Probabilistic Artificial Intelligence

Andreas Krause, Jonas Hübotter

ETH zürich

Institute for Machine Learning
Department of Computer Science

Compiled on **February 11, 2025**.

This manuscript is based on the course Probabilistic Artificial Intelligence (263-5210-00L) at ETH Zürich.

Preface

Artificial intelligence commonly refers to the science and engineering of artificial systems that can carry out tasks generally associated with requiring aspects of human intelligence, such as playing games, translating languages, and driving cars. In recent years, there have been exciting advances in learning-based, data-driven approaches towards AI, and machine learning and deep learning have enabled computer systems to perceive the world in unprecedented ways. Reinforcement learning has enabled breakthroughs in complex games such as Go and challenging robotics tasks such as quadrupedal locomotion.

A key aspect of intelligence is to not only make predictions, but reason about the *uncertainty* in these predictions, and to consider this uncertainty when making decisions. This is what "Probabilistic Artificial Intelligence" is about. The first part covers probabilistic approaches to machine learning. We discuss the differentiation between "epistemic" uncertainty due to lack of data and "aleatoric" uncertainty, which is irreducible and stems, e.g., from noisy observations and outcomes. We discuss concrete approaches towards probabilistic inference, such as Bayesian linear regression, Gaussian process models and Bayesian neural networks. Often, inference and making predictions with such models is intractable, and we discuss modern approaches to efficient approximate inference.

The second part of the manuscript is about taking uncertainty into account in sequential decision tasks. We consider active learning and Bayesian optimization — approaches that collect data by proposing experiments that are informative for reducing the epistemic uncertainty. We then consider reinforcement learning, a rich formalism for modeling agents that learn to act in uncertain environments. After covering the basic formalism of Markov Decision Processes, we consider modern deep RL approaches that use neural network function approximation. We close by discussing modern approaches in model-based RL, which harness epistemic and aleatoric uncertainty to guide exploration, while also reasoning about safety.

Guide to the Reader

The material covered in this manuscript may support a one semester graduate introduction to probabilistic machine learning and sequential decision-making. We welcome readers from all backgrounds. However, we assume familiarity with basic concepts in probability, calculus, linear algebra, and machine learning (e.g., neural networks) as covered in a typical introductory course to machine learning. In Chapter 1, we give a gentle introduction to probabilistic inference, which serves as the foundation for the rest of the manuscript. As part of this first chapter, we also review key concepts from probability theory. We provide a chapter reviewing key concepts of further mathematical background in the back of the manuscript.

Throughout the manuscript, we focus on key concepts and ideas rather than their historical development. We encourage you to consult the provided references for further reading and historical context to delve deeper into the covered topics.

Finally, we have included a set of exercises at the end of each chapter. When we highlight an exercise throughout the text, we use this question mark: ⓘ — so don't be surprised when you stumble upon it. You will find solutions to all exercises in the back of the manuscript.

Problem 1.1

We hope you will find this resource useful.

Contributing

We encourage you to raise issues and suggest fixes for anything you think can be improved. We are thankful for any such feedback!
CONTACT: pai-script@lists.inf.ethz.ch

Acknowledgements

We are grateful to Sebastian Curi for creating the original Jupyter notebooks that accompany the course at ETH Zürich and which were instrumental in the creation of many figures. We thank Hado van Hasselt for kindly contributing Figure 12.1, and thank Tuomas Haarnoja (Haarnoja et al., 2018a) and Roberto Calandra (Chua et al., 2018) for kindly agreeing to have their figures included in this manuscript. Furthermore, many of the exercises in these notes are adapted from iterations of the course at ETH Zürich. Special thanks to all instructors that contributed to the course material over the years. We also thank all students of the course in the Fall of 2022, 2023, and 2024 who provided valuable feedback on various iterations of this manuscript and corrected many

mistakes. Finally, we thank Zhiyuan Hu, Shyam Sundhar Ramesh, Leander Diaz-Bone, Nicolas Menet, and Ido Hakimi for proofreading parts of various drafts of this text.

Contents

1

Fundamentals of Inference

Boolean logic is the algebra of statements which are either true or false. Consider, for example, the statements

"If it is raining, the ground is wet." and "It is raining."

A quite remarkable property of Boolean logic is that we can combine these premises to draw logical inferences which are *new* (true) statements. In the above example, we can conclude that the ground must be wet. This is an example of logical reasoning which is commonly referred to as *logical inference*, and the study of artificial systems that are able to perform logical inference is known as *symbolic artificial intelligence*.

But is it really raining? Perhaps it is hard to tell by looking out of the window. Or we have seen it rain earlier, but some time has passed since we have last looked out of the window. And is it really true that if it rains, the ground is wet? Perhaps the rain is just light enough that it is absorbed quickly, and therefore the ground still appears dry.

This goes to show that in our experience, the real world is rarely black and white. We are frequently (if not usually) uncertain about the truth of statements, and yet we are able to reason about the world and make predictions. We will see that the principles of Boolean logic can be extended to reason in the face of uncertainty. The mathematical framework that allows us to do this is probability theory, which — as we will find in this first chapter — can be seen as a natural extension of Boolean logic from the domain of certainty to the domain of uncertainty. In fact, in the 20th century, Richard Cox and Edwin Thompson Jaynes have done early work to formalize probability theory as the "logic under uncertainty" (Cox, 1961; Jaynes, 2002).

In this first chapter, we will briefly recall the fundamentals of probability theory, and we will see how *probabilistic inference* can be used

to reason about the world. In the remaining chapters, we will then discuss how probabilistic inference can be performed efficiently given limited computational resources and limited time, which is the key challenge in *probabilistic artificial intelligence*.

1.1 Probability

Probability is commonly interpreted in two different ways. In the frequentist interpretation, one interprets the probability of an event (say a coin coming up "heads" when flipping it) as the limit of relative frequencies in repeated independent experiments. That is,

$$\text{Probability} = \lim_{N \to \infty} \frac{\#\text{ events happening in } N \text{ trials}}{N}.$$

This interpretation is natural, but has a few issues. It is not very difficult to conceive of settings where repeated experiments do not make sense. Consider the outcome:

"Person X will live for at least 80 years."

There is no way in which we could conduct multiple independent experiments in this case. Still, this statement is going to turn out either true or false, as humans we are just not able to determine its truth value beforehand. Nevertheless, humans commonly have *beliefs* about statements of this kind. We also commonly reason about statements such as

"The Beatles were more groundbreaking than The Monkees."

This statement does not even have an objective truth value, and yet we as humans tend to have opinions about it.

While it is natural to consider the relative frequency of the outcome in repeated experiments as our belief, if we are not able to conduct repeated experiments, our notion of probability is simply a subjective measure of uncertainty about outcomes. In the early 20th century, Bruno De Finetti has done foundational work to formalize this notion which is commonly called *Bayesian reasoning* or the Bayesian interpretation of probability (De Finetti, 1970).

We will see that modern approaches to probabilistic inference often lend themselves to a Bayesian interpretation, even if such an interpretation is not strictly necessary. For our purposes, probabilities will be a means to an end: the end usually being solving some task. This task may be to make a prediction or to take an action with an uncertain outcome, and we can evaluate methods according to how well they perform on this task. No matter the interpretation, the mathematical

framework of probability theory which we will formally introduce in the following is the same.

1.1.1 Probability Spaces

A probability space is a mathematical model for a random experiment. The set of all possible outcomes of the experiment Ω is called *sample space*. An *event* $A \subseteq \Omega$ of interest may be any combination of possible outcomes. The set of all events $\mathcal{A} \subseteq \mathcal{P}(\Omega)$ that we are interested in is often called the *event space* of the experiment.[1] This set of events is required to be a σ-algebra over the sample space.

[1] We use $\mathcal{P}(\Omega)$ to denote the *power set* (set of all subsets) of Ω.

Definition 1.1 (σ-algebra)**.** Given the set Ω, the set $\mathcal{A} \subseteq \mathcal{P}(\Omega)$ is a *σ-algebra* over Ω if the following properties are satisfied:

1. $\Omega \in \mathcal{A}$;
2. if $A \in \mathcal{A}$, then $\overline{A} \in \mathcal{A}$ (*closedness under complements*); and
3. if we have $A_i \in \mathcal{A}$ for all i, then $\bigcup_{i=1}^{\infty} A_i \in \mathcal{A}$ (*closedness under countable unions*).

Note that the three properties of σ-algebras correspond to characteristics we universally expect when working with random experiments. Namely, that we are able to reason about the event Ω that any of the possible outcomes occur, that we are able to reason about an event not occurring, and that we are able to reason about events that are composed of multiple (smaller) events.

Example 1.2: Event space of throwing a die

The event space $\mathcal{A}$ can also be thought of as "how much information is available about the experiment". For example, if the experiment is a throw of a die and Ω is the set of possible values on the die: $\Omega = \{1, \ldots, 6\}$, then the following $\mathcal{A}$ implies that the observer cannot distinguish between 1 and 3:

$$\mathcal{A} \doteq \{\emptyset, \Omega, \{1, 3, 5\}, \{2, 4, 6\}\}.$$

Intuitively, the observer only understands the parity of the face of the die.

Definition 1.3 (Probability measure)**.** Given the set Ω and the σ-algebra $\mathcal{A}$ over Ω, the function

$$\mathbb{P} : \mathcal{A} \to \mathbb{R}$$

is a *probability measure* on $\mathcal{A}$ if the *Kolmogorov axioms* are satisfied:

1. $0 \leq \mathbb{P}(A) \leq 1$ for any $A \in \mathcal{A}$;
2. $\mathbb{P}(\Omega) = 1$; and

3. $\mathbb{P}(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} \mathbb{P}(A_i)$ for any countable set of mutually disjoint events $\{A_i \in \mathcal{A}\}_i$.[2]

[2] We say that a set of sets $\{A_i\}_i$ is disjoint if for all $i \neq j$ we have $A_i \cap A_j = \emptyset$.

Remarkably, all further statements about probability follow from these three natural axioms. For an event $A \in \mathcal{A}$, we call $\mathbb{P}(A)$ the *probability* of A. We are now ready to define a probability space.

Definition 1.4 (Probability space)**.** A *probability space* is a triple $(\Omega, \mathcal{A}, \mathbb{P})$ where

- Ω is a sample space,
- $\mathcal{A}$ is a σ-algebra over Ω, and
- $\mathbb{P}$ is a probability measure on $\mathcal{A}$.

Example 1.5: Borel σ-algebra over $\mathbb{R}$

In our context, we often have that Ω is the set of real numbers $\mathbb{R}$ or a compact subset of it. In this case, a natural event space is the σ-algebra generated by the set of events

$$A_x \doteq \{x' \in \Omega : x' \leq x\}.$$

The smallest σ-algebra $\mathcal{A}$ containing all sets A_x is called the *Borel σ-algebra*. $\mathcal{A}$ contains all "reasonable" subsets of Ω (except for some pathological examples). For example, $\mathcal{A}$ includes all singleton sets $\{x\}$, as well as all countable unions of intervals.

In the case of discrete Ω, in fact $\mathcal{A} = \mathcal{P}(\Omega)$, i.e., the Borel σ-algebra contains *all* subsets of Ω.

1.1.2 Random Variables

The set Ω is often rather complex. For example, take Ω to be the set of all possible graphs on n vertices. Then the outcome of our experiment is a graph. Usually, we are not interested in a specific graph but rather a property such as the number of edges, which is shared by many graphs. A function that maps a graph to its number of edges is a random variable.

Definition 1.6 (Random variable)**.** A *random variable* X is a function

$$X : \Omega \to \mathcal{T}$$

where $\mathcal{T}$ is called *target space* of the random variable,[3] and where X respects the information available in the σ-algebra $\mathcal{A}$. That is,[4]

$$\forall S \subseteq \mathcal{T} : \quad \{\omega \in \Omega : X(\omega) \in S\} \in \mathcal{A}. \tag{1.1}$$

[3] For a random variable that maps a graph to its number of edges, $\mathcal{T} = \mathbb{N}_0$. For our purposes, you can generally assume $\mathcal{T} \subseteq \mathbb{R}$.

[4] In our example of throwing a die, X should assign the same value to the outcomes $1, 3, 5$.

Concrete values x of a random variable X are often referred to as *states* or *realizations* of X. The probability that X takes on a value in $S \subseteq \mathcal{T}$ is

$$\mathbb{P}(X \in S) = \mathbb{P}(\{\omega \in \Omega : X(\omega) \in S\}). \tag{1.2}$$

1.1.3 Distributions

Consider a random variable X on a probability space $(\Omega, \mathcal{A}, \mathbb{P})$, where Ω is a compact subset of $\mathbb{R}$, and $\mathcal{A}$ the Borel σ-algebra.

In this case, we can refer to the probability that X assumes a particular state or set of states by writing

$$p_X(x) \doteq \mathbb{P}(X = x) \quad \text{(in the discrete setting)}, \tag{1.3}$$

$$P_X(x) \doteq \mathbb{P}(X \leq x). \tag{1.4}$$

Note that "$X = x$" and "$X \leq x$" are merely events (that is, they characterize subsets of the sample space Ω satisfying this condition) which are in the Borel σ-algebra, and hence their probability is well-defined.

Hereby, p_X and P_X are referred to as the probability mass function (PMF) and cumulative distribution function (CDF) of X, respectively. Note that we can also *implicitly* define probability spaces through random variables and their associated PMF/CDF, which is often very convenient.

We list some common examples of discrete distributions in Appendix A.1.1. Further, note that for continuous variables, $\mathbb{P}(X = x) = 0$. Here, instead we typically use the probability density function (PDF), to which we (with slight abuse of notation) also refer with p_X. We discuss densities in greater detail in Section 1.1.4.

We call the subset $S \subseteq \mathcal{T}$ of the domain of a PMF or PDF p_X such that all elements $x \in S$ have positive probability, $p_X(x) > 0$, the *support* of the distribution p_X. This quantity is denoted by $X(\Omega)$.

1.1.4 Continuous Distributions

As mentioned, a continuous random variable can be characterized by its *probability density function* (PDF). But what is a density? We can derive some intuition from physics.

Let M be a (non-homogeneous) physical object, e.g., a rock. We commonly use $m(M)$ and $\mathrm{vol}(M)$ to refer to its mass and volume, respectively. Now, consider for a point $\boldsymbol{x} \in M$ and a ball $B_r(\boldsymbol{x})$ around $\boldsymbol{x}$ with radius r the following quantities:

$$\lim_{r \to 0} \mathrm{vol}(B_r(\boldsymbol{x})) = 0 \quad \lim_{r \to 0} m(B_r(\boldsymbol{x})) = 0.$$

They appear utterly uninteresting at first, yet, if we divide them, we get what is called the *density* of M at $\boldsymbol{x}$.

$$\lim_{r \to 0} \frac{m(B_r(\boldsymbol{x}))}{\operatorname{vol}(B_r(\boldsymbol{x}))} \doteq \rho(\boldsymbol{x}).$$

We know that the relationship between density and mass is described by the following formula:

$$m(M) = \int_M \rho(\boldsymbol{x}) \, d\boldsymbol{x}.$$

In other words, the density is to be integrated. For a small region I around $\boldsymbol{x}$, we can approximate $m(I) \approx \rho(\boldsymbol{x}) \cdot \operatorname{vol}(I)$.

Crucially, observe that even though the mass of any particular point $\boldsymbol{x}$ is zero, i.e., $m(\{\boldsymbol{x}\}) = 0$, assigning a density $\rho(\boldsymbol{x})$ to $\boldsymbol{x}$ is useful for integration and approximation. The same idea applies to continuous random variables, only that volume corresponds to intervals on the real line and mass to probability. Recall that probability density functions are normalized such that their probability mass across the entire real line integrates to one.

Example 1.7: Normal distribution / Gaussian

A famous example of a continuous distribution is the *normal distribution*, also called *Gaussian*. We say, a random variable X is *normally distributed*, $X \sim \mathcal{N}(\mu, \sigma^2)$, if its PDF is

$$\mathcal{N}(x; \mu, \sigma^2) \doteq \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{1.5}$$

We have $\mathbb{E}[X] = \mu$ and $\operatorname{Var}[X] = \sigma^2$. If $\mu = 0$ and $\sigma^2 = 1$, this distribution is called the *standard normal distribution*. The Gaussian CDF cannot be expressed in closed-form.

Note that the mean of a Gaussian distribution coincides with the maximizer of its PDF, also called *mode* of a distribution.

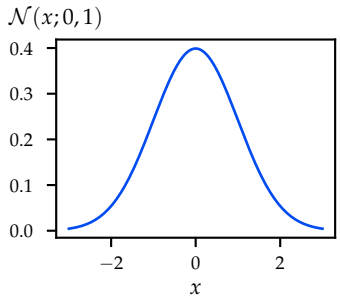


Figure 1.1: PDF of the standard normal distribution. Observe that the PDF is symmetric around the mode.

We will focus in the remainder of this chapter on continuous distributions, but the concepts we discuss extend mostly to discrete distributions simply by "replacing integrals by sums".

1.1.5 *Joint Probability*

A joint probability (as opposed to a marginal probability) is the probability of two or more events occurring simultaneously:

$$\mathbb{P}(A, B) \doteq \mathbb{P}(A \cap B). \tag{1.6}$$

In terms of random variables, this concept extends to joint distributions. Instead of characterizing a single random variable, a *joint distribution* is a function $p_{\mathbf{X}} : \mathbb{R}^n \to \mathbb{R}$, characterizing a *random vector* $\mathbf{X} \doteq [X_1 \ \cdots \ X_n]^\top$. For example, if the X_i are discrete, the joint distribution characterizes joint probabilities of the form

$$\mathbb{P}(\mathbf{X} = [x_1, \ldots, x_n]) = \mathbb{P}(X_1 = x_1, \ldots, X_n = x_n),$$

and hence describes the relationship among all variables X_i. For this reason, a joint distribution is also called a *generative model*. We use $X_{i:j}$ to denote the random vector $[X_i \ \cdots \ X_j]^\top$.

We can "sum out" (respectively "integrate out") variables from a joint distribution in a process called "marginalization":

Fact 1.8 (Sum rule). *We have that*

$$p(x_{1:i-1}, x_{i+1:n}) = \int_{X_i(\Omega)} p(x_{1:i-1}, x_i, x_{i+1:n}) \, dx_i. \tag{1.7}$$

1.1.6 Conditional Probability

Conditional probability updates the probability of an event A given some new information, for example, after observing the event B.

Definition 1.9 (Conditional probability)**.** Given two events A and B such that $\mathbb{P}(B) > 0$, the probability of A conditioned on B is given as

$$\mathbb{P}(A \mid B) \doteq \frac{\mathbb{P}(A, B)}{\mathbb{P}(B)}. \tag{1.8}$$

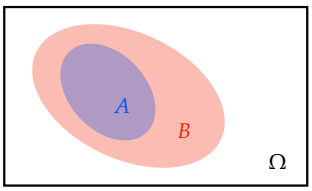


Figure 1.2: Conditioning an event A on another event B can be understood as replacing the universe of all possible outcomes Ω by the observed outcomes B. Then, the conditional probability is simply expressing the likelihood of A given that B occurred.

Simply rearranging the terms yields,

$$\mathbb{P}(A, B) = \mathbb{P}(A \mid B) \cdot \mathbb{P}(B) = \mathbb{P}(B \mid A) \cdot \mathbb{P}(A). \tag{1.9}$$

Thus, the probability that both A and B occur can be calculated by multiplying the probability of event A and the probability of B conditional on A occurring.

We say $\mathbf{Z} \sim \mathbf{X} \mid \mathbf{Y} = \boldsymbol{y}$ (or simply $\mathbf{Z} \sim \mathbf{X} \mid \boldsymbol{y}$) if $\mathbf{Z}$ follows the *conditional distribution*

$$p_{\mathbf{X}|\mathbf{Y}}(\boldsymbol{x} \mid \boldsymbol{y}) \doteq \frac{p_{\mathbf{X},\mathbf{Y}}(\boldsymbol{x}, \boldsymbol{y})}{p_{\mathbf{Y}}(\boldsymbol{y})}. \tag{1.10}$$

If $\mathbf{X}$ and $\mathbf{Y}$ are discrete, we have that $p_{\mathbf{X}|\mathbf{Y}}(\boldsymbol{x} \mid \boldsymbol{y}) = \mathbb{P}(\mathbf{X} = \boldsymbol{x} \mid \mathbf{Y} = \boldsymbol{y})$ as one would naturally expect.

Extending Equation (1.9) to arbitrary random vectors yields the product rule (also called the *chain rule of probability*):

Fact 1.10 (Product rule). *Given random variables $X_{1:n}$,*

$$p(x_{1:n}) = p(x_1) \cdot \prod_{i=2}^{n} p(x_i \mid x_{1:i-1}). \tag{1.11}$$

Combining sum rule and product rule, we can compute marginal probabilities too:

$$p(\boldsymbol{x}) = \int_{\mathbf{Y}(\Omega)} p(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{y} = \int_{\mathbf{Y}(\Omega)} p(\boldsymbol{x} \mid \boldsymbol{y}) \cdot p(\boldsymbol{y})\, d\boldsymbol{y} \tag{1.12}$$

first using the sum rule (1.7) then the product rule (1.11)

This is called the *law of total probability* (LOTP), which is colloquially often referred to as *conditioning* on $\mathbf{Y}$. If it is difficult to compute $p(\boldsymbol{x})$ directly, conditioning can be a useful technique when $\mathbf{Y}$ is chosen such that the densities $p(\boldsymbol{x} \mid \boldsymbol{y})$ and $p(\boldsymbol{y})$ are straightforward to understand.

1.1.7 Independence

Two random vectors $\mathbf{X}$ and $\mathbf{Y}$ are *independent* (denoted $\mathbf{X} \perp \mathbf{Y}$) if and only if knowledge about the state of one random vector does not affect the distribution of the other random vector, namely if their conditional CDF (or in case they have a joint density, their conditional PDF) simplifies to

$$P_{\mathbf{X}|\mathbf{Y}}(\boldsymbol{x} \mid \boldsymbol{y}) = P_{\mathbf{X}}(\boldsymbol{x}), \quad p_{\mathbf{X}|\mathbf{Y}}(\boldsymbol{x} \mid \boldsymbol{y}) = p_{\mathbf{X}}(\boldsymbol{x}). \tag{1.13}$$

For the conditional probabilities to be well-defined, we need to assume that $p_{\mathbf{Y}}(\boldsymbol{y}) > 0$.

The more general characterization of independence is that $\mathbf{X}$ and $\mathbf{Y}$ are independent if and only if their joint CDF (or in case they have a joint density, their joint PDF) can be decomposed as follows:

$$P_{\mathbf{X},\mathbf{Y}}(\boldsymbol{x}, \boldsymbol{y}) = P_{\mathbf{X}}(\boldsymbol{x}) \cdot P_{\mathbf{Y}}(\boldsymbol{y}), \quad p_{\mathbf{X},\mathbf{Y}}(\boldsymbol{x}, \boldsymbol{y}) = p_{\mathbf{X}}(\boldsymbol{x}) \cdot p_{\mathbf{Y}}(\boldsymbol{y}). \tag{1.14}$$

The equivalence of the two characterizations (when $p_{\mathbf{Y}}(\boldsymbol{y}) > 0$) is easily proven using the product rule: $p_{\mathbf{X},\mathbf{Y}}(\boldsymbol{x}, \boldsymbol{y}) = p_{\mathbf{Y}}(\boldsymbol{y}) \cdot p_{\mathbf{X}|\mathbf{Y}}(\boldsymbol{x} \mid \boldsymbol{y})$.

A "weaker" notion of independence is conditional independence.[5] Two random vectors $\mathbf{X}$ and $\mathbf{Y}$ are *conditionally independent* given a random vector $\mathbf{Z}$ (denoted $\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z}$) iff, given $\mathbf{Z}$, knowledge about the value of one random vector $\mathbf{Y}$ does not affect the distribution of the other random vector $\mathbf{X}$, namely if

$$P_{\mathbf{X}|\mathbf{Y},\mathbf{Z}}(\boldsymbol{x} \mid \boldsymbol{y}, \boldsymbol{z}) = P_{\mathbf{X}|\mathbf{Z}}(\boldsymbol{x} \mid \boldsymbol{z}), \tag{1.15a}$$

$$p_{\mathbf{X}|\mathbf{Y},\mathbf{Z}}(\boldsymbol{x} \mid \boldsymbol{y}, \boldsymbol{z}) = p_{\mathbf{X}|\mathbf{Z}}(\boldsymbol{x} \mid \boldsymbol{z}). \tag{1.15b}$$

[5] We discuss in Remark 1.11 how "weaker" is to be interpreted in this context.

Similarly to independence, we have that $\mathbf{X}$ and $\mathbf{Y}$ are conditionally independent given $\mathbf{Z}$ if and only if their joint CDF or joint PDF can be decomposed as follows:

$$P_{\mathbf{X},\mathbf{Y}|\mathbf{Z}}(\boldsymbol{x},\boldsymbol{y} \mid \boldsymbol{z}) = P_{\mathbf{X}|\mathbf{Z}}(\boldsymbol{x} \mid \boldsymbol{z}) \cdot P_{\mathbf{Y}|\mathbf{Z}}(\boldsymbol{y} \mid \boldsymbol{z}), \tag{1.16a}$$

$$p_{\mathbf{X},\mathbf{Y}|\mathbf{Z}}(\boldsymbol{x},\boldsymbol{y} \mid \boldsymbol{z}) = p_{\mathbf{X}|\mathbf{Z}}(\boldsymbol{x} \mid \boldsymbol{z}) \cdot p_{\mathbf{Y}|\mathbf{Z}}(\boldsymbol{y} \mid \boldsymbol{z}). \tag{1.16b}$$

Remark 1.11: Common causes
How can conditional independence be understood as a "weaker" notion of independence? Clearly, conditional independence does not imply independence: a trivial example is $X \perp X \mid X \not\Rightarrow X \perp X$.[6] Neither does independence imply conditional independence: for example, $X \perp Y \not\Rightarrow X \perp Y \mid X + Y$.[7]

When we say that conditional independence is a weaker notion we mean to emphasize that X and Y can be "made" (conditionally) independent by conditioning on the "right" Z even if X and Y are dependent. This is known as *Reichenbach's common cause principle* which says that for any two random variables $X \not\perp Y$ there exists a random variable Z (which may be X or Y) that causally influences both X and Y, and which is such that $X \perp Y \mid Z$.

[6] $X \perp X \mid X$ is true trivially.

[7] Knowing X and $X + Y$ already implies the value of Y, and hence, $X \not\perp Y \mid X + Y$.

1.1.8 Directed Graphical Models

Directed graphical models (also called *Bayesian networks*) are often used to visually denote the (conditional) independence relationships of a large number of random variables. They are a schematic representation of the factorization of the generative model into a product of conditional distributions as a directed acyclic graph. Given the sequence of random variables $\{X_i\}_{i=1}^n$, their generative model can be expressed as

$$p(x_{1:n}) = \prod_{i=1}^{n} p(x_i \mid \text{parents}(x_i)) \tag{1.17}$$

where $\text{parents}(x_i)$ is the set of parents of the vertex X_i in the directed graphical model. In other words, the parenthood relationship encodes a conditional independence of a random variable X with a random variable Y given their parents:[8]

$$X \perp Y \mid \text{parents}(X), \text{parents}(Y). \tag{1.18}$$

Equation (1.17) simply uses the product rule and the conditional independence relationships to factorize the generative model. This can greatly reduce the model's complexity, i.e., the length of the product.

[8] More generally, vertices u and v are conditionally independent given a set of vertices Z if Z *d-separates* u and v, which we will not cover in depth here.

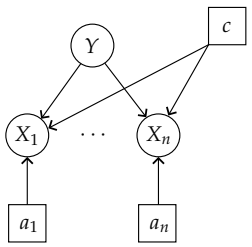


Figure 1.3: Example of a directed graphical model. The random variables $X_1, \ldots, X_n$ are mutually independent given the random variable Y. The squared rectangular nodes are used to represent dependencies on parameters $c, a_1, \ldots, a_n$.

An example of a directed graphical model is given in Figure 1.3. Circular vertices represent random quantities (i.e., random variables). In contrast, square vertices are commonly used to represent deterministic quantities (i.e., parameters that the distributions depend on). In the given example, we have that X_i is conditionally independent of all other X_j given Y. *Plate notation* is a condensed notation used to represent repeated variables of a graphical model. An example is given in Figure 1.4.

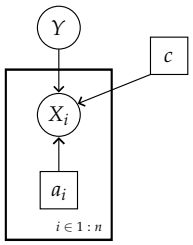


Figure 1.4: The same directed graphical model as in Figure 1.3 using plate notation.

1.1.9 Expectation

The *expected value* or *mean* $\mathbb{E}[\mathbf{X}]$ of a random vector $\mathbf{X}$ is the (asymptotic) arithmetic mean of an arbitrarily increasing number of independent realizations of $\mathbf{X}$. That is,[9]

$$\mathbb{E}[\mathbf{X}] \doteq \int_{\mathbf{X}(\Omega)} \boldsymbol{x} \cdot p(\boldsymbol{x})\, d\boldsymbol{x} \tag{1.19}$$

[9] In infinite probability spaces, absolute convergence of $\mathbb{E}[\mathbf{X}]$ is necessary for the existence of $\mathbb{E}[\mathbf{X}]$.

A very special and often used property of expectations is their *linearity*, namely that for any random vectors $\mathbf{X}$ and $\mathbf{Y}$ in $\mathbb{R}^n$ and any $\boldsymbol{A} \in \mathbb{R}^{m \times n}, \boldsymbol{b} \in \mathbb{R}^m$ it holds that

$$\mathbb{E}[\boldsymbol{A}\mathbf{X} + \boldsymbol{b}] = \boldsymbol{A}\mathbb{E}[\mathbf{X}] + \boldsymbol{b} \quad \text{and} \quad \mathbb{E}[\mathbf{X} + \mathbf{Y}] = \mathbb{E}[\mathbf{X}] + \mathbb{E}[\mathbf{Y}]. \tag{1.20}$$

Note that $\mathbf{X}$ and $\mathbf{Y}$ do not necessarily have to be independent! Further, if $\mathbf{X}$ and $\mathbf{Y}$ are independent then

$$\mathbb{E}\left[\mathbf{X}\mathbf{Y}^\top\right] = \mathbb{E}[\mathbf{X}] \cdot \mathbb{E}[\mathbf{Y}]^\top. \tag{1.21}$$

The following intuitive lemma can be used to compute expectations of transformed random variables.

Fact 1.12 (Law of the unconscious statistician, LOTUS)**.**

$$\mathbb{E}[g(\mathbf{X})] = \int_{\mathbf{X}(\Omega)} g(\boldsymbol{x}) \cdot p(\boldsymbol{x})\, d\boldsymbol{x} \tag{1.22}$$

where $g : \mathbf{X}(\Omega) \to \mathbb{R}^n$ is a "nice" function[10] and $\mathbf{X}$ is a continuous random vector. The analogous statement with a sum replacing the integral holds for discrete random variables.

[10] g being a continuous function, which is either bounded or absolutely integrable (i.e., $\int |g(\boldsymbol{x})|\, p(\boldsymbol{x})\, d\boldsymbol{x} < \infty$), is sufficient. This is satisfied in most cases.

This is a nontrivial fact that can be proven using the change of variables formula which we discuss in Section 1.1.11.

Similarly to conditional probability, we can also define conditional expectations. The expectation of a continuous random vector $\mathbf{X}$ given that $\mathbf{Y} = \boldsymbol{y}$ is defined as

$$\mathbb{E}[\mathbf{X} \mid \mathbf{Y} = \boldsymbol{y}] \doteq \int_{\mathbf{X}(\Omega)} \boldsymbol{x} \cdot p_{\mathbf{X}|\mathbf{Y}}(\boldsymbol{x} \mid \boldsymbol{y})\, d\boldsymbol{x}. \tag{1.23}$$

Observe that $\mathbb{E}[\mathbf{X} \mid \mathbf{Y} = \cdot]$ defines a deterministic mapping from $\boldsymbol{y}$ to $\mathbb{E}[\mathbf{X} \mid \mathbf{Y} = \boldsymbol{y}]$. Therefore, $\mathbb{E}[\mathbf{X} \mid \mathbf{Y}]$ is itself a random vector:

$$\mathbb{E}[\mathbf{X} \mid \mathbf{Y}](\omega) = \mathbb{E}[\mathbf{X} \mid \mathbf{Y} = \mathbf{Y}(\omega)] \tag{1.24}$$

where $\omega \in \Omega$. This random vector $\mathbb{E}[\mathbf{X} \mid \mathbf{Y}]$ is called the *conditional expectation* of $\mathbf{X}$ given $\mathbf{Y}$.

Analogously to the law of total probability (1.12), one can condition an expectation on another random vector. This is known as the *tower rule* or the *law of total expectation* (LOTE):

Theorem 1.13 (Tower rule)**.** *Given random vectors* $\mathbf{X}$ *and* $\mathbf{Y}$*, we have*

$$\mathbb{E}_{\mathbf{Y}}[\mathbb{E}_{\mathbf{X}}[\mathbf{X} \mid \mathbf{Y}]] = \mathbb{E}[\mathbf{X}]. \tag{1.25}$$

Proof sketch. We only prove the case where $\mathbf{X}$ and $\mathbf{Y}$ have a joint density. We have

$$\begin{aligned}
\mathbb{E}[\mathbb{E}[\mathbf{X} \mid \mathbf{Y}]] &= \int \left(\int \boldsymbol{x} \cdot p(\boldsymbol{x} \mid \boldsymbol{y}) \, d\boldsymbol{x} \right) p(\boldsymbol{y}) \, d\boldsymbol{y} \\
&= \int \int \boldsymbol{x} \cdot p(\boldsymbol{x}, \boldsymbol{y}) \, d\boldsymbol{x} \, d\boldsymbol{y} && \text{by definition of conditional densities (1.10)} \\
&= \int \boldsymbol{x} \int p(\boldsymbol{x}, \boldsymbol{y}) \, d\boldsymbol{y} \, d\boldsymbol{x} && \text{by Fubini's theorem} \\
&= \int \boldsymbol{x} \cdot p(\boldsymbol{x}) \, d\boldsymbol{x} && \text{using the sum rule (1.7)} \\
&= \mathbb{E}[\mathbf{X}]. && \square
\end{aligned}$$

1.1.10 Covariance and Variance

Given two random vectors $\mathbf{X}$ in $\mathbb{R}^n$ and $\mathbf{Y}$ in $\mathbb{R}^m$, their *covariance* is defined as

$$\begin{aligned}
\operatorname{Cov}[\mathbf{X}, \mathbf{Y}] &\doteq \mathbb{E}\left[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{Y} - \mathbb{E}[\mathbf{Y}])^\top \right] && (1.26) \\
&= \mathbb{E}\left[\mathbf{X}\mathbf{Y}^\top \right] - \mathbb{E}[\mathbf{X}] \cdot \mathbb{E}[\mathbf{Y}]^\top && (1.27) \\
&= \operatorname{Cov}[\mathbf{Y}, \mathbf{X}]^\top \in \mathbb{R}^{n \times m}. && (1.28)
\end{aligned}$$

Covariance measures the linear dependence between two random vectors since a direct consequence of its definition (1.26) is that given linear maps $\boldsymbol{A} \in \mathbb{R}^{n' \times n}, \boldsymbol{B} \in \mathbb{R}^{m' \times m}$, vectors $\boldsymbol{c} \in \mathbb{R}^{n'}, \boldsymbol{d} \in \mathbb{R}^{m'}$ and random vectors $\mathbf{X}$ in $\mathbb{R}^n$ and $\mathbf{Y}$ in $\mathbb{R}^m$, we have that

$$\operatorname{Cov}[\boldsymbol{A}\mathbf{X} + \boldsymbol{c}, \boldsymbol{B}\mathbf{Y} + \boldsymbol{d}] = \boldsymbol{A}\operatorname{Cov}[\mathbf{X}, \mathbf{Y}]\boldsymbol{B}^\top. \tag{1.29}$$

Two random vectors $\mathbf{X}$ and $\mathbf{Y}$ are said to be *uncorrelated* if and only if $\operatorname{Cov}[\mathbf{X}, \mathbf{Y}] = \mathbf{0}$. Note that if $\mathbf{X}$ and $\mathbf{Y}$ are independent, then Equation (1.21) implies that $\mathbf{X}$ and $\mathbf{Y}$ are uncorrelated. The reverse does not hold in general.

Remark 1.14: Correlation

The *correlation* of the random vectors $\mathbf{X}$ and $\mathbf{Y}$ is a normalized covariance,

$$\operatorname{Cor}[\mathbf{X}, \mathbf{Y}](i,j) \doteq \frac{\operatorname{Cov}\left[X_i, Y_j\right]}{\sqrt{\operatorname{Var}[X_i]\operatorname{Var}\left[Y_j\right]}} \in [-1, 1]. \tag{1.30}$$

Two random vectors $\mathbf{X}$ and $\mathbf{Y}$ are therefore uncorrelated if and only if $\operatorname{Cor}[\mathbf{X}, \mathbf{Y}] = \mathbf{0}$.

There is also a nice geometric interpretation of covariance and correlation. For zero mean random variables X and Y, $\operatorname{Cov}[X, Y]$ is an inner product.[11]

The cosine of the angle θ between X and Y (that are not deterministic) coincides with their correlation,

$$\cos\theta = \frac{\operatorname{Cov}[X, Y]}{\|X\| \, \|Y\|} = \operatorname{Cor}[X, Y]. \tag{1.31}$$

$\cos\theta$ is also called a *cosine similarity*. Thus,

$$\theta = \arccos \operatorname{Cor}[X, Y]. \tag{1.32}$$

For example, if X and Y are uncorrelated, then they are orthogonal in the inner product space. If $\operatorname{Cor}[X, Y] = -1$ then $\theta \equiv \pi$ (that is, X and Y "point in opposite directions"), whereas if $\operatorname{Cor}[X, Y] = 1$ then $\theta \equiv 0$ (that is, X and Y "point in the same direction").

[11] That is,

- $\operatorname{Cov}[X, Y]$ is symmetric,
- $\operatorname{Cov}[X, Y]$ is linear (here we use $\mathbb{E}X = \mathbb{E}Y = 0$), and
- $\operatorname{Cov}[X, X] \geq 0$.

using the Euclidean inner product formula, $\operatorname{Cov}[X, Y] = \|X\| \, \|Y\| \cos\theta$

The covariance of a random vector $\mathbf{X}$ in $\mathbb{R}^n$ with itself is called its *variance*:

$$\begin{aligned}
\operatorname{Var}[\mathbf{X}] &\doteq \operatorname{Cov}[\mathbf{X}, \mathbf{X}] && (1.33)\\
&= \mathbb{E}\left[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^\top\right] && (1.34)\\
&= \mathbb{E}\left[\mathbf{X}\mathbf{X}^\top\right] - \mathbb{E}[\mathbf{X}] \cdot \mathbb{E}[\mathbf{X}]^\top && (1.35)\\
&= \begin{bmatrix} \operatorname{Cov}[X_1, X_1] & \cdots & \operatorname{Cov}[X_1, X_n] \\ \vdots & \ddots & \vdots \\ \operatorname{Cov}[X_n, X_1] & \cdots & \operatorname{Cov}[X_n, X_n] \end{bmatrix}. && (1.36)
\end{aligned}$$

The scalar variance $\operatorname{Var}[X]$ of a random variable X is a measure of uncertainty about the value of X since it measures the average squared deviation from $\mathbb{E}[X]$. We will see that the eigenvalue spectrum of a covariance matrix can serve as a measure of uncertainty in the multivariate setting.[12]

[12] The *multivariate* setting (as opposed to the *univariate* setting) studies the joint distribution of multiple random variables.

Remark 1.15: Standard deviation

The length of a random variable X in the inner product space described in Remark 1.14 is called its *standard deviation*,

$$\|X\| = \sqrt{\mathrm{Cov}[X, X]} = \sqrt{\mathrm{Var}[X]} \doteq \sigma[X]. \tag{1.37}$$

That is, the longer a random variable is in the inner product space, the more "uncertain" we are about its value. If a random variable has length 0, then it is deterministic.

The variance of a random vector $\mathbf{X}$ is also called the *covariance matrix* of $\mathbf{X}$ and denoted by $\boldsymbol{\Sigma}_{\mathbf{X}}$ (or $\boldsymbol{\Sigma}$ if the correspondence to $\mathbf{X}$ is clear from context). A covariance matrix is symmetric by definition due to the symmetry of covariance, and is always positive semi-definite (?). **Problem 1.4**

Two useful properties of variance are the following:

- It follows from Equation (1.29) that for any linear map $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and vector $\boldsymbol{b} \in \mathbb{R}^m$,

  $$\mathrm{Var}[\boldsymbol{A}\mathbf{X} + \boldsymbol{b}] = \boldsymbol{A}\mathrm{Var}[\mathbf{X}]\boldsymbol{A}^\top. \tag{1.38}$$

  In particular, $\mathrm{Var}[-\mathbf{X}] = \mathrm{Var}[\mathbf{X}]$.
- It follows from the definition of variance (1.34) that for any two random vectors $\mathbf{X}$ and $\mathbf{Y}$,

  $$\mathrm{Var}[\mathbf{X} + \mathbf{Y}] = \mathrm{Var}[\mathbf{X}] + \mathrm{Var}[\mathbf{Y}] + 2\mathrm{Cov}[\mathbf{X}, \mathbf{Y}]. \tag{1.39}$$

  In particular, if $\mathbf{X}$ and $\mathbf{Y}$ are independent then the covariance term vanishes and $\mathrm{Var}[\mathbf{X} + \mathbf{Y}] = \mathrm{Var}[\mathbf{X}] + \mathrm{Var}[\mathbf{Y}]$.

Analogously to conditional probability and conditional expectation, we can also define conditional variance. The *conditional variance* of a random vector $\mathbf{X}$ given another random vector $\mathbf{Y}$ is the random vector

$$\mathrm{Var}[\mathbf{X} \mid \mathbf{Y}] \doteq \mathbb{E}\left[(\mathbf{X} - \mathbb{E}[\mathbf{X} \mid \mathbf{Y}])(\mathbf{X} - \mathbb{E}[\mathbf{X} \mid \mathbf{Y}])^\top \;\middle|\; \mathbf{Y}\right]. \tag{1.40}$$

Intuitively, the conditional variance is the remaining variance when we use $\mathbb{E}[\mathbf{X} \mid \mathbf{Y}]$ to predict $\mathbf{X}$ rather than if we used $\mathbb{E}[\mathbf{X}]$. One can also condition a variance on another random vector, analogously to the laws of total probability (1.12) and expectation (1.25).

Theorem 1.16 (Law of total variance, LOTV)**.**

$$\mathrm{Var}[\mathbf{X}] = \mathbb{E}_{\mathbf{Y}}[\mathrm{Var}_{\mathbf{X}}[\mathbf{X} \mid \mathbf{Y}]] + \mathrm{Var}_{\mathbf{Y}}[\mathbb{E}_{\mathbf{X}}[\mathbf{X} \mid \mathbf{Y}]]. \tag{1.41}$$

Here, the first term measures the average deviation from the mean of $\mathbf{X}$ across realizations of $\mathbf{Y}$ and the second term measures the uncertainty

in the mean of $\mathbf{X}$ across realizations of $\mathbf{Y}$. In Section 2.2, we will see that both terms have a meaningful characterization in the context of probabilistic inference.

Proof sketch of LOTV. To simplify the notation, we present only a proof for the univariate setting.

$$\begin{aligned}
\mathrm{Var}[X] &= \mathbb{E}\Big[X^2\Big] - \mathbb{E}[X]^2 \\
&= \mathbb{E}\Big[\mathbb{E}\Big[X^2 \mid Y\Big]\Big] - \mathbb{E}[\mathbb{E}[X \mid Y]]^2 && \text{by the tower rule (1.25)} \\
&= \mathbb{E}\Big[\mathrm{Var}[X \mid Y] + \mathbb{E}[X \mid Y]^2\Big] - \mathbb{E}[\mathbb{E}[X \mid Y]]^2 && \text{by the definition of variance (1.35)} \\
&= \mathbb{E}[\mathrm{Var}[X \mid Y]] + \Big(\mathbb{E}\Big[\mathbb{E}[X \mid Y]^2\Big] - \mathbb{E}[\mathbb{E}[X \mid Y]]^2\Big) \\
&= \mathbb{E}[\mathrm{Var}[X \mid Y]] + \mathrm{Var}[\mathbb{E}[X \mid Y]]. && \square \quad \text{by the definition of variance (1.35)}
\end{aligned}$$

1.1.11 Change of Variables

It is often useful to understand the distribution of a transformed random variable $Y = g(X)$ that is defined in terms of a random variable X, whose distribution is known. Let us first consider the univariate setting. We would like to express the distribution of Y in terms of the distribution of X, that is, we would like to find

$$P_Y(y) = \mathbb{P}(Y \leq y) = \mathbb{P}(g(X) \leq y) = \mathbb{P}\Big(X \leq g^{-1}(y)\Big). \tag{1.42}$$

When the random variables are continuous, this probability can be expressed as an integration over the domain of X. We can then use the substitution rule of integration to "change the variables" to an integration over the domain of Y. Taking the derivative yields the density p_Y.[13] There is an analogous change of variables formula for the multivariate setting.

[13] The full proof of the change of variables formula in the univariate setting can be found in section 6.7.2 of "Mathematics for machine learning" (Deisenroth et al., 2020).

Fact 1.17 (Change of variables formula)**.** *Let $\mathbf{X}$ be a random vector in $\mathbb{R}^n$ with density $p_{\mathbf{X}}$ and let $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^n$ be a differentiable and invertible function. Then $\mathbf{Y} = \boldsymbol{g}(\mathbf{X})$ is another random variable, whose density can be computed based on $p_{\mathbf{X}}$ and $\boldsymbol{g}$ as follows:*

$$p_{\mathbf{Y}}(\boldsymbol{y}) = p_{\mathbf{X}}(\boldsymbol{g}^{-1}(\boldsymbol{y})) \cdot \left|\det\Big(\boldsymbol{D}\boldsymbol{g}^{-1}(\boldsymbol{y})\Big)\right| \tag{1.43}$$

where $\boldsymbol{D}\boldsymbol{g}^{-1}(\boldsymbol{y})$ is the Jacobian of $\boldsymbol{g}^{-1}$ evaluated at $\boldsymbol{y}$.

Here, the term $\left|\det\big(\boldsymbol{D}\boldsymbol{g}^{-1}(\boldsymbol{y})\big)\right|$ measures how much a unit volume changes when applying $\boldsymbol{g}$. Intuitively, the change of variables swaps the coordinate system over which we integrate. The factor $\left|\det\big(\boldsymbol{D}\boldsymbol{g}^{-1}(\boldsymbol{y})\big)\right|$ corrects for the change in volume that is caused by this change in coordinates.

Intuitively, you can think of the vector field g as a perturbation to $\mathbf{X}$, "pushing" the probability mass around. The perturbation of a density $p_{\mathbf{X}}$ by g is commonly denoted by the *pushforward*

$$g_\sharp p_{\mathbf{X}} \doteq p_{\mathbf{Y}} \quad \text{where } \mathbf{Y} = g(\mathbf{X}). \tag{1.44}$$

This concludes our quick tour of probability theory, and we are well-prepared to return to the topic of probabilistic inference.

1.2 *Probabilistic Inference*

Recall the logical implication "If it is raining, the ground is wet." from the beginning of this chapter. Suppose that we look outside a window and see that it is not raining: will the ground be dry? Logical reasoning does not permit drawing an inference of this kind, as there might be reasons other than rain for which the ground could be wet (e.g., sprinklers). However, intuitively, by observing that it is not raining, we have just excluded the possibility that the ground is wet because of rain, and therefore we would deem it "more likely" that the ground is dry than before. In other words, if we were to walk outside now and the ground was wet, we would be more surprised than we would have been if we had not looked outside the window before.

As humans, we are constantly making such "plausible" inferences of our beliefs: be it about the weather, the outcomes of our daily decisions, or the behavior of others. *Probabilistic inference* is the process of updating such a prior belief $\mathbb{P}(\overline{W})$ to a posterior belief $\mathbb{P}(\overline{W} \mid \overline{R})$ upon observing $\overline{R}$ where — to reduce clutter — we write W for "The ground is wet" and R for "It is raining".

The central principle of probabilistic inference is Bayes' rule:

Theorem 1.18 (Bayes' rule)**.** *Given random vectors* $\mathbf{X}$ *in* $\mathbb{R}^n$ *and* $\mathbf{Y}$ *in* $\mathbb{R}^m$*, we have for any* $\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^m$ *that*

$$p(\boldsymbol{x} \mid \boldsymbol{y}) = \frac{p(\boldsymbol{y} \mid \boldsymbol{x}) \cdot p(\boldsymbol{x})}{p(\boldsymbol{y})}. \tag{1.45}$$

Proof. Bayes' rule is a direct consequence of the definition of conditional densities (1.10) and the product rule (1.11). □

Let us consider the meaning of each term separately:

- the *prior* $p(\boldsymbol{x})$ is the initial belief about $\boldsymbol{x}$,
- the *(conditional) likelihood* $p(\boldsymbol{y} \mid \boldsymbol{x})$ describes how likely the observations $\boldsymbol{y}$ are under a given value $\boldsymbol{x}$,

- the *posterior* $p(\boldsymbol{x} \mid \boldsymbol{y})$ is the updated belief about $\boldsymbol{x}$ after observing $\boldsymbol{y}$,
- the *joint likelihood* $p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{y} \mid \boldsymbol{x})p(\boldsymbol{x})$ combines prior and likelihood,
- the *marginal likelihood* $p(\boldsymbol{y})$ describes how likely the observations $\boldsymbol{y}$ are across all values of $\boldsymbol{x}$.

The marginal likelihood can be computed using the sum rule (1.7) or the law of total probability (1.12),

$$p(\boldsymbol{y}) = \int_{\boldsymbol{X}(\Omega)} p(\boldsymbol{y} \mid \boldsymbol{x}) \cdot p(\boldsymbol{x}) \, d\boldsymbol{x}. \tag{1.46}$$

Note, however, that the marginal likelihood is simply normalizing the conditional distribution to integrate to one, and therefore a constant with respect to $\boldsymbol{x}$. For this reason, $p(\boldsymbol{y})$ is commonly called the *normalizing constant*.

Example 1.19: Plausible inferences

Let us confirm our intuition from the above example. The logical implication "If it is raining, the ground is wet." (denoted $R \to W$) can be succinctly expressed as $\mathbb{P}(W \mid R) = 1$. Since $\mathbb{P}(W) \leq 1$, we know that

$$\mathbb{P}(R \mid W) = \frac{\mathbb{P}(W \mid R) \cdot \mathbb{P}(R)}{\mathbb{P}(W)} = \frac{\mathbb{P}(R)}{\mathbb{P}(W)} \geq \mathbb{P}(R).$$

That is, observing that the ground is wet makes it more likely to be raining. From $\mathbb{P}(R \mid W) \geq \mathbb{P}(R)$ we know $\mathbb{P}\big(\overline{R} \mid W\big) \leq \mathbb{P}\big(\overline{R}\big)$,[14] which leads us to follow that

[14] since $\mathbb{P}(\overline{X}) = 1 - \mathbb{P}(X)$

$$\mathbb{P}\big(W \mid \overline{R}\big) = \frac{\mathbb{P}\big(\overline{R} \mid W\big) \cdot \mathbb{P}(W)}{\mathbb{P}\big(\overline{R}\big)} \leq \mathbb{P}(W),$$

that is, having observed it not to be raining made the ground less likely to be wet.

Example 1.19 is called a *plausible inference* because the observation of $\overline{R}$ does not completely determine the truth value of $\overline{W}$, and hence, does not permit logical inference. In the case, however, that logical inference is permitted, it coincides with probabilistic inference.

Example 1.20: Logical inferences

For example, if we were to observe that the ground is not wet, then logical inference implies that it must not be raining: $\overline{W} \to \overline{R}$. This is called the *contrapositive* of $R \to W$.

Indeed, by probabilistic inference, we obtain analogously

$$\mathbb{P}\big(R \mid \overline{W}\big) = \frac{\mathbb{P}\big(\overline{W} \mid R\big) \cdot \mathbb{P}(R)}{\mathbb{P}\big(\overline{W}\big)} = \frac{(1 - \mathbb{P}(W \mid R)) \cdot \mathbb{P}(R)}{\mathbb{P}\big(\overline{W}\big)} = 0. \qquad \text{as } \mathbb{P}(W \mid R) = 1$$

Observe that a logical inference does not depend on the prior $\mathbb{P}(R)$: Even if the prior was $\mathbb{P}(R) = 1$ in Example 1.20, after observing that the ground is not wet, we are forced to conclude that it is not raining to maintain logical consistency. The examples highlight that while *logical* inference does not require the notion of a prior, plausible (*probabilistic*!) inference does.

1.2.1 *Where do priors come from?*

Bayes' rule necessitates the specification of a prior $p(\boldsymbol{x})$. Different priors can lead to the deduction of dramatically different posteriors, as one can easily see by considering the extreme cases of a prior that is a point density at $\boldsymbol{x} = \boldsymbol{x}_0$ and a prior that is "uniform" over $\mathbb{R}^n$.[15] In the former case, the posterior will be a point density at $\boldsymbol{x}_0$ regardless of the likelihood. In other words, no evidence can alter the "prior belief" the learner ascribed to $\boldsymbol{x}$. In the latter case, the learner has "no prior belief", and therefore the posterior will be proportional to the likelihood. Both steps of probabilistic inference are perfectly valid, though one might debate which prior is more reasonable.

[15] The latter is not a valid probability distribution, but we can still derive meaning from the posterior as we discuss in Remark 1.22.

Someone who follows the Bayesian interpretation of probability might argue that everything is conditional, meaning that the prior is simply a posterior of all former observations. While this might seem natural ("my world view from today is the combination of my world view from yesterday and the observations I made today"), this lacks an explanation for "the first day". Someone else who is more inclined towards the frequentist interpretation might also object to the existence of a prior belief altogether, arguing that a prior is *subjective* and therefore not a valid or desirable input to a learning algorithm. Put differently, a frequentist "has the belief not to have any belief". This is perfectly compatible with probabilistic inference, as long as the prior is chosen to be *noninformative*:

$$p(\boldsymbol{x}) \propto \text{const.} \tag{1.47}$$

Choosing a noninformative prior in the absence of any evidence is known as the *principle of indifference* or the *principle of insufficient reason*, which dates back to the famous mathematician Pierre-Simon Laplace.

Example 1.21: Why be indifferent?

Consider a criminal trial with three suspects, A, B, and C. The collected evidence shows that suspect C can not have committed the crime, however it does not yield any information about suspects A and B. Clearly, any distribution respecting the data must assign zero probability of having committed the crime to suspect C. However, any distribution interpolating between $(1, 0, 0)$ and $(0, 1, 0)$ respects the data. The principle of indifference suggests that the desired distribution is $(\frac{1}{2}, \frac{1}{2}, 0)$, and indeed, any alternative distribution seems unreasonable.

Remark 1.22: Noninformative and improper priors

It is not necessarily required that the prior $p(\boldsymbol{x})$ is a valid distribution (i.e., integrates to 1). Consider for example, the noninformative prior $p(\boldsymbol{x}) \propto \mathbb{1}\{\boldsymbol{x} \in I\}$ where $I \subseteq \mathbb{R}^n$ is an infinitely large interval. Such a prior which is not a valid distribution is called an *improper prior*. We can still derive meaning from the posterior of a given likelihood and (improper) prior as long as the posterior is a valid distribution.

Laplace's principle of indifference can be generalized to cases where *some* evidence is available. The *maximum entropy principle*, originally proposed by Jaynes (1968), states that one should choose as prior from all possible distributions that are *consistent* with prior knowledge, the one that makes the *least* "additional assumptions", i.e., is the least "informative". In philosophy, this principle is known as *Occam's razor* or the *principle of parsimony*. The "informativeness" of a distribution p is quantified by its *entropy* which is defined as

$$\mathrm{H}[p] \doteq \mathbb{E}_{\boldsymbol{x} \sim p}[-\log p(\boldsymbol{x})]. \tag{1.48}$$

The more concentrated p is, the less is its entropy; the more diffuse p is, the greater is its entropy.[16]

In the absence of any prior knowledge, the uniform distribution has the highest entropy,[17] and hence, the maximum entropy principle suggests a noninformative prior (as does Laplace's principle of indifference). In contrast, if the evidence perfectly determines the value of $\boldsymbol{x}$, then the only consistent explanation is the point density at $\boldsymbol{x}$. The maximum entropy principle characterizes a reasonable choice of prior for these two extreme cases and all cases in between. Bayes' rule can in fact be derived as a consequence of the maximum entropy principle in the sense that the posterior is the least "informative" distribution among all distributions that are consistent with the prior and the ob-

[16] We give a thorough introduction to entropy in Section 5.4.

[17] This only holds true when the set of possible outcomes of $\boldsymbol{x}$ finite (or a bounded continuous interval), as in this case, the noninformative prior is a proper distribution — the uniform distribution. In the "infinite case", there is no uniform distribution and the noninformative prior can be attained from the maximum entropy principle as the limiting solution as the number of possible outcomes of $\boldsymbol{x}$ is increased.

servations (?).

Problem 5.7

1.2.2 *Conjugate Priors*

If the prior $p(\boldsymbol{x})$ and posterior $p(\boldsymbol{x} \mid \boldsymbol{y})$ are of the same family of distributions, the prior is called a *conjugate prior* to the likelihood $p(\boldsymbol{y} \mid \boldsymbol{x})$. This is a very desirable property, as it allows us to recursively apply the same learning algorithm implementing probabilistic inference. We will see in Chapter 2 that under some conditions the Gaussian is *self-conjugate*. That is, if we have a Gaussian prior and a Gaussian likelihood then our posterior will also be Gaussian. This will provide us with the first *efficient* implementation of probabilistic inference.

Example 1.23: Conjugacy of beta and binomial distribution

As an example for conjugacy, we will show that the beta distribution is a conjugate prior to a binomial likelihood. Recall the PMF of the binomial distribution

$$\text{Bin}(k; n, \theta) = \binom{n}{k} \theta^k (1-\theta)^{n-k} \tag{1.49}$$

and the PDF of the beta distribution,

$$\text{Beta}(\theta; \alpha, \beta) \propto \theta^{\alpha-1}(1-\theta)^{\beta-1}, \tag{1.50}$$

We assume the prior $\theta \sim \text{Beta}(\alpha, \beta)$ and likelihood $k \mid \theta \sim \text{Bin}(n, \theta)$. Let $n_H = k$ be the number of heads and $n_T = n - k$ the number of tails in the binomial trial k. Then,

$$\begin{aligned} p(\theta \mid k) &\propto p(k \mid \theta) p(\theta) && \text{using Bayes' rule (1.45)} \\ &\propto \theta^{n_H}(1-\theta)^{n_T} \theta^{\alpha-1}(1-\theta)^{\beta-1} \\ &= \theta^{\alpha+n_H-1}(1-\theta)^{\beta+n_T-1}. \end{aligned}$$

Thus, $\theta \mid k \sim \text{Beta}(\alpha + n_H, \beta + n_T)$.

This same conjugacy can be shown for the multivariate generalization of the beta distribution, the *Dirichlet distribution*, and the multivariate generalization of the binomial distribution, the *multinomial distribution*.

1.2.3 *Tractable Inference with the Normal Distribution*

Using arbitrary distributions for learning and inference is computationally very expensive when the number of dimensions is large — even in the discrete setting. For example, computing marginal distributions using the sum rule yields an exponentially long sum in the

size of the random vector. Similarly, the normalizing constant of the conditional distribution is a sum of exponential length. Even to represent any discrete joint probability distribution requires space that is exponential in the number of dimensions (cf. Figure 1.5).

X_1	$\cdots$	X_{n-1}	X_n	$\mathbb{P}(X_{1:n})$
0	$\cdots$	0	0	0.01
0	$\cdots$	0	1	0.001
0	$\cdots$	1	0	0.213
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
1	$\cdots$	1	1	0.0003

Figure 1.5: A table representing a joint distribution of n binary random variables. The table has 2^n rows. The number of parameters is $2^n - 1$ since the final probability is determined by all other probabilities as they must sum to one.

One strategy to get around this computational blowup is to restrict the class of distributions. Gaussians are a popular choice for this purpose since they have extremely useful properties: they have a compact representation and — as we will see in Chapter 2 — they allow for closed-form probabilistic inference.

In Equation (1.5), we have already seen the PDF of the univariate Gaussian distribution. A random vector $\mathbf{X}$ in $\mathbb{R}^n$ is *normally distributed*, $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, if its PDF is

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \doteq \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{1.51}$$

where $\boldsymbol{\mu} \in \mathbb{R}^n$ is the mean vector and $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$ the covariance matrix (?). We call $\boldsymbol{\Lambda} \doteq \boldsymbol{\Sigma}^{-1}$ the *precision matrix*. $\mathbf{X}$ is also called a *Gaussian random vector* (GRV). $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is the multivariate *standard normal distribution*. We call a Gaussian *isotropic* if its covariance matrix is of the form $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}$ for some $\sigma^2 \in \mathbb{R}$. In this case, the sublevel sets of the PDF are perfect spheres as can be seen in Figure 1.6.

Problem 1.11

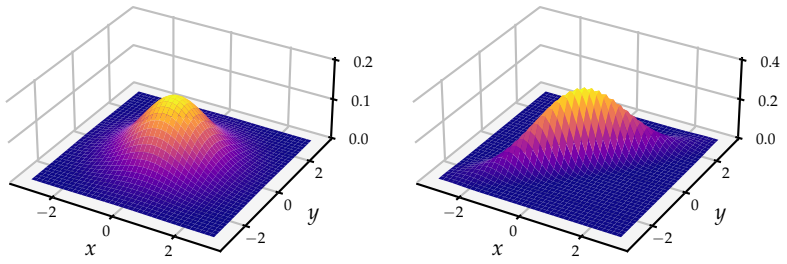


Figure 1.6: Shown are the PDFs of two-dimensional Gaussians with mean $\mathbf{0}$ and covariance matrices

$$\boldsymbol{\Sigma}_1 \doteq \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \boldsymbol{\Sigma}_2 \doteq \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}$$

respectively.

Note that a Gaussian can be represented using only $O(n^2)$ parameters. In the case of a diagonal covariance matrix, which corresponds to n independent univariate Gaussians (?), we just need $O(n)$ parameters.

Problem 1.8

In Equation (1.51), we assume that the covariance matrix $\boldsymbol{\Sigma}$ is invertible, i.e., does not have the eigenvalue 0. This is not a restriction since it can be shown that a covariance matrix has a zero eigenvalue if and only if there exists a deterministic linear relationship between some variables in the joint distribution (?). As we have already seen that a

Problem 1.6

covariance matrix does not have negative eigenvalues (?), this ensures that $\boldsymbol{\Sigma}$ and $\boldsymbol{\Lambda}$ are positive definite.[18]

Problem 1.4

[18] The inverse of a positive definite matrix is also positive definite.

An important property of the normal distribution is that it is closed under marginalization and conditioning.

Theorem 1.24 (Marginal and conditional distribution). (?) *Consider the Gaussian random vector* $\mathbf{X}$ *and fix index sets* $A \subseteq [n]$ *and* $B \subseteq [n]$. *Then, we have that for any such* marginal distribution,

$$\mathbf{X}_A \sim \mathcal{N}(\boldsymbol{\mu}_A, \boldsymbol{\Sigma}_{AA}), \tag{1.52}$$

Problem 1.9

By $\boldsymbol{\mu}_A$ we denote $[\mu_{i_1}, \dots, \mu_{i_k}]$ where $A = \{i_1, \dots i_k\}$. $\boldsymbol{\Sigma}_{AA}$ is defined analogously.

and that for any such conditional distribution,

$$\mathbf{X}_A \mid \mathbf{X}_B = \boldsymbol{x}_B \sim \mathcal{N}(\boldsymbol{\mu}_{A|B}, \boldsymbol{\Sigma}_{A|B}) \quad \textit{where} \tag{1.53a}$$

$$\boldsymbol{\mu}_{A|B} \doteq \boldsymbol{\mu}_A + \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}(\boldsymbol{x}_B - \boldsymbol{\mu}_B), \tag{1.53b}$$

Here, $\boldsymbol{\mu}_A$ characterizes the prior belief and $\boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}(\boldsymbol{x}_B - \boldsymbol{\mu}_B)$ represents "how different" $\boldsymbol{x}_B$ is from what was expected.

$$\boldsymbol{\Sigma}_{A|B} \doteq \boldsymbol{\Sigma}_{AA} - \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}\boldsymbol{\Sigma}_{BA}. \tag{1.53c}$$

Theorem 1.24 provides a closed-form characterization of probabilistic inference for the case that random variables are jointly Gaussian. We will discuss in Chapter 2, how this can be turned into an efficient inference algorithm.

Observe that upon inference, the variance can only shrink! Moreover, how much the variance is reduced depends purely on *where* the observations are made (i.e., the choice of B) but not on *what* the observations are. In contrast, the posterior mean $\boldsymbol{\mu}_{A|B}$ depends affinely on $\boldsymbol{\mu}_B$. These are special properties of the Gaussian and do not generally hold true for other distributions.

It can be shown that Gaussians are additive and closed under affine transformations (?). The closedness under affine transformations (1.78) implies that a Gaussian $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is equivalently characterized as

Problem 1.10

$$\mathbf{X} = \boldsymbol{\Sigma}^{1/2}\mathbf{Y} + \boldsymbol{\mu}. \tag{1.54}$$

where $\mathbf{Y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and $\boldsymbol{\Sigma}^{1/2}$ is the square root of $\boldsymbol{\Sigma}$.[19] Importantly, this implies together with Theorem 1.24 and additivity (1.79) that:

[19] More details on the square root of a symmetric and positive definite matrix can be found in Appendix A.2.

Any affine transformation of a Gaussian random vector is a Gaussian random vector.

A consequence of this is that given any jointly Gaussian random vectors $\mathbf{X}_A$ and $\mathbf{X}_B$, $\mathbf{X}_A$ can be expressed as an affine function of $\mathbf{X}_B$ with added independent Gaussian noise. Formally, we define

$$\mathbf{X}_A \doteq \boldsymbol{A}\mathbf{X}_B + \boldsymbol{b} + \boldsymbol{\varepsilon} \quad \text{where} \tag{1.55a}$$

$$\boldsymbol{A} \doteq \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}, \tag{1.55b}$$
$$\boldsymbol{b} \doteq \boldsymbol{\mu}_A - \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}\boldsymbol{\mu}_B, \tag{1.55c}$$
$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{A|B}). \tag{1.55d}$$

It directly follows from the closedness of Gaussians under affine transformations (1.78) that the characterization of $\mathbf{X}_A$ via Equation (1.55) is equivalent to $\mathbf{X}_A \sim \mathcal{N}(\boldsymbol{\mu}_A, \boldsymbol{\Sigma}_{AA})$, and hence, *any* Gaussian $\mathbf{X}_A$ can be modeled as a so-called *conditional linear Gaussian*, i.e., an affine function of another Gaussian $\mathbf{X}_B$ with additional independent Gaussian noise. We will use this fact frequently to represent Gaussians in a compact form.

1.3 Supervised Learning and Point Estimates

Throughout the first part of this manuscript, we will focus mostly on the *supervised learning* problem where we want to learn a function

$$f^\star : \mathcal{X} \to \mathcal{Y}$$

from labeled training data. That is, we are given a collection of labeled examples, $\mathcal{D}_n \doteq \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where the $\boldsymbol{x}_i \in \mathcal{X}$ are *inputs* and the $y_i \in \mathcal{Y}$ are *outputs* (called *labels*), and we want to find a function $\hat{f}$ that best-approximates $f^\star$. It is common to choose $\hat{f}$ from a parameterized *function class* $\mathcal{F}(\Theta)$, where each function $f_{\boldsymbol{\theta}}$ is described by some parameters $\boldsymbol{\theta} \in \Theta$.

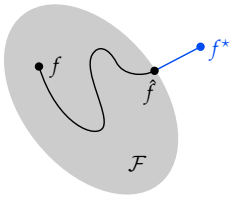


Figure 1.7: Illustration of **estimation error** and **approximation error**. $f^\star$ denotes the true function and $\hat{f}$ is the best approximation from the function class $\mathcal{F}$. We do not specify here, how one could quantify "error". For more details, see Appendix A.3.5.

Remark 1.25: What this manuscript is about and not about

As illustrated in Figure 1.7, the restriction to a function class leads to two sources of error: the *estimation error* of having "incorrectly" determined $\hat{f}$ within the function class, and the *approximation error* of the function class itself. Choosing a "good" function class / architecture with small approximation error is therefore critical for any practical application of machine learning. We will discuss various function classes, from linear models to deep neural networks, however, determining the "right" function class will not be the focus of this manuscript. To keep the exposition simple, we will assume in the following that $f^\star \in \mathcal{F}(\Theta)$ with parameters $\boldsymbol{\theta}^\star \in \Theta$.

Instead, we will focus on the problem of estimation/inference within a given function class. We will see that inference in smaller function classes is often more computationally efficient since the search space is smaller or — in the case of Gaussians — has a known tractable structure. On the other hand, larger function

classes are more expressive and therefore can typically better approximate the ground truth $f^\star$.

We differentiate between the task of *regression* where $\mathcal{Y} \doteq \mathbb{R}^k$,[20] and the task of *classification* where $\mathcal{Y} \doteq \mathcal{C}$ and $\mathcal{C}$ is an m-element set of classes. In other words, regression is the task of predicting a continuous label, whereas classification is the task of predicting a discrete class label. These two tasks are intimately related: in fact, we can think of classification tasks as a regression problem where we learn a probability distribution over class labels. In this regression problem, $\mathcal{Y} \doteq \Delta^{\mathcal{C}}$ where $\Delta^{\mathcal{C}}$ denotes the set of all probability distributions over the set of classes $\mathcal{C}$ which is an $(m-1)$-dimensional convex polytope in the m-dimensional space of probabilities $[0,1]^m$ (cf. Appendix A.1.2).

[20] The labels are usually scalar, so $k = 1$.

For now, let us stick to the regression setting. We will assume that the observations are noisy, that is, $y_i \overset{\text{iid}}{\sim} p(\cdot \mid \boldsymbol{x}_i, \boldsymbol{\theta}^\star)$ for some *known* conditional distribution $p(\cdot \mid \boldsymbol{x}_i, \boldsymbol{\theta})$ but *unknown* parameter $\boldsymbol{\theta}^\star$.[21] Our assumption can equivalently be formulated as

$$y_i = \underbrace{f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)}_{\text{signal}} + \underbrace{\varepsilon_i(\boldsymbol{x}_i)}_{\text{noise}} \tag{1.56}$$

[21] The case where the labels are deterministic is the special case of $p(\cdot \mid \boldsymbol{x}_i, \boldsymbol{\theta}^\star)$ being a point density at $f^\star(\boldsymbol{x}_i)$.

where $f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$ is the mean of $p(\cdot \mid \boldsymbol{x}_i, \boldsymbol{\theta})$ and $\varepsilon_i(\boldsymbol{x}_i) = y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$ is some independent zero-mean noise, for example (but not necessarily) Gaussian.[22] When the noise distribution may depend on $\boldsymbol{x}_i$, the noise is said to be *heteroscedastic* and otherwise the noise is called *homoscedastic*.

[22] It is crucial that the assumed noise distribution accurately reflects the noise of the data. For example, using a (light-tailed) Gaussian noise model in the presence of heavy-tailed noise will fail! We discuss the distinction between light and heavy tails in Appendix A.3.2.

1.3.1 Maximum Likelihood Estimation

A common approach to finding $\hat{f}$ is to select the model $f \in \mathcal{F}(\Theta)$ under which the training data is most likely. This is called the *maximum likelihood estimate* (or MLE):

$$\begin{aligned} \hat{\boldsymbol{\theta}}_{\text{MLE}} &\doteq \arg\max_{\boldsymbol{\theta} \in \Theta} p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}) \qquad (1.57) \\ &= \arg\max_{\boldsymbol{\theta} \in \Theta} \prod_{i=1}^{n} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}). \end{aligned}$$

using the independence of the training data (1.56)

Such products of probabilities are often numerically unstable, which is why one typically takes the logarithm:

$$= \arg\max_{\boldsymbol{\theta} \in \Theta} \underbrace{\sum_{i=1}^{n} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta})}_{\textit{log-likelihood}}. \tag{1.58}$$

We will denote the *negative log-likelihood* by $\ell_{\text{nll}}(\boldsymbol{\theta}; \mathcal{D}_n)$.

The MLE is often used in practice due to its desirable asymptotic properties as the sample size n increases. We give a brief summary here and provide additional background and definitions in Appendix A.3. To give any guarantees on the convergence of the MLE, we necessarily need to assume that $\boldsymbol{\theta}^\star$ is identifiable.[23] If additionally, ℓ_{nll} is "well-behaved" then standard results say that the MLE is *consistent* and *asymptotically normal* (Van der Vaart, 2000):

[23] That is, $\boldsymbol{\theta}^\star \neq \boldsymbol{\theta} \implies f^\star \neq f_{\boldsymbol{\theta}}$ for any $\boldsymbol{\theta} \in \Theta$. In words, there is no other parameter $\boldsymbol{\theta}$ that yields the same function $f_{\boldsymbol{\theta}}$ as $\boldsymbol{\theta}^\star$.

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} \xrightarrow{\mathbb{P}} \boldsymbol{\theta}^\star \quad \text{and} \quad \hat{\boldsymbol{\theta}}_{\text{MLE}} \xrightarrow{\mathcal{D}} \mathcal{N}(\boldsymbol{\theta}^\star, \boldsymbol{S}_n) \quad \text{as } n \to \infty. \tag{1.59}$$

Here, we denote by $\boldsymbol{S}_n$ the asymptotic variance of the MLE which can be understood as measuring the "quality" of the estimate.[24] This implies in some sense that the MLE is asymptotically unbiased. Moreover, the MLE can be shown to be *asymptotically efficient* which is to say that there exists no other consistent estimator with a "smaller" asymptotic variance.[25]

[24] A "smaller" variance means that we can be more confident that the MLE is close to the true parameter.

[25] see Appendix A.3.4.

The situation is quite different in the finite sample regime. Here, the MLE need not be unbiased, and it is susceptible to *overfitting* to the (finite) training data as we discuss in more detail in Appendix A.3.5.

1.3.2 *Using Priors: Maximum a Posteriori Estimation*

We can incorporate prior assumptions about the parameters $\boldsymbol{\theta}^\star$ into the estimation procedure. One approach of this kind is to find the mode of the posterior distribution, called the *maximum a posteriori estimate* (or MAP estimate):

$$\begin{aligned}
\hat{\boldsymbol{\theta}}_{\text{MAP}} &\doteq \arg\max_{\boldsymbol{\theta}\in\Theta} p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}) && (1.60)\\
&= \arg\max_{\boldsymbol{\theta}\in\Theta} p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}) \cdot p(\boldsymbol{\theta}) && (1.61) && \text{by Bayes' rule (1.45)}\\
&= \arg\max_{\boldsymbol{\theta}\in\Theta} \log p(\boldsymbol{\theta}) + \sum_{i=1}^{n} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) && (1.62) && \text{taking the logarithm}\\
&= \arg\min_{\boldsymbol{\theta}\in\Theta} \underbrace{-\log p(\boldsymbol{\theta})}_{\text{regularization}} + \underbrace{\ell_{\text{nll}}(\boldsymbol{\theta}; \mathcal{D}_n)}_{\text{quality of fit}}. && (1.63)
\end{aligned}$$

Here, the *log-prior* $\log p(\boldsymbol{\theta})$ acts as a regularizer. Common regularizers are given, for example, by

- $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \mathbf{0}, (2\lambda)^{-1}\boldsymbol{I})$ which yields $-\log p(\boldsymbol{\theta}) = \lambda \|\boldsymbol{\theta}\|_2^2 + \text{const}$,
- $p(\boldsymbol{\theta}) = \text{Laplace}(\boldsymbol{\theta}; \mathbf{0}, \lambda^{-1})$ which yields $-\log p(\boldsymbol{\theta}) = \lambda \|\boldsymbol{\theta}\|_1 + \text{const}$,
- a uniform prior (cf. Section 1.2.1) for which the MAP is equivalent to the MLE. In other words, the MLE is merely the mode of the posterior distribution under a uniform prior.

The Gaussian and Laplace regularizers act as simplicity biases, preferring simpler models over more complex ones, which empirically

tends to reduce the risk of overfitting. However, one may also encode more nuanced information about the (assumed) structure of $\boldsymbol{\theta}^\star$ into the prior.

An alternative way of encoding a prior is by restricting the function class to some $\widetilde{\Theta} \subset \Theta$, for example to rotation- and translation-invariant models as often done when the inputs are images. This effectively sets $p(\boldsymbol{\theta}) = 0$ for all $\boldsymbol{\theta} \in \Theta \setminus \widetilde{\Theta}$ but is better suited for numerical optimization than to impose this constraint directly on the prior.

Encoding prior assumptions into the function class or into the parameter estimation can accelerate learning and improve generalization performance dramatically, yet importantly, incorporating a prior can also inhibit learning in case the prior is "wrong". For example, when the learning task is to differentiate images of cats from images of dogs, consider the (stupid) prior that only permits models that exclusively use the upper-left pixel for prediction. No such model will be able to solve the task, and therefore starting from this prior makes the learning problem effectively unsolvable which illustrates that priors have to be chosen with care.

1.3.3 When does the prior matter?

We have seen that the MLE has desirable asymptotic properties, and that MAP estimation can be seen as a regularized MLE where the type of regularization is encoded by the prior. Is it possible to derive similar asymptotic results for the MAP estimate?

To answer this question, we will look at the asymptotic effect of the prior on the posterior more generally. *Doob's consistency theorem* states that assuming parameters are identifiable,[26] there exists $\widetilde{\Theta} \subseteq \Theta$ with $p(\widetilde{\Theta}) = 1$ such that the posterior is consistent for any $\boldsymbol{\theta}^\star \in \widetilde{\Theta}$ (Doob, 1949; Miller, 2016, 2018):

$$\boldsymbol{\theta} \mid \mathcal{D}_n \xrightarrow{\mathbb{P}} \boldsymbol{\theta}^\star \qquad \text{as } n \to \infty. \tag{1.64}$$

In words, Doob's consistency theorem tells us that for *any* prior distribution, the posterior is guaranteed to converge to a point density in the (small) neighborhood $\boldsymbol{\theta}^\star \in B$ of the true parameter as long as $p(B) > 0$.[27] We call such a prior a *well-specified prior*.

[26] This is akin to the assumption required for consistency of the MLE, cf. Equation (1.59).

[27] B can for example be a ball of radius ϵ around $\boldsymbol{\theta}^\star$ (with respect to some geometry of Θ).

Remark 1.26: Cromwell's rule
In the case where $|\Theta|$ is finite, Doob's consistency theorem strongly suggests that the prior should not assign 0 probability (or probability 1 for that matter) to any individual parameter $\boldsymbol{\theta} \in \Theta$, unless we know with certainty that $\boldsymbol{\theta}^\star \neq \boldsymbol{\theta}$. This is called *Cromwell's rule*,

and a prior obeying by this rule is always well-specified.

Under the same assumption that the prior is well-specified (and regularity conditions[28]), the *Bernstein-von Mises theorem*, which was first discovered by Pierre-Simon Laplace in the early 19th century, establishes the asymptotic normality of the posterior distribution (Van der Vaart, 2000; Miller, 2016):

$$\boldsymbol{\theta} \mid \mathcal{D}_n \xrightarrow{\mathcal{D}} \mathcal{N}(\boldsymbol{\theta}^\star, \boldsymbol{S}_n) \qquad \text{as } n \to \infty \tag{1.65}$$

and where $\boldsymbol{S}_n$ is the same as the asymptotic variance of the MLE.[29]

These results link probabilistic inference to maximum likelihood estimation in the asymptotic limit of infinite data. Intuitively, in the limit of infinite data, the prior is "overwhelmed" by the observations and the posterior becomes equivalent to the limiting distribution of the MLE.[30] One can interpret the regime of infinite data as the regime where computational resources and time are unlimited and plausible inferences evolve into logical inferences. This transition signifies a shift from the realm of uncertainty to that of certainty. The importance of the prior surfaces precisely in the non-asymptotic regime where plausible inferences are necessary due to *limited computational resources and limited time*.

[28] These regularity conditions are akin to the assumptions required for asymptotic normality of the MLE, cf. Equation (1.59).

[29] This has also been called the "Bayesian central limit theorem", which is a bit of a misnomer since the theorem also applies to likelihoods (when the prior is noninformative) which are often used in frequentist statistics.

[30] More examples and discussion can be found in section 17.8 of Le Cam (1986), chapter 8 of Le Cam and Yang (2000), chapter 10 of Van der Vaart (2000), and in Tanner (1991).

1.3.4 Estimation vs Inference

You can interpret a single parameter vector $\boldsymbol{\theta} \in \Theta$ as "one possible explanation" of the data. Maximum likelihood and maximum a posteriori estimation are examples of *estimation* algorithms which return a one such parameter vector — called a *point estimate*. That is, given the training set $\mathcal{D}_n$, they return a single parameter vector $\hat{\boldsymbol{\theta}}_n$. We give a more detailed account of estimation in Appendix A.3.

Example 1.27: Point estimates and invalid logical inferences

To see why point estimates can be problematic, recall Example 1.19. We have seen that the logical implication

"If it is raining, the ground is wet."

can be expressed as $\mathbb{P}(W \mid R) = 1$. Observing that "The ground is wet." does not permit logical inference, yet, the maximum likelihood estimate of R is $\hat{R}_{\text{MLE}} = 1$. This is logically inconsistent since there might be other *explanations* for the ground to be wet, such as a sprinkler! With only a finite sample (say independently observing n times that the ground is wet), we cannot rule out with

certainty that the ground is wet for other reasons than rain.

In practice, we never observe an infinite amount of data. Example 1.27 demonstrates that on a finite sample, point estimates may perform invalid logical inferences, and can therefore lure us into a false sense of certainty.

Remark 1.28: MLE and MAP are approximations of inference

The MLE and MAP estimate can be seen as a naive approximation of probabilistic inference, represented by a point density which "collapses" all probability mass at the mode of the posterior distribution. This can be a relatively decent — even if overly simple — approximation when the distribution is unimodal, symmetric, and light-tailed as in Figure 1.8, but is usually a very poor approximation for practical posteriors that are complex and multimodal.

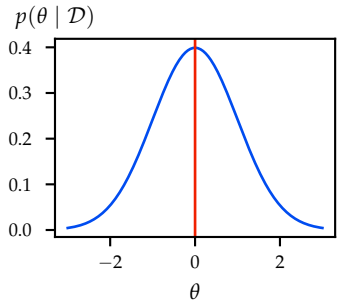


Figure 1.8: A the MLE/MAP are point estimates at the mode $\hat{\theta}$ of the posterior distribution $p(\theta \mid \mathcal{D})$.

In this manuscript, we will focus mainly on algorithms for *probabilistic inference* which compute or approximate the distribution $p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n})$ over parameters. Returning a distribution over parameters is natural since this acknowledges that given a finite sample with noisy observations, more than one parameter vector can explain the data.

1.3.5 Probabilistic Inference and Prediction

The prior distribution $p(\boldsymbol{\theta})$ can be interpreted as the degree of our belief that the model parameterized by $\boldsymbol{\theta}$ "describes the (previously seen) data best". The likelihood captures how likely the training data is under a particular model:

$$p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}) = \prod_{i=1}^{n} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}). \tag{1.66}$$

The posterior then represents our belief about the best model after seeing the training data. Using Bayes' rule (1.45), we can write it as[31]

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}) = \frac{1}{Z} p(\boldsymbol{\theta}) \prod_{i=1}^{n} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) \quad \text{where} \tag{1.67a}$$

$$Z \doteq \int_{\Theta} p(\boldsymbol{\theta}) \prod_{i=1}^{n} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) \, d\boldsymbol{\theta} \tag{1.67b}$$

is the *normalizing constant*. We refer to this process of learning a model from data as *learning*. We can then use our learned model for *prediction* at a new input $\boldsymbol{x}^\star$ by conditioning on $\boldsymbol{\theta}$,

$$p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}) = \int_{\Theta} p(y^\star, \boldsymbol{\theta} \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}) \, d\boldsymbol{\theta}$$

by the sum rule (1.7)

[31] We generally assume that

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}) = p(\boldsymbol{\theta}).$$

For our purposes, you can think of the inputs $\boldsymbol{x}_{1:n}$ as fixed deterministic parameters, but one can also consider inputs drawn from a distribution over $\mathcal{X}$.

$$= \int_\Theta p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}) \cdot p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}) \, d\boldsymbol{\theta}. \tag{1.68}$$

by the product rule (1.11) and $y^\star \perp \boldsymbol{x}_{1:n}, y_{1:n} \mid \boldsymbol{\theta}$

Here, the distribution over models $p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n})$ is called the *posterior* and the distribution over predictions $p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n})$ is called the *predictive posterior*. The predictive posterior quantifies our posterior uncertainty about the "prediction" $y^\star$, however, since this is typically a complex distribution, it is difficult to communicate this uncertainty to a human. One statistic that can be used for this purpose is the smallest set $\mathcal{C}_\delta(\boldsymbol{x}^\star) \subseteq \mathbb{R}$ for a fixed $\delta \in (0, 1)$ such that

$$\mathbb{P}(y^\star \in \mathcal{C}_\delta(\boldsymbol{x}^\star) \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}) \geq 1 - \delta. \tag{1.69}$$

That is, we believe with "confidence" at least $1 - \delta$ that the true value of $y^\star$ lies in $\mathcal{C}_\delta(\boldsymbol{x}^\star)$. Such a set $\mathcal{C}_\delta(\boldsymbol{x}^\star)$ is called a *credible set*.

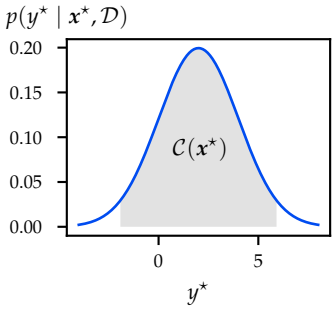


Figure 1.9: Example of a 95% credible set at $\boldsymbol{x}^\star$ where the predictive posterior is Gaussian with mean $\mu(\boldsymbol{x}^\star)$ and standard deviation $\sigma(\boldsymbol{x}^\star)$. In this case, the gray area integrates to ≈ 0.95 for

$$\mathcal{C}_{0.05}(\boldsymbol{x}^\star) = [\mu(\boldsymbol{x}^\star) \pm 1.96\sigma(\boldsymbol{x}^\star)].$$

We have seen here that the tasks of learning and prediction are intimately related. Indeed, "prediction" can be seen in many ways as a natural by-product of "reasoning" (i.e., probabilistic inference), where we evaluate the likelihood of outcomes given our learned explanations for the world. This intuition can be read off directly from Equation (1.68) where $p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta})$ corresponds to the likelihood of an outcome given the explanation $\boldsymbol{\theta}$ and $p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n})$ corresponds to our inferred belief about the world. We will see many more examples of this link between probabilistic inference and prediction throughout this manuscript.

The high-dimensional integrals of Equations (1.67b) and (1.68) are typically intractable, and represent the main computational challenge in probabilistic inference. Throughout the first part of this manuscript, we will describe settings where exact inference is tractable, as well as modern approximate inference algorithms that can be used when exact inference is intractable.

1.3.6 *Recursive Probabilistic Inference and Memory*

We have already alluded to the fact that probabilistic inference has a recursive structure, which lends itself to continual learning and which often leads to efficient algorithms. Let us denote by

$$p^{(t)}(\boldsymbol{\theta}) \doteq p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:t}, y_{1:t}) \tag{1.70}$$

the posterior after the first t observations with $p^{(0)}(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$. Now, suppose that we have already computed $p^{(t)}(\boldsymbol{\theta})$ and observe y_{t+1}. We can recursively update the posterior as follows,

$$\begin{aligned} p^{(t+1)}(\boldsymbol{\theta}) &= p(\boldsymbol{\theta} \mid y_{1:t+1}) \\ &\propto p(\boldsymbol{\theta} \mid y_{1:t}) \cdot p(y_{t+1} \mid \boldsymbol{\theta}, y_{1:t}) \end{aligned}$$

using Bayes' rule (1.45)

$$= p^{(t)}(\boldsymbol{\theta}) \cdot p(y_{t+1} \mid \boldsymbol{\theta}). \tag{1.71}$$

using $y_{t+1} \perp y_{1:t} \mid \boldsymbol{\theta}$, see Figure 2.3

Intuitively, the posterior distribution at time t "absorbs" or "summarizes" all seen data.

By unrolling the recursion of Equation (1.71), we see that regardless of the philosophical interpretation of probability, probabilistic inference is a fundamental mechanism of learning. Even the MLE which performs naive approximate inference without a prior (i.e., a uniform prior), is based on $p^{(n)}(\boldsymbol{\theta}) \propto p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})$ which is the result of n individual plausible inferences, where the $(t+1)$-st inference uses the posterior of the t-th inference as its prior.

So far we have been considering the supervised learning setting, where all data is available a-priori. However, by sequentially obtaining the new posterior and replacing our prior, we can also perform probabilistic inference as data arrives online (i.e., in "real-time"). This is analogous to recursive logical inference where derived consequences are repeatedly added to the set of propositions to derive new consequences. This also highlights the intimate connection between "reasoning" and "memory". Indeed, the posterior distribution $p^{(t)}(\boldsymbol{\theta})$ can be seen as a form of memory that evolves with time t.

1.4 *Outlook: Decision Theory*

How can we use our predictions to make concrete decisions under uncertainty? We will study this question extensively in Part II of this manuscript, but briefly introduce some fundamental concepts here. Making decisions using a probabilistic model $p(y \mid \boldsymbol{x})$ of output $y \in \mathcal{Y}$ given input $\boldsymbol{x} \in \mathcal{X}$, such as the ones we have discussed in the previous section, is commonly formalized by

- a set of possible actions $\mathcal{A}$, and
- a reward function $r(y, a) \in \mathbb{R}$ that computes the reward or utility of taking action $a \in \mathcal{A}$, assuming the true output is $y \in \mathcal{Y}$.

Standard decision theory recommends picking the action with the largest expected utility:

$$a^\star(\boldsymbol{x}) \doteq \arg\max_{a \in \mathcal{A}} \mathbb{E}_{y|\boldsymbol{x}}[r(y, a)]. \tag{1.72}$$

Here, $a^\star$ is called the *optimal decision rule* because, under the given probabilistic model, no other rule can yield a higher expected utility.

Let us consider some examples of reward functions and their corresponding optimal decisions:

Example 1.29: Reward functions

Under the decision rule from Equation (1.72), different reward functions r can lead to different decisions. Let us examine two reward functions for the case where $\mathcal{Y} = \mathcal{A} = \mathbb{R}$ (?).

Problem 1.13

- Alternatively to considering r as a reward function, we can interpret $-r$ as the loss of taking action a when the true output is y. If our goal is for our actions a to "mimic" the output y, a natural choice is the squared loss, $-r(y, a) = (y - a)^2$. It turns out that under the squared loss, the optimal decision is simply the mean: $a^\star(\boldsymbol{x}) = \mathbb{E}[y \mid \boldsymbol{x}]$.
- To contrast this, we consider the asymmetric loss,

$$-r(y, a) = c_1 \underbrace{\max\{y - a, 0\}}_{\text{underestimation error}} + c_2 \underbrace{\max\{a - y, 0\}}_{\text{overestimation error}},$$

  which penalizes underestimation and overestimation differently. When $y \mid \boldsymbol{x} \sim \mathcal{N}(\mu_{\boldsymbol{x}}, \sigma_{\boldsymbol{x}}^2)$ then the optimal decision is

$$a^\star(\boldsymbol{x}) = \mu_{\boldsymbol{x}} + \underbrace{\sigma_{\boldsymbol{x}} \cdot \Phi^{-1}\left(\frac{c_1}{c_1 + c_2}\right)}_{\text{pessimism / optimism}}$$

  where Φ is the CDF of the standard normal distribution.[32] Note that if $c_1 = c_2$, then the second term vanishes and the optimal decision is the same as under the squared loss. If $c_1 > c_2$, the second term is positive (i.e., *optimistic*) to avoid underestimation, and if $c_1 < c_2$, the second term is negative (i.e., *pessimistic*) to avoid overestimation. We will find these notions of optimism and pessimism to be useful in many decision-making scenarios.

[32] Recall that the CDF Φ of the standard normal distribution is a sigmoid with its inverse satisfying

$$\Phi^{-1}(u) \begin{cases} < 0 & \text{if } u < 0.5, \\ = 0 & \text{if } u = 0.5, \\ > 0 & \text{if } u > 0.5. \end{cases}$$

While Equation (1.72) describes how to make optimal decisions given a (posterior) probabilistic model, it does not tell us how to learn or improve this model in the first place. That is, these decisions are only optimal under the assumption that we cannot use their outcomes and our resulting observations to update our model and inform future decisions. When we start to consider the effect of our decisions on future data and future posteriors, answering "how do I make optimal decisions?" becomes more complex, and we will study this in Part II on sequential decision-making.

Discussion

In this chapter, we have learned about the fundamental concepts of probabilistic inference. We have seen that probabilistic inference is the

natural extension of logical reasoning to domains with uncertainty. We have also derived the central principle of probabilistic inference, Bayes' rule, which is simple to state but often computationally challenging. In the next part of this manuscript, we will explore settings where exact inference is tractable, as well as modern approaches to approximate probabilistic inference.

Overview of Mathematical Background

We have included brief summaries of the fundamentals of **parameter estimation** (mean estimation in particular) and **optimization** in Appendices A.3 and A.4, respectively, which we will refer back to throughout the manuscript. Appendix A.2 discusses the **correspondence of Gaussians and quadratic forms**. Appendix A.5 comprises a list of useful matrix identities and inequalities.

Problems

1.1. Properties of probability.

Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space. Derive the following properties of probability from the Kolmogorov axioms:

1. For any $A, B \in \mathcal{A}$, if $A \subseteq B$ then $\mathbb{P}(A) \leq \mathbb{P}(B)$.
2. For any $A \in \mathcal{A}$, $\mathbb{P}(\overline{A}) = 1 - \mathbb{P}(A)$.
3. For any countable set of events $\{A_i \in \mathcal{A}\}_i$,

$$\mathbb{P}\left(\bigcup_{i=1}^{\infty} A_i\right) \leq \sum_{i=1}^{\infty} \mathbb{P}(A_i). \tag{1.73}$$

   which is called a *union bound*.

1.2. Random walks on graphs.

Let G be a simple connected finite graph. We start at a vertex u of G. At every step, we move to one of the neighbors of the current vertex uniformly at random, e.g., if the vertex has 3 neighbors, we move to one of them, each with probability $1/3$. What is the probability that the walk visits a given vertex v eventually?

1.3. Law of total expectation.

Show that if $\{A_i\}_{i=1}^k$ are a partition of Ω and $\mathbf{X}$ is a random vector,

$$\mathbb{E}[\mathbf{X}] = \sum_{i=1}^{k} \mathbb{E}[\mathbf{X} \mid A_i] \cdot \mathbb{P}(A_i). \tag{1.74}$$

1.4. Covariance matrices are positive semi-definite.

Prove that a covariance matrix $\boldsymbol{\Sigma}$ is always positive semi-definite. That is, all of its eigenvalues are greater or equal to zero, or equivalently, $\boldsymbol{x}^\top \boldsymbol{\Sigma} \boldsymbol{x} \geq 0$ for any $\boldsymbol{x} \in \mathbb{R}^n$.

1.5. Probabilistic inference.

As a result of a medical screening, one of the tests revealed a serious disease in a person. The test has a high accuracy of 99% (the probability of a positive response in the presence of a disease is 99% and the probability of a negative response in the absence of a disease is also 99%). However, the disease is quite rare and occurs only in one person per 10 000. Calculate the probability of the examined person having the identified disease.

1.6. Zero eigenvalues of covariance matrices.

We say that a random vector $\mathbf{X}$ in $\mathbb{R}^n$ is not linearly independent if for some $\boldsymbol{\alpha} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$, $\boldsymbol{\alpha}^\top \mathbf{X} = 0$.

1. Show that if $\mathbf{X}$ is not linearly independent, then $\mathrm{Var}[\mathbf{X}]$ has a zero eigenvalue.
2. Show that if $\mathrm{Var}[\mathbf{X}]$ has a zero eigenvalue, then $\mathbf{X}$ is not linearly independent.
   Hint: Consider the variance of $\boldsymbol{\lambda}^\top \mathbf{X}$ where $\boldsymbol{\lambda}$ is the eigenvector corresponding to the zero eigenvalue.

Thus, we have shown that $\mathrm{Var}[\mathbf{X}]$ has a zero eigenvalue if and only if $\mathbf{X}$ is not linearly independent.

1.7. Product of Gaussian PDFs.

Let $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \in \mathbb{R}^n$ be mean vectors and $\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2 \in \mathbb{R}^{n \times n}$ be covariance matrices. Prove that

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \propto \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \cdot \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) \tag{1.75}$$

for some mean vector $\boldsymbol{\mu} \in \mathbb{R}^n$ and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$. That is, show that the product of two Gaussian PDFs is proportional to the PDF of a Gaussian.

1.8. Independence of Gaussians.

Show that two jointly Gaussian random vectors, $\mathbf{X}$ and $\mathbf{Y}$, are independent if and only if $\mathbf{X}$ and $\mathbf{Y}$ are uncorrelated.

1.9. Marginal / conditional distribution of a Gaussian.

Prove Theorem 1.24. That is, show that

1. every marginal of a Gaussian is Gaussian; and

2. conditioning on a subset of variables of a joint Gaussian is Gaussian

by finding their corresponding PDFs.

Hint: You may use that for matrices $\boldsymbol{\Sigma}$ and $\boldsymbol{\Lambda}$ such that $\boldsymbol{\Sigma}^{-1} = \boldsymbol{\Lambda}$,

- *if $\boldsymbol{\Sigma}$ and $\boldsymbol{\Lambda}$ are symmetric,*

$$\begin{aligned}
&\begin{bmatrix} \boldsymbol{x}_A \\ \boldsymbol{x}_B \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{\Lambda}_{AA} & \boldsymbol{\Lambda}_{AB} \\ \boldsymbol{\Lambda}_{BA} & \boldsymbol{\Lambda}_{BB} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_A \\ \boldsymbol{x}_B \end{bmatrix} \\
&= \boldsymbol{x}_A^\top \boldsymbol{\Lambda}_{AA} \boldsymbol{x}_A + \boldsymbol{x}_A^\top \boldsymbol{\Lambda}_{AB} \boldsymbol{x}_B + \boldsymbol{x}_B^\top \boldsymbol{\Lambda}_{BA} \boldsymbol{x}_A + \boldsymbol{x}_B^\top \boldsymbol{\Lambda}_{BB} \boldsymbol{x}_B \\
&= \boldsymbol{x}_A^\top (\boldsymbol{\Lambda}_{AA} - \boldsymbol{\Lambda}_{AB} \boldsymbol{\Lambda}_{BB}^{-1} \boldsymbol{\Lambda}_{BA}) \boldsymbol{x}_A + \\
&\qquad (\boldsymbol{x}_B + \boldsymbol{\Lambda}_{BB}^{-1} \boldsymbol{\Lambda}_{BA} \boldsymbol{x}_A)^\top \boldsymbol{\Lambda}_{BB} (\boldsymbol{x}_B + \boldsymbol{\Lambda}_{BB}^{-1} \boldsymbol{\Lambda}_{BA} \boldsymbol{x}_A),
\end{aligned}$$

- $\boldsymbol{\Lambda}_{BB}^{-1} = \boldsymbol{\Sigma}_{BB} - \boldsymbol{\Sigma}_{BA} \boldsymbol{\Sigma}_{AA}^{-1} \boldsymbol{\Sigma}_{AB}$,
- $\boldsymbol{\Lambda}_{BB}^{-1} \boldsymbol{\Lambda}_{BA} = -\boldsymbol{\Sigma}_{BA} \boldsymbol{\Sigma}_{AA}^{-1}$.

The final two equations follow from the general characterization of the inverse of a block matrix (Petersen et al., 2008, section 9.1.3).

1.10. Closedness properties of Gaussians.

Recall the notion of a *moment-generating function* (MGF) of a random vector $\mathbf{X}$ in $\mathbb{R}^n$ which is defined as

$$\varphi_{\mathbf{X}}(\boldsymbol{t}) \doteq \mathbb{E} \exp\left(\boldsymbol{t}^\top \mathbf{X}\right), \quad \text{for all } \boldsymbol{t} \in \mathbb{R}^n. \tag{1.76}$$

An MGF uniquely characterizes a distribution. The MGF of the multivariate Gaussian $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is

$$\varphi_{\mathbf{X}}(\boldsymbol{t}) = \exp\left(\boldsymbol{t}^\top \boldsymbol{\mu} + \frac{1}{2} \boldsymbol{t}^\top \boldsymbol{\Sigma} \boldsymbol{t}\right). \tag{1.77}$$

This generalizes the MGF of the univariate Gaussian from Equation (A.40).

Prove the following facts.

1. *Closedness under affine transformations:* Given an n-dimensional Gaussian $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$,

$$\boldsymbol{A}\mathbf{X} + \boldsymbol{b} \sim \mathcal{N}(\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{b}, \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top). \tag{1.78}$$

2. *Additivity:* Given two independent Gaussian random vectors $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\mathbf{X}' \sim \mathcal{N}(\boldsymbol{\mu}', \boldsymbol{\Sigma}')$ in $\mathbb{R}^n$,

$$\mathbf{X} + \mathbf{X}' \sim \mathcal{N}(\boldsymbol{\mu} + \boldsymbol{\mu}', \boldsymbol{\Sigma} + \boldsymbol{\Sigma}'). \tag{1.79}$$

These properties are unique to Gaussians and a reason for why they are widely used for learning and inference.

1.11. Expectation and variance of Gaussians.

Derive that $\mathbb{E}[\mathbf{X}] = \boldsymbol{\mu}$ and $\mathrm{Var}[\mathbf{X}] = \boldsymbol{\Sigma}$ when $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.
Hint: First derive the expectation and variance of a univariate standard normal random variable.

1.12. Non-affine transformations of Gaussians.

Answer the following questions with **yes** or **no**.

1. Does there exist any non-affine transformation of a Gaussian random vector which is Gaussian? If yes, give an example.
2. Let X, Y, Z be independent standard normal random variables. Is $\frac{X+YZ}{\sqrt{1+Z^2}}$ Gaussian?

1.13. Decision theory.

Derive the optimal decisions under the squared loss and the asymmetric loss from Example 1.29.

PART I

Probabilistic Machine Learning

Preface to Part I

As humans, we constantly learn about the world around us. We learn to interact with our physical surroundings. We deepen our understanding of the world by establishing relationships between actors, objects, and events. And we learn about ourselves by observing how we interact with the world and with ourselves. We then continuously use this knowledge to make inferences and predictions, be it about the weather, the movement of a ball, or the behavior of a friend.

With limited computational resources, limited genetic information, and limited life experience, we are not able to learn everything about the world to complete certainty. We saw in Chapter 1 that probability theory is the mathematical framework for reasoning with uncertainty in the same way that logic is the mathematical framework for reasoning with certainty. We will discuss two kinds of uncertainty: "aleatoric" uncertainty which cannot be reduced under computational constraints, and "epistemic" uncertainty which can be reduced by observing more data.

An important aspect of learning is that we do not just learn once, but continually. Bayes' rule allows us to update our beliefs and reduce our uncertainty as we observe new data — a process that is called *probabilistic inference*. By taking the former posterior as the new prior, probabilistic inference can be performed continuously and repeated indefinitely as we observe more and more data.

Our sensory information is often noisy and imperfect, which is another source of uncertainty. The same is true for machines, even if they can sometimes sense aspects of the world more accurately than humans. We discuss how one can infer latent structure of the world from sensed data, such as the state of a dynamical system like a car, in a process that is called *filtering*.

In this first part of the manuscript, we examine how we can build machines that are capable of (continual) learning and inference. First, we introduce probabilistic inference in the context of linear models which

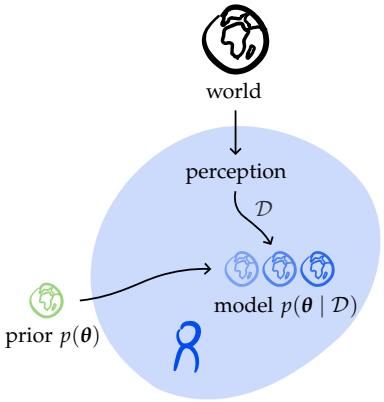


Figure 1.10: A schematic illustration of probabilistic inference in the context of the (supervised) learning of a model $\boldsymbol{\theta}$ from perceived data $\mathcal{D}$. The prior model $p(\boldsymbol{\theta})$ can equip the model with anything from substantial, to little, to no prior knowledge.

make predictions based on fixed (often hand-designed) features. We then discuss how probabilistic inference can be scaled to kernel methods and Gaussian processes which use a large (potentially infinite) number of features, and to deep neural networks which learn features dynamically from data. In these models, exact inference is typically intractable, and we discuss modern methods for approximate inference such as variational inference and Markov chain Monte Carlo. We highlight a tradeoff between curiosity (i.e., extrapolating beyond the given data) and conformity (i.e., fitting the given data), which surfaces as a fundamental principle of probabilistic inference in the regime where the data and our computational resources are limited.

2

Linear Regression

As a first example of probabilistic inference, we will study linear models for regression[1] which assume that the output $y \in \mathbb{R}$ is a linear function of the input $\boldsymbol{x} \in \mathbb{R}^d$:

$$y \approx \boldsymbol{w}^\top \boldsymbol{x} + w_0$$

where $\boldsymbol{w} \in \mathbb{R}^d$ are the weights and $w_0 \in \mathbb{R}$ is the intercept. Observe that if we define the extended inputs $\boldsymbol{x}' \doteq (\boldsymbol{x}, 1)$ and $\boldsymbol{w}' \doteq (\boldsymbol{w}, w_0)$, then $\boldsymbol{w}'^\top \boldsymbol{x}' = \boldsymbol{w}^\top \boldsymbol{x} + w_0$, implying that without loss of generality it suffices to study linear functions without the intercept term w_0. We will therefore consider the following function class of linear models

$$f(\boldsymbol{x}; \boldsymbol{w}) \doteq \boldsymbol{w}^\top \boldsymbol{x}.$$

We will consider the supervised learning task of learning weights $\boldsymbol{w}$ from labeled training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. We define the *design matrix*,

$$\boldsymbol{X} \doteq \begin{bmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_n^\top \end{bmatrix} \in \mathbb{R}^{n \times d}, \tag{2.1}$$

as the collection of inputs and the vector $\boldsymbol{y} \doteq [y_1 \cdots y_n]^\top \in \mathbb{R}^n$ as the collection of labels. For each noisy observation $(\boldsymbol{x}_i, y_i)$, we define the value of the approximation of our model, $f_i \doteq \boldsymbol{w}^\top \boldsymbol{x}_i$. Our model at the inputs $\boldsymbol{X}$ is described by the vector $\boldsymbol{f} \doteq [f_1 \ \cdots \ f_n]^\top$ which can be expressed succinctly as $\boldsymbol{f} = \boldsymbol{X}\boldsymbol{w}$.

The most common way of estimating $\boldsymbol{w}$ from data is the *least squares estimator*,

$$\hat{\boldsymbol{w}}_{\text{ls}} \doteq \arg\min_{\boldsymbol{w} \in \mathbb{R}^d} \sum_{i=1}^n (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 = \arg\min_{\boldsymbol{w} \in \mathbb{R}^d} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2, \tag{2.2}$$

[1] As we have discussed in Section 1.3, regression models can also be used for classification. The canonical example of a linear model for classification is logistic regression, which we will discuss in Section 5.1.1.

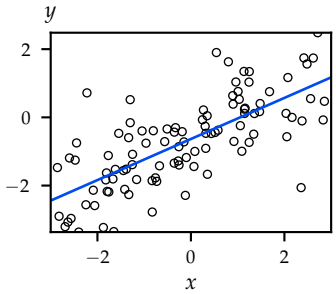


Figure 2.1: Example of linear regression with the least squares estimator (shown in blue).

minimizing the squared difference between the labels and predictions of the model. A slightly different estimator is used for *ridge regression*,

$$\hat{\boldsymbol{w}}_{\text{ridge}} \doteq \arg\min_{\boldsymbol{w}\in\mathbb{R}^d} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2 \tag{2.3}$$

where $\lambda > 0$. The squared L_2 regularization term $\lambda \|\boldsymbol{w}\|_2^2$ penalizes large $\boldsymbol{w}$ and thus reduces the "complexity" of the resulting model.[2] It can be shown that the unique solutions to least squares and ridge regression are given by

$$\hat{\boldsymbol{w}}_{\text{ls}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \qquad \text{and} \tag{2.4}$$

$$\hat{\boldsymbol{w}}_{\text{ridge}} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \tag{2.5}$$

respectively if the Hessian of the loss is positive definite (i.e., the loss is strictly convex) (?) which is the case as long as the columns of $\boldsymbol{X}$ are not linearly dependent. Least squares regression can be seen as finding the orthogonal projection of $\boldsymbol{y}$ onto the column space of $\boldsymbol{X}$, as is illustrated in Figure 2.2 (?).

[2] Ridge regression is more robust to multicollinearity than standard linear regression. *Multicollinearity* occurs when multiple independent inputs are highly correlated. In this case, their individual effects on the predicted variable cannot be estimated well. Classical linear regression is highly volatile to small input changes. The regularization of ridge regression reduces this volatility by introducing a bias on the weights towards 0.

Problem 2.1 (1)

Problem 2.1 (2)

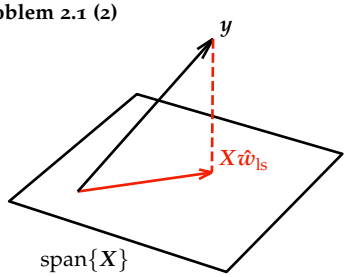


Figure 2.2: Least squares regression finds the orthogonal projection of $\boldsymbol{y}$ onto $\text{span}\{\boldsymbol{X}\}$ (here illustrated as the plane).

2.0.1 *Maximum Likelihood Estimation*

Since our function class comprises linear functions of the form $\boldsymbol{w}^\top \boldsymbol{x}$, the observation model from Equation (1.56) simplifies to

$$y_i = \boldsymbol{w}^{\star\top} \boldsymbol{x}_i + \varepsilon_i \tag{2.6}$$

for some weight vector $\boldsymbol{w}$, where for the purpose of this chapter we will additionally assume that $\varepsilon_i \sim \mathcal{N}(0, \sigma_{\text{n}}^2)$ is homoscedastic Gaussian noise.[3] This observation model is equivalently characterized by the Gaussian likelihood,

$$y_i \mid \boldsymbol{x}_i, \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{w}^\top \boldsymbol{x}_i, \sigma_{\text{n}}^2). \tag{2.7}$$

[3] ε_i is called *additive white Gaussian noise*.

using Equation (1.55)

Based on this likelihood we can compute the MLE (1.57) of the weights:

$$\hat{\boldsymbol{w}}_{\text{MLE}} = \arg\max_{\boldsymbol{w}\in\mathbb{R}^d} \sum_{i=1}^{n} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}) = \arg\min_{\boldsymbol{w}\in\mathbb{R}^d} \sum_{i=1}^{n} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2.$$

plugging in the Gaussian likelihood and simplifying

Note that therefore $\hat{\boldsymbol{w}}_{\text{MLE}} = \hat{\boldsymbol{w}}_{\text{ls}}$.

In practice, the noise variance σ_{n}^2 is typically unknown and also has to be determined, for example, through maximum likelihood estimation. It is a straightforward exercise to check that the MLE of σ_{n}^2 given fixed weights $\boldsymbol{w}$ is $\hat{\sigma}_{\text{n}}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2$ (?).

Problem 2.2

2.1 *Weight-space View*

The most immediate and natural probabilistic interpretation of linear regression is to quantify uncertainty about the weights $\boldsymbol{w}$. Recall

that probabilistic inference requires specification of a generative model comprised of prior and likelihood. Throughout this chapter, we will use the Gaussian prior,

$$\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \sigma_{\mathrm{p}}^2 \boldsymbol{I}), \tag{2.8}$$

and the Gaussian likelihood from Equation (2.7). We will discuss possible (probabilistic) strategies for choosing hyperparameters such as the prior variance σ_{p}^2 and the noise variance σ_{n}^2 in Section 4.4.

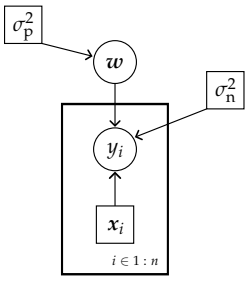


Figure 2.3: Directed graphical model of Bayesian linear regression in plate notation.

Remark 2.1: Why a Gaussian prior?

The choice of using a Gaussian prior may seem somewhat arbitrary at first sight, except perhaps for the nice analytical properties of Gaussians that we have seen in Section 1.2.3 and which will prove useful. The maximum entropy principle (cf. Section 1.2.1) provides a more fundamental justification for Gaussian priors since turns out that $\mathcal{N}$ has the maximum entropy among all distributions on $\mathbb{R}^d$ with known mean and variance (?).

Problem 5.6

Next, let us derive the posterior distribution over the weights.

$$\begin{aligned}
&\log p(\boldsymbol{w} \mid \boldsymbol{x}_{1:n}, y_{1:n}) \\
&= \log p(\boldsymbol{w}) + \log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{w}) + \text{const} && \text{by Bayes' rule (1.45)} \\
&= \log p(\boldsymbol{w}) + \sum_{i=1}^{n} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}) + \text{const} && \text{using independence of the samples} \\
&= -\frac{1}{2}\left[\sigma_{\mathrm{p}}^{-2} \|\boldsymbol{w}\|_2^2 + \sigma_{\mathrm{n}}^{-2} \sum_{i=1}^{n} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2\right] + \text{const} && \text{using the Gaussian prior and likelihood} \\
&= -\frac{1}{2}\left[\sigma_{\mathrm{p}}^{-2} \|\boldsymbol{w}\|_2^2 + \sigma_{\mathrm{n}}^{-2} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2\right] + \text{const} && \text{using } \textstyle\sum_{i=1}^{n} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 \\
&= -\frac{1}{2}\left[\sigma_{\mathrm{p}}^{-2} \boldsymbol{w}^\top \boldsymbol{w} + \sigma_{\mathrm{n}}^{-2}\left(\boldsymbol{w}^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{w} - 2\boldsymbol{y}^\top \boldsymbol{X} \boldsymbol{w} + \boldsymbol{y}^\top \boldsymbol{y}\right)\right] + \text{const} \\
&= -\frac{1}{2}\left[\boldsymbol{w}^\top (\sigma_{\mathrm{n}}^{-2} \boldsymbol{X}^\top \boldsymbol{X} + \sigma_{\mathrm{p}}^{-2} \boldsymbol{I}) \boldsymbol{w} - 2\sigma_{\mathrm{n}}^{-2} \boldsymbol{y}^\top \boldsymbol{X} \boldsymbol{w}\right] + \text{const}. && (2.9)
\end{aligned}$$

Observe that the log-posterior is a quadratic form in $\boldsymbol{w}$, so the posterior distribution must be Gaussian:

$$\boldsymbol{w} \mid \boldsymbol{x}_{1:n}, y_{1:n} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{2.10a}$$

see Equation (A.12)

where we can read off the mean and variance to be

$$\boldsymbol{\mu} \doteq \sigma_{\mathrm{n}}^{-2} \boldsymbol{\Sigma} \boldsymbol{X}^\top \boldsymbol{y}, \tag{2.10b}$$

$$\boldsymbol{\Sigma} \doteq \left(\sigma_{\mathrm{n}}^{-2} \boldsymbol{X}^\top \boldsymbol{X} + \sigma_{\mathrm{p}}^{-2} \boldsymbol{I}\right)^{-1}. \tag{2.10c}$$

This also shows that Gaussians with known variance and linear likelihood are self-conjugate, a property that we had hinted at in Section 1.2.2. It can be shown more generally that Gaussians with known

variance are self-conjugate to any Gaussian likelihood (Murphy, 2007). For other generative models, the posterior can typically not be expressed in closed-form — this is a very special property of Gaussians!

2.1.1 *Maximum a Posteriori Estimation*

Computing the MAP estimate for the weights,

$$\begin{aligned}\hat{\boldsymbol{w}}_{\text{MAP}} &= \arg\max_{\boldsymbol{w}} \log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{w}) + \log p(\boldsymbol{w}) \\ &= \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \frac{\sigma_{\text{n}}^2}{\sigma_{\text{p}}^2} \|\boldsymbol{w}\|_2^2 , \end{aligned} \tag{2.11}$$

using that the likelihood and prior are Gaussian

we observe that this is identical to ridge regression with weight decay $\lambda \doteq \sigma_{\text{n}}^2/\sigma_{\text{p}}^2$: $\hat{\boldsymbol{w}}_{\text{MAP}} = \hat{\boldsymbol{w}}_{\text{ridge}}$. Equation (2.11) is simply the MLE loss with an additional L_2-regularization (originating from the prior) that encourages keeping weights small. Recall that the MAP estimate corresponds to the mode of the posterior distribution, which in the case of a Gaussian is simply its mean $\boldsymbol{\mu}$. As to be expected, $\boldsymbol{\mu}$ coincides with the analytical solution to ridge regression from Equation (2.5).

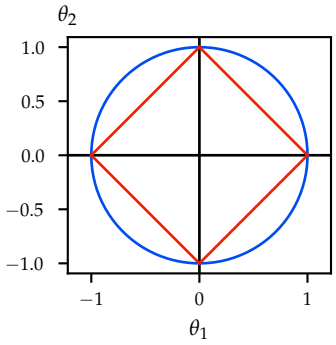


Figure 2.4: Level sets of L_2- (blue) and L_1-regularization (red), corresponding to Gaussian and Laplace priors, respectively. It can be seen that L_1-regularization is more effective in encouraging sparse solutions (that is, solutions where many components are set to exactly 0).

Example 2.2: Lasso as the MAP estimate with a Laplace prior

One problem with ridge regression is that the contribution of nearly-zero weights to the L_2-regularization term is negligible. Thus, L_2-regularization is typically not sufficient to perform variable selection (that is, set some weights to zero entirely), which is often desirable for interpretability of the model.

A commonly used alternative to ridge regression is the *least absolute shrinkage and selection operator* (or *lasso*), which regularizes with the L_1-norm:

$$\hat{\boldsymbol{w}}_{\text{lasso}} \doteq \arg\min_{\boldsymbol{w} \in \mathbb{R}^d} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_1 . \tag{2.12}$$

It turns out that lasso can also be viewed as probabilistic inference, using a Laplace prior $\boldsymbol{w} \sim \text{Laplace}(\boldsymbol{0}, h)$ with length scale h instead of a Gaussian prior.

Computing the MAP estimate for the weights yields,

$$\begin{aligned}\hat{\boldsymbol{w}}_{\text{MAP}} &= \arg\max_{\boldsymbol{w}} \log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{w}) + \log p(\boldsymbol{w}) \\ &= \arg\min_{\boldsymbol{w}} \sum_{i=1}^{n} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 + \frac{\sigma_{\text{n}}^2}{h} \|\boldsymbol{w}\|_1 \end{aligned} \tag{2.13}$$

using that the likelihood is Gaussian and the prior is Laplacian

which coincides with the lasso with weight decay $\lambda \doteq \sigma_{\text{n}}^2/h$.

To make predictions at a test point $\boldsymbol{x}^\star$, we define the (model-)predicted point $f^\star \doteq \hat{\boldsymbol{w}}_{\text{MAP}}^\top \boldsymbol{x}^\star$ and obtain the label prediction

$$y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n} \sim \mathcal{N}(f^\star, \sigma_{\text{n}}^2). \tag{2.14}$$

Here we observe that using point estimates such as the MAP estimate does not quantify uncertainty in the weights. The MAP estimate simply collapses all mass of the posterior around its mode. This can be harmful when we are highly unsure about the best model, e.g., because we have observed insufficient data.

2.1.2 *Probabilistic Inference*

Rather than selecting a single weight vector $\hat{\boldsymbol{w}}$ to make predictions, we can use the full posterior distribution. This is known as *Bayesian linear regression* (BLR) and illustrated with an example in Figure 2.5.

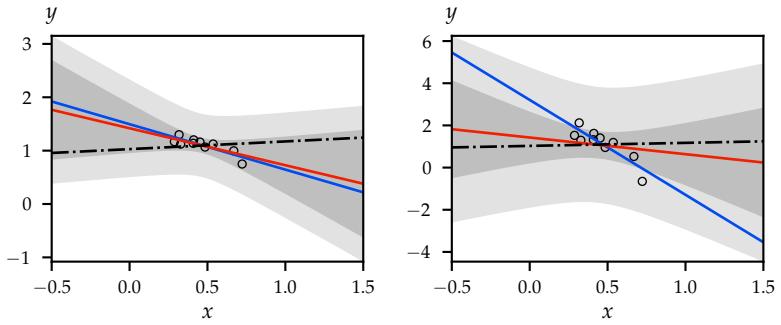


Figure 2.5: Comparison of **linear regression (MLE)**, **ridge regression (MAP estimate)**, and **Bayesian linear regression** when the data is generated according to

$$y \mid \boldsymbol{w}, \boldsymbol{x} \sim \mathcal{N}(\boldsymbol{w}^\top \boldsymbol{x}, \sigma_{\text{n}}^2).$$

The **true mean** is shown in black, the MLE in blue, and the MAP estimate in red. The dark gray area denotes the epistemic uncertainty of Bayesian linear regression and the light gray area the additional homoscedastic noise. On the left, $\sigma_{\text{n}} = 0.15$. On the right, $\sigma_{\text{n}} = 0.7$.

To make predictions at a test point $\boldsymbol{x}^\star$, we let $f^\star \doteq \boldsymbol{w}^\top \boldsymbol{x}^\star$ which has the distribution

$$f^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n} \sim \mathcal{N}(\boldsymbol{\mu}^\top \boldsymbol{x}^\star, \boldsymbol{x}^{\star\top} \boldsymbol{\Sigma} \boldsymbol{x}^\star). \tag{2.15}$$

using the closedness of Gaussians under linear transformations (1.78)

Note that this does not take into account the noise in the labels σ_{n}^2. For the label prediction $y^\star$, we obtain

$$y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n} \sim \mathcal{N}(\boldsymbol{\mu}^\top \boldsymbol{x}^\star, \boldsymbol{x}^{\star\top} \boldsymbol{\Sigma} \boldsymbol{x}^\star + \sigma_{\text{n}}^2). \tag{2.16}$$

using additivity of Gaussians (1.79)

2.1.3 *Recursive Probabilistic Inference*

We have already discussed the recursive properties of probabilistic inference in Section 1.3.6. For Bayesian linear regression with a Gaussian prior and likelihood, this principle can be used to derive an efficient online algorithm since also the posterior is a Gaussian,

$$p^{(t)}(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}^{(t)}, \boldsymbol{\Sigma}^{(t)}), \tag{2.17}$$

which can be stored efficiently using only $O(d^2)$ parameters. This leads to an efficient online algorithm for Bayesian linear regression

with time-independent(!) memory complexity $O(d)$ and round complexity $O(d^2)$ (?). The interpretation of Bayesian linear regression as an online algorithm also highlights similarities to other sequential models such as Kalman filters, which we discuss in Chapter 3. In Example 3.5, we will learn that online Bayesian linear regression is, in fact, an example of a Kalman filter.

Problem 2.5

2.2 *Aleatoric and Epistemic Uncertainty*

The predictive posterior distribution from Equation (2.16) highlights a decomposition of uncertainty wherein $\boldsymbol{x}^{\star\top}\boldsymbol{\Sigma}\boldsymbol{x}^\star$ corresponds to the uncertainty about our model due to the lack of data (commonly referred to as the *epistemic uncertainty*) and σ_n^2 corresponds to the uncertainty about the labels that cannot be explained by the inputs and any model from the model class (commonly referred to as the *aleatoric uncertainty*, "irreducible noise", or simply "(label) noise") (?).

Problem 2.6

A natural probabilistic approach is to represent epistemic uncertainty with a probability distribution over models. Intuitively, the variance of this distribution measures our uncertainty about the model and its mode corresponds to our current best (point) estimate. The distribution over weights of a linear model is one example, and we will continue to explore this approach for other models in the following chapters.

It is a practical modeling choice how much inaccuracy to attribute to epistemic or aleatoric uncertainty. Generally, when a poor model is used to explain a process, more inaccuracy has to be attributed to irreducible noise. For example, when a linear model is used to "explain" a nonlinear process, most uncertainty is aleatoric as the model cannot explain the data well. As we use more expressive models, a larger portion of the uncertainty can be explained by the data.

Epistemic and aleatoric uncertainty can be formally defined in terms of the law of total variance (1.41),

$$\mathrm{Var}[y^\star \mid \boldsymbol{x}^\star] = \underbrace{\mathbb{E}_{\boldsymbol{\theta}}\left[\mathrm{Var}_{y^\star}[y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}]\right]}_{\text{aleatoric uncertainty}} + \underbrace{\mathrm{Var}_{\boldsymbol{\theta}}\left[\mathbb{E}_{y^\star}[y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}]\right]}_{\text{epistemic uncertainty}}. \tag{2.18}$$

Here, the mean variability of predictions $y^\star$ averaged across all models $\boldsymbol{\theta}$ is the estimated *aleatoric uncertainty*. In contrast, the variability of the mean prediction $y^\star$ under each model $\boldsymbol{\theta}$ is the estimated *epistemic uncertainty*. This decomposition of uncertainty will appear frequently throughout this manuscript.

2.3 *Non-linear Regression*

We can use linear regression not only to learn linear functions. The trick is to apply a nonlinear transformation $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^e$ to the features $\boldsymbol{x}_i$, where d is the dimension of the input space and e is the dimension of the designed *feature space*. We denote the design matrix comprised of transformed features by $\boldsymbol{\Phi} \in \mathbb{R}^{n \times e}$. Note that if the feature transformation $\boldsymbol{\phi}$ is the identity function then $\boldsymbol{\Phi} = \boldsymbol{X}$.

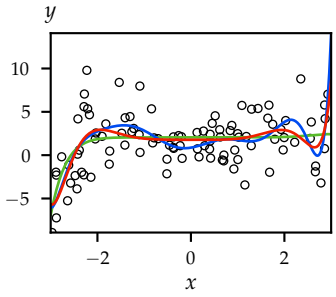


Figure 2.6: Applying linear regression with a feature space of polynomials of degree 10. The **least squares estimate** is shown in blue, **ridge regression** in red, and **lasso** in green.

Example 2.3: Polynomial regression

Let $\boldsymbol{\phi}(x) \doteq [x^2, x, 1]$ and $\boldsymbol{w} \doteq [a, b, c]$. Then the function that our model learns is given as

$$f = ax^2 + bx + c.$$

Thus, our model can exactly represent all polynomials up to degree 2.

However, to learn polynomials of degree m in d input dimensions, we need to apply the nonlinear transformation

$$\begin{aligned}\boldsymbol{\phi}(\boldsymbol{x}) = [1, x_1, \ldots, x_d, x_1^2, \ldots, x_d^2, x_1 \cdot x_2, \ldots, x_{d-1} \cdot x_d,\\ \ldots,\\ x_{d-m+1} \cdot \cdots \cdot x_d].\end{aligned}$$

Note that the feature dimension e is $\sum_{i=0}^{m} \binom{d+i-1}{i} = \Theta(d^m)$.[4] Thus, the dimension of the feature space grows exponentially in the degree of polynomials and input dimensions. Even for relatively small m and d, this becomes completely unmanageable.

The example of polynomials highlights that it may be inefficient to keep track of the weights $\boldsymbol{w} \in \mathbb{R}^e$ when e is large, and that it may be useful to instead consider a reparameterization which is of dimension n rather than of the feature dimension.

[4] Observe that the vector contains $\binom{d+i-1}{i}$ monomials of degree i as this is the number of ways to choose i times from d items with replacement and without consideration of order. To see this, consider the following encoding: We take a sequence of $d + i - 1$ spots. Selecting any subset of i spots, we interpret the remaining $d - 1$ spots as "barriers" separating each of the d items. The selected spots correspond to the number of times each item has been selected. For example, if 2 items are to be selected out of a total of 4 items with replacement, one possible configuration is "$\circ \,||\, \circ \,|$" where $\circ$ denotes a selected spot and $|$ denotes a barrier. This configuration encodes that the first and third item have each been chosen once. The number of possible configurations — each encoding a unique outcome — is therefore $\binom{d+i-1}{i}$.

2.4 *Function-space View*

Let us now look at Bayesian linear regression through a different lens. Previously, we have been interpreting it as a distribution over the weights $\boldsymbol{w}$ of a linear function $\boldsymbol{f} = \boldsymbol{\Phi}\boldsymbol{w}$. The key idea is that for a finite set of inputs (ensuring that the design matrix is well-defined), we can equivalently consider a distribution directly over the estimated function values $\boldsymbol{f}$. We call this the *function-space view* of Bayesian linear regression.

Instead of considering a prior over the weights $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \sigma_\text{p}^2 \boldsymbol{I})$ as we have done previously, we now impose a prior directly on the values of our model at the observations. Using that Gaussians are closed under linear maps (1.78), we obtain the equivalent prior

$$\boldsymbol{f} \mid \boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\Phi}\mathbb{E}[\boldsymbol{w}], \boldsymbol{\Phi}\text{Var}[\boldsymbol{w}]\boldsymbol{\Phi}^\top) = \mathcal{N}(\mathbf{0}, \underbrace{\sigma_\text{p}^2 \boldsymbol{\Phi}\boldsymbol{\Phi}^\top}_{\boldsymbol{K}}) \tag{2.19}$$

where $\boldsymbol{K} \in \mathbb{R}^{n \times n}$ is the so-called *kernel matrix*. Observe that the entries of the kernel matrix can be expressed as $\boldsymbol{K}(i,j) = \sigma_\text{p}^2 \cdot \boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\phi}(\boldsymbol{x}_j)$.

You may say that nothing has changed, and you would be right — that is precisely the point. Note, however, that the shape of the kernel matrix is $n \times n$ rather than the $e \times e$ covariance matrix over weights, which becomes unmanageable when e is large. The kernel matrix $\boldsymbol{K}$ has entries only for the finite set of observed inputs. However, in principle, we could have observed any input, and this motivates the definition of the *kernel function*

$$k(\boldsymbol{x}, \boldsymbol{x}') \doteq \sigma_\text{p}^2 \cdot \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}') \tag{2.20}$$

for arbitrary inputs $\boldsymbol{x}$ and $\boldsymbol{x}'$. A kernel matrix is simply a finite "view" of the kernel function,

$$\boldsymbol{K} = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{bmatrix} \tag{2.21}$$

Observe that by definition of the kernel matrix in Equation (2.19), the kernel matrix is a covariance matrix and the kernel function measures the covariance of the function values $f(\boldsymbol{x})$ and $f(\boldsymbol{x}')$ given inputs $\boldsymbol{x}$ and $\boldsymbol{x}'$:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \text{Cov}\big[f(\boldsymbol{x}), f(\boldsymbol{x}')\big]. \tag{2.22}$$

Moreover, note that we have reformulated[5] the learning algorithm such that the feature space is now *implicit* in the choice of kernel, and the kernel is defined by inner products of (nonlinearly transformed) inputs. In other words, the choice of kernel implicitly determines the class of functions that $\boldsymbol{f}$ is sampled from (without expressing the functions explicitly in closed-form), which encodes our prior beliefs. This is known as the *kernel trick*.

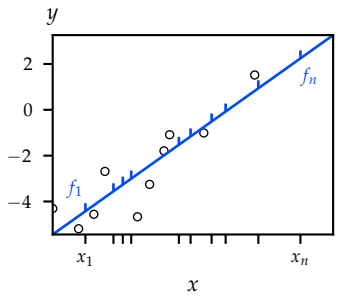


Figure 2.7: An illustration of the function-space view. The model is described by the points (x_i, f_i).

[5] we often say "kernelized"

2.4.1 *Learning and Predictions*

We have already kernelized the Bayesian linear regression prior. The posterior distribution $\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{y}$ is again Gaussian due to the closedness

properties of Gaussians, analogously to our derivation of the prior kernel matrix in Equation (2.19).

It remains to show that we can also rely on the kernel trick for predictions. Given the test point $\boldsymbol{x}^\star$, we define

$$\tilde{\boldsymbol{\Phi}} \doteq \begin{bmatrix} \boldsymbol{\Phi} \\ \boldsymbol{\phi}(\boldsymbol{x}^\star)^\top \end{bmatrix}, \quad \tilde{\boldsymbol{y}} \doteq \begin{bmatrix} \boldsymbol{y} \\ y^\star \end{bmatrix}, \quad \tilde{\boldsymbol{f}} \doteq \begin{bmatrix} \boldsymbol{f} \\ f^\star \end{bmatrix}.$$

We immediately obtain $\tilde{\boldsymbol{f}} = \tilde{\boldsymbol{\Phi}}\boldsymbol{w}$. Analogously to our analysis of predictions from the weight-space view, we add the label noise to obtain the estimate $\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{f}} + \tilde{\boldsymbol{\varepsilon}}$ where $\tilde{\boldsymbol{\varepsilon}} \doteq [\varepsilon_1 \ \cdots \ \varepsilon_n \ \varepsilon^\star]^\top \sim \mathcal{N}(\boldsymbol{0}, \sigma_\mathrm{n}^2 \boldsymbol{I})$ is the independent label noise. Applying the same reasoning as we did for the prior, we obtain

$$\tilde{\boldsymbol{f}} \mid \boldsymbol{X}, \boldsymbol{x}^\star \sim \mathcal{N}(\boldsymbol{0}, \tilde{\boldsymbol{K}}) \tag{2.23}$$

where $\tilde{\boldsymbol{K}} \doteq \sigma_\mathrm{p}^2 \tilde{\boldsymbol{\Phi}} \tilde{\boldsymbol{\Phi}}^\top$. Adding the label noise yields

$$\tilde{\boldsymbol{y}} \mid \boldsymbol{X}, \boldsymbol{x}^\star \sim \mathcal{N}(\boldsymbol{0}, \tilde{\boldsymbol{K}} + \sigma_\mathrm{n}^2 \boldsymbol{I}). \tag{2.24}$$

Finally, we can conclude from the closedness of Gaussian random vectors under conditional distributions (1.53) that the predictive posterior $y^\star \mid \boldsymbol{x}^\star, \boldsymbol{X}, \boldsymbol{y}$ follows again a normal distribution. We will do a full derivation of the posterior and predictive posterior in Section 4.1.

2.4.2 *Efficient Polynomial Regression*

But how does the kernel trick address our concerns about efficiency raised in Section 2.3? After all, computing the kernel for a feature space of dimension e still requires computing sums of length e which is prohibitive when e is large. The kernel trick opens up a couple of new doors for us:

1. For certain feature transformations $\boldsymbol{\phi}$, we may be able to find an easier to compute expression equivalent to $\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')$.
2. If this is not possible, we could approximate the inner product by an easier to compute expression.
3. Or, alternatively, we may decide not to care very much about the exact feature transformation and simply experiment with kernels that induce *some* feature space (which may even be infinitely dimensional).

We will explore the third approach when we revisit kernels in Section 4.3. A polynomial feature transformation can be computed efficiently in closed-form.

Fact 2.4. *For the polynomial feature transformation $\boldsymbol{\phi}$ up to degree m from Example 2.3, it can be shown that up to constant factors,*

$$\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}') = (1 + \boldsymbol{x}^\top \boldsymbol{x}')^m. \tag{2.25}$$

For example, for input dimension 2, the kernel $(1 + \boldsymbol{x}^\top \boldsymbol{x}')^2$ corresponds to the feature vector $\boldsymbol{\phi}(\boldsymbol{x}) = [1 \;\; \sqrt{2}x_1 \;\; \sqrt{2}x_2 \;\; \sqrt{2}x_1x_2 \;\; x_1^2 \;\; x_2^2]^\top$.

Discussion

We have explored a probabilistic perspective on linear models, and seen that classical approaches such as least squares and ridge regression can be interpreted as approximate probabilistic inference. We then saw that we can even perform *exact* probabilistic inference efficiently if we adopt a Gaussian prior and Gaussian noise assumption. These are already powerful tools, which are often applied also to nonlinear models if we treat the latent feature space — which was either human-designed or learned via deep learning — as fixed. In the next chapter, we will digress briefly from the storyline on "learning" to see how we can adopt a similar probabilistic perspective to track latent states over time. Then, in Chapter 4, we will see how we can use the function-space view and kernel trick to learn flexible nonlinear models with exact probabilistic inference, without ever explicitly representing the feature space.

Problems

2.1. Closed-form linear regression.

1. Derive the unique solutions to least squares and ridge regression from Equations (2.4) and (2.5).
2. For an $n \times m$ matrix $\boldsymbol{A}$ and vector $\boldsymbol{x} \in \mathbb{R}^m$, we call $\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{x}$ the orthogonal projection of $\boldsymbol{x}$ onto $\operatorname{span}\{\boldsymbol{A}\} = \{\boldsymbol{A}\boldsymbol{x}' \mid \boldsymbol{x}' \in \mathbb{R}^m\}$. In particular, an orthogonal projection satisfies $\boldsymbol{x} - \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{x} \perp \boldsymbol{A}\boldsymbol{x}'$ for all $\boldsymbol{x}' \in \mathbb{R}^m$. Show that $\hat{\boldsymbol{w}}_{\text{ls}}$ from Equation (2.4) is such that $\boldsymbol{X}\hat{\boldsymbol{w}}_{\text{ls}}$ is the unique closest point to $\boldsymbol{y}$ on $\operatorname{span}\{\boldsymbol{X}\}$, i.e., it satisfies $\boldsymbol{X}\hat{\boldsymbol{w}}_{\text{ls}} = \boldsymbol{\Pi}_{\boldsymbol{X}}\boldsymbol{y}$.

2.2. MLE of noise variance.

Show that the MLE of σ_{n}^2 given fixed weights $\boldsymbol{w}$ is

$$\hat{\sigma}_n^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2. \tag{2.26}$$

2.3. Variance of least squares around training data.

Show that the variance of a prediction at the point $[1 \;\; x^\star]^\top$ is smallest when $x^\star$ is the mean of the training data. More formally, show that if inputs are of the form $\boldsymbol{x}_i = [1 \;\; x_i]^\top$ where $x_i \in \mathbb{R}$ and $\hat{\boldsymbol{w}}_{\text{ls}}$ is the least squares estimate, then $\operatorname{Var}\left[y^\star \mid [1 \;\; x^\star]^\top, \hat{\boldsymbol{w}}_{\text{ls}}\right]$ is minimized for $x^\star = \frac{1}{n}\sum_{i=1}^{n} x_i$.

2.4. Bayesian linear regression.

Suppose you are given the following observations

$$\boldsymbol{X} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 2 & 1 \\ 2 & 2 \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} 2.4 \\ 4.3 \\ 3.1 \\ 4.9 \end{bmatrix}$$

and assume the data follows a linear model with homoscedastic noise $\mathcal{N}(0, \sigma_{\mathrm{n}}^2)$ where $\sigma_{\mathrm{n}}^2 = 0.1$.

1. Find the maximum likelihood estimate $\hat{\boldsymbol{w}}_{\mathrm{MLE}}$ given the data.
2. Now assume that we have a prior $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}; \boldsymbol{0}, \sigma_{\mathrm{p}}^2 \boldsymbol{I})$ with $\sigma_{\mathrm{p}}^2 = 0.05$. Find the MAP estimate $\hat{\boldsymbol{w}}_{\mathrm{MAP}}$ given the data and the prior.
3. Use the posterior $p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y})$ to get a posterior prediction for the label $y^\star$ at $\boldsymbol{x}^\star = [3 \quad 3]^\top$. Report the mean and the variance of this prediction.
4. How would you have to change the prior $p(\boldsymbol{w})$ such that

$$\hat{\boldsymbol{w}}_{\mathrm{MAP}} \to \hat{\boldsymbol{w}}_{\mathrm{MLE}}?$$

2.5. Online Bayesian linear regression.

1. Can you design an algorithm that updates the posterior (as opposed to recalculating it from scratch using Equation (2.10)) in a smarter way? The requirement is that the memory should not grow as $O(t)$.
2. If d is large, computing the inverse every round is very expensive. Can you use the recursive structure you found in the previous question to bring down the computational complexity of every round to $O(d^2)$?

The resulting efficient online algorithm is known as *online Bayesian linear regression*.

2.6. Aleatoric and epistemic uncertainty of BLR.

Prove for Bayesian linear regression that $\boldsymbol{x}^{\star\top} \boldsymbol{\Sigma} \boldsymbol{x}^\star$ is the epistemic uncertainty and σ_{n}^2 the aleatoric uncertainty in $y^\star$ under the decomposition of Equation (2.18).

2.7. Hyperpriors.

We consider a dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ of size n, where $\boldsymbol{x}_i \in \mathbb{R}^d$ denotes the feature vector and $y_i \in \mathbb{R}$ denotes the label of the i-th data point. Let ε_i be i.i.d. samples from the Gaussian distribution $\mathcal{N}(0, \lambda^{-1})$ for a given $\lambda > 0$. We collect the labels in a vector $\boldsymbol{y} \in \mathbb{R}^n$, the features in a matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, and the noise in a vector $\boldsymbol{\varepsilon} \in \mathbb{R}^n$. The labels are generated according to $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + \boldsymbol{\varepsilon}$.

To perform Bayesian Linear Regression, we consider the prior distribution over the parameter vector $\boldsymbol{w}$ to be $\mathcal{N}(\boldsymbol{\mu}, \lambda^{-1}\boldsymbol{I}_d)$, where $\boldsymbol{I}_d$ denotes the d-dimensional identity matrix and $\boldsymbol{\mu} \in \mathbb{R}^d$ is a hyperparameter.

1. Given this Bayesian data model, what is the conditional covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{y}} \doteq \mathrm{Var}[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\mu}, \lambda]$?
2. Calculate the maximum likelihood estimate of the hyperparameter $\boldsymbol{\mu}$.
3. Since we are unsure about the hyperparameter $\boldsymbol{\mu}$, we decide to model our uncertainty about $\boldsymbol{\mu}$ by placing the "*hyperprior*" $\boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_d)$. Is the posterior distribution $p(\boldsymbol{\mu} \mid \boldsymbol{X}, \boldsymbol{y}, \lambda)$ a Gaussian distribution? If yes, what are its mean vector and covariance matrix?
4. What is the posterior distribution $p(\lambda \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\mu})$?
   Hint: For any $a \in \mathbb{R}, \boldsymbol{A} \in \mathbb{R}^{n \times n}$ it holds that $\det(a\boldsymbol{A}) = a^n \det(\boldsymbol{A})$.

3

Filtering

Before we continue in Chapter 4 with the function-space view of regression, we want to look at a seemingly different but very related problem. We will study Bayesian learning and inference in the *state space model*, where we want to keep track of the state of an agent over time based on noisy observations. In this model, we have a sequence of (hidden) states $(\mathbf{X}_t)_{t \in \mathbb{N}_0}$ where $\mathbf{X}_t$ is in $\mathbb{R}^d$ and a sequence of observations $(\mathbf{Y}_t)_{t \in \mathbb{N}_0}$ where $\mathbf{Y}_t$ is in $\mathbb{R}^m$.

The process of keeping track of the state using noisy observations is also known as *Bayesian filtering* or *recursive Bayesian estimation*. Figure 3.1 illustrates this process, where an agent perceives the current state of the world and then updates its beliefs about the state based on this observation.

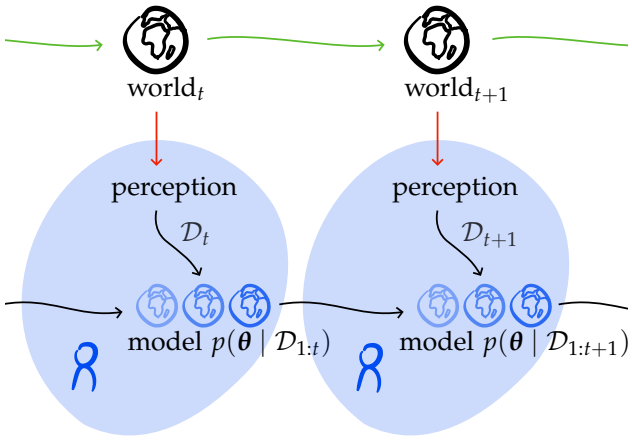


Figure 3.1: Schematic view of Bayesian filtering: An agent perceives the current state of the world and updates its belief accordingly.

We will discuss Bayesian filtering more broadly in the next section. A Kalman filter is an important special case of a Bayes' filter, which uses a Gaussian prior over the states and conditional linear Gaussians to describe the evolution of states and observations. Analogously to the

previous chapter, we will see that inference in this model is tractable due to the closedness properties of Gaussians.

Definition 3.1 (Kalman filter)**.** A *Kalman filter* is specified by a Gaussian prior over the states,

$$\mathbf{X}_0 \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{3.1}$$

and a conditional linear Gaussian *motion model* and *sensor model*,

$$\mathbf{X}_{t+1} \doteq \boldsymbol{F}\mathbf{X}_t + \boldsymbol{\varepsilon}_t \qquad \boldsymbol{F} \in \mathbb{R}^{d \times d}, \boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_x), \tag{3.2}$$

$$\mathbf{Y}_t \doteq \boldsymbol{H}\mathbf{X}_t + \boldsymbol{\eta}_t \qquad \boldsymbol{H} \in \mathbb{R}^{m \times d}, \boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_y), \tag{3.3}$$

respectively. The motion model is sometimes also called *transition model* or *dynamics model*. Crucially, Kalman filters assume that $\boldsymbol{F}$ and $\boldsymbol{H}$ are known. In general, $\boldsymbol{F}$ and $\boldsymbol{H}$ may depend on t. Also, $\boldsymbol{\varepsilon}$ and $\boldsymbol{\eta}$ may have a non-zero mean, commonly called a "drift".

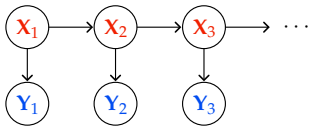


Figure 3.2: Directed graphical model of a Kalman filter with hidden states $\mathbf{X}_t$ and observables $\mathbf{Y}_t$.

Because Kalman filters use conditional linear Gaussians, which we have already seen in Equation (1.55), their joint distribution (over all variables) is also Gaussian. This means that predicting the future states of a Kalman filter is simply inference with multivariate Gaussians. In Bayesian filtering, however, we do not only want to make predictions occasionally. In Bayesian filtering, we want to *keep track* of states, that is, predict the current state of an agent online.[1] To do this efficiently, we need to update our *belief* about the state of the agent recursively, similarly to our recursive Bayesian updates in Bayesian linear regression (see Section 2.1.3).

[1] Here, *online* is common terminology to say that we want to perform inference at time t without exposure to times $t+1, t+2, \ldots$, so in "real-time".

From the directed graphical model of a Kalman filter shown in Figure 3.2, we can immediately gather the following conditional independence relations,[2]

$$\mathbf{X}_{t+1} \perp \mathbf{X}_{1:t-1}, \mathbf{Y}_{1:t-1} \mid \mathbf{X}_t, \tag{3.4}$$

$$\mathbf{Y}_t \perp \mathbf{X}_{1:t-1} \mid \mathbf{X}_t \tag{3.5}$$

$$\mathbf{Y}_t \perp \mathbf{Y}_{1:t-1} \mid \mathbf{X}_{t-1}. \tag{3.6}$$

[2] Alternatively, they follow from the definition of the motion and sensor models as linear updates.

The first conditional independence property is also known as the *Markov property*, which we will return to later in our discussion of Markov chains and Markov decision processes. This characterization of the Kalman filter, yields the following factorization of the joint distribution:

$$\begin{aligned} p(\boldsymbol{x}_{1:t}, \boldsymbol{y}_{1:t}) &= \prod_{i=1}^{t} p(\boldsymbol{x}_i \mid \boldsymbol{x}_{1:i-1}) p(\boldsymbol{y}_i \mid \boldsymbol{x}_{1:t}, \boldsymbol{y}_{1:i-1}) && \text{using the product rule (1.11)} \\ &= p(\boldsymbol{x}_1) p(\boldsymbol{y}_1 \mid \boldsymbol{x}_1) \prod_{i=2}^{t} p(\boldsymbol{x}_i \mid \boldsymbol{x}_{i-1}) p(\boldsymbol{y}_i \mid \boldsymbol{x}_i). && \text{using the conditional independence properties from (3.4), (3.5), and (3.6)} \end{aligned} \tag{3.7}$$

3.1 Conditioning and Prediction

We can describe Bayesian filtering by the following recursive scheme with the two phases, *conditioning* (also called "update") and *prediction*:

Algorithm 3.2: Bayesian filtering

start with a prior over initial states $p(\boldsymbol{x}_0)$
for $t = 1$ **to** ∞ **do**
  assume we have $p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1})$
  conditioning: compute $p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t})$ using the new observation $\boldsymbol{y}_t$
  prediction: compute $p(\boldsymbol{x}_{t+1} \mid \boldsymbol{y}_{1:t})$

Let us consider the conditioning step first:

$$\begin{aligned} p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t}) &= \frac{1}{Z} p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}) p(\boldsymbol{y}_t \mid \boldsymbol{x}_t, \boldsymbol{y}_{1:t-1}) && \text{using Bayes' rule (1.45)} \\ &= \frac{1}{Z} p(\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t-1}) p(\boldsymbol{y}_t \mid \boldsymbol{x}_t). \quad (3.8) && \text{using the conditional independence structure (3.6)} \end{aligned}$$

For the prediction step, we obtain,

$$\begin{aligned} p(\boldsymbol{x}_{t+1} \mid \boldsymbol{y}_{1:t}) &= \int p(\boldsymbol{x}_{t+1}, \boldsymbol{x}_t \mid \boldsymbol{y}_{1:t}) \, d\boldsymbol{x}_t && \text{using the sum rule (1.7)} \\ &= \int p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, y_{1:t}) p(\boldsymbol{x}_t \mid y_{1:t}) \, d\boldsymbol{x}_t && \text{using the product rule (1.11)} \\ &= \int p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t) p(\boldsymbol{x}_t \mid y_{1:t}) \, d\boldsymbol{x}_t. \quad (3.9) && \text{using the conditional independence structure (3.4)} \end{aligned}$$

In general, these distributions can be very complicated, but for Gaussians (i.e., Kalman filters) they can be expressed in closed-form.

Remark 3.3: Bayesian smoothing

Bayesian smoothing is a closely related task to Bayesian filtering. While Bayesian filtering methods estimate the current state based only on observations obtained before and at the current time step, Bayesian smoothing computes the distribution of $\mathbf{X}_k \mid \boldsymbol{y}_{1:t}$ where $t > k$. That is Bayesian smoothing estimates $\mathbf{X}_k$ based on data until *and beyond* time k. Note that if $k = t$, then Bayesian smoothing coincides with Bayesian filtering.

Analogously to Equation (3.8),

$$p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:t}) \propto p(\boldsymbol{x}_k \mid \boldsymbol{y}_{1:k}) p(\boldsymbol{y}_{k+1:t} \mid \boldsymbol{x}_k). \quad (3.10)$$

If we assume a Gaussian prior and conditional Gaussian transition and dynamics models (this is called *Kalman smoothing*), then

by the closedness properties of Gaussians, $\mathbf{X}_k \mid \boldsymbol{y}_{1:t}$ is a Gaussian. Indeed, all terms of Equation (3.10) are Gaussian PDFs and as seen in Equation (1.75), the product of two Gaussian PDFs is again proportional to a Gaussian PDF.

The first term, $\mathbf{X}_k \mid \boldsymbol{y}_{1:k}$, is the marginal posterior of the hidden states of the Kalman filter which can be obtained with Bayesian filtering.

By conditioning on $\mathbf{X}_{k+1}$, we have for the second term,

$$\begin{aligned} p(\boldsymbol{y}_{k+1:t} \mid \boldsymbol{x}_k) &= \int p(\boldsymbol{y}_{k+1:t} \mid \boldsymbol{x}_k, \boldsymbol{x}_{k+1}) p(\boldsymbol{x}_{k+1} \mid \boldsymbol{x}_k) \, d\boldsymbol{x}_{k+1} && \text{using the sum rule (1.7) and product rule (1.11)} \\ &= \int p(\boldsymbol{y}_{k+1:t} \mid \boldsymbol{x}_{k+1}) p(\boldsymbol{x}_{k+1} \mid \boldsymbol{x}_k) \, d\boldsymbol{x}_{k+1} && \text{using the conditional independence structure (3.5)} \\ &= \int p(\boldsymbol{y}_{k+1} \mid \boldsymbol{x}_{k+1}) p(\boldsymbol{y}_{k+2:t} \mid \boldsymbol{x}_{k+1}) p(\boldsymbol{x}_{k+1} \mid \boldsymbol{x}_k) \, d\boldsymbol{x}_{k+1} && \text{using the conditional independence structure (3.6)} \end{aligned} \tag{3.11}$$

Let us have a look at the terms in the product:

- $p(\boldsymbol{y}_{k+1} \mid \boldsymbol{x}_{k+1})$ is obtained from the sensor model,
- $p(\boldsymbol{x}_{k+1} \mid \boldsymbol{x}_k)$ is obtained from the transition model, and
- $p(\boldsymbol{y}_{k+2:t} \mid \boldsymbol{x}_{k+1})$ can be computed recursively backwards in time.

This recursion results in linear equations resembling a Kalman filter running backwards in time.

Thus, in the setting of Kalman smoothing, both factors of Equation (3.10) can be computed efficiently: one using a (forward) Kalman filter; the other using a "backward" Kalman filter. More concretely, in time $O(t)$, we can compute the two factors for all $k \in [t]$. This approach is known as *two-filter smoothing* or the *forward-backward algorithm*.

3.2 Kalman Filters

Let us return to the setting of Kalman filters where priors and likelihoods are Gaussian. Here, we will see that the update and prediction steps can be computed in closed form.

3.2.1 Conditioning

The conditioning operation in Kalman filters is also called the Kalman update. Before introducing the general Kalman update, let us consider a simpler example:

Example 3.4: Random walk in 1d

We use the simple motion and sensor models,[3]

$$X_{t+1} \mid x_t \sim \mathcal{N}(x_t, \sigma_x^2), \tag{3.12a}$$
$$Y_t \mid x_t \sim \mathcal{N}(x_t, \sigma_y^2). \tag{3.12b}$$

Let $X_t \mid y_{1:t} \sim \mathcal{N}(\mu_t, \sigma_t^2)$ be our belief at time t. It can be shown that Bayesian filtering yields the belief $X_{t+1} \mid y_{1:t+1} \sim \mathcal{N}(\mu_{t+1}, \sigma_{t+1}^2)$ at time $t+1$ where (?)

$$\mu_{t+1} \doteq \frac{\sigma_y^2 \mu_t + (\sigma_t^2 + \sigma_x^2) y_{t+1}}{\sigma_t^2 + \sigma_x^2 + \sigma_y^2}, \quad \sigma_{t+1}^2 \doteq \frac{(\sigma_t^2 + \sigma_x^2)\sigma_y^2}{\sigma_t^2 + \sigma_x^2 + \sigma_y^2}. \tag{3.13}$$

Although looking intimidating at first, this update has a very natural interpretation. Let us define the following quantity,

$$\lambda \doteq \frac{\sigma_t^2 + \sigma_x^2}{\sigma_t^2 + \sigma_x^2 + \sigma_y^2} = 1 - \frac{\sigma_y^2}{\sigma_t^2 + \sigma_x^2 + \sigma_y^2} \in [0, 1]. \tag{3.14}$$

Using λ, we can write the updated mean as a convex combination of the previous mean and the observation,

$$\mu_{t+1} = (1 - \lambda)\mu_t + \lambda y_{t+1} \tag{3.15}$$
$$= \mu_t + \lambda(y_{t+1} - \mu_t). \tag{3.16}$$

Intuitively, λ is a form of "gain" that influences how much of the new information should be incorporated into the updated mean. For this reason, λ is also called *Kalman gain*.

The updated variance can similarly be rewritten,

$$\sigma_{t+1}^2 = \lambda \sigma_y^2 = (1 - \lambda)(\sigma_t^2 + \sigma_x^2). \tag{3.17}$$

In particular, observe that if $\mu_t = y_{t+1}$ (i.e., we observe our prediction), we have $\mu_{t+1} = \mu_t$ as there is no new information. Similarly, for $\sigma_y^2 \to \infty$ (i.e., we do not trust our observations), we have

$$\lambda \to 0, \quad \mu_{t+1} = \mu_t, \quad \sigma_{t+1}^2 = \sigma_t^2 + \sigma_x^2.$$

In contrast, for $\sigma_y^2 \to 0$, we have

$$\lambda \to 1, \quad \mu_{t+1} = y_{t+1}, \quad \sigma_{t+1}^2 = 0.$$

The general formulas for the *Kalman update* follow the same logic as in the above example of a one-dimensional random walk. Given the

[3] This corresponds to $\boldsymbol{F} = \boldsymbol{H} = \boldsymbol{I}$ and a drift of 0.

Problem 3.1

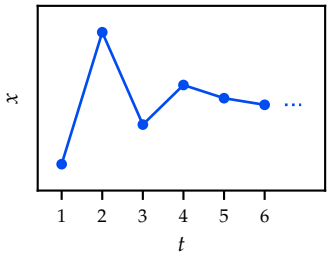


Figure 3.3: Hidden states during a random walk in one dimension.

prior belief $\mathbf{X}_t \mid \boldsymbol{y}_{1:t} \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, we have

$$\mathbf{X}_{t+1} \mid \boldsymbol{y}_{1:t+1} \sim \mathcal{N}(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) \quad \text{where} \tag{3.18a}$$

$$\boldsymbol{\mu}_{t+1} \doteq \boldsymbol{F}\boldsymbol{\mu}_t + \boldsymbol{K}_{t+1}(\boldsymbol{y}_{t+1} - \boldsymbol{H}\boldsymbol{F}\boldsymbol{\mu}_t), \tag{3.18b}$$

$$\boldsymbol{\Sigma}_{t+1} \doteq (\boldsymbol{I} - \boldsymbol{K}_{t+1}\boldsymbol{H})(\boldsymbol{F}\boldsymbol{\Sigma}_t\boldsymbol{F}^\top + \boldsymbol{\Sigma}_x). \tag{3.18c}$$

Hereby, $\boldsymbol{K}_{t+1}$ is the *Kalman gain*,

$$\boldsymbol{K}_{t+1} \doteq (\boldsymbol{F}\boldsymbol{\Sigma}_t\boldsymbol{F}^\top + \boldsymbol{\Sigma}_x)\boldsymbol{H}^\top(\boldsymbol{H}(\boldsymbol{F}\boldsymbol{\Sigma}_t\boldsymbol{F}^\top + \boldsymbol{\Sigma}_x)\boldsymbol{H}^\top + \boldsymbol{\Sigma}_y)^{-1} \in \mathbb{R}^{d\times m}. \tag{3.18d}$$

Note that $\boldsymbol{\Sigma}_t$ and $\boldsymbol{K}_t$ can be computed offline as they are independent of the observation $\boldsymbol{y}_{t+1}$. $\boldsymbol{F}\boldsymbol{\mu}_t$ represents the expected state at time $t+1$, and hence, $\boldsymbol{H}\boldsymbol{F}\boldsymbol{\mu}_t$ corresponds to the expected observation. Therefore, the term $\boldsymbol{y}_{t+1} - \boldsymbol{H}\boldsymbol{F}\boldsymbol{\mu}_t$ measures the error in the predicted observation and the Kalman gain $\boldsymbol{K}_{t+1}$ appears as a measure of relevance of the new observation compared to the prediction.

Example 3.5: Bayesian linear regression as a Kalman filter

Even though they arise from a rather different setting, it turns out that Kalman filters are a generalization of Bayesian linear regression! To see this, recall the online Bayesian linear regression algorithm from Section 2.1.3. Observe that by keeping attempting to estimate the (hidden) weights $\boldsymbol{w}^\star$ from sequential noisy observations y_t, this algorithm performs Bayesian filtering! Moreover, we have used a Gaussian prior and likelihood. This is precisely the setting of a Kalman filter!

Concretely, we are estimating the constant (i.e., $\boldsymbol{F} = \boldsymbol{I}$, $\boldsymbol{\varepsilon} = \boldsymbol{0}$) hidden state $\boldsymbol{x}_t = \boldsymbol{w}^{(t)}$ with prior $\boldsymbol{w}^{(0)} \sim \mathcal{N}(\boldsymbol{0}, \sigma_\mathrm{p}^2\boldsymbol{I})$.

Our sensor model is time-dependent, since in each iteration we observe a different input $\boldsymbol{x}_t$. Furthermore, we only observe a scalar-valued label y_t.[4] Formally, our sensor model is characterized by $\boldsymbol{h}_t = \boldsymbol{x}_t^\top$ and noise $\eta_t = \varepsilon_t$ with $\varepsilon_t \sim \mathcal{N}(0, \sigma_\mathrm{n}^2)$.

You will show in (?) that the Kalman update (3.18) is the online equivalent to computing the posterior of the weights in Bayesian linear regression.

[4] That is, $m = 1$ in our general Kalman filter formulation from above.

Problem 3.2

3.2.2 *Predicting*

Using now that the marginal posterior of $\mathbf{X}_t$ is a Gaussian due to the closedness properties of Gaussians, we have

$$\mathbf{X}_{t+1} \mid \boldsymbol{y}_{1:t} \sim \mathcal{N}(\hat{\boldsymbol{\mu}}_{t+1}, \hat{\boldsymbol{\Sigma}}_{t+1}), \tag{3.19}$$

and it suffices to compute the prediction mean $\hat{\boldsymbol{\mu}}_{t+1}$ and covariance matrix $\hat{\boldsymbol{\Sigma}}_{t+1}$.

For the mean,

$$\begin{aligned}
\hat{\boldsymbol{\mu}}_{t+1} &= \mathbb{E}[\boldsymbol{x}_{t+1} \mid \boldsymbol{y}_{1:t}] \\
&= \mathbb{E}[\boldsymbol{F}\boldsymbol{x}_t + \boldsymbol{\varepsilon}_t \mid \boldsymbol{y}_{1:t}] && \text{using the motion model (3.2)} \\
&= \boldsymbol{F}\mathbb{E}[\boldsymbol{x}_t \mid \boldsymbol{y}_{1:t}] && \text{using linearity of expectation (1.20) and } \mathbb{E}[\boldsymbol{\varepsilon}_t] = \mathbf{0} \\
&= \boldsymbol{F}\boldsymbol{\mu}_t. && \text{using the mean of the Kalman update} \qquad (3.20)
\end{aligned}$$

For the covariance matrix,

$$\begin{aligned}
\hat{\boldsymbol{\Sigma}}_{t+1} &= \mathbb{E}\left[(\boldsymbol{x}_{t+1} - \hat{\boldsymbol{\mu}}_{t+1})(\boldsymbol{x}_{t+1} - \hat{\boldsymbol{\mu}}_{t+1})^\top \;\middle|\; \boldsymbol{y}_{1:t}\right] && \text{using the definition of the covariance matrix (1.36)} \\
&= \boldsymbol{F}\mathbb{E}\left[(\boldsymbol{x}_t - \boldsymbol{\mu}_t)(\boldsymbol{x}_t - \boldsymbol{\mu}_t)^\top \;\middle|\; \boldsymbol{y}_{1:t}\right]\boldsymbol{F}^\top + \mathbb{E}\left[\boldsymbol{\varepsilon}_t\boldsymbol{\varepsilon}_t^\top\right] && \text{using (3.20), the motion model (3.2) and that } \boldsymbol{\varepsilon}_t \text{ is independent of the observations} \\
&= \boldsymbol{F}\boldsymbol{\Sigma}_t\boldsymbol{F}^\top + \boldsymbol{\Sigma}_x. && \qquad (3.21)
\end{aligned}$$

Optional Readings

Kalman filters and related models are often called *temporal models*. For a broader look at such models, read chapter 15 of "Artificial intelligence: a modern approach" (Russell and Norvig, 2002).

Discussion

In this chapter, we have introduced Kalman filters as a special case of probabilistic filtering where probabilistic inference can be performed in closed form. Similarly to Bayesian linear regression, probabilistic inference is tractable due to assuming Gaussian priors and likelihoods. Indeed, learning linear models and Kalman filters are very closely related as seen in Example 3.5, and we will further explore this relationship in Problem 4.3. We will refer back to filtering in the second part of this manuscript when we discuss sequential decision-making with partial observability of the state space. Next, we return to the storyline on "learning" using exact probabilistic inference.

Problems

3.1. Kalman update.

Derive the predictive distribution $X_{t+1} \mid y_{1:t+1}$ (3.13) of the Kalman filter described in the above example using your knowledge about multivariate Gaussians from Section 1.2.3.
Hint: First compute the predictive distribution $X_{t+1} \mid y_{1:t}$.

3.2. Bayesian linear regression as a Kalman filter.

Recall the specific Kalman filter from Example 3.5. With this model the Kalman update (3.18) simplifies to

$$\boldsymbol{k}_t = \frac{\boldsymbol{\Sigma}_{t-1}\boldsymbol{x}_t}{\boldsymbol{x}_t^\top \boldsymbol{\Sigma}_{t-1}\boldsymbol{x}_t + \sigma_\text{n}^2}, \tag{3.22a}$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \boldsymbol{k}_t(y_t - \boldsymbol{x}_t^\top \boldsymbol{\mu}_{t-1}), \tag{3.22b}$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t-1} - \boldsymbol{k}_t \boldsymbol{x}_t^\top \boldsymbol{\Sigma}_{t-1}, \tag{3.22c}$$

with $\boldsymbol{\mu}_0 = \mathbf{0}$ and $\boldsymbol{\Sigma}_0 = \sigma_\text{p}^2 \boldsymbol{I}$. Note that the Kalman gain $\boldsymbol{k}_t$ is a vector in $\mathbb{R}^d$. We assume $\sigma_\text{n}^2 = \sigma_\text{p}^2 = 1$ for simplicity.

Prove by induction that the $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ produced by the Kalman update are equivalent to $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ from the posterior of Bayesian linear regression (2.10) given $\boldsymbol{x}_{1:t}, y_{1:t}$. You may use that $\boldsymbol{\Sigma}_t^{-1}\boldsymbol{k}_t = \boldsymbol{x}_t$.

Hint: In the inductive step, first prove the equivalence of $\boldsymbol{\Sigma}_t$ and then expand $\boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t$ to prove the equivalence of $\boldsymbol{\mu}_t$.

3.3. Parameter estimation using Kalman filters.

Suppose that we want to estimate the value of an unknown constant π using uncorrelated measurements

$$y_t = \pi + \eta_t, \quad \eta_t \sim \mathcal{N}(0, \sigma_y^2).$$

1. How can this problem be formulated as a Kalman filter? Compute closed form expressions for the Kalman gain and the variance of the estimation error σ_t^2 in terms of t, σ_y^2, and σ_0^2.
2. What is the Kalman filter when $t \to \infty$?
3. Suppose that one has no prior assumptions on π, meaning that $\mu_0 = 0$ and $\sigma_0^2 \to \infty$. Which well-known estimator does the Kalman filter reduce to in this case?

4

Gaussian Processes

Let us remember our first attempt from Chapter 2 at scaling up Bayesian linear regression to nonlinear functions. We saw that we can model nonlinear functions by transforming the input space to a suitable higher-dimensional feature space, but found that this approach scales poorly if we require a large number of features. We then found something remarkable: by simply changing our perspective from a weight-space view to a function-space view, we could implement Bayesian linear regression without ever needing to compute the features explicitly. Under the function-space view, the key object describing the class of functions we can model is not the features $\boldsymbol{\phi}(\boldsymbol{x})$, but instead the kernel function which only implicitly defines a feature space. Our key observation in this chapter is that we can therefore stop reasoning about feature spaces, and instead directly work with kernel functions that describe "reasonable" classes of functions.

We are still concerned with the problem of estimating the value of a function $f : \mathcal{X} \to \mathbb{R}$ at arbitrary points $\boldsymbol{x}^\star \in \mathcal{X}$ given training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$, where the labels are assumed to be corrupted by homoscedastic Gaussian noise with variance σ_{n}^2,

$$y_i = f(\boldsymbol{x}_i) + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma_{\mathrm{n}}^2).$$

As in Chapter 2 on Bayesian linear regression, we denote by $\boldsymbol{X}$ the design matrix (collection of training inputs) and by $\boldsymbol{y}$ the vector of training labels. We will represent the unknown function value at a point $\boldsymbol{x} \in \mathcal{X}$ by the random variable $f_{\boldsymbol{x}} \doteq f(\boldsymbol{x})$. The collection of these random variables is then called a Gaussian process if any finite subset of them is jointly Gaussian:

Definition 4.1 (Gaussian process, GP)**.** A *Gaussian process* is an infinite set of random variables such that any finite number of them are jointly Gaussian and such that they are consistent under marginalization.[1]

[1] That is, if you take a joint distribution for n variables and marginalize out one of them, you should recover the joint distribution for the remaining $n-1$ variables.

The fact that with a Gaussian process, any finite subset of the random variables is jointly Gaussian is the key property allowing us to perform exact probabilistic inference. Intuitively, a Gaussian process can be interpreted as a normal distribution over functions — and is therefore often called an "infinite-dimensional Gaussian".

A Gaussian process is characterized by a *mean function* $\mu : \mathcal{X} \to \mathbb{R}$ and a *covariance function* (or *kernel function*) $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that for any set of points $A \doteq \{x_1, \dots, x_m\} \subseteq \mathcal{X}$, we have

$$f_A \doteq [f_{x_1} \ \cdots \ f_{x_m}]^\top \sim \mathcal{N}(\boldsymbol{\mu}_A, \boldsymbol{K}_{AA}) \tag{4.1}$$

where

$$\boldsymbol{\mu}_A \doteq \begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_m) \end{bmatrix}, \quad \boldsymbol{K}_{AA} \doteq \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_m) \\ \vdots & \ddots & \vdots \\ k(x_m, x_1) & \cdots & k(x_m, x_m) \end{bmatrix}. \tag{4.2}$$

We write $f \sim \mathcal{GP}(\mu, k)$. In particular, given a mean function, covariance function, and using the homoscedastic noise assumption,

$$y^\star \mid x^\star \sim \mathcal{N}(\mu(x^\star), k(x^\star, x^\star) + \sigma_{\text{n}}^2). \tag{4.3}$$

Commonly, for notational simplicity, the mean function is taken to be zero. Note that for a fixed mean this is not a restriction, as we can simply apply the zero-mean Gaussian process to the difference between the mean and the observations.[2]

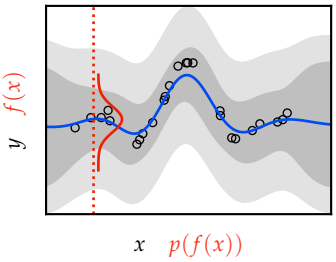


Figure 4.1: A Gaussian process can be interpreted as an infinite-dimensional Gaussian over functions. At any location x in the domain, this yields a distribution over values $f(x)$ shown in red. The blue line corresponds to the MAP estimate (i.e., mean function of the Gaussian process), the dark gray region corresponds to the epistemic uncertainty and the light gray region denotes the additional aleatoric uncertainty.

[2] For alternative ways of representing a mean function, refer to section 2.7 of "Gaussian processes for machine learning" (Williams and Rasmussen, 2006).

4.1 Learning and Inference

First, let us look at learning and inference in the context of Gaussian processes. With slight abuse of our previous notation, let us denote the set of observed points by $A \doteq \{x_1, \dots, x_n\}$. Given a prior $f \sim \mathcal{GP}(\mu, k)$ and the noisy observations $y_i = f(x_i) + \varepsilon_i$ with $\varepsilon_i \sim \mathcal{N}(0, \sigma_{\text{n}}^2)$, we can then write the joint distribution of the observations $y_{1:n}$ and the noise-free prediction $f^\star$ at a test point $x^\star$ as

$$\begin{bmatrix} \boldsymbol{y} \\ f^\star \end{bmatrix} \mid x^\star, \boldsymbol{X} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{K}}), \quad \text{where} \tag{4.4}$$

$$\tilde{\boldsymbol{\mu}} \doteq \begin{bmatrix} \boldsymbol{\mu}_A \\ \mu(x^\star) \end{bmatrix}, \quad \tilde{\boldsymbol{K}} \doteq \begin{bmatrix} \boldsymbol{K}_{AA} + \sigma_{\text{n}}^2 \boldsymbol{I} & \boldsymbol{k}_{x^\star, A} \\ \boldsymbol{k}_{x^\star, A}^\top & k(x^\star, x^\star) \end{bmatrix}, \quad \boldsymbol{k}_{x, A} \doteq \begin{bmatrix} k(x, x_1) \\ \vdots \\ k(x, x_n) \end{bmatrix}. \tag{4.5}$$

Deriving the conditional distribution using (1.53), we obtain that the Gaussian process posterior is given by

$$f \mid x_{1:n}, y_{1:n} \sim \mathcal{GP}(\mu', k'), \quad \text{where} \tag{4.6}$$

$$\mu'(\boldsymbol{x}) \doteq \mu(\boldsymbol{x}) + \boldsymbol{k}_{\boldsymbol{x},A}^\top (\boldsymbol{K}_{AA} + \sigma_\mathrm{n}^2 \boldsymbol{I})^{-1} (\boldsymbol{y}_A - \boldsymbol{\mu}_A), \tag{4.7}$$

$$k'(\boldsymbol{x}, \boldsymbol{x}') \doteq k(\boldsymbol{x}, \boldsymbol{x}') - \boldsymbol{k}_{\boldsymbol{x},A}^\top (\boldsymbol{K}_{AA} + \sigma_\mathrm{n}^2 \boldsymbol{I})^{-1} \boldsymbol{k}_{\boldsymbol{x}',A}. \tag{4.8}$$

Observe that analogously to Bayesian linear regression, the posterior covariance can only decrease when conditioning on additional data, and is independent of the observations y_i.

We already studied inference in the function-space view of Bayesian linear regression, but did not make the predictive posterior explicit. Using Equation (4.6), the predictive posterior at $\boldsymbol{x}^\star$ is simply

$$f^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n} \sim \mathcal{N}(\mu'(\boldsymbol{x}^\star), k'(\boldsymbol{x}^\star, \boldsymbol{x}^\star)). \tag{4.9}$$

4.2 Sampling

Often, we are not interested in the full predictive posterior distribution, but merely want to obtain samples of our Gaussian process model. We will briefly examine two approaches.

1. For the first approach, consider a discretized subset of points

   $$\boldsymbol{f} \doteq [f_1, \ldots, f_n]$$

   that we want to sample.[3] Note that $\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{K})$. We have already seen in Equation (1.54) that

   $$\boldsymbol{f} = \boldsymbol{K}^{1/2} \boldsymbol{\varepsilon} + \boldsymbol{\mu} \tag{4.10}$$

   where $\boldsymbol{K}^{1/2}$ is the square root of $\boldsymbol{K}$ and $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ is standard Gaussian noise.[4] However, computing the square root of $\boldsymbol{K}$ takes $O(n^3)$ time.

2. For the second approach, recall the product rule (1.11),

   $$p(f_1, \ldots, f_n) = \prod_{i=1}^{n} p(f_i \mid f_{1:i-1}).$$

   That is the joint distribution factorizes neatly into a product where each factor only depends on the "outcomes" of preceding factors. We can therefore obtain samples one-by-one, each time conditioning on one more observation:

   $$\begin{aligned} f_1 &\sim p(f_1) \\ f_2 &\sim p(f_2 \mid f_1) \\ f_3 &\sim p(f_3 \mid f_1, f_2) \\ &\vdots \end{aligned} \tag{4.11}$$

   This general approach is known as *forward sampling*. Due to the matrix inverse in the formula of the GP posterior (4.6), this approach also takes $O(n^3)$ time.

We will discuss more efficient approximate sampling methods in Section 4.5.

[3] For example, if we want to render the function, the length of this vector could be guided by the screen resolution.

[4] We discuss square roots of matrices in Appendix A.2.

4.3 Kernel Functions

We have seen that kernel functions are the key object describing the class of functions a Gaussian process can model. Depending on the kernel function, the "shape" of functions that are realized from a Gaussian process varies greatly. Let us recap briefly from Section 2.4 what a kernel function is:

Definition 4.2 (Kernel function)**.** A *kernel function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ satisfies

- $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$ for any $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$ (symmetry), and
- $\boldsymbol{K}_{AA}$ is positive semi-definite for any $A \subseteq \mathcal{X}$.

The two defining conditions ensure that for any $A \subseteq \mathcal{X}$, $\boldsymbol{K}_{AA}$ is a valid covariance matrix. We say that a kernel function is *positive definite* if $\boldsymbol{K}_{AA}$ is positive definite for any $A \subseteq \mathcal{X}$.

Intuitively, the kernel function evaluated at locations $\boldsymbol{x}$ and $\boldsymbol{x}'$ describes how $f(\boldsymbol{x})$ and $f(\boldsymbol{x}')$ are related, which we can express formally as

$$k(\boldsymbol{x}, \boldsymbol{x}') = \operatorname{Cov}\big[f(\boldsymbol{x}), f(\boldsymbol{x}')\big]. \tag{4.12}$$

If $\boldsymbol{x}$ and $\boldsymbol{x}'$ are "close", then $f(\boldsymbol{x})$ and $f(\boldsymbol{x}')$ are usually taken to be positively correlated, encoding a "smooth" function.

In the following, we will discuss some of the most common kernel functions, how they can be combined to create "new" kernels, and how we can characterize the class of functions they can model.

4.3.1 Common Kernels

First, we look into some of the most commonly used kernels. Often an additional factor σ^2 (*output scale*) is added, which we assume here to be 1 for simplicity.

1. The *linear kernel* is defined as

$$k(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\phi}) \doteq \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}') \tag{4.13}$$

   where $\boldsymbol{\phi}$ is a nonlinear transformation as introduced in Section 2.3 or the identity.

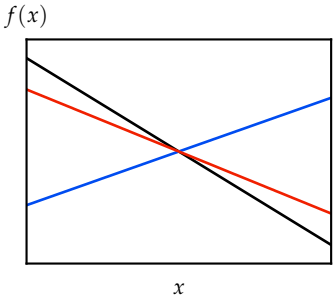


Figure 4.2: Functions sampled according to a Gaussian process with a linear kernel and $\phi = \mathrm{id}$.

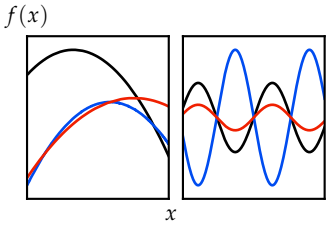


Figure 4.3: Functions sampled according to a Gaussian process with a linear kernel and $\boldsymbol{\phi}(x) = [1, x, x^2]$ (left) and $\phi(x) = \sin(x)$ (right).

Remark 4.3: GPs with linear kernel and BLR

A Gaussian process with a linear kernel is equivalent to Bayesian linear regression. This follows directly from the function-space view of Bayesian linear regression (see Section 2.4) and comparing the derived kernel function (2.20) with the definition of the linear kernel (4.13).

2. The *Gaussian kernel* (also known as *squared exponential kernel* or *radial basis function (RBF) kernel*) is defined as

$$k(\boldsymbol{x}, \boldsymbol{x}'; h) \doteq \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2h^2}\right) \tag{4.14}$$

where h is its *length scale*. The larger the length scale h, the smoother the resulting functions.[5] Furthermore, it turns out that the feature space (think back to Section 2.4!) corresponding to the Gaussian kernel is "infinitely dimensional", as you will show in (?). So the Gaussian kernel already encodes a function class that we were not able to model under the weight-space view of Bayesian linear regression.

[5] As the length scale is increased, the exponent of the exponential increases, resulting in a higher dependency between locations.

Problem 4.1

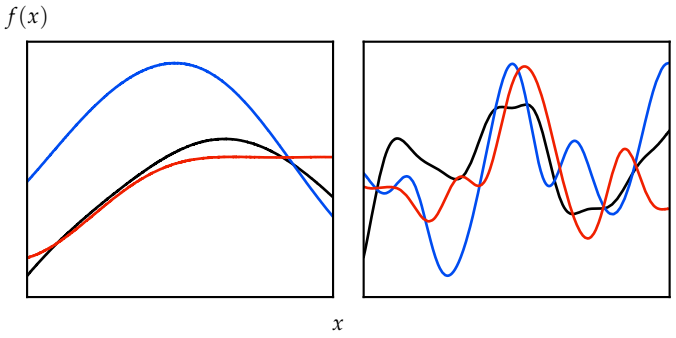


Figure 4.4: Functions sampled according to a Gaussian process with a Gaussian kernel and length scales $h = 5$ (left) and $h = 1$ (right).

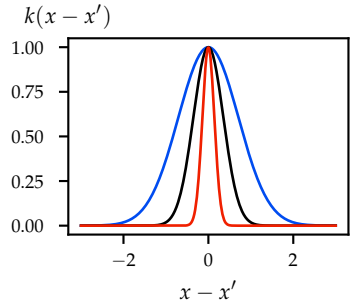


Figure 4.5: Gaussian kernel with length scales $h = 1$, $h = 0.5$, and $h = 0.2$.

3. The *Laplace kernel* (also known as *exponential kernel*) is defined as

$$k(\boldsymbol{x}, \boldsymbol{x}'; h) \doteq \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2}{h}\right). \tag{4.15}$$

As can be seen in Figure 4.7, samples from a GP with Laplace kernel are non-smooth as opposed to the samples from a GP with Gaussian kernel.

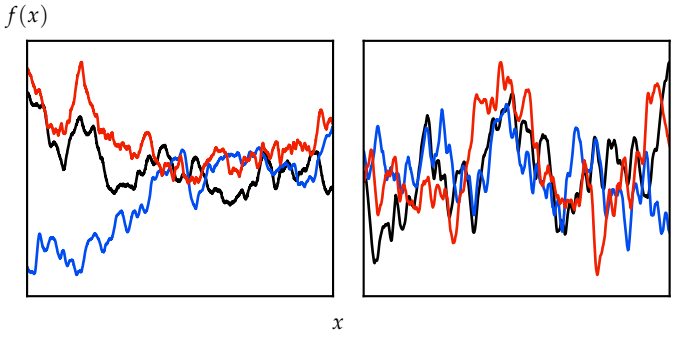


Figure 4.7: Functions sampled according to a Gaussian process with a Laplace kernel and length scales $h = 10\,000$ (left) and $h = 10$ (right).

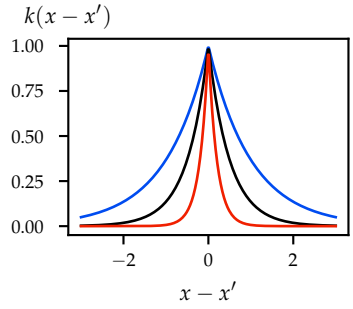


Figure 4.6: Laplace kernel with length scales $h = 1$, $h = 0.5$, and $h = 0.2$.

4. The *Matérn kernel* trades the smoothness of the Gaussian and the Laplace kernels. As such, it is frequently used in practice to model

"real world" functions that are relatively smooth. It is defined as

$$k(\boldsymbol{x}, \boldsymbol{x}'; \nu, h) \doteq \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\, \|\boldsymbol{x} - \boldsymbol{x}'\|_2}{h} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}\, \|\boldsymbol{x} - \boldsymbol{x}'\|_2}{h} \right) \tag{4.16}$$

where Γ is the Gamma function, K_ν the modified Bessel function of the second kind, and h a length scale parameter. For $\nu = 1/2$, the Matérn kernel is equivalent to the Laplace kernel. For $\nu \to \infty$, the Matérn kernel is equivalent to the Gaussian kernel. The resulting functions are $\lceil \nu \rceil - 1$ times mean square differentiable.[6] In particular, GPs with a Gaussian kernel are infinitely many times mean square differentiable whereas GPs with a Laplace kernel are mean square continuous but not mean square differentiable.

[6] Refer to Remark A.12 for the definitions of mean square continuity and differentiability.

4.3.2 *Composing Kernels*

Given two kernels $k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, they can be composed to obtain a new kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ in the following ways:

- $k(\boldsymbol{x}, \boldsymbol{x}') \doteq k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}')$,
- $k(\boldsymbol{x}, \boldsymbol{x}') \doteq k_1(\boldsymbol{x}, \boldsymbol{x}') \cdot k_2(\boldsymbol{x}, \boldsymbol{x}')$,
- $k(\boldsymbol{x}, \boldsymbol{x}') \doteq c \cdot k_1(\boldsymbol{x}, \boldsymbol{x}')$ for any $c > 0$,
- $k(\boldsymbol{x}, \boldsymbol{x}') \doteq f(k_1(\boldsymbol{x}, \boldsymbol{x}'))$ for any polynomial f with positive coefficients or $f = \exp$.

For example, the additive structure of a function $f(\boldsymbol{x}) \doteq f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})$ can be easily encoded in GP models. Suppose that $f_1 \sim \mathcal{GP}(\mu_1, k_1)$ and $f_2 \sim \mathcal{GP}(\mu_2, k_2)$, then the distribution of the sum of those two functions $f = f_1 + f_2 \sim \mathcal{GP}(\mu_1 + \mu_2, k_1 + k_2)$ is another GP.[7]

[7] We use $f \doteq f_1 + f_2$ to denote the function $f(\cdot) = f_1(\cdot) + f_2(\cdot)$.

Whereas the addition of two kernels k_1 and k_2 can be thought of as an *OR* operation (i.e., the kernel has high value if either k_1 or k_2 have high value), the multiplication of k_1 and k_2 can be thought of as an *AND* operation (i.e., the kernel has high value if both k_1 and k_2 have high value). For example, the product of two linear kernels results in functions which are quadratic.

As mentioned previously, the constant c of a scaled kernel function $k'(\boldsymbol{x}, \boldsymbol{x}') \doteq c \cdot k(\boldsymbol{x}, \boldsymbol{x}')$ is generally called the *output scale* of a kernel, and it scales the variance $\mathrm{Var}[f(\boldsymbol{x})] = c \cdot k(\boldsymbol{x}, \boldsymbol{x})$ of the predictions $f(\boldsymbol{x})$ from $\mathcal{GP}(\mu, k')$.

Optional Readings

For a broader introduction to how kernels can be used and combined to model certain classes of functions, read

- chapter 2 of "Automatic model construction with Gaussian processes" (Duvenaud, 2014) also known as the "kernel cookbook",
- chapter 4 of "Gaussian processes for machine learning" (Williams and Rasmussen, 2006).

4.3.3 *Stationarity and Isotropy*

Kernel functions are commonly classified according to two properties:

Definition 4.4 (Stationarity and isotropy)**.** A kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called

- *stationary* (or *shift-invariant*) if there exists a function $\tilde{k}$ such that $\tilde{k}(\boldsymbol{x} - \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}')$, and
- *isotropic* if there exists a function $\tilde{k}$ such that $\tilde{k}(\|\boldsymbol{x} - \boldsymbol{x}'\|) = k(\boldsymbol{x}, \boldsymbol{x}')$ with $\|\cdot\|$ any norm.

Note that stationarity is a necessary condition for isotropy. In other words, isotropy implies stationarity.

Example 4.5: Stationarity and isotropy of kernels

	stationary	isotropic
linear kernel	no	no
Gaussian kernel	yes	yes
$k(\boldsymbol{x}, \boldsymbol{x}') \doteq \exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|_{\boldsymbol{M}}^2)$ where $\boldsymbol{M}$ is positive semi-definite	yes	no

$\|\cdot\|_{\boldsymbol{M}}$ denotes the Mahalanobis norm induced by matrix $\boldsymbol{M}$

For $\boldsymbol{x}' = \boldsymbol{x}$, stationarity implies that the kernel must only depend on $\boldsymbol{0}$. In other words, a stationary kernel must depend on relative locations only. This is clearly not the case for the linear kernel, which depends on the absolute locations of $\boldsymbol{x}$ and $\boldsymbol{x}'$. Therefore, the linear kernel cannot be isotropic either.

For the Gaussian kernel, isotropy follows immediately from its definition.

The last kernel is clearly stationary by definition, but not isotropic for general matrices $\boldsymbol{M}$. Note that for $\boldsymbol{M} = \boldsymbol{I}$ it is indeed isotropic.

Stationarity encodes the idea that relative location matters more than absolute location: the process "looks the same" no matter where we shift it in the input space. This is often appropriate when we believe the same statistical behavior holds across the entire domain (e.g., no region is special). Isotropy goes one step further by requiring that

the kernel depends only on the distance between points, so that all directions in the space are treated equally. In other words, there is no preferred orientation or axis. This is especially useful in settings where we expect uniform behavior in every direction (as with the Gaussian kernel). Such kernels are simpler to specify and interpret since we only need a single "scale" (like a length scale) rather than multiple parameters or directions.

4.3.4 *Reproducing Kernel Hilbert Spaces*

We can characterize the precise class of functions that can be modeled by a Gaussian process with a given kernel function. This corresponding function space is called a *reproducing kernel Hilbert space* (RKHS), and we will discuss it briefly in this section.

Recall that Gaussian processes keep track of a posterior distribution $f \mid \boldsymbol{x}_{1:n}, y_{1:n}$ over functions. We will in fact show later that the corresponding MAP estimate $\hat{f}$ corresponds to the solution to a regularized optimization problem in the RKHS space of functions. This duality is similar to the duality between the MAP estimate of Bayesian linear regression and ridge regression we observed in Chapter 2. So what is the reproducing kernel Hilbert space of a kernel function k?

Definition 4.6 (Reproducing kernel Hilbert space, RKHS)**.** Given a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, its corresponding *reproducing kernel Hilbert space* is the space of functions f defined as

$$\mathcal{H}_k(\mathcal{X}) \doteq \left\{ f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \cdot) : n \in \mathbb{N}, \boldsymbol{x}_i \in \mathcal{X}, \alpha_i \in \mathbb{R} \right\}. \tag{4.17}$$

The inner product of the RKHS is defined as

$$\langle f, g \rangle_k \doteq \sum_{i=1}^{n} \sum_{j=1}^{n'} \alpha_i \alpha'_j k(\boldsymbol{x}_i, \boldsymbol{x}'_j), \tag{4.18}$$

where $g(\cdot) = \sum_{j=1}^{n'} \alpha'_j k(\boldsymbol{x}'_j, \cdot)$, and induces the norm $\|f\|_k = \sqrt{\langle f, f \rangle_k}$. You can think of the norm as measuring the "smoothness" or "complexity" of f. (?) **Problem 4.4 (2)**

It is straightforward to check that for all $\boldsymbol{x} \in \mathcal{X}$, $k(\boldsymbol{x}, \cdot) \in \mathcal{H}_k(\mathcal{X})$. Moreover, the RKHS inner product $\langle \cdot, \cdot \rangle_k$ satisfies for all $\boldsymbol{x} \in \mathcal{X}$ and $f \in \mathcal{H}_k(\mathcal{X})$ that $f(\boldsymbol{x}) = \langle f(\cdot), k(\boldsymbol{x}, \cdot) \rangle_k$ which is also known as the *reproducing property* (?). That is, evaluations of RKHS functions f are inner products in $\mathcal{H}_k(\mathcal{X})$ parameterized by the "feature map" $k(\boldsymbol{x}, \cdot)$. **Problem 4.4 (1)**

The *representer theorem* (Schölkopf et al., 2001) characterizes the solution to regularized optimization problems in RKHSs:

Theorem 4.7 (Representer theorem). ? *Let k be a kernel and let $\lambda > 0$. For $f \in \mathcal{H}_k(\mathcal{X})$ and training data $\{(\boldsymbol{x}_i, f(\boldsymbol{x}_i))\}_{i=1}^n$, let $\mathcal{L}(f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)) \in \mathbb{R} \cup \{\infty\}$ denote any loss function which depends on f only through its evaluation at the training points. Then, any minimizer* **Problem 4.5**

$$\hat{f} \in \operatorname*{arg\,min}_{f \in \mathcal{H}_k(\mathcal{X})} \mathcal{L}(f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)) + \lambda \|f\|_k^2 \tag{4.19}$$

admits a representation of the form

$$\hat{f}(\boldsymbol{x}) = \hat{\boldsymbol{\alpha}}^\top \boldsymbol{k}_{\boldsymbol{x}, \{\boldsymbol{x}_i\}_{i=1}^n} = \sum_{i=1}^n \hat{\alpha}_i k(\boldsymbol{x}, \boldsymbol{x}_i) \quad \text{for some } \hat{\boldsymbol{\alpha}} \in \mathbb{R}^n. \tag{4.20}$$

This statement is remarkable: the solutions to general regularized optimization problems over the generally infinite-dimensional space of functions $\mathcal{H}_k(\mathcal{X})$ can be represented as a linear combination of the kernel functions evaluated at the training points. The representer theorem can be used to show that the MAP estimate of a Gaussian process corresponds to the solution of a regularized linear regression problem in the RKHS of the kernel function, namely, ? **Problem 4.6**

$$\hat{f} \doteq \operatorname*{arg\,min}_{f \in \mathcal{H}_k(\mathcal{X})} -\log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, f) + \frac{1}{2} \|f\|_k^2 . \tag{4.21}$$

Here, the first term corresponds to the likelihood, measuring the "quality of fit". The regularization term limits the "complexity" of $\hat{f}$. Regularization is necessary to prevent overfitting since in an expressive RKHSs, there may be many functions that interpolate the training data perfectly. This shows the close link between Gaussian process regression and Bayesian linear regression, with the kernel function k generalizing the inner product of feature maps to feature spaces of possibly "infinite dimensionality". Because solutions can be represented as linear combinations of kernel evaluations at the training points, Gaussian processes remain computationally tractable even though they can model functions over "infinite-dimensional" feature spaces.

4.4 Model Selection

We have not yet discussed how to pick the hyperparameters $\boldsymbol{\theta}$ (e.g., parameters of kernels). A common technique in supervised learning is to select hyperparameters $\boldsymbol{\theta}$, such that the resulting function estimate $\hat{f}_{\boldsymbol{\theta}}$ leads to the most accurate predictions on hold-out validation data. After reviewing this approach, we contrast it with a probabilistic approach to model selection, which avoids using point estimates of $\hat{f}_{\boldsymbol{\theta}}$ and rather utilizes the full posterior.

4.4.1 Optimizing Validation Set Performance

A common approach to model selection is to split our data $\mathcal{D}$ into separate training set $\mathcal{D}^{\text{train}} \doteq \{(\boldsymbol{x}_i^{\text{train}}, y_i^{\text{train}})\}_{i=1}^n$ and validation sets $\mathcal{D}^{\text{val}} \doteq \{(\boldsymbol{x}_i^{\text{val}}, y_i^{\text{val}})\}_{i=1}^m$. We then optimize the model for a parameter candidate $\boldsymbol{\theta}_j$ using the training set. This is usually done by picking a point estimate (like the MAP estimate),

$$\hat{f}_j \doteq \arg\max_f p(f \mid \boldsymbol{x}_{1:n}^{\text{train}}, y_{1:n}^{\text{train}}). \tag{4.22}$$

Then, we score $\boldsymbol{\theta}_j$ according to the performance of $\hat{f}_j$ on the validation set,

$$\hat{\boldsymbol{\theta}} \doteq \arg\max_{\boldsymbol{\theta}_j} p(y_{1:m}^{\text{val}} \mid \boldsymbol{x}_{1:m}^{\text{val}}, \hat{f}_j). \tag{4.23}$$

This ensures that $\hat{f}_j$ does not depend on $\mathcal{D}^{\text{val}}$.

Remark 4.8: Approximating population risk

Why is it useful to separate the data into a training and a validation set? Recall from Appendix A.3.5 that minimizing the empirical risk without separating training and validation data may lead to overfitting as both the loss and $\hat{f}_j$ depend on the same data $\mathcal{D}$. In contrast, using independent training and validation sets, $\hat{f}_j$ does not depend on $\mathcal{D}^{\text{val}}$, and we have that

$$\frac{1}{m}\sum_{i=1}^m \ell(y_i^{\text{val}} \mid \boldsymbol{x}_i^{\text{val}}, \hat{f}_j) \approx \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}\Big[\ell(y \mid \boldsymbol{x}, \hat{f}_j)\Big], \tag{4.24}$$

using Monte Carlo sampling.[8] In words, for reasonably large m, minimizing the empirical risk as we do in Equation (4.23) approximates minimizing the population risk.

[8] We generally assume $\mathcal{D} \overset{\text{iid}}{\sim} \mathcal{P}$, in particular, we assume that the individual samples of the data are i.i.d.. Recall that in this setting, Hoeffding's inequality (A.41) can be used to gauge how large m should be.

While this approach often is quite effective at preventing overfitting as compared to using the same data for training and picking $\hat{\boldsymbol{\theta}}$, it still collapses the uncertainty in f into a point estimate. Can we do better?

4.4.2 Maximizing the Marginal Likelihood

We have already seen for Bayesian linear regression, that picking a point estimate loses a lot of information. Instead of optimizing the effects of $\boldsymbol{\theta}$ for a specific point estimate $\hat{f}$ of the model f, *maximizing the marginal likelihood* optimizes the effects of $\boldsymbol{\theta}$ across all realizations of f. In this approach, we obtain our hyperparameter estimate via

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} \doteq \arg\max_{\boldsymbol{\theta}} p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}) \tag{4.25}$$

using the definition of marginal likelihood in Bayes' rule (1.45)

$$= \arg\max_{\boldsymbol{\theta}} \int p(y_{1:n}, f \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})\, df \qquad \text{by conditioning on } f \text{ using the sum rule (1.7)}$$

$$= \arg\max_{\boldsymbol{\theta}} \int p(y_{1:n} \mid \boldsymbol{x}_{1:n}, f, \boldsymbol{\theta}) p(f \mid \boldsymbol{\theta})\, df. \qquad (4.26) \qquad \text{using the product rule (1.11)}$$

Remarkably, this approach typically avoids overfitting even though we do not use a separate training and validation set. The following table provides an intuitive argument for why maximizing the marginal likelihood is a good strategy.

	likelihood	prior
"underfit" model (too simple $\boldsymbol{\theta}$)	small for "almost all" f	large
"overfit" model (too complex $\boldsymbol{\theta}$)	large for "few" f small for "most" f	small
"just right"	moderate for "many" f	moderate

Table 4.1: The table gives an intuitive explanation of effects of parameter choices $\boldsymbol{\theta}$ on the marginal likelihood. Note that words in quotation marks refer to intuitive quantities, as we have infinitely many realizations of f.

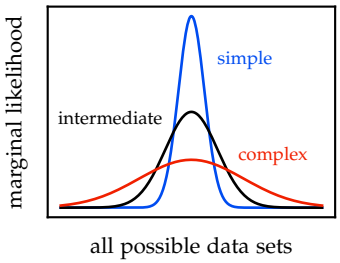


Figure 4.8: A schematic illustration of the marginal likelihood of a simple, intermediate, and complex model across all possible data sets.

For an "underfit" model, the likelihood is mostly small as the data cannot be well described, while the prior is large as there are "fewer" functions to choose from. For an "overfit" model, the likelihood is large for "some" functions (which would be picked if we were only minimizing the training error and not doing cross validation) but small for "most" functions. The prior is small, as the probability mass has to be distributed among "more" functions. Thus, in both cases, one term in the product will be small. Hence, maximizing the marginal likelihood naturally encourages trading between a large likelihood and a large prior.

In the context of Gaussian process regression, recall from Equation (4.3) that

$$y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{K}_{f,\boldsymbol{\theta}} + \sigma_{\mathrm{n}}^2 \boldsymbol{I}) \qquad (4.27)$$

where $\boldsymbol{K}_{f,\boldsymbol{\theta}}$ denotes the kernel matrix at the inputs $\boldsymbol{x}_{1:n}$ depending on the kernel function parameterized by $\boldsymbol{\theta}$. We write $\boldsymbol{K}_{y,\boldsymbol{\theta}} \doteq \boldsymbol{K}_{f,\boldsymbol{\theta}} + \sigma_{\mathrm{n}}^2 \boldsymbol{I}$. Continuing from Equation (4.25), we obtain

$$\hat{\boldsymbol{\theta}}_{\mathrm{MLE}} = \arg\max_{\boldsymbol{\theta}} \mathcal{N}(\boldsymbol{y}; \mathbf{0}, \boldsymbol{K}_{y,\boldsymbol{\theta}})$$

$$= \arg\min_{\boldsymbol{\theta}} \frac{1}{2} \boldsymbol{y}^\top \boldsymbol{K}_{y,\boldsymbol{\theta}}^{-1} \boldsymbol{y} + \frac{1}{2} \log\det(\boldsymbol{K}_{y,\boldsymbol{\theta}}) + \frac{n}{2} \log 2\pi \qquad (4.28) \qquad \text{taking the negative logarithm}$$

$$= \arg\min_{\boldsymbol{\theta}} \frac{1}{2} \boldsymbol{y}^\top \boldsymbol{K}_{y,\boldsymbol{\theta}}^{-1} \boldsymbol{y} + \frac{1}{2} \log\det(\boldsymbol{K}_{y,\boldsymbol{\theta}}) \qquad (4.29) \qquad \text{the last term is independent of } \boldsymbol{\theta}$$

The first term of the optimization objective describes the "goodness of fit" (i.e., the "alignment" of $\boldsymbol{y}$ with $\boldsymbol{K}_{y,\boldsymbol{\theta}}$). The second term characterizes the "volume" of the model class. Thus, this optimization naturally trades the aforementioned objectives.

Marginal likelihood maximization is an empirical Bayes method. Often it is simply referred to as *empirical Bayes*. It also has the nice property that the gradient of its objective (the MLL loss) can be expressed in closed-from (?),

$$\frac{\partial}{\partial \theta_j} \log p(\boldsymbol{y}_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}) = \frac{1}{2} \operatorname{tr}\left((\boldsymbol{\alpha}\boldsymbol{\alpha}^\top - \boldsymbol{K}_{\boldsymbol{y},\boldsymbol{\theta}}^{-1}) \frac{\partial \boldsymbol{K}_{\boldsymbol{y},\boldsymbol{\theta}}}{\partial \theta_j} \right) \tag{4.30}$$

where $\boldsymbol{\alpha} \doteq \boldsymbol{K}_{\boldsymbol{y},\boldsymbol{\theta}}^{-1} \boldsymbol{y}$ and $\operatorname{tr}(\boldsymbol{M})$ is the trace of a matrix $\boldsymbol{M}$. This optimization problem is, in general, non-convex. Figure 4.10 gives an example of two local optima according to empirical Bayes.

Taking a step back, observe that taking a probabilistic perspective on model selection naturally led us to consider all realizations of our model f instead of using point estimates. However, we are still using point estimates for our model parameters $\boldsymbol{\theta}$. Continuing on our probabilistic adventure, we could place a prior $p(\boldsymbol{\theta})$ on them too.[9] We could use it to obtain the MAP estimate (still a point estimate!) which adds an additional regularization term

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} \doteq \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}) \tag{4.31}$$

$$= \arg\min_{\boldsymbol{\theta}} -\log p(\boldsymbol{\theta}) - \log p(\boldsymbol{y}_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}). \tag{4.32}$$

An alternative approach is to consider the full posterior distribution over parameters $\boldsymbol{\theta}$. The resulting predictive distribution is, however, intractable,

$$p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}) = \int \int p(y^\star \mid \boldsymbol{x}^\star, f) \cdot p(f \mid \boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}, \boldsymbol{\theta}) \cdot p(\boldsymbol{\theta}) \, df \, d\boldsymbol{\theta}. \tag{4.33}$$

Recall that as the mode of Gaussians coincides with their mean, the MAP estimate corresponds to the mean of the predictive posterior.

As a final note, observe that in principle, there is nothing stopping us from descending deeper in the probabilistic hierarchy. The prior on the model parameters $\boldsymbol{\theta}$ is likely to have parameters too. Ultimately, we need to break out of this hierarchy of dependencies and choose a prior.

4.5 *Approximations*

To learn a Gaussian process, we need to invert $n \times n$ matrices, hence the computational cost is $O(n^3)$. Compare this to Bayesian linear regression which allows us to learn a regression model in $O(nd^2)$ time (even online) where d is the feature dimension. It is therefore natural to look for ways of approximating a Gaussian process.

Problem 4.7

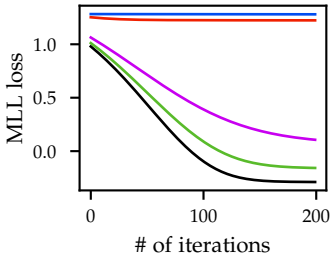


Figure 4.9: An example of model selection by maximizing the log likelihood (without hyperpriors) using a **linear**, **quadratic**, **Laplace**, **Matérn** ($\nu = 3/2$), and **Gaussian** kernel, respectively. They are used to learn the function

$$x \mapsto \frac{\sin(x)}{x} + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, 0.01)$$

using SGD with learning rate 0.1.

[9] Such a prior is called *hyperprior*.

using Bayes' rule (1.45) and then taking the negative logarithm

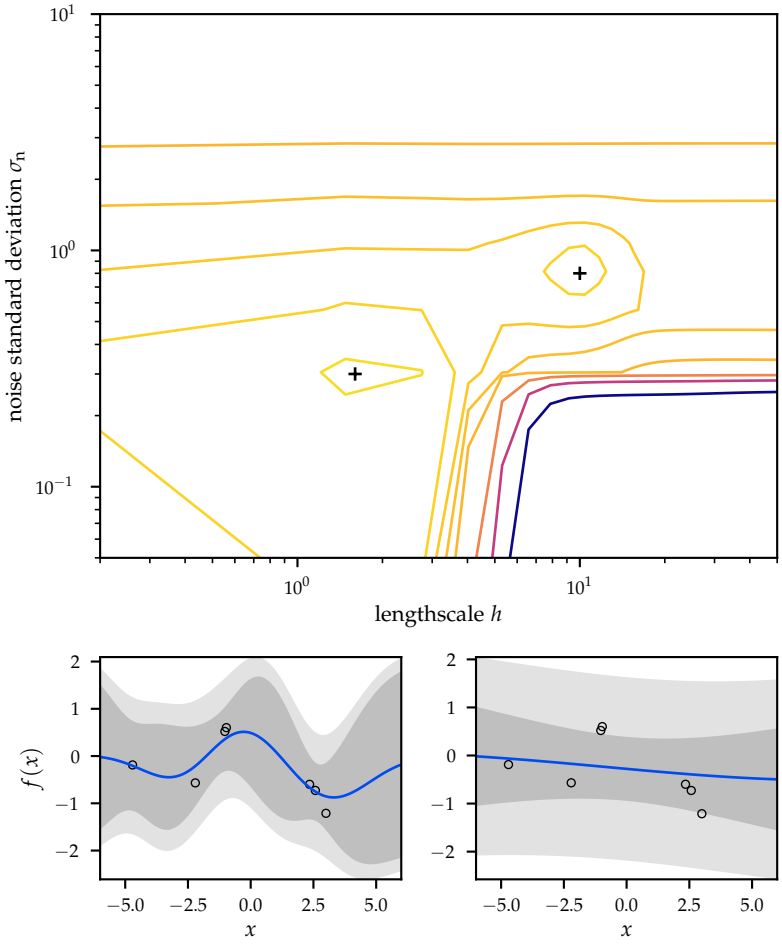


Figure 4.10: The top plot shows contour lines of an empirical Bayes with two local optima. The bottom two plots show the Gaussian processes corresponding to the two optimal models. The left model with smaller lengthscale is chosen within a more flexible class of models, while the right model explains more observations through noise. Adapted from figure 5.5 of "Gaussian processes for machine learning" (Williams and Rasmussen, 2006).

4.5.1 Local Methods

Recall that during forward sampling, we had to condition on a larger and larger number of previous samples. When sampling at a location $\boldsymbol{x}$, a very simple approximation is to only condition on those samples $\boldsymbol{x}'$ that are "close" (where $|k(\boldsymbol{x}, \boldsymbol{x}')| \geq \tau$ for some $\tau > 0$). Essentially, this method "cuts off the tails" of the kernel function k. However, τ has to be chosen carefully as if τ is chosen too large, samples become essentially independent.

This is one example of a *sparse approximation* of a Gaussian process. We will discuss more advanced sparse approximations known as "inducing point methods" in Section 4.5.3.

4.5.2 Kernel Function Approximation

Another method is to approximate the kernel function directly. The idea is to construct a "low-dimensional" feature map $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^m$ that approximates the kernel,

$$k(\boldsymbol{x}, \boldsymbol{x}') \approx \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}'). \tag{4.34}$$

Then, we can apply Bayesian linear regression, resulting in a time complexity of $O(nm^2 + m^3)$.

One example of this approach are *random Fourier features*, which we will discuss in the following.

Remark 4.9: Fourier transform

First, let us remind ourselves of Fourier transformations. The Fourier transform is a method of decomposing frequencies into their individual components.

Recall *Euler's formula* which states that for any $x \in \mathbb{R}$,

$$e^{ix} = \cos x + i \sin x \tag{4.35}$$

where i is the imaginary unit of complex numbers. The formula is illustrated in Figure 4.11. Note that $e^{-i2\pi x}$ corresponds to rotating clockwise around the unit circle in $\mathbb{R}^2$ — completing a rotation whenever $x \in \mathbb{R}$ reaches the next natural number.

We can scale x by a frequency ξ: $e^{-i2\pi\xi x}$. If $\boldsymbol{x} \in \mathbb{R}^d$, we can also scale each component j of $\boldsymbol{x}$ by a different frequency $\boldsymbol{\xi}(j)$. Multiplying a function $f : \mathbb{R}^d \to \mathbb{R}$ with the rotation around the unit circle with given frequencies $\boldsymbol{\xi}$, yields a quantity that describes the

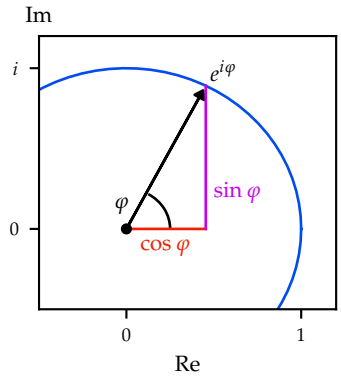


Figure 4.11: Illustration of Euler's formula. It can be seen that $e^{i\varphi}$ corresponds to a (counter-clockwise) rotation on the unit circle as φ varies from 0 to 2π.

amplitude of the frequencies $\boldsymbol{\xi}$,

$$\hat{f}(\boldsymbol{\xi}) \doteq \int_{\mathbb{R}^d} f(\boldsymbol{x}) e^{-i2\pi\boldsymbol{\xi}^\top \boldsymbol{x}} \, d\boldsymbol{x}. \tag{4.36}$$

$\hat{f}$ is called the *Fourier transform* of f. f is called the *inverse Fourier transform* of $\hat{f}$, and can be computed using

$$f(\boldsymbol{x}) = \int_{\mathbb{R}^d} \hat{f}(\boldsymbol{\xi}) e^{i2\pi\boldsymbol{\xi}^\top \boldsymbol{x}} \, d\boldsymbol{\xi}. \tag{4.37}$$

It is common to write $\boldsymbol{\omega} \doteq 2\pi\boldsymbol{\xi}$. See Figure 4.12 for an example.

Refer to "But what is the Fourier Transform? A visual introduction" (Sanderson, 2018) for a visual introduction.

Because a stationary kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ can be interpreted as a function in one variable, it has an associated Fourier transform which we denote by $p(\boldsymbol{\omega})$. That is,

$$k(\boldsymbol{x} - \boldsymbol{x}') = \int_{\mathbb{R}^d} p(\boldsymbol{\omega}) e^{i\boldsymbol{\omega}^\top (\boldsymbol{x} - \boldsymbol{x}')} \, d\boldsymbol{\omega}. \tag{4.38}$$

Fact 4.10 (Bochner's theorem)**.** *A continuous stationary kernel on $\mathbb{R}^d$ is positive definite if and only if its Fourier transform $p(\boldsymbol{\omega})$ is non-negative.*

Bochner's theorem implies that when a continuous and stationary kernel is positive definite and scaled appropriately, its Fourier transform $p(\boldsymbol{\omega})$ is a proper probability distribution. In this case, $p(\boldsymbol{\omega})$ is called the *spectral density* of the kernel k.

Remark 4.11: Eigenvalue spectrum of stationary kernels

When a kernel k is stationary (i.e., a univariate function of $\boldsymbol{x} - \boldsymbol{x}'$), its eigenfunctions (with respect to the usual Lebesgue measure) turn out to be the complex exponentials $\exp(i\boldsymbol{\omega}^\top(\boldsymbol{x} - \boldsymbol{x}'))$. In simpler terms, you can think of these exponentials as "building blocks" at different frequencies $\boldsymbol{\omega}$. The spectral density $p(\boldsymbol{\omega})$ associated with the kernel tells you how strongly each frequency contributes, i.e., how large the corresponding eigenvalue is.

A key insight of this analysis is that the rate at which these magnitudes $p(\boldsymbol{\omega})$ decay with increasing frequency $\boldsymbol{\omega}$ reveals the smoothness of the processes governed by the kernel. If a kernel allocates more "power" to high frequencies (meaning the spectral density decays slowly), the resulting processes will appear "rougher". Conversely, if high-frequency components are suppressed, the process will appear "smoother".

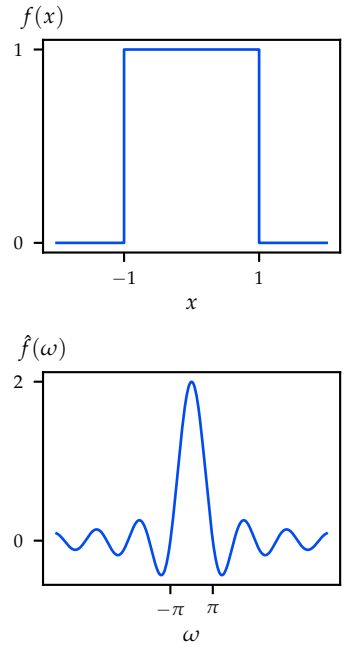


Figure 4.12: The Fourier transform of a rectangular pulse,

$$f(x) \doteq \begin{cases} 1 & x \in [-1, 1] \\ 0 & \text{otherwise,} \end{cases}$$

is given by

$$\hat{f}(\omega) = \int_{-1}^{1} e^{-i\omega x} \, dx = \frac{1}{i\omega}\left(e^{i\omega} - e^{-i\omega}\right) = \frac{2\sin(\omega)}{\omega}.$$

For an in-depth introduction to the eigenfunction analysis of kernels, refer to section 4.3 of "Gaussian processes for machine learning" (Williams and Rasmussen, 2006).

Example 4.12: Spectral density of the Gaussian kernel

The Gaussian kernel with length scale h has the spectral density

$$\begin{aligned} p(\boldsymbol{\omega}) &= \int_{\mathbb{R}^d} k(\boldsymbol{x}-\boldsymbol{x}';h) e^{-i\boldsymbol{\omega}^\top(\boldsymbol{x}-\boldsymbol{x}')}\, d(\boldsymbol{x}-\boldsymbol{x}') && \text{using the definition of the Fourier transform (4.36)} \\ &= \int_{\mathbb{R}^d} \exp\left(-\frac{\|\boldsymbol{x}\|_2^2}{2h^2} - i\boldsymbol{\omega}^\top \boldsymbol{x}\right) d\boldsymbol{x} && \text{using the definition of the Gaussian kernel (4.14)} \\ &= (2h^2\pi)^{d/2} \exp\left(-h^2 \frac{\|\boldsymbol{\omega}\|_2^2}{2}\right). && (4.39) \end{aligned}$$

The key idea is now to interpret the kernel as an expectation,

$$\begin{aligned} k(\boldsymbol{x}-\boldsymbol{x}') &= \int_{\mathbb{R}^d} p(\boldsymbol{\omega}) e^{i\boldsymbol{\omega}^\top(\boldsymbol{x}-\boldsymbol{x}')}\, d\boldsymbol{\omega} && \text{from Equation (4.38)} \\ &= \mathbb{E}_{\boldsymbol{\omega}\sim p}\left[e^{i\boldsymbol{\omega}^\top(\boldsymbol{x}-\boldsymbol{x}')}\right] && \text{by the definition of expectation (1.19)} \\ &= \mathbb{E}_{\boldsymbol{\omega}\sim p}\left[\cos(\boldsymbol{\omega}^\top\boldsymbol{x} - \boldsymbol{\omega}^\top\boldsymbol{x}') + i\sin(\boldsymbol{\omega}^\top\boldsymbol{x} - \boldsymbol{\omega}^\top\boldsymbol{x}')\right]. && \text{using Euler's formula (4.35)} \end{aligned}$$

Observe that as both k and p are real, convergence of the integral implies $\mathbb{E}_{\boldsymbol{\omega}\sim p}\left[\sin(\boldsymbol{\omega}^\top\boldsymbol{x} - \boldsymbol{\omega}^\top\boldsymbol{x}')\right] = 0$. Hence,

$$\begin{aligned} &= \mathbb{E}_{\boldsymbol{\omega}\sim p}\left[\cos(\boldsymbol{\omega}^\top\boldsymbol{x} - \boldsymbol{\omega}^\top\boldsymbol{x}')\right] \\ &= \mathbb{E}_{\boldsymbol{\omega}\sim p}\mathbb{E}_{b\sim\mathrm{Unif}([0,2\pi])}\left[\cos((\boldsymbol{\omega}^\top\boldsymbol{x}+b) - (\boldsymbol{\omega}^\top\boldsymbol{x}'+b))\right] && \text{expanding with } b-b \\ &= \mathbb{E}_{\boldsymbol{\omega}\sim p}\mathbb{E}_{b\sim\mathrm{Unif}([0,2\pi])}\left[\cos(\boldsymbol{\omega}^\top\boldsymbol{x}+b)\cos(\boldsymbol{\omega}^\top\boldsymbol{x}'+b)\right. \\ &\qquad\qquad \left. + \sin(\boldsymbol{\omega}^\top\boldsymbol{x}+b)\sin(\boldsymbol{\omega}^\top\boldsymbol{x}'+b)\right] && \text{using the angle subtraction identity, } \cos(\alpha-\beta) = \cos\alpha\cos\beta + \sin\alpha\sin\beta \\ &= \mathbb{E}_{\boldsymbol{\omega}\sim p}\mathbb{E}_{b\sim\mathrm{Unif}([0,2\pi])}\left[2\cos(\boldsymbol{\omega}^\top\boldsymbol{x}+b)\cos(\boldsymbol{\omega}^\top\boldsymbol{x}'+b)\right] && \text{using} \\ &= \mathbb{E}_{\boldsymbol{\omega}\sim p, b\sim\mathrm{Unif}([0,2\pi])}\left[z_{\boldsymbol{\omega},b}(\boldsymbol{x})\cdot z_{\boldsymbol{\omega},b}(\boldsymbol{x}')\right] && (4.40) \end{aligned}$$

$$\mathbb{E}_b[\cos(\alpha+b)\cos(\beta+b)] = \mathbb{E}_b[\sin(\alpha+b)\sin(\beta+b)]$$

for $b \sim \mathrm{Unif}([0,2\pi])$

where $z_{\boldsymbol{\omega},b}(\boldsymbol{x}) \doteq \sqrt{2}\cos(\boldsymbol{\omega}^\top\boldsymbol{x}+b)$,

$$\approx \frac{1}{m}\sum_{i=1}^{m} z_{\boldsymbol{\omega}^{(i)},b^{(i)}}(\boldsymbol{x})\cdot z_{\boldsymbol{\omega}^{(i)},b^{(i)}}(\boldsymbol{x}') \qquad (4.41)$$

using Monte Carlo sampling to estimate the expectation, see Example A.6

for independent samples $\boldsymbol{\omega}^{(i)} \overset{\text{iid}}{\sim} p$ and $b^{(i)} \overset{\text{iid}}{\sim} \mathrm{Unif}([0,2\pi])$,

$$= \boldsymbol{z}(\boldsymbol{x})^\top \boldsymbol{z}(\boldsymbol{x}') \qquad (4.42)$$

where the (randomized) feature map of random Fourier features is

$$\boldsymbol{z}(\boldsymbol{x}) \doteq \frac{1}{\sqrt{m}}\left[z_{\boldsymbol{\omega}^{(1)},b^{(1)}}(\boldsymbol{x}), \ldots, z_{\boldsymbol{\omega}^{(m)},b^{(m)}}(\boldsymbol{x})\right]^\top. \qquad (4.43)$$

Intuitively, each component of the feature map $z(x)$ projects x onto a random direction ω drawn from the (inverse) Fourier transform $p(\omega)$ of $k(x - x')$, and wraps this line onto the unit circle in $\mathbb{R}^2$. After transforming two points x and x' in this way, their inner product is an unbiased estimator of $k(x - x')$. The mapping $z_{\omega,b}(x) = \sqrt{2}\cos(\omega^\top x + b)$ additionally rotates the circle by a random amount b and projects the points onto the interval $[0, 1]$.

Rahimi et al. (2007) show that Bayesian linear regression with the feature map z approximates Gaussian processes with a stationary kernel:

Theorem 4.13 (Uniform convergence of Fourier features)**.** *Suppose $\mathcal{M}$ is a compact subset of $\mathbb{R}^d$ with diameter* $\operatorname{diam}(\mathcal{M})$*. Then for a stationary kernel k, the random Fourier features z, and any $\epsilon > 0$ it holds that*

$$\mathbb{P}\left(\sup_{x,x' \in \mathcal{M}} \left| z(x)^\top z(x') - k(x - x') \right| \geq \epsilon\right) \leq 2^8 \left(\frac{\sigma_p \operatorname{diam}(\mathcal{M})}{\epsilon}\right)^2 \exp\left(-\frac{m\epsilon^2}{8(d+2)}\right) \tag{4.44}$$

where $\sigma_p^2 \doteq \mathbb{E}_{\omega \sim p}\left[\omega^\top \omega\right]$ is the second moment of p, m is the dimension of $z(x)$, and d is the dimension of x. (?)

Note that the error probability decays exponentially fast in the dimension of the Fourier feature space.

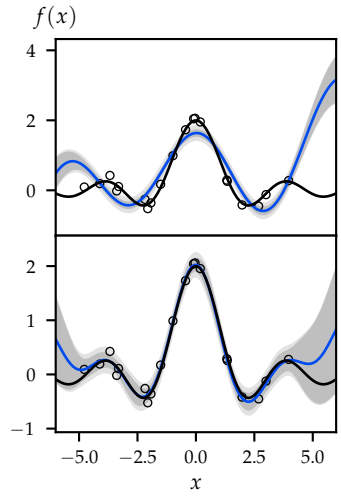


Figure 4.13: Example of random Fourier features with where the number of features m is 5 (top) and 10 (bottom), respectively. The noise-free true function is shown in black and the mean of the Gaussian process is shown in blue.

4.5.3 *Data Sampling*

Another natural approach is to only consider a (random) subset of the training data during learning. The naive approach is to subsample uniformly at random. Not very surprisingly, we can do much better.

One subsampling method is the *inducing points method* (Quinonero-Candela and Rasmussen, 2005). The idea is to summarize the data around so-called inducing points.[10] For now, let us consider an arbitrary set of inducing points,

$$U \doteq \{\bar{x}_1, \ldots, \bar{x}_k\}.$$

Then, the original Gaussian process can be recovered using marginalization,

$$p(f^\star, f) = \int_{\mathbb{R}^k} p(f^\star, f, u)\, du = \int_{\mathbb{R}^k} p(f^\star, f \mid u) p(u)\, du, \tag{4.45}$$

using the sum rule (1.7) and product rule (1.11)

where $f \doteq [f(x_1) \;\cdots\; f(x_n)]^\top$ and $f^\star \doteq f(x^\star)$ at some evaluation point $x^\star \in \mathcal{X}$. We use $u \doteq [f(\bar{x}_1) \;\cdots\; f(\bar{x}_k)]^\top \in \mathbb{R}^k$ to denote the predictions of the model at the inducing points U. Due to the marginalization property of Gaussian processes (4.1), we have that $u \sim \mathcal{N}(0, K_{UU})$.

Problem 4.8

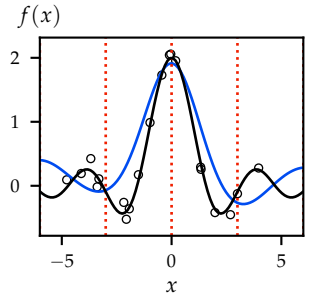


Figure 4.14: Inducing points u are shown as vertical dotted red lines. The noise-free true function is shown in black and the mean of the Gaussian process is shown in blue. Observe that the true function is approximated "well" around the inducing points.

[10] The inducing points can be treated as hyperparameters.

The key idea is to approximate the joint prior, assuming that $f^\star$ and $\boldsymbol{f}$ are conditionally independent given $\boldsymbol{u}$,

$$p(f^\star, \boldsymbol{f}) \approx \int_{\mathbb{R}^k} p(f^\star \mid \boldsymbol{u}) p(\boldsymbol{f} \mid \boldsymbol{u}) p(\boldsymbol{u}) \, d\boldsymbol{u}. \tag{4.46}$$

Here, $p(\boldsymbol{f} \mid \boldsymbol{u})$ and $p(f^\star \mid \boldsymbol{u})$ are commonly called the *training conditional* and the *testing conditional*, respectively. Still denoting the observations by $A = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ and defining $\star \doteq \{\boldsymbol{x}^\star\}$, we know, using the closed-form expression for conditional Gaussians (1.53),

$$p(\boldsymbol{f} \mid \boldsymbol{u}) \sim \mathcal{N}(\boldsymbol{f}; \boldsymbol{K}_{AU} \boldsymbol{K}_{UU}^{-1} \boldsymbol{u}, \boldsymbol{K}_{AA} - \boldsymbol{Q}_{AA}), \tag{4.47a}$$
$$p(f^\star \mid \boldsymbol{u}) \sim \mathcal{N}(f^\star; \boldsymbol{K}_{\star U} \boldsymbol{K}_{UU}^{-1} \boldsymbol{u}, \boldsymbol{K}_{\star\star} - \boldsymbol{Q}_{\star\star}) \tag{4.47b}$$

where $\boldsymbol{Q}_{ab} \doteq \boldsymbol{K}_{aU} \boldsymbol{K}_{UU}^{-1} \boldsymbol{K}_{Ub}$. Intuitively, $\boldsymbol{K}_{AA}$ represents the prior covariance and $\boldsymbol{Q}_{AA}$ represents the covariance "explained" by the inducing points.[11]

[11] For more details, refer to section 2 of "A unifying view of sparse approximate Gaussian process regression" (Quinonero-Candela and Rasmussen, 2005).

Computing the full covariance matrix is expensive. In the following, we mention two approximations to the covariance of the training conditional (and testing conditional).

Example 4.14: Subset of regressors

The *subset of regressors* (SoR) approximation is defined as

$$q_{\text{SoR}}(\boldsymbol{f} \mid \boldsymbol{u}) \doteq \mathcal{N}(\boldsymbol{f}; \boldsymbol{K}_{AU} \boldsymbol{K}_{UU}^{-1} \boldsymbol{u}, \boldsymbol{0}), \tag{4.48a}$$
$$q_{\text{SoR}}(f^\star \mid \boldsymbol{u}) \doteq \mathcal{N}(f^\star; \boldsymbol{K}_{\star U} \boldsymbol{K}_{UU}^{-1} \boldsymbol{u}, \boldsymbol{0}). \tag{4.48b}$$

Comparing to Equation (4.47), SoR simply forgets about all variance and covariance.

Example 4.15: Fully independent training conditional

The *fully independent training conditional* (FITC) approximation is defined as

$$q_{\text{FITC}}(\boldsymbol{f} \mid \boldsymbol{u}) \doteq \mathcal{N}(\boldsymbol{f}; \boldsymbol{K}_{AU} \boldsymbol{K}_{UU}^{-1} \boldsymbol{u}, \operatorname{diag}\{\boldsymbol{K}_{AA} - \boldsymbol{Q}_{AA}\}), \tag{4.49a}$$
$$q_{\text{FITC}}(f^\star \mid \boldsymbol{u}) \doteq \mathcal{N}(f^\star; \boldsymbol{K}_{\star U} \boldsymbol{K}_{UU}^{-1} \boldsymbol{u}, \operatorname{diag}\{\boldsymbol{K}_{\star\star} - \boldsymbol{Q}_{\star\star}\}). \tag{4.49b}$$

In contrast to SoR, FITC keeps track of the variances but forgets about the covariance.

The computational cost for inducing point methods SoR and FITC is dominated by the cost of inverting $\boldsymbol{K}_{UU}$. Thus, the time complexity is cubic in the number of inducing points, but only linear in the number of data points.

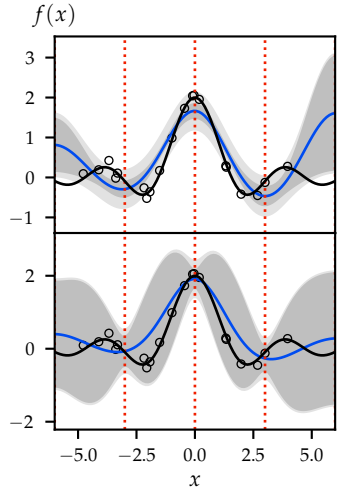


Figure 4.15: Comparison of SoR (top) and FITC (bottom). The inducing points $\boldsymbol{u}$ are shown as vertical dotted red lines. The noise-free true function is shown in black and the mean of the Gaussian process is shown in blue.

Discussion

This chapter introduced Gaussian processes which leverage the function-space view on linear regression to perform exact probabilistic inference with flexible, nonlinear models. A Gaussian process can be seen as a non-parametric model since it can represent an infinite-dimensional parameter space. Instead, as we saw with the representer theorem, such non-parametric (i.e., "function-space") models are directly represented as functions of the data points. While this can make these models more flexible than a simple linear parametric model in input space, it also makes them computationally expensive as the number of data points grows. To this end, we discussed several ways of approximating Gaussian processes.

Nevertheless, for today's internet-scale datasets, modern machine learning typically relies on large parametric models that learn features from data. These models can effectively amortize the cost of inference during training by encoding information into a fixed set of parameters. In the following chapters, we will start to explore approaches to approximate probabilistic inference that can be applied to such models.

Problems

4.1. Feature space of Gaussian kernel.

1. Show that the univariate Gaussian kernel with length scale $h = 1$ implicitly defines a feature space with basis vectors

$$\boldsymbol{\phi}(x) = \begin{bmatrix} \phi_0(x) \\ \phi_1(x) \\ \vdots \end{bmatrix} \quad \text{with} \quad \phi_j(x) = \frac{1}{\sqrt{j!}} e^{-\frac{x^2}{2}} x^j.$$

   Hint: Use the Taylor series expansion of the exponential function, $e^x = \sum_{j=0}^{\infty} \frac{x^j}{j!}$.
2. Note that the vector $\boldsymbol{\phi}(x)$ is ∞-dimensional. Thus, taking the function-space view allows us to perform regression in an infinite-dimensional feature space. What is the effective dimension when regressing n univariate data points with a Gaussian kernel?

4.2. Kernels on the circle.

Consider a dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ with labels $y_i \in \mathbb{R}$ and inputs $\boldsymbol{x}_i$ which lie on the unit circle $\mathbb{S} \subset \mathbb{R}^2$. In particular, any element of $\mathbb{S}$ can be identified with points in $\mathbb{R}^2$ of form $(\cos(\theta), \sin(\theta))$ or with the respective angles $\theta \in [0, 2\pi)$.

You now want to use GP regression to learn an unknown mapping from $\mathbb{S}$ to $\mathbb{R}$ using this dataset. Thus, you need a valid kernel k :

$\mathbb{S} \times \mathbb{S} \to \mathbb{R}$. First, we look at kernels k which can be understood as analogous to the Gaussian kernel.

1. You think of the "extrinsic" kernel $k_e : \mathbb{S} \times \mathbb{S} \to \mathbb{R}$ defined by
$$k_e(\theta, \theta') \doteq \exp\left(-\frac{\|\boldsymbol{x}(\theta) - \boldsymbol{x}(\theta')\|_2^2}{2\kappa^2}\right),$$
where $\boldsymbol{x}(\theta) \doteq (\cos(\theta), \sin(\theta))$. Is k_e positive semi-definite for all values of $\kappa > 0$?
2. Then, you think of an "intrinsic" kernel $k_i : \mathbb{S} \times \mathbb{S} \to \mathbb{R}$ defined by
$$k_i(\theta, \theta') \doteq \exp\left(-\frac{d(\theta, \theta')^2}{2\kappa^2}\right)$$
where $d(\theta, \theta') \doteq \min(|\theta - \theta'|, |\theta - \theta' - 2\pi|, |\theta - \theta' + 2\pi|)$ is the standard arc length distance on the circle $\mathbb{S}$.
You would now like to test whether this kernel is positive semi-definite. We pick $\kappa = 2$ and compute the kernel matrix $\boldsymbol{K}$ for the points corresponding to the angles $\{0, \pi/2, \pi, 3\pi/2\}$. This kernel matrix $\boldsymbol{K}$ has eigenvectors $(1, 1, 1, 1)$ and $(-1, 1, -1, 1)$.
Now compute the eigenvalue corresponding to the eigenvector $(-1, 1, -1, 1)$.
3. Is k_i positive semi-definite for $\kappa = 2$?
4. A mathematician friend of yours suggests to you yet another kernel for points on the circle $\mathbb{S}$, called the *heat kernel*. The kernel itself has a complicated expression but can be accurately approximated by
$$k_h(\theta, \theta') \doteq \frac{1}{C_\kappa}\left(1 + \sum_{l=1}^{L-1} e^{-\frac{\kappa^2}{2}l^2} 2\cos(l(\theta - \theta'))\right),$$
where $L \in \mathbb{N}$ controls the quality of approximation and $C_\kappa > 0$ is a normalizing constant that depends only on κ.
Is k_h is positive semi-definite for all values of $\kappa > 0$ and $L \in \mathbb{N}$?
Hint: Recall that $\cos(a - b) = \cos(a)\cos(b) + \sin(a)\sin(b)$.

4.3. A Kalman filter as a Gaussian process.

Next we will show that the Kalman filter from Example 3.4 can be seen as a Gaussian process. To this end, we define

$$f : \mathbb{N}_0 \to \mathbb{R}, \quad t \mapsto X_t. \tag{4.50}$$

Assuming that $X_0 \sim \mathcal{N}(0, \sigma_0^2)$ and $X_{t+1} \doteq X_t + \varepsilon_t$ with independent noise $\varepsilon_t \sim \mathcal{N}(0, \sigma_x^2)$, show that

$$f \sim \mathcal{GP}(0, k_{\text{KF}}) \quad \text{where} \tag{4.51}$$
$$k_{\text{KF}}(t, t') \doteq \sigma_0^2 + \sigma_x^2 \min\{t, t'\}. \tag{4.52}$$

This particular kernel $k(t, t') \doteq \min\{t, t'\}$ but over the continuous-time domain defines the *Wiener process* (also known as Brownian motion).

4.4. Reproducing property and RKHS norm.

1. Derive the reproducing property.
   Hint: Use $k(\boldsymbol{x}, \boldsymbol{x}') = \langle k(\boldsymbol{x}, \cdot), k(\boldsymbol{x}', \cdot) \rangle_k$.
2. Show that the RKHS norm $\|\cdot\|_k$ is a measure of smoothness by proving that for any $f \in \mathcal{H}_k(\mathcal{X})$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$ it holds that

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq \|f\|_k \, \|k(\boldsymbol{x}, \cdot) - k(\boldsymbol{y}, \cdot)\|_k \,.$$

4.5. Representer theorem.

With this, we can now derive the representer theorem (4.20).

Hint: Recall

1. *the reproducing property* $f(\boldsymbol{x}) = \langle f, k(\boldsymbol{x}, \cdot) \rangle_k$ *with* $k(\boldsymbol{x}, \cdot) \in \mathcal{H}_k(\mathcal{X})$ *which holds for all* $f \in \mathcal{H}_k(\mathcal{X})$ *and* $\boldsymbol{x} \in \mathcal{H}_k(\mathcal{X})$*, and*
2. *that the norm after projection is smaller or equal the norm before projection.*

Then decompose f *into parallel and orthogonal components with respect to* $\operatorname{span}\{k(\boldsymbol{x}_1, \cdot), \ldots, k(\boldsymbol{x}_n, \cdot)\}$.

4.6. MAP estimate of Gaussian processes.

Let us denote by $A = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ the set of training points. We will now show that the MAP estimate of GP regression corresponds to the solution of the regularized linear regression problem in the RKHS stated in Equation (4.21):

$$\hat{f} \doteq \operatorname*{arg\,min}_{f \in \mathcal{H}_k(\mathcal{X})} - \log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, f) + \frac{1}{2} \|f\|_k^2 \,.$$

In the following, we abbreviate $\boldsymbol{K} = \boldsymbol{K}_{AA}$. We will also assume that the GP has a zero mean function.

1. Show that Equation (4.21) is equivalent to

$$\hat{\boldsymbol{\alpha}} \doteq \operatorname*{arg\,min}_{\boldsymbol{\alpha} \in \mathbb{R}^n} \|\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}\|_2^2 + \lambda \, \|\boldsymbol{\alpha}\|_{\boldsymbol{K}}^2 \tag{4.53}$$

   for some $\lambda > 0$ which is also known as *kernel ridge regression*. Determine λ.
2. Show that Equation (4.53) with the λ determined in (1) is equivalent to the MAP estimate of GP regression.
   Hint: Recall from Equation (4.6) *that the MAP estimate at a point* $\boldsymbol{x}^\star$ *is* $\mathbb{E}[f^\star \mid \boldsymbol{x}^\star, \boldsymbol{X}, \boldsymbol{y}] = \boldsymbol{k}_{\boldsymbol{x}^\star, A}^\top (\boldsymbol{K} + \sigma_{\mathrm{n}}^2 \boldsymbol{I})^{-1} \boldsymbol{y}$.

4.7. Gradient of the marginal likelihood.

In this exercise, we derive Equation (4.30).

Recall that we were considering a dataset $(\boldsymbol{X}, \boldsymbol{y})$ of noise-perturbed evaluations $y_i = f(\boldsymbol{x}_i) + \varepsilon_i$ where $\varepsilon_i \sim \mathcal{N}(0, \sigma_{\mathrm{n}}^2)$ and f is an unknown

function. We make the hypothesis $f \sim \mathcal{GP}(0, k_{\boldsymbol{\theta}})$ with a zero mean function and the covariance function $k_{\boldsymbol{\theta}}$. We are interested in finding the hyperparameters $\boldsymbol{\theta}$ that maximize the marginal likelihood $p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta})$.

1. Derive Equation (4.30).
   Hint: You can use the following identities:
   (a) for any invertible matrix $\boldsymbol{M}$,

$$\frac{\partial}{\partial \theta_j} \boldsymbol{M}^{-1} = -\boldsymbol{M}^{-1} \frac{\partial \boldsymbol{M}}{\partial \theta_j} \boldsymbol{M}^{-1} \quad \textit{and} \tag{4.54}$$

   (b) for any symmetric positive definite matrix $\boldsymbol{M}$,

$$\frac{\partial}{\partial \theta_j} \log \det(\boldsymbol{M}) = \operatorname{tr}\left(\boldsymbol{M}^{-1} \frac{\partial \boldsymbol{M}}{\partial \theta_j} \right). \tag{4.55}$$

2. Assume now that the covariance function for the noisy targets (i.e., including the noise contribution) can be expressed as

$$k_{\boldsymbol{y},\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}') = \theta_0 \tilde{k}(\boldsymbol{x}, \boldsymbol{x}')$$

   where $\tilde{k}$ is a valid kernel independent of θ_0.[12]
   Show that $\frac{\partial}{\partial \theta_0} \log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = 0$ admits a closed-form solution for θ_0 which we denote by $\theta_0^\star$.
3. How should the optimal parameter $\theta_0^\star$ be scaled if we scale the labels $\boldsymbol{y}$ by a scalar s?

[12] That is, $\boldsymbol{K}_{\boldsymbol{y},\boldsymbol{\theta}}(i,j) = k_{\boldsymbol{y},\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

4.8. Uniform convergence of Fourier features.

In this exercise, we will prove Theorem 4.13.

Let $s(\boldsymbol{x}, \boldsymbol{x}') \doteq \boldsymbol{z}(\boldsymbol{x})^\top \boldsymbol{z}(\boldsymbol{x}')$ and $f(\boldsymbol{x}, \boldsymbol{x}') \doteq s(\boldsymbol{x}, \boldsymbol{x}') - k(\boldsymbol{x}, \boldsymbol{x}')$. Observe that both functions are shift invariant, and we will therefore denote them as univariate functions with argument $\boldsymbol{\Delta} \equiv \boldsymbol{x} - \boldsymbol{x}' \in \mathcal{M}_\Delta$. Notice that our goal is to bound the probability of the event $\sup_{\boldsymbol{\Delta} \in \mathcal{M}_\Delta} |f(\boldsymbol{\Delta})| \geq \epsilon$.

1. Show that for all $\boldsymbol{\Delta} \in \mathcal{M}_\Delta$, $\mathbb{P}(|f(\boldsymbol{\Delta})| \geq \epsilon) \leq 2 \exp\left(-\frac{m\epsilon^2}{4} \right)$.

What we have derived in (1) is known as a *pointwise convergence* guarantee. However, we are interested in bounding the *uniform convergence* over the compact set $\mathcal{M}_\Delta$.

Our approach will be to "cover" the compact set $\mathcal{M}_\Delta$ using T balls of radius r whose centers we denote by $\{\boldsymbol{\Delta}_i\}_{i=1}^T$. It can be shown that this is possible for some $T \leq (4 \operatorname{diam}(\mathcal{M})/r)^d$. It can furthermore be shown that

$$\forall i.\ |f(\boldsymbol{\Delta}_i)| < \frac{\epsilon}{2} \text{ and } \|\nabla f(\boldsymbol{\Delta}^\star)\|_2 < \frac{\epsilon}{2r} \implies \sup_{\boldsymbol{\Delta} \in \mathcal{M}_\Delta} |f(\boldsymbol{\Delta})| < \epsilon$$

where $\boldsymbol{\Delta}^\star = \arg\max_{\boldsymbol{\Delta} \in \mathcal{M}_\Delta} \|\nabla f(\boldsymbol{\Delta})\|_2$.

2. Prove $\mathbb{P}\left(\|\nabla f(\boldsymbol{\Delta}^\star)\|_2 \geq \frac{\epsilon}{2r}\right) \leq \left(\frac{2r\sigma_\mathrm{p}}{\epsilon}\right)^2$.
   Hint: Recall that the random Fourier feature approximation is unbiased, i.e., $\mathbb{E}[s(\boldsymbol{\Delta})] = k(\boldsymbol{\Delta})$.
3. Prove $\mathbb{P}\left(\bigcup_{i=1}^T |f(\boldsymbol{\Delta}_i)| \geq \frac{\epsilon}{2}\right) \leq 2T\exp\left(-\frac{m\epsilon^2}{16}\right)$.
4. Combine the results from (2) and (3) to prove Theorem 4.13.
   Hint: You may use that
   (a) $\alpha r^{-d} + \beta r^2 = 2\beta^{\frac{d}{d+2}}\alpha^{\frac{2}{d+2}}$ *for* $r = (\alpha/\beta)^{\frac{1}{d+2}}$ *and*
   (b) $\frac{\sigma_\mathrm{p}\mathrm{diam}(\mathcal{M})}{\epsilon} \geq 1$.
5. Show that for the Gaussian kernel (4.14), $\sigma_\mathrm{p}^2 = \frac{d}{h^2}$.
   Hint: First show $\sigma_\mathrm{p}^2 = -\mathrm{tr}(\boldsymbol{H}_{\boldsymbol{\Delta}} k(\mathbf{0}))$.

4.9. Subset of regressors.

1. Using an SoR approximation, prove the following:

$$q_{\mathrm{SoR}}(\boldsymbol{f}, f^\star) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{f}\\ f^\star\end{bmatrix}; \mathbf{0}, \begin{bmatrix}\boldsymbol{Q}_{AA} & \boldsymbol{Q}_{A\star}\\ \boldsymbol{Q}_{\star A} & \boldsymbol{Q}_{\star\star}\end{bmatrix}\right) \tag{4.56}$$

$$q_{\mathrm{SoR}}(f^\star \mid \boldsymbol{y}) = \mathcal{N}(f^\star; \boldsymbol{Q}_{\star A}\tilde{\boldsymbol{Q}}_{AA}^{-1}\boldsymbol{y}, \boldsymbol{Q}_{\star\star} - \boldsymbol{Q}_{\star A}\tilde{\boldsymbol{Q}}_{AA}^{-1}\boldsymbol{Q}_{A\star}) \tag{4.57}$$

   where $\tilde{\boldsymbol{Q}}_{ab} \doteq \boldsymbol{Q}_{ab} + \sigma_\mathrm{n}^2$.
2. Derive that the resulting model is a degenerate Gaussian process with covariance function

$$k_{\mathrm{SoR}}(\boldsymbol{x}, \boldsymbol{x}') \doteq \boldsymbol{k}_{\boldsymbol{x},U}^\top \boldsymbol{K}_{UU}^{-1}\boldsymbol{k}_{\boldsymbol{x}',U}. \tag{4.58}$$

5

Variational Inference

We have seen how to perform (efficient) probabilistic inference with Gaussians, exploiting their closed-form formulas for marginal and conditional distributions. But what if we work with other distributions?

In this and the following chapter, we will discuss two methods of approximate inference. We begin by discussing variational (probabilistic) inference, which aims to find a good approximation of the posterior distribution from which it is easy to sample. In Chapter 6, we discuss Markov chain Monte Carlo methods, which approximate the sampling from the posterior distribution directly.

The fundamental idea behind variational inference is to approximate the true posterior distribution using a "simpler" posterior that is as close as possible to the true posterior:

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}) = \frac{1}{Z} p(\boldsymbol{\theta}, y_{1:n} \mid \boldsymbol{x}_{1:n}) \approx q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) \doteq q_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) \tag{5.1}$$

where $\boldsymbol{\lambda}$ represents the parameters of the *variational posterior* $q_{\boldsymbol{\lambda}}$, also called *variational parameters*. In doing so, variational inference reduces probabilistic inference — where the fundamental difficulty lies in solving high-dimensional integrals — to an optimization problem. Optimizing (stochastic) objectives is a well-understood problem with efficient algorithms that perform well in practice.[1]

[1] We provide an overview of first-order methods such as stochastic gradient descent in Appendix A.4.

5.1 Laplace Approximation

Before introducing a general framework of variational inference, we discuss a simpler method of approximate inference known as *Laplace's method*. This method was proposed as a method of approximating integrals as early as 1774 by Pierre-Simon Laplace. The idea is to use a Gaussian approximation (that is, a second-order Taylor approxima-

tion) of the posterior distribution around its mode. Let

$$\psi(\boldsymbol{\theta}) \doteq \log p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}) \tag{5.2}$$

denote the log-posterior. Then, using a second-order Taylor approximation (A.53) around the mode $\hat{\boldsymbol{\theta}}$ of ψ (i.e., the MAP estimate), we obtain the approximation $\hat{\psi}$ which is accurate for $\boldsymbol{\theta} \approx \hat{\boldsymbol{\theta}}$:

$$\begin{aligned}\psi(\boldsymbol{\theta}) \approx \hat{\psi}(\boldsymbol{\theta}) &\doteq \psi(\hat{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{\nabla}\psi(\hat{\boldsymbol{\theta}}) + \frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{H}_\psi(\hat{\boldsymbol{\theta}})(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \\ &= \psi(\hat{\boldsymbol{\theta}}) + \frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{H}_\psi(\hat{\boldsymbol{\theta}})(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}). \qquad (5.3)\end{aligned}$$

using $\boldsymbol{\nabla}\psi(\hat{\boldsymbol{\theta}}) = 0$

Compare this expression to the log-PDF of a Gaussian:

$$\log \mathcal{N}(\boldsymbol{\theta}; \hat{\boldsymbol{\theta}}, \boldsymbol{\Lambda}^{-1}) = -\frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{\Lambda}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) + \text{const.} \tag{5.4}$$

Since $\psi(\hat{\boldsymbol{\theta}})$ is constant with respect to $\boldsymbol{\theta}$,

$$\hat{\psi}(\boldsymbol{\theta}) = \log \mathcal{N}(\boldsymbol{\theta}; \hat{\boldsymbol{\theta}}, -\boldsymbol{H}_\psi(\hat{\boldsymbol{\theta}})^{-1}) + \text{const.} \tag{5.5}$$

The *Laplace approximation* q of p is

$$q(\boldsymbol{\theta}) \doteq \mathcal{N}(\boldsymbol{\theta}; \hat{\boldsymbol{\theta}}, \boldsymbol{\Lambda}^{-1}) \propto \exp(\hat{\psi}(\boldsymbol{\theta})) \qquad \text{where} \tag{5.6a}$$

$$\boldsymbol{\Lambda} \doteq -\boldsymbol{H}_\psi(\hat{\boldsymbol{\theta}}) = -\boldsymbol{H}_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n})\big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}. \tag{5.6b}$$

Recall that for this approximation to be well-defined, the covariance matrix $\boldsymbol{\Lambda}^{-1}$ (or equivalently the precision matrix $\boldsymbol{\Lambda}$) needs to be symmetric and positive semi-definite. Let us verify that this is indeed the case for sufficiently smooth ψ.[2] In this case, the Hessian $\boldsymbol{\Lambda}$ is symmetric since the order of differentiation does not matter. Moreover, by the second-order optimality condition, $\boldsymbol{H}_\psi(\hat{\boldsymbol{\theta}})$ is negative semi-definite since $\hat{\boldsymbol{\theta}}$ is a maximum of ψ, which implies that $\boldsymbol{\Lambda}$ is positive semi-definite.

[2] ψ being twice continuously differentiable around $\hat{\boldsymbol{\theta}}$ is sufficient.

Example 5.1: Laplace approximation of a Gaussian

Consider approximating the Gaussian density $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ using a Laplace approximation.

We know that the mode of p is $\boldsymbol{\mu}$, which we can verify by computing the gradient,

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}) = -\frac{1}{2}(2\boldsymbol{\Sigma}^{-1}\boldsymbol{\theta} - 2\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}) \stackrel{!}{=} 0 \iff \boldsymbol{\theta} = \boldsymbol{\mu}. \tag{5.7}$$

For the Hessian of $\log p(\boldsymbol{\theta})$, we get

$$\boldsymbol{H}_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}) = (\boldsymbol{D}_{\boldsymbol{\theta}}(\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} - \boldsymbol{\Sigma}^{-1}\boldsymbol{\theta}))^\top = -(\boldsymbol{\Sigma}^{-1})^\top = -\boldsymbol{\Sigma}^{-1}. \tag{5.8}$$

using $(\boldsymbol{A}^{-1})^\top = (\boldsymbol{A}^\top)^{-1}$ and symmetry of $\boldsymbol{\Sigma}$

We see that the Laplace approximation of a Gaussian $p(\boldsymbol{\theta})$ is exact, which should not come as a surprise since the second-order Taylor approximation of $\log p(\boldsymbol{\theta})$ is exact for Gaussians.

The Laplace approximation matches the shape of the true posterior around its mode but may not represent it accurately elsewhere — often leading to extremely overconfident predictions. An example is given in Figure 5.1. Nevertheless, the Laplace approximation has some desirable properties such as being relatively easy to apply in a post-hoc manner, that is, after having already computed the MAP estimate. It preserves the MAP point estimate as its mean and just "adds" a little uncertainty around it. However, the fact that it can be arbitrarily different from the true posterior makes it unsuitable for approximate probabilistic inference.

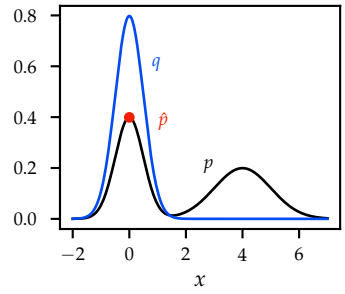


Figure 5.1: The Laplace approximation q greedily selects the mode of the true posterior distribution p and matches the curvature around the mode $\hat{p}$. As shown here, the Laplace approximation can be extremely overconfident when p is not approximately Gaussian.

5.1.1 *Example: Bayesian Logistic Regression*

As an example, we will look at Laplace approximation in the context of Bayesian logistic regression. Logistic regression learns a classifier that decides for a given input whether it belongs to one of two classes. A sigmoid function, typically the *logistic function*,

$$\sigma(z) \doteq \frac{1}{1+\exp(-z)} \in (0,1), \qquad z = \boldsymbol{w}^\top \boldsymbol{x}, \tag{5.9}$$

is used to obtain the class probabilities. *Bayesian logistic regression* corresponds to Bayesian linear regression with a Bernoulli likelihood,

$$y \mid \boldsymbol{x}, \boldsymbol{w} \sim \text{Bern}(\sigma(\boldsymbol{w}^\top \boldsymbol{x})), \tag{5.10}$$

where $y \in \{-1, 1\}$ is the binary class label.[3] Observe that given a data point $(\boldsymbol{x}, y)$, the probability of a correct classification is

$$p(y \mid \boldsymbol{x}, \boldsymbol{w}) = \begin{cases} \sigma(\boldsymbol{w}^\top \boldsymbol{x}) & \text{if } y = 1 \\ 1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x}) & \text{if } y = -1 \end{cases} = \sigma(y \boldsymbol{w}^\top \boldsymbol{x}), \tag{5.11}$$

as the logistic function σ is symmetric around 0. Also, recall that Bayesian linear regression used the prior

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}; \boldsymbol{0}, \sigma_\text{p}^2 \boldsymbol{I}) \propto \exp\left(-\frac{1}{2\sigma_\text{p}^2} \|\boldsymbol{w}\|_2^2\right).$$

Let us first find the posterior mode, that is, the MAP estimate of the weights:

$$\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}} p(\boldsymbol{w} \mid \boldsymbol{x}_{1:n}, y_{1:n})$$

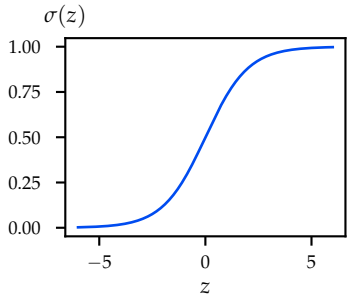


Figure 5.2: The logistic function squashes the linear function $\boldsymbol{w}^\top \boldsymbol{x}$ onto the interval $(0, 1)$.

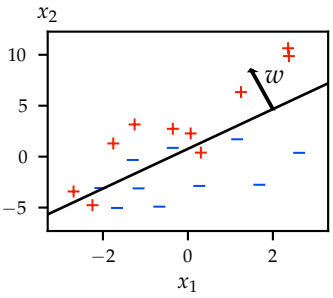


Figure 5.3: Logistic regression classifies data into two classes with a linear decision boundary.

[3] The same approach extends to Gaussian processes where it is known as *Gaussian process classification*, see Problem 5.2 and Hensman et al. (2015).

$$= \arg\max_{\boldsymbol{w}} p(\boldsymbol{w}) p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{w}) \qquad \text{using Bayes' rule (1.45)}$$

$$= \arg\max_{\boldsymbol{w}} \log p(\boldsymbol{w}) + \log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{w}) \qquad \text{taking the logarithm}$$

$$= \arg\max_{\boldsymbol{w}} -\frac{1}{2\sigma_{\mathrm{p}}^2} \|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n} \log \sigma(y_i \boldsymbol{w}^\top \boldsymbol{x}_i) \qquad \text{using independence of the observations and Equation (5.11)}$$

$$= \arg\min_{\boldsymbol{w}} \frac{1}{2\sigma_{\mathrm{p}}^2} \|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n} \log(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)). \qquad (5.12) \qquad \text{using the definition of } \sigma \text{ (5.9)}$$

Note that for $\lambda = 1/2\sigma_{\mathrm{p}}^2$, the above optimization is equivalent to standard (regularized) logistic regression where

$$\ell_{\log}(\boldsymbol{w}^\top \boldsymbol{x}; y) \doteq \log(1 + \exp(-y \boldsymbol{w}^\top \boldsymbol{x})) \qquad (5.13)$$

is called *logistic loss*. The gradient of the logistic loss is given by (?) **Problem 5.1 (1)**

$$\nabla_{\boldsymbol{w}} \ell_{\log}(\boldsymbol{w}^\top \boldsymbol{x}; y) = -y \boldsymbol{x} \cdot \sigma(-y \boldsymbol{w}^\top \boldsymbol{x}). \qquad (5.14)$$

Recall that due to the symmetry of σ around 0, $\sigma(-y \boldsymbol{w}^\top \boldsymbol{x})$ is the probability that $\boldsymbol{x}$ was *not* classified as y. Intuitively, if the model is "surprised" by the label, the gradient is large.

We can therefore use SGD with the (regularized) gradient step and with batch size 1,

$$\boldsymbol{w} \leftarrow \boldsymbol{w}(1 - 2\lambda\eta_t) + \eta_t y \boldsymbol{x} \sigma(-y \boldsymbol{w}^\top \boldsymbol{x}), \qquad (5.15)$$

for the data point $(\boldsymbol{x}, y)$ picked uniformly at random from the training data. Here, $2\lambda\eta_t$ is due to the gradient of the regularization term, in effect, performing weight decay.

Example 5.2: Laplace approx. of Bayesian logistic regression

We have already found the mode of the posterior distribution, $\hat{\boldsymbol{w}}$.

Let us denote by

$$\pi_i \doteq \mathbb{P}(y_i = 1 \mid \boldsymbol{x}_i, \hat{\boldsymbol{w}}) = \sigma(\hat{\boldsymbol{w}}^\top \boldsymbol{x}_i) \qquad (5.16)$$

the probability of $\boldsymbol{x}_i$ belonging to the positive class under the model given by the MAP estimate of the weights. For the precision matrix, we then have

$$\begin{aligned}
\boldsymbol{\Lambda} &= -\boldsymbol{H}_{\boldsymbol{w}} \log p(\boldsymbol{w} \mid \boldsymbol{x}_{1:n}, y_{1:n})\big|_{\boldsymbol{w}=\hat{\boldsymbol{w}}} \\
&= -\boldsymbol{H}_{\boldsymbol{w}} \log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{w})\big|_{\boldsymbol{w}=\hat{\boldsymbol{w}}} - \boldsymbol{H}_{\boldsymbol{w}} \log p(\boldsymbol{w})\big|_{\boldsymbol{w}=\hat{\boldsymbol{w}}} \\
&= \sum_{i=1}^{n} \boldsymbol{H}_{\boldsymbol{w}} \ell_{\log}(\boldsymbol{w}^\top \boldsymbol{x}_i; y_i)\Big|_{\boldsymbol{w}=\hat{\boldsymbol{w}}} + \sigma_{\mathrm{p}}^{-2} \boldsymbol{I} && \text{using the definition of the logistic loss (5.13)} \\
&= \sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^\top \pi_i (1 - \pi_i) + \sigma_{\mathrm{p}}^{-2} \boldsymbol{I} && \text{using the Hessian of the logistic loss (5.73) which you derive in } \textbf{Problem 5.1 (2)}
\end{aligned}$$

$$= \boldsymbol{X}^\top \mathrm{diag}_{i\in[n]}\{\pi_i(1-\pi_i)\}\boldsymbol{X} + \sigma_\mathrm{p}^{-2}\boldsymbol{I}. \tag{5.17}$$

Observe that $\pi_i(1-\pi_i) \approx 0$ if $\pi_i \approx 1$ or $\pi_i \approx 0$. That is, if a training example is "well-explained" by $\hat{\boldsymbol{w}}$, then its contribution to the precision matrix is small. In contrast, we have $\pi_i(1-\pi_i) = 0.25$ for $\pi_i = 0.5$. Importantly, $\boldsymbol{\Lambda}$ does not depend on the normalization constant of the posterior distribution which is hard to compute.

In summary, we have that $\mathcal{N}(\hat{\boldsymbol{w}}, \boldsymbol{\Lambda}^{-1})$ is the Laplace approximation of $p(\boldsymbol{w} \mid \boldsymbol{x}_{1:n}, y_{1:n})$.

5.2 *Predictions with a Variational Posterior*

How can we make predictions using our variational approximation? We simply approximate the (intractable) true posterior with our variational posterior:

$$\begin{aligned} p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}) &= \int p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n})\, d\boldsymbol{\theta} && \text{using the sum rule (1.7)} \\ &\approx \int p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}) q_\lambda(\boldsymbol{\theta})\, d\boldsymbol{\theta}. && (5.18) \end{aligned}$$

A straightforward approach is to observe that Equation (5.18) can be viewed as an expectation over the variational posterior q_λ and approximated via Monte Carlo sampling:

$$\begin{aligned} &= \mathbb{E}_{\boldsymbol{\theta}\sim q_\lambda}[p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta})] && (5.19) \\ &\approx \frac{1}{m}\sum_{j=1}^{m} p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}_j) && (5.20) \end{aligned}$$

where $\boldsymbol{\theta}_j \overset{\text{iid}}{\sim} q_\lambda$.

Example 5.3: Predictions in Bayesian logistic regression

In the case of Bayesian logistic regression with a Gaussian approximation of the posterior, we can obtain more accurate predictions.

Observe that the final prediction $y^\star$ is conditionally independent of the model parameters $\boldsymbol{w}$ given the "latent value" $f^\star = \boldsymbol{w}^\top \boldsymbol{x}^\star$:

$$\begin{aligned} p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}) &\approx \int p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{w}) q_\lambda(\boldsymbol{w})\, d\boldsymbol{w} \\ &= \int\int p(y^\star \mid f^\star) p(f^\star \mid \boldsymbol{x}^\star, \boldsymbol{w}) q_\lambda(\boldsymbol{w})\, d\boldsymbol{w}\, df^\star && \text{once more, using the sum rule (1.7)} \\ &= \int p(y^\star \mid f^\star) \int p(f^\star \mid \boldsymbol{x}^\star, \boldsymbol{w}) q_\lambda(\boldsymbol{w})\, d\boldsymbol{w}\, df^\star. && \text{rearranging terms} \end{aligned} \tag{5.21}$$

The outer integral can be readily approximated since it is only one-dimensional! The challenging part is the inner integral, which is a high-dimensional integral over the model weights $\boldsymbol{w}$. Since the posterior over weights $q_{\boldsymbol{\lambda}}(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}; \hat{\boldsymbol{w}}, \boldsymbol{\Lambda}^{-1})$ is a Gaussian, we have due to the closedness properties of Gaussians (1.78) that

$$\int p(f^\star \mid \boldsymbol{x}^\star, \boldsymbol{w}) q_{\boldsymbol{\lambda}}(\boldsymbol{w})\, d\boldsymbol{w} = \mathcal{N}(\hat{\boldsymbol{w}}^\top \boldsymbol{x}^\star, \boldsymbol{x}^{\star\top} \boldsymbol{\Lambda}^{-1} \boldsymbol{x}^\star). \tag{5.22}$$

Crucially, this is a one-dimensional Gaussian in function-space as opposed to the d-dimensional Gaussian $q_{\boldsymbol{\lambda}}$ in weight-space!

As we have seen in Equation (5.11), for Bayesian logistic regression, the prediction $y^\star$ depends deterministically on the predicted latent value $f^\star$: $p(y^\star \mid f^\star) = \sigma(y^\star f^\star)$. Combining Equations (5.21) and (5.22), we obtain

$$p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}) \approx \int \sigma(y^\star f^\star) \cdot \mathcal{N}(f^\star; \hat{\boldsymbol{w}}^\top \boldsymbol{x}^\star, \boldsymbol{x}^{\star\top} \boldsymbol{\Lambda}^{-1} \boldsymbol{x}^\star)\, df^\star. \tag{5.23}$$

We have replaced the high-dimensional integral over the model parameters $\boldsymbol{\theta}$ by the one-dimensional integral over the prediction of our variational posterior $f^\star$. While this integral is generally still intractable, it can be approximated efficiently using numerical quadrature methods such as the Gauss-Legendre quadrature or alternatively with Monte Carlo sampling.

5.3 *Blueprint of Variational Inference*

General probabilistic inference poses the challenge of approximating the posterior distribution with limited memory and computation, resource constraints also present in humans and other intelligent systems. These resource constraints require information to be compressed, and as we will see, such a compression poses a fundamental tradeoff between model accuracy (on the observed data) and model complexity (to avoid overfitting).

Laplace approximation approximates the true (intractable) posterior with a simpler one, by greedily matching mode and curvature around it. Can we find "less greedy" approaches? We can view variational probabilistic inference more generally as a family of approaches aiming to approximate the true posterior distribution by one that is closest (according to some criterion) among a "simpler" class of distributions. To this end, we need to fix a class of distributions and define suitable criteria, which we can then optimize numerically. The key ben-

efit is that we can reduce the (generally intractable) problem of high-dimensional integration to the (often much more tractable) problem of optimization.

Definition 5.4 (Variational family)**.** Let $\mathcal{P}$ be the class of all probability distributions. A *variational family* $\mathcal{Q} \subseteq \mathcal{P}$ is a class of distributions such that each distribution $q \in \mathcal{Q}$ is characterized by unique variational parameters $\boldsymbol{\lambda} \in \Lambda$.

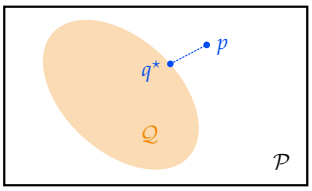


Figure 5.4: An illustration of variational inference in the space of distributions $\mathcal{P}$. The variational distribution $q^\star \in \mathcal{Q}$ is the optimal approximation of the true posterior p.

Example 5.5: Family of independent Gaussians

A straightforward example for a variational family is the family of independent Gaussians,

$$\mathcal{Q} \doteq \left\{ q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \operatorname{diag}_{i \in [d]}\{\sigma_i^2\}) \right\}, \tag{5.24}$$

which is parameterized by $\boldsymbol{\lambda} \doteq [\mu_{1:d}, \sigma^2_{1:d}]$. Such a multivariate distribution where all variables are independent is called a *mean-field distribution*. Importantly, this family of distributions is characterized by only $2d$ parameters!

Note that Figure 5.4 is a generalization of the canonical distinction between estimation error and approximation error from Figure 1.7, only that here, we operate in the space of distributions over functions as opposed the space of functions. A common notion of distance between two distributions q and p is the Kullback-Leibler divergence $\mathrm{KL}(q\|p)$ which we will define in the next section. Using this notion of distance, we need to solve the following optimization problem:

$$q^\star \doteq \arg\min_{q \in \mathcal{Q}} \mathrm{KL}(q\|p) = \arg\min_{\lambda \in \Lambda} \mathrm{KL}(q_\lambda \| p). \tag{5.25}$$

In Section 5.4, we introduce information theory and the Kullback-Leibler divergence. Then, in Section 5.5, we discuss how the optimization problem of Equation (5.25) can be solved efficiently.

5.4 *Information Theoretic Aspects of Uncertainty*

One of our main objectives throughout this manuscript is to capture the "uncertainty" about events A in an appropriate probability space. One very natural measure of uncertainty is their probability, $\mathbb{P}(A)$. In this section, we will introduce an alternative measure of uncertainty, namely the so-called "surprise" about the event A.

5.4.1 Surprise

The *surprise* about an event with probability u is defined as

$$\mathrm{S}[u] \doteq -\log u. \tag{5.26}$$

Observe that the surprise is a function from $\mathbb{R}_{\geq 0}$ to $\mathbb{R}$, where we let $\mathrm{S}[0] \equiv \infty$. Moreover, for a discrete random variable X, we have that $p(x) \leq 1$, and hence, $\mathrm{S}[p(x)] \geq 0$. But why is it reasonable to measure surprise by $-\log u$?

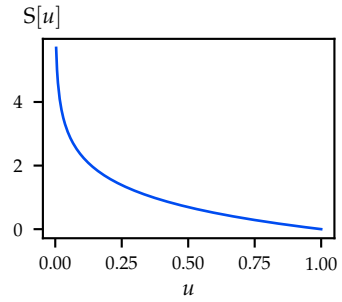


Figure 5.5: Surprise $\mathrm{S}[u]$ associated with an event of probability u.

Remarkably, it can be shown that the following natural axiomatic characterization leads to exactly this definition of surprise.

Theorem 5.6 (Axiomatic characterization of surprise)**.** *The axioms*

1. $\mathrm{S}[u] > \mathrm{S}[v] \implies u < v$ *(anti-monotonicity)*
2. S *is continuous,*
3. $\mathrm{S}[uv] = \mathrm{S}[u] + \mathrm{S}[v]$ *for independent events,*

characterize S *up to a positive constant factor.*

we are more surprised by unlikely events

no jumps in surprise for infinitesimal changes of probability

the surprise of independent events is additive

Proof. Observe that the third condition looks similar to the product rule of logarithms: $\log(uv) = \log v + \log v$. We can formalize this intuition by remembering Cauchy's functional equation, $f(x+y) = f(x) + f(y)$, which has the unique family of solutions $\{f : x \mapsto cx : c \in \mathbb{R}\}$ if f is required to be continuous. Such a solution is called an "additive function". Consider the function $g(x) \doteq f(e^x)$. Then, g is additive if and only if

$$f(e^x e^y) = f(e^{x+y}) = g(x+y) = g(x) + g(y) = f(e^x) + f(e^y).$$

This is precisely the third axiom of surprise for $f = \mathrm{S}$ and $e^x = u$! Hence, the second and third axioms of surprise imply that g must be additive and that $g(x) = \mathrm{S}[e^x] = cx$ for any $c \in \mathbb{R}$. If we replace e^x by u, we obtain $\mathrm{S}[u] = c \log u$. The first axiom of surprise implies that $c < 0$, and thus, $\mathrm{S}[u] = -c' \log u$ for any $c' > 0$. □

Importantly, surprise offers a different perspective on uncertainty as opposed to probability: the uncertainty about an event can either be interpreted in terms of its probability or in terms of its surprise, and the two "spaces of uncertainty" are related by a log-transform. This relationship is illustrated in Figure 5.6. Information theory is the study of uncertainty in terms of surprise.

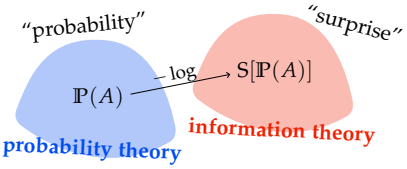


Figure 5.6: Illustration of the probability space and the corresponding "surprise space".

Throughout this manuscript we will see many examples where modeling uncertainty in terms of surprise (i.e., the information-theoretic interpretation of uncertainty) is useful. One example where we have

already encountered the "surprise space" was in the context of likelihood maximization (cf. Section 1.3.1) where we used that the log-transform linearizes products of probabilities. We will see later in Chapter 6 that in many cases the surprise $\mathrm{S}[p(x)]$ can also be interpreted as a "cost" or "energy" associated with the state x.

5.4.2 *Entropy*

The *entropy* of a distribution p is the average surprise about samples from p. In this way, entropy is a notion of uncertainty associated with the distribution p: if the entropy of p is large, we are more uncertain about $x \sim p$ than if the entropy of p were low. Formally,

$$\mathrm{H}[p] \doteq \mathbb{E}_{x\sim p}[\mathrm{S}[p(x)]] = \mathbb{E}_{x\sim p}[-\log p(x)]. \tag{5.27}$$

When $\mathbf{X} \sim p$ is a random vector distributed according to p, we write $\mathrm{H}[\mathbf{X}] \doteq \mathrm{H}[p]$. Observe that by definition, if p is discrete then $\mathrm{H}[p] \geq 0$ as $p(x) \leq 1$ $(\forall x)$.[4] For discrete distributions it is common to use the logarithm with base 2 rather than the natural logarithm:[5]

$$\mathrm{H}[p] = -\sum_x p(x)\log_2 p(x) \qquad \text{(if } p \text{ is discrete)}, \tag{5.28a}$$

$$\mathrm{H}[p] = -\int p(\boldsymbol{x})\log p(\boldsymbol{x})\,d\boldsymbol{x} \qquad \text{(if } p \text{ is continuous)}. \tag{5.28b}$$

Let us briefly recall Jensen's inequality, which is a useful tool when working with expectations of convex functions such as entropy:[6]

Fact 5.7 (Jensen's Inequality). ⓘ *Given a random variable X and a convex function $g : \mathbb{R} \to \mathbb{R}$, we have*

$$g(\mathbb{E}[X]) \leq \mathbb{E}[g(X)]. \tag{5.29}$$

Example 5.8: Examples of entropy

- *Fair Coin* $\mathrm{H}[\mathrm{Bern}(0.5)] = -2(0.5\log_2 0.5) = 1.$
- *Unfair Coin*

$$\mathrm{H}[\mathrm{Bern}(0.1)] = -0.1\log_2 0.1 - 0.9\log_2 0.9 \approx 0.469.$$

- *Uniform Distribution*

$$\mathrm{H}[\mathrm{Unif}(\{1,\dots,n\})] = -\sum_{i=1}^{n}\frac{1}{n}\log_2\frac{1}{n} = \log_2 n.$$

The uniform distribution has the maximum entropy among all discrete distributions supported on $\{1,\dots,n\}$ ⓘ. Note that a fair coin corresponds to a uniform distribution with $n = 2$. Also

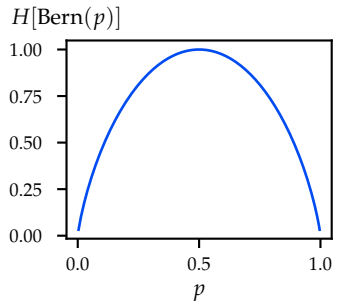


Figure 5.7: Entropy of a Bernoulli experiment with success probability p.

[4] The entropy of a continuous distribution can be negative. For example,

$$\mathrm{H}[\mathrm{Unif}([a,b])] = -\int \frac{1}{b-a}\log\frac{1}{b-a}\,dx = \log(b-a)$$

which is negative if $b - a < 1$.

[5] Recall that $\log_2 x = \frac{\log x}{\log 2}$, that is, logarithms to a different base only differ by a constant factor.

[6] The surprise $\mathrm{S}[u]$ is convex in u.

Problem 5.3 (1)

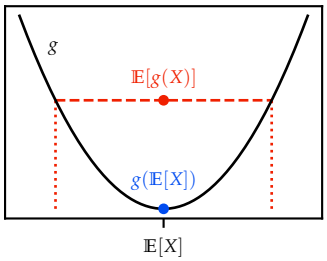


Figure 5.8: An illustration of Jensen's inequality. Due to the convexity of g, we have that g evaluated at $\mathbb{E}[X]$ will always be below the average of evaluations of g.

Problem 5.3 (2)

observe that $\log_2 n$ corresponds to the number of bits required to encode the outcome of the experiment.

In general, the entropy $\mathrm{H}[p]$ of a discrete distribution p can be interpreted as the average number of bits required to encode a sample $x \sim p$, or in other words, the average "information" carried by a sample x.

Example 5.9: Entropy of a Gaussian

Let us derive the entropy of a univariate Gaussian. Recall the PDF,

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{Z} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

where $Z = \sqrt{2\pi\sigma^2}$. Using the definition of entropy (5.28b), we obtain,

$$\begin{aligned}
\mathrm{H}\Big[\mathcal{N}(\mu, \sigma^2)\Big] &= -\int \frac{1}{Z} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \\
&\qquad \cdot \log\left(\frac{1}{Z} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\right) dx \\
&= \log Z \underbrace{\int \frac{1}{Z} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx}_{1} \\
&\qquad + \int \frac{1}{Z} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \frac{(x-\mu)^2}{2\sigma^2} dx \\
&= \log Z + \frac{1}{2\sigma^2} \mathbb{E}\Big[(x-\mu)^2\Big] && \text{using LOTUS (1.22)} \\
&= \log(\sigma\sqrt{2\pi}) + \frac{1}{2} && \text{using } \mathbb{E}\big[(x-\mu)^2\big] = \mathrm{Var}[x] = \sigma^2 \text{ (1.34)} \\
&= \log(\sigma\sqrt{2\pi e}). && (5.30) \qquad \text{using } \log\sqrt{e} = 1/2
\end{aligned}$$

In general, the entropy of a Gaussian is

$$\mathrm{H}[\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})] = \frac{1}{2} \log\det(2\pi e \boldsymbol{\Sigma}) = \frac{1}{2} \log\Big((2\pi e)^d \det(\boldsymbol{\Sigma})\Big). \quad (5.31)$$

Note that the entropy is a function of the determinant of the covariance matrix $\boldsymbol{\Sigma}$. In general, there are various ways of "scalarizing" the notion of uncertainty for a multivariate distribution. The determinant of $\boldsymbol{\Sigma}$ measures the volume of the credible sets around the mean $\boldsymbol{\mu}$, and is also called the *generalized variance*. Next to entropy and generalized variance (which are closely related for Gaussians), a common scalarization is the trace of $\boldsymbol{\Sigma}$, which is also called the *total variance*.

5.4.3 Cross-Entropy

How can we use entropy to measure our average surprise when assuming the data follows some distribution q but in reality the data follows a different distribution p?

Definition 5.10 (Cross-entropy)**.** The *cross-entropy* of a distribution q relative to the distribution p is

$$\mathrm{H}[p\|q] \doteq \mathbb{E}_{x\sim p}[\mathrm{S}[q(x)]] = \mathbb{E}_{x\sim p}[-\log q(x)]. \tag{5.32}$$

Cross-entropy can also be expressed in terms of the KL-divergence (cf. Section 5.4.4) $\mathrm{KL}(p\|q)$ which measures how "different" the distribution q is from a reference distribution p,

$$\mathrm{H}[p\|q] = \mathrm{H}[p] + \mathrm{KL}(p\|q) \geq \mathrm{H}[p]. \tag{5.33}$$

$\mathrm{KL}(p\|q) \geq 0$ is shown in Problem 5.5

Quite intuitively, the average surprise in samples from p with respect to the distribution q is given by the inherent uncertainty in p and the additional surprise that is due to us assuming the wrong data distribution q. The "closer" q is to the true data distribution p, the smaller is the additional average surprise.

5.4.4 Kullback-Leibler Divergence

As mentioned, the Kullback-Leibler divergence is a (non-metric) measure of distance between distributions. It is defined as follows.

Definition 5.11 (Kullback-Leibler divergence, KL-divergence)**.** Given two distributions p and q, the *Kullback-Leibler divergence* (or *relative entropy*) of q with respect to p,

$$\begin{aligned}\mathrm{KL}(p\|q) &\doteq \mathrm{H}[p\|q] - \mathrm{H}[p] && (5.34)\\ &= \mathbb{E}_{\boldsymbol{\theta}\sim p}[\mathrm{S}[q(\boldsymbol{\theta})] - \mathrm{S}[p(\boldsymbol{\theta})]] && (5.35)\\ &= \mathbb{E}_{\boldsymbol{\theta}\sim p}\left[\log \frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right], && (5.36)\end{aligned}$$

measures how different q is from a reference distribution p.

In words, $\mathrm{KL}(p\|q)$ measures the *additional* expected surprise when observing samples from p that is due to assuming the (wrong) distribution q and which not inherent in the distribution p already.[7]

[7] The KL-divergence only captures the additional expected surprise since the surprise inherent in p (as measured by $\mathrm{H}[p]$) is subtracted.

The KL-divergence has the following properties:

- $\mathrm{KL}(p\|q) \geq 0$ for any distributions p and q (?), **Problem 5.5 (1)**
- $\mathrm{KL}(p\|q) = 0$ if and only if $p = q$ almost surely (?), and **Problem 5.5 (2)**
- there exist distributions p and q such that $\mathrm{KL}(p\|q) \neq \mathrm{KL}(q\|p)$.

The KL-divergence can simply be understood as a shifted version of cross-entropy, which is zero if we consider the divergence between two identical distributions.

We will briefly look at another interpretation for how KL-divergence measures "distance" between distributions. Suppose we are presented with a sequence $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$ of independent samples from either a distribution p or a distribution q, both of which are known. Which of p or q was used to generate the data is, however, unknown to us, and we would like to find out. A natural approach is to choose the distribution whose data likelihood is larger. That is, we choose p if $p(\boldsymbol{\theta}_{1:n}) > q(\boldsymbol{\theta}_{1:n})$ and vice versa. Assuming that the samples are independent and rewriting the inequality slightly, we choose p if

$$\prod_{i=1}^{n} \frac{p(\boldsymbol{\theta}_i)}{q(\boldsymbol{\theta}_i)} > 1, \quad \text{or equivalently if} \quad \sum_{i=1}^{n} \log \frac{p(\boldsymbol{\theta}_i)}{q(\boldsymbol{\theta}_i)} > 0. \tag{5.37}$$

taking the logarithm

Assume without loss of generality that $\boldsymbol{\theta}_i \sim p$. Then, using the law of large numbers (A.36),

$$\frac{1}{n} \sum_{i=1}^{n} \log \frac{p(\boldsymbol{\theta}_i)}{q(\boldsymbol{\theta}_i)} \xrightarrow{a.s.} \mathbb{E}_{\boldsymbol{\theta} \sim p} \left[\log \frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} \right] = \mathrm{KL}(p \| q) \tag{5.38}$$

as $n \to \infty$. Plugging this into our decision criterion from Equation (5.37), we find that

$$\mathbb{E}\left[\sum_{i=1}^{n} \log \frac{p(\boldsymbol{\theta}_i)}{q(\boldsymbol{\theta}_i)} \right] = n \mathrm{KL}(p \| q). \tag{5.39}$$

In this way, $\mathrm{KL}(p \| q)$ measures the observed "distance" between p and q. Recall that assuming $p \neq q$ we have that $\mathrm{KL}(p \| q) > 0$ with probability 1, and therefore we correctly choose p with probability 1 as $n \to \infty$. Moreover, Hoeffding's inequality (A.41) can be used to determine "how quickly" samples converge to this limit, that is, how quickly we can distinguish between p and q.

Example 5.12: KL-divergence of Bernoulli random variables

Suppose we are given two Bernoulli distributions $\mathrm{Bern}(p)$ and $\mathrm{Bern}(q)$. Then, their KL-divergence is

$$\begin{aligned} \mathrm{KL}(\mathrm{Bern}(p) \| \mathrm{Bern}(q)) &= \sum_{x \in \{0,1\}} \mathrm{Bern}(x; p) \log \frac{\mathrm{Bern}(x; p)}{\mathrm{Bern}(x; q)} \\ &= p \log \frac{p}{q} + (1-p) \log \frac{(1-p)}{(1-q)}. \end{aligned} \tag{5.40}$$

Observe that $\mathrm{KL}(\mathrm{Bern}(p) \| \mathrm{Bern}(q)) = 0$ if and only if $p = q$.

Example 5.13: KL-divergence of Gaussians

Suppose we are given two Gaussian distributions $p \doteq \mathcal{N}(\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p)$ and $q \doteq \mathcal{N}(\boldsymbol{\mu}_q, \boldsymbol{\Sigma}_q)$ with dimension d. The KL-divergence of p and q is given by (?)

$$\mathrm{KL}(p\|q) = \frac{1}{2}\left(\mathrm{tr}(\boldsymbol{\Sigma}_q^{-1}\boldsymbol{\Sigma}_p) + (\boldsymbol{\mu}_p - \boldsymbol{\mu}_q)^\top \boldsymbol{\Sigma}_q^{-1}(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q) - d + \log\frac{\det(\boldsymbol{\Sigma}_q)}{\det(\boldsymbol{\Sigma}_p)}\right). \tag{5.41}$$

For independent Gaussians with unit variance, $\boldsymbol{\Sigma}_p = \boldsymbol{\Sigma}_q = \boldsymbol{I}$, the expression simplifies to the squared Euclidean distance,

$$\mathrm{KL}(p\|q) = \frac{1}{2}\|\boldsymbol{\mu}_q - \boldsymbol{\mu}_p\|_2^2. \tag{5.42}$$

If we approximate independent Gaussians with variances σ_i^2,

$$p \doteq \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}\{\sigma_1^2, \ldots, \sigma_d^2\}),$$

by a standard normal distribution, $q \doteq \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, the expression simplifies to

$$\mathrm{KL}(p\|q) = \frac{1}{2}\sum_{i=1}^{d}(\sigma_i^2 + \mu_i^2 - 1 - \log\sigma_i^2). \tag{5.43}$$

Here, the term μ_i^2 penalizes a large mean of p, the term σ_i^2 penalizes a large variance of p, and the term $-\log\sigma_i^2$ penalizes a small variance of p. As expected, $\mathrm{KL}(p\|q)$ is proportional to the amount of information we lose by approximating p with the simpler distribution q.

5.4.5 *Forward and Reverse KL-divergence*

$\mathrm{KL}(p\|q)$ is also called the *forward* (or *inclusive*) KL-divergence. In contrast, $\mathrm{KL}(q\|p)$ is called the *reverse* (or *exclusive*) KL-divergence. Figure 5.9 shows the approximations of a general Gaussian obtained when $\mathcal{Q}$ is the family of diagonal (independent) Gaussians. Thereby,

$$q_1^\star \doteq \arg\min_{q\in\mathcal{Q}} \mathrm{KL}(p\|q) \quad \text{and} \quad q_2^\star \doteq \arg\min_{q\in\mathcal{Q}} \mathrm{KL}(q\|p).$$

$q_1^\star$ is the result when using the forward KL-divergence and $q_2^\star$ is the result when using reverse KL-divergence. It can be seen that the reverse KL-divergence tends to greedily select the mode and underestimating the variance which, in this case, leads to an overconfident predic-

Problem 5.8

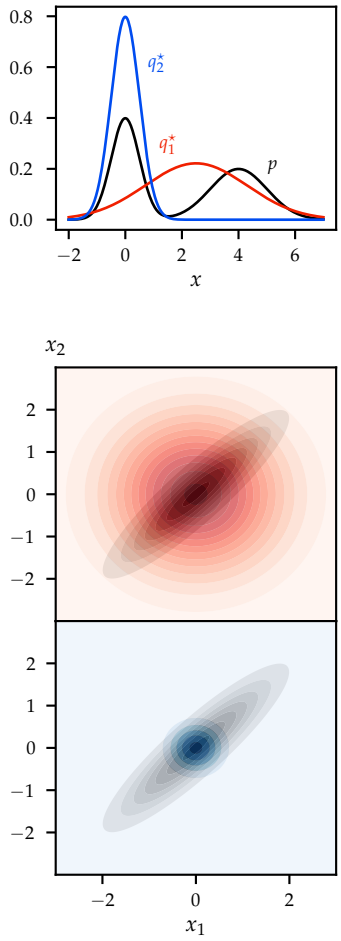


Figure 5.9: Comparison of the **forward** KL-divergence $q_1^\star$ and the **reverse** KL-divergence $q_2^\star$ when used to approximate the **true posterior** p. The first plot shows the PDFs in a one-dimensional feature space where p is a mixture of two univariate Gaussians. The second plot shows contour lines of the PDFs in a two-dimensional feature space where the non-diagonal Gaussian p is approximated by diagonal Gaussians $q_1^\star$ and $q_2^\star$. It can be seen that $q_1^\star$ selects the variance and $q_2^\star$ selects the mode of p. The approximation $q_1^\star$ is more conservative than the (overconfident) approximation $q_2^\star$.

tion. The forward KL-divergence, in contrast, is more conservative and yields what one could consider the "desired" approximation.

Recall that in the blueprint of variational inference (5.25) we used the reverse KL-divergence. This is for computational reasons. Observe that to approximate the KL-divergence $\mathrm{KL}(p\|q)$ using Monte Carlo sampling, we would need to obtain samples from p yet p is the intractable posterior distribution which we were trying to approximate in the first place. Crucially, observe that if the true posterior $p(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n})$ is in the variational family $\mathcal{Q}$, then

$$\arg\min_{q \in \mathcal{Q}} \mathrm{KL}(q \| p(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n})) \overset{a.s.}{=} p(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n}), \tag{5.44}$$

as $\min_{q \in \mathcal{Q}} \mathrm{KL}(q \| p(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n})) \overset{a.s.}{=} 0$

so minimizing reverse-KL still recovers the true posterior almost surely.

Remark 5.14: Greediness of reverse-KL

As in the previous example, consider the independent Gaussians

$$p \doteq \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}_{i \in [d]}\{\sigma_i^2\}),$$

which we seek to approximate by a standard normal distribution $q \doteq \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Using (5.41), we obtain for the reverse KL-divergence,

$$\begin{aligned} \mathrm{KL}(q\|p) &= \frac{1}{2}\left(\mathrm{tr}\Big(\mathrm{diag}\{\sigma_i^{-2}\}\Big) + \boldsymbol{\mu}^\top \mathrm{diag}\{\sigma_i^{-2}\}\boldsymbol{\mu} - d \right. \\ &\qquad \left. + \log\det\Big(\mathrm{diag}\{\sigma_i^2\}\Big)\right) \\ &= \frac{1}{2}\sum_{i=1}^{d}\left(\sigma_i^{-2} + \frac{\mu_i^2}{\sigma_i^2} - 1 + \log\sigma_i^2\right). \end{aligned} \tag{5.45}$$

Here, σ_i^{-2} penalizes small variance, μ_i^2/σ_i^2 penalizes a large mean, and $\log\sigma_i^2$ penalizes large variance. Compare this to the expression for the forward KL-divergence $\mathrm{KL}(p\|q)$ that we have seen in Equation (5.43). In particular, observe that reverse-KL penalizes large variance less strongly than forward-KL.

Note, however, that reverse-KL is not greedy in the same sense as Laplace approximation, as it does still take the variance into account and does not *purely* match the mode of p.

5.4.6 *Interlude: Minimizing Forward KL-Divergence*

Before completing the blueprint of variational inference in Section 5.5 by showing how *reverse-KL* can be efficiently minimized, we will digress briefly and relate minimizing *forward-KL* to two other well-known

inference algorithms. This discussion will deepen our understanding of the KL-divergence and its role in probabilistic inference, but feel free to skip ahead to Section 5.5 if you are eager to complete the blueprint.

Minimizing forward-KL as maximum likelihood estimation: First, we observe that minimizing the forward KL-divergence is equivalent to maximum likelihood estimation on an infinitely large sample size. The classical application of this result is in the setting where $p(\boldsymbol{x})$ is a generative model, and we aim to estimate its density with the parameterized model $q_{\boldsymbol{\lambda}}$.

Lemma 5.15 (Forward KL-divergence as MLE)**.** *Given some generative model $p(\boldsymbol{x})$ and a likelihood $q_{\boldsymbol{\lambda}}(\boldsymbol{x}) = q(\boldsymbol{x} \mid \boldsymbol{\lambda})$ (that we use to approximate the true data distribution), we have*

$$\arg\min_{\boldsymbol{\lambda}\in\Lambda} \mathrm{KL}(p\|q_{\boldsymbol{\lambda}}) \overset{a.s.}{=} \arg\max_{\boldsymbol{\lambda}\in\Lambda} \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} \log q(\boldsymbol{x}_i \mid \boldsymbol{\lambda}), \tag{5.46}$$

where $\boldsymbol{x}_i \overset{\text{iid}}{\sim} p$ are independent samples from the true data distribution.

Proof.

$$\begin{aligned}
\mathrm{KL}(p\|q_{\boldsymbol{\lambda}}) &= \mathrm{H}[p\|q_{\boldsymbol{\lambda}}] - \mathrm{H}[p] && \text{using the definition of KL-divergence (5.34)}\\
&= \mathbb{E}_{(\boldsymbol{x},y)\sim p}[-\log q(\boldsymbol{x} \mid \boldsymbol{\lambda})] + \text{const} && \text{dropping } \mathrm{H}[p] \text{ and using the definition of cross-entropy (5.32)}\\
&\overset{a.s.}{=} -\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} \log q(\boldsymbol{x}_i \mid \boldsymbol{\lambda}) + \text{const} && \text{using Monte Carlo sampling, i.e., the law of large numbers (A.36)}
\end{aligned}$$

where $\boldsymbol{x}_i \overset{\text{iid}}{\sim} p$ are independent samples. □

This tells us that *any* maximum likelihood estimate $q_{\boldsymbol{\lambda}}$ minimizes the forward KL-divergence to the empirical data distribution. Note that here, we aim to learn model parameters $\boldsymbol{\lambda}$ for estimating the probability of $\boldsymbol{x}$, whereas in the setting of variational probabilistic inference, we want to learn parameters $\boldsymbol{\lambda}$ of a distribution over $\boldsymbol{\theta}$ and $\boldsymbol{\theta}$ parameterizes a *distribution over* $\boldsymbol{x}$. This interpretation is therefore not immediately useful for probabilistic inference (i.e., in the setting where p is a posterior distribution over model parameters $\boldsymbol{\theta}$) as a maximum likelihood estimate requires i.i.d. samples from p which we cannot easily obtain in this case.[8]

[8] It is possible to obtain "approximate" samples using Markov chain Monte Carlo (MCMC) methods which we discuss in Chapter 6.

Example 5.16: Minimizing cross-entropy

Minimizing the KL-divergence between p and $q_{\boldsymbol{\lambda}}$ is equivalent to minimizing cross-entropy since $\mathrm{KL}(p\|q_{\boldsymbol{\lambda}}) = \mathrm{H}[p\|q_{\boldsymbol{\lambda}}] - \mathrm{H}[p]$ and $\mathrm{H}[p]$ is constant with respect to p.

Lets consider an example in a binary classification problem with the label $y \in \{0, 1\}$ and predicted class probability $\hat{y} \in [0, 1]$ for some fixed input. It is natural to use cross-entropy as a measure of dissimilarity between y and $\hat{y}$,

$$\begin{aligned} \ell_{\text{bce}}(\hat{y}; y) &\doteq \mathrm{H}[\text{Bern}(y) \| \text{Bern}(\hat{y})] \\ &= -\sum_{x \in \{0,1\}} \text{Bern}(x; y) \log \text{Bern}(x; \hat{y}) \\ &= -y \log \hat{y} - (1 - y) \log(1 - \hat{y}). \end{aligned} \tag{5.47}$$

This loss function is also known as the *binary cross-entropy loss* and we will discuss it in more detail in Section 7.1.3 in the context of neural networks.

Minimizing forward-KL as moment matching: Now to a second interpretation of minimizing forward-KL. *Moment matching* (also known as the *method of moments*) is a technique for approximating an unknown distribution p with a parameterized distribution $q_{\boldsymbol{\lambda}}$ where $\boldsymbol{\lambda}$ is chosen such that $q_{\boldsymbol{\lambda}}$ matches the (estimated) moments of p. For example, given the estimates $\boldsymbol{a}$ and $\boldsymbol{B}$ of the first and second moment of p,[9] and if $q_{\boldsymbol{\lambda}}$ is a Gaussian with parameters $\boldsymbol{\lambda} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$, then moment matching chooses $\boldsymbol{\lambda}$ as the solution to

$$\begin{aligned} \mathbb{E}_p[\boldsymbol{\theta}] \approx \boldsymbol{a} &\overset{!}{=} \boldsymbol{\mu} = \mathbb{E}_{q_{\boldsymbol{\lambda}}}[\boldsymbol{\theta}] \\ \mathbb{E}_p\left[\boldsymbol{\theta}\boldsymbol{\theta}^\top\right] \approx \boldsymbol{B} &\overset{!}{=} \boldsymbol{\Sigma} + \boldsymbol{\mu}\boldsymbol{\mu}^\top = \mathbb{E}_{q_{\boldsymbol{\lambda}}}\left[\boldsymbol{\theta}\boldsymbol{\theta}^\top\right]. \end{aligned}$$

[9] These estimates are computed using the samples from p. For example, using a sample mean and a sample variance to compute the estimates of the first and second moment.

using the definition of variance (1.35)

In general the number of moments to be matched (i.e., the number of equations) is adjusted such that it is equal to the number of parameters to be estimated. We will see now that the "matching" of moments is also ensured when $q_{\boldsymbol{\lambda}}$ is obtained by minimizing the forward KL-divergence within the family of Gaussians.

The Gaussian PDF can be expressed as

$$\mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{Z(\boldsymbol{\lambda})} \exp(\boldsymbol{\lambda}^\top \boldsymbol{s}(\boldsymbol{\theta})) \quad \text{where} \tag{5.48}$$

$$\boldsymbol{\lambda} \doteq \begin{bmatrix} \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} \\ \text{vec}[\boldsymbol{\Sigma}^{-1}] \end{bmatrix} \tag{5.49}$$

$$\boldsymbol{s}(\boldsymbol{\theta}) \doteq \begin{bmatrix} \boldsymbol{\theta} \\ \text{vec}[-\frac{1}{2}\boldsymbol{\theta}\boldsymbol{\theta}^\top] \end{bmatrix} \tag{5.50}$$

and $Z(\boldsymbol{\lambda}) \doteq \int \exp(\boldsymbol{\lambda}^\top \boldsymbol{s}(\boldsymbol{\theta}))\, d\boldsymbol{\theta}$, and we will confirm this in just a moment.[10] The family of distributions with densities of the form (5.48) — with an additional scaling constant $h(\boldsymbol{\theta})$ which is often 1 — is called

[10] Given a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times m}$, we use

$$\text{vec}[\boldsymbol{A}] \in \mathbb{R}^{n \cdot m}$$

to denote the row-by-row concatenation of $\boldsymbol{A}$ yielding a vector of length $n \cdot m$.

the *exponential family* of distributions. Here, $\boldsymbol{s}(\boldsymbol{\theta})$ are the *sufficient statistics*, $\boldsymbol{\lambda}$ are called the *natural parameters*, and $Z(\boldsymbol{\lambda})$ is the normalizing constant. In this context, $Z(\boldsymbol{\lambda})$ is often called the *partition function*.

To see that the Gaussian is indeed part of the exponential family as promised in Equations (5.49) and (5.50), consider

$$\begin{aligned}
\mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) &\propto \exp\left(-\frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu})\right) \\
&\propto \exp\left(\operatorname{tr}\left(-\frac{1}{2}\boldsymbol{\theta}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\theta}\right) + \boldsymbol{\theta}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right) && \text{expanding the inner product and using that } \operatorname{tr}(x) = x \text{ for all } x \in \mathbb{R} \\
&= \exp\left(\operatorname{tr}\left(-\frac{1}{2}\boldsymbol{\theta}\boldsymbol{\theta}^\top \boldsymbol{\Sigma}^{-1}\right) + \boldsymbol{\theta}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right) && \text{using that the trace is invariant under cyclic permutations} \\
&= \exp\left(\operatorname{vec}[-\tfrac{1}{2}\boldsymbol{\theta}\boldsymbol{\theta}^\top]^\top \operatorname{vec}[\boldsymbol{\Sigma}^{-1}] + \boldsymbol{\theta}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right). && \text{using } \operatorname{tr}(\boldsymbol{A}\boldsymbol{B}) = \operatorname{vec}[\boldsymbol{A}]^\top \operatorname{vec}[\boldsymbol{B}] \text{ for any } \boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}
\end{aligned}$$

This allows us to express the forward KL-divergence as

$$\begin{aligned}
\mathrm{KL}(p \| q_\lambda) &= \int p(\boldsymbol{\theta}) \log \frac{p(\boldsymbol{\theta})}{q_\lambda(\boldsymbol{\theta})}\, d\boldsymbol{\theta} \\
&= -\int p(\boldsymbol{\theta}) \cdot \boldsymbol{\lambda}^\top \boldsymbol{s}(\boldsymbol{\theta})\, d\boldsymbol{\theta} + \log Z(\boldsymbol{\lambda}) + \text{const.} && \text{using that } \textstyle\int p(\boldsymbol{\theta}) \log p(\boldsymbol{\theta})\, d\boldsymbol{\theta} \text{ is constant}
\end{aligned}$$

Differentiating with respect to the natural parameters $\boldsymbol{\lambda}$ gives

$$\begin{aligned}
\nabla_\lambda \mathrm{KL}(p \| q_\lambda) &= -\int p(\boldsymbol{\theta}) \boldsymbol{s}(\boldsymbol{\theta})\, d\boldsymbol{\theta} + \frac{1}{Z(\boldsymbol{\lambda})} \int \boldsymbol{s}(\boldsymbol{\theta}) \exp(\boldsymbol{\lambda}^\top \boldsymbol{s}(\boldsymbol{\theta}))\, d\boldsymbol{\theta} \\
&= -\mathbb{E}_{\boldsymbol{\theta}\sim p}[\boldsymbol{s}(\boldsymbol{\theta})] + \mathbb{E}_{\boldsymbol{\theta}\sim q_\lambda}[\boldsymbol{s}(\boldsymbol{\theta})].
\end{aligned}$$

Hence, for any minimizer of $\mathrm{KL}(p \| q_\lambda)$, we have that the sufficient statistics under p and q_λ match:

$$\mathbb{E}_p[\boldsymbol{s}(\boldsymbol{\theta})] = \mathbb{E}_{q_\lambda}[\boldsymbol{s}(\boldsymbol{\theta})]. \tag{5.51}$$

Therefore, in the Gaussian case,

$$\mathbb{E}_p[\boldsymbol{\theta}] = \mathbb{E}_{q_\lambda}[\boldsymbol{\theta}] \quad \text{and} \quad \mathbb{E}_p\left[-\tfrac{1}{2}\boldsymbol{\theta}\boldsymbol{\theta}^\top\right] = \mathbb{E}_{q_\lambda}\left[-\tfrac{1}{2}\boldsymbol{\theta}\boldsymbol{\theta}^\top\right],$$

implying that

$$\mathbb{E}_p[\boldsymbol{\theta}] = \boldsymbol{\mu} \quad \text{and} \quad \operatorname{Var}_p[\boldsymbol{\theta}] = \mathbb{E}_p\left[\boldsymbol{\theta}\boldsymbol{\theta}^\top\right] - \mathbb{E}_p[\boldsymbol{\theta}] \cdot \mathbb{E}_p[\boldsymbol{\theta}]^\top = \boldsymbol{\Sigma} \tag{5.52}$$

using Equation (1.35)

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the mean and variance of the approximation q_λ, respectively. That is, a Gaussian q_λ minimizing $\mathrm{KL}(p \| q_\lambda)$ has the same first and second moment as p. Combining this insight with our observation from Equation (5.46) that minimizing forward-KL is equivalent to maximum likelihood estimation, we see that if we use MLE to fit a Gaussian to given data, this Gaussian will eventually match the first and second moments of the data distribution.

5.5 Evidence Lower Bound

Let us return to the blueprint of variational inference from Section 5.3. To complete this blueprint, it remains to show that the reverse KL-divergence can be minimized efficiently. We have

$$\begin{aligned}
\mathrm{KL}(q \| p(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n})) &= \mathbb{E}_{\boldsymbol{\theta} \sim q}\left[\log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n})}\right] && \text{using the definition of the KL-divergence (5.34)} \\
&= \mathbb{E}_{\boldsymbol{\theta} \sim q}\left[\log \frac{p(y_{1:n} \mid \boldsymbol{x}_{1:n}) q(\boldsymbol{\theta})}{p(y_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n})}\right] && \text{using the definition of conditional probability (1.8)} \\
&= \log p(y_{1:n} \mid \boldsymbol{x}_{1:n}) && \text{using linearity of expectation (1.20)} \\
&\qquad \underbrace{-\mathbb{E}_{\boldsymbol{\theta} \sim q}[\log p(y_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n})] - \mathrm{H}[q]}_{-L(q, p; \mathcal{D}_n)}
\end{aligned}$$

where $L(q, p; \mathcal{D}_n)$ is called the *evidence lower bound* (ELBO) given the data $\mathcal{D}_n = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. This gives the relationship

$$L(q, p; \mathcal{D}_n) = \underbrace{\log p(y_{1:n} \mid \boldsymbol{x}_{1:n})}_{\text{const}} - \mathrm{KL}(q \| p(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n})). \tag{5.53}$$

Thus, maximizing the ELBO coincides with minimizing reverse-KL.

Maximizing the ELBO,

$$L(q, p; \mathcal{D}_n) \doteq \mathbb{E}_{\boldsymbol{\theta} \sim q}[\log p(y_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n})] + \mathrm{H}[q], \tag{5.54}$$

selects q that has large joint likelihood $p(y_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n})$ and large entropy $\mathrm{H}[q]$. The ELBO can also be expressed in various other forms:

$$\begin{aligned}
L(q, p; \mathcal{D}_n) &= \mathbb{E}_{\boldsymbol{\theta} \sim q}[\log p(y_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}) - \log q(\boldsymbol{\theta})] && \text{(5.55a)} && \text{using the definition of entropy (5.27)} \\
&= \mathbb{E}_{\boldsymbol{\theta} \sim q}[\log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log q(\boldsymbol{\theta})] && \text{(5.55b)} && \text{using the product rule (1.11)} \\
&= \underbrace{\mathbb{E}_{\boldsymbol{\theta} \sim q}[\log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})]}_{\text{log-likelihood}} \underbrace{- \mathrm{KL}(q \| p(\cdot))}_{\text{proximity to prior}}. && \text{(5.55c)} && \text{using the definition of KL-divergence (5.34)}
\end{aligned}$$

where we denote by $p(\cdot)$ the prior distribution. Equation (5.55c) highlights the connection to probabilistic inference, namely that maximizing the ELBO selects a variational distribution q that is close to the prior distribution $p(\cdot)$ while also maximizing the *average* likelihood of the data $p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})$ for $\boldsymbol{\theta} \sim q$. This is in contrast to maximum a posteriori estimation, which picks a *single* model $\boldsymbol{\theta}$ that maximizes the likelihood and proximity to the prior. As an example, let us look at the case where the prior is noninformative, i.e., $p(\cdot) \propto 1$. In this case, the ELBO simplifies to $\mathbb{E}_{\boldsymbol{\theta} \sim q}[\log p(y_{1:n}, \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})] + \mathrm{H}[q] + \text{const}$. That is, maximizing the ELBO maximizes the average likelihood of the data under the variational distribution q while regularizing q to have high

entropy. Why is it reasonable to maximize the entropy of q? Consider two distributions q_1 and q_2 under which the data is "equally" likely and which are "equally" close to the prior. Maximizing the entropy selects the distribution that exhibits the most uncertainty which is in accordance with the maximum entropy principle.[11]

[11] In Section 1.2.1, we discussed the maximum entropy principle and the related principle of indifference at length. In simple terms, the maximum entropy principle states that without information, we should choose the distribution that is maximally uncertain.

Recalling that KL-divergence is non-negative, it follows from Equation (5.53) that the evidence lower bound is a (uniform[12]) lower bound to the *evidence* $\log p(y_{1:n} \mid \boldsymbol{x}_{1:n})$:

$$\log p(y_{1:n} \mid \boldsymbol{x}_{1:n}) \geq L(q, p; \mathcal{D}_n). \tag{5.56}$$

[12] That is, the bound holds for any variational distribution q (with full support).

This indicates that maximizing the evidence lower bound is an adequate method of model selection which can be used instead of maximizing the evidence (marginal likelihood) directly (as was discussed in Section 4.4.2). Note that this inequality lower bounds the logarithm of an integral by an expectation of a logarithm over some variational distribution q. Hence, the ELBO is a family of lower bounds — one for each variational distribution. Such inequalities are called variational inequalities.

Example 5.17: Gaussian VI vs Laplace approximation

Consider maximizing the ELBO within the variational family of Gaussians $\mathcal{Q} \doteq \{q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma})\}$. How does this relate to Laplace approximation which also fits a Gaussian approximation to the posterior

$$p(\boldsymbol{\theta} \mid \mathcal{D}) \propto p(y_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n})?$$

It turns out that both approximations are closely related. Indeed, it can be shown that while the Laplace approximation is fitted *locally* at the MAP estimate $\hat{\boldsymbol{\theta}}$, satisfying

$$\begin{aligned} \mathbf{0} &= \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p(y_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}) \\ \boldsymbol{\Sigma}^{-1} &= -\boldsymbol{H}_{\boldsymbol{\theta}} \log p(y_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n})|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}, \end{aligned} \tag{5.57}$$

Gaussian variational inference satisfies the conditions of the Laplace approximation *on average* with respect to the approximation q (?):

Problem 5.10

$$\begin{aligned} \mathbf{0} &= \mathbb{E}_{\boldsymbol{\theta}\sim q}[\boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p(y_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n})] \\ \boldsymbol{\Sigma}^{-1} &= -\mathbb{E}_{\boldsymbol{\theta}\sim q}[\boldsymbol{H}_{\boldsymbol{\theta}} \log p(y_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n})]. \end{aligned} \tag{5.58}$$

For this reason, the Gaussian variational approximation does not suffer from the same overconfidence as the Laplace approximation.[13]

[13] see Figure 5.1

Example 5.18: ELBO for Bayesian logistic regression

Recall that Bayesian logistic regression uses the prior distribution $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.[14]

[14] We omit the scaling factor σ_p^2 here for simplicity.

Suppose we use the variational family $\mathcal{Q}$ of all diagonal Gaussians from Example 5.5. We have already seen in Equation (5.43) that for a prior $p \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and a variational distribution

$$q_{\boldsymbol{\lambda}} \sim \mathcal{N}(\boldsymbol{\mu}, \operatorname{diag}_{i \in [d]}\{\sigma_i^2\}),$$

we have

$$\mathrm{KL}(q \| p(\cdot)) = \frac{1}{2} \sum_{i=1}^{d} (\sigma_i^2 + \mu_i^2 - 1 - \log \sigma_i^2).$$

It remains to find the expected likelihood under models from our approximate posterior:

$$\begin{aligned} \mathbb{E}_{\boldsymbol{w} \sim q_{\boldsymbol{\lambda}}}[\log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{w})] &= \mathbb{E}_{\boldsymbol{w} \sim q_{\boldsymbol{\lambda}}}\left[\sum_{i=1}^{n} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w})\right] && \text{using independence of the data} \\ &= \mathbb{E}_{\boldsymbol{w} \sim q_{\boldsymbol{\lambda}}}\left[-\sum_{i=1}^{n} \ell_{\log}(\boldsymbol{w}; \boldsymbol{x}_i, y_i)\right]. && \text{substituting the logistic loss (5.13)} \end{aligned} \tag{5.59}$$

5.5.1 *Gradient of Evidence Lower Bound*

We have yet to discuss how the optimization problem of maximizing the ELBO can be solved efficiently. A suitable tool is stochastic gradient descent (SGD), however, SGD requires unbiased gradient estimates of the loss $\ell(\boldsymbol{\lambda}; \mathcal{D}_n) \doteq -L(q_{\boldsymbol{\lambda}}, p; \mathcal{D}_n)$. That is, we need to obtain gradient estimates of

$$\nabla_{\boldsymbol{\lambda}} L(q_{\boldsymbol{\lambda}}, p; \mathcal{D}_n) = \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{\boldsymbol{\theta} \sim q_{\boldsymbol{\lambda}}}[\log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})] - \nabla_{\boldsymbol{\lambda}} \mathrm{KL}(q_{\boldsymbol{\lambda}} \| p(\cdot)). \tag{5.60}$$

using the definition of the ELBO (5.55c)

Typically, the KL-divergence (and its gradient) can be computed exactly for commonly used variational families. For example, we have already seen a closed-form expression of the KL-divergence for Gaussians in Equation (5.41).

Obtaining the gradient of the expected log-likelihood is more difficult. This is because the expectation integrates over the measure $q_{\boldsymbol{\lambda}}$, which depends on the variational parameters $\boldsymbol{\lambda}$. Thus, we cannot move the gradient operator inside the expectation as commonly done (cf. Appendix A.1.5). There are two main techniques which are used to

rewrite the gradient in such a way that Monte Carlo sampling becomes possible.

One approach is to use *score gradients* via the "score function trick":

$$\begin{aligned}&\nabla_\lambda \mathbb{E}_{\theta\sim q_\lambda}[\log p(y_{1:n} \mid x_{1:n}, \theta)] \\ &\quad = \mathbb{E}_{\theta\sim q_\lambda}[\log p(y_{1:n} \mid x_{1:n}, \theta) \underbrace{\nabla_\lambda \log q_\lambda(\theta)}_{\text{score function}}],\end{aligned} \tag{5.61}$$

which we introduce in Section 12.3.2 in the context of reinforcement learning. More common in the context of variational inference is the so-called "reparameterization trick".

Theorem 5.19 (Reparameterization trick)**.** *Given a random variable $\varepsilon \sim \phi$ (which is independent of λ) and given a differentiable and invertible function $g : \mathbb{R}^d \to \mathbb{R}^d$. We let $\theta \doteq g(\varepsilon; \lambda)$. Then,*

$$q_\lambda(\theta) = \phi(\varepsilon) \cdot |\det(D_\varepsilon g(\varepsilon; \lambda))|^{-1}, \tag{5.62}$$

$$\mathbb{E}_{\theta\sim q_\lambda}[f(\theta)] = \mathbb{E}_{\varepsilon\sim\phi}[f(g(\varepsilon; \lambda))] \tag{5.63}$$

for a "nice" function $f : \mathbb{R}^d \to \mathbb{R}^e$.

Proof. By the change of variables formula (1.43) and using $\varepsilon = g^{-1}(\theta; \lambda)$,

$$\begin{aligned}q_\lambda(\theta) &= \phi(\varepsilon) \cdot \left|\det\Big(D_\theta g^{-1}(\theta; \lambda)\Big)\right| \\ &= \phi(\varepsilon) \cdot \left|\det\Big((D_\varepsilon g(\varepsilon; \lambda))^{-1}\Big)\right| && \text{by the inverse function theorem, } Dg^{-1}(y) = Dg(x)^{-1} \\ &= \phi(\varepsilon) \cdot |\det(D_\varepsilon g(\varepsilon; \lambda))|^{-1}. && \text{using } \det(A^{-1}) = \det(A)^{-1}\end{aligned}$$

Equation (5.63) is a direct consequence of the law of the unconscious statistician (1.22). □

In other words, the reparameterization trick allows a change of "densities" by finding a function $g(\cdot; \lambda)$ and a reference density ϕ such that $q_\lambda = g(\cdot; \lambda)_\sharp \phi$ is the pushforward of ϕ under perturbation g. Applying the reparameterization trick, we can swap the order of gradient and expectation,

$$\nabla_\lambda \mathbb{E}_{\theta\sim q_\lambda}[f(\theta)] = \mathbb{E}_{\varepsilon\sim\phi}[\nabla_\lambda f(g(\varepsilon; \lambda))]. \tag{5.64}$$

using Equation (A.5)

We call a distribution q_λ *reparameterizable* if it admits reparameterization, i.e., if we can find g and a suitable reference density ϕ which is independent of λ.

Example 5.20: Reparameterization trick for Gaussians

Suppose we use a Gaussian variational approximation,

$$q_\lambda(\boldsymbol{\theta}) \doteq \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where we assume $\boldsymbol{\Sigma}$ to have full rank (i.e., be invertible). We have seen in Equation (1.78) that a Gaussian random vector $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ following a standard normal distribution can be transformed to follow the Gaussian distribution q_λ by using the linear transformation,

$$\boldsymbol{\theta} = g(\varepsilon; \boldsymbol{\lambda}) \doteq \boldsymbol{\Sigma}^{1/2}\boldsymbol{\varepsilon} + \boldsymbol{\mu}. \tag{5.65}$$

In particular, we have

$$\boldsymbol{\varepsilon} = g^{-1}(\boldsymbol{\theta}; \boldsymbol{\lambda}) = \boldsymbol{\Sigma}^{-1/2}(\boldsymbol{\theta} - \boldsymbol{\mu}) \quad \text{and} \tag{5.66}$$

by solving Equation (5.65) for ε

$$\phi(\boldsymbol{\varepsilon}) = q_\lambda(\boldsymbol{\theta}) \cdot \left|\det\left(\boldsymbol{\Sigma}^{1/2}\right)\right|. \tag{5.67}$$

using the reparameterization trick (i.e., the change of variables formula) (5.62)

In the following, we write $\boldsymbol{C} \doteq \boldsymbol{\Sigma}^{1/2}$. Let us now derive the gradient estimate for the evidence lower bound assuming the Gaussian variational approximation from Example 5.20. This approach extends to any reparameterizable distribution.

$$\begin{aligned}
&\nabla_\lambda \, \mathbb{E}_{\boldsymbol{\theta}\sim q_\lambda}[\log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})] \\
&= \nabla_{\boldsymbol{C},\boldsymbol{\mu}} \, \mathbb{E}_{\boldsymbol{\varepsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\Big[\log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{C}\boldsymbol{\varepsilon}+\boldsymbol{\mu}}\Big] && (5.68) \\
&\qquad \text{using the reparameterization trick (5.63)} \\
&= n \cdot \nabla_{\boldsymbol{C},\boldsymbol{\mu}} \, \mathbb{E}_{\boldsymbol{\varepsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\frac{1}{n}\sum_{i=1}^{n} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{C}\boldsymbol{\varepsilon}+\boldsymbol{\mu}}\right] \\
&\qquad \text{using independence of the data and extending with } {}^{n}/_{n} \\
&= n \cdot \nabla_{\boldsymbol{C},\boldsymbol{\mu}} \, \mathbb{E}_{\boldsymbol{\varepsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\mathbb{E}_{i\sim\mathrm{Unif}([n])}\Big[\log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{C}\boldsymbol{\varepsilon}+\boldsymbol{\mu}}\Big] \\
&\qquad \text{interpreting the sum as an expectation} \\
&= n \cdot \mathbb{E}_{\boldsymbol{\varepsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\mathbb{E}_{i\sim\mathrm{Unif}([n])}\Big[\nabla_{\boldsymbol{C},\boldsymbol{\mu}} \, \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{C}\boldsymbol{\varepsilon}+\boldsymbol{\mu}}\Big] && (5.69) \\
&\qquad \text{using Equation (A.5)} \\
&\approx n \cdot \frac{1}{m}\sum_{j=1}^{m} \nabla_{\boldsymbol{C},\boldsymbol{\mu}} \, \log p(y_{i_j} \mid x_{i_j}, \boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{C}\boldsymbol{\varepsilon}_j+\boldsymbol{\mu}} && (5.70) \\
&\qquad \text{using Monte Carlo sampling}
\end{aligned}$$

where $\boldsymbol{\varepsilon}_j \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and $i_j \overset{\text{iid}}{\sim} \mathrm{Unif}([n])$. This yields an unbiased gradient estimate, which we can use with stochastic gradient descent to maximize the evidence lower bound. We have successfully recast the difficult problems of learning and inference as an optimization problem!

The procedure of approximating the true posterior using a variational posterior by maximizing the evidence lower bound using stochastic optimization is also called *black box stochastic variational inference* (Ranganath et al., 2014; Titsias and Lázaro-Gredilla, 2014; Duvenaud and

Adams, 2015). The only requirement is that we can obtain unbiased gradient estimates from the evidence lower bound (and the likelihood). We have just discussed one of many approaches to obtain such gradient estimates (Mohamed et al., 2020). If we use the variational family of diagonal Gaussians, we only require twice as many parameters as other inference techniques like MAP estimation. The performance can be improved by using natural gradients and variance reduction techniques for the gradient estimates such as control variates.

5.5.2 *Minimizing Surprise via Exploration and Exploitation*

Now that we have established a way to optimize the ELBO, let us dwell a bit more on its interpretation. Observe that the evidence can also be interpreted as the negative surprise about the observations under the prior distribution $p(\boldsymbol{\theta})$, and by negating Equation (5.56), we obtain the variational upper bound

$$\begin{aligned} \mathrm{S}[p(\boldsymbol{y}_{1:n} \mid \boldsymbol{x}_{1:n})] &\leq \underbrace{\mathbb{E}_{\boldsymbol{\theta}\sim q}[\mathrm{S}[p(\boldsymbol{y}_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n})]]}_{\text{called } \textit{energy}} - \mathrm{H}[q] \\ &= -L(q, p; \mathcal{D}_n). \end{aligned} \tag{5.71}$$

Here, $-L(q, p; \mathcal{D}_n)$ is commonly called the *(variational) free energy* with respect to q. Free energy can also be characterized as

$$-L(q, p; \mathcal{D}_n) = \underbrace{\mathbb{E}_{\boldsymbol{\theta}\sim q}[\mathrm{S}[p(\boldsymbol{y}_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})]]}_{\text{average surprise}} + \underbrace{\mathrm{KL}(q \| p(\cdot))}_{\text{proximity to prior}}, \tag{5.72}$$

analogously to Equation (5.55c)

and therefore its minimization is minimizing the average surprise about the data under the variational distribution q while maximizing proximity to the prior $p(\cdot)$. Systems that minimize the surprise in their observations are widely studied in many areas of science.[15]

[15] Refer to the *free energy principle* which was originally introduced by the neuroscientist Karl Friston (Friston, 2010).

To minimize surprise, the free energy makes apparent a natural tradeoff between two extremes: On the one hand, models $q(\boldsymbol{\theta})$ that "overfit" to observations (e.g., by using a point estimate of $\boldsymbol{\theta}$), and hence, result in a large surprise when new observations deviate from this specific belief. On the other hand, models $q(\boldsymbol{\theta})$ that "underfit" to observations (e.g., by expecting outcomes with equal probability), and hence, any observation results in a non-negligible surprise. Either of these extremes is undesirable.

As we have alluded to previously when introducing the ELBO, the two terms constituting free energy map neatly onto this tradeoff. Therein, the *entropy* (which is maximized) encourages q to have uncertainty, in other words, to "explore" beyond the finite data. In contrast, the *energy* (which is minimized) encourages q to fit the observed data closely, in

other words, to "exploit" the finite data. This tradeoff is ubiquitous in approximations to probabilistic inference that deal with limited computational resources and limited time, and we will encounter it many times more. We will point out these connections as we go along.

Since this tradeoff is so fundamental and appears in many branches of science under different names, it is difficult to give it an appropriate unifying name. The essence of this tradeoff can be captured as a *principle of curiosity and conformity*, which suggests that reasoning under uncertainty requires curiosity to entertain and pursue alternative explanations of the data *and* conformity to make consistent predictions.

Discussion

We have explored variational inference, where we approximate the intractable posterior distribution of probabilistic inference with a simpler distribution. We operationalized this idea by turning the inference problem, which requires computing high-dimensional integrals, into a tractable optimization problem. Gaussians are frequently used as variational distributions due to their versatility and compact representation.

Nevertheless, recall from Figure 5.4 that while the estimation error in variational inference can be small, choosing a variational family that is too simple can lead to a large approximation error. We have seen that for posteriors that are multimodal or have heavy tails, Gaussians may not provide a good approximation. In the next chapter, we will explore alternative techniques for approximate inference that can handle more complex posteriors.

Problems

5.1. Logistic loss.

1. Derive the gradient of $\ell_{\log}$ as given in Equation (5.14).
2. Show that

$$\boldsymbol{H}_{\boldsymbol{w}} \ell_{\log}(\boldsymbol{w}^\top \boldsymbol{x}; y) = \boldsymbol{x}\boldsymbol{x}^\top \cdot \sigma(\boldsymbol{w}^\top \boldsymbol{x}) \cdot (1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x})). \tag{5.73}$$

   Hint: Begin by deriving the first derivative of the logistic function, and use the chain rule of multivariate calculus,

$$\underbrace{\boldsymbol{D}_{\boldsymbol{x}}(\boldsymbol{f} \circ \boldsymbol{g})}_{\mathbb{R}^n \to \mathbb{R}^{m \times n}} = \underbrace{(\boldsymbol{D}_{\boldsymbol{g}(\boldsymbol{x})} \boldsymbol{f} \circ \boldsymbol{g}) \cdot \boldsymbol{D}_{\boldsymbol{x}} \boldsymbol{g}}_{\mathbb{R}^n \to \mathbb{R}^{m \times k} \cdot \mathbb{R}^{k \times n}} \tag{5.74}$$

   where $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^k$ *and* $\boldsymbol{f} : \mathbb{R}^k \to \mathbb{R}^m$.
3. Is the logistic loss $\ell_{\log}$ convex in $\boldsymbol{w}$?

5.2. Gaussian process classification.

In this exercise, we will study the use of Gaussian processes for classification tasks, commonly called *Gaussian process classification* (GPC). Linear logistic regression is extended to GPC by replacing the Gaussian prior over weights with a GP prior on f,

$$f \sim \mathcal{GP}(0, k), \quad y \mid \boldsymbol{x}, f \sim \text{Bern}(\sigma(f(\boldsymbol{x}))) \tag{5.75}$$

where $\sigma : \mathbb{R} \to (0, 1)$ is some logistic-type function. Note that Bayesian logistic regression is the special case where k is the linear kernel and σ is the logistic function. This is analogous to the relationship of Bayesian linear regression and Gaussian process regression.

In the GP regression setting of Chapter 4, y_i was assumed to be a noisy observation of $f(\boldsymbol{x}_i)$. In the classification setting, we now have that $y_i \in \{-1, +1\}$ is a binary class label and $f(\boldsymbol{x}_i) \in \mathbb{R}$ is a latent value. We study the setting where $\sigma(z) = \Phi(z; 0, \sigma_{\text{n}}^2)$ is the CDF of a univariate Gaussian with mean 0 and variance σ_{n}^2, also called a *probit likelihood*.

To make probabilistic predictions for a query point $\boldsymbol{x}^\star$, we first compute the distribution of the latent variable $f^\star$,

$$p(f^\star \mid \boldsymbol{x}_{1:n}, y_{1:n}, \boldsymbol{x}^\star) = \int p(f^\star \mid \boldsymbol{x}_{1:n}, \boldsymbol{x}^\star, \boldsymbol{f}) p(\boldsymbol{f} \mid x_{1:n}, y_{1:n}) \, d\boldsymbol{f} \tag{5.76}$$

using sum rule (1.7) and product rule (1.11), and $f^\star \perp y_{1:n} \mid \boldsymbol{f}$

where $p(\boldsymbol{f} \mid \boldsymbol{x}_{1:n}, y_{1:n})$ is the posterior over the latent variables.

1. Assuming that we can efficiently compute $p(f^\star \mid \boldsymbol{x}_{1:n}, y_{1:n}, \boldsymbol{x}^\star)$ (approximately), describe how we can find the predictive posterior $p(y^\star = +1 \mid \boldsymbol{x}_{1:n}, y_{1:n}, \boldsymbol{x}^\star)$.
2. The posterior over the latent variables is not a Gaussian as we used a non-Gaussian likelihood, and hence, the integral of the latent predictive posterior (5.76) is analytically intractable. A common technique is to approximate the latent posterior $p(\boldsymbol{f} \mid \boldsymbol{x}_{1:n}, y_{1:n})$ with a Gaussian using a Laplace approximation $q \doteq \mathcal{N}(\hat{\boldsymbol{f}}, \boldsymbol{\Lambda}^{-1})$. It is generally not possible to obtain an analytical representation of the mode of the Laplace approximation $\hat{\boldsymbol{f}}$. Instead, $\hat{\boldsymbol{f}}$ is commonly found using a second-order optimization scheme such as Newton's method.
   (a) Find the precision matrix $\boldsymbol{\Lambda}$ of the Laplace approximation.
   Hint: Observe that for a label $y_i \in \{-1, +1\}$, the probability of a correct classification given the latent value f_i is $p(y_i \mid f_i) = \sigma(y_i f_i)$, where we use the symmetry of the probit likelihood around 0.
   (b) Assume that $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}'$ is the linear kernel ($\sigma_{\text{p}} = 1$) and that σ is the logistic function (5.9). Show for this setting that the matrix $\boldsymbol{\Lambda}$ derived in (a) is equivalent to the precision matrix

$\boldsymbol{\Lambda}'$ of the Laplace approximation of Bayesian logistic regression (5.17).[16] You may assume that $\hat{f}_i = \hat{\boldsymbol{w}}^\top \boldsymbol{x}_i$.
Hint: First derive under which condition $\boldsymbol{\Lambda}$ and $\boldsymbol{\Lambda}'$ are "equivalent".

(c) Observe that the (approximate) latent predictive posterior

$$q(f^\star \mid \boldsymbol{x}_{1:n}, y_{1:n}, \boldsymbol{x}^\star) \doteq \int p(f^\star \mid \boldsymbol{x}_{1:n}, \boldsymbol{x}^\star, \boldsymbol{f}) q(\boldsymbol{f} \mid x_{1:n}, y_{1:n})\, d\boldsymbol{f}$$

which uses the Laplace approximation of the latent posterior is Gaussian.[17] Determine its mean and variance.
Hint: Condition on the latent variables $\boldsymbol{f}$ using the laws of total expectation and variance.

(d) Compare the prediction $p(f^\star \mid \boldsymbol{x}_{1:n}, y_{1:n}, \boldsymbol{x}^\star)$ you obtained in (1) (but now using the Laplace-approximated latent predictive posterior) to the prediction $\sigma(\mathbb{E}_{f^\star \sim q}[f^\star])$. Are they identical? If not, describe how they are different.

3. The use of the probit likelihood may seem arbitrary. Consider the following model which may be more natural,

$$f \sim \mathcal{GP}(0, k), \quad y = \mathbb{1}\{ \underbrace{f(\boldsymbol{x}) + \varepsilon}_{\text{GP regression}} \geq 0\}, \quad \varepsilon \sim \mathcal{N}(0, \sigma_{\text{n}}^2). \tag{5.77}$$

Show that the model from Equation (5.75) using a noise-free latent process with probit likelihood $\Phi(z; 0, \sigma_{\text{n}}^2)$ is equivalent (in expectation over ε) to the model from Equation (5.77).

5.3. Jensen's inequality.

1. Prove the finite form of Jensen's inequality.
Theorem 5.21 (Jensen's inequality, finite form)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Suppose that $\boldsymbol{x}_1, \dots, \boldsymbol{x}_k \in \mathbb{R}^n$ and $\theta_1, \dots, \theta_k \geq 0$ with $\theta_1 + \cdots + \theta_k = 1$. Then,*

$$f(\theta_1 \boldsymbol{x}_1 + \cdots + \theta_k \boldsymbol{x}_k) \leq \theta_1 f(\boldsymbol{x}_1) + \cdots + \theta_k f(\boldsymbol{x}_k). \tag{5.78}$$

Observe that if X is a random variable with finite support, the above two versions of Jensen's inequality are equivalent.
2. Show that for any discrete distribution p supported on a finite domain of size n, $\mathrm{H}[p] \leq \log_2 n$. This implies that the uniform distribution has maximum entropy.

5.4. Binary cross-entropy loss.

Show that the logistic loss (5.13) is equivalent to the binary cross-entropy loss with $\hat{y} = \sigma(\hat{f})$. That is,

$$\ell_{\log}(\hat{f}; y) = \ell_{\text{bce}}(\hat{y}; y). \tag{5.79}$$

[16] This should not be surprising since — as already mentioned — Gaussian process classification is a generalization of Bayesian logistic regression.

[17] Using the Laplace-approximated latent posterior, $[f^\star \ \boldsymbol{f}]$ are jointly Gaussian. Thus, it directly follows from Theorem 1.24 that the marginal distribution over $f^\star$ is also a Gaussian.

5.5. Gibbs' inequality.

1. Prove $\mathrm{KL}(p\|q) \geq 0$ which is also known as *Gibbs' inequality*.
2. Let p and q be discrete distributions with finite identical support A. Show that $\mathrm{KL}(p\|q) = 0$ if and only if $p \equiv q$.
   Hint: Use that if a function $f : \mathbb{R}^n \to \mathbb{R}$ is strictly convex and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in \mathbb{R}^n$, $\theta_1, \ldots, \theta_k \geq 0$, $\theta_1 + \cdots + \theta_k = 1$, we have that

$$f(\theta_1 \boldsymbol{x}_1 + \cdots + \theta_k \boldsymbol{x}_k) = \theta_1 f(\boldsymbol{x}_1) + \cdots + \theta_k f(\boldsymbol{x}_k) \tag{5.80}$$

   iff $\boldsymbol{x}_1 = \cdots = \boldsymbol{x}_k$. This is a slight adaptation of Jensen's inequality in finite-form, which you proved in Problem 5.3.

5.6. Maximum entropy principle.

In this exercise we will prove that the normal distribution is the distribution with maximal entropy among all (univariate) distributions supported on $\mathbb{R}$ with fixed mean μ and variance σ^2. Let $g(x) \doteq \mathcal{N}(x; \mu, \sigma^2)$, and $f(x)$ be any distribution on $\mathbb{R}$ with mean μ and variance σ^2.

1. Prove that $\mathrm{KL}(f\|g) = \mathrm{H}[g] - \mathrm{H}[f]$.
   Hint: Equivalently, show that $\mathrm{H}[f\|g] = \mathrm{H}[g]$. That is, the expected surprise evaluated based on the Gaussian g is invariant to the true distribution f.
2. Conclude that $\mathrm{H}[g] \geq \mathrm{H}[f]$.

5.7. Probabilistic inference as a consequence of the maximum entropy principle.

Consider the family of generative models of the random vectors $\mathbf{X}$ in $\mathcal{X}$ and $\mathbf{Y}$ in $\mathcal{Y}$:

$$\Delta^{\mathcal{X}\times\mathcal{Y}} = \left\{ q : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_{\geq 0} \;\middle|\; \int_{\mathcal{X}\times\mathcal{Y}} q(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{x}\, d\boldsymbol{y} = 1 \right\}.$$

Suppose that we observe $\mathbf{Y}$ to be $\boldsymbol{y}'$, and are looking for a (new) generative model that is consistent with this information, that is,

$$q(\boldsymbol{y}) = \int_{\mathcal{X}} q(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{x} = \delta_{\boldsymbol{y}'}(\boldsymbol{y}) \qquad \text{using the sum rule (1.7)}$$

where $\delta_{\boldsymbol{y}'}$ denotes the point density at $\boldsymbol{y}'$. The product rule (1.11) implies that $q(\boldsymbol{x}, \boldsymbol{y}) = \delta_{\boldsymbol{y}'}(\boldsymbol{y}) \cdot q(\boldsymbol{x} \mid \boldsymbol{y})$, but any choice of $q(\boldsymbol{x} \mid \boldsymbol{y})$ is possible.

We will derive that given any fixed generative model $p_{\mathbf{X},\mathbf{Y}}$, the "posterior" distribution $q_{\mathbf{X}}(\cdot) = p_{\mathbf{X}|\mathbf{Y}}(\cdot \mid \boldsymbol{y}')$ minimizes the relative entropy $\mathrm{KL}(q_{\mathbf{X},\mathbf{Y}}\|p_{\mathbf{X},\mathbf{Y}})$ subject to the constraint $\mathbf{Y} = \boldsymbol{y}'$. In other words, among all distributions $q_{\mathbf{X},\mathbf{Y}}$ that are consistent with the observation $\mathbf{Y} = \boldsymbol{y}'$, the posterior distribution $q_{\mathbf{X}}(\cdot) = p_{\mathbf{X}|\mathbf{Y}}(\cdot \mid \boldsymbol{y}')$ is the one with "maximum entropy".

1. Show that the optimization problem

$$\begin{aligned}&\arg\min_{q \in \Delta^{\mathcal{X}\times\mathcal{Y}}} \mathrm{KL}(q_{\mathbf{X},\mathbf{Y}} \| p_{\mathbf{X},\mathbf{Y}}) \\ &\text{subject to } q(\boldsymbol{y}) = \delta_{\boldsymbol{y}'}(\boldsymbol{y}) \quad \forall \boldsymbol{y} \in \mathcal{Y}\end{aligned}$$

   is solved by $q(\boldsymbol{x}, \boldsymbol{y}) = \delta_{\boldsymbol{y}'}(\boldsymbol{y}) \cdot p(\boldsymbol{x} \mid \boldsymbol{y})$.
   Hint: Solve the dual problem.
2. Conclude that $q(\boldsymbol{x}) = p(\boldsymbol{x} \mid \boldsymbol{y}')$.

5.8. KL-divergence of Gaussians.

Derive Equation (5.41).

Hint: If $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ in d dimensions, then we have that for any $\boldsymbol{m} \in \mathbb{R}^d$ and $\boldsymbol{A} \in \mathbb{R}^{d \times d}$,

$$\mathbb{E}\Big[(\mathbf{X} - \boldsymbol{m})^\top \boldsymbol{A} (\mathbf{X} - \boldsymbol{m})\Big] = (\boldsymbol{\mu} - \boldsymbol{m})^\top \boldsymbol{A} (\boldsymbol{\mu} - \boldsymbol{m}) + \mathrm{tr}(\boldsymbol{A}\boldsymbol{\Sigma}) \tag{5.81}$$

5.9. Forward vs reverse KL.

1. Consider a factored approximation $q(x, y) = q(x)q(y)$ to a joint distribution $p(x, y)$. Show that to minimize the forward $\mathrm{KL}(p\|q)$ we should set $q(x) = p(x)$ and $q(y) = p(y)$, i.e., the optimal approximation is a product of marginals.
2. Consider the following joint distribution p, where the rows represent y and the columns x:

	1	2	3	4
1	1/8	1/8	0	0
2	1/8	1/8	0	0
3	0	0	1/4	0
4	0	0	0	1/4

   Show that the reverse $\mathrm{KL}(q\|p)$ for this p has three distinct minima. Identify those minima and evaluate $\mathrm{KL}(q\|p)$ at each of them.
3. What is the value of $\mathrm{KL}(q\|p)$ if we use the approximation $q(x, y) = p(x)p(y)$?

5.10. Gaussian VI vs Laplace approximation.

In this exercise, we compare the Laplace approximation from Section 5.1 to variational inference with the variational family of Gaussians,

$$\mathcal{Q} \doteq \{q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma})\}.$$

1. Let p be any distribution on $\mathbb{R}$, and let $q^\star = \arg\min_{q \in \mathcal{Q}} \mathrm{KL}(p\|q)$. Show that $q^\star$ differs from the Laplace approximation of p.

Minimizing forward-KL is typically intractable, and we have seen that it is therefore common to minimize the reverse-KL instead:

$$\tilde{q} = \arg\min_{q \in \mathcal{Q}} \mathrm{KL}(q \| p(\cdot \mid \mathcal{D}_n)).$$

2. Show that $\tilde{q} = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ satisfies Equation (5.58):

$$\begin{aligned} \mathbf{0} &= \mathbb{E}_{\boldsymbol{\theta} \sim \tilde{q}}[\boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p(y_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n})] \\ \boldsymbol{\Sigma}^{-1} &= -\mathbb{E}_{\boldsymbol{\theta} \sim \tilde{q}}[\boldsymbol{H}_{\boldsymbol{\theta}} \log p(y_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n})]. \end{aligned}$$

Hint 1: For any positive definite and symmetric matrix $\boldsymbol{A}$, it holds that $\boldsymbol{\nabla}_{\boldsymbol{A}} \log\det(\boldsymbol{A}) = \boldsymbol{A}^{-1}$.
Hint 2: For any function f and Gaussian $p = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$,

$$\begin{aligned} \boldsymbol{\nabla}_{\boldsymbol{\mu}} \mathbb{E}_{\boldsymbol{x} \sim p}[f(\boldsymbol{x})] &= \mathbb{E}_{\boldsymbol{x} \sim p}[\boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x})] \\ \boldsymbol{\nabla}_{\boldsymbol{\Sigma}} \mathbb{E}_{\boldsymbol{x} \sim p}[f(\boldsymbol{x})] &= \frac{1}{2} \mathbb{E}_{\boldsymbol{x} \sim p}[\boldsymbol{H}_{\boldsymbol{x}} f(\boldsymbol{x})]. \end{aligned} \tag{5.82}$$

Recall the conditions satisfied by the Laplace approximation of the posterior $p(\boldsymbol{\theta} \mid \mathcal{D}) \propto p(y_{1:n}, \boldsymbol{\theta} \mid \boldsymbol{x}_{1:n})$ as detailed in Equation (5.57). The Laplace approximation is fitted *locally* at the MAP estimate $\hat{\boldsymbol{\theta}}$. Comparing Equation (5.57) to Equation (5.58), we see that Gaussian variational inference satisfies the conditions of the Laplace approximation *on average*. For more details, refer to Opper and Archambeau (2009).

5.11. Gradient of reverse-KL.

Suppose $p \doteq \mathcal{N}(\mathbf{0}, \sigma_{\mathrm{p}}^2 \boldsymbol{I})$ and a tractable distribution described by

$$q_{\lambda} \doteq \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}\{\sigma_1^2, \ldots, \sigma_d^2\})$$

where $\boldsymbol{\mu} \doteq [\mu_1 \ \cdots \ \mu_d]$ and $\boldsymbol{\lambda} \doteq [\mu_1 \ \cdots \ \mu_d \ \sigma_1 \ \cdots \ \sigma_d]$. Show that the gradient of $\mathrm{KL}(q_{\lambda} \| p(\cdot))$ with respect to $\boldsymbol{\lambda}$ is given by

$$\boldsymbol{\nabla}_{\boldsymbol{\mu}} \mathrm{KL}(q_{\lambda} \| p(\cdot)) = \sigma_{\mathrm{p}}^{-2} \boldsymbol{\mu}, \quad \text{and} \tag{5.83a}$$

$$\boldsymbol{\nabla}_{[\sigma_1 \ \cdots \ \sigma_d]} \mathrm{KL}(q_{\lambda} \| p(\cdot)) = \begin{bmatrix} \frac{\sigma_1}{\sigma_{\mathrm{p}}^2} - \frac{1}{\sigma_1} & \cdots & \frac{\sigma_d}{\sigma_{\mathrm{p}}^2} - \frac{1}{\sigma_d} \end{bmatrix}. \tag{5.83b}$$

5.12. Reparameterizable distributions.

1. Let $X \sim \mathrm{Unif}([a, b])$ for any $a \leq b$. That is,

$$p_X(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in [a, b] \\ 0 & \text{otherwise.} \end{cases} \tag{5.84}$$

Show that X can be reparameterized in terms of $\mathrm{Unif}([0, 1])$. *Hint: You may use that for any $Y \sim \mathrm{Unif}([a, b])$ and $c \in \mathbb{R}$,*

- $Y + c \sim \mathrm{Unif}([a + c, b + c])$ *and*
- $cY \sim \mathrm{Unif}([c \cdot a, c \cdot b])$.

2. Let $Z \sim \mathcal{N}(\mu, \sigma^2)$ and $X \doteq e^Z$. That is, X is logarithmically normal distributed with parameters μ and σ^2. Show that X can be reparameterized in terms of $\mathcal{N}(0,1)$.
3. Show that $\text{Cauchy}(0,1)$ can be reparameterized in terms of $\text{Unif}([0,1])$.

Finally, let us apply the reparameterization trick to compute the gradient of an expectation.

4. Let $\text{ReLU}(z) \doteq \max\{0, z\}$ and $w > 0$. Show that

$$\frac{d}{d\mu}\mathbb{E}_{x\sim\mathcal{N}(\mu,1)}\text{ReLU}(wx) = w\Phi(\mu)$$

where Φ denotes the CDF of the standard normal distribution.

6

Markov Chain Monte Carlo Methods

Variational inference approximates the entire posterior distribution. However, note that the key challenge in probabilistic inference is not learning the posterior distribution, but using the posterior distribution for predictions,

$$p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}) = \int p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n})\, d\boldsymbol{\theta}. \tag{6.1}$$

This integral can be interpreted as an expectation over the posterior distribution,

$$= \mathbb{E}_{\boldsymbol{\theta} \sim p(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n})}[p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta})]. \tag{6.2}$$

Observe that the likelihood $f(\boldsymbol{\theta}) \doteq p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta})$ is easy to evaluate. The difficulty lies in sampling from the posterior distribution. Assuming we can obtain independent samples from the posterior distribution, we can use Monte Carlo sampling to obtain an unbiased estimate of the expectation,

$$\approx \frac{1}{m} \sum_{i=1}^{m} f(\boldsymbol{\theta}^{(i)}) \tag{6.3}$$

for independent $\boldsymbol{\theta}^{(i)} \overset{\text{iid}}{\sim} p(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n})$. The law of large numbers (A.36) and Hoeffding's inequality (A.41) imply that this estimator is consistent and sharply concentrated.[1]

[1] For more details, see Appendix A.3.3.

Obtaining samples of the posterior distribution is therefore sufficient to perform approximate inference. Recall that the difficulty of computing the posterior p exactly, was in finding the normalizing constant Z,

$$p(x) = \frac{1}{Z} q(x). \tag{6.4}$$

The joint likelihood q is typically easy to obtain. Note that $q(x)$ is proportional to the probability density associated with x, but q does not

integrate to 1. Such functions are also called a *finite measure*. Without normalizing q, we cannot directly sample from it.

Remark 6.1: The difficulty of sampling — even with a PDF

Even a decent approximation of Z does not yield a general efficient sampling method. For example, one very common approach to sampling is *inverse transform sampling* (cf. Appendix A.1.3) which requires an (approximate) quantile function. Computing the quantile function given an arbitrary PDF requires solving integrals over the domain of the PDF which is what we were trying to avoid in the first place.

The key idea of Markov chain Monte Carlo methods is to construct a Markov chain, which is efficient to simulate and has the stationary distribution p.

6.1 Markov Chains

To start, let us revisit the fundamental theory behind Markov chains.

Definition 6.2 (Markov chain)**.** A *(finite and discrete-time) Markov chain* over the state space

$$S \doteq \{0, \dots, n-1\} \tag{6.5}$$

is a stochastic process[2] $(X_t)_{t \in \mathbb{N}_0}$ valued in S such that the *Markov property* is satisfied:

$$X_{t+1} \perp X_{0:t-1} \mid X_t. \tag{6.6}$$

[2] A *stochastic process* is a sequence of random variables.

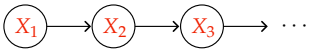


Figure 6.1: Directed graphical model of a Markov chain. The random variable X_{t+1} is conditionally independent of the random variables $X_{0:t-1}$ given X_t.

Intuitively, the Markov property states that future behavior is independent of past states given the present state.

Remark 6.3: Generalizations of Markov chains

One can also define continuous-state Markov chains (for example, where states are vectors in $\mathbb{R}^d$) and the results which we state for (finite) Markov chains will generally carry over. For a survey, refer to "General state space Markov chains and MCMC algorithms" (Roberts and Rosenthal, 2004).

Moreover, one can also consider continuous-time Markov chains. One example of such a continuous-space and continuous-time Markov chain is the *Wiener process* (cf. Remark 6.23).

We restrict our attention to *time-homogeneous* Markov chains,[3] which

[3] That is, the transition probabilities do not change over time.

can be characterized by a *transition function*,

$$p(x' \mid x) \doteq \mathbb{P}\big(X_{t+1} = x' \mid X_t = x\big). \tag{6.7}$$

As the state space is finite, we can describe the transition function by the *transition matrix*,

$$\boldsymbol{P} \doteq \begin{bmatrix} p(x_1 \mid x_1) & \cdots & p(x_n \mid x_1) \\ \vdots & \ddots & \vdots \\ p(x_1 \mid x_n) & \cdots & p(x_n \mid x_n) \end{bmatrix} \in \mathbb{R}^{n \times n}. \tag{6.8}$$

Note that each row of $\boldsymbol{P}$ must always sum to 1. Such matrices are also called *stochastic*.

The *transition graph* of a Markov chain is a directed graph consisting of vertices S and weighted edges represented by the adjacency matrix $\boldsymbol{P}$.

The current state of the Markov chain at time t is denoted by the probability distribution q_t over states S, that is, $X_t \sim q_t$. In the finite setting, q_t is a PMF, which is often written explicitly as the row vector $\boldsymbol{q}_t \in \mathbb{R}^{1 \times |S|}$. The initial state (or prior) of the Markov chain is given as $X_0 \sim q_0$. One iteration of the Markov chain can then be expressed as follows: (?) **Problem 6.1**

$$\boldsymbol{q}_{t+1} = \boldsymbol{q}_t \boldsymbol{P}. \tag{6.9}$$

It is implied directly that we can write the state of the Markov chain at time $t + k$ as

$$\boldsymbol{q}_{t+k} = \boldsymbol{q}_t \boldsymbol{P}^k. \tag{6.10}$$

The entry $\boldsymbol{P}^k(x, x')$ corresponds to the probability of transitioning from state $x \in S$ to state $x' \in S$ in exactly k steps (?). We denote this entry by $p^{(k)}(x' \mid x)$. **Problem 6.2**

In the analysis of Markov chains, there are two main concepts of interest: stationarity and convergence. We begin by introducing stationarity.

6.1.1 Stationarity

Definition 6.4 (Stationary distribution)**.** A distribution π is *stationary* with respect to the transition function p iff

$$\pi(x) = \sum_{x' \in S} p(x \mid x')\pi(x') \tag{6.11}$$

holds for all $x \in S$. It follows from Equation (6.9) that equivalently, π is stationary w.r.t. a transition matrix $\boldsymbol{P}$ iff

$$\boldsymbol{\pi} = \boldsymbol{\pi} \boldsymbol{P}. \tag{6.12}$$

After entering a stationary distribution π, a Markov chain will always remain in the stationary distribution. In particular, suppose that X_t is distributed according to π, then for all $k \geq 0$, $X_{t+k} \sim \pi$.

Remark 6.5: When does a stationary distribution exist?

In general, there are Markov chains with infinitely many stationary distributions or no stationary distribution at all. You can find some examples in Figure 6.2.

It can be shown that there exists a unique stationary distribution π if the Markov chain is *irreducible*, that is, if every state is reachable from every other state with a positive probability when the Markov chain is run for enough steps. Formally,

$$\forall x, x' \in S. \exists k \in \mathbb{N}. p^{(k)}(x' \mid x) > 0. \tag{6.13}$$

Equivalently, a Markov chain is irreducible iff its transition graph is strongly connected.

6.1.2 Convergence

Let us now consider Markov chains with a unique stationary distribution.[4] A natural next question is whether this Markov chain converges to its stationary distribution. We say that a Markov chain converges to its stationary distribution iff we have

$$\lim_{t \to \infty} q_t = \pi, \tag{6.14}$$

irrespectively of the initial distribution q_0.

[4] Observe that the stationary distribution of an irreducible Markov chain must have full support, that is, assign positive probability to every state.

Remark 6.6: When does a Markov chain converge?

Even if a Markov chain has a unique stationary distribution, it does not have to converge to it. Consider example (3) in Figure 6.2. Clearly, $\pi = (\frac{1}{2}, \frac{1}{2})$ is the unique stationary distribution. However, observe that if we start with a suitable initial distribution such as $q_0 = (1, 0)$, at no point in time will the probability of all states be positive, and in particular, the chain will not converge to π. Instead, the chain behaves periodically, i.e., its state distributions are $q_{2t} = (0, 1)$ and $q_{2t+1} = (1, 0)$ for all $t \in \mathbb{N}$. It turns out that if we exclude such "periodic" Markov chains, then the remaining (irreducible) Markov chains will always converge to their stationary distribution.

Formally, a Markov chain is *aperiodic* if for all states $x \in S$,

$$\exists k_0 \in \mathbb{N}. \forall k \geq k_0. p^{(k)}(x \mid x) > 0. \tag{6.15}$$

In words, a Markov chain is aperiodic iff for every state x, the transition graph has a closed path from x to x with length k for all $k \in \mathbb{N}$ greater than some $k_0 \in \mathbb{N}$.

This additional property leads to the concept of ergodicity.

Definition 6.7 (Ergodicity)**.** A Markov chain is *ergodic* iff there exists a $t \in \mathbb{N}_0$ such that for any $x, x' \in S$ we have

$$p^{(t)}(x' \mid x) > 0, \tag{6.16}$$

whereby $p^{(t)}(x' \mid x)$ is the probability to reach x' from x in exactly t steps. Equivalent conditions are

1. that there exists some $t \in \mathbb{N}_0$ such that all entries of $\boldsymbol{P}^t$ are strictly positive; and
2. that it is irreducible and aperiodic.

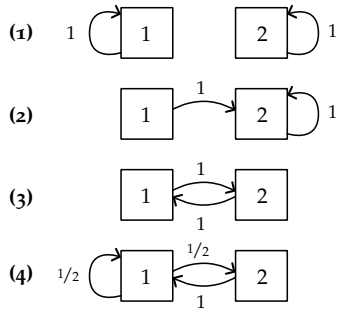


Figure 6.2: Transition graphs of Markov chains: (1) is not ergodic as its transition diagram is not strongly connected; (2) is not ergodic for the same reason; (3) is irreducible but periodic and therefore not ergodic; (4) is ergodic with stationary distribution $\pi(1) = 2/3, \pi(2) = 1/3$.

Example 6.8: Making a Markov chain ergodic

A commonly used strategy to ensure that a Markov chain is ergodic is to add "self-loops" to every vertex in the transition graph. That is, to ensure that at any point in time, the Markov chain remains with positive probability in its current state.

Take a (not necessarily ergodic) but irreducible Markov chain with transition matrix $\boldsymbol{P}$. We define the new Markov chain

$$\boldsymbol{P}' \doteq \frac{1}{2}\boldsymbol{P} + \frac{1}{2}\boldsymbol{I}. \tag{6.17}$$

It is a simple exercise to confirm that $\boldsymbol{P}'$ is stochastic, and hence a valid transition matrix. Also, it follows directly that $\boldsymbol{P}'$ is irreducible (as $\boldsymbol{P}$ is irreducible) and aperiodic as every vertex has a closed path of length 1 to itself, and therefore the chain is ergodic.

Take now $\boldsymbol{\pi}$ to be a stationary distribution of $\boldsymbol{P}$. We have that $\boldsymbol{\pi}$ is also a stationary distribution of $\boldsymbol{P}'$ as

$$\boldsymbol{\pi}\boldsymbol{P}' = \frac{1}{2}\boldsymbol{\pi}\boldsymbol{P} + \frac{1}{2}\boldsymbol{\pi}\boldsymbol{I} = \frac{1}{2}\boldsymbol{\pi} + \frac{1}{2}\boldsymbol{\pi} = \boldsymbol{\pi}. \tag{6.18}$$

using (6.12)

Fact 6.9 (Fundamental theorem of ergodic Markov chains, theorem 4.9 of Levin and Peres (2017))**.** *An ergodic Markov chain has a unique stationary distribution π (with full support) and*

$$\lim_{t \to \infty} q_t = \pi \tag{6.19}$$

irrespectively of the initial distribution q_0.

This naturally suggests constructing an ergodic Markov chain such that its stationary distribution coincides with the posterior distribution. If we then sample "sufficiently long", X_t is drawn from a distribution that is "very close" to the posterior distribution.

Remark 6.10: How quickly does a Markov chain converge?

The convergence speed of Markov chains is a rich field of research. "Sufficiently long" and "very close" are commonly made precise by the notions of *rapidly mixing* Markov chains and *total variation distance*.

Definition 6.11 (Total variation distance)**.** The total variation distance between two probability distributions μ and ν on $\mathcal{A}$ is defined by

$$\|\mu - \nu\|_{\mathrm{TV}} \doteq 2 \sup_{A \subseteq \mathcal{A}} |\mu(A) - \nu(A)| . \tag{6.20}$$

It defines the distance between μ and ν to be the maximum difference between the probabilities that μ and ν assign to the same event.

As opposed to the KL-divergence (5.34), the total variation distance is a metric. In particular, it is symmetric and satisfies the triangle inequality. It can be shown that

$$\|\mu - \nu\|_{\mathrm{TV}} \leq \sqrt{2\mathrm{KL}(\mu \| \nu)} \tag{6.21}$$

which is known as *Pinsker's inequality*. Moreover, if μ and ν are discrete distributions over the set S, it can be shown that

$$\|\mu - \nu\|_{\mathrm{TV}} = \sum_{i \in S} |\mu(i) - \nu(i)| . \tag{6.22}$$

Definition 6.12 (Mixing time)**.** For a Markov chain with stationary distribution π, its *mixing time* with respect to the total variation distance for any $\epsilon > 0$ is

$$\tau_{\mathrm{TV}}(\epsilon) \doteq \min\{t \mid \forall q_0 : \|q_t - \pi\|_{\mathrm{TV}} \leq \epsilon\}. \tag{6.23}$$

Thus, the mixing time measures the time required by a Markov chain for the distance to stationarity to be small. A Markov chain is typically said to be *rapidly mixing* if for any $\epsilon > 0$,

$$\tau_{\mathrm{TV}}(\epsilon) \in O(\mathrm{poly}(n, \log(1/\epsilon))). \tag{6.24}$$

That is, a rapidly mixing Markov chain on n states needs to be simulated for at most $\text{poly}(n)$ steps to obtain a "good" sample from its stationary distribution π.

You can find a thorough introduction to mixing times in chapter 4 of "Markov chains and mixing times" (Levin and Peres, 2017). Later chapters introduce methods for showing that a Markov chain is rapidly mixing.

6.1.3 *Detailed Balance Equation*

How can we confirm that the stationary distribution of a Markov chain coincides with the posterior distribution? The detailed balance equation yields a very simple method.

Definition 6.13 (Detailed balance equation / reversibility)**.** A Markov chain satisfies the *detailed balance equation* with respect to a distribution π iff

$$\pi(x)p(x' \mid x) = \pi(x')p(x \mid x') \tag{6.25}$$

holds for any $x, x' \in S$. A Markov chain that satisfies the detailed balance equation with respect to π is called *reversible* with respect to π.

Lemma 6.14. *Given a finite Markov chain, if the Markov chain is reversible with respect to π then π is a stationary distribution.*[5]

Proof. Let $\pi \doteq q_t$. We have,

$$\begin{aligned}
q_{t+1}(x) &= \sum_{x' \in S} p(x \mid x')q_t(x') && \text{using the Markov property (6.6)} \\
&= \sum_{x' \in S} p(x \mid x')\pi(x') \\
&= \sum_{x' \in S} p(x' \mid x)\pi(x) && \text{using the detailed balance equation (6.25)} \\
&= \pi(x) \sum_{x' \in S} p(x' \mid x) \\
&= \pi(x). && \text{using that } \textstyle\sum_{x' \in S} p(x' \mid x) = 1
\end{aligned}$$

□

[5] Note that reversibility of π is only a sufficient condition for stationarity of π, it is not necessary! In particular, there are irreversible ergodic Markov chains.

That is, if we can show that the detailed balance equation (6.25) holds for some distribution q, then we know that q is the stationary distribution of the Markov chain.

Next, reconsider our posterior distribution $p(x) = \frac{1}{Z}q(x)$ from Equation (6.4). If we substitute the posterior for π in the detailed balance equation, we obtain

$$\frac{1}{Z}q(x)p(x' \mid x) = \frac{1}{Z}q(x')p(x \mid x'), \tag{6.26}$$

or equivalently,

$$q(x)p(x' \mid x) = q(x')p(x \mid x'). \tag{6.27}$$

In words, we do not need to know the true posterior p to check that the stationary distribution of our Markov chain coincides with p, it suffices to know the finite measure q!

6.1.4 *Ergodic Theorem*

If we now suppose that we can construct a Markov chain whose stationary distribution coincides with the posterior distribution — we will see later that this is possible — it is not apparent that this allows us to estimate expectations over the posterior distribution. Note that although constructing such a Markov chain allows us to obtain samples from the posterior distribution, they are *not* independent. In fact, due to the structure of a Markov chain, by design, they are strongly dependent. Thus, the law of large numbers and Hoeffding's inequality do not apply. By itself, it is not even clear that an estimator relying on samples from a single Markov chain will be unbiased.

Theoretically, we could simulate many Markov chains separately and obtain one sample from each of them. This, however, is extremely inefficient. It turns out that there is a way to generalize the (strong) law of large numbers to Markov chains.

Theorem 6.15 (Ergodic theorem, appendix C of Levin and Peres (2017))**.** *Given an ergodic Markov chain $(X_t)_{t \in \mathbb{N}_0}$ over a finite state space S with stationary distribution π and a function $f : S \to \mathbb{R}$,*

$$\frac{1}{n} \sum_{i=1}^{n} f(x_i) \overset{a.s.}{\to} \sum_{x \in S} \pi(x) f(x) = \mathbb{E}_{x \sim \pi}[f(x)] \tag{6.28}$$

as $n \to \infty$ where $x_i \sim X_i \mid x_{i-1}$.

This result is the fundamental reason for why Markov chain Monte Carlo methods are possible. There are analogous results for continuous domains.

Note, however, that the ergodic theorem only tells us that simulating a single Markov chain yields an unbiased estimator. It does not tell us anything about the rate of convergence and variance of such an estimator. The convergence rate depends on the mixing time of the Markov chain, which is difficult to establish in general.

In practice, one observes that Markov chain Monte Carlo methods have a so-called *"burn-in" time* during which the distribution of the Markov

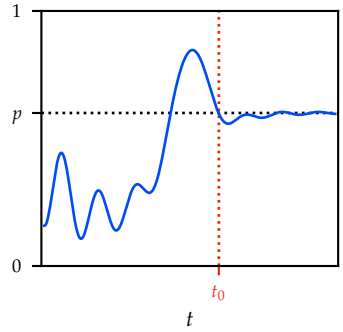


Figure 6.3: Illustration of the "burn-in" time t_0 of a Markov chain approximating the posterior $p(y^\star = 1 \mid \boldsymbol{X}, \boldsymbol{y})$ of Bayesian logistic regression. The true posterior p is shown in gray. The distribution of the Markov chain at time t is shown in red.

chain does not yet approximate the posterior distribution well. Typically, the first t_0 samples are therefore discarded,

$$\mathbb{E}[f(X)] \approx \frac{1}{T - t_0} \sum_{t=t_0+1}^{T} f(X_t). \tag{6.29}$$

It is not clear in general how T and t_0 should be chosen such that the estimator is unbiased, rather they have to be tuned.

Another widely used heuristic is to first find the mode of the posterior distribution and then start the Markov chain at that point. This tends to increase the rate of convergence drastically, as the Markov chain does not have to "walk to the location in the state space where most probability mass will be located".

6.2 *Elementary Sampling Methods*

We will now examine methods for constructing and sampling from a Markov chain with the goal of approximating samples from the posterior distribution p. Note that in this setting the state space of the Markov chain is $\mathbb{R}^n$ and a single state at time t is described by the random vector $\mathbf{X} \doteq [X_1, \ldots, X_n]$.

6.2.1 *Metropolis-Hastings Algorithm*

Suppose we are given a *proposal distribution* $r(\boldsymbol{x}' \mid \boldsymbol{x})$ which, given we are in state $\boldsymbol{x}$, proposes a new state $\boldsymbol{x}'$. Metropolis and Hastings showed that using the *acceptance distribution* $\text{Bern}(\alpha(\boldsymbol{x}' \mid \boldsymbol{x}))$ where

$$\alpha(\boldsymbol{x}' \mid \boldsymbol{x}) \doteq \min\left\{1, \frac{p(\boldsymbol{x}')r(\boldsymbol{x} \mid \boldsymbol{x}')}{p(\boldsymbol{x})r(\boldsymbol{x}' \mid \boldsymbol{x})}\right\} \tag{6.30}$$

$$= \min\left\{1, \frac{q(\boldsymbol{x}')r(\boldsymbol{x} \mid \boldsymbol{x}')}{q(\boldsymbol{x})r(\boldsymbol{x}' \mid \boldsymbol{x})}\right\} \tag{6.31}$$

similarly to the detailed balance equation, the normalizing constant Z cancels

to decide whether to follow the proposal yields a Markov chain with stationary distribution $p(\boldsymbol{x}) = \frac{1}{Z}q(\boldsymbol{x})$.

Algorithm 6.16: Metropolis-Hastings algorithm

```
initialize x ∈ ℝ^n
for t = 1 to T do
    sample x' ~ r(x' | x)
    sample u ~ Unif([0, 1])
    if u ≤ α(x' | x) then update x ← x'
    else update x ← x
```

Intuitively, the acceptance distribution corrects for the bias in the proposal distribution. That is, if the proposal distribution r is likely to propose states with low probability under p, the acceptance distribution will reject these proposals frequently. The following theorem formalizes this intuition.

Theorem 6.17 (Metropolis-Hastings theorem)**.** *Given an arbitrary proposal distribution r whose support includes the support of q, the stationary distribution of the Markov chain simulated by the Metropolis-Hastings algorithm is $p(\boldsymbol{x}) = \frac{1}{Z} q(\boldsymbol{x})$.*

Proof. First, let us define the transition probabilities of the Markov chain. The probability of transitioning from a state $\boldsymbol{x}$ to a state $\boldsymbol{x}'$ is given by $r(\boldsymbol{x}' \mid \boldsymbol{x})\alpha(\boldsymbol{x}' \mid \boldsymbol{x})$ if $\boldsymbol{x} \neq \boldsymbol{x}'$ and the probability of proposing to remain in state $\boldsymbol{x}$, $r(\boldsymbol{x} \mid \boldsymbol{x})$, plus the probability of denying the proposal, otherwise.

$$p(\boldsymbol{x}' \mid \boldsymbol{x}) = \begin{cases} r(\boldsymbol{x}' \mid \boldsymbol{x})\alpha(\boldsymbol{x}' \mid \boldsymbol{x}) & \text{if } \boldsymbol{x} \neq \boldsymbol{x}' \\ r(\boldsymbol{x} \mid \boldsymbol{x}) + \sum_{\boldsymbol{x}'' \neq \boldsymbol{x}} r(\boldsymbol{x}'' \mid \boldsymbol{x})(1 - \alpha(\boldsymbol{x}'' \mid \boldsymbol{x})) & \text{otherwise.} \end{cases} \tag{6.32}$$

We will show that the stationary distribution is p by showing that p satisfies the detailed balance equation (6.25). Let us fix arbitrary states $\boldsymbol{x}$ and $\boldsymbol{x}'$. First, observe that if $\boldsymbol{x} = \boldsymbol{x}'$, then the detailed balance equation is trivially satisfied. Without loss of generality we assume

$$\alpha(\boldsymbol{x} \mid \boldsymbol{x}') = 1, \quad \alpha(\boldsymbol{x}' \mid \boldsymbol{x}) = \frac{q(\boldsymbol{x}')r(\boldsymbol{x} \mid \boldsymbol{x}')}{q(\boldsymbol{x})r(\boldsymbol{x}' \mid \boldsymbol{x})}.$$

For $\boldsymbol{x} \neq \boldsymbol{x}'$, we then have,

$$\begin{aligned}
p(\boldsymbol{x}) \cdot p(\boldsymbol{x}' \mid \boldsymbol{x}) &= \frac{1}{Z} q(\boldsymbol{x}) p(\boldsymbol{x}' \mid \boldsymbol{x}) && \text{using the definition of the distribution } p \\
&= \frac{1}{Z} q(\boldsymbol{x}) r(\boldsymbol{x}' \mid \boldsymbol{x}) \alpha(\boldsymbol{x}' \mid \boldsymbol{x}) && \text{using the transition probabilities of the Markov chain} \\
&= \frac{1}{Z} q(\boldsymbol{x}) r(\boldsymbol{x}' \mid \boldsymbol{x}) \frac{q(\boldsymbol{x}')r(\boldsymbol{x} \mid \boldsymbol{x}')}{q(\boldsymbol{x})r(\boldsymbol{x}' \mid \boldsymbol{x})} && \text{using the definition of the acceptance distribution } \alpha \\
&= \frac{1}{Z} q(\boldsymbol{x}') r(\boldsymbol{x} \mid \boldsymbol{x}') \\
&= \frac{1}{Z} q(\boldsymbol{x}') r(\boldsymbol{x} \mid \boldsymbol{x}') \alpha(\boldsymbol{x} \mid \boldsymbol{x}') && \text{using the definition of the acceptance distribution } \alpha \\
&= \frac{1}{Z} q(\boldsymbol{x}') p(\boldsymbol{x} \mid \boldsymbol{x}') && \text{using the transition probabilities of the Markov chain} \\
&= p(\boldsymbol{x}') \cdot p(\boldsymbol{x} \mid \boldsymbol{x}'). && \text{using the definition of the distribution } p
\end{aligned}$$

□

Note that by the fundamental theorem of ergodic Markov chains (6.19), for convergence to the stationary distribution, it is sufficient for the

Markov chain to be ergodic. Ergodicity follows immediately when the transition probabilities $p(\cdot \mid \boldsymbol{x})$ have full support. For example, if the proposal distribution $r(\cdot \mid \boldsymbol{x})$ has full support, the full support of $p(\cdot \mid \boldsymbol{x})$ follows immediately from Equation (6.32). The rate of convergence of Metropolis-Hastings depends strongly on the choice of the proposal distribution, and we will explore different choices of proposal distribution in the following.

6.2.2 *Gibbs Sampling*

A popular example of a Metropolis-Hastings algorithm is Gibbs sampling as presented in Algorithm 6.18.

Algorithm 6.18: Gibbs sampling

initialize $\boldsymbol{x} = [x_1, \ldots, x_n] \in \mathbb{R}^n$
for $t = 1$ **to** T **do**
  pick a variable i uniformly at random from $\{1, \ldots, n\}$
  set $\boldsymbol{x}_{-i} \doteq [x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n]$
  update x_i by sampling according to the posterior distribution $p(x_i \mid \boldsymbol{x}_{-i})$

Intuitively, by re-sampling single coordinates according to the posterior distribution given the other coordinates, Gibbs sampling finds states that are successively "more" likely. Selecting the index i uniformly at random ensures that the underlying Markov chain is ergodic provided the conditional distributions $p(\cdot \mid \boldsymbol{x}_{-i})$ have full support.

Theorem 6.19 (Gibbs sampling as Metropolis-Hastings)**.** *Gibbs sampling is a Metropolis-Hastings algorithm. For any fixed $i \in [n]$, it has proposal distribution*

$$r_i(\boldsymbol{x}' \mid \boldsymbol{x}) \doteq \begin{cases} p(x'_i \mid \boldsymbol{x}'_{-i}) & \boldsymbol{x}' \text{ differs from } \boldsymbol{x} \text{ only in entry } i \\ 0 & \text{otherwise} \end{cases} \tag{6.33}$$

and acceptance distribution $\alpha_i(\boldsymbol{x}' \mid \boldsymbol{x}) \doteq 1$.

Proof. We show that $\alpha_i(\boldsymbol{x}' \mid \boldsymbol{x}) = 1$ follows from the definition of an acceptance distribution in Metropolis-Hastings (6.31) and the choice of proposal distribution (6.33).

By (6.31),

$$\alpha_i(\boldsymbol{x}' \mid \boldsymbol{x}) = \min\left\{1, \frac{p(\boldsymbol{x}')r_i(\boldsymbol{x} \mid \boldsymbol{x}')}{p(\boldsymbol{x})r_i(\boldsymbol{x}' \mid \boldsymbol{x})}\right\}$$

Note that $p(\boldsymbol{x}) = p(x_i, \boldsymbol{x}_{-i}) = p(x_i \mid \boldsymbol{x}_{-i})p(\boldsymbol{x}_{-i})$ using the product rule (1.11). Therefore,

$$
\begin{aligned}
&= \min\left\{1, \frac{p(x_i' \mid \boldsymbol{x}'_{-i})p(\boldsymbol{x}'_{-i})r_i(\boldsymbol{x} \mid \boldsymbol{x}')}{p(x_i \mid \boldsymbol{x}_{-i})p(\boldsymbol{x}_{-i})r_i(\boldsymbol{x}' \mid \boldsymbol{x})}\right\} \\
&= \min\left\{1, \frac{p(x_i' \mid \boldsymbol{x}'_{-i})p(\boldsymbol{x}'_{-i})p(x_i \mid \boldsymbol{x}_{-i})}{p(x_i \mid \boldsymbol{x}_{-i})p(\boldsymbol{x}_{-i})p(x_i' \mid \boldsymbol{x}'_{-i})}\right\} && \text{using the proposal distribution (6.33)} \\
&= \min\left\{1, \frac{p(\boldsymbol{x}'_{-i})}{p(\boldsymbol{x}_{-i})}\right\} \\
&= 1. && \square \quad \text{using that } p(\boldsymbol{x}'_{-i}) = p(\boldsymbol{x}_{-i})
\end{aligned}
$$

If the index i is chosen uniformly at random as in Algorithm 6.18, then the proposal distribution is $r(\boldsymbol{x}' \mid \boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^n r_i(\boldsymbol{x}' \mid \boldsymbol{x})$, which analogously to Theorem 6.19 has the associated acceptance distribution

$$
\alpha(\boldsymbol{x}' \mid \boldsymbol{x}) = \min\left\{1, \frac{p(\boldsymbol{x}')r(\boldsymbol{x} \mid \boldsymbol{x}')}{p(\boldsymbol{x})r(\boldsymbol{x}' \mid \boldsymbol{x})}\right\} = \min\left\{1, \frac{\sum_{i=1}^n p(\boldsymbol{x}')r_i(\boldsymbol{x} \mid \boldsymbol{x}')}{\sum_{i=1}^n p(\boldsymbol{x})r_i(\boldsymbol{x}' \mid \boldsymbol{x})}\right\} = 1.
$$

Corollary 6.20 (Convergence of Gibbs sampling)**.** *As Gibbs sampling is a specific example of an MH-algorithm, the stationary distribution of the simulated Markov chain is $p(\boldsymbol{x})$.*

Note that for the proposals of Gibbs sampling, we have

$$
p(x_i \mid \boldsymbol{x}_{-i}) = \frac{p(x_i, \boldsymbol{x}_{-i})}{\sum_{x_i} p(x_i, \boldsymbol{x}_{-i})} = \frac{q(x_i, \boldsymbol{x}_{-i})}{\sum_{x_i} q(x_i, \boldsymbol{x}_{-i})}. \tag{6.34}
$$

using the definition of condition probability (1.8) and the sum rule (1.7)

Under many models, this probability can be efficiently evaluated due to the conditioning on the remaining coordinates $\boldsymbol{x}_{-i}$. If X_i has finite support, the normalizer can be computed exactly.

6.3 Sampling using Gradients

In this section, we discuss more advanced sampling methods. The main idea that we will study is the interpretation of sampling as an optimization problem. We will build towards an optimization view of sampling step-by-step, and first introduce what is commonly called the "energy" of a distribution.

6.3.1 Gibbs Distributions

Gibbs distributions are a special class of distributions that are widely used in machine learning, and which are characterized by an energy.

Definition 6.21 (Gibbs distribution)**.** Formally, a *Gibbs distribution* (also called a *Boltzmann distribution*) is a continuous distribution p whose PDF is of the form

$$
p(\boldsymbol{x}) = \frac{1}{Z}\exp(-f(\boldsymbol{x})). \tag{6.35}
$$

$f : \mathbb{R}^n \to \mathbb{R}$ is also called an *energy function*. When the energy function f is convex, its Gibbs distribution is called *log-concave*.

A useful property is that Gibbs distributions always have full support.[6] It is often easier to reason about "energies" rather than probabilities as they are neither restricted to be non-negative nor do they have to integrate to 1. Note that the Gibbs distribution belongs to the exponential family (5.48) with sufficient statistic $-f(\boldsymbol{x})$.

[6] This can easily be seen as $\exp(\cdot) > 0$.

Observe that the posterior distribution can always be interpreted as a Gibbs distribution as long as prior and likelihood have full support,

$$\begin{aligned} p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}) &= \frac{1}{Z} p(\boldsymbol{\theta}) p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}) && \text{using Bayes' rule (1.45)} \\ &= \frac{1}{Z} \exp(-[-\log p(\boldsymbol{\theta}) - \log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})]). \end{aligned} \tag{6.36}$$

Thus, defining the energy function

$$f(\boldsymbol{\theta}) \doteq -\log p(\boldsymbol{\theta}) - \log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta}) \tag{6.37}$$

$$= -\log p(\boldsymbol{\theta}) - \sum_{i=1}^{n} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}), \tag{6.38}$$

yields

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}) = \frac{1}{Z} \exp(-f(\boldsymbol{\theta})). \tag{6.39}$$

Note that f coincides with the loss function used for MAP estimation (1.62). For a noninformative prior, the regularization term vanishes and the energy reduces to the negative log-likelihood $\ell_{\text{nll}}(\boldsymbol{\theta}; \mathcal{D})$ (i.e., the loss function of maximum likelihood estimation (1.57)).

Using that the posterior is a Gibbs distribution, we can rewrite the acceptance distribution of Metropolis-Hastings,

$$\alpha(\boldsymbol{x}' \mid \boldsymbol{x}) = \min\left\{1, \frac{r(\boldsymbol{x} \mid \boldsymbol{x}')}{r(\boldsymbol{x}' \mid \boldsymbol{x})} \exp(f(\boldsymbol{x}) - f(\boldsymbol{x}'))\right\}. \tag{6.40}$$

this is obtained by substituting the PDF of a Gibbs distribution for the posterior

Example 6.22: Metropolis-Hastings with Gaussian proposals

Let us consider Metropolis-Hastings with the Gaussian proposal distribution,

$$r(\boldsymbol{x}' \mid \boldsymbol{x}) \doteq \mathcal{N}(\boldsymbol{x}'; \boldsymbol{x}, \tau \boldsymbol{I}). \tag{6.41}$$

Due to the symmetry of Gaussians, we have

$$\frac{r(\boldsymbol{x} \mid \boldsymbol{x}')}{r(\boldsymbol{x}' \mid \boldsymbol{x})} = \frac{\mathcal{N}(\boldsymbol{x}; \boldsymbol{x}', \tau \boldsymbol{I})}{\mathcal{N}(\boldsymbol{x}'; \boldsymbol{x}, \tau \boldsymbol{I})} = 1.$$

Hence, the acceptance distribution is defined by

$$\alpha(\boldsymbol{x}' \mid \boldsymbol{x}) = \min\{1, \exp(f(\boldsymbol{x}) - f(\boldsymbol{x}'))\}. \tag{6.42}$$

Intuitively, when a state with lower energy is proposed, that is $f(\boldsymbol{x}') \leq f(\boldsymbol{x})$, then the proposal will always be accepted. In contrast, if the energy of the proposed state is higher, the acceptance probability decreases exponentially in the difference of energies $f(\boldsymbol{x}) - f(\boldsymbol{x}')$. Thus, Metropolis-Hastings minimizes the energy function, which corresponds to minimizing the negative log-likelihood and negative log-prior. The variance in the proposals τ helps in getting around local optima, but the search direction is uniformly random (i.e., "uninformed").

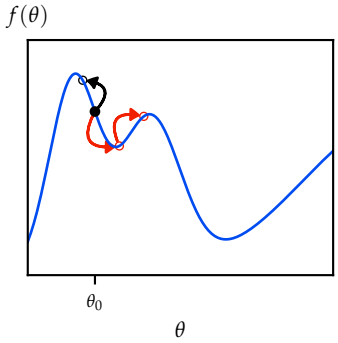


Figure 6.4: Metropolis-Hastings and Langevin dynamics minimize the energy function $f(\theta)$ shown in blue. Suppose we start at the black dot θ_0, then the black and red arrows denote possible subsequent samples. Metropolis-Hastings uses an "uninformed" search direction, whereas Langevin dynamics uses the gradient of $f(\theta)$ to make "more promising" proposals. The random proposals help get past local optima.

6.3.2 *From Energy to Surprise (and back)*

Energy-based models are a well-known class of models in machine learning where an energy function f is learned from data. These energy functions do not need to originate from a probability distribution, yet they induce a probability distribution via their Gibbs distribution $p(\boldsymbol{x}) \propto \exp(-f(\boldsymbol{x}))$. As we will see in Problem 6.7, this Gibbs distribution is the associated maximum entropy distribution. Observe that the surprise about $\boldsymbol{x}$ under this distribution is given by

$$\mathrm{S}[p(\boldsymbol{x})] = f(\boldsymbol{x}) + \log Z. \tag{6.43}$$

That is, up to a constant shift, the energy of $\boldsymbol{x}$ *coincides* with the surprise about $\boldsymbol{x}$. Energies are therefore sufficient for comparing the "likelihood" of points, and they do not require normalization.[7]

What kind of energies could we use? In Section 6.3.1, we discussed the use of the negative log-posterior or negative log-likelihood as energies. In general, *any* loss function $\ell(\boldsymbol{x})$ can be thought of as an energy function with an associated maximum entropy distribution $p(\boldsymbol{x}) \propto \exp(-\ell(\boldsymbol{x}))$.

[7] Intuitively, an energy can be used to compare the "likelihood" of two points $\boldsymbol{x}$ and $\boldsymbol{x}'$ whereas the probability $\boldsymbol{x}$ makes a statement about the "likelihood" of $\boldsymbol{x}$ relative to *all* other points.

6.3.3 *Langevin Dynamics*

Until now, we have looked at Metropolis-Hastings algorithms with proposal distributions that do not explicitly take into account the curvature of the energy function around the current state. Langevin dynamics adapts the Gaussian proposals of the Metropolis-Hastings algorithm we have seen in Example 6.22 to search the state space in an "informed" direction. The simple idea is to bias the sampling towards states with lower energy, thereby making it more likely that a proposal is accepted.

A natural idea is to shift the proposal distribution perpendicularly to the gradient of the energy function. This yields the following proposal distribution,

$$r(\boldsymbol{x}' \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}'; \boldsymbol{x} - \eta_t \nabla f(\boldsymbol{x}), 2\eta_t \boldsymbol{I}). \tag{6.44}$$

The resulting variant of Metropolis-Hastings is known as the *Metropolis adjusted Langevin algorithm* (MALA) or *Langevin Monte Carlo* (LMC). It can be shown that, as $\eta_t \to 0$, we have for the acceptance probability $\alpha(\boldsymbol{x}' \mid \boldsymbol{x}) \to 1$ using that the acceptance probability is 1 if $\boldsymbol{x}' = \boldsymbol{x}$. Hence, the Metropolis-Hastings acceptance step can be omitted once the rejection probability becomes negligible. The algorithm which always accepts the proposal of Equation (6.44) is known as the *unadjusted Langevin algorithm* (ULA).

Observe that if the stationary distribution is the posterior distribution

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n}) = \frac{1}{Z} \exp \underbrace{\left(\log p(\boldsymbol{\theta}) + \sum_{i=1}^{n} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) \right)}_{-f(\boldsymbol{\theta})}$$

with energy f as we discussed in Section 6.3.1, then the proposal $\boldsymbol{\theta}'$ of MALA/LMC can be equivalently formulated as

$$\begin{aligned} \boldsymbol{\theta}' &\leftarrow \boldsymbol{\theta} - \eta_t \nabla f(\boldsymbol{\theta}) + \boldsymbol{\varepsilon} \\ &= \boldsymbol{\theta} + \eta_t \left(\nabla \log p(\boldsymbol{\theta}) + \sum_{i=1}^{n} \nabla \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) \right) + \boldsymbol{\varepsilon} \end{aligned} \tag{6.45}$$

where $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, 2\eta_t \boldsymbol{I})$.

Remark 6.23: ULA is a discretized diffusion

The unadjusted Langevin algorithm can be seen as a discretization of Langevin dynamics, which is a continuous-time stochastic process with a drift and with random stationary and independent Gaussian increments. The randomness is modeled by a Wiener process.

Definition 6.24 (Wiener process)**.** The *Wiener process* (also known as *Brownian motion*) is a sequence of random vectors $\{\mathbf{W}_t\}_{t \geq 0}$ such that

1. $\mathbf{W}_0 = \boldsymbol{0}$,
2. with probability 1, $\mathbf{W}_t$ is continuous in t,
3. the process has independent increments,[8] and
4. $\mathbf{W}_{t+u} - \mathbf{W}_t \sim \mathcal{N}(\boldsymbol{0}, u\boldsymbol{I})$.

Consider the continuous-time stochastic process $\boldsymbol{\theta}$ defined by the

[8] That is, the "future" increments $\mathbf{W}_{t+u} - \mathbf{W}_t$ for $u \geq 0$ are independent of past values $\mathbf{W}_s$ for $s < t$.

stochastic differential equation (SDE)

$$d\boldsymbol{\theta}_t = \underbrace{-\boldsymbol{\nabla} f(\boldsymbol{\theta}_t)\, dt}_{\text{drift}} + \underbrace{\sqrt{2}\, d\mathbf{W}_t}_{\text{noise}}, \tag{6.46}$$

Such a stochastic process is also called a *diffusion (process)* and Equation (6.46) specifically is called *Langevin dynamics*. Here, the first term is called the "drift" of the process, and the second term is called its "noise". Note that if the noise term is zero then Equation (6.46) is simply an ordinary differential equation (ODE).

A diffusion can be discretized using the Euler-Maruyama method (also called "forward Euler") to obtain a discrete approximation $\boldsymbol{\theta}_k \approx \boldsymbol{\theta}(\tau_k)$ where τ_k denotes the k-th time step. Choosing the time steps such that $\Delta t_k \doteq \tau_{k+1} - \tau_k = \eta_k$ yields the approximation

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \boldsymbol{\nabla} f(\boldsymbol{\theta}_k)\, \Delta t_k + \sqrt{2}\, \Delta \mathbf{W}_k \tag{6.47}$$

where $\Delta \mathbf{W}_k \doteq \mathbf{W}_{\tau_{k+1}} - \mathbf{W}_{\tau_k} \sim \mathcal{N}(\mathbf{0}, \Delta t_k \boldsymbol{I})$. Observe that this coincides with the update rule of Langevin dynamics from Equation (6.45).

The appearance of drift and noise is closely related to the *principle of curiosity and conformity*, we encountered in the previous chapter on variational inference. The noise term (also called the *diffusion*) leads to exploration of the state space (i.e., curiosity about alternative explanations of the data), whereas the drift term (also called the *distillation*) leads to minimizing the energy or loss (i.e., conformity to the data). Interestingly, this same principle appears in both variational inference and MCMC, two very different approaches to approximate probabilistic inference. In the remainder of this manuscript we will find this to be a reoccurring theme.

For log-concave distributions, the mixing time of the Markov chain underlying Langevin dynamics can be shown to be polynomial in the dimension n (Vempala and Wibisono, 2019). You will prove this result for strongly log-concave distributions in (?) and see that the analysis is analogous to the canonical convergence analysis of classical optimization schemes. **Problem 6.9**

6.3.4 Stochastic Gradient Langevin Dynamics

Note that computing the gradient of the energy function, which corresponds to computing exact gradients of the log-prior and log-likelihood, in every step can be expensive. The proposal step of MALA/LMC can be made more efficient by approximating the gradient with an unbi-

ased gradient estimate, leading to *stochastic gradient Langevin dynamics* (SGLD) shown in Algorithm 6.25 (Welling and Teh, 2011). Observe that SGLD (6.48) differs from MALA/LMC (6.45) only by using a sample-based approximation of the gradient.

Algorithm 6.25: Stochastic gradient Langevin dynamics, SGLD

initialize $\boldsymbol{\theta}$
for $t = 1$ **to** T **do**
  sample $i_1, \dots, i_m \sim \mathrm{Unif}(\{1, \dots, n\})$ independently
  sample $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, 2\eta_t \boldsymbol{I})$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_t \Big(\boldsymbol{\nabla} \log p(\boldsymbol{\theta}) + \frac{n}{m} \sum_{j=1}^{m} \boldsymbol{\nabla} \log p(y_{i_j} \mid \boldsymbol{x}_{i_j}, \boldsymbol{\theta}) \Big) + \boldsymbol{\varepsilon}$ // (6.48)

Intuitively, in the initial phase of the algorithm, the stochastic gradient term dominates, and therefore, SGLD corresponds to a variant of stochastic gradient ascent. In the later phase, the update rule is dominated by the injected noise $\boldsymbol{\varepsilon}$, and will effectively be Langevin dynamics. SGLD transitions smoothly between the two phases.

Under additional assumptions, SGLD is guaranteed to converge to the posterior distribution for decreasing learning rates $\eta_t = \Theta(t^{-1/3})$ (Raginsky et al., 2017; Xu et al., 2018). SGLD does not use the acceptance step from Metropolis-Hastings as asymptotically, SGLD corresponds to Langevin dynamics and the Metropolis-Hastings rejection probability goes to zero for a decreasing learning rate.

6.3.5 *Hamiltonian Monte Carlo*

As MALA and SGLD can be seen as a sampling-based analogue of GD and SGD, a similar analogue for (stochastic) gradient descent with momentum is the (stochastic gradient) *Hamiltonian Monte Carlo* (HMC) algorithm, which we discuss in the following (Duane et al., 1987; Chen et al., 2014).

We have seen that if we want to sample from a distribution

$$p(\boldsymbol{x}) \propto \exp(-f(\boldsymbol{x}))$$

with energy function f, we can construct a Markov chain whose distribution converges to p. We have also seen that for this approach to work, the chain must move through all areas of significant probability with reasonable speed.

If one is faced with a distribution p which is multimodal (i.e., that has several "peaks"), one has to ensure that the chain will explore all modes, and can therefore "jump between different areas of the space".

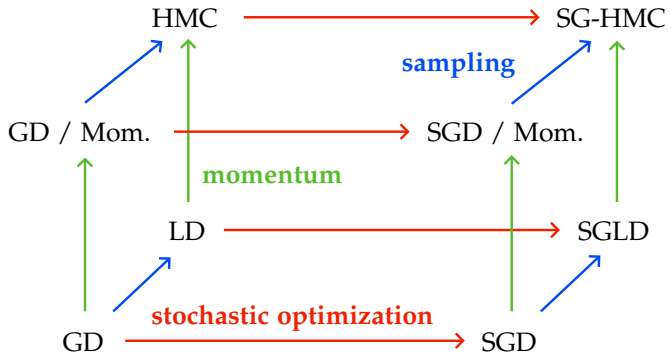


Figure 6.5: A commutative diagram of sampling and optimization algorithms. Langevin dynamics (LD) is the non-stochastic variant of SGLD.

So in general, *local* updates are doomed to fail. Methods such as Metropolis-Hastings with Gaussian proposals, or even Langevin Monte Carlo might face this issue, as they do not jump to distant areas of the state space with significant acceptance probability. It will therefore take a long time to move from one peak to another.

The HMC algorithm is an instance of Metropolis-Hastings which uses momentum to propose distant points that conserve energy, with high acceptance probability. The general idea of HMC is to *lift* samples $\boldsymbol{x}$ to a higher-order space by considering an auxiliary variable $\boldsymbol{y}$ with the same dimension as $\boldsymbol{x}$. We also lift the distribution p to a distribution on the $(\boldsymbol{x}, \boldsymbol{y})$-space by defining a distribution $p(\boldsymbol{y} \mid \boldsymbol{x})$ and setting $p(\boldsymbol{x}, \boldsymbol{y}) \doteq p(\boldsymbol{y} \mid \boldsymbol{x}) p(\boldsymbol{x})$. It is common to pick $p(\boldsymbol{y} \mid \boldsymbol{x})$ to be a Gaussian with zero mean and variance $m\boldsymbol{I}$. Hence,

$$p(\boldsymbol{x}, \boldsymbol{y}) \propto \exp\left(-\frac{1}{2m} \|\boldsymbol{y}\|_2^2 - f(\boldsymbol{x})\right). \tag{6.49}$$

Physicists might recognize the above as the canonical distribution of a Newtonian system if one takes $\boldsymbol{x}$ as the position and $\boldsymbol{y}$ as the momentum. $H(\boldsymbol{x}, \boldsymbol{y}) \doteq \frac{1}{2m} \|\boldsymbol{y}\|_2^2 + f(\boldsymbol{x})$ is called the *Hamiltonian*. HMC then takes a step in this higher-order space according to the Hamiltonian dynamics,[9]

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{\nabla}_{\boldsymbol{y}} H, \quad \frac{d\boldsymbol{y}}{dt} = -\boldsymbol{\nabla}_{\boldsymbol{x}} H, \tag{6.50}$$

reaching some new point $(\boldsymbol{x}', \boldsymbol{y}')$ and *projecting* back to the state space by selecting $\boldsymbol{x}'$ as the new sample. This is illustrated in Figure 6.6. In the next iteration, we resample the momentum $\boldsymbol{y}' \sim p(\cdot \mid \boldsymbol{x}')$ and repeat the procedure.

[9] That is, HMC follows the trajectory of these dynamics for some time.

In an implementation of this algorithm, one has to solve Equation (6.50) numerically rather than exactly. Typically, this is done using the *Leapfrog*

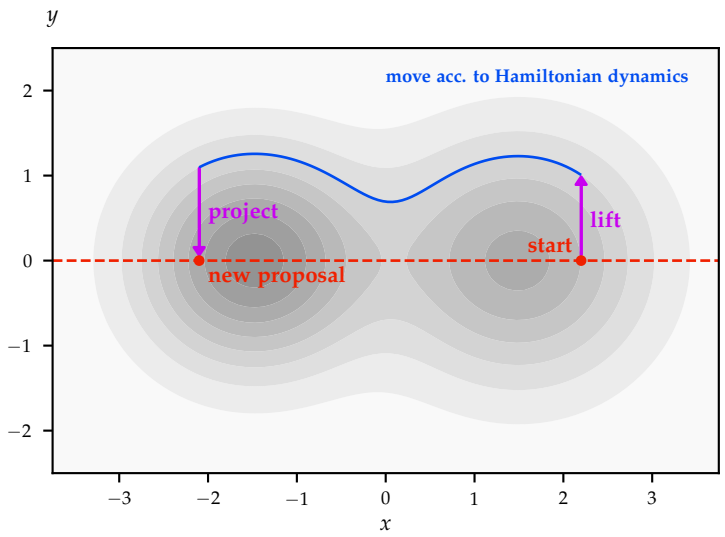


Figure 6.6: Illustration of Hamiltonian Monte Carlo. Shown is the contour plot of a distribution p, which is a mixture of two Gaussians, in the (x, y)-space.

First, the initial point in the state space is lifted to the (x, y)-space. Then, we move according to Hamiltonian dynamics and finally project back onto the state space.

method, which for a step size τ computes

$$\boldsymbol{y}(t + \tau/2) = \boldsymbol{y}(t) - \frac{\tau}{2}\nabla_{\boldsymbol{x}} f(\boldsymbol{x}(t)) \tag{6.51a}$$

$$\boldsymbol{x}(t + \tau) = \boldsymbol{x}(t) + \frac{\tau}{m}\boldsymbol{y}(t + \tau/2) \tag{6.51b}$$

$$\boldsymbol{y}(t + \tau) = \boldsymbol{y}(t + \tau/2) - \frac{\tau}{2}\nabla_{\boldsymbol{x}} f(\boldsymbol{x}(t + \tau)). \tag{6.51c}$$

Then, one repeats this procedure L times to arrive at a point $(\boldsymbol{x}', \boldsymbol{y}')$. To correct for the resulting discretization error, the proposal is either accepted or rejected in a final Metropolis-Hastings acceptance step. If the proposal distribution is symmetric (which we will confirm in a moment), the acceptance probability is

$$\alpha((\boldsymbol{x}', \boldsymbol{y}') \mid (\boldsymbol{x}, \boldsymbol{y})) \doteq \min\{1, \exp(H(\boldsymbol{x}', \boldsymbol{y}') - H(\boldsymbol{x}, \boldsymbol{y}))\}. \tag{6.52}$$

It follows that $p(\boldsymbol{x}, \boldsymbol{y})$ is the stationary distribution of the Markov chain underlying HMC. Due to the independence of $\boldsymbol{x}$ and $\boldsymbol{y}$, this also implies that the projection to $\boldsymbol{x}$ yields a Markov chain with stationary distribution $p(\boldsymbol{x})$.

So why is the proposal distribution symmetric? This follows from the time-reversibility of Hamiltonian dynamics. It is straightforward to check that the dynamics from Equation (6.50) are identical if we replace t with $-t$ and $\boldsymbol{y}$ with $-\boldsymbol{y}$. Intuitively, unlike the position $\boldsymbol{x}$, the momentum $\boldsymbol{y}$ is reversed when time is reversed as it depends on the velocity which is the time-derivative of the position.[10] In simpler terms, time-reversibility states that if we observe the evolution of a system (e.g., two billiard balls colliding), we cannot distinguish whether

[10] The momentum of the i-th coordinate is $y_i = m_i v_i$ where m_i is the mass and $v_i = \frac{dx_i}{dt}$ is the velocity.

we are observing the system evolve forward or backward in time. The Leapfrog method maintains the time-reversibility of the dynamics.

Symmetry of the proposal distribution is ensured by proposing the point $(\boldsymbol{x}', -\boldsymbol{y}')$.[11] Intuitively, this is simply to ensure that the system is run backward in time as often as it is run forward in time. Recall that the momentum is resampled before each iteration (i.e., the proposed momentum is "discarded") and observe that $p(\boldsymbol{x}', -\boldsymbol{y}') = p(\boldsymbol{x}', \boldsymbol{y}')$,[12] so we can safely disregard the direction of time when computing the acceptance probability in Equation (6.52).

[11] More formally, the proposal distribution is the Dirac delta at $(\boldsymbol{x}', -\boldsymbol{y}')$.

[12] We use here that $p(\boldsymbol{y} \mid \boldsymbol{x})$ was chosen to be symmetric around zero.

Discussion

To summarize, Markov chain Monte Carlo methods use a Markov chain to approximately sample from an intractable distribution. Note that unlike for variational inference, the convergence of many methods can be guaranteed. Moreover, for log-concave distributions (e.g., with Bayesian logistic regression), the underlying Markov chain converges quickly to the stationary distribution. Methods such as Langevin dynamics and Hamiltonian Monte Carlo aim to accelerate mixing by proposing points with a higher acceptance probability than Metropolis-Hastings with "undirected" Gaussian proposals. Nevertheless, in general, the convergence (mixing time) may be slow, meaning that, in practice, accuracy and efficiency have to be traded.

Optional Readings

- Ma, Chen, Jin, Flammarion, and Jordan (2019).
  Sampling can be faster than optimization.
- Teh, Thiery, and Vollmer (2016).
  Consistency and fluctuations for stochastic gradient Langevin dynamics.
- Chen, Fox, and Guestrin (2014).
  Stochastic gradient hamiltonian monte carlo.

Problems

6.1. Markov chain update.

Prove Equation (6.9), i.e., that one iteration of the Markov chain can be expressed as $\boldsymbol{q}_{t+1} = \boldsymbol{q}_t \boldsymbol{P}$.

6.2. k-step transitions.

Prove that the entry $\boldsymbol{P}^k(x, x')$ corresponds to the probability of transitioning from state $x \in S$ to state $x' \in S$ in exactly k steps.

6.3. Finding stationary distributions.

A news station classifies each day as "good", "fair", or "poor" based on its daily ratings which fluctuate with what is occurring in the news. Moreover, the following table shows the probabilistic relationship between the type of the current day and the probability of the type of the next day conditioned on the type of the current day.

		next day		
		good	fair	poor
current day	good	0.60	0.30	0.10
	fair	0.50	0.25	0.25
	poor	0.20	0.40	0.40

In the long run, what percentage of news days will be classified as "good"?

6.4. Example of Metropolis-Hastings.

Consider the state space $\{0,1\}^n$ of binary strings having length n. Let the proposal distribution be $r(x' \mid x) = 1/n$ if x' differs from x in exactly one bit and $r(x' \mid x) = 0$ otherwise. Suppose we desire a stationary distribution p for which $p(x)$ is proportional to the number of ones that occur in the bit string x. For example, in the long run, a random walk should visit a string having five 1s five times as often as it visits a string having only a single 1. Provide a general formula for the acceptance probability $\alpha(x' \mid x)$ that would be used if we were to obtain the desired stationary distribution used the Metropolis-Hastings algorithm.

6.5. Practical examples of Gibbs sampling.

In this exercise, we look at some examples where Gibbs sampling is useful.

1. Consider the distribution

$$p(x,y) \doteq \binom{n}{x} y^{x+\alpha-1}(1-y)^{n-x+\beta-1}, \quad x \in [n], y \in [0,1].$$

   Convince yourself that it is hard to sample directly from p and prove that it is an easy task if one uses Gibbs sampling. That is, show that the conditional distributions $p(x \mid y)$ and $p(y \mid x)$ are easy to sample from.
   Hint: Take a look at the Beta distribution (1.50).

2. Consider the following generative model $p(\mu, \lambda, x_{1:n})$ given by the likelihood $x_{1:n} \mid \mu, \lambda \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \lambda^{-1})$ and the independent priors

$$\mu \sim \mathcal{N}(\mu_0, \lambda_0^{-1}) \quad \text{and} \quad \lambda \sim \text{Gamma}(\alpha, \beta).$$

Gamma distribution The PDF of the *gamma distribution* $\text{Gamma}(\alpha, \beta)$ is defined as

$$\text{Gamma}(x; \alpha, \beta) \propto x^{\alpha-1}e^{-\beta x}, \quad x \in \mathbb{R}_{>0}.$$

A random variable $X \sim \text{Gamma}(\alpha, \beta)$ measures the waiting time until $\alpha > 0$ events occur in a Poisson process with rate $\beta > 0$. In particular, when $\alpha = 1$ then the gamma distribution coincides with the exponential distribution with rate β.

We would like to sample from the posterior $p(\mu, \lambda \mid x_{1:n})$. Show that

$$\mu \mid \lambda, x_{1:n} \sim \mathcal{N}(m_\lambda, l_\lambda^{-1}) \quad \text{and} \quad \lambda \mid \mu, x_{1:n} \sim \text{Gamma}(a_\mu, b_\mu),$$

and derive $m_\lambda, l_\lambda, a_\mu, b_\mu$. Such a prior is called a *semi-conjugate prior* to the likelihood, as the prior on μ is conjugate for any fixed value of λ and vice-versa.

3. Let us assume that $x_{1:n} \mid \alpha, c \overset{\text{iid}}{\sim} \text{Pareto}(\alpha, c)$ and assume the improper prior $p(\alpha, c) \propto \mathbb{1}\{\alpha, c > 0\}$ which corresponds to a noninformative prior. Derive the posterior $p(\alpha, c \mid x_{1:n})$. Then, also derive the conditional distributions $p(\alpha \mid c, x_{1:n})$ and $p(c \mid \alpha, x_{1:n})$, and observe that they correspond to known distributions / are easy to sample from.

6.6. Energy function of Bayesian logistic regression.

Recall from Equation (5.12) that the energy function of Bayesian logistic regression is

$$f(\boldsymbol{w}) = \lambda \|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n} \log(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)), \tag{6.53}$$

which coincided with the standard optimization objective of (regularized) logistic regression.

Show that the posterior distribution of Bayesian logistic regression is log-concave.

6.7. Maximum entropy property of Gibbs distribution.

1. Let X be a random variable supported on the finite set $\mathcal{T} \subset \mathbb{R}$.[13] Show that the Gibbs distribution with energy function $\frac{1}{T} f(x)$ for some temperature scalar $T \in \mathbb{R}$ is the distribution with maximum entropy of all distributions supported on $\mathcal{T}$ that satisfy the constraint $\mathbb{E}[f(X)] < \infty$.
   Hint: Solve the dual problem (analogously to Problem 5.7).
2. What happens for $T \to \{0, \infty\}$?

[13] The same result can be shown to hold for arbitrary compact subsets.

6.8. Energy reduction of Gibbs sampling.

Let $p(\boldsymbol{x})$ be a probability density over $\mathbb{R}^d$, which we want to sample from. Assume that p is a Gibbs distribution with energy function $f : \mathbb{R}^d \to \mathbb{R}$.

In this exercise, we will study a single round of Gibbs sampling with initial state $\boldsymbol{x}$ and final state $\boldsymbol{x}'$ where

$$x'_j = \begin{cases} x'_i & \text{if } j = i \\ x_j & \text{otherwise} \end{cases}$$

for some fixed index i and $x_i' \sim p(\cdot \mid \boldsymbol{x}_{-i})$.

Show that

$$\mathbb{E}_{x_i' \sim p(\cdot \mid \boldsymbol{x}_{-i})}\left[f(\boldsymbol{x}')\right] \leq f(\boldsymbol{x}) - \mathrm{S}[p(x_i \mid \boldsymbol{x}_{-i})] + \mathrm{H}[p(\cdot \mid \boldsymbol{x}_{-i})]. \tag{6.54}$$

That is, the energy is expected to decrease if the surprise of x_i given $\boldsymbol{x}_{-i}$ is larger than the expected surprise of the new x_i' given $\boldsymbol{x}_{-i}$, i.e., $\mathrm{S}[p(x_i \mid \boldsymbol{x}_{-i})] \geq \mathrm{H}[p(\cdot \mid \boldsymbol{x}_{-i})]$.

Hint: Recall the framing of Gibbs sampling as a variant of Metropolis-Hastings and relate this to the acceptance distribution of Metropolis-Hastings when p is a Gibbs distribution.

6.9. Mixing time of Langevin dynamics.

In this exercise, we will show that for certain Gibbs distributions, $p(\boldsymbol{\theta}) \propto \exp(-f(\boldsymbol{\theta}))$, Langevin dynamics is rapidly mixing. To do this, we will observe that Langevin dynamics can be seen as a continuous-time optimization algorithm in the space of distributions.

First, we consider a simpler and more widely-known optimization algorithm, namely the *gradient flow*

$$d\boldsymbol{x}_t = -\boldsymbol{\nabla} f(\boldsymbol{x}_t)\, dt. \tag{6.55}$$

Note that gradient descent is simply the discrete-time approximation of gradient flow just as ULA is the discrete-time approximation of Langevin dynamics. In the analysis of ODEs such as the gradient flow, so-called *Lyapunov functions* are commonly used to prove convergence of $\boldsymbol{x}_t$ to a fixed point (also called an *equilibrium*).

Let us assume that f is α-strongly convex for some $\alpha > 0$, that is,

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}) + \frac{\alpha}{2} \|\boldsymbol{y} - \boldsymbol{x}\|_2^2 \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n. \tag{6.56}$$

In words, f is lower bounded by a quadratic function with curvature α. Moreover, assume w.l.o.g. that f minimized at $f(\mathbf{0}) = 0$.[14]

[14] This can always be achieved by shifting the coordinate system and subtracting a constant from f.

1. Show that f satisfies the *Polyak-Łojasiewicz (PL) inequality*, i.e.,

$$f(\boldsymbol{x}) \leq \frac{1}{2\alpha} \|\boldsymbol{\nabla} f(\boldsymbol{x})\|_2^2 \quad \forall \boldsymbol{x} \in \mathbb{R}^n. \tag{6.57}$$

2. Prove $\frac{d}{dt} f(\boldsymbol{x}_t) \leq -2\alpha f(\boldsymbol{x}_t)$.

Thus, $\mathbf{0}$ is the fixed point of Equation (6.55) and the Lyapunov function f is monotonically decreasing along the trajectory of $\boldsymbol{x}_t$. We recall *Grönwall's inequality* which states that for any real-valued continuous functions $g(t)$ and $\beta(t)$ on the interval $[0, T] \subset \mathbb{R}$ such that $\frac{d}{dt} g(t) \leq \beta(t) g(t)$ for all $t \in [0, T]$ we have

$$g(t) \leq g(0) \exp\left(\int_0^t \beta(s)\, ds\right) \quad \forall t \in [0, T]. \tag{6.58}$$

3. Conclude that $f(\boldsymbol{x}_t) \leq e^{-2\alpha t} f(\boldsymbol{x}_0)$.

Now that we have proven the convergence of gradient flow using f as Lyapunov function, we will follow the same template to prove the convergence of Langevin dynamics to the distribution $p(\boldsymbol{\theta}) \propto \exp(-f(\boldsymbol{\theta}))$. We will use that the evolution of $\{\boldsymbol{\theta}_t\}_{t\geq 0}$ following the Langevin dynamics (6.46) is equivalently characterized by their densities $\{q_t\}_{t\geq 0}$ following the *Fokker-Planck equation*

$$\frac{\partial q_t}{\partial t} = \boldsymbol{\nabla} \cdot (q_t \boldsymbol{\nabla} f) + \Delta q_t. \tag{6.59}$$

Here, $\boldsymbol{\nabla}\cdot$ and Δ are the divergence and Laplacian operators, respectively.[15] Intuitively, the first term of the Fokker-Planck equation corresponds to the drift and its second term corresponds to the diffusion (i.e., the Gaussian noise).

[15] For ease of notation, we omit the explicit dependence of q_t, p, and f on $\boldsymbol{\theta}$.

Remark 6.26: Intuition on vector calculus
Recall that the divergence $\boldsymbol{\nabla} \cdot \mathbf{F}$ of a vector field $\mathbf{F}$ measures the change of volume under the flow of $\mathbf{F}$. That is, if in the small neighborhood of a point $\boldsymbol{x}$, $\mathbf{F}$ points towards $\boldsymbol{x}$, then the divergence at $\boldsymbol{x}$ is negative as the volume shrinks. If $\mathbf{F}$ points away from $\boldsymbol{x}$, then the divergence at $\boldsymbol{x}$ is positive as the volume increases.
The Laplacian $\Delta\varphi = \boldsymbol{\nabla} \cdot (\boldsymbol{\nabla}\varphi)$ of a scalar field φ can be understood intuitively as measuring "heat dissipation". That is, if $\varphi(\boldsymbol{x})$ is smaller than the average value of φ in a small neighborhood of $\boldsymbol{x}$, then the Laplacian at $\boldsymbol{x}$ is positive.

Regarding the Fokker-Planck equation (6.59), the second term Δq_t can therefore be understood as locally dissipating the probability mass of q_t (which is due to the diffusion term in the SDE). On the other hand, the term $\boldsymbol{\nabla} \cdot (q_t \boldsymbol{\nabla} f)$ can be understood as a Laplacian of f "weighted" by q_t. Intuitively, the vector field $\boldsymbol{\nabla} f$ moves flow in the direction of high energy, and hence, its divergence is larger in regions of lower energy and smaller in regions of higher energy. This term therefore corresponds to a drift from regions of high energy to regions of low energy.

4. Show that $\Delta q_t = \boldsymbol{\nabla} \cdot (q_t \boldsymbol{\nabla} \log q_t)$, implying that the Fokker-Planck equation simplifies to

$$\frac{\partial q_t}{\partial t} = \boldsymbol{\nabla} \cdot \left(q_t \boldsymbol{\nabla} \log \frac{q_t}{p} \right). \tag{6.60}$$

Hint: The Laplacian of a scalar field φ is $\Delta\varphi \doteq \boldsymbol{\nabla} \cdot (\boldsymbol{\nabla}\varphi)$.

Observe that the Fokker-Planck equation already implies that p is indeed a stationary distribution, as if $q_t = p$ then $\frac{\partial q_t}{\partial t} = 0$. Moreover, note

the similarity of the integrand of $\mathrm{KL}(q_t\|p)$, $q_t \log \frac{q_t}{p}$, to Equation (6.60). We will therefore use the KL-divergence as Lyapunov function.

5. Prove $\frac{d}{dt}\mathrm{KL}(q_t\|p) = -\mathrm{J}(q_t\|p)$. Here,

$$\mathrm{J}(q_t\|p) \doteq \mathbb{E}_{\boldsymbol{\theta}\sim q_t}\left[\left\|\boldsymbol{\nabla}\log\frac{q_t(\boldsymbol{\theta})}{p(\boldsymbol{\theta})}\right\|_2^2\right] \tag{6.61}$$

denotes the *relative Fisher information* of p with respect to q_t. *Hint: For any distribution q on $\mathbb{R}^n$,*

$$\int_{\mathbb{R}^n}(\boldsymbol{\nabla}\cdot q\mathbf{F})\varphi\, d\boldsymbol{x} = -\int_{\mathbb{R}^n} q\,\boldsymbol{\nabla}\varphi\cdot\mathbf{F}\, d\boldsymbol{x} \tag{6.62}$$

follows for any vector field $\mathbf{F}$ and scalar field φ from the divergence theorem and the product rule of the divergence operator.

Thus, the relative Fisher information can be seen as the negated time-derivative of the KL-divergence, and as $\mathrm{J}(q_t\|p) \geq 0$ it follows that the KL-divergence is decreasing along the trajectory.

The *log-Sobolev inequality* (LSI) is satisfied by a distribution p with a constant $\alpha > 0$ if for all q:

$$\mathrm{KL}(q\|p) \leq \frac{1}{2\alpha}\mathrm{J}(q\|p). \tag{6.63}$$

It is a classical result that if f is α-strongly convex then p satisfies the LSI with constant α (Bakry and Émery, 2006).

6. Show that if f is α-strongly convex for some $\alpha > 0$ (we say that p is "strongly log-concave"), then $\mathrm{KL}(q_t\|p) \leq e^{-2\alpha t}\mathrm{KL}(q_0\|p)$.
7. Conclude that under the same assumption on f, Langevin dynamics is rapidly mixing, i.e., $\tau_{\mathrm{TV}}(\epsilon) \in O(\mathrm{poly}(n, \log(1/\epsilon)))$.

To summarize, we have seen that Langevin dynamics is an optimization scheme in the space of distributions, and that its convergence can be analyzed analogously to classical optimization schemes. Notably, in this exercise we have studied continuous-time Langevin dynamics. Convergence guarantees for discrete-time approximations can be derived using the same techniques. If this interests you, refer to "Rapid convergence of the unadjusted Langevin algorithm: Isoperimetry suffices" (Vempala and Wibisono, 2019).

6.10. Hamiltonian Monte Carlo.

1. Prove that if the dynamics are solved exactly (as opposed to numerically using the Leapfrog method), then the acceptance probability of the MH-step is always 1.
2. Prove that the Langevin Monte Carlo algorithm from (6.44) can be seen as a special case of HMC if only one Leapfrog step is used ($L = 1$) and $m = 1$.

7
Deep Learning

We began our journey through probabilistic machine learning with linear models. In most practical applications, however, it is seen that models perform better when labels may nonlinearly depend on the inputs, and for this reason linear models are often used in conjunction with "hand-designed" nonlinear features. Designing these features is costly, time-consuming, and requires expert knowledge. Moreover, the design of such features is inherently limited by human comprehension and ingenuity.

7.1 Artificial Neural Networks

One widely used family of nonlinear functions are *artificial "deep" neural networks*,[1]

$$f : \mathbb{R}^d \to \mathbb{R}^k, \quad f(x;\theta) \doteq \varphi(W_L \varphi(W_{L-1}(\cdots \varphi(W_1 x))))) \tag{7.1}$$

where $\theta \doteq [W_1, \ldots, W_L]$ is a vector of *weights* (written as matrices $W_l \in \mathbb{R}^{n_l \times n_{l-1}}$)[2] and $\varphi : \mathbb{R} \to \mathbb{R}$ is a component-wise nonlinear function. Thus, a deep neural network can be seen as nested ("deep") linear functions composed with nonlinearities. This simple kind of neural network is also called a *multilayer perceptron*.

[1] In the following, we will refrain from using the characterizations "artificial" and "deep" for better readability.

[2] where $n_0 = d$ and $n_L = k$

In this chapter, we will be focusing mostly on performing probabilistic inference with a *given* neural network architecture. To this end, understanding the basic architecture of a multilayer perceptron will be sufficient for us. For a more thorough introduction to the field of deep learning and various architectures, you may refer to the books of Goodfellow et al. (2016) and Prince (2023).

A neural network can be visualized by a *computation graph*. An example for such a computation graph is given in Figure 7.1. The columns of the computation graph are commonly called *layers*, whereby the

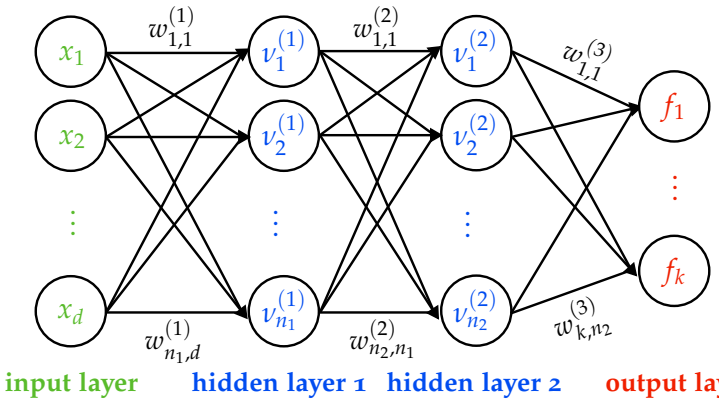


Figure 7.1: Computation graph of a neural network with two hidden layers.

left-most column is the *input layer*, the right-most column is the *output layer*, and the remaining columns are the *hidden layers*. The inputs are (as we have previously) referred to as $\boldsymbol{x} \doteq [x_1, \dots, x_d]$. The outputs (i.e., vertices of the output layer) are often referred to as *logits* and named $\boldsymbol{f} \doteq [f_1, \dots, f_k]$. The *activations* of an individual (hidden) layer l of the neural network are described by

$$\boldsymbol{v}^{(l)} \doteq \boldsymbol{\varphi}(\boldsymbol{W}_l \boldsymbol{v}^{(l-1)}) \tag{7.2}$$

where $\boldsymbol{v}^{(0)} = \boldsymbol{x}$. The activation of the i-th node is $v_i^{(l)} = \boldsymbol{v}^{(l)}(i)$.

7.1.1 Activation Functions

The nonlinearity φ is called an *activation function*. The following two activation functions are particularly common:

1. The *hyperbolic tangent* (Tanh) is defined as

$$\operatorname{Tanh}(z) \doteq \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \in (-1, 1). \tag{7.3}$$

   Tanh is a scaled and shifted variant of the sigmoid function (5.9) which we have previously seen in the context of logistic regression as $\operatorname{Tanh}(z) = 2\sigma(2z) - 1$.
2. The *rectified linear unit* (ReLU) is defined as

$$\operatorname{ReLU}(z) \doteq \max\{z, 0\} \in [0, \infty). \tag{7.4}$$

   In particular, the ReLU activation function leads to "sparser" gradients as it selects the halfspace of inputs with positive sign. Moreover, the gradients of ReLU do not "vanish" as $z \to \pm\infty$ which can lead to faster training.

It is important that the activation function is nonlinear because otherwise, any composition of layers would still represent a linear function. Non-linear activation functions allow the network to represent arbitrary functions. This is known as the *universal approximation theorem*,

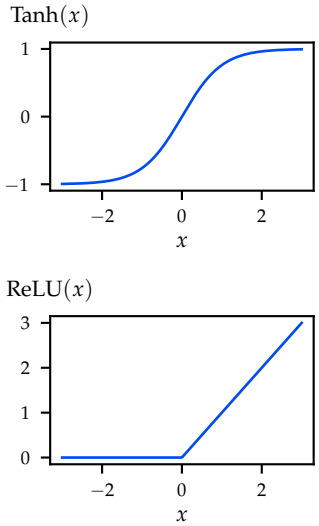


Figure 7.2: The Tanh and ReLU activation functions, respectively.

and it states that any artificial neural network with just a single hidden layer (with arbitrary width) and non-polynomial activation function φ can approximate any continuous function to an arbitrary accuracy.

7.1.2 Classification

Although we mainly focus on regression, neural networks can equally well be used for classification. If we want to classify inputs into c separate classes, we can simply construct a neural network with c outputs, $\boldsymbol{f} = [f_1, \dots, f_c]$, and normalize them into a probability distribution. Often, the *softmax function* is used for normalization,

$$\sigma_i(\boldsymbol{f}) \doteq \frac{\exp(f_i)}{\sum_{j=1}^{c} \exp(f_j)} \tag{7.5}$$

where $\sigma_i(\boldsymbol{f})$ corresponds to the probability mass of class i. The softmax is a generalization of the logistic function (5.9) to more than two classes (?). Note that the softmax corresponds to a Gibbs distribution with energies $-\boldsymbol{f}$.

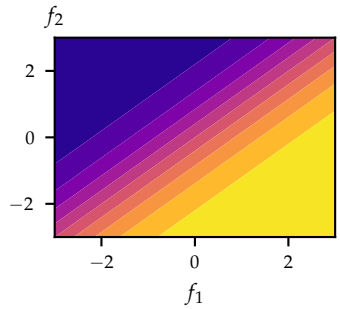


Figure 7.3: Softmax $\sigma_1(f_1, f_2)$ for a binary classification problem. Blue denotes a small probability and yellow denotes a large probability of belonging to class 1, respectively.

Problem 7.1

7.1.3 Maximum Likelihood Estimation

We will study neural networks under the lens of supervised learning (cf. Section 1.3) where we are provided some independently-sampled (noisy) data $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{n}$ generated according to an unknown process $\boldsymbol{y} \sim p(\cdot \mid \boldsymbol{x}, \boldsymbol{\theta}^\star)$, which we wish to approximate.

Upon initialization, the network does generally not approximate this process well, so a key element of deep learning is "learning" a parameterization $\boldsymbol{\theta}$ that is a good approximation. To this end, one typically considers a loss function $\ell(\boldsymbol{\theta}; \boldsymbol{y})$ which quantifies the "error" of the network outputs $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$. In the classical setting of regression, i.e., $y \in \mathbb{R}$ and $k = 1$, ℓ is often taken to be the *(empirical) mean squared error*,

$$\ell_{\text{mse}}(\boldsymbol{\theta}; \mathcal{D}) \doteq \frac{1}{n} \sum_{i=1}^{n} (f(\boldsymbol{x}_i; \boldsymbol{\theta}) - y_i)^2. \tag{7.6}$$

As we have already seen in Section 2.0.1 in the context of linear regression, minimizing mean squared error corresponds to maximum likelihood estimation under a Gaussian likelihood.

In the setting of classification where $\boldsymbol{y} \in \{0, 1\}^c$ is a one-hot encoding of class membership,[3] it is instead common to interpret the outputs of a neural network as probabilities akin to our discussion in Section 7.1.2. We denote by $q_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x})$ the resulting probability distribution over classes with PMF $[\sigma_1(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})), \dots, \sigma_c(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}))]$, and aim to find $\boldsymbol{\theta}$ such that $q_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \approx p(\boldsymbol{y} \mid \boldsymbol{x})$. In this context, it is common to

[3] That is, exactly one component of $\boldsymbol{y}$ is 1 and all others are 0, indicating to which class the given example belongs.

minimize the cross-entropy,

$$\begin{aligned} \mathrm{H}[p \| q_{\boldsymbol{\theta}}] &= \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p}[-\log q_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})] && \text{using the definition of cross-entropy (5.32)} \\ &\approx \underbrace{-\frac{1}{n}\sum_{i=1}^{n} \log q_{\boldsymbol{\theta}}(\boldsymbol{y}_i \mid \boldsymbol{x}_i)}_{\doteq \ell_{\mathrm{ce}}(\boldsymbol{\theta}; \mathcal{D})} && \text{using Monte Carlo sampling} \end{aligned} \tag{7.7}$$

which can be understood as minimizing the surprise about the training data under the model. ℓ_{ce} is called the *cross-entropy loss*. Disregarding the constant $1/n$, we can rewrite the cross-entropy loss as

$$\propto -\sum_{i=1}^{n} \log q_{\boldsymbol{\theta}}(\boldsymbol{y}_i \mid \boldsymbol{x}_i) = \ell_{\mathrm{nll}}(\boldsymbol{\theta}; \mathcal{D}) \tag{7.8}$$

Recall that $\ell_{\mathrm{nll}}(\boldsymbol{\theta}; \mathcal{D})$ is the *negative log-likelihood* of the training data, and thus, empirically minimizing cross-entropy can equivalently be interpreted as maximum likelihood estimation.[4] Furthermore, recall from Problem 5.1 that for a two-class classification problem the cross-entropy loss is equivalent to the logistic loss.

[4] We have previously seen this equivalence of MLE and empirically minimizing KL-divergence in Section 5.4.6 (minimizing the cross-entropy $\mathrm{H}[p\|q_{\boldsymbol{\theta}}]$ is equivalent to minimizing forward-KL $\mathrm{KL}(p\|q_{\boldsymbol{\theta}})$). Note that this interpretation is not exclusive to the canonical cross-entropy loss from Equation (7.7), but holds for any MLE. For example, minimizing mean squared error corresponds to empirically minimizing the KL-divergence with a Gaussian likelihood.

7.1.4 *Backpropagation*

A crucial property of neural networks is that they are differentiable. That is, we can compute gradients $\boldsymbol{\nabla}_{\boldsymbol{\theta}} \ell$ of $\boldsymbol{f}$ with respect to the parameterization of the model $\boldsymbol{\theta} = \boldsymbol{W}_{1:L}$ and some loss function $\ell(\boldsymbol{f}; \boldsymbol{y})$. Being able to obtain these gradients efficiently allows for "learning" a particular function from data using first-order optimization methods.

The algorithm for computing gradients of a neural network is called *backpropagation* and is essentially a repeated application of the chain rule. Note that using the chain rule for every path through the network is computationally infeasible, as this quickly leads to a combinatorial explosion as the number of hidden layers is increased. The key insight of backpropagation is that we can use the *feed-forward* structure of our neural network to memoize computations of the gradient, yielding a linear time algorithm. Obtaining gradients by backpropagation is often called *automatic differentiation (auto-diff)*. For more details, refer to Goodfellow et al. (2016).

Computing the exact gradient for each data point is still fairly expensive when the size of the neural network is large. Typically, stochastic gradient descent is used to obtain unbiased gradient estimates using batches of only m of the n data points, where $m \ll n$.

7.2 *Bayesian Neural Networks*

How can we perform probabilistic inference in neural networks? We adopt the same strategy which we already used for Bayesian linear

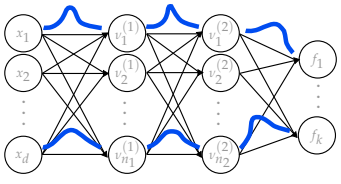


Figure 7.4: Bayesian neural networks model a distribution over the weights of a neural network.

regression, we impose a Gaussian prior on the weights,

$$\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \sigma_{\mathrm{p}}^2 \boldsymbol{I}). \tag{7.9}$$

Similarly, we can use a Gaussian likelihood to describe how well the data is described by the model $\boldsymbol{f}$,

$$y \mid \boldsymbol{x}, \boldsymbol{\theta} \sim \mathcal{N}(f(\boldsymbol{x}; \boldsymbol{\theta}), \sigma_{\mathrm{n}}^2). \tag{7.10}$$

Thus, instead of considering weights as point estimates which are learned exactly, *Bayesian neural networks* learn a distribution over the weights of the network. In principle, other priors and likelihoods can be used, yet Gaussians are typically chosen due to their closedness properties, which we have seen in Section 1.2.3 and many times since.

7.2.1 Maximum a Posteriori Estimation

Before studying probabilistic inference, let us first consider MAP estimation in the context of neural networks.

The MAP estimate of the weights is obtained by

$$\hat{\boldsymbol{\theta}}_{\mathrm{MAP}} = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}) + \sum_{i=1}^{n} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}). \tag{7.11}$$

In Section 7.1.3, we have seen that the negative log-likelihood under a Gaussian likelihood (7.10) is the squared error between label and prediction,

$$\log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) = -\frac{(y_i - f(\boldsymbol{x}_i; \boldsymbol{\theta}))^2}{2\sigma_{\mathrm{n}}^2} + \text{const.} \tag{7.12}$$

Obtaining the MAP estimate instead, simply corresponds to adding an L_2-regularization term to the squared error loss,

$$\hat{\boldsymbol{\theta}}_{\mathrm{MAP}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2\sigma_{\mathrm{p}}^2} \|\boldsymbol{\theta}\|_2^2 + \frac{1}{2\sigma_{\mathrm{n}}^2} \sum_{i=1}^{n} (y_i - f(\boldsymbol{x}_i; \boldsymbol{\theta}))^2. \tag{7.13}$$

using that for an isotropic Gaussian prior, $\log p(\boldsymbol{\theta}) = -\frac{1}{2\sigma_{\mathrm{p}}^2} \|\boldsymbol{\theta}\|_2^2 + \text{const}$

As we have already seen in Remark A.31 and the context of Bayesian linear regression (and ridge regression), using a Gaussian prior is equivalent to applying weight decay.[5] Using gradient ascent, we obtain the following update rule,

[5] Recall that weight decay regularizes weights by shrinking them towards zero.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}(1 - \lambda \eta_t) + \eta_t \sum_{i=1}^{n} \boldsymbol{\nabla} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) \tag{7.14}$$

where $\lambda \doteq 1/\sigma_{\mathrm{p}}^2$. The gradients of the likelihood can be obtained using automatic differentiation.

7.2.2 Heteroscedastic Noise

Equation (7.10) uses the scalar parameter σ_n^2 to model the aleatoric uncertainty (label noise), similarly to how we modeled the label noise in Bayesian linear regression and Gaussian processes. Such a noise model is called *homoscedastic noise* as the noise is assumed to be uniform across the domain. In many settings, however, the noise is inherently *heteroscedastic noise*. That is, the noise varies depending on the input and which "region" of the domain the input is from. This behavior is visualized in Figure 7.5.

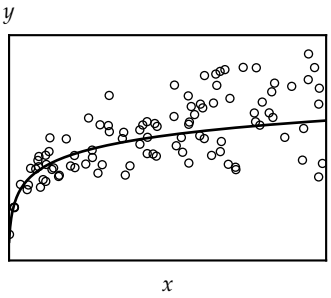


Figure 7.5: Illustration of data with variable (heteroscedastic) noise. The noise increases as the inputs increase in magnitude. The noise-free function is shown in black.

There is a natural way of modeling heteroscedastic noise with Bayesian neural networks. We use a neural network with two outputs f_1 and f_2 and define

$$y \mid \boldsymbol{x}, \boldsymbol{\theta} \sim \mathcal{N}(\mu(\boldsymbol{x}; \boldsymbol{\theta}), \sigma^2(\boldsymbol{x}; \boldsymbol{\theta})) \quad \text{where} \tag{7.15a}$$

$$\mu(\boldsymbol{x}; \boldsymbol{\theta}) \doteq f_1(\boldsymbol{x}; \boldsymbol{\theta}), \tag{7.15b}$$

$$\sigma^2(\boldsymbol{x}; \boldsymbol{\theta}) \doteq \exp(f_2(\boldsymbol{x}; \boldsymbol{\theta})). \tag{7.15c}$$

we exponentiate f_2 to ensure non-negativity of the variance

Using this model, the likelihood term from Equation (7.11) is

$$\begin{aligned}
\log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) &= \log \mathcal{N}(y_i; \mu(\boldsymbol{x}_i; \boldsymbol{\theta}), \sigma^2(\boldsymbol{x}_i; \boldsymbol{\theta})) \\
&= \log \frac{1}{\sqrt{2\pi\sigma^2(\boldsymbol{x}_i; \boldsymbol{\theta})}} - \frac{(y_i - \mu(\boldsymbol{x}_i; \boldsymbol{\theta}))^2}{2\sigma^2(\boldsymbol{x}_i; \boldsymbol{\theta})} \\
&= \underbrace{\log \frac{1}{\sqrt{2\pi}}}_{\text{const}} - \frac{1}{2}\left[\log \sigma^2(\boldsymbol{x}_i; \boldsymbol{\theta}) + \frac{(y_i - \mu(\boldsymbol{x}_i; \boldsymbol{\theta}))^2}{\sigma^2(\boldsymbol{x}_i; \boldsymbol{\theta})}\right].
\end{aligned} \tag{7.16}$$

note that the normalizing constant depends on the noise model!

Hence, the model can either explain the label y_i by an accurate model $\mu(\boldsymbol{x}_i; \boldsymbol{\theta})$ or by a large variance $\sigma^2(\boldsymbol{x}_i; \boldsymbol{\theta})$, yet, it is penalized for choosing large variances. Intuitively, this allows to attenuate losses for some data points by attributing them to large variance (when no model reflecting all data points simultaneously can be found). This allows the model to "learn" its aleatoric uncertainty. However, recall that the MAP estimate still corresponds to a point estimate of the weights, so we forgo modeling the epistemic uncertainty.

7.3 Approximate Probabilistic Inference

Naturally, we want to understand the epistemic uncertainty of our model too. However, learning and inference in Bayesian neural networks are generally intractable (even when using a Gaussian prior and likelihood) when the noise is not assumed to be homoscedastic and known.[6] Thus, we are led to the techniques of approximate inference, which we discussed in the previous two chapters.

[6] In this case, the conjugate prior to a Gaussian likelihood is not a Gaussian. See, e.g., `https://en.wikipedia.org/wiki/Conjugate_prior`.

7.3.1 Variational Inference

As we have discussed in Chapter 5, we can apply black box stochastic variational inference which — in the context of neural networks — is also known as *Bayes by Backprop*. As variational family, we use the family of independent Gaussians which we have already encountered in Example 5.5.[7] Recall the fundamental objective of variational inference (5.25),

[7] Independent Gaussians are useful because they can be encoded using only $2d$ parameters, which is crucial when the size of the neural network is large.

$$\begin{aligned}
&\arg\min_{q\in\mathcal{Q}} \mathrm{KL}(q\|p(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n})) \\
&= \arg\max_{q\in\mathcal{Q}} L(q, p; \mathcal{D}) && \text{using Equation (5.53)} \\
&= \arg\max_{q\in\mathcal{Q}} \mathbb{E}_{\boldsymbol{\theta}\sim q}[\log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})] - \mathrm{KL}(q\|p(\cdot)). && \text{using Equation (5.55c)}
\end{aligned}$$

The KL-divergence $\mathrm{KL}(q\|p(\cdot))$ can be expressed in closed-form for Gaussians.[8] Recall that we can obtain unbiased gradient estimates of the expectation using the reparameterization trick (5.68),

[8] see Equation (5.41)

$$\mathbb{E}_{\boldsymbol{\theta}\sim q}[\log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})] = \mathbb{E}_{\boldsymbol{\varepsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\Big[\log p(y_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\Sigma}^{1/2}\boldsymbol{\varepsilon}+\boldsymbol{\mu}}\Big].$$

As $\boldsymbol{\Sigma}$ is the diagonal matrix $\mathrm{diag}\{\sigma_1^2, \ldots, \sigma_d^2\}$, $\boldsymbol{\Sigma}^{1/2} = \mathrm{diag}\{\sigma_1, \ldots, \sigma_d\}$. The gradients of the likelihood can be obtained using backpropagation. We can now use the variational posterior q_λ to perform approximate inference,

$$\begin{aligned}
p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}) &= \int p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n})\, d\boldsymbol{\theta} && \text{using the sum rule (1.7) and product rule (1.11)} \\
&= \mathbb{E}_{\boldsymbol{\theta}\sim p(\cdot|\boldsymbol{x}_{1:n}, y_{1:n})}[p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta})] && \text{interpreting the integral as an expectation over the posterior} \\
&\approx \mathbb{E}_{\theta\sim q_\lambda}[p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta})] && \text{approximating the posterior with the variational posterior } q_\lambda \\
&\approx \frac{1}{m}\sum_{i=1}^{m} p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}^{(i)}) && \text{using Monte Carlo sampling} \qquad (7.17)
\end{aligned}$$

for independent samples $\boldsymbol{\theta}^{(i)} \overset{\text{iid}}{\sim} q_\lambda$,

$$= \frac{1}{m}\sum_{i=1}^{m} \mathcal{N}(y^\star; \mu(\boldsymbol{x}^\star; \boldsymbol{\theta}^{(i)}), \sigma^2(\boldsymbol{x}^\star; \boldsymbol{\theta}^{(i)})). \qquad (7.18) \quad \text{using the neural network}$$

Intuitively, variational inference in Bayesian neural networks can be interpreted as averaging the predictions of multiple neural networks drawn according to the variational posterior q_λ.

Using the Monte Carlo samples $\boldsymbol{\theta}^{(i)}$, we can also estimate the mean of our predictions,

$$\mathbb{E}[y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}] \approx \frac{1}{m}\sum_{i=1}^{m} \mu(\boldsymbol{x}^\star; \boldsymbol{\theta}^{(i)}) \doteq \overline{\mu}(\boldsymbol{x}^\star), \qquad (7.19)$$

and the variance of our predictions,

$$\begin{aligned}\mathrm{Var}[y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}] &= \mathbb{E}_{\boldsymbol{\theta}}\left[\mathrm{Var}_{y^\star}[y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}]\right] + \mathrm{Var}_{\boldsymbol{\theta}}\left[\mathbb{E}_{y^\star}[y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}]\right] && \text{using the law of total variance (1.41)} \\ &= \mathbb{E}_{\boldsymbol{\theta}}\left[\sigma^2(\boldsymbol{x}^\star; \boldsymbol{\theta})\right] + \mathrm{Var}_{\boldsymbol{\theta}}[\mu(\boldsymbol{x}^\star; \boldsymbol{\theta})]. && \text{using the likelihood (7.15a)}\end{aligned}$$

Recall from Equation (2.18) that the first term corresponds to the aleatoric uncertainty of the data and the second term corresponds to the epistemic uncertainty of the model. We can approximate them using the Monte Carlo samples $\boldsymbol{\theta}^{(i)}$,

$$\begin{aligned}\mathrm{Var}[y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, y_{1:n}] \approx \frac{1}{m}\sum_{i=1}^{m} \sigma^2(\boldsymbol{x}^\star; \boldsymbol{\theta}^{(i)}) \\ + \frac{1}{m-1}\sum_{i=1}^{m}(\mu(\boldsymbol{x}^\star; \boldsymbol{\theta}^{(i)}) - \overline{\mu}(\boldsymbol{x}^\star))^2\end{aligned} \tag{7.20}$$

using a sample mean (A.14) and sample variance (A.16).

7.3.2 *Markov Chain Monte Carlo*

As we have discussed in Chapter 6, an alternative method to approximating the full posterior distribution is to sample from it directly. By the ergodic theorem (6.28), we can use any of the discussed sampling strategies to obtain samples $\boldsymbol{\theta}^{(t)}$ such that

$$p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}) \approx \frac{1}{T}\sum_{t=1}^{T} p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}^{(t)}). \qquad \text{see (6.29)}$$

Here, we omit the offset t_0 which is commonly used to avoid the "burn-in" period for simplicity. Algorithms such as SGLD or SG-HMC are often used as they rely only on stochastic gradients of the loss function which can be computed efficiently using automatic differentiation.

Typically, for large networks, we cannot afford to store all T samples of models. Thus, we need to summarize the iterates.[9] One approach is to keep track of m *snapshots* of weights $[\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(m)}]$ according to some schedule and use those for inference (e.g., by averaging the predictions of the corresponding neural networks). This approach of sampling a subset of some data is generally called *subsampling*.

[9] That is, combine the individual samples of weights $\boldsymbol{\theta}^{(i)}$.

Another approach is to summarize (that is, approximate) the weights using sufficient statistics (e.g., a Gaussian).[10] In other words, we learn the Gaussian approximation,

$$\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad \text{where} \tag{7.21a}$$

$$\boldsymbol{\mu} \doteq \frac{1}{T}\sum_{i=1}^{T} \boldsymbol{\theta}^{(i)}, \tag{7.21b}$$

[10] A statistic is *sufficient* for a family of probability distributions if the samples from which it is calculated yield no more information than the statistic with respect to the learned parameters. We provide a formal definition in Section 8.2.

using a sample mean (A.14)

$$\boldsymbol{\Sigma} \doteq \frac{1}{T-1}\sum_{i=1}^{T}(\boldsymbol{\theta}^{(i)}-\boldsymbol{\mu})(\boldsymbol{\theta}^{(i)}-\boldsymbol{\mu})^\top, \tag{7.21c}$$

using a sample variance (A.16)

using sample means and sample (co)variances. This can be implemented efficiently using running averages of the first and second moments,

$$\boldsymbol{\mu} \leftarrow \frac{1}{T+1}(T\boldsymbol{\mu}+\boldsymbol{\theta}) \quad \text{and} \quad \boldsymbol{A} \leftarrow \frac{1}{T+1}(T\boldsymbol{A}+\boldsymbol{\theta}\boldsymbol{\theta}^\top) \tag{7.22}$$

upon observing the fresh sample $\boldsymbol{\theta}$. $\boldsymbol{\Sigma}$ can easily be calculated from these moments,

$$\boldsymbol{\Sigma} = \frac{T}{T-1}(\boldsymbol{A}-\boldsymbol{\mu}\boldsymbol{\mu}^\top). \tag{7.23}$$

using the characterization of sample variance in terms of estimators of the first and second moment (A.17)

To predict, we can sample weights $\boldsymbol{\theta}$ from the learned Gaussian.

Remark 7.1: Stochastic weight averaging

It turns out that this approach works well even without injecting additional Gaussian noise during training, e.g., when using SGD rather than SGLD. Simply taking the mean of the iterates of SGD is called *stochastic weight averaging* (SWA). Describing the iterates of SGD by Gaussian sufficient statistics (analogously to Equation (7.21)), is known as the *stochastic weight averaging-Gaussian* (SWAG) method (Izmailov et al., 2018).

7.3.3 *Dropout and Dropconnect*

We will now discuss two approximate inference techniques that are tailored to neural networks. The first is "dropout"/"dropconnect" regularization. Traditionally, *dropout regularization* (Hinton et al., 2012; Srivastava et al., 2014) randomly omits vertices of the computation graph during training, see Figure 7.7. In contrast, *dropconnect regularization* (Wan et al., 2013) randomly omits edges of the computation graph. The key idea that we will present here is to interpret this type of regularization as performing variational inference.

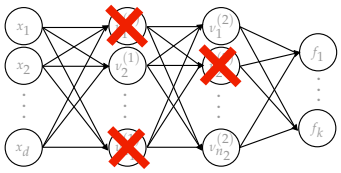


Figure 7.6: Illustration of dropout regularization. Some vertices of the computation graph are randomly omitted. In contrast, dropconnect regularization randomly omits edges of the computation graph.

In our exposition, we will focus on dropconnect, but the same approach also applies to dropout (Gal and Ghahramani, 2016). Suppose that we omit an edge of the computation graph (i.e., set its weight to zero) with probability p. Then our variational posterior is given by

$$q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) \doteq \prod_{j=1}^{d} q_j(\theta_j \mid \lambda_j) \tag{7.24}$$

where d is the number of weights of the neural network and

$$q_j(\theta_j \mid \lambda_j) \doteq p\delta_0(\theta_j) + (1-p)\delta_{\lambda_j}(\theta_j). \tag{7.25}$$

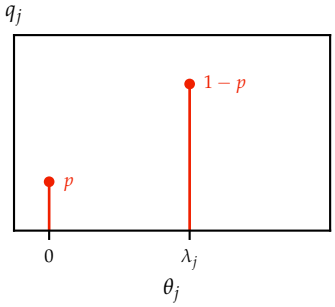


Figure 7.7: Interpretation of dropconnect regularization as variational inference. The only coordinates where the variational posterior q_j has positive probability are 0 and λ_j.

Here, δ_α is the Dirac delta function with point mass at α.[11] The variational parameters $\boldsymbol{\lambda}$ correspond to the "original" weights of the network. In words, the variational posterior expresses that the j-th weight has value 0 with probability p and value λ_j with probability $1 - p$. For fixed weights $\boldsymbol{\lambda}$, sampling from the variational posterior $q_{\boldsymbol{\lambda}}$ corresponds to sampling a vector $\boldsymbol{z}$ with entries $\boldsymbol{z}(i) \sim \text{Bern}(p)$, yielding $\boldsymbol{z} \odot \boldsymbol{\lambda}$ which is one of 2^d possible subnetworks.[12]

[11] see Appendix A.1.4

[12] $\boldsymbol{A} \odot \boldsymbol{B}$ denotes the Hadamard (element-wise) product.

The weights $\boldsymbol{\lambda}$ can be learned by maximizing the ELBO, analogously to black-box variational inference. The KL-divergence term of the ELBO is not tractable for the variational family described by Equation (7.25), instead a common approach is to use a mixture of Gaussians:

$$q_j(\theta_j \mid \lambda_j) \doteq p\mathcal{N}(\theta_j; 0, 1) + (1 - p)\mathcal{N}(\theta_j; \lambda_j, 1). \tag{7.26}$$

In this case, it can be shown that $\text{KL}(q_{\boldsymbol{\lambda}} \| p(\cdot)) \approx \frac{p}{2} \|\boldsymbol{\lambda}\|_2^2$ for sufficiently large d (Gal and Ghahramani, 2015, proposition 1). Thus,

$$\begin{aligned}
&\arg\max_{\boldsymbol{\lambda} \in \Lambda} L(q_{\boldsymbol{\lambda}}, p; \mathcal{D}) \\
&= \arg\max_{\boldsymbol{\lambda} \in \Lambda} \mathbb{E}_{\boldsymbol{\theta} \sim q_{\boldsymbol{\lambda}}}[\log p(\boldsymbol{y}_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})] - \text{KL}(q \| p(\cdot)) && \text{using Equation (5.55c)} \\
&\approx \arg\min_{\boldsymbol{\lambda} \in \Lambda} -\frac{1}{m} \sum_{i=1}^{m} \log p(\boldsymbol{y}_{1:n} \mid \boldsymbol{x}_{1:n}, \boldsymbol{\theta})\big|_{\boldsymbol{\theta} = \boldsymbol{z}^{(i)} \odot \boldsymbol{\lambda} + \boldsymbol{\varepsilon}^{(i)}} + \frac{p}{2} \|\boldsymbol{\lambda}\|_2^2 && \text{using Monte Carlo sampling} \quad (7.27)
\end{aligned}$$

where we reparameterize $\boldsymbol{\theta} \sim q_{\boldsymbol{\lambda}}$ by $\boldsymbol{\theta} = \boldsymbol{z} \odot \boldsymbol{\lambda} + \boldsymbol{\varepsilon}$ with $\boldsymbol{z}(i) \sim \text{Bern}(p)$ and $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Equation (7.27) is the standard L_2-regularized loss function of a neural network with weights $\boldsymbol{\lambda}$ and dropconnect, and it is straightforward to obtain unbiased gradient estimates by automatic differentiation.

Crucially, for the interpretation of dropconnect regularization as variational inference to be valid, we also need to perform dropconnect regularization during inference,

$$\begin{aligned}
p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}) &\approx \mathbb{E}_{\boldsymbol{\theta} \sim q_{\boldsymbol{\lambda}}}[p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta})] \\
&\approx \frac{1}{m} \sum_{i=1}^{m} p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}^{(i)}) && \text{using Monte Carlo sampling} \quad (7.28)
\end{aligned}$$

where $\boldsymbol{\theta}^{(i)} \overset{\text{iid}}{\sim} q_{\boldsymbol{\lambda}}$ are independent samples. This coincides with our earlier discussion of variational inference for Bayesian neural networks in Equation (7.17). In words, we average the predictions of m neural networks for each of which we randomly "drop out" weights.

Remark 7.2: Masksembles

A practical problem of dropout is that for any reasonable choice of dropout probability, the dropout masks $z^{(i)}$ will overlap significantly, which tends to make the predictions $p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}^{(i)})$ highly correlated. This can lead to underestimation of epistemic uncertainty. *Masksembles* (Durasov et al., 2021) mitigate this issue by choosing a fixed set of pre-defined dropout masks, which have controlled overlap, and alternating between them during training. In the extreme case of "infinitely many" masks, masksembles are equivalent to dropout since each mask is only seen once.

7.3.4 Probabilistic Ensembles

We have seen that variational inference in the context of Bayesian neural networks can be interpreted as averaging the predictions of m neural networks drawn according to the variational posterior.

A natural adaptation of this idea is to immediately learn the weights of m neural networks. The idea is to randomly choose m training sets by sampling uniformly from the data with replacement. Then, using our analysis from Section 7.2.1, we obtain m MAP estimates of the weights $\boldsymbol{\theta}^{(i)}$, yielding the approximation

$$\begin{aligned} p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}) &= \mathbb{E}_{\boldsymbol{\theta} \sim p(\cdot \mid \boldsymbol{x}_{1:n}, y_{1:n})}[p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta})] \\ &\approx \frac{1}{m} \sum_{i=1}^{m} p(y^\star \mid \boldsymbol{x}^\star, \boldsymbol{\theta}^{(i)}). \end{aligned} \tag{7.29}$$

using bootstrapping

Here, the randomly chosen m training sets induce "diversity" of the models. In practice, in the context of deep neural networks where the global minimizer of the loss can rarely be identified, it is common to use the full training data to train each of the m neural networks. Random initialization and random shuffling of the training data is typically enough to ensure some degree of diversity between the individual models (Lakshminarayanan et al., 2017). We can connect ensembles to the other approximate inference techniques we have discussed: First, ensembles can be seen as a specific kind of masksemble, where the masks are non-overlapping,[13] which mitigates the issue of correlated predictions from Remark 7.2. Second, ensembling can be combined with other approximate inference techniques such as variational inference, Laplace approximation, or SWAG to get a mixture of Gaussians as the posterior approximation.

[13] That is, the m models do not share any of their parameters. Ensembles and dropout lie on opposite ends of this spectrum.

Note that Equation (7.29) is not equivalent to Monte Carlo sampling, although it looks very similar. The key difference is that this approach does not sample from the true posterior distribution p, but instead

from the empirical posterior distribution $\hat{p}$ given the (re-sampled) MAP estimates. Intuitively, this can be understood as the difference between sampling from a distribution p directly (Monte Carlo sampling) versus sampling from an approximate (empirical) distribution $\hat{p}$ (corresponding to the training data), which itself is constructed from samples of the true distribution p. This approach is known as *bootstrapping* or *bagging* (short for *b*ootstrap *agg*regat*ing*) and plays a central role in model-free reinforcement learning. We will return to this concept in Section 11.4.1.

7.3.5 Diverse Probabilistic Ensembles

Probabilistic ensembles can be loosely interpreted as randomly initializing m "particles" $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^m$ and then pushing each particle towards regions of high posterior probability. A potential issue with this approach is that if the initialization of particles is not sufficiently diverse, the particles may "collapse" since every particle eventually converges to a local optimum of the loss function. A natural approach to mitigate this issue is to alter the objective of each particle from simply aiming to minimize the loss $-\log p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}, y_{1:n})$, which we will abbreviate by $-\log p(\boldsymbol{\theta})$, to also "push" the particles away from each other. We will see next that this can be interpreted as a form of variational inference under a very flexible variational family.

In our discussion of *variational inference*, we have seen that minimizing reverse-KL is equivalent to maximizing the evidence lower bound, and used this to derive an optimization algorithm to compute approximate posteriors. We will discuss the alternative approach of directly computing the gradient of the KL-divergence. Consider the variational family of all densities that can be expressed as smooth transformations of points sampled from a reference density ϕ. That is, we consider $\mathcal{Q}_\phi \doteq \{\mathbf{T}_\sharp \phi \mid \mathbf{T} : \Theta \to \Theta \text{ is smooth}\}$ where $\mathbf{T}_\sharp \phi$ is the density of the random vector $\boldsymbol{\theta}' = \mathbf{T}(\boldsymbol{\theta})$ with $\boldsymbol{\theta} \sim \phi$.[14] The density ϕ can be thought of as the initial distribution of the particles, and the smooth map $\mathbf{T}$ as the dynamics that push the particles towards the target density p. It can be shown that for "almost any" reference density ϕ, this variational family $\mathcal{Q}_\phi$ is expressive enough to closely approximate "almost arbitrary" distributions.[15] A natural approach is therefore to *learn* the appropriate smooth map $\mathbf{T}$ between the reference density ϕ and the target density p.

[14] Refer back to Section 1.1.11 for a recap on pushforwards.

[15] For a more detailed discussion, refer to "Stein variational gradient descent: A general purpose Bayesian inference algorithm" (Liu and Wang, 2016).

Example 7.3: Gaussian variational inference

We have seen in Section 5.5 that if ϕ is standard normal and $\mathbf{T}(\boldsymbol{x}; \{\boldsymbol{\mu}, \boldsymbol{\Sigma}^{1/2}\}) = \boldsymbol{\mu} + \boldsymbol{\Sigma}^{1/2}\boldsymbol{x}$ an affine transformation then the ELBO

can be maximized efficiently using stochastic gradient descent. However, in this case, $\mathcal{Q}_\phi$ can only approximate Gaussian-like distributions since the expressivity of the map $\mathbf{T}$ is limited under the fixed reference density ϕ.

An alternative approach to Gaussian variational inference is the following algorithm known as *Stein variational gradient descent* (SVGD), where we recursively apply carefully chosen smooth maps to the current variational approximation:

$$q_0 \xrightarrow{\mathbf{T}_0^\star} q_1 \xrightarrow{\mathbf{T}_1^\star} q_2 \xrightarrow{\mathbf{T}_2^\star} \cdots \qquad \text{where } q_{t+1} \doteq \mathbf{T}_{t\,\sharp}^\star q_t. \tag{7.30}$$

We consider maps $\mathbf{T} = \mathbf{id} + \boldsymbol{f}$ where $\mathbf{id}(\boldsymbol{\theta}) \doteq \boldsymbol{\theta}$ denotes the identity map and $\boldsymbol{f}(\boldsymbol{x})$ represents a (small) perturbation. Recall that at time t we seek to minimize $\mathrm{KL}(\mathbf{T}_\sharp q_t \| p)$, so we choose the smooth map as

$$\mathbf{T}_t^\star \doteq \mathbf{id} - \eta_t \, \boldsymbol{\nabla}_{\boldsymbol{f}} \, \mathrm{KL}(\mathbf{T}_\sharp q_t \| p)\big|_{\boldsymbol{f}=\mathbf{0}} \tag{7.31}$$

where η_t is a step size. In this way, the SVGD update (7.31) can be interpreted as a step of "functional" gradient descent.

To be able to compute the gradient of the KL-divergence, we need to make some structural assumptions on the perturbation $\boldsymbol{f}$. SVGD assumes that $\boldsymbol{f} = [f_1 \ \cdots \ f_d]^\top$ with $f_i \in \mathcal{H}_k(\Theta)$ is from some reproducing kernel Hilbert space $\mathcal{H}_k(\Theta)$ of a positive definite kernel k; we say $\boldsymbol{f} \in \mathcal{H}_k^d(\Theta)$. Within the RKHS, we can compute the gradient of the KL-divergence exactly. Liu and Wang (2016) show that in this case, the functional gradient of the KL-divergence can be expressed in closed-form as $-\boldsymbol{\nabla}_{\boldsymbol{f}} \, \mathrm{KL}(\mathbf{T}_\sharp q \| p)\big|_{\boldsymbol{f}=\mathbf{0}} = \boldsymbol{\varphi}_{q,p}^\star$ where

$$\boldsymbol{\varphi}_{q,p}^\star(\cdot) \doteq \mathbb{E}_{\boldsymbol{\theta}\sim q}[k(\cdot, \boldsymbol{\theta}) \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}) + \boldsymbol{\nabla}_{\boldsymbol{\theta}} \, k(\cdot, \boldsymbol{\theta})]. \tag{7.32}$$

SVGD then approximates q using the particles $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^m$ as follows:

Algorithm 7.4: Stein variational gradient descent, SVGD

initialize particles $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^m$
repeat
  for each particle $i \in [m]$ **do**
    $\boldsymbol{\theta}^{(i)} \leftarrow \boldsymbol{\theta}^{(i)} + \eta_t \hat{\boldsymbol{\varphi}}_{q,p}^\star(\boldsymbol{\theta}^{(i)})$ where
    $\hat{\boldsymbol{\varphi}}_{q,p}^\star(\boldsymbol{\theta}) \doteq \frac{1}{m} \sum_{j=1}^m \left[k(\boldsymbol{\theta}, \boldsymbol{\theta}^{(j)}) \boldsymbol{\nabla}_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}) + \boldsymbol{\nabla}_{\boldsymbol{\theta}^{(j)}} \, k(\boldsymbol{\theta}, \boldsymbol{\theta}^{(j)}) \right]$
until converged

Often, a Gaussian kernel (4.14) with length scale h is used to model the perturbations, in which case the repulsion term is

$$\nabla_{\boldsymbol{\theta}^{(j)}} k(\boldsymbol{\theta}, \boldsymbol{\theta}^{(j)}) = \frac{1}{h^2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(j)}) k(\boldsymbol{\theta}, \boldsymbol{\theta}^{(j)}) \tag{7.33}$$

and the negative functional gradient simplifies to

$$\hat{\boldsymbol{\varphi}}^{\star}_{q,p}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{j=1}^{m} k(\boldsymbol{\theta}, \boldsymbol{\theta}^{(j)}) \Big[\underbrace{\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta})}_{\text{drift}} + \underbrace{h^{-2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(j)})}_{\text{repulsion}} \Big]. \tag{7.34}$$

Note that SVGD has similarities to Langevin dynamics, which as seen in (?) can also be interpreted as following a gradient of the KL-divergence. Whereas Langevin dynamics perturbs particles according to a drift towards regions of high posterior probability and some random diffusion (cf. Equation (6.46)), the first term of $\hat{\boldsymbol{\varphi}}^{\star}_{q,p}(\boldsymbol{\theta})$ perturbs particles to drift towards regions of high posterior probability while the second term leads to a mutual "repulsion" of particles. Notably, the perturbations of Langevin dynamics are noisy, while SVGD perturbs particles deterministically and the randomness is exclusively in the initialization of particles. The repulsion term prevents particles from collapsing to a single mode of the posterior distribution, which is a possible failure mode of other particle-based posterior approximations such as ensembles.

Problem 6.9

Note that the above decomposition of $\hat{\boldsymbol{\varphi}}^{\star}_{q,p}(\boldsymbol{\theta})$ is once more an example of the *principle of curiosity and conformity* which we have seen to be a recurring theme in approaches to approximate inference. The repulsion term leads to exploration of the particles (i.e., "curiosity" about alternative explanations), while the drift term leads to minimization of the loss (i.e., "conformity" to the data).

Lu et al. (2019) show that under some assumptions, SVGD converges asymptotically to the target density p as $\eta_t \to 0$. SVGD's name originates from *Stein's method* which is a general-purpose approach for characterizing convergence in distribution.[16]

[16] For an introduction to Stein's method, read "Measuring sample quality with Stein's method" (Gorham and Mackey, 2015).

7.4 Calibration

A key challenge of Bayesian deep learning (and also other probabilistic methods) is the calibration of models. We say that a model is *well-calibrated* if its confidence coincides with its accuracy across many predictions. Consider a classification model that predicts that the label of a given input belongs to some class with probability 80%. If the model is well-calibrated, then the prediction is correct about 80% of the time. In other words, during calibration, we adjust the probability estimation of the model.

We will first mention two methods of estimating the calibration of a model, namely the marginal likelihood and reliability diagrams. Then, in Section 7.4.3, we survey commonly used heuristics for empirically improving the calibration.

7.4.1 Evidence

A popular method (which we already encountered multiple times) is to use the evidence of a validation set $\boldsymbol{x}_{1:m}^{\text{val}}$ of size m given the training set $\boldsymbol{x}_{1:n}^{\text{train}}$ of size n for estimating the model calibration. Here, the evidence can be understood as describing how well the validation set is described by the model trained on the training set. We obtain,

$$
\begin{aligned}
&\log p(y_{1:m}^{\text{val}} \mid \boldsymbol{x}_{1:m}^{\text{val}}, \boldsymbol{x}_{1:n}^{\text{train}}, y_{1:n}^{\text{train}}) \\
&= \log \int p(y_{1:m}^{\text{val}} \mid \boldsymbol{x}_{1:m}^{\text{val}}, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:n}^{\text{train}}, y_{1:n}^{\text{train}}) \, d\boldsymbol{\theta} && \text{using the sum rule (1.7) and product rule (1.11)} \\
&\approx \log \int p(y_{1:m}^{\text{val}} \mid \boldsymbol{x}_{1:m}^{\text{val}}, \boldsymbol{\theta}) q_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) \, d\boldsymbol{\theta} && \text{approximating with the variational posterior} \\
&= \log \int \prod_{i=1}^{m} p(y_i^{\text{val}} \mid \boldsymbol{x}_i^{\text{val}}, \boldsymbol{\theta}) q_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) \, d\boldsymbol{\theta} && \text{using the independence of the data} \qquad (7.35)
\end{aligned}
$$

The resulting integrals are typically very small which leads to numerical instabilities. Therefore, it is common to maximize a lower bound to the evidence instead,

$$
\begin{aligned}
&= \log \mathbb{E}_{\boldsymbol{\theta} \sim q_{\boldsymbol{\lambda}}} \left[\prod_{i=1}^{m} p(y_i^{\text{val}} \mid \boldsymbol{x}_i^{\text{val}}, \boldsymbol{\theta}) \right] && \text{interpreting the integral as an expectation over the variational posterior} \\
&\geq \mathbb{E}_{\boldsymbol{\theta} \sim q_{\boldsymbol{\lambda}}} \left[\sum_{i=1}^{m} \log p(y_i^{\text{val}} \mid \boldsymbol{x}_i^{\text{val}}, \boldsymbol{\theta}) \right] \qquad (7.36) && \text{using Jensen's inequality (5.29)} \\
&\approx \frac{1}{k} \sum_{j=1}^{k} \sum_{i=1}^{m} \log p(y_i^{\text{val}} \mid \boldsymbol{x}_i^{\text{val}}, \boldsymbol{\theta}^{(j)}) \qquad (7.37) && \text{using Monte Carlo sampling}
\end{aligned}
$$

for independent samples $\boldsymbol{\theta}^{(j)} \overset{\text{iid}}{\sim} q_{\boldsymbol{\lambda}}$.

7.4.2 Reliability Diagrams

Reliability diagrams take a frequentist perspective to estimate the calibration of a model. For simplicity, we assume a calibration problem with two classes, 1 and -1 (similarly to logistic regression).[17]

[17] Reliability diagrams generalize beyond this restricted example.

We group the predictions of a validation set into M interval bins of size $1/M$ according to the class probability predicted by the model, $\mathbb{P}(Y_i = 1 \mid \boldsymbol{x}_i)$. We then compare within each bin, how often the model thought the inputs belonged to the class (confidence) with how often the inputs actually belonged to the class (frequency). Formally, we

define B_m as the set of samples falling into bin m and let

$$\text{freq}(B_m) \doteq \frac{1}{|B_m|} \sum_{i \in B_m} \mathbb{1}\{Y_i = 1\} \tag{7.38}$$

be the proportion of samples in bin m that belong to class 1 and let

$$\text{conf}(B_m) \doteq \frac{1}{|B_m|} \sum_{i \in B_m} \mathbb{P}(Y_i = 1 \mid \boldsymbol{x}_i) \tag{7.39}$$

be the average confidence of samples belonging to class 1 within the bin m.

Thus, a model is well calibrated if $\text{freq}(B_m) \approx \text{conf}(B_m)$ for each bin $m \in [M]$. There are two common metrics of calibration that quantify how "close" a model is to being well calibrated.

1. The *expected calibration error* (ECE) is the average deviation of a model from perfect calibration,

$$\ell_{\text{ECE}} \doteq \sum_{m=1}^{M} \frac{|B_m|}{n} |\text{freq}(B_m) - \text{conf}(B_m)| \tag{7.40}$$

   where n is the size of the validation set.
2. The *maximum calibration error* (MCE) is the maximum deviation of a model from perfect calibration among all bins,

$$\ell_{\text{MCE}} \doteq \max_{m \in [M]} |\text{freq}(B_m) - \text{conf}(B_m)| \,. \tag{7.41}$$

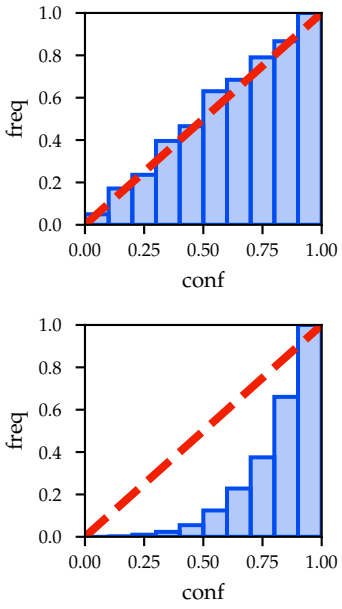


Figure 7.8: Examples of reliability diagrams with ten bins. A perfectly calibrated model approximates the diagonal dashed red line. The first reliability diagram shows a well-calibrated model. In contrast, the second reliability diagram shows an overconfident model.

7.4.3 *Heuristics for Improving Calibration*

We now survey a few heuristics which can be used empirically to improve model calibration.

1. *Histogram binning* assigns a calibrated score $q_m \doteq \text{freq}(B_m)$ to each bin during validation. Then, during inference, we return the calibrated score q_m of the bin corresponding to the prediction of the model.
2. *Isotonic regression* extends histogram binning by using variable bin boundaries. We find a piecewise constant function $\boldsymbol{f} \doteq [f_1, \dots, f_M]$ that minimizes the bin-wise squared loss,

$$\min_{M, \boldsymbol{f}, \boldsymbol{a}} \quad \sum_{m=1}^{M} \sum_{i=1}^{n} \mathbb{1}\{a_m \leq \mathbb{P}(Y_i = 1 \mid \boldsymbol{x}_i) < a_{m+1}\}(f_m - y_i)^2 \tag{7.42a}$$

$$\text{subject to} \quad 0 = a_1 \leq \cdots \leq a_{M+1} = 1, \tag{7.42b}$$

$$f_1 \leq \cdots \leq f_M \tag{7.42c}$$

   where $\boldsymbol{f}$ are the calibrated scores and $\boldsymbol{a} \doteq [a_1, \dots, a_{M+1}]$ are the bin boundaries. We then return the calibrated score f_m of the bin corresponding to the prediction of the model.

3. *Platt scaling* adjusts the logits z_i of the output layer to

$$q_i \doteq \sigma(az_i + b) \tag{7.43}$$

and then learns parameters $a, b \in \mathbb{R}$ to maximize the likelihood.

4. *Temperature scaling* is a special and widely used instance of Platt scaling where $a \doteq 1/T$ and $b \doteq 0$ for some temperature scalar $T > 0$,

$$q_i \doteq \sigma\left(\frac{z_i}{T}\right). \tag{7.44}$$

Intuitively, for a larger temperature T, the probability is distributed more evenly among the classes (without changing the ranking), yielding a more uncertain prediction. In contrast, for a lower temperature T, the probability is concentrated more towards the top choices, yielding a less uncertain prediction. As seen in Problem 6.7, temperature scaling can be motivated as tuning the mean of the softmax distribution.

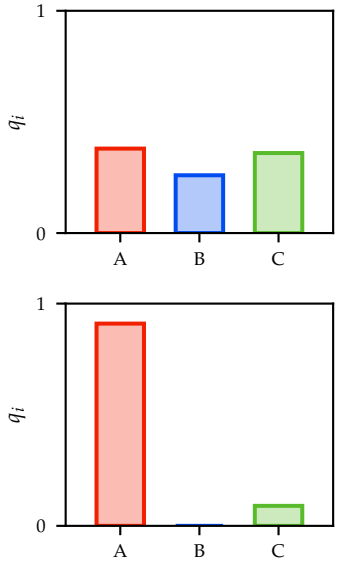


Figure 7.9: Illustration of temperature scaling for a classifier with three classes. On the top, we have a prediction with a high temperature, yielding a very uncertain prediction (in favor of class A). Below, we have a prediction with a low temperature, yielding a prediction that is strongly in favor of class A. Note that the ranking ($A \succ C \succ B$) is preserved.

Optional Readings

- Guo, Pleiss, Sun, and Weinberger (2017). *On calibration of modern neural networks*.
- Blundell, Cornebise, Kavukcuoglu, and Wierstra (2015). *Weight uncertainty in neural network*.
- Kendall and Gal (2017). *What uncertainties do we need in Bayesian deep learning for computer vision?*.

Discussion

This chapter concludes our discussion of (approximate) probabilistic inference. Across the last three chapters, we have seen numerous methods for approximating the posterior distributions of deep neural networks:

- Methods such as dropout and stochastic weight averaging are frequently used in practice. Other particle-based approaches such as ensembles and SVGD are used less frequently since they are computationally more expensive to train, but are some of the most effective methods in estimating uncertainty.
- Recently, Laplace approximations regained interest since they can be applied "post-hoc" after training simply by computing or approximating the Hessian of the loss function (Daxberger et al., 2021; Antorán et al., 2022). Still, Laplace approximations come with the limitations inherent to unimodal Gaussian approximations.

- Other work, particularly in fine-tuning, has explored approximating the posterior distribution of deep neural networks by treating them as linear functions in a fixed learned feature space, in which case one can use the tools for exact probabilistic inference from Chapters 2 and 4 (e.g., Hübotter et al., 2025).

Despite large progress in approximate inference over the past decade, efficient and reliable uncertainty estimation of large models remains an important open challenge.

Problems

7.1. Softmax is a generalization of the logistic function.

Show that for a two-class classification problem (i.e., $c = 2$), the softmax function is equivalent to the logistic function (5.9) for the univariate model $f \doteq f_1 - f_0$. That is, $\sigma_1(\boldsymbol{f}) = \sigma(f)$ and $\sigma_0(\boldsymbol{f}) = 1 - \sigma(f)$.

Thus, the softmax function is a generalization of the logistic function to more than two classes.

PART II

Sequential Decision-Making

Preface to Part II

In the first part of the manuscript, we have learned about how we can build machines that are capable of updating their beliefs and reducing their epistemic uncertainty through probabilistic inference. We have also discussed ways of keeping track of the world through noisy sensory information by filtering. An important aspect of intelligence is to use this acquired knowledge for making decisions and taking actions that have a positive impact on the world.

Already today, we are surrounded by machines that make decisions and take actions; that is, exhibit some degree of agency. Be it a search engine producing a list of search results, a chatbot answering a question, or a driving-assistance system steering a car: these systems are all perceiving the world, making decisions, and then taking actions that in turn have an effect on the world. Figure 7.10 illustrates this *perception-action loop*.

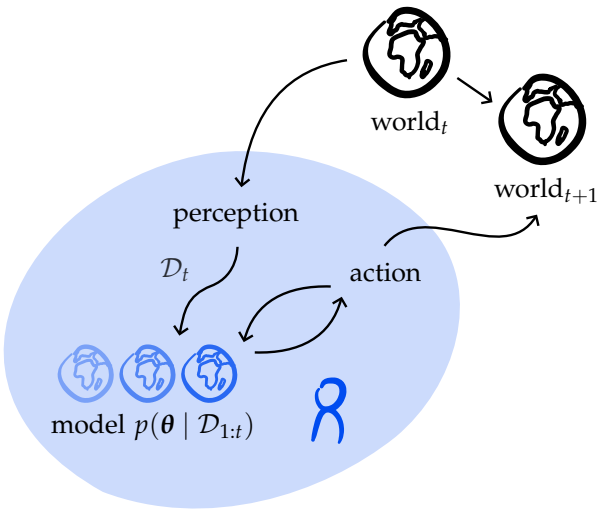


Figure 7.10: An illustration of the perception-action loop. This is a straightforward extension of our view of probabilistic inference from Figure 1.10 with the addition of an "action" component which is capable of "adaptively" interacting with the outside world and the internal world model.

In the second part of this manuscript, we will discuss the underpinning principles of building machines that are capable of making se-

quential decisions. We will see that decision-making itself can be cast as probabilistic inference, obeying the same mechanisms that we used in the first part to build learning systems.

We discuss various ways of addressing the question:

How to act, given that computational resources and time are limited?

One approach is to act with the aim to reduce epistemic uncertainty, which is the topic of active learning. Another approach is to act with the aim to maximize some reward signal, which is the topic of bandits, Bayesian optimization, and reinforcement learning.

This surfaces the exploration-exploitation dilemma where the agent has to prioritize either maximizing its immediate rewards or reducing its uncertainty about the world which might pay off in the future. We discuss that this dilemma is, in fact, in direct correspondence to the principle of curiosity and conformity which we discussed extensively throughout Part I.

Since time is limited, it is critical to be sample-efficient when learning the most important aspects of the world. At the same time, interactions with the world are often complex, and some interactions might even be harmful. We discuss how an agent can use its epistemic uncertainty to guide the exploration of its environment while mitigating risks and reasoning about safety.

8

Active Learning

By now, we have seen that probabilistic machine learning is very useful for estimating the uncertainty in our models (epistemic uncertainty) and in the data (aleatoric uncertainty). We have been focusing on the setting of supervised learning where we are given a set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ of labeled data, yet we often encounter settings where we have only little data and acquiring new data is costly.

In this chapter — and in the following chapter on Bayesian optimization — we will discuss how one can use uncertainty to effectively collect more data. In other words, we want to figure out where in the domain we should sample to obtain the most useful information. Throughout most of this chapter, we focus on the most common way of quantifying "useful information", namely the expected reduction in entropy which is also called the *mutual information*.

8.1 Conditional Entropy

We begin by introducing the notion of conditional entropy. Recall that the entropy $\mathrm{H}[\mathbf{X}]$ of a random vector $\mathbf{X}$ can be interpreted as our average surprise when observing realizations $\boldsymbol{x} \sim \mathbf{X}$. Thus, entropy can be considered as a quantification of the uncertainty about a random vector (or equivalently, its distribution).[1]

[1] We discussed entropy extensively in Section 5.4.

A natural extension is to consider the entropy of $\mathbf{X}$ given the occurrence of an event corresponding to another random variable (e.g., $\mathbf{Y} = \boldsymbol{y}$ for a random vector $\mathbf{Y}$),

$$\mathrm{H}[\mathbf{X} \mid \mathbf{Y} = \boldsymbol{y}] \doteq \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x} \mid \boldsymbol{y})}[-\log p(\boldsymbol{x} \mid \boldsymbol{y})]. \tag{8.1}$$

Instead of averaging over the surprise of samples from the distribution $p(\boldsymbol{x})$ (like the entropy $\mathrm{H}[\mathbf{X}]$), this quantity simply averages over the surprise of samples from the conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{y})$.

Definition 8.1 (Conditional entropy)**.** The *conditional entropy* of a random vector $\mathbf{X}$ given the random vector $\mathbf{Y}$ is defined as

$$\begin{aligned} \mathrm{H}[\mathbf{X} \mid \mathbf{Y}] &\doteq \mathbb{E}_{\boldsymbol{y} \sim p(\boldsymbol{y})}[\mathrm{H}[\mathbf{X} \mid \mathbf{Y} = \boldsymbol{y}]] && (8.2) \\ &= \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y}) \sim p(\boldsymbol{x},\boldsymbol{y})}[-\log p(\boldsymbol{x} \mid \boldsymbol{y})]. && (8.3) \end{aligned}$$

Intuitively, the conditional entropy of $\mathbf{X}$ given $\mathbf{Y}$ describes our average surprise about realizations of $\mathbf{X}$ given a particular realization of $\mathbf{Y}$, averaged over all such possible realizations of $\mathbf{Y}$. In other words, conditional entropy corresponds to the expected remaining uncertainty in $\mathbf{X}$ after we observe $\mathbf{Y}$. Note that, in general, $\mathrm{H}[\mathbf{X} \mid \mathbf{Y}] \neq \mathrm{H}[\mathbf{Y} \mid \mathbf{X}]$.

It is crucial to stress the difference between $\mathrm{H}[\mathbf{X} \mid \mathbf{Y} = \boldsymbol{y}]$ and the conditional entropy $\mathrm{H}[\mathbf{X} \mid \mathbf{Y}]$. The former simply corresponds to a probabilistic update of our uncertainty in $\mathbf{X}$ after we have observed the realization $\boldsymbol{y} \sim \mathbf{Y}$. In contrast, conditional entropy *predicts* how much uncertainty will remain about $\mathbf{X}$ (in expectation) after we *will* observe $\mathbf{Y}$.

Definition 8.2 (Joint entropy)**.** One can also define the *joint entropy* of random vectors $\mathbf{X}$ and $\mathbf{Y}$,

$$\mathrm{H}[\mathbf{X}, \mathbf{Y}] \doteq \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y}) \sim p(\boldsymbol{x},\boldsymbol{y})}[-\log p(\boldsymbol{x}, \boldsymbol{y})], \tag{8.4}$$

as the combined uncertainty about $\mathbf{X}$ and $\mathbf{Y}$. Observe that joint entropy is symmetric.

This gives the *chain rule for entropy*,

$$\begin{aligned} \mathrm{H}[\mathbf{X}, \mathbf{Y}] &= \mathrm{H}[\mathbf{Y}] + \mathrm{H}[\mathbf{X} \mid \mathbf{Y}] && (8.5) \\ &= \mathrm{H}[\mathbf{X}] + \mathrm{H}[\mathbf{Y} \mid \mathbf{X}]. && (8.6) \end{aligned}$$

(8.5): using the product rule (1.11) and the definition of conditional entropy (8.2)

(8.6): using symmetry of joint entropy

That is, the joint entropy of $\mathbf{X}$ and $\mathbf{Y}$ is given by the uncertainty about $\mathbf{X}$ and the additional uncertainty about $\mathbf{Y}$ given $\mathbf{X}$. Moreover, this also yields *Bayes' rule for entropy*,

$$\mathrm{H}[\mathbf{X} \mid \mathbf{Y}] = \mathrm{H}[\mathbf{Y} \mid \mathbf{X}] + \mathrm{H}[\mathbf{X}] - \mathrm{H}[\mathbf{Y}]. \tag{8.7}$$

using the chain rule for entropy (8.5) twice

A very intuitive property of entropy is its monotonicity: when conditioning on additional observations the entropy can never increase,

$$\mathrm{H}[\mathbf{X} \mid \mathbf{Y}] \leq \mathrm{H}[\mathbf{X}]. \tag{8.8}$$

Colloquially, this property is also called the "*information never hurts*" principle. We will derive a proof in the following section.

8.2 Mutual Information

Recall that our fundamental objective is to reduce entropy, as this corresponds to reduced uncertainty in the variables, which we want to predict. Thus, we are interested in how much information we "gain" about the random vector $\mathbf{X}$ by choosing to observe a random vector $\mathbf{Y}$. By our interpretation of conditional entropy from the previous section, this is described by the following quantity.

Definition 8.3 (Mutual information, MI)**.** The *mutual information* of $\mathbf{X}$ and $\mathbf{Y}$ (also known as the *information gain*) is defined as

$$\begin{aligned} \mathrm{I}(\mathbf{X};\mathbf{Y}) &\doteq \mathrm{H}[\mathbf{X}] - \mathrm{H}[\mathbf{X} \mid \mathbf{Y}] && (8.9)\\ &= \mathrm{H}[\mathbf{X}] + \mathrm{H}[\mathbf{Y}] - \mathrm{H}[\mathbf{X},\mathbf{Y}]. && (8.10) \end{aligned}$$

In words, we subtract the uncertainty left about $\mathbf{X}$ after observing $\mathbf{Y}$ from our initial uncertainty about $\mathbf{X}$. This measures the reduction in our uncertainty in $\mathbf{X}$ (as measured by entropy) upon observing $\mathbf{Y}$. Unlike conditional entropy, it follows from the definition that mutual information is symmetric. That is,

$$\mathrm{I}(\mathbf{X};\mathbf{Y}) = \mathrm{I}(\mathbf{Y};\mathbf{X}). \tag{8.11}$$

Thus, the mutual information between $\mathbf{X}$ and $\mathbf{Y}$ can be understood as the approximation error (or information loss) when assuming that $\mathbf{X}$ and $\mathbf{Y}$ are independent.

In particular, using Gibbs' inequality (cf. Problem 5.5), this relationship shows that $\mathrm{I}(\mathbf{X};\mathbf{Y}) \geq 0$ with equality when $\mathbf{X}$ and $\mathbf{Y}$ are independent, and also proves the *information never hurts* principle (8.8) as

$$0 \leq \mathrm{I}(\mathbf{X};\mathbf{Y}) = \mathrm{H}[\mathbf{X}] - \mathrm{H}[\mathbf{X} \mid \mathbf{Y}]. \tag{8.12}$$

Example 8.4: Mutual information of Gaussians

Given the Gaussian random vector $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and the noisy observation $\mathbf{Y} = \mathbf{X} + \boldsymbol{\varepsilon}$ where $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_\mathrm{n}^2 \boldsymbol{I})$, we want to find the information gain of $\mathbf{X}$ when observing $\mathbf{Y}$. Using our definitions from this chapter, we obtain

$$\begin{aligned} \mathrm{I}(\mathbf{X};\mathbf{Y}) &= \mathrm{I}(\mathbf{Y};\mathbf{X}) && \text{using symmetry (8.11)}\\ &= \mathrm{H}[\mathbf{Y}] - \mathrm{H}[\mathbf{Y} \mid \mathbf{X}] && \text{by mutual information (8.9)}\\ &= \mathrm{H}[\mathbf{Y}] - \mathrm{H}[\boldsymbol{\varepsilon}] && \text{given } \mathbf{X}\text{, the only randomness in } \mathbf{Y} \text{ originates from } \boldsymbol{\varepsilon}\\ &= \frac{1}{2}\log\Big((2\pi e)^d \det\Big(\boldsymbol{\Sigma} + \sigma_\mathrm{n}^2 \boldsymbol{I}\Big)\Big) - \frac{1}{2}\log\Big((2\pi e)^d \det\Big(\sigma_\mathrm{n}^2 \boldsymbol{I}\Big)\Big) && \text{using the entropy of Gaussians (5.31)} \end{aligned}$$

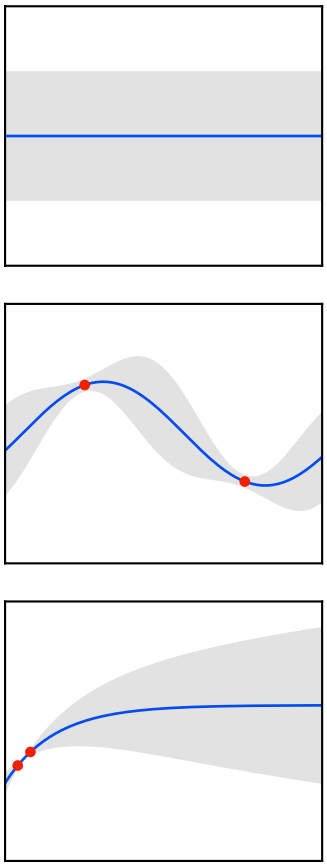


Figure 8.1: Information gain. The first graph shows the prior. The second graph shows a selection of samples with large information gain (large uncertainty reduction). The third graph shows a selection of samples with small information gain (small uncertainty reduction).

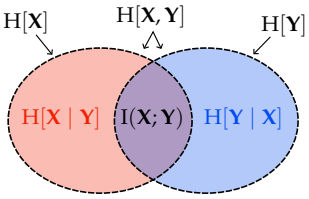


Figure 8.2: Relationship between mutual information and entropy, expressed as a Venn diagram.

$$= \frac{1}{2} \log \frac{\det\left(\boldsymbol{\Sigma} + \sigma_{\mathrm{n}}^2 \boldsymbol{I}\right)}{\det(\sigma_{\mathrm{n}}^2 \boldsymbol{I})}$$
$$= \frac{1}{2} \log \det\left(\boldsymbol{I} + \sigma_{\mathrm{n}}^{-2} \boldsymbol{\Sigma}\right). \tag{8.13}$$

Intuitively, the larger the noise σ_{n}^2 in relation to the covariance of $\mathbf{X}$, the smaller the information gain.

8.2.1 Synergy and Redundancy

It is sometimes useful to write down the mutual information of $\mathbf{X}$ and $\mathbf{Y}$ conditioned (in expectation) on a third random vector $\mathbf{Z}$. This leads us to the following definition.

Definition 8.5 (Conditional mutual information)**.** The *conditional mutual information* of $\mathbf{X}$ and $\mathbf{Y}$ given $\mathbf{Z}$ is defined as

$$\mathrm{I}(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z}) \doteq \mathrm{H}[\mathbf{X} \mid \mathbf{Z}] - \mathrm{H}[\mathbf{X} \mid \mathbf{Y}, \mathbf{Z}]. \tag{8.14}$$
$$= \mathrm{H}[\mathbf{X}, \mathbf{Z}] + \mathrm{H}[\mathbf{Y}, \mathbf{Z}] - \mathrm{H}[\mathbf{Z}] - \mathrm{H}[\mathbf{X}, \mathbf{Y}, \mathbf{Z}] \tag{8.15}$$
using the relationship of joint and conditional entropy (8.5)
$$= \mathrm{I}(\mathbf{X}; \mathbf{Y}, \mathbf{Z}) - \mathrm{I}(\mathbf{X}; \mathbf{Z}). \tag{8.16}$$

Thus, the conditional mutual information corresponds to the reduction of uncertainty in $\mathbf{X}$ when observing $\mathbf{Y}$, given we already observed $\mathbf{Z}$. It also follows that conditional mutual information is symmetric:

$$\mathrm{I}(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z}) = \mathrm{I}(\mathbf{Y}; \mathbf{X} \mid \mathbf{Z}). \tag{8.17}$$

We have seen in this chapter that entropy is monotonically decreasing as we condition on new information, and called this the "information never hurts" principle (8.8). However, the same does not hold for mutual information! That is, information about a random vector $\mathbf{Z}$ may reduce the mutual information between random vectors $\mathbf{X}$ and $\mathbf{Y}$ (?). **Problem 8.2**

Remark 8.6: Sufficient statistics and data processing inequality
A related concept is the data processing inequality (8.42) which you prove in (?) and which allows us to formalize a concept which we have seen multiple times already, namely the notion of a sufficient statistic. Consider the Markov chain $\boldsymbol{\lambda} \to \mathbf{X} \to \boldsymbol{s}(\mathbf{X})$, for example, $\boldsymbol{\lambda}$ may be parameters of the distribution of $\mathbf{X}$. By the data processing inequality (8.42), $\mathrm{I}(\boldsymbol{\lambda}; \boldsymbol{s}(\mathbf{X})) \leq \mathrm{I}(\boldsymbol{\lambda}; \mathbf{X})$. If the data processing inequality is satisfied with equality then $\boldsymbol{s}(\mathbf{X})$ is called a *sufficient statistic* of $\mathbf{X}$ for the inference of $\boldsymbol{\lambda}$. **Problem 8.2 (3)**

To understand the behavior of mutual information under conditioning, it is helpful to consider the *interaction information*

$$\mathrm{I}(\mathbf{X};\mathbf{Y};\mathbf{Z}) \doteq \mathrm{I}(\mathbf{X};\mathbf{Y}) - \mathrm{I}(\mathbf{X};\mathbf{Y} \mid \mathbf{Z}). \tag{8.18}$$

If the interaction is positive then some information about $\mathbf{X}$ that is provided by $\mathbf{Y}$ is also provided by $\mathbf{Z}$ (i.e., conditioning on $\mathbf{Z}$ decreases MI between $\mathbf{X}$ and $\mathbf{Y}$), and we say that there is *redundancy* between $\mathbf{Y}$ and $\mathbf{Z}$ (with respect to $\mathbf{X}$). Conversely, if the interaction is negative then learning about $\mathbf{Z}$ increases what can be learned from $\mathbf{Y}$ about $\mathbf{X}$, and we say that there is *synergy* between $\mathbf{Y}$ and $\mathbf{Z}$. We will see later in this chapter that the absence of synergies can lead to efficient algorithms for maximizing mutual information.

8.2.2 *Mutual Information as Utility Function*

Following our introduction of mutual information, it is natural to answer the question "where should I collect data?" by saying "wherever mutual information is maximized". More concretely, assume we are given a set $\mathcal{X}$ of possible observations of f, where $y_{\boldsymbol{x}}$ denotes a single such observation at $\boldsymbol{x} \in \mathcal{X}$,

$$y_{\boldsymbol{x}} \doteq f_{\boldsymbol{x}} + \varepsilon_{\boldsymbol{x}}, \tag{8.19}$$

$f_{\boldsymbol{x}} \doteq f(\boldsymbol{x})$, and $\varepsilon_{\boldsymbol{x}}$ is some zero-mean Gaussian noise. For a set of observations $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, we can write $\boldsymbol{y}_S = \boldsymbol{f}_S + \boldsymbol{\varepsilon}$ where

$$\boldsymbol{y}_S \doteq \begin{bmatrix} y_{\boldsymbol{x}_1} \\ \vdots \\ y_{\boldsymbol{x}_n} \end{bmatrix}, \quad \boldsymbol{f}_S \doteq \begin{bmatrix} f_{\boldsymbol{x}_1} \\ \vdots \\ f_{\boldsymbol{x}_n} \end{bmatrix}, \quad \text{and} \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_{\mathrm{n}}^2 \boldsymbol{I}).$$

Note that both $\boldsymbol{y}_S$ and $\boldsymbol{f}_S$ are random vectors. Our goal is then to find a subset $S \subseteq \mathcal{X}$ of size n maximizing the information gain between our model f and $\boldsymbol{y}_S$.

This yields the maximization objective,

$$I(S) \doteq \mathrm{I}(\boldsymbol{f}_S; \boldsymbol{y}_S) = \mathrm{H}[\boldsymbol{f}_S] - \mathrm{H}[\boldsymbol{f}_S \mid \boldsymbol{y}_S]. \tag{8.20}$$

Here, $\mathrm{H}[\boldsymbol{f}_S]$ corresponds to the uncertainty about $\boldsymbol{f}_S$ before obtaining the observations $\boldsymbol{y}_S$ and $\mathrm{H}[\boldsymbol{f}_S \mid \boldsymbol{y}_S]$ corresponds to the uncertainty about $\boldsymbol{f}_S$, in expectation, after obtaining the observations $\boldsymbol{y}_S$.

Remark 8.7: Making optimal decisions with intrinsic rewards

Note that this objective function maps neatly onto our initial consideration of making optimal decisions under uncertainty from

Section 1.4. In fact, you can think of maximizing $I(S)$ simply as computing the optimal decision rule for the utility

$$r(\boldsymbol{y}_S, S) = \mathrm{H}[f_S] - \mathrm{H}[f_S \mid \mathbf{Y}_S = \boldsymbol{y}_S], \tag{8.21}$$

with $\mathbf{Y}_S = \boldsymbol{y}_S$ denoting an event

with $I(S) = \mathbb{E}_{\boldsymbol{y}_S}[r(\boldsymbol{y}_S, S)]$ measuring the expected utility of observations $\boldsymbol{y}_S$. Such a utility or reward function is often called an *intrinsic reward* since it does not measure an "objective" external quantity, but instead a "subjective" quantity that is internal to the model of f.

Observe that picking a subset of points $S \subseteq \mathcal{X}$ from the domain $\mathcal{X}$ is a combinatorial problem. That is to say, we are optimizing a function over discrete sets. In general, such combinatorial optimization problems tend to be very difficult. It can be shown that maximizing mutual information is $\mathcal{NP}$-hard.

8.3 Submodularity of Mutual Information

We will look at optimizing mutual information in the following section. First, we want to introduce the notion of submodularity which is important in the analysis of discrete functions.

Definition 8.8 (Marginal gain)**.** Given a (discrete) function $F : \mathcal{P}(\mathcal{X}) \to \mathbb{R}$, the *marginal gain* of $\boldsymbol{x} \in \mathcal{X}$ given $A \subseteq \mathcal{X}$ is defined as

$$\Delta_F(\boldsymbol{x} \mid A) \doteq F(A \cup \{\boldsymbol{x}\}) - F(A). \tag{8.22}$$

Intuitively, the marginal gain describes how much "adding" the additional $\boldsymbol{x}$ to A increases the value of F.

When maximizing mutual information, the marginal gain is ?

Problem 8.4

$$\begin{aligned} \Delta_I(\boldsymbol{x} \mid A) &= \mathrm{I}(f_{\boldsymbol{x}}; y_{\boldsymbol{x}} \mid \boldsymbol{y}_A) && (8.23)\\ &= \mathrm{H}[y_{\boldsymbol{x}} \mid \boldsymbol{y}_A] - \mathrm{H}[\varepsilon_{\boldsymbol{x}}]. && (8.24) \end{aligned}$$

That is, when maximizing mutual information, the marginal gain corresponds to the difference between the uncertainty after observing $\boldsymbol{y}_A$ and the entropy of the noise $\mathrm{H}[\varepsilon_{\boldsymbol{x}}]$. Altogether, the marginal gain represents the reduction in uncertainty by observing $\{\boldsymbol{x}\}$.

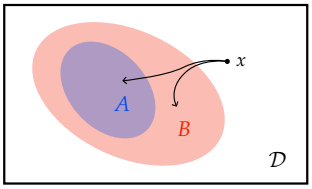


Figure 8.3: Monotone submodularity. The effect of "adding" $\boldsymbol{x}$ to the smaller set A is larger than the effect of adding $\boldsymbol{x}$ to the larger set B.

Definition 8.9 (Submodularity)**.** A (discrete) function $F : \mathcal{P}(\mathcal{X}) \to \mathbb{R}$ is *submodular* iff for any $\boldsymbol{x} \in \mathcal{X}$ and any $A \subseteq B \subseteq \mathcal{X}$ it satisfies

$$F(A \cup \{\boldsymbol{x}\}) - F(A) \geq F(B \cup \{\boldsymbol{x}\}) - F(B). \tag{8.25}$$

Equivalently, using our definition of marginal gain, we have that F is submodular iff for any $\boldsymbol{x} \in \mathcal{X}$ and any $A \subseteq B \subseteq \mathcal{X}$,

$$\Delta_F(\boldsymbol{x} \mid A) \geq \Delta_F(\boldsymbol{x} \mid B). \tag{8.26}$$

That is, "adding" $\boldsymbol{x}$ to the smaller set A yields more marginal gain than adding $\boldsymbol{x}$ to the larger set B. In other words, the function F has "diminishing returns". In this way, submodularity can be interpreted as a notion of "concavity" for discrete functions.

Definition 8.10 (Monotone submodularity)**.** A function $F : \mathcal{P}(\mathcal{X}) \to \mathbb{R}$ is called *monotone* iff for any $A \subseteq B \subseteq \mathcal{X}$ it satisfies

$$F(A) \leq F(B). \tag{8.27}$$

If F is also submodular, then F is called *monotone submodular*.

Theorem 8.11. *The objective I is monotone submodular.*

Proof. We fix arbitrary subsets $A \subseteq B \subseteq \mathcal{X}$ and any $\boldsymbol{x} \in \mathcal{X}$. We have,

$$\begin{aligned}
I \text{ is submodular} &\iff \Delta_I(\boldsymbol{x} \mid A) \geq \Delta_I(\boldsymbol{x} \mid B) && \text{by submodularity in terms of marginal gain (8.26)} \\
&\iff \mathrm{H}[y_{\boldsymbol{x}} \mid \boldsymbol{y}_A] - \mathrm{H}[\varepsilon_{\boldsymbol{x}}] \geq \mathrm{H}[y_{\boldsymbol{x}} \mid \boldsymbol{y}_B] - \mathrm{H}[\varepsilon_{\boldsymbol{x}}] && \text{using Equation (8.24)} \\
&\iff \mathrm{H}[y_{\boldsymbol{x}} \mid \boldsymbol{y}_A] \geq \mathrm{H}[y_{\boldsymbol{x}} \mid \boldsymbol{y}_B]. && \mathrm{H}[\varepsilon_{\boldsymbol{x}}] \text{ cancels}
\end{aligned}$$

Due to the "information never hurts" principle (8.8) of entropy and as $A \subseteq B$, this is always true. Moreover,

$$\begin{aligned}
I \text{ is monotone} &\iff I(A) \leq I(B) && \text{by the definition of monotinicity (8.27)} \\
&\iff \mathrm{I}(\boldsymbol{f}_A; \boldsymbol{y}_A) \leq \mathrm{I}(\boldsymbol{f}_B; \boldsymbol{y}_B) && \text{using the definition of } I \text{ (8.20)} \\
&\iff \mathrm{I}(\boldsymbol{f}_B; \boldsymbol{y}_A) \leq \mathrm{I}(\boldsymbol{f}_B; \boldsymbol{y}_B) && \text{using } \mathrm{I}(\boldsymbol{f}_B; \boldsymbol{y}_A) = \mathrm{I}(\boldsymbol{f}_A; \boldsymbol{y}_A) \text{ as } \boldsymbol{y}_A \perp \boldsymbol{f}_B \mid \boldsymbol{f}_A \\
&\iff \mathrm{H}[\boldsymbol{f}_B] - \mathrm{H}[\boldsymbol{f}_B \mid \boldsymbol{y}_A] \leq \mathrm{H}[\boldsymbol{f}_B] - \mathrm{H}[\boldsymbol{f}_B \mid \boldsymbol{y}_B] && \text{using the definition of MI (8.9)} \\
&\iff \mathrm{H}[\boldsymbol{f}_B \mid \boldsymbol{y}_A] \geq \mathrm{H}[\boldsymbol{f}_B \mid \boldsymbol{y}_B], && \mathrm{H}[\boldsymbol{f}_B] \text{ cancels}
\end{aligned}$$

which is also satisfied due to the "information never hurts" principle (8.8). □

The submodularity of I can be interpreted from the perspective of information theory. It turns out that submodularity is equivalent to the absence of synergy between observations (?). Intuitively, without synergies, acting greedily is enough to find a near-optimal solution. If there are synergies, then the combinatorial search problem is much harder, because single-step optimal actions do not necessarily lead us to the optimal solution. Consider the extreme case of having to solve a "needle in a haystack" problem, where only a single subset of $\mathcal{X}$ with size k achieves objective value 1, with all other subsets achieving objective value 0. In this case, we can do nothing but exhaustively search through all $|\mathcal{X}|^k$ combinations to find the optimal solution.

Problem 8.5

8.4 Maximizing Mutual Information

As we cannot efficiently pick a set $S \subseteq \mathcal{X}$ to maximize mutual information but know that I is submodular, a natural approach is to maximize mutual information greedily. That is, we pick the locations $\boldsymbol{x}_1$ through $\boldsymbol{x}_n$ individually by greedily finding the location with the maximal mutual information. The following general result for monotone submodular function maximization shows that, indeed, this greedy approach provides a good approximation.

Theorem 8.12 (Greedy submodular function maximization)**.** *If the set function $F : \mathcal{P}(\mathcal{X}) \to \mathbb{R}_{\geq 0}$ is monotone submodular, then greedily maximizing F is a $(1 - 1/e)$-approximation:*[2]

[2] $1 - 1/e \approx 0.632$

$$F(S_n) \geq \left(1 - \frac{1}{e}\right) \max_{\substack{S \subseteq \mathcal{X} \\ |S| = n}} F(S). \tag{8.28}$$

Proof. Fix any $n \geq 1$. Let $S^\star \in \arg\max\{F(S) \mid S \in \mathcal{X}, |S| \leq n\}$. We can assume $|S^\star| = n$ due to the monotonicity (8.27) of F. We write, $\{\boldsymbol{x}_1^\star, \ldots, \boldsymbol{x}_n^\star\} \doteq S^\star$. We have,

$$\begin{aligned}
F(S^\star) &\leq F(S^\star \cup S_t) && \text{using monotonicity (8.27)} \\
&= F(S_t) + \sum_{i=1}^{n} \Delta_F(\boldsymbol{x}_i^\star \mid S_t \cup \{\boldsymbol{x}_1^\star, \ldots, \boldsymbol{x}_{i-1}^\star\}) && \text{using the definition of marginal gain (8.22)} \\
&\leq F(S_t) + \sum_{\boldsymbol{x}^\star \in S^\star} \Delta_F(\boldsymbol{x}^\star \mid S_t) && \text{using submodularity (8.26)} \\
&\leq F(S_t) + n(F(S_{t+1}) - F(S_t)). && \text{using that } S_{t+1} = S_t \cup \{\boldsymbol{x}\} \text{ is chosen such that } \Delta_F(\boldsymbol{x} \mid S_t) \text{ is maximized (8.29)}
\end{aligned}$$

Let $\delta_t \doteq F(S^\star) - F(S_t)$. Then,

$$\delta_t = F(S^\star) - F(S_t) \leq n(F(S_{t+1}) - F(S_t)) = n(\delta_t - \delta_{t+1}).$$

This implies $\delta_{t+1} \leq (1 - 1/n)\delta_t$ and $\delta_n \leq (1 - 1/n)^n \delta_0 \leq \delta_0/e$, using the well-known inequality $1 - x \leq e^{-x}$ for all $x \in \mathbb{R}$.

Finally, observe that $\delta_0 = F(S^\star) - F(\emptyset) \leq F(S^\star)$ due to the non-negativity of F. We obtain,

$$\delta_n = F(S^\star) - F(S_n) \leq \frac{\delta_0}{e} \leq \frac{F(S^\star)}{e}.$$

Rearranging the terms yields the theorem. □

Optional Readings

The original proof of greedy maximization for submodular functions was given by "An analysis of approximations for maximiz-

ing submodular set functions" (Nemhauser et al., 1978).

For more background on maximizing submodular functions, see "Submodular function maximization" (Krause and Golovin, 2014).

Now that we have established that greedy maximization of mutual information is a decent approximation to maximizing the joint information of data, we will look at how this optimization problem can be solved in practice.

8.4.1 Uncertainty Sampling

When maximizing mutual information, at time t when we have already picked $S_t = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t\}$, we need to solve the following optimization problem,

$$\boldsymbol{x}_{t+1} \doteq \arg\max_{\boldsymbol{x} \in \mathcal{X}} \Delta_I(\boldsymbol{x} \mid S_t) \tag{8.29}$$

$$= \arg\max_{\boldsymbol{x} \in \mathcal{X}} \mathrm{I}(f_{\boldsymbol{x}}; y_{\boldsymbol{x}} \mid \boldsymbol{y}_{S_t}). \tag{8.30}$$

using Equation (8.23)

Note that $f_{\boldsymbol{x}}$ and $y_{\boldsymbol{x}}$ are univariate random variables. Thus, using our formula for the mutual information of conditional linear Gaussians (8.13), we can simplify to,

$$= \arg\max_{\boldsymbol{x} \in \mathcal{X}} \frac{1}{2} \log\left(1 + \frac{\sigma_t^2(\boldsymbol{x})}{\sigma_{\mathrm{n}}^2}\right) \tag{8.31}$$

where $\sigma_t^2(\boldsymbol{x})$ is the (remaining) variance at $\boldsymbol{x}$ after observing S_t. Assuming the label noise is independent of $\boldsymbol{x}$ (i.e., homoscedastic),

$$= \arg\max_{\boldsymbol{x} \in \mathcal{X}} \sigma_t^2(\boldsymbol{x}). \tag{8.32}$$

Therefore, if f is modeled by a Gaussian and we assume homoscedastic noise, greedily maximizing mutual information corresponds to simply picking the point $\boldsymbol{x}$ with the largest variance. This algorithm is also called *uncertainty sampling*.

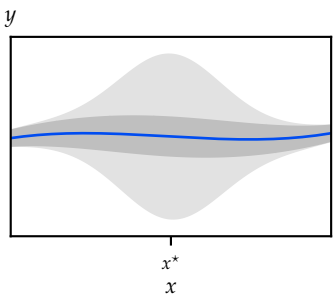


Figure 8.4: Uncertainty sampling with heteroscedastic noise. The epistemic uncertainty of the model is shown in a dark gray. The aleatoric uncertainty of the data is shown in a light gray. Uncertainty sampling would repeatedly pick points around $x^\star$ as they maximize the epistemic uncertainty, even though the aleatoric uncertainty at $x^\star$ is much larger than at the boundary.

8.4.2 Heteroscedastic Noise

Uncertainty sampling is clearly problematic if the noise is heteroscedastic. If there are a particular set of inputs with a large aleatoric uncertainty dominating the epistemic uncertainty, uncertainty sampling will continuously choose those points even though the epistemic uncertainty will not be reduced substantially (cf. Figure 8.4).

Looking at Equation (8.31) suggests a natural fix. Instead of only considering the epistemic uncertainty $\sigma_t^2(\boldsymbol{x})$, we can also consider the ale-

atoric uncertainty $\sigma_{\text{n}}^2(\boldsymbol{x})$ by modeling heteroscedastic noise, yielding

$$\boldsymbol{x}_{t+1} \doteq \arg\max_{\boldsymbol{x}\in\mathcal{X}} \frac{1}{2}\log\left(1+\frac{\sigma_t^2(\boldsymbol{x})}{\sigma_{\text{n}}^2(\boldsymbol{x})}\right) = \arg\max_{\boldsymbol{x}\in\mathcal{X}} \frac{\sigma_t^2(\boldsymbol{x})}{\sigma_{\text{n}}^2(\boldsymbol{x})}. \tag{8.33}$$

Thus, we choose locations that trade large epistemic uncertainty with large aleatoric uncertainty. Ideally, we find a location where the epistemic uncertainty is large, and the aleatoric uncertainty is low, which promises a significant reduction of uncertainty around this location.

8.4.3 *Classification*

While we focused on regression, one can apply active learning also for other settings, such as (probabilistic) classification. In this setting, for any input $\boldsymbol{x}$, a model produces a categorical distribution over labels $y_{\boldsymbol{x}}$.[3] Here, uncertainty sampling corresponds to selecting samples that maximize the entropy of the predicted label $y_{\boldsymbol{x}}$,

[3] see Section 1.3

$$\boldsymbol{x}_{t+1} \doteq \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mathrm{H}[y_{\boldsymbol{x}} \mid \boldsymbol{x}_{1:t}, y_{1:t}]. \tag{8.34}$$

The entropy of a categorical distribution is simply a finite sum of weighted surprise terms.

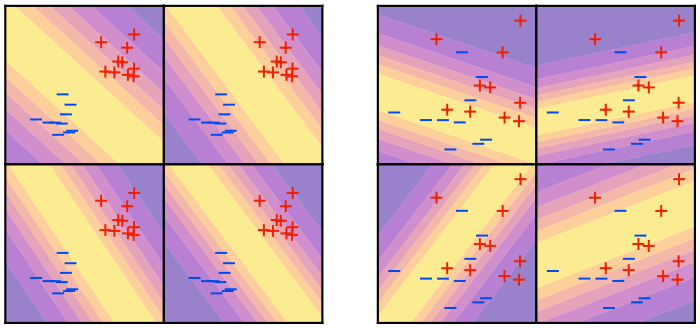


Figure 8.5: Uncertainty sampling in classification. The area with high uncertainty (as measured by entropy) is highlighted in yellow. The shown figures display each a sequence of model updates, each after one new observation. In the left figure, the classes are well-separated and uncertainty is dominated by epistemic uncertainty, whereas in the right figure the uncertainty is dominated by noise. In the latter case, if we mostly choose points $\boldsymbol{x}$ in the area of highest uncertainty (i.e., close to the decision boundary) to make observations, the label noise results in frequently changing models.

This approach generally leads to sampling points that are close to the decision boundary. Often, the uncertainty is mainly dominated by label noise rather than epistemic uncertainty, and hence, we do not learn much from our observations. This is a similar problem to the one we encountered with uncertainty sampling in the setting of heteroscedastic noise.

This naturally suggests distinguishing between the aleatoric and epistemic uncertainty of the model f (parameterized by $\boldsymbol{\theta}$). To this end, mutual information can be used similarly as we have done with uncertainty sampling for regression,

$$\boldsymbol{x}_{t+1} \doteq \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mathrm{I}(\boldsymbol{\theta}; y_{\boldsymbol{x}} \mid \boldsymbol{x}_{1:t}, y_{1:t}) \tag{8.35}$$

$$= \arg\max_{x \in \mathcal{X}} \mathrm{I}(y_x; \boldsymbol{\theta} \mid \boldsymbol{x}_{1:t}, y_{1:t})$$ using symmetry (8.11)

$$= \arg\max_{x \in \mathcal{X}} \mathrm{H}[y_x \mid \boldsymbol{x}_{1:t}, y_{1:t}] - \mathrm{H}[y_x \mid \boldsymbol{\theta}, \boldsymbol{x}_{1:t}, y_{1:t}]$$ using the definition of mutual information (8.9)

$$= \arg\max_{x \in \mathcal{X}} \mathrm{H}[y_x \mid \boldsymbol{x}_{1:t}, y_{1:t}] - \mathbb{E}_{\boldsymbol{\theta} \mid \boldsymbol{x}_{1:t}, y_{1:t}}[\mathrm{H}[y_x \mid \boldsymbol{\theta}, \boldsymbol{x}_{1:t}, y_{1:t}]] \quad (8.36)$$ using the definition of conditional entropy (8.2)

$$= \arg\max_{x \in \mathcal{X}} \underbrace{\mathrm{H}[y_x \mid \boldsymbol{x}_{1:t}, y_{1:t}]}_{\substack{\text{entropy of} \\ \text{predictive posterior}}} - \mathbb{E}_{\boldsymbol{\theta} \mid \boldsymbol{x}_{1:t}, y_{1:t}} \underbrace{\mathrm{H}[y_x \mid \boldsymbol{\theta}]}_{\substack{\text{entropy of} \\ \text{likelihood}}}. \quad (8.37)$$ using the definition of entropy (5.27) and assuming $y_x \perp \boldsymbol{x}_{1:t}, y_{1:t} \mid \boldsymbol{\theta}$

The first term measures the entropy of the averaged prediction while the second term measures the average entropy of predictions. Thus, the first term looks for points where the average prediction is not confident. In contrast, the second term penalizes points where many of the sampled models are not confident about their prediction, and thus looks for points where the models are confident in their predictions. This identifies those points $\boldsymbol{x}$ where the models *disagree* about the label y_x (that is, each model is "confident" but the models predict different labels). For this reason, this approach is known as *Bayesian active learning by disagreement* (BALD).

Note that the second term of the difference acts as a regularizer when compared to Equation (8.34). The second term mirrors our description of aleatoric uncertainty from Section 2.2. Recall that we interpreted aleatoric uncertainty as the average uncertainty for all models. Crucially, here we use entropy to "measure" uncertainty, whereas previously we have been using variance. Therefore, intuitively, Equation (8.36) subtracts the aleatoric uncertainty from the total uncertainty about the label.

Observe that both terms require approximate forms of the posterior distribution. In Chapters 5 and 6, we have seen various approaches from variational inference and MCMC methods which can be used here. The first term can be approximated by the predictive distribution of an approximated posterior which is obtained, for example, using variational inference. The nested entropy of the second term is typically easy to compute, as it corresponds to the entropy of the (discrete) likelihood distribution $p(y \mid \boldsymbol{x}, \boldsymbol{\theta})$ of the model $\boldsymbol{\theta}$. The outer expectation of the second term may be approximated using (approximate) samples from the posterior distribution obtained via variational inference, MCMC, or some other method.

Optional Readings

- Gal, Islam, and Ghahramani (2017). *Deep Bayesian active learning with image data.*

8.5 Learning Locally: Transductive Active Learning

So far we have explored how to select observations that provide us with the best predictor $f(\boldsymbol{x})$ across the entire domain $\boldsymbol{x} \in \mathcal{X}$. However, we typically utilize predictors to make predictions at a particular location $\boldsymbol{x}^\star$. It is therefore a natural question to ask how to select observations that lead to the best individual prediction $f(\boldsymbol{x}^\star)$ at the prediction target $\boldsymbol{x}^\star$. The distinction between these two settings is closely related to the distinction between two general approaches to learning: *inductive learning* and *transductive learning*. Inductive learning aims to extract general rules from the data, that is, to extract and compress the most bits of information. In contrast, transductive learning aims to make the best prediction at a particular location, that is, to extract the most bits of information *relevant to the prediction* at $\boldsymbol{x}^\star$. The concept of transduction was developed by Vladimir Vapnik in the 1980s who described its essence as follows:

> "When solving a problem of interest, do not solve a more general problem as an intermediate step. Try to get the answer that you really need but not a more general one." — Vladimir Vapnik

Remark 8.13: What are the prediction targets $x^\star$?

Typically, in transductive learning, we cannot directly observe the value $f(\boldsymbol{x}^\star)$ at the prediction target $\boldsymbol{x}^\star$, for example, when learning from a fixed dataset $\mathcal{X}' \subset \mathcal{X}$. If we *could* observe $f(\boldsymbol{x}^\star)$ directly (or perturbed by noise), solving the learning task would only require memorization. Instead, most interesting learning tasks require generalizing $f(\boldsymbol{x}^\star)$ from the behavior of f at other locations. Therefore, transductive learning becomes interesting precisely when we cannot directly observe $f(\boldsymbol{x}^\star)$.[4]

Note that this is similar to the inductive setting where we, in principle, could observe $f(\boldsymbol{x})$ at any location $\boldsymbol{x}$, but practically, we can only observe $f(\boldsymbol{x})$ at a finite number of locations. Since such an inductive model is then used to make predictions at any location $\boldsymbol{x}^\star$, it also needs to generalize from the observations.

[4] We will discuss an example of this kind in Problem 8.6.

Following the transductive paradigm, when already knowing that our goal is to predict $f(\boldsymbol{x}^\star)$ our objective is to select observations that provide the most information about $f(\boldsymbol{x}^\star)$. We can express this objective elegantly using the probabilistic framework from this chapter (MacKay, 1992; Hübotter et al., 2024):

$$\boldsymbol{x}_{t+1} \doteq \arg\max_{\boldsymbol{x} \in \mathcal{X}'} \mathrm{I}(f_{\boldsymbol{x}^\star}; y_{\boldsymbol{x}} \mid \boldsymbol{x}_{1:t}, y_{1:t}) = \arg\min_{\boldsymbol{x} \in \mathcal{X}'} \mathrm{H}[f_{\boldsymbol{x}^\star} \mid \boldsymbol{x}_{1:t}, y_{1:t}, y_{\boldsymbol{x}}]. \tag{8.38}$$

Hübotter et al. (2024) show that this objective leads to a remarkably different selection of observations compared to the inductive active learning objective we discussed earlier. Indeed, while the inductive objective focuses on selecting a *diverse* set of examples, the transductive objective also takes into account the *relevance* of the examples to the prediction target $x^\star$. We can see this tradeoff between diversity and relevance by rewriting the transductive objective as

$$\mathrm{I}(f_{x^\star}; y_x \mid x_{1:t}, y_{1:t}) = \underbrace{\mathrm{I}(f_{x^\star}; y_x)}_{\text{relevance}} - \underbrace{\mathrm{I}(f_{x^\star}; y_x; y_{1:t} \mid x_{1:t})}_{\text{redundancy}} \tag{8.39}$$

using the definition of interaction information (8.18)

where the first term measures the information gain of y_x about $f_{x^\star}$ while the second term is the interaction information which measures the redundancy of the information in y_x and $y_{1:t}$ about $f_{x^\star}$. In this way, transductive active learning describes a middle ground between traditional search and retrieval methods that focus on relevance on the one hand and inductive active learning which focuses on diversity on the other hand.

Optional Readings

- Hübotter, Sukhija, Treven, As, and Krause (2024). *Transductive active learning: Theory and applications*.

In modern machine learning, one often differentiates between a "pre-training" and a "fine-tuning" stage. During pre-training, a model is trained on a large dataset to extract general knowledge without a specific task in mind. Then, during fine-tuning, the model is adapted to a specific task by training on a smaller dataset. Whereas (inductive) active learning is closely linked to the pre-training stage, transductive active learning has been shown to be useful for task-specific fine-tuning:

- Hübotter, Bongni, Hakimi, and Krause (2025). *Efficiently Learning at Test-Time: Active Fine-Tuning of LLMs*.
- Bagatella, Hübotter, Martius, and Krause (2024). *Active Fine-Tuning of Generalist Policies*.

Discussion

We have discussed how to select the most informative data. Thereby, we focused mostly on *inductive* active learning which is applicable to "pre-training" when we aim to extract general knowledge from data, but also explored *transductive* active learning which is useful for "fine-tuning" when we aim to adapt a model to a specific task.

We focused on quantifying the "informativeness" of data by its information gain, which is a common approach, though many other viable

criteria exist:

Remark 8.14: Beyond mutual information

The problem of identifying which experiments to conduct in order to maximize learning is studied extensively in the field of *experimental design* where a set of observations S is commonly called a *design*. In the presence of a prior and likelihood, and a different possible posterior distribution for each design S, the field is also called *Bayesian experimental design* (Chaloner and Verdinelli, 1995; Rainforth et al., 2024; Mutnỳ, 2024).

As we highlighted in the beginning of this chapter, how we measure the utility (i.e., the informativeness) of a design S is crucial. Such a measure is called a *design criterion* and a design which is optimal with respect to a design criterion is called an *optimal design*. The literature studies various design criteria beyond maximizing mutual information (i.e., minimizing posterior entropy). One popular alternative is to select the observations S that minimize the trace of the posterior covariance matrix, which corresponds to minimizing the posterior average variance.

Next, we will move to the topic of optimization and ask which data we should select to find the optimum of an unknown function as quickly as possible. In the following chapter, we will focus on "Bayesian optimization" (also called "bandit optimization") where our aim is to *find and sample* the optimal point. A related task that is slightly more related to active learning is the "best-arm identification" problem where we aim only to *identify* the optimal point without sampling it. This problem is closely related transductive active learning (with the local task being defined by the location of the maximum) and so-called *entropy search* methods that minimize the entropy of the posterior distribution over the location or value of the maximum (akin to Equation (8.38)) are often used to solve this problem (Hennig and Schuler, 2012; Wang and Jegelka, 2017; Hvarfner et al., 2022).

Problems

8.1. Mutual information and KL divergence.

Show that by expanding the definition of mutual information,

$$\begin{aligned} \mathrm{I}(\mathbf{X}; \mathbf{Y}) &= \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{y})}{p(\boldsymbol{x})p(\boldsymbol{y})}\right] \\ &= \mathrm{KL}(p(\boldsymbol{x}, \boldsymbol{y}) \| p(\boldsymbol{x})p(\boldsymbol{y})) \qquad (8.40) \\ &= \mathbb{E}_{\boldsymbol{y}\sim p}[\mathrm{KL}(p(\boldsymbol{x} \mid \boldsymbol{y}) \| p(\boldsymbol{x}))], \qquad (8.41) \end{aligned}$$

where $p(\boldsymbol{x}, \boldsymbol{y})$ denotes the joint distribution and $p(\boldsymbol{x}), p(\boldsymbol{y})$ denote the marginal distributions of $\mathbf{X}$ and $\mathbf{Y}$.

8.2. Non-monotonicity of cond. mutual information.

1. Show that if $\mathbf{X} \perp \mathbf{Z}$ then $\mathrm{I}(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z}) \geq \mathrm{I}(\mathbf{X}; \mathbf{Y})$.
2. Show that if $\mathbf{X} \perp \mathbf{Z} \mid \mathbf{Y}$ then $\mathrm{I}(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z}) \leq \mathrm{I}(\mathbf{X}; \mathbf{Y})$.
3. Note that the condition $\mathbf{X} \perp \mathbf{Z} \mid \mathbf{Y}$ is the Markov property, namely, $\{\mathbf{X}, \mathbf{Y}, \mathbf{Z}\}$ form a Markov chain with graphical model $\mathbf{X} \to \mathbf{Y} \to \mathbf{Z}$. This situation often occurs when data is processed sequentially. Prove

$$\mathrm{I}(\mathbf{X}; \mathbf{Z}) \leq \mathrm{I}(\mathbf{X}; \mathbf{Y}). \tag{8.42}$$

   which is also known as the *data processing inequality*, and which says that processing cannot increase the information contained in a signal.

8.3. Interaction information.

1. Show that interaction information is symmetric.
2. Let $X_1, X_2 \sim \mathrm{Bern}(p)$ for some $p \in (0, 1)$ and independent. We denote by $Y \doteq X_1 \oplus X_2$ their XOR. Compute $\mathrm{I}(Y; X_1; X_2)$.

8.4. Marginal gain of maximizing mutual information.

Show that in the context of maximizing mutual information, the marginal gain is

$$\Delta_I(\boldsymbol{x} \mid A) = \mathrm{I}(f_{\boldsymbol{x}}; y_{\boldsymbol{x}} \mid \boldsymbol{y}_A) = \mathrm{H}[y_{\boldsymbol{x}} \mid \boldsymbol{y}_A] - \mathrm{H}[\varepsilon_{\boldsymbol{x}}].$$

8.5. Submodularity means no synergy.

Show that submodularity is equivalent to the absence of synergy between observations. That is, show that for all $A \subseteq B$,

$$\mathrm{I}(f_{\boldsymbol{x}}; y_{\boldsymbol{x}}; \boldsymbol{y}_{B \setminus A} \mid \boldsymbol{y}_A) \geq 0. \tag{8.43}$$

8.6. Transductive active learning.

Consider the prior distribution $X_i \sim \mathcal{N}(0, 1)$ with all X_i independent and consider the "output" variable

$$Z \doteq \sum_{i=1}^{100} i \cdot X_i.$$

Our observation when X_{i_t} is selected in round t is generated by

$$Y_t \doteq X_{i_t} + \varepsilon_t \quad \text{with } \varepsilon_t \overset{\text{iid}}{\sim} \mathcal{N}(0, 1).$$

For a set of inputs $S = \{i_1, \ldots, i_t\} \subseteq \{1, \ldots, 100\}$, we define the *information gain* of S as

$$F(S) \doteq \mathrm{I}(Z; Y_{1:t}) = \mathrm{H}[Z] - \mathrm{H}[Z \mid Y_{1:t}]$$

where $\mathrm{H}[Z]$ is the entropy of Z according to the prior and $\mathrm{H}[Z \mid Y_{1:t}]$ is the conditional entropy after the observations $Y_1, \ldots, Y_t$.

Note: The random vector $(X_1, \ldots, X_{100}, Z)$ is jointly Gaussian due to the closedness of Gaussians under linear transformations.

1. We define the *marginal information gain* $\Delta(j \mid S)$ for a new observation Y_j as
$$\Delta(j \mid S) \doteq F(S \cup \{j\}) - F(S).$$
Does $\Delta(j \mid S) \geq 0$ hold for all j and S?
2. Is maximizing the marginal information gain $\Delta(i \mid S)$ equivalent to picking the point $i \in \{1, \ldots, 100\}$ with maximum variance under the current posterior over X_i, that is, "equivalent to uncertainty sampling"?
3. Let us consider the alternative prior where $X_i \sim \mathrm{Bern}(0.5)$ are fair coin flips which we observe directly (i.e., $Y_t = X_{i_t}$). For which of the following definitions of Z is the acquisition function $S \mapsto F(S)$ submodular?
   (a) $Z \doteq X_1 \wedge \cdots \wedge X_{100}$ with $\wedge$ denoting the logical AND.
   (b) $Z \doteq X_1 \vee \cdots \vee X_{100}$ with $\vee$ denoting the logical OR.
   (c) $Z \doteq X_1 \oplus \cdots \oplus X_{100}$ with $\oplus$ denoting the logical XOR, i.e., the exclusive OR which returns 1 iff exactly one of the X_i is 1 and 0 otherwise.

9

Bayesian Optimization

Often, obtaining data is costly. In the previous chapter, this led us to investigate how we can optimally improve our understanding (i.e., reduce uncertainty) of the process we are trying to model. However, purely improving our understanding is often not good enough. In many cases, we want to use our improving understanding *simultaneously* to reach certain goals. This is a very common problem in artificial intelligence and will concern us for the rest of this manuscript. One common instance of this problem is the setting of optimization.

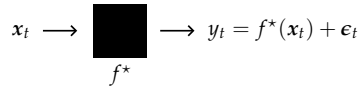


Figure 9.1: Illustration of Bayesian optimization. We pass an input x_t into the unknown function $f^\star$ to obtain noisy observations y_t.

Given some function $f^\star : \mathcal{X} \to \mathbb{R}$, suppose we want to find the

$$\arg\max_{x \in \mathcal{X}} f^\star(x). \tag{9.1}$$

Now, contrary to classical optimization, we are interested in the setting where the function $f^\star$ is unknown to us (like a "black-box"). We are only able to obtain noisy observations of $f^\star$,

$$y_t = f^\star(x_t) + \varepsilon_t. \tag{9.2}$$

Moreover, these noise-perturbed evaluations are costly to obtain. We will assume that similar alternatives yield similar results,[1] which is commonly encoded by placing a Gaussian process prior on $f^\star$. This assumed correlation is fundamentally what will allow us to learn a model of $f^\star$ from relatively few samples.[2]

[1] That is, $f^\star$ is "smooth". We will be more precise in the subsequent parts of this chapter. If this were not the case, optimizing the function without evaluating it everywhere would not be possible. Fortunately, many interesting functions obey by this relatively weak assumption.

[2] There are countless examples of this problem in the "real world". Instances are

- drug trials
- chemical engineering — the development of physical products
- recommender systems
- automatic machine learning — automatic tuning of model & hyperparameters
- and many more...

9.1 Exploration-Exploitation Dilemma

In Bayesian optimization, we want to learn a model of $f^\star$ and use this model to optimize $f^\star$ simultaneously. These goals are somewhat contrary. Learning a model of $f^\star$ requires us to explore the input space while using the model to optimize $f^\star$ requires us to focus on the most promising well-explored areas. This trade-off is commonly known as the *exploration-exploitation dilemma*, whereby

- *exploration* refers to choosing points that are "informative" with respect to the unknown function. For example, points that are far away from previously observed points (i.e., have high posterior variance);[3] and
- *exploitation* refers to choosing promising points where we expect the function to have high values. For example, points that have a high posterior mean and a low posterior variance.

In other words, the exploration-exploitation dilemma refers to the challenge of learning enough to understand $f^\star$, but not learning too much to lose track of the objective — optimizing $f^\star$.

The exploration-exploitation dilemma is yet another example of the *principle of curiosity and conformity* which we introduced in Section 5.5.2 and encountered many times since in our study of approximate probabilistic inference. We will see in subsequent chapters that sequential decision-making is intimately related to probabilistic inference, and there we will also make this correspondence more precise.

[3] We explored this topic (with strategies like uncertainty sampling) in the previous chapter.

9.2 Online Learning and Bandits

Bayesian optimization is closely related to a form of online learning. In *online learning* we are given a set of possible inputs $\mathcal{X}$ and an unknown function $f^\star : \mathcal{X} \to \mathbb{R}$. We are now asked to choose a sequence of inputs $x_1, \dots x_T$ online,[4] and our goal is to maximize our cumulative reward $\sum_{t=1}^{T} f^\star(x_t)$. Depending on what we observe about $f^\star$, there are different variants of online learning. Bayesian optimization is closest to the so-called (stochastic) "bandit" setting.

[4] *Online* is best translated as "sequential". That is, we need to pick x_{t+1} based only on our prior observations $y_1, \dots, y_t$.

9.2.1 Multi-Armed Bandits

The "*multi-armed bandits*" (MAB) problem is a classical, canonical formalization of the exploration-exploitation dilemma. In the MAB problem, we are provided with k possible actions (arms) and want to maximize our reward online within the time horizon T. We do not know the reward distributions of the actions in advance, however, so we need to trade learning the reward distribution with following the most promising action. Bayesian optimization can be interpreted as a variant of the MAB problem where there can be a potentially infinite number of actions (arms), but their rewards are correlated (because of the smoothness of the Gaussian process prior).

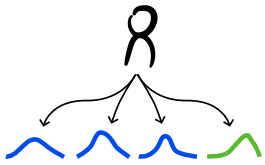


Figure 9.2: Illustration of a multi-armed bandit with four arms, each with a different reward distribution. The agent tries to identify the arm with the most beneficial reward distribution shown in green.

There exists a large body of work on this and similar problems in online decision-making. Much of this work develops theory on how to explore and exploit in the face of uncertainty. The shared prevalence of the exploration-exploitation dilemma signals a deep connection be-

tween online learning and Bayesian optimization (and — as we will later come to see — reinforcement learning). Many of the approaches which we will encounter in the context of these topics are strongly related to methods in online learning.

One of the key principles of the theory on multi-armed bandits and reinforcement learning is the principle of *"optimism in the face of uncertainty"*, which suggests that it is a good guideline to explore where we can hope for the best outcomes. We will frequently come back to this general principle in our discussion of algorithms for Bayesian optimization and reinforcement learning.

9.2.2 Regret

The key performance metric in online learning is the regret.

Definition 9.1 (Regret)**.** The *(cumulative) regret* for a time horizon T associated with choices $\{\boldsymbol{x}_t\}_{t=1}^T$ is defined as

$$R_T \doteq \sum_{t=1}^{T} \underbrace{\left(\max_{\boldsymbol{x}} f^\star(\boldsymbol{x}) - f^\star(\boldsymbol{x}_t)\right)}_{\textit{instantaneous regret}} \tag{9.3}$$

$$= T \max_{\boldsymbol{x}} f^\star(\boldsymbol{x}) - \sum_{t=1}^{T} f^\star(\boldsymbol{x}_t). \tag{9.4}$$

The regret can be interpreted as the additive loss with respect to the *static* optimum $\max_{\boldsymbol{x}} f^\star(\boldsymbol{x})$.

The goal is to find algorithms that achieve *sublinear* regret,

$$\lim_{T\to\infty} \frac{R_T}{T} = 0. \tag{9.5}$$

Importantly, if we use an algorithm which explores *forever*, e.g., by going to a random point $\tilde{\boldsymbol{x}}$ with a constant probability ϵ in each round, then the regret will grow linearly with time. This is because the instantaneous regret is at least $\epsilon(\max_{\boldsymbol{x}} f^\star(\boldsymbol{x}) - f^\star(\tilde{\boldsymbol{x}}))$ and non-decreasing. Conversely, if we use an algorithm which *never* explores, then we might never find the static optimum, and hence, also incur constant instantaneous regret in each round, implying that regret grows linearly with time. Thus, achieving sublinear regret requires *balancing* exploration and exploitation.

Typically, online learning (and Bayesian optimization) consider stationary environments, hence the comparison to the static optimum. Dynamic environments are studied in *online algorithms* (see metrical task systems[5], convex function chasing[6], and generalizations of multi-armed bandits to changing reward distributions) and reinforcement

[5] Metrical task systems are a classical example in online algorithms. Suppose we are moving in a (finite) decision space $\mathcal{X}$. In each round, we are given a "task" $f_t : \mathcal{X} \to \mathbb{R}$ which is more or less costly depending on our state $\boldsymbol{x}_t \in \mathcal{X}$. In many contexts, it is natural to assume that it is also costly to move around in the decision space. This cost is modeled by a metric $d(\cdot,\cdot)$ on $\mathcal{X}$. In *metrical task systems*, we want to minimize our total cost,

$$\sum_{t=1}^{T} f_t(\boldsymbol{x}_t) + d(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}).$$

That is, we want to trade completing our tasks optimally with moving around in the state space. Crucially, we do not know the sequence of tasks f_t in advance. Due to the cost associated with moving in the decision space, previous choices affect the future!

[6] *Convex function chasing* (or *convex body chasing*) generalize metrical task systems to continuous domains $\mathcal{X}$. To make any guarantees about the performance in these settings, one typically has to assume that the tasks f_t are convex. Note that this mirrors our assumption in Bayesian optimization that similar alternatives yield similar results.

learning. When operating in dynamic environments, other metrics such as the competitive ratio,[7] which compares against the best *dynamic* choice, are useful. As we will later come to see in Section 13.1 in the context of reinforcement learning, operating in dynamic environments is deeply connected to a rich field of research called *control*.

[7] To assess the performance in dynamic environments, we typically compare to a dynamic optimum. As these problems are difficult (we are usually not able to guarantee convergence to the dynamic optimum), one considers a multiplicative performance metric similar to the approximation ratio, the *competitive ratio*,

$$\text{cost}(\text{ALG}) \leq \alpha \cdot \text{cost}(\text{OPT}),$$

where OPT corresponds to the dynamic optimal choice (in hindsight).

9.3 Acquisition Functions

It is common to use a so-called *acquisition function* to greedily pick the next point to sample based on the current model.

Throughout our description of acquisition functions, we will focus on a setting where we model $f^\star$ using a Gaussian process which we denote by f. The methods generalize to other means of learning $f^\star$ such as Bayesian deep learning. The various acquisition functions F are used in the same way as is illustrated in Algorithm 9.2.

Algorithm 9.2: Bayesian optimization (with GPs)

initialize $f \sim \mathcal{GP}(\mu_0, k_0)$
for $t = 1$ **to** T **do**
  choose $\boldsymbol{x}_t = \arg\max_{\boldsymbol{x} \in \mathcal{X}} F(\boldsymbol{x}; \mu_{t-1}, k_{t-1})$
  observe $y_t = f(\boldsymbol{x}_t) + \epsilon_t$
  perform a probabilistic update to obtain μ_t and k_t

Remark 9.3: Model selection

Selecting a model of $f^\star$ in sequential decision-making is much harder than in the i.i.d. data setting of supervised learning. There are mainly the following two dangers:

- the data sets collected in active learning and Bayesian optimization are *small*; and
- the data points are selected *dependently* on prior observations.

This leads to a specific danger of overfitting. In particular, due to feedback loops between the model and the acquisition function, one may end up sampling the same point repeatedly.

One approach to reduce the chance of overfitting is the use of hyperpriors which we mentioned previously in Section 4.4.2. Another approach that often works fairly well is to occasionally (according to some schedule) select points uniformly at random instead of using the acquisition function. This tends to prevent getting stuck in suboptimal parts of the state space.

One possible acquisition function is uncertainty sampling (8.31), which we discussed in the previous chapter. However, this acquisition function does not at all take into account the objective of maximizing $f^\star$ and focuses solely on exploration.

Suppose that our model f of $f^\star$ is well-calibrated, in the sense that the true function lies within its confidence bounds. Consider the best lower bound, that is, the maximum of the lower confidence bound. Now, if the true function is really contained in the confidence bounds, it must hold that the optimum is somewhere above this best lower bound. In particular, we can exclude all regions of the domain where the upper confidence bound (the optimistic estimate of the function value) is lower than the best lower bound. This is visualized in Figure 9.3.

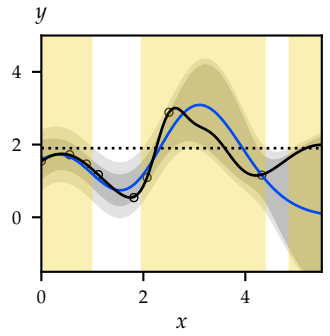


Figure 9.3: Optimism in Bayesian optimization. The **unknown function** is shown in black, our **model** in blue with gray confidence bounds. The dotted black line denotes the maximum lower bound. We can therefore focus our exploration to the yellow regions where the upper confidence bound is higher than the maximum lower bound.

Therefore, we only really care how the function looks like in the regions where the upper confidence bound is larger than the best lower bound. The key idea behind the methods that we will explore is to focus exploration on these plausible maximizers.

Note that it is crucial that our uncertainty about f reflects the "fit" of our model to the unknown function. If the model is not well calibrated or does not describe the underlying function at all, these methods will perform poorly. This is where we can use the Bayesian philosophy by imposing a prior belief that may be conservative.

9.3.1 *Upper Confidence Bound*

The principle of *optimism in the face of uncertainty* suggests picking the point where we can hope for the optimal outcome. In this setting, this corresponds to simply maximizing the *upper confidence bound* (UCB),

$$\boldsymbol{x}_{t+1} \doteq \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mu_t(\boldsymbol{x}) + \beta_{t+1}\sigma_t(\boldsymbol{x}), \tag{9.6}$$

where $\sigma_t(\boldsymbol{x}) \doteq \sqrt{k_t(\boldsymbol{x},\boldsymbol{x})}$ is the standard deviation at $\boldsymbol{x}$ and β_t regulates how confident we are about our model f (i.e., the choice of confidence interval).

This acquisition function naturally trades exploitation by preferring a large posterior mean with exploration by preferring a large posterior variance. Note that if $\beta_t = 0$ then UCB is purely exploitative, whereas, if $\beta_t \to \infty$, UCB recovers uncertainty sampling (i.e., is purely explorative).[8] UCB is an example of an optimism-based method, as it greedily picks the point where we can hope for the best outcome.

As can be seen in Figure 9.4, the UCB acquisition function is generally non-convex. For selecting the next point, we can use approximate

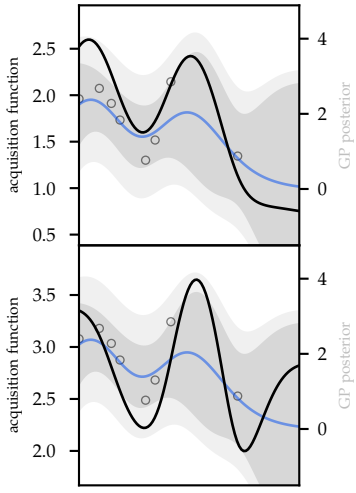


Figure 9.4: Plot of the UCB acquisition function for $\beta = 0.25$ and $\beta = 1$, respectively.

[8] Due to the monotonicity of $(\cdot)^2$, it does not matter whether we optimize the variance or standard deviation at $\boldsymbol{x}$.

global optimization techniques like Lipschitz optimization (in low dimensions) and gradient ascent with random initialization (in high dimensions). Another widely used technique is to sample some random points from the domain, score them according to this criterion, and simply take the best one.

The choice of β_t is crucial for the performance of UCB. Intuitively, for UCB to work even if the unknown function $f^\star$ is not contained in the confidence bounds, we use β_t to re-scale the confidence bounds to enclose $f^\star$ as shown in Figure 9.5. A theoretical analysis requires that β_t is chosen "correctly". Formally, we say that the sequence β_t is chosen correctly if it leads to *well-calibrated confidence intervals*, that is, if with probability at least $1 - \delta$,

$$\forall t \geq 1,\ \forall \boldsymbol{x} \in \mathcal{X} : \quad f^\star(\boldsymbol{x}) \in \mathcal{C}_t(\boldsymbol{x}) \doteq [\mu_{t-1}(\boldsymbol{x}) \pm \beta_t(\delta) \cdot \sigma_{t-1}(\boldsymbol{x})]. \quad (9.7)$$

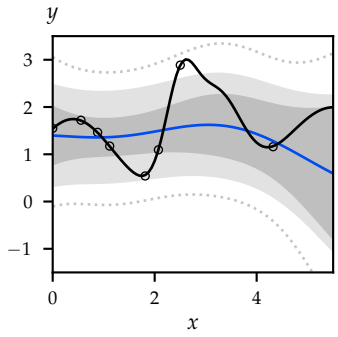


Figure 9.5: Re-scaling the confidence bounds. The dotted gray lines represent updated confidence bounds.

Bounds on $\beta_t(\delta)$ can be derived both in a "Bayesian" and in a "frequentist" setting. In the Bayesian setting, it is assumed that $f^\star$ is drawn from the prior GP, i.e., $f^\star \sim \mathcal{GP}(\mu_0, k_0)$. However, in many cases this may be an unrealistic assumption. In the frequentist setting, it is assumed instead that $f^\star$ is a fixed element of a reproducing kernel Hilbert space $\mathcal{H}_k(\mathcal{X})$ which depending on the kernel k can encompass a large class of functions. We will discuss the Bayesian setting first and later return to the frequentist setting.

Theorem 9.4 (Bayesian confidence intervals, lemma 5.5 of Srinivas et al. (2010))**.** (?) *Let $\delta \in (0, 1)$. Assuming $f^\star \sim \mathcal{GP}(\mu_0, k_0)$ and Gaussian observation noise $\epsilon_t \sim \mathcal{N}(0, \sigma_n^2)$, the sequence*

Problem 9.2

$$\beta_t(\delta) = O\left(\sqrt{\log(|\mathcal{X}|t/\delta)}\right) \quad (9.8)$$

satisfies $\mathbb{P}(\forall t \geq 1, \boldsymbol{x} \in \mathcal{X} : f^\star(\boldsymbol{x}) \in \mathcal{C}_t(\boldsymbol{x})) \geq 1 - \delta$.

Under the assumption of well-calibrated confidence intervals, we can bound the regret of UCB.

Theorem 9.5 (Regret of GP-UCB, theorem 2 of Srinivas et al. (2010))**.** (?) *If $\beta_t(\delta)$ is chosen "correctly" for a fixed $\delta \in (0, 1)$, with probability at least $1 - \delta$, greedily choosing the upper confidence bound yields cumulative regret*

Problem 9.3

$$R_T = O\left(\beta_T(\delta)\sqrt{\gamma_T T}\right) \quad (9.9)$$

where

$$\gamma_T \doteq \max_{\substack{S \subseteq \mathcal{X} \\ |S| = T}} \mathrm{I}(\boldsymbol{f}_S; \boldsymbol{y}_S) = \max_{\substack{S \subseteq \mathcal{X} \\ |S| = T}} \frac{1}{2} \log\det\left(\boldsymbol{I} + \sigma_n^{-2} \boldsymbol{K}_{SS}\right) \quad (9.10)$$

is the maximum information gain after T rounds.

Observe that if the information gain is sublinear in T then we achieve sublinear regret and, in particular, converge to the true optimum. The information gain γ_T measures how much can be learned about $f^\star$ within T rounds. If the function is assumed to be smooth (perhaps even linear), then the information gain is smaller than if the function was assumed to be "rough". Intuitively, the smoother the functions encoded by the prior, the smaller is the class of functions to choose from and the more can be learned from a single observation about "neighboring" points.

Theorem 9.6 (Information gain of common kernels, theorem 5 of Srinivas et al. (2010) and remark 2 of Vakili et al. (2021))**.** *Due to submodularity, we have the following bounds on the information gain of common kernels:*

- linear kernel

$$\gamma_T = O(d \log T), \tag{9.11}$$

- Gaussian kernel

$$\gamma_T = O\Big((\log T)^{d+1}\Big), \tag{9.12}$$

- Matérn kernel *for* $\nu > \frac{1}{2}$

$$\gamma_T = O\Big(T^{\frac{d}{2\nu+d}} (\log T)^{\frac{2\nu}{2\nu+d}}\Big). \tag{9.13}$$

The information gain of common kernels is illustrated in Figure 9.6. Notably, when all points in the domain are independent, the information gain is linear in T. This is because when the function $f^\star$ may be arbitrarily "rough", we cannot generalize from a single observation to "neighboring" points, and as there are infinitely many points in the domain $\mathcal{X}$ there are no diminishing returns. As one would expect, in this case, Theorem 9.5 does not yield sublinear regret. However, we can see from Theorem 9.6 that the information gain is sublinear for linear, Gaussian, and most Matérn kernels. Moreover, observe that unless the function is linear, the information gain grows exponentially with the dimension d. This is because the number of "neighboring" points (with respect to Euclidean geometry) decreases exponentially with the dimension which is also known as the *curse of dimensionality*.

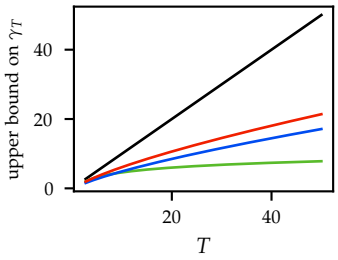


Figure 9.6: Information gain of **independent**, **linear**, **Gaussian**, and **Matérn** ($\nu \approx 0.5$) kernels with $d = 2$ (up to constant factors). The kernels with sublinear information gain have strong diminishing returns (due to their strong dependence between "close" points). In contrast, the independent kernel has no dependence between points in the domain, and therefore no diminishing returns. Intuitively, the "smoother" the class of functions modeled by the kernel, the stronger are the diminishing returns.

As mentioned, the size of the confidence intervals can also be analyzed under a frequentist assumption on $f^\star$.

Theorem 9.7 (Frequentist confidence intervals, theorem 2 of Chowdhury and Gopalan (2017))**.** *Let* $\delta \in (0, 1)$*. Assuming* $f^\star \in \mathcal{H}_k(\mathcal{X})$*, we have that with probability at least* $1 - \delta$*, the sequence*

$$\beta_t(\delta) = \|f^\star\|_k + \sigma_n \sqrt{2(\gamma_t + \log(1/\delta))} \tag{9.14}$$

satisfies $\mathbb{P}(\forall t \geq 1, \boldsymbol{x} \in \mathcal{X} : f^\star(\boldsymbol{x}) \in \mathcal{C}_t(\boldsymbol{x})) \geq 1 - \delta$.

That is, β_t depends on the information gain of the kernel as well as on the "complexity" of $f^\star$ which is measured in terms of its norm in the underlying reproducing kernel Hilbert space $\mathcal{H}_k(\mathcal{X})$.

Remark 9.8: Bayesian vs frequentist assumption

Theorems 9.4 and 9.7 provide different bounds on $\beta_t(\delta)$ based on fundamentally different assumptions on the ground truth $f^\star$: The Bayesian assumption is that $f^\star$ is drawn from the prior GP, whereas the frequentist assumption is that $f^\star$ is an element of a reproducing kernel Hilbert space $\mathcal{H}_k(\mathcal{X})$. The frequentist assumption holds uniformly for all functions $f^\star$ with $\|f^\star\|_k < \infty$, whereas the Bayesian assumption holds only under the Bayesian "belief" that $f^\star$ is drawn from the prior GP.

Interestingly, neither assumption encompasses the other. This is because if $f \sim \mathcal{GP}(0, k)$ then it can be shown that almost surely $\|f\|_k = \infty$, which implies that $f \notin \mathcal{H}_k(\mathcal{X})$ (Srinivas et al., 2010).

We remark that Theorem 9.7 holds also under the looser assumption that observations are perturbed by σ_{n}-sub-Gaussian noise (cf. Equation (A.39)) instead of Gaussian noise. The bound on γ_T from Equation (9.13) for the Matérn kernel does not yield sublinear regret when combined with the standard regret bound from Theorem 9.5, however, Whitehouse et al. (2024) show that the regret of GP-UCB is sublinear also in this case provided σ_{n}^2 is chosen carefully.

This concludes our discussion of the UCB algorithm. We have seen that its regret can be analyzed under both Bayesian and frequentist assumptions on $f^\star$.

9.3.2 *Improvement*

Another well-known family of methods is based on keeping track of a running optimum $\hat{f}_t$, and scoring points according to their improvement upon the running optimum. The *improvement* of $\boldsymbol{x}$ after round t is measured by

$$I_t(\boldsymbol{x}) \doteq (f(\boldsymbol{x}) - \hat{f}_t)_+ \tag{9.15}$$

where we use $(\cdot)_+$ to denote $\max\{0, \cdot\}$.

The *probability of improvement* (PI) picks the point that maximizes the probability to improve upon the running optimum,

$$\begin{aligned} \boldsymbol{x}_{t+1} &\doteq \arg\max_{\boldsymbol{x} \in \mathcal{X}} \mathbb{P}(I_t(\boldsymbol{x}) > 0 \mid \boldsymbol{x}_{1:t}, y_{1:t}) && (9.16) \\ &= \arg\max_{\boldsymbol{x} \in \mathcal{X}} \mathbb{P}(f(\boldsymbol{x}) > \hat{f}_t \mid \boldsymbol{x}_{1:t}, y_{1:t}) && (9.17) \end{aligned}$$

$$= \arg\max_{x \in \mathcal{X}} \Phi\left(\frac{\mu_t(x) - \hat{f}_t}{\sigma_t(x)}\right) \tag{9.18}$$

using linear transformations of Gaussians (1.78)

where Φ denotes the CDF of the standard normal distribution and we use that $f(x) \mid x_{1:t}, y_{1:t} \sim \mathcal{N}(\mu_t(x), \sigma_t^2(x))$. Probability of improvement tends to be biased in favor of exploitation, as it prefers points with large posterior mean and small posterior variance which is typically true "close" to the previously observed maximum $\hat{f}_t$.

Probability of improvement looks at *how likely* a point is to improve upon the running optimum. An alternative is to look at *how much* a point is expected to improve upon the running optimum. This acquisition function is called the *expected improvement* (EI),

$$x_{t+1} \doteq \arg\max_{x \in \mathcal{X}} \mathbb{E}[I_t(x) \mid x_{1:t}, y_{1:t}]\,. \tag{9.19}$$

Intuitively, EI seeks a large expected improvement (exploitation) while also preferring states with a large variance (exploration). Expected improvement yields the same regret bound as UCB (Nguyen et al., 2017).

The expected improvement acquisition function is often flat which makes it difficult to optimize in practice due to vanishing gradients. One approach addressing this is to instead optimize the logarithm of EI (Ament et al., 2024).

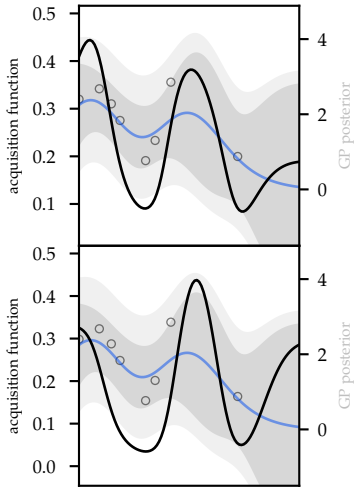


Figure 9.7: Plot of the PI and EI acquisition functions, respectively.

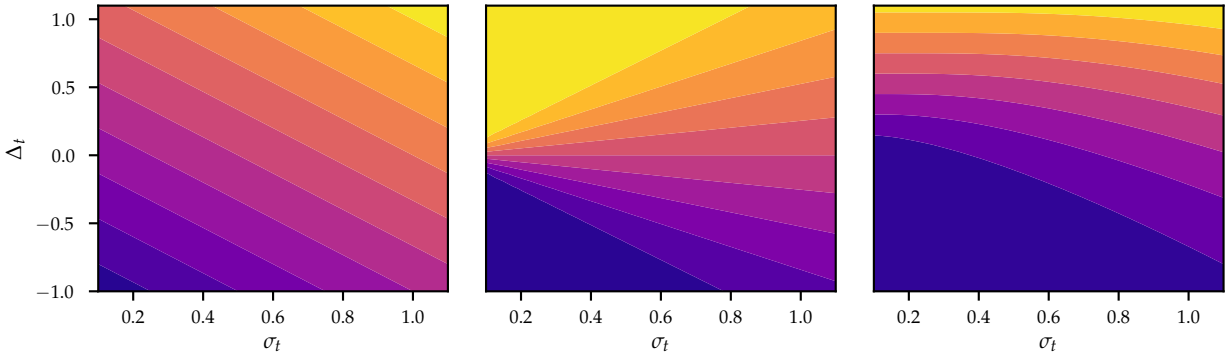


Figure 9.8: Contour lines of acquisition functions for varying $\Delta_t = \mu_t(x) - \hat{f}_t$ and σ_t. A brighter color corresponds to a larger acquisition function value. The first graph shows contour lines of UCB with $\beta_t = 0.75$, the second of PI, and the third of EI.

9.3.3 *Thompson Sampling*

We can also interpret the principle of *optimism in the face of uncertainty* in a slightly different way than we did with UCB (and EI). Suppose we select the next point according to the probability that it is optimal (assuming that the posterior distribution is an accurate representation of the uncertainty),

$$\pi(x \mid x_{1:t}, y_{1:t}) \doteq \mathbb{P}_{f \mid x_{1:t}, y_{1:t}}\left(f(x) = \max_{x'} f(x')\right) \tag{9.20}$$

$$\boldsymbol{x}_{t+1} \sim \pi(\cdot \mid \boldsymbol{x}_{1:t}, y_{1:t}). \tag{9.21}$$

This approach of sampling according to the *probability of maximality* π is called *probability matching*. Probability matching is exploratory as it prefers points with larger variance (as they automatically have a larger chance of being optimal), but at the same time exploitative as it effectively discards points with low posterior mean and low posterior variance. Unfortunately, it is generally difficult to compute π analytically given a posterior.

Instead, it is common to use a sampling-based approximation of π. Observe that the density π can be expressed as an expectation,

$$\pi(\boldsymbol{x} \mid \boldsymbol{x}_{1:t}, y_{1:t}) = \mathbb{E}_{f \mid \boldsymbol{x}_{1:t}, y_{1:t}}\big[\mathbb{1}\{f(\boldsymbol{x}) = \max_{\boldsymbol{x}'} f(\boldsymbol{x}')\}\big], \tag{9.22}$$

which we can approximate using Monte Carlo sampling (typically using a single sample),

$$\approx \mathbb{1}\{\tilde{f}_{t+1}(\boldsymbol{x}) = \max_{\boldsymbol{x}'} \tilde{f}_{t+1}(\boldsymbol{x}')\} \tag{9.23}$$

where $\tilde{f}_{t+1} \sim p(\cdot \mid \boldsymbol{x}_{1:t}, y_{1:t})$ is a sample from our posterior distribution. Observe that this approximation of π coincides with a point density at the maximizer of $\tilde{f}_{t+1}$.

The resulting algorithm is known as *Thompson sampling*. At time $t+1$, we sample a function $\tilde{f}_{t+1} \sim p(\cdot \mid \boldsymbol{x}_{1:t}, y_{1:t})$ from our posterior distribution. Then, we simply maximize $\tilde{f}_{t+1}$,

$$\boldsymbol{x}_{t+1} \doteq \arg\max_{\boldsymbol{x} \in \mathcal{X}} \tilde{f}_{t+1}(\boldsymbol{x}). \tag{9.24}$$

In many cases, the randomness in the realizations of $\tilde{f}_{t+1}$ is already sufficient to effectively trade exploration and exploitation. Similar regret bounds to those of UCB can also be established for Thompson sampling (Russo and Van Roy, 2016; Kandasamy et al., 2018).

9.3.4 *Information-Directed Sampling*

After having looked at multiple methods that aim to balance exploitation of the current posterior distribution over f for immediate returns with exploration to reduce uncertainty about f for future returns, we will next discuss a method that makes this tradeoff explicit.

Denoting the instantaneous regret of choosing $\boldsymbol{x}$ as $\Delta(\boldsymbol{x}) \doteq \max_{\boldsymbol{x}'} f^\star(\boldsymbol{x}') - f^\star(\boldsymbol{x})$ and by $I_t(\boldsymbol{x})$ some function capturing the "information gain" associated with observing $\boldsymbol{x}$ in iteration $t+1$, we can define the *information ratio*,

$$\Psi_t(\boldsymbol{x}) \doteq \frac{\Delta(\boldsymbol{x})^2}{I_t(\boldsymbol{x})}, \tag{9.25}$$

which was originally introduced by Russo and Van Roy (2016). Here exploitation reduces regret while exploration increases information gain, and hence, points $\boldsymbol{x}$ that minimize the information ratio are those that most effectively balance exploration and exploitation. We can make the key observation that the regret $\Delta(\cdot)$ decreases when $I_t(\cdot)$ decreases, as a small $I_t(\cdot)$ implies that the algorithm has already learned a lot about the function $f^\star$. The strength of this relationship is quantified by the information ratio:

Theorem 9.9 (Proposition 1 of Russo and Van Roy (2014) and theorem 8 of Kirschner and Krause (2018))**.** *For any iteration $T \geq 1$, let $\sum_{t=1}^{T} I_{t-1}(\boldsymbol{x}_t) \leq \gamma_T$ and suppose that $\Psi_{t-1}(\boldsymbol{x}_t) \leq \overline{\Psi}_T$ for all $t \in [T]$. Then, the cumulative regret is bounded by*

$$R_T \leq \sqrt{\gamma_T \overline{\Psi}_T T}. \tag{9.26}$$

Proof. By Equation (9.25), $r_t = \Delta(\boldsymbol{x}_t) = \sqrt{\Psi_{t-1}(\boldsymbol{x}_t) \cdot I_{t-1}(\boldsymbol{x}_t)}$. Hence,

$$\begin{aligned}
R_T &= \sum_{t=1}^{T} r_t \\
&= \sum_{t=1}^{T} \sqrt{\Psi_{t-1}(\boldsymbol{x}_t) \cdot I_{t-1}(\boldsymbol{x}_t)} \\
&\leq \sqrt{\sum_{t=1}^{T} \Psi_{t-1}(\boldsymbol{x}_t) \cdot \sum_{t=1}^{T} I_{t-1}(\boldsymbol{x}_t)} && \text{using the Cauchy-Schwarz inequality} \\
&\leq \sqrt{\gamma_T \overline{\Psi}_T T}. && \text{using the assumptions on } I_t(\cdot) \text{ and } \Psi_t(\cdot)
\end{aligned}$$

□

Example 9.10: How to measure "information gain"?

One possibility of measuring the "information gain" is

$$I_t(\boldsymbol{x}) \doteq \mathrm{I}(f_{\boldsymbol{x}}; y_{\boldsymbol{x}} \mid \boldsymbol{x}_{1:t}, y_{1:t}) \tag{9.27}$$

which — as you may recall — is precisely the marginal gain of the utility $I(S) = \mathrm{I}(\boldsymbol{f}_S; \boldsymbol{y}_S)$ we were studying in Chapter 8. In this case,

$$\begin{aligned}
\sum_{t=1}^{T} I_{t-1}(\boldsymbol{x}_t) &= \sum_{t=1}^{T} \mathrm{I}(f_{\boldsymbol{x}_t}; y_{\boldsymbol{x}_t} \mid \boldsymbol{x}_{1:t-1}, y_{1:t-1}) \\
&= \sum_{t=1}^{T} \mathrm{I}(\boldsymbol{f}_{\boldsymbol{x}_{1:T}}; y_{\boldsymbol{x}_t} \mid \boldsymbol{x}_{1:t-1}, y_{1:t-1}) && \text{using } \mathrm{I}(\mathbf{X}, \mathbf{Z}; \mathbf{Y}) \geq \mathrm{I}(\mathbf{X}; \mathbf{Y}) \text{ which follows from Equations (8.12) and (8.16) and is called } \textit{monotonicity of MI} \\
&= \mathrm{I}(\boldsymbol{f}_{\boldsymbol{x}_{1:T}}; \boldsymbol{y}_{\boldsymbol{x}_{1:T}}) && \text{by repeated application of Equation (8.16), also called the } \textit{chain rule of MI} \\
&\leq \gamma_T. && \text{by definition of } \gamma_T \text{ (9.10)}
\end{aligned}$$

The regret bound from Theorem 9.9 suggests an algorithm which in each iteration chooses the point which minimizes the information ratio (9.25). However, this is not possible since $\Delta(\cdot)$ is unknown due to its dependence on $f^\star$. Kirschner and Krause (2018) propose to use a surrogate to the regret which is based on the current model of $f^\star$,

$$\hat{\Delta}_t(\boldsymbol{x}) \doteq \max_{\boldsymbol{x}' \in \mathcal{X}} u_t(\boldsymbol{x}') - l_t(\boldsymbol{x}). \tag{9.28}$$

Here, $u_t(\boldsymbol{x}) \doteq \mu_t(\boldsymbol{x}) + \beta_{t+1}\sigma_t(\boldsymbol{x})$ and $l_t(\boldsymbol{x}) \doteq \mu_t(\boldsymbol{x}) - \beta_{t+1}\sigma_t(\boldsymbol{x})$ are the upper and lower confidence bounds of the confidence interval $\mathcal{C}_t(\boldsymbol{x})$ of $f^\star(\boldsymbol{x})$, respectively. Similarly to our discussion of UCB, we make the assumption that the sequence β_t is chosen "correctly" (cf. Equation (9.7)) so that the confidence interval is well-calibrated and $\Delta(\boldsymbol{x}) \leq \hat{\Delta}_t(\boldsymbol{x})$ with high probability. The resulting algorithm

$$\boldsymbol{x}_{t+1} \doteq \arg\min_{\boldsymbol{x} \in \mathcal{X}} \left\{ \widehat{\Psi}_t(\boldsymbol{x}) \doteq \frac{\hat{\Delta}_t(\boldsymbol{x})^2}{I_t(\boldsymbol{x})} \right\} \tag{9.29}$$

is known as *information-directed sampling* (IDS).

Theorem 9.11 (Regret of IDS, lemma 8 of Kirschner and Krause (2018))**.** (?) *Let $\beta_t(\delta)$ be chosen "correctly" for a fixed $\delta \in (0,1)$. Then, if the measure of information gain is $I_t(\boldsymbol{x}) = \mathrm{I}(f_{\boldsymbol{x}}; y_{\boldsymbol{x}} \mid \boldsymbol{x}_{1:t}, y_{1:t})$, with probability at least $1-\delta$, IDS has cumulative regret*

$$R_T = O\left(\beta_T(\delta)\sqrt{\gamma_T T}\right). \tag{9.30}$$

Regret bounds such as Theorem 9.11 can be derived also for different measures of information gain. For example, the argument of Problem 9.6 also goes through for the "greedy" measure

$$I_t(\boldsymbol{x}) \doteq \mathrm{I}(f_{\boldsymbol{x}_t^{\mathrm{UCB}}}; y_{\boldsymbol{x}} \mid \boldsymbol{x}_{1:t}, y_{1:t}) \tag{9.31}$$

which focuses exclusively on reducing the uncertainty at $\boldsymbol{x}_t^{\mathrm{UCB}}$ rather than globally. We compare the two measures of information gain in Figure 9.9. Observe that the acquisition function depends critically on the choice of $I_t(\cdot)$ and is less sensitive to the scaling of confidence intervals.

IDS trades exploitation and exploration by balancing the (exploitative) regret surrogate with a measure of information gain (such as those studied in Chapter 8) that is purely explorative. In this way, IDS can account for kinds of information which are not addressed by alternative algorithms such as UCB or EI (Russo and Van Roy, 2014): Depending on the measure of information gain, IDS can select points to obtain *indirect information* about other points or *cumulating information* that does not immediately lead to a higher reward but only when combined with subsequent observations. Moreover, IDS avoids selecting points which yield *irrelevant information*.

Problem 9.6

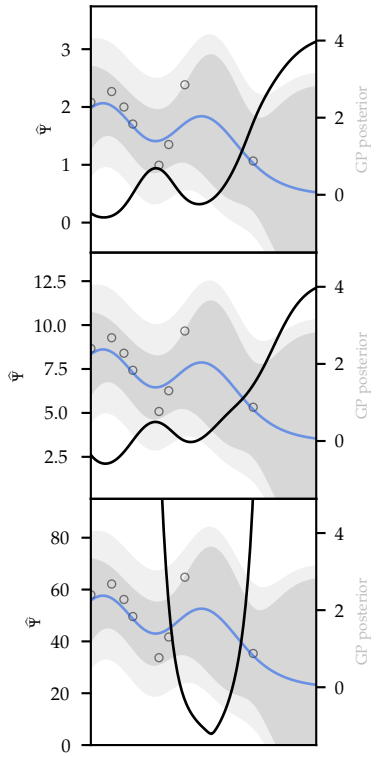


Figure 9.9: Plot of the surrogate information ratio $\widehat{\Psi}$: IDS selects its minimizer. The first two plots use the "global" information gain measure from Example 9.10 with $\beta = 0.25$ and $\beta = 0.5$, respectively. The third plot uses the "greedy" information gain measure from Equation (9.31) and $\beta = 1$.

9.3.5 *Probabilistic Inference*

As we mentioned before, computing the probability of maximality π is generally intractable. You can think of computing π as attempting to fully solve the probabilistic inference problem associated with determining the optimum of $f \mid \boldsymbol{x}_{1:t}, y_{1:t}$. In many cases, it is useful to determine the probability that a point is optimal under the current posterior distribution, and we will consider one particular example in the following.

Example 9.12: Maximizing recall

In domains such as molecular design we often use machine learning to screen candidates for further manual testing. The goal here is to suggest a small set E from a large domain of molecules $\mathcal{X}$, so that the probability of E containing the optimal molecule, i.e., the *recall*, is maximized. Note that this task is quite different from online Bayesian optimization, for example, in BO we get sequential feedback that we can use to decide which inputs to query next. Nevertheless, we will see in this section that both tasks turn out to be closely related.

Let us assume that the maximizer is unique almost surely, which with GPs is automatically the case if there are no same-mean, perfectly-correlated entries. The recall task is then intimately related to the task of determining the probability of maximality, since

$$\begin{aligned}\operatorname*{arg\,max}_{E \subseteq \mathcal{X} : |E| = k} \mathbb{P}_{f \mid \boldsymbol{x}_{1:t}, y_{1:t}} \left(\max_{\boldsymbol{x} \in E} f(\boldsymbol{x}) = \max_{\boldsymbol{x}'} f(\boldsymbol{x}') \right) \\ = \operatorname*{arg\,max}_{E \subseteq \mathcal{X} : |E| = k} \sum_{\boldsymbol{x} \in E} \pi(\boldsymbol{x} \mid \boldsymbol{x}_{1:t}, y_{1:t}).\end{aligned} \tag{9.32}$$

by noting that for $\boldsymbol{x} \neq \boldsymbol{y}$, the events $\{f(\boldsymbol{x}) = \max_{\boldsymbol{x}'} f(\boldsymbol{x}')\}$ and $\{f(\boldsymbol{y}) = \max_{\boldsymbol{x}'} f(\boldsymbol{x}')\}$ are disjoint

Thus, to obtain the recall-optimal candidates for further testing, we need to find the probability of maximality. However, as mentioned, computing the probability of maximality π is generally intractable.

Figure 9.10 depicts an example of a recall task, where the black line shows the optimal recall achieved by knowing the probability of maximality π exactly. LITE (Menet et al., 2025) is an almost-linear time approximation of π, whereas TS selects points via Thompson sampling and MEANS selects the points with the highest posterior mean. Selecting E according to probability of maximality achieves much higher recall than the other intuitive heuristics.

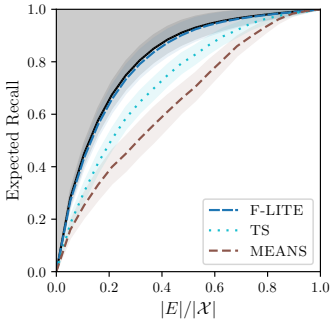


Figure 9.10: Menet et al. (2025) show in an experiment that selecting $E \subseteq \mathcal{X}$ according to the estimates of probability of maximality from LITE (here called F-LITE) is near-optimal. Intuitive heuristics such as Thompson sampling (TS) or selecting points with the highest posterior mean (MEANS) perform worse.

So how can we estimate the probability of maximality? LITE approximates π by

$$\begin{aligned}\pi(\boldsymbol{x} \mid \boldsymbol{x}_{1:t}, y_{1:t}) &= \mathbb{P}_{f|\boldsymbol{x}_{1:t},y_{1:t}}\left(f(\boldsymbol{x}) \geq \max_{\boldsymbol{x}'} f(\boldsymbol{x}')\right) \\ &\approx \mathbb{P}_{f|\boldsymbol{x}_{1:t},y_{1:t}}(f(\boldsymbol{x}) \geq \kappa^*) \qquad (9.33) \\ &= \Phi\left(\frac{\mu_t(\boldsymbol{x}) - \kappa^*}{\sigma_t(\boldsymbol{x})}\right) \qquad (9.34)\end{aligned}$$

with Φ denoting the CDF of the standard normal distribution and κ^* chosen such that the approximation of π integrates to 1, so that it is a valid distribution.

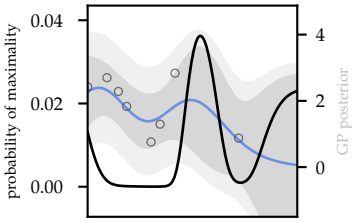


Figure 9.11: Plot of the probability of maximality as estimated by LITE.

Remarkably, LITE is intimately related to many of the BO methods we discussed in this chapter. First, and most obviously, Equation (9.33) looks similar to the PI acquisition function (9.17). But note that κ^* is not equal to the best observed value $\hat{f}_t$ as in PI, but instead typically larger. If $\kappa^* > \hat{f}_t$, then Equation (9.33) is more exploratory than PI: it puts additional emphasis on points with large posterior variance and comparatively less emphasis on points with large posterior mean, which we illustrate in Figure 9.11.

An insightful interpretation of LITE is that it balances two different kinds of exploration: *optimism in the face of uncertainty* and *entropy regularization*. To see this, let us define the following variational objective:

$$\mathcal{W}(\pi) \doteq \sum_{\boldsymbol{x} \in \mathcal{X}} \pi(\boldsymbol{x}) \cdot \Big(\underbrace{\mu_t(\boldsymbol{x})}_{\text{exploitation}} + \underbrace{\sqrt{2\,\mathrm{S}'(\pi(\boldsymbol{x}))} \cdot \sigma_t(\boldsymbol{x})}_{\text{exploration}} \Big) \qquad (9.35)$$

with the "quasi-surprise" $\mathrm{S}'(u) \doteq \frac{1}{2}(\phi(\Phi^{-1}(u))/u)^2$. The quasi-surprise $\mathrm{S}'(\cdot)$ behaves similarly to the surprise $-\ln(\cdot)$. In fact, their asymptotics coincide:

$$\mathrm{S}'(1) = 0 = -\ln(1) \quad \text{and} \quad \mathrm{S}'(u) \to -\ln(u) \text{ as } u \to 0^+.$$

The objective $\mathcal{W}$ is maximized for those probability distributions π that are concentrated around points with large mean $\mu_t(\boldsymbol{x})$ and points with large exploration bonus. We have seen that the uncertainty $\sigma_t(\boldsymbol{x})$ about $f(\boldsymbol{x})$ is the standard exploration bonus of UCB-style algorithms. In Equation (9.35), $\sigma_t(\boldsymbol{x})$ is weighted by the quasi-surprise, which acts as "entropy regularization": it increases the entropy of π by uniformly pushing $\pi(\boldsymbol{x})$ away from zero. You can think of entropy regularization as encouraging a different kind of exploration than optimism by not making deterministic decisions like in UCB.

Menet et al. (2025) show that LITE (9.34) is the solution to the varia-

tional problem (9.35) among valid probability distributions (?):

Problem 9.7

$$\arg\max_{\pi \in \Delta^{\mathcal{X}}} \mathcal{W}(\pi) = \Phi\left(\frac{\mu_t(\cdot) - \kappa^*}{\sigma_t(\cdot)}\right) \tag{9.36}$$

with κ^* such that the right-hand side sums to 1.[9] This indicates that LITE and Thompson sampling, which samples from probability of maximality, achieve exploration through two means:

1. **Optimism:** by preferring points with large uncertainty $\sigma_t(\boldsymbol{x})$ about the reward value $f(\boldsymbol{x})$.
2. **Decision uncertainty:** by assigning some probability mass to all $\boldsymbol{x}$, that is, by remaining uncertain about which $\boldsymbol{x}$ is the maximizer.

In our discussion of balancing exploration and exploitation in reinforcement learning, we will return to this dichotomy of exploration strategies.

[9] $\Delta^{\mathcal{X}}$ is the probability simplex on $\mathcal{X}$.

Remark 9.13: But why is "exploration" useful for recall?

Let us pause for a moment and reflect on why this interpretation of LITE is remarkable. This interpretation shows that LITE is more "exploratory" than simply taking the highest posterior means. However, the recall task (9.32) differs from the standard BO task in that we do not collect any further observations, which we may use downstream to make better decisions. In the recall task, we are only interested in having the best shot at including the maximizer in the set E, without obtaining any further information or making any subsequent decisions. At first sight, this seems to suggest that we should be as "exploitative" as possible. Then how can it be that "exploration" is useful?

We will explore this question with the following example: You are observing a betting game. When placing a bet, players can either place a safe bet ("playing safe") or a risky bet ("playing risky"). You now have to place a bet on their bets, and estimate which of the players will win the most money: those that play safe or those that play risky? Note that you do not care whether your guess ends up in 2nd place or last place among all players — you only care about whether your guess wins. That is, you are interested in *recalling* the best player, not in "ranking" the players.

Consider three players: one that plays safe and two that play risky. Suppose that the safe bet has payoff $S = 1$ while each risky bet has payoff $R \sim \mathcal{N}(0, 100)$. In expectation, the safe player will win the most money. However, one can see with just a little bit of algebra that the probability of either of the risky players winning the most money is $\approx$35%, whereas the safe player only wins with

probability $\approx 29\%$ (?). That is, it is in fact *optimal* to bet on either of the risky players since the best player might have vastly outperformed their expected winnings, and performed closer to their upper confidence bound. In summary, maximizing recall requires us to be "exploratory" since it is likely that the optimum among inputs is one that has performed better than expected, not simply the one with the highest expected performance.

Problem 9.8

Discussion

In this chapter, we have explored the exploration-exploitation dilemma in the context of optimizing black-box functions. To this end, we have explored various methods to balance exploration and exploitation. While computing the precise probability of maximality is generally intractable, we found that it can be understood approximately as balancing two sources of exploration: optimism in the face of uncertainty and entropy regularization.

In the following chapters, we will begin to discuss stateful settings where the black-box optimization task is known as "reinforcement learning". Naturally, we will see that the exploration-exploitation dilemma is also a central challenge in reinforcement learning, and we will revisit many of the concepts we have discussed in this chapter.

Optional Readings

- Srinivas, Krause, Kakade, and Seeger (2010).
  Gaussian process optimization in the bandit setting: No regret and experimental design.
- Golovin, Solnik, Moitra, Kochanski, Karro, and Sculley (2017).
  Google vizier: A service for black-box optimization.
- Romero, Krause, and Arnold (2013).
  Navigating the protein fitness landscape with Gaussian processes.
- Chowdhury and Gopalan (2017).
  On kernelized multi-armed bandits.

Problems

9.1. Convergence to the static optimum.

Show that any algorithm where $\lim_{t\to\infty} f^\star(\boldsymbol{x}_t)$ exists achieves sublinear regret if and only if it converges to the static optimum, that is,

$$\lim_{t\to\infty} f^\star(\boldsymbol{x}_t) = \max_{\boldsymbol{x}} f^\star(\boldsymbol{x}). \tag{9.37}$$

Hint: Use that if a sequence a_n converges to a as $n \to \infty$, then we have for the sequence

$$b_n \doteq \frac{1}{n} \sum_{i=1}^{n} a_i \tag{9.38}$$

that $\lim_{n\to\infty} b_n = a$. *This is also known as the* Cesàro mean.

9.2. Bayesian confidence intervals.

In this exercise, we derive Theorem 9.4.

1. For fixed $t \geq 1$ and $\boldsymbol{x} \in \mathcal{X}$, prove

$$\mathbb{P}(f^\star(\boldsymbol{x}) \notin \mathcal{C}_t(\boldsymbol{x}) \mid \boldsymbol{x}_{1:t-1}, y_{1:t-1}) \leq e^{-\beta_t^2/2}. \tag{9.39}$$

   Hint: Bound $\mathbb{P}(Z > c)$ *for* $Z \sim \mathcal{N}(0,1)$ *and* $c > 0$.
2. Prove Theorem 9.4.

9.3. Regret of GP-UCB.

To develop some intuition, we will derive Theorem 9.5.

1. Show that if Equation (9.7) holds, then for a fixed $t \geq 1$ the instantaneous regret r_t is bounded by $2\beta_t \sigma_{t-1}(\boldsymbol{x}_t)$.
2. Let $S_T \doteq \{\boldsymbol{x}_t\}_{t=1}^T$, and define $\boldsymbol{f}_T \doteq \boldsymbol{f}_{S_T}$ and $\boldsymbol{y}_T \doteq \boldsymbol{y}_{s_T}$. Prove

$$\mathrm{I}(\boldsymbol{f}_T; \boldsymbol{y}_T) = \frac{1}{2} \sum_{t=1}^{T} \log\left(1 + \frac{\sigma_{t-1}^2(\boldsymbol{x}_t)}{\sigma_{\mathrm{n}}^2}\right). \tag{9.40}$$

3. Combine (1) and (2) to show Theorem 9.5. We assume w.l.o.g. that the sequence $\{\beta_t\}_t$ is monotonically increasing.
   Hint: If $s \in [0, M]$ *for some* $M > 0$ *then* $s \leq C \cdot \log(1+s)$ *with* $C \doteq M/\log(1+M)$.

9.4. Sublinear regret of GP-UCB for a linear kernel.

Assume that $f^\star \sim \mathcal{GP}(0, k)$ where k is the linear kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}'.$$

In addition, we assume that for any $\boldsymbol{x} \in \mathcal{X}$, $\|\boldsymbol{x}\|_2 \leq 1$. Moreover, recall that the points in a finite set $S \subseteq \mathcal{X}$ can be written in a matrix form (the "design matrix") which we denote by $\boldsymbol{X}_S \in \mathbb{R}^{d \times |S|}$.

1. Prove that $\gamma_T = O(d \log T)$.
2. Deduce from (1) and Theorem 9.5 that $\lim_{T\to\infty} R_T/T = 0$.

9.5. Closed-form expected improvement.

Let us denote the acquisition function of EI from Equation (9.19) by $\mathrm{EI}_t(\boldsymbol{x})$. In this exercise, we derive a closed-form expression.

1. Show that

$$\mathrm{EI}_t(\boldsymbol{x}) = \int_{(\hat{f}_t - \mu_t(\boldsymbol{x}))/\sigma_t(\boldsymbol{x})}^{+\infty} (\mu_t(\boldsymbol{x}) + \sigma_t(\boldsymbol{x})\varepsilon - \hat{f}_t) \cdot \phi(\varepsilon)\, d\varepsilon \tag{9.41}$$

   where ϕ is the PDF of the univariate standard normal distribution (1.5).
   Hint: Reparameterize the posterior distribution

$$f(\boldsymbol{x}) \mid \boldsymbol{x}_{1:t}, y_{1:t} \sim \mathcal{N}(\mu_t(\boldsymbol{x}), \sigma_t^2(\boldsymbol{x}))$$

   using a standard normal distribution.
2. Using the above expression, show that

$$\mathrm{EI}_t(\boldsymbol{x}) = (\mu_t(\boldsymbol{x}) - \hat{f}_t)\Phi\left(\frac{\mu_t(\boldsymbol{x}) - \hat{f}_t}{\sigma_t(\boldsymbol{x})}\right) + \sigma_t(\boldsymbol{x})\phi\left(\frac{\mu_t(\boldsymbol{x}) - \hat{f}_t}{\sigma_t(\boldsymbol{x})}\right) \tag{9.42}$$

   where $\Phi(u) \doteq \int_{-\infty}^{u} \phi(\varepsilon)\, d\varepsilon$ denotes the CDF of the standard normal distribution.

Note that the first term of Equation (9.42) encourages exploitation while the second term encourages exploration. EI can be seen as a special case of UCB where the confidence bounds are scaled depending on $\boldsymbol{x}$:

$$\beta_t = \phi(z_t(\boldsymbol{x}))/\Phi(z_t(\boldsymbol{x}))$$

for $z_t(\boldsymbol{x}) \doteq (\mu_t(\boldsymbol{x}) - \hat{f}_t)/\sigma_t(\boldsymbol{x})$.

9.6. Regret of IDS.

We derive Theorem 9.11.

1. Prove that for all $t \geq 1$, $\hat{\Delta}_t(\boldsymbol{x}_t^{\mathrm{UCB}}) \leq 2\beta_{t+1}\sigma_t(\boldsymbol{x}_t^{\mathrm{UCB}})$ where we denote by $\boldsymbol{x}_t^{\mathrm{UCB}} \doteq \arg\max_{\boldsymbol{x}\in\mathcal{X}} u_t(\boldsymbol{x})$ the point maximizing the upper confidence bound after iteration t.
2. Using (1) and the assumption on I_t, bound $\widehat{\Psi}_t(\boldsymbol{x}_t^{\mathrm{UCB}})$.
   Hint: You may find the hint of Problem 9.3 (3) useful.
3. Complete the proof of Theorem 9.11.

9.7. Variational form of LITE.

Let $|\mathcal{X}| < \infty$ be such that we can write $\mathcal{W}$ as an objective function over the probability simplex $\Delta^{|\mathcal{X}|} \subset \mathbb{R}^{|\mathcal{X}|}$. Derive Equation (9.36).

Hint: First show that $\mathcal{W}(\cdot)$ is concave (i.e., minimizing $-\mathcal{W}(\cdot)$ is a convex optimization problem) and then use Lagrange multipliers to find the optimum.

9.8. Finding the winning player.

Consider the betting game from Remark 9.13 with two risky players, $R_1 \sim \mathcal{N}(0, 100)$ and $R_2 \sim \mathcal{N}(0, 100)$, and one safe player $S = 1$. Prove that the individual probability of any of the risky players winning the most money is larger than the winning probability of the safe player. We assume that players payoffs are mutually independent.

Hint: Compute the CDF of $R \doteq \max\{R_1, R_2\}$.

10

Markov Decision Processes

We will now turn to the topic of probabilistic planning. Planning deals with the problem of deciding which action an agent should play in a (stochastic) environment.[1] A key formalism for probabilistic planning in *known* environments are so-called Markov decision processes. Starting from the next chapter, we will look at reinforcement learning, which extends probabilistic planning to unknown environments.

[1] An environment is *stochastic* as opposed to deterministic, when the outcome of actions is random.

Consider the setting where we have a sequence of states $(X_t)_{t\in\mathbb{N}_0}$ similarly to Markov chains. But now, the next state X_{t+1} of an agent does not only depend on the previous state X_t but also depends on the last action A_t of this agent.

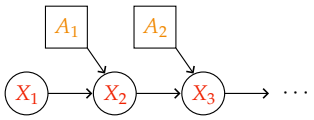


Figure 10.1: Directed graphical model of a Markov decision process with hidden states X_t and actions A_t.

Definition 10.1 ((Finite) Markov decision process, MDP)**.** A *(finite) Markov decision process* is specified by

- a (finite) set of *states* $X \doteq \{1, \dots, n\}$,
- a (finite) set of *actions* $A \doteq \{1, \dots, m\}$,
- *transition probabilities*

$$p(x' \mid x, a) \doteq \mathbb{P}\big(X_{t+1} = x' \mid X_t = x, A_t = a\big) \tag{10.1}$$

  which is also called the *dynamics model*, and
- a *reward function* $r : X \times A \to \mathbb{R}$ which maps the current state x and an action a to some reward.

The reward function may also depend on the next state x', however, we stick to the above model for simplicity. Also, the reward function can be random with mean r. Observe that r induces the sequence of rewards $(R_t)_{t\in\mathbb{N}_0}$, where

$$R_t \doteq r(X_t, A_t), \tag{10.2}$$

which is sometimes used in the literature instead of r.

Crucially, we assume the dynamics model p and the reward function r to be known. That is, we operate in a known environment. For now,

we also assume that the environment is *fully observable*. In other words, we assume that our agent knows its current state. In Section 10.4, we discuss how this method can be extended to the partially observable setting.

Our fundamental objective is to learn how the agent should behave to optimize its reward. In other words, given its current state, the agent should decide (optimally) on the action to play. Such a decision map — whether optimal or not — is called a policy.

Definition 10.2 (Policy)**.** A *policy* is a function that maps each state $x \in X$ to a probability distribution over the actions. That is, for any $t > 0$,

$$\pi(a \mid x) \doteq \mathbb{P}(A_t = a \mid X_t = x). \tag{10.3}$$

In other words, a policy assigns to each action $a \in A$, a probability of being played given the current state $x \in X$.

We assume that policies are stationary, that is, do not change over time.

Remark 10.3: Stochastic policies

We will see later in this chapter that in fully observable environments optimal policies are always deterministic. Thus, there is no need to consider stochastic policies in the context of Markov decision processes. For this chapter, you can think of a policy π simply as a deterministic mapping $\pi : X \to A$ from current state to played action. In the context of reinforcement learning, we will later see in Section 12.5.1 that randomized policies are important in trading exploration and exploitation.

Observe that a policy induces a Markov chain $(X_t^\pi)_{t \in \mathbb{N}_0}$ with transition probabilities,

$$p^\pi(x' \mid x) \doteq \mathbb{P}\big(X_{t+1}^\pi = x' \mid X_t^\pi = x\big) = \sum_{a \in A} \pi(a \mid x) p(x' \mid x, a). \tag{10.4}$$

This is crucial: if our agent follows a fixed policy (i.e., decision-making protocol) then the evolution of the process is described fully by a Markov chain.

As mentioned, we want to maximize the reward. There are many models of calculating a score from the infinite sequence of rewards $(R_t)_{t \in \mathbb{N}_0}$. For the purpose of our discussion of Markov decision processes and reinforcement learning, we will focus on a very common reward called discounted payoff.

Definition 10.4 (Discounted payoff)**.** The *discounted payoff* (also called

discounted total reward) from time t is defined as the random variable,

$$G_t \doteq \sum_{m=0}^{\infty} \gamma^m R_{t+m} \tag{10.5}$$

where $\gamma \in [0, 1)$ is the *discount factor*.

Remark 10.5: Other reward models

Other well-known methods for combining rewards into a score are

$$\underset{\text{(instantaneous)}}{G_t \doteq R_t} \quad , \quad \underset{\text{(finite-horizon)}}{G_t \doteq \sum_{m=0}^{T-1} R_{t+m}}, \quad \text{and} \quad \underset{\text{(mean payoff)}}{G_t \doteq \liminf_{T\to\infty} \frac{1}{T} \sum_{m=0}^{T} R_{t+m}}.$$

The methods that we will discuss can also be analyzed using these or other alternative reward models.

We now want to understand the effect of the starting state and initial action on our optimization objective G_t. To analyze this, it is common to use the following two functions:

Definition 10.6 (State value function)**.** The *state value function*,[2]

$$v_t^\pi(x) \doteq \mathbb{E}_\pi[G_t \mid X_t = x], \tag{10.7}$$

measures the average discounted payoff from time t starting from state $x \in X$.

[2] Recall that following a fixed policy π induces a Markov chain $(X_t^\pi)_{t\in\mathbb{N}_0}$. We define

$$\mathbb{E}_\pi[\cdot] \doteq \mathbb{E}_{(X_t^\pi)_{t\in\mathbb{N}_0}}[\cdot] \tag{10.6}$$

as an expectation over all possible sequences of states $(x_t)_{t\in\mathbb{N}_0}$ within this Markov chain.

Definition 10.7 (State-action value function)**.** The *state-action value function* (also called *Q-function*),

$$\begin{aligned} q_t^\pi(x, a) &\doteq \mathbb{E}_\pi[G_t \mid X_t = x, A_t = a] && (10.8)\\ &= r(x, a) + \gamma \sum_{x'\in X} p(x' \mid x, a) \cdot v_{t+1}^\pi(x'), && (10.9) \end{aligned}$$

by expanding the defition of the discounted payoff (10.5); corresponds to one step in the induced Markov chain

measures the average discounted payoff from time t starting from state $x \in X$ and with playing action $a \in A$. In other words, it combines the immediate return with the value of the next states.

Note that both $v_t^\pi(x)$ and $q_t^\pi(x, a)$ are deterministic scalar-valued functions. Because we assumed stationary dynamics, rewards, and policies, the discounted payoff starting from a given state x will be independent of the start time t. Thus, we write $v^\pi(x) \doteq v_0^\pi(x)$ and $q^\pi(x, a) \doteq q_0^\pi(x, a)$ without loss of generality.

10.1 Bellman Expectation Equation

Let us now see how we can compute the value function,

$$v^\pi(x) = \mathbb{E}_\pi[G_0 \mid X_0 = x]$$

using the definition of the value function (10.7)

$$= \mathbb{E}_\pi\left[\sum_{m=0}^{\infty} \gamma^m R_m \;\middle|\; X_0 = x\right]$$ using the definition of the discounted payoff (10.5)

$$= \mathbb{E}_\pi\left[\gamma^0 R_0 \;\middle|\; X_0 = x\right] + \gamma \mathbb{E}_\pi\left[\sum_{m=0}^{\infty} \gamma^m R_{m+1} \;\middle|\; X_0 = x\right]$$ using linearity of expectation (1.20)

$$= r(x, \pi(x)) + \gamma \mathbb{E}_{x'}\left[\mathbb{E}_\pi\left[\sum_{m=0}^{\infty} \gamma^m R_{m+1} \;\middle|\; X_1 = x'\right] \;\middle|\; X_0 = x\right]$$ by simplifying the first expectation and conditioning the second expectation on X_1

$$= r(x, \pi(x)) + \gamma \sum_{x' \in X} p(x' \mid x, \pi(x)) \mathbb{E}_\pi\left[\sum_{m=0}^{\infty} \gamma^m R_{m+1} \;\middle|\; X_1 = x'\right]$$ expanding the expectation on X_1 and using conditional independence of the discounted payoff of X_0 given X_1

$$= r(x, \pi(x)) + \gamma \sum_{x' \in X} p(x' \mid x, \pi(x)) \mathbb{E}_\pi\left[\sum_{m=0}^{\infty} \gamma^m R_m \;\middle|\; X_0 = x'\right]$$ shifting the start time of the discounted payoff using stationarity

$$= r(x, \pi(x)) + \gamma \sum_{x' \in X} p(x' \mid x, \pi(x)) \mathbb{E}_\pi\left[G_0 \mid X_0 = x'\right]$$ using the definition of the discounted payoff (10.5)

$$= r(x, \pi(x)) + \gamma \sum_{x' \in X} p(x' \mid x, \pi(x)) \cdot v^\pi(x'). \tag{10.10}$$ using the definition of the value function (10.7)

$$= r(x, \pi(x)) + \gamma \mathbb{E}_{x' \mid x, \pi(x)}\left[v^\pi(x')\right]. \tag{10.11}$$ interpreting the sum as an expectation

This equation is known as the *Bellman expectation equation*, and it shows a recursive dependence of the value function on itself. The intuition is clear: the value of the current state corresponds to the reward from the next action plus the discounted sum of all future rewards obtained from the subsequent states.

For stochastic policies, the above calculation can be extended to yield,

$$v^\pi(x) = \sum_{a \in A} \pi(a \mid x)\left(r(x, a) + \gamma \sum_{x' \in X} p(x' \mid x, a) v^\pi(x')\right) \tag{10.12}$$

$$= \mathbb{E}_{a \sim \pi(x)}\left[r(x, a) + \gamma \mathbb{E}_{x' \mid x, a}\left[v^\pi(x')\right]\right] \tag{10.13}$$

$$= \mathbb{E}_{a \sim \pi(x)}[q^\pi(x, a)]. \tag{10.14}$$

For stochastic policies, by also conditioning on the first action, one can obtain an analogous equation for the state-action value function,

$$q^\pi(x, a) = r(x, a) + \gamma \sum_{x' \in X} p(x' \mid x, a) \sum_{a' \in A} \pi(a' \mid x') q^\pi(x', a') \tag{10.15}$$

$$= r(x, a) + \gamma \mathbb{E}_{x' \mid x, a} \mathbb{E}_{a' \sim \pi(x')}\left[q^\pi(x', a')\right]. \tag{10.16}$$

Note that it does not make sense to consider a similar recursive formula for the state-action value function in the setting of deterministic policies as the action played when in state $x \in X$ is uniquely determined as $\pi(x)$. In particular,

$$v^\pi(x) = q^\pi(x, \pi(x)). \tag{10.17}$$

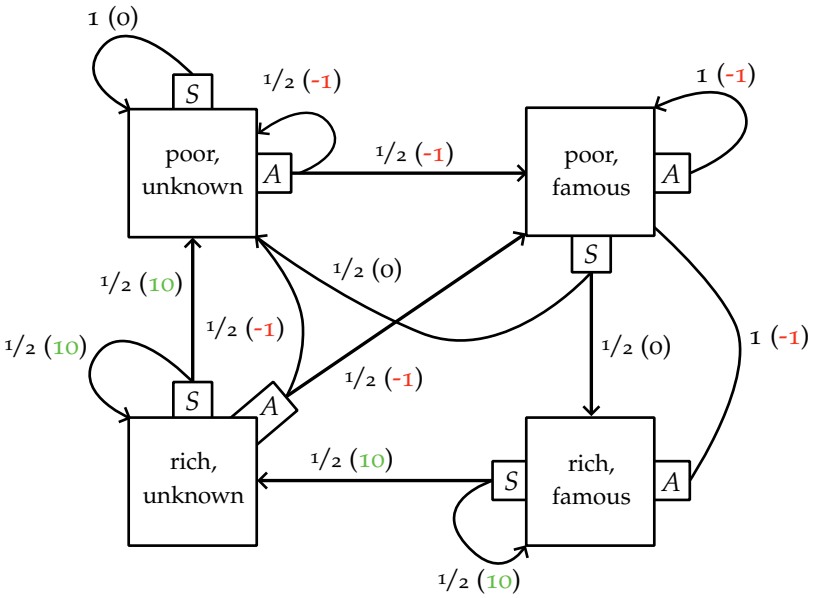


Figure 10.2: Example of an MDP, which we study in Problem 10.1. Suppose you are building a company. The shown MDP models "how to become rich and famous". Here, the action S is short for *saving* and the action A is short for *advertising*.

Suppose you begin by being "poor and unknown". Then, the greedy action (i.e., the action maximizing instantaneous reward) is to save. However, within this simplified environment, saving when you are poor and unknown means that you will remain poor and unknown forever. As the potential rewards in other states are substantially larger, this simple example illustrates that following the greedy choice is generally not optimal.

The example is adapted from Andrew Moore's lecture notes on MDPs (Moore, 2002).

10.2 *Policy Evaluation*

Bellman's expectation equation tells us how we can find the value function v^π of a fixed policy π using a system of linear equations! Using,

$$\boldsymbol{v}^\pi \doteq \begin{bmatrix} v^\pi(1) \\ \vdots \\ v^\pi(n) \end{bmatrix}, \quad \boldsymbol{r}^\pi \doteq \begin{bmatrix} r(1,\pi(1)) \\ \vdots \\ r(n,\pi(n)) \end{bmatrix}, \quad \text{and}$$
$$\boldsymbol{P}^\pi \doteq \begin{bmatrix} p(1 \mid 1,\pi(1)) & \cdots & p(n \mid 1,\pi(1)) \\ \vdots & \ddots & \vdots \\ p(1 \mid n,\pi(n)) & \cdots & p(n \mid n,\pi(n)) \end{bmatrix} \tag{10.18}$$

and a little bit of linear algebra, the Bellman expectation equation (10.10) is equivalent to

$$\boldsymbol{v}^\pi = \boldsymbol{r}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{v}^\pi \tag{10.19}$$
$$\iff \quad (\boldsymbol{I} - \gamma \boldsymbol{P}^\pi)\boldsymbol{v}^\pi = \boldsymbol{r}^\pi$$
$$\iff \quad \boldsymbol{v}^\pi = (\boldsymbol{I} - \gamma \boldsymbol{P}^\pi)^{-1}\boldsymbol{r}^\pi. \tag{10.20}$$

Solving this linear system of equations (i.e., performing matrix inversion) takes cubic time in the size of the state space.

10.2.1 Fixed-point Iteration

To obtain an (approximate) solution of v^π, we can use that it is the unique fixed-point of the affine mapping $\boldsymbol{B}^\pi : \mathbb{R}^n \to \mathbb{R}^n$,

$$\boldsymbol{B}^\pi \boldsymbol{v} \doteq \boldsymbol{r}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{v}. \tag{10.21}$$

Using this fact (which we will prove in just a moment), we can use fixed-point iteration of $\boldsymbol{B}^\pi$.

Algorithm 10.8: Fixed-point iteration

```
initialize v^π (e.g., as 0)
for t = 1 to T do
    v^π ← B^π v^π = r^π + γ P^π v^π
```

Fixed-point iteration has computational advantages, for example, for sparse transitions.

Theorem 10.9. *v^π is the unique fixed-point of $\boldsymbol{B}^\pi$.*

Proof. It is immediate from Bellman's expectation equation (10.19) and the definition of $\boldsymbol{B}^\pi$ (10.21) that v^π is a fixed-point of $\boldsymbol{B}^\pi$. To prove uniqueness, we will show that $\boldsymbol{B}^\pi$ is a contraction.

Remark 10.10: Contractions
A *contraction* is a concept from topology. In a Banach space $(\mathcal{X}, \|\cdot\|)$ (a metric space with a norm), $f : \mathcal{X} \to \mathcal{X}$ is a contraction iff there exists a $k < 1$ such that

$$\|f(\boldsymbol{x}) - f(\boldsymbol{y})\| \leq k \cdot \|\boldsymbol{x} - \boldsymbol{y}\| \tag{10.22}$$

for any $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$. By the *Banach fixed-point theorem*, a contraction admits a unique fixed-point. Intuitively, by iterating the function f, the distance to any fixed-point shrinks by a factor k in each iteration, hence, converges to 0. As we cannot converge to multiple fixed-points simultaneously, the fixed-point of a contraction f must be unique.

Let $\boldsymbol{v} \in \mathbb{R}^n$ and $\boldsymbol{v}' \in \mathbb{R}^n$ be arbitrary initial guesses. We use the L_∞ space,[3]

$$\begin{aligned}
\left\|\boldsymbol{B}^\pi \boldsymbol{v} - \boldsymbol{B}^\pi \boldsymbol{v}'\right\|_\infty &= \left\|\boldsymbol{r}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{v} - \boldsymbol{r}^\pi - \gamma \boldsymbol{P}^\pi \boldsymbol{v}'\right\|_\infty \\
&= \gamma \left\|\boldsymbol{P}^\pi (\boldsymbol{v} - \boldsymbol{v}')\right\|_\infty \\
&\leq \gamma \max_{\boldsymbol{x} \in X} \sum_{x' \in X} p(x' \mid x, \pi(x)) \cdot \left|\boldsymbol{v}(x') - \boldsymbol{v}'(x')\right|.
\end{aligned}$$

using the definition of $\boldsymbol{B}^\pi$ (10.21)

using the definition of the L_∞ norm (10.23), expanding the multiplication, and using $\left|\sum_i a_i\right| \leq \sum_i |a_i|$

[3] The L_∞ *norm* (also called *supremum norm*) is defined as

$$\|\boldsymbol{x}\|_\infty \doteq \max_i |\boldsymbol{x}(i)|. \tag{10.23}$$

$$\leq \gamma \left\| \boldsymbol{v} - \boldsymbol{v}' \right\|_\infty . \tag{10.24}$$

using $\sum_{x' \in X} p(x' \mid x, \pi(x)) = 1$ and $|\boldsymbol{v}(x') - \boldsymbol{v}'(x')| \leq \|\boldsymbol{v} - \boldsymbol{v}'\|_\infty$

Thus, by Equation (10.22), $\boldsymbol{B}^\pi$ is a contraction and by Banach's fixed-point theorem $\boldsymbol{v}^\pi$ is its unique fixed-point. □

Let $\boldsymbol{v}_t^\pi$ be the value function estimate after t iterations. Then, we have for the convergence of fixed-point iteration,

$$\begin{aligned} \|\boldsymbol{v}_t^\pi - \boldsymbol{v}^\pi\|_\infty &= \left\| \boldsymbol{B}^\pi \boldsymbol{v}_{t-1}^\pi - \boldsymbol{B}^\pi \boldsymbol{v}^\pi \right\|_\infty && \text{using the update rule of fixed-point iteration and } \boldsymbol{B}^\pi \boldsymbol{v}^\pi = \boldsymbol{v}^\pi \\ &\leq \gamma \left\| \boldsymbol{v}_{t-1}^\pi - \boldsymbol{v}^\pi \right\|_\infty && \text{using (10.24)} \\ &= \gamma^t \left\| \boldsymbol{v}_0^\pi - \boldsymbol{v}^\pi \right\|_\infty . && \text{by induction} \end{aligned} \tag{10.25}$$

This shows that fixed-point iteration converges to $\boldsymbol{v}^\pi$ exponentially fast.

10.3 Policy Optimization

Recall that our goal was to find an optimal policy,

$$\pi^\star \doteq \arg\max_\pi \mathbb{E}_\pi[G_0]. \tag{10.26}$$

We can alternatively characterize an optimal policy as follows: We define a partial ordering over policies by

$$\pi \geq \pi' \iff v^\pi(x) \geq v^{\pi'}(x) \quad (\forall x \in X). \tag{10.27}$$

$\pi^\star$ is then simply a policy which is maximal according to this partial ordering.

It follows that all optimal policies have identical value functions. Subsequently, we use $v^\star \doteq v^{\pi^\star}$ and $q^\star \doteq q^{\pi^\star}$ to denote the state value function and state-action value function arising from an optimal policy, respectively. As an optimal policy maximizes the value of each state, we have that

$$v^\star(x) = \max_\pi v^\pi(x), \quad q^\star(x, a) = \max_\pi q^\pi(x, a). \tag{10.28}$$

Simply optimizing over each policy is not a good idea as there are m^n deterministic policies in total. It turns out that we can do much better.

10.3.1 Greedy Policies

Consider a policy that acts greedily according to the immediate return. It is fairly obvious that this policy will not perform well because the agent might never get to high-reward states. But what if someone could tell us not just the immediate return, but the long-term value of

the states our agent can reach in a single step? If we knew the value of each state our agent can reach, then we can simply pick the action that maximizes the expected value. We will make this approach precise in the next section.

This thought experiment suggests the definition of a greedy policy with respect to a value function.

Definition 10.11 (Greedy policy)**.** The *greedy policy* with respect to a state-action value function q is defined as

$$\pi_q(x) \doteq \arg\max_{a \in A} q(x, a). \tag{10.29}$$

Analogously, we define the *greedy policy* with respect to a state value function v,

$$\pi_v(x) \doteq \arg\max_{a \in A} r(x, a) + \gamma \sum_{x' \in X} p(x' \mid x, a) \cdot v(x'). \tag{10.30}$$

We can use v and q interchangeably (?). **Problem 10.2**

10.3.2 *Bellman Optimality Equation*

Observe that following the greedy policy π_v, will lead us to a new value function v^{π_v}. With respect to this value function, we can again obtain a greedy policy, of which we can then obtain a new value function. In this way, the correspondence between greedy policies and value functions induces a cyclic dependency, which is visualized in Figure 10.3.

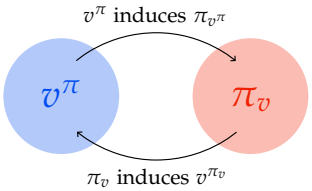


Figure 10.3: Cyclic dependency between **value function** and **greedy policy**.

It turns out that the optimal policy $\pi^\star$ is a fixed-point of this dependency. This is made precise by the following theorem.

Theorem 10.12 (Bellman's theorem)**.** *A policy $\pi^\star$ is optimal iff it is greedy with respect to its own value function. In other words, $\pi^\star$ is optimal iff $\pi^\star(x)$ is a distribution over the set* $\arg\max_{a \in A} q^\star(x, a)$.

In particular, if for every state there is a unique action that maximizes the state-action value function, the policy $\pi^\star$ is deterministic and unique,

$$\pi^\star(x) = \arg\max_{a \in A} q^\star(x, a). \tag{10.31}$$

Proof. It is a direct consequence of Equation (10.28) that a policy is optimal iff it is greedy with respect to $q^\star$. □

This theorem confirms our intuition from the previous section that greedily following an optimal value function is itself optimal. In particular, Bellman's theorem shows that there always exists an optimal policy which is deterministic and stationary.

We have seen, that $\pi^\star$ is a fixed-point of greedily picking the best action according to its state-action value function. The converse is also true:

Corollary 10.13. *The optimal value functions $v^\star$ and $q^\star$ are a fixed-point of the so-called* Bellman update*,*

$$v^\star(x) = \max_{a \in A} q^\star(x, a), \tag{10.32}$$

$$= \max_{a \in A} r(x, a) + \gamma \mathbb{E}_{x'|x,a}\left[v^\star(x')\right] \tag{10.33}$$

using the definition of the q-function (10.9)

$$q^\star(x, a) = r(x, a) + \gamma \mathbb{E}_{x'|x,a}\left[\max_{a' \in A} q^\star(x', a')\right]. \tag{10.34}$$

Proof. It follows from Equation (10.14) that

$$v^\star(x) = \mathbb{E}_{a \sim \pi^\star(x)}[q^\star(x, a)]. \tag{10.35}$$

Thus, as $\pi^\star$ is greedy with respect to $q^\star$, $v^\star(x) = \max_{a \in A} q^\star(x, a)$.

Equation (10.34) follows analogously from Equation (10.15). □

These equations are also called the *Bellman optimality equations*. Intuitively, the Bellman optimality equations express that the value of a state under an optimal policy must equal the expected return for the best action from that state. Bellman's theorem is also known as *Bellman's optimality principle*, which is a more general concept.

Bellman's optimality principle Bellman's optimality equations for MDPs are one of the main settings of Bellman's optimality principle. However, Bellman's optimality principle has many other important applications, for example in dynamic programming. Broadly speaking, Bellman's optimality principle says that optimal solutions to decision problems can be decomposed into optimal solutions to sub-problems.

The two perspectives of Bellman's theorem naturally suggest two separate ways of finding the optimal policy. Policy iteration uses the perspective from Equation (10.31) of $\pi^\star$ as a fixed-point of the dependency between greedy policy and value function. In contrast, value iteration uses the perspective from Equation (10.32) of $v^\star$ as the fixed-point of the Bellman update. Another approach which we will not discuss here is to use a linear program where the Bellman update is interpreted as a set of linear inequalities.

10.3.3 Policy Iteration

Starting from an arbitrary initial policy, policy iteration as shown in Algorithm 10.14 uses the Bellman expectation equation to compute the value function of that policy (as we have discussed in Section 10.2) and then chooses the greedy policy with respect to that value function as its next iterate.

Algorithm 10.14: Policy iteration

```
initialize π (arbitrarily)
repeat
    compute v^π
    compute π_{v^π}
    π ← π_{v^π}
until converged
```

Let π_t be the policy after t iterations. We will now show that policy iteration converges to the optimal policy. The proof is split into two parts. First, we show that policy iteration improves policies monotonically. Then, we will use this fact to show that policy iteration converges.

Lemma 10.15 (Monotonic improvement of policy iteration)**.** *We have,*

- $v^{\pi_{t+1}}(x) \geq v^{\pi_t}(x)$ *for all* $x \in X$*; and*
- $v^{\pi_{t+1}}(x) > v^{\pi_t}(x)$ *for at least one* $x \in X$*, unless* $v^{\pi_t} \equiv v^\star$*.*

Proof. We consider the Bellman update from (10.32) as the mapping $\boldsymbol{B}^\star : \mathbb{R}^n \to \mathbb{R}^n$,

$$(\boldsymbol{B}^\star \boldsymbol{v})(x) \doteq \max_{a \in A} q(x, a), \tag{10.36}$$

where q is the state-action value function corresponding to the state value function $\boldsymbol{v} \in \mathbb{R}^n$. Recall that after obtaining v^{π_t}, policy iteration first computes the greedy policy w.r.t. v^{π_t}, $\pi_{t+1} \doteq \pi_{v^{\pi_t}}$, and then computes its value function $v^{\pi_{t+1}}$.

To establish the (weak) monotonic improvement of policy iteration, we consider a fixed-point iteration (cf. Algorithm 10.8) of $\boldsymbol{v}^{\pi_{t+1}}$ initialized by $\boldsymbol{v}^{\pi_t}$. We denote the iterates by $\tilde{\boldsymbol{v}}_\tau$, in particular, we have that $\tilde{\boldsymbol{v}}_0 = \boldsymbol{v}^{\pi_t}$ and $\lim_{\tau \to \infty} \tilde{\boldsymbol{v}}_\tau = \boldsymbol{v}^{\pi_{t+1}}$.[4] First, observe that for the first iteration of fixed-point iteration,

[4] using the convergence of fixed-point iteration (10.25)

$$\begin{aligned}
\tilde{v}_1(x) &= (\boldsymbol{B}^\star \boldsymbol{v}^{\pi_t})(x) && \text{using that } \pi_{t+1} \text{ is greedy wrt. } v^{\pi_t} \\
&= \max_{a \in A} q^{\pi_t}(x, a) && \text{using the definition of the Bellman update (10.36)} \\
&\geq q^{\pi_t}(x, \pi_t(x)) \\
&= v^{\pi_t}(x) \\
&= \tilde{v}_0(x). && \text{using (10.17)}
\end{aligned}$$

Let us now consider a single iteration of fixed-point iteration. We have,

$$\tilde{v}_{\tau+1}(x) = r(x, \pi_{t+1}(x)) + \gamma \sum_{x' \in X} p(x' \mid x, \pi_{t+1}(x)) \cdot \tilde{v}_\tau(x'). \qquad \text{using the definition of } \tilde{v}_{\tau+1} \text{ (10.21)}$$

Using an induction on τ, we conclude,

$$\geq r(x, \pi_{t+1}(x)) + \gamma \sum_{x' \in X} p(x' \mid x, \pi_{t+1}(x)) \cdot \tilde{v}_{\tau-1}(x') \quad \text{using the induction hypothesis, } \tilde{v}_\tau(x') \geq \tilde{v}_{\tau-1}(x')$$
$$= \tilde{v}_\tau(x).$$

This establishes the first claim,

$$\boldsymbol{v}^{\pi_{t+1}} = \lim_{\tau \to \infty} \tilde{\boldsymbol{v}}_\tau \geq \tilde{\boldsymbol{v}}_0 = \boldsymbol{v}^{\pi_t}. \tag{10.37}$$

For the second claim, recall from Bellman's theorem (10.32) that $v^\star$ is a (unique) fixed-point of the Bellman update $\boldsymbol{B}^\star$.[5] In particular, we have $v^{\pi_{t+1}} \equiv v^{\pi_t}$ if and only if $v^{\pi_{t+1}} \equiv v^{\pi_t} \equiv v^\star$. In other words, if $v^{\pi_t} \not\equiv v^\star$ then Equation (10.37) is strict for at least one $x \in X$ and $v^{\pi_{t+1}} \not\equiv v^{\pi_t}$. This proves the strict monotonic improvement of policy iteration. □

[5] We will show in Equation (10.38) that $\boldsymbol{B}^\star$ is a contraction, implying that $v^\star$ is the *unique* fixed-point of $\boldsymbol{B}^\star$.

Theorem 10.16 (Convergence of policy iteration)**.** *For finite Markov decision processes, policy iteration converges to an optimal policy.*

Proof. Finite Markov decision processes only have a finite number of deterministic policies (albeit exponentially many). Observe that policy iteration only considers deterministic policies, and recall that there is an optimal policy that is deterministic. As the value of policies strictly increase in each iteration until an optimal policy is found, policy iteration must converge in finite time. □

It can be shown that policy iteration converges to an exact solution in a polynomial number of iterations (Ye, 2011). Each iteration of policy iteration requires computing the value function, which we have seen to be of cubic complexity in the number of states.

10.3.4 Value Iteration

As we have mentioned, another natural approach of finding the optimal policy is to interpret $v^\star$ as the fixed point of the Bellman update. Recall our definition of the Bellman update from Equation (10.36),

$$(\boldsymbol{B}^\star \boldsymbol{v})(x) = \max_{a \in A} q(x, a),$$

where q was the state-action value function associated with the state value function $\boldsymbol{v}$. The value iteration algorithm is shown in Algorithm 10.17.

We will now prove the convergence of value iteration using the fixed-point interpretation.

Theorem 10.18 (Convergence of value iteration)**.** *Value iteration converges asymptotically to an optimal policy.*

Algorithm 10.17: Value iteration

initialize $v(x) \leftarrow \max_{a \in A} r(x, a)$ for each $x \in X$
for $t = 1$ **to** ∞ **do**
  $v(x) \leftarrow (\boldsymbol{B}^\star \boldsymbol{v})(x) = \max_{a \in A} q(x, a)$ for each $x \in X$
choose π_v

Proof. Clearly, value iteration converges if $v^\star$ is the unique fixed-point of $\boldsymbol{B}^\star$. We already know from Bellman's theorem (10.32) that $v^\star$ is a fixed-point of $\boldsymbol{B}^\star$. It remains to show that it is indeed the unique fixed-point.

Analogously to our proof of the convergence of fixed-point iteration to the value function v^π, we show that $\boldsymbol{B}^\star$ is a contraction. Fix arbitrary $\boldsymbol{v}, \boldsymbol{v}' \in \mathbb{R}^n$, then

$$
\begin{aligned}
\left\| \boldsymbol{B}^\star \boldsymbol{v} - \boldsymbol{B}^\star \boldsymbol{v}' \right\|_\infty &= \max_{x \in X} \left| (\boldsymbol{B}^\star \boldsymbol{v})(x) - (\boldsymbol{B}^\star \boldsymbol{v}')(x) \right| && \text{using the definition of the } L_\infty \text{ norm (10.23)} \\
&= \max_{x \in X} \left| \max_{a \in A} q(x, a) - \max_{a \in A} q'(x, a) \right| && \text{using the definition of the Bellman update (10.36)} \\
&\leq \max_{x \in X} \max_{a \in A} \left| q(x, a) - q'(x, a) \right| && \text{using } |\max_x f(x) - \max_x g(x)| \leq \max_x |f(x) - g(x)| \\
&\leq \gamma \max_{x \in X} \max_{a \in A} \sum_{x' \in X} p(x' \mid x, a) \left| \boldsymbol{v}(x') - \boldsymbol{v}'(x') \right| && \text{using the definition of the Q-function (10.9) and } |\textstyle\sum_i a_i| \leq \sum_i |a_i| \\
&\leq \gamma \left\| \boldsymbol{v} - \boldsymbol{v}' \right\|_\infty && \text{using } \textstyle\sum_{x' \in X} p(x' \mid x, a) = 1 \text{ and } |\boldsymbol{v}(x') - \boldsymbol{v}'(x')| \leq \|\boldsymbol{v} - \boldsymbol{v}'\|_\infty
\end{aligned} \tag{10.38}
$$

where q and q' are the state-action value functions associated with $\boldsymbol{v}$ and $\boldsymbol{v}'$, respectively. By Equation (10.22), $\boldsymbol{B}^\star$ is a contraction and by Banach's fixed-point theorem $v^\star$ is its unique fixed-point. □

Remark 10.19: Value iteration as a dynamic program

Let us denote by v_t the value function estimate after the t-th iteration. Observe that $v_t(x)$ corresponds to the maximum expected reward when starting in state x and the "world ends" after t time steps. In particular, v_0 corresponds to the maximum immediate reward. This suggests a different perspective on value iteration (akin to dynamic programming) where in each iteration we extend the time horizon of our approximation by one time step.

For any $\epsilon > 0$, value iteration converges to an ϵ-optimal solution in polynomial time. However, unlike policy iteration, value iteration does not generally reach the *exact* optimum in a finite number of iterations. Recalling the update rule of value iteration, its main benefit is that each iteration only requires a sum over all possible actions a in state

x and a sum over all reachable states x' from x. In sparse Markov decision processes,[6] an iteration of value iteration can be performed in (virtually) constant time.

[6] Sparsity refers to the interconnectivity of the state space. When only few states are reachable from any state, we call an MDP sparse.

10.4 Partial Observability

So far we have focused on the fully observable setting. That is, at any time, our agent knows its current state. We have seen that we can efficiently find the optimal policy (as long as the Markov decision process is finite).

We have already encountered the partially observable setting in Chapter 3, where we discussed filtering. In this section, we consider how Markov decision processes can be extended to a partially observable setting where the agent can only access noisy observations Y_t of its state X_t.

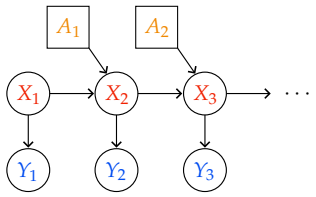


Figure 10.4: Directed graphical model of a partially observable Markov decision process with hidden states X_t, observables Y_t, and actions A_t.

Definition 10.20 (Partially observable Markov decision process, POMDP)**.** Similarly to a Markov decision process, a *partially observable Markov decision process* is specified by

- a set of *states* X,
- a set of *actions* A,
- *transition probabilities* $p(x' \mid x, a)$, and
- a *reward function* $r : X \times A \to \mathbb{R}$.

Additionally, it is specified by

- a set of *observations* Y, and
- *observation probabilities*

$$o(y \mid x) \doteq \mathbb{P}(Y_t = y \mid X_t = x). \tag{10.39}$$

Whereas MDPs are controlled Markov chains, POMDPs are controlled hidden Markov models.

Remark 10.21: Hidden Markov models
A hidden Markov model is a Markovian process with unobservable states X_t and observations Y_t that depend on X_t in a known way.

Definition 10.22 (Hidden Markov model, HMM)**.** A *hidden Markov model* is specified by

- a set of *states* X,
- *transition probabilities* $p(x' \mid x) \doteq \mathbb{P}(X_{t+1} = x' \mid X_t = x)$ (also called *motion model*), and
- a *sensor model* $o(y \mid x) \doteq \mathbb{P}(Y_t = y \mid X_t = x)$.

Following from its directed graphical model shown in Figure 10.5,

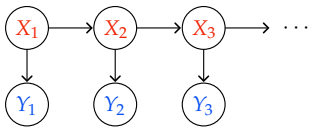


Figure 10.5: Directed graphical model of a hidden Markov model with hidden states X_t and observables Y_t.

its joint probability distribution factorizes into

$$\mathbb{P}(x_{1:t}, y_{1:t}) = \mathbb{P}(x_1) \cdot o(y_1 \mid x_1) \cdot \prod_{i=2}^{t} p(x_i \mid x_{i-1}) \cdot o(y_i \mid x_i). \tag{10.40}$$

Observe that a Kalman filter can be viewed as a hidden Markov model with conditional linear Gaussian motion and sensor models and a Gaussian prior on the initial state. In particular, the tasks of *filtering*, *smoothing*, and *predicting* which we discussed extensively in Chapter 3 are also of interest for hidden Markov models.

A widely used application of hidden Markov models is to find the most likely sequence (also called *most likely explanation*) of hidden states $x_{1:t}$ given a series of observations $y_{1:t}$,[7] that is, to find

$$\arg\max_{x_{1:t}} \mathbb{P}(x_{1:t} \mid y_{1:t}). \tag{10.41}$$

This task can be solved in linear time by a simple backtracking algorithm known as the *Viterbi algorithm*.

[7] This is useful in many applications such as speech recognition, decoding data that was transmitted over a noisy channel, beat detection, and many more.

POMDPs are a very powerful model, but very hard to solve in general. POMDPs can be reduced to a Markov decision process with an enlarged state space. The key insight is to consider an MDP whose states are the *beliefs*,

$$b_t(x) \doteq \mathbb{P}(X_t = x \mid y_{1:t}, a_{1:t-1}), \tag{10.42}$$

about the current state in the POMDP. In other words, the states of the MDP are probability distributions over the states of the POMDP. We will make this more precise in the following.

Let us assume that our prior belief about the state of our agent is given by $b_0(x) \doteq \mathbb{P}(X_0 = x)$. Keeping track of how beliefs change over time is known as *filtering*, which we already encountered in Section 3.1. Given a prior belief b_t, an action taken a_t, and a new observation y_{t+1}, the belief state can be updated as follows,

$$\begin{aligned}
b_{t+1}(x) &= \mathbb{P}(X_{t+1} = x \mid y_{1:t+1}, a_{1:t}) && \text{by the definition of beliefs (10.42)} \\
&= \frac{1}{Z}\mathbb{P}(y_{t+1} \mid X_{t+1} = x)\mathbb{P}(X_{t+1} = x \mid y_{1:t}, a_{1:t}) && \text{using Bayes' rule (1.45)} \\
&= \frac{1}{Z}o(y_{t+1} \mid x)\mathbb{P}(X_{t+1} = x \mid y_{1:t}, a_{1:t}) && \text{using the definition of observation probabilities (10.39)} \\
&= \frac{1}{Z}o(y_{t+1} \mid x) \sum_{x' \in X} p(x \mid x', a_t)\mathbb{P}\big(X_t = x' \mid y_{1:t}, a_{1:t-1}\big) && \text{by conditioning on the previous state } x', \text{ noting } a_t \text{ does not influence } X_t \\
&= \frac{1}{Z}o(y_{t+1} \mid x) \sum_{x' \in X} p(x \mid x', a_t) b_t(x') && \text{using the definition of beliefs (10.42)}
\end{aligned} \tag{10.43}$$

where

$$Z \doteq \sum_{x \in X} o(y_{t+1} \mid x) \sum_{x' \in X} p(x \mid x', a_t) b_t(x'). \tag{10.44}$$

Thus, the updated belief state is a deterministic mapping from the previous belief state depending only on the (random) observation y_{t+1} and the taken action a_t. Note that this obeys a Markovian structure of transition probabilities with respect to the beliefs b_t.

The sequence of belief-states defines the sequence of random variables $(B_t)_{t \in \mathbb{N}_0}$,

$$B_t \doteq X_t \mid y_{1:t}, a_{1:t-1}, \tag{10.45}$$

where the (state-)space of all beliefs is the (infinite) space of all probability distributions over X,[8]

$$\mathcal{B} \doteq \Delta^X \doteq \left\{ \boldsymbol{b} \in \mathbb{R}^{|X|} : \boldsymbol{b} \geq \mathbf{0}, \textstyle\sum_{i=1}^{|X|} \boldsymbol{b}(i) = 1 \right\}. \tag{10.46}$$

[8] This definition naturally extends to continuous state spaces $\mathcal{X}$.

A Markov decision process, where every belief corresponds to a state is called a belief-state MDP.

Definition 10.23 (Belief-state Markov decision process)**.** Given a POMDP, the corresponding *belief-state Markov decision process* is a Markov decision process specified by

- the *belief space* $\mathcal{B} \doteq \Delta^X$ depending on the *hidden states* X,
- the set of *actions* A,
- *transition probabilities*

$$\tau(b' \mid b, a) \doteq \mathbb{P}\big(B_{t+1} = b' \mid B_t = b, A_t = a\big), \tag{10.47}$$

- and *rewards*

$$\rho(b, a) \doteq \mathbb{E}_{x \sim b}[r(x, a)] = \sum_{x \in X} b(x) r(x, a). \tag{10.48}$$

It remains to derive the transition probabilities τ in terms of the original POMDP. We have,

$$\begin{aligned} \tau(b_{t+1} \mid b_t, a_t) &= \mathbb{P}(b_{t+1} \mid b_t, a_t) \\ &= \sum_{y_{t+1} \in Y} \mathbb{P}(b_{t+1} \mid b_t, a_t, y_{t+1}) \mathbb{P}(y_{t+1} \mid b_t, a_t). \end{aligned} \tag{10.49}$$

by conditioning on $y_{t+1} \in Y$

Using the Markovian structure of the belief updates, we naturally set,

$$\mathbb{P}(b_{t+1} \mid b_t, a_t, y_{t+1}) \doteq \begin{cases} 1 & \text{if } b_{t+1} \text{ matches the belief update} \\ & \text{of Equation (10.42) given } b_t, a_t, \text{ and} \\ & y_{t+1}, \\ 0 & \text{otherwise.} \end{cases} \tag{10.50}$$

The final missing piece is the likelihood of an observation y_{t+1} given the prior belief b_t and action a_t, which using our interpretation of beliefs corresponds to

$$\begin{aligned}\mathbb{P}(y_{t+1} \mid b_t, a_t) &= \mathbb{E}_{x \sim b_t}\Big[\mathbb{E}_{x' \mid x, a_t}\big[\mathbb{P}\big(y_{t+1} \mid X_{t+1} = x'\big)\big]\Big] \\ &= \mathbb{E}_{x \sim b_t}\Big[\mathbb{E}_{x' \mid x, a_t}\big[o(y_{t+1} \mid x')\big]\Big] \\ &= \sum_{x \in X} b_t(x) \sum_{x' \in X} p(x' \mid x, a_t) \cdot o(y_{t+1} \mid x'). \qquad (10.51)\end{aligned}$$

using the definition of observation probabilities (10.39)

In principle, we can now apply arbitrary algorithms for planning in MDPs to POMDPs. Of course, the problem is that there are infinitely many beliefs, even for a finite state space X.[9] The belief-state MDP has therefore an infinitely large belief space $\mathcal{B}$. Even when only planning over finite horizons, exponentially many beliefs can be reached. So the belief space blows-up very quickly.

We will study MDPs with large state spaces (where transition dynamics and rewards are unknown) in Chapters 12 and 13. Similar methods can also be used to approximately solve POMDPs.

A key idea in approximate solutions to POMDPs is that most belief states are never reached. A common approach is to discretize the belief space by sampling or by applying a dimensionality reduction. Examples are *point-based value iteration* (PBVI) and *point-based policy iteration* (PBPI) (Shani et al., 2013).

[9] You can think of an $|X|$-dimensional space. Here, all points whose coordinates sum to 1 correspond to probability distributions (i.e., beliefs) over the hidden states X. The convex hull of these points is also known as the $(|X|-1)$-dimensional *probability simplex* (cf. Appendix A.1.2). Now, by definition of the $(|X|-1)$-dimensional probability simplex as a polytope in $|X|-1$ dimensions, we can conclude that its boundary consists of infinitely many points in $|X|$ dimensions. Noting that these points corresponded to the probability distributions on $|X|$, we conclude that there are infinitely many such distributions.

Discussion

Even though we focus on the fully observed setting throughout this manuscript, the partially observed setting can be reduced to the fully observed setting with very large state spaces. In the next chapter, we will consider learning and planning in unknown Markov decision processes (i.e., reinforcement learning) for small state spaces. The setting of small state and action spaces is also known as the *tabular setting*. Then, in the final two chapters, we will consider approximate methods for large state and action spaces. In particular, in Section 13.1, we will revisit the problem of probabilistic planning in known Markov decision processes, but with continuous state and action spaces.

Problems

10.1. Value functions.

Recall the example of "becoming rich and famous" from Figure 10.2. Consider the policy, $\pi \equiv S$ (i.e., to always *save*) and let $\gamma = 1/2$. Show that the (rounded) state-action value function q^π is as follows:

	save	advertise
poor, unknown	**0**	0.1
poor, famous	**4.4**	1.2
rich, famous	**17.8**	1.2
rich, unknown	**13.3**	0.1

Shown in bold is the state value function v^π.

10.2. Greedy policies.

Show that if q and v arise from the same policy, that is, q is defined in terms of v as per Equation (10.9), then

$$\pi_v \equiv \pi_q. \tag{10.52}$$

This implies that we can use v and q interchangeably.

10.3. Optimal policies.

Again, recall the example of "becoming rich and famous" from Figure 10.2.

1. Show that the policy $\pi \equiv S$, which we considered in Problem 10.1, is not optimal.
2. Instead, consider the policy

$$\pi' \equiv \begin{cases} A & \text{if poor and unknown} \\ S & \text{otherwise} \end{cases}$$

   and let $\gamma = 1/2$. Show that the (rounded) state-action value function $q^{\pi'}$ is as follows:

	save	advertise
poor, unknown	0.8	**1.6**
poor, famous	**4.5**	1.2
rich, famous	**17.8**	1.2
rich, unknown	**13.4**	0.2

   Shown in bold is the state value function $v^{\pi'}$.
3. Is the policy π' optimal?

10.4. Linear convergence of policy iteration.

Denote by π_t the policy obtained by policy iteration after t iterations. Use that the Bellman operator $\boldsymbol{B}^\star$ is a contraction with the unique fixed-point $\boldsymbol{v}^\star$ to show that

$$\|\boldsymbol{v}^{\pi_t} - \boldsymbol{v}^\star\|_\infty \leq \gamma^t \|\boldsymbol{v}^{\pi_0} - \boldsymbol{v}^\star\|_\infty \tag{10.53}$$

where $\boldsymbol{v}^\pi$ and $\boldsymbol{v}^\star$ are vector representations of the functions v^π and $v^\star$, respectively.

Hint: Recall from Lemma 10.15 that $\boldsymbol{v}^{\pi_{t+1}} \geq \boldsymbol{B}^\star \boldsymbol{v}^{\pi_t} \geq \boldsymbol{v}^{\pi_t}$.

10.5. Reward modification.

A key technique for solving sequential decision problems is the modification of reward functions that leaves the optimal policy unchanged while improving sample efficiency or convergence rates. This exercise looks at simple ways of modifying rewards and understanding how these modifications affect the optimal policy.

Consider two Markov decision processes $M \doteq (X, A, p, r)$ and $M' \doteq (X, A, p, r')$ where the reward function r is modified to obtain r', and the rewards are bounded and discounted by the discount factor $\gamma \in [0, 1)$. Let $\pi^\star_M$ be the optimal policy for M.

1. Suppose $r'(x) = \alpha r(x)$, where $\alpha > 0$. Show that the optimal policy $\pi^\star$ of M is also an optimal policy of M'.
2. Given a modification of the form $r'(x) = r(x) + c$, where $c > 0$ is a constant scalar, show that the optimal policy $\pi^\star_M$ can be different from $\pi^\star_{M'}$.
3. Another way of modifying the reward function is through *reward shaping* where one supplies additional rewards to the agent to guide the learning process. When one has no knowledge of the underlying transition dynamics p, a commonly used transformation is $r'(x, x') = r(x, x') + f(x, x')$ where f is a *potential-based* shaping function defined as

$$f(x, x') \doteq \gamma\phi(x') - \phi(x), \quad \phi : X \to \mathbb{R}. \tag{10.54}$$

   Show that the optimal policy remains unchanged under this definition of f.

10.6. A partially observable fishing problem.

We model an angler's decisions while fishing, where the states are partially observable. There are two states: (1) **Fish** (F): A fish is hooked on the line. (2) **No fish** ($\overline{F}$): No fish is hooked on the line.

The angler can choose between two actions:

- **Pull up the rod** (P): If there is a fish on the line (F), there is a 90% chance of catching it (reward $+10$, transitioning to $\overline{F}$) and a 10% chance of it escaping (reward -1, transitioning to $\overline{F}$). If there is no fish ($\overline{F}$), pulling up the rod results in no catch, staying in $\overline{F}$ with a reward of -5.
- **Waiting** (W): All waiting actions result in a reward of -1. In state F, there is a 60% chance of the fish staying (remaining in F) and a 40% chance of it escaping (transitioning to $\overline{F}$). In state $\overline{F}$, there is a 50% chance of a fish biting (transitioning to F) and a 50% chance of no change (remaining in $\overline{F}$).

Suggestion: Draw the MDP transition diagram. Draw each transition with action, associated probability, and associated reward.

Since the angler cannot directly observe whether there is a fish on the line, they receive a noisy observation about the state. This observation can be:

- o_1: The signal suggests that a fish might be on the line.
- o_2: The signal suggests that there is no fish on the line.

The observation model, which defines the probability of receiving each observation given the true state is as follows:

	$\mathbb{P}(o_1 \mid \cdot)$	$\mathbb{P}(o_2 \mid \cdot)$
F	0.8	0.2
$\overline{F}$	0.3	0.7

The angler's goal is to choose actions that maximize their overall reward, balancing the chances of catching a fish against the cost of waiting and unsuccessful pulls.

1. Given an initial belief $b_0(F) = b_0(\overline{F}) = 0.5$, the angler chooses to wait and observes o_1. Compute the updated belief b_1 using the observation model and belief update equation (10.42).
2. Given belief $b_1(F) \approx 0.765$ and $b_1(\overline{F}) \approx 0.235$, compute the updated belief b_2 for both actions P (pull) and W (wait), both in the case where you observe o_1 (fish likely) and o_2 (fish unlikely).

11

Tabular Reinforcement Learning

11.1 The Reinforcement Learning Problem

Reinforcement learning is concerned with probabilistic planning in unknown environments. This extends our study of known environments in the previous chapter. Those environments are still modeled by Markov decision processes, but in reinforcement learning, we do not know the dynamics p and rewards r in advance. Hence, reinforcement learning is at the intersection of the theories of probabilistic planning (i.e., Markov decision processes) and learning (e.g., multi-armed bandits), which we covered extensively in the previous chapters.

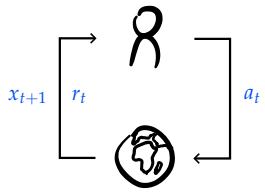


Figure 11.1: In reinforcement learning, an agent interacts with its environment in a sequence of rounds. After playing an action a_t, it observes rewards r_t and its new state x_{t+1}. The agent then uses this information to learn how to act to maximize reward.

We will continue to focus on the fully observed setting, where the agent knows its current state. As we have seen in the previous section, the partially observed setting corresponds to a fully observed setting with an enlarged state space. In this chapter, we will begin by considering reinforcement learning with small state and action spaces. This setting is often called the *tabular setting*, as the value functions can be computed exhaustively for all states and stored in a table.

Clearly, the agent needs to trade exploring and learning about the environment with exploiting its knowledge to maximize rewards. Thus, the exploration-exploitation dilemma, which was at the core of Bayesian optimization (see Section 9.1), also plays a crucial role in reinforcement learning. In fact, Bayesian optimization can be viewed as reinforcement learning with a fixed state: In each round, the agent plays an action, aiming to find the action that maximizes the reward. However, playing the same action multiple times yields the same reward, implying that we remain in a single state. In the context of Bayesian optimization, we used "regret" as performance metric: in the jargon of planning, minimizing regret corresponds to maximizing the cumulative reward.

Another key challenge of reinforcement learning is that the observed data is dependent on the played actions. This is in contrast to the setting of supervised learning that we have been considering in earlier chapters, where the data is sampled independently.

11.1.1 Trajectories

The data that the agent collects is modeled using so-called trajectories.

Definition 11.1 (Trajectory)**.** A *trajectory* τ is a (possibly infinite) sequence,

$$\tau \doteq (\tau_0, \tau_1, \tau_2, \dots), \tag{11.1}$$

of *transitions*,

$$\tau_i \doteq (x_i, a_i, r_i, x_{i+1}), \tag{11.2}$$

where $x_i \in X$ is the starting state, $a_i \in A$ is the played action, $r_i \in \mathbb{R}$ is the attained reward, and $x_{i+1} \in X$ is the ending state.

In the context of learning a dynamics and rewards model, x_i and a_i can be understood as inputs, and r_i and x_{i+1} can be understood as labels of a regression problem.

Crucially, the newly observed states x_{t+1} and the rewards r_t (across multiple transitions) are conditionally independent given the previous states x_t and actions a_t. This follows directly from the Markovian structure of the underlying Markov decision process.[1] Formally, we have,

$$X_{t+1} \perp X_{t'+1} \mid X_t, X_{t'}, A_t, A_{t'}, \tag{11.3a}$$
$$R_t \perp R_{t'} \mid X_t, X_{t'}, A_t, A_{t'}, \tag{11.3b}$$

for any $t, t' \in \mathbb{N}_0$. In particular, if $x_t = x_{t'}$ and $a_t = a_{t'}$, then x_{t+1} and $x_{t'+1}$ are independent samples according to the transition model $p(X_{t+1} \mid x_t, a_t)$. Analogously, if $x_t = x_{t'}$ and $a_t = a_{t'}$, then r_t and $r_{t'}$ are independent samples of the reward model $r(x_t, a_t)$. As we will see later in this chapter and especially in Chapter 13, this independence property is crucial for being able to learn about the underlying Markov decision process. Notably, this implies that we can apply the law of large numbers (A.36) and Hoeffding's inequality (A.41) to our estimators of both quantities.

[1] Recall the Markov property (6.6), which assumes that in the underlying Markov decision process (i.e., in our environment) the future state of an agent is independent of past states given the agent's current state. This is commonly called a Markovian structure. From this Markovian structure, we gather that repeated encounters of state-action pairs result in independent trials of the transition model and rewards.

The collection of data is commonly classified into two settings. In the *episodic setting*, the agent performs a sequence of "training" rounds (called *episodes*). In the beginning of each episode, the agent is reset to some initial state. In contrast, in the *continuous setting* (or non-episodic /

online setting), the agent learns online. Especially, every action, every reward, and every state transition counts.

The episodic setting is more applicable to an agent playing a computer game. That is, the agent is performing in a simulated environment that is easy to reset. The continuous setting is akin to an agent that is deployed to the "real world". In principle, real-world agents can be trained in simulated environments before being deployed. However, this bears the risk of learning to exploit or rely on features of the simulated environment that are not present in the real environment. Sometimes, using a simulated environment for training is downright impossible, as the real environment is too complex.

11.1.2 On-policy and Off-policy Methods

Another important distinction in how data is collected, is the distinction between on-policy and off-policy methods. As the names suggest, *on-policy* methods are used when the agent has control over its own actions, in other words, the agent can freely choose to follow any policy. Being able to follow a policy is helpful, for example because it allows the agent to experiment with trading exploration and exploitation.

In contrast, *off-policy* methods can be used even when the agent cannot freely choose its actions. Off-policy methods are therefore able to make use of purely observational data. This might be data that was collected by another agent, a fixed policy, or during a previous episode. Off-policy methods are therefore more *sample-efficient* than on-policy methods. This is crucial, especially in settings where conducting experiments (i.e., collecting new data) is expensive.

11.2 Model-based Approaches

Approaches to reinforcement learning are largely categorized into two classes. *Model-based* approaches aim to learn the underlying Markov decision process. More concretely, they learn models of the dynamics p and rewards r. They then use these models to perform planning (i.e., policy optimization) in the underlying Markov decision process. In contrast, *model-free* approaches learn the value function directly. We begin by discussing model-based approaches to the tabular setting. In Section 11.4, we cover model-free approaches.

11.2.1 Learning the Underlying Markov Decision Process

Recall that the underlying Markov decision process was specified by its dynamics $p(x' \mid x, a)$ that correspond to the probability of entering

state $x' \in X$ when playing action $a \in A$ from state $x \in X$, and its rewards $r(x, a)$ for playing action $a \in A$ in state $x \in X$. A natural first idea is to use maximum likelihood estimation to approximate these quantities.

We can think of each transition $x' \mid x, a$ as sampling from a categorical random variable of which we want to estimate the success probabilities for landing in each of the states. Therefore, as we have seen in Example A.7, the MLE of the dynamics model coincides with the sample mean,

$$\hat{p}(x' \mid x, a) = \frac{N(x' \mid x, a)}{N(a \mid x)} \tag{11.4}$$

where $N(x' \mid x, a)$ counts the number of transitions from state x to state x' when playing action a and $N(a \mid x)$ counts the number of transitions that start in state x and play action a (regardless of the next state). Similarly, for the rewards model, we obtain the following maximum likelihood estimate (i.e., sample mean),

$$\hat{r}(x, a) = \frac{1}{N(a \mid x)} \sum_{\substack{t=0 \\ x_t = x \\ a_t = a}}^{\infty} r_t. \tag{11.5}$$

It is immediate that both estimates are unbiased as both correspond to a sample mean.

Still, for the models of our environment to become accurate, our agent needs to visit *each* state-action pair (x, a) numerous times. Note that our estimators for dynamics and rewards are only well-defined when we visit the corresponding state-action pair at least once. However, in a stochastic environment, a single visit will likely not result in an accurate model. We can use Hoeffding's inequality (A.41) to gauge how accurate the estimates are after only a limited number of visits.

11.3 Balancing Exploration and Exploitation

The next natural question is how to use our current model of the environment to pick actions such that exploration and exploitation are traded effectively. This is what we will consider next.

Given the estimated MDP given by $\hat{p}$ and $\hat{r}$, we can compute the optimal policy using either policy iteration or value iteration. For example, using value iteration, we can compute the optimal state-action value function $Q^\star$ within the *estimated* MDP, and then employ the greedy policy

$$\pi(x) = \arg\max_{a \in A} Q^\star(x, a). \tag{11.6}$$

Recall from Equation (10.31) that this corresponds to always picking the best action under the *current* model (that is, π is the optimal policy). But since the model is inaccurate, while potentially quickly generating some reward, we will likely get stuck in a suboptimal state.

11.3.1 ε-greedy

Consider the other extreme: If we always pick a random action, we will eventually(!) estimate the dynamics and rewards correctly, yet we will do extremely poorly in terms of maximizing rewards along the way. To trade exploration and exploitation, a natural idea is to balance these two extremes.

Arguably, the simplest idea is the following: At each time step, throw a biased coin. If this coin lands heads, we pick an action uniformly at random among all actions. If the coin lands tails, we pick the best action under our current model. This algorithm is called *ε-greedy*, where the probability of a coin landing heads at time t is ε_t.

Algorithm 11.2: ε-greedy

```
for t = 0 to ∞ do
    sample u ∈ Unif([0, 1])
    if u ≤ ε_t then pick action uniformly at random among all actions
    else pick best action under the current model
```

The ε-greedy algorithm provides a general framework for addressing the exploration-exploitation dilemma. When the underlying MDP is learned using Monte Carlo estimation as we discussed in Section 11.2.1, the resulting algorithm is known as *Monte Carlo control*. However, the same framework can also be used in the model-free setting where we pick the best action without estimating the full underlying MDP. We discuss this approach in greater detail in Section 11.4.

Amazingly, this simple algorithm already works quite well. Nevertheless, it can clearly be improved. The key problem of ε-greedy is that it explores the state space in an uninformed manner. In other words, it explores ignoring all past experience. It thus does not eliminate clearly suboptimal actions. This is a problem, especially as we typically have many state-action pairs and recalling that we have to explore each such pair many times to learn an accurate model.

Remark 11.3: Asymptotic convergence

It can be shown that Monte Carlo control converges to an optimal policy (albeit slowly) almost surely when the learned policy is "greedy in the limit with infinite exploration".

Definition 11.4 (Greedy in the limit with infinite exploration, GLIE)**.** A sequence of policies π_t is said to be *greedy in the limit with infinite exploration* if

1. all state-action pairs are explored infinitely many times,[2]

$$\lim_{t\to\infty} N_t(x,a) = \infty \quad \text{and} \tag{11.7}$$

2. the policy converges to a greedy policy,

$$\lim_{t\to\infty} \pi_t(a \mid x) = \mathbb{1}\{a = \arg\max_{a'\in A} Q_t^\star(x,a')\} \tag{11.8}$$

where we denote by $N_t(x,a)$ the number of transitions from state x playing action a until time t, and $Q_t^\star$ is the optimal state-action value function in the estimated MDP at time t.

[2] That all state-action pairs are chosen is a fundamental requirement. There is no reason why any algorithm would converge to the true value function for *all* states when it only sees some state-action pairs finitely many times, or even not at all.

Note that ε-greedy is GLIE with probability 1 if the sequence $(\varepsilon_t)_{t\in\mathbb{N}_0}$ satisfies the Robbins-Monro (RM) conditions (A.60),

$$\varepsilon_t \geq 0 \quad \forall t, \quad \sum_{t=0}^{\infty} \varepsilon_t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \varepsilon_t^2 < \infty.$$

The RM-conditions are satisfied, for example, if $\varepsilon_t = 1/t$.

Theorem 11.5 (Convergence of Monte Carlo control)**.** *GLIE Monte Carlo control converges to an optimal policy with probability* 1*.*

Intuitively, the probability of exploration converges to zero, and hence, the policy will "eventually coincide" with the greedy policy. Moreover, the greedy policy will "eventually coincide" with the optimal policy due to an argument akin to the convergence of policy iteration,[3] and using that each state-action pair is visited infinitely often.

[3] see Lemma 10.15 and Theorem 10.16

11.3.2 *Softmax Exploration*

An alternative to using ε-greedy for trading between greedy exploitation and uniform exploration is the so-called *softmax exploration* or *Boltzmann exploration*. Given the agent is in state x, we pick action a with probability,

$$\pi_\lambda(a \mid x) \propto \exp\left(\frac{1}{\lambda} Q^\star(x,a)\right), \tag{11.9}$$

which is the Gibbs distribution with temperature parameter $\lambda > 0$. Observe that for $\lambda \to 0$, softmax exploration corresponds to greedily maximizing the Q-function (i.e., greedy exploitation), whereas for $\lambda \to \infty$, softmax exploration explores uniformly at random. This can outperform ε-greedy as the exploration is directed towards actions with larger estimated value.

11.3.3 *Optimism*

Recall from our discussion of multi-armed bandits in Section 9.2.1 that a key principle in effectively trading exploration and exploitation is *optimism in the face of uncertainty*. Let us apply this principle to the reinforcement learning setting. The key idea is to assume that the dynamics and rewards model "work in our favor" until we have learned "good estimates" of the true dynamics and rewards.

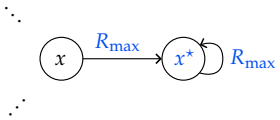


Figure 11.2: Illustration of the fairy-tale state of $R_{\max}$. If in doubt, the agent believes actions from the state x to lead to the fairy-tale state $x^\star$ with maximal rewards. This encourages the exploration of unknown states.

More formally, if $r(x, a)$ is unknown, we set $\hat{r}(x, a) = R_{\max}$, where $R_{\max}$ is the maximum reward our agent can attain during a single transition. Similarly, if $p(x' \mid x, a)$ is unknown, we set $\hat{p}(x^\star \mid x, a) = 1$, where $x^\star$ is a "fairy-tale state". The fairy-tale state corresponds to everything our agent could wish for, that is,

$$\hat{p}(x^\star \mid x^\star, a) = 1 \qquad \forall a \in A, \tag{11.10}$$

$$\hat{r}(x^\star, a) = R_{\max} \qquad \forall a \in A. \tag{11.11}$$

In practice, the decision of when to assume that the learned dynamics and reward models are "good enough" has to be tuned.

In using these optimistic estimates of p and r, we obtain an optimistic underlying Markov decision process that exhibits a bias towards exploration. In particular, the rewards attained in this MDP, are an upper bound of the true reward. The resulting algorithm is known as the $R_{\max}$ *algorithm*.

How many transitions are "enough"? We can use Hoeffding's inequality (A.41) to get a rough idea! The key here, is our observation from Equation (11.3) that the transitions and rewards are conditionally independent given the state-action pairs since, as we have discussed in Section 6.1.4 on the ergodic theorem, Hoeffding's inequality does not hold for dependent samples. In this case, Hoeffding's inequality tells us that for the absolute approximation error to be below ϵ with probability at least $1 - \delta$, we need

$$N(a \mid x) \geq \frac{R_{\max}^2}{2\epsilon^2} \log \frac{2}{\delta}. \tag{11.12}$$

see (A.42)

Lemma 11.7 (Exploration and exploitation of $R_{\max}$)**.** *Every T time steps, with high probability, $R_{\max}$ either*

Algorithm 11.6: $R_{\max}$ algorithm

add the fairy-tale state $x^\star$ to the Markov decision process
set $\hat{r}(x, a) = R_{\max}$ for all $x \in X$ and $a \in A$
set $\hat{p}(x^\star \mid x, a) = 1$ for all $x \in X$ and $a \in A$
compute the optimal policy $\hat{\pi}$ for $\hat{r}$ and $\hat{p}$
for $t = 0$ **to** ∞ **do**
    execute policy $\hat{\pi}$ (for some number of steps)
    for each visited state-action pair (x, a), update $\hat{r}(x, a)$
    estimate transition probabilities $\hat{p}(x' \mid x, a)$
    after observing "enough" transitions and rewards, recompute the optimal policy $\hat{\pi}$ according the current model $\hat{p}$ and $\hat{r}$.

- *obtains near-optimal reward; or*
- *visits at least one unknown state-action pair.*[4]

Here, T depends on the mixing time of the Markov chain induced by the optimal policy.

[4] Note that in the tabular setting, there are "only" polynomially many state-action pairs.

Theorem 11.8 (Convergence of $R_{\max}$, Brafman and Tennenholtz (2002))**.** *With probability at least $1 - \delta$, $R_{\max}$ reaches an ϵ-optimal policy in a number of steps that is polynomial in $|X|$, $|A|$, T, $1/\epsilon$, $1/\delta$, and $R_{\max}$.*

11.3.4 Challenges of Model-based Approaches

We have seen that the $R_{\max}$ algorithm performs remarkably well in the tabular setting. However, there are important computational limitations to the model-based approaches that we discussed so far.

First, observe that the (tabular) model-based approach requires us to store $\hat{p}(x' \mid x, a)$ and $\hat{r}(x, a)$ in a table. This table already has $O(n^2 m)$ entries. Even though polynomial in the size of the state and action spaces, this quickly becomes unmanageable.

Second, the model-based approach requires us to "solve" the learned Markov decision processes to obtain the optimal policy (using policy or value iteration). As we continue to learn over time, we need to find the optimal policy many times. $R_{\max}$ recomputes the policy after each state-action pair is observed sufficiently often, so $O(nm)$ times.

11.4 Model-free Approaches

In the previous section, we have seen that learning and remembering the model as well as planning within the estimated model can potentially be quite expensive in the model-based approach. We there-

fore turn to model-free methods that estimate the value function directly. Thus, they require neither remembering the full model nor planning (i.e., policy optimization) in the underlying Markov decision process. We will, however, return to model-based methods in Chapter 13 to see that promise lies in combining methods from model-based reinforcement learning with methods from model-free reinforcement learning.

A significant benefit to model-based reinforcement learning is that it is inherently off-policy. That is, any trajectory regardless of the policy used to obtain it can be used to improve the model of the underlying Markov decision process. In the model-free setting, this not necessarily true. By default, estimating the value function according to the data from a trajectory, will yield an estimate of the value function corresponding to the policy that was used to sample the data.

We will start by discussing on-policy methods and later see how the value function can be estimated off-policy.

11.4.1 On-policy Value Estimation

Let us suppose, our agent follows a fixed policy π. Then, the corresponding value function v^π is given as

$$\begin{aligned} v^\pi(x) &= r(x, \pi(x)) + \gamma \sum_{x' \in X} p(x' \mid x, \pi(x)) \cdot v^\pi(x') && \text{using the definition of the value function (10.7)} \\ &= \mathbb{E}_{R_0, X_1}[R_0 + \gamma v^\pi(X_1) \mid X_0 = x, A_0 = \pi(x)] && \text{interpreting the above expression as an expectation over the random quantities } R_0 \text{ and } X_1 \end{aligned} \tag{11.13}$$

Our first instinct might be to use a Monte Carlo estimate of this expectation. Due to the conditional independence of the transitions (11.3), Monte Carlo approximation does yield an unbiased estimate,

$$\approx r + \gamma v^\pi(x'), \tag{11.14}$$

where the agent observed the transition (x, a, r, x'). Note that to estimate this expectation we use a single(!) sample,[5] unlike our previous applications of Monte Carlo sampling where we usually averaged over m samples. However, there is one significant problem in this approximation. Our approximation of v^π does in turn depend on the (unknown) true value of v^π!

[5] The idea is that we will use this approximation repeatedly as our agent collects new data to achieve the same effect as initially averaging over multiple samples.

The key idea is to use a bootstrapping estimate of the value function instead. That is, in place of the true value function v^π, we will use a "running estimate" V^π. In other words, whenever observing a new transition, we use our previous best estimate of v^π to obtain a new estimate V^π. We already encountered bootstrapping briefly in Section 7.3.4 in the context of probabilistic ensembles in Bayesian deep learning. More generally, *bootstrapping* refers to approximating a true

quantity (e.g., v^π) by using an empirical quantity (e.g., V^π), which itself is constructed using samples from the true quantity that is to be approximated.

Due to its use in estimating the value function, bootstrapping is a core concept to model-free reinforcement learning. Crucially, using a bootstrapping estimate generally results in biased estimates of the value function. Moreover, due to relying on a single sample, the estimates from Equation (11.14) tend to have very large variance.

The variance of the estimate is typically reduced by mixing new estimates of the value function with previous estimates using a learning rate α_t. This yields the *temporal-difference learning* algorithm.

Algorithm 11.9: Temporal-difference (TD) learning

initialize V^π arbitrarily (e.g., as $\mathbf{0}$)
for $t = 0$ **to** ∞ **do**
  follow policy π to obtain the transition (x, a, r, x')
  $V^\pi(x) \leftarrow (1 - \alpha_t) V^\pi(x) + \alpha_t (r + \gamma V^\pi(x'))$ // (11.15)

The update rule is sometimes written equivalently as

$$V^\pi(x) \leftarrow V^\pi(x) + \alpha_t (r + \gamma V^\pi(x') - V^\pi(x)). \tag{11.16}$$

Thus, the update to $V^\pi(x)$ is proportional to the learning rate and the difference between the previous estimate and the renewed estimate using the new observation.

Theorem 11.10 (Convergence of TD-learning, Jaakkola et al. (1993))**.** *If $(\alpha_t)_{t \in \mathbb{N}_0}$ satisfies the RM-conditions* (A.60) *and all state-action pairs are chosen infinitely often, then V^π converges to v^π with probability* 1*.*

Importantly, note that due to the Monte Carlo approximation of Equation (11.13) with respect to transitions attained by following policy π, TD-learning is fundamentally on-policy. That is, for the estimates V^π to converge to the true value function v^π, the transitions that are used for the estimation must follow policy π.

11.4.2 *SARSA: On-policy Control*

TD-learning merely estimates the value function of a fixed policy π. To find the optimal policy $\pi^\star$, we can use an analogue of policy iteration (see Algorithm 10.14). Here, it is more convenient to use an estimate of the state-action value function q^π which can be obtained

analogously to the bootstrapping estimate of v^π (11.14),

$$\begin{aligned} q^\pi(x,a) &= r(x,a) + \gamma \sum_{x' \in X} p(x' \mid x,a) \sum_{a' \in A} \pi(a' \mid x') q^\pi(x',a') && \text{using Bellman's expectation equation (10.15)} \\ &= \mathbb{E}_{R_0,X_1,A_1}[R_0 + \gamma q^\pi(X_1,A_1) \mid X_0 = x, A_0 = a] \qquad (11.17) && \text{interpreting the above expression as an expectation over } R_0, X_1 \text{ and } A_1 \\ &\approx r + \gamma q^\pi(x',a'), \qquad (11.18) && \text{Monte Carlo approximation with a single sample} \end{aligned}$$

where the agent observed transitions (x,a,r,x') and (x',a',r',x'').

The update rule from TD-learning is therefore adapted to[6]

$$Q^\pi(x,a) \leftarrow (1-\alpha_t) Q^\pi(x,a) + \alpha_t (r + \gamma Q^\pi(x',a')). \qquad (11.19)$$

[6] Note that for deterministic policies π, $Q^\pi(x',a') = Q^\pi(x',\pi(x')) = V^\pi(x')$ if the transitions are obtained by following policy π.

This algorithm is known as *SARSA* (short for *state-action-reward-state-action*). Similar convergence guarantees to those of TD-learning can also be derived for SARSA.

Theorem 11.11 (Convergence of SARSA, Singh et al. (2000))**.** *If* $(\alpha_t)_{t \in \mathbb{N}_0}$ *satisfies the RM-conditions* (A.60) *and all state-action pairs are chosen infinitely often, then* Q^π *converges to* q^π *with probability* 1*.*

The policy iteration scheme to identify the optimal policy can be outlined as follows: In each iteration t, we estimate the value function q^{π_t} of policy π_t with the estimate Q^{π_t} obtained from SARSA. We then choose the greedy policy with respect to Q^{π_t} as the next policy π_{t+1}. However, due to the on-policy nature of SARSA, we cannot reuse any data between the iterations. Moreover, it turns out that in practice, when using only finitely many samples, this form of greedily optimizing Markov decision processes does not explore enough. At least partially, this can be compensated for by injecting noise when choosing the next action, e.g., by following an ε-greedy policy or using softmax exploration.

11.4.3 *Off-policy Value Estimation*

Consider the following slight adaptation of the derivation of SARSA (11.18),

$$\begin{aligned} q^\pi(x,a) &= r(x,a) + \gamma \sum_{x' \in X} p(x' \mid x,a) \sum_{a' \in A} \pi(a' \mid x') q^\pi(x',a') && \text{using Bellman's expectation equation (10.15)} \\ &= \mathbb{E}_{R_0,X_1}\left[R_0 + \gamma \sum_{a' \in A} \pi(a' \mid X_1) q^\pi(X_1,a') \;\middle|\; X_0 = x, A_0 = a \right] \qquad (11.20) && \text{interpreting the above expression as an expectation over } R_0 \text{ and } X_1 \\ &\approx r + \gamma \sum_{a' \in A} \pi(a' \mid x') q^\pi(x',a'), \qquad (11.21) && \text{Monte Carlo approximation with a single sample} \end{aligned}$$

where the agent observed the transition (x, a, r, x'). This yields the update rule,

$$Q^\pi(x,a) \leftarrow (1-\alpha_t)Q^\pi(x,a) + \alpha_t\left(r + \gamma \sum_{a' \in A} \pi(a' \mid x')Q^\pi(x',a')\right). \tag{11.22}$$

This adapted update rule *explicitly* chooses the subsequent action a' according to policy π whereas SARSA absorbs this choice into the Monte Carlo approximation. The algorithm has analogous convergence guarantees to those of SARSA.

Crucially, this algorithm is off-policy. That is, we can use transitions that were obtained according to *any* policy to estimate the value of a fixed policy π, which we may have never used! Perhaps this seems contradictory at first, but it is not. As noted, the key difference to the on-policy TD-learning and SARSA is that our estimate of the Q-function explicitly keeps track of the next-performed action. It does so for any action in any state. Moreover, note that the transitions that are due to the dynamics model and rewards are unaffected by the used policy. They merely depend on the originating state-action pair. We can therefore use the instances where other policies played action $\pi(x)$ in state x to estimate the performance of π.

11.4.4 Q-learning: Off-policy Control

It turns out that there is a way to estimate the value function of the optimal policy directly. Recall from Bellman's theorem (10.31) that the optimal policy $\pi^\star$ can be characterized in terms of the optimal state-action value function $q^\star$,

$$\pi^\star(x) = \arg\max_{a \in A} q^\star(x,a).$$

$\pi^\star$ corresponds to greedily maximizing the value function.

Analogously to our derivation of SARSA (11.18), only using Bellman's theorem (10.32) in place of Bellman's expectation equation (10.15), we obtain,

$$\begin{aligned} q^\star(x,a) &= r(x,a) + \gamma \sum_{x' \in X} p(x' \mid x,a) \max_{a' \in A} q^\star(x',a') && \text{using that the Q-function is a fixed-point of the Bellman update, see Bellman's theorem (10.32)} \\ &= \mathbb{E}_{R_0, X_1}\left[R_0 + \gamma \max_{a' \in A} q^\star(X_1, a') \;\middle|\; X_0 = x, A_0 = a\right] && \text{interpreting the above expression as an expectation over } R_0 \text{ and } X_1 \quad (11.23) \\ &\approx r + \gamma \max_{a' \in A} q^\star(x',a'), && \text{Monte Carlo approximation with a single sample} \quad (11.24) \end{aligned}$$

where the agent observed the transition (x, a, r, x'). Using a bootstrapping estimate $Q^\star$ for $q^\star$, we obtain a structurally similar algorithm to

TD-learning and SARSA — only for estimating the optimal Q-function directly! This algorithm is known as *Q-learning*. Whereas we have seen that the optimal policy can be found using SARSA in a policy-iteration-like scheme, Q-learning is conceptually similar to value iteration.

Algorithm 11.12: Q-learning

initialize $Q^\star(x,a)$ arbitrarily (e.g., as $\mathbf{0}$)
for $t = 0$ **to** ∞ **do**
  observe the transition (x, a, r, x')
  $Q^\star(x,a) \leftarrow (1-\alpha_t)Q^\star(x,a) + \alpha_t(r + \gamma \max_{a' \in A} Q^\star(x',a'))$
  // (11.25)

Similarly to TD-learning, the update rule can also be expressed as

$$Q^\star(x,a) \leftarrow Q^\star(x,a) + \alpha_t \left(r + \gamma \max_{a' \in A} Q^\star(x',a') - Q^\star(x,a) \right). \quad (11.26)$$

Crucially, the Monte Carlo approximation of Equation (11.23) does not depend on the policy. Thus, Q-learning is an off-policy method.

Theorem 11.13 (Convergence of Q-learning, Jaakkola et al. (1993))**.** *If* $(\alpha_t)_{t \in \mathbb{N}_0}$ *satisfies the RM-conditions* (A.60) *and all state-action pairs are chosen infinitely often (that is, the sequence of policies used to obtain the transitions is GLIE), then* $Q^\star$ *converges to* $q^\star$ *with probability* 1.

It can be shown that with probability at least $1 - \delta$, Q-learning converges to an ϵ-optimal policy in a number of steps that is polynomial in $\log |X|$, $\log |A|$, $1/\epsilon$ and $\log 1/\delta$ (Even-Dar et al., 2003).

11.4.5 *Optimistic Q-learning*

The next natural question is how to effectively trade exploration and exploitation to both visit all state-action pairs many times, but also attain a high reward.

However, as we have seen in Section 11.3, random "uninformed" exploration like ε-greedy and softmax exploration explores the state space very slowly. We therefore return to the principle of *optimism in the face of uncertainty*, which already led us to the $R_{\max}$ algorithm in the model-based setting. We will now additionally assume that the rewards are non-negative, that is, $0 \leq r(x,a) \leq R_{\max}$ $(\forall x \in X, a \in A)$. It turns out that a similar algorithm to $R_{\max}$ also exists for (model-free) Q-learning: it is called *optimistic Q-learning* and shown in Algorithm 11.14.

Algorithm 11.14: Optimistic Q-learning

initialize $Q^\star(x,a) = V_{\max} \prod_{t=1}^{T_{\text{init}}} (1-\alpha_t)^{-1}$
for $t = 0$ **to** ∞ **do**
  pick action $a_t = \arg\max_{a \in A} Q^\star(x,a)$ and observe the transition (x, a_t, r, x')
  $Q^\star(x,a_t) \leftarrow (1-\alpha_t) Q^\star(x,a_t) + \alpha_t (r + \gamma \max_{a' \in A} Q^\star(x',a'))$ // (11.27)

Here,

$$V_{\max} \doteq \frac{R_{\max}}{1-\gamma} \geq \max_{x \in X, a \in A} q^\star(x,a),$$

is an upper bound on the discounted return and T_{init} is some initialization time. Intuitively, the initialization of $Q^\star$ corresponds to the best-case long-term reward, assuming that all individual rewards are upper bounded by $R_{\max}$. This is shown by the following lemma.

Lemma 11.15. *Denote by $Q_t^\star$, the approximation of $q^\star$ attained in the t-th iteration of optimistic Q-learning. Then, for any state-action pair (x,a) and iteration t such that $N(a \mid x) \leq T_{\text{init}}$,*[7]

$$Q_t^\star(x,a) \geq V_{\max} \geq q^\star(x,a). \tag{11.28}$$

[7] $N(a \mid x)$ is the number of times action a is performed in state x.

Proof. We write $\beta_\tau \doteq \prod_{i=1}^{\tau} (1-\alpha_i)$ and $\eta_{i,\tau} \doteq \alpha_i \prod_{j=i+1}^{\tau} (1-\alpha_j)$. Using the update rule of optimistic Q-learning (11.27), we have

$$Q_t^\star(x,a) = \beta_{N(a|x)} Q_0^\star(x,a) + \sum_{i=1}^{N(a|x)} \eta_{i,N(a|x)} (r + \gamma \max_{a_i \in A} Q_{t_i}^\star(x_i, a_i)) \tag{11.29}$$

where x_i is the next state arrived at time t_i when action a is performed the i-th time in state x.

Using the assumption that the rewards are non-negative, from Equation (11.29) and $Q_0^\star(x,a) = {}^{V_{\max}}/_{\beta_{T_{\text{init}}}}$, we immediately have

$$\begin{aligned} Q_t^\star(x,a) &\geq \frac{\beta_{N(a|x)}}{\beta_{T_{\text{init}}}} V_{\max} \\ &\geq V_{\max}. \end{aligned}$$

using $N(a \mid x) \leq T_{\text{init}}$

□

Now, if T_{init} is chosen large enough, it can be shown that optimistic Q-learning converges quickly to an optimal policy.

Theorem 11.16 (Convergence of optimistic Q-learning, Even-Dar and Mansour (2001))**.** *With probability at least $1-\delta$, optimistic Q-learning obtains an ϵ-optimal policy after a number of steps that is polynomial in $|X|$,*

$|A|$, $1/\epsilon$, $\log 1/\delta$, and $R_{\max}$ where the initialization time T_{init} is upper bounded by a polynomial in the same coefficients.

Note that for Q-learning, we still need to store $Q^\star(x, a)$ for any state-action pair in memory. Thus, Q-learning requires $O(nm)$ memory. During each transition, we need to compute

$$\max_{a \in A} Q^\star(x', a')$$

once. If we run Q-learning for T iterations, this yields a time complexity of $O(Tm)$. Crucially, for sparse Markov decision processes where, in most states, only few actions are permitted, each iteration of Q-learning can be performed in (virtually) constant time. This is a big improvement of the quadratic (in the number of states) performance of the model-based $R_{\max}$ algorithm.

Discussion

We have seen that both the model-based $R_{\max}$ algorithm and the model-free Q-learning take time polynomial in the number of states $|X|$ and the number of actions $|A|$ to converge. While this is acceptable in small grid worlds, this is completely unacceptable for large state and action spaces.

Often, domains are continuous, for example when modeling beliefs about states in a partially observable environment. Also, in many structured domains (e.g., chess or multiagent planning), the size of the state and action space is exponential in the size of the input. In the final two chapters, we will therefore explore how model-free and model-based methods can be used (approximately) in such large domains.

Problems

11.1. Q-learning.

Assume the following grid world, where from state A the agent can go to the right and down, and from state B to the left and down. From states G_1 and G_2 the only action is to exit. The agent receives a reward ($+10$ or $+1$) only when exiting.

Rewards		States	
0	0	A	B
$+10$	$+1$	G_1	G_2

We assume the discount factor $\gamma = 1$ and that all actions are deterministic.

1. We observe the following two episodes:

Episode 1

x	a	x'	r
A	$\downarrow$	G_1	0
G_1	exit		10

Episode 2

x	a	x'	r
B	$\leftarrow$	A	0
A	$\downarrow$	G_1	0
G_1	exit		10

   Assume $\alpha = 0.3$, and that Q-values of all non-terminal states are initialized to 0.5. What are the Q-values

$$Q^\star(A, \downarrow), \quad Q^\star(G_1, \text{exit}), \quad Q^\star(G_2, \text{exit})$$

   learned by executing Q-learning with the above episodes?
2. Will Q-learning converge to $q^\star$ for all state-action pairs (x, a) if we repeat episode 1 and episode 2 infinitely often? If not, design a sequence of episodes that leads to convergence.
3. How does the choice of initial Q-values influence the convergence of the Q-learning algorithm when episodes are obtained off-policy?
4. Determine $v^\star$ for all states.

12

Model-free Reinforcement Learning

In the previous chapter, we have seen methods for tabular settings. Our goal now is to extend the model-free methods like TD-learning and Q-learning to large state-action spaces $\mathcal{X}$ and $\mathcal{A}$. We have seen that a crucial bottleneck of these methods is the parameterization of the value function. If we want to store the value function in a table, we need at least $O(|\mathcal{X}|)$ space. If we learn the Q function, we even need $O(|\mathcal{X}| \cdot |\mathcal{A}|)$ space. Also, for large state-action spaces, the time required to compute the value function for every state-action pair exactly will grow polynomially in the size of the state-action space. Hence, a natural idea is to learn *approximations* of these functions with a low-dimensional parameterization. Such approximations were the focus of the first few chapters and are, in fact, the key idea behind methods for reinforcement learning in large state-action spaces.

12.1 *Tabular Reinforcement Learning as Optimization*

To begin with, let us reinterpret the model-free methods from the previous section, TD-learning and Q-learning, as solving an optimization problem, where each iteration corresponds to a single gradient update. We will focus on TD-learning here, but the same interpretation applies to Q-learning. Recall the update rule of TD-learning (11.15),

$$V^\pi(x) \leftarrow (1-\alpha_t)V^\pi(x) + \alpha_t(r + \gamma V^\pi(x')).$$

Note that this looks just like the update rule of an optimization algorithm! We can parameterize our estimates V^π with parameters $\boldsymbol{\theta}$ that are updated according to the gradient of some loss function, assuming fixed bootstrapping estimates. In particular, in the tabular setting (i.e., over a finite domain), we can parameterize the value function exactly by learning a separate parameter for each state,

$$\boldsymbol{\theta} \doteq [\boldsymbol{\theta}(1), \dots, \boldsymbol{\theta}(n)], \quad V^\pi(x;\boldsymbol{\theta}) \doteq \boldsymbol{\theta}(x). \tag{12.1}$$

To re-derive the above update rule as a gradient update, let us consider the following loss function,

$$\bar{\ell}(\boldsymbol{\theta}; x, r) \doteq \frac{1}{2}(v^\pi(x) - \boldsymbol{\theta}(x))^2 \tag{12.2}$$

$$= \frac{1}{2}\Big(r + \gamma \mathbb{E}_{x'|x,\pi(x)}\big[v^\pi(x')\big] - \boldsymbol{\theta}(x)\Big)^2 \tag{12.3}$$

using Bellman's expectation equation (10.11)

Note that this loss corresponds to a standard squared loss of the difference between the parameter $\boldsymbol{\theta}(x)$ and the label $v^\pi(x)$ we want to learn.

We can now find the gradient of this loss. Elementary calculations yield,

$$\nabla_{\boldsymbol{\theta}(x)} \bar{\ell}(\boldsymbol{\theta}; x, r) = \boldsymbol{\theta}(x) - \Big(r + \gamma \mathbb{E}_{x'|x,\pi(x)}\big[v^\pi(x')\big]\Big). \tag{12.4}$$

Now, we cannot compute this derivative because we cannot compute the expectation. Firstly, the expectation is over the true value function which is unknown to us. Secondly, the expectation is over the transition model which we are trying to avoid in model-free methods.

To resolve the first issue, analogously to TD-learning, instead of learning the true value function v^π which is unknown, we learn the bootstrapping estimate V^π. Recall that the core principle behind bootstrapping as discussed in Section 11.4.1 is that this bootstrapping estimate V^π is *treated* as if it were independent of the current estimate of the value function $\boldsymbol{\theta}$. To emphasize this, we write $V^\pi(x; \boldsymbol{\theta}^{\text{old}}) \approx v^\pi(x)$ where $\boldsymbol{\theta}^{\text{old}} = \boldsymbol{\theta}$ but $\boldsymbol{\theta}^{\text{old}}$ is treated as a constant with respect to $\boldsymbol{\theta}$.[1]

[1] That is, the bootstrapping estimate $V^\pi(x; \boldsymbol{\theta}^{\text{old}})$ is assumed to be constant with respect to $\boldsymbol{\theta}(x)$ in the same way that $v^\pi(x)$ is constant with respect to $\boldsymbol{\theta}(x)$.

If we were not using the bootstrapping estimate, the following derivation of the gradient of the loss would not be this simple.

To resolve the second issue, analogously to the introduction of TD-learning in the previous chapter, we will use a Monte Carlo estimate using a single sample. Recall that this is only possible because the transitions are conditionally independent given the state-action pair.

Remark 12.1: Sample (in)efficiency of model-free methods

Taking these two shortcuts are two of the main reasons why model-free methods such as TD-learning and Q-learning are usually *sample inefficient*. This is because using a bootstrapping estimate leads to "(initially) incorrect" and "unstable" targets of the optimization problem,[2] and Monte Carlo estimation with a single sample leads to a large variance. Recall that the theoretical guarantees for model-free methods in the tabular setting therefore required that all state-action pairs are visited infinitely often.

[2] We explore this in some more capacity in Section 12.2.1 where we cover heuristic approaches to alleviate this problem to some degree.

Using the aforementioned shortcuts, let us define the loss ℓ after observing the single transition (x, a, r, x'),

$$\ell(\boldsymbol{\theta}; x, r, x') \doteq \frac{1}{2}\Big(r + \gamma \boldsymbol{\theta}^{\text{old}}(x') - \boldsymbol{\theta}(x)\Big)^2. \tag{12.5}$$

We define the gradient of this loss with respect to $\boldsymbol{\theta}(x)$ as

$$\begin{aligned}\delta_{\text{TD}} &\doteq \boldsymbol{\nabla}_{\boldsymbol{\theta}(x)}\, \ell(\boldsymbol{\theta}; x, r, x') \\ &= \boldsymbol{\theta}(x) - \Big(r + \gamma \boldsymbol{\theta}^{\text{old}}(x')\Big). \end{aligned} \tag{12.6}$$

This error term is also called *temporal-difference (TD) error*. The temporal difference error compares the previous estimate of the value function to the bootstrapping estimate of the value function. We know from the law of large numbers (A.36) that Monte Carlo averages are unbiased.[3] We therefore have,

$$\mathbb{E}_{x'|x,\pi(x)}[\delta_{\text{TD}}] \approx \boldsymbol{\nabla}_{\boldsymbol{\theta}(x)}\, \overline{\ell}(\boldsymbol{\theta}; x, r). \tag{12.7}$$

[3] Crucially, the samples are unbiased with respect to the approximate label in terms of the bootstrapping estimate only. Due to bootstrapping the value function, the estimates are not unbiased with respect to the true value function. Moreover, the variance of each individual estimation of the gradient is large, as we only consider a single transition.

Naturally, we can use these unbiased gradient estimates with respect to the loss $\overline{\ell}$ to perform stochastic gradient descent. This yields the update rule,

$$V^\pi(x; \boldsymbol{\theta}) = \boldsymbol{\theta}(x) \leftarrow \boldsymbol{\theta}(x) - \alpha_t \delta_{\text{TD}} \tag{12.8}$$

using stochastic gradient descent with learning rate α_t, see Algorithm A.29

$$= (1 - \alpha_t)\boldsymbol{\theta}(x) + \alpha_t \Big(r + \gamma \boldsymbol{\theta}^{\text{old}}(x')\Big)$$

using the definition of the temporal difference error (12.6)

$$= (1 - \alpha_t) V^\pi(x; \boldsymbol{\theta}) + \alpha_t \Big(r + \gamma V^\pi(x'; \boldsymbol{\theta}^{\text{old}})\Big). \tag{12.9}$$

substituting $V^\pi(x; \boldsymbol{\theta})$ for $\boldsymbol{\theta}(x)$

Observe that this gradient update coincides with the update rule of TD-learning (11.15). Therefore, TD-learning is essentially performing stochastic gradient descent using the TD-error as an unbiased gradient estimate.[4] Crucially, TD-learning performs stochastic gradient descent with respect to the bootstrapping estimate of the value function V^π and not the true value function v^π! Stochastic gradient descent with a bootstrapping estimate is also called *stochastic semi-gradient descent*. Importantly, the optimization target $r + \gamma V^\pi(x'; \boldsymbol{\theta}^{\text{old}})$ from the loss ℓ is now *moving* between iterations which introduces some practical challenges we will discuss in Section 12.2.1. We have seen in the previous chapter that using a bootstrapping estimate still guarantees (asymptotic) convergence to the true value function.

[4] An alternative interpretation is that TD-learning performs gradient descent with respect to the loss ℓ.

12.2 *Value Function Approximation*

To scale to large state spaces, it is natural to approximate the value function using a parameterized model, $V(\boldsymbol{x}; \boldsymbol{\theta})$ or $Q(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})$. You may think of this as a regression problem where we map state(-action) pairs to a real number. Recall from the previous section that this is a strict generalization of the tabular setting, as we could use a separate parameter to learn the value function for each individual state-action pair. Our goal for large state-action spaces is to exploit the smoothness properties[5] of the value function to condense the representation.

[5] That is, the value function takes similar values in "similar" states.

A straightforward approach is to use a linear function approximation with the hand-designed feature map $\boldsymbol{\phi}$,

$$Q(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) \doteq \boldsymbol{\theta}^\top \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{a}). \tag{12.10}$$

A common alternative is to use a deep neural network to learn these features instead. Doing so is also known as *deep reinforcement learning*.[6]

[6] Note that often non-Bayesian deep learning (i.e., point estimates of the weights of a neural network) is applied. In the final chapter, Chapter 13, we will explore the benefits of using Bayesian deep learning.

We will now apply the derivation from the previous section directly to Q-learning. For Q-learning, after observing the transition $(\boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}')$, the loss function is given as

$$\ell(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}') \doteq \frac{1}{2}\left(r + \gamma \max_{\boldsymbol{a}' \in \mathcal{A}} Q^\star(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})\right)^2. \tag{12.11}$$

Here, we simply use Bellman's optimality equation (10.32) to estimate $q^\star(\boldsymbol{x}, \boldsymbol{a})$, instead of the estimation of $v^\pi(\boldsymbol{x})$ using Bellman's expectation equation for TD-learning. The difference between the current approximation and the optimization target,

$$\delta_{\text{B}} \doteq r + \gamma \max_{\boldsymbol{a}' \in \mathcal{A}} Q^\star(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}), \tag{12.12}$$

is called the *Bellman error*. Analogously to TD-learning,[7] we obtain the gradient update,

[7] compare to Equation (12.8)

$$\begin{aligned}
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \alpha_t \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}') && (12.13)\\
&= \boldsymbol{\theta} - \alpha_t \nabla_{\boldsymbol{\theta}} \frac{1}{2}\left(r + \gamma \max_{\boldsymbol{a}' \in \mathcal{A}} Q^\star(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})\right)^2 && \text{using the definition of } \ell \text{ (12.11)}\\
&= \boldsymbol{\theta} + \alpha_t \delta_{\text{B}} \nabla_{\boldsymbol{\theta}} Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}). && (12.14) \quad \text{using the chain rule}
\end{aligned}$$

When using a neural network to learn $Q^\star$, we can use automatic differentiation to obtain unbiased gradient estimates. In the case of linear function approximation, we can compute the gradient exactly,

$$\begin{aligned}
&= \boldsymbol{\theta} + \alpha_t \delta_{\text{B}} \nabla_{\boldsymbol{\theta}} \boldsymbol{\theta}^\top \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{a}) && \text{using the linear approximation of } Q^\star \text{ (12.10)}\\
&= \boldsymbol{\theta} + \alpha_t \delta_{\text{B}} \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{a}). && (12.15)
\end{aligned}$$

In the tabular setting, this algorithm is identical to Q-learning and, in particular, converges to the true Q-function $q^\star$.[8] There are few such results in the approximate setting. Usage in practice indicates that using an approximation of the value function "should be fine" when a "rich-enough" class of functions is used.

[8] see Theorem 11.13

12.2.1 *Heuristics*

The vanilla stochastic semi-gradient descent is very slow. In this subsection, we will discuss some "tricks of the trade" to improve its performance.

Stabilizing optimization targets: There are mainly two problems. The first problem is that, as mentioned previously, the bootstrapping estimate changes after each iteration. As we are trying to learn an approximate value function that depends on the bootstrapping estimate, this means that the optimization target is "moving" between iterations. In practice, moving targets lead to stability issues. The first family of techniques we discuss here aim to "stabilize" the optimization targets.

One such technique is called *neural fitted Q-iteration* or *deep Q-networks* (DQN) (Mnih et al., 2015). DQN updates the neural network used for the approximate bootstrapping estimate infrequently to maintain a constant optimization target across multiple episodes. How this is implemented exactly varies. One approach is to clone the neural network and maintain one changing neural network ("online network") for the most recent estimate of the Q-function which is parameterized by $\boldsymbol{\theta}$, and one fixed neural network ("target network") used as the target which is parameterized by $\boldsymbol{\theta}^{\text{old}}$ and which is updated infrequently.

This can be implemented by maintaining a data set $\mathcal{D}$ of observed transitions (the so-called *replay buffer*) and then "every once in a while" (e.g., once $|\mathcal{D}|$ is large enough) solving a regression problem, where the labels are determined by the target network. This yields a loss term where the target is fixed across all transitions in the replay buffer $\mathcal{D}$,

$$\ell_{\text{DQN}}(\boldsymbol{\theta}; \mathcal{D}) \doteq \frac{1}{2} \sum_{(\boldsymbol{x},\boldsymbol{a},r,\boldsymbol{x}') \in \mathcal{D}} \left(r + \gamma \max_{\boldsymbol{a}' \in \mathcal{A}} Q^\star(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) \right)^2. \tag{12.16}$$

compare to the Q-learning loss (12.11)

The loss can also be interpreted (in an online sense) as performing regular Q-learning with the modification that the target network $\boldsymbol{\theta}^{\text{old}}$ is not updated to $\boldsymbol{\theta}$ after every observed transition, but instead only after observing $|\mathcal{D}|$-many transitions. This technique is known as *experience replay*. Another approach is *Polyak averaging* where the target network is gradually "nudged" by the neural network used to estimate the Q-function.

Maximization bias: Now, observe that the estimates $Q^\star$ are noisy estimates of $q^\star$ and consider the term,

$$\max_{\boldsymbol{a}' \in \mathcal{A}} q^\star(\boldsymbol{x}', \boldsymbol{a}') \approx \max_{\boldsymbol{a}' \in \mathcal{A}} Q^\star(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}),$$

from the loss function (12.16). This term maximizes a *noisy* estimate of $q^\star$, which leads to a *biased* estimate of $\max q^\star$ as can be seen in Figure 12.1. The fact that the update rules of Q-learning and DQN are affected by inaccuracies (i.e., noise in the estimates) of the learned Q-function is known as the "*maximization bias*".

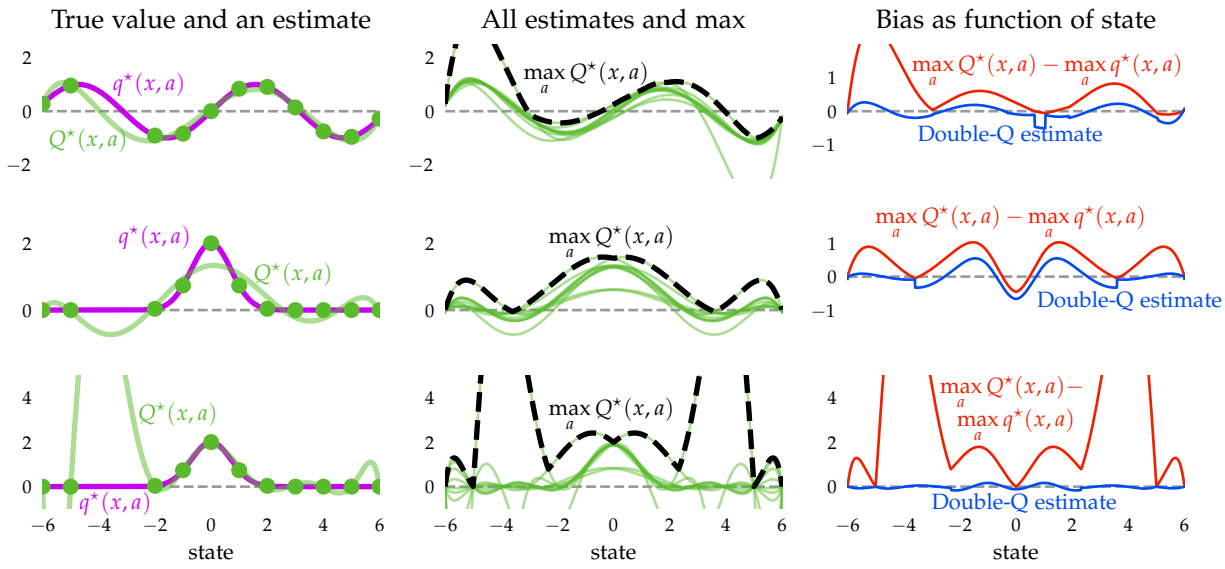


Figure 12.1: Illustration of overestimation during learning. In each state (x-axis), there are 10 actions. The left column shows the true values $v^\star(x)$ (purple line). All true action values are defined by $q^\star(x,a) \doteq v^\star(x)$. The green line shows estimated values $Q^\star(x,a)$ for one action as a function of state, fitted to the true value at several sampled states (green dots). The middle column plots show all the estimated values (green), and the maximum of these values (dashed black). The maximum is higher than the true value (purple, left plot) almost everywhere. The right column plots show the difference in red. The blue line in the right plots is the estimate used by Double Q-learning with a second set of samples for each state. The blue line is much closer to zero, indicating less bias. The three rows correspond to different true functions (left, purple) or capacities of the fitted function (left, green). Reproduced with permission from "Deep reinforcement learning with double q-learning" (Van Hasselt et al., 2016).

Double DQN (DDQN) is an algorithm that addresses this maximization bias (Van Hasselt et al., 2016). Instead of picking the optimal action with respect to the old network, it picks the optimal action with respect to the new network,

$$\ell_{\text{DDQN}}(\boldsymbol{\theta}; \mathcal{D}) \doteq \frac{1}{2} \sum_{(\boldsymbol{x},\boldsymbol{a},r,\boldsymbol{x}') \in \mathcal{D}} \Big(r + \gamma Q^\star(\boldsymbol{x}', \boldsymbol{a}^\star(\boldsymbol{x}'; \boldsymbol{\theta}); \boldsymbol{\theta}^{\text{old}}) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) \Big)^2 \tag{12.17}$$

$$\text{where} \quad \boldsymbol{a}^\star(\boldsymbol{x}'; \boldsymbol{\theta}) \doteq \arg\max_{\boldsymbol{a}' \in \mathcal{A}} Q^\star(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}). \tag{12.18}$$

Intuitively, this change ensures that the evaluation of the target network is consistent with the updated Q-function, which makes the algorithm more robust to noise.

Similarly to DQN, this can also be interpreted as the online update,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_t \Big(r + \gamma Q^\star(\boldsymbol{x}', \boldsymbol{a}^\star(\boldsymbol{x}'; \boldsymbol{\theta}); \boldsymbol{\theta}^{\text{old}}) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) \Big) \nabla_{\boldsymbol{\theta}} Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) \tag{12.19}$$

after observing a single transition $(\boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}')$ where while differentiating, $\boldsymbol{a}^\star(\boldsymbol{x}'; \boldsymbol{\theta})$ is treated as constant with respect to $\boldsymbol{\theta}$. $\boldsymbol{\theta}^{\text{old}}$ is then updated to $\boldsymbol{\theta}$ after observing $|\mathcal{D}|$-many transitions.

12.3 *Policy Approximation*

Q-learning defines a policy implicitly by

$$\pi^\star(\boldsymbol{x}) \doteq \arg\max_{\boldsymbol{a} \in \mathcal{A}} Q^\star(\boldsymbol{x}, \boldsymbol{a}). \tag{12.20}$$

Q-learning also maximizes over the set of all actions in its update step while learning the Q-function. This is intractable for large and, in particular, continuous action spaces. A natural idea to escape this limitation is to immediately learn an approximate parameterized policy,

$$\boldsymbol{\pi}^\star(\boldsymbol{x}) \approx \boldsymbol{\pi}(\boldsymbol{x};\boldsymbol{\varphi}) \doteq \boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x}). \tag{12.21}$$

Methods that find an approximate policy are also called *policy search methods* or *policy gradient methods*.

Whereas with Q-learning, exploration can be encouraged by using an ε-greedy policy, softmax exploration, or an optimistic initialization, we will see later that policy gradient methods fundamentally rely on randomized policies for exploration.

Remark 12.2: Notation
We refer to deterministic policies $\boldsymbol{\pi}$ in bold, as they can be interpreted as vector-valued functions from $\mathcal{X}$ to $\mathcal{A}$. We still refer to randomized policies by π, as for each state $\boldsymbol{x} \in \mathcal{X}$ they are represented as a PDF over actions $\mathcal{A}$.

In particular, we denote by $\pi_{\boldsymbol{\varphi}}(\boldsymbol{a} \mid \boldsymbol{x})$ the probability (density) of playing action $\boldsymbol{a}$ when in state $\boldsymbol{x}$ according to $\pi_{\boldsymbol{\varphi}}$.

12.3.1 Estimating Policy Values

We will begin by attributing a "value" to a policy. Recall the definition of the discounted payoff G_t from time t, which we are aiming to maximize,

$$G_t = \sum_{m=0}^{\infty} \gamma^m R_{t+m}. \tag{see Equation (10.5)}$$

We define $G_{t:T}$ to be the *bounded discounted payoff* until time T,

$$G_{t:T} \doteq \sum_{m=0}^{T-1-t} \gamma^m R_{t+m}. \tag{12.22}$$

Based on these two random variables, we can define the policy value function:

Definition 12.3 (Policy value function)**.** The *policy value function*,

$$J(\pi) \doteq \mathbb{E}_\pi[G_0] = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_t\right], \tag{12.23}$$

measures the expected discounted payoff of policy π.[9] We also define the bounded variant,

$$J_T(\pi) \doteq \mathbb{E}_\pi[G_{0:T}] = \mathbb{E}_\pi\left[\sum_{t=0}^{T-1} \gamma^t R_t\right]. \tag{12.24}$$

[9] We neglect here that implicitly one also averages over the initial state if this state is not fixed.

For simplicity, we will abbreviate $J(\boldsymbol{\varphi}) \doteq J(\pi_{\boldsymbol{\varphi}})$.

Remark 12.4: Notation of policy value function

We adopt the more common notation $J(\pi)$ for the policy value function, as opposed to $j(\pi)$, which would be consistent with our notation of the (true) value functions v^π, q^π. So don't be confused by this: just like the value functions v^π, q^π, the policy value function $J(\pi)$ is a deterministic object, measuring the mean discounted payoff. We will use $\widehat{J}(\pi)$ to refer to our estimates of the policy value function.

Naturally, we want to maximize $J(\boldsymbol{\varphi})$. That is, we want to solve

$$\boldsymbol{\varphi}^\star \doteq \arg\max_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi}) \tag{12.25}$$

which is a non-convex optimization problem. Let us see how $J(\boldsymbol{\varphi})$ can be evaluated to understand the optimization problem better. We will again use a Monte Carlo estimate. Recall that a fixed $\boldsymbol{\varphi}$ induces a unique Markov chain, which can be simulated. In the episodic setting, each episode (also called *rollout*) of length T yields an independently sampled trajectory,

$$\tau^{(i)} \doteq ((\boldsymbol{x}_0^{(i)}, \boldsymbol{a}_0^{(i)}, r_0^{(i)}, \boldsymbol{x}_1^{(i)}), (\boldsymbol{x}_1^{(i)}, \boldsymbol{a}_1^{(i)}, r_1^{(i)}, \boldsymbol{x}_2^{(i)}), \dots) \tag{12.26}$$

Simulating m rollouts yields the samples $\tau^{(1)}, \dots, \tau^{(m)} \sim \Pi_{\boldsymbol{\varphi}}$, where $\Pi_{\boldsymbol{\varphi}}$ is the distribution of all trajectories (i.e., rollouts) of the Markov chain induced by policy $\pi_{\boldsymbol{\varphi}}$. We denote the (bounded) discounted payoff of the i-th rollout by

$$g_{0:T}^{(i)} \doteq \sum_{t=0}^{T-1} \gamma^t r_t^{(i)} \tag{12.27}$$

where $r_t^{(i)}$ is the reward at time t of the i-th rollout. Using a Monte Carlo approximation, we can then estimate $J_T(\boldsymbol{\varphi})$. Moreover, due to the exponential discounting of future rewards, it is reasonable to approximate the policy value function using bounded trajectories,

$$J(\boldsymbol{\varphi}) \approx J_T(\boldsymbol{\varphi}) \approx \widehat{J}_T(\boldsymbol{\varphi}) \doteq \frac{1}{m} \sum_{i=1}^{m} g_{0:T}^{(i)}. \tag{12.28}$$

12.3.2 *Policy Gradient*

Policy gradient methods solve the above optimization problem (12.25) by stochastic gradient ascent on the policy parameter $\boldsymbol{\varphi}$:

$$\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} + \eta \nabla_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi}). \tag{12.29}$$

How can we compute the policy gradient? Let us first formally define the distribution over trajectories $\Pi_{\boldsymbol{\varphi}}$ that we introduced in the previous section. We can specify the probability of a specific trajectory τ under a policy $\pi_{\boldsymbol{\varphi}}$ by

$$\Pi_{\boldsymbol{\varphi}}(\tau) = p(\boldsymbol{x}_0) \prod_{t=0}^{T-1} \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t) p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, \boldsymbol{a}_t). \tag{12.30}$$

For optimizing $J(\boldsymbol{\varphi})$ we need to obtain unbiased gradient estimates:

$$\nabla_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi}) \approx \nabla_{\boldsymbol{\varphi}} J_T(\boldsymbol{\varphi}) = \nabla_{\boldsymbol{\varphi}} \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}}[G_{0:T}]. \tag{12.31}$$

Note that the expectation integrates over the measure $\Pi_{\boldsymbol{\varphi}}$, which depends on the parameter $\boldsymbol{\varphi}$. Thus, we cannot move the gradient operator inside the expectation as we have often done previously (cf. Appendix A.1.5). This should remind you of the reparameterization trick (see Equation (5.62)) that we used to solve a similar gradient in the context of variational inference. In this context, however, we cannot apply the reparameterization trick.[10] Fortunately, there is another way of estimating this gradient.

[10] This is because the distribution $\Pi_{\boldsymbol{\varphi}}$ is generally not reparameterizable. We will, however, see that reparameterization gradients are also useful in reinforcement learning. See, e.g., Sections 12.5.1 and 13.1.2.

Theorem 12.5 (Score gradient estimator)**.** *Under some regularity assumptions, we have*

$$\nabla_{\boldsymbol{\varphi}} \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}}[G_0] = \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}}\left[G_0 \nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau)\right]. \tag{12.32}$$

This estimator of the gradient is called the score gradient estimator*.*

Proof. To begin with, let us look at the so-called *score function* of the distribution $\Pi_{\boldsymbol{\varphi}}$, $\nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau)$. Using the chain rule, the score function can be expressed as

$$\nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau) = \frac{\nabla_{\boldsymbol{\varphi}} \Pi_{\boldsymbol{\varphi}}(\tau)}{\Pi_{\boldsymbol{\varphi}}(\tau)} \tag{12.33}$$

and by rearranging the terms, we obtain

$$\nabla_{\boldsymbol{\varphi}} \Pi_{\boldsymbol{\varphi}}(\tau) = \Pi_{\boldsymbol{\varphi}}(\tau) \nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau). \tag{12.34}$$

This is called the *score function trick*.[11]

[11] We have already applied this "trick" in Problem 6.9.

Now, assuming that state and action spaces are continuous, we obtain

$$\begin{aligned}
\nabla_{\boldsymbol{\varphi}} \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}}[G_0] &= \nabla_{\boldsymbol{\varphi}} \int \Pi_{\boldsymbol{\varphi}}(\tau) \cdot G_0 \, d\tau && \text{using the definition of expectation (1.19)} \\
&= \int \nabla_{\boldsymbol{\varphi}} \Pi_{\boldsymbol{\varphi}}(\tau) \cdot G_0 \, d\tau && \text{using the regularity assumptions to swap gradient and integral} \\
&= \int G_0 \cdot \Pi_{\boldsymbol{\varphi}}(\tau) \nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau) \, d\tau && \text{using the score function trick (12.34)} \\
&= \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}}\left[G_0 \nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau)\right]. && \text{interpreting the integral as an expectation over } \Pi_{\boldsymbol{\varphi}}
\end{aligned}$$

□

Intuitively, maximizing $J(\boldsymbol{\varphi})$ increases the probability of policies with high returns and decreases the probability of policies with low returns.

To use the score gradient estimator for estimating the gradient, we need to compute $\nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau)$.

$$
\begin{aligned}
&\nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau) \\
&= \nabla_{\boldsymbol{\varphi}} \left(\log p(\boldsymbol{x}_0) + \sum_{t=0}^{T-1} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t) + \sum_{t=0}^{T-1} \log p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, \boldsymbol{a}_t) \right) && \text{using the definition of the distribution over trajectories } \Pi_{\boldsymbol{\varphi}} \\
&= \nabla_{\boldsymbol{\varphi}} \log p(\boldsymbol{x}_0) + \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t) + \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\varphi}} \log p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, \boldsymbol{a}_t) \\
&= \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t). && (12.35) \quad \text{using that the first and third term are independent of } \boldsymbol{\varphi}
\end{aligned}
$$

When using a neural network for the parameterization of the policy π, we can use automatic differentiation to compute the gradients.

The expectation of the score gradient estimator (12.32) can be approximated using Monte Carlo sampling,

$$
\nabla_{\boldsymbol{\varphi}} J_T(\boldsymbol{\varphi}) \approx \nabla_{\boldsymbol{\varphi}} \widehat{J}_T(\boldsymbol{\varphi}) \doteq \frac{1}{m} \sum_{i=1}^{m} g_{0:T}^{(i)} \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t^{(i)} \mid \boldsymbol{x}_t^{(i)}). \quad (12.36)
$$

However, typically the variance of these estimates is very large. Using so-called *baselines*, we can reduce the variance dramatically (?). **Problem 12.3**

Lemma 12.6 (Score gradients with baselines)**.** *We have,*

$$
\mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}} \left[G_0 \nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau) \right] = \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}} \left[(G_0 - b) \nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau) \right]. \quad (12.37)
$$

Here, $b \in \mathbb{R}$ is called a baseline.

Proof. For the term to the right, we have due to linearity of expectation (1.20),

$$
\begin{aligned}
\mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}} \left[(G_0 - b) \nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau) \right] &= \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}} \left[G_0 \nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau) \right] \\
&\quad - \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}} \left[b \cdot \nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau) \right].
\end{aligned}
$$

Thus, it remains to show that the second term is zero,

$$
\begin{aligned}
\mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}} \left[b \cdot \nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau) \right] &= b \cdot \int \Pi_{\boldsymbol{\varphi}}(\tau) \nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau) \, d\tau && \text{using the definition of expectation (1.19)} \\
&= b \cdot \int \Pi_{\boldsymbol{\varphi}}(\tau) \frac{\nabla_{\boldsymbol{\varphi}} \Pi_{\boldsymbol{\varphi}}(\tau)}{\Pi_{\boldsymbol{\varphi}}(\tau)} \, d\tau && \text{substituting the score function (12.33), "undoing the score function trick"} \\
&= b \cdot \int \nabla_{\boldsymbol{\varphi}} \Pi_{\boldsymbol{\varphi}}(\tau) \, d\tau && \Pi_{\boldsymbol{\varphi}}(\tau) \text{ cancels} \\
&= b \cdot \nabla_{\boldsymbol{\varphi}} \int \Pi_{\boldsymbol{\varphi}}(\tau) \, d\tau \\
&= b \cdot \nabla_{\boldsymbol{\varphi}} 1 = 0. && \square \quad \text{integrating a PDF over its domain is 1 and the derivative of a constant is 0}
\end{aligned}
$$

One can even show, that we can subtract arbitrary baselines depending on *previous* states (?).

Problem 12.4

Example 12.7: Downstream returns

A commonly used state-dependent baseline is

$$b(\tau_{0:t-1}) \doteq \sum_{m=0}^{t-1} \gamma^m r_m. \tag{12.38}$$

This baseline subtracts the returns of all actions before time t. Intuitively, using this baseline, the score gradient only considers downstream returns. Recall from Equation (12.22) that we defined $G_{t:T}$ as the bounded discounted payoff from time t. It is also commonly called the (bounded) *downstream return* (or *reward to go*) beginning at time t.

For a fixed trajectory τ that is bounded at time T, we have

$$G_0 - b(\tau_{0:t-1}) = \gamma^t G_{t:T}, \tag{12.39}$$

yielding the gradient estimator,

$$\begin{aligned} \nabla_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi}) \approx \nabla_{\boldsymbol{\varphi}} J_T(\boldsymbol{\varphi}) &= \mathbb{E}_{\tau\sim\Pi_{\boldsymbol{\varphi}}}\left[G_0 \nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau)\right] && \text{using the score gradient estimator (12.32)} \\ &= \mathbb{E}_{\tau\sim\Pi_{\boldsymbol{\varphi}}}\left[\sum_{t=0}^{T-1} \gamma^t G_{t:T} \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t)\right]. && \text{using a state-dependent baseline (12.102)} \end{aligned} \tag{12.40}$$

Performing stochastic gradient descent with the score gradient estimator and downstream returns is known as the *REINFORCE algorithm* (Williams, 1992) which is shown in Algorithm 12.8.

Algorithm 12.8: REINFORCE algorithm

```
initialize policy weights φ
repeat
    generate an episode (i.e., rollout) to obtain trajectory τ
    for t = 0 to T − 1 do
        set g_{t:T} to the downstream return from time t
        φ ← φ + η γ^t g_{t:T} ∇_φ log π_φ(a_t | x_t)        // (12.41)
until converged
```

The variance of REINFORCE can be reduced further. A common tech-

nique is to subtract a term b_t from the downstream returns,

$$\nabla_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi}) = \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}} \left[\sum_{t=0}^{T} \gamma^t (G_{t:T} - b_t) \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t) \right]. \tag{12.42}$$

For example, we can subtract the t-independent mean reward to go,

$$b_t \doteq b = \frac{1}{T} \sum_{t'=0}^{T-1} G_{t':T}. \tag{12.43}$$

The main advantage of policy gradient methods such as REINFORCE is that they can be used in continuous action spaces. However, REINFORCE is not guaranteed to find an optimal policy. Even when operating in very small domains, REINFORCE can get stuck in local optima.

Typically, policy gradient methods are slow due to the large variance in the score gradient estimates. Because of this, they need to take small steps and require many rollouts of a Markov chain. Moreover, we cannot reuse data from previous rollouts, as policy gradient methods are fundamentally on-policy.[12]

[12] This is because the score gradient estimator is used to obtain gradients of the policy value function with respect to the *current* policy.

Next, we will combine value approximation techniques like Q-learning and policy gradient methods, leading to an often more practical family of methods called actor-critic methods.

12.4 On-policy Actor-Critics

Actor-Critic methods reduce the variance of policy gradient estimates by using ideas from value function approximation. They use function approximation both to approximate value functions *and* to approximate policies. The goal for these algorithms is to scale to reinforcement learning problems, where we both have large state spaces and large action spaces.

12.4.1 Advantage Function

A key concept of actor-critic methods is the advantage function.

Definition 12.9 (Advantage function)**.** Given a policy π, the *advantage function*,

$$\begin{aligned} a^\pi(\boldsymbol{x}, \boldsymbol{a}) &\doteq q^\pi(\boldsymbol{x}, \boldsymbol{a}) - v^\pi(\boldsymbol{x}) && (12.44) \\ &= q^\pi(\boldsymbol{x}, \boldsymbol{a}) - \mathbb{E}_{\boldsymbol{a}' \sim \pi(\boldsymbol{x})} \left[q^\pi(\boldsymbol{x}, \boldsymbol{a}') \right], && (12.45) \end{aligned}$$

using Equation (10.14)

measures the advantage of picking action $\boldsymbol{a} \in \mathcal{A}$ when in state $\boldsymbol{x} \in \mathcal{X}$ over simply following policy π.

It follows immediately from Equation (12.45) that for any policy π and state $\boldsymbol{x} \in \mathcal{X}$, there exists an action $\boldsymbol{a} \in \mathcal{A}$ such that $a^\pi(\boldsymbol{x}, \boldsymbol{a})$ is non-negative,

$$\max_{\boldsymbol{a} \in \mathcal{A}} a^\pi(\boldsymbol{x}, \boldsymbol{a}) \geq 0. \tag{12.46}$$

Moreover, it follows directly from Bellman's theorem (10.31) that

$$\pi \text{ is optimal } \iff \forall \boldsymbol{x} \in \mathcal{X}, \boldsymbol{a} \in \mathcal{A} : a^\pi(\boldsymbol{x}, \boldsymbol{a}) \leq 0. \tag{12.47}$$

In other words, quite intuitively, π is optimal if and only if there is no action that has an advantage in any state over the action that is played by π.

Finally, we can re-define the greedy policy π_q with respect to the state-action value function q as

$$\pi_q(\boldsymbol{x}) \doteq \arg\max_{\boldsymbol{a} \in \mathcal{A}} a(\boldsymbol{x}, \boldsymbol{a}) \tag{12.48}$$

since

$$\arg\max_{\boldsymbol{a} \in \mathcal{A}} a(\boldsymbol{x}, \boldsymbol{a}) = \arg\max_{\boldsymbol{a} \in \mathcal{A}} q(\boldsymbol{x}, \boldsymbol{a}) - v(\boldsymbol{x}) = \arg\max_{\boldsymbol{a} \in \mathcal{A}} q(\boldsymbol{x}, \boldsymbol{a}),$$

as $v(\boldsymbol{x})$ is independent of $\boldsymbol{a}$. This coincides with our initial definition of greedy policies in Equation (10.29). Intuitively, the advantage function is a shifted version of the state-action value function q that is relative to 0. Using this quantity rather than q often has numerical advantages.

12.4.2 *Policy Gradient Theorem*

Recall the score gradient estimator (12.40) that we had introduced in the previous section,

$$\nabla_{\boldsymbol{\varphi}} J_T(\boldsymbol{\varphi}) = \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}} \left[\sum_{t=0}^{T-1} \gamma^t G_{t:T} \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t) \right].$$

Previously, we have approximated the policy value function $J(\boldsymbol{\varphi})$ by the bounded policy value function $J_T(\boldsymbol{\varphi})$. We said that this approximation was "reasonable" due to the diminishing returns. Essentially, we have "cut off the tails" of the policy value function. Let us now reinterpret score gradients while taking into account the tails of $J(\boldsymbol{\varphi})$.

$$\begin{aligned} \nabla_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi}) &= \lim_{T \to \infty} \nabla_{\boldsymbol{\varphi}} J_T(\boldsymbol{\varphi}) \\ &= \sum_{t=0}^{\infty} \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}} \left[\gamma^t G_t \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t) \right]. \end{aligned} \tag{12.49}$$

substituting the score gradient estimator with downstream returns (12.40) and using linearity of expectation (1.20)

Observe that because the expectations only consider downstream returns, we can disregard all data from the trajectory prior to time t. Let us define

$$\tau_{t:\infty} \doteq ((\boldsymbol{x}_t, \boldsymbol{a}_t, r_t, \boldsymbol{x}_{t+1}), (\boldsymbol{x}_{t+1}, \boldsymbol{a}_{t+1}, r_{t+1}, \boldsymbol{x}_{t+2}), \dots), \tag{12.50}$$

as the trajectory from time step t. Then,

$$= \sum_{t=0}^{\infty} \mathbb{E}_{\tau_{t:\infty} \sim \Pi_{\boldsymbol{\varphi}}} \left[\gamma^t G_t \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t) \right].$$

We now condition on $\boldsymbol{x}_t$ and $\boldsymbol{a}_t$,

$$= \sum_{t=0}^{\infty} \mathbb{E}_{\boldsymbol{x}_t, \boldsymbol{a}_t} \left[\gamma^t \mathbb{E}_{r_t, \tau_{t+1:\infty}} [G_t \mid \boldsymbol{x}_t, \boldsymbol{a}_t] \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t) \right].$$

using that $\pi_{\boldsymbol{\varphi}}(a_t \mid x_t)$ is a constant given x_t and a_t

Observe that averaging over the trajectories $\mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}}[\cdot]$ that are sampled according to policy $\pi_{\boldsymbol{\varphi}}$ is equivalent to our shorthand notation $\mathbb{E}_{\pi_{\boldsymbol{\varphi}}}[\cdot]$ from Equation (10.6),

$$= \sum_{t=0}^{\infty} \mathbb{E}_{\boldsymbol{x}_t, \boldsymbol{a}_t} \left[\gamma^t \mathbb{E}_{\pi_{\boldsymbol{\varphi}}} [G_t \mid \boldsymbol{x}_t, \boldsymbol{a}_t] \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t) \right]$$

$$= \sum_{t=0}^{\infty} \mathbb{E}_{\boldsymbol{x}_t, \boldsymbol{a}_t} \left[\gamma^t q^{\pi_{\boldsymbol{\varphi}}}(\boldsymbol{x}_t, \boldsymbol{a}_t) \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t) \right]. \tag{12.51}$$

using the definition of the Q-function (10.8)

It turns out that $\mathbb{E}_{\boldsymbol{x}_t, \boldsymbol{a}_t}[q^{\pi_{\boldsymbol{\varphi}}}(\boldsymbol{x}_t, \boldsymbol{a}_t)]$ exhibits much less variance than our previous estimator $\mathbb{E}_{\boldsymbol{x}_t, \boldsymbol{a}_t} \mathbb{E}_{\pi_{\boldsymbol{\varphi}}}[G_t \mid \boldsymbol{x}_t, \boldsymbol{a}_t]$. Equation (12.51) is known as the *policy gradient theorem*.

Often, the policy gradient theorem is stated in a slightly rephrased form in terms of the *discounted state occupancy measure*,[13]

$$\rho^{\infty}_{\boldsymbol{\varphi}}(\boldsymbol{x}) \doteq (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t p_{\mathbf{X}_t}(\boldsymbol{x}). \tag{12.52}$$

[13] Depending on the reward setting, there exist various variations of the policy gradient theorem. We derived the variant for infinite-horizon discounted payoffs. "Reinforcement learning: An introduction" (Sutton and Barto, 2018) derive the variant for undiscounted average rewards.

The factor $(1 - \gamma)$ ensures that $\rho^{\infty}_{\boldsymbol{\varphi}}$ is a probability density as

$$\int \rho^{\infty}_{\boldsymbol{\varphi}}(\boldsymbol{x}) \, d\boldsymbol{x} = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t \int p_{\mathbf{X}_t}(\boldsymbol{x}) \, d\boldsymbol{x} = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t = 1.$$

Intuitively, $\rho^{\infty}_{\boldsymbol{\varphi}}(\boldsymbol{x})$ measures how often we visit state $\boldsymbol{x}$ when following policy $\pi_{\boldsymbol{\varphi}}$. It can be thought of as a "discounted frequency".

Theorem 12.10 (Policy gradient theorem in terms of $\rho^{\infty}_{\boldsymbol{\varphi}}$)**.** *Policy gradients can be represented in terms of the Q-function,*

$$\nabla_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi}) \propto \mathbb{E}_{\boldsymbol{x} \sim \rho^{\infty}_{\boldsymbol{\varphi}}} \mathbb{E}_{\boldsymbol{a} \sim \pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x})} \left[q^{\pi_{\boldsymbol{\varphi}}}(\boldsymbol{x}, \boldsymbol{a}) \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a} \mid \boldsymbol{x}) \right]. \tag{12.53}$$

Proof. The right hand side of Equation (12.51) can be expressed as

$$\sum_{t=0}^{\infty} \int p_{\mathbf{X}_t}(\boldsymbol{x}) \cdot \mathbb{E}_{\boldsymbol{a}_t \sim \pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x})} \left[\gamma^t q^{\pi_{\boldsymbol{\varphi}}}(\boldsymbol{x}, \boldsymbol{a}_t) \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}) \right] d\boldsymbol{x}$$

$$= \frac{1}{1-\gamma} \int \rho_{\boldsymbol{\varphi}}^{\infty}(\boldsymbol{x}) \cdot \mathbb{E}_{\boldsymbol{a} \sim \pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x})} \left[q^{\pi_{\boldsymbol{\varphi}}}(\boldsymbol{x}, \boldsymbol{a}) \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a} \mid \boldsymbol{x}) \right] d\boldsymbol{x}$$

where we swapped the order of sum and integral and reorganized terms. □

Matching our intuition, according to the policy gradient theorem, maximizing $J(\boldsymbol{\varphi})$ corresponds to increasing the probability of actions with a large value and decreasing the probability of actions with a small value, taking into account how often the resulting policy visits certain states.

Observe that we cannot use the policy gradient to calculate the gradient exactly, as we do not know $q^{\pi_{\boldsymbol{\varphi}}}$. Instead, we will use bootstrapping estimates $Q^{\pi_{\boldsymbol{\varphi}}}$ of $q^{\pi_{\boldsymbol{\varphi}}}$.

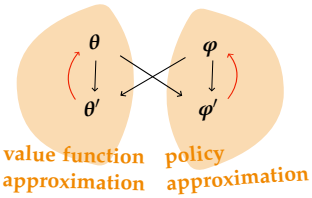


Figure 12.2: Illustration of one iteration of actor-critic methods. The dependencies between the actors and critics are shown as arrows. Methods differ in the exact order in which actor and critic are updated.

12.4.3 *A First Actor-Critic*

Actor-Critic methods consist of two components:

- a parameterized policy, $\pi(\boldsymbol{a} \mid \boldsymbol{x}; \boldsymbol{\varphi}) \doteq \pi_{\boldsymbol{\varphi}}$, which is called *actor*; and (12.54)
- a value function approximation, $q^{\pi_{\boldsymbol{\varphi}}}(\boldsymbol{x}, \boldsymbol{a}) \approx Q^{\pi_{\boldsymbol{\varphi}}}(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})$, which is called *critic*. In the following, we will abbreviate $Q^{\pi_{\boldsymbol{\varphi}}}$ by Q. (12.55)

In deep reinforcement learning, neural networks are used to parameterize both actor and critic. Therefore, in principle, the actor-critic framework allows scaling to both large state spaces and large action spaces. We begin by discussing on-policy actor-critics.

One approach in the online setting (i.e., non-episodic setting), is to simply use SARSA for learning the critic. To learn the actor, we use stochastic gradient descent with gradients obtained using single samples from

$$\nabla_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi}) \approx \nabla_{\boldsymbol{\varphi}} \widehat{J}(\boldsymbol{\varphi}) \doteq \sum_{t=0}^{\infty} \mathbb{E}_{(\boldsymbol{x}_t, \boldsymbol{a}_t) \sim \pi_{\boldsymbol{\varphi}}} \left[\gamma^t Q(\boldsymbol{x}_t, \boldsymbol{a}_t; \boldsymbol{\theta}) \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t) \right] \tag{12.56}$$

see Equation (12.51)

where Q is a bootstrapping estimate of $q^{\pi_{\boldsymbol{\varphi}}}$. This algorithm is known as *online actor-critic* or *Q actor-critic* and shown in Algorithm 12.11.

Comparing to the derivation for TD-learning from Equation (12.8), we observe that Equation (12.58) corresponds to the SARSA update rule.[14] Due to the use of SARSA for learning the critic, this algorithm is fundamentally on-policy.

[14] The gradient with respect to $\boldsymbol{\theta}_Q$ appears analogously to our derivation of approximate Q-learning (12.15).

Crucially, by neglecting the dependence of the bootstrapping estimate Q on the policy parameters $\boldsymbol{\varphi}$, we introduce bias in the gradient estimates. In other words, using the bootstrapping estimate Q means that

Algorithm 12.11: Online actor-critic

initialize parameters $\boldsymbol{\varphi}$ and $\boldsymbol{\theta}$
repeat
  use $\pi_{\boldsymbol{\varphi}}$ to obtain transition $(\boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}')$
  $\delta = r + \gamma Q(\boldsymbol{x}', \pi_{\boldsymbol{\varphi}}(\boldsymbol{x}'); \boldsymbol{\theta}) - Q(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})$
  // actor update
  $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} + \eta \gamma^t Q(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a} \mid \boldsymbol{x})$ // (12.57)
  // critic update
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \delta \nabla_{\boldsymbol{\theta}} Q(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})$ // (12.58)
until converged

the resulting gradient direction might not be a valid ascent direction. In particular, the actor is not guaranteed to improve. Still, it turns out that under strong so-called "compatibility conditions" that are rarely satisfied in practice, a valid ascent direction can be guaranteed.

12.4.4 *Improved Actor-Critics*

Reducing variance: To further reduce the variance of the gradient estimates, it turns out that a similar approach to the baselines we discussed in the previous section on policy gradient methods is useful. A common approach is to subtract the state value function from estimates of the Q-function,

$$\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} + \eta_t \gamma^t (Q(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) - V(\boldsymbol{x}; \boldsymbol{\theta})) \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a} \mid \boldsymbol{x}) \tag{12.59}$$

$$= \boldsymbol{\varphi} + \eta_t \gamma^t A(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) \nabla_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a} \mid \boldsymbol{x}) \tag{12.60}$$

using the definition of the advantage function (12.44)

where $A(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})$ is a bootstrapped estimate of the advantage function $a^{\pi_{\boldsymbol{\varphi}}}$. This algorithm is known as *advantage actor-critic* (A2C) (Mnih et al., 2016). Recall that the Q-function is an absolute quantity, whereas the advantage function is a relative quantity, where the sign is informative for the gradient direction. Intuitively, an absolute value is harder to estimate than the sign. Actor-Critic methods are therefore often implemented with respect to the advantage function rather than the Q-function.

Taking a step back, observe that policy gradient methods such as REINFORCE generally have *high variance* in their gradient estimates. However, due to using Monte Carlo estimates of G_t, the gradient estimates are *unbiased*. In contrast, using a bootstrapped Q-function to obtain gradient estimates yields estimates with a *smaller variance*, but those estimates are *biased*. We are faced with a *bias-variance tradeoff*. A natural approach is therefore to blend both gradient estimates to allow

for effectively trading bias and variance. This leads to algorithms such as *generalized advantage estimation* (GAE) (Schulman et al., 2016).

Exploration: Similarly to REINFORCE, actor-critic methods typically rely on randomization in the policy to encourage exploration, the idea being that if the policy is stochastic, then the agent will visit a diverse set of states. The inherent stochasticity of the policy is, however, often insufficient. A common problem is that the policy quickly "collapses" to a deterministic policy since the objective function is greedily exploitative. A common workaround is to use an ε-greedy policy (cf. Section 11.3.1) or to explicitly encourage the policy to exhibit uncertainty by adding an entropy term to the objective function (more on this in Section 12.6). However, note that for on-policy methods, changing the policy also changes the value function learned by the critic.

Improving sample efficiency: Actor-Critic methods typically suffer from low sample efficiency. When additionally using an on-policy method, actor-critics often need an extremely large number of interactions before learning a near-optimal policy, because they cannot reuse past data. Allowing to reuse past data is a major advantage of off-policy methods like Q-learning.

One well-known variant that slightly improves the sample efficiency is *trust-region policy optimization* (TRPO) (Schulman et al., 2015). TRPO uses multiple iterations, where in each iteration a fixed critic is used to optimize the policy.[15] During iteration k, we select

[15] Intuitively, each iteration performs a collection of gradient ascent steps.

$$\boldsymbol{\varphi}_{k+1} \leftarrow \arg\max_{\boldsymbol{\varphi}} \widehat{J}(\boldsymbol{\varphi}) \quad \text{subject to } \mathbb{E}_{\boldsymbol{x}\sim\rho^{\infty}_{\boldsymbol{\varphi}_k}} \mathrm{KL}\big(\pi_{\boldsymbol{\varphi}_k}(\cdot \mid \boldsymbol{x}) \| \pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x})\big) \leq \delta \tag{12.61}$$

for some fixed $\delta > 0$ and where

$$\widehat{J}(\boldsymbol{\varphi}) \doteq \mathbb{E}_{\boldsymbol{x}\sim\rho^{\infty}_{\boldsymbol{\varphi}_k}, \boldsymbol{a}\sim\pi_{\boldsymbol{\varphi}_k}(\cdot|\boldsymbol{x})}\big[w_k(\boldsymbol{\varphi}; \boldsymbol{x}, \boldsymbol{a}) A^{\pi_{\boldsymbol{\varphi}_k}}(\boldsymbol{x}, \boldsymbol{a})\big]. \tag{12.62}$$

Notably, $\widehat{J}$ is an expectation with respect to the *previous* policy $\pi_{\boldsymbol{\varphi}_k}$ and the previous critic $A^{\pi_{\boldsymbol{\varphi}_k}}$. TRPO uses *importance sampling* where the importance weights (called "likelihood ratios"),

$$w_k(\boldsymbol{\varphi}; \boldsymbol{x}, \boldsymbol{a}) \doteq \frac{\pi_{\boldsymbol{\varphi}}(\boldsymbol{a} \mid \boldsymbol{x})}{\pi_{\boldsymbol{\varphi}_k}(\boldsymbol{a} \mid \boldsymbol{x})},$$

are used to correct for taking the expectation over the previous policy. When $w_k \approx 1$ the policies $\pi_{\boldsymbol{\varphi}}$ and $\pi_{\boldsymbol{\varphi}_k}$ are similar, whereas when $w_k \ll 1$ or $w_k \gg 1$, the policies differ significantly. To be able to assume that the fixed critic is a good approximation within a certain "trust region" (i.e., one iteration), we impose the constraint

$$\mathbb{E}_{\boldsymbol{x}\sim\rho^{\infty}_{\boldsymbol{\varphi}_k}} \mathrm{KL}\big(\pi_{\boldsymbol{\varphi}_k}(\cdot \mid \boldsymbol{x}) \| \pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x})\big) \leq \delta$$

to optimize only in the "neighborhood" of the current policy. This constraint is also necessary for the importance weights not to blow up.

Remark 12.12: Estimating the KL-divergence

Instead of naive computation with $\mathbb{E}_{\boldsymbol{a}\sim\pi_{\boldsymbol{\varphi}_k}(\cdot|\boldsymbol{x})}[-\log w_k(\boldsymbol{\varphi};\boldsymbol{x},\boldsymbol{a})]$, the KL-divergence is commonly estimated by Monte Carlo samples of

$$\begin{aligned}&\mathrm{KL}\big(\pi_{\boldsymbol{\varphi}_k}(\cdot \mid \boldsymbol{x})\|\pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x})\big)\\&\quad= \mathbb{E}_{\boldsymbol{a}\sim\pi_{\boldsymbol{\varphi}_k}(\cdot|\boldsymbol{x})}[w_k(\boldsymbol{\varphi};\boldsymbol{x},\boldsymbol{a}) - 1 - \log w_k(\boldsymbol{\varphi};\boldsymbol{x},\boldsymbol{a})],\end{aligned}$$

which adds the "baseline" $w_k(\boldsymbol{\varphi};\boldsymbol{x},\boldsymbol{a}) - 1$ with mean 0. Observe that this estimator is unbiased, always non-negative since $\log(x) \leq x - 1$ for all x, while having a lower variance than the naive estimator.

Taking the expectation with respect to the previous policy $\pi_{\boldsymbol{\varphi}_k}$ means that we can reuse data from rollouts within the same iteration. That is, TRPO allows reusing past data as long as it can still be "trusted". This makes TRPO "somewhat" off-policy. Fundamentally, though, TRPO is still an on-policy method.

Proximal policy optimization (PPO) is a family of heuristic variants of TRPO which replace the constrained optimization problem of Equation (12.61) by the unconstrained optimization of a regularized objective (Schulman et al., 2017; Wang et al., 2020). PPO algorithms often work well in practice. One canonical PPO method uses the modified objective

$$\boldsymbol{\varphi}_{k+1} \leftarrow \arg\max_{\boldsymbol{\varphi}} \widehat{J}(\boldsymbol{\varphi}) - \lambda \mathbb{E}_{\boldsymbol{x}\sim\rho^{\infty}_{\boldsymbol{\varphi}_k}} \mathrm{KL}\big(\pi_{\boldsymbol{\varphi}_k}(\cdot \mid \boldsymbol{x})\|\pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x})\big) \tag{12.63}$$

with some $\lambda > 0$, which regularizes towards the trust region. Another common variant of PPO is based on controlling the importance weights directly rather than regularizing by the KL-divergence. PPO is used, for example, to train large-scale language models such as GPT (Stiennon et al., 2020; OpenAI, 2023) which we will discuss in more detail in Section 12.7. There we will also see that the objective from Equation (12.63) can be cast as performing probabilistic inference.

Improving computational efficiency: A practical problem with the above methods is that the estimation of the advantage function $A(\boldsymbol{x},\boldsymbol{a};\boldsymbol{\theta})$ requires training a separate critic, next to the policy (i.e., the actor) parameterized by $\boldsymbol{\varphi}$. This can be computationally expensive. In particular, when both models are large neural networks (think multiple billions of parameters each), training both models is computationally

prohibitive. Now, recall that we introduced critics in the first place to reduce the variance of the policy gradient estimates.[16] *Group relative policy optimization* (GRPO) replaces the critic in PPO with simple Monte Carlo estimates of the advantage function (Shao et al., 2024):

$$\widehat{J}(\boldsymbol{\varphi}) \doteq \mathbb{E}_{\{\tau^{(i)}\}_{i=1}^{m} \sim \Pi_{\boldsymbol{\varphi}_k}(\cdot|\boldsymbol{x})} \left[\frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T} w_k(\boldsymbol{\varphi}; \boldsymbol{a}^{(i)}) \hat{A}_{t,i}^{\pi_{\boldsymbol{\varphi}_k}} \right], \tag{12.64}$$

$$\text{where} \quad \hat{A}_{t,i}^{\pi_{\boldsymbol{\varphi}_k}} \doteq \frac{g_{t:T}^{(i)} - \text{mean}(\{\tau^{(i)}\})}{\text{std}(\{\tau^{(i)}\})}$$

estimates the advantage of action $\boldsymbol{a}^{(i)}$ at time t by comparing to the mean reward and normalizing by the standard deviation of rewards from all trajectories $\tau^{(i)}$.[17] GRPO combines Monte Carlo sampling and baselines for variance reduction with the trust-region optimization of PPO, leading to a method that is more sample efficient than naive REINFORCE while being computationally more efficient than PPO.

[16] That is, we moved from the REINFORCE policy update (12.41) to the actor-critic policy update (12.57).

[17] In Equation (12.36), we have already seen that using a Monte Carlo estimate of returns is a simple approach to reduce variance without needing to learn a critic.

12.5 *Off-policy Actor-Critics*

In many applications, sample efficiency is crucial. Either because requiring too many interactions is computationally prohibitive or because obtaining sufficiently many samples for learning a near-optimal policy is simply impossible. We therefore now want to look at a separate family of actor-critic methods, which are off-policy, and hence, allow for the reuse of past data. These algorithms use the reparameterization gradient estimates, which we encountered before in the context of variational inference,[18] instead of score gradient estimators.

[18] see Section 5.5.1

The on-policy methods that we discussed in the previous section can be understood as performing a variant of *policy iteration*, where we use an estimate of the state-action value function of the current policy and then try to improve that policy by acting greedily with respect to this estimate. They mostly vary in how improving the policy is traded with improving the estimate of its value. Fundamentally, these methods rely on policy evaluation.[19]

[19] Policy evaluation is at the core of policy iteration. See Algorithm 10.14 for the definition of policy iteration and Section 10.2 for a summary of policy evaluation in the context of Markov decision processes.

The techniques that we will introduce in this section are much more closely related to value iteration, essentially making use of Bellman's optimality principle to learn the optimal value function directly which characterizes the optimal policy.

To begin with, let us assume that the policy $\boldsymbol{\pi}$ is deterministic. We will later lift this restriction in Section 12.5.1. Recall that our initial motivation to consider policy gradient methods and then actor-critic

methods was the intractability of the DQN loss

$$\ell_{\text{DQN}}(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{2} \sum_{(\boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}') \in \mathcal{D}} \left(r + \gamma \max_{\boldsymbol{a}' \in \mathcal{A}} Q^\star(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) \right)^2$$

see Equation (12.16)

when the action space $\mathcal{A}$ is large. What if we simply replace the exact maximum over actions by a parameterized policy?

$$\ell_{\text{DQN}}(\boldsymbol{\theta}; \mathcal{D}) \approx \frac{1}{2} \sum_{(\boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}') \in \mathcal{D}} \left(r + \gamma Q^\star(\boldsymbol{x}', \boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x}'); \boldsymbol{\theta}^{\text{old}}) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) \right)^2. \tag{12.65}$$

We want to train our parameterized policy to learn the maximization over actions, that is, to approximate the greedy policy[20]

$$\boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x}) \approx \boldsymbol{\pi}^\star(\boldsymbol{x}) = \arg\max_{\boldsymbol{a} \in \mathcal{A}} Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}). \tag{12.66}$$

[20] Here, we already apply the improvement of DDQN to use the most-recent estimate of the Q-function for action selection (see Equation (12.17)).

The key idea is that if we use a "rich-enough" parameterization of policies, selecting the greedy policy with respect to $Q^\star$ is equivalent to

$$\boldsymbol{\varphi}^\star = \arg\max_{\boldsymbol{\varphi}} \mathbb{E}_{\boldsymbol{x} \sim \mu} \left[Q^\star(\boldsymbol{x}, \boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x}); \boldsymbol{\theta}) \right] \tag{12.67}$$

where $\mu(\boldsymbol{x}) > 0$ is an *exploration distribution* over states with full support.[21] We refer to this expectation by

$$\widehat{J}_\mu(\boldsymbol{\varphi}; \boldsymbol{\theta}) \doteq \mathbb{E}_{\boldsymbol{x} \sim \mu} \left[Q^\star(\boldsymbol{x}, \boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x}); \boldsymbol{\theta}) \right]. \tag{12.68}$$

[21] We require full support to ensure that all states are explored.

Commonly, the exploration distribution μ is taken to be the distribution that samples states uniformly from a replay buffer. Note that we can easily obtain unbiased gradient estimates of $\widehat{J}_\mu$ with respect to $\boldsymbol{\varphi}$:

$$\nabla_{\boldsymbol{\varphi}} \widehat{J}_\mu(\boldsymbol{\varphi}; \boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x} \sim \mu} \left[\nabla_{\boldsymbol{\varphi}} Q^\star(\boldsymbol{x}, \boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x}); \boldsymbol{\theta}) \right]. \tag{12.69}$$

see Appendix A.1.5

Analogously to on-policy actor-critics (see Equation (12.56)), we use a bootstrapping estimate of $Q^\star$. That is, we neglect the dependence of the critic $Q^\star$ on the actor $\boldsymbol{\pi}_{\boldsymbol{\varphi}}$, and in particular, the policy parameters $\boldsymbol{\varphi}$. We have seen that bootstrapping works with Q-learning, so there is reason to hope that it will work in this context too. This then allows us to use the chain rule to compute the gradient,

$$\nabla_{\boldsymbol{\varphi}} Q^\star(\boldsymbol{x}, \boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x}); \boldsymbol{\theta}) = \boldsymbol{D}_{\boldsymbol{\varphi}} \boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x}) \cdot \nabla_{\boldsymbol{a}} Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) \big|_{\boldsymbol{a} = \boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x})}. \tag{12.70}$$

This corresponds to evaluating the bootstrapping estimate of the Q-function at $\boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x})$ and obtaining a gradient estimate of the policy estimate (e.g., through automatic differentiation). Note that as $\boldsymbol{\pi}_{\boldsymbol{\varphi}}$ is vector-valued, $\boldsymbol{D}_{\boldsymbol{\varphi}} \boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x})$ is the Jacobian of $\boldsymbol{\pi}_{\boldsymbol{\varphi}}$ evaluated at $\boldsymbol{x}$.

Exploration: Now that we have estimates of the gradient of our optimization target $\widehat{J}_\mu$, it is natural to ask how we should select actions (based on $\pi_{\boldsymbol{\varphi}}$) to trade exploration and exploitation. As we have seen, policy gradient techniques rely on the randomness in the policy to explore, but here we consider deterministic policies. As our method is off-policy, a simple idea in continuous action spaces is to add Gaussian noise to the action selected by $\pi_{\boldsymbol{\varphi}}$ — also known as *Gaussian noise "dithering"*.[22] This corresponds to an algorithm called *deep deterministic policy gradients* (Lillicrap et al., 2016) shown in Algorithm 12.13. This algorithm is essentially equivalent to Q-learning with function approximation (e.g., DQN),[23] with the only exception that we replace the maximization over actions with the learned policy $\pi_{\boldsymbol{\varphi}}$.

[22] Intuitively, this adds "additional randomness" to the policy $\pi_{\boldsymbol{\varphi}}$.

[23] see Equation (12.16)

Algorithm 12.13: Deep deterministic policy gradients, DDPG

initialize $\boldsymbol{\varphi}, \boldsymbol{\theta}$, a (possibly non-empty) replay buffer $\mathcal{D} = \emptyset$
set $\boldsymbol{\varphi}^{\text{old}} = \boldsymbol{\varphi}$ and $\boldsymbol{\theta}^{\text{old}} = \boldsymbol{\theta}$
for $t = 0$ **to** ∞ **do**
  observe state $\boldsymbol{x}$, pick action $\boldsymbol{a} = \pi_{\boldsymbol{\varphi}}(\boldsymbol{x}) + \boldsymbol{\varepsilon}$ for $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \lambda \boldsymbol{I},)$
  execute $\boldsymbol{a}$, observe r and $\boldsymbol{x}'$
  add $(\boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}')$ to the replay buffer $\mathcal{D}$
  if collected "enough" data **then**
    // policy improvement step
    for some iterations **do**
      sample a mini-batch B of $\mathcal{D}$
      for each transition in B, compute the label $y = r + \gamma Q^\star(\boldsymbol{x}', \pi(\boldsymbol{x}'; \boldsymbol{\varphi}^{\text{old}}); \boldsymbol{\theta}^{\text{old}})$
      // critic update
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \boldsymbol{\nabla}_{\boldsymbol{\theta}} \frac{1}{B} \sum_{(\boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}', y) \in B} (y - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}))^2$
      // actor update
      $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} + \eta \boldsymbol{\nabla}_{\boldsymbol{\varphi}} \frac{1}{B} \sum_{(\boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}', y) \in B} Q^\star(\boldsymbol{x}, \pi(\boldsymbol{x}; \boldsymbol{\varphi}); \boldsymbol{\theta})$
      $\boldsymbol{\theta}^{\text{old}} \leftarrow (1 - \rho) \boldsymbol{\theta}^{\text{old}} + \rho \boldsymbol{\theta}$
      $\boldsymbol{\varphi}^{\text{old}} \leftarrow (1 - \rho) \boldsymbol{\varphi}^{\text{old}} + \rho \boldsymbol{\varphi}$

Twin delayed DDPG (TD3) is an extension of DDPG that uses two separate critic networks for predicting the maximum action and evaluating the policy (Fujimoto et al., 2018). This addresses the maximization bias akin to Double-DQN. TD3 also applies delayed updates to the actor network, which increases stability.

12.5.1 Randomized Policies

We have seen that randomized policies naturally encourage exploration. With deterministic actor-critic methods like DDPG, we had to inject Gaussian noise to enforce sufficient exploration. A natural question is therefore whether we can also handle randomized policies in this framework of off-policy actor-critics.

The key idea is to replace the squared loss of the critic,

$$\ell_{\text{DQN}}(\boldsymbol{\theta}; \mathcal{D}) \approx \frac{1}{2}\Big(r + \gamma Q^\star(\boldsymbol{x}', \boldsymbol{\pi}(\boldsymbol{x}'; \boldsymbol{\varphi}); \boldsymbol{\theta}^{\text{old}}) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})\Big)^2, \tag{refer to the squared loss of Q-learning (12.11)}$$

which only considers the fixed action $\boldsymbol{\pi}(\boldsymbol{x}'; \boldsymbol{\varphi}^{\text{old}})$ with an expected squared loss,

$$\ell_{\text{DQN}}(\boldsymbol{\theta}; \mathcal{D}) \approx \mathbb{E}_{\boldsymbol{a}' \sim \pi(\boldsymbol{x}'; \boldsymbol{\varphi})}\left[\frac{1}{2}\Big(r + \gamma Q^\star(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})\Big)^2\right], \tag{12.71}$$

which considers a distribution over actions.

It turns out that we can still compute gradients of this expectation.

$$\begin{aligned}
&\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{a}' \sim \pi(\boldsymbol{x}'; \boldsymbol{\varphi})}\left[\frac{1}{2}\Big(r + \gamma Q^\star(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})\Big)^2\right] \\
&= \mathbb{E}_{\boldsymbol{a}' \sim \pi(\boldsymbol{x}'; \boldsymbol{\varphi})}\left[\nabla_{\boldsymbol{\theta}} \frac{1}{2}\Big(r + \gamma Q^\star(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})\Big)^2\right].
\end{aligned} \tag{see Appendix A.1.5}$$

Similarly to our definition of the Bellman error (12.12), we define by

$$\delta_{\text{B}}(\boldsymbol{a}') \doteq r + \gamma Q^\star(\boldsymbol{x}', \boldsymbol{a}'; \boldsymbol{\theta}^{\text{old}}) - Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}), \tag{12.72}$$

the *Bellman error* for a fixed action $\boldsymbol{a}'$. Using the chain rule, we obtain

$$= \mathbb{E}_{\boldsymbol{a}' \sim \pi(\boldsymbol{x}'; \boldsymbol{\varphi})}\left[\delta_{\text{B}}(\boldsymbol{a}') \nabla_{\boldsymbol{\theta}} Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})\right]. \tag{12.73}$$

Note that this is identical to the gradient in DQN (12.14), except that now we have an expectation over actions. As we have done many times already, we can use automatic differentiation to obtain gradient estimates of $\nabla_{\boldsymbol{\theta}} Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})$. This provides us with a method of obtaining unbiased gradient estimates for the critic.

We also need to reconsider the actor update. When using a randomized policy, the objective function changes to

$$\widehat{J}_\mu(\boldsymbol{\varphi}; \boldsymbol{\theta}) \doteq \mathbb{E}_{\boldsymbol{x} \sim \mu} \mathbb{E}_{\boldsymbol{a} \sim \pi(\boldsymbol{x}; \boldsymbol{\varphi})}[Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})].$$

of which we can obtain gradients via

$$\nabla_{\boldsymbol{\varphi}} \widehat{J}_\mu(\boldsymbol{\varphi}; \boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x} \sim \mu} \nabla_{\boldsymbol{\varphi}} \mathbb{E}_{\boldsymbol{a} \sim \pi(\boldsymbol{x}; \boldsymbol{\varphi})}[Q^\star(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})]. \tag{12.74}$$ (see Appendix A.1.5)

Note that the inner expectation is with respect to a measure that depends on the parameters $\boldsymbol{\varphi}$, which we are trying to optimize. We therefore cannot move the gradient operator inside the expectation. This is a problem that we have already encountered several times. In the previous section on policy gradients, we used the score gradient estimator.[24] Earlier, in Chapter 5 on variational inference we have already seen reparameterization gradients.[25] Here, if our policy is reparameterizable, we can use the *reparameterization trick* from Theorem 5.19!

[24] see Equation (12.32)

[25] see (5.63)

Example 12.14: Reparameterization gradients for Gaussians

Suppose we use a Gaussian parameterization of policies,

$$\pi(\boldsymbol{x};\boldsymbol{\varphi}) \doteq \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x};\boldsymbol{\varphi}), \boldsymbol{\Sigma}(\boldsymbol{x};\boldsymbol{\varphi})).$$

Then, using conditional linear Gaussians, our action $\boldsymbol{a}$ is given by

$$\boldsymbol{a} = \boldsymbol{g}(\boldsymbol{\varepsilon};\boldsymbol{x},\boldsymbol{\varphi}) \doteq \boldsymbol{\Sigma}^{1/2}(\boldsymbol{x};\boldsymbol{\varphi})\boldsymbol{\varepsilon} + \boldsymbol{\mu}(\boldsymbol{x};\boldsymbol{\varphi}), \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{12.75}$$

where $\boldsymbol{\Sigma}^{1/2}(\boldsymbol{x};\boldsymbol{\varphi})$ is the square root of $\boldsymbol{\Sigma}(\boldsymbol{x};\boldsymbol{\varphi})$. This coincides with our earlier application of the reparameterization trick to Gaussians in Example 5.20.

As we have seen, not only Gaussians are reparameterizable. In general, we called a distribution (in this context, a policy) reparameterizable iff $\boldsymbol{a} \sim \pi(\boldsymbol{x};\boldsymbol{\varphi})$ is such that $\boldsymbol{a} = \boldsymbol{g}(\boldsymbol{\varepsilon};\boldsymbol{x},\boldsymbol{\varphi})$, where $\boldsymbol{\varepsilon} \sim \phi$ is an independent random variable.

Then, we have,

$$\begin{aligned}
&\nabla_{\boldsymbol{\varphi}} \mathbb{E}_{\boldsymbol{a}\sim\pi(\boldsymbol{x};\boldsymbol{\varphi})}[Q^\star(\boldsymbol{x},\boldsymbol{a};\boldsymbol{\theta})] \\
&= \mathbb{E}_{\boldsymbol{\varepsilon}\sim\phi}\left[\nabla_{\boldsymbol{\varphi}} Q^\star(\boldsymbol{x},\boldsymbol{g}(\boldsymbol{\varepsilon};\boldsymbol{x},\boldsymbol{\varphi});\boldsymbol{\theta})\right] && (12.76) && \text{using the reparameterization trick (5.63)} \\
&= \mathbb{E}_{\boldsymbol{\varepsilon}\sim\phi}\left[\nabla_{\boldsymbol{a}} Q^\star(\boldsymbol{x},\boldsymbol{a};\boldsymbol{\theta})\big|_{\boldsymbol{a}=\boldsymbol{g}(\boldsymbol{\varepsilon};\boldsymbol{x},\boldsymbol{\varphi})} \cdot \boldsymbol{D}_{\boldsymbol{\varphi}}\boldsymbol{g}(\boldsymbol{\varepsilon};\boldsymbol{x},\boldsymbol{\varphi})\right]. && (12.77) && \text{using the chain rule analogously to Equation (12.70)}
\end{aligned}$$

In this way, we can obtain unbiased gradient estimates for reparameterizable policies. This general technique does not only apply to continuous action spaces. For discrete action spaces, there is the analogous so-called *Gumbel-max trick*, which we will not discuss in greater detail here.

The algorithm that uses Equation (12.73) to obtain gradients for the critic and reparameterization gradients for the actor is called *stochastic value gradients* (SVG) (Heess et al., 2015).

12.6 Maximum Entropy Reinforcement Learning

In practice, algorithms like SVG often do not explore enough. A key issue with relying on randomized policies for exploration is that they might collapse to deterministic policies. That is, the algorithm might quickly reach a local optimum, where all mass is placed on a single action.

A simple trick that encourages a little bit of extra exploration is to regularize the randomized policies "away" from putting all mass on a single action. In other words, we want to encourage the policies to exhibit some uncertainty. A natural measure of uncertainty is entropy, which we have already seen several times.[26] This approach is known as *entropy regularization* or *maximum entropy reinforcement learning* (MERL). Canonically, entropy regularization is applied to finite-horizon rewards (cf. Remark 10.5), yielding the optimization problem of maximizing

[26] see Section 5.4

$$J_\lambda(\boldsymbol{\varphi}) \doteq J(\boldsymbol{\varphi}) + \lambda \mathrm{H}\big[\Pi_{\boldsymbol{\varphi}}\big] \tag{12.78}$$

$$= \sum_{t=1}^{T} \mathbb{E}_{(\boldsymbol{x}_t, \boldsymbol{a}_t) \sim \Pi_{\boldsymbol{\varphi}}} \big[r(\boldsymbol{x}_t, \boldsymbol{a}_t) + \lambda \mathrm{H}\big[\pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x}_t)\big] \big], \tag{12.79}$$

where we have a preference for entropy in the actor distribution to encourage exploration which is regulated by the temperature parameter λ. As $\lambda \to 0$, we recover the "standard" reinforcement learning objective (here for finite-horizon rewards):

$$J(\boldsymbol{\varphi}) = \sum_{t=1}^{T} \mathbb{E}_{(\boldsymbol{x}_t, \boldsymbol{a}_t) \sim \Pi_{\boldsymbol{\varphi}}} [r(\boldsymbol{x}_t, \boldsymbol{a}_t)]. \tag{12.80}$$

Here, for notational convenience, we begin the sum with $t = 1$ rather than $t = 0$.

12.6.1 Entropy Regularization as Probabilistic Inference

The entropy-regularized objective from Equation (12.79) leads us to a remarkable interpretation of reinforcement learning and, more generally, decision-making under uncertainty as solving an inference problem akin to variational inference. The framing of "control as inference" will lead us to contemporary algorithms for reinforcement learning as well as paint a path for decision-making under uncertainty beyond stationary MDPs.

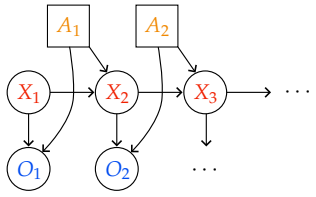


Figure 12.3: Directed graphical model of the underlying hidden Markov model with hidden states X_t, optimality variables O_t, and actions A_t.

Let us denote by $\Pi_\star$ the distribution over trajectories τ under the optimal policy $\pi^\star$. By framing the problem of optimal control as an inference problem in a hidden Markov model with hidden "optimality variables" $O_t \in \{0, 1\}$ indicating whether the played action $\boldsymbol{a}_t$ was

optimal we can derive $\Pi_\star$ analytically. That is to say, when $O_t = 1$ and $O_{t+1:T} \equiv 1$ the policy from time t onwards was optimal. To simplify the notation, we will denote the event $O_t = 1$ by $\mathcal{O}_t$.

We consider the HMM defined by the Gibbs distribution

$$p(\mathcal{O}_t \mid \boldsymbol{x}_t, \boldsymbol{a}_t) \propto \exp\left(\frac{1}{\lambda} r(\boldsymbol{x}_t, \boldsymbol{a}_t)\right), \quad \text{with } \lambda > 0 \tag{12.81}$$

which is a natural choice as we have seen in Problem 6.7 that the Gibbs distribution maximizes entropy subject to $\mathbb{E}[O_t \cdot r(\boldsymbol{x}_t, \boldsymbol{a}_t) \mid \boldsymbol{x}_t, \boldsymbol{a}_t] < \infty$.

The distribution over trajectories conditioned on optimality of actions (i.e., conditioned on $\mathcal{O}_{1:T}$) is given by

$$\Pi_\star(\tau) \doteq p(\tau \mid \mathcal{O}_{1:T}) = p(\boldsymbol{x}_1) \prod_{t=1}^{T} p(\boldsymbol{a}_t \mid \boldsymbol{x}_t, \mathcal{O}_t) p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, \boldsymbol{a}_t). \tag{12.82}$$

Using Equation (12.30). We assume here that the dynamics and initial state distribution are "fixed", that is, we assume $p(\boldsymbol{x}_1 \mid \mathcal{O}_{1:T}) = p(\boldsymbol{x}_1)$ and $p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, \boldsymbol{a}_t, \mathcal{O}_{1:T}) = p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, \boldsymbol{a}_t)$.

It remains to determine $p(\boldsymbol{a}_t \mid \boldsymbol{x}_t, \mathcal{O}_t)$ which corresponds to the optimal policy $\pi^\star(\boldsymbol{a}_t \mid \boldsymbol{x}_t)$. It is generally useful to think of the situation where the prior policy $p(\boldsymbol{a}_t \mid \boldsymbol{x}_t)$ is uniform on $\mathcal{A}$,[27] in which case by Bayes' rule (1.45), $p(\boldsymbol{a}_t \mid \boldsymbol{x}_t, \mathcal{O}_t) \propto p(\mathcal{O}_t \mid \boldsymbol{x}_t, \boldsymbol{a}_t)$, so

$$\Pi_\star(\tau) \propto \left[p(\boldsymbol{x}_1) \prod_{t=1}^{T} p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, \boldsymbol{a}_t)\right] \exp\left(\frac{1}{\lambda} \sum_{t=1}^{T} r(\boldsymbol{x}_t, \boldsymbol{a}_t)\right). \tag{12.83}$$

[27] This is not a restriction as any informative prior can be pushed into Equation (12.81).

Recall that our fundamental goal is to approximate $\Pi_\star$ with a distribution over trajectories $\Pi_{\boldsymbol{\varphi}}$ under the parameterized policy $\pi_{\boldsymbol{\varphi}}$. It is therefore a natural idea to minimize $\mathrm{KL}(\Pi_{\boldsymbol{\varphi}} \| \Pi_\star)$:[28]

[28] Observe that we cannot easily minimize forward-KL as we cannot sample from $\Pi_\star$. In the context of RL, it can be argued that the mode-seeking behavior of reverse-KL is preferable over the moment-matching behavior of forward-KL (Levine, 2018).

$$\begin{aligned}
&\arg\min_{\boldsymbol{\varphi}} \mathrm{KL}(\Pi_{\boldsymbol{\varphi}} \| \Pi_\star) \\
&= \arg\min_{\boldsymbol{\varphi}} \mathrm{H}[\Pi_{\boldsymbol{\varphi}} \| \Pi_\star] - \mathrm{H}[\Pi_{\boldsymbol{\varphi}}] && \text{using the definition of KL-divergence (5.34)} \\
&= \arg\max_{\boldsymbol{\varphi}} \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}}[\log \Pi_\star(\tau) - \log \Pi_{\boldsymbol{\varphi}}(\tau)] && \text{using the definition of cross-entropy (5.32) and entropy (5.27)} \\
&= \arg\max_{\boldsymbol{\varphi}} \mathbb{E}_{\tau \sim \Pi_{\boldsymbol{\varphi}}}\left[\sum_{t=1}^{T} r(\boldsymbol{x}_t, \boldsymbol{a}_t) - \lambda \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t)\right] && \text{using Equations (12.30) and (12.83) and simplifying} \\
&= \arg\max_{\boldsymbol{\varphi}} \sum_{t=1}^{T} \mathbb{E}_{(\boldsymbol{x}_t, \boldsymbol{a}_t) \sim \Pi_{\boldsymbol{\varphi}}}[r(\boldsymbol{x}_t, \boldsymbol{a}_t) + \lambda \mathrm{H}[\pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x}_t)]]. && \text{using the definition of entropy (5.27) and linearity of expectation (1.20)}
\end{aligned} \tag{12.84}$$

That is, entropy regularization is equivalent to minimizing the KL-divergence from $\Pi_\star$ to $\Pi_{\boldsymbol{\varphi}}$. This highlights a very natural tradeoff between exploration and exploitation, wherein $\mathrm{H}[\Pi_{\boldsymbol{\varphi}} \| \Pi_\star]$ encourages exploitation and $\mathrm{H}[\Pi_{\boldsymbol{\varphi}}]$ encourages exploration.

It can be shown that a "softmax" version of the Bellman optimality equation (10.34) can be obtained for Equation (12.84) (?):

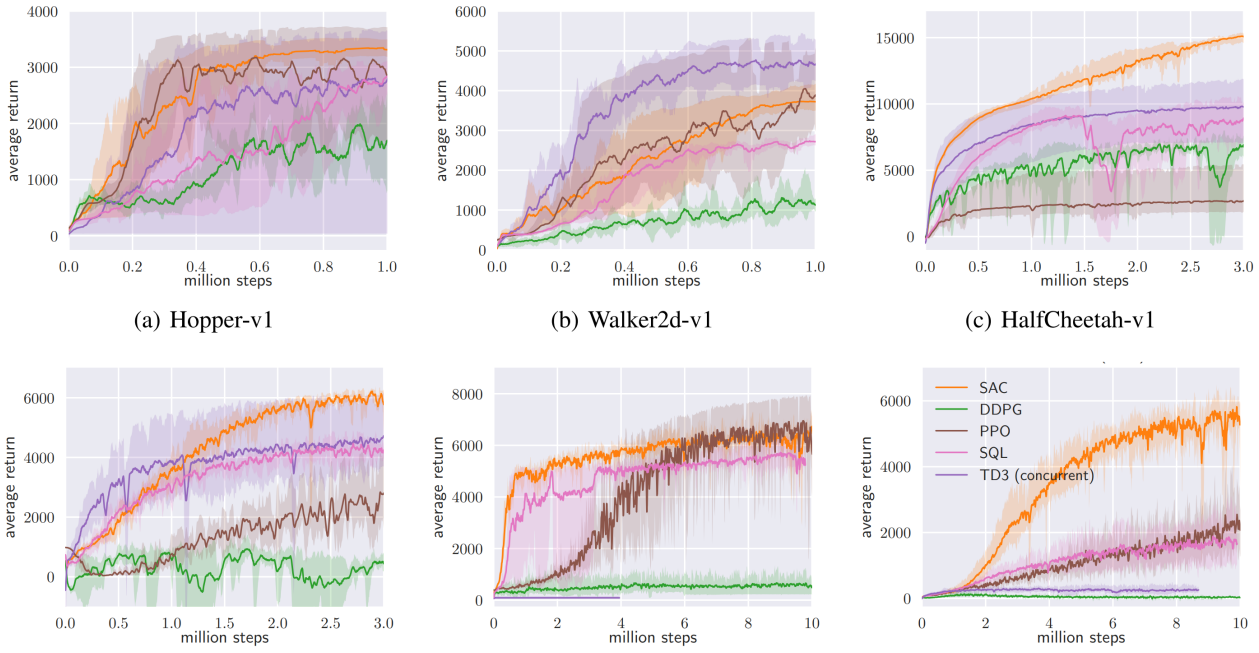
Problem 12.7

$$q^\star(\boldsymbol{x}, \boldsymbol{a}) = \frac{1}{\lambda} r(\boldsymbol{x}, \boldsymbol{a}) + \mathbb{E}_{\boldsymbol{x}' \sim \boldsymbol{x}, \boldsymbol{a}}\left[\log \int_{\mathcal{A}} \exp(q^\star(\boldsymbol{x}', \boldsymbol{a}'))\, d\boldsymbol{a}'\right] \tag{12.85}$$

with the convention that $q^\star(\boldsymbol{x}_T, \boldsymbol{a}) = 0$ for all $\boldsymbol{a}$.[29] Here, $q^\star$ is called a *soft value function*. As we will see in Problem 12.7, the optimal policy has the form $\pi^\star(\boldsymbol{a} \mid \boldsymbol{x}) \propto \exp(q^\star(\boldsymbol{x}, \boldsymbol{a}))$, that is, it simply corresponds to performing softmax exploration (11.9) with the soft value function. The second term of Equation (12.85) quantifies downstream rewards. In comparison to the "standard" Bellman optimality equation (10.34), the soft value function is less greedy which tends to encourage robustness.

[29] Note that Equation (10.34) was derived in the infinite horizon setting with discounted rewards, whereas here we study the finite horizon setting. Equation (12.85) is the natural extension of the standard Bellman optimality equation in the finite horizon setting where the downstream rewards are measured by a softmax rather than the greedy policy.

Analogously to Q-learning, the soft value function $q^\star$ can be approximated via a bootstrapped "critic" $Q^\star$ which is called *soft Q-learning* (Levine, 2018). Note that computing the optimal policy requires computing an integral over the action space, which is typically intractable for continuous action spaces. As discussed in Sections 12.3 to 12.5 and analogously to actor-critic methods such as DDPG and SVG, we can learn a parameterized policy (i.e., an "actor") $\pi_{\boldsymbol{\varphi}}$ to approximate the optimal policy $\pi^\star$. The resulting algorithm, *soft actor critic* (SAC) (Haarnoja et al., 2018a,b), is widely used. Due to its off-policy nature, it is also relatively sample efficient.



(a) Hopper-v1 (b) Walker2d-v1 (c) HalfCheetah-v1

(d) Ant-v1 (e) Humanoid-v1 (f) Humanoid (rllab)

Figure 12.4: Comparison of training curves of a selection of on-policy and off-policy policy gradient methods. Reproduced with permission from "Soft actor-critic algorithms and applications" (Haarnoja et al., 2018b).

We can also express this optimization in terms of an evidence lower

bound. The evidence lower bound for the observations $\mathcal{O}_{1:T}$ is

$$L(\Pi_{\boldsymbol{\varphi}}, \Pi_\star; \mathcal{O}_{1:T}) = \mathbb{E}_{\tau\sim\Pi_{\boldsymbol{\varphi}}}\big[\log p(\mathcal{O}_{1:T} \mid \tau) + \log \Pi(\tau) - \log \Pi_{\boldsymbol{\varphi}}(\tau)\big] \tag{12.86}$$

using the definition of the ELBO (5.55b)

where Π denotes the distribution over trajectories (12.30) under the prior policy $p(\boldsymbol{a}_t \mid \boldsymbol{x}_t)$. This is commonly written as the variational free energy

$$-L(\Pi_{\boldsymbol{\varphi}}, \Pi_\star; \mathcal{O}_{1:T}) = \mathbb{E}_{\tau\sim\Pi_{\boldsymbol{\varphi}}}[\mathrm{S}[p(\mathcal{O}_{1:T} \mid \tau)]] + \mathrm{KL}\big(\Pi_{\boldsymbol{\varphi}} \| \Pi\big) \tag{12.87}$$

$$= \underbrace{\mathrm{S}[p(\mathcal{O}_{1:T})]}_{\text{“extrinsic” value}} + \underbrace{\mathrm{KL}\big(\Pi_{\boldsymbol{\varphi}} \| \Pi_\star\big)}_{\text{“epistemic” value}}. \tag{12.88}$$

using that $\Pi_\star(\tau) = p(\tau \mid \mathcal{O}_{1:T})$

which we already encountered in Section 5.5.2 in the context of variational inference. Here, the “extrinsic” value is independent of the variational distribution $\Pi_{\boldsymbol{\varphi}}$ and can be thought of as a fixed “problem cost”, whereas the “epistemic” value can be interpreted as the approximation error or “solution cost”. To summarize, we have seen that

$$\arg\max_{\boldsymbol{\varphi}} J_\lambda(\boldsymbol{\varphi}) = \arg\min_{\boldsymbol{\varphi}} \mathrm{KL}\big(\Pi_{\boldsymbol{\varphi}} \| \Pi_\star\big) = \arg\max_{\boldsymbol{\varphi}} L(\Pi_{\boldsymbol{\varphi}}, \Pi_\star; \mathcal{O}_{1:T}).$$

Recall that the free energy $-L(\Pi_{\boldsymbol{\varphi}}, \Pi_\star; \mathcal{O}_{1:T})$ is a variational upper bound to the surprise about observations $\mathrm{S}[\Pi_\star(\mathcal{O}_{1:T})]$ when following an optimal policy.[30] For example, undesirable states incur a low reward while desirable states yield a high reward, and thus, if we expect optimality of actions (i.e., $\mathcal{O}_{1:T}$) paths leading to such states have high and low surprise, respectively.[31]

[30] see Section 5.5.2

[31] This is because, a path leading to a low reward state will include sub-optimal actions.

An agent which acts to minimize free energy with $\mathcal{O}_{1:T}$ can be thought of as hallucinating to perform optimally, and acting to minimize the surprise about having played suboptimal actions. Think, for example, about the robotics task of moving an arm to a new position. Intuitively, minimizing free energy solves this task by “hallucinating that the arm is at the goal position”, and then minimizing the surprise with respect to this perturbed world model. In this way, MERL can be understood as identifying paths of least surprise akin to the *free energy principle*.

Remark 12.15: Towards active inference

Maximum entropy reinforcement learning makes one fundamental assumption, namely, that the “biased” distribution about observations specifying the underlying HMM is

$$p(\mathcal{O}_t \mid \boldsymbol{x}_t, \boldsymbol{a}_t) \propto \exp\left(\frac{1}{\lambda} r(\boldsymbol{x}_t, \boldsymbol{a}_t)\right).$$

This assumption is key to the framing of optimal control (in an

unknown MDP) with a certain reward function as an inference problem. One could conceive other HMMs. For example, Fellows et al. (2019) propose an HMM defined in terms of the current value function of the agent:

$$p(\mathcal{O}_t \mid \boldsymbol{x}_t, \boldsymbol{a}_t) \propto \exp\left(\frac{1}{\lambda} Q(\boldsymbol{x}_t, \boldsymbol{a}_t; \boldsymbol{\theta})\right).$$

Crucially, the choice of $p(\mathcal{O}_t \mid \boldsymbol{x}_t, \boldsymbol{a}_t)$ is the only place where the reward enters the inference problem, and one can conceive of settings where a "stationary" (i.e., time-independent) reward is not assumed to exist. The general approach to decision-making as probabilistic inference presented in this section (but for possibly reward-independent and non-stationary HMMs) is known as *active inference* (Friston et al., 2015; Millidge et al., 2020, 2021; Parr et al., 2022).

12.7 *Learning from Preferences*

So far, we have been assuming that the agent is presented with a reward signal after every played action. This is a natural assumption in domains such as games and robotics — even though it often requires substantial "reward engineering" to break down complex tasks with sparse rewards to more manageable tasks (cf. Section 13.3). In many other domains such as an agent learning to drive a car or a chatbot, it is unclear how one can even quantify the reward associated with an action or a sequence of actions. For example, in the context of autonomous driving it is typically desired that agents behave "human-like" even though a different driving behavior may also reach the destination safely.

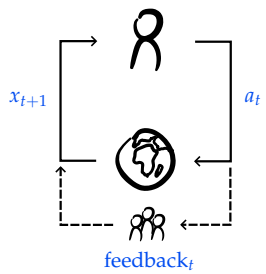


Figure 12.5: We generalize the perspective of reinforcement learning from Figure 11.1 by allowing the feedback to come from either the environment or an evaluation by other agents (e.g., humans), and by allowing the feedback to come in other forms than a numerical reward.

The task of "aligning" the behavior of an agent to human expectations is difficult in complex domains such as the physical world and language, yet crucial for their practical use. To this end, one can conceive of alternative ways for presenting "feedback" to the agent:

- The classical feedback in reinforcement learning is a numerical score. Consider, for example, a recommender system for movies. The feedback is obtained after a movie was recommended to a user by a user-rating on a given scale (often 1 to 10). This rating is informative as it corresponds to an *absolute value assessment*, allowing to place the recommendation in a complete ranking of all previous recommendations. However, numerical feedback of this type can be error-prone as it is scale-dependent (different users may ascribe different value to a recommendation rated a 7).

- An alternative feedback mechanism is comparison-based. The user is presented with k alternative actions and selects their preferred action (or alternatively returns a ranking of actions). This feedback is typically easy to obtain as humans are fast in making "this-or-that" decisions. However, in contrast to numerical rewards, the feedback provides information only on the user's *relative preferences*. That is, such preference feedback encodes fewer bits of information than score feedback, and it therefore often takes longer to learn complex behavior from this feedback.

Remark 12.16: Context-dependent feedback

We neglect here that feedback is often context-dependent (Lindner and El-Assady, 2022; Casper et al., 2023). For example, if someone is asked whether they prefer "ice cream" over "pizza" the answer may depend on whether they are hungry and the weather.

12.7.1 *Language Models as Agents*

In the following, we discuss approaches to learning from preference feedback in the context of autoregressive[32] large language models and chatbots. A chatbot is an agent (often based on a transformer architecture, Vaswani et al. (2017)) parameterized by $\boldsymbol{\varphi}$ that given a *prompt* $\boldsymbol{x}$ returns a (stochastic) *response* $\boldsymbol{y}$. The autoregressive generation of the response can be understood in terms of a policy $\pi_{\boldsymbol{\varphi}}(y_{t+1} \mid \boldsymbol{x}, y_{1:t})$ which generates the next token given the prompt and all previous tokens.[33] We denote the policy over complete responses (i.e., the chatbot) by

$$\Pi_{\boldsymbol{\varphi}}(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=0}^{T-1} \pi_{\boldsymbol{\varphi}}(y_{t+1} \mid \boldsymbol{x}, y_{1:t}). \tag{12.89}$$

In RL jargon, the agents action corresponds to the choice of next token y_{t+1} and the deterministic dynamics add this token to the current (incomplete) response $y_{1:t}$. A full trajectory $\boldsymbol{y}$ consists sequentially of all tokens $y_{1:T}$ comprising a response, and the prompt $\boldsymbol{x}$ can be interpreted as a *context* to this trajectory. Observe that Equation (12.89) is derived from the general representation of the distribution over trajectories (12.30), noting that prior and dynamics are deterministic.

The standard pipeline for applying pre-trained[34] large language models such as GPT (OpenAI, 2023) to downstream tasks consists of two main steps which are illustrated in Figure 12.6 (Stiennon et al., 2020): (1) supervised fine-tuning and (2) post-training using preference feedback.

The first step is to fine-tune the language model with supervised learning on high-quality data for the downstream task of interest. For ex-

[32] An autoregressive model predicts/-generates the next "token" as a function of previous tokens.

[33] In large language models, a "token" is usually taken to be a letter, word, or something in-between. A special token is used to terminate the response.

[34] The pre-trained language model is usually obtained by self-supervised training on a large corpus of text. *Self-supervised learning* generates labeled training data from an unlabeled data source by selectively "masking-out" parts of the data. When training an autoregressive language model, labeled training data can be obtained by repeatedly "masking-out" the next word in a sentence, and training the language model to predict this word. Such large models that can be fine-tuned and "post-trained" to various downstream tasks are also called *foundation models*.

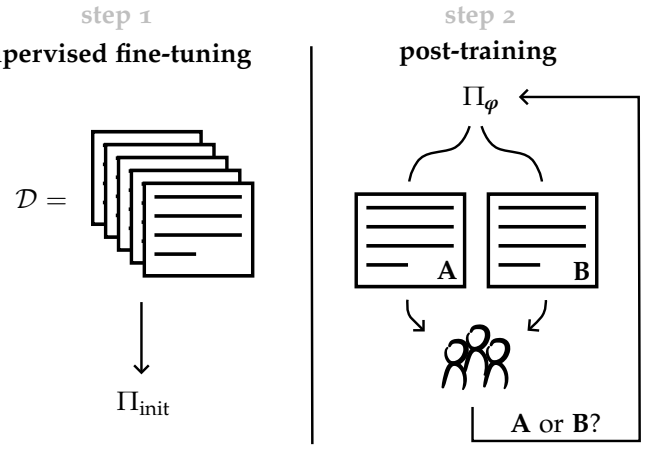


Figure 12.6: Illustration of the learning pipeline of a large language model.

ample, when the goal is to build a chatbot, this data may consist of desirable responses to some exemplary prompts. We will denote the parameters of the fine-tuned language model by $\boldsymbol{\varphi}^{\text{init}}$, and its associated policy by Π_{init}.

The second step is then to "post-train" the language model Π_{init} from the first step using human feedback. Here, it is important that Π_{init} is already capable of producing sensible output (i.e., with correct spelling and grammar). Learning this from scratch using only preference feedback would take far too long. Instead, the post-training step is used to align the agent to the task and user preferences.

In each iteration of post-training, the model is prompted with prompts $\boldsymbol{x}$ to produce pairs of answers $(\boldsymbol{y}_A, \boldsymbol{y}_B) \sim \Pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x})$. These answers are presented to human labelers who express their preference for one of the answers, denoted $\boldsymbol{y}_A \succ \boldsymbol{y}_B \mid \boldsymbol{x}$. A popular choice for modeling preferences is the *Bradley-Terry model* which stipulates that the human preference distribution is given by

$$p(\boldsymbol{y}_A \succ \boldsymbol{y}_B \mid \boldsymbol{x}, r) = \frac{\exp(r(\boldsymbol{y}_A \mid \boldsymbol{x}))}{\exp(r(\boldsymbol{y}_A \mid \boldsymbol{x})) + \exp(r(\boldsymbol{y}_B \mid \boldsymbol{x}))} \tag{12.90}$$

for some unknown latent reward model $r(\boldsymbol{y} \mid \boldsymbol{x})$ (Bradley and Terry, 1952). This can be written in terms of the logistic function σ (5.9):

$$= \sigma(r(\boldsymbol{y}_A \mid \boldsymbol{x}) - r(\boldsymbol{y}_B \mid \boldsymbol{x})). \tag{12.91}$$

as seen in Problem 7.1 this is the Gibb's distribution with energy $-r(\boldsymbol{y} \mid \boldsymbol{x})$ in a binary classification problem

Remark 12.17: Outcome rewards

Note that the Bradley-Terry model attributes reward only to "complete" responses. We call such a reward an *outcome reward*.[35] While the following discussion is on outcome rewards (which is most common in the context of language models), everything translates to individual per-step rewards.

[35] Framing this in terms of an individual "per-step" rewards which we have seen so far, this corresponds to a (sparse) reward which is zero until the final action.

The aggregated human feedback $\mathcal{D} = \{\boldsymbol{y}_A^{(i)} \succ \boldsymbol{y}_B^{(i)} \mid \boldsymbol{x}^{(i)}\}_{i=1}^n$ across n different prompts is then used to update the language model $\pi_{\boldsymbol{\varphi}}$. In the next two sections, we discuss two standard approaches to post-training: reinforcement learning from human feedback (RLHF) and direct preference optimization (DPO).

12.7.2 *Reinforcement Learning from Human Feedback*

RLHF separates the post-training step into two stages (Stiennon et al., 2020). First, the human feedback is used to learn an approximate reward model $r_{\boldsymbol{\theta}}$. This reward model is then used in the second stage to determine a refined policy $\Pi_{\boldsymbol{\varphi}}$.

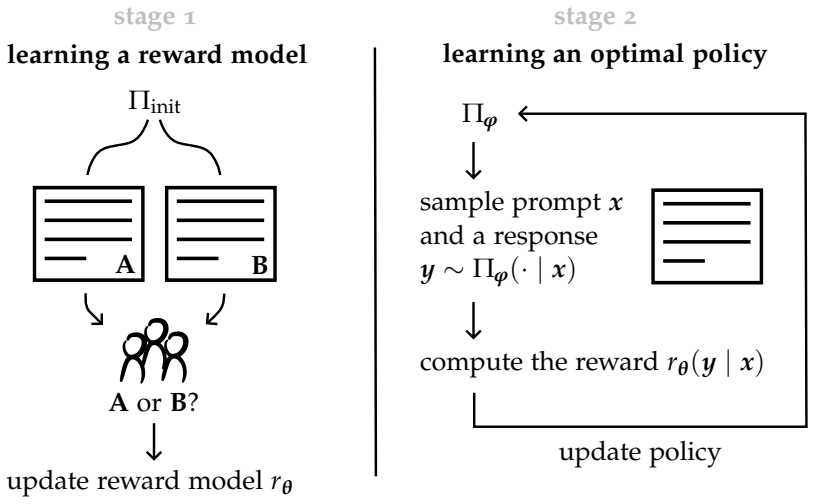


Figure 12.7: Illustration of the post-training process of RLHF.

Learning a reward model: During the first stage, the initial policy obtained by supervised fine-tuning Π_{init} is used to generate proposals $(\boldsymbol{y}_A, \boldsymbol{y}_B)$ for some exemplary prompts $\boldsymbol{x}$, which are then ranked according to the preference of human labelers. This preference data can be used to learn a reward model $r_{\boldsymbol{\theta}}$ by maximum likelihood esti-

mation (or equivalently minimizing cross-entropy):

$$\begin{aligned}&\arg\max_{\boldsymbol{\theta}} p(\mathcal{D} \mid r_{\boldsymbol{\theta}}) \\ &= \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{y}_A \succ \boldsymbol{y}_B \mid \boldsymbol{x}) \sim \mathcal{D}}[\log \sigma(r_{\boldsymbol{\theta}}(\boldsymbol{y}_A \mid \boldsymbol{x}) - r_{\boldsymbol{\theta}}(\boldsymbol{y}_B \mid \boldsymbol{x}))]\end{aligned} \tag{12.92}$$

using Equation (12.91) and SGD

This is analogous to the standard maximum likelihood estimation of the reward model in model-based RL with score feedback which we discussed in Section 11.2.1. The reward model $r_{\boldsymbol{\theta}}$ is often initialized from the initial policy π_{init} by placing a linear layer producing a scalar output on top of the final transformer layer.

Learning an optimal policy: One can now employ the methods from this and previous chapters to determine the optimal policy for the approximate reward $r_{\boldsymbol{\theta}}$. Due to the use of an *approximate* reward, however, simply maximizing $r_{\boldsymbol{\theta}}$ surfaces the so-called "reward gaming" problem which is illustrated in Figure 12.8. As the responses generated by the learned policy $\pi_{\boldsymbol{\varphi}}$ deviate from the distribution of $\mathcal{D}$ (i.e., the distribution induced by the initial policy π_{init}), the approximate reward model becomes inaccurate. The approximate reward may severely overestimate the true reward in regions far away from the training data. A common approach to address this problem is to regularize the policy search towards policies whose distribution does not stray away "too far" from the training distribution.

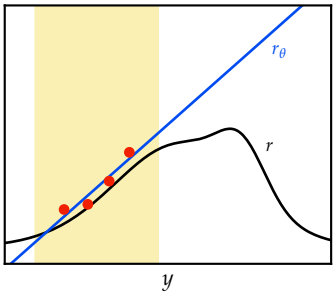


Figure 12.8: Illustration of "reward gaming". Shown in black is the true reward $r(y \mid x)$ for a fixed prompt x. Shown in blue is the approximation based on the feedback $\mathcal{D}$ to the responses shown in red. The yellow region symbolizes responses y where the approximate reward can still be "trusted".

Analogously to the trust-region methods we discussed in Section 12.4.4, the deviation from the initial policy is typically controlled by maximizing the regularized objective

$$J_\lambda(\boldsymbol{\varphi}; \boldsymbol{\varphi}^{\text{init}} \mid \boldsymbol{x}) \doteq \mathbb{E}_{\boldsymbol{y} \sim \Pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x})}[r(\boldsymbol{y} \mid \boldsymbol{x})] - \lambda \mathrm{KL}(\Pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x}) \| \Pi_{\text{init}}(\cdot \mid \boldsymbol{x}))$$

in expectation over prompts $\boldsymbol{x}$ sampled uniformly at random from the dataset $\mathcal{D}$. Note that this coincides with the PPO objective from Equation (12.63) with an outcome reward. We can expand the regularization term to obtain

$$\begin{aligned}&= \underbrace{\mathbb{E}_{\boldsymbol{y} \sim \Pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x})}[r(\boldsymbol{y} \mid \boldsymbol{x})] + \lambda \mathrm{H}[\Pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x})]}_{\text{entropy-regularized RL}} \\ &\qquad - \lambda \mathrm{H}[\Pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x}) \| \Pi_{\text{init}}(\cdot \mid \boldsymbol{x})]\end{aligned} \tag{12.93}$$

using the definition of KL-divergence (5.34)

which indicates an intimate relationship to entropy-regularized RL.[36]

[36] compare to Equation (12.79)

The optimal policy maximizing $J_\lambda(\boldsymbol{\varphi}; \boldsymbol{\varphi}^{\text{init}} \mid \boldsymbol{x})$ is (?)

Problem 12.8

$$\Pi_\star(\boldsymbol{y} \mid \boldsymbol{x}) \propto \Pi_{\text{init}}(\boldsymbol{y} \mid \boldsymbol{x}) \exp\left(\frac{1}{\lambda} r(\boldsymbol{y} \mid \boldsymbol{x})\right) \tag{12.94}$$

and can be interpreted as a probabilistic update to the prior Π_{init} where $\exp(\frac{1}{\lambda} r(\boldsymbol{y} \mid \boldsymbol{x}))$ corresponds to the "likelihood of optimality". As $\lambda \to \infty$, $\Pi_\star \to \Pi_{\text{init}}$, and as $\lambda \to 0$, the optimal policy reduces to deterministically picking the response with the highest reward.

In practice, sampling from $\Pi_\star$ explicitly is intractable, but note that any of the approximate inference methods discussed in Part I are applicable here. In the context of chatbots, it is important that the sampling from the resulting policy is efficient (i.e., the chatbot should respond quickly to prompts). Most commonly, the optimization problem of maximizing $j\boldsymbol{\varphi};\boldsymbol{\varphi}^{\text{init}} \mid \boldsymbol{x}[\lambda]$ (with the estimated reward $r_{\boldsymbol{\theta}}$) is solved approximately within a parameterized family of policies. This is typically done using policy gradient methods such as PPO (Stiennon et al., 2020) or GRPO (Guo et al., 2025).

Remark 12.18: Other (non-preference) reward models

Note that the post-training pipeline we described here is agnostic to the choice of reward model. That is, instead of post-training our language model to comply with human preferences, we could have also post-trained it to maximize any other reward signal. More recently, works have explored the use of reward models for challenging "reasoning problems" such as mathematical calculations, where the reward is usually based on the correctness of the answer.[37] One prominent example for this are "reasoning" models such as the DeepSeek-R1 model (Guo et al., 2025), which was trained with GRPO.

[37] On training data where the answer to problem $\boldsymbol{x}$ is known to be $y^\star(\boldsymbol{x})$, the reward is simply $r(\boldsymbol{y} \mid \boldsymbol{x}) = \mathbb{1}\{y^\star(\boldsymbol{x}) \in \boldsymbol{y}\}$. That is, the reward is 1 if the response $\boldsymbol{y}$ contains the correct answer and 0 otherwise.

12.7.3 *Direct Preference Optimization*

Observe that the reward model can be expressed in terms of its associated optimal policy:

$$r(\boldsymbol{y} \mid \boldsymbol{x}) = \lambda \log \frac{\Pi_\star(\boldsymbol{y} \mid \boldsymbol{x})}{\Pi_{\text{init}}(\boldsymbol{y} \mid \boldsymbol{x})} + \text{const.} \tag{12.95}$$

by reorganizing the terms of Equation (12.94)

In particular, it follows that given a fixed prior Π_{init}, *any* policy $\Pi_{\boldsymbol{\varphi}}$ has a family of associated reward models with respect to which it is optimal! We denote by

$$r_{[\boldsymbol{\varphi}]}(\boldsymbol{y} \mid \boldsymbol{x}) \doteq \lambda \log \frac{\Pi_{\boldsymbol{\varphi}}(\boldsymbol{y} \mid \boldsymbol{x})}{\Pi_{\text{init}}(\boldsymbol{y} \mid \boldsymbol{x})} \tag{12.96}$$

the "simplest" of these reward models. Remembering the characterization of the optimal policy from Equation (12.94) it follows immediately that $\Pi_{\boldsymbol{\varphi}}$ is optimal with respect to $r_{[\boldsymbol{\varphi}]}$.

Instead of first learning an approximate reward model and then finding the associated optimal policy, DPO exploits the relationship of

Equation (12.95) to learn the optimal policy directly (Rafailov et al., 2023). Substituting $r_{\boldsymbol{\theta}}$ in the maximum likelihood estimation from Equation (12.92), yields the objective

$$\begin{aligned}
&\mathbb{E}_{(\boldsymbol{y}_A \succ \boldsymbol{y}_B \mid \boldsymbol{x}) \sim \mathcal{D}} \left[\log \sigma \Big(r_{[\boldsymbol{\varphi}]}(\boldsymbol{y}_A \mid \boldsymbol{x}) - r_{[\boldsymbol{\varphi}]}(\boldsymbol{y}_B \mid \boldsymbol{x}) \Big) \right] \\
&= \mathbb{E}_{(\boldsymbol{y}_A \succ \boldsymbol{y}_B \mid \boldsymbol{x}) \sim \mathcal{D}} \left[\log \sigma \left(\lambda \log \frac{\Pi_{\boldsymbol{\varphi}}(\boldsymbol{y}_A \mid \boldsymbol{x})}{\Pi_{\text{init}}(\boldsymbol{y}_A \mid \boldsymbol{x})} - \lambda \log \frac{\Pi_{\boldsymbol{\varphi}}(\boldsymbol{y}_B \mid \boldsymbol{x})}{\Pi_{\text{init}}(\boldsymbol{y}_B \mid \boldsymbol{x})} \right) \right]. \qquad \text{using Equation (12.95)}
\end{aligned} \tag{12.97}$$

Gradients can be computed via automatic differentiation:

$$\begin{aligned}
\lambda \mathbb{E}_{(\boldsymbol{y}_A \succ \boldsymbol{y}_B \mid \boldsymbol{x}) \sim \mathcal{D}} \Bigg[& \underbrace{\sigma \Big(r_{[\boldsymbol{\varphi}]}(\boldsymbol{y}_B \mid \boldsymbol{x}) - r_{[\boldsymbol{\varphi}]}(\boldsymbol{y}_A \mid \boldsymbol{x}) \Big)}_{\text{weight according to error of reward estimate}} \\
& \Big[\underbrace{\nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\boldsymbol{y}_A \mid \boldsymbol{x})}_{\text{increase likelihood of } \boldsymbol{y}_A} - \underbrace{\nabla_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\boldsymbol{y}_B \mid \boldsymbol{x})}_{\text{decrease likelihood of } \boldsymbol{y}_B} \Big] \Bigg].
\end{aligned} \tag{12.98}$$

Intuitively, DPO successively increases the likelihood of preferred responses $\boldsymbol{y}_A$ and decreases the likelihood of dispreferred responses $\boldsymbol{y}_B$. Examples $(\boldsymbol{y}_A \succ \boldsymbol{y}_B \mid \boldsymbol{x})$ are weighted by the strength of regularization λ and by the degree to which the implicit reward model incorrectly orders the responses.

Discussion

In this chapter, we studied central ideas in actor-critic methods. We have seen two main approaches to use policy-gradient methods. We began, in Section 12.3, by introducing the REINFORCE algorithm which uses policy gradients and Monte Carlo estimation, but suffered from large variance in the gradient estimates of the policy value function. In Section 12.4, we have then seen a number of actor-critic methods such as A2C and GAE behaving similarly to policy iteration that exhibit less variance, but are very sample inefficient due to their on-policy nature. TRPO improves the sample efficiency slightly, but not fundamentally.

In Section 12.5, we discussed a second family of policy gradient techniques that generalize Q-learning and are akin to value iteration. For reparameterizable policies, this led us to algorithms such as DDPG, TD3, SVG. Importantly, these algorithms are significantly more sample efficient than on-policy policy gradient methods, which often results in much faster learning of a near-optimal policy. In Section 12.6, we discussed entropy regularization which frames reinforcement learning as probabilistic inference, and we derived the SAC algorithm which is widely used and works quite well in practice.

Finally, in Section 12.7, we studied two canonical approaches to learning from preference feedback: RLHF which separately learns reward model and policy and DPO which learns a policy directly. We have seen that RLHF is akin to model-based RL as it explicitly learns a reward model through maximum likelihood estimation. In contrast, DPO is more closely related to model-free RL and policy gradient methods as it learns the optimal policy directly.

Optional Readings

- **A3C:** Mnih, Badia, Mirza, Graves, Lillicrap, Harley, Silver, and Kavukcuoglu (2016).
  Asynchronous methods for deep reinforcement learning.
- **GAE:** Schulman, Moritz, Levine, Jordan, and Abbeel (2016).
  High-dimensional continuous control using generalized advantage estimation.
- **TRPO:** Schulman, Levine, Abbeel, Jordan, and Moritz (2015).
  Trust region policy optimization.
- **PPO:** Schulman, Wolski, Dhariwal, Radford, and Klimov (2017).
  Proximal policy optimization algorithms.
- **DDPG:** Lillicrap, Hunt, Pritzel, Heess, Erez, Tassa, Silver, and Wierstra (2016).
  Continuous control with deep reinforcement learning.
- **TD3:** Fujimoto, Hoof, and Meger (2018).
  Addressing function approximation error in actor-critic methods.
- **SVG:** Heess, Wayne, Silver, Lillicrap, Erez, and Tassa (2015).
  Learning continuous control policies by stochastic value gradients.
- **SAC:** Haarnoja, Zhou, Abbeel, and Levine (2018a).
  Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.
- **DPO:** Rafailov, Sharma, Mitchell, Ermon, Manning, and Finn (2023).
  Direct preference optimization: Your language model is secretly a reward model.

Problems

12.1. Q-learning and function approximation.

Consider the MDP of Figure 12.9 and set $\gamma = 1$.

1. Using Bellman's theorem, prove that $v^\star(x) = -|x-4|$ is the optimal value function.
2. Suppose we observe the following episode:

x	a	x'	r
3	-1	2	-1
2	1	3	-1
3	1	4	-1
4	1	4	0

   We initialize all Q-values to 0. Compute the updated Q-values using Q-learning with learning rate $\alpha = 1/2$.
3. We will now approximate the Q-function with a linear function. We let

$$Q(x, a; \boldsymbol{w}) \doteq x w_0 + a w_1 + w_2$$

   where $\boldsymbol{w} = [w_0 \; w_1 \; w_2]^\top \in \mathbb{R}^3$.
   Suppose we have $\boldsymbol{w}^{\text{old}} = [1 \;\; -1 \;\; -2]^\top$ and $\boldsymbol{w} = [-1 \;\; 1 \;\; 1]^\top$, and we observe the transition $\tau = (2, -1, -1, 1)$. Use the learning rate $\alpha = 1/2$ to compute $\boldsymbol{\nabla}_{\boldsymbol{w}} \ell(\boldsymbol{w}; \tau)$ and the updated weights $\boldsymbol{w}' = \boldsymbol{w} - \alpha \boldsymbol{\nabla}_{\boldsymbol{w}} \ell(\boldsymbol{w}; \tau)$.

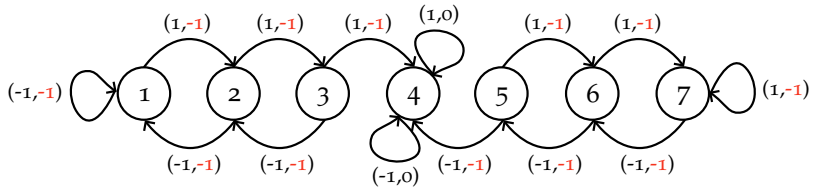


Figure 12.9: MDP studied in Problem 12.1. Each arrow marks a (deterministic) transition and is labeled with (action, reward).

12.2. Eligibility vector.

The vector $\boldsymbol{\nabla}_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t)$ is commonly called *eligibility vector*. In the following, we assume that the action space $\mathcal{A}$ is finite and denote it by A.

If we parameterize $\pi_{\boldsymbol{\varphi}}$ as a softmax distribution

$$\pi_{\boldsymbol{\varphi}}(a \mid \boldsymbol{x}) \doteq \frac{\exp(h(\boldsymbol{x}, a, \boldsymbol{\varphi}))}{\sum_{b \in A} \exp(h(\boldsymbol{x}, b, \boldsymbol{\varphi}))} \tag{12.99}$$

with linear preferences $h(\boldsymbol{x}, a, \boldsymbol{\varphi}) \doteq \boldsymbol{\varphi}^\top \boldsymbol{\phi}(\boldsymbol{x}, a)$ where $\boldsymbol{\phi}(\boldsymbol{x}, a)$ is some feature vector, what is the form of the eligibility vector?

12.3. Variance of score gradients with baselines.

In this exercise, we will see a sufficient condition for baselines to reduce the variance of score gradient estimators.

1. Suppose for a random vector $\mathbf{X}$, we want to estimate $\mathbb{E}[f(\mathbf{X})]$ for some function f. Assume that you are given a function g and also its expectation $\mathbb{E}[g(\mathbf{X})]$. Instead of estimating $\mathbb{E}[f(\mathbf{X})]$ directly, we will instead estimate $\mathbb{E}[f(\mathbf{X}) - g(\mathbf{X})]$ as we know from linearity of expectation (1.20) that

$$\mathbb{E}[f(\mathbf{X})] = \mathbb{E}[f(\mathbf{X}) - g(\mathbf{X})] + \mathbb{E}[g(\mathbf{X})].$$

   Prove that if $\frac{1}{2}\mathrm{Var}[g(\mathbf{X})] \leq \mathrm{Cov}[f(\mathbf{X}), g(\mathbf{X})]$, then

$$\mathrm{Var}[f(\mathbf{X}) - g(\mathbf{X})] \leq \mathrm{Var}[f(\mathbf{X})]. \tag{12.100}$$

2. Consider estimating $\boldsymbol{\nabla}_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi})$. Prove that if $b^2 \leq 2b \cdot r(\boldsymbol{x}, \boldsymbol{a})$ for every state $\boldsymbol{x} \in \mathcal{X}$ and action $\boldsymbol{a} \in \mathcal{A}$, then

$$\mathrm{Var}_{\tau\sim\Pi_{\boldsymbol{\varphi}}}\big[(G_0 - b)\boldsymbol{\nabla}_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau)\big] \leq \mathrm{Var}_{\tau\sim\Pi_{\boldsymbol{\varphi}}}\big[G_0 \boldsymbol{\nabla}_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau)\big]. \tag{12.101}$$

12.4. Score gradients with state-dependent baselines.

For a sequence of *state-dependent baselines* $\{b(\tau_{0:t-1})\}_{t=1}^{T}$ where

$$\tau_{0:t-1} \doteq (\tau_0, \tau_1, \ldots, \tau_{t-1}),$$

show that

$$\begin{aligned} &\mathbb{E}_{\tau\sim\Pi_{\boldsymbol{\varphi}}}\big[G_0 \boldsymbol{\nabla}_{\boldsymbol{\varphi}} \log \Pi_{\boldsymbol{\varphi}}(\tau)\big] \\ &= \mathbb{E}_{\tau\sim\Pi_{\boldsymbol{\varphi}}}\left[\sum_{t=0}^{T-1} (G_0 - b(\tau_{0:t-1}))\boldsymbol{\nabla}_{\boldsymbol{\varphi}} \log \pi_{\boldsymbol{\varphi}}(\boldsymbol{a}_t \mid \boldsymbol{x}_t)\right] \end{aligned} \tag{12.102}$$

where we write $b(\tau_{0:-1}) = 0$.

12.5. Policy gradients with downstream returns.

Suppose we are training an agent to solve a computer game. There are only two possible actions, specifically:

1. *do nothing*; and
2. *move*.

Each episode lasts for four ($T = 3$) time steps. The policy π_θ is completely determined by the parameter $\theta \in [0, 1]$. Here, for simplicity, we have assumed that the policy is independent of the current state. The probability of moving (action 2) is equal to θ and the probability of doing nothing (action 1) is $1 - \theta$.

Initially, $\theta = 0.5$. One episode is played with this initial policy and the results are

$$\text{actions} = (1, 0, 1, 0), \quad \text{rewards} = (1, 0, 1, 1).$$

Compute the policy gradient estimate with downstream returns, discount factor $\gamma = 1/2$, and the provided *single* sample $\tau \sim \Pi_\theta$.

12.6. Policy gradient with an exponential family.

1. Suppose, we can choose between two actions $a \in \{0, 1\}$ in each state. A natural stochastic policy is induced by a Bernoulli distribution,

$$a \sim \text{Bern}(\sigma(f_{\boldsymbol{\varphi}}(\boldsymbol{x}))), \tag{12.103}$$

   where σ is the logistic function from Equation (5.9). First, write down the expression for $\pi_{\boldsymbol{\varphi}}(a \mid \boldsymbol{x})$. Then, derive the expression for $\boldsymbol{\nabla}_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi})$ in terms of q^{π_φ}, $\sigma(f_{\boldsymbol{\varphi}}(\boldsymbol{x}))$, and $\boldsymbol{\nabla}_{\boldsymbol{\varphi}} f_{\boldsymbol{\varphi}}(\boldsymbol{x})$ using the policy gradient theorem.
2. The Bernoulli distribution is part of a family of distributions that allows for a much simpler derivation of the gradient than was necessary in (1). A univariate *exponential family* is a family of distributions whose PDF (or PMF) can be expressed in canonical form as

$$\pi_{\boldsymbol{\varphi}}(a \mid \boldsymbol{x}) = h(a) \exp(a f_{\boldsymbol{\varphi}}(\boldsymbol{x}) - A(f_{\boldsymbol{\varphi}}(\boldsymbol{x}))) \tag{12.104}$$

   where h, f, and A are known functions.[38] Derive the expression of the policy gradient $\boldsymbol{\nabla}_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi})$ for such a distribution.
3. Can you relate the results of the previous two exercises (1) and (2)? What are h and A in case of the Bernoulli distribution?
4. The Gaussian distribution with unit variance $\mathcal{N}(f_{\boldsymbol{\varphi}}(\boldsymbol{x}), 1)$ is of the same canonical form with

$$A(f_{\boldsymbol{\varphi}}(\boldsymbol{x})) = \frac{f_{\boldsymbol{\varphi}}(\boldsymbol{x})^2}{2}. \tag{12.105}$$

   Determine the policy gradient $\boldsymbol{\nabla}_{\boldsymbol{\varphi}} J(\boldsymbol{\varphi})$.
5. For a Gaussian policy, can we instead apply the reparameterization trick (5.65) that we have seen in the context of variational inference? If yes, how? If not, why?

[38] Observe that this form is equivalent to the form introduced in Equation (5.48) where $f_{\boldsymbol{\varphi}}(\boldsymbol{x})$ is the natural parameter, and we let $A(f) = \log Z(f)$.

12.7. Soft value function.

In this exercise, we derive the optimal policy solving Equation (12.84) and the soft value function from Equation (12.85).

1. We let $\beta(\boldsymbol{a}_t \mid \boldsymbol{x}_t) \doteq \exp(\frac{1}{\lambda} r(\boldsymbol{x}_t, \boldsymbol{a}_t))$, $Z(\boldsymbol{x}) \doteq \int_{\mathcal{A}} \beta(\boldsymbol{a} \mid \boldsymbol{x}) \, d\boldsymbol{a}$, and denote by $\hat{\pi}(\cdot \mid \boldsymbol{x})$ the policy $\beta(\cdot \mid \boldsymbol{x}) / Z(\boldsymbol{x})$. Show that

$$\begin{aligned} &\mathrm{KL}\big(\Pi_{\boldsymbol{\varphi}} \| \Pi_\star\big) \\ &\quad = \sum_{t=1}^{T} \mathbb{E}_{\boldsymbol{x}_t \sim \Pi_{\boldsymbol{\varphi}}} \big[\mathrm{KL}\big(\pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x}_t) \| \hat{\pi}(\cdot \mid \boldsymbol{x}_t)\big) - \log Z(\boldsymbol{x}_t)\big]. \end{aligned} \tag{12.106}$$

2. Show that if the space of policies parameterized by $\boldsymbol{\varphi}$ is sufficiently expressive, $\pi^\star(\boldsymbol{a} \mid \boldsymbol{x}) \propto \exp(q^\star(\boldsymbol{x}, \boldsymbol{a}))$ solves Equation (12.84).

12.8. PPO as probabilistic inference.

1. Consider the same generative model as was introduced in Section 12.6.1 when we interpreted entropy-regularized RL as probabilistic inference. Only now, assume that $T = 1$ [39] and suppose that the prior over actions is $p(\boldsymbol{y} \mid \boldsymbol{x}) = \Pi_{\text{init}}(\boldsymbol{y} \mid \boldsymbol{x})$ rather than uniform. Define the distribution over optimal trajectories $\Pi_\star(\boldsymbol{y} \mid \boldsymbol{x})$ as before in Equation (12.82). Show that for any context $\boldsymbol{x}$,

$$\arg\min_{\boldsymbol{\varphi}} \mathrm{KL}\big(\Pi_{\boldsymbol{\varphi}}(\cdot \mid \boldsymbol{x}) \| \Pi_\star(\cdot \mid \boldsymbol{x})\big) = \arg\max_{\boldsymbol{\varphi}} J_\lambda(\boldsymbol{\varphi}; \boldsymbol{\varphi}^{\text{init}} \mid \boldsymbol{x}). \tag{12.107}$$

2. Conclude that the policy maximizing $J_\lambda(\boldsymbol{\varphi}; \boldsymbol{\varphi}^{\text{init}} \mid \boldsymbol{x})$ is

$$\Pi_\star(\boldsymbol{y} \mid \boldsymbol{x}) \propto \Pi_{\text{init}}(\boldsymbol{y} \mid \boldsymbol{x}) \exp\left(\frac{1}{\lambda} r(\boldsymbol{y} \mid \boldsymbol{x})\right). \tag{12.108}$$

[39] since, in this section, we have been considering outcome rewards

13

Model-based Reinforcement Learning

In this final chapter, we will revisit the model-based approach to reinforcement learning. We will see some advantages it offers over model-free methods. In particular, we will use the machinery of probabilistic inference, which we developed in the first chapters, to quantify uncertainty about our model and use this uncertainty for planning, exploration, and reliable decision-making.

To recap, in Chapter 11, we began by discussing model-based reinforcement learning which attempts to learn the underlying Markov decision process and then use it for planning. We have seen that in the tabular setting, computing and storing the entire model is computationally expensive. This led us to consider the family of model-free approaches, which learn the value function directly, and as such can be considered more economical in the amount of data that they store.

In Chapter 12, we saw that using function approximation, we were able to scale model-free methods to very large state and action spaces. We will now explore similar ideas in the model-based framework. Namely, we will use function approximation to condense the representation of our model of the environment. More concretely, we will learn an approximate dynamics model $f \approx p$ and approximate rewards r, which is also called a *world model*.

There are a few ways in which the model-based approach is advantageous. First, if we have an accurate model of the environment, we can use it for planning — ideally also for interacting safely with the environment. However, in practice, we will rarely have such a model. In fact, the accuracy of our model will depend on the past experience of our agent and the region of the state-action space. Understanding the uncertainty in our model of the environment is crucial for planning. In particular, quantifying uncertainty is necessary to drive safe(!) exploration and avoid undesired states.

Moreover, modeling uncertainty in our model of the environment can be extremely useful in deciding where to explore. Learning a model can therefore help to dramatically reduce the sample complexity over model-free techniques. Often times this is crucial when developing agents for real-world use as in such settings, exploration is costly and potentially dangerous.

Algorithm 13.1 describes the general approach to model-based reinforcement learning.

Algorithm 13.1: Model-based reinforcement learning (outline)

```
start with an initial policy π and no (or some) initial data 𝒟
for several episodes do
    roll out policy π to collect data
    learn a model of the dynamics f and rewards r from data
    plan a new policy π based on the estimated models
```

We face three main challenges in model-based reinforcement learning. First, given a fixed model, we need to perform planning to decide on which actions to play. Second, we need to learn models f and r accurately and efficiently. Third, we need to effectively trade exploration and exploitation. We will discuss these three topics in the following.

13.1 Planning

There exists a large literature on planning in various settings. These settings can be mainly characterized as

- discrete or continuous action spaces;
- fully- or partially observable state spaces;
- constrained or unconstrained; and
- linear or nonlinear dynamics.

In Chapter 10, we have already seen algorithms such as policy iteration and value iteration, which can be used to solve planning exactly in tabular settings. In the following, we will now focus on the setting of continuous state and action spaces, fully observable state spaces, no constraints, and nonlinear dynamics.

13.1.1 Deterministic Dynamics

To begin with, let us assume that our dynamics model is deterministic and known. That is, given a state-action pair, we know the subsequent state,

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{a}_t). \tag{13.1}$$

We continue to focus on the setting of infinite-horizon discounted returns (10.5), which we have been considering throughout our discussion of reinforcement learning. This yields the objective,

$$\max_{\boldsymbol{a}_{0:\infty}} \sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{x}_t, \boldsymbol{a}_t) \quad \text{such that } \boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{a}_t). \tag{13.2}$$

Now, because we are optimizing over an infinite time horizon, we cannot solve this optimization problem directly. This problem is studied ubiquitously in the area of *optimal control*. We will discuss one central idea from optimal control that is widely used in model-based reinforcement learning, and will later return to using this idea for learning parametric policies in Section 13.1.3.

Planning over finite horizons: The key idea of a classical algorithm from optimal control called *receding horizon control* (RHC) or *model predictive control* (MPC) is to iteratively plan over finite horizons. That is, in each round, we plan over a finite time horizon H and carry out the first action.

Algorithm 13.2: Model predictive control, MPC

1 **for** $t = 0$ **to** ∞ **do**
2 observe $\boldsymbol{x}_t$
3 plan over a finite horizon H,
$$\max_{\boldsymbol{a}_{t:t+H-1}} \sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} r(\boldsymbol{x}_\tau, \boldsymbol{a}_\tau) \quad \text{such that } \boldsymbol{x}_{\tau+1} = \boldsymbol{f}(\boldsymbol{x}_\tau, \boldsymbol{a}_\tau) \tag{13.3}$$
4 carry out action $\boldsymbol{a}_t$

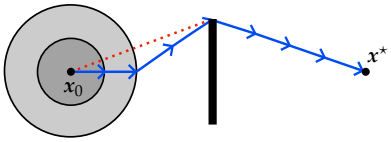


Figure 13.1: Illustration of model predictive control in a deterministic transition model. The agent starts in position $\boldsymbol{x}_0$ and wants to reach $\boldsymbol{x}^\star$ despite the black obstacle. We use the reward function $r(\boldsymbol{x}) = -\|\boldsymbol{x} - \boldsymbol{x}^\star\|$. The gray concentric circles represent the length of a single step. We plan with a time horizon of $H = 2$. Initially, the agent does not "see" the black obstacle, and therefore moves straight towards the goal. As soon as the agent sees the obstacle, the optimal trajectory is "replanned". The dotted red line corresponds to the optimal trajectory, the agent's steps are shown in blue.

Observe that the state $\boldsymbol{x}_\tau$ can be interpreted as a deterministic function $\boldsymbol{x}_\tau(\boldsymbol{a}_{t:\tau-1})$, which depends on all actions from time t to time $\tau - 1$ and the state $\boldsymbol{x}_t$. To solve the optimization problem of a single iteration, we therefore need to maximize,

$$J_H(\boldsymbol{a}_{t:t+H-1}) \doteq \sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} r(\boldsymbol{x}_\tau(\boldsymbol{a}_{t:\tau-1}), \boldsymbol{a}_\tau). \tag{13.4}$$

This optimization problem is in general non-convex. If the actions are continuous and the dynamics and reward models are differentiable, we can nevertheless obtain analytic gradients of $\widehat{J}_H$. This can be done using the chain rule and "backpropagating" through time, analogously to backpropagation in neural networks.[1] Especially for large horizons H, this optimization problem becomes difficult to solve exactly due to local optima and vanishing/exploding gradients.

[1] see Section 7.1.4

Tree search: Often, heuristic global optimization methods (also called "search methods") are used to optimize $\widehat{J}_H$. An example are *random shooting methods*, which find the optimal choice among a set of random proposals. Of course, obtaining a "good" set of randomly proposed action sequences is crucial. A naive way of generating the proposals is to pick them uniformly at random. This strategy, however, usually does not perform very well as it corresponds to suggesting random walks of the state space. Alternatives are to sample from a Gaussian distribution or using the *cross-entropy method* which gradually adapts the sampling distribution by reweighing samples according to the rewards they produce.

Algorithm 13.3: Random shooting methods

1 generate m sets of random samples, $\boldsymbol{a}_{t:t+H-1}^{(i)}$

2 pick the sequence of actions $\boldsymbol{a}_{t:t+H-1}^{(i^\star)}$ where

$$i^\star = \arg\max_{i \in [m]} J_H(\boldsymbol{a}_{t:t+H-1}^{(i)}) \tag{13.5}$$

The evolution of the state can be visualized as a tree where — if dynamics are deterministic — the branching is fully determined by the played actions. For this reason, classical tree search algorithms can be employed, such as *alpha-beta pruning* or *Monte Carlo tree search* (MCTS), which was used for example by "AlphaZero" (Silver et al., 2016, 2017) and can be viewed as an advanced variant of a shooting method.

Finite-horizon planning with sparse rewards: A common problem of finite-horizon methods is that in the setting of sparse rewards, there is often no signal that can be followed. You can think of an agent that operates in a kitchen and tries to find a box of candy. Yet, to get to this box, it needs to perform a large number of actions. In particular, if this number of actions is larger than the horizon H, then the local optimization problem of MPC will not take into account the reward for choosing this long sequence of actions. Thus, the box of candy will likely never be found by our agent.

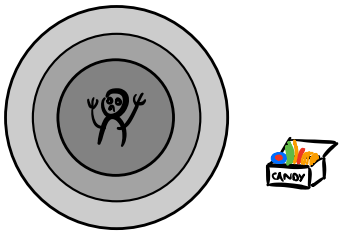


Figure 13.2: Illustration of finite-horizon planning with sparse rewards. When the finite time horizon does not suffice to "reach" a reward, the agent has no signal to follow.

A solution to this problem is to amend a long-term value estimate to the finite-horizon sum. The idea is to not only consider the rewards attained *while* following the actions $\boldsymbol{a}_{t:t_H-1}$, but to also consider the value of the final state $\boldsymbol{x}_{t+H}$, which estimates the discounted sum of

future rewards.

$$\widehat{J}_H(\boldsymbol{a}_{t:t+H-1}) \doteq \underbrace{\sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} r(\boldsymbol{x}_\tau(\boldsymbol{a}_{t:\tau-1}), \boldsymbol{a}_\tau)}_{\text{short-term}} + \underbrace{\gamma^H V(\boldsymbol{x}_{t+H})}_{\text{long-term}}. \tag{13.6}$$

Intuitively, $\gamma^H V(\boldsymbol{x}_{t+H})$ is estimating the tail of the infinite sum.

Remark 13.4: Planning generalizes model-free methods!

Observe that for $H = 1$, when we use the value function estimate associated with this MPC controller, maximizing J_H coincides with using the greedy policy π_V. That is,

$$\boldsymbol{a}_t = \arg\max_{\boldsymbol{a} \in \mathcal{A}} \widehat{J}_1(\boldsymbol{a}) = \pi_V(\boldsymbol{x}_t). \tag{13.7}$$

Thus, by looking ahead for a single time step, we recover the approaches from the model-free setting in this model-based setting! Essentially, if we do not plan long-term and only consider the value estimate, the model-based setting reduces to the model-free setting. However, in the model-based setting, we are now able to use our model of the transition dynamics to anticipate the downstream effects of picking a particular action $\boldsymbol{a}_t$. This is one of the fundamental reasons for why model-based approaches are typically severely more sample efficient than model-free methods.

To obtain the value estimates, we can use the approaches we discussed in detail in Section 11.4, such as TD-learning for on-policy value estimates and Q-learning for off-policy value estimates. For large state-action spaces, we can use their approximate variants, which we discussed in Section 12.1 and Section 12.2. To improve value estimates, we can obtain "artificial" data by rolling out policies within our model. This is a key advantage over model-free methods as, once a sufficiently accurate model has been learned, data for value estimation can be generated efficiently in simulation.

13.1.2 Stochastic Dynamics

How can we extend this approach to planning to a stochastic transition model? A natural extension of model predictive control is to do what is called *stochastic average approximation* (SAA) or *trajectory sampling* (Chua et al., 2018). Like in MPC, we still optimize over a deterministic sequence of actions, but now we will average over all resulting trajectories.

Algorithm 13.5: Trajectory sampling

for $t = 0$ **to** ∞ **do**
  observe $\boldsymbol{x}_t$
  optimize expected performance over a finite horizon H,

$$\max_{\boldsymbol{a}_{t:t+H-1}} \mathbb{E}_{\boldsymbol{x}_{t+1:t+H}} \left[\sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} r_\tau + \gamma^H V(\boldsymbol{x}_{t+H}) \,\middle|\, \boldsymbol{a}_{t:t+H-1}, f \right] \tag{13.8}$$

  carry out action $\boldsymbol{a}_t$

Intuitively, trajectory sampling optimizes over a much simpler object — namely a deterministic sequence of actions of length H — than finding a policy, which corresponds to finding an optimal decision tree mapping states to actions. Of course, using trajectory sampling (from an arbitrary starting state) implies such a policy. However, trajectory sampling never computes this policy explicitly, and rather, in each step, only plans over a finite horizon.

Figure 13.3: Illustration of trajectory sampling. High-reward states are shown in brighter colors. The agent iteratively plans a finite number of time steps into the future and picks the best initial action.

Computing the expectation exactly is typically not possible as this involves solving a high-dimensional integral of nonlinear functions. Instead, a common approach is to use Monte Carlo estimates of this expectation. This approach is known as *Monte Carlo trajectory sampling*. The key issue with using sampling based estimation is that the sampled trajectory (i.e., sampled sequence of states) we obtain, depends on the actions we pick. In other words, the measure we average over depends on the decision variables — the actions. This is a problem that we have seen several times already! It naturally suggests using the reparameterization trick.[2]

[2] see Theorem 5.19 and Equation (12.76)

Previously, we have used the reparameterization trick to reparameterize variational distributions (see Section 5.5.1) and to reparameterize policies (see Section 12.5.1). It turns out that we can use the exact same approach for reparameterizing the transition model. We say that a (stochastic) transition model f is *reparameterizable* iff $\boldsymbol{x}_{t+1} \sim f(\boldsymbol{x}_t, \boldsymbol{a}_t)$ is such that $\boldsymbol{x}_{t+1} = g(\varepsilon; \boldsymbol{x}_t, \boldsymbol{a}_t)$, where $\varepsilon \sim \phi$ is a random variable that is independent of $\boldsymbol{x}_t$ and $\boldsymbol{a}_t$. We have already seen in Example 12.14 (in the context of stochastic policies) how a Gaussian transition model can be reparameterized.

In this case, $\boldsymbol{x}_\tau$ is determined recursively by $\boldsymbol{a}_{t:\tau-1}$ and $\varepsilon_{t:\tau-1}$,

$$\begin{aligned} \boldsymbol{x}_\tau &= \boldsymbol{x}_\tau(\varepsilon_{t:\tau-1}; \boldsymbol{a}_{t:\tau-1}) \\ &\doteq g(\varepsilon_{\tau-1}; g(\dots; (\varepsilon_{t+1}; g(\varepsilon_t; \boldsymbol{x}_t, \boldsymbol{a}_t), \boldsymbol{a}_{t+1}), \dots), \boldsymbol{a}_{\tau-1}). \end{aligned} \tag{13.9}$$

This allows us to obtain unbiased estimates of J_H using Monte Carlo approximation,

$$\widehat{J}_H(\boldsymbol{a}_{t:t+H-1}) \approx \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{\tau=t}^{t+H-1}\gamma^{\tau-t}r(\boldsymbol{x}_\tau(\boldsymbol{\varepsilon}^{(i)}_{t:\tau-1};\boldsymbol{a}_{t:\tau-1}),\boldsymbol{a}_\tau) + \gamma^H V(\boldsymbol{x}_{t+H}(\boldsymbol{\varepsilon}^{(i)}_{t:t+H-1};\boldsymbol{a}_{t:t+H-1}))\right) \tag{13.10}$$

where $\boldsymbol{\varepsilon}^{(i)}_{t:t+H-1} \overset{\text{iid}}{\sim} \phi$ are independent samples. To optimize this approximation we can again compute analytic gradients or use shooting methods as we have discussed in Section 13.1.1 for deterministic dynamics.

13.1.3 Parametric Policies

When using algorithms such as model predictive control for planning, planning needs to be done online before each time we take an action. This is called *closed-loop control* and can be expensive. Especially when the time horizon is large, or we encounter similar states many times (leading to "repeated optimization problems"), it can be beneficial to "store" the planning decision in a (deterministic) policy,

$$\boldsymbol{a}_t = \boldsymbol{\pi}(\boldsymbol{x}_t;\boldsymbol{\varphi}) \doteq \boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x}_t). \tag{13.11}$$

This policy can then be trained offline and evaluated cheaply online, which is known as *open-loop control*.

This is akin to a problem that we have discussed in detail in the previous chapter when extending Q-learning to large action spaces. There, this led us to discuss policy gradient and actor-critic methods. Recall that in Q-learning, we seek to follow the greedy policy,

$$\boldsymbol{\pi}^\star(\boldsymbol{x}) = \arg\max_{\boldsymbol{a}\in\mathcal{A}} Q^\star(\boldsymbol{x},\boldsymbol{a};\boldsymbol{\theta}), \qquad \text{see Equation (12.20)}$$

and therefore had to solve an optimization problem over all actions. We accelerated this by learning an approximate policy that "mimicked" this optimization,

$$\boldsymbol{\varphi}^\star = \arg\max_{\boldsymbol{\varphi}} \underbrace{\mathbb{E}_{\boldsymbol{x}\sim\mu}\left[Q^\star(\boldsymbol{x},\boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x});\boldsymbol{\theta})\right]}_{\widehat{J}_\mu(\boldsymbol{\varphi};\boldsymbol{\theta})} \qquad \text{see Equation (12.67)}$$

where $\mu(\boldsymbol{x}) > 0$ was some exploration distribution that has full support and thus leads to the exploration of all states. The key idea was that if we use a differentiable approximation Q and a differentiable parameterization of policies, which is "rich enough", then both optimizations are equivalent, and we can use the chain rule to obtain

gradient estimates of the second expression. We then used this to derive the *deep deterministic policy gradients* (DDPG) and *stochastic value gradients* (SVG) algorithms. It turns out that there is a very natural analogue to DDPG/SVG for model-based reinforcement learning.

Instead of maximizing the Q-function directly, we use finite-horizon planning to estimate the immediate value of the policy within the next H time steps and simply use the Q-function to approximate the terminal value (i.e., the tails of the infinite sum). Then, our objective becomes,

$$\widehat{J}_{\mu,H}(\boldsymbol{\varphi};\boldsymbol{\theta}) \doteq \mathbb{E}_{\boldsymbol{x}_0\sim\mu,\boldsymbol{x}_{1:H}|\boldsymbol{\pi}_{\boldsymbol{\varphi}},f}\left[\sum_{\tau=0}^{H-1}\gamma^\tau r_\tau + \gamma^H Q^\star(\boldsymbol{x}_H, \boldsymbol{\pi}_{\boldsymbol{\varphi}}(\boldsymbol{x}_H);\boldsymbol{\theta})\right] \tag{13.12}$$

This approach naturally extends to randomized policies using reparameterization gradients, which we have discussed in Section 12.5.1. Analogously to Remark 13.4, for $H = 0$, this coincides with the DDPG objective! For larger time horizons, the look-ahead takes into account the transition model for planning next time steps. This tends to help dramatically in improving policies *much more rapidly* between episodes. Instead of just gradually improving policies a little by slightly adapting the policy to the learned value function estimates (as in model-free RL), we use the model to anticipate the consequences of actions multiple time steps ahead. This is at the heart of model-based reinforcement learning.

Essentially, we are using methods such as Q-learning and DDPG/SVG as subroutines within the framework of model predictive control to do much bigger steps in policy improvement than to slightly improve the next picked action. To encourage exploration, it is common to extend the objective in Equation (13.12) by an additional entropy regularization term as seen in Section 12.6.

Remark 13.6: Planning as inference

We have been using *Monte Carlo rollouts* to estimate the expectation of Equation (13.12). That is, we have been using a Monte Carlo approximation (e.g., a sample mean) using samples obtained by "rolling out" the induced Markov chain of a fixed policy.

It would certainly be preferable to compute the expectation exactly, however, this is generally not possible as this involves solving a high dimensional integral. Recall that we faced the same problem when studying inference in the first half of the manuscript. In both problems, we need to approximate high-dimensional integrals (i.e., expectations). This suggests a deep connection between

the problems of planning and inference. It is therefore not surprising that many techniques for approximate inference that we have seen earlier can also be applied to planning.

- *Monte Carlo approximation* which we have been focusing on during our discussion of planning is a very simple inference algorithm — approximating an expectation by sampling from the distribution that is averaged over. This allowed us to obtain unbiased gradient estimates (which may have high variance).
- An alternative approach is *moment matching* (cf. Section 5.4.6). Instead of approximating the expectation, here, we approximate the distribution over trajectories using a tractable distribution (e.g., a Gaussian) and "matching" their moments. This then allows us to analytically compute gradients of the expectation in Equation (13.12). A prominent example of this approach is *probabilistic inference for learning control* (PILCO).
- In Section 12.6, we have used *variational inference* for planning, and seen that it coincides with entropy regularization as implemented by the *soft actor critic* (SAC) algorithm.

13.2 Learning

Thus far, we have considered known environments. That is, we assumed that the transition model f and the rewards r are known. In reinforcement learning, f and r are (of course!) not known. Instead, we have to estimate them from data. This will also be crucial in our later discussion of exploration in Section 13.3 where we explore methods of driving data collection to learn what we need to learn about the world.

First, let us revisit one of our key observations when we first introduced the reinforcement learning problem. Namely, that the observed transitions $\boldsymbol{x}'$ and rewards r are conditionally independent given the state-action pairs $(\boldsymbol{x}, \boldsymbol{a})$.[3] This is due to the Markovian structure of the underlying Markov decision process.

[3] see Equation (11.3)

This is the key observation that allows us to treat the estimation of the dynamics and rewards as a simple regression problem (or a density estimation problem when the quantities are stochastic rather than deterministic). More concretely, we can estimate the dynamics and rewards off-policy using the standard supervised learning techniques we discussed in earlier chapters, from a replay buffer

$$\mathcal{D} = \{(\underbrace{\boldsymbol{x}_t, \boldsymbol{a}_t}_{\text{"input"}}, \underbrace{r_t, \boldsymbol{x}_{t+1}}_{\text{"label"}})\}_t. \tag{13.13}$$

Here, $\boldsymbol{x}_t$ and $\boldsymbol{a}_t$ are the "inputs", and r_t and $\boldsymbol{x}_{t+1}$ are the "labels" of the

regression problem. Due to the conditional independence of the labels given the inputs, we have independent label noise (i.e., "independent training data") which is the basic assumption that we have been making throughout our discussion of techniques for probabilistic machine learning in Part I.

The key difference to supervised learning is that the set of inputs depends on how we act. That is, the current inputs arise from previous policies, and the inputs which we will observe in the future will depend on the model (and policy) obtained from the current data: we have feedback loops! We will come back to this aspect of reinforcement learning in the next section on exploration. For now, recall we only assume that we have used an arbitrary policy to collect some data, which we then stored in a replay buffer, and which we now want to use to learn the "best-possible" model of our environment.

13.2.1 Probabilistic Inference

In the following, we will discuss how we can use the techniques from probabilistic inference, which we have seen in Part I, to learn the dynamics and reward models. Thereby, we will focus on learning the transition model f as learning the reward model r is completely analogous. For learning deterministic dynamics or rewards, we can use for example Gaussian processes (cf. Chapter 4) or deep neural networks (cf. Chapter 7). We will now focus on the setting where the dynamics are stochastic, that is,

$$\boldsymbol{x}_{t+1} \sim f(\boldsymbol{x}_t, \boldsymbol{a}_t; \boldsymbol{\psi}). \tag{13.14}$$

Example 13.7: Conditional Gaussian dynamics

We could, for example, use a conditional Gaussian for the transition model,

$$\boldsymbol{x}_{t+1} \sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}_t, \boldsymbol{a}_t; \boldsymbol{\psi}), \boldsymbol{\Sigma}(\boldsymbol{x}_t, \boldsymbol{a}_t; \boldsymbol{\psi})). \tag{13.15}$$

As we have seen in Equation (A.7), we can rewrite the covariance matrix $\boldsymbol{\Sigma}$ as a product of a lower-triangular matrix $\mathcal{L}$ and its transpose using the Cholesky decomposition $\boldsymbol{\Sigma} = \mathcal{L}\mathcal{L}^\top$ of $\boldsymbol{\Sigma}$. This allows us to represent the model by only $n(n+1)/2$ parameters. Moreover, we have learned that Gaussians are reparameterizable, which we have seen to be useful for planning.

Note that this model reduces to a deterministic model if the covariance is zero. So the stochastic transition models encompass all deterministic models. Moreover, in many applications (such as robotics), it is often useful to use stochastic models to attribute

slight inaccuracies and measurement noise to a small uncertainty in the model.

A first approach might be to obtain a point estimate for f, either through maximum likelihood estimation (which we have seen to overfit easily) or through maximum a posteriori estimation. If, for example, f is represented as a deep neural network, we have already seen how to find the MAP estimate of its weights in Section 7.2.1.

Remark 13.8: The key pitfall of point estimates

However, using point estimates leads to a *key pitfall* of model-based reinforcement learning. Using a point estimate of the model for planning — even if this point estimate is very accurate — often performs *very poorly*. The reason is that planning is very good at overfitting (i.e., exploiting) small errors in the transition model. Moreover, the errors in the model estimate compound over time when using a longer time horizon H. The key to remedy this pitfall lies in being robust to misestimated models. This naturally suggests quantifying the uncertainty in our model estimate and taking it into account during planning.[4] In the following section, we will rediscover that estimates of epistemic uncertainty are also extremely useful for driving (safe) exploration — something that we have already encountered in our discussion of Bayesian optimization.

[4] Quantifying the uncertainty of an estimate is a problem that we have spent the first few chapters exploring. Notably, refer to

- Section 2.2 for a description of epistemic and aleatoric uncertainty;
- Chapter 4 for our use of uncertainty estimates in the context of Gaussian processes;
- Chapter 7 for our use of uncertainty estimates in the context of Bayesian deep learning; and
- Chapters 8 and 9 for our use of epistemic uncertainty estimates to drive exploration.

In the following, we will differentiate between the epistemic uncertainty and the aleatoric uncertainty. Recall from Section 2.2 that epistemic uncertainty corresponds to our uncertainty about the model, $p(f \mid \mathcal{D})$, while aleatoric uncertainty corresponds to the uncertainty of the transitions in the underlying Markov decision process (which can be thought of as "irreducible" noise), $p(\boldsymbol{x}_{t+1} \mid f, \boldsymbol{x}_t, \boldsymbol{a}_t)$.

Intuitively, probabilistic inference of dynamics models corresponds to learning a distribution over possible models f and r given prior *beliefs*, where f and r characterize the underlying Markov decision process. This goes to show another benefit of the model-based over the model-free approach to reinforcement learning. Namely, that it is much easier to encode prior knowledge about the transition and rewards model.

Example 13.9: Inference with conditional Gaussian dynamics

Let us revisit inference with our conditional Gaussian dynamics

model from Equation (13.15),

$$x_{t+1} \sim \mathcal{N}(\mu(x_t, a_t; \psi), \Sigma(x_t, a_t; \psi)).$$

Recall that in the setting of Bayesian deep learning, most approximate inference techniques represented the approximate posterior using some form of a mixture of Gaussians,[5]

$$p(x_{t+1} \mid \mathcal{D}, x_t, a_t) \approx \frac{1}{m} \sum_{i=1}^{m} \mathcal{N}(\mu(x_t, a_t; \psi^{(i)}), \Sigma(x_t, a_t; \psi^{(i)})). \tag{13.16}$$

Hereby, the epistemic uncertainty is represented by the variance between mixture components, and the aleatoric uncertainty by the average variance within the components.[6]

[5] see
- Equation (7.18) for variational inference;
- Equation (7.21a) for Markov chain Monte Carlo;
- Equation (7.28) for dropout regularization; and
- Equation (7.29) for probabilistic ensembles.

[6] see Section 7.3.1

In supervised learning, we often conflated the notions of epistemic and aleatoric uncertainty. In the context of planning, there is an important consequence of the decomposition into epistemic and aleatoric uncertainty. Recall that the epistemic uncertainty corresponds to a distribution over Markov decision processes f, whereas the aleatoric uncertainty corresponds to the randomness in the transitions within one such MDP f. Crucially, this randomness in the transitions must be consistent within a single MDP! That is, once we selected a single MDP for planning, we should disregard the epistemic uncertainty and solely focus on the randomness of the transitions. Then, to take into account epistemic uncertainty, we should average our plan across the different realizations of f. This yields the following Monte Carlo estimate of our reward $\widehat{J}_H$,

$$\widehat{J}_H(a_{t:t+H-1}) \approx \frac{1}{m} \sum_{i=1}^{m} \widehat{J}_H(a_{t:t+H-1}; f^{(i)}) \qquad \text{where} \tag{13.17}$$

$$\widehat{J}_H(a_{t:t+H-1}; f) \doteq \sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} r(x_\tau(\varepsilon^{(i)}_{t:\tau-1}; a_{t:\tau-1}, f), a_\tau) + \gamma^H V(x_{t+H}). \tag{13.18}$$

Here, $f^{(i)} \overset{\text{iid}}{\sim} p(f \mid \mathcal{D})$ are independent samples of the transition model, and $\varepsilon^{(i)}_{t:t+H-1} \overset{\text{iid}}{\sim} \phi$ parameterizes the dynamics analogously to Equation (13.9):

$$\begin{aligned} &x_\tau(\varepsilon_{t:\tau-1}; a_{t:\tau-1}, f) \\ &\doteq f(\varepsilon_{\tau-1}; f(\dots; (\varepsilon_{t+1}; f(\varepsilon_t; x_t, a_t), a_{t+1}), \dots), a_{\tau-1}). \end{aligned} \tag{13.19}$$

Observe that the epistemic and aleatoric uncertainty are treated differently. Within a particular MDP f, we ensure that randomness (i.e., ale-

atoric uncertainty) is simulated consistently using our previous framework from our discussion of planning.[7] The Monte Carlo samples of f take into account the epistemic uncertainty about the transition model. In our previous discussion of planning, we assumed the Markov decision process f to be fixed. Essentially, in Equation (13.17) we are now using Monte Carlo trajectory sampling as a subroutine and average over an "ensemble" of Markov decision processes.

[7] see Equation (13.10)

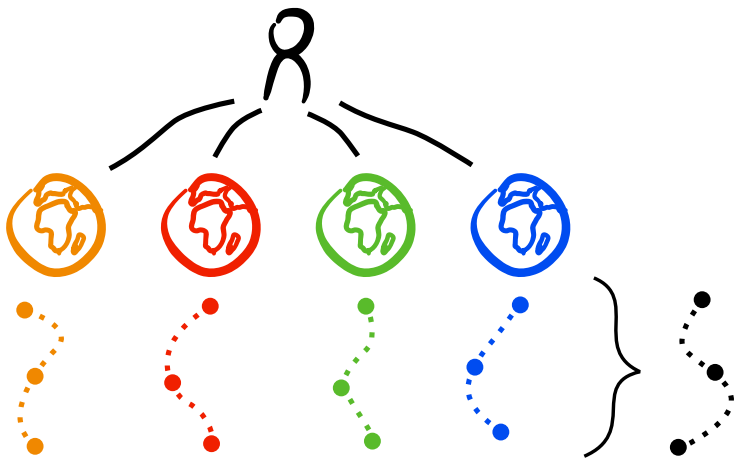


Figure 13.4: Illustration of planning with epistemic uncertainty and Monte Carlo sampling. The agent considers m alternative "worlds". Within each world, it plans a sequence of actions over a finite time horizon. Then, the agent averages all optimal initial actions from all worlds. Crucially, each world by itself is *consistent*. That is, its transition model (i.e., the aleatoric uncertainty of the model) is constant.

The same approach that we have seen in Section 13.1.3 can be used to "compile" these plans into a parametric policy that can be trained offline, in which case, we write $\widehat{J}_H(\pi)$ instead of $\widehat{J}_H(\boldsymbol{a}_{t:t+H-1})$.

This leads us to a first greedily exploitative algorithm for model-based reinforcement learning, which is shown in Algorithm 13.10. This algorithm is purely exploitative as it greedily maximizes the expected reward with respect to the transition model, taking into account epistemic uncertainty.

Algorithm 13.10: Greedy exploitation for model-based RL

start with (possibly empty) data $\mathcal{D} = \emptyset$ and a prior $p(f) = p(f \mid \mathcal{D})$
for several episodes **do**
  plan a new policy π to (approximately) maximize,

$$\max_{\pi} \mathbb{E}_{f \sim p(\cdot \mid \mathcal{D})} \left[\widehat{J}_H(\pi; f) \right] \tag{13.20}$$

  roll out policy π to collect more data
  update posterior $p(f \mid \mathcal{D})$

In the context of Gaussian process models, this algorithm is called

probabilistic inference for learning control (PILCO) (Deisenroth and Rasmussen, 2011), which was the first demonstration of how sample efficient model-based reinforcement learning can be. As was mentioned in Remark 13.6, the originally proposed PILCO uses moment matching instead of Monte Carlo averaging.

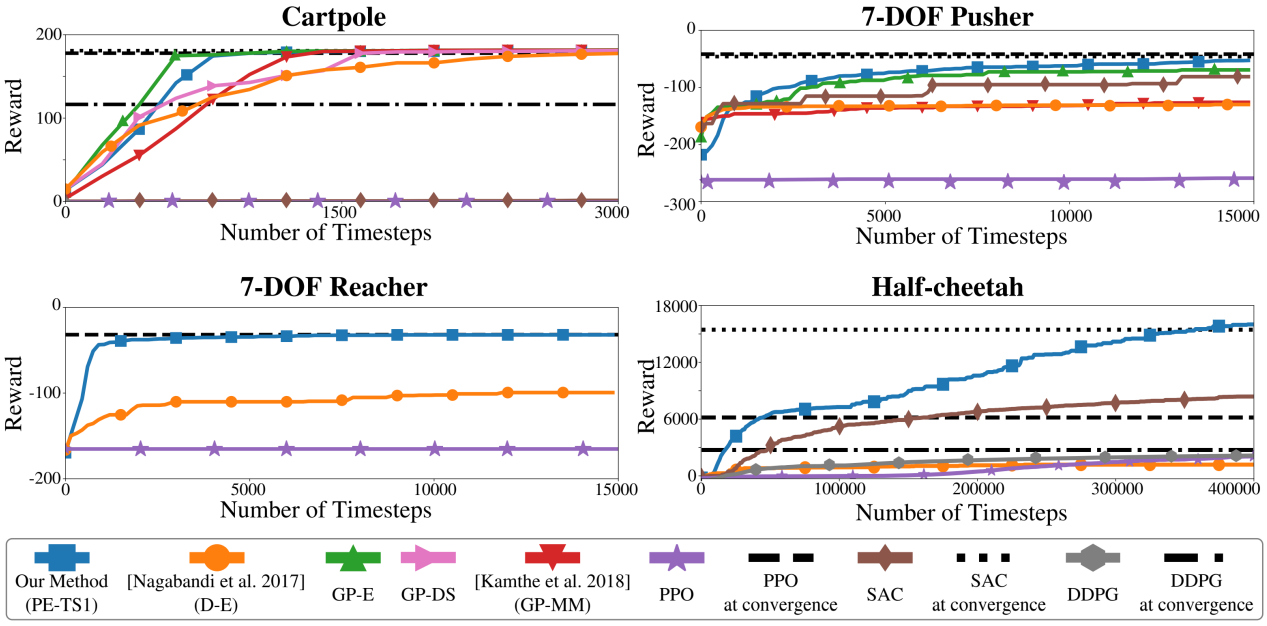


Figure 13.5: Sample efficiency of model-free and model-based reinforcement learning. DDPG and SAC are shown as constant (black) lines, because they take an order of magnitude more time steps before learning a good model. They also compare the PETS algorithm (blue) to deterministic ensembles (orange), where the transition models are assumed to be deterministic (or to have covariance 0). Reproduced with permission from "Deep reinforcement learning in a handful of trials using probabilistic dynamics models" (Chua et al., 2018).

In the context of neural networks, this algorithm is called *probabilistic ensembles with trajectory sampling* (PETS) (Chua et al., 2018), which is one of the state-of-the-art algorithms. PETS uses an ensemble of conditional Gaussian distributions over weights, trajectory sampling for evaluating the performance and model predictive control for planning.

Notably, PETS does not explicitly explore. Exploration only happens due to the uncertainty in the model, which already drives exploration to some extent. In many settings, however, this incentive is not sufficient for exploration — especially when rewards are sparse.

Optional Readings

- **PILCO:** Deisenroth and Rasmussen (2011).
  PILCO: A model-based and data-efficient approach to policy search.
- **PETS:** Chua, Calandra, McAllister, and Levine (2018).
  Deep reinforcement learning in a handful of trials using probabilistic dynamics models.

13.2.2 Partial Observability

Depending on the task, it may be difficult to learn the observed dynamics directly. For example, when learning an artificial player for a computer game based only on the games' visual interface it may be difficult to predict the next frame in pixel space. Instead, a common approach is to treat the visual interface as an observation in a POMDP with a hidden latent space (cf. Section 10.4), and to learn the dynamics within the latent space and the observation model separately.

An example of this approach is the *deep planning network* (PlaNet) algorithm which learns such a POMDP via variational inference and uses the cross-entropy method for closed-loop planning of action sequences (Hafner et al., 2019). The *Dreamer* algorithm replaces the closed-loop planning of PlaNet by open-loop planning with entropy regularization (Hafner et al., 2020, 2021). Notably, neither PlaNet nor Dreamer explicitly take into account epistemic model uncertainty during planning but rather use point estimates.

13.3 Exploration

Exploration is critical when interacting with unknown environments, such as in reinforcement learning. We first encountered the exploration-exploitation dilemma in our discussion of Bayesian optimization, where we aimed at maximizing an unknown function in as little time as possible by making noisy observations.[8] Within the framework of Bayesian optimization, we used so-called acquisition functions for selecting the next point at which to observe the unknown function. Observe that these acquisition functions are analogous to policies in the setting of reinforcement learning. In particular, the policy that uses greedy exploitation like we have seen in the previous section is analogous to simply picking the point that maximizes the mean of the posterior distribution. In the context of Bayesian optimization, we have already seen that this is insufficient for exploration and can easily get stuck in locally optimal solutions. As reinforcement learning is a strict generalization of Bayesian optimization, there is no reason why such a strategy should be sufficient now.

[8] see Chapter 9

Recall from our discussion of Bayesian optimization that we "solved" this problem by using the epistemic uncertainty in our model of the unknown function to pick the next point to explore. This is what we will now explore in the context of reinforcement learning.

One simple strategy that we already investigated is the addition of some *exploration noise*. In other words, one follows a greedy exploitative strategy, but every once in a while, one chooses a random action

(like in ε-greedy); or one adds additional noise to the selected actions (known as Gaussian noise "dithering"). We have already seen that in difficult exploration tasks, these strategies are not systematic enough.

Two other approaches that we have considered before are Thompson sampling (cf. Section 9.3.3) and, more generally, the principle of *optimism in the face of uncertainty* (cf. Section 9.2.1).

13.3.1 Thompson Sampling

Recall from Section 9.3.3 the main idea behind Thompson sampling: namely, that the randomness in the realizations of f from the posterior distribution is already enough to drive exploration. That is, instead of picking the action that performs best on average across all realizations of f, Thompson sampling picks the action that performs best for a *single realization* of f. The epistemic uncertainty in the realizations of f leads to variance in the picked actions and provides an additional incentive for exploration. This yields Algorithm 13.11 which is an immediate adaptation of greedy exploitation and straightforward to implement.

Algorithm 13.11: Thompson sampling

start with (possibly empty) data $\mathcal{D} = \emptyset$ and a prior $p(f) = p(f \mid \mathcal{D})$
for several episodes **do**
  sample a model $f \sim p(\cdot \mid \mathcal{D})$
  plan a new policy π to (approximately) maximize,

$$\max_{\pi} \widehat{J}_H(\pi; f) \tag{13.21}$$

  roll out policy π to collect more data
  update posterior $p(f \mid \mathcal{D})$

13.3.2 Optimistic Exploration

We have already seen in the context of Bayesian optimization and tabular reinforcement learning that optimism is a central pillar for exploration. But how can we explore optimistically in model-based reinforcement learning?

Let us consider a set $\mathcal{M}(\mathcal{D})$ of *plausible models* given some data $\mathcal{D}$. Optimistic exploration would then optimize for the most advantageous model among all models that are plausible given the seen data.

Example 13.12: Plausible conditional Gaussians

In the context of conditional Gaussians, we can consider the set of all models such that the prediction of a single dimension i lies in some confidence interval,

$$f_i(\boldsymbol{x}, \boldsymbol{a}) \in \mathcal{C}_i \doteq [\mu_i(\boldsymbol{x}, \boldsymbol{a} \mid \mathcal{D}) - \beta_i \sigma_i(\boldsymbol{x}, \boldsymbol{a} \mid \mathcal{D}), \\ \mu_i(\boldsymbol{x}, \boldsymbol{a} \mid \mathcal{D}) + \beta_i \sigma_i(\boldsymbol{x}, \boldsymbol{a} \mid \mathcal{D})], \tag{13.22}$$

where β_i tunes the width of the confidence interval, analogously to Section 9.3.1. The set of plausible models is then given as

$$\mathcal{M}(\mathcal{D}) \doteq \{f \mid \forall \boldsymbol{x} \in \mathcal{X}, \boldsymbol{a} \in \mathcal{A}, i \in [d] : f_i(\boldsymbol{x}, \boldsymbol{a}) \in \mathcal{C}_i\}. \tag{13.23}$$

When compared to greedy exploitation, instead of taking the optimal step on average with respect to all realizations of the transition model f, optimistic exploration as shown in Algorithm 13.13 takes the optimal step with respect to the most optimistic model among all transition models that are consistent with the data.

Algorithm 13.13: Optimistic exploration

start with (possibly empty) data $\mathcal{D} = \emptyset$ and a prior $p(f) = p(f \mid \mathcal{D})$
for several episodes **do**
  plan a new policy π to (approximately) maximize,

$$\max_{\pi} \max_{f \in \mathcal{M}(\mathcal{D})} \widehat{J}_H(\pi; f) \tag{13.24}$$

  roll out policy π to collect more data
  update posterior $p(f \mid \mathcal{D})$

In general, the joint maximization over π and f is very difficult to solve computationally. Yet, remarkably, it turns out that this complex optimization problem can be reduced to standard model-based reinforcement learning with a fixed model. The key idea is to consider an agent that can control its "luck". In other words, we assume that the agent believes it can control the outcome of its actions — or rather choose which of the plausible dynamics it follows. The "luck" of the agent can be considered as additional decision variables. Consider the optimization problem,

$$\pi_{t+1} \doteq \arg\max_{\pi} \max_{\boldsymbol{\eta}(\cdot) \in [-1,1]^d} \widehat{J}_H(\pi; \widehat{f}_t) \tag{13.25}$$

with "optimistic" dynamics

$$\widehat{f}_{t,i}(\boldsymbol{x}, \boldsymbol{a}) \doteq \mu_{t,i}(\boldsymbol{x}, \boldsymbol{a}) + \beta_{t,i} \eta_i(\boldsymbol{x}, \boldsymbol{a}) \sigma_{t,i}(\boldsymbol{x}, \boldsymbol{a}). \tag{13.26}$$