Lecture Notes for CIE6128

Lecture Notes for CIE6128

Lecturer
Ruoyu Sun
Assistant Professor, University of Illinois
sundirac@gmail.com

Lecturer
Mingyi Hong
Assistant Professor, University of Minnesota
poborskypaul@gmail.com

Tex Written By
Jie Wang
Undergraduate Student, CUHK(SZ)
116010214@link.cuhk.edu.cn

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

ENJOY MATH!

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

J. Wang. *Lecture Notes for CIE6128.* , vol. XX, no. XX, pp. 1–112, 2019.

ISBN: XXX-X-XXXXX-XXX-X

Volume XX, Issue XX, 2019

Editorial Board

Editorial Scope

Topics

Information for Librarians

Contents

Lecture Notes for CIE6128

ABSTRACT

How does deep learning work? Without any guidance, it is very difficult to find the correct setting that makes a deep neural network perform well. Researchers have spent a few decades to find the right combination of choices that lead to recent success of deep neural-nets. However, most, if not all, existing courses and textbooks did not discuss the underlying theory in depth. In this course, we will go through the current theoretical understanding of neural networks based on research in recent years. Topics include the theory of training dynamics (smart initialization and dynamical isometry, normalization methods), popular algorithms (SGDR, Adam, etc.), landscape of non-convex optimization, adversarial attacks and robustness, GANs (generative adversarial networks), etc.)

1

Introduction to Deep Learning

1.1 Motivation

Example: is AI like Alchemy? A core of deep learning is to optimize a loss function with respect to a collection of model parameters. In NIPS 2017 talk, Rahimi gives a typical example for choosing parameters for a 2-layer neural network.

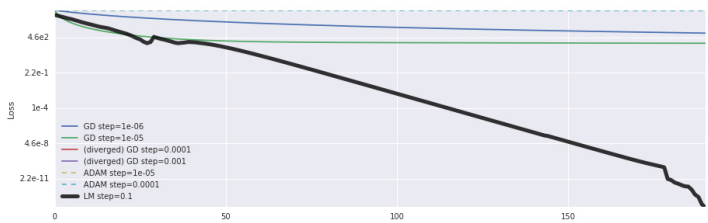


Figure 1.1: Numerical Results for solving $\min_{W_1,W_2} \hat{\mathbb{E}}\|W_1W_2x - Ax\|^2$

The gradient descent for solving this problem gets stuck at the objective value (error) around 400. In particular,

- It does not converge to a stationary point
- This phenomenon is not due to the statistical noise floor of the

dataset. Computing the expectation of the loss and directly minimize it leads to the same result.

Rahimi reveals that gradient descent cannot usually solve a 2-layer linear network. However, we expect it to solve a 1000-layer neural-net with 1 million data. It seems we don't understand neural-net optimization at all.

LeCun has a different pespective. He thinks that having theory for deep learning is important, but another goal is to invent new methods, new tricks, etc. Perhaps we don't have or don't need theory for neural networks.

Comment from Ruoyu Sun

1. Do we need to study deep learning?

   For application researchers, there is no doubt. For applications, Neural-nets (not necessarily current deep learning) will probably become fundamental tool in many fields. For theoretical researchers such as optimizers, mathematicians, statisticians, this is still a big question. Deep learning is possible to become a theoretical field.

2. Do we need theory for deep learning?

   Yes, since we need to *udnerstand* why deep learning works. There are tons of hard deep learning problems, and we need to solve them. Moreover, theory can help deep learning work for other fields such as reinforcement learning, physics, biology, etc.

3. This course tries to offer answers to these questions, but it cannot answer them completely, not even a 20% answer. We will analysis deep learning from the perspective of optimization, and understand why some heuristics can help.

1.2 Outline

Pre-requisite

- Pre-requisites: Calculus, Linear Algebra, Probability, Optimization, and Basics of Deep Learning.

- Related Subjects (helpful but not required): Basic knowledge from compute vision and NLP, the trend of Deep Learning and Artificial Intelligence.

Course Objective and Audience This course will talk about optimization theory of deep learning. This is not intended for practitioners with little interest in theory, and want to know how to tune parameters. Intended audience:

- Theorists (e.g., optimizers, machine learners, statisticians, mathematicians, theoretical computer scientists) who want to work on or interested in theory of deep learning;

- Practitioners interested in theory;

- Those who curious about frontiers of optimization in deep learning.

Also note that the theory discussed in this course does not necessarily solve real problems in deep learning, but it offers the logic of thinking that is the most helpful.

1.3 Neural Network Basis

Firstly, we will give mathematical descriptions of fully-connected neural networks. The Figure (1.2) gives a demostration of 3-layer fully-connected neural network. Neural networks give a function $f : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ parameterized with $\theta \in \mathbb{R}^{d_\theta}$:

- Input: $x \in \mathbb{R}^{d_x}$;

- Output: $y \in \mathbb{R}^{d_y}$;

- A L-layer fully connected neural network consists of $L - 1$ hidden layers.

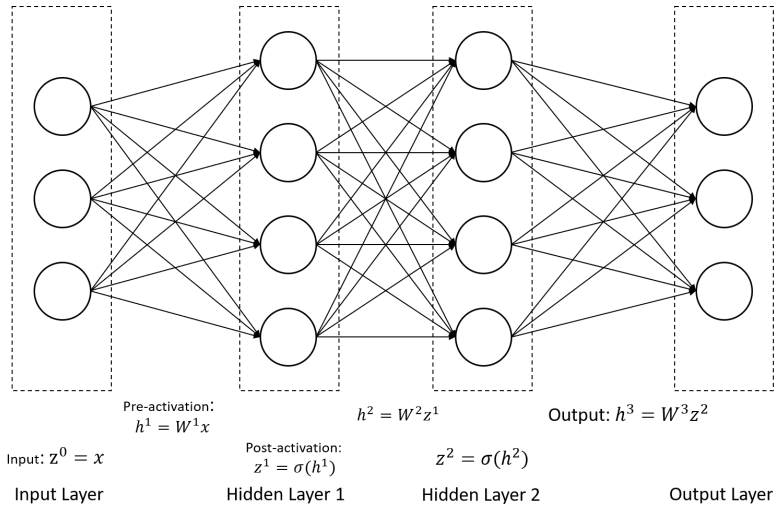


Figure 1.2: Example of a 3-layer fully-connected neural network.

- The values of the next hidden layer are a linear transformation of previous values, and then followed by a non-linear function:

$$\begin{aligned} &\text{pre-activiation}: \quad h^\ell = W^\ell z^{\ell-1}, \quad \ell = 1, \dots, L \\ &\text{post-activation}: \quad z^\ell = \phi(h^\ell), \quad \ell = 1, \dots, L \end{aligned}$$

  Here $W^\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ denotes the weight matrix, and ϕ denotes the nonlinear function.

Using the notions above, the function f_θ can be written as

$$f_\theta(x) = W^L \phi(W^{L-1} \phi(\cdots \phi(Wx))) \tag{1.1}$$

Remark 1.1. The bias of the neural network is skipped. In general, it should be $z^\ell = \phi(W^\ell z^{\ell-1} + b^\ell)$.

Remark 1.2. There are other kinds of neural network, such as CNN, RNN, and ResNet.

Why & When & How do we need neural-nets? Consider the image classificaiton scenario. Imagine there is an orcle telling you the label

of the input image, say f^*. The motivation for using deep learning is that it can approximate such a function very well. The classical way for applying deep learning is the supervised learning:

> Given data (x_i, y_i) for $i = 1, \ldots, n$, we need to find a model f_θ from a set of model candiates $\mathcal{F}$ such that $f_\theta(x_i) \approx y_i$, $i = 1, \ldots, n$.

This problem is often formulated as a finite-sum optimization problem:

$$\min_\theta \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i),$$

where $\ell(\cdot, \cdot)$ denotes the loss function.

1. When the representation model $f_\theta(x) = Wx$, and the loss is quadratic, the problems becomes a least-square linear regression problem

2. When the representation is a 2-layer neural network, say $f_\theta(x) = W^2W^1x$ and the loss is quadratic, the problem becomes

   $$\min_{W^2, W^1} \sum_i \|y_i - W^2W^1x_i\|^2 = \|Y - W^2W^1X\|_F^2$$

   When $X = I$, this problem reduces to the matrix factorization problem:

   $$\min_{U,V} \|Y - UV\|_F^2 \tag{1.2}$$

   For matrix $Y \in \mathbb{R}^{m \times n}$ and decision variables $U \in \mathbb{R}^{m \times p}, V \in \mathbb{R}^{p \times n}$, if $p < \text{rank}(Y)$, then the optimal solution

   $$U^* = \begin{pmatrix} \sqrt{\sigma_1}u_1 & \cdots & \sqrt{\sigma_p}u_p \end{pmatrix}, \quad V^* = \begin{pmatrix} \sqrt{\sigma_1}v_1 & \cdots & \sqrt{\sigma_p}v_p \end{pmatrix}^{\mathrm{T}}$$

   where $\{u_i, v_i, \sigma_i\}_{1:p}$ are from the first p terms of the SVD decomposition of Y.

1.4 Gradient Explosion/Vanishing

Consider minimizing a quadratic loss for multi-layer neural network with scalar input:

$$\min_{w_1, \ldots, w_L} F(w) \triangleq \frac{1}{2}(1 - w_1 \cdots w_L)^2$$

Solving by Classical Gradient Descent If applying gradient descent, the step size should be bounded by 1 over the Lipschitz constant β of the objective function. Here we compute this constant:

$$\frac{\partial F}{\partial w_1} = -(1 - w_1 \cdots w_L) w_2 \cdots w_L$$
$$\frac{\partial^2 F}{\partial w_1^2} = (w_2 \cdots w_L)^2$$

Note that $L \approx \lambda_{\max}(\nabla_w^2 F)$. Here we simply use $\frac{\partial^2 F}{\partial w_1^2}$ to loosely study how large (small)β is.

- When $w_2 = \cdots = w_L = 2$, then $\beta = 2^{\mathcal{O}(L)}$
- When $w_2 = \cdots = w_L = \frac{1}{2}$, then $\beta = 2^{-\mathcal{O}(L)}$

Take $L = 100$ as an example. For the first initialization, the step-size is too big; for the second initialization, the step size is too small. This issue is called *Lipschitz constant explosion/vanishing*.

2

Back Propagation and Initialization

2.1 Review

- Neural-net and formulation; (section 1.3)
- Training difficulty; (section 1.4)

  Example 2.1. Consider the multi-layer ($L = 7$) linear neural network with scalar input. The function shape of the loss function $y(w) \triangleq (w^7 - 1)^2$ is presented in Figure (2.1)

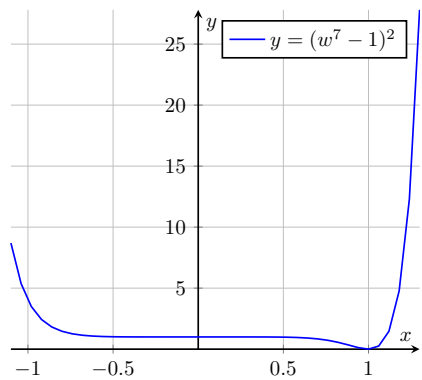


Figure 2.1: Function Shape of $(w^7 - 1)^2$

From Figure (2.1) we can see that when $x \in [-0.5, 0.5]$, the gradient of the loss function *nearly* vanishes; when $x > 1.2$, the gradient exploses into infinite. These two bad region makes the training (optimization) process of such neural network very difficult.

How to rescue the gradient vanishing/explosion during DL training? The "good" region of the loss function is small. There are two ways to rescue this phenomenon:

1. By proper initialization, it's possible to find a good region;
2. By techniques such as *Batch Normalization*, we can change the landscape of the loss function.

2.2 Back Propagation

Suppose that the loss function is of finite-sum form:

$$F(\theta) \triangleq \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$$

with $f_\theta(x_i) = W^L(\phi(W^{L-1}\phi(\cdots \phi(W(x)))))$, and the weight matrices W^ℓ are parameterized by θ. The direct motivation of *back propagation* is to apply gradient descent [1] to minimize the loss function:

$$\theta(t+1) = \theta(t) - \alpha_t \nabla F(\theta(t)).$$

The non-trivial part during this process is how to tuning parameters α_t and how to compute $\nabla F(\theta(t))$. The *back propagation* (BP) technique is one efficient strategy to compute the gradient by chain rule, since it avoids repeating the same computations.

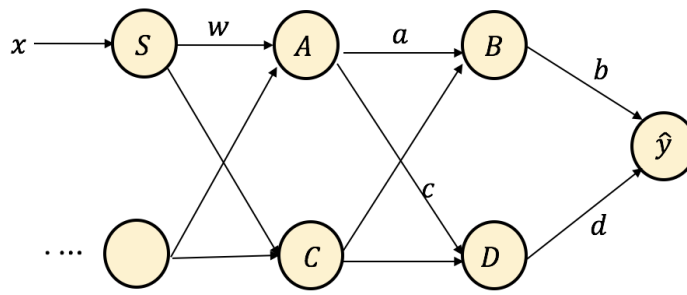
Understanding BP in Level I: Scalar Form of Gradient Most courses/blogs teach how to do BP in scalar version, i.e., to compute the derivative of a scalar-valued function over a scalar variable, which are based on two rules:

[1] Usually we use stochastic gradient descent method in DL since this method is more efficient

- Chain Rule: $f(g(w))$ with $f, g \in \mathbb{R}$,
$$\frac{\mathrm{d}f(g(w))}{\mathrm{d}w} = \frac{\mathrm{d}f}{\mathrm{d}g}\frac{\mathrm{d}g}{\mathrm{d}w}$$
- Sum rule: $g(w) \triangleq f_1(w) + f_2(w)$ with $w \in \mathbb{R}$,
$$\frac{\mathrm{d}g}{\mathrm{d}w} = \frac{\mathrm{d}f_1}{\mathrm{d}w} + \frac{\mathrm{d}f_2}{\mathrm{d}w}$$

We give an example on how to apply these two rules to compute the scalar form of the gradient of the loss function:

Example 2.2. Consider a 2-layer neural network with scalar output. We are interested in computing the derivative of this output $\hat{y}$ over a scalar parameter w. This function w.r.t. w can be represented in graph:



The computation of $\frac{\partial \hat{y}}{\partial w}$ can be summarized as follows:

Step 1: Decompose into multiple paths The path from the parameter w to the output $\hat{y}$ undergoes two paths:
$$w \to A \to B \to \hat{y}$$
$$w \to A \to D \to \hat{y}$$

Step 2: Take gradient of each path by Chain rule These paths corresponds to the functions (w.r.t. w) as follows:
$$f_1(w) = b \cdot \phi(a \cdot \phi(w \cdot x)$$
$$f_2(w) = d \cdot \phi(c \cdot \phi(w \cdot x)$$

The derivative of $f_1(w)$ is computed by the Chain rule:

$$\frac{\partial f_1}{\partial w} = [b \cdot \phi'(a \cdot \phi(w \cdot x_1)] \cdot [a \cdot \phi'(w \cdot x)] \cdot [x]$$

The derivative of $f_2(w)$ can be computed similarly.

The coding is doable in this understanding level.

Understanding BP in Level II: Matrix Form of Gradient Firstly let's review some matrix calculus knowledge by an example.

Example 2.3. Consider a 2-layer linear network[2] $f_\theta(x) = UVx$. Given n data points (x_i, y_i), the goal is to minimize the loss function

$$F \triangleq \frac{1}{n} \sum_{i=1}^{n} \|UVx_i - y_i\|^2,$$

with U, V to be determined. The question is how to take gradient of F w.r.t. the matrix V? Or even simpler, how to compute $\frac{\partial F}{\partial V}$ with $F \triangleq \|UV - Y\|_F^2$? Here suppose that $U \in \mathbb{R}^{d_y \times d_1}$, $V \in \mathbb{R}^{d_1 \times d_x}$, $Y \in \mathbb{R}^{d_y \times d_x}$.

- Let's try to compute the gradient by "standard" Chain rule. Define $H = U \cdot V$, $E = H - Y$, and $F = \|E\|_F^2$.

$$\begin{aligned} \frac{\partial F}{\partial V} &= \frac{\partial F}{\partial E} \frac{\partial E}{\partial H} \frac{\partial H}{\partial V} \\ &= (2E) \cdot I \cdot (U) \end{aligned}$$

  Then check the dimension. We find $E \in \mathbb{R}^{d_y \times d_x}$ and $U \in \mathbb{R}^{d_y \times d_1}$. The matrix-multiplication is undefined! If we want to make the dimension matched, we should write

$$\frac{\partial F}{\partial V} = 2U^{\mathrm{T}} E.$$

Sometimes it's problematic to write gradient by checking matrix dimensions. For instance, if $d_y = d_x$ in practice, this method is invalid.

[2] The weight matrices U, V are parameterized by θ

Another way is to write down scalar-input scalar-output derivatives and then form the whole matrix[3]. However, this way is tedious in practice.

The reason why our method is problematic is that we probably applied the Chain rule incorrectly. Wikipedia provides the Chain rule for vector-valued functions:

Proposition 2.1 (Vector-Function Chain Rule)**.** For vector-input vector-output functions

$$x \in \mathbb{R}^n \to g(x) \in \mathbb{R}^m \to F(x) \triangleq f(g(x)) \in \mathbb{R}^k,$$

the chain rule is

$$\frac{\partial F}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \frac{\partial g(x)}{\partial x},$$

where

$$\frac{\partial f(g(x))}{\partial x} = \left[\frac{\partial f_i(g(x_j))}{\partial x_j}\right]_{ij} \in \mathbb{R}^{k\times m}, \quad \frac{\partial g(x)}{\partial x} = \left[\frac{\partial g_i}{\partial x_j}\right]_{ij} \in \mathbb{R}^{m\times n}$$

denotes the Jacobian matrices.

- Consider the objective function $F = \|UVx - y\|_F^2$. The goal is to apply proposition (2.1) to write $\frac{\partial F}{\partial V}$.

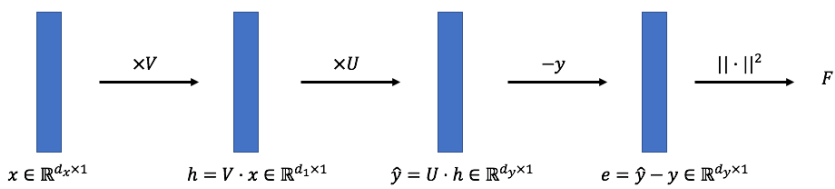


Figure 2.2: Diagram for the operator F

As a result,

$$\frac{\partial F}{\partial V} = \frac{\partial F}{\partial e}\frac{\partial e}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h}\frac{\partial h}{\partial V}$$

[3]LeCun, CS224 Note, https://web.stanford.edu/class/cs224n/

In this formula, the LHS is of the form (∂ scalar/∂ matrix), which should be a matrix; the first term in RHS is of the form (∂ scalar/∂ vector), which should be a vector; the second and third term in RHS are of the form (∂ vector/∂ vector), which should be a matrix; the forth term is of the form (∂ vector/∂ matrix), which should be a tensor. Here we discuss the issues for computing these derivatives:

1. Issue 1: computing derivative of a scalar over a vector.
   The issue for computing $\frac{\partial F}{\partial e}$ is on the confusion of the different notions of derivatives.
   - By definition of Jacobian matrices from proposition (2.1), $\frac{\partial F}{\partial e} \in \mathbb{R}^{\text{fan-out} \times \text{fan-in}} = \mathbb{R}^{1 \times d_y}$, which is a row vector;
   - By definition of gradient, we assume $\frac{\partial F}{\partial e}$ is a column vector instead, i.e., a vector of dimension $d_y \times 1$.
   - Moreover, the notion of Jacobian and gradient coincides[4] for the case fan-out > 1, e.g.,
   $$\frac{\partial (Wx)}{\partial x} = W.$$

   Based on the issues above, one solution is to define the *general Jacobian* to unify the notions of gradient and Jacobian. Before that, from now on, we define $\frac{\partial f}{\partial x}$ as a row vector if f is scalar-valued, otherwise $\frac{\partial f}{\partial x}$ denotes the Jacobian matrix. Moreover, define the *general Jacobian*
   $$\frac{\tilde{\partial} f}{\tilde{\partial} x} = \begin{cases} \dfrac{\partial f}{\partial x}, & \text{if fan-out}> 1 \text{ and fan-in}> 1 \\ \left(\dfrac{\partial f}{\partial x}\right)^{\mathrm{T}}, & \text{if fan-out} = 1 \end{cases}$$

   The proposition (2.1) always holds for *general Jacobian*. Intermediately,
   $$\frac{\tilde{\partial} F}{\tilde{\partial} h} = \frac{\tilde{\partial} F}{\tilde{\partial} e} \frac{\tilde{\partial} e}{\tilde{\partial} h} \implies \left(\frac{\tilde{\partial} F}{\tilde{\partial} h}\right)^{\mathrm{T}} = \left(\frac{\tilde{\partial} e}{\tilde{\partial} h}\right)^{\mathrm{T}} \left(\frac{\tilde{\partial} F}{\tilde{\partial} e}\right)^{\mathrm{T}}$$

[4] At least in some references, e.g., Matrix Differentiation, available at https://atmos.washington.edu/~dennis/MatrixCalculus.pdf

Or equivalently,

$$\begin{array}{ccccc} \frac{\partial F}{\partial h} & = & \left(\frac{\partial e}{\partial h}\right)^{\mathrm{T}} & \left(\frac{\partial F}{\partial e}\right) & \\ \uparrow & & \uparrow & \uparrow & \\ \frac{\partial \text{ scalar}}{\partial \text{ vector}} & & \frac{\partial \text{ vector}}{\partial \text{ vector}} & \frac{\partial \text{ scalar}}{\partial \text{ vector}} & \end{array} \tag{2.1}$$

2. Issue 2: computing derivative of a vector over a matrix.
   There are two ways to solve this issue:
   - The first way is to reduce matrix into vectors, i.e., in order to compute $\frac{\partial F}{\partial V}$, it suffices to consider $\frac{\partial F}{\partial V(:,k)}$ and then combine to form a tensor.
   - The other is to use Lemma (2.1) that deals vector-matrix derivative into the vector-vector cases.

   Let's consider the second way in this lecture.

   Lemma 2.1. For $g(V) \triangleq \phi(Vx)$ with $x \in \mathbb{R}^{d\times 1}$ and $V \in \mathbb{R}^{k\times d}$, define $h = Vx$. Then

$$\frac{\partial g}{\partial V} = \frac{\partial \phi}{\partial h} x^{\mathrm{T}}$$

   Now we give an example for applying Lemma (2.1) to compute $\frac{\partial F}{\partial V}$:

$$\begin{aligned} \frac{\partial F}{\partial V} &= \frac{\partial F}{\partial h} x^{\mathrm{T}} && \text{(2.2a)} \\ &= \left(\frac{\partial e}{\partial h}\right)^{\mathrm{T}} \left(\frac{\partial F}{\partial e}\right) x^{\mathrm{T}} && \text{(2.2b)} \\ &= \left(\frac{\partial e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h}\right)^{\mathrm{T}} \left(\frac{\partial F}{\partial e}\right) x^{\mathrm{T}} && \text{(2.2c)} \\ &= (I \cdot U)^{\mathrm{T}} 2e \cdot x^{\mathrm{T}} && \text{(2.2d)} \\ &= 2U^{\mathrm{T}} e x^{\mathrm{T}} \end{aligned}$$

   where (2.2a) is because of Lemma (2.1) and $F(V) = F(Vx)$; (2.2b) is by the substitution of (2.1); (2.2c) is by the Chain rule stated in proposition (2.1); (2.2d) is by direct calculation.

Exercise:

$$\frac{\partial \|AWB + C\|_F^2}{\partial W} = 2A^{\mathrm{T}}(AWB + C)B^{\mathrm{T}}$$

BP for General Deep Non-linear Network Now derive the gradient of fully-connected neural network with quadratic loss. The objective f_θ is defined based on the following diagram:

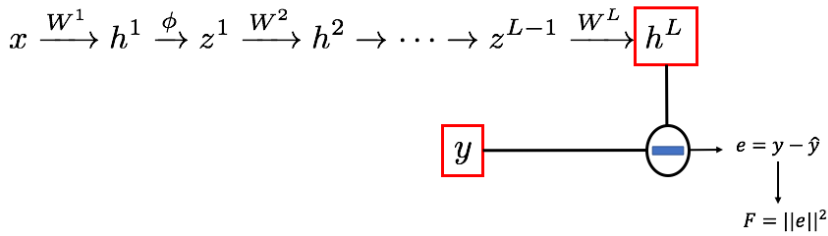


Figure 2.3: Diagram for the operator F

Then the derivative $\frac{\partial F}{\partial W^1}$ is computed as follows:

$$\frac{\partial F}{\partial W^1} = \frac{\partial F}{\partial h^1} x^{\mathrm{T}} \tag{2.3a}$$

$$= \left(\frac{\partial e}{\partial h^1}\right)^{\mathrm{T}} \left(\frac{\partial F}{\partial e}\right) x^{\mathrm{T}} \tag{2.3b}$$

$$= \left(\frac{\partial e}{\partial h^1}\right)^{\mathrm{T}} 2e \cdot x^{\mathrm{T}}$$

$$= \left(\frac{\partial e}{\partial h^L} \frac{\partial h^L}{\partial z^{L-1}} \cdots \frac{\partial h^1}{\partial z^1} \frac{\partial z^1}{\partial h^1}\right)^{\mathrm{T}} 2e \cdot x^{\mathrm{T}} \tag{2.3c}$$

$$= \left(W^L D^{L-1} W^{L-1} D^{L-2} \cdots W^2 D^1\right)^{\mathrm{T}} 2e \cdot x^{\mathrm{T}} \tag{2.3d}$$

where (2.3a) is by Lemma (2.1); (2.3b) follows the similar trick as in (2.1); (2.3c) is by the Chain rule stated in proposition (2.1); in (2.3d) the matrix $D^\ell \triangleq \mathrm{diag}(\phi'(h_i^\ell))_{i=1}^{d_\ell}$, with ϕ' denotes the derivative of ϕ. The general formula $\frac{\partial F}{\partial W^\ell}$ is left as exercise:

$$\frac{\partial F}{\partial W^\ell} = (W^L D^{L-1} \cdots W^{\ell+1} D^\ell)^{\mathrm{T}} \cdot 2e \cdot (z^{\ell-1})^{\mathrm{T}}$$

This formula can be expressed in a recursive way, which is the mechanism of the BP technique. BP is an efficient way to compute all gradients

$\frac{\partial F}{\partial W^\ell}$ for $\ell = 1, \ldots, L$. The navie computation complexity is $\mathcal{O}(d^2L^2)$; while the BP complexity is $\mathcal{O}(d^2L)$.

2.3 Initialization methods for handling Training Difficulty

We have discussed the *gradient explosion or vanishing* issue. The step size for the gradient descent method is one over the Lipschitz constant, which will be super-small/super-large in gradient explosion/vanishing cases. From the landscape in Fig. (2.1), we can see that w^7 grows more active compared with the input $x = 1$. To solve this problem, the direct idea is to control the "energy" of output compared with the input, i.e., for linear network $y = W^L W^{L-1} \cdots W^1 \cdot x$, we want to have

$$\|W^L W^{L-1} \cdots W^1 \cdot x\| \approx \|x\|.$$

Or even simpler, maybe it's enough to let $\|W^\ell x\| \approx \|x\|$ for $\ell = 1, \ldots, L$. Assume W^ℓ is initialized to be a random matrix. After simulation we found that the energy (ℓ_2 norm) for the output after activation is much larger than the previous input.

```
clear;
d = 100; % dimension for weight matrix W
maxit = 10; % maximum iteration number

x = ones(d,1);  norm0 = norm(x);
for i = 1:maxit
    W = randn(d,d);
    x = W*x;
    rato = norm(x)/norm0
end
```

There are different ways to deal with this problem:

- Sparsity Solution: Set many entries of W to be 0;
- Orthogonalization: Generate orthogonal random weight matrix; (to be discussed in the future)
- Scalization: Normalize each entry of W by some constant C.

We find that if each entry of W (assume to be square matrix first) is divided by $\sqrt{d}$, then the energy of $\|W \cdot x\|$ is very close to $\|x\|$.

Informal Xavier Initialization: for the special case where $d = d_x = d_1 = \cdots = d_{L-1} = d_L$, initialize

$$W_{i,j}^{\ell} \sim \mathcal{N}(0,1) \cdot \frac{1}{\sqrt{d}}$$

Supporting Analysis

1. Claim 1: For fixed x, if entries of W are i.i.d. such that

$$W_{i,j} \sim \mathcal{N}(0, 1/d), \text{[5]} \tag{2.4}$$

then

$$\mathbb{E}\|Wx\|^2 = \|x\|^2.$$

Proof. Two-line proof: $\mathbb{E}\|Wx\|^2 = x^{\mathrm{T}}\mathbb{E}[W^{\mathrm{T}}W]x$ and evaluate the term $\mathbb{E}[W^{\mathrm{T}}W]$. □

Sometimes x are also initialized as random number. Therefore, there is a stronger version of claim 1.

2. Claim 2: if x_i's are i.i.d., and previous conditon holds[6], and x is independent of W, then

$$\mathbb{E}\|Wx\|^2 = \|x\|^2.$$

Remark 2.1. 1. If the input and the output dimension are not the same, there is an in-consistency in (2.4). In this case, we try $W_{i,j}^{\ell} \sim \mathcal{N}(0, 2/(d_{\text{fan-in}} + d_{\text{fan-out}}))$. This is the formal Xavier Initialization.

2. The claims 1 and 2 are only about feed-forward neural network. For the back-ward case, i.e., $e^1 = (W^L W^{L-1} \cdots W^1)^{\mathrm{T}} e$, we need to have $W_{ij} \sim \mathcal{N}(0, 1/d_{\text{fan-in}})$.

[5] for the case fan-in $\neq$ fan-out, use $W_{i,j} \sim \mathcal{N}(0, 1/d_{\text{fan-out}})$

[6] Again, for the case fan-in $\neq$ fan-out, follow the setting in claim 1.

3. The conditions for claims 1 and 2 can be weakened a little bit, e.g., the Gaussian assumptions of W are not needed but only the mean and variance assumptions.

4. For non-linear activation such as relu function, the He Initialization / Kaming Initialization is needed. The initution is that $\mathbb{E}[\text{Relu}(w^2)] = 1/2$. In this case, initialize

$$\mathbb{E}W_{ij}^{\ell} = 0, \quad \text{Var}(W_{ij}^{\ell}) = \frac{2}{\text{fan-in}} \quad \text{or} \frac{2}{\text{fan-out}}$$

3

Taming Explosion/Vanishing: Initialization

3.1 Reviewing

1. Does Xevier intialization allow $W_{ij}^{\ell} \sim C \cdot \text{rand}$?
   No. From the assignment we know that $\mathbb{E}W_{ij}^{\ell}$ should be zero.
2. How to pick variance of $W_{i,j}$ for non-linear activation functions?
   - Relu activiation: twice the variance.
   - other types of activiation: to be discussed today
3. Does the Chain rule work for derivative of matrix over vector?
   Not directly. We need to derive a different form

3.2 Motivation

Three topics to be discussed today:

- The difference for training *wide* versus *narrow* neural network
- Mean-field approximation
- Dynamical Isometry (spectrum analysis)

The motivation is that engineers believe that the initialization for training a neural network is important, otherwise the gradient explosion/vanishing will happen. These topics discuss the initialization with connection to gradient explosion/vanishing from different perspectives.

Example 3.1. Review the code in section (2.3). This experiment has seveal limitations:

- It only shows the signal strength does not change too much over linear networks, but what will happen if the signal undergoes a non-linear activation.
- It only shows that the signal strength does not change too much after one-layer. Does it assert that after more layers, the signal strength still remains nearly the same?
  - Last time we have shown that $\mathbb{E}(\|z^{\ell}\|^2) \approx \mathbb{E}(\|z^{\ell-1}\|^2)$. Therefore, it seems to be true that
    $$\mathbb{E}(\|z^{L}\|^2) \approx \mathbb{E}(\|z^{L-1}\|^2) \approx \cdots \approx \mathbb{E}(\|x\|^2)$$
  - However, we can run a simulation to verify the cases. If we set $L \leftarrow 100, d \leftarrow 10$, and run the following code in MATLAB:

    ```
    clear;
    L = 100;
    d = 10; % dimension for weight matrix W
    maxit = 1; % maximum iteration number

    x = ones(d,1);  norm0 = norm(x);
    for i = 1:maxit
        for l = 1:L
            W = randn(d,d)/sqrt(d);
            x = W*x;
        end
        rato = norm(x)/norm0
    end
    ```

    Then we find that the ratio of output signal strength over the input signal strength is below 10^{-3}.

We should give some more precise theoretical analysis.

1. Firstly we give a rigorous proof for that the signal strength after one-layer linear network keeps nearly the same for Xaiver Initialization. Consider the input vector $x = \mathbf{1} \in \mathbb{R}^d$ and after one-layer linear network, $z = Wx \in \mathbb{R}^d$ such that $W_{i,j} \sim \mathcal{N}(0, 1/d)$. Therfore,

$$z = \begin{pmatrix} \sum_{j=1}^d W_{1,j} \\ \sum_{j=1}^d W_{2,j} \\ \vdots \\ \sum_{j=1}^d W_{d,j} \end{pmatrix} \implies \|z\|^2 = \sum_{i=1}^d \underbrace{\left(\sum_{j=1}^d W_{i,j} \right)^2}_{\xi_i}$$

where $d \cdot \xi_i$ is the sum of the square of d i.i.d standard normal random variables, i.e., $d \cdot \xi_i \sim \chi^2(d)$. Therefore,

$$\mathbb{E}[\xi_i] = 1.$$

Moreover,

$$\begin{aligned} \mathbb{P}(|\xi_i - 1| \le \epsilon) &= \mathbb{P}(|\chi^2(d) - d| \le \epsilon \cdot d) \\ &\triangleq 1 - F((1-\epsilon)d, d) - (1 - F((1+\epsilon)d, d)) \\ &\ge 1 - ((1-\epsilon)e^{\epsilon})^{d/2} - ((1+\epsilon)e^{-\epsilon})^{d/2} \end{aligned}$$

where $F(x, d)$ denotes teh cdf of the random variable $\chi^2(d)$, and the last inequality follows from the *Chernoff bounds* on the lower and upper tails of $F(\cdot, d)$. Therefore, we conclude that the signal strength after one-layer is close to the original with high probability.

2. Then consider the signal strength after multi-layer linear network. Unrigorously, suppose that in each layer $\xi_i \approx 1 + \varepsilon$, and $\|z^\ell\|^2 = \|z^{\ell-1}\|^2(1+\varepsilon)$. Therefore, the final signal strength

$$\|z^L\|^2 \approx \|z^{L-1}\|^2(1+\epsilon) \approx \cdots \approx \|x\|^2(1+\varepsilon)^L$$

Here the parameter ε depends on d, i.e., the number of neuros in each layer. Roughly speaking, $\epsilon = \mathcal{O}(1/d)$. As a result,

$$\|z^L\|^2 = \|x\|^2(1 \pm \frac{1}{d})^L \approx \|x\|^2 e^{L/d}.$$

Remark 3.1. 1. The signal strength ratio $\|z^L\|^2/\|x\|^2 = \mathcal{O}(\exp(L/d))$, where L is the depth of the layers, d is the number of neuros in each layer. Therefore, L/d controls the width of the neural network. When L/d is very large, i.e., the neural network is very narrow, it's likely that gradient explosion/vanishing will happen.

2. For the toy example of fully connected linear network we have tried before, i.e., in Example (3.1), when $L/d > 10$, the gradient explosion/vanishing will happen; it works well if we set $L/d < 1/10$.

3. This theoretical analysis also depends on other factors such as input data and architecture. For instance, if we train CIFAR10/MINST using super-small L/d (e.g., $d = 3$million neurals in the widest layer, $L > 50$ layers, and L/d small), we still cannot train it well. This is different from our prediction that when L/d is small then one can train it well. It might be related to CNN, but could also be related to other issues such as landscape.

Bibliography For different neural network architectures such as CNN, we need to perform the similar experiments but with "fair" criteria. The paper (**Glorot10understandingthe**) checked the CNN architecture. No one have ever performed the similar verifications for SOTA architecture.

The paper (**NIPS2018_7338**) gives total regiorous proof for previous theoretical analysis based on the martingale theory:

- Failure mode 1: the signal strength normalized by the size in each layer scale in the final layer increases/decreases exponentially with the depth, i.e., $\mathbb{E}[M_L] \triangleq \mathbb{E}[1/d\|z^L\|^2] \to 0$ or ∞ as $L \to \infty$

- Failure mode 2: The empirical variance of the signal strength normalized by the size in each layer, say $\mathrm{Var}\{M_{1:L}\}$, grows exponentially with the depth. More precisely,

$$\mathbb{E}\left[\mathrm{Var}\{M_{1:L}\}\right] = \mathcal{O}\left(\exp\left(\sum_{i=1}^{L}\frac{1}{d_i}\right)\right)$$

In particular, if all d_i's are the same, then $\mathbb{E}\left[\mathrm{Var}\{M_{1:L}\}\right] = \mathcal{O}(\exp(L/d))$.

There is some gap between Ruoyu Sun's claim and the work in this paper. What Ruoyu Sun claimed is that the variance of M_L depends on $\exp(L/d)$; but this paper claims that the empirical variance of $M_{1:L}$ depends on $\exp(L/d)$. Nevertheless, the paper actually proved that the variance of M_L depends on $\exp(L/d)$ in their theorem, but they did not advertise this result in the abstract and introduction. In summary, this paper and others give a formal theory of why super-deep & super-narrow neural networks are hard to train.

3.3 General Activation

Now we discuss the answer to Question 2 raised at the beginning of this lecture.

Problem Setting Given input vector $X \in \mathbb{R}^{d\times 1}$ and random weight matrix $W \in \mathbb{R}^{d\times d}$. Define the output vector $z = \phi(Wx)$, where $\phi : \mathbb{R} \to \mathbb{R}$ is a given activation function. We are interested in the sufficient condition ensuring $\mathbb{E}[\|z\|^2] = \mathbb{E}[\|x\|^2]$.

3.3.1 The scalar-input one-layer case

Consider the case where $d = 1$ and $L = 1$. Then $x \in \mathbb{R}$ and $z = \phi(wx)$ with $w \sim \mathcal{N}(0, c)$. We want to choose c such that $\mathbb{E}[z^2] = x^2$. This problem reduces to solving a non-linear equation in terms of c:

$$x^2 = \mathbb{E}[z^2] = \mathbb{E}_w[(\phi(wx))^2] = \int (\phi(tx))^2 \,\mathrm{d}t \frac{1}{\sqrt{2\pi}} e^{-1/2t^2}$$

3.3.2 The vector-input one-layer case

Then consider the case where $d > 1$ and $L = 1$. First write down z explicitly in terms of x and w:

$$z = \phi(Wx) \implies \begin{pmatrix} z_1 \\ \vdots \\ z_d \end{pmatrix} = \begin{pmatrix} \phi(\sum_{j=1}^d w_{1j}x_j) \\ \vdots \\ \phi(\sum_{j=1}^d w_{dj}x_j) \end{pmatrix}$$

As a result,

$$\|z\|^2 = \sum_{i=1}^{d} \phi\left(\sum_{j=1}^{d} w_{ij}x_j\right)^2,$$

which implies

$$\begin{aligned}\mathbb{E}[\|z\|^2] &= \mathbb{E}\left[\sum_{i=1}^{d} \phi\left(\sum_{j=1}^{d} w_{ij}x_j\right)^2\right] \\ &= d \cdot \mathbb{E}\left[\phi\left(\sum_{j=1}^{d} w_{ij}x_j\right)^2\right] = \|x\|^2\end{aligned}$$

where the last inequality is because that $\left\{\phi\left(\sum_{j=1}^{d} w_{ij}x_j\right)^2\right\}_{i=1:d}$ are i.i.d. This is a single equation on scalar c, which is solvable.

3.3.3 The vector-input multi-layer case

We cannot apply the techniques similar as in the previous two cases. For instance, consider the case where $d = 1$ and $L = 2$. If we have

$$z^2 = \sigma(w^2\sigma(w^1x))$$

Then we know the pdf of w^1x for fixed x, but it's hard to know the pdf of $z^1 \triangleq \sigma(w^1x)$. It's even harder to know the pdf of $z^2 = \sigma(w^2z^1)$. Instead, for the linear activation case, we have

$$\mathbb{E}\xi_1\xi_2\xi_3 = \mathbb{E}\xi_1\mathbb{E}\xi_2\mathbb{E}\xi_3, \quad \text{provided that } \xi_{1:3} \text{ are independent}$$

However, the expectation $\mathbb{E}(\xi_1(\phi(\xi_2(\phi(\xi_3)))))$ is hard to compute.

Mean-field approximation The solution is to apply the mean-field approximation, the idea in which is to approximate intermediate variabeles by Gaussian random variables.

Proposition 3.1 (Informal Claim)**.** Suppose that the $(\ell-1)$-th layer has $d_{\ell-1}$ neurons, which is denoted as $h^{\ell-1}_{1:d_{\ell-1}}$. The variables for the pre-activation in ℓ-th layer, denoted as $h^\ell = W^\ell\phi(h^{\ell-1})$ (or $h^\ell_i = \sum_{j=1}^{d_{\ell-1}} W^\ell_{ij}\phi(h^{\ell-1}_j)$ for $i = 1 : d_\ell$) can be approximated by Gaussian

random variables, provided that $d_{\ell-1}$ is very large. As $d_{\ell-1} \to \infty$, the variable $h^{\ell}_{1:d_\ell}$ converge to Gaussain.

This method is first applied in the paper (**NIPS2016_6322**) to analysis the propagation of the variance of pre-activations. This technique is a novel application of the central limit theorem (**Bill86**):

Theorem 3.1 (Lyapunov Central Limit Theorem)**.** The sum of n independent random variables $X_1, \ldots, X_n$ converges to a Gaussian random variable as $n \to \infty$, provided that Lyapunov's condition is satisfied.

Now we apply the mean-field approximation technique in some examples, although the ∞-width neural network assumption is not satisfied:

Example 3.2. Consider the case where $d = 1$ and $L = 3$. We have

$$h^1 = w^1\phi(x), h^2 = w^2\phi(h^1), h^3 = w^3\phi(h^2)$$

with $w^{1:3} \sim \mathcal{N}(0, \sigma_w^2)$ and $x \sim \mathcal{N}(0, q_{\text{in}})$. In order to control the signal strength of h^3, it suffices to control its variance (why?).

By the hint in the note[1], we have

$$\mathbb{E}[h^1] = 0, \quad \text{Var}(h^1) = \mathbb{E}[\|wx\|^2] = \sigma_w^2 \int_{-\infty}^{\infty} \phi(t\sqrt{q_{\text{in}}})^2 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{t^2}{2}\right) \mathrm{d}t$$

Similarly, we have

$$q^2 = \sigma_w^2 \int_{-\infty}^{\infty} \phi(t\sqrt{q^1})^2 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{t^2}{2}\right) \mathrm{d}t,$$

where $q^i \triangleq \text{Var}(h^i)$ for $i = 1, 2$.

The general form for q^ℓ can be computed recursively:

$$q^\ell = T(q^{\ell-1}) \triangleq \sigma_w^2 \int_{-\infty}^{\infty} \phi(t\sqrt{q^{\ell-1}})^2 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{t^2}{2}\right) \mathrm{d}t$$

Remark 3.2. It's not true for the case $d = 1$, since h^ℓ are actually not Gaussian. In multi-dimension case, this statement becomes more

[1] Ruoyu Sun, Mean-field Approximation: Step-by-Step Approach

rigorous. For neural network with infinite-width, if $\mathbb{E}w_{ij}^{\ell} = 0$ and $\mathrm{Var}(w_{ij}^{\ell}) = \sigma_w^2/\text{fan-out}$, then

$$q^{\ell} = T(q^{\ell-1}; \sigma_w^2) \triangleq \sigma_w^2 \int \phi(t\sqrt{q^{\ell-1}})^2 \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{t^2}{2}\right) \mathrm{d}t \tag{3.1}$$

Example 3.3. We can use the Eq. (3.1) to find proper σ_w^2 such that $\{q^{\ell}\}$ does not explode or vanish.

- For linear neural network, Eq. (3.1) reduces to

  $$q^{\ell} = \sigma_w^2 q^{\ell-1}.$$

  Therefore, we choose $\sigma_w^2 = 1$, which is the Xavier initialization.

- For relu activation, Eq. (3.1) reduces to

  $$q^{\ell} = \frac{1}{2}\sigma_w^2 q^{\ell-1}$$

  Therefore, we choose $\sigma_w^2 = 2$, which is the Kaiming initialization.

- For other types of activation, e.g., $q^{\ell} = (q^{\ell-1})^2\sigma_w^2$, it's difficult to pick σ_w^2 only to get the desired result. In this case, we pick $\sigma_w^2 = 1$ and $q^1 = 1$. Note that $q^1 = \sigma_w^2\|x\|^2\frac{1}{d_0}$. One key message here is that to achieve the desired stability of signal propagation, one needs to properly choose the input strength q^0 such that q^1 is the fixed point of the propagation equation (3.1). We just gave a toy example that the fixed point is $q^1 = 1$, but in general, the fixed point $q^* = T(q^*)$ has no closed-form and needs to be computed by other methods.

3.4 Dynamical Isometry

The motivation is that for neural network with input $x \in \mathcal{X}$ and output $y = \mathcal{W}(x) \in \mathcal{Y}$, we want to have $\|x\| \approx \|y\|$.

Bibliography Dynamical Isometry has gain popularity in the research of deep learning. It is first raised in the paper (**DBLP**) to analysis the behavior of deep non-linear networks. Following the previous work,

the same authors also extends the dynamical isometry theory to a large number of activation functions (**Pennington2018TheEO**). This theory also has some applications in practical training. The paper (**pmlr-v80-xiao18a**) applies dynamical isometry theory to train 10000-layer vanilla CNN without tricks such as Batch Normalization or ResNet. The paper (**li2018on**) applies this theory to tranin a deep autoencoder without any other tricks as well. The paper (**08987**) applies this theory to train 10000-long LSTM. The Dynamical Isometry is the peak of the design of initialization. There are other examples for the smart design of initalization. The paper (**zhang2018residual**) proposes a *FixUp Initialization* scheme to train ResNet without Batch Normalization, which is based on another analysis.

3.4.1 Dynamical Isometry for Linear Networks

For linear network case, given $y = W^L \cdots W^1 x$, the goal is to ensure $\|y\| \approx \|x\|$. The previous technique is to choose W^ℓ to be a Gaussain matrix.

There is another perspective from singular values. It suffices to let singular values of W^ℓ to be close to 1. The simplest solution (**Saxe14exactsolutions**) is to set W^ℓ is an orthogonal matrix, provided that $d_0 = \cdots = d_L$, and the intuition is that the norm is orthogonal invariant.

Proposition 3.2 (Key Observation for Orthogonality)**.** If W^ℓ are orthogonal matrices for $\ell = 1, \ldots, L$, and $d \triangleq d_0 = \cdots = d_L$, then

$$\begin{aligned} \|z^\ell\| &= \|x\|, \quad \forall \ell \\ \|e^\ell\| &= \|e\|, \quad \forall \ell \\ \left\| \frac{\partial F}{\partial W^\ell} \right\| &= 2\|e\|\|x\|, \quad \forall \ell \end{aligned}$$

The paper (**Saxe14exactsolutions**) also runs simulation and finds that the orthogonal initialization in deep neural linear network, unlike the case for Gaussian initialization, enjoys *depth-independent* training time.

The goal that we want to achieve, i.e., all singular values of $W^L \cdots W^0$ are close to 1, is called the *dynamical isometry*.

Remark 3.3. There are two reasons that people prefer not to use orthogonal initialization. The first is that the initialization for CNN is totally different since it is not a fully connected neural network; the second is that the case for non-linear network will change a lot.

In the next lecture we will talk about the non-linear network. In particular, we will talk about the *DeltaOrthogonal* frequently used in tensorflow.

4

Three Tricks in Training of Neural Network

4.1 Reviewing

- Why Kaiming initialization is not good enough for deep narrow network (i.e., large L/d)?

  Answer: Error accumulation, i.e.,

  $$\text{error} \sim \exp(L/d)$$

- Besides random initialization, what else is possible?

  Answer: orthogonalization.

- Besides weight matrices $W^{1:L}$, what eles can be tuned at initialization?

  Answer: Input strength (size) $\|x\|^2$.

Motivation Up to today, there are three tricks to train a deep neural network:

1. Intialization;
2. Batch Normalization;

3. Residual units (architecture)

Back to 2012, there were 6-10 tricks to train the AlexNet. As time goes by, scientists found that many of these tricks are not necessary. In this lecture, we will talk about highlights in these three tricks.

4.2 Intialization: Dynamical Isometry

Up to now, Dynamical Isometry is the peak of initilization theory for deep learning. Given a neural network $\mathrm{NN} : \mathcal{X} \to \mathcal{Y}$ such that

$$y = \mathrm{NN}(x; W),$$

the goal is to choose W such that $\|y\| \approx \|x\|, \forall x$. More precisely, we want to make singular values of the input-output Jacobian J all close to 1. Following the notation setting in section (1.3), we have

$$J = \frac{\partial z^L}{\partial z^0} = \prod_{\ell=1}^{L} \frac{\partial z^\ell}{\partial z^{\ell-1}} = \prod_{\ell=1}^{L} (D^\ell W^\ell)$$

with $D^\ell = \mathrm{diag}(\phi'(\{h_i^\ell\}_{i=1}^{d_{\ell_i}}))$.

The key turns to the understanding of the distribution of eigenvalues of the matrix JJ^{T} (i.e., singular values of J). There is a classical theory for the eigenvalues of random matrix, which claims that its distribution is like a semi-circle:

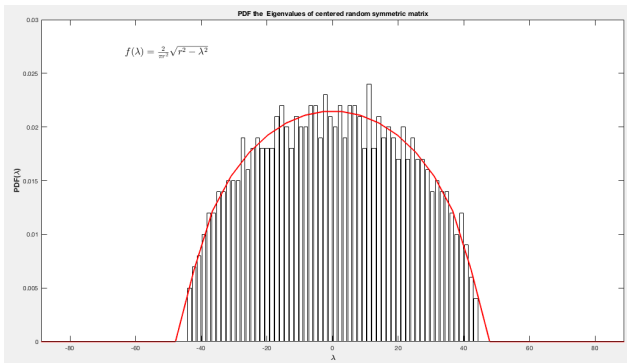


Figure 4.1: The pdf of eigenvalues for the scaled (symmetric) random matrix $\frac{1}{\sqrt{N}}W$ with $N = 1000$ and $w_{i,j}$ follows normal distribution.The simulation code is in https://www.mathworks.com/matlabcentral/fileexchange/46464-wigner-s-semicircle-law

Theorem 4.1 (Wigner's semi-circle law)**.** Given a symmetric matrix $W \in \mathbb{R}^{N\times N}$ whose entries $W_{i,j}, i \geq j$ are independent random variables, the asymptotic distribution (as $N \to \infty$) of the eigenvalues of W is like a semi-circle.

The semi-circle law is related to our problem: The Jacobian J is a random matrix, This semi-circle law is related to our problemm where the randomness comes from W. More precisely, it is a product of non-linear function of random matrices. In order to resolve the nonlinearity, we need to use the free probability(i.e., S-transform) tool to calculate the distribution of singular values of J.

The possibility of dynamical isometry The paper (**DBLP**) gives possibility for realizing dynamical isometry for deep *nonlinear* neural networks in different scenarios.

Table 4.1: Summarization of the results for the possibilities for realizing dynamical isometry for deep neural networks with different types of activations and initializations

	Scenarios	possibilities
Section (2.5.1)	Guassian Initialization +Relu Activation	No
Section (2.5.1)	Guassian Initialization +Hard-tanh Activation	No
Section (2.5.2)	Orthogonal Initialization +Relu Activation	No
Section (2.5.2)	Orthogonal Initialization +Hard-tanh Activation	Yes

In particular, for the last scenario, one can achieve dynamical isometry by tuning $\|x\|^2, \sigma_w^2, \sigma_b^2$ properly[1]. In this case, the deep neural network enjoys depth-independent training time.

The mechanism for dynamical isometry is to realize three goals simultaneously, i.e., relies on solving three types of equations:

- Feedforward propagation: keep variance

[1]Here $\sigma_w^2 \triangleq \frac{1}{d}\mathrm{var}(w_{i,j}^{\ell})$ and σ_b^2 refers to the variance of bias for pre-activation

- Backforward propagation: keep variance
- Eigenvalues of $J^{\mathrm{T}}J$ close to 1.

The realization relies on choosing $\|x\|^2, \sigma_w^2, \sigma_b^2$. The failure of first three scenarios in Table (4.1) is due to the loss of freedom of these variables.

Why "Orthogonal" is "Difficult" Before the paper (**pmlr-v80-xiao18a**) comes out, the orthogonal initialization for CNN is difficult, One reason is that the orthogonality for this architecture is not well-defined:

- For fully connected neural network, we say the weight matrix is orthogonal if $W^{\mathrm{T}}W = I$.
- For CNN, consider the pre-activation process, i.e., $y = W \otimes x$, it is unclear how to define the orthogonality for tensor product. To resolve this issue, the paper (**pmlr-v80-xiao18a**) defines the orthogonality as $\|y\| = \|x\|, \forall x$.

After designing the orthogonal weights for CNN, they are able to train 10000-layer-net for CIFAR10. This result indicates that initialization is enough for training ultra-deep neural network.

4.3 Batch Normalization

Motivation Previous data analysis knowledge tells us that the success of data processing also depends on the *input normalization* (preprocessing of data), even for linear regression. It's nature to do the similar thing for the training of neural networks.

Example 4.1. Review the linear regression problem. Given the data points (x_i, y_i) for $i = 1, \ldots, n$, one wants to select parameter w^* such that the quadratic loss is minimized:

$$w^* = \arg\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (w^{\mathrm{T}} x_i - y_i)^2$$

For the data matrix $X = [x_1, x_2, \ldots, x_n]$, one should always do the row normalization:

$$\begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \xrightarrow{\eta} \begin{bmatrix} \tilde{x}_1 & \tilde{x}_2 & \cdots & \tilde{x}_n \end{bmatrix}$$

where $\tilde{x}_i \triangleq \frac{x_i - \mu}{\sigma}$ with $\mu = \left[\frac{1}{n}\sum_i x_{i,j}\right]_j$ and $\sigma = \left[1/n \cdot \sum_i (a_{i,j} - \mu)^2\right]_j$. Here the minus and division operator is performed *component-wisely*.

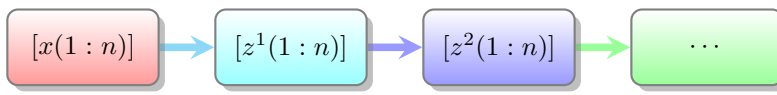
From the perspective of optimization, this opertion improves the *convergence speed*, since the operator η will reduce the condition number[2] of the data matrix.

From the experience of linear regression, we gain some knowledge for the training of neural nets:

- It is necessary to do pre-processing of the input.
- Besides, for each layer, we might have the same issue that the condition number for variables before activation is large or small.

Before (**Glorot10understandingthe**), people uses the layerwise-per training technique, which also brings up some problems such as the inefficiency of training. Therefore, we need other techniques to solve the issue.

Consider the neural network represented in the figure below:



where z^ℓ's are variables before activation. The goal is to make matrices

$$Z^\ell = \begin{pmatrix} z^\ell(1) & \cdots & z^\ell_1(n) \end{pmatrix} = \begin{pmatrix} z^\ell_1(1) & \cdots & z^\ell_1(n) \\ \vdots & \ddots & \vdots \\ z^\ell_d(1) & \cdots & z^\ell_d(n) \end{pmatrix}$$

become well-conditioned for each $\ell = 1, \ldots, L$, i.e., each row has zero mean and unit variance.[3]

To fully understand how the batch normalization achieves the row normalization for each data matrices Z^ℓ, consider the toy example presented below:

[2] $\kappa(X) = \lambda_{\max}(X)/\lambda_{\min}(X)$ denotes the condition number of X, see Prof. Luo's Note Lecture 2 for more detailed explanations.

[3] The i-th row denotes the data points for the i-th feature, and the row normalization makes these data points well-scaled

Example 4.2 (Toy Example)**.** Consider an aritificial problem defined as

$$\begin{array}{ll} \min_h & F(h) \triangleq \sum_{i=1}^n g(h_i) \\ \text{with} & \sum_i h_i = 0 \\ & \sum_i h_i^2 = 1 \end{array} \tag{4.1}$$

Combining with gradent descent method, there are at least 3 ways to deal with it:

Method 1: Pure Algorithmic Correction The intuitive way is that in each iteration one performs the gradient descent, and then project the new iterates within the constraint set. However, the projection into this constraint set could be diffucult due to its non-convexity. Therefore, we modify the projection step with the *normalization operator* η:

$$\begin{array}{ll} \eta(h_{1:n}) & \triangleq \frac{h_{1:n}-\mu}{\sigma} \\ \text{with} & \mu = \frac{1}{n}\sum_i h_i \\ & \sigma = \frac{1}{n}\sum_i \|h_i - \mu\|^2 \end{array}$$

The whole algorithm is presented below:

$$\begin{cases} h^{t+1/2} \leftarrow h - \nabla F(h) \\ h^{t+1} \leftarrow \eta(h^{t+1/2}) \end{cases}$$

Method 2: Constrained optimization Consider this problem as a constrained optimization problem, and thus it's natural to solve this problem from its dual (or solve primal and dual simultaneously), e.g., applying Lagrangian method or ADMM.

Remark 4.1. The method 1 solves the problem in an algorithmic way, i.e., modify the gradient project method in a heuristic way; The method 2 is more about reformulation, i.e., reformulate this problem from its dual and therefore solve the new convex problem instead. We observe that both of them don't work well in practice. The method 3 reformulates this problem in another way and we found it works well.

Method 3: New way of Formulation We reformulate this problem into unconstrained optimization, by substituting the constraint into the

objective function:

$$\min_h \quad F(\eta(h)) \tag{4.2}$$

Then the optimal solution of the origin problem (3.1) is tractable by setting $h^* = \eta(\arg\min_h F(\eta(h)))$. For this unconstrained optimization problem, we can apply the gradient descent to solve it.

Motivation for Batch Normalization (BN) The insights of BN is similar to that in Method 3. Consider a 2-layer neural network computing

$$e = F_2(F_1(x, W_1), W_2) \tag{4.3}$$

where F_1, F_2 are arbitary transformations, and the parameters W_1, W_2 are to be chosen so as to minimize the loss e. Learning W_2 can be viewed as if the inputs $z \triangleq F_1(x, W_1)$ are fed into the sub-network

$$e = F_2(z, W_2).$$

The goal for BN is to ensure the distribution of nonlinearity inputs (i.e., z) remains more stable during the training process, i.e., pick W_2, W_2 to minimize the loss function and within the constraint set

$$\mathbb{E}[x] = 0, \mathrm{Var}[x] = 1, \qquad \mathbb{E}[z] = 0, \mathrm{Var}[z] = 1.$$

From the experience of Method 3, it suffices to reformulate (4.3) by adding the *normalization* process before the non-linear activation in each layer.

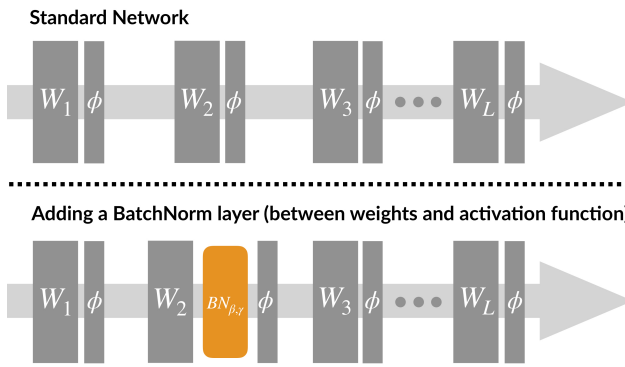


Figure 4.2: Adding the Bacth Normalization process in the *second* layer

The *normalization* process can be changed in different scenarios, e.g., for capsule-net, change it with the *clustering* process.

Gradient Computation if adding the BN The standard one-layer neural network can be described as a non-linear parametric function

$$z = g(Wu),$$

where g is the non-linear transformation, W is the weight matrix. By adding the BN, the standard function is replaced with

$$z = g(\mathrm{BN}(Wu)),$$

where the BN transform is applied independently to each dimension of $x = Wu$. We set $h = Wu$, then we show how to compute $\frac{\partial z}{\partial h}$:

1. Step 1: decompose the transformation from h to z into paths.

   Note that $z = z(h, \mu(h), \sigma(h))$. Therefore, the paths could be:

   $$\begin{aligned} &h \to z; && \text{provided that } \mu, \sigma \text{ are fixed} \\ &h \to \mu \to z; \\ &h \to \sigma \to z. \end{aligned}$$

2. Step 2: Apply the Chain Rule.

   $$\frac{\tilde{\partial} z}{\tilde{\partial} h} = \left.\frac{\tilde{\partial} z}{\tilde{\partial} h}\right|_{\mu,\sigma \text{ fixed}} + \frac{\tilde{\partial} z}{\tilde{\partial} \mu}\frac{\tilde{\partial} \mu}{\tilde{\partial} h} + \frac{\tilde{\partial} z}{\tilde{\partial} \sigma}\frac{\tilde{\partial} \sigma}{\tilde{\partial} h}$$

   in which the notion of generalized Jacobian is adpoted.

Remark 4.2 (Representation Power)**.** Re-consider the formula (4.1). We find that

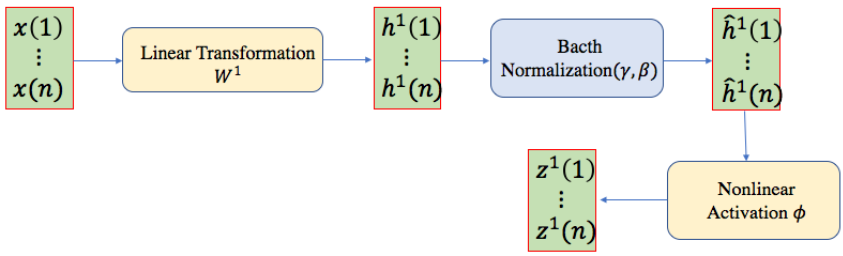
$$\{F(h) \mid h \in \mathbb{R}^n\} \supseteq \{F(\hat{h}) \mid \sum_i \hat{h} = 0, \sum_i \hat{h}^2 = 1\}.$$

Therefore, the $\min F(h)$ is not necessarily equivalent to $\min F(\eta(h))$. In order to resolve this issue, we need to "store" the representation power by scale and shift $\eta(h)$ in (4.2), i.e., introducing γ, β to form a new problem

$$\min_{h,\gamma,\beta} F(\gamma \cdot \eta(h) + \beta) \tag{4.4}$$

Now the set $\{F(h) \mid h \in \mathbb{R}^n\} = \{F(\gamma \cdot \eta(h) + \beta) \mid h \in \mathbb{R}^n, \gamma, \beta \in \mathbb{R}\}$.

In the remaining of this section, let's discuss more about applying BN in practice.



Remark 4.3. Batch Normalization is applied before the non-linear activation in each layer.

Remark 4.4. The function $\mathrm{BN}_{\gamma,\beta}(\cdot)$ is applied to each row of the matrix

$$\begin{pmatrix} h^1(1) & \cdots & h^1(n) \end{pmatrix}.$$

Remark 4.5. In practice, the BN is applied for mini-batch with batch size B. In other words, the total n inputs are separated into N batches, where each batch contains B inputs. The BN is performed on each single batch. This technique changes the problem form:

Standard Formulation	$\min_\theta \frac{1}{n}\sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$
Mini-Batch Formulation	$\min_\theta \frac{1}{N}\sum_{i=1}^{N} \tilde{\ell}(\tilde{f}_\theta(x_{i,1:B}), (y_{i,1:B}))$

Previously, the neural network is a single-input-single-output (SISO) function:

$$x_i \xrightarrow{f_\theta} \hat{y}_i$$

Now, with mini-batch BN, the neural network is a multi-input-multi-output (MIMO) function:

$$x_{i,1:B} \xrightarrow{\tilde{f}_\theta} \hat{y}_{i,1:B}$$

Remark 4.6. In each batch i, $\tilde{f}_\theta$ is also dependent on the parameter $\bar{\mu}_i, \bar{\sigma}_i$. Therefore, for the whole neural network $\tilde{f}_\theta(\bar{\mu}, \bar{\sigma})$, there could be an issue about how to select the parameters $\bar{\mu}$ and $\bar{\sigma}$.

Remark 4.7. For the problem (4.4), if we choose $\gamma = \|h\|$ and $\beta = \frac{1}{n}\sum_i h_i$, then the representation power is strong, but we may have difficulty in training, i.e, solving this optimization problem; if we choose $\gamma = 1$ and $\beta = 0$, it will lead to the weak representation power. Therefore, the choice of γ, β is a trade-off.

Remark 4.8. An reasonable mini-bacth size is needed to calculate μ, σ in each single bacth, i.e., $B \geq 16, 32, \ldots$. For non-ImageNet tasks, it will be an issue on how to choose B.

Remark 4.9. Besides BN, there are other types of normalization methods:

- Layer normalization; (which is standard in language processing)
- Instance normalization;
- Weight normalization;
- Group normalization; (**Wu_2018_ECCV**)
- Column normalization. The intuition is that there are lots of headache come from normalizing rows, it may be better to normalize columns instead. We skip the discussion for column normalization in this lecture.

4.4 ResNet

Recommended Reading: (**He2016res**).

Motivation It is intuitive that more layers for neural networks will lead smaller traning error. However, some numerical experiments show it is not true in general:

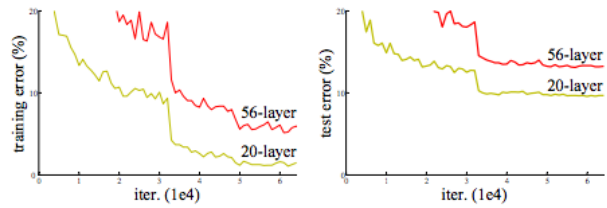


Figure 4.3: Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is also observed.

Let's analysis what is the most possible reason leading to this phenomena. Let's introduce few terminologies first. In data science, the testing error can be separated into three types of errors:

- Representation Error, which is related to the representation power of the fitting model.
- Optimziation Error, which is related to how well we minimize the loss function.
- Generalization Error, which is related to the over-fitting issue.

In Fig. 4.3 we observe that the training error for 56-layer network is larger. There are two factors related to the training error:

$$\text{Tranining error} = \begin{cases} \text{Representation Error} + \\ \quad \text{Optimziation Error} \end{cases}$$

It seems that the high training error for deeper neural network is because of the optimziation error, since deeper neural network admits stronger power in representation.

The optimization error for solving $\min_\theta F(\theta)$ using iteration formula can also be decomposed into two types of errors:

$$\underbrace{[F(\hat{\theta}) - F^*]}_{\text{Optimization Error}} = \underbrace{[F(\hat{\theta}) - F(\theta^\infty)]}_{\text{Convergence Error}} + \underbrace{[F(\theta^\infty) - F^*]}_{\text{Global-optimality Gap}}$$

Here θ^∞ is the solution we can obtain if given ∞ many iterations, i.e., the limit point where the optimiztion algorithm converge to.

- If $F(\theta^\infty) - F^*$ is large, one often needs reformulation of the original problem and smart initialization. Check (**frankle2018the**) with its comments (**WinNT**) in Zhihu.
- If $F(\hat{\theta}) - F(\theta^\infty)$ is large, one often improves his algorithm to accelerate, such as momentum-based acceleration.

It seems that the global-optimality gap is large for our training, and therefore some smart initialization is needed. Moreover, one may ask does 56-layer have more "representation power" than 20-layer networks? Not neceesarily, unless the extra 36-layer can be approximated to the identity operator.

It's known that the deep neural network is sensitivity to initialization, i.e., we can only travel a small region around the initialization. Therefore, it seems that the high training error for deep neural network is because that we may never travel have chance to travel to the identity operator for the extra 36-layer operations.

Solution The solution for solving this issue is to design the architecture to recover the identity operator. The Resnet is designed:

$$f_\theta(x) = \mathcal{F}(x, \{W_i\}) + x,$$

where x is the input vectors of the layers. More generally, for same-width network in the layer $\ell = 1, \ldots, L$,

$$z^\ell = \phi(W^\ell z^{\ell-1}) + z^{\ell-1},$$

Remark 4.10. The dimension of input and output does not always match. Two tricks may be needed:

- Perform a linear projection W_s by the shortcut connections to match the dimensions:

$$f_\theta(x) = \mathcal{F}(x, \{W_i\}) + W_s x.$$

- Use pooling to change the dimension.

Remark 4.11. ResNet uses 2-layer and 3-layer net as residual module.

Remark 4.12. Motivated by LSTM, the paper (**srivastava2015highway**) uses "gate" to resolve this issue, but their performance is no better than ResNet.

5

ResNet Initialization and Landscape Analysis

5.1 Reviewing

- Three major tricks for State-of-The-art (SoTa) Deep Learning Training:
  - *Initialization*;
  - *Batch Normalization* (BN);
  - *ResNet* (or other architectures).
- Key ideas of Batch Normalization:
  - Motivation: Reduce condition number by normalization.
  - Treat Normalization as ϕ, a non-linear transformation.
- Error Decomposition (*An* perspective from Prof. Ruoyu Sun):

$$\text{Testing Error} = \begin{cases} \text{Representation Error} \\ \text{Optimization Error} \end{cases}$$

$$\text{Optimization Error} = \begin{cases} \text{Finite-time Error} \\ \infty\text{-time Error} \end{cases}$$

- How to train a 10000-layer neural network with only one trick "*initilization*"?
  - Special Orthogonal Initial point (DeltaOrthogonal);
  - Based on the idea of Dynamical Isometry.

5.2 Initialization for ResNet

The architecture for classic ResNet is presented below:

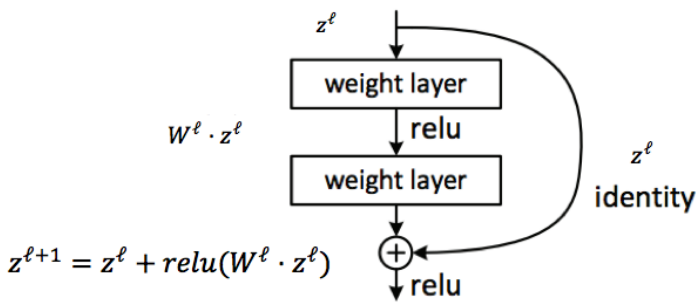


Figure 5.1: ResNet framework with Relu activation

How to initialize the ResNet architecture? Can we follow the previous knowleage for handling FNN? Let's give some analysis by first considering the simple linear network (with the same width).

Example 5.1. Let's consider the L-layer linear network with the same width d, i.e., the nonlinear activation is an identity operator. As a result, the output-input function is expressed as:

$$y = (I + W^L) \cdots (I + W^1)x, \quad \text{where } W^\ell \in \mathbb{R}^{d \times d}$$

By the Lecture 2 knowledge, we know that Xavier initialization works for fully connected linear neural networks. Now we perform the simulation of Xavier initialization for ResNet. The matlab code for the toy example where $L = 10$, $d = 100$, and input x is all-one vector, is presented below.

```
clear;
L = 10;
d = 100; % dimension for weight matrix W
maxit = 10; % maximum iteration number

x = ones(d,1);  norm0 = norm(x);
for i = 1:maxit
    for l = 1:L
        W = randn(d,d)/sqrt(d);
        x = W*x + x;
    end
    rato = norm(x)/norm0
end
```

Unfortunately, we find that $\|y\|/\|x\| \approx 10^{14}$ in this case. Now the question is the following:

1. Why does (**He2016res**) succeed?

   It seems that combining the trick *Batch Normalization* saves him

2. Why does our simulation fail?

   In previous toy example, Xavier initialization works when the output in each layer is the multiplication of the input with a Gaussian matrix, but in our example, $(I + W^\ell)$ is not Gaussian.

Therefore, we need to re-derive the whole *initialization theory* for ResNet, i.e., for what kind of W^ℓ, we have $\|y\|/\|x\| \approx \mathcal{O}(1)$?

1. Choose $W^\ell = \mathcal{N}(0, 1/d) - I$, then the original case for linear FNN is recovered, but we no longer enjoy the advantage of the new architecture.

2. Consider the case where $d = 1$ first, i.e., the output-input function is expressed as

   $$y = (1 + w^L) \cdots (1 + w^1)x$$

   It's feasible to choose $w^\ell = 0$ for all ℓ, but it does not have much representation power, since any point near this initial point will

be strongly attracted to it.[1] It's reasonable to assume that w^ℓ follows Gaussian distribution, i.e., $w^\ell = \mathcal{N}(0, \cdot c)$. The question turns to the choice of c. In this case,

$$\mathbb{E}[\|y\|^2] = (1+c)^L \|x\|^2$$

Therefore, we should choose $c = \frac{1}{L}$, which implies $\mathbb{E}[\|y\|^2] = \mathcal{O}(\|x\|^2)$.

For more general d, we should make $W_{i,j}^\ell = \mathcal{N}(0, \frac{1}{d} \cdot \frac{1}{L})$. The proof outline is as follows:

- Consider how the term $\mathbb{E}[\|y\|^2]$ scales with $\|x\|^2$. Observe that

$$\mathbb{E}[\|z^\ell\|^2 \mid z^{\ell-1}] = (z^{\ell-1})^{\mathrm{T}} \left[\mathbb{E}(I + W^\ell)^2\right] (z^{\ell-1})$$

- It's easy to show that when $W_{i,j}^\ell = \mathcal{N}(0, \frac{1}{d} \cdot \frac{1}{L})$,

$$\mathbb{E}(I+W)^2 = \mathbb{E}(I + 2W + W^2) = I + \mathbb{E}W^2 = (1 + 1/L)I.$$

- Therefore,

$$\mathbb{E}[\|y\|^2] = (1+1/L)^L I \cdot \|x\|^2 = \mathcal{O}(\|x\|^2).$$

Bibilogrphy In practice, people notice that ResNet performs much better than standard architectures when networks are very deep. The paper (**Balduzzi2017**) gives (partial) explanations for this phenomenon, and claims that one reason is that the gradients in ResNet (with BN) are far more resistant to shattering, which decays sublinearly. Then this paper proposes a new initialization scheme accordingly, which outperforms the classic He-initialization for standard architectures.

Scaling of the Residuals The formal analysis in the paper (**Balduzzi2017**) is as follows. Consider a (variant of) ResNet[2] framework for Batch Normalization Disabled case:

$$z^{\ell+1} = \alpha(z^\ell + \beta \cdot W^\ell \cdot \mathrm{relu}(z^{\ell-1}))$$

where α and β are rescaling factors.

[1] The formal definition of the strong attraction is presented in the paper (**Zhang2000**).

[2] There are several variants of classic ResNet.

1. With classic initilization without batch normalization trick, set $\alpha = \beta = 1$, then the variance of the gradient at $z^\ell[i]$ is 2^L.

2. α-scaling: A solution to the exploding variance of resnets is to rescale layers $\alpha = 1/\sqrt{2}$, then $\mathrm{Var}(z^\ell[i]) = 1$.

3. β-scaling: In practice, α-rescaling is not used. Instead, combining the bacth normalization trick and β-scaling[3] gives $\mathrm{Var}(z^\ell[i]) = \beta^2(L-1)+1$. Furthermore, when $\beta = 1/\sqrt{L-1}$, we see that the variance keeps constant:

$$\mathrm{Var}(z^\ell[i]) \equiv 2.$$

Is normalization fundamental? It's believed that normalization trick is fundamental in state-of-the-art training. The paper (**zhang2018residual**) challenges this belief by proposing *fixed-update initialization* scheme on *ResNet* to achieve state-of-the-art performance in image classification and machine translation. This initialization is motivated by solving the gradient explosion/vanishing problem at the beginning of training via properly rescaling a standard initialization. This is the only work that achieves such good results without the normalization method

This work is amazing for two reasons:

1. Batch Normalization can be annoying, since its practical implementation can have many bugs, especially for ResNet.

2. It shows the importance of basic logic, i.e., the combination of previous work could have gained more advantages. The paper (**Glorot10understandingthe**) first proposes Xavier-initialization; and the same authors propose Relu function in (**glorot2011relu**). Combining these two tricks, the Kaiming initialization is proposed in (**DDR2919332**); and the same authors propose ResNet in (**He2016res**). Moreover, this paper proposes the Fixup initialization by combining these two tricks that enjoy more advantages.

The theme of this course so far: discussion on how the theory shaped the current practice.

[3]It's suggested in (**Szegedy2016Inceptionv4IA**) that $\beta \in [0.1, 0.3]$

5.3 Landscape of Neural-Nets

Motivation We are interested in when and why the non-convexity is not a big issue for the training of neural-nets. To answer this question, recall that the Optimization Error is decomposed into two kinds of errors:

$$\text{Optimization Error} \begin{cases} \text{finite-time error} \\ \infty\text{-time error} \end{cases}$$

It's believed that the major issue is due to the ∞-time error, since in practice most algorithms do converge in a reasonable time. The ∞-time error is about the optimality gap between the global minima and the local minima that the algorithm has converged, which is related to the landscape of the loss function for the training of neural-nets. This lecture will talk about both positive and negative results about the landscape of the loss function.

5.3.1 Positive Result: Linear Network has nice landscape

Consider the loss function for linear neural network with depth L:

$$F(\theta) = \|y - W^L \phi(W^{L-1}(\phi(\cdots W^1 x)))\|^2$$

The non-convexity of this loss function comes from the multi-layer and the non-linear activation. The case for $L = 2$ reduces to the Matrix Factorization problem:

$$F(\theta) = \|Y - W^2 W^1 x\|_F^2.$$

The landscape for matrix factorization is well-studied:

Scalar Case Analysis The loss function for the scalar case is given by:

$$F(u, v) = (1 - uv)^2, \qquad u, v \in \mathbb{R}.$$

The 3-D plot for this loss function is in Fig. 5.2.

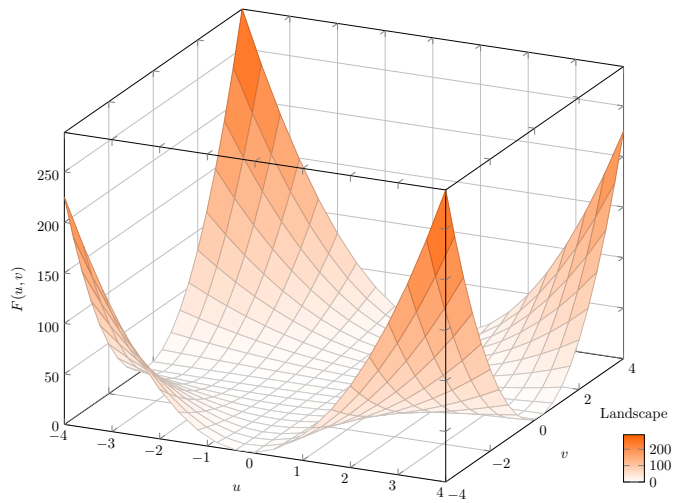


Figure 5.2: 3-D plot for the loss function $F(u,v) = (1-uv)^2$

From the landscape we can see that every local minima is a global-minima, i.e., there is no bad local minima. We give a proof for this claim:

1. Step 1: By taking gradient equal to zero, we find stationary points must satisfy
$$\text{either } uv = 1 \text{ or } u = v = 0$$

2. Step 2: It suffices to show that $(u,v) = (0,0)$ is not a local-minima. Note that
$$F(\epsilon_1, \epsilon_2) = (\epsilon_1\epsilon_2 - 1)^2 < 1$$
When $(\epsilon_1, \epsilon_2) > 0$ are sufficiently small, we find the function value decreases, i.e., $(0,0)$ is not a local-minima.

Remark 5.1. However, starting from $(u,v) = (0,0)$, the gradient descent gets stuck. Is this finding bad enough? Actually not. The paper (**pmlr-v49-lee16**) shows that gradient descent, if converges, it only converges to a local minimizer, almost surely with random initialization. The recent trend for non-convex optimization is not satisfied with the

convergence to first order stationary point[4], but the convergence to second order stationary point.[5] The reason is that empirically second order stationary point is as good as global minima. The tutorial (**ISIT2019**) in ISIT2019 gives summarization on the recent progress in non-convex optimization, which is highly recommended to read.

However, for people working on optimization, focusing on second order stationary point is not good enough, since high-order saddle points do exist for deep neural-nets. Now let's focus on the second order stationary point only.

Proposition 5.1. Consider the function $F(u) = \|M - uu^{\mathrm{T}}\|_F^2$, where M is PSD, any SOSP of this function is global-minima.

Proof. 1. Step 1: Check the gradient:

$$\nabla_u F = 4(M - uu^{\mathrm{T}})u = 0 \implies Mu = (u^{\mathrm{T}}u)u$$

Therefore, if u is SOSP, then $(\|u\|^2, u)$ is an eigen-pair of M.

2. Check the Hessian of the SOSP u:

$$\nabla^2 F(u) = 4(uu^{\mathrm{T}} + \|u\|^2 I - M) \succeq 0.$$

Combining these steps, we can show that $\|u\|^2 = \lambda_{\max}(M)$, i.e., u is a global-minima. □

Re-thinking Convexity From this proof we can partially answer why many people work on convex optimization, and why non-convex optimization is not scary:

1. By sub-gradient inequality, we can show that FOSP together with convexity implies global-minima. Therefore, it suffices to design algorithms searching for FOSP.

2. Define $G \triangleq \{\text{SOSP implies global-minima}\}$. We find many instances belong to the set G:

$$\{\text{convex problems}\} \subseteq G, \qquad \{\min_u \|M - uu^{\mathrm{T}}\|_F^2\} \subseteq G.$$

Therefore, for non-convex problems belonging to the set G, it suffices to design algorithms searching for SOSP.

[4] ϵ-First order stationary point (FOSP) means $\|\nabla f(x)\| \le \epsilon$

[5] Second order stationary point (SOSP) means $\nabla f(x) = 0$ and $\nabla^2 f(x) \succeq 0$.

Bibliography The paper (**BALDI198953**) shows that any SOSP for the 2-layer linear network quadratic loss function is global-minima under mild conditions; the paper (**NIPS2016_6112**) shows that any SOSP for the deep linear network quadratic loss function is global-minima under mild conditions. There are many later works extending to other loss functions. Up to now, we find that multi-layer may not be an issue.

5.3.2 Negative Result: Nonlinearity doesn't necessarily imply Global-optimality for SOSP

Now consider the non-linear activation. For the 1-dimension case, suppose the loss function $F(w) = (y - \phi(wx))^2$. W.l.o.g., $x = y = 1$, which follows that

$$F(w) = (1 - \phi(w))^2.$$

We are interested in whether all SOSP are global-minimas.

Relu Activation Now we draw the landscape of $F(w)$ if $\phi(w) = \max\{w, 0\}$:

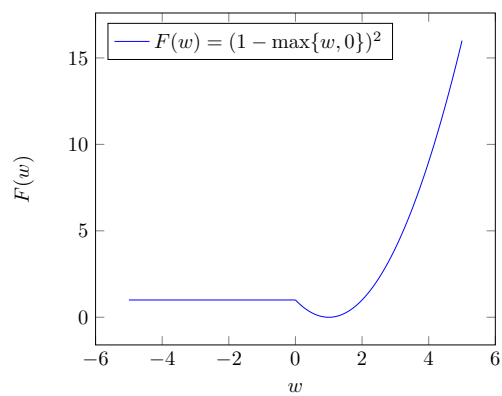


Figure 5.3: 2-D plot for the loss function $F(w) = (1 - w_+)^2$

We find that for $w < 0$, w is still a SOSP, but no longer a global-minima. Let's try other kinds of activation functions.

- When $\phi(w) = w^2$, we find SOSPs are still the global-minima:

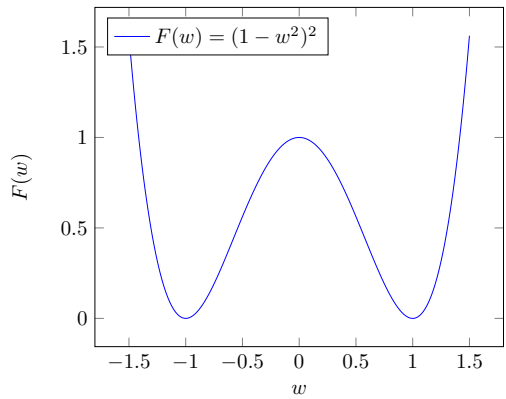


Figure 5.4: 2-D plot for the loss function $F(w) = (1 - w^2)^2$

- When $\phi(w) = \text{sigmoid}(w) = \frac{1}{1+e^{-w}}$, we find SOSP is still the global-minima:

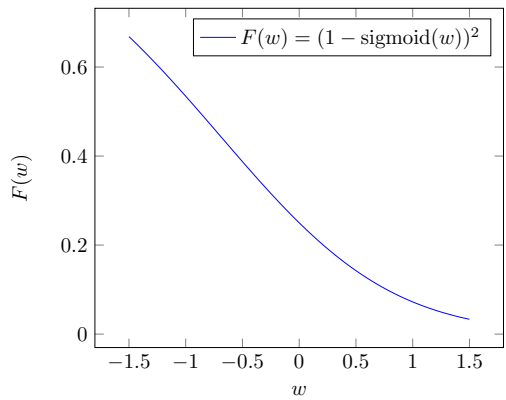


Figure 5.5: 2-D plot for the loss function $F(w) = (1 - \text{sigmoid}(w))^2$

Therefore, we conclude that the landscape of loss function is fine for most non-linear activations. but ReLu is not good in terms of landscape. Prof. Ruoyu Sun suggests that we can try Leaky Relu or Softplw if Relu fails (e.g., for GAN training).

In the next lecture we will try the sum of loss functions, i.e.,

$$F(w) = (y_1 - \phi(w)x_1)^2 + (y_2 - \phi(w)x_2)^2.$$

The question is that if two functions both have good landscapes, does the summation still have a good landscape?

6

Landscape Analysis and Representation

6.1 Reviewing

1. For ResNet, what should be the right initialization?

   β-scaling: scale the variance of the weight matrix W by $1/L$.

   Remark 6.1. Summarization for initialization tricks of the weight matrix W^ℓ:

   - Gaussian random W^ℓ
   - Orthogonal W^ℓ
   - For W^ℓ of the form Identity + Gaussian matrix H, scale the variance of the Gaussian matrix by $1/L$.

   Remark 6.2. The insights behind the hw1, question 5 are close to the initialization tricks for ResNet. In order to gurantee $\|z^L\|/\|x\| = \mathcal{O}(1)$, the trick for the ResNet is to make $\mathbb{E}[(I+H)^L] = \mathcal{O}(I)$; In order to gurantee $\mathbb{E}[\|Wx\|^2] = \mathbb{E}[\|x\|^2]$, it suffices to make $\mathbb{E}[WW^{\mathrm{T}}] = I$, or more specifically,

   - W and x are independent; $W_{i,j}$'s are independent;
   - $\mathbb{E}W_{i,j} = 0, \forall i,j$; $\sum_j \mathbb{E}(W_{i,j}^2) = 1$.

2. When does the non-convexity not scary?

   When the objective funciton has no sub-optimal local minima, i.e., each second order stationary point is global minima.

3. Under which condition, a deep neurel-net loss function has no sub-optimal local minima?

   - Mutli-layer is not an issue.
   - Most non-linear activation functions seems not an issue; However, for ReLU function, there do exists sub-optimal local minima. Therefore, the non-linearity sometimes does cause an issue.

Outline

- Non-linear neural-nets landscape analysis
- Universal Approximation Theorem

6.2 Landscape analysis for non-linear neural-nets

6.2.1 Negative Result: The sum of two good-landscape function have good landscape

A function with good-landscape means that there exists sub-optimal local minima. It's reasonable to think that the sum of two good-landscape function has a good landscape, since convex functions have good landscape, and the convexity holds under summation.

Unfortunately, the paper (**NIPS19951028**) gives counter-examples for this statement. Consider the error functions

$$E_1(w) = (y_1 - \phi(wx_1))^2, \qquad E_2(w) = (y_2 - \phi(wx_2))^2,$$

then it's possible that the error function $E \triangleq E_1 + E_2$ contains the local minimas of E_1 and E_2. In this way more local minimas can be produced, which may lead to bad landscape.

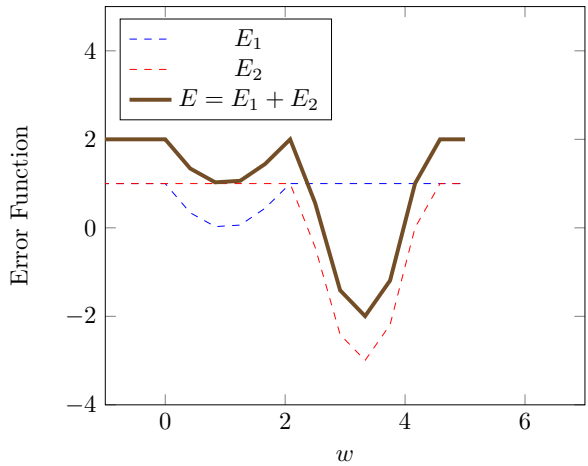


Figure 6.1: Illustraion of a counter-example for 1-dimension case

We made the assumption that the loss function is bounded[1], then the paper (**NIPS19951028**) further shows that the number of local minima of the error function grows can grow exponentially in the dimension.

Theorem 6.1 (Theorem 3.4 in **(NIPS19951028)**). Let ϕ and $\ell(\cdot, \cdot)$ satisfy the assumption; then for all $n \geq 1$, there exists data $\{(x_i, y_i)\}_{i=1}^n$ such that

$$F(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \phi(w x_i))$$

has $\lfloor n/d \rfloor^d$ distinct local minima.

This seems a negative result, but we consider adding some extra conditions to make it positive. First introduce the notion of *minimum-containing* set:

Definition 6.1. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuous function. Then an open and bounded set $U \in \mathbb{R}^d$ is called a *minimum-containing* set for f if for each w on the boundary of U, there is a $w^* \in U$ such that $f(w^*) < f(w)$.

[1] but ReLU does not satisfy this assumption

Graphically speaking, the minimum-containing set for a loss function is its "hole" part. In Fig. 6.1 we can see that E_1 and E_2 has different holes, which makes E has two distinct holes, which results in two distinct local minima.

The question is that does this phenomena happen frequently? In other words, is it possible to add mild (practical) conditions to eliminate extra *minimum-containing* set?

Theorem 6.2 (Theorem 5.1 in (**NIPS19951028**))**.** Let ϕ and ℓ satisfy the previous condition, and further assume that ϕ is monotone and ℓ is qusi-motonone, i.e., $L(y, y + r_1) \leq L(y, y + r_2)$ for $0 \leq r_1 \leq r_2$ or $r_2 \leq r_2 \leq 0$. Given a sequence of data $\{(x_i, y_i)\}_{i=1}^n$, assume that there exists parameter w such that $\phi(wx_i) = y_i$ for all i. As a result, there is only one minimum-containing set in the loss function $F(w)$.

Remark 6.3. In the 1-layer neural net with non-linear activation, we can see that the loss function has a good landscape only if each dataset is *realizable* [2], i.e., there exists parameter w such that $\phi(wx_i) = y_i$. The intuition is that in this case each component of the loss function share the same minimizer w^*. Therefore, the necessary condition for a loss function to have a good landscape is that representation power of the neural net is "enough".

6.3 Over-Parameterized Networks

It's a common sense that deep (over-parameterized) networks are effiective descriptors for our physical world. However, it requires long computation time, large storage space and otherwise. Due to the limited resource, it is popular to do the network pruning of a large network to get a reasonable effiective descriptor. See (**frankle2018the**) and (**NIPS2015_5784**) for related work.

[2] This condition is recently also called the interpolation property

However, it is not effiective to train a small network directly. One possible reason is that bigger networks may have better landscape. The evidence is that training larger network is in general "easier" than smaller one in practice.

Does Current Neural-net have too many parameters? This is not clearly understood now. Prof. Ruoyu Sun makes an analogy from the function fitting example. To fit an underlying function $\mathbb{R}^d \to \mathbb{R}$ given n samples, tuning n parameters are enough. However, to fit an underlying function $\mathbb{R}^d \to \mathbb{R}^{d_y}$ given n samples, it seems at least we need $n \cdot d_y$ samples. From this analogy, we infer that given n samples, each layer has n degree of freedom, and we call the phenomena that, training neural network with more than n neuros each layer, the *over-parametrization*.

Bibliography There are three classical works on the over-parametrization issue of neural networks before 2000. The paper (**BALDI198953**) simply shows that the landscape for the loss function of 1-hidden layer linear neural network is good; following this work, however, the paper (**NIPS19951028**) shows that adding non-linearity activation can create many bad local minima; suprisingly, (**410380**) shows that under the assumption of over-parametrization, 1-hidden layer nonlinear neural network has good landscape. However, Prof. Ruoyu Sun claims that the statement from this paper is wrong. He gives extension work in (**ruoyusun2018**).

Let's discuss the work (**410380**) in detail. Consider quadratic loss for 1-hidden layer nonlinear neural network under the assumption that the number of hidden neuron in each layer is more than the number of samples:

$$\min_{W_1, W_2} \|Y - W_2\sigma(W_1 X)\|_F^2$$

To understand thir work, define the following properties:

- Property [PT]: Starting from any initial point, there exists a *small* perturbation Δ and a strictly decreasing path from $\theta + \Delta$ to a global minima

- Property [P]: Starting from any initial point, there exists a strictly decreasing path from θ to a global minima, which further implies that there is no bad sub-optimal local minima.

Their work essentially shows the over-parameterized problem has the property [PT] instead of the property [P]. The paper (**ruoyusun2018**) is the first one that finds this mistake. We can easily see that hte property [PT] does not necessarily imply no bad local minima exist by considering the counter-example in the Figure 6.2.

In fact, the property [PT] only implies that no suboptimal strict local minimum exists, but suboptimal local minima can possibly exist. We say the existence of strict local minimum as the existence of bad basin:

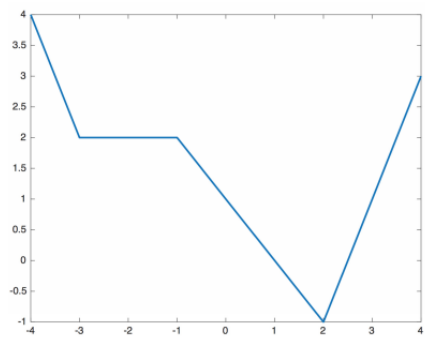


Figure 6.2: No bad basin

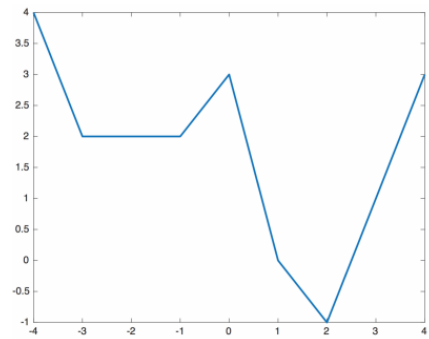


Figure 6.3: Example of bad basin

What we really need to worry about is the existence of bad basin in our loss function, i.e., exitence of strict local minima. However, such cases are rara in neural-nets due to its symmetry. For instance, matrix factorization is one of the cores in the deep learning training:

$$\min_{X,Y} \|A - XY\|_F^2$$

As long as we pick the solution pair (X, Y), the solution pair $(XT, T^{-1}Y)$ admits the same objective value for any orthogonal matrix T.

6.3.1 Empirical Evidence for Landscape

There is some evidence that large neural nets have nice landscape, but whether it is true, or how this can help us design better netowkr are still open problems. Since these results are empirical, the details are skipped, but only few works are listed.

Bibliography The paper (**43404**) examines that, on a straight path from initialization to solution, a variety of state-of-the-art neural networks never encounter any significant obstacles (basins). The paper (**NIPS2018_8095**) and (**Gotmare2018**) empirically finds that the optima of neural network loss functions are connected by simple curves over which training and test accuracy are nearly constant, which is called the *Mode Connectivity* phenomena. These empirical findings may potentially help us understand the landscape of loss functions, and the insights for robustness during training.

Remark 6.4.

1. The landscape for large neural network is nice, but it is not the case where each local-minima is global minima.
2. It would be helpful to analysis the landscape from geometric point of view, and we hope to generalize the empirical findings to theory.
3. There are recent results in other applications of neural network, such as reinforcement learning (Google Brain is working on it), GAN (**DBLPkarol**), and otherwise.

6.4 Representation Power

Finally, we give some quick introduction to the representation power of neural network.

Motivation Suppose we have a bunch of points in a unit square, and we have two classes, inside the circle is class one, outside the circle is class two. The goal is to build a classifier to classify these two classes, then it is never possible to apply linear classifier to get positive results, since linear classifier does not have strong representation power. We can formalize the notion of representation power with math.

Formulation of Representation Power

- Given a target domain $\mathcal{D} \subseteq \mathbb{R}^d$, which is usually assumed to be *compact*. For simplicity suppose that $\mathcal{D} = [0,1]^d$.
- Given a target function $f(x) \in \mathcal{C}(\mathcal{D})$.
- Given a candidate family $\mathcal{F}$:

$$\begin{aligned}\mathcal{F} &= \{f \mid f = v^{\mathrm{T}}\phi(wx+b), v \in \mathbb{R}^{1\times m}, w \in \mathbb{R}^{m\times b}, b \in \mathbb{R}^m, \text{some } m \in \mathbb{N}\} \\ &= \text{span}\{\phi(\langle w, x\rangle + w_0), w \in \mathbb{R}^d, w_0 \in \mathbb{R}\}\end{aligned}$$

- We say that $\mathcal{F}$ represents f if for any ε, there exists $f \in \mathcal{F}$ such that $\|f - g\| \leq \varepsilon$. In other words, the candidate family $\mathcal{F}$ has *enough representation power* w.r.t. the target function f.

Sufficient Condition for Enough Representation Power Few sufficient conditions given for building the enough representation power of F:

- When $\phi(\cdot)$ is a bounded, non-constant and continuous function, then $\mathcal{F}$ can represent any continuos mapping f (**Hornik1991**).
- If $\phi(\cdot)$ is bounded, non-constant, then $\mathcal{F}$ can represent any function f in $\mathcal{L}^p(\mu)$. (see Stone-Weierstrass Theorem for detail in real analysis note (**jiewang2019**)).
- ReLU is not a bounded function, but we can still show that the ReLU activation has enough representation power by some easy-following argument.

In next lecture, we will give some introduction to GAN.

7

Representation and GAN

7.1 Reviewing

1. When do neural-nets have bad local-minima?

   - Classical results in (**NIPS19951028**) show that for unrealizable case, non-linear activation can easily create bad local minima;
   - Recent results in (**ruoyusun2018**) show that even for over-parameterized case (thus also realizable), the bad local minima do exist for a class of non-linear activations.

2. People claims that "over-parametrization" smooths the landscape, any rigorous result in this claim?

   There are a few rigorous results for this claim. For instance, (**ruoyusun2018**) shows that the loss function of over-parametrized networks has no bad basin (or more precisely, it is a "weakly global" function); for many smooth enough activations, over-paramterized networks satisfy a stronger geometrical property PT (i.e. at any point, after a tiny generic perturbation, there is a strictly decreasing path from the perturbed point to a global minimum).

3. How to empirically check nice or bad landscape for any problem (continuous optimization)?

   Check values along paths connecting interesting points

4. When does the neural-nets have enough representation power?

   The most important factor is the activation function. It should be bounded and non-constant; although ReLU is not bounded, it also makes the neural-nets have enough representation power; but linear/quadratic activation does not.

7.2 Representation: depth separation

7.2.1 A simple proof of threhold activation has enough representation power

Consider the dimension $d = 1$ first. The non-linear activation is $\phi(t) = 1\{t \geq 0\}$. It suffices to show that

$$\overline{\text{span}\{\phi(at+b)\}} = \mathcal{C}(\mathcal{D})$$

for any compact domain $\mathcal{D} \in \mathbb{R}$. Define the pulse function $\psi(t) = 1\{0 \leq t < 1\}$, which can be expressed as $\psi = \phi(t) - \phi(1-t)$. It suffices to use the pulse function to approximate any continuous function. The general idea is shown in the Figure. 7.1.

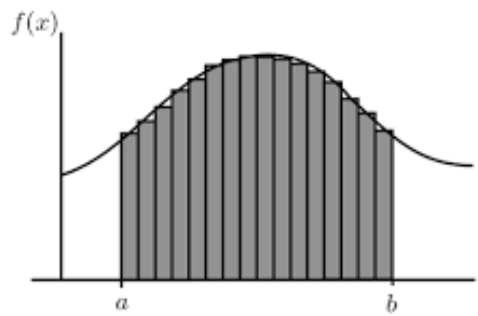


Figure 7.1: The pulse function can approximate any continuous function.

We can see that the insights are very similar to those in Riemann integration, and the proof follows the similar idea as well (**jiewang20191**).

Proof. Suppose that $\mathcal{D} = [0, 1]$ and our target function $f^* \in \mathcal{C}[0, 1]$. The continuity of f^* together with the compactness of $\mathcal{D}$ implies that the function f^* is uniformly continuous, i.e., $\forall \varepsilon > 0$, there exists δ such that for any $|x - x^*| < \delta$,

$$|f - f^*| < \varepsilon.$$

Pick a partition

$$\mathcal{P} = \{b_0 := 0, b_1 = h, b_2 = 2h, \ldots, b_K := Kh := 1\}, \quad \text{with } \frac{1}{K} < \Delta.$$

Therefore, define the approximation function

$$f(x) = \sum_{i=1}^{K} a_i \psi\left(\frac{x - b_i}{b_{i+1} - b_i}\right) \in \text{span}\{\phi(at + b)\},$$

where $a_i \triangleq f^*(b_i), i = 0, \ldots, K - 1$. Then it's easy to verify that $|f(x) - f^*(x)| < \varepsilon$ for any $x \in [0, 1]$. □

Remark 7.1. The first step in the proof explains why we need to define the *compact* domain.

Bibliography Then we discuss the representation power for other kinds of activations. For sigmoid function $\phi(t) = \frac{1}{1+e^{-t}}$, it suffices to show that it can approximate the threshold function very well; for other types of functions such as switch function, some techniques from function analysis are needed. The paper (**3561150**) shows that the sigmoidal-type activation has enough representation power by using arguments from real analysis; the paper (**Barron1994**) further gives an mean integrated squared error between the estimated network and a target function f, in terms of number of neurons and the input dimension; the Kolmogorov–Arnold representation theorem actually has solved this problem by using that every multivariate continuous function can be represented as a superposition of continuous functions of one variable, which is also related to Hilbert's thirteenth problem. The proof in this representation theorem contains the multi-resolution idea, and VCG/Receptor has the similar idea.

7.2.2 Depth Separation (Analysis for ReLU Activation)

It's a common belief that *deep* neural network usually gains better performance. We want to analysis this claim from the perspective of representation power. To show the power of depth, one way is to construct a function represented "*deep*"-net, then show this function is difficult to be represented by shallow networks.

1. Consider a function ψ frequently studied in the dynamical systems literature, which can be represented with the ReLU activation ϕ:

$$\psi(x) = \phi(2\phi(x) - 4\phi(x - 0.5)).$$

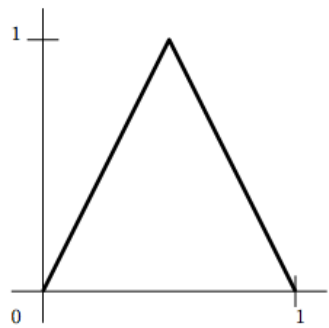


Figure 7.2: The function ψ, which has one "peak"

2. Construct a function $f^*(x) = \psi^{(L)}(x)$, a the composition of L ψ functions, which has 2^{L-1} peaks. See $\psi^{(2)}$ for instance:

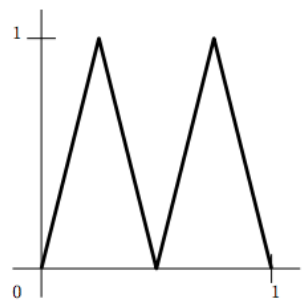


Figure 7.3: The function $\psi^{(2)}$, which has two "peaks"

It suffices to show $f^*(x) = \psi^{(L)}(x)$ can be represented by deep neural-nets, but it is difficult to be represented by a shallow network, i.e., we need $\mathcal{O}(2^L)$ neurons of a shallow network for representation. The intuition is that the depth (i.e., function composition) increases oscillation exponentially; while width (i.e., linear combination) increases oscillation linearly.

Definition 7.1. We say that f is K-sawtooth if f is piecewise affine with K pieces. For example, the ReLU function is 2-sawtooth, and ψ is 4-sawtooth.

We can show that function composition is stronger to produce more "sawtooth" than addition:

Lemma 7.1. If f is a-sawtooth, g is b-sawtooth, then $f + g$ is at most $(a + b)$-sawtooth, $f \circ g$ is at most ab-sawtooth.

By using this lemma, we can show the converse error bound on the representation power of shallow network:

Theorem 7.2. Given an underlying function F and data points $\{(x_i, y_i \triangleq F(x_i))\}_{i=1}^n, y_i \in \{0, 1\}$, define the classification error for the approximation function f:

$$R(f) = \frac{1}{n} \sum_{i=1}^{n} 1 \left\{ \text{sign}(f(x_i) - 1/2) \neq y_i \right\}.$$

Construct the data points $x_i = \frac{i}{2^{L^*}}$, $y_i = \psi^{(L^*)}(x_i), i = 1, \ldots, 2^{L^*}$. As a result, $y = (0, 1, 0, 1, \ldots)$, and $F(x) = \psi^{(L^*)}(x)$. If a ReLU neural network f has L layers, and width $m < 2^{(L^*-k)/L-1}$, then $R(f) > \frac{1}{2} - \frac{1}{3}\frac{1}{2^{k-1}}$.

Corollary 7.3. If f has L layers with width $m < 2^{(L^*-1)/L-1}$, then the error function is lower bounded by a constant: $R(f) > 1/6$.

Corollary 7.4. If f has no more than $\sqrt{L^*}$ layers, then we need at least $m > 2^{\mathcal{O}(\sqrt{L^*})}$ neurons to get the error less than $1/6$.

Remark 7.2. There is an Implicit assumption on Theorem (7.2), i.e., the neural network is fully connected feedforward. It does not apply to ResNet and RNN. The paper (**NIPS2018_7855**) shows the representation power for ResNet.

Remark 7.3. The representation power on other kinds of neural-nets is a popular problem, such as graph neural-nets and meta-learning.

Summarization There are three criteria for the performance of neural-nets:

$$\begin{cases} \text{Representation Error} \\ \text{Optimization Error} \\ \text{Generalization Error} \end{cases}$$

The important factors for the success of neural-nets are as follows:

1. The depth of neural-nets relates to the representation error;
2. The width of neural-nets relates to the landscape of neural-nets, which further influences the optimization error;
3. The initialization and normalization techniques relate to the convergence performance of optimization;
4. The architecture design influences the representation error; and the optimization error;
5. The SGD algorithm influences the speed for convergence during the optimization process, and people believe that it also tends to give a solution with low generalization error

Remark 7.4. Why the over-parametrization of neural-nets usually do not lead to over-fitting? Prof. Ruoyu Sun gives his understanding of this question. Consider true data $\{(x_i, y_i)\}_{i=1}^n$ generated by $f^*(x_i) = y_i$. We want to approximate f^* with f, by using these n data points. Let W^* denote the representation-power threshold, and n^* denotes the threshold for the number of data points, under which the approximation is likely to be bad. The number of parameters of f is more than W^* will not cause over-fitting, but when $n < n^*$, it is likely to cause over-fitting.

7.3 GAN

Now we turn from supervised learning to unsupervised learning. Prof. Ruoyu Sun will give basic formulation about GAN this lecture, and

Prof. Mingyi Hong will provide the introduction to Adversarial Attack & Defense in the next lecture.

Motivation

> Richard Feymann: What I cannot create, I do not understand

We wish to learn the data distribution $\mathbb{P}_d$, e.g., a style of writing of articles. In order to do so, we build a generative model which generates $\mathbb{P}_g$, e.g., imitates writing articles; and a classifier which judges whether the received sample comes from $\mathbb{P}_d$ or $\mathbb{P}_g$, e.g., give comments to the written samples. Finally, we want $\mathbb{P}_g \approx \mathbb{P}_d$, e.g., the generated article has a similar style of the original one.

We are interested in solving the optimization problem

$$\min_{\mathbb{P}_g} \Phi(\mathbb{P}_d, \mathbb{P}_g)$$

The question is that what distance measure should the Φ be? Statiscians tend to pick $\Phi(P, Q) = \mathrm{KL}(P, Q)$ or $\Phi(P, Q) = \mathrm{JS}(P, Q)$ empirically. However, it is not clear whether these distance metrics are good metrics. An ideal metric should satisfy the following property: if two images are from the same class, then their distance is small; if they are from different classes, then their distance is large. Is the JS distance between the images of two cats is smaller than the distance between a cat and a dog? This is not clear. The major issue here is that common distances may not capture the right representation of the images. To explain the solution, next, we use an example of fake paintings.

Discussion Suppose we want to generate an appropriate painting, denoted as X. Given an artist paint something, denoted as $\hat{X}$; and hire a critic to judge whether it is good or bad. We use $D(x)$ to represent the probability that the input x is thought by the critic to come from the data P_d rather than P_g. The evaluation score can be modeled as

$$L^{\mathrm{GAN}}(\mathbb{P}_d, D) = \mathbb{E}_{x\sim\mathbb{P}_d}[\log D(x)] + \mathbb{E}_{\hat{x}\sim\mathbb{P}_g}[\log(1 - D(\hat{x}))]$$

Pick the *best* critic, i.e., the most strict critic, the distance measure is

$$\Phi(\mathbb{P}_d, \mathbb{P}_g) = \max_D L^{\text{GAN}}(\mathbb{P}_d, D)$$

Therefore, the optimization for GAN is a minimax problem:

$$\min_{\mathbb{P}_g} \max_D \mathbb{E}_{x\sim\mathbb{P}_d}[\log D(x)] + \mathbb{E}_{\hat{x}\sim\mathbb{P}_g}[\log(1 - D(\hat{x}))] \tag{7.1}$$

When proposing a new mode, the sanity-check is needed, i.e., ensure that the global optimum equals whatever we want, i.e., the optimal solution $\mathbb{P}_g^* = \mathbb{P}_d$.

Theorem 7.5 ((NIPS2014_5423)). The global minimum of the problem (7.1) is achieved if and only if $\mathbb{P}_g^* = \mathbb{P}_d$. Moreover, this optimization problem is equivalent to minimizing the Jensen-Shannon divergence

$$\Phi(\mathbb{P}_g, \mathbb{P}_d) = -\log 4 + 2\text{JSD}(\mathbb{P}_d \| \mathbb{P}_g).$$

Proof. Finding the global minima of the problem consists of two steps: first, we need to specify the range of the objective function; second, we need to identify some points that achieve the extreme of the range.

- Question 1: What is the range of the objectvie function?

  We find that $D(x) \in (0, 1)$, and therefore $L^{GAN} \in (-\infty, 0)$. It seems that it is meanless to solve an optimization problem with negative infinite value. In fact, the objective is lower bounded since the maximum criteria help.

- Question 2: Check when does the objective achieve the optimum.

  Consider the finite support distribution for simplicity. Denote the pmf from $\mathbb{P}_d$ and $\mathbb{P}_g$ as $\{q_1, \ldots, q_n\}$, $\{p_1, \ldots, p_n\}$, respectively. It suffices to solve

  $$\begin{array}{ll} \min_{p\in\mathcal{P}^n} \max_{d_i\in(0,1)} & \sum_{i=1}^n q_i \log d_i + \sum_{i=1}^n p_i \log(1 - d_i) \\ \text{with} & \mathcal{P}^n = \{p \mid \sum_i p_i = 1, p_i \geq 0\} \end{array}$$

  Consider the maximum optimization first:

  $$\max_{d_i\in(0,1)} \sum_{i=1}^n q_i \log d_i + \sum_{i=1}^n p_i \log(1 - d_i)$$

It is decomposable in terms of i. For each single problem $\max_{d_i} q_i \log d_i + p_i \log(1 - d_i)$, we find the optimal solution is $d_i = \frac{q_i}{q_i + p_i}$. Substituting this solution into $\Phi(\mathbb{P}_g, \mathbb{P}_d)$, we imply

$$\begin{aligned}\Phi(q, p) &= \sum_i q_i \log \frac{q_i}{q_i + p_i} + \sum_i p_i \log \frac{p_i}{q_i + p_i} \\ &= \mathrm{JSD}(p \| q) - 2 \log 2\end{aligned}$$

We find that it suffices to minimize $\Phi(q, p) \in (-2 \log 2, 0)$, which is a valid probem now. After solving this minimization problem, we obtain the optimal d:

$$d_i^*(p_i) = \frac{q_i}{q_i + p_i} = \begin{cases} 1, & \text{if } p_i = 0, \text{ juage as a bad generator} \\ 0, & \text{if } q_i = 0, p_i > 0, \text{ juage as invalid} \\ 1/2, & \text{juage as a good generator} \end{cases},$$

i.e., for certain data point i, the discrimintator returns a probability $q_i/(q_i + p_i)$. At optimal $p^* = q$, $d_i^*(p^*) = 1/2, \forall i$.

□

Remark 7.5. This result is misleading somehow. For instance, images are continuous distributions, so it is impossible to expect the generated image exactly match the original image, i.e., we can never achieve values for d_i other than $\{0, 1\}$.

Remark 7.6. This proof justifies GAN by relating it to Jensen–Shannon divergence, but in the beginning we think that this distance is not good.

Motivation of W-GAN The Jensen–Shannon divergence is not a good metric in some settings. For instance, it is impossible to measure the distance between two distributions with the different supporting set, but Wasserstein distance givens a reasonable measure. The p-th Wasserstein distance between two probability measures μ, ν is defined as

$$W_p(\mu, \nu) = \min_{p \sim \Gamma(\mu, \nu)} \left(\mathbb{E}_{(x,y) \sim \mathcal{P}} |x - y|^p \right)^{1/p}$$

where $\Gamma(\mu, \nu)$ denotes the set of all couplings of μ and ν. When $p = 1$, finding the Wasserstein distance reduces to solving an LP problem.

Moreover, the W_1 distance can be re-expressed using duality of LP:

$$W_1(\mu, \nu) = \sup_{|f|_L \leq 1} \mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{y\sim\nu}[f(y)]$$

where the supremum is taken over all the 1-Lipschitz functions f. The W-GAN solves the following problem:

$$\min_{\mathbb{P}_g} \max_{|f|_L \leq 1} \mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{\hat{x}\sim\nu}[f(\hat{x})]$$

Remark 7.7. The origin GAN using Jensen–Shannon divergence also works, but using Wasserstein distance is better. However, Wasserstein distance does has some disadvantages. For example, it is not generalizable, i.e., to approximate $W_1(\mu, \nu)$, we require $\exp(d)$ samples from (μ, ν), where d is the supporting dimension of μ and ν, even when they are Gassuain distributions. One solution is to realize that the objective functions in the original GAN and W-GAN are actually using different distance metrics called "neural network distance". Again, we emphasize that neural-network distance is not a new distance, but a distance with good generalization property, and is used by everyone although the distance metric is not explicitly defined before.

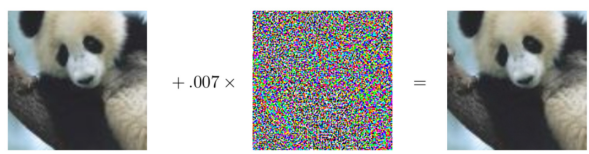
8

Adversarial Learning

In this lecture, we will discuss adversarial attack & defense for the neural network. We will give mathematical descriptions about attack and defense, and discuss the current trend in this field.

8.1 Introduction to Adversarial Learning

Motivation The earliest research on the adversarial attack is on (**43405**) and (**42503**), where (**42503**) shows that a small but specified designed perturbation of image changes the prediction of a neural net, while (**43405**) presents an example that a panda image with a small perturbation still looks like the same for humans, but it is classified as a gibbon by the neural network.



$\boldsymbol{x}$
"panda"
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Remark 8.1. Some comments on this adversary example:

1. An adversary attack is usually performed on several well-known deep learning models, such as *GooLeNet*.

2. The performance of an adversary attack is the *Robust Error*, i.e., the proportion of data samples that can be effectively attacked by our attacking method. In (**43405**), the robust error was 87.5% on the CIFAR-10 dataset, which was later increased to 100% by the *C&W* attack in (**7958570**). Therefore, each image can be easily attacked.

3. Transferability issue: adversarial examples designed for one kind of neural-nets can attack other kinds of neural-nets.

4. The targetted attack is also easy, i.e., we can manipulate the prediction to whatever target we want.

5. It is hard to defend.

6. A good attack can achieve

   - high robust error;
   - small number of queries of the model;
   - low magnitude of perturbation.

Type of attacks Based on whether the hacker knows the model or not, adversarial attacks can be classified as:

- White box attack: the hacker can access everything of the victim model;

- Black box attack: the hacker can only access top-k confidence scores or labels.

Based on whether the attack is aimed to manipulate the prediction, adversarial attacks can be classified as:

- Targetted attack: an example in class-A is classified into class-B;

- Untargeted attack: an example in class-A is classified into a class other than A.

Based on what kinds of data the hacker can access, adversarial attacks can be classified as:

- Evasion attack: the hacker can change testing data. We focus on this kind of attack in this lecture
- Poison attack: the hacker can change the training data.

Bibliography In 2014, the papers (**43405**) and (**42503**) were the first ones that introduced white-box attacks; In 2016, the papers (**7958570**) introduced black-box attacks; In 2017, (**Chen2017**) first performed the black box attacks by zeroth-order optimization methods; In 2018, (**Ilyas2018**) introduced the zeroth-order attacks by querying the objective as less as possible.

8.2 Mathematical Formulation of Adversary Attack

Consider the supervised setting, i.e., given the data points $\{(x_i, y_i)\}_{i=1}^n$, the modeler wants to generate a neural network f_θ such that $f(\theta; x_i) \approx y_i$ for each i. Now we give various formulations for different types of attacks.

8.2.1 Un-targetted Attack

The goal of an untargeted attacker is that given a data instance x, the perturbed input $x + \delta$ will make the neural-nets learn a wrong model. This gives an optimization formulation:

$$\min_{\delta \in \mathbb{R}^d} \quad \|\delta\|_p \tag{8.1a}$$

$$\text{s.t.} \quad f(x + \delta) \neq y \tag{8.1b}$$

$$x + \delta \in [0, 1]^d \tag{8.1c}$$

where (8.1a) is to make the energy of the pertubation as small as possible; (8.1b) is to mislead the model; (8.1c) is to make the perturbed

input well-defined. The inequality constraint (8.1b) makes the problem hard to solve. Thus we often reformulates this problem as

$$\begin{aligned} \max \quad & J(f(x+\delta), y) && \text{(8.2a)} \\ \text{s.t.} \quad & \|\delta\|_p \leq \epsilon && \text{(8.2b)} \\ & x + \delta \in [0,1]^d && \text{(8.2c)} \end{aligned}$$

where $J(\cdot, \cdot)$ is some loss function; the objective (8.2a) is to maximize the loss between output for perturbed data and original one; the constraint (8.2b) is to keep the perturbation in a small magnitude ε. The decision variable in this problem is the input perturbation δ instead of parameters in the neural-nets f.

The *fast gradient sign method* (FGSM) (**43405**) can be used to solve this optimization problem. First linearize the objective (8.2a) and project it into the norm constrained set, we imply that the update should be

$$\eta = \epsilon \cdot \text{sign}(\nabla_x J(f(x), y)),$$

and the adversarial attack is performed by setting $\tilde{x} = x + \eta$.

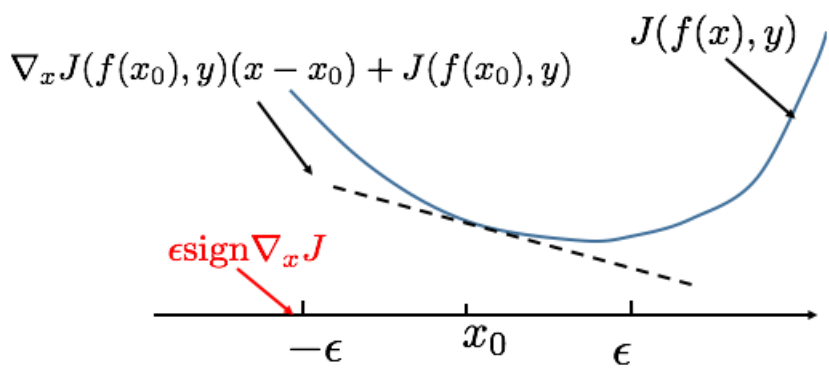


Figure 8.1: Illustration for Fast gradient sign method

The update rule is very easy, and only one-time update is performed.

8.2.2 Targetted Attack

The goal of targetted attack is to add a perturbation such that the output label is a specific class. Suppose that the target class is $t \in \{1, \dots, M\}$. Similar to the idea in un-targetted attack, the typical formulation is

$$\begin{aligned} \min_{\delta \in \mathbb{R}^d} \quad & \|\delta\|_p && \text{(8.3a)} \\ \text{s.t.} \quad & f(x+\delta) = t && \text{(8.3b)} \\ & x + \delta \in [0,1]^d && \text{(8.3c)} \end{aligned}$$

This kind of attack is called *Carlini-Wagner* attack, proposed in the paper (**7958570**). Solving (8.3) is in general harder than solving (8.1). We view this problem as an equality-constrained optimization, which can be solved using *penalty method*:

- Step 1: Find a function $g(\cdot)$ such that

$$g(\tilde{x}) \leq 0 \implies f(\tilde{x}) = t$$

  - For example, construct $g(\tilde{(x)}) = \left(\frac{1}{2} - [Z(\tilde{x})]_t\right)_+$, whre $([Z(\tilde{x})]_t)_{t=1}^M$ is the output before the softmax layer. If $g(\tilde{(x)}) \leq 0$, then the t-th entry before the softmax layer is larger than $\frac{1}{2}$, i.e., $f(\tilde{x}) = t$.
  - Similar to the idea above, construct

$$g(\tilde{x}) = \left(\max_{i \neq t}\{[Z(\tilde{x})]_i\} - [Z(\tilde{x})]_t\right)_+$$

- Step 2: Penalize the equality constraint into the objective function:

$$\begin{aligned} \min_{\delta} \quad & \|\delta\|_p + c \cdot g(x+\delta) && \text{(8.4a)} \\ \text{s.t.} \quad & x + \delta \in [0,1]^d && \text{(8.4b)} \end{aligned}$$

Solving Optimization Problems in Adversary Attack Either Zeroth-order or First-order optimization algorithm can be used to solve the problems like (8.1) or (8.4). The advantage of first-order methods, such

as projected gradient descent and Adam (**kingma-adam**), are fast convergence rates, but they need the gradient information, i.e., using backpropagation of the neural-nets, i.e., white box attack is needed, which is in-practical in life. Therefore, people consider the zeroth-order methods, also called the *derivative-free methods*. This method estimates gradient information using objective evaluation. However, this method is *slower* than gradient methods, by order of low polynomial of problem dimension.

8.2.3 Zeroth-order Optmization

The basic ideas of zeroth-order methods are to approximate the gradient information by calling objective value several times. At point x, the Gausaain vector u is generated with correlation B^{-1}. Then the gradient at x can be approximated as:

$$g_\mu(x) = \frac{f(x+\mu u) - f(x)}{\mu} \cdot Bu,$$

or

$$\hat{g}_\mu(x) = \frac{f(x+\mu u) - f(x-\mu u)}{2\mu} \cdot Bu.$$

Given T iterations for running the algorithm, the optimality measures are $h_T := \mathbb{E}[f(x_T)] - f^*$ for convex problems, or $\tilde{h}_T := \mathbb{E}[\|\nabla f(x_T)\|^2]$ for non-convex problems. The typical zeroth order method is the random gradient free algorithm (**Nesterov2011**), i.e., in each iteration perform the gradient-descent-like update:

$$x_{k+1} = x_k - hB^{-1}g_\mu(x_k).$$

For convex objective f, $h_T = \mathcal{O}(d/T) + \mathcal{O}(\epsilon)$, where d is the problem dimension. To achieve $h_T \leq \epsilon$, $T = \mathcal{O}(d/\epsilon)$ iterations are needed. For non-conex f, $\tilde{h}_T = \mathcal{O}(d/T) + \mathcal{O}(\epsilon)$. To achieve $\tilde{h}_T \leq \epsilon$, $T = \mathcal{O}(d/\epsilon)$ iterations are needed.

8.3 Adversarial Defense

Currently, there are two kinds of defense strategies:

- Adversarial training: Defender generates training data using a known attack, and then tries to improve his model. However, the model may still be vulnerable under new kinds of attacks. Moreover, it does not perform very well even for given attacks.

- Defensive distilling: Based on distilling the network knowledge, which is effective, i.e., reducing robust error from 95% to 5%. However, it does not perform well for the $C\&W$ attacks

8.3.1 Certified Defense

This defense method borrows the idea of robust optimization, which aims to solve the min-max problem:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathbb{P}} \qquad \max_{\tilde{x}} \left[1\{f_\theta(\tilde{x}) \neq y\}\right] \tag{8.5a}$$

$$\text{s.t.} \qquad \|\tilde{x} - x\|_p \leq \varepsilon \tag{8.5b}$$

The objective (8.5a) is to minimize the percentage of getting an error among all samples, and the constraint (8.5b) assumes the perturbation magnitude is bounded by ε.

The state of art for certified defense method is shown in the following table (**SPA3327757**):

Accuracy	MINST	CIFAR-10
$\varepsilon = 0.1$	3%	34%
$\varepsilon = 0.3$	34%	
$\varepsilon = 2/255$		36%
$\varepsilon = 2/255$		70%

Table 8.1: The entries in the table are the robust error, the ε denotes the norm of perturbation.

Generally speaking, it is very challenging to make models robust to different kinds of attacks. It is still an active research area.

8.4 Optimization Algorithms

8.4.1 Part I: Gradient Descent (GD) method

Consider the unconstrained problem

$$\min_x f(x)$$

Apply the Gradient Descent (GD) method:

$$x^{k+1} = x^k - \gamma \nabla f(x^k)$$

Following questions need to be considered:

1. When does the algorithm work?
   - Assumption for the problem:
     (a) The objective function is L-smooth;
     (b) The level set $X_\delta = \{x \mid f(x) \leq \delta\}$ is bounded for all δ.
2. What kind of solution can it compute?
   - It finds the ϵ-first-order-stationary-points, i.e., x such that $\|\nabla f(x)\| \leq \epsilon$.
3. How fast can we compute it?
   - Sublinear convergence rate. We encourage the reader to first go through the convergence proof for convex objective functions in the appendix. The related contents are typed on the undergraduate course MAT3220, *Optimization II*.

By assuming that f is L-lipschitz which is not necessarily convex, we give a convergence proof for the gradient descent method.

Theorem 8.1. Suppose that the objective function f is L-lipschitz, and gradient descent method is applied in each iteration:

$$x^{k+1} = x^k - \frac{1}{L}\nabla f(x^k).$$

Define $T^* := \min\{i : \|\nabla f(x^i)\|^2 \leq \epsilon\}$, then $T^* = \mathcal{O}(1/\epsilon)$.

Proof. 1. Step 1: GD method always makes significant progress in each iteration.

$$f(x^{r+1}) - f(x^r) \leq \langle \nabla f(x^r), x^{r+1} - x^r \rangle + \frac{L}{2}\|x^{r+1} - x^r\|^2 \tag{8.6a}$$

$$\leq -\frac{1}{L}\|\nabla f(x^r)\|^2 + \frac{L}{2}\|x^{r+1} - x^r\|^2 \tag{8.6b}$$

$$\leq -\frac{1}{2L}\|\nabla f(x^r)\|^2 \tag{8.6c}$$

where (8.6a) is by the Lipschitzness of f; (8.6b) is by the identity $x^{r+1} - x^r = -1/L \nabla f(x^r)$; (8.6c) is by the inequality $\|x^{r+1} - x^r\| \leq \frac{1}{L}\|\nabla f(x^r)\|$.

2. Step 2: Considering the best performance in first T^* iteration.

   By (8.6c), we imply

$$f(x^\infty) - f(x^0) \leq -\frac{1}{2L}\sum_{r=0}^{\infty}\|\nabla f(x^r)\|^2 \tag{8.7}$$

   Moreover, since T^* is the first iteration where the iterate reaches the ϵ-suboptimality point,

$$\epsilon \leq \frac{1}{T^*}\sum_{r=1}^{T^*}\|\nabla f(x^r)\|^2$$

$$\leq \frac{1}{T^*}\sum_{r=0}^{\infty}\|\nabla f(x^r)\|^2 \leq \frac{2L}{T^*}[f(x^0) - f(x^\infty)]$$

   It follows that

$$T^* \leq \frac{2L[f(x^0) - f(x^\infty)]}{\epsilon} = \mathcal{O}(1/\epsilon).$$

□

Remark 8.2. 1. The convergence order does matter, but before caring about that, we need to pay attention to the *optimality measure*, i.e., whether $\|\nabla f(x)\| \leq \epsilon$ or $\|\nabla f(x)\|^2 \leq \epsilon^2$.

2. To reach ϵ-FOSP, we need $\mathcal{O}(1/\epsilon^2)$ iteration; in deep learning training, the gradient descent operation in each iteration is expansive.

3. The gradient descent, if converge, then it converges to SOSP with probability one, and it will escape strict saddle points.

   However, the gradient descent does not necessarily converge to SOSP. Moreover, the convergence of gradient descent to SOSP is too slow, and perturbation can rescue this situation. Take one special optimization as an example.

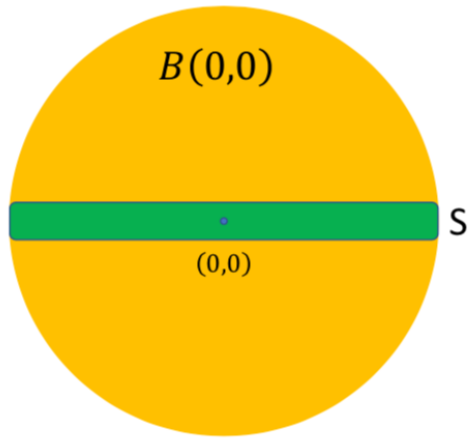


Figure 8.2: Gradient Descent cannot escape saddle point efficiently

The objective is a quadratic function

$$f(x) = \frac{1}{2} x^{\mathrm{T}} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} x$$

The gradient descent formula is given by:

$$x^{k+1} = \begin{pmatrix} 1-\gamma & 0 \\ 0 & 1+\gamma \end{pmatrix}$$

In this case, the gradient descent will get stuck around the line of the saddle point $(0, 0)$, but perturbed gradient descent motivates the iterates to explore more areas around the saddle point.

In the next lecture, modern optimization methods will be introduced.

9

Optimization Algorithms

9.1 Reviewing

- In order to reach $\sqrt{\epsilon}$-FOSP such that $\|\nabla f(\theta^r)\|^2 \leq \epsilon$, the $\mathcal{O}(1/\epsilon)$ iterations are needed.
- Observe that each update of gradient descent is equivalent to minimizing a taylor-expansion-based quadratic upper bound on the objective function:

  $$x^{k+1} = x^k - t_k \nabla f(x^k) = \arg\min_{x \in \mathcal{X}} \frac{1}{2t_k} \|x - x^k\|^2 + \nabla^{\mathrm{T}} f(x^k)(x - x^k).$$

  Therefore, gradient descent is one special case of successive convex approximation (SCA) technique. See the survey paper (**Razaviyayn2014Suc** on the SCA technique for detail.

9.2 Variants of Gradient Descent (GD) Method

9.2.1 Scaled GD

Consider the minimization problem

$$\min_{\theta} F(\theta).$$

The intuition of the scaled GD is to left-multiply the gradient $\nabla F(\theta_t)$ with a positive definite matrix to avoid osillation:

$$\theta_{t+1} = \theta_t - \alpha_t D_t \nabla F(\theta_t), \quad D_t \succ 0.$$

Example 9.1. Consider the linear regression problem

$$\min_{\theta} \ \frac{1}{2}\|A\theta - b\|^2, \tag{9.1}$$

with

$$A = \begin{pmatrix} a_1 & \cdots & a_d \end{pmatrix},$$
$$\theta = (\theta_i)_{i=1}^d.$$

In practice, the size for the feature a_1 can be very differnt from that of a_2. In order to resolve this issue, the data transformation technique is applied, i.e., introduce a new variable

$$w \triangleq S\theta, \qquad \text{where } S = \operatorname{diag}(s_1, \ldots, s_d).$$

It suffices to solve the new problem where the data matrix AS^{-1} is well-conditioned:

$$\min_{w} \|AS^{-1}W - b\|^2 \tag{9.2}$$

- The GD method for solving (9.1) is given by

$$\theta_{t+1} = \theta_t - \gamma A^{\mathrm{T}}(A\theta_t - b)$$

- The GD method for solving the transformed problem (9.2) is given by

$$w_{t+1} = w_t - \gamma S^{-1} A^{\mathrm{T}}(AS^{-1}w_t - b)$$

  Left-multiplying S^{-1} for this iteration gives

$$\theta_{t+1} = \theta_t - \gamma S^{-2} A^{\mathrm{T}}(A\theta_t - b)$$

Therefore, solving the transformed problem (9.2) is equivalent to the scaled GD method by setting $D \equiv S^{-2}$.

Example 9.2 (Newton's Method). Newton's method is a special case of scaled GD by setting $D_t \equiv (\nabla^2 F(\theta_t))^{-1}$:

$$\theta_{t+1} = \theta_t - \alpha_t (\nabla^2 F(\theta_t))^{-1} \nabla F(\theta_t)$$

Newton's method performs good in linear regression problem. If we choose $\alpha_t = 1$, then Newton's method gives the optimal solution in one iteration:

$$\begin{aligned} \theta_{t+1} &= \theta_t - \alpha_t (\nabla^2 F(\theta_t))^{-1} \nabla F(\theta_t) \\ &= \theta_t - (A^{\mathrm{T}} A)^{-1} \cdot [A^{\mathrm{T}}(A\theta - b)] \\ &= (A^{\mathrm{T}} A)^{-1} A^{\mathrm{T}} b. \end{aligned}$$

Example 9.3 (Gauss-Newton Method). Consider the non-linear regression problem

$$\min_{\theta} \; F(\theta) \triangleq \frac{1}{2} \sum_{i=1}^{n} [f_i(\theta)]^2$$

The gradient and Hessian are computed as follows:

$$\begin{aligned} \nabla F(\theta) &= \sum_{i=1}^{n} f_i(\theta) \nabla_\theta f_i(\theta) \\ \nabla^2 F(\theta) &= \sum_{i=1}^{n} \nabla_\theta f_i(\theta) \nabla_\theta^{\mathrm{T}} f_i(\theta) + \underbrace{\sum_{i=1}^{n} f_i(\theta) \nabla^2 f_i(\theta)}_{S(\theta)} \end{aligned}$$

The Gauss-Newton method sets

$$D_t \equiv \left(\sum_{i=1}^{n} \nabla_\theta f_i(\theta) \nabla_\theta^{\mathrm{T}} f_i(\theta) \right)^{-1},$$

which is an approximation of the inverse of the Hessian matrix. There are two advantages:

1. It reduces the Hessian computation time into gradient computation time.

2. The D_t is positive definite, which avoids the numerical instability of Newton's method. Actually, in the paper (**Zhang2000**), Prof. Yin Zhang compares the difference between these two methods.

He found that the Gauss-Newton method possesses a desirable behavior of only seeking global (or good local) minima, while the Newton method is blindly attracted to all kinds of FOSPs.

Remark 9.1. There are generally two choices of step size for GD and variants of GD method:

- Constant step size: $\alpha_t \in (0, 1/L)$;
- Diminishing stepsize: $\alpha_t \to 0$ but satisfies the infinite travel condition
$$\sum_{t=1}^{\infty} \alpha_t = \infty.$$

For constrained minimization problem, the projected gradient descent method is applied.

9.2.2 Stochastic Gradient Descent (SGD)

Consider minimizing an objective with the finite-sum form:

$$F(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f_\theta(x_i), y_i) \triangleq \frac{1}{N}\sum_{i=1}^{N} F_i(\theta)$$

The SGD iteration is given by

$$\theta_{t+1} = \theta_t - \alpha_t \nabla F_{j[t]}(\theta_t),$$

where $j[t]$ is the sample randomly picked from $\{1, \ldots, N\}$ at the t-th iteration.

Remark 9.2. 1. The SGD works under the following assumptions:

- Each component function F_i is Lipschitz continuous, i.e., $\|\nabla F_i(\theta) - \nabla F_i(\xi)\| \le L \cdot \|\theta - \xi\|$;
- The variance for estimation of gradient is uniformly bounded:
$$\mathbb{E}\left[\|\nabla F_{j[t]}(x) - \nabla f(x)\|^2\right] \le \sigma^2$$

2. Stochastic gradient method is not a descent method, though it is frequently referred to as stochastic gradient descent in the literature.

3. We will show that α_t has to be a dimishing step-size.

4. If $j[t]$ are selected like the form $\{1, \dots, N, 1 \dots, N, \dots\}$, then it reduces to the incremental gradient algorithm.

5. People prefer to use SGD in machine learning field instead of GD. The disadvantage for SGD is that it needs $\mathcal{O}(1/\epsilon^2)$ iterations to get $\sqrt{\epsilon}$-FOSP, while GD needs only $\mathcal{O}(1/\epsilon)$ iterations. However, the complexity of the number of gradient calls in each inner iteration is different. The computation of the gradient of component functions per iteration for SGD is $\mathcal{O}(1)$, but GD requires $\mathcal{O}(N)$ calls. Due to this fact, SGD is faster than GD in practice.

	# gradient calls per iteration	# iterations
GD	n	$\boldsymbol{\epsilon^{-1}}$
SGD	$\mathbf{1}$	ϵ^{-2}

Table 9.1: Comparison of SGD and GD

Theorem 9.1. Suppose that the objective function satisfies the assumptions mentioned before, and the SGD is applied in each iteration:

$$\theta_{t+1} = \theta_t - \alpha_t g_t, \qquad \text{where } g_t := \nabla F_{j[t]}(\theta_t).$$

Define $T^* := \min\{i : \|\nabla F(\theta_i)\|^2 \le \epsilon\}$, then $T^* = \mathcal{O}(1/\epsilon^2)$.

Proof. 1. Step 1: SGD always makes significant progess in each iteration.

$$\begin{aligned} F(\theta_{t+1}) &\le F(\theta_t) + \langle \nabla F(\theta_t), \theta_{t+1} - \theta_t \rangle + \frac{L}{2}\|\theta_{t+1} - \theta_t\|^2 \\ &= F(\theta_t) - \alpha_t \langle \nabla F(\theta_t), g_t \rangle + \frac{L\alpha_t^2}{2}\|g_t\|^2 \end{aligned}$$

Then taking conditional expectation both sides given θ_t leads to

$$\begin{aligned} \mathbb{E}[F(\theta_{t+1}) \mid \theta_t] &\le F(\theta_t) - \alpha_t \langle \nabla F(\theta_t), \mathbb{E}[g_t \mid \theta_t] \rangle + \frac{L\alpha_t^2}{2}\mathbb{E}[\|g_t\|^2 \mid \theta_t] \\ &= F(\theta_t) - \alpha_t \|\nabla F(\theta_t)\|^2 + \frac{L\alpha_t^2}{2}\mathbb{E}[\|g_t\|^2 \mid \theta_t] \end{aligned} \tag{9.3}$$

Furthermore, by the equality $\|a\|^2 \leq 2\|a-b\|^2 + 2\|b\|^2$,

$$\begin{aligned}\mathbb{E}[\|g_t\|^2 \mid \theta_t] &\leq 2\mathbb{E}[\|g_t - \nabla F(\theta_t)\|^2 \mid \theta_t] + 2\|\nabla F(\theta_t)\|^2 \\ &= 2\sigma^2 + 2\|\nabla F(\theta_t)\|^2\end{aligned}$$

Substituting this identity into the RHS of (9.3) gives

$$\begin{aligned}\mathbb{E}[F(\theta_{t+1}) \mid \theta_t] &\leq F(\theta_t) + \left(-\alpha_t + L\alpha_t^2\right)\|\nabla F(\theta_t)\|^2 + L\alpha_t^2\sigma^2 \\ &\leq F(\theta_t) - \frac{\alpha_t}{2}\|\nabla F(\theta_t)\|^2 + L\alpha_t^2\sigma^2\end{aligned}$$

where the last inequality is by choosing sufficiently small α_t. Therefore, the expected decrease in each iteration is bounded by the gradient term plus some variance:

$$\mathbb{E}[F(\theta_{t+1})] - \mathbb{E}[F(\theta_t)] \leq -\frac{\alpha_t}{2}\|\nabla F(\theta_t)\|^2 + L\alpha_t^2\sigma^2$$

2. Step 2: Considering the best performance in the first T^* iteration.

   Suppose that the step size $\alpha_t \equiv \alpha$ for all t. Then summing over the inequality above for $t = 1, \ldots, T^*$ gives

$$\begin{aligned}\frac{1}{T^*}\sum_{t=0}^{T^*}\mathbb{E}[\|\nabla F(\theta_t)\|^2] &\leq \frac{2\mathbb{E}[F(\theta_0) - F(\theta_{T^*+1})]}{\alpha T^*} + 2L\alpha\sigma^2 \\ &\leq \frac{2[F(\theta_0) - F(\theta_\infty)]}{\alpha T^*} + 2L\sigma^2\alpha\end{aligned}$$

   Choosing $\alpha = 1/\sqrt{T}$ gives

$$\frac{1}{T^*}\sum_{t=0}^{T^*}\mathbb{E}[\|\nabla F(\theta_t)\|^2] \leq \frac{2[F(\theta_0) - F(\theta_\infty)] + 2L\sigma^2}{\sqrt{T^*}}$$

   Moreover, since T^* is the first iteration where the iterate reaches the ϵ-suboptimality point,

$$\epsilon \leq \frac{1}{T^*}\sum_{t=0}^{T^*}\mathbb{E}[\|\nabla F(\theta_t)\|^2] \leq \frac{2[F(\theta_0) - F(\theta_\infty)] + 2L\sigma^2}{\sqrt{T^*}}$$

   Or equivalently,

$$T^* \leq \frac{2[F(\theta_0) - F(\theta_\infty)] + 2L\sigma^2}{\epsilon^2} = \mathcal{O}(\epsilon^{-2}).$$

□

9.3 Momentum-based Method

Now we introduce several famous momentum methods.

9.3.1 Heavy-Ball Method

Each update of Heavy-Ball Method uses the information from the last and the last second iterate points:

$$\begin{cases} m_t = \beta m_{t-1} + (1-\beta)\nabla F(\theta_t) \\ \theta_{t+1} = \theta_t - \alpha_t m_t \end{cases}$$

Remark 9.3. For convex objective function, the heavy-ball method can accelerate the convergence rate from $\mathcal{O}(1/\epsilon)$ iterations into $\mathcal{O}(1/\sqrt{\epsilon})$ iterations. However, it is not clear how the heavy-ball method behaves in general non-convex problems.

9.3.2 Adaptive Gradient methods (AdaGrad) (Duchi2001)

The AdaGrad can be viewed as a scaled version of the SGD:

$$\begin{aligned} &\theta_{t+1} = \theta_t - \alpha_t (D^t)^{-1/2} g_t, \\ \text{where} \quad &D^t = \tfrac{1}{t}\textstyle\sum_{s=1}^{t} g_s \cdot g_s^{\mathrm{T}} \end{aligned}$$

The inituition is that the step size of SGD is hard to control. The AdaGrad resolves this issue by rescaling the step size of i-th coordinate from α_t to $\frac{\alpha_t}{\sqrt{\frac{1}{t}\sum_{s=1}^{t} g_i g_j}}$ In practice, the inverse of D^t is difficult to compute, and therefore we approximate its inverse by simply inversing the diagonal entries of D^t:

$$\theta_{t+1} = \theta_t - \alpha_t \operatorname{diag}\left(\frac{1}{t}\sum_{s=1}^{t} g_s \cdot g_s^{\mathrm{T}}\right)^{-1/2} g_t$$

Or equivalently, this iteration can be written as

$$\begin{cases} v_t = \dfrac{1}{t}\displaystyle\sum_{s=0}^{t} g_s \circ g_s \\ \theta_{t+1} = \theta_t - \alpha_t (v^t)^{-1/2} \circ g_t \end{cases}$$

9.3.3 RMS-Prop (RMSProp)

The RMSprop algorithm is a variant version of AdaGrad, by doing exponential decay of previous information v_{t-1}:

$$\begin{cases} v_t = \beta v_{t-1} + (1-\beta) g_t \circ g_t \\ \theta_{t+1} = \theta_t - \alpha_t (v^t)^{-1/2} \circ g_t \end{cases}$$

Remark 9.4. Unfortunately, there is no theory concerning how to choose β optimally in literature.

9.3.4 Adam (kingma-adam)

Finally, let's introduce the Adam algorithm, which not simply enjoys features from heavy-ball method such that the gradient estimate is momentum, but also borrows ideas from RMSprop such that the step size is adaptively chosen with exponential decay.

$$\begin{cases} m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t \circ g_t \\ \theta_{t+1} = \theta_t - \alpha_t v_t^{-1/2} \circ m_t \end{cases}$$

Bibliography In 2018 ICLR, the paper (**2018on**) gives a counter-example to show that Adam does not converge for convex problems. Specifically, consider the problem

$$\min_{\theta} F(\theta) = \sum_{j=1}^{n} F_j(\theta),$$

with

$$F_j(\theta) = \begin{cases} 5.5\theta^2, & j = 1 \\ -0.5\theta^2, & j \neq 1 \end{cases}$$

The special parameter of Adam $\beta_1 = 0, \beta_2 = 1$ gives the signSGD:

$$\theta_{t+1} = \theta_t - \alpha_t \text{sign}(g_t)$$

which is unlikely to converge. Then they propose AMSGrad to correct this algorithm. Later in 2019 ICLR, the paper (**chen2018on**) shows that Adam-type algorithms do not converge for non-convex problems, and they propose the AdaFom to correct previous algorithms' behaviors.

Summary The Adam-Type Method can be expressed as follows:

$$\begin{cases} m_t = \beta_{1,t} m_{t-1} + (1-\beta_{1,t}) g_t \\ \hat{v}_t = h_t(g_1, \ldots, g_t) \\ \theta_{t+1} = \theta_t - \alpha_t m_t / \sqrt{\hat{v}_t} \end{cases}$$

Some popular variants of the Adam algorithm is shown in the Table below:

$\hat{v}_t$ \ $\beta_{1,t}$	$\beta_{1,t} = 0$	$\beta_{1,t} \le \beta_{1,t-1}$, $\beta_{1,t} \xrightarrow[t\to\infty]{} b \ge 0$	$\beta_{1,t} = \beta_1$
$\hat{v}_t = 1$	SGD	N/A*	Heavy-ball method
$\hat{v}_t = \frac{1}{t}\sum_{i=1}^{t} g_i^2$	AdaGrad	AdaFom	AdaFom
$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$, $\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$	AMSGrad	AMSGrad	AMSGrad
$\hat{v}_t = \beta_2 \hat{v}_{t-1} + (1-\beta_2) g_t^2$	RMSProp	N/A	Adam

* N/A stands for an informal algorithm that was not defined in literature.

Figure 9.1: Variants of Adam algorithm

9.4 Nonconvex nonconcave minimax optimization

Consider the minimax problem

$$\min_x \max_y f(x, y)$$

If we can obtain $y^* = h(x)$, then it suffices to solve the minimization problem

$$\min_x g(x) \triangleq f(x, h(x)).$$

In practice, the function $g(x)$ is always defined implicitly. This problem is the basis of generative adversarial networks and robust training. The gradient descent ascent (GDA) Algorithm is widely used in this problem, but this algorithm cannot even solve a linear problem

$$\min_x \max_y x^{\mathrm{T}} A y.$$

Remark 9.5. 1. Now there are two state-of-the-art algorithms for solving the minimax problem. The first one is the optimistic

GDA; and the second one is to applying BCD heuristic, i.e., approximately optimizing the inner problem in terms of y for few iterations, and then perform gradient descent for the outer problem for one iteration. The assumptions for the second algorithm is that the problem in terms of y is concave, and the constraint set is bounded.

2. It is even not clear what is the local optimality for the minimax problem in the general setting.

3. The GDA is widely used, but its convergence properties remain to be understood.

Appendices

Basic Algorithms for Nonlinear Programming

.1 Gradient Algorithms

.1.1 Preliminaries: convergence analysis

Consider an iterative algorithm for solving the optimization problem $\min f(x)$, producing iterates $\{x^0, x^1, \dots\}$.

1. The possible error measurements are as follows. The stopping criteria depends on these error measurements.

   - $e(x^k) := \|x^k - x^*\|$;
   - $e(x^k) = f(x^k) - f(x^*)$;

   where x^* denotes the underlying optimal solution.

2. We say the algorithm converges if $\lim_{k\to\infty} e(x^k) = 0$

3. There are different types of convergence rate:

   (a) R-linear convergence: there exists $a \in (0, 1)$ such that $e(x^k) \leq Ca^k$;

   (b) Q-linear convergence: there exists $a \in (0, 1)$ such that $\frac{e(x^{k+1})}{e(x^k)} \leq a$;

(c) Sub-linear convergence: $e(x^k) \leq C/k^p$ for some $p > 0$.

question: when say about convergence rate, do we need to specify which error measurements we use?

.1.2 The (Sub)gradient algorithm for Unconstrained Optimization

Consider an unconstrained optimization problem $\min f(x)$, where f may not necessarily be smooth. Let $\{t_k > 0 \mid k = 0, 1, \dots\}$ be a sequence of step-sizes. Let's study the simpleest first order optimization algorithm.

Algorithm 1 The (Sub)gradient Algorithm

Input: Initial guess $x^0 \in \mathcal{X}$
Output: Optimal solution $\hat{x}$
For $k = 0, 1, \dots,$ **do**

- Take $d^k \in \partial f(x^k)$;
- $x^{k+1} \leftarrow x^k - t_k d^k$

end for.

Worst Case Bounds Consier a *convex optimization model* where f is a completely unknown function. The first order type algorithm esentially produces a sequence of iterates $\{x^k \mid k = 0, 1, 2, \dots\}$ in such a way that x^k is in the *affine space* spanned by

$$x^0, g(x^0), \dots, g(x^{k-1}), \text{ where } g(\cdot) = \partial f(\cdot).$$

- Suppose f is *Lipschitz continuous* and no other information is known, we can construct an example such that

$$\min_{x \in \text{Span}\{x^0, g(x^0), \dots, g(x^{k-1})\}} f(x) - f(x^*) \geq \mathcal{O}(\frac{1}{\sqrt{k}}), \ \forall k = 1, 2, \dots, \lfloor \frac{n}{2} \rfloor$$

Therefore, the first order type algorithm can never reach the convergence rate faster than $\mathcal{O}(\frac{1}{\sqrt{k}})$.

- Additionally, if we know f is *differentiable* and ∇f is *Lipschitz continuous*, then we can construct an example such that

$$\min_{x\in\text{Span}\{x^0,g(x^0),\ldots,g(x^{k-1})\}} f(x)-f(x^*) \geq \mathcal{O}(\frac{1}{k^2}),\ \forall k=1,2,\ldots,\lfloor\frac{n}{2}\rfloor$$

  Therefore, the first order type algorithm can never reach the convergence rate faster than $\mathcal{O}(\frac{1}{k^2})$ for optimizing this class of function.

.1.3 Gradient Algorithm with Exact Line-Search

First we discuss the optimization with a uniform convex function. This assumption is by default unless specifically mentioned. A nice Q-linear convergence result is obtained:

Theorem .2. Suppose there exists $0 < m \leq M$ such that $0 \succ mI \succeq \nabla^2 f(x) \succeq MI$ (i.e., f is *uniformly convex*), and an exact line search is performed per iteration:

$$t_k := \arg\min_t f(x^k - t\nabla f(x^k)),$$

then

$$f(x^{k+1}) - f(x^*) \leq \left(1-\frac{m}{M}\right)[f(x^k)-f(x^*)] \tag{4}$$

Proof. • **(Uniform Convexity implies Strongly Convexity)** For $\forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \text{dom}(f)$, by mean-value theorem,

$$f(\boldsymbol{x}_2) = f(\boldsymbol{x}_1)+\langle\nabla f(\boldsymbol{x}_1), \boldsymbol{x}_2-\boldsymbol{x}_1\rangle+\frac{1}{2}(\boldsymbol{x}_2-\boldsymbol{x}_1)^{\mathrm{T}}\nabla^2 f(\boldsymbol{\xi})(\boldsymbol{x}_2-\boldsymbol{x}_1),$$

where $\boldsymbol{\xi}$ is some number between $\boldsymbol{x}_2$ and $\boldsymbol{x}_1$. Applying the uniform convexity of f, we derive the strongly convexity property:

$$\frac{m}{2}\|\boldsymbol{x}_1-\boldsymbol{x}_2\|_2^2 \leq f(\boldsymbol{x}_2)-f(\boldsymbol{x}_1)-\langle\nabla f(\boldsymbol{x}_1), \boldsymbol{x}_2-\boldsymbol{x}_1\rangle \leq \frac{M}{2}\|\boldsymbol{x}_1-\boldsymbol{x}_2\|_2^2 \tag{5}$$

- **(Applying Strongly Convexity Property)** On the one hand, by setting $\boldsymbol{x}_1 = \boldsymbol{x}^*$ and $\boldsymbol{x}_2 = \boldsymbol{x}$ in (5), we obtain:

$$\frac{m}{2}\|\boldsymbol{x}-\boldsymbol{x}^*\|_2^2 \leq f(\boldsymbol{x})-f(\boldsymbol{x}^*) \leq \frac{M}{2}\|\boldsymbol{x}-\boldsymbol{x}^*\|_2^2 \tag{6}$$

On the other hand, by setting $\boldsymbol{x}_1 = \boldsymbol{x}$ and $\boldsymbol{x}_2 = \boldsymbol{x}^*$ in (5), we obtain

$$\begin{aligned}\frac{m}{2}\|\boldsymbol{x}-\boldsymbol{x}^*\|_2^2 &\le f(\boldsymbol{x}^*) - f(\boldsymbol{x}) - \langle \nabla f(\boldsymbol{x}), \boldsymbol{x}^* - \boldsymbol{x}\rangle \\ &\le f(\boldsymbol{x}^*) - f(\boldsymbol{x}) + \|\nabla f(\boldsymbol{x})\| \cdot \|\boldsymbol{x}^* - \boldsymbol{x}\| \\ &\le -\frac{m}{2}\|\boldsymbol{x}-\boldsymbol{x}^*\|_2^2 + \|\nabla f(\boldsymbol{x})\| \cdot \|\boldsymbol{x}^* - \boldsymbol{x}\|\end{aligned}$$

which implies $m\|\boldsymbol{x}-\boldsymbol{x}^*\| \le \|\nabla f(\boldsymbol{x})\|$. Similarly, we get

$$m\|\boldsymbol{x}-\boldsymbol{x}^*\| \le \|\nabla f(\boldsymbol{x})\| \le M\|\boldsymbol{x}-\boldsymbol{x}^*\| \tag{7}$$

- **(Upper Bounding left and right side of (4))** Moreover, we upper bounding the left side of (4) by setting $\boldsymbol{x}_2 = \boldsymbol{x}^{k+1}$ and $\boldsymbol{x}_1 = \boldsymbol{x}^k$ in (5):

$$\begin{aligned}f(\boldsymbol{x}^{k+1}) - f(\boldsymbol{x}^k) &\le \langle \nabla f(\boldsymbol{x}^k), \boldsymbol{x}^{k+1} - \boldsymbol{x}^k\rangle + \frac{M}{2}\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\|^2 \\ &\le -\frac{1}{2M}\|\nabla f(x^k)\|^2\end{aligned} \tag{8}$$

where the second inequality is active when $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \frac{1}{M}\nabla f(\boldsymbol{x}^k)$. On the other hand, by setting $\boldsymbol{x}_2 = \boldsymbol{x}^*$ and $\boldsymbol{x}_1 = \boldsymbol{x}^k$ in (5), we obtain

$$\begin{aligned}f(\boldsymbol{x}^k) - f(\boldsymbol{x}^*) &\le \langle \nabla f(x^k), \boldsymbol{x}^k - \boldsymbol{x}^*\rangle - \frac{m}{2}\|\boldsymbol{x}^k - \boldsymbol{x}^*\|_2^2 \\ &\le \|\nabla f(x^k)\|\|\boldsymbol{x}^k - \boldsymbol{x}^*\| - \frac{m}{2}\|\boldsymbol{x}^k - \boldsymbol{x}^*\|_2^2 \\ &\le \frac{1}{2m}\|\nabla f(\boldsymbol{x}^k)\|^2\end{aligned} \tag{9}$$

Therefore, substituting (9) into (8), we obtain

$$f(\boldsymbol{x}^{k+1}) - f(\boldsymbol{x}^k) \le -\frac{m}{M}[f(\boldsymbol{x}^k) - f(\boldsymbol{x}^*)]$$

Or equivalently,

$$f(\boldsymbol{x}^{k+1}) - f(\boldsymbol{x}^*) \le \left(1 - \frac{m}{M}\right)[f(\boldsymbol{x}^k) - f(\boldsymbol{x}^*)]$$

□

question: this proof also holds for $t_k = \frac{1}{M}$. Thus what is the intuition behind the line search.

question: is uniformly convex and strongly convex talking about the same thing?

.1.4 Gradient Algorithm with Diminishing Step Sizes

Consider a pre-scribed diminishing step size $\{\alpha_k\} \to 0$ but satisfies the infinite travel condition $\sum_{k=1}^{\infty} \alpha_k = \infty$.

In this case, for sufficiently large k, we have $\alpha_k \leq \frac{1}{M}$ and simiar to the idea in (8),

$$f(\boldsymbol{x}^{k+1}) \leq f(\boldsymbol{x}^k) - \frac{\alpha_k}{2}\|\nabla f(\boldsymbol{x}^k)\|^2$$

which implies that $\nabla f(\boldsymbol{x}^k)$ cannot be bounded away from 0 whenever $f(\boldsymbol{x}^k)$ is finitely lower bounded. In other words, if a finite minimum exists for $f(\boldsymbol{x}^k)$, then the iterates satisfy $\lim_{k\to\infty} \inf \|\nabla f(\boldsymbol{x}^k)\| = 0$.

We can further show the whole sequence $f(\boldsymbol{x}^k)$ converges:

Proof. w.l.o.g., assume the inequality below holds for $k = 1, 2, \ldots$, i.e., $\alpha_k \leq \frac{1}{M}$:

$$f(\boldsymbol{x}^{k+1}) \leq f(\boldsymbol{x}^k) - \frac{\alpha_k}{2}\|\nabla f(\boldsymbol{x}^k)\|^2$$

Therefore, for any $k = 1, 2, \ldots,$

$$\begin{aligned} f(\boldsymbol{x}^{k+1}) - f(\boldsymbol{x}^*) &\leq f(\boldsymbol{x}^k) - f(\boldsymbol{x}^*) - \frac{\alpha_k}{2}\|\nabla f(\boldsymbol{x}^k)\|^2 \\ &\leq (1 - m\alpha_k)[f(\boldsymbol{x}^k) - f(\boldsymbol{x}^*)] \end{aligned}$$

where the second inequality is by applying (9). It follows that

$$f(\boldsymbol{x}^n) - f(\boldsymbol{x}^*) \leq [f(\boldsymbol{x}^1) - f(\boldsymbol{x}^*)] \prod_{k=1}^{n} (1 - m\alpha_k) \to 0,$$

i.e., $\lim_{n\to\infty} f(\boldsymbol{x}^n) = f(\boldsymbol{x}^*)$.

There is another way to show the convergence of $\{\nabla f(\boldsymbol{x}^k)\}$:

$$\|\nabla f(\boldsymbol{x}^n)\|^2 \leq 2M[f(\boldsymbol{x}^n) - f(\boldsymbol{x}^*)] \implies \lim_{n\to\infty} \nabla f(\boldsymbol{x}^k) = \mathbf{0}.$$

□

We summarize the results above as a theorem for the convergence of the gradient algorithm with diminishing step sizes:

Theorem .3. Suppose there exists $0 < m \leq M$ such that $0 \succ mI \succeq \nabla^2 f(x) \succeq MI$ (i.e., f is *uniformly convex*), and the dimishing step size is performed per iteration:

$$\alpha_k \to 0, \text{ but } \sum_{k=1}^{\infty} \alpha_k = \infty,$$

then either $f(\boldsymbol{x}^k) \to -\infty$ or else $\{f(\boldsymbol{x}^k)\}$ converges to a finite value and $\nabla f(\boldsymbol{x}^k) \to \mathbf{0}$.

.1.5 Gradient Algorithm with Armijo's Rule

Consider a general iterative descent algorithm $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha_k \boldsymbol{d}^k$. The Armiijo's rule for choosing step sizes is as follows:

> Let $\gamma \in (0, 1)$ (question: 1/2?). Start with $s > 0$ and continue with $\beta s, \beta^2 s, \ldots$, until $\beta^\ell s$ falls within the set of α with the condition
>
> $$f(\boldsymbol{x}^k) - f(\boldsymbol{x}^k + \alpha \boldsymbol{d}^k) \geq -\gamma\alpha \cdot \nabla^{\mathrm{T}} f(\boldsymbol{x}^k)\boldsymbol{d}^k$$

In this case we have $\alpha_k = s\beta^\ell$ and

$$f(\boldsymbol{x}^k) \geq f(\boldsymbol{x}^k + \alpha_k \boldsymbol{d}^k) - \gamma\alpha_k \nabla^{\mathrm{T}} f(\boldsymbol{x}^k)\boldsymbol{d}^k \tag{10a}$$
$$f(\boldsymbol{x}^k) < f(\boldsymbol{x}^k + \alpha_k/\beta \boldsymbol{d}^k) - \gamma\alpha_k/\beta \cdot \nabla^{\mathrm{T}} f(\boldsymbol{x}^k)\boldsymbol{d}^k \tag{10b}$$

We can analysis the convergence result for gradient algorithm, i.e., $\boldsymbol{d}^k = -\nabla f(\boldsymbol{x}^k)$:

- From the (10b) and the Taylor expansion on $f(\boldsymbol{x}^k + \alpha_k \boldsymbol{d}^k)$ we obtain:

  $$f(\boldsymbol{x}^k) + \gamma\alpha_k/\beta \cdot \nabla^{\mathrm{T}} f(\boldsymbol{x}^k)\boldsymbol{d}^k < f(\boldsymbol{x}^k) + \alpha_k/\beta \nabla^{\mathrm{T}} f(\boldsymbol{x}^k)\boldsymbol{d}^k + \frac{M}{2}(\alpha_k/\beta)^2 \|\boldsymbol{d}^k\|^2$$

  Or equivalently, $\alpha_k > \frac{2\beta(1-\gamma)}{M}$

- Combining the (10a), (9) and the bound on α_k, we obtain

  $$f(\boldsymbol{x}^k + \alpha_k \boldsymbol{d}^k) \leq f(\boldsymbol{x}^k) - 4\beta\gamma(1-\gamma)\frac{m}{M}[f(\boldsymbol{x}^k) - f(\boldsymbol{x}^*)]$$

Therefore, we get the Q-linear convergence for Armijo's rule:

$$f(\boldsymbol{x}^{k+1}) - f(\boldsymbol{x}^*) \leq \left(1 - 4\beta\gamma(1-\gamma)\frac{m}{M}\right)[f(\boldsymbol{x}^k) - f(\boldsymbol{x}^*)]$$

.1.6 The Gradient Algorithm for non-strongly convex case

The estimations of convergence so far are based on the assumption that $m > 0$. Now we discuss the case where $m = 0$. The function is convex but not necessarily strongly convex.

Assume that the set of optimal solutions is a bounded set, and that there is a bounded *level set*. If still apply the exact line search, the iterates will be bounded. Note that the inequalities below still hold:

$$f(\boldsymbol{x} + \alpha\boldsymbol{d}) \leq f(\boldsymbol{x}) - \frac{1}{2M}\|\nabla f(\boldsymbol{x})\|^2$$
$$f(\boldsymbol{x}) - f(\boldsymbol{x}^*) \leq \|\nabla f(\boldsymbol{x})\| \cdot \|\boldsymbol{x} - \boldsymbol{x}^*\|$$

Assume that $\|\boldsymbol{x}^k - \boldsymbol{x}^*\| \leq C$, and let $e(\boldsymbol{x}^k) = f(\boldsymbol{x}^k) - f(\boldsymbol{x}^*)$, Using the inequalities above, it's easy to show that

$$e(\boldsymbol{x}^{k+1}) \leq e(\boldsymbol{x}^k) - c[e(\boldsymbol{x}^k)]^2, \text{ where } c = \frac{1}{2MC^2}.$$

which follows that

$$\begin{aligned}\frac{1}{e(\boldsymbol{x}^{k+1})} &\geq \frac{1}{e(\boldsymbol{x}^k)} + \frac{c}{1 - c \cdot e(\boldsymbol{x}^k)} \\ &\geq \frac{1}{e(\boldsymbol{x}^k)} + c \\ &\geq \cdots \\ &\geq \frac{1}{e(\boldsymbol{x}^1)} + k \cdot c\end{aligned}$$

Therefore, we obtain the *sublinear rate of convergence*:

$$e(\boldsymbol{x}^{k+1}) \leq \frac{e(\boldsymbol{x}^1)}{1 + k(c \cdot e(\boldsymbol{x}^1))}$$

.1.7 Linear Convergence without Second Order Differentiability

Acutally, the assumptions on the existence of $\nabla^2 f$ is unnecessaryin Theorem (.2). We can weaken the condition by the inequality below to

obtain the same linear convergence result:

$$\sigma\|\boldsymbol{x}-\boldsymbol{y}\|^2 \leq \langle \nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle \leq L\|\boldsymbol{x}-\boldsymbol{y}\|^2, \ \forall \boldsymbol{x}, \boldsymbol{y}, \quad (11)$$

where $0 < \sigma \leq L < \infty$.

Remark .6.

- The condition (11) can be implied by uniform convexity.
- The interpretation of (11) is that, restricting f to any line segment between $\boldsymbol{x}$ and $\boldsymbol{y}$, the function $h(t) := f(x + t(y - x))$ satisfies

$$0 \leq \frac{h'(t) - h'(s)}{t - s} \leq L, \quad \forall 0 \leq s < t \leq 1,$$

  i.e., the slope of ∇f is bounded.
- The condition (11) implies the strong convexity, which can be shown by appying the directional derivative and (11):

$$\frac{\sigma}{2}\|\boldsymbol{y}-\boldsymbol{x}\|^2 \leq f(\boldsymbol{y}) - f(\boldsymbol{x}) - \langle \nabla f(\boldsymbol{x}), \boldsymbol{y}-\boldsymbol{x}\rangle \leq \frac{L}{2}\|\boldsymbol{y}-\boldsymbol{x}\|^2$$

Therefore, we can use the same logic to show the following inequalities:

$$\begin{aligned} f(\boldsymbol{x} - \alpha\nabla f(\boldsymbol{x})) - f(\boldsymbol{x}) &\leq -\frac{1}{2L}\|\nabla f(\boldsymbol{x})\|^2 \\ \sigma\|\boldsymbol{x}-\boldsymbol{x}^*\|^2 &\leq \|\nabla f(\boldsymbol{x})\|\|\boldsymbol{x}-\boldsymbol{x}^*\| \\ f(\boldsymbol{x}^*) &\geq f(\boldsymbol{x}) - \frac{1}{2\sigma}\|\nabla f(\boldsymbol{x})\|^2 \end{aligned}$$

nad therefore,

$$f(\boldsymbol{x} - \alpha\nabla f(\boldsymbol{x})) - f(\boldsymbol{x}^*) \leq \left(1 - \frac{\sigma}{L}\right)[f(\boldsymbol{x}) - f(\boldsymbol{x}^*)].$$

.2 The Pure Newton's Method

Now we discuss a particularly important method in optimization: Newton's method.

Motivation This method is a *linearlization scheme* for solving a nonlinear equation.

- For scalar form of nonlinear equation $g(x) = 0$, we apply Taylor's expansion on the root $\hat{x}$:

  $$g(\hat{x}) = g(x) + g'(x)(\hat{x} - x) + o(|\hat{x} - x|)$$

  Ignoring the high order part we get an approximation, i.e., iterative formula

  $$\bar{x} = x - \frac{g(x)}{g'(x)}.$$

- Consider a n-dimensional equation $g_{1:n}(x_1, \ldots, x_n) = 0$, we have a similar solution

  $$g(\hat{\boldsymbol{x}}) = g(\boldsymbol{x}) + J(g(\boldsymbol{x})) \cdot (\hat{\boldsymbol{x}} - \boldsymbol{x}) + o(\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|)$$

  where $J(g(\boldsymbol{x}))$ denotes the Jacobian matrix of g:

  $$\mathbb{R}^{n \times n} \ni J(g(\boldsymbol{x})) := \left[\frac{\partial g_i(\boldsymbol{x})}{\partial x_j} \right]$$

Therefore, the unconstrained optimization problem suffices to solve a nonlinear equation $\nabla f(\boldsymbol{x}) = 0$, and the iterative formula is

$$\bar{\boldsymbol{x}} = \boldsymbol{x} - [\nabla^2 f(\boldsymbol{x})]^{-1} \nabla f(\boldsymbol{x}), \quad \text{(Newton's Method)}$$

Remark .7.

1. Newton's direction may not necessarily exist;
2. It is a descent direction for strongly convex functions;
3. However, the function may not necessarily decrease even for strongly convex function.
4. It minimizes a strongly convex *quadratic* function in just *one* step.
5. The pure form of Newton's method can be modified by taking another step length.

.2.1 Local Convergence Analysis

We analysis the convergence rate for Newton's method under the convexity and continuity conditions first:

> **Assumption**: The function f is *convex*, *twice continuously differentiable*, and that $\nabla^2 f(\boldsymbol{x}^*)$ is non-singular for local minimum $\boldsymbol{x}^*$.

A key inequality for the analysis is

$$\nabla f(\boldsymbol{y}) = \nabla f(\boldsymbol{x}) + \int_0^1 \nabla^2 f(\boldsymbol{x} + t(\boldsymbol{y} - \boldsymbol{x})) \cdot (\boldsymbol{y} - \boldsymbol{x})\, \mathrm{d}t$$

Suppose that $\boldsymbol{x}^k$ is close to $\boldsymbol{x}^*$ enough, then $\nabla^2 f(\boldsymbol{x}^k)$ is non-singular as well due to the continuity of determinant function. It follows that

$$\begin{aligned}
&\boldsymbol{x}^{k+1} - \boldsymbol{x}^* = \boldsymbol{x}^k - \boldsymbol{x}^* - [\nabla^2 f(\boldsymbol{x}^k)]^{-1} \nabla f(\boldsymbol{x}^k) \\
&= [\nabla^2 f(\boldsymbol{x}^k)]^{-1} [\nabla^2 f(\boldsymbol{x}^k)(\boldsymbol{x}^k - \boldsymbol{x}^*) - \nabla f(\boldsymbol{x}^k)] \\
&= [\nabla^2 f(\boldsymbol{x}^k)]^{-1} \left[\nabla^2 f(\boldsymbol{x}^k)(\boldsymbol{x}^k - \boldsymbol{x}^*) - \int_0^1 \nabla^2 f(\boldsymbol{x}^* + t(\boldsymbol{x}^k - \boldsymbol{x}^*))(\boldsymbol{x}^k - \boldsymbol{x}^*)\, \mathrm{d}t \right] \\
&= [\nabla^2 f(\boldsymbol{x}^k)]^{-1} \left\{ \int_0^1 [\nabla^2 f(\boldsymbol{x}^k) - \nabla^2 f(\boldsymbol{x}^* + t(\boldsymbol{x}^k - \boldsymbol{x}^*))](\boldsymbol{x}^k - \boldsymbol{x}^*)\, \mathrm{d}t \right\}
\end{aligned}$$

Thererfore,

$$\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\| \leq \|\boldsymbol{x}^k - \boldsymbol{x}^*\| \cdot \|[\nabla^2 f(\boldsymbol{x}^k)]^{-1}\| \cdot \int_0^1 \|\nabla^2 f(\boldsymbol{x}^k) - \nabla^2 f(\boldsymbol{x}^* + t(\boldsymbol{x}^k - \boldsymbol{x}^*))]\|\, \mathrm{d}t$$

Since $\boldsymbol{x}^k$ is close to $\boldsymbol{x}^*$, $\|[\nabla^2 f(\boldsymbol{x}^k)]^{-1}\|$ is bounded. Since $\nabla^2 f(\boldsymbol{x})$ is continuous, the integration term goes to zero as $\|\boldsymbol{x}^k - \boldsymbol{x}^*\| \to 0$. Thus we imply $\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\| = o(\|\boldsymbol{x}^k - \boldsymbol{x}^*\|)$, ensuring a superlinear convergence.

> **Extra Assumption**: The term $\nabla^2 f(\boldsymbol{x})$ is *Lipschitz continuous*: there exists $L_2 > 0$ such that
>
> $$\|\nabla^2 f(\boldsymbol{x}) - \nabla^2 f(\boldsymbol{y})\| \leq L_2 \|\boldsymbol{x} - \boldsymbol{y}\|, \quad \forall \boldsymbol{x}, \boldsymbol{y}.$$

This extra assumption will ensure a *quadratic convergence rate*:

$$\begin{aligned}
&\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\| \\
&\leq \|[\nabla^2 f(\boldsymbol{x}^k)]^{-1}\| \cdot \|\boldsymbol{x}^k - \boldsymbol{x}^*\| \cdot \int_0^1 \|\nabla^2 f(\boldsymbol{x}^k) - \nabla^2 f(\boldsymbol{x}^* + t(\boldsymbol{x}^k - \boldsymbol{x}^*))]\|\, \mathrm{d}t \\
&\leq \frac{L_2}{2} \|[\nabla^2 f(\boldsymbol{x}^k)]^{-1}\| \cdot \|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2.
\end{aligned}$$

Further Assumption: Based on the previous two assumptions, we assume that f is *strongly convex*.

In this case, it is easy to show that

$$\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\| \leq \frac{L_2}{2m}\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2.$$

This inequality introduces a *region of attraction*, i.e., as soon as $\boldsymbol{x}^k$ falls into the neighborhood of $\boldsymbol{x}^*$ with radius $2m/L_2$, the iterates will be trapped in the neighborhood and converge to $\boldsymbol{x}^*$ *quadratically*.

Remark .8. The pure form of Newton's method, however, has several drawbacks:

1. It in general does not guarantees global convergence if no additional assumption is given. Fortunartely, if f is strongly convex, then the Newton's method with *line-search* (e.g., with Armijo's step-length rule) will be globally convergent with a globally linear convergence rate.

2. If f is not *strictly* convex, then $\nabla^2 f$ may be singular. Even worse, if f is not convex, then the Newton's direction may not be a *descent direction*. In next section we will discuss how to handle such a situation.

.3 Practical Implementation of Newton's method

.3.1 Cholesky Factorization

First let's introduce a technique in optimization algorithms that can reduce computational complexity: the *Cholesky factorization*.

Consider the case where $\nabla^2 f(\boldsymbol{x}^k) \succ 0$, and the Newton's direction can be found by solving the linear system

$$\nabla^2 f(\boldsymbol{x}^k)\boldsymbol{d} = -\nabla f(\boldsymbol{x}^k).$$

Directly computing the inverse of $\nabla^2 f(\boldsymbol{x}^k)$ is computationally expansive, which motivates us to apply the *Cholesky factorization* as follows:

1. First apply the Cholesky factorization to get $\nabla^2 f(\boldsymbol{x}^k) = \boldsymbol{L}_k \boldsymbol{L}_k^{\mathrm{T}}$, where $\boldsymbol{L}_k$ is a lower triangular matrix, resulting in the following Newton's equation
$$\boldsymbol{L}_k \boldsymbol{L}_k^{\mathrm{T}} \boldsymbol{d} = -\nabla f(\boldsymbol{x}^k).$$
2. Firstly solve the lower triangular system below by forward substitution:
$$\boldsymbol{L}_k \boldsymbol{y} = -\nabla f(\boldsymbol{x}^k)$$
The complexity for this process is $\mathcal{O}(n^2)$.
3. Then solve the triangular system below by backforward substitution:
$$\boldsymbol{L}_k^{\mathrm{T}} \boldsymbol{d} = \boldsymbol{y}_k$$
Again, this step takes complexity $\mathcal{O}(n^2)$.

The basic Cholesky factorization algorithm is as follows:

Algorithm 2 Basic Cholesky factorization Algorithm

Input: A positive definite $n \times n$matrix $\boldsymbol{A}$
Output: Lower triangular matrix $\boldsymbol{L}$ such thtat $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$
For $j = 1 : n$**, do**

- **For** $i = j + 1 : n$**, do**
  - $l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{jk} l_{ik}\right) / l_{jj}$

  end for.
  $l_{jj} = \left(a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2\right)^{1/2}$.

end for.

Remark .9. If $\boldsymbol{A}$ is not positive semidefinite, then at a certain stage we will encounter a j such that
$$a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 < 0.$$
In that case, the Cholesky decomposition cannot proceed. Note that the Cholesky decomposition takes about $\mathcal{O}(n^3)$ operations.

.3.2 Modified Newton's method

In case the Hessian matrix is not positive definite, the following remedies can be applied:

> If there occurs $a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 < 0$ for a certain j, then we simply increase a_{jj} so that the quantity becomes positive again. (question: increase how much?)

This remedy has the same effect of changing $\nabla^2 f(\boldsymbol{x}^k)$ into $\nabla^2 f(\boldsymbol{x}^k) + \Delta^k \succ 0$, where Δ^k is non-negative (question: positive or non-negative?) diagonal, which suffices to solve the regularized equation

$$(\nabla^2 f(\boldsymbol{x}^k) + \Delta^k)\boldsymbol{d} = -\nabla f(\boldsymbol{x}^k).$$

Moreover, we may use the direction with Armijo's line search technique to guarantee the global convergence. (how to show?)

.3.3 The Trust Region Approach

Another way to handle the case that $\nabla^2 f(\boldsymbol{x}^k)$ is indefinite is to use the *trust region approach*. It is the complement of the line search approach.

The direction $\boldsymbol{d}^k$ for each iteration suffices to consider the trust region subproblem

$$\begin{array}{ll} \min & \langle \nabla f(\boldsymbol{x}^k), \boldsymbol{d} \rangle + \frac{1}{2}\boldsymbol{d}^{\mathrm{T}} \nabla^2 f(\boldsymbol{x}^k)\boldsymbol{d} \\ \text{such that} & \|\boldsymbol{d}\| \leq \delta \end{array}$$

where $\delta > 0$ is called the trust region radius.

Remark .10. It can be shown that when δ is sufficiently small, $f(\boldsymbol{x}^k + \boldsymbol{d}^k) < f(\boldsymbol{x}^k)$, i.e., $\boldsymbol{d}^k$ is the descent direction. This trust region subproblem can be efficiently solved (question: which method? curious about it)

.3.4 Implementation of Least Squares Problem

Consider solving the *nonlinear least square problem* (NLSP)

$$\min \quad f(x) = \tfrac{1}{2} \textstyle\sum_{i=1}^m f_i^2(x)$$

Firstly note that

$$\nabla f(x) = \sum_{i=1}^{m} f_i(x) \nabla f_i(x)$$
$$\nabla^2 f(x) = \sum_{i=1}^{m} [\nabla f_i(x) \nabla^{\mathrm{T}} f_i(x) + f_i(x) \nabla^2 f_i(x)]$$

The so-called Gauss-Newton method is a *quasi-Newton's method*, specialized to this NLSP:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha_k \left(\sum_{i=1}^{m} \nabla f_i(\boldsymbol{x}^k) \nabla^{\mathrm{T}} f_i(\boldsymbol{x}^k) \right)^{-1} \left(\sum_{i=1}^{m} f_i(\boldsymbol{x}^k) \nabla f_i(\boldsymbol{x}^k) \right)$$

Remark .11. It works well when f_i's are not *too linear*, or when at the optimality, f_i's are close to zero.

A variantion of the Gauss-Newton's method operates as follows:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha_k \left(\sum_{i=1}^{m} \nabla f_i(\boldsymbol{x}^k) \nabla^{\mathrm{T}} f_i(\boldsymbol{x}^k) + \lambda_k \boldsymbol{I} \right)^{-1} \left(\sum_{i=1}^{m} f_i(\boldsymbol{x}^k) \nabla f_i(\boldsymbol{x}^k) \right)$$

which is called the *Levenberg-Marquardt method*.

Note that if consider solving the equation $f_{1:n}(\boldsymbol{x}_{1:n}) = \boldsymbol{0}$, the Gauss-Newton direction is just the Newton direction itself.