Machine Learning and Neural Networks III (MATH3431)

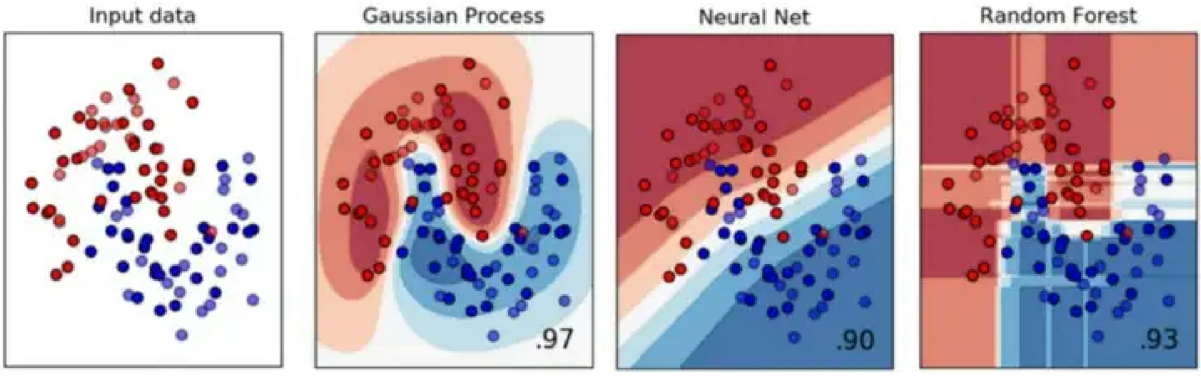
Epiphany term

Georgios P. Karagiannis

georgios.karagiannis@durham.ac.uk

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

2023/02/16 at 11:27:31



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 It is very comprehensive (compared to others) and it goes deep enough for the module although it may be a bit outdated.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

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

Handout 0: Learning problem: Definitions, notation, and formulation –A recap

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:

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

Supervised learning problems involve applications where the training data $z \in \mathcal{Z}$ comprises 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 recover 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 $y \in \mathcal{Y}$ 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 $y \in \mathcal{Y} \subseteq \mathbb{R}$. Hypothesis is a linear function $h : \mathcal{X} \to \mathcal{Y}$ (that learner wishes to learn) to approximate mapping $x \to y$. The hypothesis set $\mathcal{H} = \left\{x \to \langle w, x\rangle : w \in \mathbb{R}^d\right\}$. We can use the loss $\ell\left(h, (x, y)\right) = \left(h(x) - y\right)^2$.

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

Definition 5. Data generation model $g\left(\cdot\right)$ is the probability distribution over $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, unknown to the learner that has generated the data.

Definition 6. We denote as $\mathfrak{A}\left(\mathcal{S}\right)$ the hypothesis (outcome) that a learning algorithm $\mathfrak{A}$ returns given training sample S.

Definition 7. (Loss function) Given any set of hypothesis $\mathcal{H}$ and some domain $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, a loss function $\ell\left(\cdot\right)$ is any function $\ell : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}_+$. The purpose of loss function $\ell\left(h, z\right)$ is to quantify the "error" for a given hypothesis h and example z –the greater the error the greater its value of the loss.

Example 8. (Cont. Example 3) In regression problems $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ and $\mathcal{Y} \subset \mathbb{R}$ is uncountable, a loss function can be

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

Example 9. In binary classification problems with $h : \mathcal{X} \to \mathcal{Y}$ a learner where $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ and $\mathcal{Y} = \{0, 1\}$ is discrete, a loss function can be

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

Definition 10. (Risk function) The risk function $R_g\left(h\right)$ of h is the expected loss of the hypothesis $h \in \mathcal{H}$, w.r.t. probability distribution g over domain Z; i.e.

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

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

Remark 11. In learning, an ideal way to obtain an optimal predictor h^* is to compute the risk minimizer

$$h^* = \arg\min_{\forall h} \left(R_g\left(h\right)\right)$$

Example 12. (Cont. Ex. 8) The risk function is $R_g\left(h\right) = \mathrm{E}_{z\sim g}\left(h(x)-y\right)^2$, and it measures the quality of the hypothesis function $h : \mathcal{X} \to \mathcal{Y}$, (or equiv. the validity of the class of hypotheses $\mathcal{H}$) against the data generating model g, as the expected square difference between the predicted values form h and the true target values y at every x.

Note 13. 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 resort to the Empirical risk function.

Definition 14. (Empirical risk function) The empirical risk function $\hat{R}_S\left(h\right)$ of h is the expectation of loss of h over a given sample $S = (z_1, ..., z_m) \in \mathcal{Z}^m$; i.e.

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

Example 15. (Cont. Example 12) Given given sample $S = \left\{\left(x_i, y_i\right); i = 1, ..., m\right\}$ the empirical risk function is $\hat{R}_S\left(h\right) = \frac{1}{m}\sum_{i=1}^{m}\left(h(x_i) - y_i\right)^2$.

Example 16. 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 $f_\theta\left(y\right) := f\left(y|\theta\right)$ labeled by an unknown parameter θ. The goal is to learn θ. If we assume loss function

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

then the risk is

$$R_g\left(\theta\right) = \mathrm{E}_{z\sim g}\left(\log\left(\frac{g\left(z\right)}{f_\theta\left(z\right)}\right)\right) = \mathrm{E}_{z\sim g}\left(\log\left(g\left(z\right)\right)\right) - \mathrm{E}_{z\sim g}\left(\log\left(f_\theta\left(z\right)\right)\right) \tag{2.2}$$

whose minimizer is

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

as the first term in (2.2) is constant. Note that in the Maximum Likelihood Estimation technique the MLE θ_{MLE} is the minimizer of

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

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

Definition 17. 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 18. Consider the multiple linear regression problem $x \longmapsto y$ where $x \in \mathcal{X} \subseteq \mathbb{R}^d$ and $y \in \mathcal{Y} \subseteq \mathbb{R}$. Till now, we set the learning problem $(\mathcal{H}, \mathcal{Z}, \ell)$ in the linear regression with hypothesis class $\mathcal{H} = \left\{x \to \langle w, x\rangle : w \in \mathbb{R}^d\right\}$, loss $\ell\left(h, (x, y)\right) = \left(h(x) - y\right)^2$, and examples domain $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ with $\mathcal{X} \in \mathbb{R}^d$ and $\mathcal{Y} \in \mathbb{R}$. Because learning problem involves only linear functions as predictors $h\left(x\right) = \langle w, x\rangle$, this learning problem could be defined equivalently with a hypothesis class $\mathcal{H} = \left\{w \in \mathbb{R}^d\right\}$ and loss function loss $\ell\left(w, (x, y)\right) = \left(\langle w, x\rangle - y\right)^2$. The latter will be mainly used.

Appendix A. Useful bits

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

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

Also some standard formulas.

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

$$\begin{cases} f\left(\mathrm{E}\left(x\right)\right) \leq \mathrm{E}\left(f\left(x\right)\right) & \text{if } f \text{ is convex} \\ f\left(\mathrm{E}\left(x\right)\right) \geq \mathrm{E}\left(f\left(x\right)\right) & \text{if } f \text{ is concave} \end{cases}$$

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

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

Handout 1: 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 (theoretical) analysis of the learning problems and their solution that we will discuss (eg stochastic gradient descent) later on. Also learning problems with such characteristics can be learned more efficiently.

Note 2. Most of the ML problems discussed in the course (eg, Artificial neural networks, Gaussian process regression) are usually non-convex.

Note 3. 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. Convex learning problems

Definition 4. 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 5. Multiple linear regression $< w, x > \rightarrow y$ with $y \in \mathbb{R}$, hypothesis class $\mathcal{H} = \left\{ w \in \mathbb{R}^d \right\}$ and loss $\ell\left(w, (x, y)\right) = \left(< w, x > -y\right)^2$ with

$$w^* = \arg\min_{\forall w} \mathrm{E}\left(< w, x > -y\right)^2$$

or

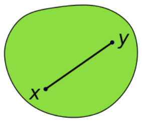
$$w^{**} = \arg\min_{\forall w} \frac{1}{m} \sum_{i=1}^{m} \left(< w, x_i > -y\right)^2$$

is a convex learning problem for reasons that will be discussed below.

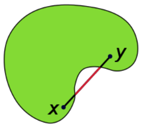
3. Convexity

Definition 6. A set C is convex if for any $u, v \in C$, the line segment between u and v is contained in C. Namely,

- for any $u, v \in C$ and for any $\alpha \in [0,1]$ we have that $\alpha u + (1-\alpha) v \in C$.



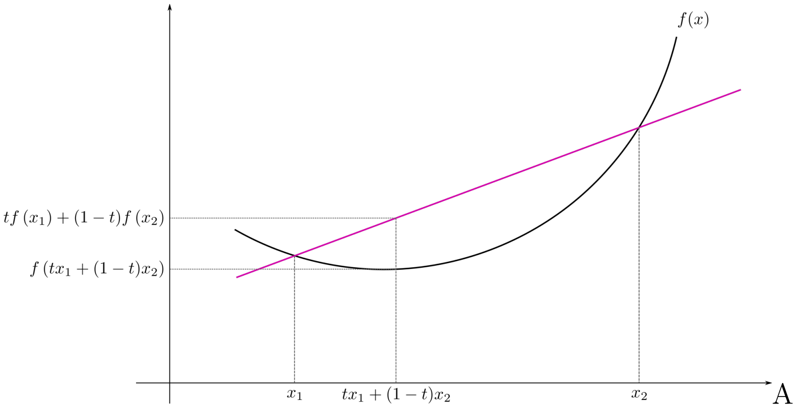
A convex set



A non-convex set

Definition 7. Let C be a convex set. A function $f : C \to R$ is convex function if for any $u, v \in C$ and for any $\alpha \in [0,1]$

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



A convex function

Example 8. 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

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

Proposition 9. *Every local minimum of a convex function is the global minimum.*

Proposition 10. *Let $f : C \to \mathbb{R}$ be convex function. The tangent of f at $w \in C$ is below f, namely*

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

Proposition 11. *Let $f : \mathbb{R}^d \to \mathbb{R}$ such that $f(w) = g(\langle w, x \rangle + y)$ for some $x \in \mathbb{R}^d$, $y \in \mathbb{R}$. If g is convex function then f is convex function.*

Proof. See Exercise 1 in the Exercise sheet. □

Example 12. Consider the regression problem $x \mapsto y$ with $x \in \mathbb{R}^d$, $y \in \mathbb{R}$ and predictor $h(x) = \langle w, x \rangle$. The loss function $\ell(w, (x,y)) = (\langle w, x \rangle + y)^2$ is convex because $g(a) = (a)^2$ is convex and Proposition 11.

Example 13. 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 14. $g(x) = |x|$ is convex according to Example 13, as $g(x) = |x| = \max(-x, x)$.

4. Lipschitzbness

Definition 15. 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

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

Conclusion 16. That means: a Lipschitz function $f(x)$ cannot change too drastically wrt x.

Example 17. 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 $C = \{x \in \mathbb{R} : |x| < \rho/2\}$.

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

Solution.

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

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

(2) It is

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

Theorem 18. *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 19. Let functions g be ρ-Lipschitz. Then f with $f(x) = g(< v, x > +b)$ is $(\rho\,|v|)$-Lipschitz.

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

Note 20. So, given Examples 17 and 19, in the linear regression setting using loss $\ell(w, z) = \left(w^\top x - z\right)^2$, the loss unction is -Lipschitz for a given z and and bounded $\|w\| < \rho$.

5. Smoothness

Definition 21. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is β-smooth if its gradient is β-Lipschitz; namely for all $v, w \in \mathbb{R}^d$

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

Theorem 22. *Function $f : \mathbb{R}^d \to \mathbb{R}$ is β-smooth iff*

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

Remark 23. If $f : \mathbb{R}^d \to \mathbb{R}$ is β-smooth then (5.2) holds, and if it is convex as well then

$$f(v) \geq f(w) + < \nabla f(w), v - w >$$

holds. Hence if both conditions imply upper and lower bounds

$$f(v) - f(w) \in \left(< \nabla f(w), v - w >, < \nabla f(w), v - w > +\frac{\beta}{2}\|v - w\|^2\right)$$

Remark 24. If $f : \mathbb{R}^d \to \mathbb{R}$ is β-smooth then for $v, w \in \mathbb{R}^d$ such that $v = w - \frac{1}{\beta}\nabla f(w)$ then by (5.2), it is

$$\frac{1}{2\beta}\|\nabla f(w)\|^2 \leq f(w) - f(v)$$

If additionally $f(x) > 0$ for all $x \in \mathbb{R}^d$ then

$$\|\nabla f(w)\|^2 \leq 2\beta f(w)$$

which provides assumptions to bound the gradient.

Theorem 25. *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.*

Solution. See Exercise 3 from the Exercise sheet

Example 26. 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 (25), f is $\left(2\left\|x\right\|^2\right)$-smooth.

6. Non-convex learning problems (surrogate treatment)

Remark 27. A a learning problem may involve non-convex loss function $\ell\left(w,z\right)$ which implies a non-convex risk function $R_g\left(w\right)$. 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 $\ell\left(w,z\right)$ by a convex surrogate loss function $\tilde{\ell}\left(w,z\right)$ for all w, and use $\tilde{\ell}\left(w,z\right)$ instead of $\ell\left(w,z\right)$.

Example 28. 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

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

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

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

which is convex (Example 14) wrt w. Note that:

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

Then we can compute

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

instead of

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

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

Remark 29. (Intuitions...) Using a convex surrogate loss function instead the convex one, fascilitates 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

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

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.

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

Handout 0: Learning problem: Definitions, notation, and formulation –A recap

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

1. 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) is called the procedure where the ML model is tuned. **Training data** (or observations, sample data set, exemplars) is a set of observables $\{z_i \in \mathcal{Z}\}$ used to tune the parameters of the ML model. **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 focus here) and the unsupervised learning.

Supervised learning problems involve applications where the training data $z \in \mathcal{Z}$ comprises examples of the input vectors $x \in \mathcal{X}$ along with their corresponding target vectors $y \in \mathcal{Y}$; i.e. $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. **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 recover 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 domain. The goal of learning is to discover a function which predicts $y \in \mathcal{Y}$ from $x \in \mathcal{X}$.

Unsupervised learning problems involve applications where the training data $z \in \mathcal{Z}$ consists 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 in learning

Definition 1. The learner's output is a function, $h : \mathcal{X} \rightarrow \mathcal{Y}$ 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. $h \in \mathcal{H}$.

Definition 3. Training data set $\mathcal{S}$ of size m is any finite sequence of pairs $((x_i, y_i); i = 1, ..., m)$ in $\mathcal{X} \times \mathcal{Y}$. This is the information that the learner has assess.

Definition 4. 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.

Definition 5. We denote as $\mathfrak{A}(\mathcal{S})$ the hypothesis (outcome) that a learning algorithm $\mathfrak{A}$ returns given training sample S.

Definition 6. (Loss function) Given any set of hypothesis $\mathcal{H}$ and some domain $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, a loss function $\ell(\cdot)$ is any function $\ell : \mathcal{H} \times \mathcal{Z} \rightarrow \mathbb{R}_+$. The purpose of loss function $\ell(h, z)$ is to quantify the "error" for a given hypothesis h and example z –the greater the error the greater its value of the loss.

Example 7. In binary classification problems where $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ and $\mathcal{Y} = \{0, 1\}$ is discrete, a loss function can be

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

Example 8. In regression problems $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ and $\mathcal{Y}$ is uncountable, a loss function can be

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

Definition 9. (Risk function) The risk function $R_g(h)$ of h is the expected loss of the hypothesis $h \in \mathcal{H}$, w.r.t. probability distribution g over domain Z; i.e.

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

Remark 10. In learning, an ideal way to obtain an optimal predictor h^* is to compute the risk minimizer

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

Example 11. (Cont. Ex. 8)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 $h : \mathcal{X} \rightarrow \mathcal{Y}$ as the expected square difference between the predicted values h and the true target values y at every x.

Remark 12. 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). Suboptimally, one may resort to the Empirical risk function.

Definition 13. (Empirical risk function) The empirical risk function $\hat{R}_S(h)$ of h is the expectation of loss of h over a given sample $S = (z_1, ..., z_m) \in \mathcal{Z}^m$; i.e.

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

Example 14. (Cont. Example 11) 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 15. 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 $f(y|\theta)$ where the parameter θ is unknown. 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

$$R_g(\theta) = \mathrm{E}_{z\sim g}\left(\log\left(\frac{g(z)}{f_\theta(z)}\right)\right) = \mathrm{E}_{z\sim g}(\log(g(z))) - \mathrm{E}_{z\sim g}(\log(f_\theta(z))) \tag{2.2}$$

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 (2.2) is constant. Note that in the Maximum Likelihood Estimation technique the MLE θ_{MLE} is the minimizer of

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

where $S = \{y_1, ..., y_m\}$ is an IID sample from g. Hence, MLE θ_{MLE} can be considered as the minimizer of the empirical risk $R_S(\theta) = \frac{1}{m}\sum_{i=1}^{m}(-\log(f_\theta(z_i)))$.

Example 16. (Linear Regression) Consider the multiple linear regression problem $x \longmapsto y$ where $x \in \mathcal{X} \subseteq \mathbb{R}^d$ and $y \in \mathcal{Y} \subseteq \mathbb{R}^d$.

- The hypothesis is a linear function $h : \mathcal{X} \to \mathcal{Y}$ (that learner wishes to learn) to approximate mapping $x \longmapsto y$. The hypothesis set $\mathcal{H} = \left\{< w, x > \longmapsto y : w \in \mathbb{R}^d\right\}$. We can use the loss $\ell(h, (x, y)) = (h(x) - y)^2$.
- Equivalently, learning problem can be set differently because the predictor (linear function) is parametrized by $w \in \mathbb{R}^d$ as $< w, x > \longmapsto y$. Set $\mathcal{H} = \left\{w \in \mathbb{R}^d\right\}$. The set of examples is $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. The loss is $\ell(w, (x, y)) = (< w, x > -y)^2$.

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

Handout 2: 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:

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

Further reading

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

1. Motivations

Note 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) $\hat{R}\left(h\right)=\frac{1}{n}\sum_{i=1}^{n}\ell\left(h,z_{i}\right)$ given a finite sample $\{z_{i};i=1,...,n\}$ generated from the data generating model $g(\cdot)$ and using loss $\ell\left(\cdot\right)$; that is

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

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

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

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

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

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

Definition 4. Given a per-specified learning rate $\eta_{t}>0$, the Gradient Descent (GD) algorithm for the solution of the minimization problem (2.1) is given in Algorithm 1

```
Algorithm 1 Gradient descent algorithm with learning rate $\eta_t$
For $t = 1, 2, 3, ...$ iterate:
    (1) compute
(2.2)        $w^{(t+1)} = w^{(t)} - \eta_t \nabla f\left(w^{(t)}\right)$
    (2) terminate if a termination criterion is satisfied, e.g.
                If $t \geq T_{\max}$ then STOP
```

where

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

is the gradient of f at w.

Remark 5. Interpreting GD, GD produces a chain $\left\{w^{(t)}\right\}$ that drifts towards the 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.

Remark 6. For more intuitive explanation, 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

See Handout 1

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

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).

Remark 7. Given T GD algorithm iterations, 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 8. GD output converges to a local minimum, $w_{\text{GD}}^{(T)} \rightarrow w_*$ (in some sense), under different sets of regularity conditions (some are weaker other stronger). Section 4 has a brief analysis.

Remark 9. The parameter η_t is called learning rate, step size, or 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 8) often imply restrictions on the decay of $\{\eta_t\}$ which guide the practitioner to parametrize it properly. Some popular choices of learning rate η_t are:

(1) <u>constant</u>; $\eta_t = \eta$, for where $\eta > 0$ is a small value. The rationale is that GD chain $\{w_t\}$ performs constant small steps towards the (local) minimum w_* and then oscillate around it.

(2) <u>decreasing and converging to zero</u>; $\eta_t \searrow$ with $\lim_{t\rightarrow\infty} \eta_t = 0$. 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 begining to explore the area for discovering possible minima. Also it reduces the size of those steps with the iterations (controled by ς) such that eventually when the chain $\{w_t\}$ is close to a possible minimum w_* value to converge and do not overshoot.

(3) <u>decreasing and converging to a tiny value</u> τ_*; $\eta_t \searrow$ with $\lim_{t\rightarrow\infty} \eta_t = \tau_*$ E.g. $\eta_t = \left(\frac{C}{t}\right)^{\varsigma} + \tau_*$ with $\varsigma \in (0.5, 1]$, $C > 0$, and $\tau_* \approx 0$. Same as previously, but the algorithm aims at oscillating around the detected local minimum.

(4) <u>constant until an iteration T_0 and then decreasing</u>; Eg $\eta_t = \left(\frac{C}{\max(t, T_0)}\right)^{\varsigma}$ with $\varsigma \in [0.5, 1]$ and $C > 0$, and $T_0 < T$. The rationale is that at the first stage of the iterations (when $t \leq T_0$) the algorithm may need a constant large steps for a significant number of iterations T_0 in order to explore the domain; and hence in order for the chain $\{w_t\}$ to reach the area around the (local) minimum w_*. In the second stage, hoping that the chain $\{w_t\}$ may be in close proximity to the (local) minimum w_* the algorithm progressively performs smaller steps to converge towards the minimum w_*. The first stage ($t \leq T_0$) is called burn-in; the values $\{w_t\}$ produced during the burn-in ($t \leq T_0$) are are often discarded/ignored from the output of the GD algorithm.

- Parameters $C, \varsigma, \tau_*, T_0$ may be chosen based on pilot runs.

Remark 10. There are several practical termination criteria that can be used in GD Algorithm 1(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.

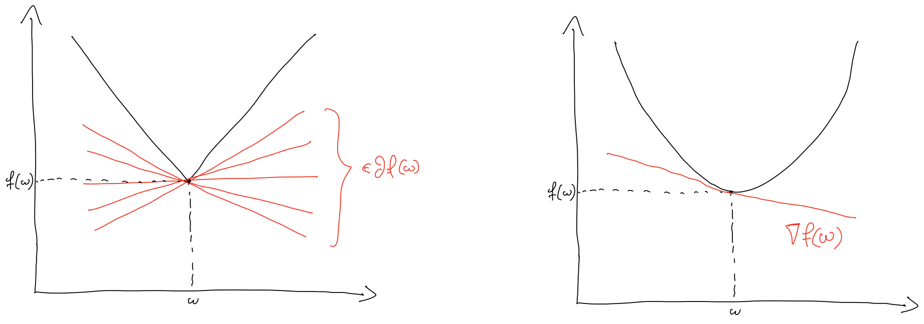
3. GD for non-differentiable functions (using sub-gradients)

Note 11. 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 12. 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 13. 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

Notation 14. The set of subgradients of function $f : S \to \mathbb{R}$ at $w \in S$ is denoted by $\partial f(w)$.

Definition 15. 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

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

3.1. **Construction of subgradient.**

Note 16. We discuss how to construct subgradients in practice.

Fact 17. *Some properties of subgradient sets that help for their construction*

1. *If function $f : S \rightarrow \mathbb{R}$ 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*

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

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

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

Example 18. Consider the function $f : \mathbb{R} \rightarrow \mathbb{R}_{+}$ 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 17, 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

Assumption 19. *For the sake of the analysis of the GD, let us consider:*

1. *constant learning rate $\eta_t = \eta$,*
2. *GD output $w_{GD}^{(T)} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$*

Also as we will see, function $f(\cdot)$ should be convex and Lipschitz fin order for the following results to hold

Notation 20. w^{*} is the minimizer in (2.1).

Note 21. We consider Lemma 22 as given Fact.

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 2nd reference. □

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

Proposition 24. *Consider the minimization problem (2.1). Given Assumptions 19, the error can be bounded as*

$$f\left(w_{GD}^{(T)}\right)-f\left(w^{*}\right)\leq\frac{1}{T}\sum_{t=1}^{T}\langle w^{(t)}-w^{*},v_{t}\rangle \tag{4.2}$$

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

Proof. It is[1]

$$\begin{aligned}
f\left(w_{\text{GD}}^{(T)}\right)-f\left(w^{*}\right)= & f\left(\frac{1}{T}\sum_{t=1}^{T}w_{t}\right)-f\left(w^{*}\right)\\
\leq & \frac{1}{T}\sum_{t=1}^{T}\left(f\left(w_{t}\right)-f\left(w^{*}\right)\right) && \text{(by Jensen's inequality)} \qquad (4.3)\\
\leq & \frac{1}{T}\sum_{t=1}^{T}\langle w^{(t)}-w^{*},\nabla f\left(w^{(t)}\right)\rangle && \text{(by convexity of } f\left(\cdot\right))
\end{aligned}$$

□

Note 25. The following provides conditions under which the gradient (or sub-gradient) is bounded. This is necessary in order to bound (4.2) with (4.1) in a meaningful manner.

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

Proof. $\Longrightarrow$ Let $f:S\rightarrow\mathbb{R}$ 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 $u:=w+\epsilon\frac{v}{\left\Vert v\right\Vert }$ where $u\in S$. So $\langle u-w,v\rangle=\epsilon\left\Vert v\right\Vert$ and $\left\Vert u-w\right\Vert =\epsilon$.
- From the subgradient definition we get

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

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

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

Therefore $\left\Vert v\right\Vert \leq\rho$.

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

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

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

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

Note 27. The following summaries Lemma 22, and Propositions 24, 26 with respect to the GD algorithm.

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

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

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

Solution. Straightforward from Lemma 22, and Propositions 24, 26.

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

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

$$f\left(w_{\text{GD}}^{(T)}\right) - f(w^{*}) \leq \frac{B\rho}{\sqrt{T}} \tag{4.4}$$

(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).

Remark 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

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

[3] Code is available in `https://github.com/georgios-stats/Machine_Learning_and_Neural_Networks_III_Epiphany_2023/tree/main/Lecture_handouts/code/02.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

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

The GD Algorithm 1 with learning rate η is

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

(1) compute

$$w^{(t+1)} = w^{(t)} - \eta v_t, \tag{5.3}$$

where $v_t = \begin{pmatrix} 2w_1^{(t)} + 2w_2^{(t)}\bar{x} - 2\bar{y} \\ 2w_1^{(t)}\bar{x} + 2w_2^{(t)}\overline{x^2} - 2y^\top x \end{pmatrix}$

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

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

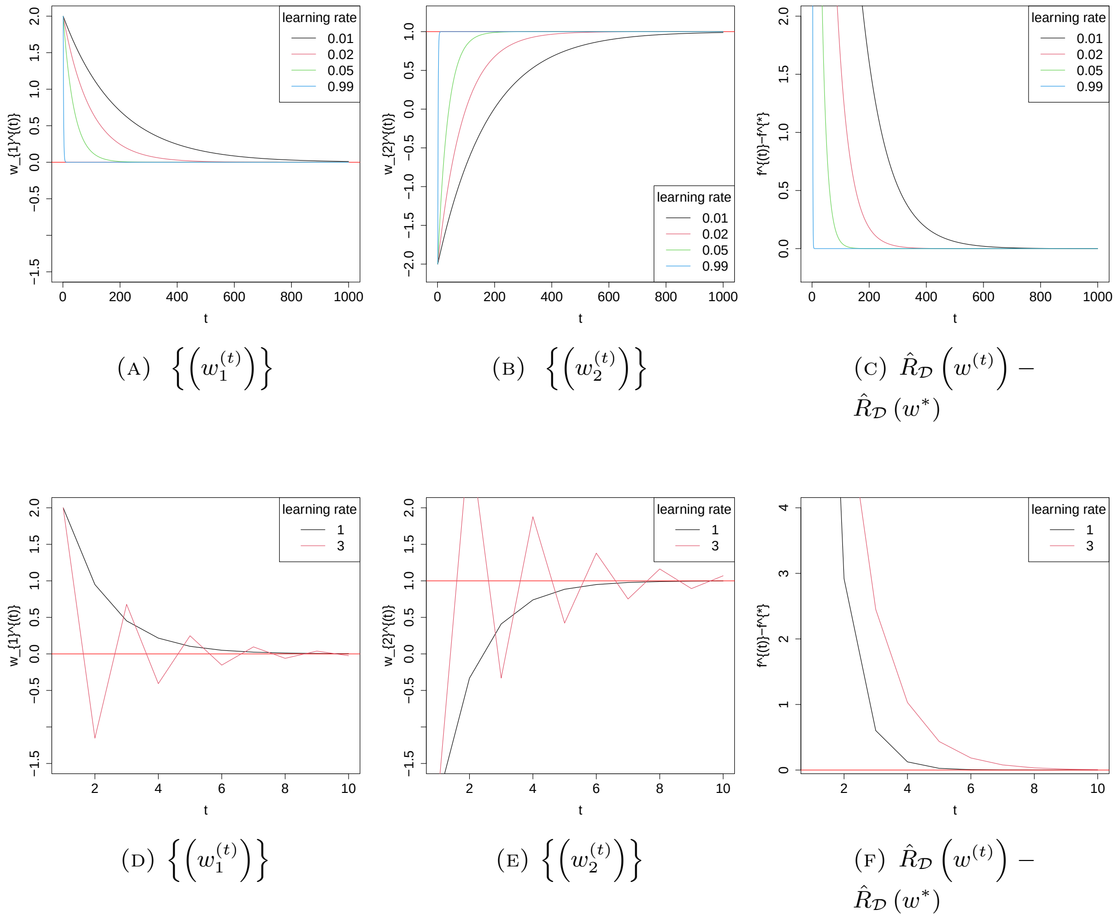
This is because $\hat{R}_{\mathcal{D}}(w)$ is differentiable in $\mathbb{R}^2$ so $\partial \hat{R}_{\mathcal{D}}(w) = \left\{\nabla \hat{R}_{\mathcal{D}}(w)\right\}$ and because

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

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 5.1a and 5.1b.



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

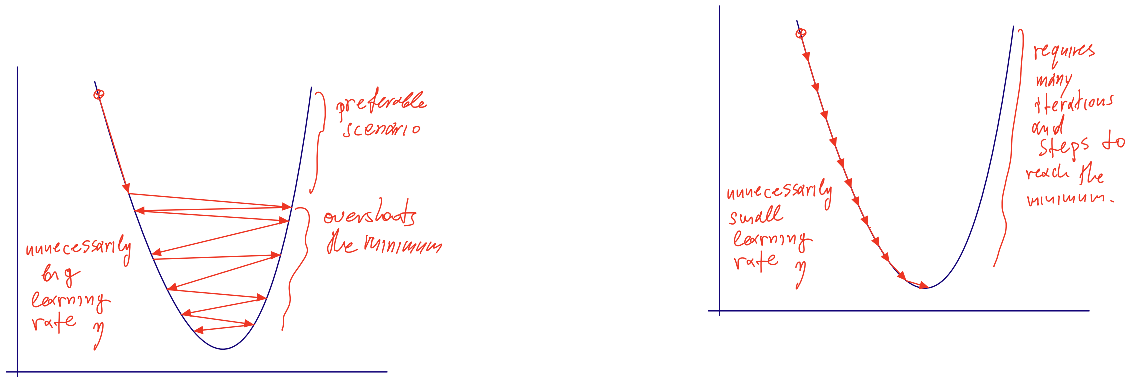
(B) $\left\{\left(w_2^{(t)}\right)\right\}$

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


preferable scenario
overshoots the minimum
unnecessarily big learning rate η
requires many iterations and steps to reach the minimum.
unnecessarily small learning rate η

(A) Unnecessarily large learning rate

(B) Unnecessarily small learning rate

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

Handout 3: 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:

- 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

Further reading

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

1. Motivations for stochastic gradient descent

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

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

Remark 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 $\hat{R}\left(w\right)=\frac{1}{n}\sum_{i=1}^{n}\ell\left(w,z_{i}\right)$ which ideally is used as a proxy when data size n is big (big-data).

Remark 3. The implementation of GD may be computationally impractical even in problems where we need to minimize an ERF $\hat{R}_{n}\left(w\right)$ if we have big data ($n\approx$big). This is because GD requires the recursive computation of the exact gradient $\nabla\hat{R}_{n}\left(w\right)=\frac{1}{n}\sum_{i=1}^{n}\nabla\ell\left(w,z_{i}\right)$ using all the data $\{z_{i}\}$ at each iteration. That may be too slow.

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

2. Stochastic gradient descent

2.1. Description.

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

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

where here $f:\mathbb{R}^{d}\rightarrow\mathbb{R}$, 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 (2.1)*

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.*

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

Remark 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.

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

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

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

(2) or the best value discovered

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

(3) or the last value discovered

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

Note 10. SGD output converges to a local minimum, $w_{\text{SGD}}^{(T)} \rightarrow w_*$ (in some sense), under different sets of regularity conditions. Section 4 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) $\sum_{t=1}^{\infty} \eta_t^2 < \infty$

Remark 12. The popular learning rates $\{\eta_t\}$ in Remark 9 in Handout 2 satisfy Condition 11 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.

Remark 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 $t \rightarrow \infty$; if this was not the case then

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

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. Stochastic gradient descent with projection

Remark 14. 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 $w_{\text{SGD}} \notin \mathcal{H}$. SGD can be modified to address this issue, as in Algorithm 15, by including a projection step guarantying $w \in \mathcal{H}$.

Algorithm 15. *Stochastic Gradient Descent with learning rate $\eta_t > 0$ and with projection in $\mathcal{H}$ for problem in (2.1)*

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

(1) *compute*

$$w^{\left(t+\frac{1}{2}\right)} = w^{(t)} - \eta_t v_t, \tag{3.1}$$

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*

$$w^{(t+1)} = \arg\min_{w \in \mathcal{H}} \left(\left\| w - w^{\left(t+\frac{1}{2}\right)} \right\| \right) \tag{3.2}$$

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

4. Analysis of SGD (Algorithm 6)

Note 16. Recall that the stochasticity of SGD comes from the stochastic sub-gradients $\{v_t\}_{\forall t}$; hence the expectations below are under these random vectors' distributions.

Theorem 17. *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

$$\mathrm{E}\left(f\left(w_{\text{SGD}}^{(T)}\right)\right) - f(w^*) \le \frac{\|w^*\|^2}{2\eta T} + \frac{\eta}{2}\frac{1}{T}\sum_{t=1}^{T} \mathrm{E}\,\|v_t\|^2 \tag{4.1}$$

Proof. Let $v_{1:t} = (v_1, ..., v_t)$. By Jensens' inequality (or see (4.3) in Handout 2)

$$\mathrm{E}\left(f\left(w_{\text{SGD}}^{(T)}\right) - f(w^*)\right) \le \mathrm{E}\left(\frac{1}{T}\sum_{t=1}^{T}\left(f\left(w^{(t)}\right) - f(w^*)\right)\right) = \frac{1}{T}\sum_{t=1}^{T}\mathrm{E}\left(f\left(w^{(t)}\right) - f(w^*)\right) \tag{4.2}$$

I will try to use Lemma 22 from Handout 2, hence I need to show

$$\mathrm{E}\left(f\left(w^{(t)}\right) - f(w^*)\right) \le \mathrm{E}\left(\langle w^{(t)} - w^*, v_t\rangle\right) \tag{4.3}$$

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

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

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

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

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

$$\begin{aligned}\mathrm{E}_{v_{1:t-1}}\left(\langle w^{(t)}-w^*,\mathrm{E}_{v_{1:t}}\left(v_t|v_{1:t-1}\right)\rangle\right)\geq&\mathrm{E}_{v_{1:t-1}}\left(f\left(w^{(t)}\right)-f\left(w^*\right)\right)\\ =&\mathrm{E}_{v_{1:T}}\left(f\left(w^{(t)}\right)-f\left(w^*\right)\right)\end{aligned}\tag{4.4}$$

Hence combining (4.4), (4.3), and (4.2)

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

Then Lemma 22 in Handout 2 implies

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

□

Remark 18. The upper bound in (4.1) depends on the variation of v_t as

$$\mathrm{E}\|v_t\|^2=\sum_{j=1}^{d}\mathrm{Var}\left(v_{t,j}\right)+\sum_{j=1}^{d}\left(\mathrm{E}\left(v_{t,j}\right)\right)^2\tag{4.5}$$

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 (4.5) is constant as $v_{t,j}$ is the unbiased estimator of the sub-gradient by construction.

Proposition 19. *(Cont. Theorem 17) 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 $\eta_t=\sqrt{\frac{B^2}{\rho^2 T}}$ for T steps, and output $w_{\mathrm{SGD}}^{(T)}=\frac{1}{T}\sum_{t=1}^{T}w^{(t)}$. Then*

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

$$\mathrm{E}\left(f\left(w_{\mathrm{SGD}}^{(T)}\right)\right)-f\left(w^*\right)\leq\frac{B\rho}{\sqrt{T}}\tag{4.6}$$

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

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

Proof. It follows from 17. □

Remark 20. Lemma 22 in Handout 2 holds even when projection steps are added in the vanilla SGD algorithm, hence the above analysis holds for the SGD with projections (Algorithm 15) too.

5. Implementation of SGD in the learning Problem 1

Note 21. Below we show SGD can be implement in the learning Problem 1. Proposition 22 allows a convenient implementation of SGD to the learning problem (Problem 1).

Proposition 22. *For a randomly drawn $z \sim g(\cdot)$, 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

$$\mathrm{E}_{z\sim g}(v|w) = \nabla_w R_g(w)$$

Note 24. Below we show how SGD can be implement in the learning Problem 1.

Note 25. 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 sampling (Algorithm 26) is an implementation of the SGD (Algorithm 6) in the learning Problem 1.

Algorithm 26. *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*

Remark 27. Step 1 can be presented 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 $\tilde{v}_{t,j} \in \partial_w \ell\left(w^{(t)}, z_j\right)$.

Remark 28. A projection step, such as (3.2) can be added right after step 2 in Algorithm 26, if needed.

Remark 29. 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 26 can become

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

Definition 30. Online Stochastic gradient descent is the special case of Algorithm 26 using subsamples of size one ($m = 1$), namely, only one example is randomly chosen in Step 1 of Algorithm 26.

Remark 31. In theory, using larger batch size m has the benefit that reduces the variance $\text{Var}\left(v_t | w^{(t)}\right)$ of v_t at iteration t due to averaging effect, stabilizes the SGD algorithm, and reduces the error bound (4.1); see Remark 18.

Remark 32. 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 in a scenario with big-data: 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.

Remark 33. 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.

6. Stochastic Variance Reduced Gradient (SVRG)

Remark 34. Recall the upper bound of the error (4.1) in SGD depends on the variance of the stochastic gradient, as shown in Remark 4.5. Hence the algorithm may be improved by reducing the variance of each element of v_t.

Remark 35. Control variate is a general way to perform variance reduction. Let random variables $v \in \mathbb{R}$, and $y \in \mathbb{R}$. Let $z = v + c\,(y - \text{E}\,(y))$ for some constant $c \in \mathbb{R}$. It is $\text{E}_c\,(z) = \text{E}\,(v)$ and

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

which is minimized for

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

hence

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

Note 36. For simplicity, SVRG is presented in the case where $\kappa = 1$ and the loss function $\ell\,(w, z)$ is differentiable wrt w hence its gradient exists. However it is applicable to the more general cases introduced.

Remark 37. Every m 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

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

given that a random $\tilde{z}^{(t)}$ has been collected from the sample. The symbols below the brackets are given with reference to Remark 35.

Algorithm 38. *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 39. Iterations of SVRG are computationally faster than those of full GD, but SVRG can still match the theoretical convergence rate of GD.

Remark 40. How often we get snapshots, κ, in Algorithm 38 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.

7. Preconditioned SGD; The AdaGrad algorithm

Remark 41. 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 42. To address this, SGD (Algorithm 6) can be modified in the update step as in Algorithm 43 by using a preconditioner P_t that accounts for the curvature (or geometry in general of f).

Algorithm 43. *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
        (7.1)    $w^{(t+1)} = w^{(t)} - \eta_t P_t v_t,$
        where $v_t$ is a random vector such that $E\left(v_t | w^{(t)}\right) \in \partial f\left(w^{(t)}\right)$, and $P_t$ a preconditioner
    (2) terminate if a termination criterion is satisfied, e.g.
        If $t \geq T$ then STOP
```

Remark 44. A natural choice of P_t can be $P_t := [H_t + \epsilon I_d]^{-1}$, where H_t is the Hessian matrix $[H_t]_{i,j} = \frac{\partial^2}{\partial w_i \partial w_j} f(w)\Big|_{w=w^{(t)}}$ (ie. the gradient of the gradient's elements), and ϵ is a tiny $\epsilon > 0$ to mitigate machine error when Hessian elements are close to zero.

Remark 45. 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 46. AdaGrad dynamically incorporates knowledge of the geometry of function $f(\cdot)$ (to be minimized) in earlier iterations to perform more informative gradient-based learning.

Remark 47. 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 48. Hence, this strategy often improves convergence performance over standard stochastic gradient descent in settings where data is sparse and sparse features w's are more informative.

Definition 49. Adaptive Stochastic Gradient Decent (AdaGrad) can be presented in terms of preconditioned SGD (Algorithm 43) with preconditioner $P_t = [I_d \text{diag}(G_t) I_d + \epsilon I_d]^{-1/2}$ in (7.1)

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

where $\text{diag}(A) := (A_{1,1}, A_{2,2}, ..., A_{d,d})^\top$ is the vector of the diagonal of a matrix A, $G_t = \sum_{\tau=1}^{t} v_\tau^\top v_\tau$ 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 50. 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) implies that the j-th dimesion 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 51. 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.

8. Example[1]

Note 52. We continue Example in Section 5 in Handout 2. Recall, we considered a hypothesis space $\mathcal{H}$ of linear functions $h : \mathbb{R}^2 \to \mathbb{R}$ with $h(w) = w_1 + w_2 x$, $w = (w_1, w_2)^\top$, and $\ell\left(w, z = (x, y)^\top\right) = (y_i - w_1 - w_2 x)^2$. Here we consider a big dataset $\mathcal{S}_n = \{z_1, ..., z_n\}$ with $n = 10^6$ examples.

Example 53. The batch SGD algorithm (Algorithm 26) 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 8.1a & 8.1d, 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 54. 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 $\tilde{\mathcal{S}}_1 = \{z_{j^*}\}$

[1] Code can be found in `https://github.com/georgios-stats/Machine_Learning_and_Neural_Networks_III_Epiphany_2023/tree/main/Lecture_handouts/code/03.Stochastic_gradient_descent`

(2) compute

$$w^{(t+1)} = \begin{bmatrix} w_1^{(t)} \\ w_2^{(t)} \end{bmatrix} - \eta_t \left(\begin{bmatrix} 2\left(w_1^{(t)} - \tilde{w}_1^{(t)}\right) + 2w_2^{(t)} x_{j^*} \\ 2\left(w_2^{(t)} - \tilde{w}_2^{(t)}\right)\bar{x} + 2\left(w_2^{(t)}\left(x_{j^*}\right)^2 - \tilde{w}_2^{(t)}\left(x_{j^*}\right)^2\right) \end{bmatrix} + \nabla \hat{R}_{\mathcal{D}}\left(\tilde{w}\right) \right) \tag{8.1}$$

(3) if modulo $(t, \kappa) = 0$,

(a) set $\tilde{w} = w^{(t)}$

(b) compute

$$\frac{1}{n}\sum_{i=1}^{n} \ell\left(\tilde{w}, z_i\right) = \begin{pmatrix} 2\tilde{w}_1^{(t)} + 2\tilde{w}_2^{(t)}\bar{x} - 2\bar{y} \\ 2\tilde{w}_1^{(t)}\bar{x} + 2\tilde{w}_2^{(t)}\overline{x^2} - 2y^{\top}x \end{pmatrix} = \nabla \hat{R}_{\mathcal{D}}\left(\tilde{w}\right) \tag{8.2}$$

(4) if a termination criterion is satisfied STOP

Because (8.2) is actually the gradient of the Risk function at $\tilde{w}$ and

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

In Figure 8.1c we observe the frequency of the snapshots has improved the convergence; however this is not a panacea as seen in Figure 8.1f.

Example 55. 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 $\left\{1, ..., n = 10^6\right\}$, and set $\tilde{\mathcal{S}}_1 = \{z_{j^*}\}$

(2) compute

$$\begin{aligned} v_t =& \begin{pmatrix} 2w_1^{(t)} + 2w_2^{(t)} x_{j^*} - 2y_{j^*} \\ 2w_1^{(t)}\bar{x} + 2w_2^{(t)} x_{j^*}^2 - 2y_{j^*}x_{j^*} \end{pmatrix} \\ G_t =& G_{t-1} + \begin{pmatrix} v_{t,1}^2 \\ v_{t,2}^2 \end{pmatrix} \\ w^{(t+1)} =& \begin{pmatrix} w_1^{(t)} - \frac{\eta_t}{\sqrt{G_{t,1}+\epsilon}} v_{t,1} \\ w_2^{(t)} - \frac{\eta_t}{\sqrt{G_{t,2}+\epsilon}} v_{t,2} \end{pmatrix}, \end{aligned}$$

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

because

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

In Figures 8.1g & 8.1h, 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.

Example 56. Consider a (rather naive) loss function $\ell\left(w, z = (x, y)\right) = -\cos\left(0.5\left(y - w_1 - w_2 x\right)\right)$, 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

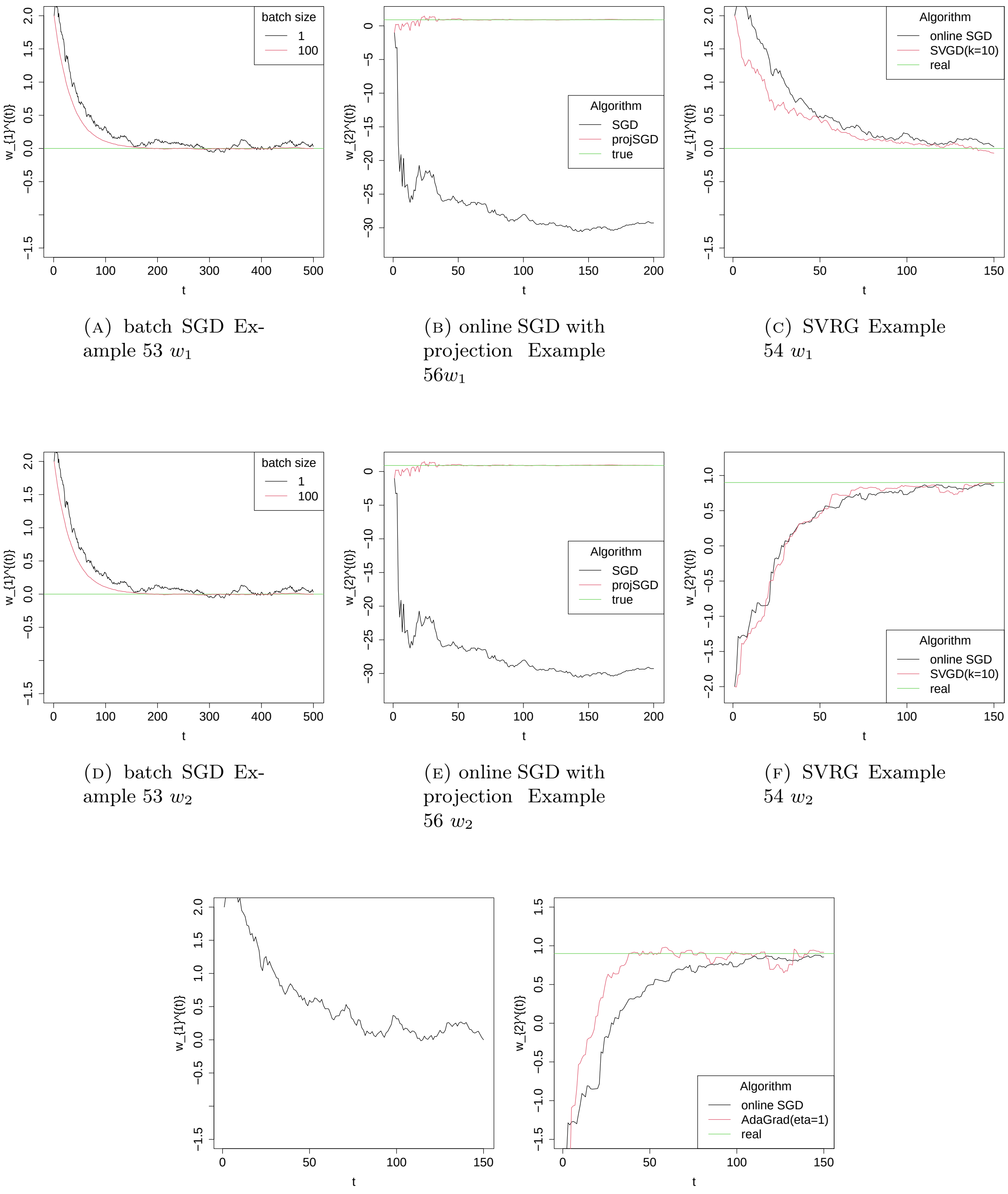
$$\begin{aligned} w^{(t+1/2)} =& w^{(t)} - \frac{50}{t} \begin{pmatrix} \sin\left(y_{j^*} - w_1^{(t)} - w_2^{(t)} x_{j^*}\right) \\ \sin\left(y_{j^*} - w_1^{(t)} - w_2^{(t)} x_{j^*}\right) x_{j^*} \end{pmatrix} \\ w^{(t+1/2)} =& \arg \min_{\forall w: \|w\| \leq 1.5} \left(\left\| w - w^{(t+1/2)} \right\| \right) \end{aligned}$$

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

  because

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

  In Figures 8.1b & 8.1e, 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.



(a) batch SGD Example 53 w_1

(b) online SGD with projection Example $56w_1$

(c) SVRG Example 54 w_1

(d) batch SGD Example 53 w_2

(e) online SGD with projection Example 56 w_2

(f) SVRG Example 54 w_2

(g) online AdaGrad Example $55w_1$

(h) online AdaGrad Example $55w_2$

Figure 8.1. Simulations of the Examples

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

Handout 4: 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:

- 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).
- Bottou, L. (2012). Stochastic gradient descent tricks. In Neural networks: Tricks of the trade (pp. 421-436). Springer, Berlin, Heidelberg.
- 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.
- Teh, Y. W., Thiery, A. H., & Vollmer, S. J. (2016). Consistency and fluctuations for stochastic gradient Langevin dynamics. Journal of Machine Learning Research, 17.
- 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
- 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

Remark 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

$$\begin{cases} z_{1:n}|w & \sim f\left(z_{1:n}|w\right) \\ w & \sim f\left(w\right) \end{cases} \tag{1.1}$$

Remark 3. 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)

Remark 4. 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)

Remark 5. 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.

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

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

Note that (I) may be interpreted as an empirical rist 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 6. 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 7. 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 8. Given the Bayesian model (1.1), and rearranging (1.4), MAP estimate w^* of w can be computed as

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

Remark 9. GD is particularly suitable to solve (2.1) when w has high dimensionality.

Algorithm 10. *Gradient descent (Algorithm 1 Handout 2) with learning rate $\eta_t \geq 0$ can be used to solve (2.1) by using the update rule as*

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

(1) *Compute*

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

Remark 11. The implementation of other GD variants (eg (3.2) in Handout 2) is straightforward, based on Algorithm 10.

Remark 12. SGD is particularly suitable to solve (2.1) when w has high dimensionality, and in big-data problems since the repetitive computation of the sum in (2.2) is prohibitively expensive. Yet consider the benefits of SGD against GD as discussed in Remarks 32 and 33 of Handout 3.

Algorithm 13. *Batch Stochastic Gradient Descent (Algorithm 26 in Handout 3) with learning rate $\eta_t \geq 0$ and batch size m can be used to solve (2.1) 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*

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

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

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

Remark 15. The implementation of the SGD variants (Algorithms 26, 38, 43, 49 in Handout 3)) is straightforward based on Algorithm 13 and (2.4).

3. Fully Bayesian learning via SGLD

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

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

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

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

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

Remark 18. Stochastic gradient Langevin dynamics (SGLD) algorithm is able to approximately produce samples from the posterior distribution (1.2) of parameters w given the available data $z_{1:n}$. That allows to recover the whole posterior distribution (1.2) (hence account for the uncertainty in parameters) and approximate posterior expectations (1.3) by averaging out (3.1) according to the Monte Carlo integration (Remark 17).

Remark 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 (hence account for uncertainty in parameters) and approximate posterior expectations based on the Monte Carlo integration (Remark 17).

Note 20. SGLD relies on injecting the 'right' amount of noise 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 the true posterior distribution.

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*

$$w^{(t+1)} = w^{(t)} + \eta_t \left(\frac{n}{m} \sum_{i \in J^{(t)}} \nabla \log f \left(z_i | w \right) + \nabla \log f \left(w \right) \right) + \sqrt{\eta_t} \sqrt{\tau} \epsilon_t \tag{3.2}$$

where $\epsilon_t \overset{\text{iid}}{\sim} \mathrm{N}\left(0, 1\right)$.

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

Remark 22. The first few iterations from 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.

Remark 23. 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 (aka the end tail of the generated chain).

Remark 24. SGLD (Algorithm 21) generates as output a random chain $\left\{w^{(t)}\right\}$ that is approximately distributed according to a distribution with density such as

$$f_{\tau}\left(w|z_{1:n}\right) \propto \exp\left(\frac{1}{\tau}\prod_{i=1}^{n} f\left(z_i|w\right) f\left(w\right)\right) \tag{3.3}$$

$$\propto \exp\left(\frac{1}{\tau} L_n\left(w\right) f\left(w\right)\right) \tag{3.4}$$

under regularity conditions. Conditions 25 on the learning rate are rather inevitable and should be satisfied.

Condition 25. 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) $\sum_{t=1}^{\infty} \eta_t^2 < \infty$

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

Remark 27. SGLD for $\tau = 1$ approximately simulates from the posterior (1.2).

Remark 28. The popular learning rates $\{\eta_t\}$ in Remark 9 in Handout 2 satisfy Condition 25 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.

Remark 29. (Mathematically speaking) The stochastic chain in (3.2) can be viewed as a discretization of the continuous-time Langevin diffusion described by the stochastic differential equation

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

where $\{B\left(t\right)\}$ 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

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

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

Remark 30. (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 10 -but this is until η_t or $\nabla \log\left(L_n\left(w\right)\right)$ 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

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

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.

Remark 31. 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 (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.

Remark 32. 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.

Remark 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.

Remark 34. Another more efficient estimator for the expectation (1.3) is the weighted arithmetic average

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

Because 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) will overemphasize the tail end of the sequence where there is higher correlation among the samples resulting in higher variance in the estimator.

Remark 35. 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 3. Hence the update (3.2) becomes

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

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

Remark 36. '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. Practical solutions 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., truncating it) if it is gonna result an overflow.

To be explai
the computer
tical and Lec

Gradient scaling: involves normalizing the gradient vector such that vector norm (magnitude) equals a defined value. More formally, given any gradient v on an example, 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 is a clipping threshold impaling that clipping will take place at iteration t if $\|v_t\| > c$.

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

Beware that unreasonable clipping may introduce significant bias and hence it should not be applied unnecessarily.

4. Examples [2]

We continue the Example 33 in Handout 1, and Example 8 in Handout 2. Consider the Bayesian Normal linear regression model

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

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

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

Because SGD (Algorithm 13) 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

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

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

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

[2] Code is available from `https://github.com/georgios-stats/Machine_Learning_and_Neural_Networks_III_Epiphany_2023/tree/main/Lecture_handouts/code/04.Stochastic_gradient_Langevine_dynamics`

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

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

Hence for the log sampling PDF we have

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

...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

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

...so

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

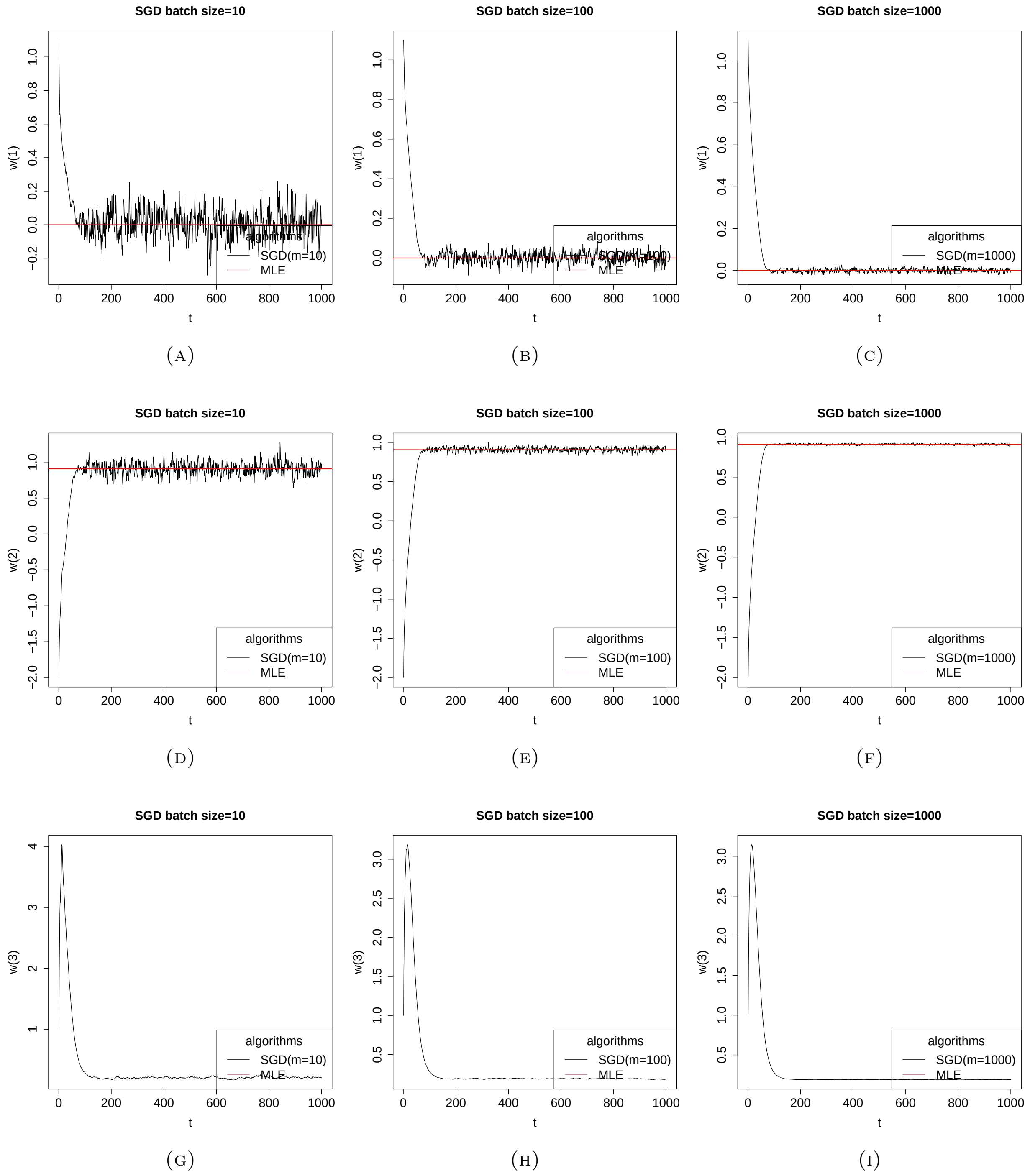
To implement SGD (Algorithm 13) and SGLD (Algorithm 21) , we just need to plug in the computed gradients (4.1) and (4.2) for $w=(\beta,\gamma)\in\mathbb{R}^d\times\mathbb{R}$ with $\gamma=\log\left(\sigma^2\right)$. After running SGD and SGLD with the computed gradients with respect to $w=(\beta,\gamma)\in\mathbb{R}^d\times\mathbb{R}$ 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 $\left(\beta,\sigma^2\right)$.

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

- 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

$\{w^{(t)}\}$ 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 $\tau = 1$.

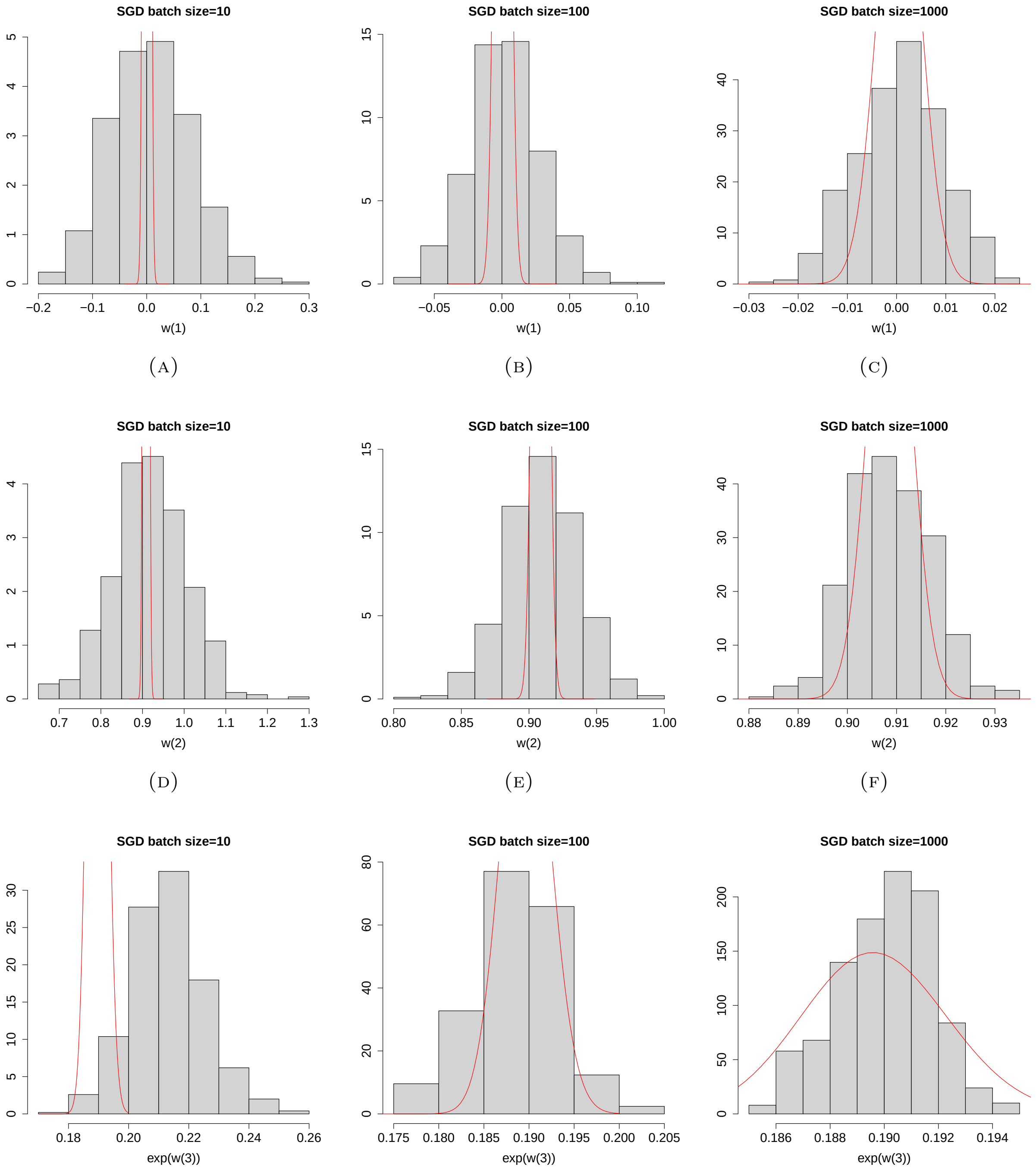


(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

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



(A)

(B)

(C)

(D)

(E)

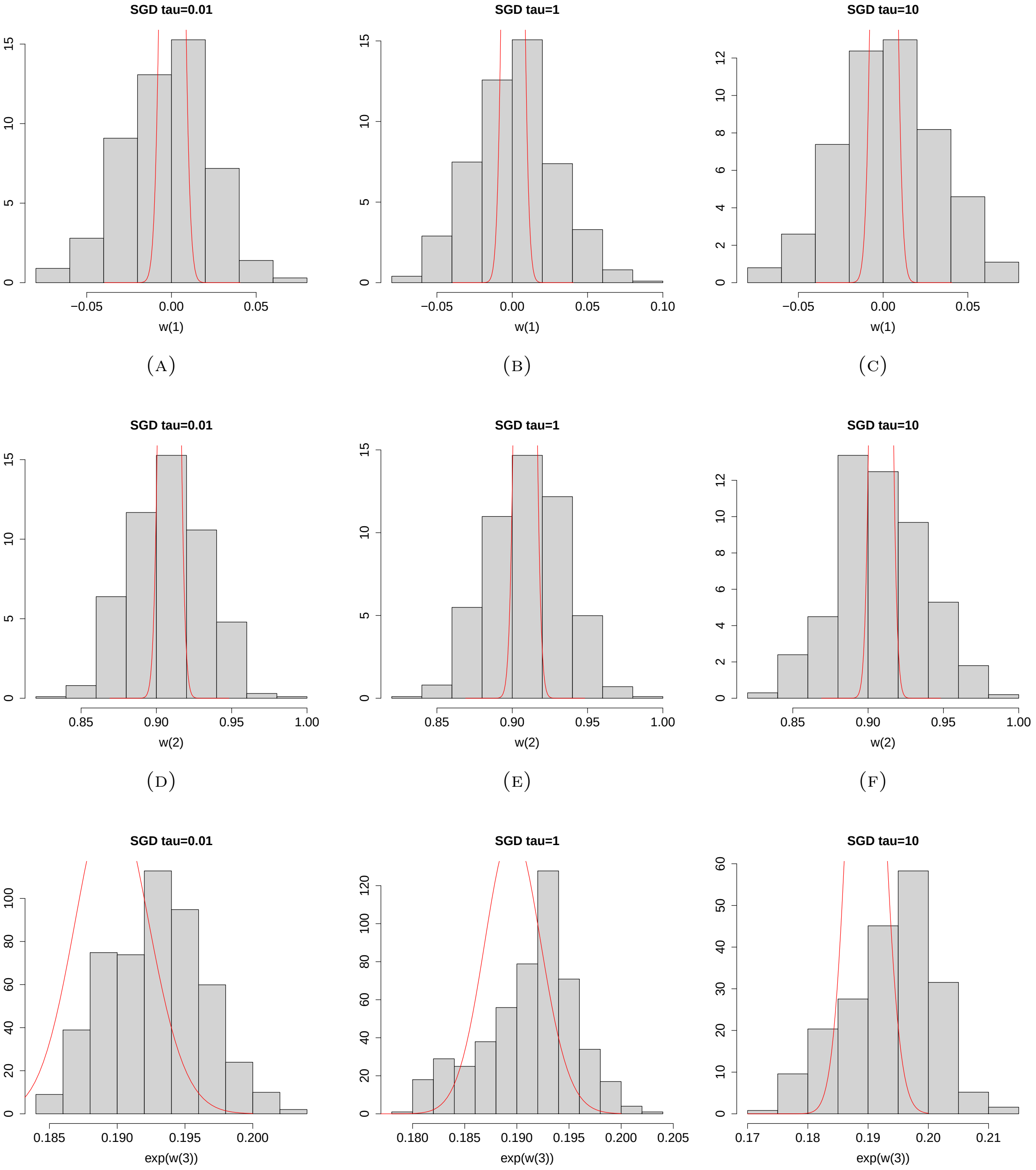
(F)

(G)

(H)

(I)

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



(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

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, 2023

Draft Handout 5: 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) Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.
- Ch. 20 Neural Networks

(2) Bishop, C. M., & Nasrabadi, N. M. (2006). Pattern recognition and machine learning (Vol. 4, No. 4, p. 738). New York: springer.
- Ch. 5 Neural Networks

(3) Neural Networks for Pattern Recognition
- Ch. 4 The multi-layer perceptron

(4) 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.

1. Intro and motivation [1]

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.

Note 2. 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.

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

where the sum is over neuron i connected to neuron j; $n_j\left(t\right)$ 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...

[1] In this Section, formulas not needed to be memorized.

Remark 3. 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

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

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 4. 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 $y \in \mathbb{R}^q$. A 2 layer artificial neural network is

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

for $k = 1, ..., q$. One may choose with $\sigma_2(\alpha) = \alpha$, $\sigma_1(\alpha) = 1/(1+\exp(-\alpha))$. The only left here is to learn unknown parameters $\left\{w_{(\cdot),\cdot,\cdot}\right\}$.

2. Feedforward Neural Network (Mathematical set-up)

Note 5. An (Artificial) Neural Network (NN) can be depicted as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose nodes $\mathcal{V}$ correspond to neurons and edges $\mathcal{E}$ correspond to links between them.

Note 6. A Feed-Forward Neural Network (FFNN) or else multi-layer perceptron is a special case of NN which can be depicted by a directed acyclic graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. [2] (See Figure 2.1).

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

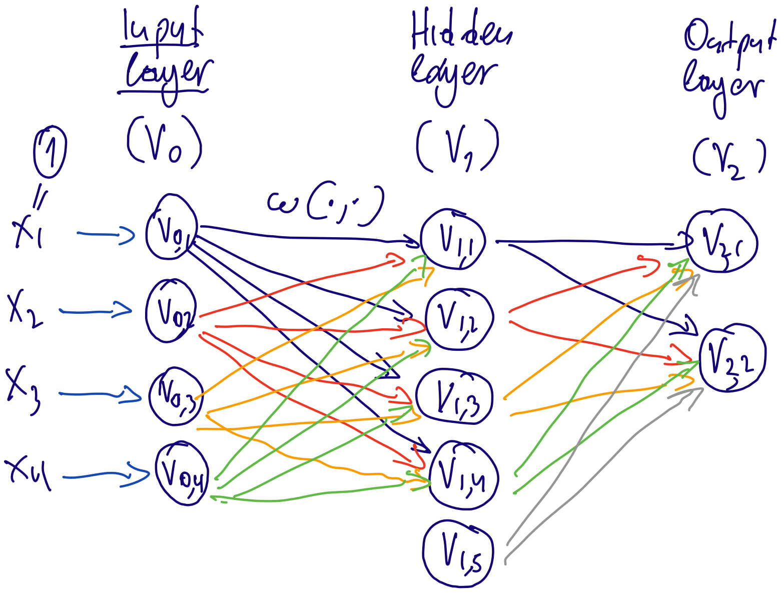


FIGURE 2.1. Feed forward neural network (1 hidden layer)

Note 7. 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 8. The first layer V_0 is called **input layer**. If x has d dimensions, then the first layer V_0 contains d nodes.

Note 9. The last layer V_T is called **output layer**. If y has q dimensions, then the last layer V_T contains q nodes.

Note 10. The intermediate layers $\{V_1, ..., V_{T-1}\}$ are called **hidden layers.**

Note 11. In a neural network, the nodes of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ correspond to neurons.

Notation 12. The **i-th neuron of the t-th layer** is denoted as $v_{t,i}$.

Note 13. The output of neuron i in the input layer V_0 is simply x_i that is $o_{0,i}(x) = x_i$for $i = \{1, ..., d\}$
.

Note 14. 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 15. We define a weight function $w : \mathcal{E} \to \mathbb{R}$ over the edges $\mathcal{E}$.

Note 16. 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

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

Note 17. 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 18. We denote by $o_{t,i}(x) := \sigma_{t-1}(\alpha_{t-1,i}(x))$ **the output of neuron** $v_{t,i}$ when the network is fed with the input x.

Remark 19. 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

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

Note 20. The input of a neuron is obtained by taking a weighted sum of the outputs of all the neurons connected to it, where the weighting is according to w.

Definition 21. To sum-up: The the feed-forward NN formula in a layer by layer manner is performed (defined) according to the following recursion.

Note 22. At $t = 0$, for $i = 1, ..., |V_0|$

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

At $t > 0$, for $i = 1, ..., |V_{t+1}|$

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

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 $\alpha_{t+1,i}(x)$.

Note 24. Depth of the NN is the number of the layers T.

Note 25. Size of the network is the number $|V|$.

Note 26. Width of the NN is the number $\max_{\forall t}(|V_t|)$.

Note 27. The architecture of the neural network is defined by the triplet $(\mathcal{V}, \mathcal{E}, \sigma_t)$.

Note 28. The neural network can be fully specified by the quadruplet $(\mathcal{V}, \mathcal{E}, \sigma_t, w)$.

Example 29. Figure 2.1 denotes a NN with depth 2, size 11, width 5. The neuro with no incoming edges has$o_{1,5} = \sigma(0)$.

Notation 30. To easy the notation, we denote the weights as $w_{(t),j,i} := w\left(\left(v_{t,j}, v_{t+1,i}\right)\right)$. Using this notation, $w_{(t),j,i} = 0$ is equivalent in (2.2) to $\left(v_{t,j}, v_{t+1,i}\right) \notin \mathcal{E}$ and means that the link $v_{t,j} \rightarrow v_{t+1,i}$ is not in the network.

Note 31. Often a **constant neuron** $v_{t,0}$ (at each layer t and $i = 0$) which outputs 1; i.e. $o_{0,0}\left(x\right) = 1$ and $o_{t,0}\left(x\right) = 1$. The corresponding weight $w_{(t),k,0}$ is called **bias**. This resembles to the constant term in the linear regression.

Example 32. (Cont. Example 4) The 2 layer neural network

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

can be written according to the recursion in Note **??** as

- Input layer

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

- Hidden layer

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

- Hidden layer

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

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

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

by considering the first x as 1, i.e.g $x_1 = 1$ similar to the linear regression models.

Remark 33. Activation functions σ_t are non-increasing functions often sigmoids or threshold functions. Their choice is problem-dependent. some examples

- Identity function: $\sigma\left(\alpha\right) = \alpha$ 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 34. 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 35. 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 Definition 21.

Note 36. Learning $(\mathcal{V}, \mathcal{E}, \sigma)$ is a model selection task, as it involves learning the architecture of the neural network.

Note 37. We assume that the architecture $(\mathcal{V}, \mathcal{E}, \sigma)$ of the neural network is fixed (given), 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 $\left\{w_{(t),j,i}\right\}$ where $w_{(t),j,i} := w((v_{t,j}, v_{t+1,i}))$.

Note 38. 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 $(\mathcal{V}, \mathcal{E}, \sigma)$. To simplify notation we will use h_w instead of $h_{\mathcal{V},\mathcal{E},\sigma,w}$.

Note 39. 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 40. We consider a loss function $\ell(w, z)$ at some value of weight vector $w \in \mathbb{R}^{|\mathcal{E}|}$ and at some example $z = (x, y)$.

Example 41. For instance, loss function can be based on

(1) some norm $\ell(w, z) = \frac{1}{2}\|h_w(x) - y\|_2^2$, or
(2) the pdf/pmf of the sampling distribution $f(y|w)$ of the target y given the weight vector w as $\ell(w, z) = -\log(f(y|w))$

Definition 42. Error function is a performance measure that can be defined as

$$\text{EF}(w|z) = -\sum_{i=1}^{n} \ell(w, z_i^*)$$

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 $\mathcal{S}$.

4. Classical learning of Neural Network

Problem 43. Find $h_w \in \mathcal{H}_{\mathcal{V},\mathcal{E},\sigma}$ essentially find $w \in \mathbb{R}^{|\mathcal{E}|}$ under loss $\ell(\cdot,\cdot)$, and training data-set $\mathcal{S} = \{z_i = (x_i, y_i)\,; i = 1, ..., n\}$. Compute $w^* \in \mathbb{R}^{|\mathcal{E}|}$, by minimizing the risk function $R_g(w)$

$$w^* = \arg\min_{w\in\mathcal{H}} \left(R_g(w)\right) = \arg\min_{w\in\mathcal{H}} \left(\mathrm{E}_{z\sim g}\left(\ell(w,z)\right)\right) \tag{4.1}$$

or by minimizing the empirical risk function $\hat{R}_S(w)$

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

Problem 44. Often neural network models are over-parameterized, namely the dimensionality of $w \in \mathbb{R}^{|\mathcal{E}|}$ is too large. One can resort to shrinkage methods. For a given architecture $(\mathcal{V}, \mathcal{E}, \sigma)$, 'node selection' can be performed if $w_{(t),j,i} = 0$ which 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 active (essentially not in the network).
The problem is to find w^* such that

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

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

where $J(w;\lambda)$ is a shrinkage term such as

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

Note 45. 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 46. Training a neural network model is usually a high-dimensional problem (essentially w has high dimensionality) 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 stochastic learning algorithm.

Note 47. To address Problem 43, the recursion of the SGD with batch size m is

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

To address the Problem 44the recursion of the SGD with batch size m is

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

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

Definition 48. 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 49. 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 $e \in \mathcal{E}$.

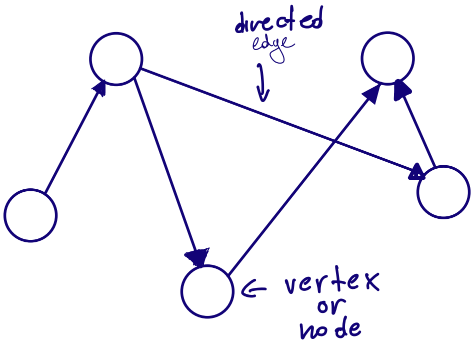


Figure A.1. A directed graph