Machine Learning and Neural Networks III (MATH3431)

Epiphany term

Georgios P. Karagiannis

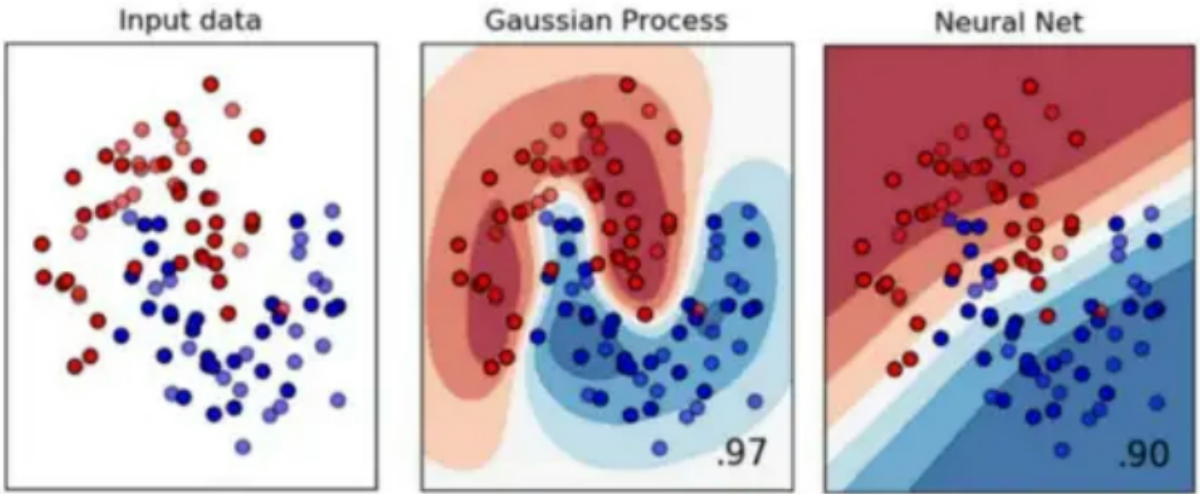
georgios.karagiannis@durham.ac.uk

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

2024/03/03 at 16:29:21

Concepts

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



Reading list

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

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-recogniti`
- 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 It is very comprehensive (compared to others) and it goes deep enough for the module although it may be a bit outdated.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:

- 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.

- Murphy, K. P. (2012). Machine learning: a probabilistic perspective. MIT press.
  - A popular textbook in 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 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.

- Barber, D. (2012). Bayesian reasoning and machine learning. Cambridge University Press.
  - A textbook in machine learning methods from a Bayesian point of view. It discusses all the concepts introduced apart from ANN and stochastic gradient algorithms. It aims to be more 'statistical' than those of Murphy and Bishop. The level on difficulty is easy.

- 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.
  - Theoretical aspects about machine learning algorithms. The level on difficulty is advanced as it requires moderate knowledge of probability.

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, 2024

Handout 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 loose 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. (loose) Notation & definitions in learning

Definition 1. 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 2. We often denote the set of hypothesis as $\mathcal{H}$; i.e. $h \in \mathcal{H}$.

Example 3. (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 $\mathcal{H} = \left\{ x \to \langle w, x \rangle : w \in \mathbb{R}^d \right\}$.

Example 4. (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 5. Training data set $\mathcal{S}$ of size m is any finite sequence of pairs $(z_i = (x_i, y_i) ; i = 1, ..., m)$ in $\mathcal{X} \times \mathcal{Y}$; i.e. $\mathcal{S} = \{(x_i, y_i) ; i = 1, ..., m\}$. This is the information that the learner has assess.

Definition 6. Data generation model $g(\cdot)$ is the probability distribution over $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, unknown to the learner that has generated the data. E.g. $z \sim g$.

Definition 7. We denote as $\mathfrak{A}(\mathcal{S})$ the hypothesis (outcome) that a learning algorithm $\mathfrak{A}$ returns given training sample S.

Definition 8. (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 9. (Cont. Example 3) 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 10. (Cont. Example 4) In binary classification problems with hypothesis $h : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{Y} = \{0, 1\}$ is discrete, a loss function can be

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

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

Definition 11. (Risk function) The risk function $R_g(h)$ 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.

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

Remark 12. In learning, an ideal way to obtain an optimal predictor h^* is to compute the minimizer of the risk; i.e.

$$h^* = \arg\min_{\forall h}(R_g(h)) \tag{2.2}$$

Example 13. (Cont. Ex. 9) The risk function is $R_g(h) = \mathrm{E}_{z\sim g}(h(x)-y)^2$, 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.

Note 14. 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, one may use the Empirical risk function instead of the Risk function in (2.2).

Definition 15. (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.

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

Remark 16. Given Empirical Risk Function (ERF) $\hat{R}_S(h)$ of h the optimal predictor h^* is the minimizer of the ERF; i.e.

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

Example 17. (Cont. Example 13) Given given sample $S = \{(x_i, y_i); i = 1, ..., m\}$ the empirical risk function is $\hat{R}_S(h) = \frac{1}{m}\sum_{i=1}^{m}(h(x_i)-y_i)^2$.

Example 18. (Cont. Example 10) Given given sample $S = \{(x_i, y_i); i = 1, ..., m\}$ the empirical risk function is $\hat{R}_S(h) = \frac{1}{m}\sum_{i=1}^{m}1(h(x_i) \neq y_i)$.

Remark 19. If the Hypothesis set $\mathcal{H}$ is a known parametric family of functions; i.e. $\mathcal{H} = \{h_w(\cdot); w \in \mathcal{W}\}$ parameterized by unknown $w \in \mathcal{W}$, then we can equivalently consider $\mathcal{H} = \{w \in \mathcal{W}\} = \mathcal{W}$ keeping in mind that the learner's output is restricted to $h_w(\cdot)$.

Example 20. Consider the multiple linear regression problem with regressors $x \in \mathcal{X} \subseteq \mathbb{R}^d$ and response $y \in \mathcal{Y} \subseteq \mathbb{R}$. Because it involves only linear functions as predictors $h_w(x) = \langle w, x\rangle$, we could consider a hypothesis class $\mathcal{H} = \left\{w \in \mathbb{R}^d\right\} = \mathbb{R}^d$ and loss function loss $\ell(w,(x,y)) = (\langle w, x\rangle - y)^2$ for computational simplicity. The latter will be mainly used.

Example 21. 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

$$R_g(\theta) = \mathrm{E}_{z\sim g}\left(\log\left(\frac{g(z)}{f_\theta(z)}\right)\right) = \mathrm{E}_{z\sim g}(\log(g(z))) - \mathrm{E}_{z\sim g}(\log(f_\theta(z))) \tag{2.4}$$

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 (2.4) 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 $R_S(\theta) = \frac{1}{m}\sum_{i=1}^{m}(-\log(f_\theta(z_i)))$.

Definition 22. 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 23. 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 $\ell_{\mathrm{sq}}(h, (x, y)) = (h(x) - y)^2$.

Appendix A. Useful things

Below are some standard notation used as default in the notes except in cases that is defined otherwise.

- $q-$norm: When $x \in \mathbb{R}^d$ $\|x\|_q := \left(\sum_{j=1}^{d} x_j^q\right)^{1/q}$
- Manhattan norm: When $x \in \mathbb{R}^d$ $\|x\|_1 := \sum_{j=1}^{d} |x_j|$
- Euclidean norm: When $x \in \mathbb{R}^d$ $\|x\|_2 := \sqrt{\sum_{j=1}^{d} x_j^2}$. When $\|\cdot\|$ we will assume the Euclidean norm.
- Infinity norm or maximum norm: $\|x\|_\infty := \max_{\forall j} |x_j|$
- Inner product of x , y: If $x, y \in \mathbb{R}^d$ then $\langle x, y\rangle = x^\top y$. So $\langle x, x\rangle = \|x\|^2$

Also some standard formulas.

- Jensens' inequality: If If $x \in \mathbb{R}^d$ and $f : \mathbb{R}^d \to \mathbb{R}$ then

$$\begin{cases} f(\mathrm{E}(x)) \le \mathrm{E}(f(x)) & \text{if } f \text{ is convex} \\ f(\mathrm{E}(x)) \ge \mathrm{E}(f(x)) & \text{if } f \text{ is concave} \end{cases}$$

- Cauchy–Schwarz inequality: If $x, y \in \mathbb{R}^d$ then $|\langle x, y\rangle|^2 \le \langle x, x\rangle\langle y, y\rangle$ equiv. $|\langle x, y\rangle| \le \|x\|\,\|y\|$.

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2024

Handout 2: Elements of convex learning problems

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

Aim. To introduce elements of convexity, Lipschitzness, and smoothness that can be used 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

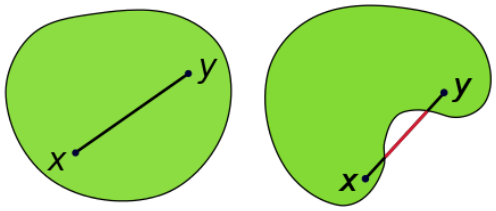
Note 1. We introduce concepts of convexity and smoothness that facilitate the analysis and understanding of the learning problems and their solutions that we will discuss (eg stochastic gradient descent, SVM) later on. Also learning problems with such characteristics can be learned more efficiently.

Note 2. Some of the ML problems discussed in the course (eg, Artificial neural networks, Gaussian process regression) are non-convex. To overcome this problem, we will introduce the concept of surrogate loss function that allows a non-convex problem to be handled with the tools introduced int he convex setting.

2. Convexity

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

Note 4. Namely, a set C is convex if for any $u, v \in C$, the line segment between u and v is contained in C.



(A) (B)

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

Example 5. For instance $\mathbb{R}^d$ for $d \geq 1$ is a convex set.

Definition 6. 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 $\alpha \in [0, 1]$

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

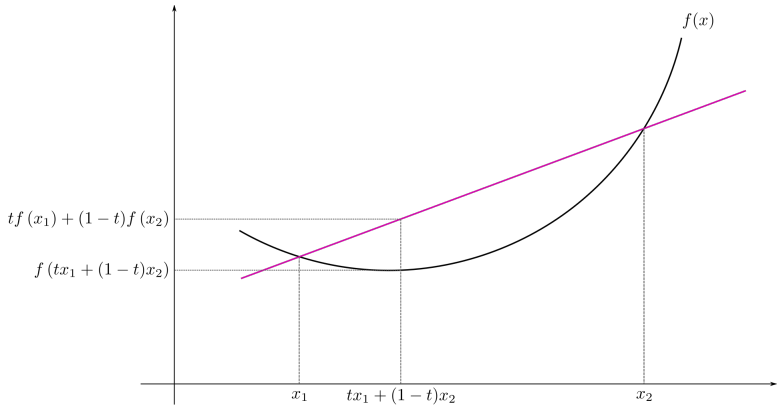


FIGURE 2.2. A convex function

Example 7. The function $f : \mathbb{R} \to \mathbb{R}_+$ with $f(x) = x^2$ is convex function. For any $u, v \in C$ and for any $\alpha \in [0, 1]$ it is

$$\left(\alpha u + (1-\alpha)\, v\right)^2 - \alpha\left(u\right)^2 + (1-\alpha)\left(v\right)^2 = -\alpha\left(1-\alpha\right)\left(u-v\right)^2 \leq 0$$

Proposition 8. *Every local minimum of a convex function is the global minimum.*

Proposition 9. *Let $f : C \to \mathbb{R}$ be convex function. The tangent of f at $w \in C$ is below f, namely*

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

Proposition 10. *Let $f : \mathbb{R}^d \to \mathbb{R}$ such that $f\left(w\right) = g\left(\left\langle w, x\right\rangle + y\right)$ 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 11. 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 10.

Proposition 12. *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 $w_j > 0$*

Solution.

(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 13. $g(x) = |x|$ is convex according to Example 12, as $g(x) = |x| = \max(-x, x)$.

3. Lipschitzness

Definition 14. Let $C \in \mathbb{R}^d$. Function $f : \mathbb{R}^d \to \mathbb{R}^k$ is ρ-Lipschitz over C if for every $w_1, w_2 \in C$ we have that

$$\|f(w_1) - f(w_2)\| \leq \rho \|w_1 - w_2\|. \qquad \text{Lipschitz condition} \tag{3.1}$$

Conclusion 15. That means: a Lipschitz function $f(x)$ cannot change too drastically wrt x.

Example 16. 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 $C = \{x \in \mathbb{R} : |x| < \rho/2\}$.

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

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

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

Theorem 17. *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.*

Solution. See Exercise 2 from the exercise sheet

Example 18. 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 19. So, given Examples 16 and 18, 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 $\|w\| < \rho$.

4. Smoothness

Definition 20. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is β-smooth if its gradient is β-Lipschitz; namely for all $v, w \in \mathbb{R}^d$

$$\|\nabla f(w_1) - \nabla f(w_2)\| \leq \beta\,\|w_1 - w_2\|\,. \tag{4.1}$$

Theorem 21. *Function $f : \mathbb{R}^d \to \mathbb{R}$ is β-smooth iff*

$$f(v) \leq f(w) + \langle \nabla f(w), v - w\rangle + \frac{\beta}{2}\,\|v - w\|^2 \tag{4.2}$$

Theorem 22. *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.*

Proof. See Exercise 3 from the Exercise sheet □

Example 23. 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.

Solution. It is $f(w) = g(< w, x > +y)$ for $g(a) = a^2$. g is 2-smooth since

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

Hence from Theorem 22, f is $\left(2\,\|x\|^2\right)$-smooth.

Example 24. 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.

Solution. Follows from Example 23.

5. Convex learning problems

Definition 25. 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 ℓ is a convex function for each example $z \in \mathcal{Z}$.

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 $y \in \mathbb{R}$. This imposes a convex learning problem due to Examples 5 and 12.

Definition 27. 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 $\|w\| \leq B$, and the loss function $\ell(\cdot, z)$ is convex and ρ-Lipschitz function for all $z \in \mathcal{Z}$.

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-Lipschitz-Bounded Learning Problem if $\mathcal{H} = \left\{w \in \mathbb{R}^d : \|w\| \leq B\right\}$ due to Examples 12, and 16(2).

Definition 29. 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 $\|w\| \leq B$, and the loss function $\ell(\cdot, z)$ is convex, nonnegative, and β-smooth 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-Smooth-Bounded Learning Problem if $\mathcal{H} = \left\{w \in \mathbb{R}^d : \|w\| \leq B\right\}$ due to Examples 12, and 24.

Proposition 31. *If ℓ is a convex loss function and the class $\mathcal{H}$ is convex, then the ERM$_\mathcal{H}$ problem, of minimizing the empirical risk $\hat{R}_\mathcal{S}(w)$ over $\mathcal{H}$, is a convex optimization problem (that is, a problem of minimizing a convex function over a convex set).*

Proof. The ERM$_\mathcal{H}$ problem is

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

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 Proposition (12). Hence ERM rule is a problem of minimizing a convex function subject to the constraint that the solution should be in a convex set. □

Example 32. 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

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

or

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

is a convex learning problem –from Proposition 31.

Note 33. Problems like that in Proposition 31 can be efficiently solved with algorithms such as Stochastic Gradients Descent to be introduced later.

6. Non-convex learning problems (surrogate treatment)

Remark 34. A learning problem may involve non-convex loss function $\ell(w,z)$ which implies a non-convex risk function $R_g(w)$. However, our learning algorithm will be analyzed in the convex setting. A suitable treatment to overcome this difficulty 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 35. 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

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

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

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

which is convex (Example 12) wrt w. Note that:

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

Then we can compute

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

instead of

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

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

Remark 36. (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 $\tilde{w}_{\text{alg}}$ is the output of the learning algorithm under $\tilde{R}_g(\cdot)$ then we have the upper bound

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

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.

7. Strong convexity

Note 37. Strong convexity is a central concept in regularization, e.g. Ridge, as it makes a convex loss function strongly convex by adding a shrinkage term.

Definition 38. (Strongly convex functions) A function f is λ-strongly convex function is for all w, u, and $\alpha \in (0,1)$ we have

$$f(\alpha w + (1-\alpha)u) \leq af(w) + (1-\alpha)f(u) - \frac{\lambda}{2}\alpha(1-\alpha)\|w-u\|^2 \tag{7.1}$$

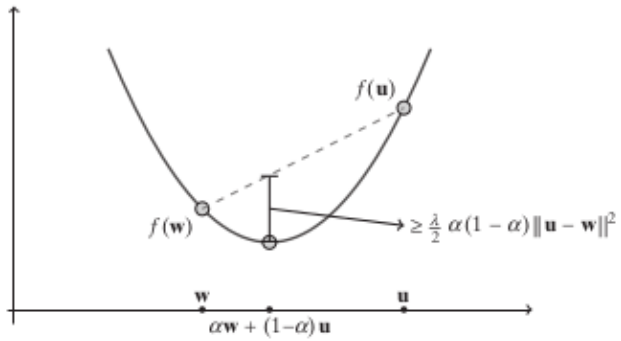


FIGURE 7.1. Strongly convex function

Proposition 39.

(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

Proof. Both can be checked from the definition by substitution. □

Lemma 40. *If* f *is* λ-strongly convex and $w^* = \arg\min_w f(w)$ is a minimizer of f then for any w

$$f(w) - f(w^*) \geq \frac{\lambda}{2}\|w-w^*\|^2$$

Proof. Exercise 7 in the Exercise sheet. □

Proposition 41. *If* ℓ *is a convex loss function and the class* $\mathcal{H}$ *is convex, then the Ridge ERM*$_\mathcal{H}$ *problem, with learning rule*

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

is a 2λ-strongly convex learning problem.

Proposition 42. *(*ERM *with Ridge regularization) If* ℓ *is a convex loss function, 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 the ERM*$_\mathcal{H}$ *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 Proposition 31, $\lambda\|\cdot\|_2^2$ is 2λ-strongly convex, hence $\hat{R}_\mathcal{S}(w) + \lambda\|w\|_2^2$ is a 2λ-strongly convex function. Hence the above ERM$_\mathcal{H}$ problem is a strongly convex optimization problem. □

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2024

Handout 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. (too advanced)

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. It may be easier to work with expectations: by using Markov inequality (1.1) becomes

$$\Pr_{\mathcal{S}\sim g}\left(R_g\left(\mathfrak{A}\left(\mathcal{S}_m\right)\right) - \min_{h'\in\mathcal{H}}\left(R_g\left(h'\right)\right) \leq \epsilon\right) \geq 1 - \frac{1}{\epsilon}\mathrm{E}_{\mathcal{S}\sim g}\left(R_g\left(\mathfrak{A}\left(\mathcal{S}_m\right)\right) - \min_{h'\in\mathcal{H}}\left(R_g\left(h'\right)\right)\right) \tag{1.2}$$

and hence we need to work with expectations and bounded above.

Hint: Markov inequality $\Pr(X \geq a) \leq \frac{1}{a}\mathrm{E}(X)$ for $X > 0$.

Remark 6. About 1.2: The accuracy parameter ϵ determines how far the output rule $\mathfrak{A}(\mathcal{S}_m)$ of $\mathfrak{A}$ can be from the optimal one (this corresponds to the 'approximately correct'). The confidence parameter δ indicates how likely the classifier is to meet that accuracy requirement (corresponds to the "probably" part of "PAC").

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

Note 7. Let $R^* = \min_{\forall h}(R(h))$ 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

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

Note 8. Over-fitting can be represented by $R_g(\mathfrak{A}(\mathcal{S})) - \hat{R}_{\mathcal{S}}(\mathfrak{A}(\mathcal{S}))$. However, $\hat{R}_{\mathcal{S}}(\cdot)$ 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 9. For a learning algorithm $\mathfrak{A}$, as a measure of over-fitting we consider the expected difference between true Risk and empirical Risk

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

Definition 10. We say that learning algorithm $\mathfrak{A}$ suffers from over-fitting when (2.2) is 'too' large.

Note 11. The expected Risk of a learning algorithm $\mathfrak{A}(\mathcal{S})$ can be decomposed as

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

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

Proposition 13. *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*

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

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

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

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 $\Pr\left(\xi \leq \left(\epsilon\delta\right)/\delta\right) = 1-\delta$ implies $\Pr\left(\xi \leq \epsilon\right) = 1-\delta$. Now substitute back ξ and we conclude the proof. □

Note 14. Hence, we aim to design a learning algorithm $\mathfrak{A}\left(\mathcal{S}\right)$ 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 $\mathfrak{A}\left(\mathcal{S}\right)$.

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 $\epsilon : \mathbb{N} \to \mathbb{R}$ if for every distribution g

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

Note 18. The following result demonstrates the direct association of stability and over-fitting as defined in Definitions 10 & 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 $\mathcal{H}_i=\left\{w\in\mathbb{R}^{d}:\left\|w\right\|_{2}<i\right\}$ 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 $\tilde{R}_{\mathcal{S}}(w) = \hat{R}_{\mathcal{S}}(w) + \lambda \|w\|_2^2$. $\tilde{R}_{\mathcal{S}}(\cdot)$ is 2λ-strongly convex as the sum of a convex function $\hat{R}_{\mathcal{S}}(\cdot)$ (Assumption 30) and a 2λ-strongly convex function $J(\cdot;\lambda) = \lambda \|\cdot\|_2^2$ (results directly from Definition 38 in Handout 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*

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

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

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

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 $\tilde{R}_{\mathcal{S}^{(i)}}\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right)\right) \leq \tilde{R}_{\mathcal{S}^{(i)}}(\mathfrak{A}(\mathcal{S}))$, it is

$$\tilde{R}_{\mathcal{S}}\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right)\right) - \tilde{R}_{\mathcal{S}}(\mathfrak{A}(\mathcal{S})) \leq \frac{\ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right), z_i\right) - \ell(\mathfrak{A}(\mathcal{S}), z_i)}{m} + \frac{\ell(\mathfrak{A}(\mathcal{S}), z') - \ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right), z'\right)}{m} \tag{4.5}$$

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

$$\lambda \left\|\mathfrak{A}\left(\mathcal{S}^{(i)}\right) - \mathfrak{A}(\mathcal{S})\right\| \leq \frac{\ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right), z_i\right) - \ell(\mathfrak{A}(\mathcal{S}), z_i)}{m} + \frac{\ell(\mathfrak{A}(\mathcal{S}), z') - \ell\left(\mathfrak{A}\left(\mathcal{S}^{(i)}\right), z'\right)}{m} \tag{4.6}$$

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

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

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

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

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

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

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

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

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

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 $J\left(\cdot;\lambda\right) = \lambda\left\|\cdot\right\|^2$, $\lambda > 0$, 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 $\epsilon\left(m\right) = 2\frac{\rho^2}{\lambda m}$; i.e.

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

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 $J\left(\cdot;\lambda\right) = \lambda\left\|\cdot\right\|^2$, $\lambda > 0$, and learning rule trained against an iid sample $\mathcal{S} = \{z_i\}_{i=1}^m$ from g has Expected Risk bound

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

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.* $\mathcal{H} = \left\{w \in \mathbb{R}^d : \left\|w\right\|_2 \leq B\right\}$.

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

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

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

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

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

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

Appendix A. Recall strong convex functions from Handout 2.

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

$$f\left(aw + (1-\alpha)u\right) \leq af\left(w\right) + (1-\alpha)f\left(u\right) - \frac{\lambda}{2}\alpha\left(1-\alpha\right)\left\|w-u\right\|^2$$

- The function $f\left(w\right) = \lambda\left\|w\right\|_2^2$ is 2λ-strongly convex
- If $f$is λ-strongly convex and g is convex then $f+g$ is λ-strongly convex
- If $f$is λ-strongly convex and u is a minimizer of f then for any w

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

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2023

Handout 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

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

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. Description

Problem 2. For the sake of notation simplicity and generalization, we will present Gradient Descent (GD) in the following minimization problem

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

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

Assumption 3. *Assume (for now) that $f : \mathbb{R}^d \to \mathbb{R}$ is a differentiable function.*

Algorithm 4. *Gradient Descent (GD) algorithm with learning rate $\eta_t > 0$ for the solution of the minimization problem (2.1)*

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

(1) *compute*

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla f\left(w^{(t)}\right) \tag{2.2}$$

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

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

Note 5. Recall that the gradient of $f : \mathbb{R}^d \to \mathbb{R}$ at w is

$$\nabla f(w) = \left(\frac{\partial}{\partial x_1} f(x), \ldots, \frac{\partial}{\partial x_d} f(x)\right)^\top \Bigg|_{x=w}.$$

Note 6. (Intuition) GD produces a chain $\left\{w^{(t)}\right\}$ that drifts towards a minimum w^*. It evolves directed towards the opposite direction than that of the gradient $\nabla f(\cdot)$ and at a rate controlled by the learning rate η_t.

Note 7. (More intuition) Consider the (1st order) Taylor polynomial for the approximation of $f(w)$ in a small area around u (i.e. $\|v - u\| = \text{small}$)

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

Assuming convexity for f, it is

$$f(u) \geq \underbrace{f(w) + \langle u - w, \nabla f(w) \rangle}_{=P(u;w)} \tag{2.3}$$

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 $P(w)$ of $f(w)$ at $w^{(t)}$

$$P\left(w; w^{(t)}\right) = f\left(w^{(t)}\right) + \langle w - w^{(t)}, \nabla f\left(w^{(t)}\right) \rangle. \tag{2.4}$$

while hoping that this mechanism would push the produced chain $\left\{w^{(t)}\right\}$ towards the minimum because of (2.3). Hence we could recursively minimize the linear approximation (2.4) and the distance between the current state $w^{(t)}$ and the next w value to produce $w^{(t+1)}$; namely

$$\begin{aligned} w^{(t+1)} &= \arg\min_{\forall w} \left(\frac{1}{2}\left\|w - w^{(t)}\right\|^2 + \eta P\left(w; w^{(t)}\right)\right) \\ &= \arg\min_{\forall w} \left(\frac{1}{2}\left\|w - w^{(t)}\right\|^2 + \eta\left(f\left(w^{(t)}\right) + \langle w - w^{(t)}, \nabla f\left(w^{(t+1)}\right)\rangle\right)\right) \\ &= w^{(t)} - \eta \nabla f\left(w^{(t)}\right) \end{aligned} \tag{2.5}$$

where parameter $\eta > 0$ controls the trade off in (2.5).

Note 8. 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)

$$w_{\text{GD}}^{(T)} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)} \tag{2.6}$$

(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

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

Note 9. GD output (e.g. in Note 8) 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 10. 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 9) 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) constant; $\eta_t = \eta$, 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) decreasing and converging to zero; $\eta_t \searrow$ 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) decreasing and converging to a tiny value 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) constant until an iteration T_0 and then decreasing; 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 $t \le T_0$) 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 ($t \le T_0$) is called burn-in; the values $\{w_t\}$ produced during the burn-in ($t \le T_0$) 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 11. There are several practical termination criteria that can be used in GD Algorithm 4(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.

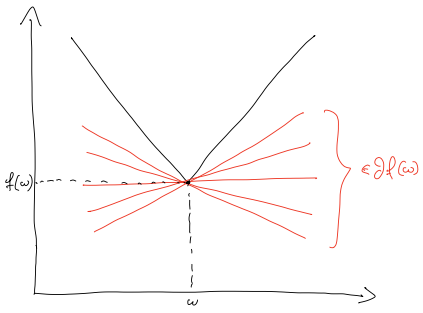
3. GD for non-differentiable functions (using sub-gradients)

Note 12. In several learning problems the function to be minimized is not differentiable. GD can be extended to address such problems with the use of subgradients.

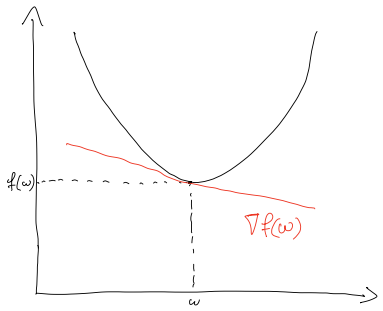
Definition 13. Vector v is called subgradient of a function $f : S \to \mathbb{R}$ at $w \in S$ if

$$\forall u \in S, \quad f(u) \geq f(w) + \langle u - w, v \rangle \tag{3.1}$$

Note 14. Essentially there may be more than one subgradients of the function at a specific point. As seen by (3.1), 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 (3.1) in the non-differentiable case



(B) gradient satisfying the equality in (3.1) in the differentiable case

Definition 15. The set of subgradients of function $f : S \to \mathbb{R}$ at $w \in S$ is denoted by $\partial f(w)$.

Algorithm 16. *The Gradient Descent algorithm using subgradients in non-differentiable cases, results by replacing the gradient $\nabla f\left(w^{(t)}\right)$ in (2.2) with any of subgradient v_t from the set of subgradients $\partial f\left(w^{(t)}\right)$ at $w^{(t)}$; namely (2.2) is replaced by*

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

3.1. **Construction of subgradient.**

Note 17. We discuss how to construct subgradients in practice.

Fact 18. *Some properties of subgradient sets that help for their 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 (3.1) 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 19. 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 18, 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 (3.1) it is

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

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

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

4. Analysis of Gradient descent (Algorithm 4)

Note 20. Recall we address the minimization Problem 2 under Assumptions 21.

Assumption 21. *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 $w_{GD}^{(T)} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$*

No need to memorize

Lemma 22. *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*

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

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

Note 23. To find an upper bound of the GD error, we try to bound the error $f\left(w_{\mathrm{GD}}^{(T)}\right) - f\left(w^*\right)$ with purpose to use Lemma 22.

Proposition 24. *Consider the minimization problem (2.1). Given Assumptions 21, the error can be bounded as*

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

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

Proof. It is[1]

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

□

Note 25. Proposition 24 shows that it is important to bound the (sub-)gradient in (4.2) in a meaningful manner. Lemma 26 shows that if we assume Lipschitzness as well, the (sub-)gradient has the so-called self-bounded behavior.

Lemma 26. [2] $f : S \to \mathbb{R}$ *is* ρ*-Lipschitz over an open convex set* S *if and only if for all* $w \in S$ *and* $v \in \partial f\left(w\right)$ *it is* $\left\|v\right\| \leq \rho$.

Proof. $\Longrightarrow$ Let $f : S \to \mathbb{R}$ be ρ-Lipschitz over convex set S, $w \in S$ and $v \in \partial f\left(w\right)$.

- Since S is open we get that there exist $\epsilon > 0$ such as $u := w + \epsilon\frac{v}{\left\|v\right\|}$ where $u \in S$. So $\left\langle u - w, v\right\rangle = \epsilon\left\|v\right\|$ and $\left\|u - w\right\| = \epsilon$.
- From the subgradient definition we get

$$f\left(u\right) - f\left(w\right) \geq \left\langle u - w, v\right\rangle = \epsilon\left\|v\right\|$$

- From the Lipschitzness of $f\left(\cdot\right)$ we get

$$f\left(u\right) - f\left(w\right) \leq \rho\left\|u - w\right\| = \rho\epsilon$$

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

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

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

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

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

Note 27. The following summaries Proposition 24 and Lemma 26 with respect to the GD algorithm satisfying Assumption 21.

Proposition 28. *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

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

where $\|\nabla f(\cdot)\| \leq \rho$.

Solution. Straightforward from Lemma 22 and Proposition 24.

Note 29. The following shows that a given 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.

Proposition 30. *(Cont Prop. 28) 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 $w_{\mathrm{GD}}^{(T)} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$. Then*

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

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

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

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

Proof. Part 1 is a simple substitution from Proposition 28, and part 2 is implied from part 1. □

Note 31. The result on Proposition 30 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).

Note 32. The above results from the analysis of the GD also hold for the GD with subgradients; just replace $\nabla f(\cdot)$ with any v_t such that $v_t \in \partial f(\cdot)$.

5. Examples [3]

Example 33. Consider the simple Normal linear regression problem where the dataset $\{z_i = (y_i, x_i)\}_{i=1}^n \in \mathcal{D}$ is generated from a Normal data generating model

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

[3] Code is available in `https://github.com/georgios-stats/Machine_Learning_and_Neural_Networks_III_Epiphany_2024/tree/main/Lecture_handouts/code/04.Gradient_descent/example_1.R`

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

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

The GD Algorithm 4 with learning rate η is

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

(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.

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

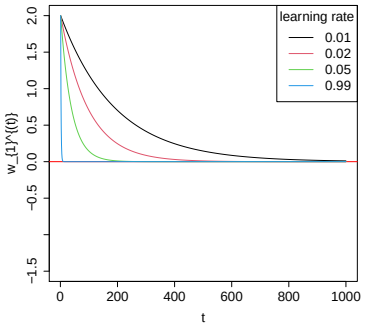
This is because $\hat{R}_{\mathcal{D}}(w)$ is differentiable in $\mathbb{R}^2$ so $\partial\hat{R}_{\mathcal{D}}(w) = \left\{\nabla\hat{R}_{\mathcal{D}}(w)\right\}$ and because

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

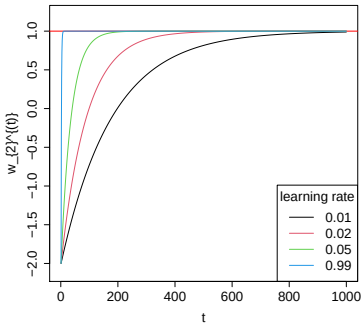
Consider data size $n = 100$, and parameters $\rho = 0.2$, $\sigma_y^2 = 1$ and $\sigma_x^2 = 1$. Then the real value (5.2) that I need to learn equals to $w^* = (0, 1)^\top$. Consider a GD seed $w_0 = (2, -2)$, and total number of iterations $T = 1000$.

Figures 5.1a, 5.1b, and 5.1c present trace plots of the chain $\left\{\left(w^{(t)}\right)\right\}$ and error $\hat{R}_{\mathcal{D}}\left(w^{(t)}\right) - \hat{R}_{\mathcal{D}}(w^*)$ 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.

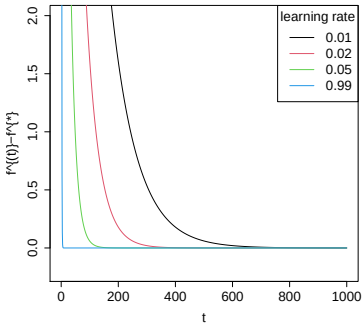
Figures 5.1d, 5.1e, and 5.1f present trace plots of the chain $\left\{\left(w^{(t)}\right)\right\}$, and of the error $\hat{R}_{\mathcal{D}}\left(w^{(t)}\right) - \hat{R}_{\mathcal{D}}(w^*)$ produced by running GD for $T = 1000$ total iterations and for learning rate $\eta = 1.0$ (previously considered) 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 (2.2) that the produced chain overshoots the global minimum; see the cartoon in Figures 5.1g and 5.1h.



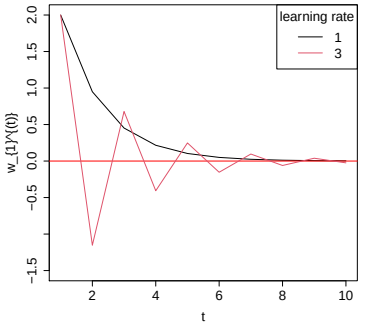
(A) $\left\{\left(w_1^{(t)}\right)\right\}$



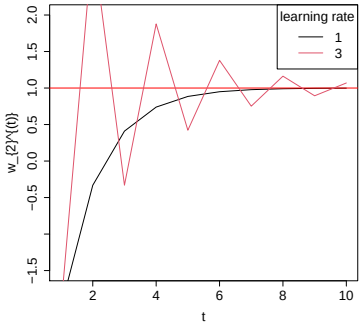
(B) $\left\{\left(w_2^{(t)}\right)\right\}$



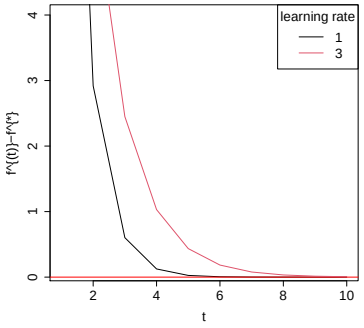
(C) $\hat{R}_{\mathcal{D}}\left(w^{(t)}\right) - \hat{R}_{\mathcal{D}}\left(w^*\right)$



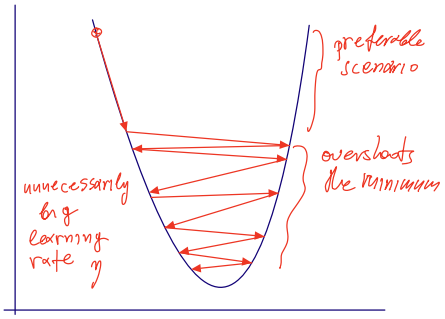
(D) $\left\{\left(w_1^{(t)}\right)\right\}$



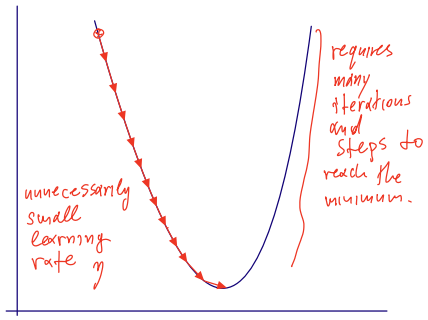
(E) $\left\{\left(w_2^{(t)}\right)\right\}$



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



(G) Unnecessarily large learning rate



(H) Unnecessarily small learning rate

FIGURE 5.1

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2024

Handout 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.

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

(4) 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).

Code of the Examples can be found in `https://github.com/georgios-stats/Machine_Learning_and_Neural_Networks_III_Epiphany_2024/tree/main/Lecture_handouts/code/05.Stochastic_gradient_descent`

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 $w^* \in \mathcal{H}$, where $\mathcal{H}$ is a class of hypotheses, of the risk function (RF) $R(w) = \mathrm{E}_{z\sim g}(\ell(w,z))$ given an unknown data generating model $g(\cdot)$ and using a known tractable loss $\ell(\cdot,\cdot)$; that is

$$w^* = \arg\min_{\forall w\in\mathcal{H}} (R_g(w)) = \arg\min_{\forall w\in\mathcal{H}} (\mathrm{E}_{z\sim g}(\ell(w,z))) \tag{1.1}$$

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) involves an integral which may be computationally intractable. Instead it aims to minimize the ERF $\hat{R}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(w, z_i)$ which ideally is used as a proxy when data size n is big (big-data).

Note 3. The implementation of GD may be computationally impractical even in problems where we need to minimize an 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 $\nabla\hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla\ell(w, z_i)$ using 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

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

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

Algorithm 6. *Stochastic Gradient Descent (SGD) with learning rate $\eta_t > 0$ for the solution of the minimization 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.*

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

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)

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

(2) or the best value discovered

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

(3) or the last value discovered

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

Note 10. SGD output converges to a local minimum, $w_{\text{SGD}}^{(T)} \to w_*$ (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 8 in Handout 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

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

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

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

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

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

I will try to use Lemma 22 from Handout 4, hence I need to show

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

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

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

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 22 in Handout 4 "Gradient descent" implies

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

□

Note 16. The upper bound in (3.1) depends on the variation of v_t as

$$\mathrm{E}\left\|v_{t}\right\|^{2}=\sum_{j=1}^{d}\mathrm{Var}\left(v_{t,j}\right)+\sum_{j=1}^{d}\left(\mathrm{E}\left(v_{t,j}\right)\right)^{2} \tag{3.5}$$

where d is the dimension of $v_t=\left(v_{t,1},...,v_{t,d}\right)^{\top}$. The second term on the right hand side of (3.5) is constant as $v_{t,j}$ is the unbiased estimator of the sub-gradient by construction.

Proposition 17. *(Cont. Proposition 15) Let $f(\cdot)$ be a convex function, and let $\mathcal{H}=\{w\in\mathbb{R}:\|w\|\leq B\}$. Let $E\|v_t\|^2\leq\rho^2$. Assume we run SGD algorithm of $f(\cdot)$ with learning rate $\eta_t=\sqrt{\frac{B^2}{\rho^2T}}$ 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 T iterations*

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

Proof. It follows from Proposition 15. Condition $\mathrm{E}\left\|v_t\right\|^2 \leq \rho^2$ can be achieved, for instance by assuming Lipschitz (See Lemma 26 from "Handout 4: Gradient descent"). □

4. Stochastic gradient descent with projection

Note 18. Consider the scenario in Problem 1 where the learning problem requires to discover w^* in the restricted/bounded set $\mathcal{H}$. Assume the function to be minimized is convex in the restricted hypothesis set $\mathcal{H}$, e.g. $\mathcal{H} = \{w : \|w\| \leq B\}$, but non-convex in $\mathbb{R}^d$. Direct implementation of vanilla SGD (Algorithm 6) may produce a chain stepping out $\mathcal{H}$ and hence an output $w_{\mathrm{SGD}} \notin \mathcal{H}$. SGD can be modified to address this issue, as in Algorithm 19, by including a projection step guaranteeing $w \in \mathcal{H}$.

Algorithm 19. *Stochastic Gradient Descent with learning rate $\eta_t > 0$ and with projection in $\mathcal{H}$ for Problem 5*

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

(1) *compute*

$$w^{\left(t+\frac{1}{2}\right)} = w^{(t)} - \eta_t v_t, \tag{4.1}$$

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*

$$w^{(t+1)} = \arg\min_{w \in \mathcal{H}} \left(\left\| w - w^{\left(t+\frac{1}{2}\right)} \right\| \right) \tag{4.2}$$

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

Note 20. The analysis in Section 3 holds for the SGD with projection after a slight modification in the math proofs; see Exercise 13 from the Exercise sheet.

5. Implementation of SGD in the learning Problem 1

Note 21. Proposition 22 allows a convenient implementation of SGD to the learning problem (Problem 1).

Proposition 22. *For a randomly drawn example $z \sim g(\cdot)$, the sub-gradient v of $\ell(w, z)$ at point w is an unbiased estimator of the sub-gradient of the risk $R_g(w)$ at point w.*

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

$$\ell(u, z) - \ell(w, z) \geq \langle u - w, v \rangle \tag{5.1}$$

It is

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

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

Example 23. Consider that loss ℓ is differentiable wrt w. If $v = \nabla_w \ell\,(w, z)$, then

Note 24. 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; $z_i \overset{\text{ind}}{\sim} g$. Batch SGD (Algorithm 25) is an implementation of the SGD (Algorithm 6) in the learning Problem 1.

Algorithm 25. *Batch Stochastic Gradient Descent with learning rate $\eta_t > 0$, and batch size m, for Problem 1.*

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

(1) *get a random sub-sample $\left\{\tilde{z}_j^{(t)}; j = 1, ..., m\right\}$ of size m with or without replacement from the complete data-set $\mathcal{S}_n$.*

(2) *compute*

$$w^{(t+1)} = w^{(t)} - \eta_t v_t, \tag{5.2}$$

where $v_t = \frac{1}{m}\sum_{j=1}^{m} \tilde{v}_{t,j}$ and $\tilde{v}_{t,j} \in \partial_w \ell\left(w^{(t)}, \tilde{z}_j^{(t)}\right)$.

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

Note 26. Step 1 can be described equivalently as

(1) Randomly generate a set $\mathcal{J}^{(t)} \subseteq \{1, ..., n\}^m$ of m indices from 1 to n with or without replacement, and set a $\tilde{\mathcal{S}}_m = \left\{z_i; i \in \mathcal{J}^{(t)}\right\}$.

Hence $v_t = \frac{1}{m}\sum_{j \in \mathcal{J}^{(t)}} v_{t,j}$ where $\tilde{v}_{t,j} \in \partial_w \ell\left(w^{(t)}, z_j\right)$.

Note 27. 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 25 can become

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

Definition 28. Online Stochastic gradient descent is the special case of Algorithm 25 using sub-samples of size one ($m = 1$), namely, only one example is randomly chosen in Step 1 of Algorithm 25.

Note 29. 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 30. 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 31. In practice, for a given fixed computational time, it is possible for a batch 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 vs online SGD 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 32. *(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 26 from "Handout 4: Gradient descent"); i.e. if ℓ is ρ-Lipschitz, then $\|v_t\| \leq \rho$ where $v_t \in \partial_w \ell\left(w^{(t)}, \tilde{z}_j^{(t)}\right)$. □

Example 33. We continue Example 33 in Section 4 of Handout 4. Recall, we considered a hypothesis space $\mathcal{H}$ of linear functions $h : \mathbb{R}^2 \to \mathbb{R}$ with $h\left(w\right) = w_1 + w_2 x$, $w = \left(w_1, w_2\right)^\top$, and $\ell\left(w, z = \left(x, y\right)^\top\right) = \left(y_i - w_1 - w_2 x\right)^2$. Here we consider a big dataset $\mathcal{S}_n = \{z_1, ..., z_n\}$ with $n = 10^6$ examples.

The batch SGD algorithm (Algorithm 25) with learning rate η_t and batch size $m = 10$ is

- For $t = 1, 2, 3, ...$ iterate:
  (1) Randomly generate a set $\mathcal{J}^{(t)}$ by drawing $m = 10$ numbers from $\left\{1, ..., n = 10^6\right\}$, and set $\tilde{\mathcal{S}}_m = \left\{z_i; i \in \mathcal{J}^{(t)}\right\}$
  (2) compute
  $$w^{(t+1)} = w^{(t)} - \eta_t v_t,$$
  where
  $$v_t = \begin{pmatrix} 2w_1^{(t)} + 2w_2^{(t)} \frac{1}{m} \sum_{j \in \mathcal{J}^{(t)}} x_j - 2\frac{1}{m} \sum_{j \in \mathcal{J}^{(t)}} y_j \\ 2w_1^{(t)} \bar{x} + 2w_2^{(t)} \frac{1}{m} \sum_{j \in \mathcal{J}^{(t)}} x_j^2 - 2 \sum_{j \in \mathcal{J}^{(t)}} y_j x_j \end{pmatrix}$$
  ,
  (3) if $t \geq T = 1000$ STOP

because

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

In Figures A.1a & A.1d, 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 $\left\{w^{(t)}\right\}$.

Example 34. Consider a (rather naive) loss function $\ell(w, z=(x,y)) = -\cos(0.5(y - w_1 - w_2 x))$, a hypothesis class $\mathcal{H} = \left\{w \in \mathbb{R}^2 : \|w\| \leq 1.5\right\}$, and assume that inputs x in dataset $\mathcal{D}$ 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 (batch size $m = 1$) 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

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

$$w^{(t+1/2)} = \underset{\|w\| \leq 1.5}{\arg\min}\left(\left\|w - w^{(t+1/2)}\right\|\right)$$

  (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.

6. Stochastic Variance Reduced Gradient (SVRG) Ref [3]

Remark 35. Recall the upper bound of the error (3.1) in SGD depends on the variance of the stochastic gradient, as shown in (3.5) of Remark 16. Hence the algorithm may be improved by reducing the variance of each element of v_t.

Remark 36. Control variate is a general way to perform variance reduction. Let random variables $v \in \mathbb{R}$, and $y \in \mathbb{R}$. Let $z = v + c(y - \mathrm{E}(y))$ for some constant $c \in \mathbb{R}$. It is $\mathrm{E}_c(z) = \mathrm{E}(v)$ and

$$\mathrm{Var}_c(z) = \mathrm{Var}(v) + c^2 \mathrm{Var}(y) + 2c\mathrm{Cov}(x, y)$$

which is minimized for

$$c^* = -\frac{\mathrm{Cov}(v, y)}{\mathrm{Var}(y)}$$

hence

$$\mathrm{Var}_{c^*}(z) = \mathrm{Var}(v) - \frac{(\mathrm{Cov}(v, y))^2}{\mathrm{Var}(y)}$$

Note 37. For simplicity, SVRG is presented in the case where $c = 1$ and the loss function $\ell(w, z)$ is differentiable wrt w at every z hence its gradient exists. However it is applicable to the more general cases introduced.

Remark 38. Every κ iterations, SVRG keeps a snapshot $\tilde{w}$, and computes the gradient using all data i.e. $\frac{1}{n}\sum_{i=1}^{n}\ell\left(\tilde{w},z_i\right)$. At each iteration t, the update is

$$w^{(t+1)}=w^{(t)}-\eta_t\underbrace{\left[\underbrace{\nabla\ell\left(w^{(t)},\tilde{z}^{(t)}\right)}_{=v}-\underbrace{1}_{=c}\underbrace{\left(\nabla\ell\left(\tilde{w},\tilde{z}^{(t)}\right)-\frac{1}{n}\sum_{i=1}^{n}\nabla\ell\left(\tilde{w},z_i\right)\right)}_{y}\right]}_{z}$$

given that a random $\tilde{z}^{(t)}$ has been collected from the sample. The symbols below the brackets are given with reference to Remark 36.

Algorithm 39. *Stochastic Variance Reduced Gradient with learning rate $\eta_t>0$ for Problem 1.*

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

(1) *randomly get an example $\tilde{z}^{(t)}$ from S_n.*

(2) *compute*

$$w^{(t+1)}=w^{(t)}-\eta_t\left[\nabla\ell\left(w^{(t)},\tilde{z}^{(t)}\right)-\nabla\ell\left(\tilde{w},\tilde{z}^{(t)}\right)+\frac{1}{n}\sum_{i=1}^{n}\nabla\ell\left(\tilde{w},z_i\right)\right] \tag{6.1}$$

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

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

Remark 40. Iterations of SVRG are computationally faster than those of full GD, but SVRG can still match the theoretical convergence rate of GD.

Remark 41. How often we get snapshots, aka κ, in Algorithm 39 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 and hence the bigger the performance improvement; however the iterations will be slower.

Example 42. 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 $\mathcal{J}^{(t)}=\{j^*\}$ by drawing one number j^* from $\left\{1,...,n=10^6\right\}$, and set $\tilde{\mathcal{S}}_1=\{z_{j^*}\}$
  (2) compute

$$w^{(t+1)}=\begin{bmatrix}w_1^{(t)}\\w_2^{(t)}\end{bmatrix}-\eta_t\left(\begin{bmatrix}2\left(w_1^{(t)}-\tilde{w}_1^{(t)}\right)+2\left(w_2^{(t)}-\tilde{w}_2^{(t)}\right)x_{j^*}\\2\left(w_2^{(t)}-\tilde{w}_2^{(t)}\right)x_{j^*}+2\left(w_2^{(t)}-\tilde{w}_2^{(t)}\right)\left(x_{j^*}\right)^2\end{bmatrix}+\nabla\hat{R}_{\mathcal{D}}\left(\tilde{w}\right)\right) \tag{6.2}$$

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

(b) compute

$$\frac{1}{n}\sum_{i=1}^{n}\ell\left(\tilde{w},z_i\right)=\begin{pmatrix}2\tilde{w}_1^{(t)}+2\tilde{w}_2^{(t)}\bar{x}-2\bar{y}\\ 2\tilde{w}_1^{(t)}\bar{x}+2\tilde{w}_2^{(t)}\overline{x^2}-2y^\top x\end{pmatrix}=\nabla\hat{R}_{\mathcal{D}}\left(\tilde{w}\right) \tag{6.3}$$

(4) if a termination criterion is satisfied STOP

Because (6.3) is actually the gradient of the Risk function at $\tilde{w}$ and

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

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.

7. Preconditioned SGD; The AdaGrad algorithm

Ref [4]

Remark 43. All SGD (introduced earlier) are first order methods in the sense they consider only the gradient. Their advantage is each iteration is fast. The disadvantage is that they ignore the curvature of the space and hence can be slower to converge in cases the curvature changes eg. among dimensions of w.

Remark 44. To address this, SGD (Algorithm 6) can be modified in the update step as in Algorithm 45 by using a preconditioner P_t that accounts for the curvature (or geometry in general of f).

Algorithm 45. *Preconditioned Stochastic Gradient Descent with learning rate $\eta_t > 0$, and preconditioner P_t for the solution of the minimization problem (2.1)*

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

(1) *compute*

$$w^{(t+1)}=w^{(t)}-\eta_t P_t v_t, \tag{7.1}$$

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.*

If $t\geq T$ then STOP

Remark 46. A natural choice of P_t can be $P_t:=\left[H_t+\epsilon I_d\right]^{-1}$, where H_t is the Hessian matrix $[H_t]_{i,j}=\frac{\partial^2}{\partial w_i\partial w_j}f\left(w\right)\Big|_{w=w^{(t)}}$ (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.

Remark 47. If the preconditioner P_t is set to be the inverse of the full Hessian, it may be too expensive to perform matrix operations in (7.1) with the full Hessian, and such operations can be too unstable/inaccurate due to the random error induced by the stochasticity of the gradient.

7.1. Adaptive Stochastic Gradient Decent (AdaGrad).

Remark 48. AdaGrad aims to dynamically incorporate knowledge of the geometry of function $f(\cdot)$ (to be minimized) in earlier iterations to perform more informative gradient-based learning.

Remark 49. AdaGrad 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).

Remark 50. Hence, this strategy often improves convergence performance over standard stochastic gradient descent in settings where the data are sparse and sparse features w's are more informative.

Algorithm 51. *Adaptive Stochastic Gradient Decent (AdaGrad) can be presented in terms of preconditioned SGD (Algorithm 45) with preconditioner* $P_t = [diag(G_t) + \epsilon I_d]^{-1/2}$ *in (7.1)*

$$w^{(t+1)} = w^{(t)} - \eta_t \left[diag(G_t) + \epsilon I_d\right]^{-1/2} v_t, \tag{7.2}$$

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,* $G_t = \sum_{\tau=1}^{t} v_\tau^\top v_\tau$ *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.*

Remark 52. AdaGrad algorithm 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

$$[G_t]_{j,j} = \sum_{\tau=1}^{t} (v_{\tau,j})^2 \tag{7.3}$$

where j denotes the j-th dimension of w. Hence (7.2) and (7.2)imply that the j-th dimension of w is updated as

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

Remark 53. The accumulation of positive terms in (7.3) makes the sum keep growing during training and causes the learning rate to shrink and becoming 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 (7.2) a (user specified rate) $\eta_t \geq 0$ and tuning it properly via pilot runs.

Example 54. The 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 a set $\mathcal{J}^{(t)} = \{j^*\}$ by drawing one number from $\{1, ..., n = 10^6\}$, and set $\tilde{\mathcal{S}}_1 = \{z_{j^*}\}$

(2) compute

$$v_t = \begin{pmatrix} 2w_1^{(t)} + 2w_2^{(t)} x_{j^*} - 2y_{j^*} \\ 2w_1^{(t)} \bar{x} + 2w_2^{(t)} x_{j^*}^2 - 2y_{j^*} x_{j^*} \end{pmatrix}$$

$$G_t = G_{t-1} + \begin{pmatrix} v_{t,1}^2 \\ v_{t,2}^2 \end{pmatrix}$$

$$w^{(t+1)} = \begin{pmatrix} w_1^{(t)} - \frac{\eta_t}{\sqrt{G_{t,1}+\epsilon}} v_{t,1} \\ w_2^{(t)} - \frac{\eta_t}{\sqrt{G_{t,2}+\epsilon}} v_{t,2} \end{pmatrix},$$

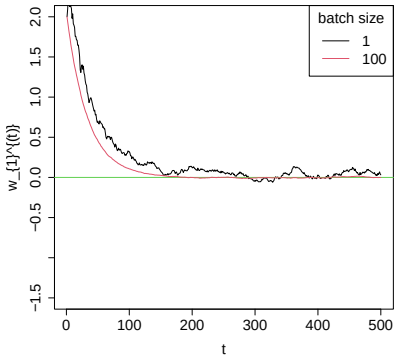
(3) if $t \geq T = 1000$ STOP

because

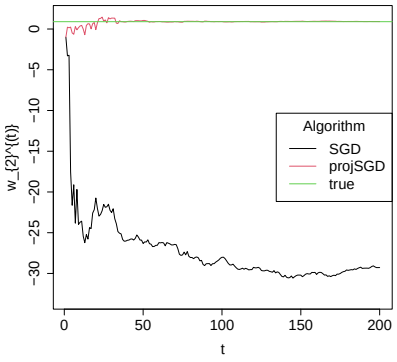
$$v_t = \nabla \ell \left(w^{(t)}, z_{j^*} \right) = \begin{pmatrix} \frac{\partial}{\partial w_1^{(t)}} \ell \left(w^{(t)}, z_{j^*} \right) \\ \frac{\partial}{\partial w_2^{(t)}} \ell \left(w^{(t)}, z_{j^*} \right) \end{pmatrix} = \begin{pmatrix} 2w_1^{(t)} + 2w_2^{(t)} x_{j^*} - 2y_{j^*} \\ 2w_1^{(t)}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

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.

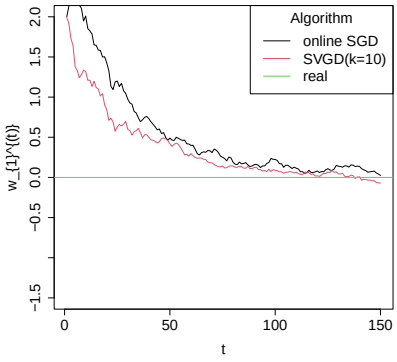
Appendix A. Plots



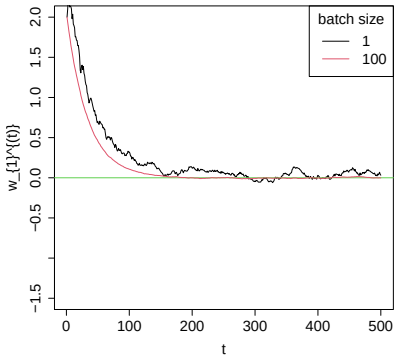
(A) batch SGD Example 33 w_1



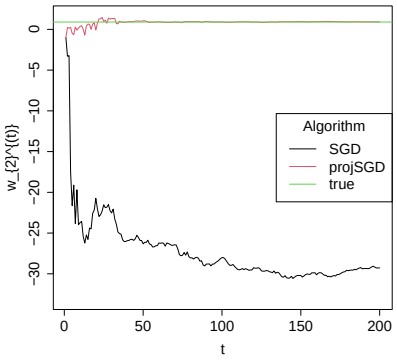
(B) online SGD with projection Example 34w_1



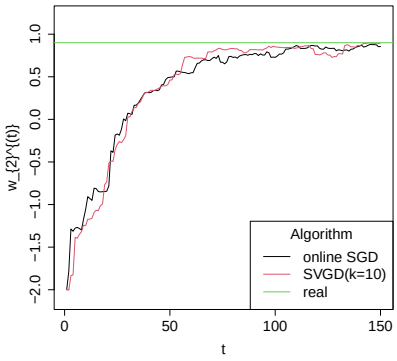
(C) SVRG Example 42 w_1



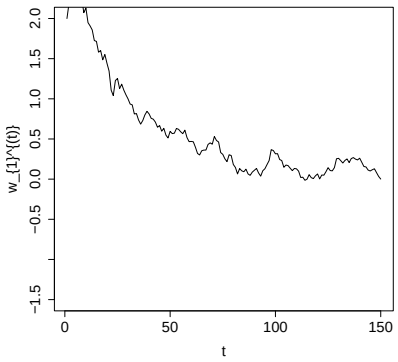
(D) batch SGD Example 33 w_2



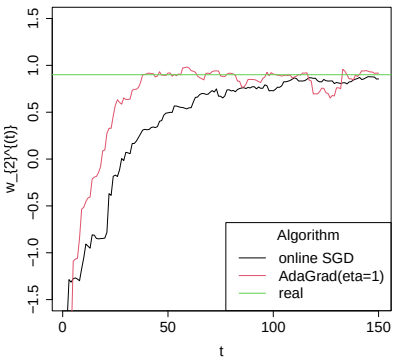
(E) online SGD with projection Example 34 w_2



(F) SVRG Example 42 w_2



(G) online AdaGrad Example 54w_1



(H) online AdaGrad Example 54w_2

FIGURE A.1. Simulations of the Examples

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2024

Handout 6: 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 $w \in \Theta \subseteq \mathbb{R}^d$ 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.

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

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

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

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

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

Algorithm 11. *Gradient descent (Algorithm 4 "Handout 4: Gradient descent ") with learning rate $\eta_t \geq 0$ can be used to address Problem 9 by using the update rule as*

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. The implementation of other GD variants (eg (3.2) in Handout 2) is straightforward, based on Algorithm 11.

Note 13. 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 (Remarks 30 and 31 of Handout 3).

Algorithm 14. *Batch Stochastic Gradient Descent (Algorithm 25 in "Handout 5: Stochastic gradient descent") with learning rate $\eta_t \geq 0$ and batch size m can be used to address Problem 9 by using the update rule as*

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, and set a $\tilde{\mathcal{S}}_m = \left\{z_i; i \in \mathcal{J}^{(t)}\right\}$.*

(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 15. 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 11 from the Exercise sheet.

Note 16. The implementation of the SGD variants (Algorithms 25, 39, 45, 51 in Handout 3)) is straightforward based on Algorithm 14 and (2.4).

3. Fully Bayesian learning via SGLD

Problem 17. Fully Bayesian learning, computationally, is the problem of recovering the posterior distribution $f(w|z_{1:n})$ of w given $z_{1:n}$ that admits density (1.2). For a given Bayesian model (1.1), the Bayesian estimator of $h := h(w)$ id often computed as the posterior expectation of $h(w)$ given the data $\mathcal{S}_n$

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

Note 18. Monte Carlo integration aims at approximating (1.3) when intractable, by using Central Limit Theorem or Law of Large Numbers arguments as $\hat{h}_T \approx \mathrm{E}_f(h(w)|z_{1:n})$ 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 19. Stochastic gradient Langevin dynamics (SGLD) algorithm is able to generate a sample approximately distributed according to the posterior distribution (1.2). 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 (Remark 18).

Note 20. SGLD (see Algorithm 21), as output, generates a random chain $\left\{w^{(t)}\right\}$ that is approximately distributed according to a distribution that admits 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.

Ref [1]

Algorithm 21. *Stochastic Gradient Langevin Dynamics (SGLD) with learning rate $\eta_t > 0$, batch size m, and temperature $\tau > 0$ is*

---

(1) *Generate a random set $\mathcal{J}^{(t)} \subseteq \{1, ..., n\}^m$ of m indices from 1 to n with or without replacement, and set a $\tilde{\mathcal{S}}_m = \left\{z_i; i \in \mathcal{J}^{(t)}\right\}$.*

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

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

About intuitions (why it works) ...

Note 22. 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 $\left\{w^{(t+1)}\right\}$ converges to samples that could have been drawn from distribution with density (3.2).

Note 23. (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

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

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

Note 24. (Heuristically speaking) 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 $\nabla \log\left(L_n(w)\right)$ 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 25. One can argue that, the output of SGLD is also an "almost" minimizer of the empirical risk for large enough t. A draw from the Gibbs distribution (3.6) is approximately a minimizer of

[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 $0 \leq s \leq t$.

(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 26. 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 27. Regarding the learning rate (or gain) $\{\eta_t\}$ should satisfy the following conditions in order for SGLD Algorithm 21 to generate an output $\left\{w^{(t)}\right\}$ approximately distributed according to (3.2)

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

About the temperature alg. parameter...

Note 28. The popular learning rates $\{\eta_t\}$ in Note 10 in "Handout 4: Gradient descent " satisfy Condition 27 and hence can be used in SGD too. Once parametrized, η_t can be tuned based on pilot runs using a reasonably small number of data.

Note 29. 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 30. The first few iterations of SGLD (Algorithm 21) 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 31. The output of SGLD (Algorithm 21) $\left\{w^{(t)}\right\}$ 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 32. SGLD for $\tau = 1$ approximately simulates from the posterior (1.2).

About facilitating inference...

Note 33. 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 $\widehat{h_T(w)} \to \mathrm{E}_f\left(h(w)\,|z_{1:n}\right)$ based on LLN arguments.

Note 34. 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 35. 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.

Preconditioned SGLD...

Note 36. Certain dimensions may have a vastly larger curvature leading to much bigger gradients. In this case a symmetric preconditioning matrix $P_t > 0$ can transform all dimensions to the same scale; this is similar to the SGD case in "Handout 5: Stochastic gradient descent". Hence the update (3.4) becomes

$$w^{(t+1)} = w^{(t)} + \eta_t P_t \left(\frac{n}{m} \sum_{i \in J^{(t)}} \nabla \log f\left(z_i | w\right) + \nabla \log f\left(w\right) \right) + \sqrt{2\eta_t \tau} P_t^{\frac{1}{2}} \epsilon_t \tag{3.9}$$

where $P_t^{\frac{1}{2}}$ is such that $P_t^{\frac{1}{2}} \left(P_t^{\frac{1}{2}}\right)^{\top} = P_t$.

About the exploding gradient phenomenon...

Note 37. '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 38. 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 39. 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 40. Gradient clipping involves forcing the gradient values (element-wise) to a specific minimum or maximum value if the gradient exceeded an expected range.

Note 41. Unreasonably frequent scaling or clipping may introduce significant bias and hence it should not be applied carelessly.

4. Examples [2]

We continue the Example 33 in 4: Gradient descent.

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

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

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

$$\begin{aligned} \mathrm{N}\left(x|\mu,\Sigma\right) &= \left(\frac{1}{2\pi}\right)^d \frac{1}{|\Sigma|} \exp\left(-\frac{1}{2}\left(x-\mu\right)^\top \Sigma^{-1}\left(x-\mu\right)\right) \\ \mathrm{IG}\left(x|a,b\right) &= \frac{b^a}{\Gamma\left(a\right)} x^{-a-1} \exp\left(-\frac{b}{x}\right) 1\left(x \geq 0\right) \end{aligned}$$

Performing computationally convenient transformations. Because SGD (Algorithm 14) and SGLD (Algorithm 21) 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

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

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

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

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 $\gamma = \log\left(\sigma^2\right)$.

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

[2] Code is available from `https://github.com/georgios-stats/Machine_Learning_and_Neural_Networks_III_Epiphany_2024/tree/main/Lecture_handouts/code/06.Stochastic_gradient_Langevine_dynamics`

Hence for the log sampling PDF we have

$$\nabla_w \log\left(f\left(z_i|w\right)\right) = \left(\frac{\mathrm{d}}{\mathrm{d}\beta}\log\left(f\left(z_i|w\right)\right) \; ; \; \frac{\mathrm{d}}{\mathrm{d}\gamma}\log\left(f\left(z_i|w\right)\right)\right)$$

$$\frac{\mathrm{d}}{\mathrm{d}\beta}\log\left(f\left(z_i|w\right)\right) = \left(y_i - x_i^\top \beta\right) x_i \exp\left(-\gamma\right)$$

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

...so

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

Hence for the log a priori PDF we have

$$\nabla_w \log\left(f\left(w\right)\right) = \left(\frac{\mathrm{d}}{\mathrm{d}\beta}\log\left(f\left(w\right)\right) \; ; \; \frac{\mathrm{d}}{\mathrm{d}\gamma}\log\left(f\left(w\right)\right)\right)$$

$$\frac{\mathrm{d}}{\mathrm{d}\beta}\log\left(f\left(w\right)\right) = -\exp\left(-\gamma\right) V^{-1}\left(\beta - \mu\right)$$

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

...so

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

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

To implement SGD (Algorithm 14) and SGLD (Algorithm 21) , 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 $\gamma = \log\left(\sigma^2\right)$. After running SGD and SGLD with the computed gradients with respect to $w = (\beta, \gamma) \in \mathbb{R}^d \times \mathbb{R}$ with $\gamma = \log\left(\sigma^2\right)$, 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 $\left(\beta, \sigma^2\right)$.

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 $\tau = 1$.

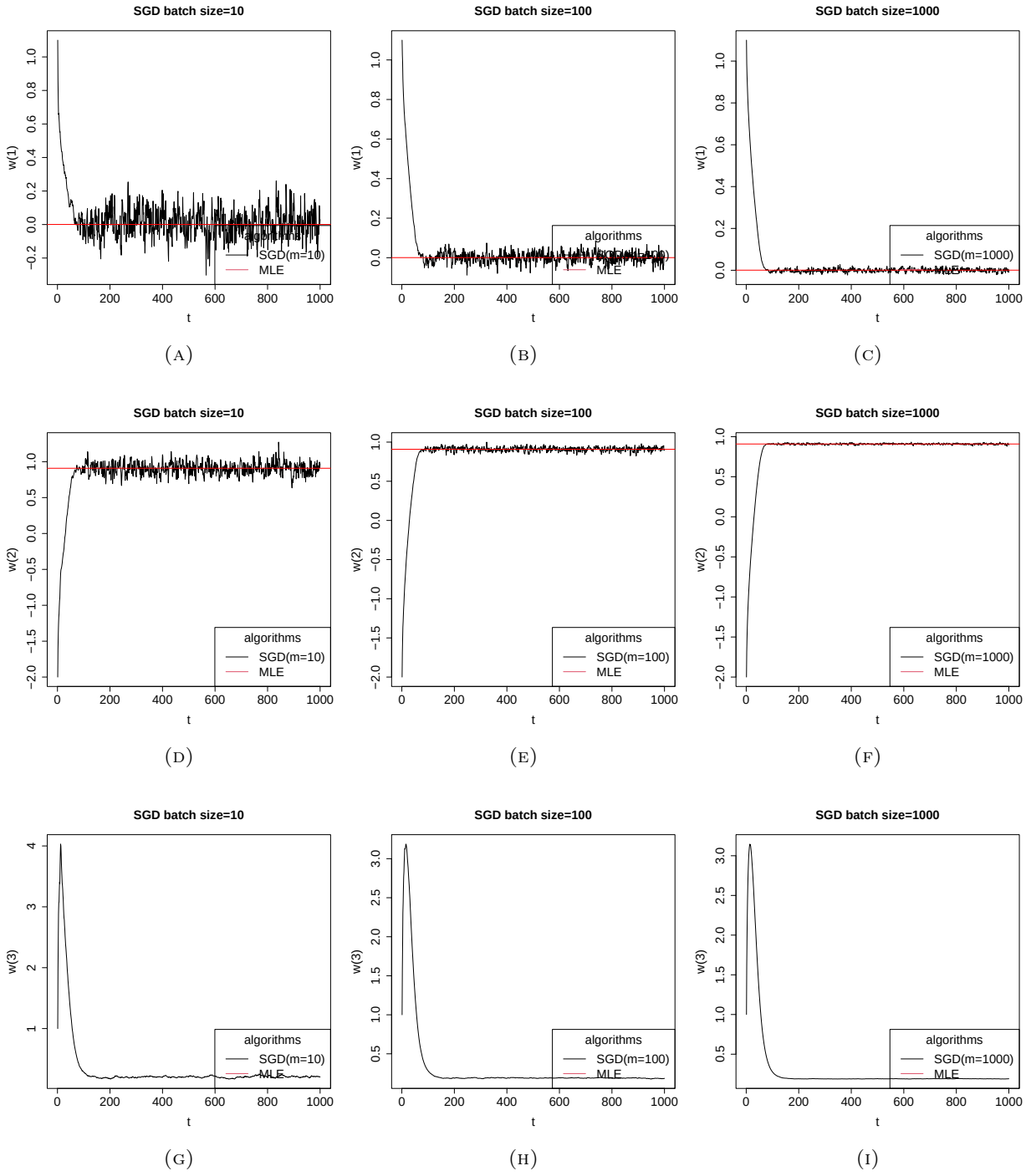


FIGURE 4.1. Bayesian learning via SGD (SGD vs (exact)MLE) Study on batch size m.

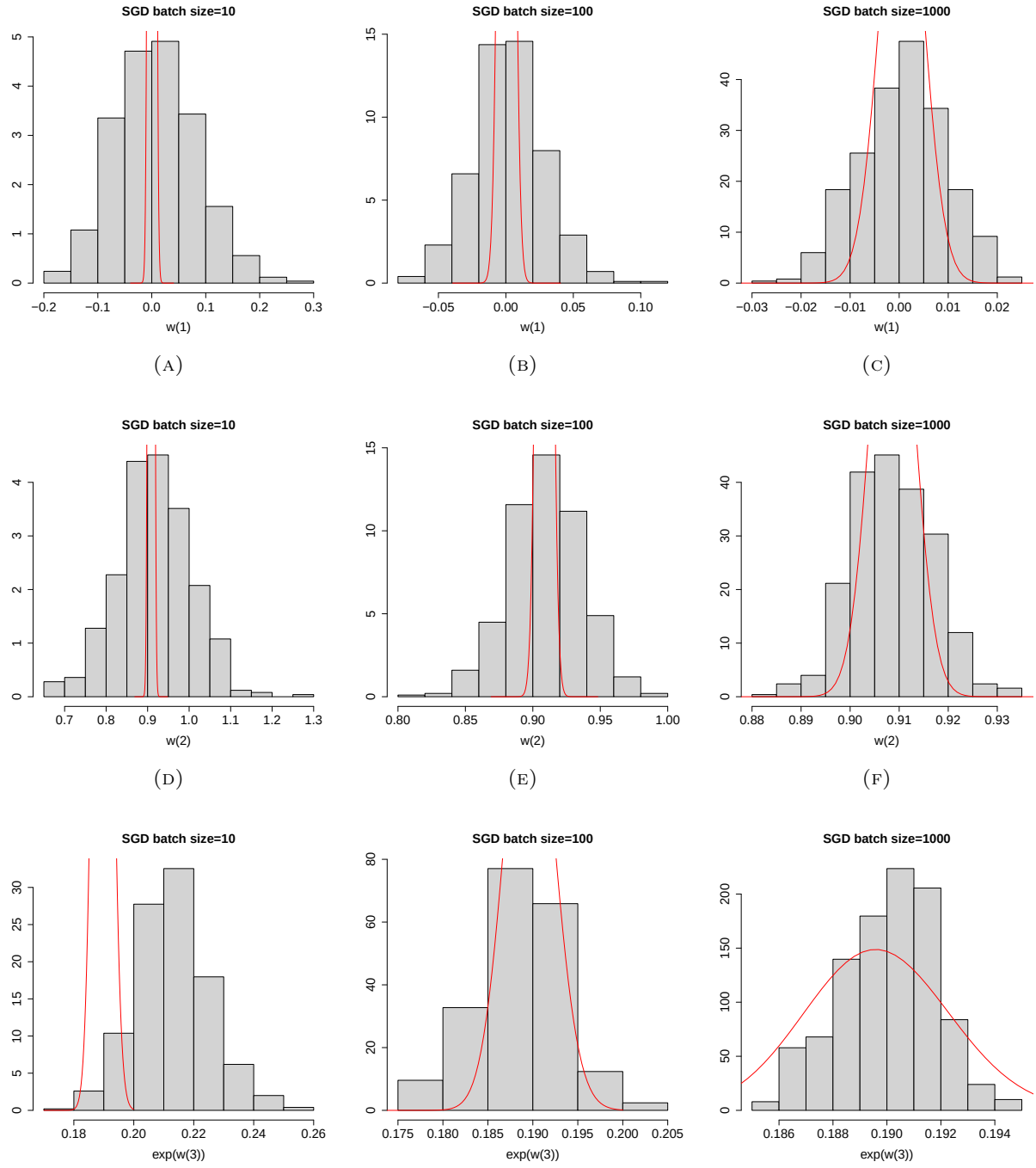


FIGURE 4.2. Bayesian learning: SGLD vs exact posterior (in red). Temperature $\tau = 1$. Study on batch size m

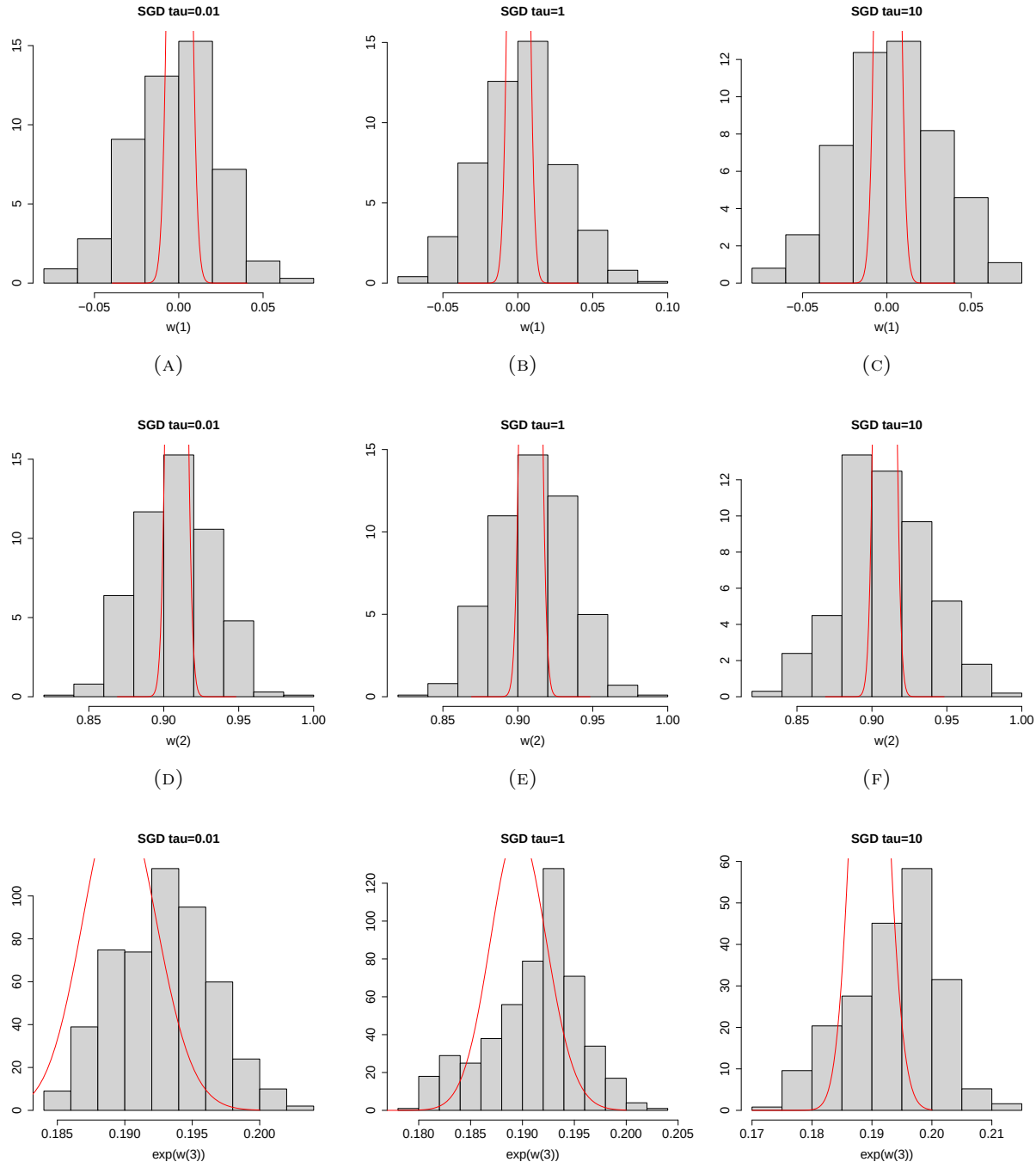


FIGURE 4.3. Bayesian learning: SGLD vs exact posterior (in red). Batch size = 100. Study on τ

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2024

Handout 7: 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

(5) Strang, G. (2019). Linear algebra and learning from data. Wellesley-Cambridge Press.

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).

Note 12. 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.

Note 13. 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.

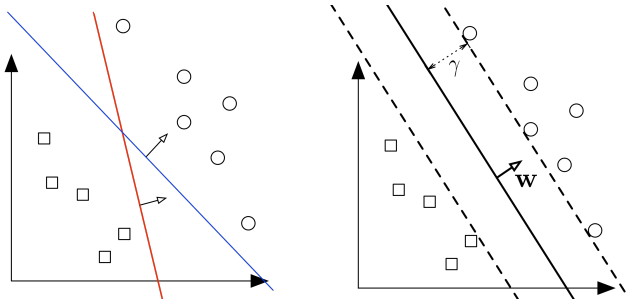


FIGURE 1.1

Note 14. Support Vector Machines (SVM) aims at learning the maximum margin separating hyperplane Figure (1.1; Right).

2. Hard Support Vector Machine

Assumption 15. *Assume the training sample* $S = \{(x_i, y_i)\}_{i=1}^m$ *is linearly separable.*

Definition 16. 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 15.

Problem 17. (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:

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

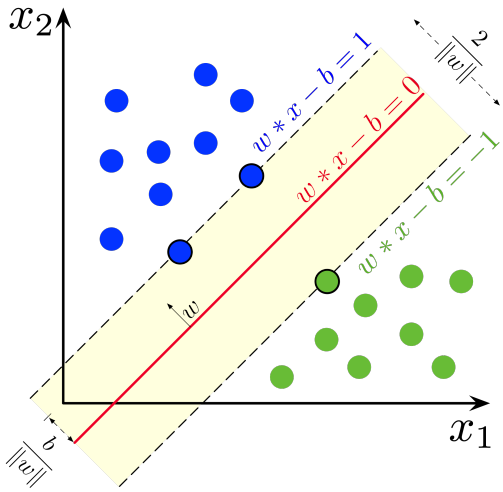
$$\hat{w} = \frac{\tilde{w}}{\|\tilde{w}\|}, \text{ and } \hat{b} = \frac{\tilde{b}}{\|\tilde{w}\|}$$

Note 18. Following we show why Problem 17 produces a Hard-SVM hyperplane stated in Note 16.

Proposition 19. *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 20. On the right, see the geometry of Problem 17.



Note 21. 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 22. Hard-SVM in Problem 17 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 17)

(1) Based on Note 16, and Proposition 19, the closest point in the training set to the separating hyperplane has distance

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

Without loss of generality, we make 2.3 identifiable by just pick those with $\|w\| = 1$. Hence, by Definition 16, the Hard-SVM hypothesis should be such as

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

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

(2) If there is a solution in (2.4) (namely, linearly separable dataset), then (2.4) is equivalent to (proof is omitted)

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

(3) Next we show that (2.6) is equivalent to the solution of Problem 17; i.e. $(w^*, b^*) = \left(\hat{w}, \hat{b} \right)$.
Let $\gamma^* := \min_i \left(|\langle w^*, x_i \rangle + b^*| \right)$. 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 condition (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 23. 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 24. Hard-SVM assumes the strong Assumption 15 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 25. 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 15. Soft-SVM rule is given as the solution to the quadratic optimization problem 26.

Problem 26. (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

Solve

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

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

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

Note 27. To relax the linearly separable training set Assumption 15, 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 28. 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 29. 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 30. Proposition 31 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 31. *The solution of Problem 26 is equivalent to the Ridge regularized loss minimization problem*

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

corresponding to a learning problem under the hinge loss function

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

and hence the hinge Empirical Risk Function

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

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

Proof. In Algorithm 26, we consider (3.1) as

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

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

Example 32. Consider the Soft-SVM as a Ridge regularized loss minimization 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 $\ell^{\text{hinge}}\left((w,b), z\right) = \max\left(0, 1 - y\left(\langle w, x\rangle + b\right)\right)$ is $\|x\|$-Lipschitz, it is

  $$\mathrm{E}_{S\sim g}\left(R_g^{\text{hinge}}\left(\mathfrak{A}(S)\right)\right) \leq R_g^{\text{hinge}}(u) + \lambda\|u\|^2 + \frac{2\rho^2}{\lambda m}$$

  [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

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

- If we choose $\lambda = \sqrt{\frac{2\rho^2}{B^2 m}}$ then

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

- Assume one was interested in minimizing a $0-1$ Risk $R_g^{0-1}(\cdot)$ under the $0-1$ loss $\ell^{0-1}\left((w,b), z\right) := 1_{(y\langle w,x\rangle \leq 0)}$. But $\ell^{0-1}(\cdot, z)$ is non-convex for all z. It is $\ell^{0-1}\left((w,b), z\right) \leq \ell^{\text{hinge}}\left((w,b), z\right)$ 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

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

Example 33. Given Proposition 31, Soft-SVM in Problem 26 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 $v_{b,t} = \begin{cases} -y_i^{(t)} & \text{if } y_i^{(t)} \left(\langle w^{(t)}, x_i^{(t)} \rangle + b^{(t)} \right) < 1 \\ 0 & \text{otherwise} \end{cases}$.

4. Duality and sparsity

4.1. **Lagrangian duality .** Ref [4]

Notation 34. 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 35. (Primal problem) Consider we have the following minimization problem that we will call Primal problem

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

Definition 36. (Lagrangian function) To the Primal problem 35, we associate the Lagrangian function $L : \mathbb{R}^q \times \mathbb{R}^m \times \mathbb{R}^n$ with

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

Note 37. Note that the Primal problem is equivalent to

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

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

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

Definition 38. (Lagrangian dual problem) The associated Lagrangian dual problem is

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

Definition 39. (Dual function) To the Dual problem, we associate a function $\tilde{L} : \mathbb{R}^m \times \mathbb{R}^n$ with

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

called dual function.

Proposition 40. *In general it is $p^* \geq d^*$; i.e.*

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

Definition 41. We call the general case $p^* \geq d^*$ weak duality.

Definition 42. When $p^* = d^*$ we say we have a strong duality.

Proposition 43. *(Strong duality via Slater condition) If the primal problem (4.1) is convex, and satisfies the weak Slater's condition, i.e.*

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

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

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

Proposition 44. *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*

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

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

Note 45. The Hard-SVM Problem 17 (2.2) and (2.2), can be rewritten in the form

$$\min_w \left(\frac{1}{2} \left\| w \right\|_2^2 + g\left(w\right) \right) \tag{4.4}$$

with

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

Hence we get the weak duality

$$\begin{aligned}
& \min_{w,b} \max_{\alpha \geq 0} \left(\frac{1}{2} \left\| w \right\|_2^2 + \sum_{i=1}^{m} \alpha_i \left(1 - y_i \left(\left\langle w, x_i \right\rangle + b \right)\right) \right) \\
& \geq \max_{\alpha \geq 0} \min_{w,b} \left(\underbrace{\frac{1}{2} \left\| w \right\|_2^2 + \sum_{i=1}^{m} \alpha_i \left(1 - y_i \left(\left\langle w, x_i \right\rangle + b \right)\right)}_{= L(w,b;\alpha)} \right)
\end{aligned} \tag{4.5}$$

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

$$0 = \nabla_w L(w,\alpha,b)\big|_{(w^*,b^*)} \implies w^* = \sum_{i=1}^m \alpha_i y_i x_i \tag{4.6}$$

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

The dual function is

$$\begin{aligned}\tilde{L}(\alpha) = \min_{w,b}(L(w,\alpha,b)) &= \frac{1}{2}\left\|\sum_{i=1}^m \alpha_i y_i x_i\right\|_2^2 - \sum_{i=1}^m \alpha_i \left(y_i\left(\langle\sum_{j=1}^m \alpha_j y_j x_j, x_i\rangle + b\right) - 1\right)\\ &= \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m\sum_{j=1}^m \alpha_i\alpha_j y_i y_j \langle x_j x_i\rangle\end{aligned}$$

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 46. The Dual problem of the Hard-SVM (primal) problem 17 is

$$\alpha^* = \arg\max_{\alpha\in\mathbb{R}^m:\alpha\geq 0}\left(\sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m\sum_{j=1}^m \alpha_i\alpha_j y_i y_j \langle x_j, x_i\rangle\right) \tag{4.8}$$

$$\text{subject to } 0 = \sum_{i=1}^m \alpha_i y_i$$

Note 47. Once the optimal α^* is computed from (4.8), the optimal weights w^* can be computed (from (4.6)) as

$$w^* = \sum_{i=1}^m \alpha_i^* y_i x_i \tag{4.9}$$

Note 48. If $\alpha_i^* = 0$, the example (x_i, y_i) does not contribute to (4.9). If $\alpha_i^* \neq 0$, (x_i, y_i) contributes to (4.9). 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).

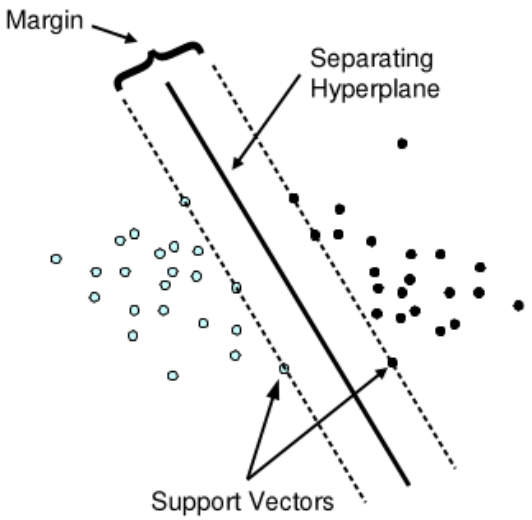


FIGURE 4.1

Note 49. Given optimal α^* (4.9) becomes

$$w^* = \sum_{i \in \mathcal{I}} \alpha_i^* y_i x_i \tag{4.10}$$

where $\mathcal{I} = \{i : y_i (\langle w^*, x_i \rangle + b^*) - 1 = 0\}$.

Note 50. The bias b parameter can be computed by KKT condition (4.3), it is

$$\alpha_i^* (y_i (\langle w^*, x_i \rangle + b^*) - 1) = 0$$

Consider $\mathcal{I} = \{i : y_i (\langle w^*, x_i \rangle + b^*) - 1 = 0\}$ then, for $i \in \mathcal{I}$ it is

$$y_i^2 (\langle w^*, x_i \rangle + b^*) - y_i = 0 \overset{y_i^2=1}{\Longrightarrow} \langle w^*, x_i \rangle + b^* - y_i = 0$$

and summing up all $i \in \mathcal{I}$, I get

$$\sum_{i \in \mathcal{I}} (\langle w^*, x_i \rangle + b^* - y_i) = 0 \implies b^* = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} (y_i - \langle w^*, x_i \rangle) \tag{4.11}$$

Note 51. Therefore the predictive halfspace hypothesis is given as

$$\begin{aligned} h_{w,b}(x) &= \operatorname{sign} (\langle w^*, x \rangle + b^*) \\ &= \operatorname{sign} \left(\sum_{i \in \mathcal{I}} \alpha_i^* y_i \langle x_i, x \rangle + b^* \right) \end{aligned} \tag{4.12}$$

with $\mathcal{I} = \{i : y_i (\langle w^*, x_i \rangle + b^*) - 1 = 0\}$.

Note 52. Compared to the Primal problem (Problem 17), the dual problem (Note 46) 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 53. Compared to the Primal problem, the dual problem (Note 46), can provide sparse solutions relying on a small number of support vectors (see Eq. 4.10, 4.11, and 4.12).

Note 54. (Snapshot of the next concept "The kernel trick") The importance of the existence of the dual problem is that eg (4.8) and (4.12) 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 (4.12), 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 4.8 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 55. In the Soft-SVM case, the solution is the same as in the Hard-SVM (4.9) and (4.12), the only difference is that in the quadratic programming problem (4.8) 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 **??** from the Exercise sheet).

Note 56. Sparsity and duality in Soft SVM can be seen in Exercise 15 in the Exercise sheet.

Note 57. Relevance Vector Machine (a version of Bayesian SVM) can be seen in Exercise 16 in the Exercise sheet.

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2024

Handout 8: Kernel methods

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:

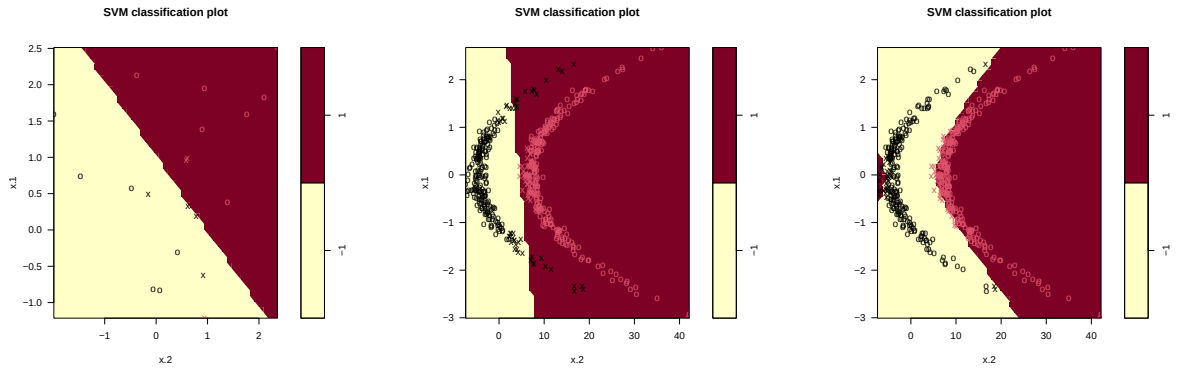
(1) Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.
- Ch. 16.2 Support Vector Machine

(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 $x = (x_1, x_2)^\top \in \mathbb{R}^2$. 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 Computer practical 4

Note 2. The accuracy of predictive rule could be improved if I could take into account the curvature 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 learning task that involves a hypothesis class $\mathcal{H} = \{x \mapsto \langle w, x\rangle : w \in \mathbb{R}^n\}$ where the predictive rule $h \in \mathcal{H}$ is defined over domain set $\mathcal{X}$ and is to be trained against data set $\mathcal{S} = \{z_i = (x_i, y_i)\}_{i=1}^m$ drawn from data generating process $G(\cdot)$. The basic paradigm for improving expressive power of $\mathcal{H}$ via embedding involves:

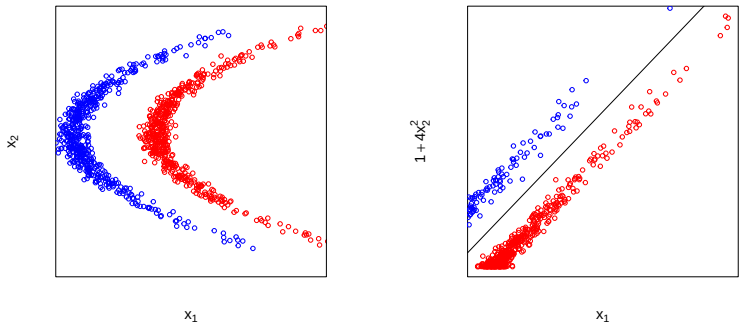
(1) Choose an embedding $\psi : \mathcal{X} \to \mathcal{F}$ with $\psi(x) := (\psi_1(x), ..., \psi_d(x))^{\top}$ for some feature space $\mathcal{F}$.
(2) Create the image sequence $\mathcal{S}^{\psi} = \left\{z_i^{\psi} = (\psi(x_i), y_i)\right\}_{i=1}^m$ from the original training set $\mathcal{S}$.
(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 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. 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. Eg, 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



(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

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).

Note 9. As feature mapping ψ any function that maps the original instances $\mathcal{X}$ into some Hilbert space $\mathcal{F}$ can be used.

Definition 10. A Hilbert space is a vector space, with an inner product, which is also complete.

Lemma 11. *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 $v \in \mathcal{X}$.*

Note 12. Feature space $\mathcal{F}$ is a Hilbert space preferably due to Lemma 11 that enables the Kernel trick via the representation Theorem 19. Eg, a Euclidean space such as $\mathbb{R}^d$ for some d. That includes infinite dimensional spaces.

Note 13. Using a $\psi(x) = (\psi_1(x), \psi_2(x) ...)^\top$ 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.

Example 14. ; Example Consider the Soft-SVM in "Handout 7: Support Vector Machines". Consider a given embedding $\psi : \mathcal{X} \to \mathcal{F}$ with $\psi(x) = (\psi_1(x), \psi_2(x) ...)^\top$. The learning rule becomes

$$h^\psi(x) = h(\psi(x)) = \text{sign}(\langle w, \psi(x)\rangle + b) \tag{2.1}$$

. In "Handout 7, Problem 26 becomes

Solve

$$(w^*, b^*, \xi^*) = \underset{(w,b,\xi)}{\arg\min} \left(\lambda \|w\|_2^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right)$$
$$\text{subject to: } y_i \left(\langle w, \psi(x_i) \rangle + b \right) \geq 1 - \xi_i, \ \forall i = 1, ..., m$$
$$\xi_i \geq 0, \ \forall i = 1, ..., m$$

3. The Kernel trick

Note 15. We discuss a duality in the learning problem 16 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 16. (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

$$\underset{w}{\text{minimize}} \left(\ f\left(\langle w, \psi(x_1) \rangle, ..., \langle w, \psi(x_m) \rangle \right) + R\left(\|w\| \right) \ \right), \tag{3.1}$$

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 17. (Cont. Example 14) Soft SVM problem has a solution equivalent to (see Proposition 31, Handout 7)

$$(w^*, b^*) = \underset{(w,b)}{\arg\min} \left(\frac{1}{m} \sum_{i=1}^{m} \max\left(0, 1 - y_i \left(\langle w, \psi(x_i) \rangle + b \right)\right) + \lambda \|w\|_2^2 \right)$$

then in terms of (3.1) we get

$$f(\alpha_1, ..., \alpha_m) = \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i \alpha_i\}, \text{ and } R(\beta) = \lambda \beta^2$$

Definition 18. 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 19. *(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 16.*

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 20. 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 19, the Learning problem 16, 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

$$\underset{\alpha}{\text{minimize}} \left(f\left(\sum_{i=1}^m \alpha_i K(x_i, x_1), ..., \sum_{i=1}^m \alpha_i K(x_i, x_m)\right) + R\left(\sqrt{\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j K(x_i, x_j)}\right)\right), \tag{3.2}$$

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 21. In Learning Problem 16, 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 22. (Cont. Example 17) 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 23. (Polynomial Kernels) Let $x \in \mathcal{X} \subseteq \mathbb{R}^n$. Assume 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 $\psi(x)$ is a vector of elements $\psi_J(x) = \prod_{i=1}^r x_{J_i}$ for $J \in \{1, ..., n\}^r$ and $r \leq k$. This learning problem can be equivalently be addressed with the k degree polynomial kernel

$$K(x, x') = \left(1 + \langle x, x'\rangle\right)^k$$

Solution. It is

$$\begin{aligned} K\left(x,x'\right) &= \left(1+\langle x,x'\rangle\right)^k = \left(\sum_{j=0}^{n} x_j x'_j\right)^k \quad \text{(by setting } x_0 = x'_0 = 1\text{)} \\ &= \sum_{J\in\{0,1,\ldots,n\}^k} \prod_{i=1}^{k} x_{J_i} x'_{J_i} = \sum_{J\in\{0,1,\ldots,n\}^k} \left(\prod_{i=1}^{k} x_{J_i}\right)\left(\prod_{i=1}^{k} x'_{J_i}\right) \\ &= \langle \psi\left(x\right), \psi\left(x'\right)\rangle \end{aligned}$$

where $\psi\left(x\right)$ is as defined.

Example 24. (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)

$$K\left(x,x'\right) = \exp\left(-\frac{1}{2\sigma^2}\left\|x-x'\right\|_2^2\right).$$

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 $\psi\left(\cdot\right)$.

Solution. It is

$$\begin{aligned} K\left(x,x'\right) &= \exp\left(-\frac{1}{2\sigma^2}\left\|x-x'\right\|_2^2\right) = \exp\left(\frac{1}{\sigma^2}xx' - \frac{1}{2}x^2 - \frac{1}{2}\left(x'\right)^2\right) \\ &= \exp\left(\frac{1}{\sigma^2}xx'\right)\exp\left(-\frac{1}{2\sigma^2}x^2\right)\exp\left(-\frac{1}{2\sigma^2}\left(x'\right)^2\right) \\ &= \sum_{k=0}^{\infty}\frac{\left(xx'/\sigma^2\right)^k}{k!}\exp\left(-\frac{1}{2\sigma^2}x^2\right)\exp\left(-\frac{1}{2\sigma^2}\left(x'\right)^2\right) \quad \text{(...by Taylor Expansion)} \\ &= \sum_{k=0}^{\infty}\left[\frac{x^k}{\sqrt{k!}\sigma^k}\exp\left(-\frac{1}{2\sigma^2}x^2\right)\right]\left[\frac{\left(x'\right)^k}{\sqrt{k!}\sigma^k}\exp\left(-\frac{1}{2\sigma^2}\left(x'\right)^2\right)\right] \end{aligned}$$

hence it is $K\left(x,x'\right) = \langle \psi\left(x\right), \psi\left(x'\right)\rangle$ with $\psi_k\left(x\right) = \frac{x^k}{\sqrt{k!}\sigma^k}\exp\left(-\frac{1}{2\sigma^2}x^2\right)$.

4. Construction of kernels

Note 25. 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 22).

Note 26. 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 19 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 27. Theorem 30 provides sufficient and necessary conditions to check whether the specified function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ 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 28. Gram matrix is called the $m \times m$ matrix G s.t. $[G]_{i,j} = K(x_i, x_j)$.

Definition 29. A symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is positive semi-definite if its Gram matrix G, $[G]_{i,j} = K(x_i, x_j)$, is a positive semi-definite matrix.

Theorem 30. *A symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ 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 $\psi : \mathcal{X} \to \mathbb{R}^d$ 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 $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is positive semi-definite. Let $\mathbb{R}^f = \{f : \mathcal{X} \to \mathbb{R}\}$. 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 $K(x, x') = \langle \psi(x), \psi(x') \rangle$. □

Claim 31. In Figure 1.1c, we could see that examples/points can be distinguished by some ellipse, so it was reasonable for the scientist to define ψ as a vector with elements all the monomials up to order ; alternatively we could use a degree 2 polynomial kernel.

Proposition 32. *A powerful technique for constructing new kernels is to build them out of simpler kernels as building blocks. Below are some properties. Assume $K_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $K_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ 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

Solution. We present the first two and the rest are proved similarly.

See the proof as a solution to an Example

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

$$\begin{aligned} K(x, x') &= K_1(x, x') K_2(x, x') = (\psi_1(x))^\top \psi_1(x') (\psi_2(x))^\top \psi_2(x') \\ &= (\psi_1(x))^\top \psi_2(x) (\psi_1(x'))^\top \psi_2(x') = \left((\psi_1(x))^\top \psi_2(x)\right)^\top (\psi_1(x'))^\top \psi_2(x') \end{aligned}$$

which can be represented as an inner product of feature vectors.

Note 33. 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.

Machine Learning and Neural Networks (MATH3431) Epiphany term, 2024

Handout 9: Artificial neural networks

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

Aim. To introduce the Artificial neural network as a model and procedure in classical and Bayesian framework. Motivation, set-up, description, computation, implementation, tricks. We focus on the Feedforward network.

Reading list & references:

(1) Bishop, C. M., & Bishop, H. (2024). Deep learning: foundations and concepts. New York: Springer. [a]
- Ch. 5, 6, 8, 9

(2) Bishop, C. M. (1995). Neural networks for pattern recognition. Oxford university press.
- Ch. 4 The multi-layer perceptron

(3) LeCun, Y., Bottou, L., Orr, G. B., & Müller, K. R. (2002). Efficient backprop. In Neural networks: Tricks of the trade (pp. 9-50). Berlin, Heidelberg: Springer Berlin Heidelberg.

[a] As alternative & more compact see Ch. 5 Neural Networks from "Bishop, C. M. (2006). Pattern recognition and machine learning (Vol. 4, No. 4, p. 738). New York: Springer."

1. Intro and motivation

Note 1. Artificial Neural Networks (NN) are statistical models which have mostly been developed from the algorithmic perspective of machine learning. They were originally created as an attempt to model the act of thinking by modeling neurons in a brain. In ML, NN are used as global approximators.

Example 2. Consider a regression problem with predictive rule $h : \mathbb{R}^d \to \mathbb{R}^q$, suitable for cases where the examples (data) consist of input $x \in \mathbb{R}^d$, and output targets $y \in \mathbb{R}^q$. An example of a 2 layer artificial neural network formulating the predictive rule h is

$$h_k(x) = \sigma_2\left(w_{2,k,0} + \sum_j w_{2,k,j}\sigma_1\left(w_{1,j,0} + \sum_i w_{1,j,i}x_i\right)\right), \text{ for } k = 1, ..., q$$

where $\{\sigma_j(\cdot)\}$ are known functions and $\{w_{\cdot,\cdot,\cdot}\}$ are unknown weights/coefficients. E.g., one may choose with $\sigma_2(\alpha) = \alpha$, $\sigma_1(\alpha) = 1/(1 + \exp(-\alpha))$.

Note 3. The original biological motivation for feed-forward NN stems from McCulloch & Pitts (1943) who published a seminal model of a NN as a binary thresholding device in discrete time, i.e.

$$n_j(t) = 1\left(\sum_{\forall i \to j} w_{j,i} n_i(t-1) > \theta_i\right)$$

no need to memorize the formula

where the sum is over neuron i connected to neuron j; $n_j(t)$ is the output of neuron i at time t and $0 < w_{j,i} < 1$ are attenuation weights. Thus the effect is to threshold a weighted sum of the inputs at value θ_i. Perhaps, such a mathematical model involving compositions of interconnected non-linear functions could be able to mimic human's learning mechanism and be implemented in a computing environment (with faster computational abilities) with purpose to discover patters, make predictions, cluster, classify, etc...

Note 4. Mathematically, NN are rooted in the classical theorem by Kolmogorov stating (informally) that every continuous function $h(\cdot)$ on $[0,1]^d$ can be written as

no need to memorize the formula

$$h(x) = \sum_{i=1}^{2d+1} F_i \left(\sum_{j=1}^{d} G_{i,j}(x_j) \right) \tag{1.1}$$

where $\{G_{i,j}\}$ and $\{F_i\}$ are continuous functions whose form depends on f. Perhaps, one may speculate that functions $\{G_{i,j}\}$ and $\{F_i\}$ can be approximated by sigmoids or threshold functions of the form $\sigma\left(w^\top x\right)$ allowing the number of the tunable coefficients w to be high enough such that they can represent any function -hence the property of NN as global approximators.

Example 5. Recall soft-SVM in (Example 14 in Handout 8: Kernel methods); the predictive rule h^ψ had a formula

$$h^\psi(x) = \text{sign}\left(b + \langle w, \psi(x) \rangle\right) = \text{sign}\left(b + \sum_j w_j \psi_j(x) \right), \tag{1.2}$$

with some embedding $\psi(x) = (\psi_1(x), \psi_2(x) ...)^\top$ and unknown coefficients $\{w_j\}$. Assume there is no prior info about how to specify the formula of the "ideal" embedding ψ; then the scientist could try to parameterize ψ such as

$$\psi_j(x) = \zeta\left(\omega_{j,0} + \sum_i \omega_{j,i} x_i \right), \tag{1.3}$$

for some chosen function $\zeta(\cdot)$ and unknown coefficients $\{\omega_{j,i}\}$, hopping that the unknown coefficients $\{\omega_{j,i}\}$ can be successfully tuned during the learning procedure against the dataset and hence result in an adaptive semi-parametric way to construct the the embedding ψ from the training data without his/her interference. Based on Note 4, and assuming that a linear combination of $\{\zeta(\cdot)\}$, specified as sigmoids, could acceptably enough represent the fluctuations of $\{\psi_j(x)\}$ in (1.2), such an attempt to "adaptively construct the embedding ψ" might be successful.

Example 6. (Cont Example 5) Having this in the mind, and while wanting to improve expressiveness, the scientist could become even more greedy, and consider the use of another embedding in (1.3) as

$$\psi_j(x) = \zeta\left(\omega_{j,0} + \sum_i \omega_{j,i} \phi_i(x) \right),$$

whose bases $\phi(x) = (\phi_1(x), \phi_2(x) ...)^\top$ were parameterised as

$$\phi_i(x) = \xi\left(\varpi_{i,0} + \sum_s \varpi_{i,s} x_s\right)$$

for some chosen sigmoid $\xi(\cdot)$ and unknown coefficients $\{\varpi_{i,s}\}$. This recursive basis function composition creates a deep hierarchical structure which may improve approximation of the ideal embedding $\{\psi_j(x)\}$, and expressiveness of predictive rule h^ψ by sacrificing "interpretability" and "parsimony" of the "model".

2. Feedforward Neural Network (Mathematical set-up)

Note 7. An (Artificial) Neural Network (NN) is an interconnection of several sigmoids or threshold functions associated and arranged in layers, such that the outputs of one layer form the input of the next layer. Formally the structure of these interconnections can be depicted as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose nodes $\mathcal{V}$ correspond to vertices/neurons and edges $\mathcal{E}$ correspond to links between them.

Note 8. A Feed-Forward Neural Network (FFNN) or else multi-layer perceptron is a special case of NN whose vertices can be numbered so that all connections go from a neuron (vertex) to one with a higher number. Hence, neurons (vertices) have one-way connections to other neurons such that the output of a lower numbered neuron feeds the input of the higher numbered neuron (in a forward manner). The neurons can be arranged in layers so that connections go from one layer to a later layer. FFNN can be depicted by a directed acyclic graph[1], $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. (See Figure 2.1).

Note 9. Here we restrict ourselves to FFNN only.

Note 10. We assume that the network is organized in layers. The set of nodes is decomposed into a union of (nonempty) disjoint subsets

$$V = \cup_{t=0}^{T} V_t$$

such that every edge in $\mathcal{E}$ connects a node from V_t to a node from V_{t+1}, for $t = 1, ..., T$.

Note 11. The first layer V_0 is called **input layer**. If x has d dimensions, then the first layer V_0 contains d nodes.

Note 12. The last layer V_T is called **output layer**. If y has q dimensions, then the last layer V_T contains q nodes.

Note 13. The intermediate layers $\{V_1, ..., V_{T-1}\}$ are called **hidden layers.**

Note 14. In a neural network, the nodes of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ correspond to neurons.

Note 15. The **i-th neuron of the t-th layer** is denoted as $v_{t,i}$.

Note 16. The output of neuron i in the input layer V_0 is simply x_i that is $o_{0,i}(x) = x_i$ for $i = \{1, ..., d\}$.

[1](its vertices can be numbered so that all connections go from one vertex to another with a higher number)

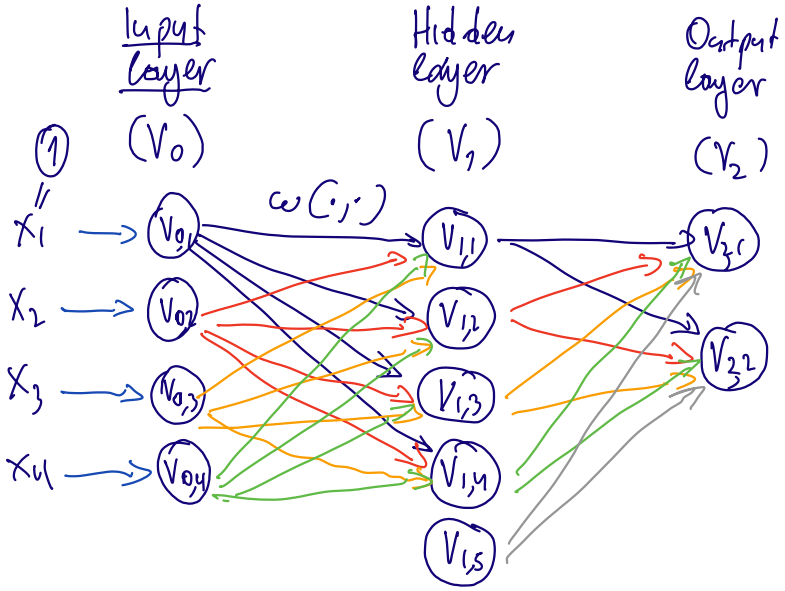


FIGURE 2.1. Feed forward neural network (1 hidden layer)

Note 17. Each edge in the graph $(v_{t,j}, v_{t+1,i})$ links the output of some neuron $v_{t,j}$ to the input of another neuron $v_{t+1,i}$; i.e. $(v_{t,j}, v_{t+1,i}) \in \mathcal{E}$.

Note 18. We define a **weight function** $w : \mathcal{E} \to \mathbb{R}$ over the edges $\mathcal{E}$.

Note 19. **Activation** of neuron i at hidden layer 1 is the weighted sum of the outputs $o_{0,i}(x) = x_i$ of the neurons in V_0 which are connected to $v_{1,i}$ where weighting is according to function w, that is

$$\alpha_{1,i}(x) = \sum_{\forall j:(v_{0,j},v_{1,i})\in\mathcal{E}} w\left((v_{0,j}, v_{1,i})\right) x_i \tag{2.1}$$

Note 20. Each single neuron $v_{t,i}$ is modeled as a simple scalar function, $\sigma_t(\cdot) : \mathbb{R} \to \mathbb{R}$, called **activation function** at layer t.

Note 21. **The output of neuron** $v_{t,i}$ when the network is fed with the input x is denoted as $o_{t,i}(x) := \sigma_{t-1}(\alpha_{t-1,i}(x))$.

Note 22. Activation of neuron i at layer t is the weighted sum of the outputs $o_{t-1,j}(x)$ of the neurons in V_{t-1} which are connected to $v_{t,i}$ where weighting is according to function w, that is

$$\alpha_{t,i}(x) = \sum_{\forall j:(v_{t-1,j},v_{t,i})\in\mathcal{E}} w\left((v_{t-1,j}, v_{t,i})\right) o_{t-1,j}(x) \tag{2.2}$$

Note 23. The input to $v_{t+1,i}$ is activation $\alpha_{t+1,i}(x)$ namely a weighted sum of the outputs $o_{t,j}(x)$ of the neurons in V_t which are connected to $v_{t+1,i}$, where weighting is according to w. The output of $v_{t+1,i}$ is the application of the activation function $\sigma_{t+1}(\cdot)$ on its input $\alpha_{t+1,i}(x)$.

Algorithm 24. *The feed-forward NN formula in a layer by layer manner is performed (defined) according to the following recursion.*

At $t = 0$**,:** *for* $i = 1, ..., |V_0|$

$$o_{0,i}(x) := x_i$$

At $t = 0, ..., T-1$**,:** *for* $i = 1, ..., |V_{t+1}|$

$$\begin{aligned}\alpha_{t+1,i}(x) =& \sum_{\forall j:(v_{t,j},v_{t+1,i})\in\mathcal{E}} w((v_{t,j},v_{t+1,i}))\, o_{t,j}(x)\\ o_{t+1,i}(x) =& \sigma_{t+1}(\alpha_{t+1,i}(x))\end{aligned}$$

Note 25. **Depth** of the NN is the number of the layers excluding the input layer; i.e. T.

Note 26. **Size** of the network is the number $|V|$.

Note 27. **Width** of the NN is the number $\max_{\forall t}(|V_t|)$.

Note 28. The **architecture** of the neural network is defined by the triplet $(\mathcal{V}, \mathcal{E}, \sigma_t)$.

Note 29. The neural network can be fully specified by the quadruplet $(\mathcal{V}, \mathcal{E}, \sigma_t, w)$.

Example 30. Figure 2.1 denotes a NN with depth 2, size 11, width 5. The neuron with no incoming edges has $o_{1,5} = \sigma(0)$.

Notation 31. To easy the notation, we denote the weights as $w_{(t+1),i,j} := w((v_{t,j}, v_{t+1,i}))$. Using this notation, $w_{(t+1),i,j} = 0$ is equivalent in (2.2) to $(v_{t,j}, v_{t+1,i}) \notin \mathcal{E}$ and means that the link $v_{t,j} \to v_{t+1,i}$ is not in the network.

Note 32. Often a **constant neuron** $v_{t,0}$ (at each layer t and $i = 0$) which outputs 1; i.e. $o_{0,0}(x) = 1$ and $o_{t,0}(x) = 1$. The corresponding weight $w_{(t),k,0}$ is called **bias**. This resembles to the constant term in the linear regression.

Example 33. (Cont. Example 2) The 2 layer neural network

$$h_k(x) = \sigma_{(2)}\left(w_{(2),k,0} + \sum_{\forall j} w_{(2),k,j}\sigma_{(1)}\left(w_{(1),j,0} + \sum_{\forall i} w_{(1),j,i}x_i\right)\right) \tag{2.3}$$

can be written according to the recursion in Algorithm 24 as

- Input layer

$$o_{(0),i}(x) = \begin{cases}1 & i = 0\\ x_i & i = 1, ..., d\end{cases}$$

- Hidden layer

$$\begin{aligned}\alpha_{(1),j}(x) &= w_{(1),j,0} + \sum_{\forall j} w_{(1),j,i} x_i \\ o_{(1),j}(x) &= \sigma_{(1)}\left(\alpha_{(1),j}(x)\right) \\ &= \sigma_{(1)}\left(w_{(1),j,0} + \sum_{\forall j} w_{(1),j,i} x_i\right)\end{aligned}$$

- Output layer

$$\begin{aligned}\alpha_{(2),k}(x) &= w_{(2),k,0} + \sum_{\forall j} w_{(2),k,j} o_{(2),k}(x) \\ o_{(2),k}(x) &= \sigma_{(2)}\left(\alpha_{(2),k}(x)\right) \\ &= \sigma_{(2)}\left(w_{(2),k,0} + \sum_{\forall j} w_{(2),k,j} o_{(2),k}(x)\right)\end{aligned}$$

If $h_k(x)$ returns values in $\mathbb{R}$, we can choose $\sigma_{(2)}$ as the identity function i.e., $\sigma_{(2)}(\alpha) = \alpha$. Note that (2.3) is also presented more compact

$$h_k(x) = \sigma_{(2)}\left(\sum_{\forall j} w_{(2),k,j} \sigma_{(1)}\left(\sum_{\forall i} w_{(1),j,i} x_i\right)\right) \tag{2.4}$$

by considering the constant neuron associated with first x which is set equal to 1, e.g $x_1 = 1$, similar to the linear regression models.

Note 34. Activation functions σ_t are non-increasing functions often sigmoids or threshold functions. Their choice is problem-dependent. some examples

- Identity function: $\sigma(\alpha) = \alpha$ cannot be used in hidden layers
- Threshold sigmoid: $\sigma(\alpha) = 1(\alpha > 0)$
- Logistic sigmoid: $\sigma(\alpha) = \frac{1}{1+\exp(-\alpha)}$
- Rectified linear unit: $\text{RELU}(\alpha) = \max(\alpha, 0)$

Example 35. Examples on the choice of the activation function at the output layer T:

- In the univariate regression problem with prediction rule $h(\cdot) \in \mathbb{R}$, and examples $z_i = (x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$, we can choose $\sigma_T(\alpha) = \alpha$ to get $h(\alpha) = \sigma_T(\alpha) = \alpha$.
- In the binary logistic regression problem with prediction rule $h(\cdot) \in [0,1]$ and examples $z_i = (x_i, y_i) \in \mathbb{R}^d \times \{0,1\}$, we can choose $\sigma_T(\alpha) = \frac{1}{1+\exp(-\alpha)}$ to get $h(\alpha) = \sigma_T(\alpha) = \frac{1}{1+\exp(-\alpha)} = \frac{\exp(\alpha)}{1+\exp(\alpha)}$.

3. Learning Neural Networks

Note 36. Assume we are interested in a prediction rule $h_{\mathcal{V},\mathcal{E},\sigma,w} : \mathbb{R}^{|V_0|} \to \mathbb{R}^{|V_T|}$ which is modeled as a feed-forward Neural Network with $(\mathcal{V}, \mathcal{E}, \sigma, w)$; that is

$$h_{\mathcal{V},\mathcal{E},\sigma,w}(x) = o_T(x)$$

where $o_T = \left(o_{T,1}, ..., o_{T,|V_T|}\right)^\top$ is according to the Algorithm 24.

Note 37. Learning the architecture $(\mathcal{V}, \mathcal{E}, \sigma)$ of a neural network is a model selection task (recall the variable selection in linear regression).

Note 38. We assume that the architecture $(\mathcal{V}, \mathcal{E}, \sigma)$ of the neural network $(\mathcal{V}, \mathcal{E}, \sigma, w)$ is fixed (given/known), and that there is interest in learning the weight function $w : \mathcal{E} \to \mathbb{R}$ or equivalently in vector form the vector of weights $\left\{w_{(t+1),i,j}\right\}$ where $w_{(t+1),i,j} := w\left((v_{t,j}, v_{t+1,i})\right)$.

Note 39. The class of hypotheses is

$$\mathcal{H}_{\mathcal{V},\mathcal{E},\sigma} = \{h_{\mathcal{V},\mathcal{E},\sigma,w} : \text{ for all } w : \mathcal{E} \to \mathbb{R}\}$$

for given $(\mathcal{V}, \mathcal{E}, \sigma)$.

Notation 40. To simplify notation we will use h_w instead of $h_{\mathcal{V},\mathcal{E},\sigma,w}$ as $\mathcal{V}, \mathcal{E}, \sigma$ is fixed here.

Note 41. Assume that there is available a training set of examples (data-set) $\mathcal{S} = \{z_i = (x_i, y_i) ; i = 1, ..., n\}$ with $x_i \in \mathcal{X} = \mathbb{R}^{|V_0|}$ and $y_i \in \mathcal{Y}$.

Note 42. To learn the unknown w, we need to specify a loss function $\ell(w, z)$ at some value of weight vector $w \in \mathbb{R}^{|\mathcal{E}|}$ and at some example $z = (x, y)$.

Definition 43. Error function is a performance measure that can be defined as

$$\text{EF}\left(w | \{z_i^*\}\right) = \sum_{i=1}^{n} \ell\left(w, z_i^*\right) \tag{3.1}$$

where $\mathcal{S}^* = \{z_i^* = (x_i^*, y_i^*) ; i = 1, ..., n^*\}$ is a set of examples which does not necessarily need to be the training set $\mathcal{S}$.

Example 44. (Regression problem) Assume we wish to predict the mapping $x \overset{h(\cdot)}{\to} y$, where $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$. Consider a predictive rule $h_w : \mathbb{R}^d \to \mathbb{R}$. Assume a training data-set $\{z_i = (x_i, y_i)\}$.

- The output activation function can be the identity function $\sigma_T(a) = a$. This is because $h_w(x) = o_T(x) = \sigma_T(\alpha_T(x))$ by Algorithm 24 and Note 36. Since h_w returns in $\mathbb{R}$ and the output activation $\alpha_T(x)$ returns in $\mathbb{R}$, then mapping $\sigma_T(a) = a$ suffices.
- A reasonable loss can be the Euclidean distance

$$\ell\left(w, z = (x, y)\right) = \frac{1}{2}\left(h_w(x) - y\right)^2$$

- Alternatively, if I consider a statistical model as

$$y_i | x, w \overset{\text{ind}}{\sim} \text{N}\left(\mu_i, \beta^{-1}\right), \text{ where } \mu_i = h_w(x_i) \tag{3.2}$$

for some fixed $\beta > 0$, the implied loss is

$$\begin{aligned}\ell(w, z=(x,y)) =& -\log\left(\mathrm{N}\left(y|h_w(x), \beta^{-1}\right)\right)\\ =& -\left(-\frac{1}{2}\log\left(\frac{1}{2\pi}\right) - \frac{1}{2}\log\left(\beta^{-1}\right) - \frac{1}{2}\frac{(h_w(x)-y)^2}{\beta^{-1}}\right)\\ =& \frac{1}{2}\beta(h_w(x)-y)^2 + \text{const...}\end{aligned}$$

- The Error function is

$$\mathrm{EF}(w|z) = \sum_{i=1}^{n}\ell(w, z_i=(x_i,y_i)) = \frac{1}{2}\beta\sum_{i=1}^{n}(h_w(x_i)-y_i)^2 + \text{const...}$$

Example 45. (Multi-output regression problem) Assume we wish to predict the mapping $x \overset{h(\cdot)}{\to} y$, where $x \in \mathbb{R}^d$ and $y \in \mathbb{R}^q$. Consider a predictive rule $h_w : \mathbb{R}^d \to \mathbb{R}^q$. Assume a training data-set $\{z_i = (x_i, y_i)\}$.

- The output activation function is the identity function $\sigma_T(a) = a$. This is because $h_{w,k}(x) = o_{T,k}(x) = \sigma_T(\alpha_{T,k}(x))$ for $k = 1, ..., q$ by Algorithm 24 and Note 36. Since h_w returns in $\mathbb{R}^q$ and the output activation is $\alpha_T(x)$ in $\mathbb{R}$, then mapping $\sigma_T(a) = a$ suffices.
- A reasonable loss can be an ellipsoidal norm

$$\ell(w, z=(x,y)) = (h_w(x)-y)^\top P(h_w(x)-y), \text{ for some } P > 0$$

- Alternatively, if I consider a statistical model as

$$y_i|x_i, w \overset{\text{ind}}{\sim} \mathrm{N}\left(\mu, P^{-1}\right), \text{ where } \mu_i = h_w(x_i) \tag{3.3}$$

for some fixed precision matrix $P > 0$, the implied loss is

$$\begin{aligned}\ell(w, z=(x,y)) =& -\log\left(\mathrm{N}\left(y|h_w(x), P^{-1}\right)\right)\\ =& -\left(-\frac{q}{2}\log\left(\frac{1}{2\pi}\right) + \frac{1}{2}\log(|P|) - \frac{1}{2}(h_w(x)-y)^\top P(h_w(x)-y)\right)\\ =& \frac{1}{2}(h_w(x)-y)^\top P(h_w(x)-y) + \text{const...}\end{aligned}$$

- The Error function is

$$\begin{aligned}\mathrm{EF}(w|z) =& \sum_{i=1}^{n}\ell(w, z_i=(x_i,y_i))\\ =& \frac{1}{2}\sum_{i=1}^{n}(h_{w,k}(x_i)-y_{k,i})^\top P(h_{w,k}(x)-y_{k,i}) + \text{const...}\end{aligned}$$

where $y_{k,i}$ is the k-th dimension of the i-th example in the dataset.

Example 46. (Binary classification problem) Assume we wish to classify objects with features $x \in \mathbb{R}^d$ in 2 categories. Consider a predictive rule $h_w : \mathbb{R}^d \to (0,1)$ as a classification probability i.e, $h_w(x) = \Pr(x \text{ belongs to class 1})$. Assume a training data-set $\{z_i = (x_i, y_i)\}$ with $y_i \in \{0, ..., q\}$ labeling the class.

- A suitable output activation function can be the logistic sigmoid

$$\sigma_T(a) = \frac{1}{1+\exp(-a)}$$

This is because $h_w(x) = o_T(x) = \sigma_T(\alpha_T(x))$ by Algorithm 24 and Note 36. Since h_w returns in $(0,1)$ and the output activation is $\alpha_T(x)$ in $\mathbb{R}$, then the aforesaid mapping suffices.

- A reasonable loss (among others) can be the cross entropy

$$\ell(w, z=(x,y)) = -y\log(h_w(x)) - (1-y)\log(1-h_w(x))$$

- If I consider a statistical model as

$$y_i|x_i, w \overset{\text{ind}}{\sim} \text{Bernoulli}(p_i), \text{ where } p_i = h_w(x_i) \tag{3.4}$$

with mass function

$$f(y|x,w) = h_w(x)^y (1-h_w(x))^{1-y}$$

the implied loss is

$$\begin{aligned}\ell(w, z=(x,y)) &= -\log(f(y|x,w)) \\ &= -y\log(h_w(x)) - (1-y)\log(1-h_w(x))\end{aligned}$$

- The Error function given a set of examples $\{z_i^* = (x_i^*, y_i^*)\}$ is

$$\begin{aligned}\text{EF}(w|z^*) &= \sum_{i=1}^{n} \ell(w, z_i^* = (x_i^*, y_i^*)) \\ &= -\sum_{i=1}^{n} y_i^* \log(h_w(x_i^*)) - (1-y_i^*)\log(1-h_w(x_i^*))\end{aligned}$$

Example 47. (Multi-class classification problem) Assume we wish to classify objects with features $x \in \mathbb{R}^d$ in q categories. Consider a predictive rule $h_w : \mathbb{R}^d \to \mathcal{P}$, with $\mathcal{P} = \left\{\varpi \in (0,1)^q : \sum_{j=1}^q \varpi_j = 1\right\}$ and $h_w = (h_{w,1}, ..., h_{w,q})^\top$, as a classification probability i.e, $h_{w,k}(x) = \Pr(x \text{ belongs to class } k)$. Assume a training data-set $\{z_i = (x_i, y_i)\}$ where y_i, with $y_{i,k} \in \{0,1\}$ and $\sum_{k=1}^q y_{i,k} = 1$, labeling the class to which the i-the example belongs. Then it is

- A suitable output activation function can be the softmax function

$$\sigma_T(a_k) = \frac{\exp(a_k)}{\sum_{k'=1}^q \exp(a_{k'})}, \text{ for } k = 1, ..., q \tag{3.5}$$

This is because $h_{w,k}(x) = o_{T,k}(x) = \sigma_{T,k}(\alpha_{T,k}(x))$ by Algorithm 24 and Note (36). Since h_w is essentially a probability vector and the output activation is $\alpha_{T,k}(x)$ in $\mathbb{R}$ the aforesaid mapping suffices. Unfortunately, (3.5) is invariant to additive transformations $a \leftarrow a + c$ i.e, $\sigma_T(a_k) = \sigma_T(a_k + \text{const})$.

- Another suitable output activation function can be

$$\tilde{\sigma}_T(a_k) = \frac{\exp(a_k)}{1+\sum_{k'=1}^{q-1} \exp(a_{k'})}, \text{ for } k = 1, ..., q-1 \tag{3.6}$$

This is because $h_{w,k}(x) = o_{T,k}(x) = \sigma_{T,k}(\alpha_{T,k}(x))$ by Algorithm 24 and Note 36. Since h_w is essentially a probability vector and the output activation is $\alpha_{T,k}(x)$ in $\mathbb{R}$ the aforesaid mapping suffices as $h_{w,k}(x) = o_{T,k}(x)$ for $k = 1, ..., q-1$ and $h_{w,q}(x) = 1 - \sum_{k=1}^{q-1} h_{w,k}(x)$. The advantage of (3.6) compared to (3.5) is that the former uses one less output neurons (less unknown parameters to learn). Also (3.6) is not invariant to additive transformations $a \leftarrow a + c$ unlike (3.5)i.e, $\tilde{\sigma}_T(a_k) \neq \tilde{\sigma}_T(a_k + \text{const})$.

- A reasonable loss can be the cross entropy

$$\ell(w, z = (x, y)) = -\sum_{k=1}^{q} y_k \log(h_{w,k}(x))$$

- If I consider a statistical model as

$$y_i|x, w \sim \text{Multinomial}(p_i), \text{ where } p_i = h_w(x_i)$$

with mass function

$$f(y_i|x, w) = \prod_{k=1}^{q} h_{w,k}(x_i)^{y_{i,k}}$$

the implied loss is

$$\begin{aligned}\ell(w, z = (x, y)) &= -\log(f(y|x, w)) \\ &= -\sum_{k=1}^{q} y_k \log(h_{w,k}(x))\end{aligned}$$

which is the cross-entropy.

- The Error function given a set of examples $\{z_i^* = (x_i^*, y_i^*)\}$ is

$$\begin{aligned}\text{EF}(w|z^*) &= \sum_{i=1}^{n} \ell(w, z_i^* = (x_i^*, y_i^*)) \\ &= -\sum_{i=1}^{n}\sum_{k=1}^{q} y_{k,i}^* \log(h_{w,k}(x_i^*))\end{aligned}$$

where $y_{k,i}$ is the k-th dimension of the i-th example in the dataset.

4. Classical learning of Neural Network

Note 48. Our purpose is to find optimal $h_w \in \mathcal{H}_{\mathcal{V},\mathcal{E},\sigma}$ under loss $\ell(\cdot, \cdot)$ and against training data-set $\mathcal{S} = \{z_i = (x_i, y_i); i = 1, ..., n\}$.

4.1. **Standard learning problem.**

Note 49. Essentially, this is an optimization problem, where the objective is to minimize either the risk function $R_g(w)$ or the empirical risk function $\hat{R}_S(w)$.

Problem 50. Compute optimal $w^* \in \mathbb{R}^{|\mathcal{E}|}$ by minimizing the risk function $R_g(w)$

$$w^* = \arg\min_w (R_g(w)) = \arg\min_w (\text{E}_{z\sim g}(\ell(w, z))) \tag{4.1}$$

Problem 51. Compute optimal $w^* \in \mathbb{R}^{|\mathcal{E}|}$ by minimizing the empirical risk function $\hat{R}_S(w)$

$$w^* = \arg\min_w \left(\hat{R}_S(w)\right) = \arg\min_w \left(\frac{1}{n}\sum_{i=1}^{n} \ell(w, z_i)\right) \tag{4.2}$$

4.2. **Regularized loss optimization.**

Note 52. Often neural network models are over-parameterized, in the sense that the dimensionality of $w \in \mathbb{R}^{|\mathcal{E}|}$ is too large, with negative consequences in its predictability. This can be addressed by learning the architecture NN, precisely by learning which of the edges of the FFNN significantly contribute to the NN model and should be kept, and which do not contribute and may be removed (or be inactive).

Note 53. To address Note 52, one can resort to shrinkage methods e.g., LASSO, Ridge, which indirectly allow edge/weight selection/elimination by shrinking the values of the weights $\left\{w_{(t),i,j}\right\}$ towards zero, and setting some of them as $w_{(t),i,j} = 0$ if their absolute value is small enough. This is based on the observation in (2.2) i.e., $w_{(t),i,j} = 0$ is equivalent to $(v_{t,j}, v_{t+1,i}) \notin \mathcal{E}$ which implies that the link $v_{t,j} \to v_{t+1,i}$ is not active (essentially not in the neural network).

Problem 54. Compute $h_w \in \mathcal{H}_{\mathcal{V},\mathcal{E},\sigma}$ (essentially compute $w \in \mathbb{R}^{|\mathcal{E}|}$) under loss $\ell(\cdot,\cdot)$ and shrinkage term (or weight decay) $J(w;\lambda)$, and against training data-set $\mathcal{S} = \{z_i = (x_i, y_i); i = 1, ..., n\}$. Compute $w^* \in \mathbb{R}^{|\mathcal{E}|}$, according to the minimization:

$$w^* = \arg\min_w \left(R_g(w) + J(w;\lambda)\right) \tag{4.3}$$

$$= \arg\min_w \left(\mathrm{E}_{z\sim g}\left(\ell(w, z) + J(w;\lambda)\right)\right) \tag{4.4}$$

and set $w^*_{t,i,j} = 0$ if $w^*_{t,i,j}$ is less than a threshold user specific value $\xi > 0$ i.e. $\left|w^*_{t,i,j}\right| < \xi$.

Note 55. Popular shrinkage terms $J(w;\lambda)$ are

- Ridge: $J(w;\lambda) = \lambda \|w\|_2^2$ Term 1
- LASSO: $J(w;\lambda) = \lambda \|w\|_1$ Term 1
- Elastic net: $J(w;\lambda = (\lambda_1, \lambda_2)) = \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2^2$

5. Stochastic gradient descent for classical training of NN

Note 56. Training a neural network model is usually a high-dimensional problem (essentially w has high dimensionality to provide a better approximation) and a big-data problem (essentially we need a large number of training examples to learn a large number of weights). For this reason, Stochastic Gradient Descent (and its variations) is a suitable computational learning tool.

Note 57. To address Problem 50, the recursion of the SGD with batch size m is

$$w^{(t+1)} = w^{(t)} - \eta_t \frac{1}{m}\sum_{j=1}^{m} \partial_w \ell\left(w^{(t)}, z_j^{(t)}\right)$$

Note 58. To address the Problem 54, the recursion of the SGD with batch size m is

$$w^{(t+1)} = w^{(t)} - \eta_t \left[\frac{1}{m} \sum_{j=1}^{m} \partial_w \ell \left(w^{(t)}, z_j^{(t)} \right) + \partial_w J \left(w; \lambda \right) \right]$$

for some positive λ which is user specified, or chosen via cross validation.

Note 59. The learning problem associated to the Neural network model is (almost always) non-convex due to the non-convex loss with respect to the w's. Upon implementing SGD in the learning problem of Neural Network, the theoretical results in Section 4 (Handout 2) and Section 3 (Handout 3) will not be effective due to the violation of the assumptions.

Note 60. Practical guidelines for the use of SGD in the learning problem of NN: Ref [3]

(1) Utilize a learning rate η_t that changes over the iterations. The choice of the sequence η_t is more significant. In practice, it is tuned by a trial and error manner: given a validation data-set $\mathcal{S}^*$ you may perform cross validation based on Error Function (3.1).
(2) Re-run the SGD procedure several times and by using different settings (learning rate η_t, batch size m) and different seed $w^{(0)}$ (randomly choosen) each time. Possibly, by luck, at some trial, we will initialize the SGD process with a random seed $w^{(0)}$ producing a trace leading to a good local minimum w^*.
(3) The output w^*_{SGD} returned by SGD is the best discovered w tested by using a performance measure (Error Function) using a validation set $\mathcal{S}^* = \{z_i^* = (x_i^*, y_i^*) \, ; i = 1, ..., n^*\}$; Eg.

$$w^*_{\text{SGD}} = \arg\min_{w^{(t)}} \left(\text{EF} \left(w^{(t)} | \, \{z_i^*\} \right) \right).$$

5.1. **Error backpropagation.**

Note 61. The error back-propagation procedure is an efficient algorithm for the computation of the gradient $\nabla_w \ell \left(w, z \right)$ of the loss function $\ell \left(w, z \right)$ at some value of w and some example $z = (x, y)$ as required for the implementation of stochastic gradient based algorithm to train the FFNN.

Note 62. Error backpropagation assumes that $\ell \left(w, z \right)$ is differentiable at w for each value of z; $\nabla_w \ell \left(w, z \right)$ exists –however extensions exist.

Notation 63. Let $V_t = \{v_{t,1}, ..., v_{t,k_t}\}$ be the t-th layer of a NN and $k_t = |V_t|$ the number of neuron in layer $t = 1, ...T$.

Algorithm 64. *(Error backpropagation)*

Requires: *The FFNN* $(\mathcal{V}, \mathcal{E}, \sigma, w)$ *with the values of the a weight vector* $w \in \mathbb{R}^{|\mathcal{E}|}$*, and example value* $z = (x, y)$

Returns: $\nabla_w \ell(w, z) = \left(\frac{\partial}{\partial w_{t,i,j}} \ell(w, z) \, ; \forall t, i, j \right)$

Initialize:

$$\text{Set } w_{t+1,j,i} = \begin{cases} w(v_{t,i}, v_{t+1,j}) & \text{if } (v_{t,i}, v_{t+1,j}) \in \mathcal{E} \\ 0 & \text{if } (v_{t,i}, v_{t+1,j}) \notin \mathcal{E} \end{cases}$$

Forward pass:

(1) *For* $i = 1, ..., d$

Set:

$$o_{0,i} = x_i$$

(2) *For* $t = 1, ..., T$ *; For* $i = 1, ..., k_t$

Compute:

$$\alpha_{t,i} = \sum_{j=1}^{k_{t-1}} w_{t,i,j} o_{t-1,j}$$

Compute:

$$o_{t,i} = \sigma_t(\alpha_{t,i})$$

Backward pass:

(1) *For* $t = T$ *; For* $i = 1, ..., k_T$

Compute:

$$\delta_{T,i} = \frac{\partial \ell_T}{\partial \alpha_{T,i}}(\alpha_T) \tag{5.1}$$

where ℓ_T *is the loss as a function of the vector of activations* $\alpha_T = (\alpha_{T,1}, ..., \alpha_{T,k_T})^\top$.

(2) *For* $t = T - 1, ..., 1$ *; For* $i = 1, ..., k_t$

Compute:

$$\delta_{t,i} = \frac{d}{d\alpha_{t+1,i}} \sigma_{t+1}(\alpha_{t+1,i}) \sum_{j=1}^{k_{t+1}} w_{t+1,j,i} \delta_{t+1,j} \tag{5.2}$$

Output:

For $t = 1, ..., T$*;* $i = 1, ..., k_t$ *;* $j = 1, ..., k_{t-1}$

If edge $(v_{t-1,j}, v_{t,i}) \in \mathcal{E}$*, set:*

$$\frac{\partial \ell}{\partial w_{t,i,j}}(w, z) = \delta_{t,i} \, o_{t-1,j}$$

Note 65. In (5.1) and (5.2), $\{\delta_{t,i}\}$ are called errors.

Example 66. Consider the multi-output regression problem, assume a predictive rule $h_w : \mathbb{R}^d \to \mathbb{R}^q$ with $h_w = (h_{w,1}, ..., h_{w,q})$ and

$$h_k(x) = \sigma_2\left(\sum_{j=1}^{c} w_{2,k,j}\sigma_1\left(\sum_{i=1}^{d} w_{1,j,i}x_i\right)\right)$$

with activation functions $\sigma_2(a) = a$, and $\sigma_1(a) = \tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}$, and loss $\ell(w,z) = \frac{1}{2}\sum_{k=1}^{q}(h_k(x) - y_k)^2$ for example $z = (x, y)$. Perform the error back propagation steps to compute the elements of $\nabla_w \ell(w,z)$ at some w and z.

Solution.

Forward pass:

Set: $o_{0,i} = x_i$ for $i = 1, ..., d$

Compute:

at $t = 1$**:** for $j = 1, ...c$

comp: $\alpha_{1,j} = \sum_{i=1}^{d} w_{1,j,i}x_i$

comp: $o_{1,j} = \tanh(\alpha_{1,j})$

at $t = 2$**:** for $k = 1, ...q$

comp: $\alpha_{2,k} = \sum_{j=1}^{c} w_{2,k,j}o_{1,j}$

comp: $o_{2,k} = \alpha_{2,k}$

get: $h_k = o_{2,k}$

Note that $\frac{\mathrm{d}}{\mathrm{d}\xi}\sigma_1(\xi) = 1 - (\sigma_1(\xi))^2$ and that $\frac{\mathrm{d}}{\mathrm{d}\xi}\sigma_2(\xi) = 1$.

Backward pass:

at $t = 2$**:** for $k = 1, ...q$

comp:

$$\delta_{2,k} = \frac{\partial}{\partial\alpha_{2,k}}\ell_T = \sum_{j=1}^{q}\frac{\partial\ell_T}{\partial o_{2,j}}(o_{2,j})\frac{\partial o_{2,j}}{\partial\alpha_{2,k}}(\alpha_{2,k}) = \frac{\partial\ell_T}{\partial o_{2,k}}(o_{2,k})\frac{\partial o_{2,k}}{\partial\alpha_{2,k}}(\alpha_{2,k}) = h_k - y_k$$

at $t = 1$**:** for $j = 1, ...c$

comp:

$$\delta_{1,j} = \frac{\mathrm{d}}{\mathrm{d}\xi}\sigma_1(\xi)\Big|_{\xi=\alpha_{1,j}}\sum_{k=1}^{q} w_{2,k,j}\delta_{2,k} = \left(1 - \left(\underbrace{\sigma_1(\alpha_{1,j})}_{=o_{1,j}}\right)^2\right)\sum_{k=1}^{q} w_{2,k,j}\delta_{2,k}$$

$$= \left(1 - (o_{1,j})^2\right)\sum_{k=1}^{q} w_{2,k,j}\delta_{2,k}$$

Output:

$$\frac{\partial\ell(w,z)}{\partial w_{1,j,i}} = \delta_{1,j}x_i \text{ and } \frac{\partial\ell(w,z)}{\partial w_{2,k,j}} = \delta_{2,k}o_{1,j}$$

Note 67. We show that the Algorithm 64 is valid. The Forward pass is valid due to Algorithm 24. Now, we work on the Backward pass.

- Let $\alpha_t = (\alpha_{t,1}, ..., \alpha_{t,k_t})^\top$ be the vector of the activations of the t-th layer of a NN.
- Let $\ell_t(\cdot)$ be the loss function $\ell(w, z)$ as a function of the vector of the activations α_t at t-th layer.
- It is

$$\frac{\partial \ell}{\partial w_{t,i,j}}(w, z) = \frac{\partial \ell_t}{\partial \alpha_{t,i}}(\alpha_t) \frac{\partial \alpha_{t,i}}{\partial w_{t,i,j}} = \delta_{t,i} \frac{\partial \alpha_{t,i}}{\partial w_{t,i,j}} \tag{5.3}$$

  where

$$\delta_{t,i} = \frac{\partial \ell_t}{\partial \alpha_{t,i}}(\alpha_t)$$

  for $i = 1, ..., k_t$, $j = 1, ..., k_{t-1}$ and $t = 1, ..., T$.
- Regarding the second part of (5.3), it is

$$\frac{\partial \alpha_{t,i}}{\partial w_{t,i,j}} = \frac{\partial}{\partial w_{t,i,j}} \sum_{s=1}^{k_{t-1}} w_{t,i,s} o_{t-1,s} = o_{t-1,i}$$

  for $t = 1, ..., T$ and $i = 1, ..., k_t$.
- I will try to find a way to compute δ_t given δ_{t+1} recursively from V_T to V_0.
  - For $t = T$, and $i = 1, ..., k_T$, it is

$$\delta_{T,i} = \frac{\partial \ell_T}{\partial \alpha_{T,i}}(\alpha_T)$$

  - For $t < T$, and $i = 1, ..., k_t$, it is

$$\delta_{t,i} = \frac{\partial \ell_t}{\partial \alpha_{t,i}}(\alpha_t) = \sum_{j=1}^{k_{t+1}} \frac{\partial \ell_{t+1}}{\partial \alpha_{t+1,j}}(\alpha_{t+1}) \frac{\partial \alpha_{t+1,j}}{\partial \alpha_{t,i}} = \sum_{j=1}^{k_{t+1}} \delta_{t+1,j} \frac{\partial \alpha_{t+1,j}}{\partial \alpha_{t,i}}$$

    and

$$\begin{aligned} \frac{\partial \alpha_{t+1,j}}{\partial \alpha_{t,i}} &= \frac{\partial}{\partial \alpha_{t,i}} \sum_{s=1}^{k_t} w_{t+1,j,s} o_{t,s} = \frac{\partial}{\partial \alpha_{t,i}} \sum_{s=1}^{k_t} w_{t+1,j,s} \sigma_{t+1}(\alpha_{t+1,s}) \\ &= w_{t+1,j,i} \frac{\mathrm{d}}{\mathrm{d}\alpha_{t+1,i}} \sigma_{t+1}(\alpha_{t+1,i}) \end{aligned}$$

    hence

$$\delta_{t,i} = \frac{\mathrm{d}}{\mathrm{d}\alpha_{t+1,i}} \sigma_{t+1}(\alpha_{t+1,i}) \sum_{j=1}^{k_{t+1}} \delta_{t+1,j} w_{t+1,j,i}$$

Note 68. Error back-propagation idea can also be used to efficiently compute the gradient $\nabla_x \ell(w, z = (x, y))$ of the loss function $\ell(w, z = (x, y))$ with respect to the inputs x. The idea is the same, use chain rule to find a recursive procedure from V_T to V_0.

5.2. **Preconditioning and computation of the Hessian.**

Note 69. Recall (Handout 3, Algorithm 45), that SGD may be improved by using a suitable preconditioner $P_t > 0$ as

$$w^{(t+1)} = w^{(t)} - \eta_t P_t \nabla_w \ell\left(w^{(t)}, z^{(t)}\right)$$

such a preconditioner can be the $P_t := [H_t + \epsilon I_d]^{-1}$ where H_t is the Hessian of $\ell\left(w^{(t)}, z^{(t)}\right)$ and $\epsilon > 0$.

Note 70. The computation of the Hessian H_t can be done by using error propagation ideas for

$$[H_t]_{i,j} = \left.\frac{\partial^2}{\partial w_i \partial w_j} f(w)\right|_{w=w^{(t)}}$$

We will not go further to exact computations.

Note 71. The exact computation of the Hessian H_t of the loss $\ell(\cdot, z)$ in NN setting can be computationally expensive and hence approximations are often used (with hope to work well). One can implement the general purpose AdaGrad (Section 7.1, Handout 3). In what follows we present other alternatives tailored to the NN model.

Note 72. Consider the regression problem with predictive rule $h : \mathbb{R}^d \to \mathbb{R}$ with $h_w(x) = o_T(x) = \alpha_T(x)$, and loss function $\ell(w, z = (x, y)) = \frac{1}{2}(h_w(x) - y)^2$. Then the Hessian of ℓ is

$$\begin{aligned} H &= \frac{\mathrm{d}(\nabla_w \ell)}{\mathrm{d}w} = \frac{\mathrm{d}}{\mathrm{d}w}\left(\nabla_w h_w(x)(h_w(x) - y)^\top\right) \\ &= \left(\frac{\mathrm{d}}{\mathrm{d}w}\nabla_w h_w(x)\right)(h_w(x) - y)^\top + \nabla_w h_w(x)(\nabla_w h_w(x))^\top \end{aligned} \tag{5.4}$$

We can expect that $h_w(x) \approx y$ provided that the network is well trained due to the global approximation ability of the FFNN. Yet, we can expect that $h_w = \mathrm{E}(y)$ provided that we train the network under the quadratic loss and because $\arg\min_h \mathrm{E}_{y\sim g}(h - y)^2 = \mathrm{E}_{y\sim g}(y)$. Hence a reasonable approximation can be

$$\begin{aligned} H &\approx \nabla_w h_w(x)(\nabla_w h_w(x))^\top \\ &\overset{\sigma_T(a)=a}{=} \nabla_w \alpha_T(x)(\nabla_w \alpha_T(x))^\top \end{aligned}$$

Consequently, the approximation of the Hessian of the Error function $\mathrm{EF}(w|\{z_i\}) = \sum_{i=1}^n \ell(w, z_i = (x_i, y_i))$ is

$$H_n = \sum_{i=1}^n \frac{\mathrm{d}(\nabla_w \mathrm{EF})}{\mathrm{d}w} \approx \sum_{i=1}^n \nabla_w h_w(x_i)(\nabla_w h_w(x_i))^\top \tag{5.5}$$

Note 73. (Cont. Note 72) To efficiently compute the inverse H_n^{-1} of (5.5) I can utilize Woodbury identity

$$\left(M + vv^\top\right)^{-1} = M^{-1} - \frac{\left(M^{-1}v\right)\left(v^\top M^{-1}\right)}{1 + v^\top M^{-1} v} \tag{5.6}$$

No need to memorize Woodbury identity

offering a way to avoid the computational expensive task of directly inverting the high dimensional H_n. Given $v_i = \nabla_w h_w(x)$, it is

$$(H_n)^{-1} = \left(\sum_{i=1}^{n} v_i v_i^\top\right)^{-1} = \left(\sum_{i=1}^{n-1} v_i v_i^\top + v_n v_n^\top\right)^{-1} = \left(H_{n-1} + v_n v_n^\top\right)^{-1} \tag{5.7}$$
$$= H_{n-1}^{-1} - \frac{\left(H_{n-1}^{-1} v_n\right)\left(v_n^\top H_{n-1}^{-1}\right)}{1 + v_n^\top H_{n-1}^{-1} v_n}$$

In practice I start with $H_0 = \epsilon I$ with $\epsilon > 0$ small, and iterate (5.7).

Note 74. Likewise and by modifying (5.4), one can compute the corresponding approximations for the classification problems or problems with different loss functions.

6. Bayesian Artificial Neural Networks

Note 75. Consider a feed-forward Neural Network with $(\mathcal{V}, \mathcal{E}, \sigma, w)$. Assume the architecture $(\mathcal{V}, \mathcal{E}, \sigma)$ of a neural network is fixed/known, and interest lies in learning the weights $\{w_{t,i,j}\}$.

Problem 76. Assume there is available a training set of examples (data-set) $\mathcal{S} = \{z_i = (x_i, y_i); i = 1, ..., n\}$ with $x_i \in \mathcal{X} = \mathbb{R}^{|V_0|}$ and $y_i \in \mathcal{Y}$. The Bayesian NN model is then

$$\begin{cases} y_i | x_i, w & \overset{\text{ind}}{\sim} f(y_i | x_i, w), \ i = 1, ..., n \ \text{(sampling distribution)} \\ w & \sim f(w) \qquad\qquad\qquad\quad \text{(prior distribution)} \end{cases}$$

where the sampling distribution is specified either according to the experimental design (how $\{z_i\}$) are collected or based on subjective judgments, while the prior distribution is specified based on subjective manner.

Note 77. In most of the real applications, it is difficult (almost impossible) to specify the sampling distribution based on the experimental design or judgments, and almost impossible to specify the prior of the weights based on subjective judgments.

Note 78. One could specify the sampling distribution based on the loss function $\ell(h_w(x), z = (x, y))$ as

$$f(y_i | x_i, w) \propto \exp\left(-\ell\left(h_w(x_i), (x_i, y_i)\right)\right), \tag{6.1}$$

based on the argument that sampling distribution in the posterior distribution contraindicates how far the theoretical model $h_w(x_i)$ (as casted in a NN) is from the corresponding observation y_i via the likelihood.

Note 79. The prior density of the weights may be specified based on some shrinkage term $J(w; \lambda)$ (Note 55) as

$$f(w) \propto \exp\left(-J(w; \lambda)\right) \tag{6.2}$$

based on the argument that I often model the predictive rule $h_w(\cdot)$ with an over-parametrized NN (with many weighs, more than needed) and many of them should be shrunk to zero. Also as we

see in Note 88, careless training of NN tends to produce over-fitted NN with large weights (in abs values).

Note 80. Regarding prior it is often, $w \sim \mathrm{N}\left(0, I\lambda^{-1}\right)$ for some small λ controlling prior uncertainty about weights, i.e. 50% chances for the weight to be above or below zero.

Note 81. The corresponding posterior (according to the Bayes theorem) is

$$f\left(w|\left\{z_i\right\}\right) = \frac{\prod_{i=1}^{n} f\left(z_i|w\right) f\left(w\right)}{\int \prod_{i=1}^{n} f\left(z_i|w'\right) f\left(w'\right) \mathrm{d}w'} \propto \prod_{i=1}^{n} f\left(z_i|w\right) f\left(w\right)$$

and given 6.1 and 6.2, in log scale I get

$$\log\left(f\left(w|\left\{z_i\right\}\right)\right) = -\sum_{i=1}^{n} \ell\left(h_w\left(x_i\right), \left(x_i, y_i\right)\right) - J\left(w;\lambda\right) + \text{const}$$

that resembles the EF with a shrinkage term in classical learning.

Example 82. (Regression problem) Sampling distribution can be specified as in (6.1) based on (Euclidean) loss

$$\ell\left(w, z = (x, y)\right) = \frac{\beta}{2}\left(h_w\left(x\right) - y\right)^2$$

which (based on (3.4)) can build a density

$$f\left(y_i|x_i, w\right) \propto \exp\left(-\ell\left(h_w\left(x_i\right), \left(x_i, y_i\right)\right)\right) = -\frac{\beta}{2}\left(h_w\left(x_i\right) - y_i\right)^2,$$

This implies a sampling distribution

$$y_i|x_i, w \sim \mathrm{N}\left(\mu_i, \beta^{-1}\right), \text{ where } \mu_i = h_w\left(x_i\right) \tag{6.3}$$

for some fixed $\beta > 0$ based on (3.2). Prior can be specified with density (6.2) where

$$J\left(w;\lambda\right) = \frac{\lambda}{2}\left\|w\right\|_2^2$$

which is based on the Ridge shrinkage term. This implies a prior

$$w \sim \mathrm{N}\left(0, I\lambda^{-1}\right) \tag{6.4}$$

The resulted posterior density is

$$f\left(w|\left\{z_i\right\}\right) \propto \exp\left(-\frac{\beta}{2}\sum_{i=1}^{n}\left(h_w\left(x_i\right) - y_i\right)^2 - \frac{\lambda}{2}\left\|w\right\|_2^2\right) \tag{6.5}$$

Example 83. (Binary classification problem) Sampling distribution can be specified as in (6.1) based on (cross-entropy) loss

$$\ell\left(w, (x, y)\right) = y\log\left(h_w\left(x\right)\right) + \left(1 - y\right)\left(1 - \log\left(h_w\left(x\right)\right)\right)$$

which (based on (3.4)) can build a density

$$\begin{aligned} f\left(y_i|x_i, w\right) &\propto \exp\left(-\ell\left(h_w\left(x_i\right), \left(x_i, y_i\right)\right)\right) \\ &= -y_i\log\left(h_w\left(x_i\right)\right) - \left(1 - y_i\right)\left(1 - \log\left(h_w\left(x_i\right)\right)\right), \end{aligned}$$

This implies a sampling distribution

$$y_i|x_i, w \sim \text{Bernoulli}\left(p_i\right), \text{ where } p_i = h_w\left(x_i\right)$$

To encourage sparsity, prior can be specified as in (6.2) by using the LASSO shrinkage term

$$J\left(w;\lambda\right) = \lambda \left\|w\right\|_1;$$

The resulted posterior density is

$$f\left(w|\left\{z_i\right\}\right) \propto \exp\left(-\sum_{i=1}^{n}\left[y_i \log\left(h_w\left(x_i\right)\right) + \left(1-y_i\right)\left(1-\log\left(h_w\left(x_i\right)\right)\right)\right] - \lambda\left\|w\right\|_1\right)$$

Note 84. Sampling from the posterior can be performed via SGLD due to the high-dimensionality in the weights w and the big size of the training data set. Recall the recursion of the SGLD with batch size m is

$$w^{(t+1)} = w^{(t)} + \eta_t\left(\frac{n}{m}\sum_{j\in\mathcal{J}^{(t)}}\nabla_w \log\left(f\left(z_j^{(t)}|w^{(t)}\right)\right) + \nabla_w \log\left(f\left(w^{(t)}\right)\right)\right) + \sqrt{\eta_t}\sqrt{\tau}\epsilon_t, \tag{6.6}$$

for $\epsilon_t \sim \text{N}\left(0,1\right)$. (See Algorithm 21 in Handout 4: Bayesian Learning via Stochastic gradient and Stochastic gradient Langevin dynamics).

Note 85. Given sample values $\left\{w^{(t)}\right\}_{t=1}^{T}$ produced from SGLD recursions (6.6), learning of any function $h_w\left(x\right)$ of the w's can be performed via

(1) Monte Carlo estimation namely averaging the samples values $\left\{w^{(t)}\right\}_{t=1}^{T}$ as

$$\widehat{h_w}\left(x\right) = \frac{1}{T}\sum_{t=1}^{T} h_{w^{(t)}}\left(x\right), \quad \text{(Monte Calro estimator)}$$

(2) Maximum-A-posteriori (MAP) estimation namely find the best $\hat{w}^*$ among the samples values $\left\{w^{(t)}\right\}_{t=1}^{T}$ that maximizes the posterior i.e

$$\hat{w}^* = \arg\max_{w\in\left\{w^{(t)}\right\}_{t=1}^{T}}\left(\log f\left(\left\{z_i\right\}|w\right) + \log f\left(w\right)\right)$$

and compute

$$\widehat{h_w}\left(x\right) = h_{\hat{w}^*}\left(x\right), \quad \text{(MAP estimator)}$$

7. Theoretical aspects

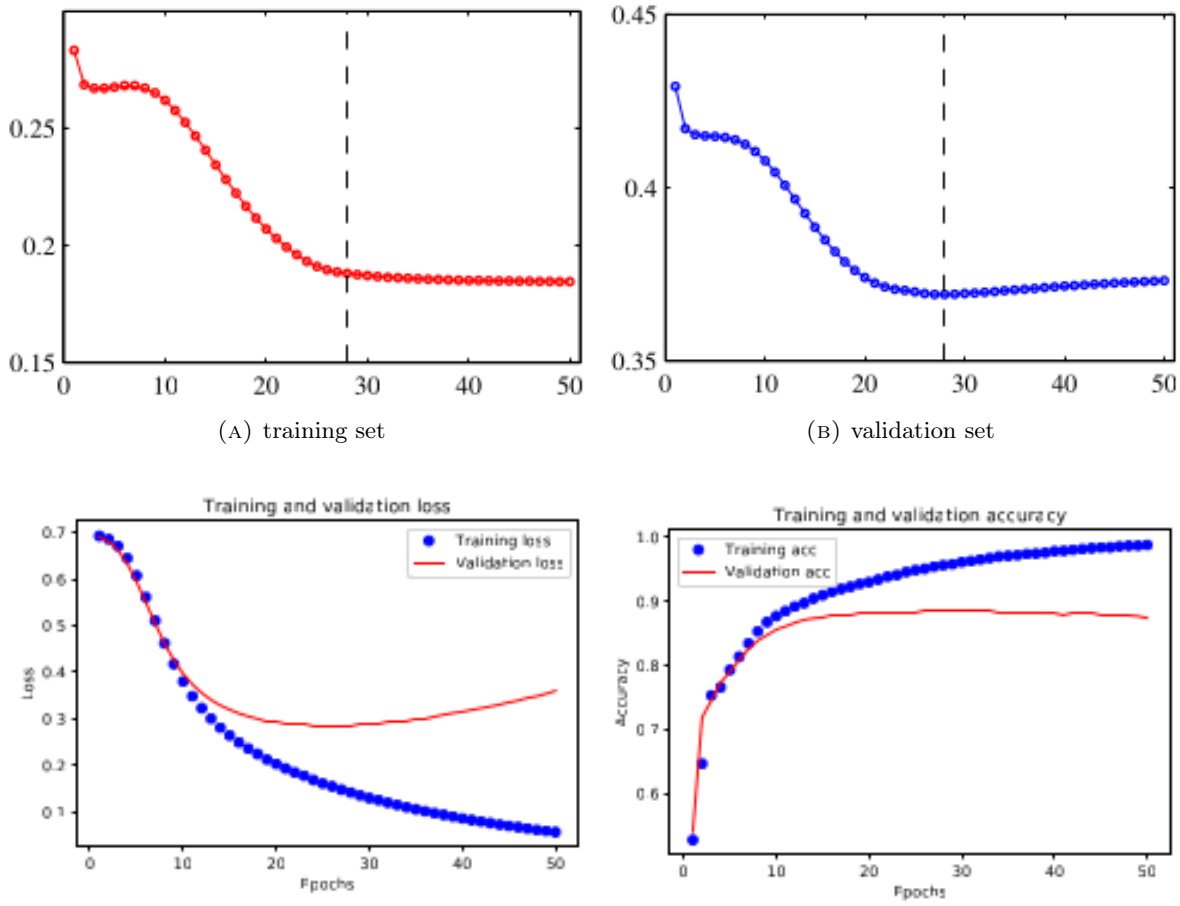
Note 86. We will not go this direction here. For the interested student, I suggest for starters, Ch 5 from "Ripley, B. D. (2007). Pattern recognition and neural networks. Cambridge university press."; and for advanced Ch 30 from "Devroye, L., Györfi, L., & Lugosi, G. (2013). A probabilistic theory of pattern recognition (Vol. 31). Springer Science & Business Media."

8. Comments, guidelines, and discussions

8.1. **Over-fitting issues.**

Note 87. Neural Networks can be "over-parametrized", for instance by consisting of a large number of layers each of them having a large number of neurons able to represent each feacture/characteristic of the pattern of interest to be learned. Careless training may produce NN models with bad generalization predictive properties.

Note 88. Training of non-linear (non-convex) NN models corresponds to an iterative reduction of the Error Function defined on the training data-set. It has been observed that the Error Function defined on the validation data-set (independent to the training data set) often shows a decrease at first, followed by an increase as the NN starts to overfit. This overfit is often associated to the production of weight values larger than needed to be generalised. It is desirable to avoid this over-fitting with purpose to obtain a NN with good generalization performance.



(A) training set (B) validation set

(C) loss function (D) accuracy

FIGURE 8.1. Behavior of the Error Function wrt the iterations, against a training set and against a validation set.

Note 89. **Early stopping** is a way of limiting the effective network complexity by halting training before a minimum of the training error has been reached. During training, we monitor the NN performance against the EF defined on the validation data-set, and stop the training procedure

when just before the EF defined on the validation data set start increasing (aka before overfitting signs) to prevent the model memorizing too much information about the training set.

Note 90. **Regularization** as in Section 4.2 can alternatively be used to address the above issue by using a shrinkage term preventing the weights to grow too much away from zero.

8.2. **Non-identifiabilities.**

Note 91. NN Learning Problems (E.g, Problems 50, 51, 54, and 76) are often non-identifiable with respect to weights $\{w_{t,i,j}\}$.

Example 92. Consider the FFNN in Figure 8.2 with 1 hidden layer of M neurons, activation function $\sigma_1(a) = \tanh(a)$ and and full connectivity in both layers. If we flip the sign of all of the w's feeding into a particular hidden unit (for a given input pattern) then the sign of the activation of the hidden unit will be reversed because $\tanh(-a) = -\tanh(a)$. This transformation can be compensated by changing the sign of all of the w's leading out of that hidden unit. By changing the signs of a particular group of w's, the input-output mapping function represented by the FFNN is unchanged, and so there are two different weight vectors resulting the same mapping function. I can do M such flips (there are M hidden nuerons) leading to 2^M equivalent parameter settings. Similarly, we can permulte the labels of the neurons in hidden layer without changing the loss function; there are $M!$ such permutations. Hence the equivalent parameter settings in total are $2^M M!$. –This is not harmful in training or predictive rul computation as we just need to find one such parameter setting.

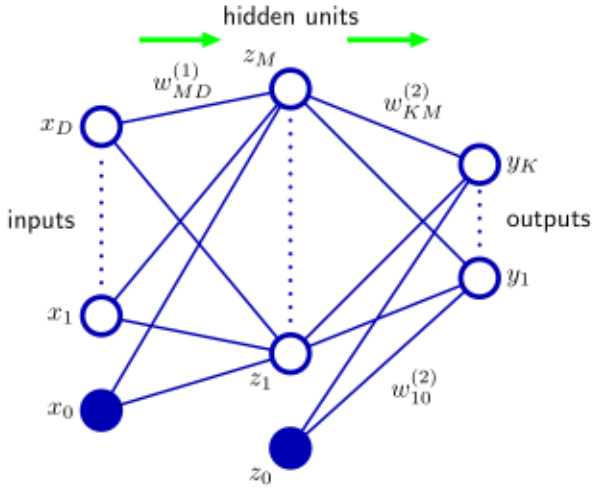


FIGURE 8.2. A FFNN

8.3. **Non-convexity.**

Note 93. Learning problems with NN are non-convex, hence the loss function to be minimized has many local minima. The consequence is that the SGD/SGLD learning algorithms may be trapped in a local optima and never reach any of the global ones. The number of local minima is exaggerated

due to the symmetries discussed in Note 91. Due to the large number of data (big-data) used to train the NN (in real life) such local minima become more shallow while the global minima more picky. Due to this and the stochastic nature of SGD/SGLD algorithms, SGD/SGLD may be able (by chance) to escape from such local minima/maxima and reach the global ones.

Note 94. To address local optima issue, run SGD/SGLD learning algorithms multiple times by initializing them with different seeds each time.

8.4. **Skip-layers 2.4.**

Note 95. The general definition of feed forward Neural Network allows "skip-layer" connections that is the directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in NN architecture includes edges which may not necessarily connect neurons between consecutive layers, namely $(v_{t,j}, v_{t+k,i}) \in \mathcal{E}$ for some $k > 1$.

Example 96. For instance, a fully connected FFNN with 1 hidden layer is

$$h_k(x) = \sigma_{(2)}\left(\underbrace{\sum_{\forall i} w^{\text{skip}}_{(1),k,i} x_i}_{\text{skip layer terms}} + \sum_{\forall j} w_{(2),k,j}\sigma_{(1)}\left(\sum_{\forall i} w_{(1),j,i} x_i\right)\right) \tag{8.1}$$

Compared to (2.4) in Example 33 and (2.4), (8.1) links the input to the output.

Note 97. In skip-layer FFNN cases, the error back propagation (Section 5.1) has to be adjusted properly.

Note 98. FFNN are mainly used without skip-layers because theory (Section 7) states that FFNN are global approximators even without skip layers, as well as because of computational inconvenience and modeling parsimony.

Appendix A. About graphs

Definition 99. A directed graph is an ordered pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ comprising

- a set of nodes $\mathcal{V}$ (or vertices), where a nodes are abstract objects, and
- a set of edges $\mathcal{E} = \{(v, u) \,|v \in \mathcal{E}, u \in \mathcal{E}, v \neq u\}$ (or directed edges, directed links, arrows) which are ordered pairs of vertices (that is, an edge is associated with two distinct vertices).

Definition 100. An edge-weighted graph or a network is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ equipped with a weight function $w : \mathcal{E} \to \mathbb{R}$ that assigns a number (the weight) to each edge $e \in \mathcal{E}$.

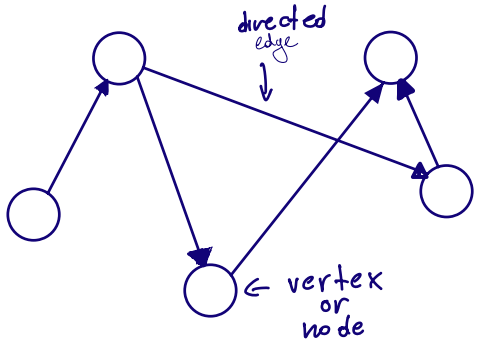


Figure A.1. A directed graph

Appendix B. About Partial derivatives

Note 101. Consider a function $f : \mathbb{R}^n \to \mathbb{R}$.

Definition 102. The partial derivative of $f : \mathbb{R}^n \to \mathbb{R}$ at the point $a = (a_1, \ldots, a_n) \in U \subseteq \mathbb{R}^n$ with respect to the i-th variable is denoted as

$$\frac{\partial f}{\partial x_i}(a) \text{ or } \left.\frac{\partial f}{\partial x_i}(x)\right|_{x=a}$$

and defined as

$$\begin{aligned}\left.\frac{\partial f}{\partial x_i}(x)\right|_{x=a} &= \lim_{h\to 0} \frac{f(a_1, ..., a_{i-1}, a_i + h, a_{i+1}, ..., a_n) - f(a_1, ..., a_{i-1}, a_i, a_{i+1}, ..., a_n)}{h} \\ &= \lim_{h\to 0} \frac{f(a + he_i) - f(a)}{h}\end{aligned}$$

where e_i is a $0-1$ vector with only one ace in the i-th location.

Remark 103. Essentially, Definition 102 says that the partial derivative of $f : \mathbb{R}^n \to \mathbb{R}$ at the point $a = (a_1, \ldots, a_n) \in U \subseteq \mathbb{R}^n$ with respect to the i-th variable is

$$\frac{\partial f}{\partial x_i}(a) = \left.\frac{\mathrm{d}g}{\mathrm{d}h}(h)\right|_{h=0}$$

the derivative of function $g(h) := f(a_1, ..., a_{i-1}, a_i + h, a_{i+1}, ..., a_n)$ at value 0.

Example 104. Consider a function f with $f(x_1, x_2) = x_1^2 + x_1 x_2^3$. Comput is partial derivatives at $a = (2, 3)^\top$; i.e. $\frac{\partial f}{\partial x_1}(a)$ and $\frac{\partial f}{\partial x_2}(a)$.

Solution. It is

$$\begin{aligned}
\frac{\partial f}{\partial x_1}(a) &= \frac{\partial f}{\partial x_1}(x)\Big|_{x=a} = \frac{\mathrm{d}}{\mathrm{d}x_1}\left(x_1^2 + x_1 x_2^3\right)\Big|_{x=a} = 2x_1 + x_2^3\big|_{x=a} \\
&= 2a_1 + a_2^3 = 4 + 27 = 31 \\
\frac{\partial f}{\partial x_2}(a) &= \frac{\partial f}{\partial x_2}(x)\Big|_{x=a} = \frac{\mathrm{d}}{\mathrm{d}x_2}\left(x_1^2 + x_1 x_2^3\right)\Big|_{x=a} = 2x_1 x_2^2\big|_{x=a} \\
&= 2a_1 a_2^2 = 4 + 27 = 72
\end{aligned}$$