Notes on neural computation

Luca Mazzucato

Departments of Biology, Mathematics, and Physics
Institute of Neuroscience
University of Oregon
Eugene, OR, 97403

Email address: `lmazzucato at scgp.stonybrook.edu` (Luca Mazzucato)

September 7, 2021

Contents

1. Introduction

These is a series of lecture notes on Computational Neuroscience. The way we will approach the topic in this class is very pedagogical and draws from my own training as a theoretical physicist. Understanding the neural code is the biggest challenge in this field. There is no consensus on what the essential ingredients are, that make the brain work/compute the way it does. Because of this, our approach will be very low-brow and pragmatic.

Let us give a working example. Try and ask a theoretical physicist: how can a horse run and win the Kentucky Derby? The physicist will start first of all with a crude model of a horse: spherically symmetric object, 2 meter radius, weighs 500 Kg and mostly made of water. This model explains the size of a horse race track, and how many horses can fit in the lane. Next we need to change the model from a sphere to a sausage-like thing, with four segments coming out from the bottom (legs). We will build a model of how the joints and muscles of the horse move. At this point the model starts to be realistic and makes order of magnitude predictions on the average speed of the race, and eventually we will probably get into the fine prints of what it is that makes a horse win races. Finally, we will be able to check if our predictions are correct, by comparing our findings with the special traits that made Lexington the best horse of the mid-nineteenth century. Maybe we should even build a model of how the horse's brain works!

But let us go back to Neuroscience. First thing, we will identify a particular function that our brain performs. We will try to simplify this particular function to its very essence, eliminating at first most ingredients that seem to us not essential. At this point, we will build a model to explain how this skeletal version of our brain function works. In order to do this, we will need a model neuron to begin with, then we will put together assemblies of neurons and study their features. We would probably get the gist of the function, at least from the "Artificial Intelligence" point of view: we will have a model that can simulate more or less well the brain function. The first crucial ingredient of such brain function model is the neuron. Neurons come in a large variety and are extremely complex cells. To build a model of a brain function, we need to isolate the features of the neuronal cell that we think are sophisticated enough to implement the algorithm, but not too complicated or the model would be baroque and, for example, hard to simulate.

Our real goal is to understand how the brain works! So we will ask the question whether this apparently successful model is in fact plausible from the biological point of view. It probably won't be any close to the real thing, so we will have to go back to our assumptions, revise them, add more structure. Until we see some resemblance to what we believe happens in our brain, when we perform that function. And we can play along all the way until we will be able to solve the neural code problem. Summarizing:

- Pick a brain function.
- What is the essential, simplest mechanism that can realize it? (AI point of view)
- Is this simple realization related to the brain? (Computational Neuroscience point of view)
- What kind of model neuron suffices to realize such function?
- Is this simple model good enough to be compared to the actual measurements and make predictions?
- Add details until we find reasonable agreement with neurobiological data, or abandon model and start all over again.

1.1. Structure of the course

What kind of language can we use to describe brain functions? There are at least two coarse-grained levels of analysis: the biological level of flesh and blood; the more abstract level in which the bits and pieces come together to describe the **emergent phenomena** of brain functions.

The biological level is the most fundamental one. All insights about the brain come from some experimental discovery, some new electrophysiological mechanism unveiled, some new neurotransmitter, and so on. The main focus of this series of lectures will be on how to use these insights to build models that explain what makes the brain such a special piece of flesh: three pounds of meat that are the most complex and fantastic system we ever encountered in nature. Unlike the hydrogen atom, the Standard Model, or a neutron star, our brain is such a sophisticated system that it can look into itself to understand how it works. Well, at least it hopes to do so...

Statistical mechanics

The keyword of the computational neuroscience approach to the brain is **emergence**. The single neuron is a very simple, yet powerful, computational unit. However, each brain function is the product of the collective interactions of millions of neurons, possibly all of the hundred billion neurons that make up our brain. Each neuron has on average 10^4 synapses that connect it to an equal number of other neurons. It is the largest parallel computer on Earth. The crucial insight is that complex brain functions are very likely to be collective phenomena arising from large networks of neurons. The natural language to describe collective emergent phenomena is *statistical mechanics*. The goal of this discipline of physics is to explain the macroscopic behavior of thermodynamics quantities, such as energy, temperature, entropy, by using the microscopic degrees of freedom that secretly make up the system we are looking at.

The perfect analogy is from condensed matter theory. A spin chain system is made of single units whose dynamics is rather boring: they have two states, spin up/spin down and interact with a nearest neighbor interaction that might flip or not each spin, according to the total field at that point. It turns out however that, once you look at the spin chain from afar, you find that the normal modes of vibration of the system, namely the eigenstates of its hamiltonian, are collective excitations that involve all the spins at the same time and propagate as a coherent wave along the whole spin chain: the phonons.

A second, more striking analogy, is the holographic correspondence in the context of string theory. Suppose you start with your favorite field theory and look at its renormalization group flow, as you move from the UV fixed point towards the IR. We would like to ask what the effective description of the degrees of freedom at low energy is. Take the particular example in which the theory is conformal. We look at the field theory from larger and larger distances, getting a coarse-grained description of the fields that enter into the lagrangian. It turns out that the correct description of the theory in this framework is a gravity with a negative cosmological constant, in one more dimension. It is as if the field theory develops an *emergent* extra direction, related to the energy, and its physics is described by a completely different set of degrees of freedom, that we would never guess from the UV. Nevertheless, it is always the same object that we study, from the different micro- or macro-scopic points of view.

Methods of statistical mechanics are a central tool to understand the emergence of brain functions from the collection of single neurons. The first part of the lectures is going to be a recap of the basic ideas of ensembles, mean field theory and the Ising model, including several exercises, whose solution technique is directly applicable to neuroscience problems. In particular, we will solve the one-dimensional long range Ising model, which is similar to the Hopfield model of associative memory.

Associative memory

The second part of the lecture will address the simplest example of a brain function that we can attack using statistical mechanics methods: associative memory. Historically, this is the landmark example of a mathematical model, whose emergent behavior is very reminiscent of a brain function. We will regard it as the "harmonic oscillator" of computational neuroscience. In theoretical physics, the harmonic oscillator is the basic object that we can completely solve and understand in every small detail. Even if it usually does not provide a realistic description of the physical system we want to study, it certainly gives a starting point to which we can add structure, like interactions, to make closer to the real world. Even the Standard Model of electroweak interactions has a basic structure that is a perturbation around free fields (a collection of an infinite number of harmonic oscillators).

The model of associative memory we will study is called the Hopfield network. It is a system of spins $S_i = \pm 1$, just like the Ising model, whose interactions are long range, with hamiltonian

$$H = -\frac{1}{2} \sum_{ij} w_{ij} S_i S_j \ .$$

This is our first model neuron, hard to beat as far as simplicity is concerned: a spin with two states. The crucial ingredient in the Hopfield network is the coupling matrix w_{ij}, which is a model for the synaptic connections between different neurons.

The goal of the model is to give us a hint of how auto-associative memory works. Auto-associative memory is very elementary: I give you a clue about a pattern, say "Ame" and you have to retrieve to whole pattern, stored as a memory: "America". In terms of our Hopfield network, the stored memory is a fixed configuration of spins $S_i = \xi_i$. Upon presenting the system just part of the pattern, say half of it, the configuration of spins evolves dynamically according to some update rule

$$S_i(t+1) = \mathrm{sgn}\left(w_{ij} S_j\right) \ ,$$

to end up in the full stored pattern ξ_i.

It is believed that the storage of patterns as long-term memory (from hours to years) is encoded in the *synaptic strengths*. Namely, the information about the pattern is encoded in the particular values taken by the synaptic connection between the neurons. The main character of our story is therefore the matrix of synaptic weights,

$$w_{ij} = \xi_i \xi_j \ .$$

This particular form of the synaptic weights is called the *Hebb rule*, and leads to a strikingly efficient memory retrieval.

We will go through the details of two aspects of this auto-associative memory system:

- *Storage capacity*: How many patterns can we store in such network?
- *Efficiency*: How can we make sure that spurious memory states do not compromise our retrieval task?

We will postpone to later chapters the question of how we can learn some new memory.

Probability theory and statistical inference

Our brain is a device that allows us to detect the current state of affairs around us, use this guess to make predictions about the future, and then make decisions based on those predictions. Since we live in an *uncertain* world, our guesses are irreducibly probabilistic. For example, our brain takes partial clues coming from the sensory inputs and estimates what the causes of those stimuli where. In other words, the brain is performing *inference* about the world around it, given certain inputs. The proper language to describe inference is clearly probability theory. That is why we devote a short interlude to recap the basics of Bayesian statistics, in particular the laws of conditional probabilities, enshrined into Bayes' theorem. The probability $P(x|y)$ of x occurring, given that y occurred is

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \ .$$

This simple formula gives us a key to interpret a lot of tasks the brain performs.

The statistical mechanics approach to studying emergent phenomena is on the other hand perfectly suited to be applied to probability theory. After all, the hamiltonian of the Ising model gives us a natural measure of the probability $P(\{S_i\})$ of a particular spin configuration at a fixed temperature $1/\beta$

$$P(\{S_i\}) = \frac{e^{-\beta H(S_i)}}{\sum_{\{S_i\}} e^{-\beta H(S_i)}} \ .$$

where the temperature is not the temperature of our brains, but rather the noise in delivering the neural signal. A typical quantity we want to estimate is the probability distribution $P(w|\xi)$ of the weights w_{ij} that can encode a certain stored memory pattern ξ_i, called the *likelihood* of the parameters, given the data.

At the same time, we need probability theory to analyze the data coming from our neuroscience experiments and isolate the meaning of the neural code. So inference is doubly relevant to this field, being both the method we used to extract information from the experimental data and, often, a simple model of brain function itself that we are after.

Supervised learning

So far, we discussed the simplest possible system that, once the memories are stored, can act as an auto-associative memory and retrieve the full pattern given a partial clue. We did not say anything about how the storing process works. The most striking feature of the brain, from many perspective, is the fact that it can *learn* new stuff at all. If you repeat ten times a ten digits important phone number, chances are you are going to remember it when needed. How do memories form?

It is believed that *synaptic plasticity* is the neurophysiological mechanism responsible for long-term memory.[1] This means that all of the information is stored in the particular values that the synaptic weights w_{ij} take. The question is: how do we modify the weights to store new information?

We will not be interested so much in the biochemical details of synaptic plasticity, as rather in understanding the simplest model of synaptic update that can realize a particular brain function (and then checking whether this is biologically plausible). We will consider first the simplest way of learning: learning with a teacher, or supervised learning. Our brain is given a particular input ξ_i, it performs its calculations and spits an answer $\mathcal{O}$. The teacher gives the neuron the *correct* answer, call it ζ, and the neuron adjusts its synaptic weights w_{ij} in small steps, upon presentation of the training patterns $\xi - \zeta$,

[1] Short-term (a.k.a. working) memory is related to persistent activity of firing neurons instead.

until the output reproduces the correct answer.

A good learning rule, which is also biologically plausible, is the delta rule:

$$w_{ij}^{new} = w_{ij}^{old} + \delta_i \xi_j \ ,$$

where $\delta_i = \zeta_i - \mathcal{O}_i$ is the error. It is possible to recast this learning rule into an extremization problem, where the cost function G to be minimized measures the distance between the correct answer ζ_i and the actual output $\mathcal{O}_i$. This is called the gradient descent learning algorithm

$$w_{ij}^{new} = w_{ij}^{old} - \frac{\partial G}{\partial w_{ij}} \ .$$

We will show how to implement the supervised learning algorithm into increasingly more realistic systems. We will start with the *perceptron*, a single neuron with a simple activity rule

$$\mathcal{O} = \mathrm{sgn}\,(\sum_j w_j \xi_j) \ .$$

This device can perform the AND and OR functions, but not the XOR. If we collect more layers of perceptrons into a multi-layer feed-forward network, the signal moves unidirectionally from the inputs, through the layers and to the outputs: this device can perform the XOR task and is, in fact, a universal approximator. The delta rule takes a simple local form in these networks, called error back-propagation.

Empowered with probability theory, we are in the perfect position for interpreting learning as an inference process that our brain makes on the external world. It will be amusing to recast the learning rules into the language of conditional probability and see that the gradient descent algorithm corresponds to a *maximum likelihood* procedure, that produces the set of synaptic weights w_{ij} that are *most likely* to reproduce the set of training data $\xi - \zeta$ provided by the teacher.

Unsupervised learning

We have considered the most obvious way of learning something: learning with a teacher. A neuron receives some inputs, performs its computation, then spits an output. There's a teacher outside that tells the neuron what is the *correct* output. If the two outputs do not agree, the neuron adjusts its synaptic weights according to a supervised learning algorithm. It is clear that in general there will not be a teacher around to tell each output neuron if its computation was correct. We will turn next to the case of learning in which there is no teacher (unsupervised). Namely, we have a neural network with some input and a learning rule, which is only based on inputs ξ and/or outputs $\mathcal{O}$. The network is fed some inputs, spits out an output and updates its weights automatically. The learning rule we will adopt is the simplest form of the celebrated Hebb rule for synaptic plasticity: If neuron A and neuron B fire together, the weight connecting them gets strengthened:

$$\Delta w_{ij} = \mathcal{O}_i \xi_j \ .$$

We will show that this simple rule, with some little adjustments, performs the Principal Component Analysis of the input signal: it computes the eigenvectors of the covariance matrix of the input.

Unsupervised learning is what happens for instance when you are just a newborn baby, coming out of the womb for the first time. Your cortex must learn how to use the inputs coming through your senses, say the visual signal from the retina. But there is no Maxwell devil sitting on top of your retina, instructing your neurons about their tasks. Unsupervised learning with a multilayer feedforward architecture explains how the neurons in the visual cortex learn to detect the orientations of the edges in the images of the world around us!

In the supervised learning algorithms, our model neurons were able to learn very specific set of patterns $\xi - \zeta$. However, in the real world we are not really sure about what are the causes of our stimuli. In fact, we can at most estimate our degree of belief that a certain stimulus is produced by a certain cause A, rather than a different one B. To overcome this limitation, we will introduce a more sophisticated device: the Boltzmann machine. It has two main features:

- It learns probability distributions, rather than fixed patterns.

- It contains *hidden units*, that can detect higher order correlations among the inputs.

The learning algorithm for this device is going to be extremely interesting. It consists of two terms called "wake" and "sleep" terms. The "wake" term is a Hebbian rule whose goal is to make the weights as close as possible as the ones that produce the training distribution. The "sleep" term works when the machine is not connected to external inputs, but is in a sense "dreaming", or evolving according to its internal dynamics. This last term removes the correlations generated by the sleeping mode.

Reinforcement learning

Unsupervised learning has had a big success in explaining how the brain develops basic abilities of detecting features in the sensory stimuli coming from the outside world. However, once we grow up, we learn and make decisions based on our own interaction with the environment. Usually, we turn right and walk ten blocks to get to a grocery store, where we can find our favorite brand of granulated sugar. But since we are very curious animals, we'd like to explore the neighborhood to check if there are other grocery stores, either closer or that carry a better kind of delicious granulated sugar. Since we can't afford a smartphone with GPS, we must just take the chance and walk around town. After walking for a while, sometimes we will find a new grocery store with amazing sugar, sometimes we will just run into stray dogs that bark at us. By performing our daily exploration, it happens that after a few trials, we somehow learn where the best grocery store in town is and what the quickest way to get there is.[2]

This kind of learning is still somehow supervised, because when we walk the right way we get rewarded, otherwise we get punished or just bored. But in this kind of learning, there is no omniscient teacher that supervises each of our

[2] Yes, we are the subjects of the maze task experiment, performed on us by the little white rats in our lab dadams [1].

billions of neurons and reveals the correct spike that leads to the sugar. In real life, most of the time we do not have a "training set" of input-output matches to update our synaptic weights. We just get an overall reward signal: some quantity of sugar. Based on that, we must modify our synaptic weights in order to learn how to get to our treat. This kind of biologically plausible learning procedure is called reinforcement learning.

A procedure that realizes this intuition is one that maximizes the reward: it is called reinforcement learning. We will discuss how to implement a gradient ascent method using the average reward $\langle R \rangle$ as an error function function to learn our way through town:

$$\Delta w_i = \eta \frac{\partial \langle R \rangle}{\partial w_i} \ .$$

This implements the delta rule of reinforcement learning, that updates the weight based on the difference between the actual reward R and the expected reward V

$$\Delta w_i = \eta (R - V) \xi_i \ .$$

This method works great if the decision-making task we face is simple. Namely, if we just have to decide a left or right turn, and then get a reward or not. But suppose that our grocery store is two blocks from us. At the first block, we have to decide where to turn. Here, we don't get an immediate reward, but we must choose again at the second block and only then get rewarded. In case we miss the store, how can we decide that we did a mistake at the first block, rather than at the second block, or both? Or imagine the case in which there are two different grocery stores: the one that carries our favorite sugar is five blocks away but there is a closer one two blocks away, with a so and so brand of sugar. How are we going to decide what is the best course of action at an intermediate traffic light? This is the *temporal credit assignment* problem.

We clearly need a way to assign different values to different places, say each block, given a particular course of action: the total amount of reward we expect from that block on. We will see how to solve this problem in a particular model called temporal difference (**TD**) learning. By comparing the total future reward we will get to our current estimate of such reward, we can adjust our course of action to maximize the total reward. This is called the **actor/critic** model of reinforcement learning.

The game has two players. We start by deciding a *policy*, namely with what probability we turn left or right at each block. Then the first player, called the critic, evaluates how much reward we would get under this course of action and the rewards we got during our previous explorations, namely how much effective our policy proved so far: in this phase we compute the difference between actual and expected reward at each block. In the second step, the actor plays and uses the critic's evaluation of the policy to improve the policy itself, based on such temporal differences. This scheme is a rough approximation that explains successfully a wide variety of conditioning experiments.

Spiking neurons

The first part of these notes is concerned with conceptual issues mostly. Our goal there was to isolate and explain piece by piece what are the simplest computational mechanism that can potentially describe some aspects of our brain functions. We considered auto-associative memory and the various learning algorithm, supervised, unsupervised, reinforcement, as tasks to be realized in networks of very simple neurons: either Ising spins, or perceptrons.

In the second part of these notes, we will take a more pragmatic point of view and try to make direct contact with the flesh and blood neuronal cells inside our brain. Whereas the perceptron captures some crude aspects of the neuron computational abilities, it is clearly very far from reproducing its biological features. We will take a closer look at a real neuron and see how it actually processes the inputs and spits the output using the **action potential**. This electro-chemical signal is the fundamental ingredient in neural computations and we will refer to it as a **spike**. We can describe spike generation using a set of coupled differential equations that go under the name of the **Hodgkin-Huxley model** of the neuron: it describes the dynamics of ion channels and gates that conspire in generating the action potential.

In order to reproduce many experimental observations, though, we will not need the full glory of the Hodgkin-Huxley neuron. It suffices to consider a much simpler model, known as the **leaky integrate-and-fire** neuron. The membrane potential $V(t)$ of the neuron is schematically encoded into a simple RC circuit, described by the differential equation

$$\tau_m \frac{d}{dt} V(t) = -[V(t) - V_{rest}] + RI \ .$$

The pre-synaptic neurons excite a current I through the ion channels at the synapses. The current flows into our post-synaptic neuron, whose membrane resistance is R, and its potential $V(t)$ increases above its resting value V_{rest}. The rule of the game is that whenever $V(t)$ reaches the *threshold* value V_{th}, the neuron emits an action potential (spike) and resets to its resting potential V_{rest}. The leaky part of the model is the fact that, when there is no input currents I, the potential relaxes exponentially to its resting value with a time constant $\tau_m \sim 10$ ms.

Stochastic neurons

When you look at neurons *in vivo* in the cerebral cortex, it is often the case that the input current, generated by the thousands of signals coming from the pre-synaptic neurons, looks like a noisy current. In the diffusion limit for the incoming currents, one can reproduce the features of such a live neuron using a stochastic differential equation called the Ornstein-Uhlenbeck process

$$\tau dV = -(V - V_{rest})dt + RIdt + \sigma \xi_t \sqrt{dt} \ .$$

The input current has two terms. The first term I is the mean current, while the last term represents the noise contribution to the current, characterized by the zero mean Gaussian process ξ_t and its variance σ.

At this point, we will need to take a short detour into the study of stochastic differential equations and the Fokker-Planck equation.[3] Using these powerful

[3] Such methods are useful in a wide variety of context. In particular, they are the same methods that hedge funds exploit to skim billions out of the market by high frequency trading.

tools we are now well equipped to build realistic models of networks of neurons in the cortex. We will consider one particular example, that has been widely studied in the literature: short-term or working memory.

Prologue: Probability and Inference

2. Probability theory and Statistical Inference

section:inference

Before we delve into neural networks, we need to have a quick refreshment on some basic notions of probability theory that are central in any Neuroscience discussion.

The approach to statistics that we will adopt is the bayesian one. This approach is typically concerned not with the probability that some event will occur but rather with the confidence that the data we measured is described by a particular model.

Bayesian inference is going to be essential to address the two different facets of Computational Neuroscience.

- **Neural encoding:** This is the process by which our brain turns a stimulus (flashlights, music, a tiger trying to eat us) into an internal representation build out of neural activity. In other words, how the brain translates the external world into a pattern of neural spikes. The way we study this is, for example, by having some animals in our lab respond to some accurately prepared stimulus we present to them. By putting some electrodes in their cortex, we can record the neural activity *evoked* by the particular controlled stimulus they receive. Once we understand how the stimulus is encoded into a pattern of spikes, we can predict how neurons will behave when the animal is presented with a different stimulus. Well, hopefully.
- **Neural decoding:** This is the inverse problem, and a very hard one. Given some recording of neural activity, we want to infer what our brain means by that: namely, try to guess what stimulus caused that particular pattern of spikes. Something like reading your mind.

Neural encoding: external stimulus → internal representation it generates

Neural decoding: internal representation → external stimulus that caused it

We are in a pretty singular position here: we are using statistical inference to understand a machine whose functions include... doing statistical inference. There's one more layer: the inference your brain is doing is focussed on understanding how to use inference to understand how your brain is doing inference...

It is clearly a vicious cycle, but we have to break the loop and start somewhere. We might as well just start studying what statistical inference is.

2.1. Probabilities

A few definitions and facts about probabilities. We will not give any proofs, but will be concerned mostly with explaining the general ideas with a few examples. We are concerned with the study of a random variable X, which might be a coin we toss, a lab test result, a raster of spikes from a multi-electrode recording of the brain.

Ensemble. The random variable X defines a triple (x, A_X, P_X) where

- x is the outcome of a measurement of the random variable X: the value a random variable takes
- $A_X = (a_1, \ldots a_I)$ is the set of possible values that the outcome x can take
- $P_X = (p_1, \ldots, p_I)$ is the probability that a measurement outcome takes a particular value: $P(x = a_i) = p_i$, where $\sum_i p_i = 1$.

For instance, if the random variable X is a regular die, then its set of possible values is $A_X = (1, 2, 3, 4, 5, 6)$ and the respective probabilities (if you are not cheating) is $P_X = (1/6, 1/6, 1/6, 1/6, 1/6, 1/6)$.

Joint ensemble. We can have a joint ensemble XY of two random variables X and Y. Each outcome is an ordered pair (x, y) where the first entry comes from $A_X = (a_1, \ldots, a_I)$ and the second entry comes from $A_Y = (b_1, \ldots, b_J)$. The *joint probability* of having outcomes $X = x$ and $Y = y$ is denoted by $P(x, y)$.

Marginal probability. Suppose we have a joint ensemble of two random variables X and Y, whose joint probability is $P(x, y)$. We can extract from this the *marginal probability* $P(x)$ of the variable X by "integrating out" the variable Y, giving the *sum rule* of probability theory:

$$P(x = a_i) = \sum_{b_j} P(x = a_i, y = b_j) \ .$$

2.2. Conditional probabilities

Conditional Probability. Given a joint ensemble of the random variables X and Y, we can define the *conditional probability of* $x = a_i$*, given* $y = b_j$ as

$$P(x = a_i | y = b_j) = \frac{P(x = a_i, y = b_j)}{P(y = b_j)} \ ,$$

valid if $P(y = b_j) \neq 0$.

This last definition leads us to the crucial *product rule* of probability theory, analogous to the chain rule in calculus. It tells us how to relate the joint probability to the different conditional probabilities

$$P(x, y) = P(x|y)P(y) = P(y|x)P(x) \ .$$

Independent variables. The two random variables X and Y are *independent* if and only if their joint ensemble

$$P(x, y) = P(x)P(y) \ .$$

This is intuitive since $P(x, y) = P(x|y)P(y)$ but $P(x|y) = P(x)$ is independent of y.

It is easy to relate the marginal probability to the conditional one:

Sum rule.

$$\begin{aligned} P(x) = & \quad \textstyle\sum_y P(x, y) \\ = & \quad \textstyle\sum_y P(x|y)P(y) \ . \end{aligned}$$

Now, using these definitions and the product rule, we can introduce the most important piece of information regarding probability theory, that we will use over and over again in any possible form.

Bayes' theorem.

$$
\begin{aligned}
P(y|x) = & \quad \frac{P(x|y)P(y)}{P(x)} \\
= & \quad \frac{P(x|y)P(y)}{\sum_{y'} P(x|y')P(y')} \ .
\end{aligned}
$$

A word of caution regarding probability theory. Conditional probabilities can be counterintuitive. Our brain is very good at estimating the trajectory of a truck coming towards us at a traffic light, but is not wired up to estimate conditional probabilities on the spot. So we must pay close attention to what we are doing, as in the following illuminating example[4].

Kitty-telepathy

Your friend Fredrick and you are having a lovely dinner at his favorite restaurant Prego. After several drinks, the conversation drifts towards the incidence of some rare diseases that may give people paranormal powers. Fredrik apparently discovered a new human gene $k001$ that is correlated to kitty-telepathy. The results of his recent experiments confirm that whoever carries the gene has these innate superpowers to talk to their kitties. Fredrik tells you that in their epidemiological studies, they found that only 1% of the population carry the gene.

You always suspected to have a special connection to your favorite kitty Cookie kitty [1], but so far you thought it was just normal empathy. Fredrik tells you that you may find out if you have mental superpowers, by testing for gene $k001$. After drinking up the last few trappist beers on your table, you and Fredrik decide it is a great time for doing science and take a cab back to your lab on 175th Street in Manhattan and submit yourself to the test. It turns out that Fredrik always brings with him in his backpack the testing equipment for gene $k001$. That's weird, you think, but hey.

[4] A wonderful book discussing statistical biases in human reasoning is kahneman2011thinking [2]

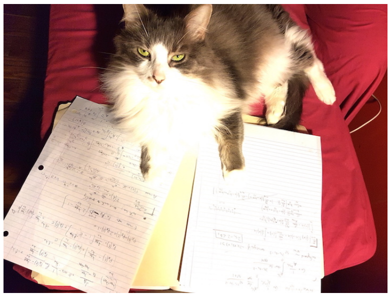

Figure 1: Cookie: a case study of kitty-telepathy.

kitty

It's hard to focus while intoxicated, so it takes a while to get yourself tested. Results finally come in: you indeed test positive for the gene $k001$! You and Fredrik leave the lab, and when you get back home and see Cookie, you stare at her and try to communicate using telepathy. She doesn't look impressed though.

The next day, late in the afternoon and still hungover, you ask yourself: What is the probability that, scoring positive in the test, you are a kitty-whisperer?

Before sitting down and calculating, you realize that, while this kind of genetic screening is nearly 100% reliable, the night before the two of you were pretty dazed, which might have affected the lab results. So you look up on the National Institute of Health guidelines, and find that the official reliability of a genetic test while intoxicated is 90%. This means that 90% of the time when you have a gene and you test for it while intoxicated, it yields a positive result. 90% of the time, when you do not have a gene and you test for it while intoxicated, you find negative results.

Solution

First rule of statistical inference: write down all of the probabilities involved in the problem. This is the easiest way to figure out the answer. Let us introduce the following random variables:

$$t = 1 \qquad \text{Positive test result,}$$

$$t = 0 \qquad \text{Negative test result,}$$

and also

$$g = 1 \qquad \text{You have the gene,}$$

$$g = 0 \qquad \text{You do not have the gene.}$$

The NIH guidelines of test reliability while intoxicated give you the conditional probability of t given g

$$P(t = 1|g = 1) = 0.9\ , \qquad P(t = 1|g = 0) = 0.1\ ,$$

$$P(t = 0|g = 1) = 0.1\ , \qquad P(t = 0|g = 0) = 0.9\ ,$$

The presence of the gene among the population gives you the marginal probability of g

$$P(g = 1) = 0.01\ , \qquad P(g = 0) = 0.99\ .$$

You want to compute the probability of actually having the gene, given the positive test result $P(g = 1|t = 1)$. We can use Bayes' theorem and compute

$$P(g = 1|t = 1) = \frac{P(t = 1|g = 1)P(g = 1)}{P(t = 1)}\ .$$

The marginal probability of having a positive test result can be obtained by summing up

$$P(t = 1) = P(t = 1|g = 1)P(g = 1) + P(t = 1|g = 0)P(g = 0)\ .$$

Hence we find

$$\begin{aligned} P(g = 1|t = 1) = &\ \tfrac{P(t=1|g=1)P(g=1)}{P(t=1|g=1)P(g=1)+P(t=1|g=0)P(g=0)} \\ = &\ \tfrac{0.9\times 0.01}{0.9\times 0.01+0.1\times 0.99} \\ = &\ 8.3\%\ . \end{aligned} \tag{1}$$

This 8.3% expresses our *confidence* that we have the gene, given the lab test result.

I bet your first guess for the probability went like: "Well if the reliability of the test while intoxicated was 90% and I scored positive, my chances of actually having the gene is 90%." But that was clearly wrong, and you must apply Bayes' theorem to get the correct result. Even if your test was positive, you have only small chances of actually having the gene. Note that this probability is very sensitive to the reliability of the test, which enters in the marginal probability of testing positive $P(t = 1)$, in the denominator. If you improved only slightly the test reliability from 90% to 95%, for example, you would double the probability of having the gene given a positive test.

2.3. Forward, posterior, likelihood

There are many different meanings we can associate to a particular probability.

Suppose we come up with a model of what happens when we toss a coin in the air. I want to know what is the probability of having heads or tails, using my model that says that all results are equiprobable. Then I predict $P(x = \text{heads}) = 1/2 = P(x = \text{tails})$ is a forward probability.

Forward probability. Given a generative model that describes a process, the probability distribution of the random variable (data) generated by that model is called a *forward probability* (like when you do a computer simulation).

Imagine that you have two different random variables: a die and 6 urns filled with colored balls. Each urn contains ten balls in total and urn x contains x green and $6 - x$ red balls. You first throw a die and get a number $x = 1, \ldots, 10$ that determines which urn x you pick. Then, given an urn, you pick a random ball and write down its color. A fun fact about this example. The forward probability of finding n green balls out of N draws from the urn x varies depending on the procedure for the draw. If you draw *with replacement*, then you can use the binomial distribution:

$$P(n_{\text{balls}}|x, N) = \binom{N}{n} x^n (1 - x)^{N-n}, \qquad \text{with replacement}$$

because after every draw, you put the ball back into the urn and draw again, so every draw comes from the same distribution. However, if you draw *without* *replacement*, then you must you the hypergeometric distribution:

$$P(n_{\text{balls}}|x, N) = \frac{\binom{x10}{n}\binom{10(1-x)}{N-n}}{\binom{10}{n}}, \qquad \text{without replacement}$$

because after every draw, the number of balls left in the urn changes and so every draw comes from a different probability distribution.

At the end of the experiment, you realize that you forgot to write down the sequence generated by the die! So you want to infer the probability of the die outcomes, given the distribution of balls of different colors $P(\text{die}|\text{balls})$. This is an example of inverse probabilities.

Inverse probabilities. Given a set of measurements, a model that generates the data, and another *unobserved variable* in our model, we call an inverse probability the conditional probability of the unobserved variable, *given* the observed data.

Inverse probabilities invariably need the application of Bayes' theorem:

$$P(\text{die}|\text{balls}) = \frac{P(\text{balls}|\text{die})P(\text{die})}{P(\text{balls})}\ .$$

Let us introduce some useful terminology for inverse probabilities. In this example of the die, the probability of the outcome of the tossing of the die $P(x_{\text{die}})$ is called the **prior**. $P(n_{\text{balls}}|x_{\text{die}})$ is a function of two variables n_{balls} and x_{die}. For fixed x_{die} it is a probability distribution over n_{balls}. When given a series of observations of n_{balls}, then it is a function of x_{die}: The probability of a certain distribution of balls, given the distribution of die outcomes $P(n_{\text{balls}}|x_{\text{die}})$ is called the **likelihood** of the die. The probability we are after, inferring the distribution of die outcomes from the ball distribution $P(\text{die}|\text{balls})$ is called the **posterior** probability, and it is what we will be after most of the time. The marginal probability of observing the data $P(\text{balls})$ is called the **evidence**.

Let us rephrase this in general terms. We are assuming some model of the world, that contains certain parameters that we collectively denote as θ. The model is based on some other assumptions $\mathcal{H}$. We make an experiment and we observe some data D, that we think was generated by our model. We want to infer the posterior probability of the parameters θ of our model, given the observed data. This tells us with what *confidence* we can believe that the data we measured has been generated by a certain value of the parameters in our model, rather than a different value. Using Bayes' theorem we find

$$P(\theta|D,\mathcal{H}) = \frac{P(D|\theta,\mathcal{H})P(\theta|\mathcal{H})}{P(D|\mathcal{H})}\ . \qquad (2) \qquad \boxed{\texttt{posteriorprob}}$$

The various quantities in the previous equation are called:

$$\mathrm{Posterior\,of}\,\theta = \frac{\mathrm{Likelihood\,of}\,\theta\,\mathrm{given}\,D \times \mathrm{Prior\,over}\,\theta}{\mathrm{Evidence\,of\,D}}\ .$$

The evidence does not depend on the model parameters θ, so it is usually not necessary if we are just estimating the most likely value of the parameters given the data. In this case, we just extremise the posterior probability over the parameters. But it is the crucial quantity that enters model selections, when we compare different models that may generate the observed data and we want to find which model is favored by the data.

The crucial point in the discussion is the interpretation of posterior probabilities. The parameters θ in the model are not random variables. What does their probability distribution mean then? The posterior probability $P(\theta|D,\mathcal{H})$ gives the *degree of belief* in the parameters of our model, given the observed data D. In particular, by extremizing the posterior with respect to the parameters θ, we can derive the most likely value of the parameters, given the observed data.

2.4. Inference

We are all theorists here. Everybody has its favorite model of how stuff happens, and we would like to use probability theory to tell good models from bad models. That is, we are going to *infer* how likely our favorite model is to describe what we see. When you have data from an experiment and you want to test if your favorite model has anything to do with the results of the experiment, remember the following mantra:

> Posterior probabilities measure the *degree of belief* in your assumptions (your favorite theory), given a set of empirical data.

Our hidden assumption here is that we cannot of course make any statistical inference without making prior assumptions, since it is precisely those assumptions that we are going to test and quantify how much we can rely on them. A corollary to this method is that

Our inference task only depends on the set of data we observe and on the prior distribution we assumed for the parameters of our favorite model.

Armed with this wisdom, let us see how to test our model of a dentist shop. We will first describe the set of data we measure. Then we will try to invent a model for a probability distribution that we believe describes our data, and we will discuss the parameters in this model. We want to infer what are the value of the parameters in our model which are most likely, given the observed data. To do that, we will have to make an assumption on the prior distribution of the parameters in our model.

At the dentist waiting room

You go to the dentist very often, because you really like when he picks up the drill and starts to open holes in your teeth. The dentist is open all day long and the waiting room is always crowded, people seem to enjoy this. You want to measure the average waiting time for a patient in line: the inter-patients interval $x_i = t_{i+1} - t_i$ between two consecutive patients being called in. It can take up to an arbitrarily short time, because sometimes patients misremember their appointment and are sent away in a few seconds. However, every one hour you must leave the waiting room to drink a coffee, or you would just fall asleep and jeopardize the whole endeavor. Your data is therefore covering the interval $x \in [0, 60]$ minutes. Your data points consist of a set of time intervals $A_X = \{x_1, \dots, x_n\}$.

What is the theoretical model for the distribution of inter-patients intervals you are measuring? Let us derive it from first principles. The times t_i at which the next patient is called into the dentist room are our starting point. The probability of a patient being called between time t and $t + \Delta t$ is uniform and proportional to the time interval itself

$$P(\text{patient called in } [t, t+\Delta t]) = r\Delta t \ ,$$

where r is a constant rate. The probability that no patients are called in the same interval is

$$P(\text{no patient called in } [t, t+\Delta t]) = 1 - r\Delta t \ ,$$

The probability of a patient being called in $[t_1, t_1 + \Delta t]$ and another patient being called in $[t_2, t_2 + \Delta t]$ does not depend on t_1, t_2 but only on the length of the interval. The two events are independent, hence the joint probability is the product of the two separate probabilities

$$P(\text{patient called in } [t_1, t_1 + \Delta t] \, AND \, \text{patient called in } [t_2, t_2 + \Delta t]) = (r\Delta t)^2 \ .$$

Let us denote by $N(T)$ the random variable describing the total number of patients that are called into the dentist room during a time window of T minutes. What is the probability that n patients are called in the dentist room during a long interval T? Let us split the total interval T in M small temporal bins of length Δt such that $T = M\Delta t$. Such probability is given by the probability of n bins being full, times the probability of $M - n$ bins being empty, times the number of ways we can ditribute the n patients among the M bins

$$P(N(T) = n) = (r\Delta t)^n (1 - r\Delta t)^{M-n} \frac{M!}{n!(M-n)!} \ .$$

Taking the limit $M \to \infty$ and $\Delta t \to 0$, while keeping fixed the time window $T = M\Delta t$

$$\begin{aligned} P(N(T) = n) = & \quad \tfrac{(rT)^n}{M^n} \left(1 - \tfrac{rT}{M}\right)^{M-n} \tfrac{M!}{n!(M-n)!} \\ = & \quad \tfrac{(rT)^n}{M^n} \underbrace{\left(1 - \frac{rT}{M}\right)^M}_{\to e^{-rT}} \underbrace{\left(1 - \frac{rT}{M}\right)^{-n}}_{\to 1} \tfrac{1}{n!} \underbrace{\frac{M!}{(M-n)!}}_{\sim M^n} \end{aligned}$$

and we find the Poisson distribution:

$$P(N(T) = n) = \frac{(rT)^n}{n!} e^{-rT} \ . \tag{3}$$

poisson

It is easy to check that the probability is normalized to one: $\sum_{n=0}^{\infty} P(N(T) = n) = 1$.

We are almost there. We want the probability that our random variable X which describes the time interval between patients takes the value T: $P(X = T)$. Now, the probability that the next patient is called after T minutes is

$$P(X > T) = P(N(T) = 0) = e^{-rT} \ .$$

On the other hand, the probability that the next patient is called before T minutes is

$$P(X < T) = 1 - P(X > T) = 1 - e^{-rT} \ . \tag{4}$$

probabint

We can define the probability of observing an inter-patient interval X smaller than T as

$$P(X < T) = \int_0^T P(X = x)dx \ . \tag{5}$$

defpxt

The fundamental theorem of calculus tells us that

$$f(T) = \frac{d}{dT}\int^T f(x)dx \ . \tag{6}$$

fundamental

fundamental defpxt probabint

Putting together (6) with (5) and (4), we find that

$$P(X = T) = \frac{d}{dT}\int_0^T P(X = x)dx = \frac{d}{dT}(1 - e^{-rT}) \ ,$$

and finally obtain the probability that the next patient gets in at time $X = T$

$$P(X = T) = re^{-rT} \ .$$

Using the sheer power of logic, we came up with a model for the probability distribution of the time intervals X between a patient and the next patient. The probability that the time interval is equal to T is

$$P(X = T) = re^{-rT} \ . \tag{7}$$

expdistribution

Note that the model depends on a single parameter r, which is the frequency at which patients are called in.

Now let us go back to our original inference problem. We are at the dentist and we measure the intervals x_i between two consecutive patients getting called in, so our data is a set of time intervales $\{x_1, \ldots, x_N\}$, all of which are inside the window $x_i \in [0, 60]$. We want to estimate, given the data, what the *most likely value of the parameter* r is.

You know that the mean of the exponential distribution[5] is $1/r$. Your first guess at the parameter is to compare the mean $\bar{x} = \sum_{i=1}^{N} x_i/N$ of the data set with $1/r$. But in this way, we would get a really misleading estimate, in the case in which the true value of the parameter r^{-1} is larger than the time interval $[0, 60]$ in which we are measuring our data. We could try to fit our curve to the data, but still that will not help in the case of a very small frequency $r^{-1} >> 60$.

The clue is that our estimate of the parameters must rely *exclusively* on the set of data that we are given: we have to restrict our probability distribution to the interval $x \in [0, 60]$. Namely, we start with a probability distribution

$$P(X = x|r) = \begin{cases} re^{-rx}/Z(r) \ , & 0 < x < 60 \\ 0 \ , & \text{otherwise} \end{cases} \qquad (8) \qquad \boxed{\texttt{model}}$$

By imposing that the total probability sum up to one, we fix the normalization constant

$$Z(r) = \int_0^{60} re^{-rx} dx = 1 - e^{-60r} \ . \qquad (9) \qquad \boxed{\texttt{normalization}}$$

Now we can use Bayes' theorem to derive the posterior probability of our parameter r, given the observed data $\{x_1, \ldots, x_n\}$

$$P(r|\{x_1, \ldots, x_n\}) = \frac{P(\{x_1, \ldots, x_n\}|r)P(r)}{P(\{x_1, \ldots, x_n\})} \ , \qquad (10) \qquad \boxed{\texttt{posterior}}$$

where $P(\{x_1, \ldots, x_n\}|r)$ is the *likelihood* that our data set is generated by a distribution with parameter r, $P(r)$ is the *prior distribution* of the possible values of the frequency. Since we are after the value of the parameter r that maximizes the probability ($\overset{\texttt{posterior}}{10}$), we do not need to evaluate the *marginal probability* of the data in the denominator.

[5] The mean of the exponential distribution is $\langle X \rangle = \int_0^\infty xP(X = x)dx = \int_0^\infty xre^{-rx}dx = 1/r$.

We can easily estimate the likelihood using our model ($\overset{\texttt{model}}{8}$) given the data and its normalization constant ($\overset{\texttt{normalization}}{9}$), since the single events (patients getting into the room) are independent and so their joint probability factorizes

$$P(\{x_1,\ldots,x_n\}|r) = \prod_{i=1}^{N} P(x_i|r) = \frac{r^N e^{-r(x_1+\ldots+x_N)}}{Z(r)^N}\ . \tag{11}$$

likelihood

At this point, we must make an explicit *assumption* on the prior probability distribution of the frequencies $P(r)$. A reasonable assumption is that the probability distribution of having a frequency takes any value inside a certain interval $[r_{min}, r_{max}]$ is constant: $P(r) = const$, independent of r. It is reasonable to restrict r inside a finite interval due to human life constraints. With this assumption, the prior probability is independent of the parameter and hence it does not enter in the maximization of the posterior probability ($\overset{\texttt{posterior}}{10}$). Hence, we find that maximizing the posterior distribution ($\overset{\texttt{posterior}}{10}$) is equivalent to maximizing the likelihood of the model given the data ($\overset{\texttt{likelihood}}{11}$).

As you are probably suspecting, the assumption on the prior probability distribution of our parameters is a crucial step in the inference problem. However, different priors would be better suited for the dentist's office. You noticed that patients *arrive* every 15 minutes or so, and you reason that it would make sense for the dentist to have patients come in at roughly a rate that is close to the actual rate of getting called in. So you can formulate a different prior, namely a truncated gaussian distribution with mean $\bar{r}^{-1} = 15$ minutes and variance $\sigma_r^2 = 1$ minutes^{-2}. Remember that the prior should be normalized such that its integral gives 1:

$$P(r|\bar{r},\sigma_r^2) = \begin{cases} e^{-\frac{(r-\bar{r})^2}{2\sigma_r^2}} / Z_{prior}(\bar{r},\sigma_r^2)\ , & r \geq 0 \\ 0\ , & \text{otherwise} \end{cases} \tag{12}$$

gaussianprior

where $Z_{prior}(\bar{r},\sigma_r^2) = \int_0^\infty P(r)dr$. When your prior depended on the parameter itself, the maximization procedure would yield a completely different result. Bayes' theorem is very good in forcing you to make all of your assumptions explicit: it is much easier to question them once they are exposed (and not hidden under the carpet of baroque models).

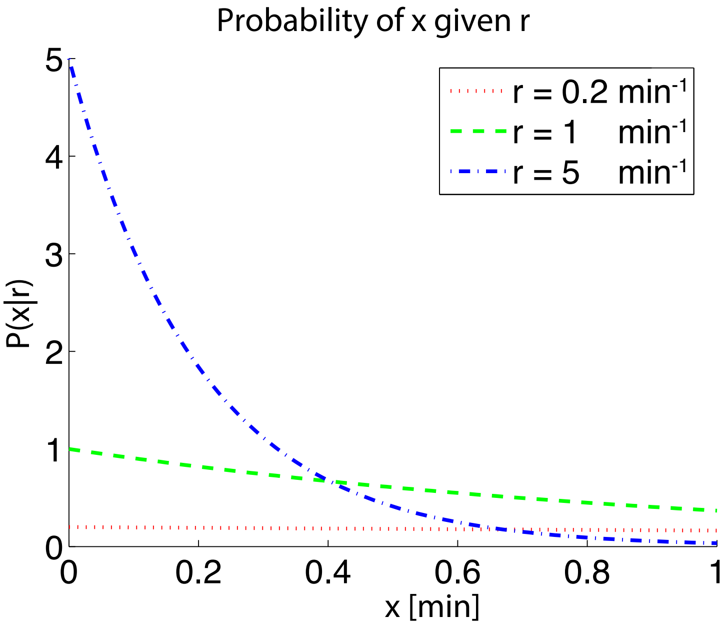

Figure 2: Fixing different values of our model parameter r, this is the likelihood of measuring a certain inter-patient interval x. It is the usual exponential distribution. Pofxgivenr

Let us make several comments on this inference procedure.

- If we consider the probability distribution (model 8) as the probability of generating the data x given a fixed values of the parameter r, we see in Fig. Pofxgivenr 2 that it is just an exponential function of x and it does not have any extremum inside the interval $[0, 60]$.

- On the other hand if we regard the same quantity as a function of the parameter r, given a fixed set of data $\{x\}$, the story changes completely and an extremum emerges! The particular features of the data set in Fig. Pofrgivenx 3 change completely the posterior probability as a function of the parameters. If we only have one data point, the posterior probability only has a maximum if the data point is very small. As soon as the interval gets closer to the limit of sixty minutes, it looses significance and the posterior probability flattens out, losing its maximum.

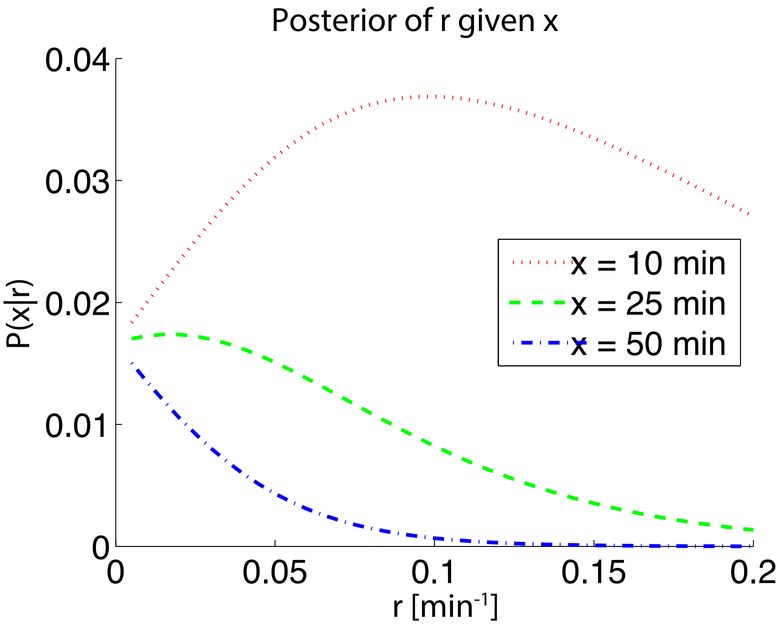

Figure 3: Given three different data points x, this is the probability distribution of the r. The most likely value of the parameter r is the one that maximizes the distribution. If x is small, there is a clear maximum. But if x is close to the upper end of the interval range 60, our model becomes inconclusive, namely the maximum is at the edge of the support interval for r. Pofrgivenx

- As we add more data points, the posterior probability becomes very peaked around a particular value of the parameter, so the inference procedure is very efficient.

A new mantra we have to adhere to when inferring posterior probabilities is the "likelihood principle".

> We have our favorite model $\mathcal{H}$, that generates the likelihood of data d, depending on some parameters θ. Suppose we observe a particular set of data d_1. Then all our inferences and predictions should depend only on the observed data $P(d_1|\theta)$ and not on any other sort of imagined data d.

2.5. Information and entropy

We want to introduce now some quantities that will be useful when considering learning algorithms. We are going to interpret the learning procedure as maximizing the information content retained by our brain: but we need to define properly what we mean by information.

Given a random variable X (which comes with its outcome x, its possible values $A = (a_1, \ldots, a_I)$ and its probability distribution $P = (p_1 \ldots, p_I)$, we can define ask what the information content $h(x)$ of one observation x is. The lowest the probability $p(x)$, the largest the information content, so $h(x)$ should be an monotonically decreasing function of $p(x)$. Moreover, if we observed two independent events x and y (namely, $p(x, y) = p(x)p(y)$, the information gained should be the sum of the information gained by observing each one of them separately: $h(x, y) = h(x) + h(y)$. These two requirements, plus the constrain that information is always non-negative, imply that the

Shannon information content of the data point x

$$h(x) = \log_2 \frac{1}{P(x)} \ .$$

The most unlikely the data point, the highest its information content. The entropy $H(X)$ of an ensemble X is given by the expectation value of its information content

$$H(X) = \langle h(x) \rangle = \sum_x P(x) \log_2 \frac{1}{P(x)} = \sum_{i=1}^{I} p_i \log_2 \frac{1}{p_i} \ . \tag{13}$$

shannonentropy

The entropy encodes the amount of information in a probability distribution. Some properties and definitions:

- Entropy is always positive

$$H(X) \geq 0 \ .$$

- Entropy is maximized by the uniform distribution

$$H(X) \leq \log |A_X| \ . \tag{14}$$

max:uniform

max:uniform

Exercise: Prove (14) by maximizing the Shannon entropy subject to the contraint $\sum_i p_i = 1$.

- Entropy of a joint ensemble XY is always larger or equal than the sum of the entropy of the separate ensembles X and Y

$$H(X,Y) \leq H(X) + H(Y) \ ,$$

  and the equality holds if and only if the two ensembles X and Y are independent.

- The *conditional entropy* of X given $y = b_k$ represents the additional information required to specify X (uncertainty of X) when the value $y = b_k$ is already known:

$$H(X|y = b_k) = \sum_x P(x|y = b_k) \log \frac{1}{P(x|y = b_k)} \ .$$

- The *conditional entropy* of X given Y represents the additional information required to specify X when the full distribution Y is known:

$$\begin{aligned} H(X|Y) &= \sum_y P(y) H(X|y) = \sum_y P(y) \sum_x P(x|y) \log \frac{1}{P(x|y)} \\ &= \sum_{x,y} P(x,y) \log \frac{1}{P(x|y)} \ . \end{aligned} \qquad (15)$$

The chain rule for information content follows from the product rule:

$$\log \frac{1}{P(x,y)} = \log \frac{1}{P(x|y)} + \log \frac{1}{P(y)} \ ,$$

from which we find the chain rule for entropy:

$$H(X,Y) = H(X|Y) + H(Y) \ ,$$

showing that the uncertainty in x and y is given by the uncertainty of y plus the uncertainty in x given that y is known.

The *mutual information* between X and Y measures the average reduction in the uncertainty of X which we gain by learning about Y:

$$I(X;Y) = H(X) - H(X|Y) \ .$$

Below, we will derive the mutual information from a different perspective.

In case of a continuous probability density $p(x)$, such that $\int dx\, p(x) = 1$, Shannon entropy (shannonentropy 13) is replaced by the differential entropy

$$H(X) = -\int dx\, p(x) \log p(x) \ .$$

One can show that the **maximum entropy distribution** given a fixed value of the mean μ and variance σ^2 is obtained with the normal distribution

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \ ,$$

whose entropy is $H(x) = \frac{1}{2}(1 + \log 2\pi\sigma^2)$.

Now let us imagine we are trying to model an unknown probability distribution $P(x)$ with an approximate distribution $Q(x)$. We would like to transmit the values of x to a receiver over a channel, by using our approximate distribution $Q(x)$. The true distribution has an information content $E[\log P(x)]_{P(x)} = -\sum_x P(x) \log P(x)$, while the information content of our transmission code is only $E[\log Q(x)]_{P(x)} = -\sum_x P(x) \log Q(x)$. So the average additional information content the receiver would need to reconstruct the original distribution is the difference between them, or the **relative entropy** or **Kullback-Leibler divergence** between them

$$D_{KL}(P||Q) = -\sum_x P(x) \ln \frac{Q(x)}{P(x)} \ . \qquad (16) \qquad \boxed{\texttt{KL}}$$

It satisfies the **Gibbs inequality**

$$D_{KL}(P||Q) \geq 0 \ ,$$

where the equality holds only if $P = Q$.

Now we will derive the Gibbs inequality using properties of convex function. A function $f(x)$ is called convex ("smile") it sits below its chord. Namely, given an interval $[a, b]$ then any point within this interval can be written as $\lambda a+(1-\lambda)b$ so that a convex function satisfies

$$f(\lambda a + (1-\lambda)b) \leq \lambda f(a) + (1-\lambda)f(b) \ ,$$

and, by induction, we can generalize this to

$$f(\sum_i \lambda_i x_i) \leq \sum_i \lambda_i f(x_i) \ ,$$

where $\sum_i \lambda_i = 1$ with $\lambda_i \geq 0$. But the latter two requirements define a probability distribution, so that we conclude that if f is convex, then **Jensen's inequality** holds:

$$E[f(X)] \geq f(E[X]) \ .$$

Using the fact that $-\log u$ is convex, we find that

$$\begin{aligned} D_{KL}(P||Q) &= \sum_x P(x) \log \frac{P(x)}{Q(x)} = E[f\left(\frac{Q(x)}{P(x)}\right)]_{P(x)} \geq f(E\left[\frac{Q(x)}{P(x)}\right]_{P(x)}) \\ &= -\log \sum_x p(x) \frac{q(x)}{p(x)} = 0. \end{aligned} \tag{17}$$

Consider the joint probability distribution $p(x, y)$ between two random variables X and Y, recall that it factorizes iff the two variables are independent $p(x, y) = p(x)p(y)$. Suppose we are trying to broadcast the joint probability $p(x, y)$ over a channel by approximating it with the factorized distribution $p(x)p(y)$. The additional amount of information we would need to supply to reconstruct the full joint distribution is the KL divergence between X and Y, which is called the *mutual information* between the two variables:

$$MI(X, Y) = D_{KL}\left(p(x, y)||p(x)p(y)\right) = -\int dx\, dy\, p(x, y) \ln \frac{p(x)p(y)}{p(x, y)} \ . \tag{18}$$

The mutual information is always non-negative, and it vanishes iff the two distributions are independent.

2.6. Assignments

Exercise 1

At the dentist's office, our inference problem consists in learning the posterior probability $P(r|D)$ in (posterior 10) of the model parameter r given the observations $D = \{x_1, \ldots, x_N\}$, where r has a certain prior distribution $P(r)$. The likelihood is (likelihood 11) where the observations are restricted to the interval $x_i \in [0, 60]$ minutes, for $i = 1, \ldots, N$. Let us fix a simple prior distribution, $P(r) = const$ over the positive real line. You have collected in total 5 observations: $D = \{10, 15, 50, 3, 12\}$.

Question 1: Plot the posterior distribution of r as you increase the number of observations from one to five. What do you observe?

Question 2: Compare the posteriors in the previous case, with the posterior you obtain by using two different priors: the gaussian prior in (gaussianprior 12) and an exponential prior $P(r) = \bar{r}^{-1} e^{-r/\bar{r}}$ with $\bar{r}^{-1} = 15$ minutes.

Exercise 2

Prove statement (max:uniform 14). Given a random variable X (which comes with its outcome x, its possible values $A_X = (a_1, \ldots, a_N)$ and its probability distribution $P_X = (p_1 \ldots, p_N)$, prove that the uniform distribution $p_i = 1/N$ is the one with the maximum entropy.

Hint: This is a constrained maximization problem, where you extremize the entropy (shannonentropy 13) subject to the constraint $\sum_i p_i = 1$. You may use a Lagrange multiplier.

Guide to the literature

My favorite book on information theory is the one by MacKay mackay [3]. It is freely available online at the author's webpage. This chapter is a lightning summary of some of the topics in that book, in particular Chapters 2, 3 and 8. The last part on information theory relies in part on Chapter 1 of Bishop's book bishop [4].

3. Bias/Variance Tradeoff

Estimation vs. Prediction. Given some data, we may wish to do different things with it. For example, we might want to fit some model to the data, with the goal of performing *estimation* of some quantity like the data mean, variance, entropy, and so on. The goal of Machine Learning is a bit different: to learn structures from data for making *predictions*. Its methods are usually applied to complex high dimensional data. Let us consider the differences between estimation vs. prediction.

3.1. Cross-validation

Let us consider a random variable X and a dataset $D_{train} = \{x_i, y_i\}_{i=1}^{N}$, consisting of independent variables x_i and dependent variables or target labels y_i, one for each datapoint. This is a supervised learning problem, where we want to learn the relationship between dependent and independent variables. It could be a regression problem (e.g., linear regression), or a classification problem (cats vs. dogs image classification). We are also given a model $p(D|\theta)$. When we *fit* the model to the dataset, we look for some value of the parameter $\hat{\theta}$ that best describes the data. Two different examples are: The MAP estimate: $\hat{\theta}_{MAP} = \text{argmax}_{\theta} p(D_{train}|\theta)p(\theta)$; The MLE estimate: $\hat{\theta}_{MLE} = \text{argmax}_{\theta} p(D_{train}|\theta)$, corresponding to the MAP estimate with uniform prior $p(\theta)$.

If we are looking for a model fit (or *paramter estimation*), we are done: we found the $\hat{\theta}$ that best fit the training set D_{train}. However, if we are in the business of doing *predictions*, then this is not the correct procedure. When making predictions, we have to find the $\hat{\theta}$ that best *predicts* an unobserved test dataset X_{test}. This is also called *generalization*, and it is quite a bit trickier than just estimating some parameters.

Let us be more precise and define our problem with three ingredients:

- A **dataset** $D_{train} = \{x_i, y_i\}_{i=1}^{N}$.

- A **model**, represented by a function $f : x \to y$ with parameters θ. E.g., a regression problem $y = f(x, \theta) = a_0 + a_1 x$, with parameters $\theta = (a_0, a_1)$.

- A **cost function** $C[y, f(x, \theta)]$, which tells us how well the model fits the data. E.g. least squares $C = \sum_{i=1}^{N}[y_i - f(x_i, \theta)]^2$.

If we want to find the model parameters that may best predict yet unobserved data, we will proceed as follows:

1. Split the dataset into two disjoint sets: a **training set** and a **test set**: $D = D_{train} \bigcup D_{test}$.
2. Fit the model to the training set to find $\hat{\theta}$. E.g., $\hat{\theta} = \text{argmin}_{\theta} C[y_{train}, f(x_{train}, \theta)]$.
3. Evaluate the model performance on the test set $C[y_{test}, f(x_{test}, \hat{\theta})]$, using the parameters $\hat{\theta}$ estimated on the training set at the previous step.

We define the in-sample error or *training error* as $Error^2_{in} = C[y_{train}, f(x_{train}, \hat{\theta})]$, and the out-of-sample error or *test error* as $Error^2_{out} = C[y_{test}, f(x_{test}, \hat{\theta})]$. Steps 1-3 go under the name of **cross-validation**. The parameters $\hat{\theta}$ that minimize the test error E_{out} give an unbiased estimate of the prediction or generalization ability of the model. A model endowed with these parameters can give the most accurate estimation of the label y_{new} associated with some new data point x_{new}, which was not used to train or test the model.

3.2. Poynomial regression

Why should we follow these cross-validation steps, instead of just fitting a model to the whole dataset? Let us consider a very complicated dataset (X, y), where we do not have a simple model that explains it. Rather, we have many alternative hypotheses for possible models $f_1(x, \theta)$, $f_2(x, \theta), \dots f_p(x, \theta)$ that might explain it equally well, but with an increasing level of model complexity (i.e., an increasing number p of model parameters). In this scenario, the model that gives the best fit on the training set (lowest $Error^2_{in}$) is *not* the model that best predicts the test data (lowest $Error^2_{out}$). In this case, we need to perform **model selection**, looking for the best model $\hat{p} = \text{argmin}_p \, Error^2_{out}$. The field of research that deals with these issues is called "Statistical learning theory." We will now consider in great details a simple example of how this all works in practice. We will find out how prediction depends on 1) the sample size (i.e., # of points in the training set), the measurement noise, and the prior knowledge about the system (i.e., model complexity).

Consider a dataset $D = (X, y)$ with observations X and targets y, and the following model

$$y_i = f(x_i, \theta) + \epsilon_i \ ,$$

for $i = 1, \ldots, N$, where we call $f(x_i, \theta)$ the model prediction, and ϵ_i is the measurement noise, which we take from a normal distribution $\epsilon \sim \mathcal{N}(0, \sigma^2)$ so that $\langle \epsilon_i \rangle = 0$ and $\langle \epsilon_i \epsilon_j \rangle = \delta_{ij} \sigma^2$. Here, $\langle \ldots \rangle$ denotes an average over different experiments. Given one particular value of x_i, then the corresponding value of y_i, measured over and over again across different experiments, will have a normal distribution with mean $f(x_i, \theta)$ and variance σ^2, since $\epsilon = y - f(x, \theta) \sim \mathcal{N}(0, \sigma^2)$. The likelihood $\mathcal{L}$ of a data point given the model is then

$$\mathcal{L}(y|X, \theta, \sigma) = \mathcal{N}(y|f(x, \theta), \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{[y - f(x, \theta)]^2}{2\sigma^2} \right] \ .$$

We now split the dataset into training and test sets as above and perform cross-validation. The first step is to train the model, namely, we find parameters θ that maximize the likelihood $\mathcal{L}$, or minimize the cost function defined as $\mathcal{C} = -\log \mathcal{L}$, estimated on the training set. Assuming each datapoint has been measured independently (the noise is i.i.d., independent and identically distributed for all data points), then the probability of the whole dataset is the product of the probabilities of each data points, and we find:

$$\begin{aligned} \mathcal{L}_{train} &= \prod_{i=1}^{N_{train}} \mathcal{N}(y_i|f(x_i, \theta), \sigma^2) \\ &= \prod_{i=1}^{N_{train}} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{[y_i - f(x_i, \theta)]^2}{2\sigma^2} \right] \ . \end{aligned}$$

The cost function $\mathcal{C} = -\log \mathcal{L}$ then reads:

$$\begin{aligned}
\mathcal{C}_{train} &= \sum_{i=1}^{N_{train}} \left[-\frac{1}{2} \log 2\pi\sigma^2 - \frac{[y_i - f(x_i, \theta)]^2}{2\sigma^2} \right] \\
&= -\frac{N_{train}}{2} \log 2\pi + \frac{N_{train}}{2} \log \frac{1}{\sigma^2} - \frac{1}{2\sigma^2} \sum_i [y_i - f(x_i, \theta)]^2 \ . \qquad (19)
\end{aligned}$$

We found that the cost function is proportional to the **mean squared error**: MSE=$\frac{1}{N_{train}} \sum_i [y_i - f(x_i, \theta)]^2$. This is a general result: whenever the measurement noise is gaussian, then the log-likelihood is proportional to the MSE. That is why it is convenient to work with log-likelihood and gaussian noise. In other words, the MLE estimate for gaussian noise is equivalent to minimizing the MSE. Training the model amounts to find the parameters

$$\hat{\theta} = \text{argmin}_\theta \, \mathcal{C}_{train} = \text{argmax}_\theta \log \mathcal{L}_{train} \ .$$

MLE for polynomial regression. The extremum of the cost function is obtained at $\frac{\partial}{\partial\theta} \log \mathcal{L}_{train} = 0$, namely $\sum_i^{N_{train}} [y_i - f_p(x_i, \theta)] \partial f_p(x_i, \theta)/\partial\theta = 0$. In the specific case of polynomial regression, the model is

$$f_p(x, \theta) = \sum_{\alpha=0}^{p} \theta_\alpha x^\alpha = \theta_0 + \theta_1 x + \ldots + \theta_p x^p \ . \qquad (20)$$

polymodel

Let us define the following quantities: a column vector whose rows are the targets y_i. A matrix $X_i^\alpha = x_i^\alpha$, i-th row and α-th column represents the i-th observation raised to the power α. A column vector of parameters θ_α; and a column vector of measurement noise ϵ_i, one for each observation. Then the whole polynomial regression model can be recast in matrix notation as $y_i = \sum_{\alpha=1}^{p} x_i^\alpha \theta_\alpha + \epsilon_i$ or $y = X\theta + \epsilon$, where $f_p = X\theta$. We then find that $\partial f_p \partial \theta_\alpha = x^\alpha$ and then

$$\frac{\partial \log \mathcal{L}}{\partial \theta_\alpha} = 0 \quad \leftrightarrow \quad [y - X\theta]^T X = 0 \ , \qquad (21)$$

where $(A_{ij})^T = A_{ji}$. One finds:

$$\begin{aligned}
y^T X - \theta^T X^T X = 0 \quad &\leftrightarrow \quad X^T y - X^T X \theta = 0 \\
&\leftrightarrow \quad (X^T X)^{-1} X^T y - (X^T X)^{-1} X^T X \theta = 0 \\
&\leftrightarrow \quad (X^T X)^{-1} X^T y - \theta = 0 \\
&\leftrightarrow \quad \hat{\theta} = (X^T X)^{-1} X^T y \qquad (22)
\end{aligned}$$

where $(A_{ij})^T = A_{ji}$. In general, $(X^T X)^{-1}$ exists if $p < N_{train}$ and the points x_i are all distinct. We can also estimate the measurement variance σ^2 for the polynomial regression from $\frac{\partial}{\partial\sigma^2}\log\mathcal{L}_{train} = 0$, obtaining

$$\hat{\sigma}^2 = \frac{1}{N_{train}}\sum_i [y_i - f_p(x_i,\hat{\theta})]^2 = MSE.$$

Thus, the MSE of the model fit represents the estimate of the measurement error variance.

After we optimized the model parameters $\hat{\theta}$ for the p-th model on the training set, we then estimate the test error, or the cost function, given those parameters, using

$$\mathcal{L}_{test}(p) = \prod_{i=1}^{N_{test}} \mathcal{N}(y_i|f_p(x_i,\hat{\theta}),\hat{\sigma}^2)\ .$$

Among all the different polynomials $f_p(x,\theta)$, which model will give the best prediction for new, unobserved pairs (x_{new}, y_{new})? That's the model with the lowest test error

$$\hat{p} = \mathrm{argmax}_p\, \mathcal{L}_{test}(y|f_p(x|\hat{\theta})\ .$$

3.3. Model selection: Bias and variance

We know consider an explicit example of model selection wherewe generate a dataset from a ground truth model, and then fit a number of alternative models to the data. We will figure out under what circumstances the best fit model recovers the ground truth, and how to understand the origin of the errors in the model fit, as one varies the complexity of the model.

Ground truth. Let us consider a ground truth model

$$\textbf{Ground truth}\qquad y = f_2(x,\theta) = \theta_0 + x\theta_1 + x^2\theta_2 \tag{23}$$

groundtruth

for specific numerical values of the parameters $(\theta_0,\theta_1,\theta_2) = (2, 0.5, 1)$. We assume gaussian measurement noise and generate a dataset $D = \{x_i, y_i\}_{i=1}^N$ according to

$$y_i = f_2(x_i,\theta) + \epsilon_i\ , \tag{24}$$

In a scenario where we are given the dataset D, but we do not know how it was generated, can we infer what model was used to generate it?

In order to do this, we first have to make an assumption and pick a family of parametric models among which to perform our model selection. We will find out whether D was generated by a polynomial $\overset{\texttt{polymodel}}{(20)}$ of degree p and our

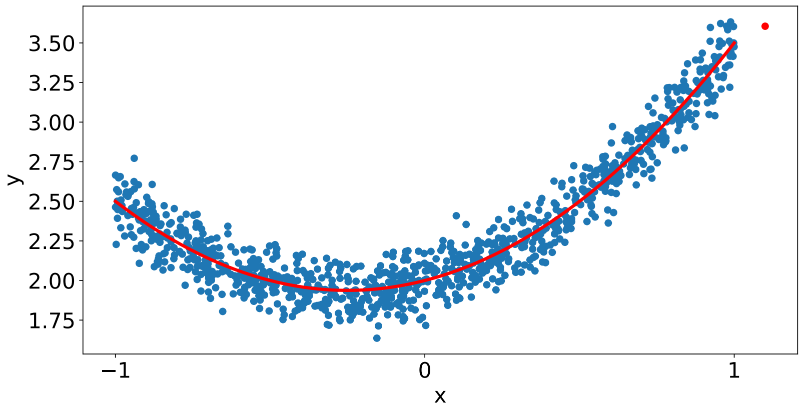

Figure 4: Data generated from the model (groundtruth 23) with $(\theta_0, \theta_1, \theta_2) = (2, 0.5, 1)$ (red line) using measurement noise with $\sigma_\epsilon = 0.11$. Training set (blue dots) and test value (red dot). groundtruthfig

goal is to select which p explains the data more accurately. The most predictive model will be the model with the smallest generalization error, i.e. the smallest $Error^2_{out}$ error on the test set. We will thus perform cross-validation in the three steps outlined above. We will first split our dataset into training and test sets: $D = D_{train} \bigcup D_{test}$. Then we will fit each model f_p to the training set D_{train} to optimize the parameters $\hat{\theta}(p)$. Then we will estimate the generalization error $Error^2_{out}(p)$ of each model on the test set D_{test}. The model with the lowest $Error^2_{out}$ will be our winner: $\hat{p} = \text{argmin}_p \, Error^2_{out}(p)$.

Let us walk through a number of examples to understand how the nature of the error changes as we increase the complexity p of the polynomial. For this exercise, we will consider different training sets $D_1, D_2, \ldots$ generated from the same model (groundtruth 23). The different training sets are generated using different sets of independent variables $\{x_i\}_{i=1}^N$, and the corresponding targets y_i are obtained from different draws of the measurement noise ϵ_i. This scenario closely resembles an experiment where you have multiple measurements $D_1, D_2, \ldots$ of the same quantities $\{x_i\}_{i=1}^N$. For simplicity, we will consider a test set comprising a single point $D_{test} = (x_{test}, y_{test})$.

Constant model $p = 0$. The simplest model is

$$y = f_0(x, \theta) = \theta_0 \ , \tag{25}$$

which just gives the mean value of the data $\hat{\theta}_0 = mean(x)$. Let us consider a first training set D_1 and fit the model to it so we obtain the first estimate of the model $f_0|_{D_1}$ with parameter $\hat{\theta}_0|_{D_1}$, where we explicitly include a dependence on the training set D_1 we used. This is a very simple model, and we find a very large training error:

$$\begin{aligned} Error^2_{in}|_{D_1} &= \mathbb{E}_{D_1}\left[y_i - f_0(x_i, \hat{\theta})\right] \\ &= \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} [y_i - f_0(x_i, \hat{\theta}|_{D_1})]^2 \\ &= \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} [y_i - \hat{\theta}_0|_{D_1})]^2 \ . \end{aligned} \tag{26}$$

Here, we used the notation $\mathbb{E}_{D_1}[g(x_i)]$ to denote the expectation value with respect to $x_i \in D_1$, namely across data points belonging to the first training set D_1. The model prediction for the test data point also contains a large error:

$$Error^2_{test}|_{D_1} = [y_{test} - f_0(x_{test}, \theta|_{D_1})]^2 \ .$$

We emphasized the fact that errors depend on the training set D_1 used for estimating the model parameters. Such a simple model thus gives large training and test errors. However, let us perform the experiment a second time, using a different training set D_2 generated from the same set of x_i, but with different draws of the measurement noise ϵ_i. We optimize to obtain a new estimate of the model parameter $\hat{\theta}_0|_{D_2}$, which we can use to calculate the new training and test errors.

Now we introduce the two crucial quantities to understand model selection: bias and variance. The **bias** is defined as the difference between the true value of the test set and its average model prediction - where the average is done over

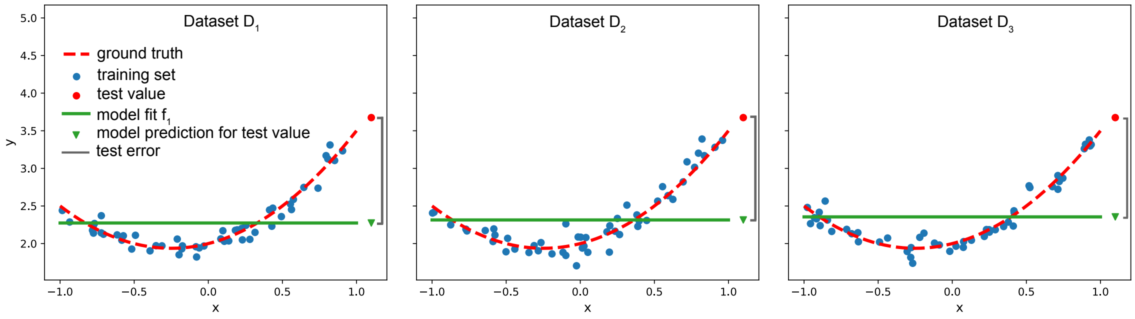

Figure 5: Model f_0 fit (green lines) to three different training sets D_1, D_2, D_3 (blue dots) generated from the same ground truth model as in Fig. groundtruthfig (4). The model has very low complexity: it has only one parameter θ_0 which estimates the mean of the y values in the training set. Each fit gives a different model prediction (green triangle) for the point in the test set (red dot). The test set prediction from each model is off by a different prediction error (square of the grey bar): the model errors are large (high bias), but they are very similar across training sets (low variance). modelfitzero

the predictions obtained from different training sets:

$$\begin{aligned}
\textbf{Bias} &= \text{True value} - \text{Average model prediction} \\
&= y_{test} - \mathbb{E}_{exp}\left[f_0(x_{test}, \hat{\theta}|_{D_s}\right] \\
&= \mathbb{E}_{exp}\left[y_{test} - f_0(x_{test}, \hat{\theta}|_{D_s}\right] \\
&= \frac{1}{n_{exp}} \sum_{s=1}^{n_{exp}} \left[y_{test} - f_0(x_{test}, \hat{\theta}|_{D_s})\right] \; . \qquad (27)
\end{aligned}$$

Here, we introduced the new notation $\mathbb{E}_{exp}\left[g(D_s)\right]$ to denote the expectation value across experiments D_s, where each experiment is represented by a training set. Each prediction $f_0(x_{test}, \hat{\theta}|_{D_s})$ is done using the parameters $\theta|_{D_s}$ optimized on the training set D_s. Remember that $\theta|_{D_s}$ from different training sets are different. As you can see, the bias is very large for this simple model, which does not have enough complexity to capture the structure of the data. In this case, we say that models with large bias are *underfitting* the training set. This large bias is going to give a large MSE on both the training and the test sets.

There is another clear feature emerging from the predictions $f_0(x_{test}, \hat{\theta}|_{D_s})$.

Although the difference between the true and predicted values $y_{test} - f_0(x_{test}, \hat{\theta}|_{D_s})$ is always large, the predicted values $f_0(x_{test}, \hat{\theta}|_{D_s})$ are very similar across the different experiments. In other words, the variance of the prediction is very low. Let us formalize this observation and consider the predictions $f_0(x_{test}, \hat{\theta}|_{D_s})$ we obtained when using models whose parameters $\theta|_{D_s}$ were fit on different training set D_s. The **variance** of these predictions across training sets is:

$$\begin{aligned}
\textbf{Variance} &= \text{Variance of model predictions across different training sets} \\
&= \text{var}_{exp}\left[f_0(x_{test}, \hat{\theta}|_{D_s})\right] \\
&= \frac{1}{n_{exp}-1}\sum_{s=1}^{n_{exp}}\left[\mathbb{E}_{exp}[f_0(x_{test}, \hat{\theta}|_{D_s})] - f_0(x_{test}, \hat{\theta}\||_{D_s})\right]^2 \quad , \quad (28)
\end{aligned}$$

where $\mathbb{E}_{exp}[f_0(x_{test}, \hat{\theta}|_{D_s})] = \frac{1}{n_{exp}}\sum_{s=1}^{n_{exp}} f_0(x_{test}, \hat{\theta}|_{D_s})$ is the average prediction across datasets. Note that this quantity does not depend at all on the actual error the models are making in predicting the true value of the test data point, but it only concerns the comparison between the different models. What we find is that our simple model has a very low variance: the model predictions are very similar to each other, because the parameters estimated $\hat{\theta}_0|_{D_s}$ from different training sets D_s are very similar to each other. We thus arrived at the following crucial insight:

Simple models have high bias and low variance.

Polynomial regression model of degree $p = 7$.

On the other hand of the spectrum, let us consider now a very complex model f_7 where we fit a degree 7 polynomial to the data (Fig. modelfitseven 6). The model fit to each training set D_s has a much smaller training error $Error^2_{in}$, because the model tries to wiggle around all the training data points - namely, it is fitting the noise in the training set, a phenomenon called *overfitting*. Fitting the model to different training sets D_s gives completely different results for the test prediction each time. This is evident from the fact that the model predictions for the test data point are completely different from each other: this highly complex model has a very high variance! This contributes a very large MSE on the test set. The reason for this is that complex models tend to *overfit* the noise in the training set: the cyan curve in (Fig. modelfitseven 6) attempts to pass through the ripples in the data which are due to the measurement noise, but the model treats those as signals. Because of this, the MSE of each model on its training set is very small. On the other hand, because the errors on the prediction of the test data point have either sign (namely, they are very different across experiments), in the end they mostly cancel out when taking the average over different experiments D_s, leaving a small bias overall. We thus the second crucial insight:

Complex models have high variance and low bias.

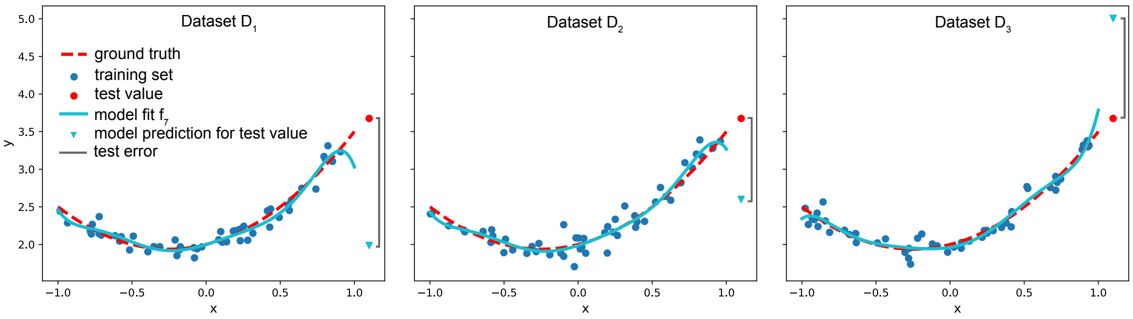

Figure 6: Model f_7 fit (purple lines) to three training sets D_1, D_2, D_3 (blue dots, same as in Fig. (modelfitzero 5)). The model has very high complexity: it is a polynomial regression of degree 7. Each fit gives a very different model prediction (cyan triangle) for the point in the test set (red dot). The model errors are very different across training sets (high variance). modelfitseven

Polynomial regression model of degree $p = 2$.

What is emerging from our exercise so far is that a model that is too simple has a large MSE on the test set, that is driven mostly by its large bias. A model that is too complex also has a large MSE on the test set, driven mostly by the large variance across different training sets. What about an intermediate model? Let us fit a quadratic polynomial f_2 to the data (Fig. modelfittwo 7). In this case, we see that the model still has a bias: the difference between true and predicted value of the test data point has the same sign and similar amount across experiments. On the other hand, the model prediction is very consistent across experiments: it has small variance. It thus seems that models with intermediate complexity have an overall small MSE on the test set, because although they both have bias and variance, their sum is small and contributes a small test MSE in total.

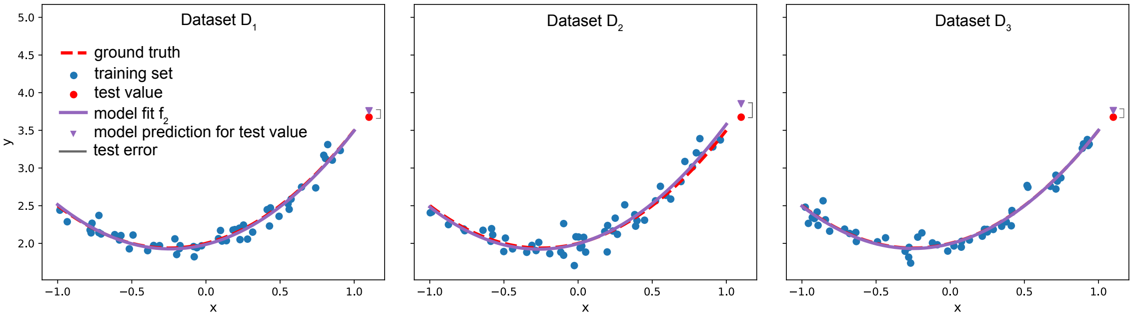

Figure 7: Model f_2 fit (purple lines) to three training sets D_1, D_2, D_3 (blue dots, same as in Fig. (modelfitzero 5)). The model has intermediate complexity: it is a quadratic regression (degree 2). Each fit gives a very different model prediction (purple triangle) for the point in the test set (red dot). The model errors are very different across training sets (high variance). modelfittwo

3.4. Bias-variance tradeoff

In Fig. tradeoffdistribution 8 one can see the bias and variance (across D_s experiments for $s = 1, \ldots, n_{exp} = 100$) of different polynomial regression models with degree p to the same data generated from the ground truth model in Fig. groundtruthfig 4. Models with low complexity make predictions which have a very large bias: the average prediction is far from the true value of the test set. However, the variance of model predictions is low since all predictions are similar to each other (the histogram has a small width). Models of increasing complexity have low bias, namely, the average prediction is close to the true value of the test set. However, model predictions vary wildly across different training sets, so the complex models have very large variance. Intermediate models have small bias and variance, and give the most accurate predictions for the test set. The total test error $Error^2_{out}$ is thus minimized for intermediate complexity and the model with the smallest test error is in fact $p = 2$. In fact, we found that the data was generated from a ground truth quadratic polynomial, which we recovered as the best model class f_2 to describe the data.

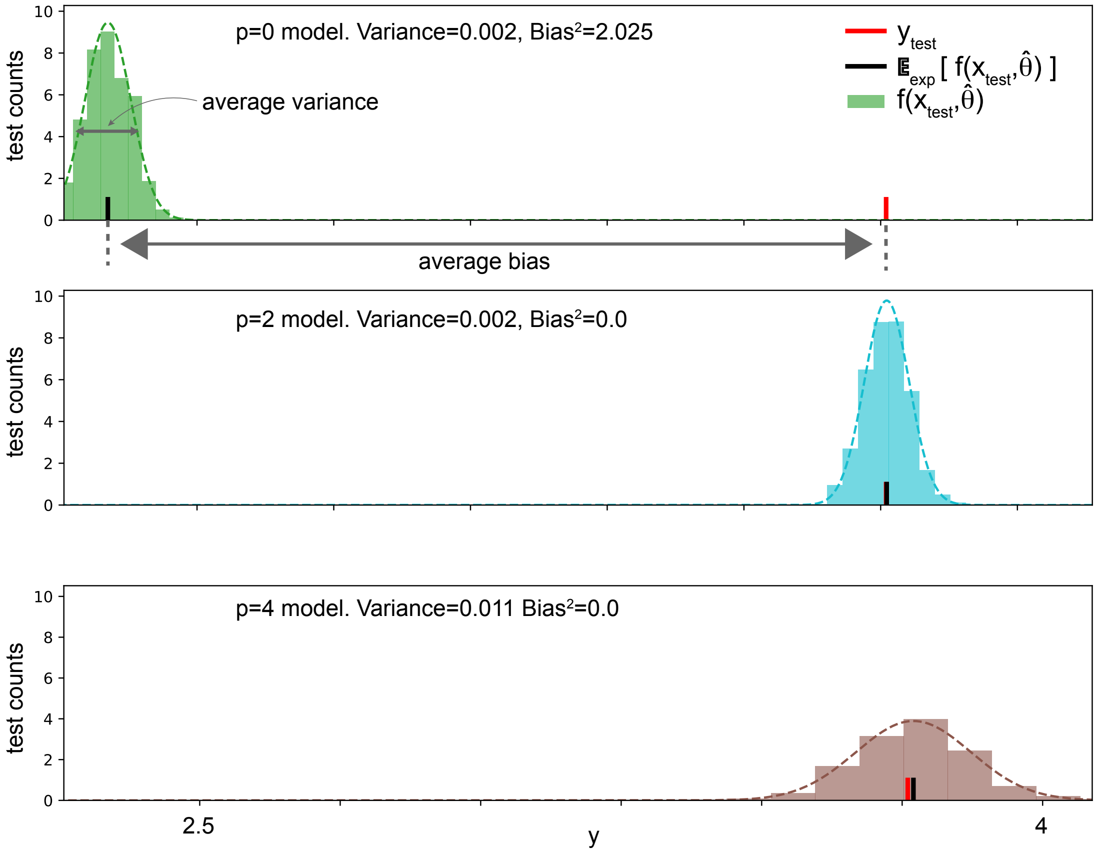

Figure 8: Different models (f_0, f_2, f_4, top to bottom) optimized on $n_{exp} = 100$ different training sets gives a distribution of $n_{exp} = 100$ model predictions for the test value (x_{test}, y_{test}). The difference between the average test prediction (black bar) and the true test value (red bar) is the bias, while the variance of the distribution of model predictions is the variance. Models of increasing complexity have larger variance and smaller bias.

tradeoffdistribution

The general result of the bias-variance tradeoff is that the average squared test error is the sum of three terms: the bias squared, the variance, and the irreducible measurement error:

Bias-variance theorem: Error^2_{test}=Bias^2+Variance+Measurement noise.

theoreomstatement

As one increases the complexity p of the model (the degree of the polynomial regression in our example), the following phenomena occur:

- The training error decreases: more complex models tend to *overfit* the training sets.
- The variance increases: more complex models overfit the training sets and thus give very different predictions for the test values across different experiments.
- The bias decreases: simpler models struggle to capture the structure in the data and tend to *underfit* the training sets.
- The goldilock zone of intermediate complexity give models with small bias and variance, yielding the best generalization ability with the smallest test error.

This general tradeoff is described in Fig. tradeoffsummary 9 for the example of polynomial regression we just worked out. We will now proceed to give a proof of the bias-variance tradeoff theorem.

3.5. Proof of the bias-variance tradeoff

Let us formalize our framework. We are given a training set $D = \{x_i, y_i\}_{i=1}^N$ generated as $y_i = f(x_i) + \epsilon_i$ with $\epsilon_i \sim \mathcal{N}(0, \sigma_\epsilon^2)$, and our ground truth values $y = f(x)$. We fit a model $g(x, \theta)$ by optimizing the cost function $C[y, g(x, \theta) = \sum_i [y_i - g(x_i, \theta)]^2$ yielding the parameters $\hat{\theta}|_D = \text{argmin}_\theta \, C$. Note that $\hat{\theta}|_D$ is a random variable, function of the training set D. Note also that in our notations, the model g is different from the function f used to generate the data.

Now suppose we have many training sets (or experiments) D_s, $s = 1, \ldots, n_{exp}$, each comprising different values of (x_i, y_i). Each training set gives a different

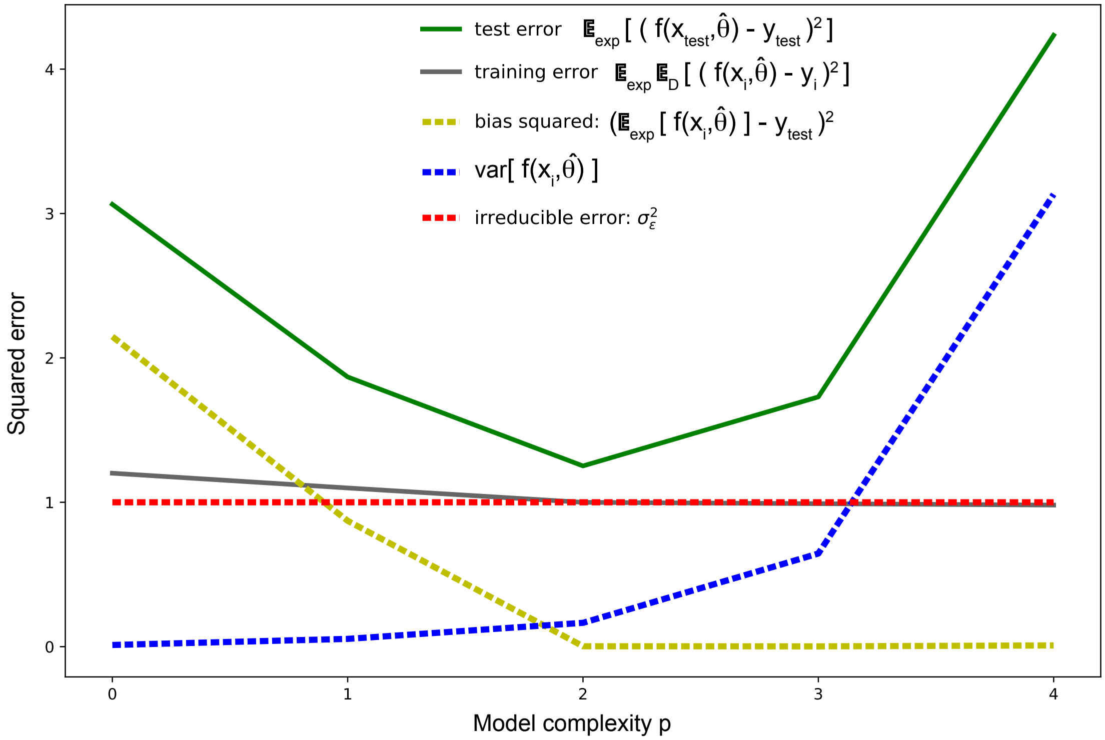

Figure 9: Bias-variance tradeoff for the polynomial regression yields the best model at degree $p = 2$.

tradeoffsummary

set of optimized model parameters $\hat{\theta}|_{D_s} \equiv \hat{\theta}_s$ for short. We can then consider the expectation over the different datasets

$$\mathbb{E}_{exp}[\hat{\theta}] = \frac{1}{n_{exp}} \sum_{s=1}^{n_{exp}} \hat{\theta}|_{D_s} \ .$$

Let us consider a test set comprising a single data point $x_{test} \equiv x$ and its target $y_{test} \equiv y = f(x) + \epsilon$, and make a prediction $f(x, \hat{\theta}_s)$ given the parameters optimized on the training set D_s. We will obtain different predictions depending on the training set used to optimize our model parameters. We can then consider the average prediction over all training sets:

$$\mathbb{E}_{exp}[f(x, \hat{\theta})] = \frac{1}{n_{exp}} \sum_{s=1}^{n_{exp}} f(x, \hat{\theta}_s) \ .$$

At this point we can define the bias and variance as follows:

$$\begin{aligned}\mathrm{bias}[f(x,\hat{\theta})] &= \mathbb{E}_{exp}[f(x,\hat{\theta})] - \underbrace{y}_{\text{ground truth}} && (29)\\ \mathrm{var}[f(x,\hat{\theta})] &= \mathbb{E}_{exp}\left[f(x,\hat{\theta}) - \mathbb{E}_{exp}[f(x,\hat{\theta})]\right]^2 && (30)\end{aligned}$$

Our goal is to prove the following decomposition for the average test error:

$$Theorem: \qquad \mathbb{E}_{exp}\left[(y - f(x,\hat{\theta}))^2\right] = \mathrm{bias}^2 + \mathrm{var} + \sigma_\epsilon^2 \ . \qquad (31)$$

Note that there are two sources of variability in our problem:

- Each training set D_s has its own set of x_i's.
- For each training set D_s, we can run different experiments each time drawing a different realization of the measurement noise

  $$y_i = f(x_i) + \epsilon_i \ ,$$

  $$y_i' = f(x_i) + \epsilon_i' \ ,$$

  where $\mathbb{E}_\epsilon[\epsilon] = 0$ and $\mathbb{E}_\epsilon[\epsilon^2] = \sigma_\epsilon^2$.

The model parameters for a given training set can thus be averaged over noise realizations: $\mathbb{E}_\epsilon[\hat{\theta}_s]$. We will thus calculated a slightly different expression of the generalization error (generror 31), averaged over both experiments and noise realization for fixed experiment:

$$\begin{aligned}\mathbb{E}_{exp,\epsilon}\left[(y - f(x,\hat{\theta}))^2\right] &= \mathbb{E}_{exp,\epsilon}\left[(y - f(x) + f(x) - f(x,\hat{\theta}))^2\right]\\ &= \overbrace{\mathbb{E}_{exp,\epsilon}\left[\underbrace{(y - f(x))^2}_{\epsilon^2}\right]}^{\sigma_\epsilon^2} + \mathbb{E}_{exp,\epsilon}\left[(f(x) - f(x,\hat{\theta}))^2\right] + \\ &\quad 2\mathbb{E}_{exp,\epsilon}\left[(y - f(x))(f(x) - f(x,\hat{\theta}))\right] \ , && (32)\end{aligned}$$

where we used the fact that $y - f(x) = \epsilon$. The cross-term in the last line reads:

$$\begin{aligned}
\mathbb{E}_{exp,\epsilon}\left[(y - f(x))(f(x) - f(x,\hat{\theta}))\right] &= \mathbb{E}_{exp,\epsilon}\left[\epsilon\ \underbrace{(f(x) - f(x,\hat{\theta}))}_{\text{does not depend on}\,\epsilon}\right] \ , \\
&= \mathbb{E}_{exp}\left[\underbrace{\mathbb{E}_{\epsilon}[\epsilon]}_{=0}(f(x) - f(x,\hat{\theta}))\right] = 0 \ ,
\end{aligned}$$

where we used the fact that the term $f(x) - f(x,\hat{\theta})$ does not depend on the measurement noise so we can factor it out of the $\mathbb{E}_{\epsilon}$. The average generalization error thus reduces to

$$\begin{aligned}
\mathbb{E}_{exp,\epsilon}\left[(y - f(x,\hat{\theta}))^2\right] &= \sigma_{\epsilon}^2 + \mathbb{E}_{exp,\epsilon}\left[(f(x) - f(x,\hat{\theta}))^2\right] \\
&= \sigma_{\epsilon}^2 + \underbrace{\mathbb{E}_{exp}\left[(f(x) - f(x,\hat{\theta}))^2\right]}_{\equiv A}
\end{aligned} \tag{33}$$

where we used the fact that $f(x) - f(x,\hat{\theta})$ does not depend on ϵ, so we dropped the expectation over the noise from the last term. Let us now focus on calculating this last term A by adding and subtracting the term $\mathbb{E}_{exp}\left[f(x,\hat{\theta})\right]$:

$$\begin{aligned}
A &= \mathbb{E}_{exp}\left[\left(f(x) - \mathbb{E}_{exp}\left[f(x,\hat{\theta})\right] + \mathbb{E}_{exp}\left[f(x,\hat{\theta})\right] - f(x,\hat{\theta})\right)^2\right] \\
&= \mathbb{E}_{exp}\left[\left(f(x) - \mathbb{E}_{exp}\left[f(x,\hat{\theta})\right]\right)^2\right] + \mathbb{E}_{exp}\left[\left(\mathbb{E}_{exp}\left[f(x,\hat{\theta})\right] - f(x,\hat{\theta})\right)^2\right] \\
&\quad +2\mathbb{E}_{exp}\left[\left(f(x) - \mathbb{E}_{exp}\left[f(x,\hat{\theta})\right]\right)\left(\mathbb{E}_{exp}\left[f(x,\hat{\theta})\right] - f(x,\hat{\theta})\right)\right]
\end{aligned}$$

Let us consider the term in the last line: because $f(x) - \mathbb{E}_{exp}\left[f(x,\hat{\theta})\right]$ does not depend on the training sets anymore (it is an average over training sets) we can pass it through the outermost $\mathbb{E}_{exp}$ and find

$$2\left(f(x) - \mathbb{E}_{exp}\left[f(x,\hat{\theta})\right]\right)\mathbb{E}_{exp}\left[\left(\mathbb{E}_{exp}\left[f(x,\hat{\theta})\right] - f(x,\hat{\theta})\right)\right] \ ,$$

and now the second factor yields

$$\mathbb{E}_{exp}\left[\left(\mathbb{E}_{exp}\left[f(x,\hat{\theta})\right] - f(x,\hat{\theta})\right)\right] = \mathbb{E}_{exp}\left[f(x,\hat{\theta})\right] - \mathbb{E}_{exp}\left[f(x,\hat{\theta})\right] = 0 \ .$$

So the cross-term vanished and we obtain

$$A = \mathbb{E}_{exp}\left[\left(f(x)-\mathbb{E}_{exp}\left[f(x,\hat{\theta})\right]\right)^2\right]+\mathbb{E}_{exp}\left[\left(\mathbb{E}_{exp}\left[f(x,\hat{\theta})\right]-f(x,\hat{\theta})\right)^2\right] \tag{34}$$

Now, consider the first term in A above:

$$\mathbb{E}_{exp}\left[\left(f(x)-\mathbb{E}_{exp}\left[f(x,\hat{\theta})\right]\right)^2\right]=\left(f(x)-\mathbb{E}_{exp}\left[f(x,\hat{\theta})\right]\right)^2=\text{bias}^2\ ,$$

where we used the fact that neither $f(x)$ nor $\mathbb{E}_{exp}\left[f(x,\hat{\theta})\right]$ depend on the training sets, so we can drop the outermost $\mathbb{E}_{exp}$ and realize that it is the bias squared (biasdef 29). The second term in A is just the variance (vardef 30). So we find that $A=\text{bias}^2+\text{var}$ and, collecting all terms together obtain the expression for the average generalization error:

$$\begin{aligned}\mathbb{E}_{exp,\epsilon}\left[(y-f(x,\hat{\theta}))^2\right] &= \sigma_\epsilon^2+\underbrace{\left(f(x)-\mathbb{E}_{exp}\left[f(x,\hat{\theta})\right]\right)^2}_{\text{bias}^2}+\underbrace{\mathbb{E}_{exp}\left[\left(\mathbb{E}_{exp}\left[f(x,\hat{\theta})\right]-f(x,\hat{\theta})\right)^2\right]}_{\text{var}} \\ &= \sigma_\epsilon^2+\text{bias}^2+\text{var}\ , \end{aligned} \tag{35}$$

which proves the bias-variance decomposition theorem (generror 31), albeit a slightly different version of it where we also averaged over multiple realization of the measurement noise for each training set. This explains the experimental results we found in Fig. (tradeoffsummary 9).

Guide to the literature

The bias-variance tradeoff is covered in the review *A high bias, low variance introduction to ML for physicists.* mehta2019high [5], Chapters II and III, and in Chapter 3.2 of Bishop's book bishop [4]. Our proof of the bias-variance decomposition theorem is adapted from these references. The example of model selection using polynomial regression is adapted from the *Towards data science* blog post towardsbiasvar [6].

Learning

4. Supervised learning

supervisedlearning

There are many ways to characterize the perceptron, its update rule, its learning algorithm. The way we will present it is going to be just an example among the various possibilities one can find in the literature. We will consider the perceptron as a very rough approximation to an actual neuron, the same way physicists regard a sphere as a first order approximation of a horse. The specific learning rule we will adopt allows us to interpret the perceptron as a machine that performs statistical inference, in the very precise sense we defined in Chapter section:inference 2, according to the principles of information theory.

4.1. Perceptron

Our goal is to start from a simple model neuron and gradually move towards something that looks a bit more realistic, step by step. The first step is to separate the input patterns ξ_i from the wanted output patterns ζ_i, that we will call **targets**.

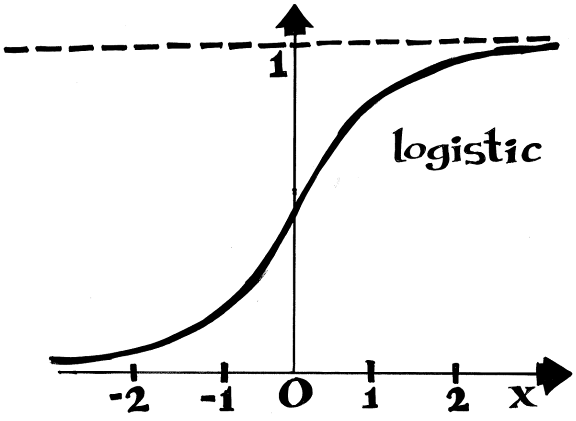

Figure 10: The sigmoid function in equation (sigmoid 36), a.k.a. the logistic function.

logistic

We will consider an architecture called *feedforward* network. Let us start from the simplest example of a feedforward network: the *single-layer perceptron*. Consider the system in Fig. perceptronfig 11 that has the following architecture. We have a set of inputs ξ_i, for $i = 1, \ldots, N$, that may be either external stimuli, or outputs

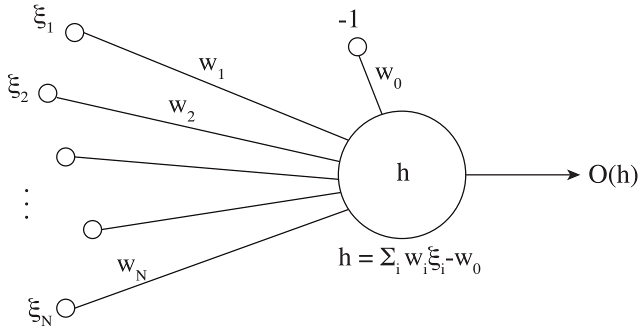

Figure 11: Architecture of a single layer perceptron with N inputs ξ_i, weights w_i, activation function $\mathcal{O}$ and threshold w_0.

perceptronfig

coming from other networks. All these inputs feed into a single computational unit, our model neuron, through a set of weights w_i. The activation of the neuron (the post synaptic potential) is

$$h = \sum_{j=1}^{N} w_j \xi_j \ .$$

The neuron processes the information and computes its output $\mathcal{O}$ as usual

$$\mathcal{O} = f(h) \ ,$$

where the *activity* $f(h)$ of the neuron may take different forms, depending on the problem at hand. The most common activities found in the literature are deterministic (first four) or stochastic (last one)

1. Linear:
$$f(h) = h \ ,$$
2. Rectified linear:
$$f(h) = \begin{cases} 0 & \text{for}\, h < 0 \\ h & \text{for}\, h \geq 0. \end{cases}$$
3. Sigmoid:
$$f(h) = \frac{1}{1+e^{-h}} \ , \tag{36}$$
sigmoid
4. Sign:
$$f(h) = \mathrm{sgn}\,(h) \ ,$$

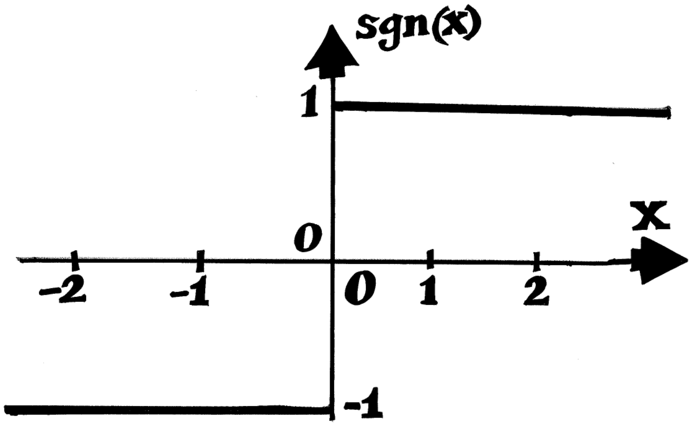

Figure 12: The function sgn (x). sgnx

5. Threshold:

$$f(h) = \begin{cases} 0 & \text{for}\, h < 0 \\ 1 & \text{for}\, h \geq 0. \end{cases} \tag{37}$$

perthreshold

6. Stochastic:

$$f(h) = \begin{cases} 1 & \text{with probability}\ \frac{1}{1+e^{-h}} \\ -1 & \text{otherwise.} \end{cases}$$

For the time being we will stick to the "zero noise" deterministic activity rules and see which operations can be performed by our perceptron. We will consider the stochastic rule later in the context of Boltzmann machines.

andorxor *4.1.1. Examples: AND, OR & XOR*

To get familiar with the device, we will first solve two simple examples. Consider a single-layer perceptron with two inputs and one output unit. We are given a fixed set of two inputs $\{\xi_1, \xi_2\}$ and one target ζ, that we collect in the *truth table*. Our task is to find a set of weights $\{w_1, w_2, w_0\}$ such that

$$\zeta^\mu = \text{sgn}\,(\sum_{i=1,2} w_i \xi_i^\mu - w_0)\ , \qquad \mu = 1, \ldots, p\ , \tag{38}$$

perceptron

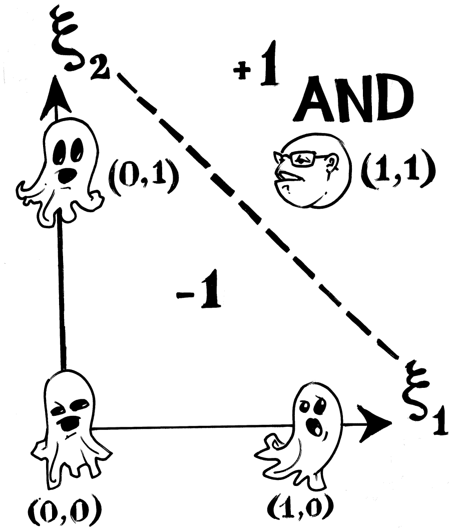

Figure 13: AND function. The inputs are linearly separable and the perceptron can perform the task. Ghosts and Pacmans belong respectively to the -1 and $+1$ class as in the truth table. The dashed line represents Eq.(40).

ANDpic

where we introduced the threshold w_0.[6] In our simple examples we are going to take just binary inputs $\xi = \{0, 1\}$ and four possible targets $p = 4$.

The truth tables for the AND, OR and XOR functions are the following

truthtables

AND function		
INPUT		TARGET
ξ_1	ξ_2	ζ
0	0	-1
0	1	-1
1	0	-1
1	1	+1

OR function		
INPUT		TARGET
ξ_1	ξ_2	ζ
0	0	-1
0	1	+1
1	0	+1
1	1	+1

XOR function		
INPUT		TARGET
ξ_1	ξ_2	ζ
0	0	-1
0	1	+1
1	0	+1
1	1	-1

Let us consider the AND function in the first table. The output is positive only if both inputs are activated. If we solve (38) for w_i, given the patterns in

[6] Sometimes the presence of the threshold w_0 is made implicit by introducing an extra input with a fixed value $\xi_0 = -1$.

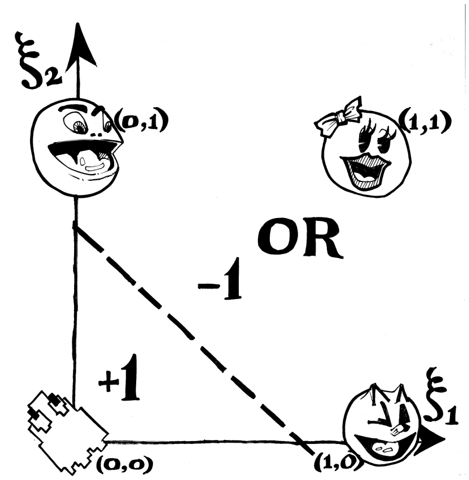

Figure 14: OR function. The inputs are linearly separable and the perceptron can perform the task. Ghosts and Pacmans belong respectively to the -1 and $+1$ class as in the truth table. The dashed line represents the equation $3\xi_2 = -3\xi_1 + 2$. ORpic

the truth table, we find the following conditions

$$\begin{cases} -w_0 < 0 \\ w_2 - w_0 < 0 \\ w_1 - w_0 < 0 \\ w_1 + w_2 - w_0 > 0 \end{cases} \Rightarrow \begin{cases} w_0 > 0 \\ w_1 + w_2 > 0 \\ w_{1,2} < w_0 \end{cases}$$

We can immediately find a possible solution:

$$w_1 = w_2 = 1 \ , \qquad w_0 = 1.5 \ . \tag{39}$$ andw

Note that if we rescale the weights by an overall constant, we still obtain a valid solution, since the only thing that matters is the sign of the activation, not its actual value. Now let us make a picture of the AND function in the ξ-plane. It is easy to see that the white dots are separated from the black dot by the line defined by the equation

$$\sum_{i=1,2} w_i \xi_i + w_0 = 0 \ , \tag{40}$$ separable

or in other words $2\xi_2 = -2\xi_1 + 3$.

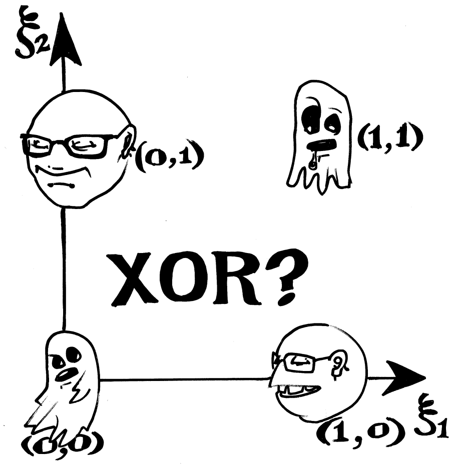

Figure 15: XOR function. The inputs are *not* linearly separable and the perceptron cannot perform the task. Ghosts and Pacmans belong respectively to the -1 and $+1$ class as in the truth table. XORnotpic

Let us solve for the OR function now, in the second truth table. The output is positive if at least one of the inputs is activated. If we solve (perceptron 38) for w_i, given the patterns in the truth table, we find the following conditions

$$\begin{cases} -w_0 < 0 \\ w_2 - w_0 > 0 \\ w_1 - w_0 > 0 \\ w_1 + w_2 - w_0 > 0 \end{cases} \Rightarrow \begin{cases} w_0 > 0 \\ w_{1,2} > w_0 \end{cases}$$

We can immediately find a possible solution:

$$w_1 = w_2 = 1 \ , \qquad w_0 = 0.5 \ . \tag{41}$$

orw

Now let us make a picture of the OR function in the ξ-plane. The separating line this time is defined by the equation (separable 40) which now reads $3\xi_2 = -3\xi_1 + 2$.

Consider now the XOR function in the third truth table. The output is positive if only one of the two inputs is activated, but not both. If we try to solve (perceptron 38) for w_i, given the patterns in the truth table, we find the following conditions

$$\begin{cases} -w_0 < 0 \\ w_2 - w_0 > 0 \\ w_1 - w_0 > 0 \\ w_1 + w_2 - w_0 < 0 \end{cases} \Rightarrow \begin{cases} w_0 > 0 \\ w_{1,2} > w_0 \\ w_1 + w_2 < w_0 \end{cases} \quad ??$$

which, low and behold, does not admit solutions! We have found the interesting result that the perceptron cannot implement the XOR function. Let us draw the picture of the XOR function in the ξ-plane and understand why our new toy is broken.

What is the difference between the AND and OR functions as opposed to the XOR? It's easier to just look at the pictures. In the case of AND and OR, we can draw a line that separates all the black dots from all the white dots. However, it is obviously impossible to draw such a line in the case of the XOR function. This is the final judgement on the single-layer perceptron. It is a **classifier** that can classify all your input patterns according to their category YES/NO, but only in the case in which the inputs are **linearly separable**, like in the AND and OR functions. Linearly separable obviously means that you can separate the various patterns with a line. If the inputs are *not* linearly separable, like in the XOR function, then the perceptron *fails*. We will first find out how we can teach the perceptron to solve the tasks that it is able to solve, and then generalize it to a more powerful device that can solve the XOR function, too.

4.1.2. Gradient descent learning

We have found that, in the case of the linearly separable data in the AND and OR tasks in Section andorxor 4.1.1, there is a solution for the weights w_i that classifies the inputs correctly. The question we want to address now is the following. Suppose we start with zero weights. Is there an algorithm that allows the network to learn the classification task? Namely, can we give a procedure to change the weights incrementally to a point where they correctly classify the data? This procedure is our operational definition of "learning."

We will look at a learning algorithm in which we know the correct target ζ that the output $\mathcal{O}$ of a neuron must reproduce. This way of learning is called "supervised" or "learning with a teacher," since we are given the correct value of the output. If we start with initial synaptic weights set to zero, we want to find an algorithm that minimizes the difference between the wanted target ζ and the actual output $\mathcal{O}$. Consider the model of the perceptron ($\overset{\texttt{sigmoid}}{36}$) with inputs ξ_i, for $i = 1, \ldots, N$ and activity rule

$$f(h) = \frac{1}{1+e^{-h}} \ ,$$

where $h = \sum_{i=1}^{N} w_i \xi_i$ and we changed the output from $\{\pm 1\}$ to the interval $\{0, 1\}$. The activity rule describes the probability that the input ξ_i belongs to class $+1$ rather than 0. Our learning task consists of a training set D made of p input-target pairs $\{\xi_i^\mu, \zeta^\mu\}_{\mu=1}^{p}$ that we want to learn. Our first learning algorithm, called the *delta rule*, consists of the following steps

1. Compute the activation in response to a given input ξ_i^μ

$$h^\mu = \sum_{j=1}^{N} w_j \xi_j^\mu \ .$$

2. Evaluate the output of the unit

$$\mathcal{O}^\mu = \frac{1}{1+e^{-h^\mu}} \ . \qquad (42) \qquad \boxed{\texttt{activityrule}}$$

3. The teacher gives you the target ζ^μ which is the *correct* value that the output should have, given such input. The *error signal* is

$$\delta^\mu = \zeta^\mu - \mathcal{O}^\mu \ .$$

Adjust the weights w_i so as to reduce the error

$$\Delta w_i = \eta \delta^\mu \xi_i \ , \qquad (43) \qquad \boxed{\texttt{deltaw}}$$

where η is called the learning rate. This learning rule is called the **delta rule**.

4. Repeat the procedure for the whole training set D. Possibly use the whole training set multiple times.

We called this algorithm the "on-line" gradient descent because at each step in the training we pause and update the weights. Alternatively, one can use a "batch" algorithm in which we only update the weights at the end of each full training cycle

$$\Delta w_i = \sum_{\mu=1}^{p} \eta \delta^\mu \xi_i^\mu \ , \tag{44}$$

perceptronlearningG

where $\delta^\mu = \zeta^\mu - \mathcal{O}^\mu$.

Does the perceptron learning rule (deltaw 43) converge, so that the network learns how to reproduce the wanted targets? It can be shown that the perceptron can learn the classification task in a finite number of steps, but only in the case in which the input patterns are separable, of course, otherwise there is no solution to the classification task in the first place.

Our **supervised learning** procedure minimizes the following *error function*

$$G(w) = -\sum_{\mu=1}^{p} [\zeta^\mu \ln \mathcal{O}(\xi^\mu, w) + (1-\zeta^\mu)\ln(1-\mathcal{O}(\xi^\mu, w)] \ . \tag{45}$$

errorfunction

We can identify each term as the *relative entropy* (KL 16) of the distribution of the targets $(\zeta^\mu, 1-\zeta^\mu)$ with respect to the actual outputs of the neuron $(\mathcal{O}^\mu, 1-\mathcal{O}^\mu)$.[7] The error function is always positive and zero only when the outputs coincides with the targets (at which point we learned the lesson). If we use the sigmoid function (sigmoid 36) as activation rule and differentiate the error function (errorfunction 45) with respect to the synaptic weights we find[8]

$$\frac{\partial G}{\partial w_i} = -\sum_{\mu=1}^{p} (\zeta^\mu - \mathcal{O}^\mu)\xi_i \ . \tag{46}$$

gradientG

[7] In the expression (KL 16), consider the distribution of the empirical data $P(x) = (\zeta, 1-\zeta)$ and the distribution of outputs of our neuron $Q(x) = (\mathcal{O}, 1-\mathcal{O})$. One finds that $D_{KL}(P||Q) = G(w) + \zeta \ln \zeta + (1-\zeta)\ln(1-\zeta)$. This is equal to (errorfunction 45) up to the last two constant terms, that do not depend on the weights. Since we are only interested in minimizing these functions, the result is the same.

[8] When using the sigmoid function as activation there is a crucial simplification. Denoting $\mathcal{O}(h) = 1/(1+e^{-h})$, we find $\partial_h \mathcal{O} = \mathcal{O}(1-\mathcal{O})$. This term cancels the analogous term in the denominator of (gradientG 46), leading to this simple result.

The quantity $\zeta^\mu - \mathcal{O}^\mu$ appearing in the gradient is the *error* on the training session μ: the difference between output and target. We found the batch version perceptronlearningG (44) of the learning rule.

One can take other expressions for the cost function, depending on the features of the algorithm one wants to implement. Another very common error function is simply

$$G(w) = \frac{1}{2}\sum_{\mu=1}^{p}[\zeta^\mu - \mathcal{O}(\xi^\mu, w)]^2 \ ,$$

which gives the update rule deltaw (43) with a different error

$$\delta^\mu = f'(h^\mu)(\zeta^\mu - \mathcal{O}^\mu) \ .$$

Note the presence of the derivative of the activity function, which drops out of the expression when using the error function errorfunction (45).

The gradient descent procedure suffers from a serious problem. Who is going to tell us when it is time to stop? Namely, we can repeat the training cycles as many times as we want and the weights are going to become very large in absolute value. This is clearly a problem of *overfitting.* A first guess would be to stop early with the training session, to avoid the parameters to grow too large. But this sounds a bit of a subjective procedure. A better way to solve this problem is to add a "mass term" to the error function errorfunction (45), whose effect is to discourage large values of the weights

$$M(w) = G(w) + \alpha E(w) \ , \qquad (47) \qquad \boxed{\texttt{newerror}}$$

where the new term is a regulator that we can choose for instance as

$$E(w) = \frac{1}{2}\sum_i w_i^2 \ . \qquad (48) \qquad \boxed{\texttt{eofw}}$$

The quantity α is called the weight decay rate. The new on-line learning rule obtained from a descent of the error function is

$$\Delta w_i = \eta \delta^\mu \xi_i^\mu - \alpha w_i \ . \qquad (49) \qquad \boxed{\texttt{learningrule}}$$

A comment on the capacity of the perceptron, defined as the ratio of the number of patterns or memories we want to store, divided by the number of input neurons. One can show that the perceptron capacity equals two, namely we can store twice as many patterns as the number of input neurons.

4.1.3. Learning as inference

What does our brain do, when it learns something? It is inferring some facts about the world around us. We would like to rephrase the learning procedure we just introduced as an inference task. This will enlighten the connection between network of neurons and our earlier detour on probability theory.

Consider our single-perceptron architecture with activity rule (activityrule 42). We plug in some inputs ξ_i, the neuron looks at them and spits out a number $\mathcal{O}$. This is a number between zero and one. It is natural to interpret it as the probability that the input ξ_i belongs to the class $\zeta = 1$, given the particular set of weights w_i:

$$\begin{aligned} P(\zeta = 1|w, \xi_i) = & \quad \mathcal{O} \ , \\ P(\zeta = 0|w, \xi_i) = & \quad 1 - \mathcal{O} \ . \end{aligned} \tag{50}$$

We can recast these two equations into a single one

$$P(\zeta|w, \xi_i) = \mathcal{O}^{\zeta}(1 - \mathcal{O})^{1-\zeta} \ .$$

The right hand side is nothing else than minus the exponential of the error function (errorfunction 45), so we conclude that

$$P(\zeta|w, \xi) = e^{-G(w)} \ . \tag{51}$$

likeliw

If we define our data to be the targets ζ, and the inputs ξ_i to be some fixed given quantity (our hypothesis $\mathcal{H}$), then the parameters we want to estimate with our inference task are the weights w_i. The expression in (likeliw 51) is then the *likelihood* that the parameter w describe the data ζ. The relative entropy $G(w)$ is then interpreted as the log likelihood. Confident in our starting point, let us try to interpret the whole supervised learning algorithm as an inference problem, according to formula (posteriorprob 2).

We would like to find the probability distribution of the weights w_i, given our classification task with data ζ and fixed inputs. We have to make an assumption on the *prior* distribution of the weights. Let us assume that the weights have a Gaussian prior ditribution

$$P(w) = e^{-\frac{\alpha}{2}\sum_i w_i^2}/Z(\alpha) = e^{-\alpha E(w)}/Z_\alpha \ ,$$

where the normalization constant is $Z_\alpha = (2\pi/\alpha)^{N/2}$ and $E(w)$ was introduced in (48). The prior of the weights is just a Gaussian with zero mean and variance $\sigma_w^2 = 1/\alpha$. But this is just the weight decay term in the cost function!

All the pieces are in place now to use Bayes' theorem. The total cost function (47) that governs the learning process gives the *inference* of the parameters w, given the data

$$\begin{aligned} P(w|\zeta) &= \tfrac{P(\zeta|w)P(w)}{P(D)} \\ &= e^{-M(w)}/Z_M \ , \end{aligned} \tag{52}$$

where the normalization constant $Z_M = Z_\alpha P(D)$ does not depend on the weights. The learning algorithm in (49), which is found by minimizing the cost function $M(w)$ using gradient descent, computes the *most probable* value of the weights w_i, a.k.a. *maximum a posteriori*, which we will denote as w_i^{MAP}. Note that the cost function in (52) describes log probabilities. That's very convenient given the fact that, while costs and errors sum, probabilities multiply.

4.1.4. Prediction ability

While the gradient descent learning algorithm gives us just a particular value of the weights - the most probable one - the posterior (52) gives us a much more informative quantity: the degree of belief that a certain set of weights correctly classifies our data. This is the quantity we need when we want to make predictions!

What do we mean exactly by prediction? We started from a training set D consisting of p inputs $\{\xi_i^\mu\}_{\mu=1}^p$, and a set of p data points $\{\zeta^\mu\}_{\mu=1}^p$ giving us the class of each set of inputs. By minimizing the cost function (47) using the gradient descent learning algorithm (49), we computed the most probable value of the weights w_i^*. Now we want to use our trained perceptron to make the following prediction: given the new input ξ_i^{p+1}, estimate to which class it belongs, given the previous set of input-output matches:

$$P(\zeta^{p+1}|\xi_i^{p+1}, D) \ .$$

The first guess would be to run our neural network using the most probable set of weights w_i^* and just compute $\mathcal{O}(\xi_i^{p+1}, w^*)$. But the problem is that our most probable set of weights is *uncertain*. That's just the value that extremizes the posterior probability distribution (posteriorw 52) of the weights, but the fact that we have a probability distribution in the first place means that such extremum is uncertain. What usually happens, in fact, is that the most probable value of the parameters is likely to give an over-confident prediction on the class of the new input.

It is clear that we have to give a prediction that takes into account the uncertainty on the parameter estimation. We need to use Bayes' theorem!

Let us marginalize over the posterior distribution of uncertain parameters we just evaluated:

$$P(\zeta^{p+1}|\xi_i^{p+1}, D) = \int [dw] P(\zeta^{p+1}|\xi_i^{p+1}, w) P(w|D) \ ,$$

where the integral is over all the synaptic weights $[dw] = \prod_{i=1}^N dw_i$. The first factor $P(\zeta^{p+1}|\xi_i^{p+1}, w)$ in the integral is the probability of classifying the input in a certain way (classification 50), given some value of the weights. The second factor $P(w|D)$ is just the posterior probability distribution of the weights (posteriorw 52) that we constructed with our training set

$$P(w|D) = e^{-M(w)}/Z_M \ , \qquad (53)$$

weightdistr

where

$$Z_M = \int [dw'] e^{-M(w')} \ .$$

The final expression for the probability that the new input ξ_i^{p+1} belongs to class ζ^{p+1}, given the training set, is

$$P(\zeta^{p+1}|\xi_i^{p+1}, D) = \int [dw] P(\zeta^{p+1}|\xi_i^{p+1}, w) \frac{e^{-M(w)}}{Z_M} \ , \qquad (54)$$

generali

giving

$$\begin{aligned} P(\zeta^{p+1} = 1|\xi_i^{p+1}, D) = & \quad \tfrac{1}{Z_M} \int [dw] e^{-M(w)} \mathcal{O}(\xi_i^{p+1}, w) \ , \\ P(\zeta^{p+1} = 0|\xi_i^{p+1}, D) = & \ \tfrac{1}{Z_M} \int [dw] e^{-M(w)} \Big(1 - \mathcal{O}(\xi_i^{p+1}, w)\Big) \ , \end{aligned} \tag{55}$$

The devil is in the details of course. The ball now is in the game of those who know how to compute integrals. Given the fact that for a typical network of neurons the dimensionality of the integral is in the thousands at least, we need a way to implement this integral. It may be done with Monte Carlo or other methods.

Laplace approximation. One can compute an approximate generalization probability (54) by performing a Laplace approximation of the weight distribution (53). We assume that a maximum a posteriori w_{MAP} of the posterior weight distribution (53) was found with a learning algorithm:

$$w_{MAP} = \text{argmax}_w P(w|D) = \text{argmin}_w M(w).$$

Then one can Taylor expand the cost function as

$$M(w) \simeq M(w_{MAP}) + \frac{1}{2}(w - w_{MAP})^T A (w - w_{MAP}) + \dots \ ,$$

where

$$A_{ij} = \frac{\partial^2}{\partial w_i \partial w_j} M(w)|_{w=w_{MAP}} \ ,$$

is the *Hessian* of the posterior, which is guaranteed to be positive definite if the learning algorithm found a local minimum. We can then approximate the posterior distribution (53) with a new gaussian distribution

$$Q(w; w_{MAP}, A) = \frac{1}{(2\pi \det A)^{N/2}} \exp\left[-\frac{1}{2}(w - w_{MAP})^T A (w - w_{MAP})\right] \ ,$$

where A^{-1} represent the covariance matrix of the weights w_i and can be used to estimate confidence intervals.

4.1.5. Perceptron capacity

We saw earlier that given $p = 4$ points on a $N = 2$ plane, we could solve the AND and OR tasks, but not the XOR task, because it was not linearly separable. In general, given a set of p points in an N dimensional space, we can design 2^p different sets of black and white coloring ("dichotomies").[9] Only those dichotomies that are linearly separable can be implemented with a perceptron. *Given a set of p points in N dimensions, what is the number $C(p, N)$ of dichotomies that can be classified?*

We expect that all 2^p dichotomies can be learned when $p << N$, and that only a small fraction of dichotomies can be classified if $p >> N$. The *shattering dimension* or *VC dimension* (Vapnik-Chervonenkis) of a classifier with N dimensional inputs is defined as the maximum number p_{VC} of points for which any dichotomy can be learned. A slightly less restrictive definition, from a physicist point of view, is the *capacity*, defined as the maximum number p_{max} of points for which *almost all* dichotomies can be learned. We will compute both numbers for the perceptron. In particular, we will find that, in the limit of large input space N, the capacity is

$$p_{max} = 2N \ ,$$

We will find that up to $p_{VC} = N$ all dichotomies can be learned, and up to $p_{max} = 2N$ *almost all* dichotomies can be implemented, but for $p > 2N$ there's a sharp transition such that *almost no* dichotomies can be implemented.

Given a set of p points in an N dimensional space, how many dichotomies $C(p, N)$ are linearly separable, i.e., can be classified by a perceptron? We will specialize to the case of a perceptron with a threshold function (perthreshold 37) and with zero bias, namely, with activation $h = \sum_{i=1}^{N} w_i \xi_i$, where ξ_i are N-dimensional vectors representing the points to color in black or white. A linearly separable dichotomy can be classified by a perceptron if there is a hyperplane passing through the origin which separates the two sets of colors.

Let's start with examples. If $N = 1$, given p points on a line, we can color them in 2^p different way. However, the maximum number of dichotomies is 2:

[9] A dichotomy or a set of black/white colorings is also referred to as an input/output (I/O) function.

black for negative points, and white for positive points, or viceversa, as the only hyperplane here is the origin. Analogously, if we have only one point $p = 1$, in arbitrary dimension N, we still have only 2 dichotomies, as we can paint the point black or white. Hence, we found that:

$$C(1, N) = 2 \ , \qquad C(p, 1) = 2 \ . \tag{56}$$

cpnboundary

We will now consider p points in N dimensions distributed in *general position*: no more than d points can lie on a d dimensional plane through the origin, for any $d < N$. For example, in $N = 2$ dimensions, we do not allow for pairs of points to lie on the same line through the origin; or in $N = 3$ we do not allow for three points to lie on a plane through the origin or for pairs of points to lie on a line through the origin.

Let us consider now points on a plane $N = 2$. Three points on a plane in general position can be colored in 6 different ways by lines through the origin: $C(3, 2) = 6$. Notice that in general when $p \leq N$ all 2^p colorings are implementable, but for $p > N$ one has less than 2^p implementable colorings. In particular, when we studied colorings of $p = 4$ points on the $N = 2$ plane, we found that AND and OR functions were implementable, but XOR was not. It is easy to see that $C(4, 2) = 8$. In general given p point on the plane in general position, one can draw a separating line at any point and split the set into black and white. Then, one can rotate the line counterclockwise until it crosses the closest point, and switch that color. By proceeding point by point until the line rotates by 360 degrees, we achieved all the possible $2p$ dichotomies, so: $C(p, 2) = 2p$.

We will proceed by induction to obtain a recursion relation for $C(p, N)$. Suppose we have $C(p, N)$ linearly separable dichotomies for p points in N dimensions, and now we add one more point Q. The new dichotomies $C(p+1, N)$ follow into two categories:

A) For each of the D old dichotomies whose hyperplanes pass through the new point Q, we can slightly displace their hyperplanes to the left or right of Q (because Q is in general position) and generate two new colorings.

B) For the old dichotomies whose hyperplane does no pass through Q, then Q is on one side of their hyperplane so that gives a single coloring.

So from the colorings in B and half of those in A we get back the same dychotomies as before, plus we have an extra amount D. Thus we find:

$$C(p+1, N) = C(p, N) + D \ .$$

How many are the D dychotomies whose hyperplanes pass through Q? Constraining a hyperplane to pass through two points (Q and the origin O) is the same as projecting the N dimensional space into the $N-1$ dimensional space perpendicular to OQ, which gives $D = C(p, N-1)$, so that

$$C(p+1, N) = C(p, N) + C(p, N-1) \ . \tag{57}$$

recursionc

If we iterate this recursion relation over p for fixed N we find:

$$C(p,N) = \binom{p-1}{0} C(1,N) + \binom{p-1}{1} C(1,N-1) + \binom{p-1}{2} C(1,N-2) + \ldots$$
$$+ \binom{p-1}{p-1} C(1, N-p+1) \ .$$

where if $p > N$ we set to zero all terms with negative argument $C(1,m) = 0$ with $m < 0$. We find

$$C(p,N) = \sum_{i=0}^{N-1} \binom{p-1}{i} C(1, N-i) \ ,$$

but we remember the boundary condition $C(1,N) = 2$ for any N:

$$C(p,N) = 2 \sum_{i=0}^{N-1} \binom{p-1}{i} \ ,$$

In particular if $p \leq N$ we can perform the binomial sum and find $C(p,N) = 2^p$, so that

$$C(p,N) = \begin{cases} 2 \sum_{i=0}^{N-1} \binom{p-1}{i} & \text{for}\, p > N \ , \\ 2^p & \text{for}\, p \leq N \ , \end{cases} \tag{58}$$

capacity

Let us compare $C(p,N)$ with the total number of dichotomies we can draw, which is 2^p. For $p \leq N$, then all the possible dichotomies can be implemented by a threshold function, but for $p > N$ the number of implementable dichotomies drops. The line $p = N$ is then the *maximum number of points for which any dichotomy can be implemented* by a threshold function. This is an important quantity called the VC dimensions of this class of functions, after Vapnik-Chervonenkis. The VC dimension of a perceptron (binary threshold function) in N dimensions is thus N.

If $p > N$ we can realize less than 2^p dichotomies, however, we want to estimate the number of points such that we can realize *almost* all dichotomies for large N. We can consider the gaussian approximation to the binomial coefficient for large N and find:

$$\begin{aligned} C(p,N) \quad &\simeq_{N\to\infty} \quad 2^p \Phi\left[\frac{N-1-\frac{p-1}{2}}{\sqrt{p-1}/2}\right] \ , && (59) \\ &\simeq \quad 2^p \Phi\left[\sqrt{p}\left(\frac{2N}{p}-1\right)\right] \ , && (60) \end{aligned}$$

where $\Phi[x] = \int_{-\infty}^{x} dx e^{-x^2/2}/\sqrt{2\pi}$ is the cumulative gaussian. Using the fact that $\Phi[0] = 1/2$ we find that for $p = 2N$ we can still realize half of the dichotomies. In particular, for large N, $C(p,N)/2^p$ has a steep drop from 1 to 0 at $p = 2N$, such that for $p < 2N$ the fraction of implementable dichotomies is closed to one and for $p > 2N$ it sharply drops to zero. Thus, we conclude that for large N the capacity of a perceptron is $p = 2N$.

4.2. Multi-layer perceptron

In section andorxor 4.1.1 we experimented with our single-layer perceptron and realized that, while it can perform the AND and OR tasks, it cannot perform the XOR task. This is because it can only classify data that is linearly separable. This is clearly a big drawback, since our brain can perform the XOR task at light speed. For this reason, people initially abandoned the idea of using the perceptron.

4.2.1. XOR

It turns out that if we just add *more layers* to the feed-forward architecture we easily overcome this problem. Let us recall the truth table for the XOR

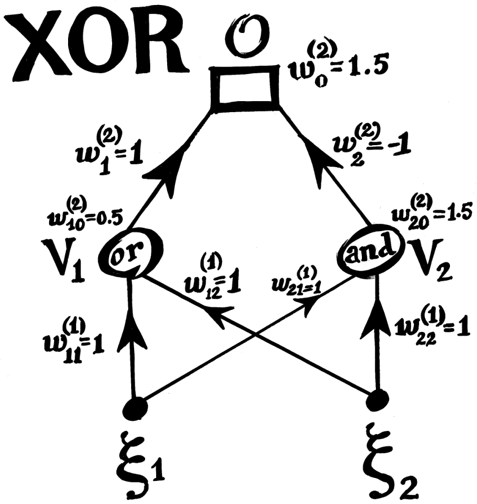

Figure 16: XOR function, realized as a two-layer feedforward architecture. The first layer consists of an OR (left) and AND (right) functions. The second layer takes the OR and the NOT AND and produces a XOR output.

XORpic

function

XOR function	
INPUT	TARGET
ξ_1 ξ_2	ζ
0 0	-1
0 1	1
1 0	1
1 1	-1

We can split this function into two subsequent logical steps

$$\mathcal{O} = \xi_1 \text{ XOR } \xi_2 = (\xi_1 \text{ OR } \xi_2) \text{ AND NOT } (\xi_1 \text{ AND } \xi_2) \ .$$

This gives us a hint. We will build a network with two layers. The first layer will have two units: the first one V_1 computes OR as in $(\overset{\texttt{orw}}{41})$, the second one V_2 computes AND as in $(\overset{\texttt{andw}}{39})$

$$V_1 = \xi_1 \text{ OR } \xi_2 \ , \qquad V_2 = \xi_1 \text{ AND } \xi_2 \ .$$

The second layer will have a single unit $\mathcal{O}$ that implements the AND function as in (39) between the first unit output V_1 and the NOT of the second unit V_2 (NOT is implemented by simply taking the output and turning it into its opposite).

$$\mathcal{O} = V_1 \text{ AND } (\text{NOT } V_2) \ .$$

This simple **multi-layer perceptron** architecture is shown in Fig. 16. The units V_i in the intermediate layer are called **hidden units**.

4.2.2. Error back-propagation

A multi-layer feed-forward network with one hidden layer can perform the XOR task that was not available to the simple perceptron. In Fig. 16 we gave an example of the weights that realize that device. Multi-layer feed-forward networks are *universal approximators*. This means that given any function, we can always find a certain architecture that can can approximate that function arbitrarily well. Once again, the question we face now is: what learning algorithm can we use to teach the network how to perform this task?

We will show how gradient descent of an error function works as well for the multi-layer architecture. First we will derive the back-propagation procedure in a network with two layers, and then we will state the general algorithm.

We will use the relative entropy (47) as a cost function. Our multi-layer network has 2 layers. The first layer is just the usual input ξ_i, for $j = 1, \ldots, N_0$. The intermediate hidden layer collects the inputs and spits out the outputs

$$V_i = g(h_i) \ , \qquad i = 1, \ldots, N_h \ ,$$

$$h_i = \sum_{j=1}^{N_0} w_{ij} \xi_j \ ,$$

where the activity of the hidden layer g can be either the sigmoid function (36) or any of the other alternatives, linear, ReLu, sign, step function, or stochastic. The last layer is the output

$$\mathcal{O} = f(h) \ ,$$

$$h = \sum_{i=1}^{N_h} W_i V_i \ ,$$

that we take as a single unit for simplicity. We will set the output unit $\mathcal{O}$ to the sigmoid function (sigmoid 36), which allows us to simplify the formulas when taking the gradient descent as in (gradientG 46).

While the signal is transmitted forward, the error is propagated backward. Let us see how this works. We apply the gradient descent procedure to change the synaptic weights. We can apply the procedure layer by layer, starting from the output and working our way down to the first layer. To avoid cumbersome notations, we will assume that we only have one training pattern (ξ_j, ζ), so we can drop the index μ from the training set.[10] The output unit update rule is

$$\begin{aligned} \Delta W_i = & \quad -\eta \tfrac{\partial G}{\partial W_i} \\ = & \quad \eta \delta_{\mathcal{O}} V_i \ , \end{aligned}$$

where the delta rule is $\delta_{\mathcal{O}} = \zeta - \mathcal{O}$. It clearly coincides with the single-layer perceptron learning rule (perceptronlearningG 44). Let us now update the weights in the hidden layer instead. We just differentiate the error function G with respect to the corresponding weights w_{ij} by applying the chain rule

$$\begin{aligned} \Delta w_{ij} = & \quad -\eta \tfrac{\partial G}{\partial w_{ij}} \\ = & \quad -\eta \underbrace{\frac{\partial G}{\partial \mathcal{O}}}_{-\frac{\zeta-\mathcal{O}}{\mathcal{O}(1-\mathcal{O})}} \textstyle\sum_{l=1}^{N_h} \underbrace{\frac{\partial \mathcal{O}}{\partial V_l}}_{\mathcal{O}(1-\mathcal{O})W_l} \underbrace{\frac{\partial V_l}{\partial w_{ij}}}_{g'(h_l)\delta_{il}\xi_j} \\ = & \quad \eta \delta_{\mathcal{O}} W_i g'(h_i) \xi_j \\ = & \quad \eta \delta_i \xi_j \ , \end{aligned}$$

In the last step we introduce the hidden delta rule $\delta_i = \delta_{\mathcal{O}} W_i g'(h_i)$, where $h_i = \sum_{k=1}^{N_0} w_{ik} \xi_k$.

[10] Alternatively, we can still have p patterns in our training set, but we are using the stochastic gradient descent procedure in which we update the weights at every step of the training procedure.

The pattern of error back-propagation is now clear. Let us summarize the general rule. In a multi-layer network of M layers, let us denote by $V_{i_m}^{(m)}$ the output of the m-th layer and by $w_{i_m j_{m-1}}^{(m)}$ the weights connecting the unit $V_{j_{m-1}}^{(m-1)}$ to $V_{i_m}^{(m)}$. $V_{i_0}^{(0)} = \xi_i$, while $V_{i_M}^{(M)} = \mathcal{O}_{i_M}$. The online back-propagation gradient descent algorithm is

1. Start with small w's and a training pattern (ξ_i, ζ).
2. Propagate the signal through the layers all the way to the output via the activity rule
$$V_{i_m}^{(m)} = f(h_{i_m}) = f\left(\sum_{j_{m-1}} w_{i_m j_{m-1}}^{(m)} V_{j_{m-1}}^{(m-1)}\right) \ .$$
3. Now back-propagate the error. Compute the delta's for the output layer
$$\delta_i^M = \zeta - \mathcal{O} \ .$$
4. Compute the delta's for the previous layers, propagating backwards through the network
$$\delta_{j_m}^{(m)} = \sum_{i_{m+1}} \delta_{i_{m+1}}^{(m+1)} w_{i_{m+1} j_m}^{(m+1)} g'(h_{j_m}^{(m)}) = [\delta^{(m+1)} w^{(m+1)T}]_{j_m} g'(h_{j_m}^{(m)}) \ , \qquad m = M-1, \ldots, 1.$$
5. Update the weights using the gradient descent
$$\Delta w_{i_m j_{m-1}}^{(m)} = \eta \delta^{(m)} V_{j_{m-1}}^{(m-1)} \ .$$
6. Repeat for the next pattern.

While the back-propagation algorithm is widely used in the machine learning community, it is less clear if it has to do with the brain.

4.2.3. Problems with backpropagation

Even though gradient descent via backprop is theoretically sound, in practice, it is hard to train deep architectures with it. The main problem is that because the update rules in a lower layer m close to the input depends on the product of $M - m + 1$ gradients from the chain rule, this leads to either *vanishing or exploding gradients.* To understand the origin of this problem, let us consider a simple (M+1)-layer network with one neuron per layer with activity rule $f(h^{(m)})$ and identical weights connecting all layers $w^{(m)} = w$. The update rule for weights in the bottom layer is then given by

$$\Delta w^{(1)} = \eta \delta^{(M+1)} \left[\prod_{m=1}^{M} w f'(h^{(m)}) \right] \xi = \eta \delta^{(M+1)} w^M \left[\prod_{m=1}^{M} f'(h^{(m)}) \right] \xi \ ,$$

Let us assume that derivatives of activity rules across layers have a similar value $f'(h^{(m)}) = f'_0$ and find

$$\Delta w^{(1)} \simeq \delta^{(M+1)} (w f'_0)^M \xi \ .$$

For large M we see that if $wf'_0 > 1$ the gradient explodes, while if $wf'_0 < 1$ the gradient vanishes. It is thus necessary that $wf'_0 \simeq$ at every training step in order to perform meaningful training. This means that i) weights should stay around unity and ii) neural activity should not saturate as it would occur for the sigmoid at large positive or negative values of its argument h where $|f'(h) \to 0$. In the case of layers with larger widths, one can extend this argument to see that the singular values of the weight matrices $w^{(m)}_{i_m i_{m-1}}$ must stay of order 1 at each training step. A popular way to avoid saturation is to use ReLUs (whose derivative is positive and constant for positive argument) instead of saturating functions, and to initialize the weights in clever ways. Let us list some popular methods:

- A simple way to prevent weights from blowing up is to use L_2 regularization, a.k.a. weight decay, which we did in (learningrule 49), or L_1 regularization with a prior such as $\sum_i |w_i|$ which leads to sparse weights, or a combination of both L_1 and L_2 called *elastic net.*

- Another way to prevent overfitting (large weights) is to use stochastic gradient descent, as opposed to batch gradient descent, which is akin to introducing noise in the gradient; and to use mini-batches with random permutations of small subsets of training data in each mini-batch.

- Clever weight initialization include initializing the weights using a gaussian distribution with zero mean and a variance that scales as $1/N$, where N is the width of the layer

- To choose the learning rate appropriately.

- To whiten the input data, namely, to rescale it so that it has zero mean and input variance, so that all ξ_i contribute with the same magnitude to the update rule.

- Early stopping: Split your training set into a smaller training and a validation set, and use only the smaller training set for training, while monitoring the generalization error in the validation set which is a proxy for the error in the test set (which is always left untouched). Typically, the validation error would decrease until reaching a minimum at some training epoch t_0, then start increasing again. One then stops training early at the t_0 epoch.

- Dropout: this is the most successful way of training deep architecture by avoiding the emergence of correlations between weights which may lead to vanishing gradients. For each mini-batch in the stochastic gradient descent, drop neurons from the network (and set to zero the corresponding weights) with a probability $1 - p$, then update only the surviving weights, until reaching the end of training with weights w_{train}. When estimating the test set generalization performance, one should then use $w_{test} = pw_{train}$. This procedure is motivated by ensemble methods, namely, one assumes that an "ensemble" of networks have been trained.

Guide to the literature

There are several books about supervised learning, most of them have a machine learning spin. Our goal in this chapter though is to build increasingly complex models that can describe our brain functions, rather than finding efficient methods to solve optimization problems. The discussion of the perceptron, the AND, OR and XOR examples, the multi-layer network, and the back-propagation learning rule are adapted from Hertz' book hertz [7]. The gradient descent algorithm, learning as an inference problem, and the prediction ability are adapted from MacKay's book on information theory mackay [3]. The calculation of the perceptron capacity combines both Hertz' book hertz [7] and MacKay's book mackay [3]. The regularization methods are discussed in mehta2019high [5].

Learning in a linear perceptron

Consider a perceptron with activity rule $\mathcal{O}(h)$, where $h = \sum_{i=1}^{N} w_i \xi_i$ is the activation for an N-dimensional input pattern ξ_i. The training set includes p patterns $\{\xi_i^\mu, \zeta^\mu\}_{\mu=1,\ldots,p}$, where ζ^μ are the targets.

Question 1: Derive the perceptron learning rule starting from the following cost function:

$$E = \frac{1}{2} \sum_{\mu=1}^{p} \left[\zeta^\mu - \mathcal{O}(h^\mu)\right]^2 \ , \qquad (61) \qquad \boxed{\texttt{cost}}$$

by applying gradient descent. What is the difference between the learning rule obtained from this cost function and the one obtained in class using relative entropy?

Question 2: Now, specialize to the case of a linear activity $\mathcal{O}(h) = h$. What is the learning rule in this case? Find a simple set of $\mathcal{O}^\mu$ that solve the gradient descent of ($\overset{\texttt{cost}}{61}$).

Question 3: Consider the following matrix representing the overlap, or correlation, between the input patterns:

$$Q^{\mu\nu} = \frac{1}{N} \sum_{i=1}^{N} \xi_i^\mu \xi_i^\nu \ .$$

If the input patterns ξ_i^μ are linearly independent, then Q can be inverted such that $\sum_{\nu=1}^{p} (Q^{-1})^{\mu\nu} Q^{\nu\rho} = \delta^{\mu\rho}$, where $\delta^{\mu\rho}$ is 1 if $\mu = \rho$ and zero otherwise. Verify that

$$w_i^* = \frac{1}{N} \sum_{\mu,\nu=1}^{p} \zeta^\mu (Q^{-1})^{\mu\nu} \xi_i^\nu \ , \qquad (62) \qquad \boxed{\texttt{wstar}}$$

realizes the solution to gradient descent found in Question 2.

Question 4: Consider the following family of weights:

$$\tilde{w}_i = w_i^* + \lambda y_i \ ,$$

where λ is a real-valued parameter and w_i^* is a solution to the gradient descent for the linear perceptron $\mathcal{O}(h) = h$, such as ($\overset{\texttt{wstar}}{62}$). Find the condition on y_i such that $\tilde{w}_i$ is a solution for any real value of the parameter λ. What is a geometric interpretation of y_i in the N dimensional space of inputs ξ_i^μ?

Question 5: If such a y_i exists, then gradient descent solutions can be found with arbitrarily large weights, leading to the problem of overfitting. What is a simple modification to the cost function in ($\overset{\texttt{cost}}{61}$), which only depends on the weights but not on the targets ζ^μ, such that the weights cannot grow arbitrarily large thus preventing overfitting? (Hint: Remember the Bayesian interpretation of the cost function).

5. Unsupervised learning

So far we have considered the most obvious way of learning something: learning with a teacher. A neuron receives some inputs, performs its computation, then spits an output. There's a teacher outside that tells the neuron what is the *correct* target. If target and output do not agree, the neuron adjusts its synaptic weights according to minimize an error function. In this way we make the output resemble as close as possible the target.

Sometimes, there will not be a teacher around to tell each output neuron if its computation was correct. In this Chapter we will consider the case of learning in which there is no teacher (hence the title). Namely, we have a neural network that receives some input and spits some output, and solely based on these two quantity it will adjust its weights. The purpose of this network may be to detect features in the input signal. Namely, if the input signal has some redundancy, it may capture the important information hidden in the input signal.

Imagine you are a newborn child. You just came out of the womb and your senses are recording an overload of inputs for the first time. You never opened your eyes before, so your visual cortex, though wired up by spontaneous activity during gestation, is not really familiar with the new signal arriving from the retina through the optical nerve. The simplest thing that your visual cortex can do is to detect some correlations among the inputs it receives. We will see shortly that the Hebb rule (hebbrule 63) of synaptic plasticity (or some slight modification thereof) achieves this goal.

5.1. Learning

The crucial property of the brain is that it can **learn**. From the neurobiological point of view, the learning process is believed to be related to *synaptic plasticity*. Synapses form, disappear, strengthen and weaken dynamically, as the network of neurons performs some tasks.

When should a synapse change? Let us do a little experiment. Every night you are watching your favorite show, CSI Eugene. In every episode, the villain shoots somebody with a pistol. The first time you see the villain pointing the gun at the victim, you wander what would happen next. Easy answer: you discover that when the villain shoots and hits, the victim falls to the floor. This bizarre phenomenon "villain shoots → victim falls" reoccur reliably every time you watch the show. You come to associate the act of shooting with the act of falling to the floor. Whenever you see somebody shooting or hear a gun shot, you imagine immediately a body falling down. You have *learned* that shooting and falling are associated. To put it in the words of Donald Hebb (1949)

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

In other words, there must be a synaptic connection from the axon A to the post-synaptic neuron B, that is activated only when both neurons fire together. This is the celebrated **Hebb rule** of learning that can be summarized in the following mantra

Two neurons that fire together are wired together.

Whenever we see the event coded by neuron A occurring together with the one coded by neuron B, the synaptic connection between the two neurons must somehow strengthen. You usually think about this phenomenon as *learning the correlation* between the events A and B. Denote by ξ the input coming into our unit B from the pre-synaptic neuron A and by $\mathcal{O}$ the output of the post-synaptic neuron B. We can formulate the change of the synaptic weights in terms of the Hebb rule

$$\Delta w_{BA} = w_{BA}^{new} - w_{BA}^{old} = \eta \mathcal{O}_B \xi_A \ , \tag{63}$$

hebbrule

where η is called the learning rate. Whenever neurons A and B fire simultaneously, the strength of the synaptic connection from A to B increases. The biological mechanism through which the neurons implement this rule is believed to be due to the presence of Calcium channels on the membrane of the synapse. In this Section, we will first discuss the neurobiological plausibility of this hypothesis. Then we will implement the Hebb rule in the perceptron example. Then, we will consider the meaning of the Hebb rule in the context of information theory.

5.1.1. Neurobiology of the Hebb rule

Do the actual synapses in the brain obey the Hebb rule? We will convey a simplified picture of a plausible mechanism for synaptic plasticity and address this question in the following order

- Anatomy of the hippocampus, where studies of synaptic plasticity and memory have led to most success.
- The biochemical mechanism present at the synapse, inducing plasticity.
- Biological implementation of quantitative aspects of the Hebb rule.

There are two different kinds of memory, that are believed to be radically different, as far as their biological origin is concerned: short-term and long-term memories.

Short-term memory lasts from a few seconds to a few minutes. If I tell you quickly a phone number that does not bear any particular significance for you, it is probably going to resound in our head for a second and then you will forget it, and will not be able to recall it later. It is believed that this kind of memory is due to the *sustained activity* of a neural circuit. A certain number of neurons is experiencing a persistent oscillatory activity related to an attractor point of the network dynamics, that lasts for a while (remembering) and then gets dumped to zero (forgetting). This kind of memory is likely not related to adjustments in the synaptic weights and is a sort of “software” property of the neural circuits.

On the other hand, long-term memory, for instance when we recall memory of past experiences or events, is believed to be a hardwired property of the brain, which is encoded in the strength of the synapses. When we memorize a new experience, we are changing the synaptic weights. In a sense, the neuroscientist’s version of the saying “we are our memories” is rather “we are our synapses." Most of what we know about synaptic plasticity and its relation to memory comes from the study of the hippocampus. This is a bananiform part of the brain, which lies deep inside our skull in the medio-temporal region. We have two copies of the hippocampus, one in the left and one in the right brain emisphere. The hippocampus is not part of the neo-cortex, but it is a very old structure that can be found already in reptiles and it is called archi-cortex, so it is an ancient part of the brain from an evolutionary perspective.

The input to the hippocampus comes from the entorhinal cortex, a part of the neo-cortex related to navigation and spatial informations. The entorhinal cortex axons connect onto the granule cells in the dentate gyrus (called "mossy fibers"), which in turn synapse on the dendrites of the CA3 hippocampal pyramidal cells. The CA3 cells have a forest of recurrent connections among themselves, and project to the CA1 pyramidal cells, which in turn send the signal back to the entorhinal cortex and also broadcast to the rest of the brain. The role of the hippocampus may be to encode episodic memory: the explicit memory of events, places or sensory inputs in general, as opposed to the procedural memory related to performing tasks. This has been discovered on patients whose hippocampus got removed to cure debilitating forms of epilepsy: they lost the ability to remember new events, even if they could still remember abstract concepts or acquired skills.

The study of synaptic plasticity in the hippocampus has been carried out by taking slices of the brain and stimulating and recording from single pairs of pre-synaptic CA3 and post-synaptic CA1 cells. In this way, one can analyze the dependence of the synaptic weights on the pre- and post-synaptic potentials. A stimulating electrode is placed in the pre-synaptic CA3 cell and a very weak current is injected in the axon to produce a single spike, a bit as if you are pointing a taser to the soma of the neuron. A recording electrode is placed in the post-synaptic dendrite of the CA1 cell, to measure the strength of the excitatory post-synaptic potential generated by the arrival of the signal.

Biochemical mechanism for plasticity

The way the electro-chemical signal is transmitted at the interface between two neurons is through the exchange of glutamate neurotransmitter through the synapse. At the arrival of the action potential to the pre-synaptic terminal, the bouton releases glutamate, that diffuses in the intra-synaptic space and gets captured by receptors on the spine of the post-synaptic dendrite. There are two types of receptors on the post-synaptic terminal: AMPA and NMDA. Both are activated by the glutamate. The AMPA receptor is the one mostly responsible for the generation of the post-synaptic spike. After getting activated by the glutamated binding, its ion channel opens up and lets in an influx of Na^+ ions that depolarize the post-synaptic membrane potential. When the excitatory post-synaptic potential (EPSP) reaches a threshold (that varies from neuron to neuron), the neuron emits an action potential, that we usually refer to as a "spike." The NMDA receptors, on the other hand, do not contribute much to the action potential: they are involved in the implementation of synaptic plasticity.

We are interested here in the fact that the NMDA receptor has a voltage dependent gate. The NMDA, even when activated by the glutamate, is still blocked by the presence of a Mg^{2+} ion. When the depolarization of the membrane potential reaches a certain level, the NMDA throws out the Mg^{2+} and its ion channel opens up to an influx of Calcium ions from the extracellular fluid into the post-synaptic spine. The unblocking of the active NMDA receptor only happens if the post-synaptic potential reaches a threshold, related to the firing of an action potential by the post-synaptic neuron. The NMDA receptors are "coincidence detectors," namely they permit Calcium influx only when they are simultaneously activated by glutamate (incoming spike) and the post-synaptic neuron is emitting a spike. The AMPA receptors are not permeable to Ca^{2+}, only the NMDA receptors are. It is the presence of Ca^{2+} which in turn triggers the change in the synaptic strength.

Let us consider first the Long Term Potentiation (LTP) of the synapse. The Hebb rule states that the concurrent firing of the pre- and post-synaptic neurons strengthens the synapse. After the arrival of the pre-synaptic signal and the depolarization leading to the EPSP, the NMDA receptors only unblock when the “backpropagation” signal from the EPSP travels back through the post-synaptic dendrites and hits the spine, where the NMDA are located. This mechanism makes sure that an EPSP is needed in order to open the NMDA receptors, allowing the Ca^{2+} influx.

After Ca^{2+} invades the post-synaptic spine, two things happen. First, such ions activate new AMPA receptors that join their brothers already stuck to the membrane of the spine. The more AMPA receptors, the more sensitive the post-synaptic dendrite is to the glutamate shot and the more efficient the signal transmission becomes. Secondly, the presence of Calcium stimulates the growth of the spine itself, through the expression of particular proteins inside the dendrite, thereby potentiating the synapse once again. A confirmation of this picture comes from the fact that many synapses are *silent*: they only contain NMDA receptors, responsible for plasticity, but no AMPA receptors, the ones that convey the electrical signal. It has been observed that upon arrival of the spike and the subsequent burst of EPSP, the NMDA receptors open up and allow Ca^{2+} to enter the spine, which in turn bring some AMPA receptors, previously absent, to the surface of the spine, therefore making the synapse talk.

The other facet of synaptic plasticity as summarized in the Hebb rule is the weakening of the synapse, when either one of the pre- and post-synaptic spikes occur, but not both combined. This leads to Long Term Depression (LTD). It is clear that we need LTD for the correct functioning of the brain, otherwise we would just potentiate the strength of the synapse until they all saturate - runaway excitation! This has been observed again in the CA3-CA1 synapse. If one induces a low frequency stimulation in the pre-synaptic terminal at around 1 Hz, but with no EPSP generated, the synaptic weight decreases. One of the various mechanisms for LTD is based on NMDA receptors. The arrival of a large number of pre-synaptic spikes, while the post-synaptic neuron remain silent, keeps low but sustained levels of Ca^{++} in the spine, which causes AMPA receptors to be progressively depotentiated and finally removed.

Quantitative results

The Hebb rule (hebbrule 63) is quantitative in its predictions. In particular, we can ask the following questions

1. If we induce LTP not just once but repeatedly on the same synapse, does the increase in synaptic strength add up linearly? Is the change in w_{BA} *independent* of w_{BA} itself?
2. Since pre- and post-synaptic activity together induce LTP (or either absence induces LTD), how does that depend on the exact timing of the two spikes?

It is clear that the change in the synaptic weight must be democratic, independent of the value of the weight itself. If it depended on the strength itself, it would clearly advantage strong connections unfairly, leading to a situation in which very few connections become very strong exponentially, while the remaining ones are just silent - a winner-take-all scenario. The answer to question one is that the rule (63) is not totally realistic: the LTP does not add up linearly. After strengthening a synapse through correlated pre- and post-synaptic activity, the synapse becomes refractory to further changes for at least one hour. The synaptic change occurs in a stochastic “all or none” fashion. Either it strengthens of a certain fixed amount, described by η in (63), or it does not. And it is more likely to occur depending on the strength of the pairing. The fact that the synaptic changes are “digital” rather than “analogue” goes in the direction of a stable and more efficient learning procedure. Since the spine itself and number of NMDA receptors on its surface are quite small, it would be extremely unprecise to add a number of AMPA receptors that vary with the strength of the synapse. This digital updating is the most efficient way to make it work.

The answer to the second question is revealed in a breakthrough experiment that led to the discovery of *spike timing-dependent synaptic plasticity* (STDP), reported in Fig. 17. In this experiment, we look at a pair of pre- and post-synaptic cells and study the synaptic change by varying the coincidence timing of the pre- and post-synaptic spikes. The results show that

- The maximum LTP is achieved if the post-synaptic spike follows the arrival of the pre-synaptic one by 10 ms. This is clearly what is needed if we

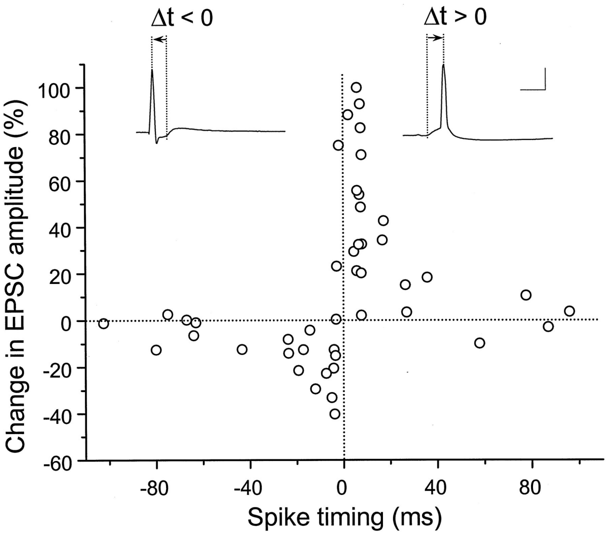

Figure 17: Spike time-dependent synaptic plasticity, from bipoo [8]: "Critical window for the induction of synaptic potentiation and depression. The percentage change in the EPSC amplitude at 20-30 min after the repetitive correlated spiking (60 pulses at 1 Hz) was plotted against the spike timing. Spike timing was defined by the time interval (Δt) between the onset of the EPSP and the peak of the postsynaptic action potential during each cycle of repetitive stimulation, as illustrated by the traces above."

bipoofig

want to interpret the EPSP as *being caused* by the pre-synaptic signal. The delay is due to the finite membrane back-propagation time.

- If the post-synaptic spike precedes the pre-synaptic arrival by at least 10 ms, LTD occurs instead. This is clearly in line with causality again, since if this happens over and over again, it is better to turn the synapse off.

- If the pre- and post-synaptic spikes are exactly simultaneous, nothing happens.

- If the pre- and post-synaptic spikes occur at a distance of more than 100 ms, then again no change takes place. The spontaneous activity of the brain typically has a frequency around 10 Hz, which means that the chance of having pre- and post-synaptic spikes every 100 ms or more, caused by random activity, is very high. In this case, we expect that nothing should occur.

5.2. *Hebb rule*

It is an empirical observation that two neurons that fire together are wired together. If neuron i fires every time neuron j fires, then they are connected by some synapses. The rule of thumb of synaptic plasticity, called Long Term Potentiation (LTP), states that in this situation the synaptic weight w_{ij} increases in strength. If we call $\mathcal{O}_i$ the output of neuron i and ξ_j the output of neuron j (which is the input of neuron i, we can summarize this rule as (Hebb rule)

$$\Delta w_{ij} = \eta \mathcal{O}_i \xi_j \ , \qquad (64)$$

where η is the (small) learning rate. On the other hand, if neuron j fires, but neuron i never fires in response, it means that the two activities are not related. Hence, the strength of their synaptic connection, if any, must be rather weak. If we start with some synaptic weight w_{ij} and fire the two neurons in an uncorrelated manner, the weight should decrease. This is the observed phenomenon of Long Term Depression (LTD).

There is an interesting way of describing both LTP and LTD in an unified manner

$$\Delta w_{ij} = (\mathcal{O}_i - \langle \mathcal{O}_i \rangle)(\xi_i - \langle \xi_i \rangle) \qquad (65)$$

covariancerule

where $\langle \cdot \rangle$ denotes the average over time. It is clear that if the two neurons are firing at a similar rate with respect to their average, their synaptic connection strengthens, otherwise it weakens.

PCA

5.3. *Principal Component Analysis*

Let us take a closer look at the plasticity rule (covariancerule 65). We use the vector notation $\mathbf{w} = (w_1, \ldots, w_N)$ and $\xi = (\xi_1, \ldots, \xi_N)$ and take a single unit with a linear activity rule

$$\mathcal{O} = \mathbf{w} \cdot \xi \ , \qquad (66)$$

linearar

Let us also assume that inputs have zero mean

$$\langle \xi_i \rangle = 0 \ ,$$

where $\langle \xi_i \rangle = \frac{1}{p}\sum_{\mu=1}^{p} \xi_i^{\mu}$. Then the rule ($\overset{\texttt{covariancerule}}{65}$) reads

$$\Delta \mathbf{w} = \mathbf{C} \cdot \mathbf{w}\ , \qquad (67) \qquad \boxed{\texttt{updatecov}}$$

where we introduced the *correlation matrix* of the input signals

$$\mathbf{C} = C_{ij} = \langle \xi_i \xi_j \rangle\ .$$

If we consider the update rule ($\overset{\texttt{updatecov}}{67}$) as a continuous rather than a discrete phenomenon replacing $\Delta w \to dw/dt$, we find

$$\frac{d\mathbf{w}}{dt} = \mathbf{C} \cdot \mathbf{w}\ . \qquad (68) \qquad \boxed{\texttt{weightc}}$$

The $N \times N$ covariance matrix is symmetric, so it has N real eigenvalues. It is also positive definite, since $\mathbf{x}^T \cdot \mathbf{C} \cdot \mathbf{x} = [\mathbf{x} \cdot (\xi - \langle \xi \rangle)]^2 \geq 0$ for any $\mathbf{x}$. This means that it has real non-negative eigenvalues, which we can order

$$\lambda^{(1)} \geq \ldots \geq \lambda^{(N)} \geq 0\ .$$

It is easy to solve ($\overset{\texttt{weightc}}{68}$) by diagonalizing the covariance matrix

$$C \cdot \mathbf{e}^{(k)} = \lambda^{(k)} \mathbf{e}^{(k)}\ . \qquad (69) \qquad \boxed{\texttt{PCA}}$$

We can expand the weights along the eigenvector basis $\mathbf{e}^{(k)} = (e_1^{(k)}, \ldots, e_N^{(k)})$ as

$$\mathbf{w}(t) = \sum_{k=1}^{N} c^{(k)}(t) \mathbf{e}^{(k)}\ . \qquad (70) \qquad \boxed{\texttt{weigen}}$$

The evolution equation ($\overset{\texttt{weightc}}{68}$) translates into an equation for the coefficients

$$\frac{dc^{(k)}}{dt} = \lambda^{(k)} c^{(k)}\ ,$$

with initial condition $c^{(k)}(0) = \mathbf{w}(0) \cdot \mathbf{e}^{(k)}$. The solution is

$$c^{(k)}(t) = c^{(k)}(0) e^{\lambda^{(k)} t}\ ,$$

from which we can reconstruct the time dependence of the synaptic weights

$$\mathbf{w}(t) = \sum_{k=1}^{N} [\mathbf{w}(0) \cdot \mathbf{e}^{(k)}] e^{\lambda^{(k)} t} \mathbf{e}^{(k)}\ . \qquad (71) \qquad \boxed{\texttt{woft}}$$

Let us stare at this solution for a moment. We are interested in its behavior at late times t, when the learning procedure is supposed to have delivered its results. Due to the exponential factor in (woft 71), if $\mathbf{w}(0) \cdot \mathbf{e}^{(1)} \neq 0$, the late time behavior is determined by its largest positive eigenvalue $\lambda^{(1)}$

$$\mathbf{w}(t) \propto \mathbf{e}^{(1)} \ ,$$

giving a late time weight matrix aligned with the eigenvector corresponding to the largest eigenvalue of the covariance matrix, which is called the *first principal component*. After the learning procedure, the output $\mathcal{O}$ therefore is going to spit the projection of the input ξ along the principal component $\mathbf{e}^{(1)}$

$$\mathcal{O} \propto \mathbf{e}^{(1)} \cdot \xi \ .$$

Let us discuss the geometric significance of the Principal Component Analysis (PCA). Imagine that our input data (we take two inputs, $N = 2$, for simplicity) is a random sample coming from a two dimensional Gaussian distribution with difference variance in the two directions. After training, the weight vector will align in the direction where the data is more spread, namely the direction with the largest variance. Since the entropy of a random variable sampled from a Gaussian distribution grows with its variance as $\log_2 \sigma$, we see that the PCA algorithm maximizes the amount of information the output carries about the input. If we are given a noisy input signal like the one in the picture, and we were told to represent the data with just one object, we would pick the first principal component, that encodes the basic facts about the data.

It is not all rainbows, flowers and lepricons though. We immediately see a problem with our naive learning rule (covariancerule 65). Due to the exponential factor in (woft 71) and the fact that the eigenvalue $\lambda^{(1)}$ is positive, the value of the weights (its norm $\mathbf{w} \cdot \mathbf{w} = |\mathbf{w}|^2$) blows up at late times, which is clearly an unpleasant feature. This learning algorithm predicts our brain is going to explode! This is not a very realistic feature, so we need to fix it.

5.3.1. Oja's rule

We already encountered the problem of runaway weights in the context of the gradient descent learning rule (49). There we saw that the easiest way to prevent the weights from getting too large is just to add a weight decay term. In the case of Hebbian plasticity, the best way to implement such a procedure goes under the name of the *Oja's rule*

$$\Delta w_{ij} = \eta \left(\mathcal{O}_i \xi_j - \alpha \mathcal{O}_i^2 w_{ij} \right) \ . \tag{72}$$

ojarule

This rule is very powerful. Let us see what its main feature is. Consider for simplicity a single output unit with linear activity rule (66), in the vector notation of Section 69. Oja's rule reads $\Delta \mathbf{w} = \eta \mathcal{O} \left(\xi - \alpha \mathcal{O} \mathbf{w} \right)$. Let us take the projection of this vector equation onto the weight vector $\mathbf{w}$ itself. By observing that $\mathbf{w} \cdot \Delta \mathbf{w} = \frac{1}{2} \Delta (\mathbf{w} \cdot \mathbf{w}) = \frac{1}{2} \Delta |\mathbf{w}|^2$, we find

$$\Delta |\mathbf{w}|^2 = \eta \mathcal{O}^2 (1 - \alpha |\mathbf{w}|^2) \ .$$

At late times, once the learning procedure has been implemented and $\Delta \mathbf{w} \sim 0$, Oja's rule gives a weight vector of finite norm $|\mathbf{w}|^2 = 1/\alpha$. This learning rule therefore solves the runaway weights problem found at the end of Section 69.

Let us now prove that Oja's rule converges to the first Principal Component. After learning, we have $\langle \Delta \mathbf{w} \rangle = \mathbf{0}$, where $\langle ... \rangle$ represents the average over the patterns $\mu = 1, \ldots, p$. We then find:

$$0 = \langle \Delta w_i \rangle = \langle \mathcal{O} \xi_i - \mathcal{O}^2 w_i \rangle = (Cw)_i - (w^T C w) w_i \ ,$$

where $C_{ij} = \langle \xi_i \xi_j \rangle = \sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu$. We thus found that after learning

$$Cw = \lambda w \ ,$$

so that w converges to an eigenvector of C. Its eigenvalue is $\lambda = w^T C w = \sum_{k=1}^{N} \lambda^{(k)} (c^{(k)})^2$ where we expanded the weights along the principal components using (70). This implies that $\sum_{k=1}^{N} (c^{(k)})^2 = 1$, namely, that the weight vector has unit norm, $|w|^2 = 1$ (remember we set $\alpha = 1$ in Oja's rule (72).

We showed that Oja's rule drives the weight vector to align to an eigenvector of the covariance matrix C, but which one? In order to find it, we will perform a stability analysis. We will assume that the weights are aligned to the k-th eigenvector $e^{(k)}$, and perturb it by a small vector ϵ

$$w = e^{(k)} + \epsilon \ ,$$

and we will ask whether the update rule $\langle \Delta w \rangle$ is going to amplify the perturbation, bringing the weights away from $e^{(k)}$, or rather dump the perturbation, bringing the weights back to $e^{(k)}$. Namely, we will calculate whether the solution $w = e^{(k)}$ is stable. This analysis will show that all perturbations get amplified except for $k = 1$, the first Principal Component. Let us calculate the update to first order in the perturbation:

$$\begin{aligned}
\langle \Delta\epsilon \rangle &= \langle \Delta w \rangle = Cw - (w^T C w)w = C(e^{(k)} + \epsilon) - [(e^{(k)} + \epsilon)^T C(e^{(k)} + \epsilon)] \\
&= Ce^{(k)} + C\epsilon - (e^{(k)T} C e^{(k)})e^{(k)} - 2(\epsilon^T C e^{(k)})e^{(k)} - (e^{(k)T} C e^{(k)})\epsilon + \mathcal{O}(\epsilon^2) \\
&= C\epsilon - 2\lambda^{(k)}(\epsilon^T e^{(k)})e^{(k)} - \lambda^{(k)}\epsilon + \mathcal{O}(\epsilon^2) \ , \qquad (73)
\end{aligned}$$

where we used ($\overset{\text{PCA}}{69}$) and dropped higher order terms in ϵ. Let us decompose this increment onto the principal component basis $e^{(l)}$ and find the directions where it grows or shrinks

$$e^{(l)T}\langle \Delta\epsilon \rangle = (e^{(l)T}\epsilon)[\lambda^{(l)} - \lambda^{(k)}(1 + 2\delta^{kl})] + \mathcal{O}(\epsilon^2) \ ,$$

where we used orthonormality of the eigenvectors $(e^{(k)T}e^{(l)}) = \delta^{kl}$. We find that the increment projected onto orthogonal directions $l \neq k$ to the original eigenvector $e^{(k)}$

$$e^{(l)T}\langle \Delta\epsilon \rangle = [\lambda^{(l)} - \lambda^{(k)}](e^{(l)T}\epsilon) \ , \qquad (74) \qquad \boxed{\texttt{edecrease}}$$

grows if $\lambda^{(l)} > \lambda^{(k)}$, hence the solution will be unstable. This occurs unless w was aligned to the first principal component $k = 1$, which has the largest eigenvalues $\lambda^{(1)}$. In the latter case, the solution is always stable because it decreases both in orthogonal directions ($\overset{\text{edecrease}}{74}$), since $\lambda^{(l)} < \lambda^{(1)}$ for $l \neq 1$, and also in the direction of $e^{(1)}$ itself:

$$e^{(k)T}\langle \Delta\epsilon \rangle = -2\lambda^{(k)}(e^{(k)T}\epsilon).$$

Finally, let us prove that the solution $w = e^{(1)}$ also maximizes the variance $\langle \mathcal{O}^2 \rangle$ of the output:

$$\langle \mathcal{O}^2 \rangle = \langle (w^T \xi) \rangle = (w^T C w) = \lambda^{(1)} \ ,$$

where $\lambda^{(1)}$ is the direction of largest variance hence maximizing the quadratic form $(w^T C w)$.

We thus conclude that Oja's rule implements the Principal Component Analysis of the input data, producing an output which is the projection of the input onto its first principal component, with a finite value of the weight vector. Because the length of the weight vector is fixed, Oja's rule introduces a *competition* between the different weights. When one of the weights increase, others must decrease to keep the length finite. This rule is also *local*: for each output unit, its implementation only depends on the inputs connected to that output.

5.3.2. Oja's rule from a cost function

We will derive here Oja's rule as a gradient descent of a cost function. Consider a perceptron with linear activity rule $\mathcal{O}(h) = h$, where $h = \sum_{i=1}^{N} w_i \xi_i$ and N-dimensional input patterns ξ_i. The training set includes p patterns $\{\xi_i^\mu\}_{\mu=1,\dots,p}$ with zero mean $\frac{1}{p}\sum_{i=1}^{p} \xi_i^\mu = 0$. Consider the following cost function

$$\begin{aligned} E &= -\frac{1}{2p}\sum_{i=1}^{p} \mathcal{O}^\mu \mathcal{O}^\mu + \frac{\alpha}{4}(\sum_{i=1}^{N} w_i^2)^2 \ , \\ &= -\frac{1}{2}(w^T C w) + \frac{\alpha}{4}(\sum_{i=1}^{N} w_i^2)^2 \ , \end{aligned} \qquad (75)$$

where $C_{ij} = \frac{1}{p}\sum_{i=1}^{p} \xi_i^\mu \xi_j^\mu$. The gradient of the cost function (costoja 75) is

$$\frac{\partial E(w)}{\partial w_i} = -\sum_{j=1}^{N} C_{ij} w_j + \alpha w_i (\sum_{j=1}^{N} w_j^2)$$

The learning rule is:

$$\Delta w_i = -\eta \frac{\partial E(w)}{\partial w_i} = \eta [(Cw)_i - \alpha w_i (w^T w)]$$

The extrema w^* of the cost function (costoja 75) are of two kinds: $w_i^* = 0$; and the solutions to

$$(Cw)_i = (w^T w) w_i \ ,$$

which is the eigenvalue equation for C, giving as solution

$$w_i^* = \sqrt{\frac{\lambda^{(k)}}{\alpha}} e_i^{(k)}$$

where $e^{(k)}$ and $\lambda^{(k)}$ are the eigenvectors and eigenvalues of C. Which extrema are saddle points and which is is the global minimum of the cost function? Let us compute the Hessian H_{ij} of the cost function (costoja 75):

$$H_{ij} \equiv \frac{\partial^2 E(w)}{\partial w_i \partial w_j} = -C_{ij} + \alpha \delta_{ij} (w^T w) - 2\alpha w_i w_j$$

At the extremum $w_i^* = 0$ the Hessian is given by $-C_{ij}$ whose eigenvalues are all negative or zero, so it cannot be a minimum. At the other extrema where $w_i \sim e_i^{(k)}$, the Hessian amounts to

$$H_{ij} | w^* = -C_{ij} + \lambda^{(k)} \delta_{ij} + 2\lambda^{(k)} e_i^{(k)} e_j^{(k)}$$

We can project the Hessian onto the basis of Principal Components $e^{(l)}$ and find:

$$(e^{(l)T} H e^{(l)}) | w^* = -\lambda^{(l)} + \lambda^{(k)} (1 + 2\delta^{kl}) = \begin{cases} 2\lambda^{(l)} \ , & l = k \\ \lambda^{(k)} - \lambda^{(l)} \ , & l \neq k \end{cases} \tag{76}$$

The only minimum where all projections are positive is where $\lambda^{(k)} - \lambda^{(l)} > 0$ for all l, attained at $k = 1$. So the only minimum is where w is aligned to the first Principal Component. Having a cost function allows a simple implementation of Oja's rule using tensorflow (see below).

In the case of a single output unit, Oja's rule gives us the first principal component of the input. Suppose that we have instead multiple output units. A simple generalization of Oja's rule (ojarule 72) can be introduced in order to extract the other principal components, namely the components of the input signal along the either eigenvectors of the covariance matrix, but we will not discuss it here. The PCA technique works well in the case in which the input signal distribution is Gaussian. When this is not the case, then we must resort to other kind of analysis to separate the indipendent sources that generate the data: Indipendent Component Analysis is a technique that helps in this case.

Guide to the literature

The discussion of the neurobiological evidence for synaptic plasticity and the Hebb rule is loosely taken from Paul Adams' notes pauladams [9], based on various clues accumulating during the past decades and the final discovery of the spike-time dependent synaptic plasticity. The various unsupervised learning rules are discussed in hertz [7], Chapter 8. The cost function for the Oja's rule was found in yuille1989quadrature [10].

Further reading

For those who would like to learn more about the theory of supervised learning in neural networks, here is a list of recommendations.

- *A high bias, low variance introduction to ML for physicists.* mehta2019high [5] This review covers the topics we discussed in class, with an emphasis on a theoretical approach using variational methods and mean field theory. In addition, Chapters 14-18 explain deep generative models such as Restricted Boltzmann Machines, and self-supervised models such as Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs). It includes awesome notebooks to play with the networks.
- *Deep Learning.* goodfellow2016deep [11] This is a comprehensive book on deep learning reviews all the modern tools for training deep networks, with emphasis on the practical aspects. It includes lectures and tutorials as well.Research. On the more research-oriented side, recently there has been a lot of work on applying methods from statistical physics to understand the theory of deep learning.

Some interesting recent developments are:

- *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks.* saxe2013exact [12] A thorough analysis of deep linear networks where one can solve learning exactly.

- *Statistical mechanics of deep learning.* bahri2020statistical [13] A survey of the recent interesting directions and methods in deep learning including expressivity, the error landscape, parameter initialization, generalization performance, and probabilistic models.

Interlude: Statistical Mechanics

6. Recap of Statistical Mechanics

We will give now a flash summary of key concepts and methods in statistical mechanics, as based on a graduate level class. These methods are widely used in all the rest of the course and this Section comes in handy. After reminding ourselves the definition of the most common statistical ensembles, microcanonical, canonical and grand canonical, we will rederive them by observing that they all maximize a quantity, called the statistical entropy. Then we will introduce the Ising model with nearest neighbor interaction and solve exactly the one-dimensional case, to show that there is no phase transition. It is a general phenomenon, that one-dimensional systems with short-range interactions do not display phase-transitions. If in the same system we introduce long-range "non-local" interactions instead, we can produce phase transitions even in one dimension! We will compare the nearest neighbor interaction with the long-range model. Most natural systems are not exactly solvable, unlike the Ising model. One therefore needs to develop some approximations to compute the observables. The basic tool we will discuss is mean field theory.

6.1. Ensembles

Statistical mechanics is about understanding from a microscopic point of view how everyday observables emerge, like energy, pressure, magnetization, number of particles. Given a macroscopic system with, say, some fixed energy, there are many different ways (microscopic states) in which we can arrange the tiny constituents to realize such energy configurations. The number of ways we can arrange microstates to generate a given macroscopic configuration makes up what we call entropy.

The postulate of equilibrium statistical thermodynamics is that, when we measure a macroscopic observable of our system, like the energy, we are actually averaging over a large number of microscopic states, consistent with the measured value of the observable. Such ensemble of microstates is called the *Gibbs ensembles.*

Typical systems we encounter in Neuroscience are a collection of a fixed number of N neurons, that form a neural network, in contact with some external bath of incoming signals. In all our applications to Neuroscience, it will suffice to use only one particular kind of ensemble, called *canonical.* We assume that our system has a fixed number N of particles and a fixed volume V, and it is in contact with an external heat reservoir at fixed temperature T, with which it is in equilibrium. The only thing our system can do is then exchanging heat with the thermal bath, adjusting its energy E. We will see when studying memory that temperature can be interpreted as the amount of noise in the signal transmission among neurons.

The macroscopic system is characterized by the value of the energy observable at fixed temperature. The microstate s with energy E_s occurs with a probability

$$\mathcal{P}_s = \mathcal{P}(E_s) = \frac{e^{-\beta E_s}}{Z(V,N,T)} \ , \qquad (77) \qquad \boxed{\texttt{probe}}$$

where we introduced the *canonical partition function*

$$Z(V,N,T) = \sum_r e^{-\beta E_r}, \qquad (78) \qquad \boxed{\texttt{partitionz}}$$

where r labels all of the microstates in the theory, including their degeneracy. In this way the probabilities sum up to unity. The factor

$$\beta = \frac{1}{kT} \ ,$$

introduces the temperature T and the Boltzmann constant $k = 1.3806504(24) \times 10^{-23}\, J/K = 8.617343(15) \times 10^{-5}\, \mathrm{eV}/K$.

In classical mechanics, the partition function is determined by the integral of the density of states $\rho(E)$

$$Z(V,N,T) = \int dE\, \rho(E) e^{-\beta E} \ .$$

In a quantum mechanical system, suppose we have an energy operator H with a discrete spectrum of eigenvalues E_r and eigenvectors $|\psi_r\rangle$. The system is completely characterized by its density matrix

$$\rho = \sum_r |\psi_r\rangle \mathcal{P}_r \langle\psi_r = \frac{1}{Z} \sum_r |\psi_r\rangle e^{-\beta E_r} \langle\psi_r| \ , \qquad (79) \qquad \boxed{\texttt{densitymatrix}}$$

where $\mathcal{P}_r = e^{-\beta E_r}/Z$ is the probability (probe 77) of having an eigenstate ψ_r and

$$Z(V, N, T) = \mathrm{Tr}\, e^{-\beta H} \ ,$$

which gives us back (partitionz 78). Given an observable A, corresponding to an operator acting on the quantum mechanical Hilbert space, its macroscopic average is given by

$$\langle A \rangle = \mathrm{Tr}\, \rho A \ .$$

6.2. Entropy

Let us derive the different statistical ensembles from a more current point of view, introducing the concept of *statistical entropy*, related to information theory. Suppose we have a series of events $\{e_r\}$, for $r = 1, \ldots, M$, each event occurring with probability $\mathcal{P}_r \geq 0$, subject to the constraint

$$\sum_r \mathcal{P}_r = 1 \ . \tag{80}$$

probone

The entropy of the probability distribution $\{\mathcal{P}_r\}$ is

$$S = -\sum_{r=1}^{M} \mathcal{P}_r \ln \mathcal{P}_r \ .$$

If we only have one event, the entropy is obviously vanishing. If all event are equally likely, namely $\mathcal{P}_r = 1/M$, the entropy is maximal: $S = \ln M$. You got the basic point: the less information carried by the probability distribution, the larger the corresponding entropy. One can prove in fact that $0 \leq S \leq \ln M$, by finding the extrema of S subject to the constraint (probone 80). If we have two series of events $\{e_r\}$ and $\{e_{r'}\}$ with probability distributions $\mathcal{P}$ and $\mathcal{P}$, the entropy of the total system satisfies the inequality $S(\mathcal{P} \otimes \mathcal{P}') \leq S(\mathcal{P}) + S(\mathcal{P}')$, where the equality holds in case the two sets are statistically independent. The entropy is therefore an *extensive* quantity.

A quantum mechanical system with density matrix (densitymatrix 79) has a statistical entropy

$$S = -k \mathrm{Tr}\, \rho \ln \rho = -k \sum_r \mathcal{P}_r \ln \mathcal{P}_r \ ,$$

where we introduced the Boltzmann constant to identify the statistical entropy with the thermodynamic one.

The *Fundamental Postulate of Equilibrium Statistical Mechanics* gives us a way to derive the various ensembles from the definition of statistical entropy. Suppose we are given a system in which a certain number of observables are fixed to specific macroscopic values $\langle A_i \rangle$. Among the many density matrices that are compatible with the values $\langle A_i \rangle = \mathrm{Tr}\, \rho A_i$, we must choose the one that maximizes the entropy S. Such density matrix describes the Boltzmann, or equilibrium, distribution.

Let us put work out the consequences of this principle. We build our trial entropy functional

$$\frac{1}{k}\tilde{S} = -\mathrm{Tr}\, \rho \ln \rho + \sum_i \lambda_i (\mathrm{Tr}\, \rho A_i - \langle A_i \rangle) - \lambda_0 (\mathrm{Tr}\, \rho - 1) \ ,$$

where the λ's are Lagrange multipliers. Using the fact that $d\mathrm{Tr}\, f(A) = \mathrm{Tr}\, f'(A) dA$, we can extremize the trial entropy

$$-\frac{1}{k} d\tilde{S} = \mathrm{Tr}\, d\rho \left(\ln \rho + 1 - \sum_i \lambda_i A_i + \lambda_0 \right) = 0 \ ,$$

requiring that the quantity inside the parenthesis vanishes for all ρ and derive an expression for the density matrix of the Boltzmann equilibrium distribution

$$\rho_B = \frac{1}{Z} e^{\sum_i \lambda_i A_i} \ , \qquad Z = \mathrm{Tr}\, e^{\sum_i \lambda_i A_i} \ .$$

The entropy is then given by $S = -k \mathrm{Tr}\, \rho_B \ln \rho_B$ that turns into the familiar Legendre transform of $\ln Z$ with respect to the Lagrange multipliers λ_i

$$S = -k \ln Z + k \sum_i \lambda_i \langle A_i \rangle \ ,$$

where

$$\langle A_i \rangle = \partial_{\lambda_i} \ln Z \ . \tag{81}$$

averageA

If we fix the average value of the energy $E = \langle H \rangle$, we derive the canonical ensemble (densitymatrix 79), that we can read also as

$$\rho = \frac{1}{Z} e^{-\beta H} \ . \tag{82}$$

gibbs

The Lagrange multiplier for the energy constraint is the inverse temperature $\beta = 1/kT$ and the thermodynamic potential in this case is called the Helmholtz free energy (the analogue of the effective action in quantum field theory)

$$F(V, N, T) = -\frac{1}{\beta} \ln Z = E - TS.$$

If we fix other average values of other operators $\langle A_i \rangle$, we can derive different thermodynamics potentials, according to our specific problem in the lab. For instance, fixing both average energy and particle number gives the grand potential for the grand canonical ensemble, that depends on both the temperature and the chemical potential (sometimes called fugacity).

The effect of probing our system with an external stimulus is parametrized by the dependence of our averaged observable $\langle A_i \rangle$ on its corresponding Lagrange multiplier. We will call this the "response." Thanks to (averageA 81), this is equal to the second variation of the logarithm of the partition function. But the latter quantity defines the quadratic fluctuation of our observable around its average value. We just established the very useful *fluctuation-response* theorem

$$\partial_{\lambda_j} \langle A_i \rangle = \langle (A_i - \langle A_i \rangle)(A_j - \langle A_j \rangle) \rangle \ . \qquad (83)$$

fluctuationresponse

One can prove that the matrix $\partial_{\lambda_i} \partial_{\lambda_j} \ln Z$ is a positive definite quantity, which implies that $\ln Z$ is a convex function of the λ's, while the entropy S is a concave function of the $\langle A_i \rangle$'s.

subsec:nearest-neighbor

6.3. Nearest-neighbor interactions

Let us illustrate the cuteness of the canonical ensemble in the simplest example of an interacting system: the one dimensional Ising model with nearest-neighbor interactions. It turns out that the partition function is exactly solvable, using for instance the transfer matrix method. Along the way, we will introduce the concept of a thermodynamic limit and connected correlation functions.

The Ising model was invented as an approximation to the Heisenberg model of ferromagnetism. In this model, we are given a collection of lattice sites, each

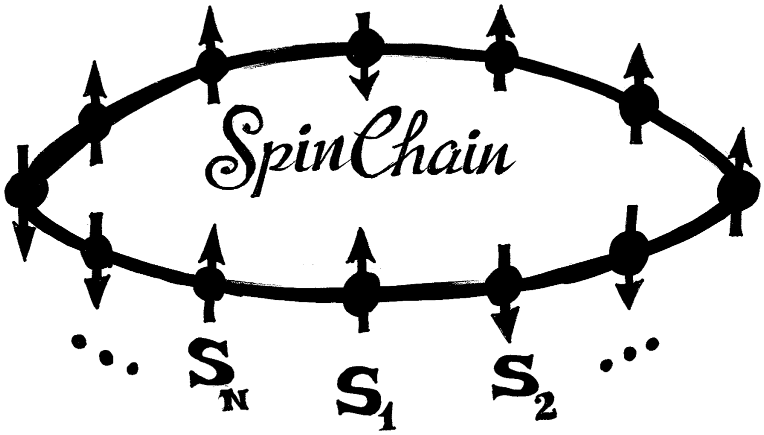

Figure 18: The spin chain realization of the Ising model with periodic boundary conditions. The arrows denote the spins S_i in (onedising 84). Spin up is $S_i = 1$, spin down is $S_i = -1$. spinchain

of which hosts a spin one-half variable $\vec{\sigma}_i$ and hamiltonian

$$H = -J \sum_{<i,j>} \vec{\sigma}_i \cdot \vec{\sigma}_j - \mu \sum_i \vec{B} \cdot \vec{\sigma}_i \ ,$$

where μ is the magnetic moment of the spins, J is the nearest-neighbor interaction strength and B is the external magnetic field. The interaction sum $< i, j >$ runs only on nearest-neighbors. However, this model is not exactly solvable, so we replace it with a simpler one that is very easy to solve.

The Ising model consists of N spin variables $S_i = \pm 1$, lying on a spin chain with periodic boundary conditions and hamiltonian

$$H = -J \sum_{i=1}^{N} S_i S_{i+1} - \mu B \sum_{i=1}^{N} S_i \ , \qquad (84)$$ onedising

We fix periodic boundary conditions $S_{N+1} = S_1$. The partition function is

$$Z = \sum_{\{S_i\}} e^{-\beta H} \ . \qquad (85)$$ isingz

We now evaluate (isingz 85) exactly. Let us define $K = \beta J$, $K' = \beta \mu B$ and rewrite

$$Z = \sum_{S_1 = \pm 1} \ldots \sum_{S_N = \pm 1} \prod_{i=1}^{N} T(S_i, S_{i+1}) = \operatorname{Tr} T^N \ , \qquad (86)$$ isingzone

where we introduced the *transfer matrix*

$$T_{i,i+1} = \exp\left(KS_iS_{i+1} + \frac{K'}{2}(S_i + S_{i+1})\right) = \begin{pmatrix} e^{K+K'} & e^{-K} \\ e^{-K} & e^{K-K'} \end{pmatrix} . \qquad (87) \qquad \boxed{\texttt{transfer}}$$

Thanks to the trace appearing in $(\overset{\texttt{isingzone}}{86})$, we can solve the problem simply by diagonalizing the transfer matrix, which is symmetric and real

$$T = RDR^{-1} \ , \qquad D = \mathrm{diag}(\lambda_+, \lambda_-) \ ,$$

where R is a similarity transformation, that at zero external magnetic field reads

$$R = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = R^{-1} \ ,$$

and the eigenvalues $\lambda_+ > \lambda_-$ are

$$\lambda_\pm = e^K \left(\cosh K' \pm \sqrt{\sinh^2 K' + e^{-4K}}\right) \ .$$

The partition function is finally given by

$$Z(N,T) = \lambda_+^N + \lambda_-^N \ . \qquad (88) \qquad \boxed{\texttt{zlambda}}$$

The thermodynamic limit of a statistical mechanical system is obtained by taking $N, V \to \infty$ while keeping the ratio V/N fixed. A quantity that scales like N is called *extensive* and is well behaved under this limit. Let us compute the thermodynamic limit of the free energy of our Ising model, in which the volume V is by construction proportional to the total number of spins N. We can see from $(\overset{\texttt{zlambda}}{88})$ that $F = -(1/\beta)\ln Z$ scales like N. In thermodynamic limit, only the largest eigenvalue λ_+ survives and we can define the free energy per spin f in this limit

$$f(T) = -\lim_{N\to\infty} \frac{1}{\beta N} \ln Z(N,T) = -\frac{1}{\beta} \ln \lambda_+ \ . \qquad (89) \qquad \boxed{\texttt{freen}}$$

We want to see what happens to our spin chain when we dial the external magnetic field. The relevant observable in this case is the magnetization. Since this is an extensive quantity that scales like N, when we take the thermodynamic limit the correct observable is the magnetization per spin, which stays finite

$$m = \frac{\mu}{N}\langle \sum_i S_i \rangle = -\frac{1}{\beta N}\frac{\partial \ln Z}{\partial B} = -\frac{\partial f}{\partial B}. \qquad (90) \qquad \boxed{\texttt{magnetization}}$$

Now, thanks to the fluctuation-response theorem (fluctuationresponse 83), we know that the variation $\partial_B m$ of the magnetization in response to a change in the external magnetic field (which is the Lagrange multiplier associated with the magnetization) is given by the connected two-point function of the spins. Let us derive this result in our particular case

$$\frac{1}{\beta N}\partial_B m = -\frac{1}{(\beta N)}\partial_B^2 f = \frac{1}{(\beta N)^2}\partial_B^2 \ln Z = \frac{1}{(\beta N)^2}[Z^{-1}\partial_B^2 Z - (Z^{-1}\partial_B Z)^2] \ ,$$

and recalling that $Z^{-1}\partial_B Z = \mu\beta\langle\sum_i S_i\rangle$ and $Z^{-1}\partial_B^2 Z = (\mu\beta)^2\langle\sum_{ij} S_i S_j\rangle$ we finally obtain

$$\begin{aligned}\frac{1}{\beta N}\partial_B m = & \ \tfrac{\mu^2}{N^2}\textstyle\sum_{ij}\left(\langle S_i S_j\rangle - \langle S_i\rangle\langle S_j\rangle\right) \\ = & \ \langle m^2\rangle - \langle m\rangle^2 \ .\end{aligned}$$

Hence, the response of the magnetization to a variation of the magnetic field is given by the fluctuation of the spins. We therefore want to compute the two-point function of the spins.

Let us compute the correlation function of a pair of spins at two different sites

$$\begin{aligned}\langle S_i S_j\rangle = & \ Z^{-1}\textstyle\sum_{\{S\}} S_i S_j e^{-\beta H} \\ = & \ Z^{-1}\textstyle\sum_{\{S\}} \mathrm{Tr}\left(\ldots T_{i-1,i} S_i T_{i,i+1}\ldots T_{j-1,j} S_j T_{j,j+1}\ldots\right) \ , \\ = & \ Z^{-1}\textstyle\sum_{\{S\}} \mathrm{Tr}\left(\ldots T_{i-1,i} \sigma_3 T_{i,i+1}\ldots T_{j-1,j} \sigma_3 T_{j,j+1}\ldots\right) \ , \\ = & \ Z^{-1}\textstyle\sum_{\{S\}} \mathrm{Tr}\left(\ldots R^{-1} D_{i-1,i} R\sigma_3 R^{-1} D_{i,i+1} R\ldots\right. \\ & \left.\ldots R^{-1} D_{j-1,j} R\sigma_3 R^{-1} D_{j,j+1} R\ldots\right) \ ,\end{aligned}$$

where $T_{i,j}$ is the transfer matrix (transfer 87). Let us set for simplicity the external magnetic field $B=0$. It is immediate to realize that $S_i T_{i,j} = (\sigma_3 T)_{i,j}$, where σ_3 is the usual Pauli matrix and now we can diagonalize the transfer matrix once again and observe that $R\sigma_3 R^{-1} = \sigma_1$. After a brief and straightforward manipulation we find

$$\langle S_i S_j\rangle = Z^{-1}\mathrm{Tr}\, D^{i-1}\sigma_1 D^{j-i}\sigma_1 D^{N-j+1} \ ,$$

where D is the diagonal matrix in (87). The effect of inserting the spin variables in the trace is tantamount as flipping the eigenvalues of the transfer matrix as $\sigma_1 D \sigma_1 = \mathrm{diag}(\lambda_-, \lambda_+)$ and one finally obtains, in the thermodynamic limit

$$\langle S_i S_j \rangle = (\lambda_-/\lambda_+)^{j-i} = (\tanh[J/kT])^{|j-i|} \ . \qquad (91)$$

twopoint

We learn a general feature of the thermodynamic limit: the largest eigenvalue controls the free energy, while its ratio to the second largest one controls the correlation function.

The two-point function determines how strongly a perturbation inserted at the i-th site will propagate to the distant site j. Hence, it is useful to introduce the concept of *correlation length* ξ and rewrite

$$\langle S_i S_j \rangle = \exp\left(\frac{r_{ij}}{\xi}\right) \ .$$

In our simple Ising model, with unit lattice spacing, the distance is $r_{ij} = |j-i|$ and from (91) we find

$$\xi = 1/|\ln\tanh(J/kT)| \ . \qquad (92)$$

correlationlength

The correlation length in the one-dimensional Ising model is a monotonically increasing function of the temperature, which shows there is no phase transition.

Alas, no phase transition in this model. At this point, you have two choices: you might be very discouraged and quit, just like Ising himself did after finishing the previous computation, or you may keep an optimistic mindset and move to the next model. If you bear with me a few more paragraphs, you will be rewarded by a real phase transition.

longrange

6.4. Long-range interactions

It is a general lore that systems in one spacial dimensions do not have phase transitions. However, this is only true if the interactions are short-range, as in the case of the 1d Ising model with nearest-neighbor interactions that we just solved. A very cute example of a phase transition, that we will discuss now, happens when we consider the same 1d Ising model, but we we change the couplings to obtain a non-local theory. This example is similar in spirit to the spin glasses we will introduce later on. Consider the hamiltonian

$$H = -\frac{J}{2N}\sum_{i,j} S_i S_j - \mu B \sum_i S_i \ , \tag{93}$$

nonlocalh

where now the sum in the interaction term runs over *all* the lattice sites, not just the nearest-neighbors as in ($\overset{\texttt{onedising}}{84}$). Note that the coupling strength is decreasing as we increase the total number of spins, which is necessary for the system to have a thermodynamic limit. The model is still exactly solvable, by resorting to a clever trick called the *Hubbard-Stratonovich* transformation. Since we are summing over all spins, we can rewrite

$$\sum_{i,j} S_i S_j = \left(\sum_i S_i\right)^2 - N.$$

The hamiltonian thus reads

$$H = -\frac{J}{2N}\Sigma^2 - \mu B \Sigma - \frac{J}{2} \ ,$$

where $\Sigma = \sum_i S_i$. In the following, we will drop the last term since it will be irrelevant in the large N limit. We now convert the last quadratic term in Σ into a linear term, by noting the following property of the Gaussian integral

$$e^{a^2\Sigma^2} = \int_{-\infty}^{+\infty} \frac{dx}{\sqrt{\pi}} e^{-x^2 - 2a\Sigma x} \ ,$$

where in our case $a = \sqrt{\beta J/2N}$. The partition function for the theory ($\overset{\texttt{nonlocalh}}{93}$) is thus

$$\begin{aligned} Z = & \textstyle\sum_{\{S_i\}} \exp\left(\frac{\beta J}{2N}\sum_{i,j} S_i S_j + \beta\mu B \sum_i S_i\right) \\ = & \textstyle\int \frac{dx}{\sqrt{\pi}} e^{-x^2} \sum_{\{S_i\}} e^{(\beta\mu B - 2ax)\sum_i S_i} \ . \end{aligned} \tag{94}$$

It is easy to perform the sum over the spin configurations now, since it just factorizes, leading to the partition function

$$\begin{aligned} Z = & \textstyle\int \frac{dx}{\sqrt{\pi}} e^{-x^2} \sum_{\{S_i\}} \prod_{\{S_i\}} e^{(\beta\mu B - 2ax)S_i} \ . \\ = & \textstyle\int \frac{dx}{\sqrt{\pi}} e^{-x^2} \left(2\cosh(\beta\mu B - 2ax)\right)^N \ , \end{aligned}$$

where we used $2\cosh x = e^x + e^{-x}$. Setting $w = 2ax - \beta\mu B$ we finally obtain the exact partition function

$$Z = \sqrt{\frac{N}{2J\beta\pi}} \int dw\, e^{-Ns(w)} \ , \qquad s(w) = \frac{(w+\beta\mu B)^2}{2J\beta} - \ln(2\cosh w) \ . \tag{95}$$

$\boxed{\texttt{partiz}}$

We want to study the occurrence of phase transitions in the thermodynamic limit. First of all, we note that, in the limit $N \to \infty$, the integral in $(\overset{\texttt{partiz}}{95})$ can be evaluated by the saddle point approximation and it localizes around the minimum w_0 of the function $s(w)$. By Taylor-expanding around the minimum and shifting the integration variable $w = w_0 + u/\sqrt{N}$, we can easily evaluate the integral

$$\int dw e^{-Ns(w)} = \int \frac{du}{\sqrt{N}} e^{-Ns(w_0) - \frac{u^2}{2}s''(w_0) + \cdots} = e^{-Ns(w_0)} \sqrt{\frac{2\pi}{Ns''(w_0)}} + \cdots \ ,$$

where we omitted terms that drop in the thermodynamic limit. The partition function reads

$$Z = \left[J\beta s''(w_0)\right]^{-1/2} e^{-Ns(w_0)} [1 + \mathcal{O}(1/\sqrt{N})] \ .$$

In the thermodynamic limit, only the exponential part survives and we find a free energy density

$$f = \lim_{N\to\infty} -\frac{kT}{N} \ln Z = kTs(w_0) \ .$$

To investigate whether a phase transition occurs, let us consider the magnetization per spin $(\overset{\texttt{magnetization}}{90})$. When the external magnetic field vanishes, the Hamiltonian of our Ising model $(\overset{\texttt{onedising}}{84})$ displays the obvious $\mathbb{Z}_2$ that flips the spin signs $S_i \to -S_i$. The magnetization $(\overset{\texttt{magnetization}}{90})$ is odd under this symmetry, so we expect it to vanish at $B = 0$. On the other hand, we know that the ground state configuration of the system must be a minimum of the free energy $F = E - TS$. Two competing effects are at work here. On one hand, we want to minimize the energy by aligning all of the spins in the same direction. At the same time, we want to maximize the entropy, which requires having a random distribution of spins. At very high temperature, the entropy wins and the magnetization vanishes. But as we lower the temperature, the "ordering effect" of the energy starts to kick in and it is conceivable that at some value of the temperature energy wins over entropy and we develop a non-zero magnetization, even at zero external field.

The way to proceed in order to investigate this issue is to first turn on a non-zero external magnetic field B, breaking the Z_2 symmetry explicitly. After taking the thermodynamic limit, we take the limit of zero external magnetic field, thus restoring the original Z_2 symmetry of the hamiltonian

$$m_{\pm} = \lim_{B\to 0\pm} \lim_{N\to\infty} m \ . \tag{96}$$

thermolim

The *order of limits* in $\overset{\texttt{thermolim}}{(96)}$ is absolutely crucial: if we reverse the order there is nothing to check, since $m = 0$ from the very beginning in that case. But if the thermodynamic limit is taken first, there is still a possible way out: the $\mathbb{Z}_2$ spin-flip symmetry may be *spontaneously broken*. This means that the ground state has fewer symmetries than the hamiltonian. A quantity such as m, which acquires a non-zero expectation value in the phase with broken symmetry, is called an *order parameter*.

A little computation saves a lot of speculations, as the saying goes. Let us first evaluate the magnetization in the nearest-neighbor Ising model we already solved and check that it vanishes. We evaluate $\overset{\texttt{thermolim}}{(96)}$ using the free energy density $\overset{\texttt{freen}}{(89)}$ and find

$$m_{\pm} = \lim_{B\to 0\pm} \lim_{N\to\infty} \frac{1}{\beta} \left[\frac{\partial \ln \lambda_+}{\partial B} \right]_{\beta} = 0 \ . \tag{97}$$

magnespin

The magnetization per spin vanishes in the nearest-neighbor model, which is consistent with the fact that the correlation length $\overset{\texttt{correlationlength}}{(92)}$ is a continuous function. Much ado about nothing there...

Let us now go back to our long range Ising model. The magnetization per spin $\overset{\texttt{magnespin}}{(97)}$ is given by

$$m = -\frac{\partial f}{\partial B} = -\frac{\mu}{\beta J}(w_0 + \beta\mu B) \ .$$

At zero external magnetic field, the magnetization $m \neq 0$ if $w_0 \neq 0$. Remember now that w_0 was defined as the minimum of the function $s(w)$ and the condition

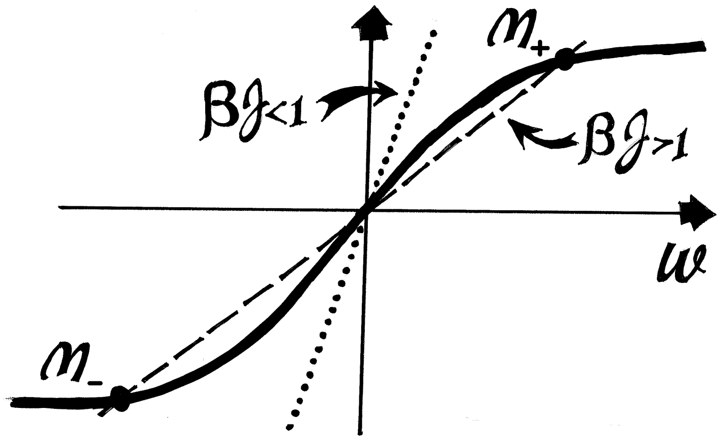

Figure 19: Magnetization in the long range Ising model. Graphical depiction of Eq. (98): the filled line is the right hand side, the dashed and pointy lines are the left hand side at $B = 0$, below and above the critical temperature respectively. There are two solutions corresponding to non-zero values of the magnetization, a positive m_+ and a negative m_-, at $T < J/k$. phasetransition

$s'(w_0) = 0$ gives

$$w_0 + \beta\mu B = \beta J \tanh w_0 \ . \qquad (98)$$

wphase

We need to look for solutions of (98) at zero external magnetic field. The graphical solution to (98) is plotted in Fig. 19. First of all, it is obvious that when the couplings $J < 0$, a.k.a. the anti-ferromagnetic coupling, the only solution is $w_0 = 0$ and the magnetization vanishes for any value of the temperature. But in the ferromagnetic case $J > 0$ we have two different regimes:

- If $\beta J < 1$ there is again only one solution, $w_0 = 0$ and the magnetization vanishes.

- If $\beta J > 1$ two new solutions appear, a positive one for $B \to 0^+$ and a negative one for $B \to 0^-$. It is easy to check that both solutions are minima, while the third solution at $w_0 = 0$ is a maximum.

As a result, we have found that $m \neq 0$ below the critical temperature $T_c = J/k$ and we have a phase transition! The $\mathbb{Z}_2$ spin-flip symmetry is spontaneously broken.

wphase

The equation (98) that determines the magnetization is the same as the equation appearing in the *mean-field* approximation to ferromagnetism. While the usual mean-field theory is just an approximation in the case of nearest-neighbor interactions, we find here that mean-field theory is exact in the thermodynamic limit, when the interactions are long-range. Mean-field theory assumes that each spin interacts in the same way with all the others, which is just an approximation in the former case, but it is an exact statement in the latter case instead. The exactness of mean-field theory will show up again in the spin glass models, whose interactions are similar to the ones we studied in the present example.

Below the critical temperature, where two spontaneous magnetization states $\pm m$ appear, the phase space of the system splits into two separate parts. Such an ensemble represents all the microscopic states of the system which are compatible with a fixed macroscopic value of the magnetization. It is not possible to have trajectories that span the whole of the phase space anymore, but trajectories are confined to either of the two regions. In other words, the spontaneous symmetry breaking phase brings along a breaking of ergodicity.

In the symmetry breaking phase, the original Gibbs measure of the full system $\mathrm{P}[S] = e^{-\beta H[S]}/Z$ does not satisfy cluster decomposition anymore. While at high temperatures we have $\langle S_i S_j \rangle = 0 = \langle S_i \rangle \langle S_j \rangle$ for $|i-j| \to \infty$, below the critical temperature we find instead $\langle S_i S_j \rangle = m^2 \neq \langle S_i \rangle \langle S_j \rangle = 0$. There is a canonical decomposition of the Gibbs measure into a direct sum of probability measures that satisfy cluster decomposition and ergodicity

$$\mathrm{P}[S] = \frac{1}{2}\mathrm{P}_+[S] + \mathrm{P}_-[S] \ ,$$

where $\mathrm{P}_\pm[S] = \lim_{B\to 0^\pm} \lim_{N\to\infty} \frac{1}{Z} e^{-\beta H[S] + B\sum_i S_i}$. This means that, below the critical temperature and for zero magnetic field, the phase space of the theory gets split into two different components. These two regions never talk to each other, namely any configuration has a trajectory in phase space that stays in one superselection sector for all times and never crosses to the other superselection sector. Superselection sectors described by $\mathrm{P}_\pm[S]$ are called "pure states". Note that in presence of a non-zero external magnetic field B, ergodicity is restored.

6.5. Mean Field Theory

The Ising model with nearest neighbor interactions can be solved analytically in low dimensions. However, no solution is known for higher dimensions, including the three dimensional case which is everybody's favourite. So we need to develop some approximation method that can lead us to reliable results. The most common method was invented by Weiss in the far 1907 and goes under the name of *mean field approximation.* We will first illustrate the general idea, then apply it to the nearest neighbor Ising model, and finally show an amazing result: that this approximation is in fact exact in the case where the interaction is long range. The idea is that we want to estimate the free energy ln Z using a variational approach, by exploiting its *convexity.*

Consider two positive definite hermitian operators X, Y. Then

$$\mathrm{Tr}\, X \ln Y - \mathrm{Tr}\, X \ln X \leq \mathrm{Tr}\, Y - \mathrm{Tr}\, X \ .$$

We would like to use this inequality as follows. We have a system, described by the true density matrix ρ, which is very hard to solve. We would like to approximate it with another system, described by an alternative density matrix ρ_λ, that we know how to use to compute expectation values. If we take as our hermitian positive definite operators two different density matrices, the true density matrix ρ and a trial one ρ_λ, both of whose traces are unity, we immediately derive

$$-\,\mathrm{Tr}\, \rho_\lambda \ln \rho_\lambda \leq -\mathrm{Tr}\, \rho_\lambda \ln \rho \ . \qquad (99)$$

inerho

Substituting the usual expressions for Gibbs distribution

$$\rho = \frac{e^{-\beta H}}{Z} \ , \qquad \rho_\lambda = \frac{e^{-\beta H_\lambda}}{Z_\lambda} \ ,$$

we turn (inerho 99) into an inequality among the respective thermodynamic quantities

$$F \leq F_\lambda + \langle H - H_\lambda \rangle_\lambda \equiv \Phi(\lambda) \ , \qquad (100)$$

meanfieldf

where the expectation value in the last term is taken with respect to the trial density matrix ρ_λ, namely $\langle \mathcal{O} \rangle_\lambda = \mathrm{Tr}\, \rho_\lambda \mathcal{O}$. The inequality ($\overset{\texttt{meanfieldf}}{100}$) is the heart of the mean field approximation. It states that the true free energy F, computed using the hamiltonian H, is smaller than any variational estimate based on a trial hamiltonian H_λ. The crucial point is that, even if we do not know how to evaluate the l.h.s. of ($\overset{\texttt{meanfieldf}}{100}$), we do know how to compute the r.h.s. which is an approximation. Smaller successive estimates give better and better approximations. It can be shown that if $H - H_\lambda \sim \epsilon$, then $F - \Phi(\lambda) \sim \epsilon^2$.

6.5.1. Ising mean field theory

Let us apply this formal machinery to our favourite example: the Ising model. It can be either the short range on in Section $\overset{\texttt{subsec:nearest-neighbor}}{6.3}$ or the long range one in Section $\overset{\texttt{longrange}}{6.4}$. It will apply just as well to the Hopfield network of associative memory in the next Chapter. While in the short range case this really is an approximation, in the latter two cases this will give the exact result. Consider an Ising hamiltonian

$$H = -\frac{1}{2}\sum_{ij} J_{ij} S_i S_j - \mu \sum_i B_i S_i \ ,$$

and the trial hamiltonian H_λ that we will use to approximate the true one

$$H_\lambda = -\mu \sum_i \lambda_i S_i \ .$$

The idea is that we take an effective site dependent magnetic field λ_i to approximate the true spin dynamics. Of course, the partition function for our trial hamiltonian is exactly solvable

$$Z_\lambda = \mathrm{Tr}\, e^{-\beta H_\lambda} = \prod_i 2\cosh \beta\mu\lambda_i \quad \Rightarrow \quad F_\lambda = -\frac{1}{\beta}\sum_i \ln 2\cosh\beta\mu\lambda_i \ .$$

We finally derive the trial magnetization at the i-th site

$$\langle S_i \rangle_\lambda = m_i = -\frac{1}{\mu}\frac{\partial F_\lambda}{\partial_{\lambda_i}} = \tanh \beta\mu\lambda_i \ . \tag{101}$$ localm

Now it's time to compute the approximated free energy, that we called $\Phi(\lambda)$ in

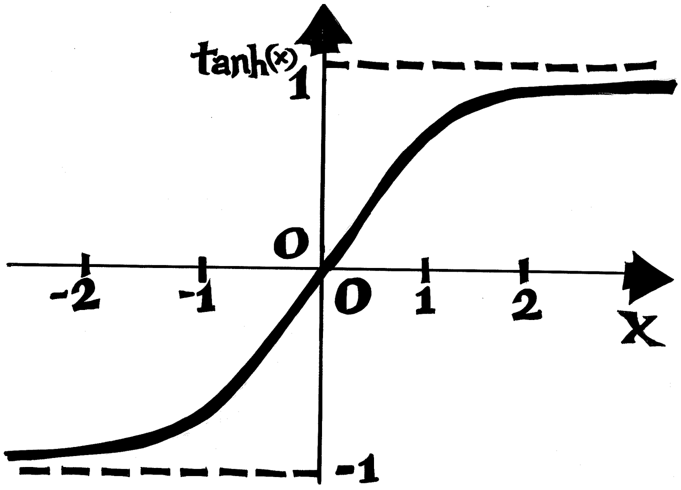

Figure 20: The graph of the magnetization m in (101) as a function of $x = \beta\mu\lambda$. `tanhx`

(100). We first need to evaluate the difference of the true and trial hamiltonians using the trial density matrix. This is easy since in the trial density matrix expectation values of spins just factorize $\langle S_i S_j \rangle_\lambda = \langle S_i \rangle_\lambda \langle S_j \rangle_\lambda = m_i m_j$ and we find

$$\langle H - H_\lambda \rangle_\lambda = -\frac{1}{2}\sum_{ij} J_{ij} m_i m_j - \mu \sum_i (B_i - \lambda_i) m_i \ .$$

Putting the pieces together, the approximate free energy is then just

$$\Phi(\lambda) = -\frac{1}{\beta}\sum_i \ln 2\cosh \beta\mu\lambda_i - \frac{1}{2}\sum_{ij} J_{ij} m_i m_j - \mu \sum_i (B_i - \lambda_i) m_i \ . \qquad (102)$$ `trialf`

The estimate closest to the true free energy will be the minimum of the trial free energy (102), so we need to extremize it. Instead of extremizing it with respect to the effective magnetic field λ_i, we can extremize it with respect to the local magnetization m_i in (101), since the latter is a monotonic function of the former as we can see in Fig. 20

$$\frac{\partial \Phi}{\partial \lambda_i} = 0 \quad \Leftrightarrow \quad \frac{\partial \Phi}{\partial m_i} = 0 \ .$$

Let us extremize the trial free energy

$$\begin{aligned}\frac{\partial\Phi(\lambda)}{\partial m_i} &= \textstyle\sum_j \frac{\partial F_\lambda}{\partial\lambda_j}\frac{\partial\lambda_j}{\partial m_i} - \sum_j J_{ij}m_j - \mu(B_i-\lambda_i) + \mu\sum_j m_j\frac{\partial\lambda_j}{\partial m_i} \\ &= \textstyle -\sum_j J_{ij}m_j - \mu(B_i-\lambda_i)\ , \qquad (103)\end{aligned}$$

where the first and the last term in the first line cancel due to $\overset{\texttt{localm}}{\text{(101)}}$. Imposing $\partial\Phi/\partial m_i = 0$ we get

$$\mu\lambda_i = \mu B_i + \sum_j J_{ij}m_j\ .$$

But since $\lambda_i = \frac{1}{\beta\mu}\tanh^{-1} m_i$, we finally obtain the fundamental equation of mean field theory

$$m_i = \tanh\beta\left(\mu B_i + \sum_j J_{ij}m_j\right)\ . \qquad (104)$$

fundamean

Integrating $\overset{\texttt{difff}}{\text{(103)}}$ we obtain the approximate free energy

$$\Phi = \langle H\rangle_{\mathrm{P}} - TS[\mathrm{P}]\ ,$$

where the energy

$$\langle H\rangle_{\mathrm{P}} = -\frac{1}{2}\sum_{ij}J_{ij}m_im_j - \mu\sum_i B_im_i$$

and the entropy

$$S[\mathrm{P}] = -\sum_i\left(\frac{1+m_i}{2}\ln\frac{1+m_i}{2} + \frac{1-m_i}{2}\ln\frac{1-m_i}{2}\right)$$

are both computed using the probability distribution

$$\mathrm{P} = \prod_i\left(\frac{1+m_i}{2}\delta_{S_i,1} + \frac{1-m_i}{2}\delta_{S_i,-1}\right)\ .$$

6.5.2. Critical exponents

If we take uniform external field $B_i = B$ and uniform couplings $J_{ij} = J$, with nearest-neighbor interactions only in the hamiltonian, the mean field equation $\overset{\texttt{fundamean}}{\text{(104)}}$ reads

$$\tanh^{-1} m - \beta dJm = \beta\mu B\ ,$$

where d is the number of nearest neighbors to each lattice site. We want to find the behavior of the magnetization m near the critical temperature T_c for the phase transition. In this regime, m is small and we can expand $\tanh^{-1} m \simeq m + m^3/3$ and find

$$m(1 - \beta dJ + m^2/3) = \beta\mu B \ ,$$

which gives the spontaneous magnetization

$$\lim_{B\to 0^\pm} m(B) = \begin{cases} \pm\sqrt{3}\sqrt{1-\beta dJ} \ , & \text{for}\, T < T_c \\ 0 \ , & \text{for}\, T > T_c \end{cases}$$

where the critical temperature is $T_c = dJ$. The critical exponent β, not to be confused with the temperature $\beta = 1/T$, is defined as $m \sim |T - T_c|^\beta$ and in this case we have

$$\beta = 1/2 \ .$$

Guide to the literature

A very good text that covers everything from the basics of statistical mechanics and thermodynamics to transport processes, which we pillaged to write this chapter, is lebellac [14]. In dalvit [15] the reader can find a never-ending collection of clever problems with solutions, from which we took the 1d Ising model with long-range interactions.

Memory

7. Associative Memory

chapter-hopfield

The philosophy that I will follow throughout these lectures is the physicist's approach to nature:

- Take a particular phenomenon in its full complexity.
- Separate its salient features from the irrelevant details.
- Create a model that reproduces the basic barebone features of the phenomenon, even if it might look oversimplified and not realistic at first.
- Starting from this skeletal model, add new features that bring it closer and closer to the real world.
- Predict new quantitative observations, adding new features until you get them right.
- Celebrate the achievement with your friends!

Following this simple algorithm, we will start by looking at the problem of associative memory, building an oversimplified model at first, good enough to show that the problem can be formalized successfully in the first place. In this Chapter, we will discuss a model of associative memory, called the Hopfield network. We will first describe the computational task we are after, then we will introduce the network and see that it is a long range Ising model. We will study the equilibrium solution of the network, first heuristically and then providing an exact solution in the case of a finite number of stored patterns.

Suppose I ask you: what is the capital of France? After a few milliseconds, you recall from your archive of stored memories, under the categories *France: capital* the correct answer: Paris. The recovered answer is 100% exact. But suppose now that I ask you what is the capital of Burkina Faso. You might just not know it: no retrieval in this case. Or more likely, you might have heard it or read it once in the past and vaguely remember its sound or a few letters in its word. So your brain is going to scan for a while and try to retrieve its exact sound and spelling, maybe something like “Ugadugu,” or perhaps “Ougagougou”... In this case, you more or less remember it, even if you cannot recall the *exact* answer. The retrieval for the categories *Burkina Faso: capital* is working with some error, but it is still extremely useful if, say, you find yourself in a cab in central Africa and want to go there. By the way, I cannot remember the correct answer myself... try it on the best associative memory database: Google!

Suppose now that you have a geography exam tomorrow, the topic is: all the world’s capitals. You already know some, but you need to take an atlas and study the rest of them. You work very hard and make some progress. But no matter what, there is just no way you can memorize all of them... How many can you remember? In other words, what is the storage capacity, let us call it α, of your associative memory?

In this chapter, we will introduce a model of associative memory, named after John J. Hopfield. It is a simple statistical mechanical model, which retains some basic features of the complicated story we discussed above, yet it is simple enough that we can solve it analytically. Even if it is not a fully realistic model, the fact that it can be solved exactly is very important. I think of it as the theoretical neuroscience version of what the *harmonic oscillator* is in physics. It is rare to find a physical system that behaves as a harmonic oscillator, but many phenomena can be treated as a perturbative expansion around it. Likewise, our brain is not really described by the Hopfield model, but the basic mechanism for memory retrieval might be related to it. Moreover, we can write down computer programs based on the Hopfield network, that perfom amusing tasks like pattern recognition.

7.1. Address-based vs. content-based memory

7.2. Hamming distance

We want to construct a model that solves the following problem. I am traveling along central Africa in a taxi, headed towards the capital of Burkina Faso for an important meeting, but I just cannot remember the name of it. The taxi driver seems to remember it: ”Ouga...something?”, which sounds like a close shot. ”Bingo!”, you shout, recalling suddenly that your business meeting is about establishing a new lottery program. As you listen to the first few letters of the driver’s hint, the correct answer immediately bursts into your head, as if a light bulb suddenly popped in front of your eyes: ”Ougadougou!”.

This is the task we will solve in this Chapter: given a partial clue about the name of a capital city, retrieve the full name. Namely, given an input series of binary bits that are somewhat similar to a certain stored pattern of zeros and ones, we want a system that dynamically recovers the full answer - a.k.a. an error-correcting code.

Our memory consists of a set of p of stored patterns $\{\tilde{\xi}_i^\mu\}$, where the index $\mu = 1, \ldots, p$ labels the patterns, each of which is a vector with N entries that can take values either zero or one, where the index $i = 1, \ldots, N$ denotes the i-th entry in each pattern. This is the set of all the capitals that we know. When the taxi driver gave us a hint, or an *input* N-vector ρ_i, for $i = 1, \ldots, N$, each entry ρ_i taking values either zero or one. How close is the input hint ρ to each of the p stored patterns ξ^μ?

We can compare the input and the stored pattern bit by bit and collect a zero if they agree and a one if they disagree. The sum of all the ones is called the *Hamming distance* between the two strings of bits

$$d(\rho, \xi^\mu) = \frac{1}{N} \sum_{i=1}^{N} [(1-\rho_i)\xi_i^\mu + (1-\xi_i^\mu)\rho_i] \ , \tag{105}$$

hamming

which we normalized such as $d = 0$ when the strings are identical and $d = 1$ when they are complementary. The associative memory task can be rephrased neatly as follows:

Our memory database contains p stored patterns ξ^μ, for $\mu = 1, \ldots, p$. Given a new input pattern ρ, find among the stored patterns ξ^μ the one closest to ρ, according to the Hamming distance.

We need to invent a model that solves this task.

7.3. The Hopfield Model

A horse is a very complicated biological system. However, for simulation purposes, physicists and mathematicians are perfectly fine with modeling a horse with a (topologically equivalent) three-dimensional sphere. In the same spirit, the sophisticated biological systems that we call neurons (they come in many different kinds) will be idealized as follows. In this chapter, a neuron will be a box with N input lines and N output lines. The input lines are the synaptic connections coming in from all the other N neurons in our neural network. The output lines fan out the output to all the other neurons. Each synapse between the presynaptic neuron j and the postsynaptic neuron i will be weighted by a factor w_{ij}, called the *synaptic weight.* Here is a translation between the quantities we would like to study in our brain and their Hopfield net realization:

What a neuron does is: it collects all of the incoming information from all the other neurons $n_j(t)$ at time t and it sums them according to the respective synaptic weights. If the total sum is larger than a certain quantity μ_i, that we call the *threshold* for the neuron i, then its output $n_i(t+1)$ is equal to 1 at time $t+1$. We say that the neuron emits a *spike.* If the total sum is less than the threshold, it gives zero. We can think of our neuron as doing the job of a small computer. Hence, our neural network is a collection of N computers hooked up in parallel. Altogether, we have a parallel supercomputer! This can be conveniently characterized by the following equation

$$n_i(t+1) = \Theta\left(\sum_{j=1}^{N} w_{ij} n_j(t) - \mu_i\right) \ , \tag{106}$$

$\boxed{\texttt{neuron}}$

where $\Theta(x)$ is the step function, equal to one if the argument is positive and equal to zero otherwise.

In the following, we will think of our neural network as a collection of Ising spins $S_i = \pm 1$. Hence, we will translate from our binary variables to the spin variables: $S_i = 2n_i - 1$ and the update rule $(\overset{\texttt{neuron}}{\boxed{106}})$ reads

$$S_i(t+1) = \mathrm{sgn}\left(\sum_{j=1}^{N} w_{ij} S_j(t) - \theta_i\right) \ , \tag{107}$$

$\boxed{\texttt{updatethreshold}}$

where $\theta_i = 2\mu_i - \sum_{j=1}^{N} w_{ij}$ and sgn is the sign function

$$\mathrm{sgn}(x) = \begin{cases} +1, & x \geq 0 \\ -1 & x < 0 \end{cases}$$

In this chapter, we will be considering only equilibrium thermodynamics, namely what happens once the system relaxes to the local minumum of its potential. In any case, we will keep in mind that our update rule (updatethreshold 107) will be such that at each time step t we pick at random one of the neurons and update it, then at the next time step, we will pick another one, and so on. One could choose a probabilistic update rule, namely at each time step every neuron has a certain probability of updating, and the results will not change.

We consider a simplified model in which we set the thresholds $\theta_i = 0$, since they do not really affect the discussion, so our update rule will be

$$S_i(t+1) = \mathrm{sgn}\left(\sum_{j=1}^{N} w_{ij} S_j(t)\right) \ , \qquad (108)$$

update

Suppose now that after studying really hard we have memorized just a single pattern ξ, the capital of France. In order for this pattern to be a fixed point of the dynamical evolution, it must be stable under the update rule (update 108)

$$\xi_i = \mathrm{sgn}\left(\sum_{j=1}^{N} w_{ij} \xi_j\right) \ , \qquad (109)$$

stability

which we call the *stability condition.* If we start with a configuration of neurons ξ, we will simply stay there. Since $\sum_i \xi_i \xi_i = N$, we see immediately that, to achieve such a stable configuration, we can just set the weights equal to

$$w_{ij} = \frac{1}{N} \xi_i \xi_j \ . \qquad (110)$$

onepattern

Next, choose an input configuration $S_i \neq \xi_i$ different from our stored pattern. Let us say that a fraction ϵ of the entries of S are different from the entries of ξ. In our previous binary language, the Hamming distance (hamming 105) between the two patterns is then $d(S, \xi) = \epsilon$. What happens when we apply the update rule (update 108)?

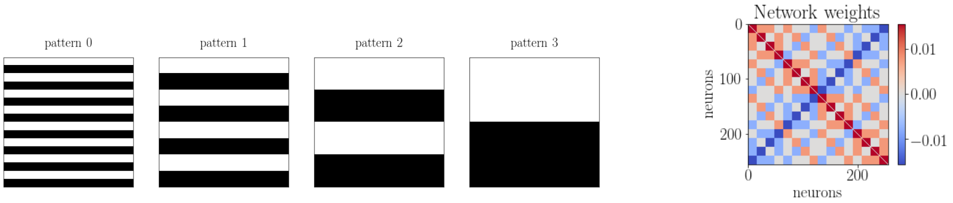

Figure 21: **Orthogonal patterns.** Storing four orthogonal patterns (orthogonalpatterns 114) in the synaptic weights of a Hopfield network with $N = 16^2$ neurons, using the Hebb rule (hopfieldweight 111). Patterns are represented on a 16×16 square grid.

orthogonalpatternsfig

Consider the following quantity, that we can call the *local magnetic field*

$$h_i = \sum_{j=1}^{N} w_{ij} S_j = \frac{1}{N} \sum_{j=1}^{N} \xi_i \xi_j S_j = \xi_i \frac{N - 2\epsilon}{N} \ .$$

If the stored and the input patterns differ for less than half their entries, namely if $\epsilon < N/2$, then $h_i > 0$ and $\mathrm{sgn}\,(h_i) = \mathrm{sgn}\,(\xi_i)$. The update rule then gives back the stored pattern!

$$\xi_i = \mathrm{sgn}\,(h_i) \ .$$

We say that the stored pattern ξ is an *attractor*, namely the network of neurons, given the noisy pattern S, will eventually relax to the correct pattern ξ. Actually, if more than half of the bits are different from ξ than the network will relax in the opposite pattern $-\xi_i$, which is called *reverse state*. Our network therefore has two different attractors ξ and $-\xi$ and its configuration space is divided into two basins of attraction.

We would like to store more patterns, say p of them, in the same memory database. Suppose we start with a *tabula rasa* set of weights J_{ij}, which are all equal to zero. We can add patterns by a simple generalization of (onepattern 110) and, by starting with a *tabula rasa* situation with zero synaptic strengths, change the weights according to

$$w_{ij} \to w_{ij} + \frac{1}{N} \xi_i^\mu \xi_j^\mu \ ,$$

for each pattern we want to add. This is called learning problem and we will study it in full detail later. As long as all the store patterns ξ^μ are orthogonal, this rule makes sure that the patterns themselves are stable under the update rule (108). The weight matrix at the end of the training phase will be given by

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} \xi_i^{\mu} \xi_j^{\mu} \ . \tag{111}$$

hopfieldweight

Are the stored patterns themselves stable under the update rule (108), using the weights (111)? Pick a particular pattern $\xi^{\bar{\nu}}$, then the local field at the unit i, in the configuration $\xi^{\bar{\nu}}$, is

$$h_i = \sum_{j=1}^{N} w_{ij} \xi_j^{\bar{\nu}} = \xi_i^{\bar{\nu}} + \frac{1}{N} \sum_{\mu \neq \bar{\nu}}^{p} \sum_{j=1}^{N} \xi_i^{\mu} \xi_j^{\mu} \xi_j^{\bar{\nu}} \ . \tag{112}$$

hip

The stability condition requires that $\operatorname{sgn}(h_i) = \operatorname{sgn}(\xi_i^{\bar{\nu}})$ for all i. This means that the last term in (112), which we call *crosstalk term*, must be smaller than one, so that it does not flip the sign of h_i and (109) holds

$$\left| \frac{1}{N} \sum_{\mu \neq \bar{\nu}}^{p} \sum_{j=1}^{N} \xi_i^{\mu} \xi_j^{\mu} \xi_j^{\bar{\nu}} \right| < 1 \ . \tag{113}$$

crosstalk

If this condition holds, all the patterns are stable. Moreover, if this term is small, a configuration which is close to a given pattern will converge to it under the update rule. Hence, the patterns are attractor of the network dynamics. Note that if the patterns are actually *orthogonal*, i.e.

$$\frac{1}{N} \sum_{j=1}^{N} \xi_i^{\mu} \xi_i^{\nu} = \delta^{\mu\nu} \ , \tag{114}$$

orthogonalpatterns

as in Fig. (21) then the crosstalk is strictly zero.

If we start with a configuration $\{S_i\}$ close (in Hamming distance) to a certain pattern $\xi^{\bar{\nu}}$, but far from the other patterns $\xi^{\mu \neq \bar{\nu}}$, then we will relax to the close pattern as in Fig. (22). We can easily see this, assuming

$$d(S, \xi^{\bar{\nu}}) = \frac{\epsilon}{2} N << N \ ,$$

$$d(S, \xi^{\mu \neq \bar{\nu}}) = \frac{1-\epsilon}{2} N \ ,$$

with $\epsilon << 1$, we find a local magnetic field

$$h_i = (1-\epsilon)\, \xi_i^{\bar{\nu}} + \epsilon \sum_{\mu \neq \bar{\nu}}^{p} \xi_i^{\mu} \ ,$$

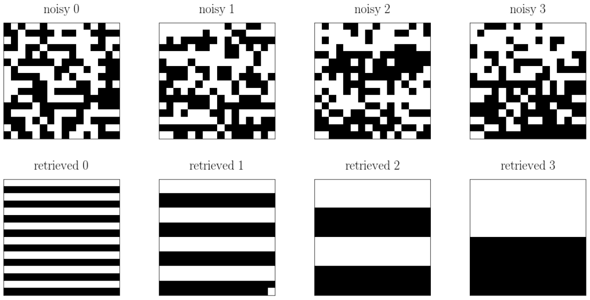

Figure 22: **Orthogonal pattern retrieval**. In the network of Fig. (orthogonalpatternsfig 21), storing 4 orthogonal patterns, we initialized the network with corrupted patterns (30% of the spins were flipped, top row) and we let the network asynchronous (update 108) dynamics evolve until it retrieves the correct patterns (bottom row), showing that the stored patterns are stable fixed points (stability 109). orthogonalpatternsretrieva

and clearly $\operatorname{sgn} h_i = \operatorname{sgn} \xi_i^{\nu}$, provided that p is small enough too. Our main concern now is going to be estimating the range of values of p where the system works as a good memory device.

Dictionary

brain	→	Hopfield network
N neurons	→	N spins S_i
synaptic weights	→	couplings w_{ij}
p memories	→	p stored random patterns ξ^{μ}
memory retrieval state	→	attractor
synaptic noise	→	temperature T

7.3.1. Example

In Fig. (orthogonalpatternsfig 21), we trained a Hopfield network with $N = 16^2$ neurons to store 4 orthogonal patterns in the synaptic weights. When visualizing the activity on a 16×16 square grid, it is clear the patterns are orthogonal. When starting from initial conditions representing a corrupt version of each pattern (where we flipped 30% of spins in each patterns), the network retrieves the correct patterns (Fig. (orthogonalpatternsretrieval 22)).

To illustrate the spectacular robustness of the Hopfield network to brain damage in Fig. (orthogonalpatternsdamage 23), we randomly set to zero 60% of the synaptic weights. We then test memory retrieval in the damaged network by initializing the network in corrupt versions of the patterns (with 10% spin flips). For each initial condition, the network retrieves patterns which have the highest overlap with the corresponding stored patterns, with the following overlap matrix (rows=retrieved patterns, columns=stored patterns):

$$\begin{pmatrix} 0.39 & 0 & 0 & 0.02 \\ -0.02 & 0.40 & 0.02 & -0.06 \\ -0.02 & 0.02 & 0.41 & 0.05 \\ 0.02 & -0.04 & 0.07 & 0.38 \end{pmatrix} \tag{115}$$

damageoverlap

capacity

7.3.2. Storage capacity

How many patterns can we store in our memory system? We will estimate the storage capacity

$$\alpha_c = \frac{p_{max}}{N}$$

by fixing the number of errors that we consider acceptable, and estimating how many patterns p we can store allowing at most such error in the retrieval of a pattern.

The stability of a pattern ξ^μ depends on the size of the crosstalk term (crosstalk 113)

$$C_i^\nu = \frac{1}{N} \sum_{\mu\neq\bar{\nu}}^{p} \sum_{j=1}^{N} \xi_i^\mu \xi_j^\mu \xi_j^\nu \ .$$

If it is larger than one, it can flip the sign of h_i^ν and cause a spin flip, making the i-th bit of the ν-th pattern unstable. Let us estimate the probability that the crosstalk term is larger than one, by assuming that the patterns ξ_i^ν are *random* collections of ±1's, with equal probability of having either signs. The crosstalk term C_i^ν is a sum of $N(p-1)$ random numbers $\pm1/N$. Its distribution is a binomial distribution with zero mean and variance $\sigma^2 = \langle\xi^2\rangle = \sum^{N(p-1)}[\frac{1}{2}(1/N)^2 + \frac{1}{2}(-1/N)^2] = (p-1)/N$. In the limit of large Np, this distribution is well approximated by a normal distribution with

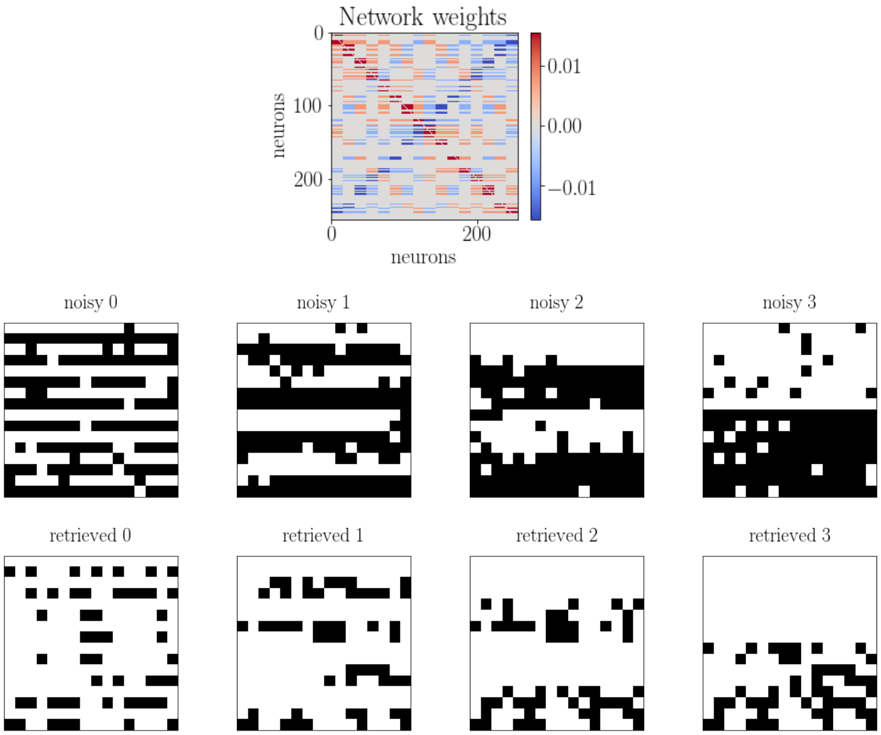

Figure 23: **Damaged brain still functions**. In the network of Fig. (orthogonalpatternsfig 21), storing 4 orthogonal patterns, we erased 60% of the synaptic weights (top) and initialized the network with corrupted patterns (10% of the spins were flipped, center). We let the network asynchronous (update 108) dynamics evolve until it retrieves the correct patterns (bottom), which have the highest overlap (damageoverlap 115) with the corresponding stored patterns despite large brain damage. orthogonalpatternsdamage

the same mean and variance $\sigma = \sqrt{p/N}$

$$P(C_i^\nu) = \frac{1}{\sqrt{2\pi}\sigma} e^{(C_i^\nu)^2/2\sigma^2} \ .$$

If $C_i^\nu > 1$ than the corresponding pattern element is unstable and it will flip. We will consider this an error in the retrieval procedure, since the stable pattern, which is the one we retrieve, is not the original one we stored, but one that differs from it by one unit. Hence, the probability of having one error in a pattern element is given by the probability of having $C > 1$

$$P(C > 1) = \int_1^{+\infty} e^{x^2/2\sigma^2} \frac{dx}{\sqrt{2\pi}\sigma} = \frac{1}{2}[1 - \mathrm{erf}(1/\sqrt{2\sigma^2})] = \frac{1}{2}[1 - \mathrm{erf}(\sqrt{N/2p})] \ ,$$

We define the error function as

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x dz\, e^{-z^2} \ . \tag{116}$$

deferror

If we accept a 1% chance of error in each bit ξ_i^ν, i.e. $P(C > 1) < 0.01$, the largest number p of random patterns we can store is $\alpha = p/N = 0.185$. In Fig.? we can see different estimates of the capacity, obtained by allowing a different percentage of error. This calculation is a gross estimate of the number of bits that will flip, in the stability condition (stability 109), at the first step of the update rule (update 108). But nothing prevents more bits to flip at the second step of the update rule, and so on. So the storage capacity we estimate in this way is just an *upper bound* on the true capacity, and soon we will compute such quantity more reliably.

If instead we require that each single pattern be recalled *perfectly*, say with 99% accuracy, since each pattern contains N bits, we have to require $[1-P(C > 1)]^N < 0.01$ (every bit is independent). In the large N limit, this reads $P(C > 1) < 0.01/N$. This implies $p/N \to 0$, in which regime we can approximate $1 - \mathrm{erf}[x] \sim e^{-x^2}/\sqrt{\pi}x$ and one can easily find the condition

$$\alpha < 1/2 \log N \ .$$

Putting everything together, we see that if we allow a small percentage of error in each bit, the capacity is finite, but if we allow only a small percentage of error in the full pattern in the large N limit, then the capacity is proportional to $1/\log N$.

We estimated the capacity of a Hopfield network with N units storing p *random* patterns (with equal probability of ±1 for each spin). We defined the capacity as the largest value p_{max} where all stored patterns can be retrieved (Fig. (hopfieldcapacity 24)).

spuriousstates

7.3.3. Spurious States

If the only stable states under the update rule (update 108) were just the stored patterns ξ^μ, we would be in great shape: no need for further analysis! However,

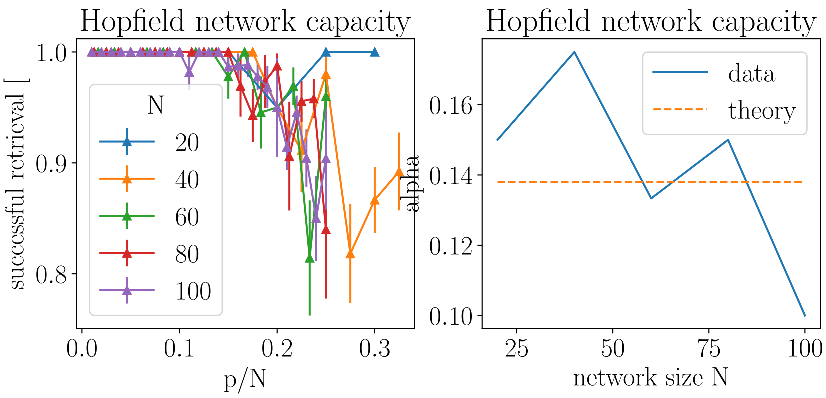

Figure 24: **Network capacity**. In a Hopfield network with N units storing p *random* patterns (with equal probability of ± 1 for each spin), as the fraction $\alpha = p/N$ of stored patterns increases, the network fails to successfully retrieve all patterns starting from initial conditions given by corrupted versions of the each pattern (with 5% spin flips). Network capacity, defined as the largest α_c for which all patterns are successfully retrieved, approaches the theoretical value $\alpha_c = 0.138$. hopfieldcapacity

it turns out that there are several more stable states which are not part of the stored memories: we call these states *spurious states*.

First of all, since the update rule (update 108) are both invariant under the Z_2 spin flip symmetry $S_i \to -S_i$, the reversed pattern $-\xi^\mu$ is also an attractor, with the same exact energy as the parent pattern itself. But we do not regard this as a spurious states. If you remember the capital of Burkina Faso as $-$Ouagadougou instead of just Ouagadougou, you are going to be fine anyway.

But there are plenty of spurious states. The simplest is the combination of three stored patterns in Fig. (orthogonalspurious 26)

$$\xi_i^{mix} = \operatorname{sgn}\left(\pm \xi_i^{\mu_1} \pm \xi_i^{\mu_2} \pm \xi_i^{\mu_3}\right) . \qquad (117)$$ mix

The local field generated by the spurious state is

$$h_i = \frac{1}{N} \sum_{\mu=1}^{p} \sum_{j=1}^{N} \xi_i^\mu \xi_j^\mu \xi_j^{mix} = \frac{1}{2} \xi_i^{mix} + \frac{1}{N} \sum_{\mu \neq \mu_1, \mu_2, \mu_3}^{p} \sum_{j=1}^{N} \xi_i^\mu \xi_j^\mu \xi_j^{mix} ,$$

where we used the fact that the overlap between ξ_i^{mix} and each of its component patterns is $\frac{1}{2}$ on average. But the last crosstalk term is small, if we assume that

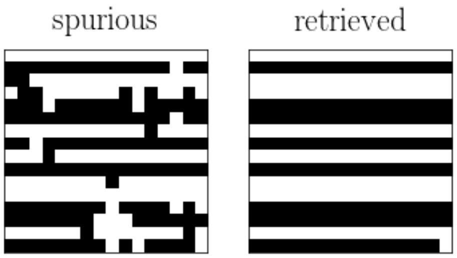

Figure 25: **Retrieving a spurious state**. In the network of Fig. (21), storing 4 orthogonal patterns, we initialize the network in the vicinity of the spurious state $\xi_i^{sp} = \text{sign}(\xi_i^1 - \xi_i^2 + \xi_i^3)$ (10% of spins flipped, left). We let the network asynchronous (108) dynamics evolve until it retrieves the spurious pattern (right), showing that spurious states are attractors at zero temperature.

orthogonalspurious

the patterns are far from each other in Hamming distance, so we find that $\text{sgn}\, h_i = \text{sgn}\, \xi_i^{mix}$ and the spurious states are stable under the update rule. Any combination of an *odd* number of patterns, with either signs, is a spurious state. There are other kind of very subtle spurious states we will discuss later on, called *spin glass states.* There are many more spurious states than actual patterns! We need to make sure that we get rid of all the spurious states, or clearly our memory device is going to be useless. The way to do this is by introducing some kind of *noise* into our memory device. Contrarily to what you may think, a finite amount of noise will lift all of the spurious states, leaving only the pattern and its reverse as the local minima of our Hamiltonian. The way we can describe noise in the language of Ising spins is by introducing a non-zero temperature.

7.3.4. Basins of attraction

A useful way to characterize distances in the space of patterns is in terms of the overlaps between a spin configuration S_i and a stored pattern ξ^μ

$$m^m u = \frac{1}{N}\sum_{i=1}^{N} S_i \xi^\mu \ ,$$

where the overlap is 1 if $S_i = \xi^\mu$ and -1 is $S_i = -\xi^\mu$. We can estimate the basin of attraction of pattern ξ^μ as follows. We initialize the network in the a random pattern S_i with overlap m^μ with ξ^μ. For large overlap, the dynamics converges to the pattern ξ^μ. When the overlap is close to -1, the dynamics will converge to $-\xi^\mu$, which is also an attractor. So, there will be a largest value of the overlap m^μ where the retrieval fails to converge to ξ^μ 50% of the time. We thus define the basin of attraction of pattern ξ^μ to be that value of the overlap m^μ. For fixed network size, as one increases the number of stored pattern p, their average basin of attraction of a stored pattern ξ^μ shrinks, as well as the basin of attraction of $-\xi^\mu$. Basins of attraction of spurious states are always very small at zero temperature, and they vanish when the temperature is above $T_3 = 0.46$ (see below).

7.3.5. Temperature

If we have a non-zero temperature $T = 1/\beta$, our spin variable at site i becomes a probabilistic random variable. The probability of having a value of the spin $S_i = \pm 1$ is proportional to its Boltzmann factor $e^{-\beta\epsilon_i}$. The energy at the i-th site $\epsilon_i = -S_i h_i$ depends on the local magnetic field $h_i = \sum_{j=1}^{N} w_{ij} S_j$. We replace the deterministic update rule ($\overset{\texttt{update}}{108}$) with the probability of having $S_i = \pm 1$, given by

$$\mathrm{P}(S_i = \pm 1) = \frac{e^{\beta h_i S_i}}{e^{\beta h_i S_i} + e^{-\beta h_i S_i}} = \frac{1}{1 + e^{\mp 2\beta h_i}} \ . \tag{118}$$

glauber

Our system at non-zero temperature will be described by the *Glauber dynamics.* We pick a random spin S_i at each update step, and update it according to ($\overset{\texttt{glauber}}{118}$). The expectation value of the spin variable is then

$$\langle S_i \rangle = \mathrm{P}(S_i = +1) - \mathrm{P}(S_i = -1) = \tanh \beta h_i \to_{\beta\to\infty} \mathrm{sgn}\, h_i \ . \tag{119}$$

expects

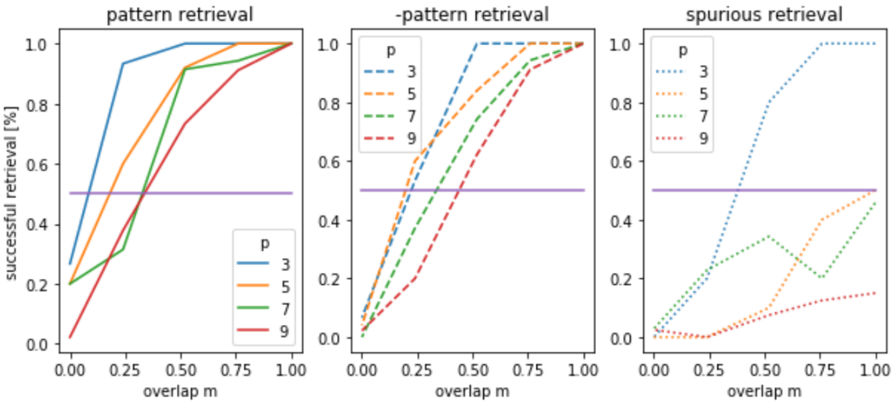

Figure 26: **Basin of attraction of stored patterns**. In a Hopfield network with $N = 50$ units storing p *random* patterns, we estimate the basin of attractions of the stored patterns ξ^μ (left), of their opposites $-\xi^\mu$ (center) and of the spurious states given by $\xi_i^{sp} = \text{sign}(\xi_i^1 + \xi_i^2 + \xi_i^3)$ (right). The size of each basin is defined as the value of the overlap where pattern retrieval success rate drops below 50%. Basins of ξ^μ and $-\xi^\mu$ have the same size, while basins of spurious states are very small. Basins decrease when increasing the number of stored patterns p.

orthogonalspurious

The interpretation of this expectation value is the following. At each instant in time, S_i has a definite value ±1, but due to the disorder effect of the temperature, it flips quickly from up to down, spending more time in the up configuration if the local magnetic field is positive, or in the down configuration otherwise. The average over a long period of time give (expects 119). At $T = 0$, i.e. $\beta \to \infty$, we get back to the deterministic update rule in (update 108).

What do temperature and this new stochastic formalism mean in the context of neural networks? First of all, the "temperature" $T = 1/\beta$ has nothing to do with the real temperature in our brain (that is 37^{o}C). It is simply a formal way to introduce stochastic fluctuations in our neural update rule: it represents the fact that neural signals are *noisy*. The larger the temperature, the more noise in our neural network.

7.4. Heuristic Solution

In this Section, we will study the Hopfield network using a heuristic method, based on mean field theory. Let us start with the expectation value of the spin variable (119) we just derived

$$\langle S_i \rangle = \tanh \beta h_i \ ,$$

but now we are going to replace the actual fluctuating local field $h_i = \sum_{j=1}^{N} w_{ij} S_j$ with its expectation value

$$h_i \to \langle h_i \rangle = \sum_{j=1}^{N} w_{ij} \langle S_i \rangle \ ,$$

so that the mean field dynamics of our system becomes simply

$$\langle S_i \rangle = \tanh \beta \sum_{j=1}^{N} w_{ij} \langle S_j \rangle \ , \qquad w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} \xi_i^{\mu} \xi_j^{\mu} \ . \qquad (120) \qquad \boxed{\texttt{meanfield}}$$

This is still a complicated system of N coupled nonlinear equations, but at least we got rid of all the stochastic complications! The idea is that we are focusing on a single spin at the i-th site, and how this is influenced by the average local field generated by all the surrounding spins, neglecting their fluctuations. We are going to study the stable time-independent configurations of spins $\{S_i\}$. This is the *equilibrium state* of our system, which is guaranteed to exist, due to the presence of an energy function, that we will introduce in Section 7.5.

The mean field procedure turns out to be actually *exact* in the case of long range interactions, just as we saw in Section 6.4. This is roughly speaking because h_i, in the case of infinite range interaction, is a sum over an infinite number of contributions, and a kind of central limit theorem comes into play.

7.4.1. $p/N \to 0$ case

We will consider first the case of $p << N$, namely the number of stored pattern remains fixed in the large N thermodynamic limit. We can recognize immediately the mean field equation (120) as the equation for the ferromagnet (98) at zero external field. Since we saw that $S_i = \xi_i^{\nu}$ is a stable solution at zero temperature, let us make a natural *ferromagnetic ansatz* for the solution at non-zero temperature. This ansatz assumes that our spin configuration is proportional to one of the stored patterns, let us say the first one

$$\text{ferromagnetic ansatz}: \;\; S_i = m\xi_i^1 \;, \tag{121}$$

ferro

where m is the analogue of the spontaneous magnetization. We find

$$m\xi_i^1 = \tanh\beta m \left(\xi_i^1 + \frac{1}{N}\sum_{\mu=2}^{p}\sum_{j=1}^{N} \xi_i^\mu \xi_j^\mu \xi_j^1 \right) \;. \tag{122}$$

finitep

The second term in the argument of the right hand side is the crosstalk term and it vanishes in the large N limit, assuming that the finite number of stored patterns are far from each other in Hamming space. Since $\tanh(-x) = -\tanh x$ we find

$$m = \tanh\beta m \;. \tag{123}$$

firstm

This equation is exactly the same as the one for the spontaneous magnetization of a ferromagnet (wphase 98). It is easy to see that there are non-vanishing solutions for m only below a critical temperature $T < T_c = 1$. The magnetization as a function of the temperature is in Fig.?. This is the smoking gun of a phase transition in the behavior of our system.

What is the meaning of m, the analogue of the spontaneous magnetization, in our associative memory model? Of course, our brain is not literally a magnet, or whenever we remember something, we would see silverware flying accross the room from the kitchen towards our skull. Remembering the definition of the expectation value of the spin (expects 119), we can see that

$$\begin{aligned} m = & \;\; \tfrac{\langle S_i \rangle}{\xi_i^1} = \tfrac{1}{\xi_i^1}\mathrm{P}(S_i = +1) + \tfrac{-1}{\xi_i^1}\mathrm{P}(S_i = -1) \\ = & \;\; \underbrace{\mathrm{P}(S_i = \xi_i^1)}_{\text{correct}} - \underbrace{\mathrm{P}(S_i = -\xi_i^1)}_{\text{wrong}} \;, \end{aligned}$$

where the second line can be checked by explicit substitution. m is the probability of having the correct result minus the probability of making a mistake, which is nothing but one minus the probability of being correct. The probability is the same for every site, so we can sum over N sites and find the average number of corrects bits in our configuration $\{S_i\}$ as

$$\langle N_{correct} \rangle = N \times \mathrm{P}(S_i = \xi_i^1) = \frac{1+m}{2} N \ .$$

Above the critical temperature, $T > T_c$, the spontaneous magnetization vanishes and so our stable configuration will get just half of the bits right, which is what we would expect if we picked just a random pattern. In other words, there is a critical amount of noise, in our large network, above which the system just does not work anymore. But below a certain level of noise, for $T < T_c$, there is a phase transition and we start to improve the reliability of our associative memory very quickly.

The second issue here has to do with the spurious states we found in Section spuriousstates 7.3.3. Whenever the spurious states are stable equilibrium configurations of our network, we are in trouble, because we have a large chance of retrieving a memory that is, in fact, spurious. Hence, we must address the question of their stability, for which we will need some more advanced statistical mechanical techniques, so we postpone this analysis to Section?. For the moment, we just quote the result. It turns out that the spurious states, which are stable configurations at zero temperature, have themselves a critical temperature T_n, where n is the number of patterns we are mixing in (mix 117), in that example $n = 3$. For $T > T_n$, the corresponding spurious state becomes unstable and does not bother us anymore. Luckily, T_n is a monotonically decreasing function of n and the highest temperature $T_3 < T_c$, so that there is a range of temperature

$$T_3 < T < T_c \ ,$$

in which only the retrieval states (and their inverse) are the stable configurations of the network.

We are encountering here for the first time an absolutely crucial aspect of neural networks and, we believe, a fundamental feature of our brain. While a large noise destroys the collective behavior of the network, the presence of a low but significant *noise level* in our brain activity increases dramatically the reliability of our cognitive mechanism. This is going to be one of the *leit motifs* of our exploration.

7.4.2. Storage Capacity

Let us consider now the following question: how many patterns or memories can we store in our device, before the retrieval mechanism starts to break down? In other words, what is the storage capacity $\alpha = p/N$ of our network, in the case in which the number of stored patterns p scales like N?

We gave a rough estimate of the capacity in Section capacity 7.3.2, where we saw that, if we want to recover a pattern with 1% accuracy, the upper bound on the number of stored patterns is below $p_{max} < 0.185N$. Here we will derive a much sharper bound.

Let us recollect our simple derivation in (finitep 122) and see what goes wrong in the case in which the number of stored patterns scales like N. Clearly, in the crosstalk term

$$\frac{1}{N}\sum_{\mu=2}^{p}\sum_{j=1}^{N}\xi_i^\mu\xi_j^\mu\xi_j^1 \ ,$$

the first sum over the number of patterns has order N elements, so it might easily get to a result of order one and potentially reverse the sign in the update rule, and we cannot just neglect it. It requires a more careful analysis.

If we look at the crosstalk term for inspiration on the road to pursue, we can introduce two useful quantities that will simplify our life. The first one is the *overlap* m_ν between the expectation value of our spin variable $\langle S_i\rangle$ and a particular pattern ξ_i^ν

$$m_\nu = \frac{1}{N}\sum_{i=1}^{N}\xi_i^\nu\langle S_i\rangle \ , \tag{124}$$

overlap

for *all* the patterns, not just the one that we are going to (hopefully) retrieve. Suppose our configuration is in the basin of attraction of the first pattern ξ_i^1. Then m_1 is of order unity, as we saw earlier. Let us estimate the order of magnitude of the overlap with the wrong patterns $\xi^{\nu\neq 1}$. $m_{\nu\neq 1}$ is the sum over N random variables (we are assuming here again that our patterns are random variables), each of which can take the values $\pm 1/N$ with equal probability. The computation is the same as for the crosstalk term C_i^ν, except that we do not have now the sum over p patterns that we had there

$$\langle m_{\nu\neq 1}^2 \rangle = \sum_{i=1}^{N} \left[\frac{1}{2} \left(\frac{1}{N} \right)^2 + \frac{1}{2} \left(-\frac{1}{N} \right)^2 \right] = \frac{1}{N} \ .$$

We find a mean value $\langle m_{\nu\neq 1} \rangle = 0$ and a variance $\langle m_{\nu\neq 1}^2 \rangle = \frac{1}{N}$. Note that the Hamming distance (105) between our average configuration $\langle S_i \rangle$ and the "non-retrieved" patterns $\xi^{\nu\neq 1}$ is $d_\nu = \frac{N}{2}(1 - m_\nu)$.

The second useful quantity is the total *mean square overlap* between our spin configuration $\langle S_i \rangle$ and all the "non-retrieved" patterns $\xi_i^{\nu\neq 1}$

$$r = \frac{1}{\alpha} \sum_{\nu\neq 1} m_\nu^2 \ . \tag{125}$$

totaloverlap

Since $m_\nu^2 \sim 1/N$ and $\alpha = p/N$, we expect r to be of order unity, since it is the sum of p terms.

What do we expect to find? There are two possible scenarios:

- *Successful retrieval*: the late time spin configuration $\{S_i\}$ aligns with the pattern ξ_i^1. In this case, according to (121)

$$m_1 \neq 0 \ .$$

- *No retrieval*: the late time spin configuration $\{S_i\}$ does not align with the pattern ξ_i^1. In this case

$$m_1 = 0 \ .$$

The mean square overlap is non-zero because, even if each $m_{\nu\neq 1} \propto 1/N \to 0$, the sum in (125) involves p positive terms and in our case p scales like N.

We shall now compute the largest value of p above which no successful retrieval can happen.

Let us perform now the computation of the maximum storage capacity α of the system, by starting again from the mean field equation (120)

$$\langle S_i \rangle = \tanh \left(\frac{\beta}{N} \sum_{\mu=1}^{p} \sum_{j=1}^{N} \xi_i^\mu \xi_j^\mu \langle S_i \rangle \right) = \tanh \beta \sum_{\mu=1}^{p} \xi_i^\mu m_\mu \ . \tag{126}$$

meanf

We assume that the retrieved pattern is again ξ_i^1. We will first compute $m_{\nu\neq 1}$ and then use it to estimate the total square overlap with the wrong patterns r. Then we will derive m_1, the overlap with the retrieved pattern.

We take the sum $(1/N)\sum_i \xi_i^\nu$ of both sides of (126)

$$m_\nu = \frac{1}{N}\sum_{i=1}^{N} \xi_i^\nu \tanh\beta\left(\sum_{\mu=1}^{p} \xi_i^\mu m_\mu\right) \ , \tag{127}$$

meanmnu

and, using $(\xi_i^1)^2 = 1$ and $\tanh x\,\xi_i = \xi_i \tanh x$ we find, in the case where the index $\nu \neq 1$

$$m_{\nu\neq 1} = \frac{1}{N}\sum_{i=1}^{N} \xi_i^{\nu\neq 1}\xi_i^1 \tanh\beta\big(\underbrace{m_1}_{\mathcal{O}(1)} + \underbrace{\xi_i^{\nu\neq 1}\xi_i^1 m_{\nu\neq 1}}_{\mathcal{O}(1/\sqrt{N})} + \underbrace{\sum_{\mu\neq 1,\nu}^{p} \xi_i^\mu \xi_i^1 m_\mu\big)}_{\mathcal{O}(\sqrt{N})} \ .$$

We splitted the argument into three terms. The first one is the overlap with the retrieved pattern and is of order one. The second one is the overlap with the "non-retrieved" ν, the same as the l.h.s., and we saw that this is of order $1/\sqrt{N}$, so it is small in the large N limit. The third term scales as $p/\sqrt{N}$ which is large, since $p \sim N$. Hence we can expand the r.h.s. using $\tanh(x+\epsilon) = \tanh x + \epsilon(1-\tanh^2 x) + \mathcal{O}(\epsilon^2)$, where x is equal to the first and third term and ϵ is the second term

$$\begin{aligned} m_\nu = & \quad \tfrac{1}{N}\textstyle\sum_{i=1}^{N} \xi_i^\nu \xi_i^1 \tanh\beta(m_1 + \sum_{\mu\neq 1,\nu}^{p} \xi_i^\mu \xi_i^1 m_\mu) \\ & + \tfrac{\beta}{N}\textstyle\sum_{i=1}^{N} m_\nu \Big(1 - \tanh^2\beta(m_1 + \sum_{\mu\neq 1,\nu}^{p} \xi_i^\mu \xi_i^1 m_\mu)\Big) + \mathcal{O}(m_\nu^2) \ , \end{aligned} \tag{128}$$

Let us consider the last term in (128). We are going to make a crucial assumption that will drastically simplify our duty. We assume that both the stored patterns $\xi^{\nu\neq 1}$ and the overlaps $m_{\nu\neq 1}$ are *random variables.* The overlaps m_ν have mean zero and variance $\alpha r/p$, where we are using the definition (125). The quantity $\xi_i^1\xi_i^{\nu\neq 1}$ is also a random variable, so the expression $\Xi_i = \sum_{\mu\neq 1,\nu}^{p} \xi_i^\mu \xi_i^1 m_\mu$ is in turn a random variable with mean zero and variance αr, since we are summing $p-2 \sim p$ terms, each of which has mean zero and variance $\alpha r/p$. Let us make use of the celebrated

Central limit theorem: Given a distribution of random variables $\{\Xi_i\}$ with mean μ and variance σ^2, the sample average $\frac{1}{N}\sum_{i=1}^{N}\Xi_i$ of the random variables approaches a Gaussian (normal) distribution with same mean and variance, in the limit $N \to \infty$.

Because of the central limit theorem the *site average* $\frac{1}{N}\sum_{i=1}^{N}$ can then be recast into an average over a Gaussian ditribution with the same mean and variance as the variables we were going to average over sites. In other words we will replace

$$\frac{1}{N}\sum_{i=1}^{N} f(\Xi_i) \to \int \frac{dz}{\sqrt{2\pi\sigma}} e^{z^2/2\sigma^2} f(z) = \int \frac{dz}{\sqrt{2\pi}} e^{z^2/2} f(\sigma z)\ , \tag{129}$$

gaussian

since our random variable $\Xi_i = \sum_{\mu\neq 1,\nu}^{p} \xi_i^\mu \xi_i^1 m_\mu$ has zero mean and variance $\sigma^2 = \alpha r$. Hence the last term in $(\overset{\texttt{mnu}}{128})$ can be recast into

$$\frac{\beta}{N}\sum_{i=1}^{N} m_\nu \left(1 - \tanh^2 \beta(m_1 + \sum_{\mu\neq 1,\nu}^{p} \xi_i^\mu \xi_i^1 m_\mu)\right) \to \beta m_\nu (1-q)\ ,$$

where we introduced for convenience the notation

$$q \equiv \int \frac{dz}{\sqrt{2\pi}} e^{-z^2/2} \tanh^2 \beta(m_1 + \sqrt{\alpha\, r} z)\ . \tag{130}$$

defq

Hence the final expression for $m_{\nu\neq 1}$ in $(\overset{\texttt{mnu}}{128})$ becomes

$$m_\nu = \frac{\frac{1}{N}\sum_i \xi_i^\nu \xi_i^1 \tanh \beta(m_1 + \Xi_i)}{1 - \beta(1-q)}\ , \tag{131}$$

mnutwo

where again $\Xi_i = \sum_{\mu\neq 1,\nu}^{p} \xi_i^\mu \xi_i^1 m_\mu$.

Now we want to compute the total overlap r with the "non-retrieved" patterns $(\overset{\texttt{totaloverlap}}{125})$. We just take the square of $(\overset{\texttt{mnutwo}}{131})$ and sum.

$$\begin{aligned} r = &\quad \tfrac{1}{\alpha}\textstyle\sum_{\nu\neq 1} m_\nu^2 \\ = &\quad \tfrac{1}{\alpha}\tfrac{1}{[1-\beta(1-q)]^2}\textstyle\sum_{\nu\neq 1}\left[\tfrac{1}{N^2}\sum_i\sum_j \xi_i^\nu \xi_i^1 \xi_j^\nu \xi_j^1 \tanh\beta(m_1+\Xi_i)\tanh\beta(m_1+\Xi_j)\right] \\ = &\quad \tfrac{1}{\alpha}\tfrac{1}{[1-\beta(1-q)]^2}\textstyle\sum_{\nu\neq 1}\tfrac{1}{N^2}\sum_i \tanh^2\beta(m_1+\Xi_i)\ , \end{aligned}$$

where in the last line we used the fact that only the terms $i = j$ in the double sum survive. This is because ξ^ν does not appear in the argument of the tanh, so we can average the $\xi_i^\nu \xi_i^1 \xi_j^\nu \xi_j^1$ factor outside of the tanh separately. Since the i's and j's are non-correlated random patterns, only the $i = j$ term survives (all the patterns square to one). Finally we can replace the last sum $\frac{1}{N}\sum_i f(\Xi_i)$ with the average over the Gaussian distribution ($\overset{\texttt{gaussian}}{\text{129) and}}$, after using ($\overset{\texttt{defq}}{130}$), end up with

$$r = \frac{q}{[1-\beta(1-q)]^2} \ . \tag{132}$$ $\boxed{\texttt{defr}}$

We can easily derive the overlap m_1 with the retrieved pattern from the mean field equation ($\overset{\texttt{meanmnu}}{\text{127) by}}$ using ($\overset{\texttt{gaussian}}{129}$)

$$m_1 = \int \frac{dz}{\sqrt{2\pi}} e^{-z^2/2} \tanh\beta(m_1 + \sqrt{\alpha\, r} z) \ . \tag{133}$$ $\boxed{\texttt{defmone}}$

$\boxed{\texttt{lowT}}$

7.4.3. Storage Capacity at Low Temperature

Let us summarize what we have obtained so far. We computed the overlap m_1 between the spin configuration and the retrieved pattern and the total overlap r between the spin configuration and all the other "non-retrieved" patterns. We can scan the values of α for which the coupled equations ($\overset{\texttt{defr}}{132}$) and ($\overset{\texttt{defmone}}{133}$) admit solutions. This is going to give us the storage capacity. In the limit $T \to 0$ or $\beta \to \infty$, our expressions simplify considerably.

By recalling the fact that $\partial_z \tanh z = 1 - \tanh^2 z$ and integrating by parts, we find

$$\begin{aligned} 1 - q &= \int \frac{dz}{\sqrt{2\pi}} e^{-z^2/2} [1 - \tanh^2\beta(m_1 + \sqrt{\alpha\, r} z)] \\ &= \int \frac{dz}{\sqrt{2\pi}} \frac{1}{\sqrt{\alpha r}} z e^{-z^2/2} \tanh\beta(m_1 + \sqrt{\alpha r} z) \ . \end{aligned}$$

Since for $a \to \infty$

$$1 - \tanh^2 a(x+b) \to \frac{2}{a}\delta(x+b) \ ,$$

we find

$$\beta(1-q) \simeq \sqrt{\frac{2}{\pi\alpha r}} e^{-m_1^2/2\alpha r} \equiv C \ .$$

Clearly, $q \to 1$ and

$$r \simeq \frac{1}{(1-C)^2} \ , \tag{134}$$ $\boxed{\texttt{rzero}}$

while, using $\tanh a(x+b) \to \mathrm{sgn}\,(x+b)$ for $a \to \infty$, we find

$$m_1 \simeq \int \frac{dz}{\sqrt{2\pi}} e^{-z^2/2} \mathrm{sgn}\,(m_1 + \sqrt{\alpha r} z) \ ,$$

which gives

$$m_1 \simeq \mathrm{erf}(m_1/\sqrt{2\alpha r}) \ . \tag{135}$$ $\boxed{\texttt{monezero}}$

where the error function was defined in (deferror 116). Collecting (rzero 134) and (monezero 135) we conclude that in the limit of zero temperature our order parameters behave as

$$m_1 \simeq \mathrm{erf}(m_1/\sqrt{2\alpha r}) \ ,$$

$$r \simeq 1 + \sqrt{\frac{2}{\pi\alpha r}} e^{\frac{-m_1^2}{2\alpha r}} \ . \tag{136}$$

Let us derive a simple expression from these two coupled equations. We can divide the first equation by the square root of the second

$$\frac{m_1}{\sqrt{r}} = \frac{\mathrm{erf}(m_1/\sqrt{2\alpha r})}{1 + \sqrt{\frac{2}{\pi\alpha}} e^{\frac{-m_1^2}{2\alpha r}}} \ ,$$

and introduce the new variable $y = m_1/\sqrt{2\alpha r}$. We obtain the equation

$$y = \frac{\mathrm{erf}\, y}{\sqrt{2\alpha} + \frac{2}{\sqrt{\pi}} e^{-y^2}} \ . \tag{137}$$

finaleq

This is going to be interpreted as an implicit equation for y, parametrical in the storage capacity $\alpha = p/N$. We plotted in Fig. alphac 27 the graphical solution to (finaleq 137), where we divided both sides by y. This way it is easier to find the solutions. The plot shows the left. Depending on the value of α, equation (finaleq 137) can have either one or three solutions:

- For any value of $\alpha > 0$, we have a solution at $y = 0$, for which $m_1 = 0$ and $r = 1$, namely we do not have any retrieval of patterns, but just an overlap between our stable configuration and all the patterns.

- Below a critical value of the storage capacity $\alpha < \alpha_c$, where (numerically)

  $$\alpha_c \simeq 0.138 \ , \tag{138}$$

  alphac

  there are two more solutions $y_1 < y_2$. The largest of the two is a stable local minimum and represents a correct retrieval state with $m_1 \neq 0$.

Even if for $\alpha > \alpha_c$ the retrieval vanishes, $m_1 = 0$, as we approach the critical value of the storage capacity $\alpha = \alpha_c$, there is a dramatic improvement in the efficiency of the retrieval, which jumps discontinuously from zero to $m \simeq 97\%$. This means that we go discontinuously from a very good associative memory below α_c, to a completely useless memory as soon as we reach α_c.

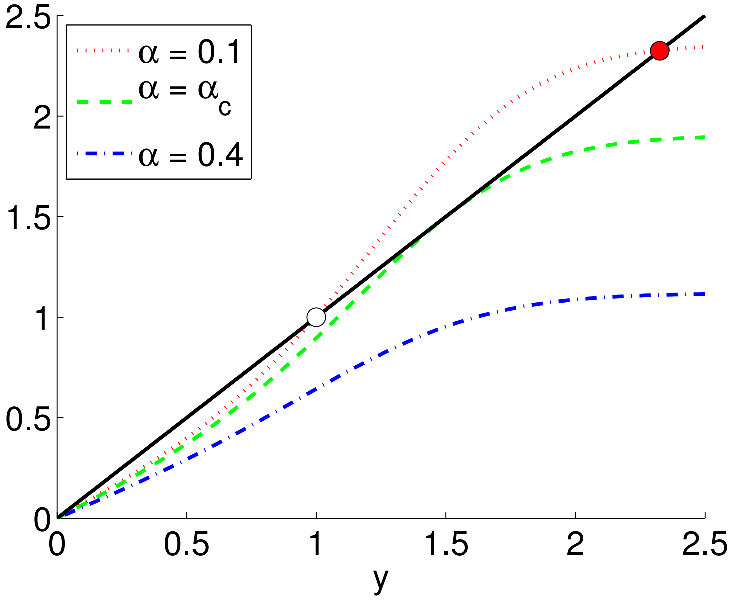

Figure 27: Graphical solution to (finaleq 137). The black solid line represents its left hand side; the dotted, dashed and dashdotted lines represent the right hand side for different values of the capacity α. If $\alpha < \alpha_c$, (finaleq 137) admits two solutions, the full circle represents the stable retrieval state while the empty circle represents an unstable solution. For $\alpha > \alpha_c$ the only solution is the zero overlap state $y = 0$. alphac

energy

7.5. Absence of Spurious States

So far we have derived an estimate of the maximum storage capacity of the Hopfield network, using mean field arguments. But we have not addressed yet the question of the efficiency of the network in performing the task of correct retrieval of stored memories. The stored memories are attractor points of the spin configuration. However, if there are other stable or metastable attractors, which are not stored patterns but just spurious states, this would jeopardize the whole memory endeavor. Starting with our input spin configuration $\{S_i\}$, we would end up in a spurious attractor instead of a stored pattern. Hence, the analysis of the spurious states is of paramount importance and we will address it now.

In order to properly address the stability of spurious states, we need to recast the Hopfield network into a proper statistical mechanics system. The endpoint ($\overset{\texttt{finaleq}}{137}$) of our derivation of the storage capacity using mean field theory is not going to be affected by this more formal derivation. Our goal is instead to look for a range of parameters where the retrieval state is stable, while the spurious states are all unstable.

This is the main insight of Hopfield's, who introduced a *Hamiltonian*, or energy function, into the game. The crucial properties we require the Hamiltonian to satisfy are

- It has *minima*, at least local, when a configuration is equal to one of the stored patterns.

- It *decreases*, or remains constant, as the system evolves according to its dynamical rule (which we called update rule).

If we can come up with such a function, then we can start from an initial configuration $\{S_i\}$, let the configuration evolve according to its dynamical rule, and finally end up in the stored pattern at equilibrium, which are true *attractors* of the system, minima of the free energy. For this last step to occur, we need to make sure that the only (local) minima are the stored patterns, namely that *there are no spurious minima.*

What is our first guess for the Hamiltonian? Clearly, we can just take the Hamming distance between our initial configuration $\{S_i\}$ and the stored pattern ξ_i^μ. If we rewrite the Hamming distance ($\overset{\texttt{hamming}}{105}$), which was originally defined for binary variables with values $(0, 1)$, into a distance for our spin ±1 variables, we find that

$$d(S, \xi) = -\frac{1}{2}\sum_{i=1}^{N} S_i \xi_i^\mu + \frac{N}{2} \ .$$

We can remove the irrelevant constant term and take our candidate Hamiltonian to be simply the square of such Hamming distance

$$H = -\frac{1}{2N}\sum_{\mu=1}^{p}(\sum_{i=1}^{N} S_i \xi_i^\mu)^2 + \frac{p}{2} = -\frac{1}{2}\sum_{i,j} w_{ij} S_i S_j \ , \qquad (139) \qquad \boxed{\texttt{hamone}}$$

We are guaranteed that the stored patterns ξ^μ are local minima. Let us check that it is a decreasing function of our spin flip update rule. Under the update rule

$$S_i' = \text{sgn}\left(\sum_{i=1}^{N} w_{ij} S_j\right) \ ,$$

suppose we have a particular spin flip $S_{\bar{i}}' = -S_{\bar{i}}$, we find

$$\begin{aligned} H(S') - H(S) = & \ -(S_{\bar{i}}' - S_{\bar{i}}) \textstyle\sum_{j\neq\bar{i}} w_{\bar{i}j} S_j \\ = & \ \ 2S_{\bar{i}} \textstyle\sum_{j\neq\bar{i}} w_{\bar{i}j} S_j < 0 \ , \end{aligned}$$

since $\text{sgn}\left[\sum_{j\neq\bar{i}} w_{\bar{i}j} S_j\right] = -1$. The energy decreases at each spin flip, given the dynamical rule (update 108). In order for such a *Lyapunov function* to exist, we must require that the weights are symmetric

$$w_{ij} = w_{ji} \ .$$

This is not a realistic feature, since neurons in the brain do not have symmetric couplings among them. However, having a Hamiltonian simplifies dramatically the computation, so we will stick to this requirement for the time being. Another feature of the weights is that we set to zero the self couplings $w_{ii} = 0$. This can be easily achieved by simply shifting the Hamiltonian by a constant term proportional to p.

Since our couplings w_{ij} can be of either signs, we expect the ground states of our system to be complicated. In fact, the Hopfield network, with random patterns, is an example of a *spin glass*, a very interesting condensed matter system that displays magnetic frustration and a lot of amusing features, among which we mention the *ultrametric* structure of its spin glass states.

7.5.1. Free Energy

In order to study the stability of the spurious states, we need to compute the free energy density of the Hopfield network. The hamiltonian is

$$H = -\frac{1}{2N}\sum_{\mu=1}^{p}(\sum_{i=1}^{N} S_i \xi_i^\mu)^2 + \frac{p}{2} - \sum_{\mu=1}^{p} h^\mu \sum_{i=1}^{N} \xi_i^\mu S_i \ , \qquad (140)$$

hopfieldham

where h^μ is an external field that will be useful in interpreting the overlap functions. The constant shift $p/2$ removes the self couplings in the weight matrix, setting effectively $w_{ii} = 0$. We are going to compute the free energy per unit spin

$$-\frac{1}{\beta N}\langle\langle \ln Z_\xi \rangle\rangle \ ,$$

where the notation $\langle\langle \ldots \rangle\rangle$ means that we are averaging it over the distribution of stored patterns ξ_i^μ, that we take to be *random variables.* The partition function

$$Z_\xi = \mathrm{Tr}_{\{S_i\}} e^{-\beta H[S_i]} \ ,$$

is computed with the hamiltonian (140), at fixed patterns ξ. Let us evaluate

$$Z_\xi = e^{-\beta p/2}\mathrm{Tr}_{\{S_i\}} \prod_{\mu=1}^{p} \exp\left(\frac{\beta}{2N}(\sum_{i=1}^{N} S_i \xi_i^\mu)^2 + \beta h^\mu \sum_{i=1}^{N} \xi_n^\mu S_i\right) \ . \qquad (141)$$

defz

We will use the same tricks that led us to the free energy of the long range Ising model in Section 6.4. First, it is hard to take the trace since the spin variables appear quadratically in the exponential. Let us use the Gaussian integration trick (94) to linearize this term, at the price of introducing a new set of p auxiliary variables m^μ

$$Z_\xi = e^{-\beta p/2} \int d\vec{m} \sqrt{\frac{\beta N}{2\pi}} \mathrm{Tr}_{\{S_i\}} \exp\left[-\frac{\beta N}{2}\vec{m}^2 + \beta(\vec{m} + \vec{h}) \sum_{i=1}^{N} \vec{\xi}_i S_i\right] \ ,$$

where we introduced the shorthand notation $\vec{m} = (m^1, \ldots, m^p)$ to denote a vector with p components. We can easily perform the trace over spins since they appear just linearly

$$\mathrm{Tr}_{\{S_i\}} \exp\left[\beta(\vec{m} + \vec{h}) \sum_{i=1}^{N} \vec{\xi}_i S_i\right] = \prod_{i=1}^{N} 2\cosh\beta(\vec{m} + \vec{h})\vec{\xi}_i \ .$$

Bringing this factor to the exponential we get

$$\begin{aligned} Z_\xi = & \left(\tfrac{\beta N}{2\pi}\right)^{p/2} \textstyle\int d\vec{m}\, e^{-\beta N f(m^\mu)} \ , \\ f(m^\mu) = & \tfrac{\alpha}{2} + \tfrac{1}{2}\vec{m}^2 - \tfrac{1}{\beta N} \textstyle\sum_{i=1}^{N} \ln\cosh\beta(\vec{m} + \vec{h}) \cdot \vec{\xi}_i \ , \qquad (142) \end{aligned}$$

At this point, note that the integrand in the partition function is an exponential whose argument has an overall factor of N. Since we are only interested in the large N limit, we can therefore evaluate this integral by the saddle point method, just as we did in the case of (95). The free energy per unit spin f in the large N limit is given by

$$-\frac{1}{\beta N}\langle\langle \ln Z_\xi \rangle\rangle = \langle\langle \min_{m^\mu} f(m^\mu) \rangle\rangle \ . \tag{143}$$

freenergy

In other words, we need to evaluate the function $f(m^\mu)$ at its minimum. Its stationary points are the solutions to the saddle point equation

$$\frac{\partial f(m^\mu)}{\partial \vec{m}} = 0 \ ,$$

which reads

$$\vec{m} = \frac{1}{N}\sum_{i=1}^{N} \vec{\xi}_i \tanh \beta(\vec{m}+\vec{h})\cdot\vec{\xi}_i \ , \tag{144}$$

saddle

which coincides with the mean field equation (127) we derived heuristically, in the case of zero external field $\vec{h}=0$. In the case of long range interactions, mean field theory is often an exact statement. The external field $\vec{h}$ is useful to derive the meaning of our auxiliary variables $\vec{m}$. By taking the derivative of the free energy f with respect to $\vec{h}$ we find, by using the definition of the partition function (141)

$$-\frac{1}{\beta N}\left.\frac{\partial \ln Z_\xi}{\partial \vec{h}}\right|_{\vec{h}=0} = -\frac{1}{N}\sum_{i=1}^{N}\langle S_i\rangle\vec{\xi}_i \ .$$

On the other hand, by computing the same variation of the final expression of free energy

$$-\frac{1}{\beta N}\left.\frac{\partial \ln Z_\xi}{\partial \vec{h}}\right|_{\vec{h}=0} = -\frac{1}{N}\sum_{i=1}^{N}\vec{\xi}_i \tanh\beta\vec{m}\cdot\vec{\xi}_i = -\vec{m} \ ,$$

where we used the saddle point value (144). Hence, we see that the p auxiliary variables $\vec{m}$ describe precisely the overlap between the spin configuration and the p stored patterns. They are the same quantities we introduced in (124). From now on we will set to zero the external field $\vec{h}=0$, since it served its purpose.

Let us consider the saddle point equation (144) for the overlap functions. At finite N, the expression in (144) depends on the particular realization of the stored patterns ξ_i^μ. However, in the large N limit the random fluctuations are suppressed and, thanks to the sum $(1/N)\sum_i$, the physical quantities *self-average*. If the number p of stored patterns is much smaller than the number of terms in the sum, namely in the regime $N >> 2^p$, we can then replace everywhere the sum over the patterns with their expectation value with respect to a distribution with same mean and variance, as we did above in ($\overset{\texttt{gaussian}}{129}$). Once we do this, we are automatically taking care of the "averaging over the patterns", denoted by $\langle\langle\ldots\rangle\rangle$ in ($\overset{\texttt{freenergy}}{143}$).

Summarizing, the solution to the Hopfield network is given by the mean field equation for the overlap and the free energy density

$$\begin{aligned} \vec{m} = & \quad \langle\langle\vec{\xi}\tanh\beta\vec{m}\cdot\vec{\xi}\rangle\rangle \ , && (145)\\ f = & \quad \tfrac{1}{2}\vec{m}^2 - \tfrac{1}{\beta}\langle\langle\ln 2\cosh\beta\vec{m}\cdot\vec{\xi}\rangle\rangle \ , && (146)\end{aligned}$$

where

$$\langle\langle g(\vec{\xi})\rangle\rangle = \int d\vec{\xi}\mathrm{P}(\vec{\xi})g(\vec{\xi}) \ . \qquad (147) \qquad \boxed{\texttt{average}}$$

At fixed N, the distribution of the stored random patterns is

$$\begin{aligned} \mathrm{P}(\{\xi_i^\mu\}) = & \quad \textstyle\prod_{\mu=1}^p\prod_{i=1}^N \mathrm{p}(\xi_i^\mu) \ , && (148)\\ \mathrm{p}(\xi_i^\mu) = & \quad \tfrac{1}{2}\delta(\xi_i^\mu - 1) + \tfrac{1}{2}\delta(\xi_i^\mu + 1) \ . \end{aligned}$$

In the large N limit, the probability distribution becomes

$$\mathrm{P}(\xi) = e^{-\vec{\xi}^2/2}/\sqrt{2\pi} \ . \qquad (149) \qquad \boxed{\texttt{patterns}}$$

as in ($\overset{\texttt{gaussian}}{129}$).

7.5.2. Solutions

There are many solutions of the saddle point equation ($\overset{\texttt{meq}}{145}$). Above $T > T_c = 1$, the only stable state is the disordered state with $\vec{m} = 0$. Below the critical temperature $T < T_c$, a plethora of new solutions appear as we lower the temperature. These new solutions have their own critical temperatures, below which they become stable solutions. Here we are in the regime $\alpha = p/N \to 0$ of finite number of stored patterns. However, the stability of the various solutions depends crucially on both the value of α and the temperature. In fact, the Hopfield network phase diagram can be nicely drawn as a function of $\alpha(T)$, as we will see at the end of the Chapter. The solutions at finite p are:

- The "retrieval" state in which our configuration overlaps only with the retrieved pattern, say ξ^1

  $$\vec{m} = (m, 0, \ldots, 0) \ .$$

  The saddle point equation ($\overset{\texttt{meq}}{145}$) reads in this case

  $$m = \langle\langle \xi^1 \tanh \beta m \xi^1 \rangle\rangle \ .$$

  Since the series expansion of the tanh only contains odd powers of its argument, we can easily perform the average over the pattern ξ^1. By applying ($\overset{\texttt{average}}{147}$) with the distribution in ($\overset{\texttt{patterns}}{149}$) we find

  $$m = \tanh \beta m \ ,$$

  which we already discussed in ($\overset{\texttt{firstm}}{123}$). The retrieval state appears as we lower the temperature below the critical one $T < T_c = 1$. The equilibrium spin configuration is aligned with the retrieved pattern

  $$\langle S_i \rangle = \xi_i^1 \tanh \beta m \ .$$

- The spurious states in which our configuration overlaps with more than one stored pattern; we will denote them as "symmetric states". We denote by m_n the overlap state with n different patterns

  $$\vec{m} = m_n(\underbrace{1, \ldots, 1}_{n}, \underbrace{0, \ldots, 0}_{p-n}) \ . \qquad (150) \qquad \boxed{\texttt{symmetricm}}$$

  For instance, in ($\overset{\texttt{mix}}{117}$) above we used the $n = 3$ symmetric state. These states have a $2^n \binom{n}{p}$ degeneracy, where the 2^n factor is due to the sign flip $\pm m$ and the binomial coefficient takes care of the different ways to stick

n entries in p bins. The saddle point equation ($\overset{\texttt{meq}}{145}$) for this state reads

$$
\begin{aligned}
m^\mu &= \langle\langle \xi^\mu \tanh \beta m_n \sum_{\nu=1}^{n} \xi^\nu \rangle\rangle \ , &\qquad &\text{for}\, \mu \le n \ , \\
m^\mu &= 0 \ , & &\text{for}\, \mu > n \ ,
\end{aligned}
$$

Let us introduce the following quantity

$$z_n = \sum_{\nu=1}^{n} \xi^\nu \ .$$

It is a sum over n random variables ξ^ν, each of which comes from the distribution ($\overset{\texttt{probxi}}{148}$). Hence, z has a binomial distribution

$$p(z_n) = 2^{-n} \binom{n}{k} \ , \qquad k = (z_n + n)/2 \ , \tag{151}$$

zdistribution

where k is the number of positive entries in the sum that defines z. The saddle point equation reads

$$m_n = \frac{1}{n} \langle\langle z_n \tanh \beta m_n z_n \rangle\rangle \ , \tag{152}$$

mneq

where the average is taken over the distribution of z. These "symmetric state" solutions exist in the whole range of temperature $0 < T < T_c = 1$. These configurations overlap equally with n different patterns and give a wrong retrieval

$$\langle S_i \rangle = \tanh \beta m_n z_n \ .$$

- The "asymmetric states" which are given by

$$\vec{m} = (\underbrace{m, \ldots, m}_{2k}, \underbrace{\epsilon, \ldots, \epsilon}_{l}, \underbrace{0, \ldots, 0}_{p-2k-l}) \ , \tag{153}$$

asymmetric

that we will briefly describe later. There are also solutions with more than two different kind of entries. The transition temperature for the appearance of asymmetric solutions, in which some of the n components have different magnitudes, are all lower than T_c. We quote from $\overset{\texttt{amit85}}{[16]}$: "The appearance of these additional solutions becomes apparent by following the change in stability of the symmetric saddle points $(\overset{\texttt{mneq}}{152})$ as the temperature is reduced. When a particular saddle point changes stability in a certain direction, it does not usually exchange stability with another existing symmetric saddle point, which lies in that direction. Instead, a new, asymmetric saddle point between the two existing saddle points appears. The highest temperature in which a change in stability occurs is when the $n = 2$ symmetric solution becomes unstable to the mixing of more memories. The eigenvalue that controls this stability is λ_2." This is how these spurious states appear.

- In the regime in which $\alpha = p/N \neq 0$, there is a new phase at low temperature, which is called *spin glass*. The spin glass states are a proliferating number of degenerate minima, that become infinite in the limit of large N. They come from the "symmetric states" in $(\overset{\texttt{symmetricm}}{150})$ in the limit in which n and p become very large. These states are characterized by a very low overlap with all the stored patterns. They are ordered states, but clearly spurious ones.

Now that we listed the solutions to the mean field equations, we can proceed to study their stability properties. We want to see if there is a range of parameters in which the good retrieval states are the only stable configurations of the Hopfield network.

sec:stability

7.5.3. Stability

A solution is stable if it is a minimum of the effective potential. This means that the matrix of the second derivatives of the effective potential must be positive definite. Hence, to address this question we need to find the eigenvalues of the second derivative of the effective potential, with respect to our order parameters, the overlaps. The effective potential is the free energy density, namely the saddle point value of the function $(\overset{\texttt{feq}}{146})$, with the argument fixed to be a solution to (meq 145). Let us introduce the stability matrix

$$A^{\mu\nu} = \frac{\partial^2 f(m)}{\partial m^\mu \partial m^\nu}\bigg|_{\vec{m}=m_n(\underbrace{1,..,1}_{n},0,..,0)} ,$$

and evaluate it on the symmetric solutions m_n in (symmetricm 150). In the case $n=1$ we recover the retrieval state, while for $n>1$ we have spurious states

$$\begin{aligned} A^{\mu\nu} = & \quad \delta^{\mu\nu} - \tfrac{\beta}{N}\textstyle\sum_{i=1}^N \xi_i^\mu \xi_i^\nu (1-\tanh^2 \beta\vec{m}\cdot\vec{\xi_i}) \\ = & \quad \delta^{\mu\nu} - \beta\langle\langle \xi^\mu\xi^\nu(1-\tanh^2\beta\vec{m}\cdot\vec{\xi})\rangle\rangle \\ = & \quad \delta^{\mu\nu} - \beta\langle\langle \xi^\mu\xi^\nu(1-\tanh^2\beta m_n \textstyle\sum_{\rho=1}^n \xi^\rho)\rangle\rangle \\ = & \quad \delta^{\mu\nu} - \beta\langle\langle \xi^\mu\xi^\nu(1-\tanh^2\beta m_n z_n)\rangle\rangle , \end{aligned}$$

where $z_n = \sum_{\rho=1}^n \xi^\rho$. Let us consider separately the diagonal and the off-diagonal terms in the stability matrix. Since $(\xi^\mu)^2 = 1$, we find that the diagonal elements $\mu=\nu$, for $\mu = 1,\ldots,p$ are (remember we are not summing over the μ index)

$$A^{\mu\mu} = 1 - \beta(1-q) ,$$

where we introduced the quantity

$$q = \langle\langle \tanh^2 \beta m z_n\rangle\rangle . \qquad (154) \qquad \boxed{\texttt{defqq}}$$

We used the fact that $\langle\langle\xi^\mu\xi^\nu\rangle\rangle = \delta^{\mu\nu}$. The off-diagonal terms for $\mu,\nu \le n$ are all equal and given by

$$A^{\mu\nu} = \beta\langle\langle \xi^1\xi^2 \tanh^2\beta m_n z_n\rangle\rangle_z \equiv \beta Q , \qquad (155) \qquad \boxed{\texttt{defQ}}$$

where in last equality defines the quantity Q. The off-diagonal elements for $\mu,\nu > n$ all vanish, because when taking the average over patterns we will always have a single power of a pattern $\xi^{\mu>n}$ appearing alone and its average vanishes. Summarizing, the stability matrix has the form

$$A^{\mu\nu} = \left(\begin{array}{c|c} \overbrace{\begin{matrix} 1-\beta(1-q) & \beta Q & \beta Q & \cdot \\ \beta Q & 1-\beta(1-q) & \beta Q & \cdot \\ \beta Q & \beta Q & \cdot & \\ \cdot & \cdot & & \cdot \end{matrix}}^{n} & 0 \\ \hline 0 & \underbrace{\begin{matrix} 1-\beta(1-q) & 0 & \cdot \\ 0 & 1-\beta(1-q) & 0 \\ \cdot & 0 & \cdot \end{matrix}}_{p-n} \end{array}\right)$$

We can easily derive the eigenvalues of this matrix

- A non-degenerate eigenvalue, corresponding to the longitudinal fluctuation in the amplitude of m_n

$$\lambda_1 = 1 - \beta[1 - q - (n-1)Q] \ .$$

- An eigenvalue with degeneracy $p-n$, corresponding to the fluctuations in the direction that mix with more memories, namely the $p-n$ memories that were not part of the solution m_n

$$\lambda_2 = 1 - \beta(1-q) \ .$$

- An eigenvalue with degeneracy $n-1$, corresponding to fluctuations producing anisotropies among the relative weights of the n memories equally mixed in m_n

$$\lambda_3 = 1 - \beta[1 - q + Q] \ . \tag{156}$$

lambdathree

One can prove that $Q > 0$ for all $T < T_c$, hence the smaller eigenvalue is λ_3. To study the stability of the solution, we need to study the sign of λ_3. In the range of parameters where λ_3 is negative, the corresponding solution is unstable. If λ_3 is positive, the corresponding solution is stable. Since for $n = 1$ there is no eigenvalue λ_3, we conclude that the retrieval state is always a local minimum, hence it is always (meta)stable below the critical temperature. Let us find out what happens to the spurious states!

Let us estimate λ_3 both around the critical temperature $T \simeq T_c = 1$ and at very low temperature $T \simeq 0$. We need to estimate the two quantities q and Q defined in (defqq 154) and (defQ 155).

Let us start in the vicinity of the critical temperature. The overlap function (mneq 152) vanishes at the critical temperature, so we can take it to be small and expand in powers of m, using $\tanh x \simeq x - x^3/3$ and finding

$$m_n = \frac{1}{n}\beta m_n \left(\langle\langle z_n^2 \rangle\rangle - \frac{1}{3}\beta^2 m_n^2 \langle\langle z_n^4 \rangle\rangle \right) \ . \qquad (157)$$

mexpand

We need to evaluate the average over $z_n = \sum_{\mu=1}^{n} \xi^\mu$, which has a probability distribution (zdistribution 151). This is the same distribution as a random walk on a one-dimensional lattice

$$\langle\langle z_n^2 \rangle\rangle = n \ , \qquad \langle\langle z_n^4 \rangle\rangle = n(3n-2) \ ,$$

and plugging this back into (mexpand 157) and solving for m we obtain

$$m_n^2 \simeq \frac{T_c - T}{\beta^2} \frac{3}{3n-2} \ ,$$

confirming that m is small just below the critical temperature, and

$$q = \langle\langle \tanh^2 \beta m_n z_n \rangle\rangle \simeq (\beta m_n)^2 \langle\langle z_n^2 \rangle\rangle = n(\beta m_n)^2 = \frac{3n(T_c - T)}{3n-2} \ .$$

Estimating $Q = Q^{\mu\nu} = \langle\langle \xi^\mu \xi^\nu \tanh^2 \beta m_n z_n \rangle\rangle$, with $\mu \neq \nu$, can be done in the following way. We first note that, since $Q^{\mu\nu}$ only has $n(n-1)$ off-diagonal components, and all of them are equal, we have

$$Q = \frac{1}{n(n-1)} \left(\sum_{\mu,\nu=1}^{n} \langle\langle \xi^\mu \xi^\nu \tanh^2 \beta m_n z_n \rangle\rangle - n \langle\langle \tanh^2 \beta m_n z_n \rangle\rangle \right) \ ,$$

where the last term, which equals q defined in (defqq 154), removes the diagonal contribution in the sum. We recognize in the first term the appearance of the square of z_n

$$\sum_{\mu,\nu=1}^{n} \langle\langle \xi^\mu \xi^\nu \tanh^2 \beta m_n z_n \rangle\rangle] = \langle\langle z_n^2 \tanh^2 \beta m_n z_n \rangle\rangle] \simeq \langle\langle z_n^4 \rangle\rangle (\beta m_n)^2 \ ,$$

where in the last step we used the fact that m_n is small in this regime. We finally obtain

$$Q \simeq \frac{2}{n} q \ .$$

Using these estimates, we find the behavior of the eigenvalues of the stability matrix close to the critical temperature

$$\lambda_1 \simeq 2(T_c - T) > 0 \ ,$$

$$\lambda_2 \simeq \frac{2(T_c - T)}{3n-2} > 0 \ ,$$

$$\lambda_3 \simeq -\frac{4(T_c - T)}{3n-2} < 0 \ .$$

Hence, just below the critical temperature, the $\lambda_3 < 0$ and the spurious solutions m_n with $n > 1$ are all unstable! Only the retrieval state $n = 1$ is stable. This is very good news for our associative memory device.

Let us now repeat the eigenvalue estimate at low temperature. In the low temperature limit $\beta \to \infty$ we find

$$m_n = \frac{1}{n}\langle\langle z \tanh \beta m_n z_n \rangle\rangle \to \frac{1}{n}\langle\langle z \mathrm{sgn}\, m_n z_n \rangle\rangle = \frac{1}{n}\langle\langle |z_n| \rangle\rangle \ .$$

??Why last step, is $m > 0$??

In the case of odd n, the smallest value of $|z_n|$ is of course one, which gives $q \to 1$ and $Q \to 0$. The eigenvalues behave as

$$\lambda_1, \lambda_2, \lambda_3 \sim 1 \ .$$

Near $T \sim 0$, the eigenvalues are all positive and the spurious solutions are stable. Since they are stable at low temperature and unstable near the critical temperature, we conclude that there is a temperature range with upper bound given by T_c in which the only (locally) stable states are the $2p$ retrieval ones and their inverses.

Let us estimate the lower bound of this stability window. We want to estimate the temperature at which the last eigenvalue λ_3 flips sign. The odd solution m_n is stable in the range $T < T_n$, where $0 < T_n < T_c = 1$. The condition $\lambda_3 = 0$ from ($\overset{\texttt{lambdathree}}{156}$) reads $1 - \beta(1 - q + Q) = 0$, where $\beta = 1/T_n$. We find

$$T_n = 1 - \langle\langle (1 - \xi^1 \xi^2) \tanh^2 \frac{m_n(T_n) z_n}{T_n} \rangle\rangle \ ,$$

where $m_n(T_n)$ is the solution to ($\overset{\texttt{mneq}}{152}$). One can study numerically the solution of this equation and find that it is a decreasing function of n. We are interested in the critical temperature of the first spurious state to become stable as we lower the temperature: this is the $n = 3$ one

$$T_3 \simeq 0.461 \ ,$$

Our stability analysis leads us to a very important conclusion:

> In the regime of finite storage capacity p, there is a range of temperatures $0.461 < T < 1$ in which the only locally stable equilibrium solutions of the Hopfield network are the retrieval states.

This means that it is absolutely crucial that our memory neural network is *noisy*. If the noise is too large, the memory does not work at all. If the noise is too small, there are many spurious solutions that jeopardize the memory retrieval. But when the noise is inside a reasonable window, our associative memory works just fine! Hence, we have found that the Hopfield network can definitely do the job of the associative memory.

There are a few subtle points that need be mentioned

- The spurious "symmetric states" ($\overset{\texttt{mneq}}{152}$) are solutions for any $T < T_c$. But while the ones for odd n become stable below a certain temperature T_n, the ones for even n are always unstable.

- The spurious "asymmetric states" ($\overset{\texttt{asymmetric}}{153}$) solutions appear only below a certain critical temperature T_{asym}, but they are almost always unstable. Numerical simulations show that the highest temperature at which one of the asymmetric states become stable is $T_{asym} \sim 0.452 < T_3$, so as long as we are in the useful range $T_3 < T < 1$ we are still perfectly safe.

- But maybe we do not want to get rid completely of the spurious states. After all, humans sometimes come up with some creative new idea. Or sometimes, when you are trying to remember something after you were given some hint, you end up remembering something else. At the end of day, memories are not an exact replica of the experience that led to store the pattern, but they are wild revisitations of the original event. So maybe having some spurious states is not so bad after all...

- The number of spurious states grows exponentially with the number of stored patterns. When we retrieve a spurious memory in the Hopfield network, however, there is no tag attached to it that warns us we are making a mistake. This funny feature might be a mathematical model that embodies the way of saying "The more we learn, the more we deceive ourselves..."

7.6. Phase Diagram

Let us summarize the full phase diagram of the Hopfield network, in the plane $\alpha - T$, depending on both the *load parameter* α and the temperature T. We have several salient features, that we can extract from Fig?

- *Zero temperature*: The limit of low temperature $\beta \to \infty$, that we studied in Section lowT 7.4.3, corresponds to the horizontal axis in Fig?. For large load $\alpha > \alpha_c$, there is no retrieval solution at all: the network does not have any memory. Our analytical estimate of the capacity (alphac 27) gives $\alpha_c \simeq 0.138$.[11] As we decrease α below the critical value, two solutions with $m \neq 0$ appear, the larger one is the stable solution, corresponding to a retrieval state, which deviates only slightly from the stored pattern. Close to $\alpha \sim \alpha_c$, $m = 0.967$ implying a 1.5% error in the retrieval of the stored pattern. The network is thus able to slightly lower its free energy by relaxing a small fraction of spins towards the direction of the other patterns (with a non-zero total overlap r with them, see (totaloverlap 125). As we decrease α, the accuracy becomes perfect very quickly.

- *Spin glass phase*: It is easy to see that the low temperature equations (lowteq 136) admit a locally stable solution with

  $$m = 0 \;, \qquad r = (1 + \sqrt{2/\pi\alpha})^2 \;.$$

  It can be shown that this solution, called the *spin glass state*, is the limit of the spurious symmetric state solutions (symmetricm 150), which mix n patterns, when $n, p \to \infty$. This is an ordered state, in which the magnetization is zero but its square is nonzero. At low temperatures, the spin glass state is the ground state in the range $0.051 < \alpha < 0.138$. As we approach the vertical axis $\alpha \to 0$, the spin glass state freezes into the symmetric state

[11] This is the same as the one we would obtain by using the so-called *replica trick*, using a replica symmetric ansatz. It turns out that this estimate is slightly off the numerical simulation using Monte Carlo methods, which gives the value $\alpha_c \simeq 0.145$ and is in agreement with the exact replica symmetry breaking solution.

solutions. We studied their stability in Section sec:stability 7.5.3 and concluded that, for $0.461 < T < 1$, the only stable states are the retrieval ones, and all the spurious states are unstable.

- For general α and T, there is a line $T_{ret}(\alpha)$ below which the retrieval states appear and are locally stable, but the global minima are the spin glass states. There is another line $T_c(\alpha) < T_{ret}(\alpha)$, below which retrieval states become the global minima.

Guide to the literature

First, the original papers. The Hopfield network was introduced by John J. Hopfield in Hopfield:1982pe [17]. Its exact solution in the case of a finite number of stored patterns appeared in amit85 [16]. The case in which the number of stored patterns scales like N was solved in amit85b [18], by using the replica method in the theory of spin glasses. The authors computed the capacity of such model and found the famous result $\alpha \sim 0.138$. A useful review article by the same authors is Amit:1985ff [19]). The original papers are very pedagogical and I recommend their reading. Some funny comments are quoted from pargavirasoro [20].

There are several books on the solution on the Hopfield-like models for associative memory. The reprints of the original papers can be found in the great book parisi [21]. This book contains many other applications of the theory of spin glasses, of which the Hopfield network is one of the most celebrated examples. The heuristic derivation of the solution of the Hopfield network can be found in geszti,hertz [22, 7]. In particular, hertz [7] is a great pedagogical introduction to the theory of neural networks, and includes also a formal derivation of the exact solution of the Hopfield model, using the replica method. The neural network book by Amit amitmodeling [23] contains a thorough analysis of every little aspect of the Hopfield network and various generalization, it is like a expanded version of his papers. A review of the theory of spin glasses from a more modern perspective, with applications to some current string theory problems, is Denef:2011ee [24].

8. Boltzmann Machines

We encountered two kind of machines so far. The first associative memory system was the Hopfield network. In this system, all spin-like units in the system are visible and the weights are symmetric and the architecture is that of a recurrent network. The inputs were fixed values ξ_i, while the update rule was stochastic (the presence of noise greatly enhanced the reliability of the memory retrieval task). On the other hand, we have studied the multi-layer perceptron architecture. Here, only some of the units are accessible: the inputs and outputs, but not the hidden units. The weights are not symmetric, but feedforward. The inputs and outputs are fixed patterns, while the update rule is deterministic.

In this Section, we will introduce a new system that has a new property: it learns probability distributions, rather than fixed input-output patterns! In real life, it is sometimes important to generate different outputs given the same sensory inputs. A probability distribution as an output allows for this possibility. Suppose you are in the forest and a lion is running after you. You work your way through the bushes running as fast as you can and at some point you find the lion's face with his mouth wide open and all his teeth in plain sight, ten feet in front of you. The same thing happens every week, and you have to decide whether to jump left or right to avoid easing the lion's task. If every time you see the lion's face in front of you (same external stimulus ξ_i), you decide to jump to the right, clearly the lion is going to learn your pattern and eat you soon. But if your response to the external stimulus is probabilistic, you have way higher chances of survival!

Let us pause and go back to the Hopfield network for a second. At zero temperature, the Hopfield network update rule (108) minimizes the energy function $H(S_i) = -\frac{1}{2}\sum_{ij} w_{ij} S_i S_j$ in (139), given a deterministic result. At non-zero temperature, we replaced the deterministic rule (108) with the probabilistic one (118), called *Glauber dynamics.* The network, in this case, is approximating the probability distribution given by the energy function: $e^{-\beta H(S)}/Z$. What if we create a system that, instead of approximating such probability distribution, actively implements it?

Consider a Hopfield network with the stochastic activity rule

$$f(h_i) = \begin{cases} 1 & \text{with probability } \frac{1}{1+e^{-h_i}} \\ -1 & \text{otherwise} \ , \end{cases}$$

with the usual activations $h_i = \sum_j w_{ij} S_j$. This rule implements the Gibbs sampling of the probability distribution $e^{-\beta H(S)}/Z$.[12]

Our learning task is the following. Instead of having the system store a fixed set of patterns $D = \{\xi_i^\mu\}_{\mu=1}^p$, we would like it to learn a *probability distribution* of patterns $P(D)$. By using Bayes' theorem, we want to find the weights that approximate such distribution:

$$P(w|D) = \frac{P(D|w)P(w)}{P(D)} \ . \tag{158}$$

inferenceboltzmann

Using a gradient descent algorithm, we have to minimize the likelihood $P(D|w) = \prod_{\mu=1}^p P(\xi_i^\mu|w)$, with

$$P(\xi_i^\mu|w) = \frac{e^{-H(\xi,w)}}{Z(w)} \ ,$$

and $Z(w) = \sum_{S_i} e^{-H(S_i)}$. It is more convenient to extremize the log likelihood instead, so that we can turn the product into a sum and take one pattern at a time. Let us consider only one training pattern ξ_i^μ for simplicity, namely let us look at the stochastic gradient descent algorithm. Setting the temperature $\beta = 1$ for simplicity, we find

$$\begin{aligned} \frac{\partial}{\partial w_{ij}} \ln P(\xi_i^\mu|w) &= \tfrac{\partial}{\partial w_{ij}} \left(-H(\xi_i^\mu; w) - \ln Z(w)\right) \ , \\ &= \underbrace{\xi_i^\mu \xi_j^\mu}_{data} - \underbrace{\langle S_i S_j \rangle}_{model} \ . \end{aligned} \tag{159}$$

This last equation encodes the first basic feature of the Boltzmann machine. Its learning algorithm contains two terms:

[12] Given a probability distribution $P(S_1, \ldots, S_N)$ it is usually very hard to sample all S_i's in one shot. An easier way to proceed is to sample the units one at a time. Fix $S_2, \ldots, S_N$ and sample S_1. Then fix $S_1, S_3, \ldots, S_N$ and sample S_2 and so on. We are computing $P(S_1|S_2, \ldots, S_N)$, $P(S_2|S_1, S_3, \ldots, S_N)$ and so on, one at a time.

1. The first term $\xi_i \xi_j$ is the usual Hebb rule, applied to the empirical distribution of data that the machine "sees", coming in from the outside world. It is called the **wake** term, because that is what the machine sees when it is connected to the world. It detects the correlations among the external stimuli and increases the weights accordingly, much like in the unsupervised learning scheme.
2. The second term is called the **dream** term, since it is the correlation between the same units, but when it is generated by the internal dynamics of the machine. Namely, when the machine is sleeping it dreams about producing some correlations. The sign on the second term is negative: this is an anti-Hebbian rule, namely the internal correlations *decrease* the weights.

If we start from a zero set of weights, the first "awake" step of the learning procedure in (boltzmannlearning 159) produces $\Delta w_{ij} = \eta \xi_i \xi_j$, which is the Hebb rule for the Hopfield network. However, from the second step the learning algorithm does not give the Hopfield weights. Since the learning rule is symmetric, it will produce symmetric weights even when the initial distribution of weights is not symmetric. Finally, note that by including in the inference task (inferenceboltzmann 158) the prior distribution over the weights $P(w)$, as we did in (newerror 47), we can generate a weight decay term (learningrule 49) that regularizes the learning algorithm.

Note that maximizing the log likelihood $\ln P(\xi_i^\mu | w)$ is equivalent to minimizing the Kullback-Leibler divergence (KL 16)

$$G(w) = \sum_{\xi_i^\mu} P(\xi_i^\mu) \ln \frac{P(\xi_i^\mu)}{P(\xi_i^\mu | w)} \ ,$$

that measures the difference between the probability distribution of the training data $P(\xi_i^\mu)$ and its likelihood $P(\xi_i^\mu | w)$ as generated by the Boltzmann machine.

8.0.1. Higher order correlations

We have just built a machine that learns and generates probability distributions. However, it is still only sensitive to quadratic correlations. But in the real world, quadratic correlations are not very useful, consider for example image recognition. They just tell us what happens to a pixel in our retina, given the value of another pixel. They are not really enough to help us realize that a person is different from a plastic doll. These advanced tasks require a machine that can detect (very) high order correlations. How can we improve our machine in this direction?

The first guess is to add higher-order couplings to our hamiltonian, something like

$$H(S) = -\frac{1}{2}\sum_{ij} w_{ij} S^i S^j - \frac{1}{4!}\sum_{ijkl} t_{ijkl} S^i S^j S^k S^l + \dots$$

But this sounds unlikely, since the number of couplings needed to capture higher order statistics increases factorially and it is definitely not a plausible model for our brain.

We can draw a useful hint from physics, namely from the Fermi theory of weak interactions. At the time, people were interested in studying beta decay, namely a neutron decaying into an electron, a proton, and an anti-neutrino. The operator that generates this quartic interaction contains four fermi fields

$$\mathcal{L} = G_F(\bar{\psi}\sigma^\mu\psi)(\bar{\psi}\sigma^\mu\psi) \ . \tag{160}$$

fermi

However, this operator is ill defined once we take into account quantum corrections (it is non-renormalizable). The way to fix this problem is to introduce a new field W_μ, which is a vector boson that couples to the fermion bilinear and has a mass term

$$\mathcal{L}' = -g(\bar{\psi}\sigma^\mu\psi)W_\mu + \frac{1}{2}m_W W_\mu W^\mu \ , \tag{161}$$

vectorboson

where the mass m_W of the vector boson is very large, compared to the energy at which we are probing the system. By integrating out the vector boson W_μ in $(\overset{\texttt{vectorboson}}{161})$ we recover the original model $(\overset{\texttt{fermi}}{160})$ and the relation between the higher order coupling G_F and the microscopic couplings g, m_W: $G_F = g^2/2m_W$. But the new model $(\overset{\texttt{vectorboson}}{161})$ has much nicer properties, in particular, quantum corrections are more tractable (in fact, this insight together with the discovery of the

Higgs mechanism led to the construction of the Standard Model of electroweak interactions). At energies much lower than the W-boson mass, we cannot produce such bosons as external particle in our experiment, and hence they are virtual particle, invisible to an external observer.

The hint we draw from this story is that if we want to have a model that encode higher order correlations, we need to introduce hidden units, which are coupled to the external ones, but we cannot probe directly. The higher order correlations among the external units are detected by these hidden units. In fact, this is pretty much what happens in the case of multi-layer networks with hidden units we saw above. Such networks are universal approximators, thanks to the role played by their hidden units.

We improve our Boltzmann machine now by introducing two sets of units: visible ones S_i and hidden ones h_i. We will collectively denote them by $y_i = (S_i, h_i)$. When we *clamp* the visible units to their training patterns we will use the notation $y_i^\mu = (\xi_i^\mu, h_i)$, still leaving the hidden units free to fluctuate. The hamiltonian couples all units, visible and hidden

$$H(y) = -\frac{1}{2}\sum_{ij} w_{ij} y_i y_j \ ,$$

but the likelihood of producing the training set ξ_i^μ is obtained by *tracing* over the hidden units

$$\begin{array}{rcl} P(D|w) = & & \sum_h P(\xi_i, h_i|w) \\ = & & \frac{1}{Z(w)} \mathrm{Tr}_{\,h} \exp\left(\frac{1}{2}\sum_{ij} w_{ij} y_i^\mu y_j^\mu\right) \ , \end{array}$$

where the partition function is $Z(w) = \mathrm{Tr}_{\,S,h} e^{-H(y)}$. The gradient descent procedure applied to the log likelihood of (hiddenboltz 162) is similar to the one with no hidden units (boltzmannlearning 159), except that now the "wake" term includes a trace over the hidden units

$$\frac{\partial}{\partial w_{ij}} \ln P(\xi_i^\mu|w) = \quad \frac{1}{Z_h}\sum_h y_i^\mu y_j^\mu e^{-H(y^\mu)} - \langle y_i y_j \rangle,$$

where $Z_h = \mathrm{Tr}_{\,h} e^{-H(y^\mu)}$ is the hidden units partition function, for fixed visible units. The two terms have the same interpretation as the ones we gave above

1. The first term is a Hebbian contribution that describes "wake" state of the network. It measure the correlations among the units when the external states are *clamped* to the training set values ξ_i^μ, but now the internal hidden units are free to fluctuate. So this sample is obtained by fixing the visible units and averaging over the hidden ones.
2. The second term is the anti-Hebbian "dream" term, in which both the free and the hidden units are free to fluctuate according to the internal dynamics.

Again, we can add the weight decay term by including the prior distribution over the weights $P(w)$.

Guide to the literature

Unsupervised learning is explained in many excellent books. A pedagogical introduction to both Hebbian learning, PCA and Oja's rule can be found in Hertz' book hertz [7]. A discussion more focussed on the neurobiological aspects of synaptic plasticity, with the neat example of ocular dominance, can be found in abbottdayan [25]. The Boltzmann machine is explained very well in the books by Hertz hertz [7] and MacKay mackay [3].

9. Reinforcement learning

Inside our brain, there is no Maxwell devil telling each neuron if its answer to the stimuli was correct or not, as was the case for the supervised learning algorithms in Chapter supervisedlearning 4. Nonetheless, there must be some sort of feedback signal that tells us about the quality of our actions. If we go see the last Mel Gibson's movie, for which we had high expectations, and we realize it really sucks, we are not going to see his next movie. If we vote republican expecting that job creators will thrive and resurrect the economy, but they start two wars instead and engineer the worst financial crisis since the Great Depression, we are not going to vote for them again later. If we wander around the neighborhood in search of a grocery store and we find it, we are probably going to learn how to get there after a couple of times.

In all these cases, we perform some actions, based on some internal estimate of future rewards, and get a feedback signal from the environment, that in turn shifts our previous estimate to a more realistic one, that we use to change our future actions. The feedback signal is an overall indicator of the effectiveness of our choise: a global yes or no feedback. It is called a *reinforcement* signal. Our goal is to maximize the total return, which is a sum of all the rewards we will get in the future.

9.1. The maze task

Let us work out the theory of reinforcement learning step by step in a simple example that contains all the ingredients. We have a maze, with one starting point and four endpoints, see Fig.???. We put a thirsty Mickey Mouse at the entrance of the maze, at location A, and let him figure out how to get to the drinks. A certain amount of juice is delivered at each endpoints (reward): 0, 2 or 5 ml. Our hero can only move forward at each T-cross, making a decision whether to turn left (L) or right (R). Different paths lead to different rewards. The goal for thirsty Mickey Mouse is to maximize the total amount of juice in a trial (long term return).

We call locations A, B, C our *states* s; at each state, we have two options for which action a to take (turning either L or R) and we have to make a decision. If the hero turns left at B, he gets no juice. But if he turns right, he gets 5 ml. We denote by $r^a_{ss'}$ the reward obtained from state s, taking action a, and ending up in state s' . Summarizing:

$$\begin{aligned} \text{States:} \qquad & s = A, B, C \ . & (162) \\ \text{Actions:} \qquad & a = L, R \ . & (163) \\ \text{Rewards:} \qquad & r^R_{AB} = 5 \ , & (164) \\ & r^L_{AC} = 2 \ , & \\ & r^a_{ss'} = 0 \text{ otherwise} \ . & \end{aligned}$$

In our simplified maze task, the actions a available at each state are the same, either left or right, but we could have more general tasks in which at every state we can select among a variety of different actions. Another simplification is that while the choice of which action to take may be probabilistic, after choosing a particular action, the next state is known with certainty. Namely, if we decide to turn L at A, we end up in B with probability one. In more general cases, a choice of an action might lead to a probability distribution of ending up in different states, but it is not the case in our maze task.

At each point in the maze (state), Mickey Mouse must select an action, following a certain policy:

A **policy** π is a collection of all the possible (state, action) pairs, that specifies completely the behavior of the **agent**.

It can be either stochastic (we give the probability of choosing a certain action in a certain state) or deterministic (only one action is accessible for each state). We will consider here *stochastic policies*, namely we will give the *probability*

$$\text{Prob(choosing action a in state s)} = \pi(s, a)$$

that in a certain state the rat will select a particular action

$$\text{Policy:} \quad \{\pi(s, a)\} \quad \text{for } s = A, B, C \text{ and } a = L, R \ .$$

We can parameterize a policy in a simple way, by assigning to each (state,action) pair a number called the action-value function

> An **action-value function** $Q^{\pi}(s,a)$, under policy π, is the total future return expected by starting in state s, taking action a and following policy π from there.

The probability of selecting action a in the state s is then given by the simple *softmax* probability distribution

$$\pi(s,a) = \frac{e^{\beta Q(s,a)}}{\sum_{b=L,R} e^{\beta Q(s,b)}} \ . \tag{165}$$

`softmax`

We introduced here the seeds of the notorious *exploration/exploitation dilemma*. The goal of the agent is to maximize the long term return, the collection of all the future rewards in the trial. To do this, he might stick to a course of action he already knows, that will deliver the highest return (exploitation). This is called a *greedy* policy, because it tends to maximize the known return. But sometimes it is a good idea to explore new options, to discover potential policies that deliver higher rewards (exploration)! The parameter β in ($\overset{\texttt{softmax}}{165}$) controls the balance between these two different possibilities. If β is very large, the probability distribution will localize around a single choice per state, namely the one with higher action-value function $Q(s,a)$, so that we will always take action a in state s. This will give a deterministic greedy policy, all exploitation and no exploration. On the other hand, if β is very small, all actions are equiprobable, regardless of their expected returns, so we are in exploration mode. One can have different kind of policies from ($\overset{\texttt{softmax}}{165}$). Another popular choice is the ϵ-greedy policy, where the rat chooses almost always the action that gives the largest immediate reward (i.e. greedy) with probability $1-\epsilon$, except sometimes when with probability ϵ he chooses a random action. Intuitively, it is clear that an explorative policy is advantageous in the first few trials, when we still have no idea where the rewards are. As long as we start to learn about the environment, we can tune down the exploration a bit and concentrate on the most favorable actions instead, still leaving some room for exploration once in a while.

We will divide the reinforcement learning problem into two different steps and our agent (Mickey Mouse the rat) into two different personalities: the *critic* and the *actor*. The actor uses the existing policy to perform the task and, on the way, collects eventual rewards. The critic uses the rewards to perform estimates of the value functions and compares them to the expectations from the previous trials, to evaluate whether it performed better or worse than expected, and improves the policy accordingly. The process of having rounds of actor-critic-actor-critic-..., called *policy iteration*, allows the agent to improve its estimates better and better and use in turn these great estimates to change its behavior incrementally, approaching an *optimal policy*, namely a policy that gives the largest possible return.

9.1.1. Value functions

Let us assume that a particular policy $\{\pi(s,a)\}$ has been chosen. We will evaluate its performance, to decide how it compares to our expectations. But first, we need to describe what we mean exactly by having "expectations."

The goal of reinforcement learning is to obtain the largest amount of return. We we define the future return at time t as the sum of all the future rewards we will collect from time t until the end of the trial

$$R_t = r_{t+1} + r_{t+2} + \ldots = \sum_{k=0,1,\ldots} r_{t+1+k} \ .$$

We can have either an infinite number of steps or just a finite of them, so we take care of the notation appropriately in the sum over future rewards, that we left open-ended in the previous expression. Our conventions are that by taking the action a_t in state s_t at time t, we move to state s_{t+1} and collect the reward r_{t+1}.

Usually, rewards that are far off in the future are less appealing than immediate rewards. If I ask you to decide whether to get ten dollars now or twenty dollars one year from now, you will most likely choose the immediate cash, even if the sum you would earn later on is larger. Therefore, we introduce a *discount factor* $0 \leq \gamma \leq 1$, that counts how far in the future the expected rewards are going to be delivered. The return is defined as the discounted sum of all future rewards

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0,1,\ldots} \gamma^k r_{t+1+k} \ , \tag{166}$$

`return`

where in our notation, r_{t+1} is the reward we collect at time step $t+1$ by taking a particular action at time t. If $\gamma = 0$, we are only interested in the immediate reward. If $\gamma = 1$, all future rewards count the same. Now, the crucial point is that the return ($\overset{\texttt{return}}{166}$) satisfies a very simple recursive property

$$\begin{aligned} R_t &= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \ldots) \\ &= r_{t+1} + \gamma R_{t+2} \ . \end{aligned} \tag{167}$$

This recursion relation is at the very heart of the temporal difference method of assigning credit, as we will shortly realize.

At each point in the maze, $s = A, B, C$, we associate a particular *state-value function* $V^\pi(s)$. This describe the total amount of future rewards r_i that we *expect to collect*, starting from that state s and following a fixed policy

A **state-value function** $V^\pi(s)$ under policy π is the expected future return obtained by starting in state s and following policy π.

Let us work out the state-value functions for the maze task in Fig.???. Consider an exploratory policy, in which we assign equal probability for turning L or R at each node of the maze. This is quite realistic: the first time the rat is put in the maze, he will just take a random path and see what happens. In this case, the action-value functions $Q(s,a)$ in ($\overset{\texttt{softmax}}{165}$) all vanish, and we find that the probabilities are all the same

$$\pi(s,a) = 1/2 \qquad \forall\, s, a \tag{168}$$

`policyhalf`

We want to evaluate the state-value functions for this policy, by weighing the rewards according to the probability of selecting that particular action. This is going to give us the true value of the state-value functions for our given policy π in ($\overset{\texttt{policyhalf}}{168}$). States B or C are the terminal states: we take an action and end the task, getting or not a certain amount of immediate reward. The values of states B and C are

$$V^{\pi}(B) = \frac{1}{2} \times 0 + \frac{1}{2} \times 5 = 2.5 \ , \qquad V^{\pi}(C) = \frac{1}{2} \times 0 + \frac{1}{2} \times 2 = 1 \ . \tag{169}$$

mazebc

State B is more valuable than state C, under the current policy in which every action L or R is chosen with the same probability. We can easily derive a formal expression for the state-value functions in the terminal states $s = B, C$

$$V^{\pi}(s) = \sum_{a=L,R} \pi(s,a) \sum_{s'} P^{a}_{ss'} r^{a}_{ss'} \ . \tag{170}$$

statevalueimmediate

This is nothing but the expectation value of the immediate reward, which is also equal to the total future return since this is the last state of the maze task

$$V^{\pi}(s) = \langle r_{t+1} \rangle = \langle R_t \rangle \ , \tag{171}$$

statereturn

where the average is taken according to the probability distribution $\{\pi(s,a)\}$ that specifies our current policy. This is in agreement with our definition of the state-value function as an average measure of the future return. In our simple case, whereas the choice of L or R action is probabilistic, once we have chosen an action, the next state is known with probability one. Namely, if we decide to turn L at B, we end up at the end of the task and collect reward zero, with probability one, and so on. But in general, the result of taking a certain action might be stochastic too, for instance if by taking action L in state B we had a certain probability of getting to state s' with a certain reward and to a different state s" with a different reward. We included in (statevalueimmediate 170) the probability $P^{a}_{ss'}$ of ending up in state s' by taking action a in state s, even if in the simple maze task this expression is always diagonal $P^{a}_{ss'} = \delta^{a}_{s'}$ so the second sum over s' in (statevalueimmediate 170) just drops and we get back to (mazebc 169).

A convenient way to illustrate a task is by use of the *backup diagrams.* The diagram in Fig.?? encodes the rat's maze task we are describing. The empty circles represent states, and the full circles state-action pairs. Time always flows from top to bottom. In the maze task, once we choose an action in a particular state, the next state is known with certainty. Hence, full circles only have single lines emanating from them. This corresponds to the choice $P^a_{ss'} = \delta^a_{s'}$ in (statevalueimmediate 170). In general, one can have stochastic actions, such that once an action is chosen in a particular state, the next state is chosen among different options, weighted by a probability distribution. This more general case, which requires the full expression in (statevalueimmediate 170), is exemplified in Fig.??.

Let us go back to our maze task. We evaluated the state value of states followed by immediate reward. But what if the reward comes only after several steps? How can we evaluate the total worth of state A? Actions taken from there do not lead to immediate rewards (more precisely, they lead to states with zero immediate reward). On the other hand, from the next states we can choose an action and get our final rewards. Should we give credit only to the terminal states B and C, or also to the previous state A, which might eventually lead to a reward too? This is called the *temporal credit assignment problem.* Value functions allow us to easily solve this problem.

We will just back-propagate the value functions weighted by the probability of ending up in each state, according to the current equiprobable policy:

$$V(A) = \frac{1}{2}\gamma V(B) + \frac{1}{2}\gamma V(C) = \frac{1}{2} \times 0.8 \times 2.5 + \frac{1}{2} \times 0.8 \times 1 = 1.4. \qquad (172) \qquad \boxed{\texttt{valueofa}}$$

We introduced here a discount factor of $\gamma = 0.8$, that we already encountered in the definition of return in (return 166). This accounts for the fact that future rewards are less appealing than immediate rewards. The total worth of state A is somewhat in between the worths of states B and C in (mazebc 169). Let us derive this intuitive result from the definition of state-value function.

By exploiting the recursive formula for the future return (recursive 167) we obtain

$$V^\pi(s_t) = \langle R_t \rangle = \langle r_{t+1} + \gamma R_{t+1} \rangle = \langle r_{t+1} \rangle + \gamma \langle R_{t+1} \rangle \ . \qquad (173) \qquad \boxed{\texttt{recuru}}$$

In this last equation, the first term $\langle r_{t+1} \rangle$ is the average immediate reward we obtain by ending up in state s' at time $t+1$

$$\langle r_{t+1} \rangle = \sum_{a=L,R} \pi(s,a) \sum_{s'} P^a_{ss'} r^a_{ss'} \ ,$$

with $P^a_{ss'} = \delta^a_{s'}$ in our simple maze task, but we keep this factor for general tasks. The immediate reward is zero in the case of state A since there is no immediate reward associated to ending up in B or C from A. To evaluate the second term in (173), it is best to look at the backup diagram in Fig.??. Starting in $s_t = A$, namely from state A at time t, we have two possible states $s_{t+1} = B, C$ at time $t+1$, depending on whether we choose action $a_t = L, R$. The future return R_{t+1} from time $t+1$ must average over the two possibilities, using the estimate of the future returns given by the corresponding state-value functions

$$\langle R_{t+1} \rangle = \sum_{a=L,R} \pi(s,a) \sum_{s'=B,C} P^a_{ss'} V^\pi(s')$$

Putting everything together we find

$$V^\pi(s) = \sum_{a=L,R} \pi(s,a) \sum_{s'=B,C} P^a_{ss'} \left(r^a_{ss'} + \gamma V^\pi(s') \right) \ , \qquad (174)$$

bellman

where, again, in our simple maze task we set $P^a_{ss'} = \delta^a_{s'}$, but in general we allow the result of an action to be stochastic. Eq. (174) is called the *Bellman equation*. This equation gives a recursion relation that the state-value functions must satisfy and it applies to the generic backup diagram in Fig.??. Note that we have one Bellman equation for each state. Hence, the Bellman equations completely determine the state-value functions. We immediately see that the expression (172) we obtained intuitively is nothing that the Bellman equation (174) for state A.

9.2. Actor/critic model

As we anticipated above, we will now describe a realistic reinforcement learning algorithm. Thanks to this algorithm, the agent is going to learn the state-value functions for the maze task and use them to improve its policy until it reaches an optimal policy. The algorithm requires the agent to have two personalities. The first one, called the *actor*, is a very handsome Hollywood star. No, just kidding. The actor interacts with the environment, it picks an action a in each state s according to the policy $\pi(s,a)$. The second personality, called the critic, is very mean and always criticizes what the actor does. It compares the reward at each step with previous estimates of the return encoded in the current state-value functions, and updates them online, at every step of the task. Then, the critic takes the new updated estimates of the value functions, and uses them to improve the policy. At this point, the actor comes into the game again and proceeds with the task in the next state, according to the current policy. Since the job of the actor is obvious, we will mainly focus on the critic's challenge.

9.2.1. Policy evaluation

We just computed the true value of the state-value functions (mazebc 169) and (valueofa 172) for each state in the maze task under the equiprobable policy (policyhalf 168). However, the rat does not know anything about the task, in particular, he has no way of knowing the state-value functions. He has got no idea of the rewards ahead of him, that he would need to compute the exact state value functions $V^{\pi}(s)$ we just derived. He needs to learn the state-value functions by successive approximations. Suppose the rat starts from a random reference value of the state-value functions $V(s)$ and then performs the first trial in the maze (actor). At the end of the first task, the rat is going to observe some reward (including the no-reward case) and, based on the observed values, he will be able to build a better estimate of the state-value functions (critic). This process is called *policy evaluation*.

Suppose we start the first task with some random initialized values of the state value functions $V(s)$, for every state s. This is reasonable since we do not know anything about the task, we even ignore what the states themselves are, so when we encounter them for the first time we initialize their value in some way, random is just as good. Suppose moreover that we start with our initial equiprobable policy (policyhalf 168), just because we do not know any better. Now, let us perform the task for the first time following the policy, and collect a certain amount of reward in each state. In our interpretation (statereturn 171), the state-value functions are an estimate of the total future return. If we already knew the *exact* state-value functions $V^{\pi}(s)$ related to the current policy, we would find that they satisfy

$$V^{\pi}(s_t) = r_{t+1} + \gamma V^{\pi}(s_{t+1}) \ ,$$

for the current choice of actions, as shown in (recuru 173). But obviously we do not know the exact state-value functions, only their approximation!

The best we can do is the following. Suppose we stored the current state-value functions $V(s)$ and $V(s')$, for two consecutive states s and s', connected by the choice of the action a in state s, as shown in the backup diagram Fig.??. Now, during the next trial, we take the same action $a_t = a$ in state $s_t = s$ and end in state $s_{t+1} = s'$, getting an immediate reward $r_{s'}$. We can get a better estimate of the state-value function $V(s)$ by computing the **temporal difference error**

$$\delta = r_{s'} + \gamma V(s') - V(s) \ , \tag{175}$$

Again, this error would be zero if we knew the exact value of the state-value functions $V^{\pi}(s)$ associated to our current policy π, but unfortunately we do not know it and must learn it from experience. Hence, the error is non-zero. It derives its name from the fact that the estimates of the total future return $R(s) = V(s_t = s)$ at time t and $R(s) = r_{t+1} + \gamma V(s_{t+1} = s')$ at time step $t+1$ come from two successive estimates, the first one performed in state s and the second one performed one step later in state s'. Again, what we need to compute the error is the estimates of the state-value functions $V(s)$ and $V(s')$ evaluated by the previous trial, and the new immediate reward $r_{s'}$ collected in the present trial.

The temporal difference (TD) error gives us an evaluation of the accuracy of the current estimates $V(s)$. If it is positive, it means that the reward is higher than we previously thought. If it is negative, it means that the reward is worse than we expected. We can therefore use this TD-error to *learn* a better policy. After moving from state s to state s' and collecting the reward $r_{s'}$, we can update our current state-value function for state s according to

$$V(s) \longleftarrow V(s) + \alpha \left(r_{s'} + \gamma V(s') - V(s) \right) \ . \tag{176}$$

TDupdate

If the trial went better than expected with $\delta > 0$, we want to increase the value of the state s, if it went worse than expected $\delta < 0$, we want to decrease the value of state s.

The TD-error policy evaluation has a lot of nice features:

- It learns directly from the experience, without assuming any model for the environment. It builds a model on the go.

- It is an example of *online* policy evaluation. It updates the current estimates of the state-value functions as one proceeds state by state, incrementally. It is also called a "bootstrap" method, since it updates estimates based on previous estimates, without waiting for the final outcome of the trial, after collecting all of the rewards.

- Due to its online nature, it is efficient also in the case of a "continuous" stream of events, namely states which are not clearly separated in different number of episodes.

- One can prove that the update rule (TDupdate 176) converges to the exact state-value functions $V^{\pi}(s)$ for the current policy.

Let us summarize an algorithmic procedure to policy evaluation, in a procedural way convenient for software implementation:

- Initialize $V(s)$ arbitrarily for all states s. Choose a policy $\{\pi(s,a)\}$ to evaluate.

- Repeat for each time step t:

  1. Pick a state s.
  2. Pick an action a in state s, according to the policy $\pi(s,a)$.
  3. Act with a and move from state s to state s', collect reward r, compute the TD-error $\delta = r + \gamma V(s') - V(s)$.
  4. Update $V(s)$ according to

$$V(s) \longleftarrow V(s) + \alpha\delta \ .$$

  5. Move to next state s'.

9.2.2. Policy improvement

The second job of the critic is to use better estimates of the state-value functions to improve the current policy $\{\pi(s,a)\}$. If the TD-error δ is positive, more reward than expected lies ahead, so we should increase the preference for the action that gave more reward, and decrease the preference for the other actions=. Viceversa, if the TD-error is negative, we should decrement the corresponding probability. Recall that the policy preferences are given by the softmax distribution ($\overset{\texttt{softmax}}{165}$), that we repeat here

$$\pi(s,a) = \frac{e^{\beta Q(s,a)}}{\sum_{b=L,R} e^{\beta Q(s,b)}} \ ,$$

whose parameters are the action-value functions $Q(s,a)$. The critic should then perform an update of the action-value functions, according to the intuition outlined above. The simplest way to update the policy is to replace

$$Q(s,a) \longleftarrow Q(s,a) + \rho\,[1 - \pi(s,a)]\,\delta \ , \qquad (177) \quad \boxed{\texttt{control}}$$

for the action-value corresponding to the action a that was taken and led to the TD-error δ, while at the same time updating

$$Q(s,a') \longleftarrow Q(s,a') - \rho\pi(s,a')\delta \ , \qquad (178) \quad \boxed{\texttt{control}}$$

for all the action-values a' that were not taken in state s. The parameter ρ controls the amount of change in the policy control algorithm. Note the last term in ($\overset{\texttt{control}}{178}$), proportional to the probability of not choosing a or choosing a'.

9.3. A neural network implementation

Evidence for the relevance of reinforcement learning for our brain comes from the study basal ganglia, among other examples. In this section, we describe a schematic neural circuitry in Fig.??, inspired by the basal ganglia, that realizes the neuron as computing the state-value function.

Our hero is a linear model of the neuron, a perceptron, in which the output V, the input patterns ξ_i^μ, where μ labels the different patterns, and the synaptic weights $w_i(t)$ are related via

$$V[\xi^\mu(t)] = \sum_i w_i(t)\xi_i^\mu(t) \ . \tag{179}$$

linearnode

We interpret the input pattern $s_t = \{\xi_i^\mu(t)\}$ at time t, for a particular pattern μ, as a particular state. The output $V[\xi^\mu(t)]$ is interpreted as the state-value function corresponding to such state. We will use this system to predict future rewards, given a fixed deterministic policy.

Suppose the agent follows a deterministic policy and at time t it takes an action a in response to the particular state $\{\xi_i^\mu(t)\}$. The estimate of the state-value function of state $\xi_i^\mu(t)$ is the output $V[\xi^\mu(t)]$ in (linearnode 179). There must be a sort of memory system in the model neuron, that keeps track of the state and of the output, in order to compute the TD-error. The memory of the last state, which is referred to as **eligibility trace**, can be implemented as a form of synaptic memory. We represented this as a little filled circle on the synapse in Fig.???. We need to store the value of the output $V(t)$ for one time step, in order to compute the TD-error. This is implemented by an axonal delay of one time step from the output neuron to the evaluation neuron. We represented this delay by a thick arrow connecting the output neuron and the evaluation neuron.

Now let us move to time step $t+1$. Here, a reward $r(t+1)$ is administered by the environment, as a result of taking action a at time t. In the basal ganglia, this is believed to be given by a temporary shot of the dopamine neurotransmitter, which makes the neurons feel good. Always at time $t+1$, a new state $\xi_i^\mu(t+1)$ hits the neuron, which estimates again the state-value function as its output $V(t+1)$. Now, the output signal $V(t+1)$ goes through two different pipelines. One is the slow axonal connection, that we use for the next step evaluation at time $t+2$ and we neglect it. There is a second "fast" connection, that weighs in the output $V(t+1)$ the discount factor γ and brings $\gamma V(t+1)$ to the evaluation unit, always at time $t+1$.

At this point, everything is ready for the evaluation neuron to do its job. The action taken at time t prompts the environment to feedback a reward

$r(t+1)$, collected directly by the neuron. We have the new discounted state-value function $\gamma V(t+1)$ from the fast loop, and we have the previous state-value function $V(t)$, corresponding to the previous state, arriving with a delay due to the slow connection. The evaluation neuron then computes

$$\delta(t+1) = r(t+1) + \gamma V(t+1) - V(t) \ ,$$

and feeds this error back onto the incoming synapses, where the eligibility trace of the previous state $\xi_i^\mu(t)$ lingers. The synapses are then updated using the learning rule

$$w(t+1)_i = w(t)_i + \alpha\delta(t+1)\xi_i^\mu(t) \ .$$

This circuit realizes a simple implementation of the TD-error in reinforcement learning.

Guide to the literature

Reinforcement learning is a very active research field in many different disciplines: neuroscience, computer science, machine learning, optimization, control theory and many others. The maze task we described is adapted from abbottdayan [25], with a few changes. This book is the cornerstone of theoretical neuroscience and contains a great chapter devoted to reinforcement learning, comparing theory and experiments, especially with rats. The general theory of reinforcement learning, that we derived from the particular maze task, can be found in suttonbarto [26], which is mainly concerned with a machine learning perspective and is extremely clear and pedagogical. The notations we adopted in this Chapter are from this book. The neural circuit that implements the TD-error is taken from trappenberg [27].

Appendix A. Extras

Appendix A.1. 2d Ising

As we saw in Section subsec:nearest-neighbor 6.3, the 1d Ising model with nearest-neighbor interactions does not undergo a phase transition at low temperature. In dimensions higher than one the nearest-neighbor Ising model does display a phase transition. In this Section, we will give a couple of heuristic arguments to show that the 2d Ising model has a thermodynamic limit and undergoes a phase transition. These two results can be established exactly using the solution of the model, provided by Onsager, however we will just stick to Peierls' hand-waving philosophy to develop some intuition.

Consider a two-dimensional lattice of spins $S_i = \pm 1$ and hamiltonian

$$H = -J \sum_{<i,j>} S_i S_j \ ,$$

where $< i, j >$ is non-vanishing only for nearest-neighbor spins. Let us leave the boundary spins free to fluctuate and prove that this model has a well defined thermodynamic limit. Let us start with a square lattice whose edge has length $2N$, for a total of $4N^2$ spins, and denote its partition function by Z_{2N}. Now split the system into four subsystem, each of which is a square whose edge has length N with partition function Z_N. [ADD FIG] If we want to estimate the energy of the large system by multiplying the partition functions of the smaller subsystems, the error we are making corresponds to neglecting the energies of the four internal edges and amounts to $\pm 4JN$. The estimate for the partition function is therefore bounded by

$$Z_N^4 e^{-4JN\beta} \leq Z_{2N} \leq Z_N^4 e^{+4JN\beta} \ .$$

Translating this into a bound for the free energy densities $f_N = -(\beta N)^{-1} \ln Z_N$ we immediately see that

$$|f_{2N} - f_N| \leq J/N \to 0 \ ,$$

as $N \to \infty$. We have proven that the free energy densities form a Cauchy sequence and hence they converge to a finite thermodynamic limit. At the same time, we have found that boundary effects are washed away by the thermodynamic limit and we can completely neglect them.

Next we want to argue that, unlike its 1d cousin, the 2d Ising model has a phase transition.

References

dadams [1] D. Adams, *The Hitchhiker's Guide to the Galaxy,* Ballantine Books, New York (1979).

kahneman2011thinking [2] D. Kahneman, P. Egan, Thinking, fast and slow, Vol. 1, Farrar, Straus and Giroux New York, 2011.

mackay [3] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press (2003), available at `http://www.inference.phy.cam.ac.uk/mackay/itila/`.

bishop [4] C. M. Bishop, Pattern recognition and machine learning, springer, 2006.

mehta2019high [5] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, D. J. Schwab, A high-bias, low-variance introduction to machine learning for physicists, Physics reports.

towardsbiasvar [6] G. Papachristoudis, The Bias-Variance Tradeoff `https://towardsdatascience.com/the-bias-variance-tradeoff-8818f41e39e9`.

hertz [7] J. Hertz, A. S. Krogh, R. G. Palmer, *Introduction to the Theory of Neural Computation,* Addison-Wesley (1991).

bipoo [8] G. Q. Bi, M. M. Poo, *Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type*, The Journal of Neuroscience 18 (24) (1998) 10464–72, available at `http://www.jneurosci.org/content/18/24/10464.full.pdf+html`.

pauladams [9] P. Adams, Lecture notes for the Stony Brook University course BIO 338: *"From Synapse to Circuit: Self-organisation of the Brain"*, Available at `http://syndar.org/PAdams/intro%20to%20lecture%20notes.htm`.

yuille1989quadrature [10] A. Yuille, D. Kammen, D. Cohen, Quadrature and the development of orientation selective cortical cells by hebb rules, Biological Cybernetics 61 (3) (1989) 183–194.

goodfellow2016deep [11] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT press, 2016.

saxe2013exact [12] A. M. Saxe, J. L. McClelland, S. Ganguli, Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, arXiv preprint arXiv:1312.6120.

bahri2020statistical [13] Y. Bahri, J. Kadmon, J. Pennington, S. S. Schoenholz, J. Sohl-Dickstein, S. Ganguli, Statistical mechanics of deep learning, Annual Review of Condensed Matter Physics available at `https://www.annualreviews.org/doi/pdf/10.1146/annurev-conmatphys-031119-050745`.

lebellac [14] M. Le Bellac, F. Mortessagne, G. G. Batrouni, *Equilibrium and Non-equilibrium Statistical Thermodynamics,* Cambridge University Press (2004).

dalvit [15] D. A. R. Dalvit, J. Frastai, I. D. Lawrie, *Problems on Statistical Mechanics,* IOP (1999).

amit85 [16] D. J. Amit, H. Gutfreund, H. Sompolinsky, *Spin Glass Models of Neural Networks*, Physical Review A 32 (1985) 1007–1018, available at `http://neurophysics.huji.ac.il/~haim/papers/amit-somp-A-85.pdf`.

Hopfield:1982pe [17] J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proc.Nat.Acad.Sci. 79 (1982) 2554–2558. `doi:10.1073/pnas.79.8.2554`.

amit85b [18] D. J. Amit, H. Gutfreund, H. Sompolinsky, *Storing an Infinite Number of Patterns in a Spin-Glass Model of Neural Networks*, Physical Review Letters 55 (1985) 1530–1533, available at `http://neurophysics.huji.ac.il/~haim/papers/amit-somp-85.pdf`.

Amit:1985ff [19] D. J. Amit, H. Gutfreund, H. Sompolinsky, *Spin Glass Models Of Neural Networks,* Available at http://neurophysics.huji.ac.il/~haim/papers/amit-gutfreund-somp-87.pdf.

pargavirasoro [20] N. Parga, M. Virasoro, *The ultrametric organization of memories in a neural network*, J. Physique 47 (1986) 1857–1864, available at http://hal.archives-ouvertes.fr/docs/00/21/03/82/PDF/ajp-jphys_1986_47_11_1857_0.pdf.

parisi [21] M. Mezard, G. Parisi, M. Virasoro, *Spin Glass Theory and Beyond,* World Scientific (1987).

geszti [22] T. Geszti, *Physical Models of Neural Networks,* World Scientific (1990).

amitmodeling [23] D. J. Amit, *Modeling Brain Functions: The world of attractor neural networks,* Cambridge University Press (1989).

Denef:2011ee [24] F. Denef, TASI lectures on complex structures* Temporary entry *. arXiv:1104.0254.

abbottdayan [25] P. Dayan, L. F. Abbott, *Theoretical Neuroscience*, MIT Press (2001), exercises online at http://www.gatsby.ucl.ac.uk/~dayan/book/.

suttonbarto [26] R. S. Sutton, A. G. Barto, *Reinforcement learning: an Introduction*, MIT Press (1998), available at http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html.

trappenberg [27] T. P. Trappenberg, *Fundamentals of Computational Neuroscience*, Oxford University Press (2002),.