Machine Learning and Neural Networks III (MATH3431)

Epiphany term

Georgios P. Karagiannis

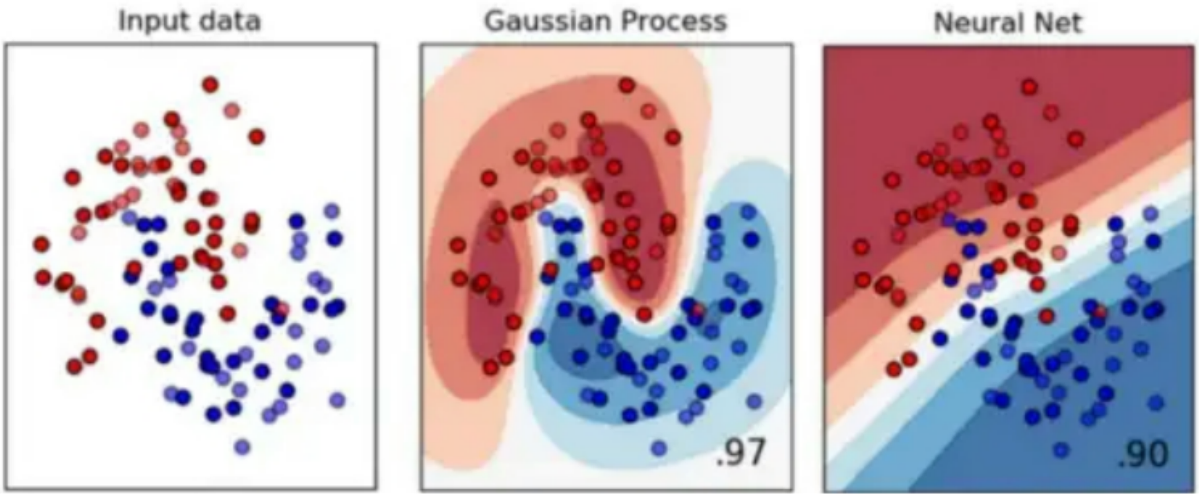
georgios.karagiannis@durham.ac.uk

Department of Mathematical Sciences (Office MCS3088)
Durham University
Stockton Road Durham DH1 3LE UK

2025/02/17 at 17:50:42

Concepts

- Convex learning problems
- Stochastic learning
- Support vector machines
- Artificial neural networks
- Kernel methods, trick
- Gaussian process regression



These lecture Handouts have been derived based on the reading list below.

Main texts:

- Bishop, C. M. (2006). Pattern recognition and machine learning. New York: Springer.
  - It is a classical textbook in machine learning (ML) methods. It discusses all the concepts introduced in the course (not necessarily in the same depth). It is one of the main textbooks in the module. The level on difficulty is easy.
  - Students who wish to have a textbook covering traditional concepts in machine learning are suggested to get a copy of this textbook. It is available online from the Microsoft's website `https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/`
- Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.
  - It has several elements of theory about machine learning algorithms. It is one of the main textbooks in the module. The level on difficulty is advanced as it requires moderate knowledge of maths.
- Bishop, C. M. (1995). Neural networks for pattern recognition. Oxford university press.
  - It is a classical textbook about 'traditional' artificial neural networks (ANN). It is very comprehensive (compared to others) and it goes deep enough for the module although it may be a bit outdated. It is one of the main textbooks in the module for ANN. The level on difficulty is moderate.

Supplementary textbooks:

- Vapnik, V. (1999). The nature of statistical learning theory. Springer science & business media.
  - Important textbook in the statistical machine learning theory. To have have an in deep understanding about statistical learning, one has to read it together with the textbook of Shalev-Shwartz, S., & Ben-David, S. (2014)
- Devroye, L., Györfi, L., & Lugosi, G. (2013). A probabilistic theory of pattern recognition (Vol. 31). Springer Science & Business Media.
  - Important textbook in the theoretical aspects about machine learning algorithms. The level on difficulty is advanced as it requires moderate knowledge of probability.
- Theodoridis, S. (2015). Machine learning: a Bayesian and optimization perspective. Academic press.
  - A textbook in general machine learning methods. It includes a large collection of different machine learning methods. It does not go too deep into the theoretical details of the method.
- Murphy, K. P. (2012). Machine learning: a probabilistic perspective. MIT press.
  - A popular textbook in general machine learning methods. It discusses all the concepts introduced in the module. It focuses more on the probabilistic/Bayesian framework but not with great detail. It can be used as a comparison textbook

for brief reading about ML methods just to see another perspective than that in (Bishop, C. M., 2006). The level on difficulty is easy.

- Murphy, K. P. (2022). Probabilistic machine learning: an introduction. MIT press.
  - A textbook in general machine learning methods. It covers a smaller number of ML concepts than (Murphy, K. P., 2012) but it contains more fancy/popular topics such as deep learning ideas. It is suggested to be used in the same manner as (Murphy, K. P., 2012). The level on difficulty is easy.
- Bishop, C. M., & Bishop, H. (2024). Deep learning: foundations and concepts. New York: Springer.
  - A textbook in machine learning methods focusing on Neural Networks philosophy and deep learning. Essentially, this is an update of the earlier book "Bishop, C. M. (1995). Neural networks for pattern recognition. Oxford university press." where: the concepts about Neural networks, stochastic gradient descent have been extended; new deep learning concepts have been added; and the kernel methods are left out.
- Ripley, B. D. (2007). Pattern recognition and neural networks. Cambridge university press.
  - A classical textbook in artificial neural networks (ANN) that also covers other machine learning concepts. It contains interesting theory about ANN.
  - It is suggested to be used as a supplementary reading for neural networks as it contains a few interesting theoretical results. The level on difficulty is moderate.
- Williams, C. K., & Rasmussen, C. E. (2006). Gaussian processes for machine learning (Vol. 2, No. 3, p. 4). Cambridge, MA: MIT press.
  - A classic book in Gaussian process regression (GPR) that covers the material we will discuss in the course about GPR. It can be used as a companion textbook with that of (Bishop, C. M., 2006). The level on difficulty is easy.

Preparatory textbooks:

- Strang, G. (2019). Linear algebra and learning from data. Wellesley-Cambridge Press.
  - Material regarding linear algebra concepts.
- Boyd, S. P., & Vandenberghe, L. (2004). Convex optimization. Cambridge university press.
  - Material regarding Convex optimization, quadratic optimization, Lagrange duality.

Contents

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2025

Lecture notes 1: Machine learning -A recap on: definitions, notation, and formalism

Lecturer & author: Georgios P. Karagiannis georgios.karagiannis@durham.ac.uk

Aim. To get some definitions and set-up about the learning procedure; essentially to formalize what introduced in term 1.

Reading list & references:

- Bishop, C. M. (2006). Pattern recognition and machine learning. New York: Springer.
  - Ch. 1 Introduction
- Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.
  - Ch. 1 Introduction

1. General Introductions and definitions

Pattern recognition is the automated discovery of patterns and regularities in data $z \in \mathcal{Z}$. **Machine learning (ML)** are statistical procedures for building and understanding probabilistic methods that 'learn'. **ML algorithms** $\mathfrak{A}$ build a (probabilistic/deterministic) model able to make predictions or decisions with minimum human interference and can be used for pattern recognition. **Learning** (or training, estimation, fitting) is called the procedure where the ML model is tuned. **Training data** (or observations, sample data set, examples) is a set of observables $\{z_i \in \mathcal{Z}\}$ used to tune the parameters of the ML model. By $\mathcal{Z}$ we denote the examples (or observables) domain. **Test set** is a set of available examples/observables $\{z_i'\}$ (different than the training data) used to verify the performance of the ML model for a given a measure of success. **Measure of success** (or performance) is a quantity that indicates how bad the corresponding ML model or Algorithm performs (eg quantifies the failure/error), and can also be used for comparisons among different ML models; eg, **Risk function** or **Empirical Risk Function**. Two main problems in ML are the supervised learning (we will focus on this here) and the unsupervised learning.

Supervised learning problems involve applications where the training data $z \in \mathcal{Z}$ comprise examples of the input vectors $x \in \mathcal{X}$ along with their corresponding target vectors $y \in \mathcal{Y}$; i.e. $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. By $\mathcal{X}$ we denote the inputs (or instances) domain, and by $\mathcal{Y}$ we denote the target domain. **Classification problems** are those which aim to assign each input vector x to one of a finite number of discrete categories of y. **Regression problems** are those where the output y consists of one or more continuous variables. All in all, the learner wishes to discover an unknown pattern (i.e. functional relationship) between components $x \in \mathcal{X}$ that serves as inputs and components $y \in \mathcal{Y}$ that act as outputs; i.e. $x \longmapsto y$. Hence, $\mathcal{X}$ is the input domain, and $\mathcal{Y}$ is the output (or target) domain. The goal of learning is to discover a function which predicts (or help us make decisions about) $y \in \mathcal{Y}$ from $x \in \mathcal{X}$.

Unsupervised learning problems involve applications where the training data $z \in \mathcal{Z}$ consist of a set of input vectors $x \in \mathcal{X}$ without any corresponding target values ; i.e. $\mathcal{Z} = \mathcal{X}$. In clustering the goal is to discover groups of similar examples within the data of it is to discover groups of similar examples within the data.

2. Notation & definitions in learning

Note 1. The aim is to learn the mapping $x \to y$ where $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ with purpose to predict $y \in \mathcal{Y}$ from $x \in \mathcal{X}$. Denote $z = (x, y)^\top$ and $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$.

Definition 2. The learner's output is a function, $h : \mathcal{X} \to \mathcal{Y}$ which predicts $y \in \mathcal{Y}$ from $x \in \mathcal{X}$. It is also called hypothesis, prediction rule, predictor, or classifier.

Notation 3. We often denote the set of hypothesis as $\mathcal{H}$; i.e. $h \in \mathcal{H}$.

Example 4. (Linear Regression)[1] Consider the regression problem where the goal is to learn the mapping $x \to y$ where $x \in \mathcal{X} \subseteq \mathbb{R}^d$ and $y \in \mathcal{Y} \subseteq \mathbb{R}$. A hypothesis is a linear function $h : \mathcal{X} \to \mathcal{Y}$ (that learner wishes to learn) with $h(x) = \langle w, x \rangle$ approximating the mapping $x \to y$. The hypothesis set is $\mathcal{H} = \left\{ x \to \langle w, x \rangle : w \in \mathbb{R}^d \right\}$.

Example 5. (Binary Classification) Consider the classification problem where the goal is to learn the mapping $x \to y$ where $x \in \mathcal{X} \subseteq \mathbb{R}^d$ and $y \in \mathcal{Y} = \{-1, +1\}$. A hypothesis can be a function $h : \mathcal{X} \to \mathcal{Y}$ with $h(x) = \text{sign}(\langle w, x \rangle)$ approximating the mapping $x \to y$. The hypothesis set $\mathcal{H} = \left\{ x \to \text{sign}(\langle w, x \rangle) : w \in \mathbb{R}^d \right\}$.

Definition 6. (Loss function) Given any set of hypothesis $\mathcal{H}$ and some domain $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, a loss function $\ell(\cdot)$ is any function $\ell : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}_+$. Loss function $\ell(h, z)$ for $h \in \mathcal{H}$ and $z \in \mathcal{Z}$ is specified according to the purpose the machine learning algorithm. It reflects how the "error" is quantified for a given hypothesis h and a given example z. The rule is "the greater the error the greater the value of the loss".

Example 7. (Cont. Example 4) In regression problems $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ and $\mathcal{Y} \subset \mathbb{R}$ is uncountable, a potential loss function is

$$\ell_{\text{sq}}(h, (x, y)) = (h(x) - y)^2$$

Example 8. (Cont. Example 5) In binary classification problems with hypothesis $h : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{Y} = \{0, 1\}$ is discrete, a loss function can be

$$\ell_{0-1}(h, (x, y)) = 1(h(x) \neq y),$$

Definition 9. A learning problem with hypothesis class $\mathcal{H}$, examples domain $\mathcal{Z}$, and loss function ℓ may be denoted with a triplet $(\mathcal{H}, \mathcal{Z}, \ell)$.

Example 10. The standard multiple linear regression problem with regressors $x \in \mathcal{X} \subseteq \mathbb{R}^d$ and response $y \in \mathcal{Y} \subseteq \mathbb{R}$, is a learning problem with examples domain $\mathcal{Z} = \mathcal{X} \times \mathcal{Y} = \mathbb{R}^{d+1}$, hypothesis class $\mathcal{H} = \left\{ x \to \langle w, x \rangle : w \in \mathbb{R}^d \right\}$, and loss function $\ell_{\text{sq}}(h, (x, y)) = (h(x) - y)^2$.

[1] $\langle w, x \rangle = w^\top x$

Example 11. The binary classification regression problem with regressors $x \in \mathcal{X} \subseteq \mathbb{R}^d$ and response $y \in \mathcal{Y} = \{-1, +1\}$, is a learning problem with examples domain $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, hypothesis class $\mathcal{H} = \left\{x \to \text{sign}\left(\langle w, x\rangle\right) : w \in \mathbb{R}^d\right\}$, and loss function $\ell_{0-1}\left(h, (x, y)\right) = 1\left(h(x) \neq y\right) = 1\left(\langle w, x\rangle y < 0\right)$.

Definition 12. Data generation model $g\left(\cdot\right)$ is the probability distribution over $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, unknown to the learner which describes the probabilistic law that realizations $z = (x, y) \in \mathcal{Z}$ are realized.

Conventions...

Note 13. If the Hypothesis set $\mathcal{H}$ is a known parametric family of functions; i.e. $\mathcal{H} = \{h_w\left(\cdot\right); w \in \mathcal{W}\}$ parameterized by unknown $w \in \mathcal{W}$, then we can equivalently consider the convention $\mathcal{H} = \{w \in \mathcal{W}\} = \mathcal{W}$ keeping in mind that the learner's output is restricted to formula $h_w\left(\cdot\right)$.

Example 14. Because it involves only linear functions as predictors $h_w\left(x\right) = \langle w, x\rangle$, we could consider a hypothesis class $\mathcal{H} = \left\{w \in \mathbb{R}^d\right\} = \mathbb{R}^d$ and loss function loss $\ell\left(w, (x, y)\right) = \left(\langle w, x\rangle - y\right)^2$ for simplicity.

Example 15. Because it involves only linear functions as predictors $h_w\left(x\right) = \text{sign}\left(\langle w, x\rangle\right)$, we could consider a hypothesis class $\mathcal{H} = \left\{w \in \mathbb{R}^d\right\} = \mathbb{R}^d$ and loss function loss $\ell\left(w, (x, y)\right) = 1\left(\langle x, w\rangle y < 0\right)$ for simplicity.

2.1. **Learning...**

Definition 16. (Risk function) The risk function $R_g\left(h\right)$ of h is the expected loss of the hypothesis $h \in \mathcal{H}$, w.r.t. the data generation model (which is a probability distribution) g over domain Z; i.e.

$$R_g\left(h\right) = \mathrm{E}_{z\sim g}\left(\ell\left(h, z\right)\right) \tag{2.1}$$

Definition 17. (Risk minimization learning) The Risk minimization (RM) learning paradigm computes the optimal predictor h^* as the minimizer of the risk $R_g\left(h\right)$ of h; i.e.

$$h^* = \arg\min_h \left(R_g\left(h\right)\right) \tag{2.2}$$

Example 18. (Cont. Ex. 7) The risk function is $R_g\left(h_w\right) = \mathrm{E}_{z\sim g}\left(h_w(x) - y\right)^2 = \mathrm{E}_{z\sim g}\left(\langle w, x\rangle - y\right)^2$, and it measures the quality of the hypothesis function $h : \mathcal{X} \to \mathcal{Y}$, (or equiv. the validity of the class of hypotheses $\mathcal{H}$) against the data generating model g, as the expected square difference between the predicted values form h and the true target values y at every x.

Example 19. (Cont. Ex.) The risk function is $R_g\left(h_w\right) = \mathrm{E}_{z\sim g}\left(1\left(h_w(x) \neq y\right)\right) = \mathrm{Pr}_{z\sim g}\left(\langle x, w\rangle y < 0\right)$.

Note 20. Computing the risk minimizer may be practically challenging due to the integration w.r.t. the unknown data generation model g involved in the expectation (2.1). Sub-optimally, instead of the Risk function in (2.2), one use the Empirical risk function as a Monte Carlo approximation of it.

Definition 21. Training data set $\mathcal{S}$ of size m is any finite sequence of pairs $\left(z_i = (x_i, y_i); i = 1, ..., m\right)$, called examples, in $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$; i.e. $\mathcal{S} = \{(x_i, y_i); i = 1, ..., m\}$. This is the information that the learner has assess.

Definition 22. (Empirical risk function) The Empirical Risk Function (ERF) $\hat{R}_S(h)$ of h is the expectation of loss of h over a given sample $S=(z_1,...,z_m)\in\mathcal{Z}^m$; i.e.

$$\hat{R}_S(h)=\frac{1}{m}\sum_{i=1}^{m}\ell(h,z_i).$$

Definition 23. (Empirical risk minimization learning) The Empirical risk minimization (ERM) learning paradigm computes the optimal predictor h^* as the minimizer of the ERF $\hat{R}_S(h)$ of h; i.e.

$$h^*=\arg\min_h\left(\hat{R}_S(h)\right) \tag{2.3}$$

Example 24. (Cont. Example 18) Given given sample $S=\{(x_i,y_i);i=1,...,m\}$ the empirical risk function is $\hat{R}_S(h)=\frac{1}{m}\sum_{i=1}^{m}(h(x_i)-y_i)^2=\frac{1}{m}\sum_{i=1}^{m}\left(x_i^\top w-y_i\right)^2$.

Example 25. (Cont. Example 8) Given given sample $S=\{(x_i,y_i);i=1,...,m\}$ the empirical risk function is $\hat{R}_S(h)=\frac{1}{m}\sum_{i=1}^{m}1(h(x_i)\neq y_i)=\frac{1}{m}\sum_{i=1}^{m}1(\langle x_i,w\rangle y_i<0)$.

Definition 26. We denote as $\mathfrak{A}(\mathcal{S})$ the hypothesis (outcome) that a learning algorithm $\mathfrak{A}$ returns given training sample S.

3. Regularized loss minimization (RLM) learning

Note 27. Consider a learning problem $(\mathcal{H},\mathcal{Z},\ell)$ with training data set $\mathcal{S}$.

Definition 28. (Regularized loss minimization learning) Regularized loss minimization learning paradigm computes the optimal rule $h^*\in\mathcal{H}$ by jointly minimizing the Risk function $R_g(h)$ (or Empirial Risk Function $\hat{R}_{\mathcal{S}}(h)$) and a regularization function $J:\mathcal{H}\to\mathbb{R}$ where $J(h)$ increases in value with the complexity of the hypothesis $h\in\mathcal{H}$. Formally, the Regularized loss minimization rule h^* is

$$h^*=\arg\min_{h\in\mathcal{H}}\left(R_g(h)+J(h)\right),\ \text{given the Risk function}$$
$$h^*=\arg\min_{h\in\mathcal{H}}\left(\hat{R}_{\mathcal{S}}(w)+J(h)\right),\ \text{given the Empirical risk function}$$

3.1. ℓ_0, ℓ_1,and ℓ_2 norm regularization problems.

Note 29. Consider a parameterized hypothesis $h_w(\cdot)$ (e.g. $h_w(x)=\eta(\langle x,w\rangle)$) with $\eta(\cdot)$ a function and $w\in\mathbb{R}^d$. Consider the hypothesis class in the simplified form $\mathcal{H}=\mathcal{W}=\left\{w\in\mathbb{R}^d\right\}=\mathbb{R}^d$ of Note 13. Assume we want the vector w to be sparse, in the sense that "sparser" means more zero elements.

3.1.1. ℓ_0 *norm regularization.*

Note 30. The problem of minimizing the empirical risk subject to a budget of k features can be written as

$$\underset{w\in\mathcal{H}}{\text{minimize}}R(w)$$
$$\text{subject to}\,\|w\|_0\leq k_0$$

where $\|w\|_0 = \#\{j : w_j \neq 0\}$. By Lagrange multiplies we can re-write it as

$$w^* = \arg\min_{w \in \mathcal{H}} \left(R(w) + \lambda_0 \|w\|_0\right)$$

for some $\lambda_0 \geq 0$ (k_0 dependent). Hence, $h^*(x) = h_{w^*}(x)$.

Example 31. (Cont. Example 14) The ℓ_0-RLM rule in Example 14 becomes

$$w^* = \arg\min_{w} \left(\mathrm{E}_{z\sim g}\left(\langle w, x_i\rangle - y\right)^2 + \lambda_0 \|w\|_0\right)$$

using the Risk function, and

$$w^* = \arg\min_{w} \left(\frac{1}{m}\sum_{i=1}^{m}\left(\langle w, x_i\rangle - y_i\right)^2 + \lambda_0 \|w\|_0\right)$$

using the Empirical risk function.

3.1.2. *ℓ_1 norm regularization (LASSO).*

Note 32. Solving ℓ_0 norm optimization problem is computationally hard. To simplify computations, we can replace $J(\cdot) = \|\cdot\|_0$ with $J(\cdot) = \|\cdot\|_1$ i.e.

$$\begin{aligned} &\underset{w\in\mathcal{H}}{\text{minimize}} R(w) \\ &\text{subject to } \|w\|_1 \leq k_1 \end{aligned}$$

where $\|w\|_1 = \sum_j |w_j|$. By Lagrange multiplies we can re-write it as

$$w^* = \arg\min_{w} \left(R(w) + \lambda_1 \|w\|_1\right)$$

for some $\lambda_1 \geq 0$ (k_1 dependent). Hence, $h^*(x) = h_{w^*}(x)$.

Example 33. (Cont. Example 14) The ℓ_1-RLM rule in Example 14 becomes

$$w^* = \arg\min_{w} \left(\mathrm{E}_{z\sim g}\left(\langle w, x\rangle - y\right)^2 + \lambda_1 \sum_{j=1}^{d} |w_j|\right)$$

using the Risk function , and

$$w^* = \arg\min_{w} \left(\frac{1}{m}\sum_{i=1}^{m}\left(\langle w, x_i\rangle - y_i\right)^2 + \lambda_1 \sum_{j=1}^{d} |w_j|\right)$$

using the Empirical risk function.

3.1.3. *ℓ_2 norm regularization (Ridge).*

Note 34. To further simplify computations, we can consider $J(\cdot) = \|\cdot\|_2$ i.e.

$$\begin{aligned} &\underset{w}{\text{minimize}} R(w) \\ &\text{subject to } \|w\|_2^2 \leq k_2 \end{aligned}$$

where $\|w\|_2^2 = \sum_j w_j^2$. By Lagrange multiplies we can re-write it as

$$w^* = \arg\min_w \left(R(w) + \lambda_2 \|w\|_2^2 \right)$$

for some $\lambda_1 \geq 0$ (k_1 dependent). Hence, $h^*(x) = h_{w^*}(x)$.

Example 35. (Cont. Example 14) The ℓ_2-RLM rule in Example 14 becomes

$$w^* = \arg\min_w \left(\mathrm{E}_{z\sim g} \left(\langle w, x \rangle - y \right)^2 + \lambda_2 \sum_{j=1}^{d} w_j^2 \right)$$

using the Risk function, and

$$w^* = \arg\min_w \left(\frac{1}{m} \sum_{i=1}^{m} (h(x_i) - y_i)^2 + \lambda_2 \sum_{j=1}^{d} w_j^2 \right)$$

using the Empirical risk function.

Example 36. (Cont. Example 15) The ℓ_2-RLM rule in Example 15 becomes

$$w^* = \arg\min_w \left(\Pr_{z\sim g} \left(\langle x, w \rangle y < 0 \right) + \lambda_2 \sum_{j=1}^{d} w_j^2 \right)$$

using the Risk function, and

$$w^* = \arg\min_w \left(\frac{1}{m} \sum_{i=1}^{m} 1 \left(\langle x_i, w \rangle y_i < 0 \right) + \lambda_2 \sum_{j=1}^{d} w_j^2 \right)$$

using the Empirical risk function.

Example 37. Consider a learning problem where the true data generation distribution (unknown to the learner) is $g(z)$, the statistical model (known to the learner) is given by a sampling distribution $f_\theta(y) := f(y|\theta)$ labeled by an unknown parameter θ. The goal is to learn θ. If we assume loss function

$$\ell(\theta, z) = \log\left(\frac{g(z)}{f_\theta(z)}\right)$$

then the risk is

$$R_g(\theta) = \mathrm{E}_{z\sim g}\left(\log\left(\frac{g(z)}{f_\theta(z)}\right)\right) = \mathrm{E}_{z\sim g}(\log(g(z))) - \mathrm{E}_{z\sim g}(\log(f_\theta(z))) \tag{A.1}$$

whose minimizer is

$$\theta^* = \arg\min_{\forall\theta}(R_g(\theta)) = \arg\min_{\forall\theta}(\mathrm{E}_{z\sim g}(-\log(f_\theta(z))))$$

as the first term in (A.1) is constant. Note that in the Maximum Likelihood Estimation technique the MLE θ_{MLE} is the minimizer

$$\theta_{\mathrm{MLE}} = \arg\min_{\theta}\left(\frac{1}{m}\sum_{i=1}^{m}(-\log(f_\theta(z_i)))\right)$$

where $S = \{z_1, ..., z_m\}$ is an IID sample from g. Hence, MLE θ_{MLE} can be considered as the minimizer of the empirical risk $R_S(\theta) = \frac{1}{m}\sum_{i=1}^{m}(-\log(f_\theta(z_i)))$.

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2025

Lecture notes 2: Elements of convex learning problems

Lecturer & author: Georgios P. Karagiannis georgios.karagiannis@durham.ac.uk

Aim. To introduce convex learning problems that can be used as a framework for the analysis of stochastic gradient related learning algorithms.

Reading list & references:

- Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.
  - Ch. 12 Convex Learning Problems

Further reading

- Bishop, C. M. (2006). Pattern recognition and machine learning. New York: Springer.

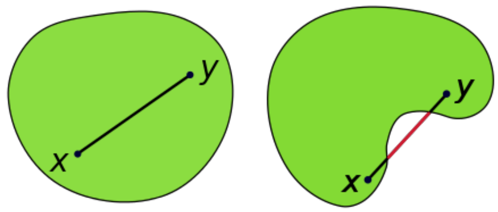
1. Motivations

Note 1. We introduce learning problems associated with convexity, Lipschitzness and smoothness to be able to analyze and understand the machine learning (ML) tools we will discuss later on (eg stochastic gradient descent, SVM). We discuss ways allowing to address non-convex ML problems (eg, Artificial neural networks, Gaussian process regression) in the convex setting.

2. Convexity

Note 2. Convexity is a central concept in learning, e.g. least squares, as it often considers a Euclidean distance as a Risk function to be minimized.

Definition 3. A set C is convex if for any $u, v \in C$ and for any $\alpha \in [0, 1]$ we have that $\alpha u + (1 - \alpha) v \in C$.



(A) (B)

FIGURE 2.1. (2.1a) is a Convex set ; (2.1b) is a non-convex set

Example 4. For instance $\mathbb{R}^d$ for $d \geq 1$ is a convex set.

Definition 5. Let C be a convex set. A function $f : C \to R$ is convex function if for any $u, v \in C$ and for any $\alpha \in [0, 1]$

$$f(\alpha u + (1-\alpha) v) \leq \alpha f(u) + (1-\alpha) f(v)$$

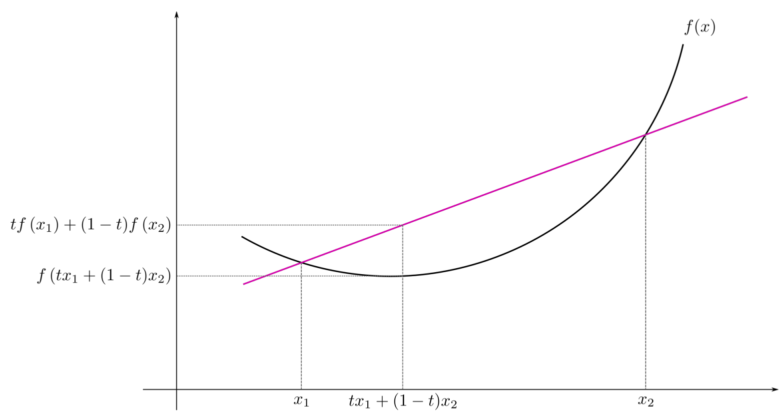


FIGURE 2.2. A convex function

Example 6. The function $f : \mathbb{R}^d \to \mathbb{R}_+$ with $f(x) = \|x\|_2^2$ is convex function. For any $u, v \in C$ and for any $\alpha \in [0, 1]$ it is

$$\begin{aligned}
&\|\alpha u + (1-\alpha) v\|_2^2 - \alpha \|u\|_2^2 - (1-\alpha) \|v\|_2^2 \\
&= (\alpha u + (1-\alpha) v)^\top (\alpha u + (1-\alpha) v) - \alpha \|u\|_2^2 - (1-\alpha) \|v\|_2^2 \\
&= ... = -\alpha (1-\alpha) \|u - v\|_2^2 \leq 0
\end{aligned}$$

Note 7. Every local minimum of a convex function is the global minimum.

Note 8. Let $f : C \to \mathbb{R}$ be convex function. The tangent of $f$at $w \in C$ is below f, namely

$$\forall u \in C \quad f(u) \geq f(w) + \langle \nabla f(w), u - w \rangle$$

Proposition 9. *Let $f : \mathbb{R}^d \to \mathbb{R}$ such that $f(w) = g(\langle w, x\rangle + y)$ for some $x \in \mathbb{R}^d$, $y \in \mathbb{R}$. If g is convex function then f is convex function.*

Proof. See Exercise 1 in the Exercise sheet. □

Example 10. Consider the regression problem with regressor $x \in \mathbb{R}^d$, and response $y \in \mathbb{R}$ and predictor rule $h(x) = \langle w, x\rangle$. The loss function $\ell(w,(x,y)) = (\langle w, x\rangle - y)^2$ is convex because $g(a) = (a)^2$ is convex and Proposition 9.

Note 11. Let $f_j : \mathbb{R}^d \to \mathbb{R}$ convex functions for $j = 1, ..., r$. Then:

(1) $g(x) = \max_{\forall j}(f_j(x))$ is a convex function
(2) $g(x) = \sum_{j=1}^{r} w_j f_j(x)$ is a convex function where $w_j > 0$

Proof.

(1) For any $u, v \in \mathbb{R}^d$ and for any $\alpha \in [0,1]$

$$\begin{aligned} g(\alpha u + (1-\alpha) v) &= \max_{\forall j}(f_j(\alpha u + (1-\alpha) v)) \\ &\leq \max_{\forall j}(\alpha f_j(u) + (1-\alpha) f_j(v)) && (f_j \text{ is convex}) \\ &\leq \alpha \max_{\forall j}(f_j(u)) + (1-\alpha)\max_{\forall j}(f_j(v)) && (\max(\cdot) \text{ is convex}) \\ &\leq \alpha g(u) + (1-\alpha) g(v) \end{aligned}$$

(2) For any $u, v \in \mathbb{R}^d$ and for any $\alpha \in [0,1]$

$$\begin{aligned} g(\alpha u + (1-\alpha) v) &= \sum_{j=1}^{r} w_j f_j(\alpha u + (1-\alpha) v) \\ &\leq \alpha \sum_{j=1}^{r} w_j f_j(u) + (1-\alpha) \sum_{j=1}^{r} w_j f_j(v) && (f_j \text{ is convex}) \\ &\leq \alpha g(u) + (1-\alpha) g(v) \end{aligned}$$

□

Example 12. $g(x) = \|x\|_1$ is convex according to Note 11, since $|x_j| = \max(-x_j, x_j)$ and $\|x\|_1 = \sum_j |x_j|$.

3. Strong convexity

Note 13. Strong convexity is a central concept in Ridge regularization, as it makes a convex loss function strongly convex by adding a shrinkage term.

Definition 14. (Strongly convex functions) A function f is λ-strongly convex function is for all w, u, and $\alpha \in (0,1)$ we have

$$f(\alpha w + (1-\alpha) u) \leq \alpha f(w) + (1-\alpha) f(u) - \frac{\lambda}{2}\alpha(1-\alpha)\|w-u\|^2 \tag{3.1}$$

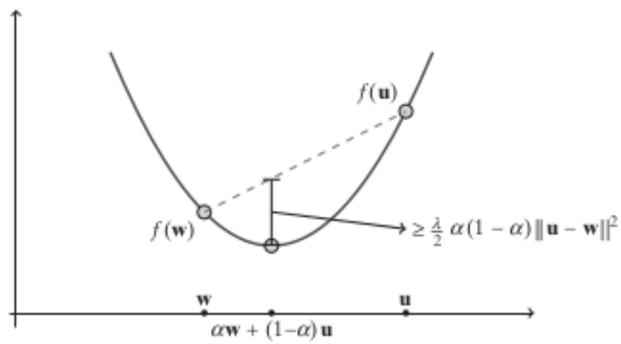


FIGURE 3.1. Strongly convex function

Note 15. The following can be checked from the definition by substitution

(1) The function $f(w) = \lambda\|w\|^2$ is 2λ-strongly convex
(2) If $f$is λ-strongly convex and g is convex then $f+g$ is λ-strongly convex

4. Lipschitzness

Definition 16. Let $C \subseteq \mathbb{R}^d$. Function $f : C \to \mathbb{R}^k$ is ρ-Lipschitz over C if for every $w_1, w_2 \in C$ we have that

$$\|f(w_1) - f(w_2)\| \leq \rho\|w_1 - w_2\|. \qquad \text{Lipschitz condition} \tag{4.1}$$

Note 17. That means: a Lipschitz function $f(x)$ cannot change too drastically wrt x.

Example 18. Consider the function $f : \mathbb{R} \to \mathbb{R}_+$ with $f(x) = x^2$.

(1) f is not a ρ-Lipschitz in $\mathbb{R}$.
(2) f is a ρ-Lipschitz in $C = \{x \in \mathbb{R} : |x| < \rho/2\}$.

Solution.

(1) For $x_1 = 0$ and $x_2 = 1+\rho$, it is

$$|f(x_2) - f(x_1)| = (1+\rho)^2 > \rho(1+\rho) = \rho|x_2 - x_1|$$

(2) It is

$$|f(x_2) - f(x_1)| = |x_2^2 - x_1^2| = |(x_2+x_1)(x_2-x_1)| \leq 2\rho/2(x_2-x_1) = \rho|x_2-x_1|$$

Note 19. Let functions g_1 be ρ_1-Lipschitz and g_2 be ρ_2-Lipschitz. Then f with $f(x) = g_1(g_2(x))$ is $\rho_1\rho_2$-Lipschitz.

[See Exercise 2 from the exercise sheet]

Example 20. Let functions g be ρ-Lipschitz. Then f with $f(x) = g(< v, x > +b)$ is $(\rho|v|)$-Lipschitz.

Solution. It is

$$\begin{aligned}|f(w_1) - f(w_2)| =& |g(< v, w_1 > +b) - g(< v, w_2 > +b)| \leq \rho |< v, w_1 > +b - < v, w_2 > -b| \\ \leq& \rho \left|v^\top w_1 - v^\top w_2\right| \leq \rho |v| |w_1 - w_2|\end{aligned}$$

Note 21. So, given Examples 18 and 20, in the linear regression setting using loss $\ell(w, z = (x, y)) = \left(w^\top x - y\right)^2$, the loss function is -Lipschitz for a given $z = (x, y)$ and and bounded $\|w\| < \rho$.

5. Smoothness

Definition 22. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is β-smooth if its gradient is β-Lipschitz; namely for all $v, w \in \mathbb{R}^d$

$$\|\nabla f(w_1) - \nabla f(w_2)\| \leq \beta \|w_1 - w_2\| . \tag{5.1}$$

Note 23. Function $f : \mathbb{R}^d \to \mathbb{R}$ is β-smooth iff

$$f(v) \leq f(w) + \langle \nabla f(w), v - w\rangle + \frac{\beta}{2}\|v - w\|^2 \tag{5.2}$$

Note 24. Let $f : \mathbb{R}^d \to \mathbb{R}$ with $f(w) = g(\langle w, x\rangle + y)$ $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$. Let $g : \mathbb{R} \to \mathbb{R}$ be a β-smooth function. Then f is a $\left(\beta\|x\|^2\right)$-smooth.

[See Exercise 3 from the Exercise sheet]

Example 25. Let $f(w) = (< w, x > +y)^2$ for $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$. Then f is $\left(2\|x\|^2\right)$-smooth.

To show that we set $f(w) = g(< w, x > +y)$ where $g(a) = a^2$. g is 2-smooth since

$$\|g'(w_1) - g'(w_2)\| = \|2w_1 - 2w_2\| \leq 2\|w_1 - w_2\| .$$

Hence from Theorem 24, f is $\left(2\|x\|^2\right)$-smooth.

Example 26. Consider the regression problem with predictor rule $h(x) = \langle w, x\rangle$, loss function $\ell(w, (x, y)) = (\langle w, x\rangle - y)^2$, feature $x \in \mathbb{R}^d$, and target $y \in \mathbb{R}$. Then $\ell(w, \cdot)$ is $\left(2\|x\|^2\right)$-smooth.

[Follows from Example 25.]

Definition 27. Convex learning problem is a learning problem $(\mathcal{H}, \mathcal{Z}, \ell)$ that the hypothesis class $\mathcal{H}$ is a convex set, and the loss function $\ell(\cdot, z)$ is a convex function for each example $z \in \mathcal{Z}$.

Example 28. Consider the regression problem with predictor rule $h(x) = \langle w, x \rangle$, loss function $\ell(w, (x, y)) = (\langle w, x \rangle - y)^2$, feature $x \in \mathbb{R}^d$, and target $y \in \mathbb{R}$. This imposes a convex learning problem due to Examples 4 and 11.

Definition 29. Convex-Lipschitz-Bounded Learning Problem $(\mathcal{H}, \mathcal{Z}, \ell)$ with parameters ρ, and B, is called the learning problem whose the hypothesis class $\mathcal{H}$ is a convex set, for all $w \in \mathcal{H}$ it is $\|w\| \le B$, and the loss function $\ell(\cdot, z)$ is convex and ρ-Lipschitz function for all $z \in \mathcal{Z}$.

Example 30. Consider the regression problem with predictor rule $h(x) = \langle w, x \rangle$, loss function $\ell(w, (x, y)) = (\langle w, x \rangle - y)^2$, feature $x \in \mathbb{R}^d$, and target $y \in \mathbb{R}$. This imposes a Convex-Lipschitz-Bounded Learning Problem if $\mathcal{H} = \left\{w \in \mathbb{R}^d : \|w\| \le B\right\}$ due to Examples 11, and 18(2).

Definition 31. Convex-Smooth-Bounded Learning Problem $(\mathcal{H}, \mathcal{Z}, \ell)$ with parameters β, and B, is called the learning problem whose the hypothesis class $\mathcal{H}$ is a convex set, for all $w \in \mathcal{H}$ it is $\|w\| \le B$, and the loss function $\ell(\cdot, z)$ is convex, nonnegative, and β-smooth function for all $z \in \mathcal{Z}$.

Example 32. Consider the regression problem with predictor rule $h(x) = \langle w, x \rangle$, loss function $\ell(w, (x, y)) = (\langle w, x \rangle - y)^2$, feature $x \in \mathbb{R}^d$, and target $y \in \mathbb{R}$. This imposes a Convex-Smooth-Bounded Learning Problem if $\mathcal{H} = \left\{w \in \mathbb{R}^d : \|w\| \le B\right\}$ due to Examples 11, and 26.

Note 33. (Empirical risk minimization learning problem) If ℓ is a convex loss function and the class $\mathcal{H}$ is convex, then the $\text{ERM}_{\mathcal{H}}$ problem, of minimizing the empirical risk $\hat{R}_{\mathcal{S}}(w)$ over $\mathcal{H}$, is a convex optimization problem

Proof. The Empirical risk minimization (ERM) problem $(\mathcal{H}, \mathcal{Z}, \ell)$ is

$$w^* = \underset{w \in \mathcal{H}}{\arg\min} \left\{\hat{R}_{\mathcal{S}}(w)\right\}$$

given a sample $\mathcal{S} = \{z_1, ..., z_m\}$ for $\hat{R}_{\mathcal{S}}(w) = \frac{1}{m} \sum_{i=1}^{m} \ell(w, z_i)$. $\hat{R}_{\mathcal{S}}(w)$ is a convex function from Note (11). Hence ERM rule is a problem of minimizing a convex function subject to the constraint that the solution should be in a convex set. $\square$

Example 34. Multiple linear regression with predictor rule $h(x) = \langle w, x\rangle$, loss function $\ell(w,(x,y)) = (\langle w, x\rangle - y)^2$, feature $x \in \mathbb{R}^d$, and target $y \in \mathbb{R}$ where

$$w^* = \arg\min_w \mathrm{E}\left(\langle w, x\rangle - y\right)^2$$

or

$$w^{**} = \arg\min_w \frac{1}{m}\sum_{i=1}^{m}\left(\langle w, x_i\rangle - y_i\right)^2$$

is a convex learning problem –from Note 33.

Note 35. (Ridge / ℓ_2-RLM problem) If ℓ is a convex loss function w.r.t. w, the class $\mathcal{H}$ is convex, and $J(\cdot;\lambda) = \lambda\|\cdot\|_2^2$ with $\lambda > 0$ then $\hat{R}_{\mathcal{S}}(w) + \lambda\|w\|_2^2$ is a 2λ-strongly convex function, and hence the ℓ_2-RLM problem

$$w^* = \arg\min_{w\in\mathcal{H}}\left\{\hat{R}_{\mathcal{S}}(w) + \lambda\|w\|_2^2\right\}$$

is a strongly convex optimization problem (i.e. the learning rule is the minimizer of a strongly convex function over a convex set).

Proof. $\hat{R}_{\mathcal{S}}(\cdot)$ is a convex function from Note 33, $\lambda\|\cdot\|_2^2$ is 2λ-strongly convex, hence $\hat{R}_{\mathcal{S}}(w) + \lambda\|w\|_2^2$ is a 2λ-strongly convex function. Hence the above Ridge RLM problem is a strongly convex optimization problem. □

7. Non-convex learning problems (surrogate treatment)

Note 36. A learning problem may involve non-convex loss function $\ell(w,z)$ w.r.t. w which implies a non-convex risk function $R_g(w)$. A suitable treatment to analyze the associated learning tools within the convex setting would be to upper bound the non-convex loss function $\ell(w,z)$ by a convex surrogate loss function $\tilde{\ell}(w,z)$ for all w, and use $\tilde{\ell}(w,z)$ instead of $\ell(w,z)$.

Example 37. Consider the binary classification problem with inputs $x \in \mathcal{X}$, outputs $y \in \{-1,+1\}$; we need to learn $w \in \mathcal{H}$ from hypothesis class $\mathcal{H} \subset \mathbb{R}^d$ with respect to the loss

$$\ell(w,(x,y)) = 1_{(y\langle w,x\rangle \le 0)}$$

with $y \in \mathbb{R}$, and $x \in \mathbb{R}^d$. Here $\ell(\cdot)$ is non-convex. A convex surrogate loss function can be

$$\tilde{\ell}(w,(x,y)) = \max\left(0, 1 - y\langle w, x\rangle\right)$$

which is convex (Example 11) wrt w. Note that:

- $\tilde{\ell}(w,(x,y))$ is convex wrt w ; because $\max(\cdot)$ is convex
- $\ell(w,(x,y)) \le \tilde{\ell}(w,(x,y))$ for all $w \in \mathcal{H}$

Then we can compute

$$\tilde{w}_* = \arg\min_{\forall x} \left(\tilde{R}_g\left(w\right) \right) = \arg\min_{\forall x} \left(\mathrm{E}_{(x,y)\sim g}\left(\max\left(0, 1 - y\langle w, x\rangle\right)\right)\right)$$

instead of

$$w_* = \arg\min_{\forall x} \left(R_g\left(w\right)\right) = \arg\min_{\forall x} \left(\mathrm{E}_{(x,y)\sim g}\left(1_{(y\langle w,x\rangle \leq 0)}\right)\right)$$

Of course by using the surrogate loss instead of the actual one, we introduce some approximation error in the produced output $\tilde{w}_* \neq w_*$.

Note 38. (Intuitions...) Using a convex surrogate loss function instead the convex one, facilitates computations but introduces extra error to the solution. If $R_g(\cdot)$ is the risk under the non-convex loss, $\tilde{R}_g(\cdot)$ is the risk under the convex surrogate loss, and $\tilde{w}_{\mathrm{alg}}$ is the output of the learning algorithm under $\tilde{R}_g(\cdot)$ then we have the upper bound

$$R_g(\tilde{w}_{\mathrm{alg}}) \leq \underbrace{\min_{w\in\mathcal{H}}\left(R_g(w)\right)}_{\mathrm{I}} + \underbrace{\left(\min_{w\in\mathcal{H}}\left(\tilde{R}_g(w)\right) - \min_{w\in\mathcal{H}}\left(R_g(w)\right)\right)}_{\mathrm{II}} + \underbrace{\epsilon}_{\mathrm{III}}$$

where term I is the approximation error measuring how well the hypothesis class performs on the generating model, term II is the optimization error due to the use of surrogate loss instead of the actual non-convex one, and term III is the estimation error due to the use of a training set and not the whole generation model.

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2025

Lecture notes 3: Learnability and stability in learning problems

Lecturer & author: Georgios P. Karagiannis georgios.karagiannis@durham.ac.uk

---

Aim. To introduce concepts PAC, fitting vs stability trade off, stability, and their implementation in regularization problems and convex problems.

Reading list & references:

- Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.
  - Ch. 2, 3, 13
- Vapnik, V. (2000). The nature of statistical learning theory. Springer science & business media.

1. Learnable Probably Approximately Correct (PAC) Learning

Note 1. We formally define the broad learning problem we will work on.

Note 2. Learning algorithms $\mathfrak{A}$ use training data sets $\mathcal{S}$ which may miss characteristics of the unknown data-generating process g. Essentially, approximations (aka errors) are inevitable; some characteristics of data generating process g will be missed even if we use a very representative training set $\mathcal{S}$. We cannot hope that the learning algorithm will find a hypothesis whose error is smaller than the minimal possible error.

Note 3. The PAC learning problem requires no prior assumptions about the data-generating process g. It requires that the learning algorithm $\mathfrak{A}$ will find a predictor $\mathfrak{A}(\mathcal{S})$ whose error is not much larger than the best possible error of a predictor in some given benchmark hypothesis class. So essentially, in practice, the researcher's effort falls on the hypothesis class $\mathcal{H}$.

Definition 4. (Agnostic PAC Learnability for General Loss Functions) A hypothesis class $\mathcal{H}$ is agnostic PAC learnable with respect to a domain $\mathcal{Z}$ and a loss function $\ell : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}_+$, if there exist a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm $\mathfrak{A}$ with the following property:

- for every $\epsilon \in (0,1)$, $\delta \in (0,1)$, and distribution g over $\mathcal{Z}$, when running algorithm $\mathfrak{A}$ given training set $\mathcal{S}_m = \{z_1, ..., z_m\}$ with $z_i \overset{\text{IID}}{\sim} g$ for $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ then $\mathfrak{A}$ returns

$\mathfrak{A}(\mathcal{S}) \in \mathcal{H}$ such as

$$\Pr_{\mathcal{S}\sim g}\left(R_g\left(\mathfrak{A}\left(\mathcal{S}_m\right)\right) - \min_{h'\in\mathcal{H}}\left(R_g\left(h'\right)\right) \leq \epsilon\right) \geq 1-\delta \tag{1.1}$$

Note 5. About (1.1): The accuracy parameter ϵ determines how far the output rule $\mathfrak{A}(\mathcal{S}_m)$ of $\mathfrak{A}$ can be from the optimal rule (this corresponds to the 'approximately correct' part of "PAC"). The confidence parameter δ indicates how likely the classifier is to meet that accuracy requirement (corresponds to the "probably" part of "PAC").

Note 6. It may be easier to work with expectations by using Markov inequality in (1.1).

Proposition 7. *Let $\mathfrak{A}$ be a learning algorithm that guarantees the following:*

- *If $m \geq m_{\mathcal{H}}(\epsilon)$ then for every distribution g, it is*

$$E_{\mathcal{S}\sim g}\left(R_g\left(\mathfrak{A}\left(\mathcal{S}_m\right)\right)\right) \leq \min_{h'\in\mathcal{H}}\left(R_g\left(h'\right)\right) + \epsilon$$

Then $\mathfrak{A}$ satisfies the PAC guarantee in Definition 4:

- *for every $\delta \in (0,1)$, if $m \geq m_{\mathcal{H}}(\epsilon\delta)$* then

$$\Pr_{\mathcal{S}\sim g}\left(R_g\left(\mathfrak{A}\left(\mathcal{S}_m\right)\right) - \min_{h'\in\mathcal{H}}\left(R_g\left(h'\right)\right) \leq \epsilon\right) \geq 1-\delta$$

Proof. Let $\xi = R_g\left(\mathfrak{A}\left(\mathcal{S}_m\right)\right) - \min_{h'\in\mathcal{H}}\left(R_g\left(h'\right)\right)$. From Markov's inequality $\Pr\left(\xi \geq \mathrm{E}\left(\xi\right)/\delta\right) \leq \frac{1}{\mathrm{E}(\xi)/\delta}\mathrm{E}\left(\xi\right) = \delta$. Namely, $\Pr\left(\xi \leq \mathrm{E}\left(\xi\right)/\delta\right) = 1-\delta$. But it is given that if $m \geq m_{\mathcal{H}}\left(\left(\epsilon\delta\right)\right)$ for every distribution g it is $\mathrm{E}\left(\xi\right) \leq \left(\epsilon\delta\right)$. So by substitution $\Pr\left(\xi \leq \left(\epsilon\delta\right)/\delta\right) \geq 1-\delta$ implies $\Pr\left(\xi \leq \epsilon\right) \geq 1-\delta$. Now substitute back ξ and we conclude the proof. □

2. Analysis of the risk based on the trade-off fitting vs stability

Note 8. Let $R^* = \min_{\forall h}\left(R\left(h\right)\right)$ be an ideal/optimal (hence minimum) Risk, and $\mathfrak{A}(\mathcal{S})$ the learning rule from a learning algorithm $\mathfrak{A}$ trained against dataset $\mathcal{S}$. The Risk of a learning algorithm $\mathfrak{A}$ can be decomposed as

$$R_g\left(\mathfrak{A}\left(\mathcal{S}\right)\right) - R^* = \hat{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}\right)\right) - R^* + \underbrace{R_g\left(\mathfrak{A}\left(\mathcal{S}\right)\right) - \hat{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}\right)\right)}_{\text{over-fitting}} \tag{2.1}$$

Note 9. Over-fitting can be represented by $R_g\left(\mathfrak{A}\left(\mathcal{S}\right)\right) - \hat{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}\right)\right)$. However, $\hat{R}_{\mathcal{S}}\left(\cdot\right)$ is a random variable and, here, for our computational convenience, we focus on its expectation w.r.t. $\mathcal{S} \sim g$. Hence, we provide the following (arguable) definition for over-fitting on which we base our analysis.

Definition 10. For a learning algorithm $\mathfrak{A}$, as a measure of over-fitting we consider the expected difference between true Risk and empirical Risk

$$\mathrm{E}_{\mathcal{S}\sim g}\left(R_g\left(\mathfrak{A}\left(\mathcal{S}\right)\right) - \hat{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}\right)\right)\right) \tag{2.2}$$

Definition 11. We say that learning algorithm $\mathfrak{A}$ suffers from over-fitting when (2.2) is 'too' large.

Note 12. The expected Risk of a learning algorithm $\mathfrak{A}(\mathcal{S})$ can be decomposed as

$$\mathrm{E}_{\mathcal{S}\sim g}\left(R_g\left(\mathfrak{A}\left(\mathcal{S}\right)\right)\right)=\underbrace{\mathrm{E}_{\mathcal{S}\sim g}\left(\hat{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}\right)\right)\right)}_{\text{(I)}}+\underbrace{\mathrm{E}_{\mathcal{S}\sim g}\left(R_g\left(\mathfrak{A}\left(\mathcal{S}\right)\right)-\hat{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}\right)\right)\right)}_{\text{(II)}} \tag{2.3}$$

by applying expectations in (2.1) and ignoring R^*. (I) indicates how well $\mathfrak{A}(\mathcal{S})$ fits the training set $\mathcal{S}$, and (II) indicates the discrepancy between the true and empirical risks of $\mathfrak{A}(\mathcal{S})$. In Section 3, we argue that the over-fitting term (II) is directly related to a certain type of stability of $\mathfrak{A}(\mathcal{S})$.

Note 13. The following result connects the need for upper bounding (and minimizing this bound) the Expected Risk in (2.3) and PAC learning (Definition 4).

Note 14. Hence, we aim to design a learning algorithm $\mathfrak{A}(\mathcal{S})$ that both fits the training set and is stable; i.e, keep both (I) and (II) in (2.3) small. As seen later, in certain learning problems, there may be a trade-off between empirical risk term (I) and (II) in (2.3). Consequently, we aim to upper bound (2.3) by upper bounding (I) and (II) individually and decide which term we should benefit against the other to achieve a certain total bound.

3. Stability and its association with over-fitting

Notation 15. Consider a learning problem $(\mathcal{H},\mathcal{Z},\ell)$. Let $\mathcal{S}=\{z_1,...,z_m\}$ be a training sample, and $\mathfrak{A}$ be a learning algorithm with output $\mathfrak{A}(\mathcal{S})$.

Note 16. It is reasonable to consider that a learning algorithm $\mathfrak{A}$ can be stable if a small change of the algorithm input does not change the algorithm output much. Mathematically formalizing this, we can say that if $\mathcal{S}^{(i)}=\{z_1,...,z_{i-1},z',z_{i+1},...,z_m\}$ is another training dataset equal to $\mathcal{S}$ but the ith element being replaced by another independent example $z'\sim g$, then a good learning algorithm $\mathfrak{A}$ would produce a small value of

$$\ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right),z_i\right)-\ell\left(\mathfrak{A}\left(\mathcal{S}\right),z_i\right)\geq 0$$

Definition 17. A learning algorithm $\mathfrak{A}$ is **on-average-replace-one-stable** with rate a decreasing function $\epsilon:\mathbb{N}\rightarrow\mathbb{R}$ if for every distribution g

$$\mathop{\mathrm{E}}_{\substack{\mathcal{S}\sim g\\ z'\sim g,\, i\sim\mathrm{U}\{1,...,m\}}}\left(\ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right),z_i\right)-\ell\left(\mathfrak{A}\left(\mathcal{S}\right),z_i\right)\right)\leq\epsilon\left(m\right) \tag{3.1}$$

Note 18. The following result demonstrates the direct association of stability and over-fitting as defined in Definitions 11 & 17.

Theorem 19. *For any learning algorithm* $\mathfrak{A}$ *it is*

$$E_{\mathcal{S}\sim g}\left(R_g\left(\mathfrak{A}\left(\mathcal{S}\right)\right)-\hat{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}\right)\right)\right)=E_{\substack{\mathcal{S}\sim g,\, z'\sim g\\ i\sim U\{1,...,m\}}}\left(\ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right),z_i\right)-\ell\left(\mathfrak{A}\left(\mathcal{S}\right),z_i\right)\right) \tag{3.2}$$

where g *is a distribution,* $\mathcal{S}=\{z_1,...,z_m\}$ *and* $\mathcal{S}^{(i)}=\{z_1,...,z_{i-1},z',z_{i+1},...,z_m\}$ are *training datasets with* $z',z_1,...,z_m\overset{\text{iid}}{\sim}g$.

Proof. As $z',z_1,...,z_m\overset{\text{iid}}{\sim}g$, then for every i

$$\mathrm{E}_{\mathcal{S}\sim g}\left(R_g\left(\mathfrak{A}\left(\mathcal{S}\right)\right)\right)=\mathrm{E}_{\substack{\mathcal{S}\sim g\\ z'\sim g}}\left(\ell\left(\mathfrak{A}\left(\mathcal{S}\right),z'\right)\right)=\mathrm{E}_{\substack{\mathcal{S}\sim g\\ z'\sim g}}\left(\ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right),z_i\right)\right)$$

and

$$\mathrm{E}_{\mathcal{S}\sim g}\left(\hat{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}\right)\right)\right)=\mathrm{E}_{\substack{\mathcal{S}\sim g\\ i\sim U\{1,...,m\}}}\left(\ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right),z_i\right)\right)$$

□

Note 20. Hence, we desire to design learning algorithms corresponding to as small as possible (3.2).

4. Implementation in regularized loss learning problems

Definition 21. A Regularized Loss Minimization (RLM) learning problem $(\mathcal{H},\mathcal{Z},\ell)$ with a regularization function $J:\mathcal{H}\to\mathbb{R}$ aims at a RLM learning rule that is the minimizer

$$h^*=\arg\min_{h\in\mathcal{H}}\left(\hat{R}_{\mathcal{S}}\left(h\right)+J\left(h\right)\right) \tag{4.1}$$

where $\hat{R}_{\mathcal{S}}\left(\cdot\right)=\frac{1}{m}\sum_{i=1}^{m}\ell\left(\cdot,z_i\right)$, and $\mathcal{S}=\{z_1,...,z_m\}\subset$ an IID training sample.

Remark 22. The motivation for considering the regularization function J in (4.1) is to: (1.) control complexity and (2.) improve stability; as we will see later.

Note 23. Here, we make our example more specific and narrow it to the Ridge RLM learning problem (could have been LASSO etc.).

Definition 24. The Ridge RLM learning problem $(\mathcal{H},\mathcal{Z},\ell)$, with $\mathcal{H}=\mathcal{W}\subset\mathbb{R}^d$, uses regularization function $J\left(w;\lambda\right)=\lambda\left\|w\right\|_2^2$ with $\lambda>0$, $w\in\mathcal{W}$ and produces learning rule

$$\mathfrak{A}\left(\mathcal{S}\right)=\arg\min_{w\in\mathcal{W}}\left(\hat{R}_{\mathcal{S}}\left(w\right)+\lambda\left\|w\right\|_2^2\right) \tag{4.2}$$

Note 25. Recall (Term 1) how the regularization function in Ridge RLM learning problem $(\mathcal{H}, \mathcal{Z}, \ell)$ penalizes complexity. Essentially, it implies a sequence of hypothesis $\mathcal{H}_1 \subset \mathcal{H}_2 \subset ...$ with $\mathcal{H}_i = \left\{w \in \mathbb{R}^d : \|w\|_2 < i\right\}$ due to duality of the corresponding minimization problem.

Note 26. Below, we will try to analyze the behavior of Ridge RLM learning rule (4.2) w.r.t. the Risk decomposition (2.3). In particular, to upper bounded w.r.t. the shrinkage term λ, training sample size m, and other characteristics.

4.1. **Bounding the empirical risk (I) in (2.3).**

Note 27. From (4.2), we have

$$\begin{aligned}\hat{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}\right)\right) &\leq \hat{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}\right)\right) + \lambda \left\|\mathfrak{A}\left(\mathcal{S}\right)\right\|_2^2 \\ &\leq \hat{R}_{\mathcal{S}}\left(w\right) + \lambda \left\|w\right\|_2^2; \quad \forall w \in \mathcal{W}\end{aligned}$$

and by taking expectations w.r.t. $\mathcal{S}$, it is

$$\mathrm{E}_{\mathcal{S}\sim g}\left(\hat{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}\right)\right)\right) \leq R_g\left(w\right) + \lambda \left\|w\right\|_2^2; \quad \forall w \in \mathcal{W} \tag{4.3}$$

because $\mathrm{E}_{\mathcal{S}\sim g}\left(\hat{R}_{\mathcal{S}}\left(\cdot\right)\right) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{E}_{\mathcal{S}\sim g}\left(\ell\left(\cdot, z_i\right)\right) = R_g\left(\cdot\right)$.

Note 28. From (4.3), we observe that part (I) in the expected risk decomposition (2.3), (aka the upper bound of the expected empirical risk) increases with the regularization term $\lambda > 0$. (!!!) –Although anticipated, it did not start well.

4.2. **Bounding the empirical risk (II) in (2.3).**

Note 29. As we will see to bound (II) in (2.3) we will have to impose an additional condition on the behavior of loss function apart from convexity; e.g. Lipstchitzness.

Assumption 30. *The loss function $\ell(\cdot, z)$ in (4.2) is convex for any $z \in \mathcal{Z}$.*

Note 31. Let $\tilde{R}_{\mathcal{S}}\left(w\right) = \hat{R}_{\mathcal{S}}\left(w\right) + \lambda \left\|w\right\|_2^2$. $\tilde{R}_{\mathcal{S}}\left(\cdot\right)$ is 2λ-strongly convex as the sum of a convex function $\hat{R}_{\mathcal{S}}\left(\cdot\right)$ (Assumption 30) and a 2λ-strongly convex function $J\left(\cdot;\lambda\right) = \lambda \left\|\cdot\right\|_2^2$ (results directly from Definition 14 in Lecture notes 2).

Fact 32. *(To be used in Note 33) If f is* λ-strongly convex and u is a minimizer of f then for any w

$$f\left(w\right) - f\left(u\right) \geq \frac{\lambda}{2}\left\|w - u\right\|^2$$

Note 33. Let $\mathfrak{A}\left(\mathcal{S}\right)$ be the Ridge learning algorithm output minimizing (4.2). Because $\tilde{R}_{\mathcal{S}}\left(\cdot\right)$ is 2λ-strongly convex and $\mathfrak{A}\left(\mathcal{S}\right)$ is its minimizer, according to Fact 32, for $\mathfrak{A}\left(\mathcal{S}\right)$ and any $w \in \mathcal{W}$, it is

$$\tilde{R}_{\mathcal{S}}\left(w\right) - \tilde{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}\right)\right) \geq \lambda \left\|w - \mathfrak{A}\left(\mathcal{S}\right)\right\|^2, \quad \forall w \in \mathcal{W} \tag{4.4}$$

Note 34. Also, for any $w, u \in \mathcal{W}$, it is

$$\begin{aligned}\tilde{R}_{\mathcal{S}}(w) - \tilde{R}_{\mathcal{S}}(u) &= \left(\hat{R}_{\mathcal{S}}(w) + \lambda \|w\|_2^2\right) - \left(\hat{R}_{\mathcal{S}}(u) + \lambda \|u\|_2^2\right) \\ &= \left(\hat{R}_{\mathcal{S}^{(i)}}(w) + \lambda \|w\|_2^2\right) - \left(\hat{R}_{\mathcal{S}^{(i)}}(u) + \lambda \|u\|_2^2\right) \\ &\quad + \frac{\ell(w, z_i) - \ell(u, z_i)}{m} + \frac{\ell(u, z') - \ell(w, z')}{m} \\ &= \tilde{R}_{\mathcal{S}^{(i)}}(w) - \tilde{R}_{\mathcal{S}^{(i)}}(u) + \frac{\ell(w, z_i) - \ell(u, z_i)}{m} + \frac{\ell(u, z') - \ell(w, z')}{m}\end{aligned}$$

Choosing $w = \mathfrak{A}\left(\mathcal{S}^{(i)}\right)$ and $u = \mathfrak{A}(\mathcal{S})$, and the fact that $\tilde{R}_{\mathcal{S}^{(i)}}\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right)\right) \le \tilde{R}_{\mathcal{S}^{(i)}}(\mathfrak{A}(\mathcal{S}))$, it is

(4.5)

$$\tilde{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right)\right) - \tilde{R}_{\mathcal{S}}(\mathfrak{A}(\mathcal{S})) \le \frac{\ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right), z_i\right) - \ell(\mathfrak{A}(\mathcal{S}), z_i)}{m} + \frac{\ell(\mathfrak{A}(\mathcal{S}), z') - \ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right), z'\right)}{m}$$

Note 35. Then (4.4) and (4.5) imply

(4.6)

$$\lambda \left\|\mathfrak{A}\left(\mathcal{S}^{(i)}\right) - \mathfrak{A}(\mathcal{S})\right\|^2 \le \frac{\ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right), z_i\right) - \ell(\mathfrak{A}(\mathcal{S}), z_i)}{m} + \frac{\ell(\mathfrak{A}(\mathcal{S}), z') - \ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right), z'\right)}{m}$$

Note 36. Now that we brought it in form (4.6), we can impose/use an additional assumption on the loss to bound it. In this example we use Lipstchitzness.

Assumption 37. *The loss function $\ell(\cdot, z)$ in (4.2) is convex and ρ-Lipschitz for any $z \in \mathcal{Z}$.*

Note 38. Given ρ-Lipschitzness in Assumption 37, it is

$$\ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right), z_i\right) - \ell(\mathfrak{A}(\mathcal{S}), z_i) \le \rho \left\|\mathfrak{A}\left(\mathcal{S}^{(i)}\right) - \mathfrak{A}(\mathcal{S})\right\| \tag{4.7}$$

$$\ell(\mathfrak{A}(\mathcal{S}), z') - \ell\left(\mathcal{S}^{(i)}, z'\right) \le \rho \left\|\mathfrak{A}\left(\mathcal{S}^{(i)}\right) - \mathfrak{A}(\mathcal{S})\right\| \tag{4.8}$$

and hence plugging 4.7 and (4.8) in (4.6) yields

$$\left\|\mathfrak{A}\left(\mathcal{S}^{(i)}\right) - \mathfrak{A}(\mathcal{S})\right\| \le 2\frac{\rho}{\lambda m} \tag{4.9}$$

Note 39. Plugging (4.9) in (4.7) yields

$$\ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right), z_i\right) - \ell(\mathfrak{A}(\mathcal{S}), z_i) \le 2\frac{\rho^2}{\lambda m}$$

Note 40. Using Theorem 19, we get an upper bound for the stability / over-fitting

$$\mathrm{E}_{\mathcal{S}\sim g}\left(R_g(\mathfrak{A}(\mathcal{S})) - \hat{R}_{\mathcal{D}}(\mathfrak{A}(\mathcal{S}))\right) = \mathrm{E}_{\substack{\mathcal{S}\sim g,\, z'\sim g \\ i \sim \mathrm{U}\{1,\ldots,m\}}}\left(\ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right), z_i\right) - \ell(\mathfrak{A}(\mathcal{S}), z_i)\right) \le 2\frac{\rho^2}{\lambda m}$$

Note 41. After this saga, the researcher could come to the conclusion that: A Ridge RLM learning problem $(\mathcal{H}, \mathcal{Z}, \ell)$ with the loss function which is convex and ρ-Lipschitz, regularizer $J(\cdot;\lambda) = \lambda \|\cdot\|^2$, $\lambda > 0$, and learning rule trained against an iid sample $\mathcal{S} = \{z_i\}_{i=1}^m$ from g is on-averave-replace-one-stable with rate $\epsilon(m) = 2\frac{\rho^2}{\lambda m}$; i.e.

$$\mathrm{E}_{\mathcal{S}\sim g}\Big(R_g(\mathfrak{A}(\mathcal{S})) - \hat{R}_{\mathcal{S}}(\mathfrak{A}(\mathcal{S}))\Big) \le 2\frac{\rho^2}{\lambda m} \tag{4.10}$$

Note 42. From (4.10), we see that stability improves (and over-fitting decreases) as the shrinkage parameter λ increases.

4.3. **Bounding the Risk (2.3).**

Note 43. Given the bounds (4.3) and (4.10), the decomposition of the expected Risk in (2.3) yields that: A Ridge RLM learning problem $(\mathcal{H}, \mathcal{Z}, \ell)$ with the loss function which is convex and ρ-Lipschitz, regularizer $J(\cdot;\lambda) = \lambda \|\cdot\|^2$, $\lambda > 0$, and learning rule trained against an iid sample $\mathcal{S} = \{z_i\}_{i=1}^m$ from g has Expected Risk bound

$$\mathrm{E}_{\mathcal{S}\sim g}(R_g(\mathfrak{A}(\mathcal{S}))) \le \underbrace{R_g(w) + \lambda \|w\|_2^2}_{\text{(I)}} + 2\underbrace{\frac{\rho^2}{\lambda m}}_{\text{(II)}}; \quad \forall w \in \mathcal{W} \tag{4.11}$$

Note 44. From (4.11), we see that there is a trade-off between Empirical Risk (I) and stability/overfitting (II) with regards the regularization parameter λ. It is desirable to use the optimal $\lambda > 0$ corresponding to the smallest bound in (4.11); it has to both fit the training data well (but perhaps not too well) and be very stable to different training data from the same g (but perhaps not too stable)!

Assumption 45. *Assume that the learning problem is additionally B-bounded by $B > 0$; i.e. $\mathcal{H} = \left\{w \in \mathbb{R}^d : \|w\|_2 \le B\right\}$.*

Note 46. If additionally the learning problem is B-bounded then the upper bound in (4.11) becomes

$$\mathrm{E}_{\mathcal{S}\sim g}(R_g(\mathfrak{A}(\mathcal{S}))) \le \underbrace{\min_{w\in\mathcal{H}} R_g(w) + \lambda B^2}_{\text{(I)}} + 2\underbrace{\frac{\rho^2}{\lambda m}}_{\text{(II)}} \tag{4.12}$$

Note 47. Furthermore, if we choose a shrinkage parameter $\lambda_m = \sqrt{\frac{2}{m}}\frac{\rho}{B}$ in (4.2) the upper bound in (4.11) becomes

$$\mathrm{E}_{\mathcal{S}\sim g}(R_g(\mathfrak{A}(\mathcal{S}))) \le \min_{w\in\mathcal{H}} R_g(w) + \rho B\sqrt{\frac{8}{m}} \tag{4.13}$$

4.4. **Deriving PAC guarantees.**

Note 48. In particular, if the size m of the training sample $\mathcal{S}$ is

$$m \geq \frac{\lambda_m^2 B^2 8}{\epsilon^2} = \frac{8\rho^2 B^2}{\epsilon^2}$$

for some $\epsilon > 0$ then for every distribution g, the upper bound in (4.11) becomes

$$\mathrm{E}_{\mathcal{S}\sim g}\left(R_g\left(\mathfrak{A}\left(\mathcal{S}\right)\right)\right) \leq \min_{w\in\mathcal{H}} R_g\left(w\right) + \epsilon \tag{4.14}$$

and hence we can have a PAC guarantee that is summarized in the following.

Summary 49. Let $(\mathcal{H}, \mathcal{Z}, \ell)$ be a learning problem convex-Lipschitz-bounded with parameters ρ and B, and let m be the training data size. The associated Ridge Regularized Loss Minimization rule with regularization function $\lambda_m = \sqrt{\frac{2}{m}\frac{\rho}{B}}$ satisfies

$$\mathrm{E}_{\mathcal{S}\sim g}\left(R_g\left(\mathfrak{A}\left(\mathcal{S}\right)\right)\right) \leq \min_{w\in\mathcal{H}} R_g\left(w\right) + \lambda_m B\sqrt{\frac{8}{m}}$$

Moreover, if $m \geq \frac{8\rho^2 B^2}{\epsilon^2}$, $\epsilon > 0$ then for every distribution g we can get a PAC -guarantee

$$\mathrm{E}_{\mathcal{S}\sim g}\left(R_g\left(\mathfrak{A}\left(\mathcal{S}\right)\right)\right) \leq \min_{w\in\mathcal{H}} R_g\left(w\right) + \epsilon$$

Lecture notes 4: Gradient descent

Lecturer & author: Georgios P. Karagiannis georgios.karagiannis@durham.ac.uk

Aim. To introduce gradient descent, its motivation, description, practical tricks, analysis in the convex scenario, and implementation.

Reading list & references:

(1) Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.
   - Ch. 14.1 Gradient Descent

1. Motivations

Problem 1. Consider a learning problem $(\mathcal{H}, \mathcal{Z}, \ell)$. Learning may involve the computation of the minimizer $h^* \in \mathcal{H}$, where $\mathcal{H}$ is a class of hypotheses, of the empirical risk function (ERF) $\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} \ell(h, z_i)$ given a finite sample $\{z_i; i = 1, ..., n\}$ generated from the data generating model $g(\cdot)$ and using loss $\ell(\cdot)$; that is

$$h^* = \arg\min_h \left(\hat{R}(h)\right) = \arg\min_h \left(\frac{1}{n}\sum_{i=1}^{n} \ell(h, z_i)\right) \tag{1.1}$$

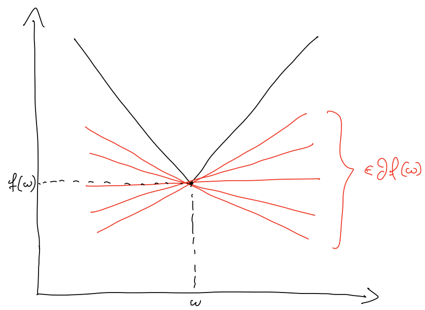
or its corresponding regularized version. If analytical minimization of (1.1) is impossible or impractical, numerical procedures can be applied; eg Gradient Descent (GD) algorithms. Such approaches introduce numerical errors in the solution.

2. subgradient

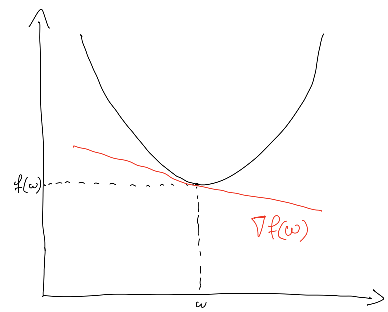
Definition 2. Vector v is called subgradient of a function $f : S \to \mathbb{R}$ at $w \in S$ if

$$\forall u \in S, \quad f(u) \geq f(w) + \langle u - w, v\rangle \tag{2.1}$$

Note 3. There may be more than one subgradients of a function at a specific point. As seen by (2.1), subgradients are the slopes of all the lines passing through the point $(w, f(w))$ and been under the function $f(\cdot)$.



(A) subgradients satisfying (2.1) in the non-differentiable case



(B) gradient satisfying the equality in (2.1) in the differentiable case

Definition 4. The set of subgradients of function $f : S \to \mathbb{R}$ at $w \in S$ is denoted by $\partial f(w)$.

Fact 5. *Properties of subgradient sets useful for the subgradient construction*

(1) *If function $f : S \to \mathbb{R}$ is differentiable at w then the only subgradient of f at w is the gradient $\nabla f(w)$, and (2.1) is equality; i.e. $\partial f(w) = \{\nabla f(w)\}$.*

(2) *for constants α, β and convex function $f(\cdot)$, it is*

$$\partial (\alpha f(w) + \beta) = \alpha (\partial f(w)) = \{\alpha v : v \in \partial f(w), \}$$

(3) *for convex functions $f(\cdot)$ and $g(\cdot)$, it is*

$$\partial (f(w) + g(w)) = \partial f(w) + \partial g(w) = \{v + u : v \in \partial f(w), \, and u \in \partial g(w)\}$$

Example 6. Consider the function $f : \mathbb{R} \to \mathbb{R}_+$ with $f(w) = |w| = \begin{cases} w & w \geq 0 \\ -w & w < 0 \end{cases}$. Find the set of subgradients $\partial f(w)$ for each $w \in \mathbb{R}$.

Solution: Using Fact 5, it is $\partial f(w) = 1$ for $w > 0$ and $\partial f(w) = -1$ for $w < 0$ as f is differentiable for $x \neq 0$. At $x = 0$, f is not differentiable; hence from condition (2.1) it is

$$\forall u \in \mathbb{R}, \;\; |u| \geq |0| + (u - 0) v$$

which is satisfied for $v \in [-1, 1]$. Hence,

$$\partial f(w) = \begin{cases} \{-1\} & , w < 0 \\ [-1, 1] & , w = 0 \\ \{1\} & , w > 0 \end{cases}$$

Problem 7. For the sake of notation simplicity and generalization, we will present Gradient Descent (GD) in the following minimization problem

$$w^{*}=\arg\min_{w\in\mathcal{H}}\left(f\left(w\right)\right) \tag{3.1}$$

where $f:\mathcal{H}\rightarrow\mathbb{R}$, and $w\in\mathcal{H}\subseteq\mathbb{R}^{d}$; $f\left(\cdot\right)$ is the function to be minimized, e.g., $f\left(\cdot\right)$ can be an empirical risk function $\hat{R}\left(\cdot\right)$.

Algorithm 8. *Gradient Descent (GD) algorithm with learning rate $\eta_{t}>0$ for the solution of the minimization problem (3.1)*

For $t=1,2,3,...$ iterate:

(1) *compute*

$$w^{(t+1)}=w^{(t)}-\eta_{t}v_{t};\ \ \textit{where}\ \ v_{t}\in\partial f\left(w^{(t)}\right) \tag{3.2}$$

(2) *terminate if a termination criterion is satisfied, e.g.*

$$\textit{If}\ t\geq T_{\max}\ \ \textit{then}\ \ \textit{STOP}$$

Note 9. (Intuition) Assume f is differentiable. GD produces a chain $\left\{w^{(t)}\right\}$ that drifts towards a minimum w^{*}. The chain is directed towards the opposite direction of that of the gradient $\nabla f\left(\cdot\right)$ and at a rate controlled by the learning rate η_{t}.

Note 10. (More intuition) Assume f is differentiable. Consider the (1st order) Taylor polynomial for the approximation of $f\left(w\right)$ in a small area around u (i.e. $\|v-u\|=\text{small}$)

$$f\left(u\right)\approx P\left(u\right)=f\left(w\right)+\langle u-w,\nabla f\left(w\right)\rangle$$

Assuming convexity for f, it is

$$f\left(u\right)\geq\underbrace{f\left(w\right)+\langle u-w,\nabla f\left(w\right)\rangle}_{=P(u;w)} \tag{3.3}$$

meaning that P lower bounds f. Hence we could design an updating mechanism producing $w^{(t+1)}$ which is nearby $w^{(t)}$ (small steps) and which minimize the linear approximation $P\left(w\right)$ of $f\left(w\right)$ at $w^{(t)}$

$$P\left(w;w^{(t)}\right)=f\left(w^{(t)}\right)+\langle w-w^{(t)},\nabla f\left(w^{(t)}\right)\rangle. \tag{3.4}$$

while hoping that this mechanism would push the produced chain $\left\{w^{(t)}\right\}$ towards the minimum because of (3.3). Hence we could recursively minimize the linear approximation (3.4) and the distance between the current state $w^{(t)}$ and the next w value to produce $w^{(t+1)}$; namely

$$
\begin{aligned}
w^{(t+1)} =& \arg\min_{\forall w} \left(\frac{1}{2} \left\| w - w^{(t)} \right\|^2 + \eta P\left(w; w^{(t)}\right) \right) \\
=& \arg\min_{\forall w} \left(\frac{1}{2} \left\| w - w^{(t)} \right\|^2 + \eta \left(f\left(w^{(t)}\right) + \langle w - w^{(t)}, \nabla f\left(w^{(t+1)}\right) \rangle \right) \right) \\
=& w^{(t)} - \eta \nabla f\left(w^{(t)}\right)
\end{aligned}
\tag{3.5}
$$

where parameter $\eta > 0$ controls the trade off in (3.5).

Note 11. GD output (e.g. in Note 14) converges to a local minimum, $w_{\text{GD}}^{(T)} \to w_*$ (in some sense), under different sets of regularity conditions (some are weaker other stronger). Section 4 has a brief analysis.

Note 12. The parameter η_t is called learning rate (or step size, gain). It determines the size of the steps GD takes to reach a (local) minimum. $\{\eta_t\}$ is a non-negative sequence and it is chosen by the practitioner. In principle, regularity conditions (Note 11) often imply restrictions on the decay of $\{\eta_t\}$ which guide the practitioner to parametrize it properly. Some popular choices of learning rate η_t are:

(1) <u>constant</u>; $\eta_t = \eta$, for where $\eta > 0$ is a small value. The rationale is that GD chain $\{w_t\}$ performs constant small steps towards the (local) minimum w_* and then oscillate around it.
(2) <u>decreasing and converging to zero</u>; $\eta_t \searrow$ with $\lim_{t\to\infty} \eta_t = 0$. E.g. $\eta_t = \left(\frac{C}{t}\right)^{\varsigma}$ where $\varsigma \in [0.5, 1]$ and $C > 0$. The rationale is that GD algorithm starts by performing larger steps (controlled by C) at the beginning to explore the area for discovering possible minima. Also it reduces the size of those steps with the iterations (controled by ς) such that eventually when the chain $\{w_t\}$ is close to a possible minimum w_* value to converge and do not overshoot.
(3) <u>decreasing and converging to a tiny value</u> τ_*; $\eta_t \searrow$ with $\lim_{t\to\infty} \eta_t = \tau_*$ E.g. $\eta_t = \left(\frac{C}{t}\right)^{\varsigma} + \tau_*$ with $\varsigma \in (0.5, 1]$, $C > 0$, and $\tau_* \approx 0$. Same as previously, but the algorithm aims at oscillating around the detected local minimum.
(4) <u>constant until an iteration T_0 and then decreasing</u>; Eg $\eta_t = \left(\frac{C}{\max(t,T_0)}\right)^{\varsigma}$ with $\varsigma \in [0.5, 1]$ and $C > 0$, and $T_0 < T$. The rationale is that at the first stage of the iterations (when $t \leq T_0$) the algorithm may need a constant large steps for a significant number of iterations T_0 in order to explore the domain; and hence in order for the chain $\{w_t\}$ to reach the area around the (local) minimum w_*. In the second stage, hoping that the chain $\{w_t\}$ may be in close proximity to the (local) minimum w_* the algorithm progressively performs smaller steps to converge towards the minimum w_*. The first stage ($t \leq T_0$) is called burn-in; the values $\{w_t\}$ produced during the burn-in ($t \leq T_0$) are are often discarded/ignored from the output of the GD algorithm.

- Parameters $C, \varsigma, \tau_*, T_0$ may be chosen based on pilot runs against a small fraction of the training data set.

Note 13. There are several practical termination criteria that can be used in GD Algorithm 8(step 2). They aim to terminate the recursion in practice. Some popular termination criteria are

(1) terminate when the gradient is sufficiently close to zero; i.e. if $\left\|\nabla f\left(w^{(t)}\right)\right\| \leq \epsilon$, for some pre-specified tiny $\epsilon > 0$ then STOP
(2) terminate when the chain $w^{(t)}$ does not change; i.e. if $\left\|w^{(t+1)} - w^{(t)}\right\| \leq \epsilon \left\|w^{(t)}\right\|$ for some pre-specified tiny $\epsilon > 0$ then STOP
(3) terminate when a pre-specified number of iterations T is performed; i.e. if $t \geq T$ then STOP

Here (1) may be deceive if the chain is in a flat area, (2) may be deceived if the learning rate become too small, (3) is obviously a last resort.

Note 14. Given T iterations of GD algorithm, the output of GD can be (but not a exclusively),

(1) the average (after discarding the first few iterations of $w^{(t)}$ for stability reasons)

$$w_{\text{GD}}^{(T)} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)} \tag{3.6}$$

(2) or the best value discovered

$$w_{\text{GD}}^{(T)} = \arg\min_{\forall w_t}\left(f\left(w^{(t)}\right)\right)$$

(3) or the last value discovered

$$w_{\text{GD}}^{(T)} = w^{(T)}$$

4. Analysis of Gradient descent (Algorithm 8)

Note 15. We analyze the behavior of GD addressing the minimization Problem 7 under Assumptions 16.

Assumption 16. *For the sake of the analysis of the GD, let us consider:*

(1) *The function $f(\cdot)$ is convex and Lipschitz*
(2) *GD has a constant learning rate $\eta_t = \eta$,*
(3) *GD output $w_{GD}^{(T)} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$*

Lemma 17. *Let $\{v_t; t = 1, ..., T\}$ be a sequence of vectors. Any algorithm with $w^{(1)} = 0$ and $w^{(t+1)} = w^{(t)} - \eta v_t$ for $t = 1, ..., T$ satisfies*

No need to memorize

$$\sum_{t=1}^{T}\langle w^{(t)} - w^{*}, v_t\rangle \leq \frac{\|w^{*}\|^2}{2\eta} + \frac{\eta}{2}\sum_{t=1}^{T}\|v_t\|^2 \tag{4.1}$$

Proof. Omitted; see the Exercise 12 in the Exercise sheet 1. □

Note 18. To find an upper bound of the GD error, we try to bound the error $f\left(w_{\text{GD}}^{(T)}\right) - f(w^{*})$ with purpose to use Lemma 17.

Note 19. Consider the minimization problem (3.1). Given Assumptions 16, the error can be bounded as

$$f\left(w_{\text{GD}}^{(T)}\right) - f(w^{*}) \leq \frac{\|w^{*}\|^2}{2\eta T} + \frac{\eta}{2}\frac{1}{T}\sum_{t=1}^{T}\|v_t\|^2 \tag{4.2}$$

where $v_t \in \partial f\left(w^{(t)}\right)$. If $f(\cdot)$ is differentiable then $v_t = \nabla f\left(w^{(t)}\right)$

Proof. It is[1]

$$
\begin{aligned}
f\left(w_{\mathrm{GD}}^{(T)}\right)-f\left(w^{*}\right)= & f\left(\frac{1}{T} \sum_{t=1}^{T} w_{t}\right)-f\left(w^{*}\right) \\
\leq & \frac{1}{T} \sum_{t=1}^{T}\left(f\left(w_{t}\right)-f\left(w^{*}\right)\right) && \text{(by Jensen's inequality)} \quad (4.3)\\
\leq & \frac{1}{T} \sum_{t=1}^{T}\langle w^{(t)}-w^{*}, v_{t}\rangle && \text{(by convexity of } f(\cdot)) \\
\leq & \frac{\left\|w^{*}\right\|^{2}}{2 \eta T}+\frac{\eta}{2} \frac{1}{T} \sum_{t=1}^{T}\left\|v_{t}\right\|^{2} && \text{(by roceeding Lemma)} \quad (4.4)
\end{aligned}
$$

□

Note 20. Note 19 shows that it is important to bound the (sub-)gradient in (4.2) in a meaningful manner. Lemma 21 shows that if we assume Lipschitzness as well, the (sub-)gradient has the so-called self-bounded behavior.

Lemma 21. [2] *$f: S \rightarrow \mathbb{R}$ is ρ-Lipschitz over an open convex set S if and only if for all $w \in S$ and $v \in \partial f(w)$ it is $\|v\| \leq \rho$.*

Solution: *$\Longrightarrow$ Let $f: S \rightarrow \mathbb{R}$ be ρ-Lipschitz over convex set S, $w \in S$ and $v \in \partial f(w)$.*

- *Since S is open we get that there exist $\epsilon>0$ such as $u:=w+\epsilon \frac{v}{\|v\|}$ where $u \in S$. So $\langle u-w, v\rangle=\epsilon\|v\|$ and $\|u-w\|=\epsilon$.*
  - *From the subgradient definition we get*
  $$f(u)-f(w) \geq\langle u-w, v\rangle=\epsilon\|v\|$$
  - *From the Lipschitzness of $f(\cdot)$ we get*
  $$f(u)-f(w) \leq \rho\|u-w\|=\rho \epsilon$$

  Therefore $\|v\| \leq \rho$.

Proof. For $\Longleftarrow$ see Exercise 4 in the Exercise sheet. □

Note 22. The following summaries Note 19 and Lemma 21 with respect to the GD algorithm under Assumption 16.

Note 23. Let $f(\cdot)$ be a convex and ρ-Lipschitz function. If we run GD algorithm of f with learning rate $\eta>0$ for T steps the output $w_{\mathrm{GD}}^{(T)}=\frac{1}{T} \sum_{t=1}^{T} w^{(t)}$ satisfies

$$f\left(w_{\mathrm{GD}}^{(T)}\right)-f\left(w^{*}\right) \leq \frac{\left\|w^{*}\right\|^{2}}{2 \eta T}+\frac{\eta}{2} \frac{1}{T} \sum_{t=1}^{T}\left\|v_{t}\right\|^{2}$$

[1] Jensen's inequality for convex $f(.)$ is $f(\mathrm{E}(x)) \leq \mathrm{E}(f(x))$

[2] If this was a Homework there would be a Hint:

- If S is open there exist $\epsilon>0$ such as $u=w+\epsilon \frac{v}{\|v\|}$ such as $u \in S$

where $\|v_t\| \leq \rho$, $v_t \in \partial f\left(w^{(t)}\right)$. If $f(\cdot)$ is differentiable then $v_t = \nabla f\left(w^{(t)}\right)$.

Proof. Straightforward from Lemma 17 and Note 19. □

Note 24. The following shows that given a learning rate depending on the iteration t, we can reduce the upper bound of the error as well as find the number of required iterations to achieve convergence.

Note 25. (Cont Prop. 23) Let $f(\cdot)$ be a convex and ρ-Lipschitz function, and let $\mathcal{H} = \{w \in \mathbb{R} : \|w\| \leq B\}$. Assume we run GD algorithm of $f(\cdot)$ with learning rate $\eta_t = \sqrt{\frac{B^2}{\rho^2 T}}$ for T steps, and output $w_{\text{GD}}^{(T)} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$. Then

(1) upper bound on the sub-optimality is

$$f\left(w_{\text{GD}}^{(T)}\right) - f(w^*) \leq \frac{B\rho}{\sqrt{T}} \tag{4.5}$$

(2) a given level off accuracy ε such that $f\left(w_{\text{GD}}^{(T)}\right) - f(w^*) \leq \varepsilon$ can be achieved after T iterations

$$T \geq \frac{B^2\rho^2}{\varepsilon^2}.$$

Proof. Part 1 is a simple substitution from Proposition 23, and part 2 is implied from part 1. □

The result on Note 25 heavily relies on setting suitable values for B and ρ which is rather a difficult task to be done in very complicated learning problems (e.g., learning a neural network).

5. Examples [3]

Example 26. Consider the simple Normal linear regression problem where the dataset $\{z_i = (y_i, x_i)\}_{i=1}^{n} \in \mathcal{S}$ is generated from a Normal data generating model

$$\begin{pmatrix} y_i \\ x_i \end{pmatrix} \overset{\text{iid}}{\sim} \text{N}\left(\begin{pmatrix} \mu_y \\ \mu_x \end{pmatrix}, \begin{bmatrix} \sigma_y^2 & \rho\sqrt{\sigma_y^2\sigma_x^2} \\ \rho\sqrt{\sigma_y^2\sigma_x^2} & \sigma_x^2 \end{bmatrix}\right) \tag{5.1}$$

for $i = 1, ..., n$. Consider a hypothesis space $\mathcal{H}$ of linear functions $h : \mathbb{R}^2 \to \mathbb{R}$ with $h(w) = w_1 + w_2 x$. The exact solution (which we pretend we do not know) is given as

$$\begin{pmatrix} w_1^* \\ w_2^* \end{pmatrix} = \begin{pmatrix} \mu_y - \rho\frac{\sigma_y}{\sigma_x}\mu_x \\ \rho\frac{\sigma_y}{\sigma_x} \end{pmatrix}. \tag{5.2}$$

To learn the optimal $w^* = (w_1^*, w_2^*)^\top$, we consider a loss $\ell\left(w, z_i = (x_i, y_i)^\top\right) = (y_i - [w_1 + w_2 x_i])^2$, which leads to the minimization problem

$$w^* = \arg\min_w \left(\hat{R}_{\mathcal{S}}(w)\right) = \arg\min_w \left(\frac{1}{n}\sum_{i=1}^{n}(y_i - w_1 - w_2 x_i)^2\right)$$

The GD Algorithm 8 with learning rate η is

For $t = 1, 2, 3, ...$ iterate:

[3] Code is available in `https://github.com/georgios-stats/Machine_Learning_and_Neural_Networks_III_Epiphany_2025/tree/main/Lecture_notes/code/example_GD.R`

(1) compute

$$w^{(t+1)} = w^{(t)} - \eta v_t, \qquad \text{where } v_t = \begin{pmatrix} 2w_1^{(t)} + 2w_2^{(t)}\bar{x} - 2\bar{y} \\ 2w_1^{(t)}\bar{x} + 2w_2^{(t)}\overline{x^2} - 2y^\top x \end{pmatrix} \tag{5.3}$$

(2) terminate if a termination criterion is satisfied, e.g.

$$\text{If } t \geq T \text{ then STOP}$$

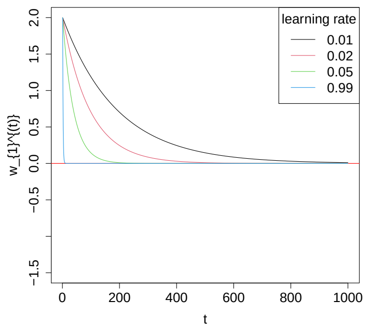
This is because $\hat{R}_{\mathcal{D}}(w)$ is differentiable in $\mathbb{R}^2$ so $\partial \hat{R}_{\mathcal{D}}(w) = \left\{\nabla \hat{R}_{\mathcal{D}}(w)\right\}$ and because

$$\nabla \hat{R}_{\mathcal{D}}(w) = \begin{pmatrix} \frac{\mathrm{d}}{\mathrm{d}w_1}\hat{R}_{\mathcal{S}}(w) \\ \frac{\mathrm{d}}{\mathrm{d}w_2}\hat{R}_{\mathcal{S}}(w) \end{pmatrix} = \ldots = \begin{pmatrix} 2w_1^{(t)} + 2w_2^{(t)}\bar{x} - 2\bar{y} \\ 2w_1^{(t)}\bar{x} + 2w_2^{(t)}\overline{x^2} - 2y^\top x \end{pmatrix}$$

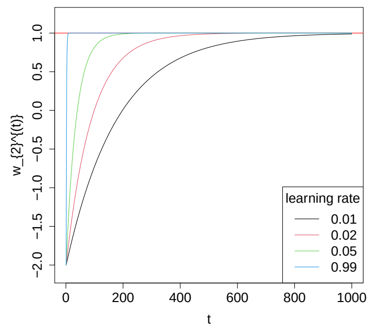
Consider data size $n = 100$, and parameters $\rho = 0.2$, $\sigma_y^2 = 1$ and $\sigma_x^2 = 1$. Then the real value (5.2) that I need to learn equals to $w^* = (0,1)^\top$. Consider a GD seed $w_0 = (2,-2)$, and total number of iterations $T = 1000$.

Figures 5.1a, 5.1b, and 5.1c present trace plots of the chain $\left\{\left(w^{(t)}\right)\right\}$ and error $\hat{R}_{\mathcal{S}}\left(w^{(t)}\right) - \hat{R}_{\mathcal{S}}(w^*)$ produced by running GD for $T = 1000$ total iterations and for different (each time) constant learning rates $\eta \in \{0.01, 0.02, 0.05, 0.99\}$. We observe that the larger learning rates under consideration were able to converge faster to the minimum w^*. This is because they perform larger steps and can learn faster -this is not a panacea.

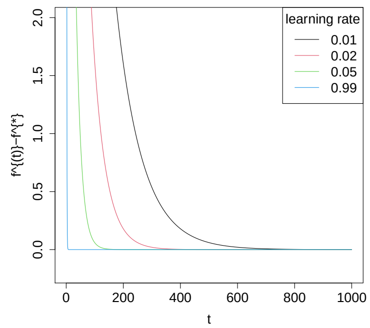
Figures 5.1d, 5.1e, and 5.1f present trace plots of the chain $\left\{\left(w^{(t)}\right)\right\}$, and of the error $\hat{R}_{\mathcal{S}}\left(w^{(t)}\right) - \hat{R}_{\mathcal{S}}(w^*)$ produced by running GD for $T = 1000$ total iterations and for learning rate $\eta = 1.0$ (previously considered) and a very big learning rate $\eta = 3.0$. We observe that the very big learning rate $\eta = 3.0$ presents slower convergence to the minimum w^*. This is because it creates unreasonably big steps in (3.2) that the produced chain overshoots the global minimum; see the cartoon in Figures 5.1g and 5.1h.



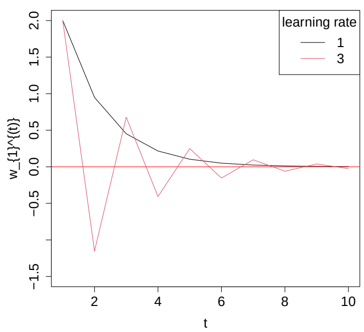
(A) $\left\{\left(w_1^{(t)}\right)\right\}$



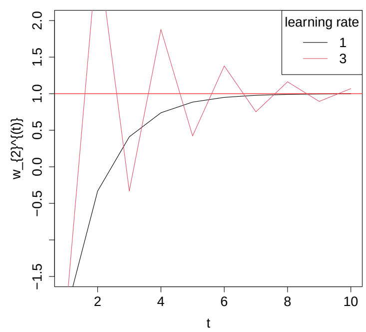
(B) $\left\{\left(w_2^{(t)}\right)\right\}$



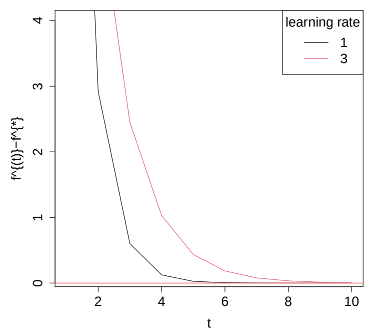
(C) $\hat{R}_{\mathcal{S}}\left(w^{(t)}\right) - \hat{R}_{\mathcal{S}}\left(w^{*}\right)$



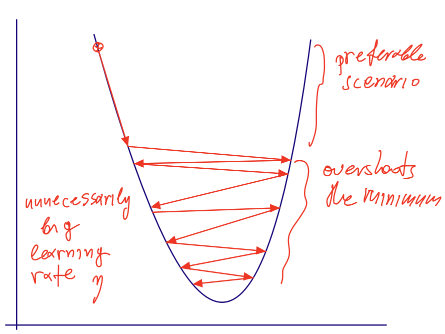
(D) $\left\{\left(w_1^{(t)}\right)\right\}$



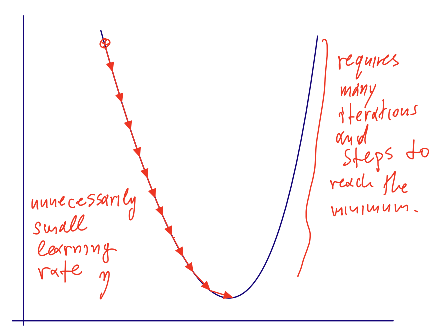
(E) $\left\{\left(w_2^{(t)}\right)\right\}$



(F) $\hat{R}_{\mathcal{S}}\left(w^{(t)}\right) - \hat{R}_{\mathcal{S}}\left(w^{*}\right)$



(G) Unnecessarily large learning rate



(H) Unnecessarily small learning rate

FIGURE 5.1

Lecture notes 6: Variations of stochastic gradient descent

Lecturer & author: Georgios P. Karagiannis georgios.karagiannis@durham.ac.uk

Aim. To introduce the stochastic gradient descent variations for restricted sets, adaptive step functions, and acceleration/variance reduction.

Reading list & references:

(1) Johnson, Rie, and Tong Zhang. "Accelerating stochastic gradient descent using predictive variance reduction." Advances in neural information processing systems 26 (2013).

(2) Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." Journal of machine learning research 12, no. 7 (2011).

1. Stochastic gradient descent with projection

Problem 1. Consider the minimization problem

$$w^{*}=\arg\min_{w}\left(f\left(w\right)\right) \tag{1.1}$$

where $f:\mathbb{R}^{d}\rightarrow\mathbb{R}$, $w\in\mathcal{W}$, and assume $\mathcal{W}\subset\mathbb{R}^{d}$ is a restricted domain.

Note 2. In Problem 1 if for instance, $\mathcal{W}=\left\{ w\in\mathbb{R}^{d}:\left\Vert w\right\Vert \leq B\right\}$ for $B>0$, f is convex over $\mathcal{W}$, but f is non-convex in $\mathbb{R}^{d}/\mathcal{W}$, direct implementation of (standard) SGD (Algorithm 6, Lect. Notes 5) may produce a chain stepping out $\mathcal{W}$ and hence an output $w_{\text{SGD}}\notin\mathcal{W}$.

Note 3. Stochastic gradient descent with projection (prjSGD) is a modified SGD suitable for applications with restricted sets and relies on the consideration of a projection step guaranteeing $w\in\mathcal{W}$.

Algorithm 4. *Stochastic Gradient Descent with learning rate $\eta_{t}>0$ and with projection in $\mathcal{W}$ for Problem1*

For $t=1,2,3,...$ iterate:

(1) *compute*

$$w^{\left(t+\frac{1}{2}\right)}=w^{(t)}-\eta_{t}v_{t}, \tag{1.2}$$

where v_{t} is a random vector such that $E\left(v_{t}|w^{(t)}\right)\in\partial f\left(w^{(t)}\right)$

(2) *compute*

$$w^{(t+1)}=\arg\min_{w\in\mathcal{W}}\left(\left\Vert w-w^{\left(t+\frac{1}{2}\right)}\right\Vert \right) \tag{1.3}$$

(3) *terminate if a termination criterion is satisfied*

Note 5. The analysis of SGD with projection is given as a Exercise 13 from the Exercise sheet.

Example 6. [1] Consider a (rather naive) loss function $\ell(w, z = (x, y)) = -\cos(0.5(y - w_1 - w_2 x))$, a hypothesis class $\mathcal{H} = \left\{h_w(x) = w^\top x : w \in \mathcal{W}\right\}$ and $\mathcal{W} = \left\{w \in \mathbb{R}^2 : \|w\| \leq 1.5\right\}$, and assume that inputs x in dataset S are such that $x \in [-1, 1]$. Note that $-\cos(\cdot)$ is convex in $[-1.5, 1.5]$ and non-convex in $\mathbb{R}$. Consider learning rate $\eta_t = 50/t$ reducing to zero, and seed $w^{(0)} = (1.5, 1.5)$. An unconstrained SGD may produce a minimizer/solution outside $\mathcal{H}$ because η_t is too large at the first few iterations. We can design the online SGD ($m = 1$) with projection to $\mathcal{H}$ as
For $t = 1, 2, 3, ...$ iterate:

(1) randomly generate a set $\mathcal{J}^{(t)} = \{j^*\}$ by drawing one number from $\left\{1, ..., n = 10^6\right\}$, and set $\tilde{\mathcal{S}}_1 = \{z_{j^*}\}$

(2) compute

$$\begin{aligned}
w^{(t+1/2)} =& w^{(t)} - \frac{25}{t}\begin{pmatrix} \sin\left(0.5\left(y_{j^*} - w_1^{(t)} - w_2^{(t)} x_{j^*}\right)\right) \\ \sin\left(0.5\left(y_{j^*} - w_1^{(t)} - w_2^{(t)} x_{j^*}\right)\right) x_{j^*} \end{pmatrix} \\
w^{(t+1/2)} =& \arg\min_{\|w\| \leq 1.5}\left(\left\|w - w^{(t+1/2)}\right\|\right)
\end{aligned}$$

(3) if $t \geq T = 1000$ STOP

because

$$v_t = \nabla \ell\left(w^{(t)}, z_{j^*}\right) = \begin{pmatrix} \frac{\partial}{\partial w_1^{(t)}} \ell\left(w^{(t)}, z_{j^*}\right) \\ \frac{\partial}{\partial w_2^{(t)}} \ell\left(w^{(t)}, z_{j^*}\right) \end{pmatrix} = \begin{pmatrix} 0.5\sin\left(0.5\left(y_{j^*} - w_1^{(t)} - w_2^{(t)} x_{j^*}\right)\right) \\ 0.5\sin\left(0.5\left(y_{j^*} - w_1^{(t)} - w_2^{(t)} x_{j^*}\right)\right) x_{j^*} \end{pmatrix}$$

In Figures A.1b & A.1e, we observe that the SGD got trapped outside $\mathcal{H}$ due to the unreasonably large learning rate at the beginning of the iterations, while the SGD with projection step managed to stay in $\mathcal{H}$ and converge.

2. Preconditioned SGD abd the AdaGrad algorithm

Problem 7. Consider the minimization problem

$$w^* = \arg\min_{w \in \mathcal{W}} (f(w)) \tag{2.1}$$

where $f : \mathcal{W} \to \mathbb{R}$, and assume $\mathcal{W} \equiv \mathbb{R}^d$.

Note 8. All SGD (introduced earlier) are first order methods, in the sense they consider only the gradient, and hence they ignore the curvature of the space. Consequently they may present slow converge in cases the curvature changes, eg. among dimensions of w.

Note 9. Preconditioned Stochastic Gradient Descent (Algorithm 10) is a modified SGD aiming to expedite convergence of SGD by using a preconditioner P_t that accounts for the curvature (or geometry in general) of f.

[1] `https://github.com/georgios-stats/Machine_Learning_and_Neural_Networks_III_Epiphany_2025/tree/main/Lecture_notes/code/example_projSGD.R`

Algorithm 10. *Preconditioned Stochastic Gradient Descent with learning rate $\eta_t > 0$, and preconditioner P_t for the solution of the minimization problem (3.1)*

For $t = 1, 2, 3, ...$ iterate:

(1) *compute*

$$w^{(t+1)} = w^{(t)} - \eta_t P_t v_t, \tag{2.2}$$

where v_t is a random vector such that $E\left(v_t | w^{(t)}\right) \in \partial f\left(w^{(t)}\right)$, and P_t is a preconditioner

(2) *terminate if a termination criterion is satisfied, e.g.*

$$\text{If } t \geq T \text{ then STOP}$$

Note 11. A natural choice of P_t can be $P_t := [H_t + \epsilon I_d]^{-1}$, where H_t is the Hessian matrix $[H_t]_{i,j} = \frac{\partial^2}{\partial w_i \partial w_j} f(w)\Big|_{w=w^{(t)}}$ (ie. the gradient of the gradient's elements), and ϵ is a tiny $\epsilon > 0$ to mitigate machine error when Hessian elements are close to zero.

Note 12. Using the inverse of the full Hessian as preconditioner P_t may be too expensive to perform matrix operations in (2.2) and operations can be too unstable/inaccurate due to the random error induced by the stochasticity of the gradient.

2.1. **Adaptive Stochastic Gradient Decent (AdaGrad).**

Note 13. AdaGrad (Algorithm 15) is a modified SGD algorithm that aims to adaptively adjust the learning rate of SGD, perform more informative gradient-based learning, and by improve convergence performance over standard SGD by dynamically incorporating knowledge of the geometry of function $f(\cdot)$ in earlier iterations.

Note 14. It is particularly suitable in cases where the data are sparse and sparse features w's are more informative. It aims to perform larger updates (i.e. high learning rates) for those dimensions of w that are related to infrequent features (largest partial derivative) and smaller updates (i.e. low learning rates) for frequent ones (smaller partial derivative).

Algorithm 15. *Adaptive Stochastic Gradient Decent (AdaGrad) is a preconditioned SGD (Algorithm 10) with preconditioner $P_t = [diag(G_t) + \epsilon I_d]^{-1/2}$ where notation $diag(A)$ denotes a $d \times d$ matrix whose diagonal is the d dimensional diagonal vector $(A_{1,1}, A_{2,2}..., A_{d,d})$ of $d \times d$ matrix A and whose off-diagonal elements are zero, $G_t = \sum_{\tau=1}^{t} v_\tau v_\tau^\top$ is the sum of the outer products of the gradients $\{v_\tau; \tau \leq t\}$ up to the state t, and $\epsilon > 0$ is a tiny value (eg, 10^{-6}) set for computational stability is case the gradient becomes too close to zero. I.e.*

$$w^{(t+1)} = w^{(t)} - \eta_t \left[diag(G_t) + \epsilon I_d\right]^{-1/2} v_t. \tag{2.3}$$

Note 16. AdaGrad individually adapts the learning rate of each dimension of w_t by scaling them inversely proportional to the square root of the sum of all the past squared values of the gradient

$\{v_{\tau};\tau\leq t\}$. This is because

$$[G_t]_{j,j}=\sum_{\tau=1}^{t}(v_{\tau,j})^2 \tag{2.4}$$

where j denotes the j-th dimension of w. Hence (2.3) and (2.3)imply that the j-th dimension of w is updated as

$$w_j^{(t+1)}=w_j^{(t)}-\eta_t\frac{1}{\sqrt{[G_t]_{j,j}+\epsilon}}v_{t,j}.$$

Note 17. The accumulation of positive terms in (2.4) makes the sum keep growing during training and causes the learning rate to shrink and become infinitesimally small. This offers an automatic way to choose a decreasing learning rate simplifying setting the learning rate; however it may result in a premature and excessive decrease in the effective learning rate. This can be mitigated by still considering in (2.3) a (user specified rate) $\eta_t\geq 0$ and tuning it properly via pilot runs.

Example 18. [2] AdaGrad with $\eta_t=1$, $\epsilon=10^{-6}$, and batch size $m=1$ is

- Set $G_0=(0,0)^\top$.
- For $t=1,2,3,...$ iterate:
  (1) randomly generate $j^{(t)}$ from $\left\{1,...,n=10^6\right\}$
  (2) compute
$$\begin{aligned}v_t&=\begin{pmatrix}2w_1^{(t)}+2w_2^{(t)}x_{j^{(t)}}-2y_{j^{(t)}}\\2w_1^{(t)}\bar{x}+2w_2^{(t)}x_{j^{(t)}}^2-2y_{j^{(t)}}x_{j^{(t)}}\end{pmatrix}\\G_t&=G_{t-1}+\begin{pmatrix}v_{t,1}^2\\v_{t,2}^2\end{pmatrix}\\w^{(t+1)}&=\begin{pmatrix}w_1^{(t)}-\frac{\eta_t}{\sqrt{G_{t,1+\epsilon}}}v_{t,1}\\w_2^{(t)}-\frac{\eta_t}{\sqrt{G_{t,2+\epsilon}}}v_{t,2}\end{pmatrix},\end{aligned}$$
  (3) if $t\geq T=1000$ STOP

because

$$v_t=\nabla\ell\left(w^{(t)},z_{j^{(t)}}\right)=\begin{pmatrix}\frac{\partial}{\partial w_1^{(t)}}\ell\left(w^{(t)},z_{j^{(t)}}\right)\\\frac{\partial}{\partial w_2^{(t)}}\ell\left(w^{(t)},z_{j^{(t)}}\right)\end{pmatrix}=\begin{pmatrix}2w_1^{(t)}+2w_2^{(t)}x_{j^{(t)}}-2y_{j^{(t)}}\\2w_1^{(t)}\bar{x}+2w_2^{(t)}x_{j^{(t)}}^2-2y_{j^{(t)}}x_{j^{(t)}}\end{pmatrix}$$

In Figures A.1g & A.1h, we see that AdaGrad with $\eta=1$ works (I did not try to tune it), however to make vanilla SGD to work I have to tune $\eta=0.03$ otherwise for $\eta=1.0$ it did not work.

[2] `https://github.com/georgios-stats/Machine_Learning_and_Neural_Networks_III_Epiphany_2025/tree/main/Lecture_notes/code/example_AdaGrad.R`

3. Stochastic Variance Reduced Gradient (SVRG)

Note 19. (Recall) Consider minimization problem $w^* = \arg\min_w \{f(w)\}$. Consider SGD with learning rate $\eta > 0$, output $w_{\text{SGD}}^{(T)}$, and general recursion

$$w^{(t+1)} = w^{(t)} - \eta_t v_t$$

where $\mathrm{E}\left(v_t | w^{(t)}\right) \in \partial f\left(w^{(t)}\right)$. Recall that the upper bound of the error depends on the variance of the stochastic sub-gradient i.e. $\mathrm{E}\left\|v_t - \mathrm{E}(v_t)\right\|^2$.

$$\mathrm{E}\left(f\left(w_{\text{SGD}}^{(T)}\right)\right) - f(w^*) \le \frac{\|w^*\|^2}{2\eta T} + \frac{\eta}{2}\frac{1}{T}\sum_{t=1}^{T}\mathrm{E}\|v_t\|^2 \tag{3.1}$$

$$\mathrm{E}\|v_t\|^2 = \underbrace{\mathrm{E}\|v_t - \mathrm{E}(v_t)\|^2}_{=\mathrm{tr}(\mathrm{Var}(v_t))} + \underbrace{\|\mathrm{E}(v_t)\|^2}_{\text{const.}} \tag{3.2}$$

A way to improve the SGD may be by reducing / controlling the variances at each dimension of v_t.

Problem 20. Consider a learning problem $(\mathcal{H}, \mathcal{Z}, \ell)$ with risk function $R_g(w) = \mathrm{E}_{z\sim g}(\ell(h_w, z))$ given an unknown data generating model $g(\cdot)$, using a known tractable loss $\ell(\cdot,\cdot)$, and hypothesis $h_w \in \mathcal{H}$. Consider a training data set $\mathcal{S}_n = \{z_i = (x_i, y_i); z_i \in \mathcal{Z}, i = 1, ..., n\}$. We need to find minimizer $w^* = \arg\min_w (R_g(w))$.

3.1. Stochastic gradient descent using (mini-batches).

Note 21. A way to reduce variance in (3.2) (and hence achieve faster convergence rate) is to consider a larger batch size $m = \left|\mathcal{J}^{(t)}\right|$. Note that

$$\mathrm{E}_{z\sim g}\|v_t - \mathrm{E}(v_t)\|^2 = \mathrm{E}_{z\sim g}\left\|\nabla_w \frac{1}{\left|\mathcal{J}^{(t)}\right|}\sum_{j\in\mathcal{J}^{(t)}}\ell(w, z_j) - \nabla_w R_g(w)\right\|^2 = \frac{\mathrm{E}_{z_j\sim g}\|\nabla_w \ell(w, z_j) - \nabla_w R_g(w)\|^2}{\left|\mathcal{J}^{(t)}\right|}$$

where to top part of the fraction is the gradient variance for SGD with $m = 1$. Using larger batch size $m = \left|\mathcal{J}^{(t)}\right|$ reduces the stochastic gradient variance and hence the convergence rate.

3.2. Stochastic Variance Reduced Gradient (SVRG).

Control variate methods.

Note 22. Control variate is a general way to perform variance reduction. Let $x \in \mathbb{R}^d$ be an unbiased estimator of θ. The goal is to compute $v = x + c(y - \mathrm{E}(y))$ by finding a suitable control variable $y \in \mathbb{R}^d$ and constant $c \in \mathbb{R}$ such that $\mathrm{tr}(\mathrm{Var}_c(v)) < \mathrm{tr}(\mathrm{Var}(x))$ with purpose to use it as an unbiased estimator of θ instead of x $\mathrm{E}_c(v) = \mathrm{E}(x) = \theta$.

Example 23. Let random variables $x \in \mathbb{R}^d$, and $y \in \mathbb{R}^d$. Let $v = x + c(y - \mathrm{E}(y))$ for some constant $c \in \mathbb{R}$. It is $\mathrm{E}_c(v) = \mathrm{E}(x)$. Also

$$\mathrm{E}_c\|v - \mathrm{E}_c(v)\|^2 = \mathrm{E}\|x - \mathrm{E}(x)\|^2 + c^2\mathrm{E}\|y - \mathrm{E}(y)\|^2 + 2c\mathrm{E}\langle x - \mathrm{E}(x), y - \mathrm{E}(y)\rangle$$

which is minimized for

$$0 = \frac{\mathrm{d}}{\mathrm{d}c}\mathrm{E}_c\|v - \mathrm{E}_c(v)\|^2\Big|_{c=c^*} \implies c^* = -\frac{\mathrm{E}\langle x - \mathrm{E}(x), y - \mathrm{E}(y)\rangle}{\mathrm{E}\|y - \mathrm{E}(y)\|^2}$$

with

$$\mathrm{E}_{c^*}\|v-\mathrm{E}_c(v)\|^2=\mathrm{E}\|x-\mathrm{E}(x)\|^2-\frac{\mathrm{E}\langle x-\mathrm{E}(x),y-\mathrm{E}(y)\rangle}{\mathrm{E}\|y-\mathrm{E}(y)\|^2} \tag{3.3}$$

Variance reduced methods.

Definition 24. A SGD algorithm with learning rate $\eta>0$ and general recursion

$$w^{(t+1)}=w^{(t)}-\eta_t v_t,\ t=1,2,...$$

where $\mathrm{E}\left(v_t|w^{(t)}\right)\in\partial f\left(w^{(t)}\right)$ is called variance reduced SGD if it presents smaller stochastic gradient variance compared to standard SGD and has the property

$$\mathrm{E}\left\|v_t-\nabla f\left(w^{(t)}\right)\right\|^2\rightarrow 0,\ \text{as}\, t\rightarrow\infty$$

where $\mathrm{E}(\cdot)$ is w.r.t. all random variables till iteration t.

Stochastic Variance Reduced Gradient (SVRG) .

Algorithm 25. *Stochastic Variance Reduced Gradient with learning rate $\eta_t>0$ for Problem 1.*

For $t=1,2,3,...$ iterate:

(1) *randomly draw an example $z_{j^{(t)}}$ indexed by $j^{(t)}\in\{1,...,n\}$.*

(2) *compute*

$$w^{(t+1)}=w^{(t)}-\eta_t v_t \tag{3.4}$$

where

$$v_t=\underbrace{\nabla\ell\left(w^{(t)},z_{j^{(t)}}\right)}_{=x}\underbrace{-}_{=c}\left(\underbrace{\nabla\ell\left(\tilde{w},z_{j^{(t)}}\right)}_{=y}-\frac{1}{n}\sum_{i=1}^{n}\nabla\ell\left(\tilde{w},z_i\right)\right) \tag{3.5}$$

.

(3) *if modulo$(t,\kappa)=0$, set $\tilde{w}=w^{(t)}$*

(4) *if a termination criterion is satisfied STOP*

Note 26. Prepend, we do not know c in (3.3). The variance of gradient v_t^{SVRG} is

$$\mathrm{E}_{z\sim g}\left\|v_t^{\mathrm{SVRG}}-\mathrm{E}\left(v_t^{\mathrm{SVRG}}\right)\right\|^2=\mathrm{E}_{z\sim g}\left\|\overbrace{\nabla_w\ell\left(w^{(t)},z_{j^{(t)}}\right)+c\left(\nabla_w\ell\left(\tilde{w},z_{j^{(t)}}\right)-\frac{1}{n}\sum_{i=1}^{n}\nabla_w\ell\left(\tilde{w},z_i\right)\right)}^{v_t^{\mathrm{SVRG}}=}-\nabla_w R_g\left(w^{(t)}\right)\right\|^2$$

$$\leq\mathrm{E}_{z\sim g}\left\|\nabla_w\ell\left(w^{(t)},z_{j^{(t)}}\right)+c\nabla_w\ell\left(\tilde{w},z_{j^{(t)}}\right)\right\|^2$$

As optimal c is often intractable, we sub-optimally set $c=-1$ and get

$$\mathrm{E}_{z\sim g}\left\|v_t^{\mathrm{SVRG}}-\mathrm{E}\left(v_t^{\mathrm{SVRG}}\right)\right\|^2\leq\mathrm{E}_{z\sim g}\left\|\nabla_w\ell\left(w^{(t)},z_{j^{(t)}}\right)-\nabla_w\ell\left(\tilde{w},z_{j^{(t)}}\right)\right\|^2$$

If $\ell(\cdot, z_j)$ is β_j-smooth, then

$$\mathrm{E}_{z\sim g}\left\|v_t^{\mathrm{SVRG}} - \mathrm{E}\left(v_t^{\mathrm{SVRG}}\right)\right\|^2 \leq \max_j\left(\{\beta_j\}\right)\left\|w^{(t)} - \tilde{w}\right\|^2$$

Hence to bound/control variance, point $\tilde{w}$ should be chosen to be as close as possible to $w^{(t)}$. Perhaps by taking a snapshot $\tilde{w} \leftarrow w^{(t)}$ every κ-th iteration.

Note 27. The frequency of the snapshots κ is specified by the researcher. The smaller the κ, the more frequent snapshots, and the more correlated the baseline y will be with the objective x in (3.5) and hence the bigger the performance improvement (See (3.3)); however the iterations will be slower.

Note 28. Iterations of SVRG are computationally faster than those of full GD but SVRG can still match the theoretical convergence rate of GD.

Example 29. [3] The SVRG with learning rate $\eta_t > 0$ and batch size $m = 1$ is

For $t = 1, 2, 3, ...$ iterate:

(1) randomly generate a set $\mathcal{J}^{(t)} = \left\{j^{(t)}\right\}$ by drawing one number $j^{(t)}$ from $\left\{1, ..., n = 10^6\right\}$.

(2) compute

$$w^{(t+1)} = \begin{bmatrix} w_1^{(t)} \\ w_2^{(t)} \end{bmatrix} - \eta_t\left(\begin{bmatrix} 2\left(w_1^{(t)} - \tilde{w}_1\right) + 2\left(w_2^{(t)} - \tilde{w}_2\right)x_{j^{(t)}} \\ 2\left(w_2^{(t)} - \tilde{w}_2\right)x_{j^{(t)}} + 2\left(w_2^{(t)} - \tilde{w}_2\right)\left(x_{j^{(t)}}\right)^2 \end{bmatrix} + \nabla\hat{R}_{\mathcal{S}}(\tilde{w})\right) \tag{3.6}$$

(3) if $\mathrm{modulo}(t, \kappa) = 0$, set $\tilde{w} = w^{(t)}$

(4) compute

$$\frac{1}{n}\sum_{i=1}^{n}\ell(\tilde{w}, z_i) = \begin{pmatrix} 2\tilde{w}_1 + 2\tilde{w}_2\bar{x} - 2\bar{y} \\ 2\tilde{w}_1\bar{x} + 2\tilde{w}_2\overline{x^2} - 2y^\top x \end{pmatrix} = \nabla\hat{R}_{\mathcal{S}}(\tilde{w}) \tag{3.7}$$

(5) if a termination criterion is satisfied STOP

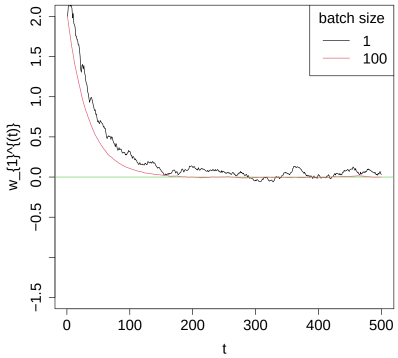
Because (3.7) is actually the gradient of the Risk function at $\tilde{w}$ and

$$\nabla\ell\left(w^{(t)}, z_{j^{(t)}}\right) - \nabla\ell\left(\tilde{w}, z_{j^{(t)}}\right) = \begin{pmatrix} 2\left(w_1^{(t)} - \tilde{w}_1\right) + 2\left(w_2^{(t)} - \tilde{w}_2\right)x_{j^{(t)}} \\ 2\left(w_2^{(t)} - \tilde{w}_2\right)x_{j^{(t)}} + 2\left(w_2^{(t)}\left(x_{j^{(t)}}\right)^2 - \tilde{w}_2\left(x_{j^{(t)}}\right)^2\right) \end{pmatrix}$$

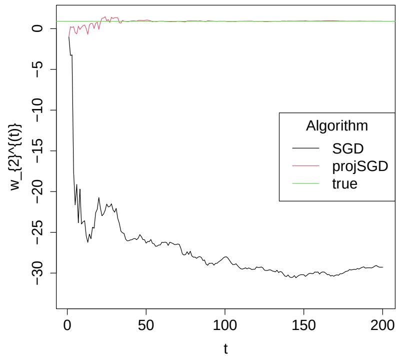
In Figure A.1c we observe the frequency of the snapshots has improved the convergence; however this is not a panacea as seen in Figure A.1f.

[3] `https://github.com/georgios-stats/Machine_Learning_and_Neural_Networks_III_Epiphany_2025/tree/main/Lecture_notes/code/example_SVRG.R`

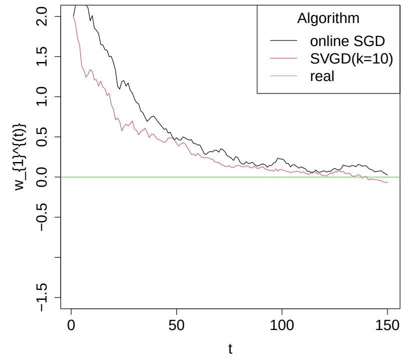
Appendix A. Plots



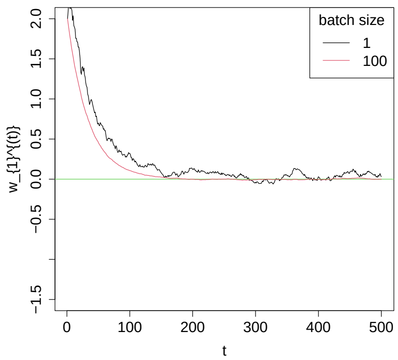
(A) batch SGD Example 30 w_1



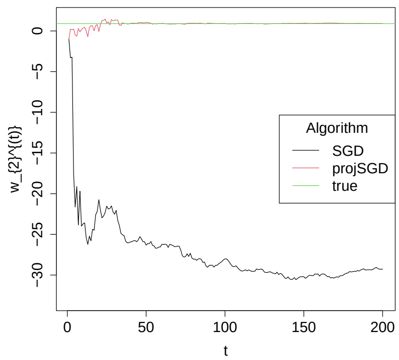
(B) online SGD with projection Example $6w_1$



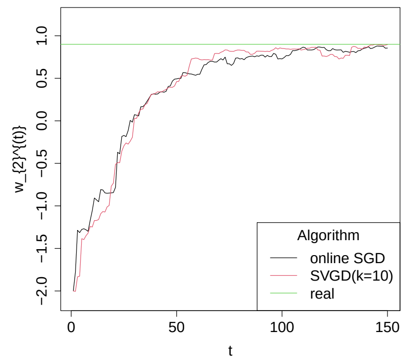
(C) SVRG Example 29 w_1



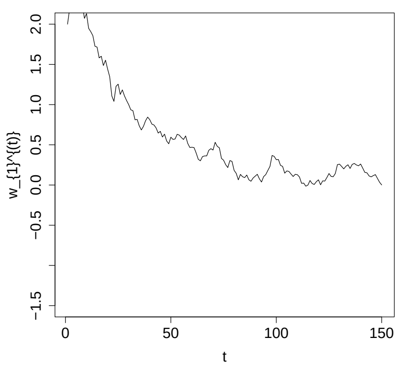
(D) batch SGD Example 30 w_2



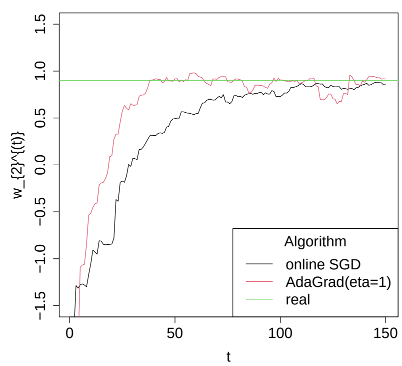
(E) online SGD with projection Example 6 w_2



(F) SVRG Example 29 w_2



(G) online AdaGrad Example $18w_1$



(H) online AdaGrad Example $18w_2$

Figure A.1. Simulations of the Examples

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2025

Lecture notes 5: Stochastic gradient descent

Lecturer & author: Georgios P. Karagiannis georgios.karagiannis@durham.ac.uk

Aim. To introduce the stochastic gradient descent (motivation, description, practical tricks, analysis in the convex scenario, and implementation).

Reading list & references:

(1) Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.
- Ch. 14.3 Stochastic Gradient Descent (SGD), 14.5 Variants, 14.5 Learning with SGD

(2) Bottou, L. (2012). Stochastic gradient descent tricks. In Neural networks: Tricks of the trade (pp. 421-436). Springer, Berlin, Heidelberg.

1. Motivations for stochastic gradient descent

Problem 1. Consider a learning problem $(\mathcal{H}, \mathcal{Z}, \ell)$. Learning may involve the computation of the minimizer $w^* \in \mathcal{W}$ of the risk function (RF) $R_g(w) = \mathrm{E}_{z\sim g}(\ell(h_w, z))$ given an unknown data generating model $g(\cdot)$, using a known tractable loss $\ell(\cdot,\cdot)$, and hypothesis $h_w \in \mathcal{H}$; that is

$$w^* = \arg\min_w (R_g(w)) = \arg\min_w (\mathrm{E}_{z\sim g}(\ell(h_w, z))) \tag{1.1}$$

Note 2. Gradient descent (GD) cannot be directly utilized to address Problem 1 (i.e., minimize the Risk function) because g is unknown, and because (1.1) may involve a computationally intractable integration. Instead GD aims to minimize the ERF $\hat{R}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(h_w, z_i)$ which ideally can be considered as a proxy when data size n is big (big-data).

Note 3. The implementation of GD may be computationally impractical even in problems aiming to minimize ERF $\hat{R}_n(w)$ if we have big data ($n \approx$big). This is because GD requires the recursive computation of the exact gradient $\partial_w \hat{R}_n(w) = \left\{\frac{1}{n}\sum_{i=1}^{n} v_i : v_i \in \partial_w \ell(h_w, z_i)\right\}$ using (required to scan through) all the data $\{z_i\}$ at each iteration. That may be too slow.

Note 4. Stochastic gradient descent (SGD) aims at solving (1.1), and overcoming the issues in Notes 2 & 3 by using an unbiased estimator of the actual gradient (or some sub-gradient) based on a sample (a single example or a set of examples) properly drawn from g.

2. Stochastic gradient descent

Problem 5. For the sake of notation simplicity and generalization, we present Stochastic Gradient Descent (SGD) in the following minimization problem

$$w^* = \arg\min_w (f(w)) \tag{2.1}$$

where here $f:\mathbb{R}^d\rightarrow\mathbb{R}$, and $w\in\mathcal{W}\subseteq\mathbb{R}^d$; $f(\cdot)$ is the unknown function to be minimized, e.g., $f(\cdot)$ can be the risk function $R_g(w)=\mathrm{E}_{z\sim g}(\ell(h_w,z))$.

Algorithm 6. *Stochastic Gradient Descent (SGD) with learning rate $\eta_t>0$ for Problem 5*

For $t=1,2,3,...$ iterate:

(1) *compute*

$$w^{(t+1)}=w^{(t)}-\eta_t v_t, \tag{2.2}$$

where v_t is a random vector such that $E\left(v_t|w^{(t)}\right)\in\partial f\left(w^{(t)}\right)$

(2) *terminate if a termination criterion is satisfied, e.g.*

If $t\geq T_{\max}$ then STOP

Note 7. If f is differentiable at $w^{(t)}$, it is $\partial f\left(w^{(t)}\right)=\left\{\nabla f\left(w^{(t)}\right)\right\}$. Hence v_t is such as $\mathrm{E}\left(v_t|w^{(t)}\right)=\nabla f\left(w^{(t)}\right)$ in Algorithm 6 step 1.

Note 8. Assume f is differentiable (for simplicity). To compare SGD with GD, we can re-write (2.2) in the SGD Algorithm 6 as

$$w^{(t+1)}=w^{(t)}-\eta_t\left[\nabla f\left(w^{(t)}\right)+\xi_t\right], \tag{2.3}$$

where

$$\xi_t:=v_t-\nabla f\left(w^{(t)}\right)$$

represents the (observed) noise introduced in (2.2) by using a random realization of the exact gradient.

Note 9. Given T SGD algorithm iterations, the output of SGD can be (but not a exclusively)

(1) the average (after discarding the first few iterations of $w^{(t)}$ for stability reasons)

$$w_{\text{SGD}}^{(T)}=\frac{1}{T}\sum_{t=1}^{T}w^{(t)} \tag{2.4}$$

(2) or the best value discovered

$$w_{\text{SGD}}^{(T)}=\arg\min_{\{w_t\}}\left(f\left(w^{(t)}\right)\right)$$

(3) or the last value discovered

$$w_{\text{SGD}}^{(T)}=w^{(T)}$$

Note 10. SGD output converges to a local minimum, $w_{\text{SGD}}^{(T)}\rightarrow w_*$ (in some sense), under different sets of regularity conditions –Section 3 has a brief analysis. To achieve this, Conditions 11 on the learning rate are rather inevitable and should be satisfied.

Condition 11. Regarding the learning rate (or gain) $\{\eta_t\}$ should satisfy conditions

(1) $\eta_t\geq 0$,

(2) $\sum_{t=1}^{\infty}\eta_t=\infty$
(3) $\sum_{t=1}^{\infty}\eta_t^2<\infty$

Note 12. The popular learning rates $\{\eta_t\}$ in Note 14 in Lect. notes 4 "Gradient descent" satisfy Condition 11 and hence can be used in SGD too. Once parameterized, η_t can be tuned based on pilot runs using a reasonably small fraction of the training data set.

Note 13. (Intuition on Condition 11). Assume that v_t is bounded. Condition 11(3) aims at reducing the effect of the randomness in v_t (introduced noise ξ_t) because it implies $\eta_t \searrow 0$ as $t\to\infty$; if this was not the case then

$$w^{(t+1)}-w^{(t)}=-\eta_t v_t\to 0$$

may not be satisfied and the chain $\left\{w^{(t)}\right\}$ may not converge. Condition 11(2) prevents η_t from reducing too fast and allows the generated chain $\left\{w^{(t)}\right\}$ to be able to converge. E.g., after t iterations

$$\begin{aligned}\left\|w^{(t)}-w^*\right\| &= \left\|w^{(t)}\pm w^{(0)}-w^*\right\| \geq \left\|w^{(0)}-w^*\right\|-\left\|w^{(t)}-w^{(0)}\right\| \\ &\geq \left\|w^{(0)}-w^*\right\|-\sum_{t=0}^{\infty}\left\|w^{(t+1)}-w^{(t)}\right\| = \left\|w^{(0)}-w^*\right\|-\sum_{t=0}^{T-1}\left\|\eta_t v_t\right\|\end{aligned}$$

However if it was $\sum_{t=1}^{\infty}\eta_t<\infty$ it would be $\sum_{t=0}^{\infty}\|\eta_t v_t\|<\infty$ and hence $w^{(t)}$ would never converge to w^* if the seed $w^{(0)}$ is far enough from w^*.

3. Analysis of SGD (Algorithm 6)

Note 14. Recall (Note 8) that the stochasticity of SGD comes from the stochastic sub-gradients $\{v_t\}$ in (2.2); hence the expectations below are under these random vectors' distributions.

Proposition 15. *Let $f(\cdot)$ be a convex function. If we run SGD algorithm of f with learning rate $\eta_t>0$ for T steps, the output $w_{\text{SGD}}^{(T)}=\frac{1}{T}\sum_{t=1}^{T}w^{(t)}$* satisfies

$$\mathrm{E}\left(f\left(w_{\text{SGD}}^{(T)}\right)\right)-f\left(w^*\right)\leq\frac{\left\|w^*\right\|^2}{2\eta T}+\frac{\eta}{2}\frac{1}{T}\sum_{t=1}^{T}\mathrm{E}\left\|v_t\right\|^2 \tag{3.1}$$

Proof. Let $v_{1:t}=(v_1,...,v_t)$. By Jensens' inequality (or see 4.3 in Lect. notes 4)

$$\mathrm{E}\left(f\left(w_{\text{SGD}}^{(T)}\right)-f\left(w^*\right)\right)\leq\mathrm{E}\left(\frac{1}{T}\sum_{t=1}^{T}\left(f\left(w^{(t)}\right)-f\left(w^*\right)\right)\right)=\frac{1}{T}\sum_{t=1}^{T}\mathrm{E}\left(f\left(w^{(t)}\right)-f\left(w^*\right)\right) \tag{3.2}$$

I will try to use Lemma 17 from Lect. notes 4, hence I need to show

$$\mathrm{E}\left(f\left(w^{(t)}\right)-f\left(w^*\right)\right)\leq\mathrm{E}\left(\langle w^{(t)}-w^*,v_t\rangle\right) \tag{3.3}$$

where the expectation is under $v_{1:T}$. It is

$$\begin{aligned}\mathrm{E}_{v_{1:T}}\left(\langle w^{(t)}-w^*,v_t\rangle\right) =&\mathrm{E}_{v_{1:t}}\left(\langle w^{(t)}-w^*,v_t\rangle\right)\\ =&\mathrm{E}_{v_{1:t-1}}\left(\mathrm{E}_{v_{1:t}}\left(\langle w^{(t)}-w^*,v_t\rangle|v_{1:t-1}\right)\right) \qquad \text{(law of total expectation)}\end{aligned}$$

But $w^{(t)}$ is fully determined by $v_{1:t-1}$, (see (2.2)) so

$$\mathrm{E}_{v_{1:t-1}}\left(\mathrm{E}_{v_{1:t}}\left(\langle w^{(t)}-w^{*},v_{t}\rangle|v_{1:t-1}\right)\right)=\mathrm{E}_{v_{1:t-1}}\left(\langle w^{(t)}-w^{*},\mathrm{E}_{v_{1:t}}\left(v_{t}|v_{1:t-1}\right)\rangle\right)$$

As $w^{(t)}$ is fully determined by $v_{1:t-1}$ then $\mathrm{E}_{v_{1:t}}\left(v_t|v_{1:t-1}\right)=\mathrm{E}_{v_{1:t}}\left(v_t|w^{(t)}\right)\in\partial f\left(w^{(t)}\right)$, hence $\mathrm{E}_{v_{1:t}}\left(v_t|v_{1:t-1}\right)$ is a sub-gradient. By sub-gradient definition

$$\begin{aligned}\mathrm{E}_{v_{1:t-1}}\left(\langle w^{(t)}-w^{*},\mathrm{E}_{v_{1:t}}\left(v_{t}|v_{1:t-1}\right)\rangle\right)&\geq\mathrm{E}_{v_{1:t-1}}\left(f\left(w^{(t)}\right)-f\left(w^{*}\right)\right)\\&=\mathrm{E}_{v_{1:T}}\left(f\left(w^{(t)}\right)-f\left(w^{*}\right)\right)\end{aligned}\tag{3.4}$$

Hence combining (3.4), (3.3), and (3.2)

$$\mathrm{E}\left(f\left(w_{\mathrm{SGD}}^{(T)}\right)-f\left(w^{*}\right)\right)\leq\frac{1}{T}\sum_{t=1}^{T}\mathrm{E}\left(\langle w^{(t)}-w^{*},v_{t}\rangle\right)$$

Then Lemma 17 in Lect. notes 4 “Gradient descent” implies

$$\mathrm{E}\left(f\left(w_{\mathrm{SGD}}^{(T)}\right)-f\left(w^{*}\right)\right)\leq\mathrm{E}\left(\frac{1}{T}\frac{\|w^{*}\|^{2}}{2\eta}+\frac{\eta}{2}\frac{1}{T}\sum_{t=1}^{T}\|v_{t}\|^{2}\right)=\frac{\mathrm{E}\|w^{*}\|^{2}}{2\eta T}+\frac{\eta}{2}\frac{1}{T}\sum_{t=1}^{T}\mathrm{E}\|v_{t}\|^{2}$$

□

Note 16. The upper bound in (3.1) depends on $\mathrm{E}\|v_t\|^2$ which has to be bounded and hence the variation of v_t because

$$\mathrm{E}\|v_{t}\|^{2}=\mathrm{E}\left(\|v_{t}-\mathrm{E}\left(v_{t}\right)\|^{2}\right)+\|\mathrm{E}\left(v_{t}\right)\|^{2}=\mathrm{tr}\left(\mathrm{Var}\left(v_{t}\right)\right)+\|\mathrm{E}\left(v_{t}\right)\|^{2}\tag{3.5}$$

where $\mathrm{E}\left(\|v_t-\mathrm{E}\left(v_t\right)\|^2\right)=\mathrm{tr}\left(\mathrm{Var}\left(v_t\right)\right)$ is the sum of all the variances at each dimension of v_t and $\|\mathrm{E}\left(v_t\right)\|^2$ is a finite constant as $\mathrm{E}\left(v_t\right)\in\partial_w f\left(w_t\right)$ is the unbiased estimator of the sub-gradient by construction.

Proposition 17. *(Cont. Proposition 15) Let $f(\cdot)$ be a convex function over the set $\mathcal{W}=\{w:\|w\|\leq B\}$. Let $E\|v_t\|^2\leq\rho^2$. Assume we run SGD algorithm of $f(\cdot)$ with learning rate $\eta_t=\sqrt{\frac{B^2}{\rho^2 T}}$ for T steps, and output $w_{\mathrm{SGD}}^{(T)}=\frac{1}{T}\sum_{t=1}^{T}w^{(t)}$. Then*

(1) *upper bound on the sub-optimality is*

$$\mathrm{E}\left(f\left(w_{\mathrm{SGD}}^{(T)}\right)\right)-f\left(w^{*}\right)\leq\frac{B\rho}{\sqrt{T}}\tag{3.6}$$

(2) *a given level of accuracy ε such that $E\left(f\left(w_{SGD}^{(T)}\right)\right)-f\left(w^{*}\right)\leq\varepsilon$ can be achieved after Titerations*

$$T\geq\frac{B^{2}\rho^{2}}{\varepsilon^{2}}.$$

Proof. It follows from Proposition 15. Condition $\mathrm{E}\|v_t\|^2\leq\rho^2$ can be achieved, for instance by assuming Lipschitz (See Lemma 21 from “Lect. notes 4: Gradient descent”). □

4. Implementation of SGD in the learning Problem 1

Note 18. Note 19 implements SGD to the learning problem (Problem 1)

$$w^{*}=\arg\min_{w}\left(R_{g}\left(w\right)\right)=\arg\min_{w}\left(\mathrm{E}_{z\sim g}\left(\ell\left(h_{w},z\right)\right)\right)$$

Note 19. For a randomly drawn example $z\sim g(\cdot)$, the sub-gradient v of $\ell\left(w,z\right)$ at point w is an unbiased estimator of the sub-gradient of the risk $R_{g}\left(w\right)$ at point w. I.e. if $v\in\partial_{w}\ell\left(w,z\right)$ where $z\sim g(\cdot)$ then $\mathrm{E}_{z\sim g}\left(v|w\right)\in\partial_{w}R_{g}\left(w\right)$.

Proof. Let v be a sub-gradient of $\ell\left(w,z\right)$ at point w, then

$$\ell\left(u,z\right)-\ell\left(w,z\right)\geq\left\langle u-w,v\right\rangle \tag{4.1}$$

It is

$$\begin{aligned}R_{g}\left(u\right)-R_{g}\left(w\right)&=\mathrm{E}_{z\sim g}\left(\ell\left(u,z\right)-\ell\left(w,z\right)|w\right)\geq\mathrm{E}_{z\sim g}\left(\left\langle u-w,v\right\rangle|w\right)\\&=\left\langle u-w,\mathrm{E}_{z\sim g}\left(v|w\right)\right\rangle\end{aligned}$$

Hence, by definition, v is such that $\mathrm{E}_{z\sim g}\left(v|w\right)$ is a sub-gradient of $R_{g}\left(w\right)$. □

Note 20. Similarly, the average $\bar{v}=\frac{1}{m}\sum_{i=1}^{m}v_{i}$ of sub-gradients $v_{i}\in\partial_{w}\ell\left(w,\tilde{z}_{i}\right)$ at point w given a randomly drawn set of m examples $\{\tilde{z}_{i}\sim g(\cdot);\,i=1,...,m\}$ is an unbiased estimator of the risk function sub-gradient; i.e. $\mathrm{E}_{z\sim g}\left(\bar{v}|w\right)\in\partial_{w}R_{g}\left(w\right)$.

Example 21. Assume a differentiable loss function $\ell\left(\cdot,z\right)$ at each z and set sub-gradient $v=\nabla_{w}\ell\left(w,z\right)$, then

$$\mathrm{E}_{z\sim g}\left(v|w\right)=\mathrm{E}_{z\sim g}\left(\nabla_{w}\ell\left(h_{w},z\right)|w\right)=\nabla_{w}\mathrm{E}_{z\sim g}\left(\ell\left(h_{w},z\right)|w\right)=\nabla_{w}R_{g}\left(w\right)$$

Note 22. Assume there is available a finite dataset $\mathcal{S}_{n}=\{z_{i};i=1,...,n\}$ of size n which consists of independent realizations z_{i} from the data generating distribution g; $z_{i}\overset{\text{ind}}{\sim}g$. SGD (Algorithm 23) is an implementation of the SGD (Algorithm 6) in the learning Problem 1.

Algorithm 23. *Stochastic Gradient Descent with learning rate $\eta_{t}>0$, and batch size m, for Problem 1.*

For $t=1,2,3,...$ iterate:

1. *randomly generate a set $\mathcal{J}^{(t)}\subseteq\{1,...,n\}^{m}$ of m indices from 1 to n with or without replacement.*
2. *compute*

$$w^{(t+1)}=w^{(t)}-\eta_{t}v_{t}, \tag{4.2}$$

   where $v_{t}=\frac{1}{\left|\mathcal{J}^{(t)}\right|}\sum_{j\in\mathcal{J}^{(t)}}\nu_{t,j}$ and $\nu_{t,j}\in\partial_{w}\ell\left(w^{(t)},z_{j}\right)$ for any $j\in\mathcal{J}^{(t)}$.

3. *terminate if a termination criterion is satisfied*

Note 24. If it is possible to sample anytime fresh examples z_i directly from the data generation model g instead of just having access to only a given finite dataset of examples $\mathcal{S}_n$, then step 1 in Algorithm 23 can become

(1) sample $\tilde{z}_j^{(t)} \sim g(\cdot)$ for $j = 1, ..., m$.

Note 25. Algorithm 23 may be called online SGD when using sub-samples of size one ($m = 1$) and batch SGD when using sub-samples of larger sizes ($m > 1$) .

Note 26. In theory, using larger batch size m has the benefit that reduces the variance of v_t at iteration t due to averaging effect, stabilizes the SGD algorithm, and reduces the error bound (3.1); see Remark 16.

Note 27. In practice, for a given fixed computational time, using smaller batch size m has the benefit that the algorithm iterates faster as each iteration processes less number of examples. E.g., consider the extreme cases GD vs online SGD utilized in a scenario of big-data (large training data set): if the dataset consists of several replications of the same values, GD (using all the data) has to process the same information multiple times, while the online SGD (using only one example at a time) would avoid this issue.

Note 28. In practice, for a given fixed computational time, it is possible for a SGD with smaller batch size m to present better generalization properties (wrt the theoretical assumptions) than those with larger m (GD is included). It is observed for the former to be often less prone to getting stuck in shallow local minima because of the additional amount of "noise" E.g., consider the extreme cases GD ($\mathcal{J}^{(t)} \equiv \{1, ..., n\}$) vs online SGD ($m = 1$) in a scenario with non-convex risk function (e.g. our theoretical assumptions as violated): if the Risk function presents local minima, considering less examples randomly chosen each time may cause fluctuations in the gradient that allow the chain to accidentally jump/escape to an area with a lower minimum.

Proposition 29. *(Implementing Proposition 17) Consider a convex-Lipschitz-bounded problem* $(\mathcal{H}, \mathcal{Z}, \ell)$ *with parameters* ρ *and* B. *Then for every* $\epsilon > 0$, *if we run SGD Algorithm 6 to minimize Problem 1 for* $T \geq \frac{B^2}{\epsilon^2}\rho^2$ *iterations then it produces output* $w_{SGD}^{(T)}$ *satisfying*

$$E\left(R_g\left(w_{SGD}^{(T)}\right)\right) \leq \min_{\forall w \in \mathcal{H}} \left(R_g\left(w\right)\right) + \epsilon$$

and hence we can achieve PAC guarantees.

Proof. It is just re-writing the formula in part 2 of Proposition 17. Notice that here, condition $\mathrm{E}\left\|v_t\right\|^2 \leq \rho^2$ is satisfied by the self-bounding property from Lipschitzness of the loss (see Lemma 21 from "Lect. notes 4: Gradient descent"); i.e. if ℓ is ρ-Lipschitz, then $\|v_t\| \leq \rho$ where $\nu_t \in \partial_w \ell\left(w^{(t)}, z_j\right)$ at $j \in \mathcal{J}^{(t)}$. $\square$

Example 30. [1] We continue Example 26 in Section 4 of Lect. notes 4. Recall, we considered a hypothesis space $\mathcal{H}$ of linear functions $h : \mathbb{R}^2 \rightarrow \mathbb{R}$ with $h\left(w\right) = w_1 + w_2 x$, $w = \left(w_1, w_2\right)^\top$,

[1] https://github.com/georgios-stats/Machine_Learning_and_Neural_Networks_III_Epiphany_2025/tree/main/Lecture_notes/code/example_SGD.R

and $\ell\left(w, z=(x, y)^{\top}\right)=(y_i-w_1-w_2 x)^2$. Here we consider a big dataset $\mathcal{S}_n=\{z_1, \ldots, z_n\}$ with $n=10^6$ examples.

The batch SGD algorithm (Algorithm 23) with learning rate η_t and batch size $m=10$ is
For $t=1,2,3, \ldots$ iterate:

(1) Randomly generate a set $\mathcal{J}^{(t)}$ by drawing $m=10$ numbers from $\left\{1, \ldots, n=10^6\right\}$
(2) compute

$$w^{(t+1)}=w^{(t)}-\eta_t v_t,$$

where

$$v_t=\begin{pmatrix} 2 w_1^{(t)}+2 w_2^{(t)} \frac{1}{m} \sum_{j \in \mathcal{J}^{(t)}} x_j-2 \frac{1}{m} \sum_{j \in \mathcal{J}^{(t)}} y_j \\ 2 w_1^{(t)} \bar{x}+2 w_2^{(t)} \frac{1}{m} \sum_{j \in \mathcal{J}^{(t)}} x_j^2-2 \sum_{j \in \mathcal{J}^{(t)}} y_j x_j \end{pmatrix}$$

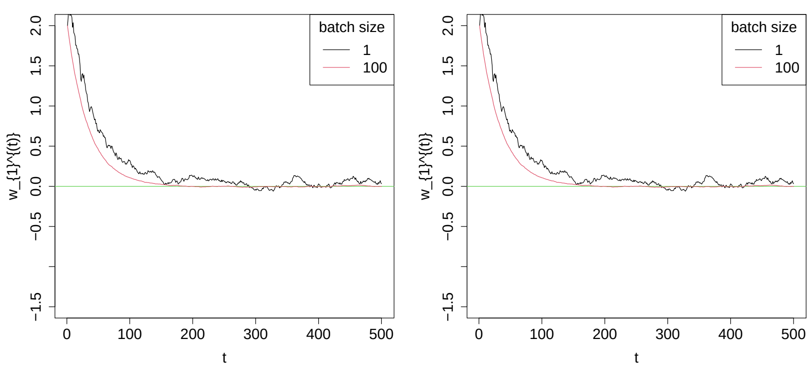
(3) if $t \geq T=1000$ STOP

because

$$v_t=\frac{1}{m} \sum_{j \in \mathcal{J}^{(t)}} \nabla_w \ell\left(w^{(t)}, z_j\right)=\ldots=\begin{pmatrix} 2 w_1^{(t)}+2 w_2^{(t)} \frac{1}{m} \sum_{j \in \mathcal{J}^{(t)}} x_j-2 \frac{1}{m} \sum_{j \in \mathcal{J}^{(t)}} y_j \\ 2 w_1^{(t)} \bar{x}+2 w_2^{(t)} \frac{1}{m} \sum_{j \in \mathcal{J}^{(t)}} x_j^2-2 \sum_{j \in \mathcal{J}^{(t)}} y_j x_j \end{pmatrix}$$

In Figures A.1a & A.1b, we observe that increasing the batch size has improved the convergence however this is not a panacea. Also it had reduced the oscillations of chain $\left\{w^{(t)}\right\}$.

Appendix A. Plots



(A) SGD for w_1 (B) SGD for w_2

Figure A.1. Simulations of the Example 30

Lecture notes 7: Bayesian Learning via Stochastic gradient and Stochastic gradient Langevin dynamics

Lecturer & author: Georgios P. Karagiannis georgios.karagiannis@durham.ac.uk

Aim. To introduce the Bayesian Learning, and Stochastic gradient Langevin dynamics (description, heuristics, and implementation). Stochastic gradient descent will be implemented in the Bayesian learning too.

Reading list & references:

(1) Welling, M., & Teh, Y. W. (2011). Bayesian learning via stochastic gradient Langevin dynamics. In Proceedings of the 28th international conference on machine learning (ICML-11) (pp. 681-688).
(2) Bottou, L. (2012). Stochastic gradient descent tricks. In Neural networks: Tricks of the trade (pp. 421-436). Springer, Berlin, Heidelberg.
(3) Sato, I., & Nakagawa, H. (2014). Approximation analysis of stochastic gradient Langevin dynamics by using Fokker-Planck equation and Ito process. In International Conference on Machine Learning (pp. 982-990). PMLR.
(4) Teh, Y. W., Thiery, A. H., & Vollmer, S. J. (2016). Consistency and fluctuations for stochastic gradient Langevin dynamics. Journal of Machine Learning Research, 17.
(5) Vollmer, S. J., Zygalakis, K. C., & Teh, Y. W. (2016). Exploration of the (non-) asymptotic bias and variance of stochastic gradient Langevin dynamics. The Journal of Machine Learning Research, 17(1), 5504-5548. −>further reading for theory
(6) Nemeth, C., & Fearnhead, P. (2021). Stochastic gradient Markov chain Monte Carlo. Journal of the American Statistical Association, 116(533), 433-450.

1. Bayesian learning and motivations

Note 1. Bayesian methods are appealing in their ability to capture uncertainty in learned parameters and avoid overfitting. Arguably with large datasets there will be little overfitting. Alternatively, as we have access to larger datasets and more computational resources, we become interested in building more complex models (eg from a logistic regression to a deep neural network), so that there will always be a need to quantify the amount of parameter uncertainty.

Note 2. Consider a Bayesian statistical model with sampling distribution (statistical model) $f(z|w)$ labeled by an unknown parameter $w \in \mathcal{W} \subseteq \mathbb{R}^d$ that follows a prior distribution $f(w)$. Assume a dataset $\mathcal{S}_n = \{z_i; i = 1, ..., n\}$ of size n containing independently drawn examples. Let $L_n(w) := f(z_{1:n}|w)$ denote the likelihood of the observables $\{z_i \in \mathcal{Z}\}_{i=1}^{n}$ give parameter w. The Bayesian model is denoted as

$$\begin{cases} z_i|w & \overset{\text{ind}}{\sim} f\left(z_i|w\right), \;\; i=1,...,n \\ w & \sim f\left(w\right) \end{cases} \tag{1.1}$$

Note 3. With regards to the Bayesian model in Note 2, we denote the likelihood of the observables $\{z_i \in \mathcal{Z}\}_{i=1}^{n}$ given the parameter w as

$$L_n\left(w\right) := f\left(z_{1:n}|w\right) = \prod_{i=1}^{n} f\left(z_i|w\right)$$

Note 4. Bayesian learning (inference) relies on the posterior distribution density

$$f\left(w|z_{1:n}\right) = \frac{L_n\left(w\right) f\left(w\right)}{\int L_n\left(w\right) f\left(w\right) \mathrm{d}w} \tag{1.2}$$

which quantifies the researcher's belief (or uncertainty) about the unknown parameter w learned given examples $\{z_i \in \mathcal{Z}\}_{i=1}^{n}$. It is often intractable; hence there is often a need to numerically compute it. (Section 3)

Note 5. Point estimation of a function h of w is often performed via computation of the posterior expectation w given the examples in $\mathcal{S}_n$

$$\mathrm{E}_f\left(h\left(w\right)|z_{1:n}\right) = \int h\left(w\right) f\left(w|z_{1:n}\right) \mathrm{d}w \tag{1.3}$$

It is often intractable; hence there is often a need to numerically compute it. (Section 3)

Note 6. Point estimation of w can also be performed via maximum a-posteriori (MAP) estimator w^* of w that is the maximizer w^* of (1.2) i.e.

$$w^* = \arg\max_{w}\left(f\left(w|z_{1:n}\right)\right) \tag{1.4}$$

$$= \arg\max_{w}\left(\underbrace{\log\left(L_n\left(w\right)\right)}_{\text{(I)}} + \underbrace{\log\left(f\left(w\right)\right)}_{\text{(II)}}\right) \tag{1.5}$$

Note that (I) may be interpreted as an empirical risk function, and (II) can be interpreted as a shrinkage term in terms of shrinkage methods (like LASSO, Ridge). It is often intractable; hence there is often a need to numerically compute it. (Section 2)

Note 7. To describe the learning algorithms Gradient Descent (GD), Stochastic Gradient Descent (SGD), and Stochastic Gradient Langevin Dynamics (SGLD), we consider that the examples (data) in (1.1) are independent realizations from the sampling distribution i.e. $z_i|w \sim f\left(\cdot|w\right)$ for $i=1,...,n$ and that w is continuous (not discrete).

Note 8. In what follows, we first present the implementation of GD and SGD addressing MAP learning, and then we introduce the implementation of SGLD addressing posterior density and expectation learning.

2. Maximum A Posteriori (MAP) learning via GD and SGD

Problem 9. Given the Bayesian model (1.1), and rearranging (1.4), MAP estimate w^* of w can be computed as

$$w^* = \arg\max_w \left(\log\left(L_n\left(w\right)\right) + f\left(w\right)\right) = \arg\max_w \left(\sum_{i=1}^{n} \log\left(f\left(z_i|w\right)\right) + \log\left(f\left(w\right)\right)\right) \tag{2.1}$$

Note 10. GD is particularly suitable to solve (2.1) when w has high dimensionality.

Algorithm 11. *Gradient descent implementation in Problem 9 with learning rate $\eta_t \geq 0$.*

For $t = 1, 2, 3, ...$ iterate:

(1) *Compute*

$$w^{(t+1)} = w^{(t)} + \eta_t \left(\sum_{i=1}^{n} \nabla_w \log\left(f\left(z_j|w^{(t)}\right)\right) + \nabla_w \log\left(f\left(w^{(t)}\right)\right)\right) \tag{2.2}$$

Note 12. SGD is particularly suitable to solve (2.1) when w has high dimensionality, and in big-data problems since the repetitive computation of the sum in (2.2) is prohibitively expensive. Yet consider the benefits of SGD against GD as discussed in (Notes 27 and 28 of Lect. Notes 5).

Algorithm 13. *Stochastic Gradient Descent implementation in with learning rate $\eta_t \geq 0$ and batch size m*

For $t = 1, 2, 3, ...$ iterate:

(1) *generate a random set $\mathcal{J}^{(t)} \subseteq \{1, ..., n\}^m$ of m indices from 1 to n with or without replacement*
(2) *compute*

$$w^{(t+1)} = w^{(t)} + \eta_t \left(\frac{n}{m} \sum_{j \in \mathcal{J}^{(t)}} \nabla_w \log\left(f\left(z_j|w^{(t)}\right)\right) + \nabla_w \log\left(f\left(w^{(t)}\right)\right)\right) \tag{2.3}$$

Note 14. Recursion (2.3) is justified in terms of SGD theory as

$$\mathrm{E}_{\mathcal{J}^{(t)} \sim \text{simple-random-sampling}} \left(\frac{n}{m} \sum_{j \in \mathcal{J}^{(t)}} \nabla_w \log\left(f\left(z_j|w^{(t)}\right)\right)\right) = \sum_{i=1}^{n} \nabla_w \log\left(f\left(z_i|w^{(t)}\right)\right) \tag{2.4}$$

see Exercise 15 from the Exercise sheet.

Note 15. The implementation of the SGD variants (Algorithms 23, 25, 10, 15 in Handout 3)) is straightforward based on Algorithm 13 and (2.4).

3. Fully Bayesian learning via SGLD

Problem 16. Fully Bayesian learning, computationally, is the problem of recovering the posterior distribution $f\left(w|z_{1:n}\right)$ of w given $z_{1:n}$ that admits density (1.2). For a given Bayesian model (1.1), the Bayesian estimator of $h := h(w)$ is often computed as the posterior expectation of $h(w)$ given the data $\mathcal{S}_n$

$$\mathrm{E}_f\left(h(w)\,|z_{1:n}\right) = \int h(w)\, f(w|z_{1:n})\,\mathrm{d}w \tag{1.3}$$

Note 17. Monte Carlo integration aims at approximating (1.3) when intractable, by using Central Limit Theorem or Law of Large Numbers arguments as $\hat{h}_T \approx \mathrm{E}_f\left(h(w)\,|z_{1:n}\right)$ where

$$\hat{h}_T = \frac{1}{T}\sum_{t=1}^{T} h\left(w^{(t)}\right) \tag{3.1}$$

where $\left\{w^{(t)}\right\}$ are T simulations drawn (approximately) from the posterior distribution (1.2). This theory is subject to conditions we skip.

Note 18. Stochastic gradient Langevin dynamics (SGLD) algorithm generates as output a random chain $\left\{w^{(t)}\right\}$ approximately distributed according to a distribution admitting density such as

$$f_\tau(w|z_{1:n}) \propto \exp\left(\frac{1}{\tau}\left(\log L_n(w) + \log f(w)\right)\right) \tag{3.2}$$

$$\propto \exp\left(\frac{1}{\tau}\left(\sum_{i=1}^{n}\log\left(f(z_i|w)\right) + \log\left(f(w)\right)\right)\right) \tag{3.3}$$

for any $\tau > 0$ under regularity conditions. That allows to recover the whole posterior distribution (1.2) (hence account for uncertainty in $h = h(w)$) and approximate posterior expectations based on the Monte Carlo integration (Note 17).

Algorithm 19. *Stochastic Gradient Langevin Dynamics (SGLD) with learning rate $\eta_t > 0$, batch size m, and temperature $\tau > 0$ is* Ref [1]

---

1. *Generate a random set $\mathcal{J}^{(t)} \subseteq \{1, ..., n\}^m$ of m indices from 1 to n with or without replacement*
2. *Compute*

$$w^{(t+1)} = w^{(t)} + \eta_t\left(\frac{n}{m}\sum_{i\in J^{(t)}}\nabla\log f\left(z_i|w^{(t)}\right) + \nabla\log f\left(w^{(t)}\right)\right) + \sqrt{2\eta_t\tau}\epsilon_t \tag{3.4}$$

where $\epsilon_t \overset{iid}{\sim} N(0, I)$.

3. *Terminate if a termination criterion is satisfied; E.g., $t \geq T_{max}$ for a prespecified $T_{max} > 0$.*

How / why it works...

Note 20. (Mathematically speaking) The stochastic chain in (3.4) can be viewed as a discretization of the continuous-time Langevin diffusion described by the stochastic differential equation Ref [3]

$$\mathrm{d}W(t) = -\nabla_w\left[-\log f\left(W(t)\,|z_{1:n}\right)\right]\mathrm{d}t + \sqrt{2\tau}\mathrm{d}B(t)\,,\ t \geq 0 \tag{3.5}$$

where $\{B(t)\}$ is a standard Brownian motion[1] in $\mathbb{R}^d$ (i.e.). Under suitable assumptions on f, it can be shown that a Gibbs distribution with PDF such as

$$f^{*}(w|z_{1;n}) \propto \exp\left(-\frac{1}{\tau}\left[-\log f(w|z_{1:n})\right]\right) \tag{3.6}$$

is the unique invariant distribution of (3.5), and that the distributions of $W(t)$ converge rapidly to f^{*} as $t \to \infty$.

Note 21. (Heuristically speaking) SGLD relies on injecting the 'right' amount of noise (3.4) to a standard stochastic gradient optimization recursion (2.2), such that, as the stepsize η_t properly reduces, the produced chain $\{w^{(t+1)}\}$ converges to samples that could have been drawn from distribution with density (3.2). In the initial phase of running, the stochastic gradient noise will dominate the injected noise ϵ_t and the algorithm will imitate an efficient SGD Algorithm 11 -but this is until η_t or $\nabla \log(L_n(w))$ become small enough. In the later phase of running, the injected noise ϵ_t will dominate the stochastic gradient noise, so the SGLD will imitate a Langevin dynamics for the target distribution (1.2). The aim is for to the algorithm to transition smoothly between the two phases. Whether the algorithm is in the stochastic optimization phase or Langevin dynamics phase depends on the variance of the injected noise versus that of the stochastic gradient.

Note 22. One can argue that, the output of SGLD is also an "almost" maximizer of the (minus) empirical risk with shrinkage for large enough t. A draw from the Gibbs distribution (3.6) is approximately a maximizer of (2.1). Also one can show that the SGLD recursion tracks the Langevin diffusion (3.5) in a suitable sense. Hence, both imply that the distributions of $W(t)$ will be close to the Gibbs distribution (3.6) for all sufficiently large t.

About the learning rate alg. parameter...

Note 23. To guarantee the algorithm to work it is important for the step sizes η_t to decrease to zero, so that the mixing rate of the algorithm will slow down with increasing number of iterations t. Then, we can keep the step size η_t constant once it has decreased below a critical level.

Condition 24. Regarding the learning rate (or gain) $\{\eta_t\}$ should satisfy the following conditions in order for SGLD Algorithm 19 to generate an output $\{w^{(t)}\}$ approximately distributed according to (3.2)

(1) $\eta_t \geq 0$,
(2) $\sum_{t=1}^{\infty} \eta_t = \infty$
(3) $\sum_{t=1}^{\infty} \eta_t^2 < \infty$

Note 25. The popular learning rates $\{\eta_t\}$ in Note 12 in Lect. Notes 4 can be used. . Once parametrized, η_t can be tuned based on pilot runs using a reasonably small number of data.

[1] A a continuous-time stochastic process: (1) $B(0) = 0$; (2) $B(t)$ is almost surely continuous ; (3) $B(t)$ has independent increaments ; (4) $B(t) - B(s) \sim \mathrm{N}(0, t-s)$ for $0 \leq s \leq t$.

About the temperature alg. parameter...

Note 26. The temperature parameter $\tau > 0$ is user specified and aims at controlling (eg; inflating) the variance of the produced chain for instance with practical purpose to escape from local modes (otherwise energy barriers) in non-convex problems.

About the output of the alg...

Note 27. The first few iterations of SGLD (Algorithm 19) involve values generated at the beginning of the running algorithm while the chain have not yet converged to (or reached) an area of substantial posterior mass. Hence they are discarded from the output of the SGLD. These values are called burn-in.

Note 28. The output of SGLD (Algorithm 19) $\left\{w^{(t)}\right\}$ includes the generated values of w produced during the last few iterations of the running algorithm and after discarding the burn-in generated values.

Note 29. SGLD for $\tau = 1$ approximately simulates from the posterior (1.2).

About facilitating inference...

Note 30. Expectation (1.3), can be estimated as an arithmetic average

$$\widehat{h_T(w)} = \frac{1}{T}\sum_{t=1}^{T} h\left(w^{(t)}\right) \tag{3.7}$$

as $\widehat{h_T(w)} \rightarrow \mathrm{E}_f\left(h(w)\,|z_{1:n}\right)$ based on LLN arguments.

Note 31. Another more efficient estimator for the expectation (1.3) is the weighted arithmetic average

$$\widehat{\hat{h}_T(w)} = \sum_{t=T_0+1}^{T} \frac{\eta_t}{\sum_{t=T_0+1}^{T}\eta_{t'}} h\left(w^{(t)}\right) \tag{3.8}$$

Note 32. Estimator (3.8) has a smaller standard error than estimator (3.7). As the step size η_t decreases, the mixing rate of the chain $\left\{w^{(t)}\right\}$ decreases as well and the simple sample average (3.7) overemphasizes in the tail end of the sequence where there is higher correlation among the samples resulting in higher variance in the estimator.

About the exploding gradient phenomenon...

Note 33. 'Exploding gradients' is the practical phenomenon in which large updates to weights during training can cause a numerical overflow or underflow due to the machine error of the computer.

Note 34. Practical solutions to 'Exploding gradient' involve, at each iteration t, checking the magnitude of the gradient v_t, (e.g., Euclidean norm $\|v_t\|$), and instantly changing it (e.g., gradient scaling or clipping) if it is gonna result an overflow.

Note 35. Gradient scaling involves normalizing the gradient vector such that vector norm (magnitude) equals a user-specified value. More formally, given a gradient v_t, gradient clipping can be used in a standard recursion

$$w^{(t+1)} = w^{(t)} + \eta_t v_t$$

as

$$w^{(t+1)} = w^{(t)} + \eta_t \text{clip}\left(v_t, c\right)$$

where

$$\text{clip}\left(v, c\right) = v \min\left(1, \frac{c}{\|v\|}\right)$$

and $c > 0$ is a clipping threshold impaling that clipping will take place at iteration t if $\|v_t\| > c$.

Note 36. Gradient clipping involves forcing the gradient values (element-wise) to a specific minimum or maximum value if the gradient exceeded an expected range.

Note 37. Unreasonably frequent scaling or clipping may introduce significant bias and hence it should not be applied carelessly.

4. Examples [2]

We continue the Example 26 in 4: Gradient descent.

Specifying the Bayesian model. Consider the Bayesian Normal linear regression model

$$\begin{cases} y_i|\beta, \sigma^2 \sim \text{N}\left(x_i^\top \beta, \sigma^2\right) & \text{sampling distribution } f\left(y_i|\beta, \sigma^2\right) \\ \beta|\sigma^2 \sim \text{N}\left(\mu, \sigma^2 V\right) & \text{prior } f\left(\beta|\sigma^2\right) \\ \sigma^2 \sim \text{IG}\left(\phi, \psi\right) & \text{prior } f\left(\sigma^2\right) \end{cases}$$

and $f\left(\beta, \sigma^2\right) = f\left(\beta|\sigma^2\right) f\left(\sigma^2\right)$, $\beta \in \mathbb{R}^d$, and $\sigma^2 \in \mathbb{R}_+$. Note given densities

$$\text{N}\left(x|\mu, \Sigma\right) = \left(\frac{1}{2\pi}\right)^d \frac{1}{|\Sigma|} \exp\left(-\frac{1}{2}\left(x-\mu\right)^\top \Sigma^{-1}\left(x-\mu\right)\right)$$

$$\text{IG}\left(x|a, b\right) = \frac{b^a}{\Gamma\left(a\right)} x^{-a-1} \exp\left(-\frac{b}{x}\right) 1\left(x \geq 0\right)$$

Performing computationally convenient transformations. Because SGD (Algorithm 13) and SGLD (Algorithm 19) can handle cases where $w \in \mathbb{R}^d$ in a straightforward manner than what they do when $w = \left(\beta, \sigma^2\right) \in \mathbb{R}^d \times \mathbb{R}_+$ which requires an additional projection step; we consider a transformation $w = (\beta, \gamma) \in \mathbb{R}^d \times \mathbb{R}$ with $\gamma = \log\left(\sigma^2\right)$. Hence, the Bayesian model becomes

$$\begin{cases} y_i|\beta, \sigma^2 \sim \text{N}\left(x_i^\top \beta, \exp\left(\gamma\right)\right) & \text{sampling distribution } f\left(y_i|\beta, \gamma\right) \\ \beta|\sigma^2 \sim \text{N}\left(\mu = 0, \exp\left(\gamma\right) V\right) & \text{prior } f\left(\beta|\gamma\right) \\ \gamma \sim f_\gamma\left(\gamma\right) & \text{prior } f\left(\gamma\right) \end{cases}$$

[2] Code is available from `https://github.com/georgios-stats/Machine_Learning_and_Neural_Networks_III_Epiphany_2025/tree/main/Lecture_notes/code/06.Stochastic_gradient_Langevine_dynamics`

where $f_\gamma(\gamma)$ is computed according to the method of bijective transformation of random variables; i.e.

$$f_\gamma(\gamma) = \mathrm{IG}(\exp(\gamma)|\phi,\psi)\left|\frac{\mathrm{d}}{\mathrm{d}\gamma}\exp(\gamma)\right| = \mathrm{IG}(\exp(\gamma)|\phi,\psi)\exp(\gamma)$$

Computing the require quantities for SGLD. Then we can compute the required gradients in order to run the SGD, and SGLD with respect to $w=(\beta,\gamma)$ with $\gamma=\log(\sigma^2)$.

$$\begin{aligned}
\log(f(z_i=(x_i,y_i)|w)) =& -\frac{1}{2}\log(2\pi)-\frac{1}{2}\gamma-\frac{1}{2}\left(y_j-x_i^\top\beta\right)^2\exp(-\gamma)\\
\log(f(w=(\beta,\gamma))) =& \log(f(\beta|\gamma))+\log(f(\gamma))\\
\log(f(\beta|\gamma)) =& -\frac{d}{2}\log(2\pi)-\frac{d}{2}\gamma-\frac{1}{2}|V|-\frac{1}{2}\exp(-\gamma)(\beta-\mu)^\top V^{-1}(\beta-\mu)\\
\log(f(\gamma)) =& \psi\log(\phi)-\log(\Gamma(\phi))-(\phi+1)\gamma-\psi\exp(-\gamma)+\gamma
\end{aligned}$$

Hence for the log sampling PDF we have

$$\begin{aligned}
\nabla_w\log(f(z_i|w)) &= \left(\frac{\mathrm{d}}{\mathrm{d}\beta}\log(f(z_i|w))\ ;\ \frac{\mathrm{d}}{\mathrm{d}\gamma}\log(f(z_i|w))\right)\\
\frac{\mathrm{d}}{\mathrm{d}\beta}\log(f(z_i|w)) &= \left(y_i-x_i^\top\beta\right)x_i\exp(-\gamma)\\
\frac{\mathrm{d}}{\mathrm{d}\gamma}\log(f(z_i|w)) &= -\frac{1}{2}+\frac{1}{2}\left(y_j-x_i^\top\beta\right)^2\exp(-\gamma)
\end{aligned}$$

...so

$$\nabla_w\log(f(z_i|w)) = \begin{pmatrix}\left(y_i-x_i^\top\beta\right)x_i\exp(-\gamma)\\ -\frac{1}{2}+\frac{1}{2}\left(y_j-x_i^\top\beta\right)^2\exp(-\gamma)\end{pmatrix} \tag{4.1}$$

Hence for the log a priori PDF we have

$$\begin{aligned}
\nabla_w\log(f(w)) &= \left(\frac{\mathrm{d}}{\mathrm{d}\beta}\log(f(w))\ ;\ \frac{\mathrm{d}}{\mathrm{d}\gamma}\log(f(w))\right)\\
\frac{\mathrm{d}}{\mathrm{d}\beta}\log(f(w)) &= -\exp(-\gamma)V^{-1}(\beta-\mu)\\
\frac{\mathrm{d}}{\mathrm{d}\gamma}\log(f(w)) &= -\frac{d}{2}+\frac{1}{2}\exp(-\gamma)(\beta-\mu)^\top V^{-1}(\beta-\mu)-(\phi+1)+\psi\exp(-\gamma)+1
\end{aligned}$$

...so

$$\nabla_w\log(f(w)) = \begin{pmatrix}-\exp(-\gamma)V^{-1}(\beta-\mu)\\ \frac{d}{2}+\frac{1}{2}\exp(-\gamma)(\beta-\mu)^\top V^{-1}(\beta-\mu)-(\phi+1)+\psi\exp(-\gamma)+1\end{pmatrix} \tag{4.2}$$

Implementation of SGLD. We consider $\mu=0$, $\phi=1$, $\psi=1$, and $V=100I_d$, for our simulations below.

To implement SGD (Algorithm 13) and SGLD (Algorithm 19) , we just need to plug in the computed gradients (4.1) and (4.2) for $w=(\beta,\gamma)\in\mathbb{R}^d\times\mathbb{R}$ with $\gamma=\log(\sigma^2)$. After running SGD and SGLD with the computed gradients with respect to $w=(\beta,\gamma)\in\mathbb{R}^d\times\mathbb{R}$ with $\gamma=\log(\sigma^2)$, and obtaining chains $\left\{w^{(t)}=\left(\beta^{(t)},\gamma^{(t)}\right)\right\}_{t=1}^T$, we can just perform transformation $\left\{\left(\sigma^{(t)}\right)^2=\exp\left(\gamma^{(t)}\right)\right\}_{t=1}^T$ if we are interested in learning (β,σ^2).

In Figures 4.1, we ran the SGD for different batch sizes m and compared it against the exact MLE. We observe that SGD trace converges to the exact MLE. The oscillations are due to the stochastic gradient (ie, noise in the gradient).

In Figures 4.2, we ran the SGLD for different batch sizes m but the same temperature $\tau = 1$ and compared it against the exact posterior densities. We observe that the histograms of $\left\{w^{(t)}\right\}$ produced from SGLD are closer to the curves representing the exact posteriors when the batch size m is bigger. As we said this is not a panacea; if the landscape of the exact psterior density was multimodal (aka not convex but with had several maxima), then the SGLD using smaller batch sizes could have performed better, in the sense that the inflated noise from the stochastic gradient could accidentally make the generated chain to pass the low mass barrier and visit a different mode, unlike the one with larger batch-size and hence smaller variation.

In Figures 4.3, we ran the SGLD for different temperatures τ but the same batch sizes $m = 100$ and compared it against the exact posterior densities. We observe that increasing the temperature τ may increase the variation of the produced chain. We can use a large τ at the beginning of the run of the algorithm to perform an exploration of the space (this is particularly useful for non-convex/multimodal densities as it allows visiting different modes), and later on we can use a smaller temperature such as $\tau = 1$.

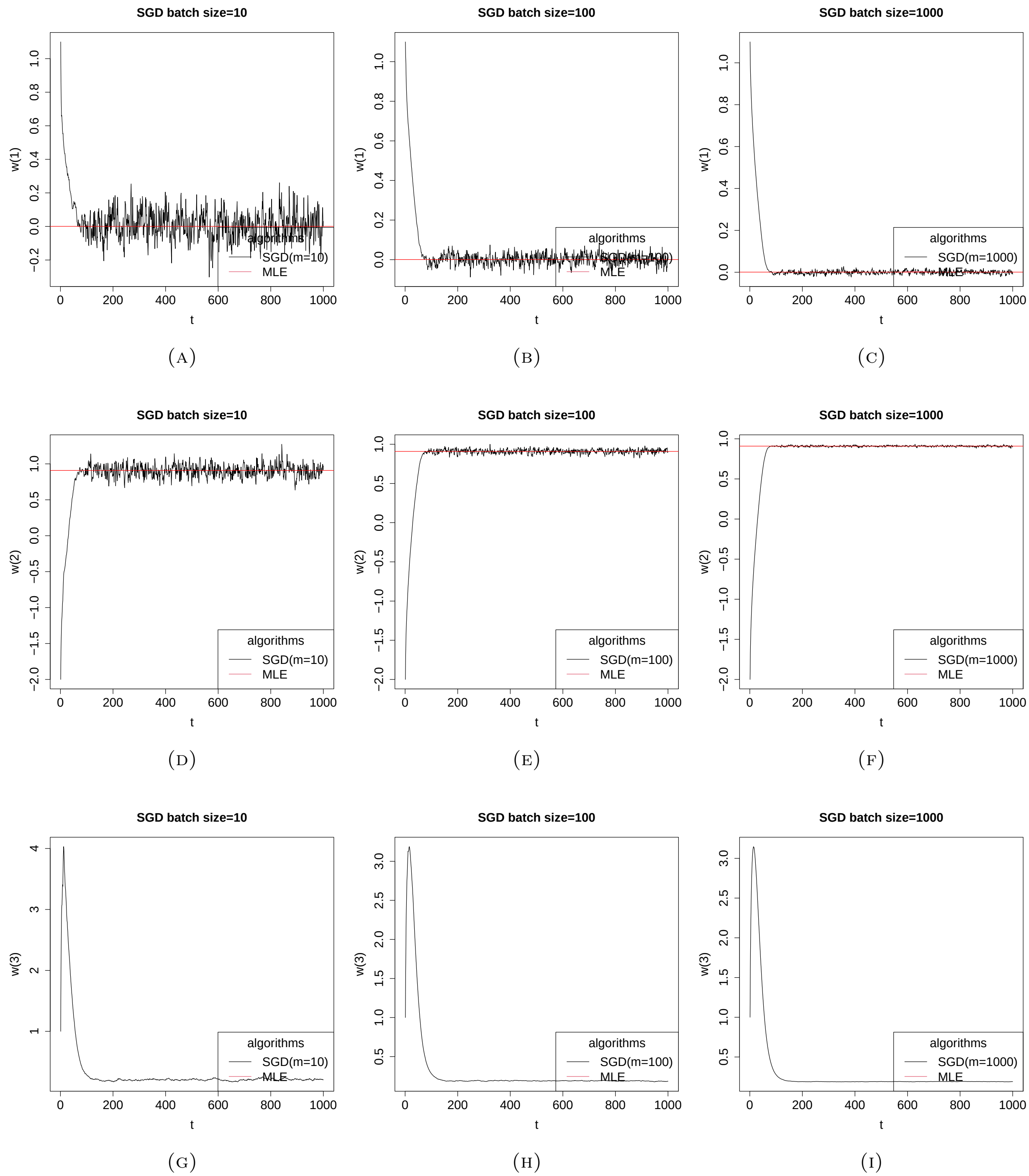


Figure 4.1. Bayesian learning via SGD (SGD vs (exact)MLE) Study on batch size m.

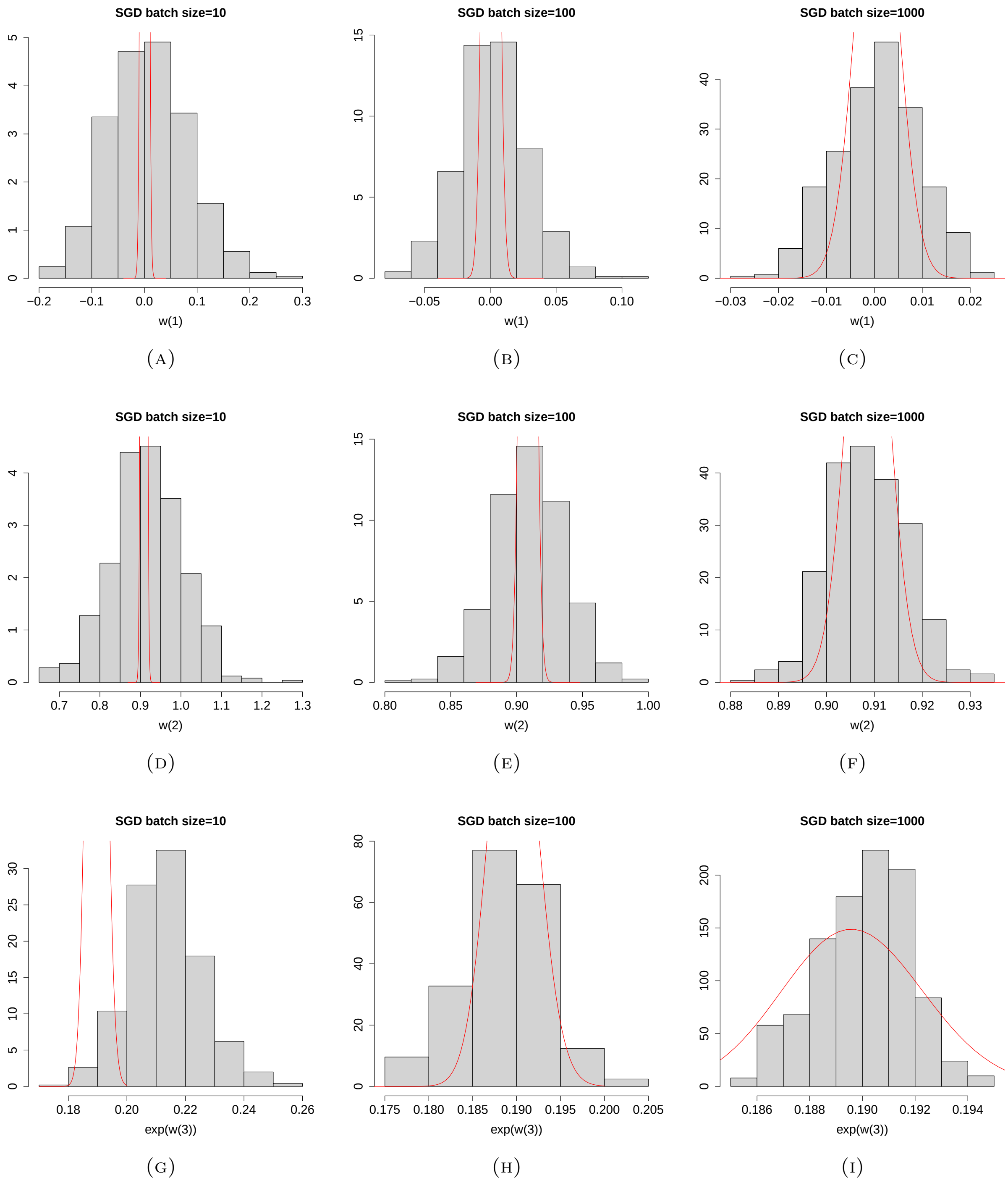


FIGURE 4.2. Bayesian learning: SGLD vs exact posterior (in red). Temperature $\tau = 1$. Study on batch size m

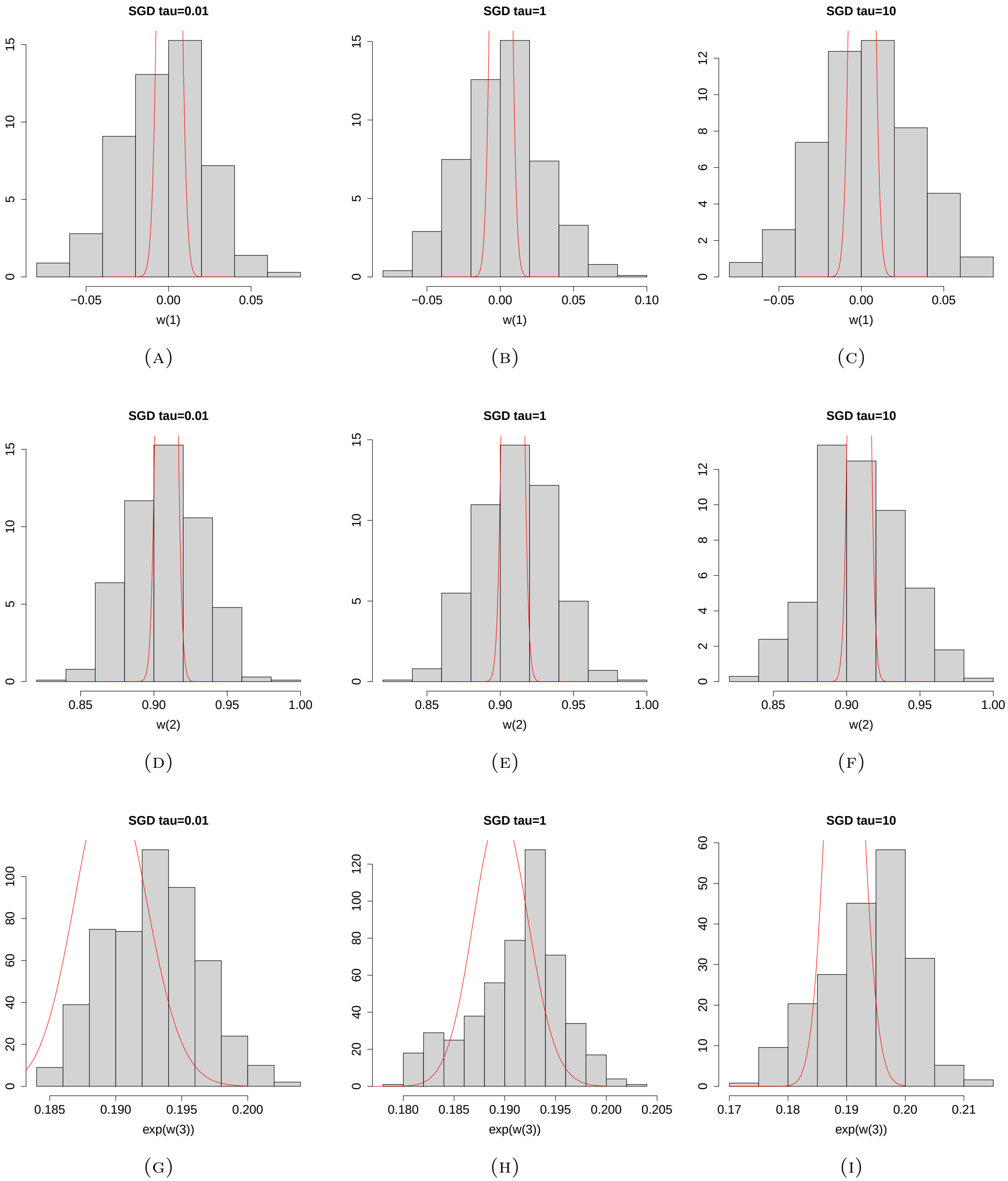


FIGURE 4.3. Bayesian learning: SGLD vs exact posterior (in red). Batch size = 100. Study on τ

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2025

Lecture notes 8: Support Vector Machines

Lecturer & author: Georgios P. Karagiannis georgios.karagiannis@durham.ac.uk

Aim. To introduce the Support Vector Machines as a procedure. Motivation, set-up, description, computation, and implementation. We focus on the classical treatment.

Reading list & references:

(1) Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.
- Ch. 15 (pp. 167-170, 171-172, 176-177) Support Vector Machine

(2) Bishop, C. M. (2006). Pattern recognition and machine learning (Vol. 4, No. 4, p. 738). New York: Springer.
- Ch. 7.1 Sparse Kernel Machines/Maximum marginal classifiers

(3) Vapnik, V. (2013). The nature of statistical learning theory. Springer science & business media.

(4) Boyd, S. P., & Vandenberghe, L. (2004). Convex optimization. Cambridge university press.
- Ch. 4, 5

1. Intro and motivation

Note 1. Support Vector Machines (SVM) is a ML procedure for learning linear predictors in high-dimensional feature spaces with regards the sample complexity challenges. Due to a duality property, SVM have sparse solutions, so that predictions for new inputs depend only on quantities evaluated at a subset of the whole training dataset.

Definition 2. Let $w \neq 0$. **Hyperplane** in space $\mathcal{X} \subseteq \mathbb{R}^d$ is called the sub-set

$$S = \left\{ x \in \mathbb{R}^d : \langle w, x \rangle + b = 0 \right\}. \tag{1.1}$$

Note 3. Hyperplane (1.1) separates $\mathcal{X}$ in two half-spaces

$$S_+ = \left\{ x \in \mathbb{R}^d : \langle w, x \rangle + b > 0 \right\}$$

and

$$S_- = \left\{ x \in \mathbb{R}^d : \langle w, x \rangle + b < 0 \right\}$$

Definition 4. Halfspace (hypothesis space) is hypotheses class $\mathcal{H}$ designed for binary classification problems, $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathcal{Y} = \{-1, +1\}$ defined as

$$\mathcal{H} = \left\{ x \longmapsto \text{sign}\left(\langle w, x \rangle + b \right) : w \in \mathbb{R}^d, b \in \mathbb{R} \right\},$$

where b is called bias.

Definition 5. Each **halfspace hypothesis** $h \in \mathcal{H}$ has form

$$h_{w,b}(x) = \text{sign}(\langle w, x\rangle + b). \tag{1.2}$$

It takes an input in $\mathcal{X} \subseteq \mathbb{R}^d$ and returns an output in $\mathcal{Y} = \{-1, +1\}$. We may refer to it as halfspace (w, b) as well.

Note 6. Let $S = \{(x_i, y_i)\}_{i=1}^m$ be a training set of examples with $x_i \in \mathbb{R}^d$ the features and $y_i \in \{-1, +1\}$ the labels.

Note 7. Our goal is to train a halfspace hypothesis $h_{w,b}(x)$ in (1.2) against a training dataset S, with puppose to be able to classify a future feature x as $y = -1$ or $y = 1$.

Definition 8. The training set S is **linearly separable** if there exists a halfspace (w, b) such that for all $i = 1, ..., n$

$$y_i = \text{sign}(\langle w, x_i\rangle + b)$$

or equivalently

$$y_i(\langle w, x_i\rangle + b) > 0 \tag{1.3}$$

Note 9. Halfspaces (w^*, b^*) satisfying the linearly separable condition (1.3) are ERM hypothesis under the $0-1$ loss function $\ell^{0-1}((w, b), z) = 1_{(y_i \neq \text{sign}(\langle w, x_i\rangle + b))}$ and Empirical Risk $R_S^{0-1}(w, b) = \frac{1}{m}\sum_{i=1}^m \ell((w, b), z_i) \geq 0$, i.e.

$$(w^*, b^*) = \arg\min_{w,b}\left(R_S^{0-1}(w, b)\right) = \arg\min_{w,b}\left(\frac{1}{m}\sum_{i=1}^m \ell^{0-1}((w, b), z_i)\right)$$

as $R_S^{0-1}(w^*, b^*) = 0$.

Definition 10. Margin of a hyperplane with respect to a training set is defined to be the minimal distance between a point in the training set and the hyperplane.

Note 11. There are several different halfspaces (w, b) satisfying (1.3) for the same linearly separable training dataset S; see Figure (1.1; left). In Figure (1.1; right) the margin γ is the distance from the hyperplane (solid line) to the closest points in either class (which touch the parallel dotted lines). Among the hyperplanes satisfying (1.3), a reasonable/desirable halfspace (w, b) is the one with the maximum margin γ; see Figure (1.1; right). The rational is that if a hyperplane has a large margin, then it will still separate the training set even if we slightly perturb each instance.

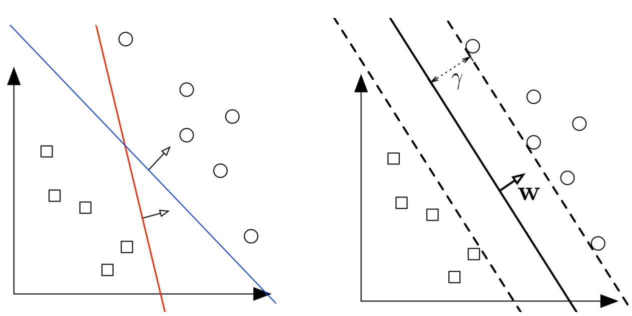


FIGURE 1.1

Note 12. Support Vector Machines (SVM) aims at learning the maximum margin separating hyperplane Figure (1.1; Right).

2. Hard Support Vector Machine

Assumption 13. *Assume the training sample* $S = \{(x_i, y_i)\}_{i=1}^m$ *is linearly separable.*

Definition 14. Hard Support Vector Machine (Hard-SVM) is the learning rule in which we return an ERM hyperplane that separates the training set with the largest possible margin given Assumption 13.

Problem 15. (Hard-SVM) Given a linearly separable training sample $S = \{(x_i, y_i)\}_{i=1}^m$ the Hard-SVM rule for the binary classification problem is the solution to the quadratic optimization problem:

$$\text{Solve } \left(\tilde{w}, \tilde{b}\right) = \arg\min_{(w,b)} \frac{1}{2} \|w\|_2^2 \tag{2.1}$$

$$\text{subject to: } y_i \left(\langle w, x_i \rangle + b\right) \geq 1, \ \forall i = 1, ..., m \tag{2.2}$$

Scale

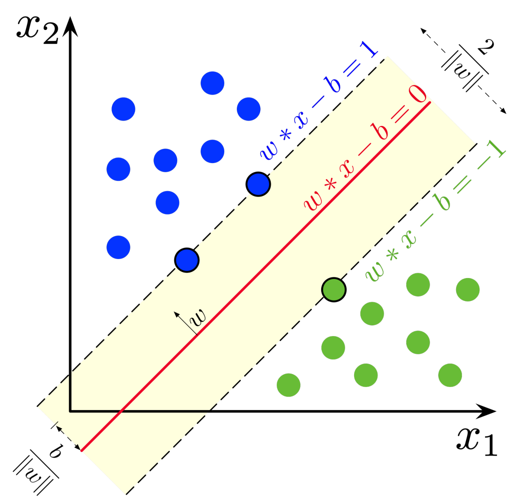
$$\hat{w} = \frac{\tilde{w}}{\|\tilde{w}\|}, \text{ and } \hat{b} = \frac{\tilde{b}}{\|\tilde{w}\|}$$

Note 16. Following we show why Problem 15 produces a Hard-SVM hyperplane stated in Note 14.

Lemma 17. *The distance between a point* x *and the hyperplane defined by* (w, b) *with is* $|\langle w, x \rangle + b| \, / \, \|w\|$.

Proof. We skip it. □

Note 18. On the right, see the geometry of Problem 15.



Note 19. Hard-SVM selects two parallel hyperplanes that separate the two classes of data so that the distance between them is as large as possible. The predictive hyperplane (rule) is the hyperplane that lies halfway between them.

Note 20. Hard-SVM in Problem 15 searches for the hyperplane with minimum norm w among all those that separate the data and have distance greater or equal to 1.

Proof. (Sketch of the proof of Problem 15)

(1) Let's formulate the Hard SVM problem according to the SVM definition under the linear separation assumption.

(a) Based on Note 14, and Lemma 17, the closest point in the training set to the separating hyperplane has distance

$$\min_i \left(|\langle w, x_i \rangle + b| \,/\, \|w\| \right) \tag{2.3}$$

According to Definition 14, the Hard-SVM hypothesis can be formulated as having to find

$$(w^*, b^*) = \arg\max_{(w,b)} \left(\min_i \left(\frac{1}{\|w\|} \, |\langle w, x_i \rangle + b| \right) \right) \tag{2.4}$$

$$\text{subject to } \; y_i \left(\langle w, x_i \rangle + b \right) > 0, \; \forall i = 1, ..., m \tag{2.5}$$

(b) We observe that (2.4) & (2.5) is non-identifiable in the sense that $(\kappa w^*, \kappa b^*)$ for any $\kappa \neq 0$ are solutions. To make (2.3) identifiable (without loss of generality) we pick only (w^*, b^*) with $\|w\| = 1$. Additionally, because we consider only (w^*, b^*) which correctly separate the linearly separable training dataset into the two the categories (i.e. (2.5)), it is $|\langle w, x_i \rangle + b| = y_i \left(\langle w, x_i \rangle + b \right)$. Hence, (2.4) & (2.5) is equivalent to computing

$$(w^*, b^*) = \arg\max_{(w,b): \|w\|=1} \left(\min_i \left(y_i \left(\langle w, x_i \rangle + b \right) \right) \right) \tag{2.6}$$

(2) Next we show that the solution of (2.6) is equivalent to the solution of Problem 15; i.e. $(w^*, b^*) = \left(\hat{w}, \hat{b} \right)$.

Let $\gamma^* := \min_i \left(|\langle w^*, x_i \rangle + b^*| \right)$ be the margin from (w^*, b^*). Firstly, because

$$y_i \left(\langle w^*, x_i \rangle + b^* \right) \geq \gamma^* \;\Leftrightarrow\; y_i \left(\langle \frac{w^*}{\gamma^*}, x_i \rangle + \frac{b^*}{\gamma^*} \right) \geq 1$$

$\left(\frac{w^*}{\gamma^*}, \frac{b^*}{\gamma^*} \right)$ satisfies conditions (2.1) and (2.2). Secondly, I have $\|\tilde{w}\| \leq \left\| \frac{w^*}{\gamma^*} \right\| = \frac{1}{\gamma^*}$ because of (2.1) and because of $\|w^*\| = 1$. Hence, for all $i = 1, ..., m$, it is

$$y_i \left(\langle \hat{w}, x_i \rangle + \hat{b} \right) = \frac{1}{\|\tilde{w}\|} y_i \left(\langle \tilde{w}, x_i \rangle + \tilde{b} \right) \geq \frac{1}{\|\tilde{w}\|} \geq \gamma^*$$

Hence $\left(\hat{w}, \hat{b} \right)$ is the optimal solution of (2.6).

□

Definition 21. Homogeneous halfspaces in SVM is the case where the halfspaces pass from the origin; that is when the bias term in (2.2) is zero $b = 0$.

3. Soft Support Vector Machine

Note 22. Hard-SVM assumes the strong Assumption 13 that the training set is linearly separable. This might not always the case, and hence there is need to derive a procedure that weakens this assumption.

Note 23. Soft Support Vector Machine (Soft-SVM) aims to relax the strong assumption of Hard-SVM that the training set is linearly separable (2.5) with purpose to extend the scope of application.

Soft-SVM does not assume Assumption 13. Soft-SVM rule is given as the solution to the quadratic optimization problem 24.

Problem 24. (Soft-SVM) Given a training sample $S = \{(x_i, y_i)\}_{i=1}^m$ the Soft-SVM rule for the binary classification learning problem is solution to the quadratic optimization Problem

$$\text{Solve} \quad (w^*, b^*, \xi^*) = \underset{(w,b,\xi)}{\arg\min} \left(\frac{1}{2} \|w\|_2^2 + C \frac{1}{m} \sum_{i=1}^m \xi_i \right) \tag{3.1}$$

$$\text{subject to: } y_i (\langle w, x_i \rangle + b) \geq 1 - \xi_i, \ \forall i = 1, ..., m \tag{3.2}$$

$$\xi_i \geq 0, \ \forall i = 1, ..., m \tag{3.3}$$

Note 25. To relax the linearly separable training set Assumption 13, Soft-SVM relies on replacing the "harder" constraint (2.2) with the "softer" one in (3.2) through the introduction of non-negative unknown quantities (slack variables) $\{\xi_i\}_{i=1}^m$ controlling how much the separability assumption (2.2) is violated. Because any point that is misclassified has $\xi_i > 1$, it follows that $\sum_{i=1}^m \xi_i$ is an upper bound on the number of misclassified points.

Note 26. Parameter $C > 0$ controls the trade-off between the slack variable penalty and the margin. It controls the trade-off between minimizing training errors and controlling model complexity.

Note 27. Soft-SVM learns all (w, b, ξ) via the minimization part in (3.1) where the trade off between the two terms is controlled via the user specified parameter C.

Note 28. Proposition 29 shows that the Soft-SVM is a binary classification learning problem under the hinge loss function and with a regularization term biasing toward low norm separators.

Proposition 29. *The solution of Problem 24 is equivalent to the Ridge regularized loss minimization problem*

$$(w^*, b^*) = \underset{(w,b)}{\arg\min} \left(R_S^{hinge} + \lambda \|w\|_2^2 \right) \tag{3.4}$$

with the hinge loss function

$$\ell^{hinge}((w, b), z) = \max(0, 1 - y(\langle w, x \rangle + b))$$

Empirical Risk Function

$$R_S^{hinge}((w, b)) = \frac{1}{m} \sum_{i=1}^m \max(0, 1 - y_i(\langle w, x_i \rangle + b))$$

regularization parameter $\lambda = \frac{1}{2C}$ *and regularization term* $\|\cdot\|_2^2$.

Proof. In Algorithm 24, we consider (3.1) as

$$\underset{(w,b)}{\arg\min} \left(\min_{\xi} \left(\lambda \|w\|_2^2 + \frac{1}{m} \sum_{i=1}^m \xi_i \right) \right) \tag{3.5}$$

with $\lambda = \frac{1}{2C}$. Consider (w, b) fixed and focus on the inside minimization. From (3.2), it is $\xi_i \geq 1 - y_i (\langle w^*, x_i \rangle + b^*)$, and from (3.3), it is $\xi_i \geq 0$. If $y_i (\langle w, x_i \rangle + b) \geq 1$, the best assignment in 3.5 is $\xi_i = 0$ because it is $\xi_i \geq 0$ from (3.3) and I need to minimize (3.5) wrt ξ_i's. If $y_i (\langle w, x_i \rangle + b) \leq 1$, the best assignment in (3.5) is $\xi_i = 1 - y_i (\langle w, x_i \rangle + b)$ because I need to minimize w.r.t ξ_i's. Hence $\xi_i = \max(0, 1 - y_i (\langle w, x_i \rangle + b))$. □

Example 30. Consider the Soft-SVM as a Ridge regularized loss minimization learning problem in (3.4) on a training dataset $S = \{(x_i, y_i)\}_{i=1}^m$ with $z_i = (x_i, y_i) \overset{\text{ind}}{\sim} g$ where g is a data generating process on $\mathcal{X} \times \{0, 1\}$ with $\mathcal{X} = \{x : \|x\| \leq \rho\}$. Let $\mathfrak{A}(S)$ be the solution of the Soft-SVM. Then:

- Because $\ell^{\text{hinge}}((w, b), z) = \max(0, 1 - y(\langle w, x \rangle + b))$ is $\|x\|$-Lipschitz, it is
$$\mathrm{E}_{S\sim g}\left(R_g^{\text{hinge}}(\mathfrak{A}(S))\right) \leq R_g^{\text{hinge}}(u) + \lambda \|u\|^2 + \frac{2\rho^2}{\lambda m}$$
[Hint: Note 43 Handout 3: Learnability and stability in learning problems]
- If we assume a bounded learning problem, i.e. $\mathcal{H} = \{\|(w, b)\| \leq B\}$ for $B > 0$ then
$$\mathrm{E}_{S\sim g}\left(R_g^{\text{hinge}}(\mathfrak{A}(S))\right) \leq \min_{\|(w,b)\| \leq B} R_g^{\text{hinge}}((w, b)) + \lambda B^2 + \frac{2\rho^2}{\lambda m}$$
- If we choose $\lambda = \sqrt{\frac{2\rho^2}{B^2 m}}$ then
$$\mathrm{E}_{S\sim g}\left(R_g^{\text{hinge}}(\mathfrak{A}(S))\right) \leq \min_{\|(w,b)\| \leq B} R_g^{\text{hinge}}((w, b)) + \sqrt{\frac{8\rho^2 B^2}{m}}$$
- Assume one was interested in minimizing a $0 - 1$ Risk $R_g^{0-1}(\cdot)$ under the $0 - 1$ loss $\ell^{0-1}((w, b), z) := 1_{(y\langle w,x\rangle \leq 0)}$. But $\ell^{0-1}(\cdot, z)$ is non-convex for all z. It is $\ell^{0-1}((w, b), z) \leq \ell^{\text{hinge}}((w, b), z)$ for all z. By using the convex hinge loss ℓ^{hinge} as a surrogate loss and implementing the Soft-SVM procedure, the error under $0 - 1$ Risk $R_g^{0-1}(\cdot)$ is

$$\mathrm{E}_{S\sim g}\left(R_g^{0-1}(\mathfrak{A}(S))\right) \leq \mathrm{E}_{S\sim g}\left(R_g^{\text{hinge}}(\mathfrak{A}(S))\right) \leq \min_{\|(w,b)\| \leq B} R_g^{\text{hinge}}((w, b)) + \sqrt{\frac{8\rho^2 B^2}{m}}$$

Example 31. Given Proposition 29, Soft-SVM in Problem 24 can be learned via any variation of SGD, eg online SGD (batch size $m = 1$) with recursion

$$\varpi^{(t+1)} = \varpi^{(t)} - \eta_t v_t$$
$$\begin{bmatrix} w^{(t+1)} \\ b^{(t+1)} \end{bmatrix} = \begin{bmatrix} w^{(t)} \\ b^{(t)} \end{bmatrix} - \eta_t \begin{bmatrix} v_{w,t} \\ v_{b,t} \end{bmatrix}$$

where $v_{w,t} = \begin{cases} -y_i^{(t)} x_i^{(t)} & \text{if } y_i^{(t)}\left(\langle w^{(t)}, x_i^{(t)} \rangle + b^{(t)}\right) < 1 \\ 0 & \text{otherwise} \end{cases}$ and $v_{b,t} = \begin{cases} -y_i^{(t)} & \text{if } y_i^{(t)}\left(\langle w^{(t)}, x_i^{(t)} \rangle + b^{(t)}\right) < 1 \\ 0 & \text{otherwise} \end{cases}$.

Recal Lagrange mukltipliers and duality from previous years

4.1. **Lagrangian duality (recall from previous years).**

Notation 32. Let $f:\mathbb{R}^q\to\mathbb{R}$ be an objective function, let $\{g_i:\mathbb{R}^q\to\mathbb{R}\}_{i=1}^m$ inequality constraint convex functions, and let $\{h_j:\mathbb{R}^q\to\mathbb{R}\}_{j=1}^n$ equality constraint functions.

Definition 33. (Primal problem) Consider we have the following minimization problem that we will call Primal problem

$$p^*=\min_x\left(f\left(x\right)\right) \quad (4.1)$$
$$\text{s.t.} g_i\left(x\right)\leq 0,\ \ \forall i=1,...,m$$
$$h_j\left(x\right)=0,\ \ \forall j=1,...,n$$

Definition 34. (Lagrangian function) To the Primal problem 33, we associate the Lagrangian function $L:\mathbb{R}^q\times\mathbb{R}^m\times\mathbb{R}^n$ with

$$L\left(x,\alpha,\beta\right)=f\left(x\right)+\sum_{i=1}^m\alpha_i g_i\left(x\right)+\sum_{j=1}^n\beta_j h_j\left(x\right) \quad (4.2)$$

Note 35. Note that the Primal problem is equivalent to

$$p^*=\min_x\left(\max_{\alpha\geq 0,\beta}\left(L\left(x,\alpha,\beta\right)\right)\right)$$

For instance, if I ignore the $\{h_j\}$ terms which is zero for a suitable solution for simplicity, I observe that

$$\max_{\alpha\geq 0,\beta}\left(L\left(x,\alpha,\beta\right)\right)=\begin{cases}f\left(x\right) & g_i\left(x\right)\leq 0,\forall i\\ \infty, & g_i\left(x\right)>0,\exists i\end{cases}$$

Definition 36. (Lagrangian dual problem) The associated Lagrangian dual problem is

$$d^*=\max_{\alpha,\beta}\left(\min_x\left(L\left(x,\alpha,\beta\right)\right)\right)$$
$$\text{s.t.}\alpha_i\geq 0$$
$$\beta_j\in\mathbb{R}$$

Definition 37. (Dual function) To the Dual problem, we associate a function $\tilde{L}:\mathbb{R}^m\times\mathbb{R}^n$ with

$$\tilde{L}\left(\alpha,\beta\right):=\min_x\left(L\left(x,\alpha,\beta\right)\right)$$

called dual function.

Proposition 38. *In general it is $p^*\geq d^*$; i.e.*

$$\mathit{min}_x\max_{\alpha\geq 0,\beta}\left(L\left(x,\alpha,\beta\right)\right)\geq\mathit{max}_{\alpha\geq 0,\beta}\min_x\left(L\left(x,\alpha,\beta\right)\right)$$

Definition 39. We call the general case $p^*\geq d^*$ weak duality.

Definition 40. When $p^* = d^*$ we say we have a strong duality.

Proposition 41. *(Strong duality via Slater condition) If the primal problem (4.1) is convex, and satisfies the weak Slater's condition, i.e.*

$$(\exists x_0 \in \mathcal{D}) \; : \; (g_i(x_0) < 0, \;\; \forall i = 1, ..., n) \textit{ and } (h_i(x_0) = 0, \;\; \forall i = 1, ..., n)$$

then strong duality holds, that is: $p^* = d^*$. It other words

$$\min_x \max_{\alpha \geq 0, \beta} (L(x, \alpha, \beta)) = \max_{\alpha \geq 0, \beta} \min_x (L(x, \alpha, \beta))$$

Proposition 42. *When f, $\{h_j\}$, $\{g_i\}$ are convex and the Slater condition holds, Karush–Kuhn–Tucker (KKT) conditions are necessary and sufficient conditions for x^* to be local minimum*

$$\begin{aligned}
0 =& \nabla f(x^*) + \sum_{j=1}^{n} \beta_j \nabla h_j(x^*) + \sum_{i=1}^{m} \alpha_i \nabla g_i(x^*) && \textit{Stationarity} \\
& g_i(x^*) \leq 0, \;\; \forall i = 1, ..., m && \textit{Primal feasibility} \\
& h_j(x^*) = 0, \;\; \forall j = 1, ..., n && \\
& \alpha_i \geq 0 \;\; \forall i = 1, ..., m && \textit{Dual feasibility} \\
& \alpha_i g_i(x^*) = 0, \;\; \forall i = 1, ..., m && \textit{Complementary slackness}
\end{aligned} \tag{4.3}$$

4.2. **Implementation in the Hand SVM.**

Note 43. Consider the Primal problem in Hard-SVM Algorithm 15 (2.2) and (2.2).

Note 44. According to Definition 34 the Lagrangian function is

$$L(w, b, \alpha) = \frac{1}{2} \|w\|_2^2 - \sum_{i=1}^{m} \alpha_i (y_i(\langle w, x_i \rangle + b) - 1) \tag{4.4}$$

where $\{\alpha_i\}_{i=1}^m$ are non-negative Lagrange multipliers.

Insight with regards to Note 35

Note 45. (Insight with regards to Note 35) The Hard-SVM Problem 15 (2.2) and (2.2), can be rewritten in the form

$$\min_w \left(\frac{1}{2} \|w\|_2^2 + g(w) \right) \tag{4.5}$$

with

$$g(w) = \max_{\alpha \geq 0} \sum_{i=1}^{m} \alpha_i (1 - y_i(\langle w, x_i \rangle + b)) = \begin{cases} 0 & \text{if } \; y_i(\langle w, x_i \rangle + b) \geq 1 \\ \infty & \text{else} \end{cases}$$

Hence we get the weak duality

$$\min_{w,b} \max_{\alpha \geq 0} \left(\frac{1}{2} \|w\|_2^2 + \sum_{i=1}^{m} \alpha_i \left(1 - y_i \left(\langle w, x_i \rangle + b\right)\right) \right)$$

$$\geq \max_{\alpha \geq 0} \min_{w,b} \left(\underbrace{\frac{1}{2} \|w\|_2^2 + \sum_{i=1}^{m} \alpha_i \left(1 - y_i \left(\langle w, x_i \rangle + b\right)\right)}_{=L(w,b;\alpha)} \right) \tag{4.6}$$

Note 46. Strong duality (equality) holds because of the convexity. Here, keeping α fixed, $L(w, \alpha, b)$ is minimized when

$$0 = \nabla_w L(w, \alpha, b)|_{(w^*, b^*)} \implies w^* = \sum_{i=1}^{m} \alpha_i y_i x_i \tag{4.7}$$

$$0 = \nabla_b L(w, \alpha, b)|_{(w^*, b^*)} \implies 0 = \sum_{i=1}^{m} \alpha_i y_i \tag{4.8}$$

The dual Lagrangian function is

$$\tilde{L}(\alpha) = \min_{w,b} \left(L(w, \alpha, b)\right) = \frac{1}{2} \left\| \sum_{i=1}^{m} \alpha_i y_i x_i \right\|_2^2 - \sum_{i=1}^{m} \alpha_i \left(y_i \left(\langle \sum_{j=1}^{m} \alpha_j y_j x_j, x_i \rangle + b \right) - 1 \right)$$

$$= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_j x_i \rangle$$

Then dual problem is

$$\max_{\alpha \in \mathbb{R}^m : \alpha \geq 0} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_j x_i \rangle \right)$$

Note 47. The Dual problem of the Hard-SVM (primal) problem 15 is

$$\alpha^* = \arg \max_{\alpha \in \mathbb{R}^m : \alpha \geq 0} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_j, x_i \rangle \right) \tag{4.9}$$

$$\text{subject to } 0 = \sum_{i=1}^{m} \alpha_i y_i$$

Note 48. Once the optimal α^* is computed from (4.9), the optimal weights w^* can be computed (from (4.7)) as

$$w^* = \sum_{i=1}^{m} \alpha_i^* y_i x_i \tag{4.10}$$

Note 49. If $\alpha_i^* = 0$, the example (x_i, y_i) does not contribute to (4.10). If $\alpha_i^* \neq 0$, (x_i, y_i) contributes to (4.10). Moreover if $\alpha_i^* \neq 0$ the KKT condition (4.3)

$$\alpha_i^* \left(y_i \left(\langle w^*, x_i \rangle + b^* \right) - 1 \right) = 0$$

implies that $y_i\left(\langle w^*, x_i\rangle + b^*\right) = 1$ meaning that x_i is on the boundary of the margin. Features x_i associated to $\alpha_i^* \neq 0$ which contribute to the evaluation of the optimal weights w^* and hence the evaluation of the separating predictive rule $h_{w,b}$ are called **supporting vectors** (see Figure 4.1).

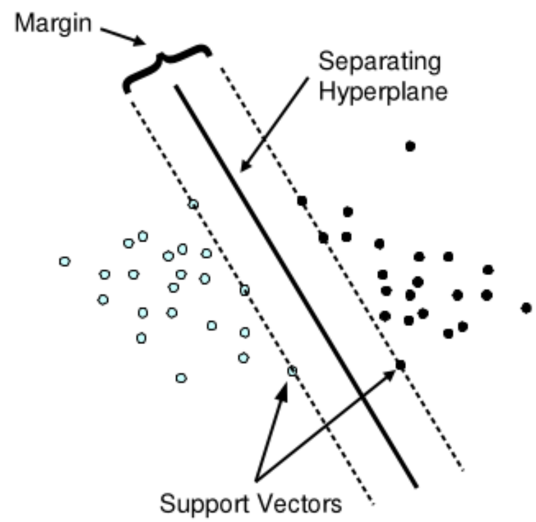


FIGURE 4.1

Note 50. Given optimal α^* (4.10) becomes

$$w^* = \sum_{i\in\mathcal{I}} \alpha_i^* y_i x_i \tag{4.11}$$

where $\mathcal{I} = \{i : y_i\left(\langle w^*, x_i\rangle + b^*\right) - 1 = 0\}$.

Note 51. The bias b parameter can be computed by KKT condition (4.3), it is

$$\alpha_i^*\left(y_i\left(\langle w^*, x_i\rangle + b^*\right) - 1\right) = 0$$

Consider $\mathcal{I} = \{i : y_i\left(\langle w^*, x_i\rangle + b^*\right) - 1 = 0\}$ then, for $i \in \mathcal{I}$ it is

$$y_i^2\left(\langle w^*, x_i\rangle + b^*\right) - y_i = 0 \overset{y_i^2=1}{\Longrightarrow} \langle w^*, x_i\rangle + b^* - y_i = 0$$

and summing up all $i \in \mathcal{I}$, I get

$$\sum_{i\in\mathcal{I}}\left(\langle w^*, x_i\rangle + b^* - y_i\right) = 0 \implies b^* = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}\left(y_i - \langle w^*, x_i\rangle\right) \tag{4.12}$$

Note 52. Therefore the predictive halfspace hypothesis is given as

$$\begin{aligned} h_{w,b}(x) &= \text{sign}\left(\langle w^*, x\rangle + b^*\right) \\ &= \text{sign}\left(\sum_{i\in\mathcal{I}} \alpha_i^* y_i \langle x_i, x\rangle + b^*\right) \end{aligned} \tag{4.13}$$

with $\mathcal{I} = \{i : y_i\left(\langle w^*, x_i\rangle + b^*\right) - 1 = 0\}$.

Note 53. Compared to the Primal problem (Problem 15), the dual problem (Note 47) is computationally desirable in cases that the training data size m is smaller than the number of the dimensions d of the feature space; i.e. $d \gg m$. The solution of the Primal quadratic programming problem in $d+1$ and complexity $O(d+1)$, while the dual quadratic programming problem has m variables and complexity $O(m)$.

Note 54. Compared to the Primal problem, the dual problem (Note 47), can provide sparse solutions relying on a small number of support vectors (see Eq. 4.11, 4.12, and 4.13).

Note 55. (Snapshot of the next concept "The kernel trick") The importance of the existence of the dual problem is that eg (4.9) and (4.13) involves the inner products between instances and does not require the direct access to specific elements of features within instance. For instance, if we consider the hypothesis (1.2) as an expansion of bases $h_{w,b}(x) = \text{sign}(\langle w, \phi(x)\rangle + b)$ with $\phi(x) = (\phi_1(x), ..., \phi_d(x))$ with a large d such as $d \gg m$ then, based on (4.13), the separation rule is

$$h_{w,b}(x) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i y_i k(x_i, x) + b\right)$$

and 4.9 becomes

$$\alpha^* = \arg \max_{\alpha \in \mathbb{R}^m : \alpha \geq 0} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j)\right)$$

with $k(x', x) = \langle \phi(x'), \phi(x)\rangle$. As we will see in the "kernel methods" lecture, in specific case, one can specify the "kernel" function $k(\cdot, \cdot)$ (having with specific desirable properties) avoiding the direct specification of a possibly high-dimensional dictionary of features $\{\phi_j(\cdot)\}$.

Note 56. In the Soft-SVM case, the solution is the same as in the Hard-SVM (4.10) and (4.13), the only difference is that in the quadratic programming problem (4.9) it is subject to $\alpha_j \in [0, C]$ where $C = 1/2\lambda$, it is called "cost" and controls the cost of having the constraint violation by adding the $\xi_i' s$. (See Exercise 16 from the Exercise sheet).

Note 57. Sparsity and duality in Soft SVM can be seen in Exercise 16 in the Exercise sheet.

Note 58. Relevance Vector Machine (a version of Bayesian SVM) can be seen in Exercise 17 in the Exercise sheet.

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2025

Lecture notes 9: The Kernel trick

Lecturer & author: Georgios P. Karagiannis georgios.karagiannis@durham.ac.uk

Aim. To introduce the ideas of learning machines by introducing data into high-dimensional feature spaces for accuracy gains; introduce the kernel trick, and kernel functions.

Reading list & references:

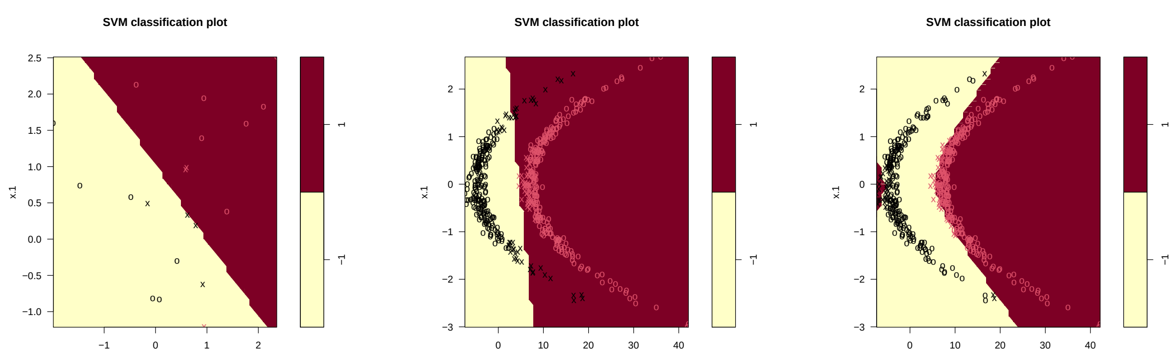
(1) Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.
   - Ch. 16.2 Support Vector Machine

(2) Bishop, C. M. (2006). Pattern recognition and machine learning (Vol. 4, No. 4, p. 738). New York: Springer.
   - Ch. 6.1, 6.2 Kernel methods

(3) Vapnik, V. (2013). The nature of statistical learning theory. Springer science & business media.

1. Intro and motivation

Note 1. Consider the Soft SVM with predictive rule $h(x) = \text{sign}(\eta(x))$ with separator $\eta(x) = \langle w, x\rangle + b$ where $w = (w_1, w_2)^\top \in \mathbb{R}^2$ and $x = (x_1, x_2)^\top \in \mathcal{X} \subseteq \mathbb{R}^2$. It can address learning problems where the data can (up to some degree of violation) be separated by a line (Figure 1.1a). In more challenging cases where the geometry is strongly non-linear this can totally fail (Figure 1.1b).



(a) Linearly separable data set, with linear separator

(b) Non-linearly separable data set, with linear separator

(c) Non-linearly separable data set, with quadratic separator

Figure 1.1. Soft SVM from the Computer lab

Note 2. The accuracy of predictive rule could be improved if I could take into account the curvature in the feature space $\mathcal{X}$ by adding a quadratic term in the 2nd dimension as $h^{\psi}(x) = \text{sign}\left(\eta^{\psi}(x)\right)$ with separator $\eta^{\psi}(x) = \langle w', \psi(x) \rangle + b'$ for some $\psi : \mathbb{R}^2 \to \mathbb{R}^3$ with $\psi(x) = \left(x_1, x_2, x_2^2\right)$ and learning w' and b' . It works, (Figure 1.1c).

Note 3. In other words, in order to improve the expressiveness of a given hypothesis class $\mathcal{H} = \{x \mapsto f(\langle w, x \rangle)\}$ (for some function f) with purpose to learn a more accurate predictive rule, it is reasonable to consider an embedding $\psi(x)$ and work on the learning problem with $\mathcal{H} = \{x \mapsto f(\langle w, \psi(x) \rangle)\}$. Such an embedding $\psi(x)$ can possibly be a vector of basis functions such as polynomials, splines, etc...

Note 4. The above may drastically increase the dimensionality of the problem hence the computational cost and required training dataset size. This challenge is addressed by the Kernel trick which allows the design of cheap more expressive extensions of many well known algorithms.

2. Improving expressive power via embeddings in feature spaces

Note 5. To make the class of hypotheses more expressive with purpose to improve accuracy, we can first map the original instance space $x \in \mathcal{X}$ into another feature space $\mathcal{F}$ (possibly of a higher dimension) via an embedding $\psi : \mathcal{X} \to \mathcal{F}$ and then learn a hypothesis in that space.

Note 6. Consider a given ERM learning problem $(\mathcal{H}, \ell, \mathcal{Z})$ that involves hypothesis class $\mathcal{H}$, predictive rule $h \in \mathcal{H}$, loss function $\ell : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}_{+}$, training dataset $\mathcal{S} = \{z_i = (x_i, y_i)\}_{i=1}^{m}$ from data generating process $G(\cdot)$ defined over $\mathcal{Z} = \mathcal{Y} \times \mathcal{X}$.

Note 7. The introduction of embedding $\psi : \mathcal{X} \to \mathcal{F}$ induces

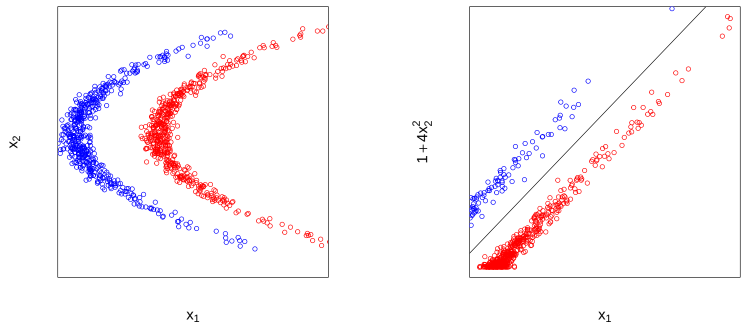
(1) a probability distribution G^{ψ} over domain $\mathcal{F} \times \mathcal{Y}$ with $G^{\psi}(A) = G\left(\psi^{-1}(A)\right)$ for every set $A \subseteq \mathcal{F} \times \mathcal{Y}$ given a data generating process $G(\cdot)$.
(2) predictive rule $h^{\psi}(\cdot) := h \circ \psi(\cdot) = h(\psi(\cdot))$
(3) risk function $R_{G^{\psi}}(h) := R_G(h \circ \psi)$, as

$$R_G(h \circ \psi) = \int \ell(h \circ \psi, z = (x, y))\, \mathrm{d}G(z) = \int \ell\left(h, z^{\psi}\right) \mathrm{d}G^{\psi}(x, y) = R_{G^{\psi}}(h)$$

Note 8. The basic paradigm for improving expressive power of $\mathcal{H}$ via embedding $\psi : \mathcal{X} \to \mathcal{F}$ involves:

(1) Choose an embedding $\psi : \mathcal{X} \to \mathcal{F}$ with $\psi(x) := (\psi_1(x), ..., \psi_d(x))^{\top}$ for some feature space $\mathcal{F}$.
(2) Create the image sequence $\mathcal{S}^{\psi} = \left\{z_i^{\psi} = (\psi(x_i), y_i)\right\}_{i=1}^{m}$ from the original training set $\mathcal{S}$.
(3) Train a linear predictor h against $\mathcal{S}^{\psi}$.
(4) Predict the label or the output of a new point x^{new} by $h^{\psi}(x^{\text{new}}) := h \circ \psi(x^{\text{new}}) = h(\psi(x^{\text{new}}))$

Note 9. For a specific learning task, the success of the learning paradigm in Note 6 depends on choosing an embedding ψ that provides a suitable deformation for the image of the data generating process (or the training dataset) to be as close as possible to what could be accurately addressed by the specific learning task.



(A) Inputs in the original space x_1 vs x_2

(B) Projection of inputs tot he feature space $\psi_1(x) = x_1$ vs $\psi_2(x) = 4x_2^2+1$

FIGURE 2.1. Projection of the inputs living in the original space to the feature space

Example 10. In SVM, ψ will make the image of the data distribution (close to being) linearly separable in the feature space $\mathcal{F}$, thus making the resulting learning algorithm a good learner for a given task (Figure 2.1). This requires prior knowledge of the problem (In Section 4, we see popular recipes for that).

Example 11. (Example) Consider the Soft-SVM in "Handout 8: Support Vector Machines". Consider a given embedding $\psi : \mathcal{X} \rightarrow \mathcal{F}$ with $\psi(x) = (\psi_1(x), \psi_2(x) \ldots)^\top$. The learning rule becomes

$$h^{\psi}(x) = h(\psi(x)) = \text{sign}(\langle w, \psi(x)\rangle + b). \tag{2.1}$$

In "Handout 8, Problem 24 becomes

Solve

$$\text{Solve: } (w^*, b^*, \xi^*) = \underset{(w,b,\xi)}{\arg\min} \left(\lambda \|w\|_2^2 + \frac{1}{m}\sum_{i=1}^{m} \xi_i \right)$$

$$\text{subject to: } y_i(\langle w, \psi(x_i)\rangle + b) \geq 1 - \xi_i, \ \forall i = 1, \ldots, m$$

$$\xi_i \geq 0, \ \forall i = 1, \ldots, m$$

Note 12. Feature space $\mathcal{F}$ is preferably a Hilbert space due to Lemma 15 that enables the Kernel trick via the representation Theorem 21. Eg, a Euclidean space such as $\mathbb{R}^d$ for some d. That includes infinite dimensional spaces.

Note 13. Any function that maps the original instances $\mathcal{X}$ into some Hilbert space $\mathcal{F}$ can be used as feature mapping ψ .

Recal Inner product and Hilbert spaces

Definition 14. A Hilbert space $\mathcal{F}$ is a a real or complex inner product (hence vector) space which is also complete[a] metric space with respect to the distance function (norm) $\|\cdot\| : \mathcal{F} \to \mathbb{R}$ induced by the inner product $\langle \cdot, \cdot \rangle : \mathcal{F} \times \mathcal{F} \to \mathbb{C}$ as $\|\psi\| = \sqrt{\langle \psi, \psi \rangle}$ for all $\psi \in \mathcal{F}$.

Lemma 15. *If $\mathcal{X}$ is a linear subspace of a Hilbert space, then every $x \in \mathcal{X}$ can be written as $x = u + v$ where $u \in \mathcal{X}$ and $\langle u, v \rangle = 0$ for all $v \in \mathcal{X}$.*

[a] I.e., every Cauchy sequence converges to some element of $\mathcal{F}$

Note 16. Using a $\psi(x) = (\psi_1(x), \psi_2(x) \ldots)^\top$ that is high dimensional (d is too large) may improve accuracy (expressiveness) of the learner (e.g. recall in polynomial regression increasing the polynomial degree). However this increases the computational effort/cost required to perform calculations to minimize the associated risk function in the high dimensional space, as well as we need more data. This is addressed via the Kernel trick.

3. The Kernel trick

Note 17. We discuss a duality in the learning Problem 18 that facilitates the implementation of the extension to a possibly high dimensional feature space (hence improving the expressiveness/accuracy) by using kernel functions (hence reducing dimensionality, computational cost, and required data size).

Problem 18. (RLM learning problem) Consider a prediction rule $h : \mathcal{X} \to \mathcal{Y}$ with $h(x) = \langle w, \psi(x) \rangle$ which is trained against a training sample $\{z_i = (x_i, y_i)\}_{i=1}^m$ with the following general optimization problem

$$\underset{w}{\text{minimize}} \left(\; f(\langle w, \psi(x_1) \rangle, \ldots, \langle w, \psi(x_m) \rangle) \; + \; R(\|w\|) \; \right), \tag{3.1}$$

where $f : \mathbb{R}^m \to \mathbb{R}$ is an arbitrary function and $R : \mathbb{R}_+ \to \mathbb{R}$ is a monotonically non-decreasing function.

Example 19. (Cont. Example 11) Soft SVM problem has a solution equivalent to (see Proposition 29, Handout 8 Support Vector Machines)

$$(w^*, b^*) = \underset{(w,b)}{\arg\min} \left(\frac{1}{m} \sum_{i=1}^m \max(0, 1 - y_i (\langle w, \psi(x_i) \rangle + b)) + \lambda \|w\|_2^2 \right)$$

then in terms of (3.1) we get

$$f(\alpha_1, \ldots, \alpha_m) = \frac{1}{m} \sum_{i=1}^m \max\{0, 1 - y_i \alpha_i\}, \text{ and } R(\beta) = \lambda \beta^2$$

Definition 20. Kernel function K is defined as $K : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$ with $K(x, x') = \langle \psi(x), \psi(x') \rangle$ given an embedding $\psi(x)$ of some domain space $\mathcal{X}$ into some Hilbert space $\mathcal{F}$. Kernel functions describe inner products in the feature space $\mathcal{F}$.

Theorem 21. *(Representation theorem) Assume mapping $\psi : \mathcal{X} \to \mathcal{F}$ where $\mathcal{F}$ is a Hilbert space. There exists a vector $\alpha \in \mathbb{R}^m$ such that $w = \sum_{i=1}^{m} \alpha_i \psi(x_i)$ is the optimal solution of (3.1) in Problem 18.*

Proof. Let w^* be the optimal solution of (3.1). Because w^* is element of Hilbert space, it can be written as $w^* = \sum_{i=1}^{m} \alpha_i \psi(x_i) + u$ where $\langle u, \psi(x_i) \rangle = 0$ for all $i = 1, ..., m$. Set $w := w^* - u$.

Because $\|w^*\|^2 = \|w\|^2 + \|u\|^2$ it is $\|w\| \leq \|w^*\|$ implying that

$$R(\|w\|) \leq R(\|w^*\|).$$

Because $\langle w, \psi(x_i) \rangle = \langle w^* - u, \psi(x_i) \rangle = \langle w^*, \psi(x_i) \rangle$ for all $i = 1, ..., m$, it is

$$f(\langle w, \psi(x_1) \rangle, ..., \langle w, \psi(x_m) \rangle) = f(\langle w^*, \psi(x_1) \rangle, ..., \langle w^*, \psi(x_m) \rangle)$$

Then the objective function of (3.1) at w is less than or equal to that of the minimizer w^* which implies that $w = \sum_{i=1}^{m} \alpha_i \psi(x_i)$ is an optimal solution. □

Note 22. Let $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel function with $K(x, x') = \langle \psi(x), \psi(x') \rangle$. According to the representation Theorem 21, the Learning Problem 18, can be equivalently addressed by re-writing the learning predictive rule as

$$h_\alpha(x) = \sum_{i=1}^{m} \alpha_i K(x_i, x)$$

and learning $\{\alpha_i\}$ as the solutions of

$$\text{(3.2)}\quad \underset{\alpha}{\text{minimize}} \left(f\left(\sum_{i=1}^{m} \alpha_i K(x_i, x_1), ..., \sum_{i=1}^{m} \alpha_i K(x_i, x_m) \right) + R\left(\sqrt{\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j K(x_i, x_j)} \right) \right),$$

This is because

$$\langle w, \psi(x_j) \rangle = \langle \sum_{i=1}^{m} \alpha_i \psi(x_i), \psi(x_j) \rangle = \sum_{i=1}^{m} \alpha_i \langle \psi(x_i), \psi(x_j) \rangle = \sum_{i=1}^{m} \alpha_i K(x_i, x_j)$$

and

$$\|w\|^2 = \langle \sum_{i=1}^{m} \alpha_i \psi(x_i), \sum_{j=1}^{m} \alpha_j \psi(x_j) \rangle = \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \langle \psi(x_i), \psi(x_j) \rangle = \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j K(x_i, x_j)$$

Note 23. In Learning Problem 18, direct access to elements $\psi(\cdot)$ in the feature space is not necessary because equivalently one can calculate or just specify the associated kernel function (that is inner products in the feature space).

Example 24. (Cont. Example 19) In Soft SVM the predictive rule becomes $h(x) = \text{sign}\left(\sum_j \alpha_j K(x_i, x_j) \right)$ and the learning task becomes

$$\underset{\alpha}{\text{minimize}} \left(\lambda \sum_i \sum_j \alpha_i \alpha_j K(x_i, x_j) + \frac{1}{m} \sum_i \max\left(0, 1 - y_i \sum_j \alpha_j K(x_i, x_j) \right) \right)$$

which can be addressed via SGD. This form of SVM is called Kernel SVM because we can just directly specify the kernel function $K(\cdot,\cdot)$ without the need to even think about feature mapping $\psi(\cdot)$ (which is eliminated and replaced by the kernel).

Example 25. (Polynomial Kernels) Let $x \in \mathcal{X} \subseteq \mathbb{R}^n$. We want to extend the linear mapping $x \mapsto \langle w, x\rangle$ to the k degree polynomial mapping $x \mapsto h(x)$. The multivariate polynomial can be written as $h(x) = \langle w, \psi(x)\rangle$, where $\psi : \mathbb{R}^n \to \mathbb{R}^d$ with $\psi(x)$ being a vector of elements $\psi_J(x) = \frac{\sqrt{k!}}{\sqrt{j_0!}}(\sqrt{c})^{j_0}\prod_{i=1}^n \frac{(k!)^{\frac{1}{2n}}}{(j_i!)^{\frac{1}{2}}}(x_i)^{j_i}$ for $J = (j_0, ..., j_n)^\top$ and $j_0 + j_1 + \cdots + j_n = k$. This learning problem can be equivalently be addressed with the k degree polynomial kernel

$$K(x,t) = (c + \langle x, t\rangle)^k \text{; where } x, t \in \mathcal{X}$$

Solution: It is

$$K(x,t) = (c+\langle x,t\rangle)^k = \text{ (by setting } x_0 = t_0 = \sqrt{c}) = \left(\sum_{j=0}^n x_j t_j\right)^k = \text{(by Multinomial theorem)} =$$

$$= \sum_{J:j_0+j_1+\cdots+j_n=k} \frac{k!}{j_0! j_1! \cdots j_n!}\prod_{i=0}^n (x_i t_i)^{j_i} = \sum_{J:j_0+j_1+\cdots+j_n=k}\left[\underbrace{\prod_{i=0}^n \frac{(k!)^{\frac{1}{2n}}}{(j_i!)^{\frac{1}{2}}}(x_i)^{j_i}}_{\psi_J(x)}\right]\left[\prod_{i=0}^n \frac{(k!)^{\frac{1}{2n}}}{(j_i!)^{\frac{1}{2}}}(t_i)^{j_i}\right]$$

$$= \langle \psi(x), \psi(x')\rangle \qquad \text{where } \psi(x) \text{ is as defined.}$$

Solution. ▶ In Figure 1.1c, points can be distinguished by some ellipse, hence ψ could be defined as a vector of monomials up to order. Alternatively a degree 2 polynomial kernel could be used.

Example 26. (Radial basis kernel) Let the original input space be $x \in \mathcal{X} \subseteq \mathbb{R}$. Consider the Radial Basis Functions Kernel (or Gaussian kernel)

$$K(x,x') = \exp\left(-\frac{1}{2\sigma^2}\|x-x'\|_2^2\right).$$

Show that it is a kernel indeed, by presenting it as an inner product in a feature space of infinite dimension, and state the bases of the mapping $\psi(\cdot)$.
Solution: It is

$$\begin{aligned}
K(x,x') =& \exp\left(-\frac{1}{2\sigma^2}\|x-x'\|_2^2\right) = \exp\left(\frac{1}{\sigma^2}xx' - \frac{1}{2}x^2 - \frac{1}{2}(x')^2\right)\\
=& \exp\left(\frac{1}{\sigma^2}xx'\right)\exp\left(-\frac{1}{2\sigma^2}x^2\right)\exp\left(-\frac{1}{2\sigma^2}(x')^2\right)\\
=& \sum_{k=0}^\infty \frac{(xx'/\sigma^2)^k}{k!}\exp\left(-\frac{1}{2\sigma^2}x^2\right)\exp\left(-\frac{1}{2\sigma^2}(x')^2\right) \text{ (...by Taylor Expansion)}\\
=& \sum_{k=0}^\infty\left[\frac{x^k}{\sqrt{k!}\sigma^k}\exp\left(-\frac{1}{2\sigma^2}x^2\right)\right]\left[\frac{(x')^k}{\sqrt{k!}\sigma^k}\exp\left(-\frac{1}{2\sigma^2}(x')^2\right)\right]
\end{aligned}$$

hence it is $K(x,x') = \langle \psi(x), \psi(x')\rangle$ with $\psi_k(x) = \frac{x^k}{\sqrt{k!}\sigma^k}\exp\left(-\frac{1}{2\sigma^2}x^2\right)$.

Note 27. The kernel formulated as an inner product in a feature space allows us to build interesting extensions of many well known algorithms by making use of the kernel trick and without the need to have direct access to the feature space (E.g. Example 24).

Note 28. Specifying a kernel function K is a way to express prior knowledge without the need to have direct access to the feature space. This is consequence of the Representation Theorem 21 that kernel is the inner product of feature mappings ψ which sufficiently replaces them in the learning problem, and the fact that ψ is a way to express and utilize prior knowledge about the problem at hand.

Note 29. Theorem 32 provides sufficient and necessary conditions to check whether the specified function $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is indeed a kernel function; i.e. if K can be written as inner product $K(x, x') = \langle \psi(x), \psi(x') \rangle$ of feature functions $\psi(x)$.

Definition 30. Gram matrix is called the $m \times m$ matrix G s.t. $[G]_{i,j} = K(x_i, x_j)$.

Definition 31. A symmetric function $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is positive semi-definite if its Gram matrix G, $[G]_{i,j} = K(x_i, x_j)$, is a positive semi-definite matrix.

Theorem 32. *A symmetric function $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ implements an inner product in some Hilbert space is a valid kernel function if and only if it is positive semi-definite i.e. its Gram matrix G , $[G]_{i,j} = K(x_i, x_j)$, is a positive semi-definite matrix.*

Proof. Assume K is a valid kernel function (i.e. it implements an inner product in some Hilbert space) $K(x, x') = \langle \psi(x), \psi(x') \rangle$; let's consider $\psi : \mathcal{X} \rightarrow \mathbb{R}^d$ for simplicity. Let G be its Gram matrix with $G = \Psi^\top \Psi$ and $\psi(x_i)$ is the i-th column of Ψ. For any $\xi \in \mathbb{R}^d - \{0\}$

$$\xi^\top G \xi = \sum_i \sum_j \xi_i K(x_i, x_j) \xi_j = \sum_i \sum_j \xi_i \langle \psi(x_i), \psi(x_j) \rangle \xi_j = \sum_i \sum_j \langle \xi_i \psi(x_i), \psi(x_j) \xi_j \rangle$$

$$= \langle \sum_i \xi_i \psi(x_i), \sum_j \psi(x_j) \xi_j \rangle = \left\| \sum_i \xi_i \psi(x_i) \right\|_2^2 \geq 0$$

Assume the symmetric function $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is positive semi-definite. Let $\mathbb{R}^f = \{f : \mathcal{X} \rightarrow \mathbb{R}\}$. For $x \in \mathcal{X}$ let function ψ over $\mathbb{R}^f$ with $\psi(x) = K(\cdot, x)$. This allows to define a vector space consisting of all the linear combinations of elements of the form $K(\cdot, x)$, having an inner product

$$\langle \sum_i \alpha_i K(\cdot, x_i), \sum_j \beta_j K(\cdot, x_j) \rangle = \sum_i \sum_i \alpha_i \beta_j \underbrace{\langle K(\cdot, x_i), K(\cdot, x_j) \rangle}_{=K(x_i, x_j)}.$$

This is satisfies all the properties of inner product, s.t. it is symmetric, linearity, positive definite as $K(x, x') \geq 0$. Then there is some feature vector ψ such that $K(x, x') = \langle \psi(x), \psi(x') \rangle$. $\square$

Note 33. A powerful technique for constructing new kernels is to build them out of simpler kernels as building blocks. Below are some properties. Assume $K_1 : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ and $K_2 : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ are valid kernels, then the following are kernels too

(1) $K(x,x') = K_1(x,x') + K_2(x,x')$
(2) $K(x,x') = K_1(x,x')\, K_2(x,x')$
(3) $K(x,x') = K_1(x_1,x_1') + K_2(x_2,x_2')$, where $x = (x_1,x_2)^\top$, $x' = (x_1',x_2')^\top$
(4) $K(x,x') = K_1(x_1,x_1')\, K_2(x_2,x_2')$, where $x = (x_1,x_2)^\top$, $x' = (x_1',x_2')^\top$
(5) $K(x,x') = f(x)\, K_1(x,x')\, f(x')$ for any function f
(6) $K(x,x') = K_1(f(x), f(x'))$ for any function f

See the proof as a solution to an Example

Proof. We present the first two and the rest are proved similarly.

For (1). Let Gram matrix, G_j induced by kernel function K_j. For any $\xi \in \mathbb{R}^d - \{0\}$

$$\xi^\top G_3 \xi = \xi^\top (G_1 + G_2)\, \xi = \xi^\top G_1 \xi + \xi^\top G_2 \xi \geq 0$$

For (2). Assume that $K_j(x,x') = (\psi_j(x))^\top \psi_j(x)$. Then

$$\begin{aligned} K\left(x,x'\right) &= K_1\left(x,x'\right) K_2\left(x,x'\right) = (\psi_1(x))^\top \psi_1\left(x'\right) (\psi_2(x))^\top \psi_2\left(x'\right) \\ &= (\psi_1(x))^\top \psi_2(x) \left(\psi_1\left(x'\right)\right)^\top \psi_2\left(x'\right) = \left((\psi_1(x))^\top \psi_2(x)\right)^\top \left(\psi_1\left(x'\right)\right)^\top \psi_2\left(x'\right) \end{aligned}$$

which can be represented as an inner product of feature vectors.

□

Note 34. The concept of a kernel formulated as an inner product in a feature space allows us to build interesting extensions of many well known algorithms by making use of the kernel trick. One example was the Kernel SVM. Some other popular cases are Gaussian process regression, and Kernel PCA.