Supervised Descent Method

Xuehan Xiong

CMU-RI-TR-15-28

September 2015

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Fernando De la Torre, Chair
Srinivasa Narasimhan
Kris Kitani
Aleix Martinez

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Robotics.

Keywords: nonlinear optimization, global optimization, non-convex optimization, nonlinear least squares, face alignment, facial feature tracking, pose estimation, inverse kinematics, imitation learning, learning from demonstration, policy derivation, neural network, stacking, gradient boosting, sequential prediction

For my mom, dad, and Jing

Abstract

In this dissertation, we focus on solving Nonlinear Least Squares problems using a supervised approach. In particular, we developed a Supervised Descent Method (SDM), performed thorough theoretical analysis, and demonstrated its effectiveness on optimizing analytic functions, and four other real-world applications: Inverse Kinematics, Rigid Tracking, Face Alignment (frontal and multi-view), and 3D Object Pose Estimation.

In Rigid Tracking, SDM was able to take advantage of more robust features, such as, HoG and SIFT. Those non-differentiable image features were out of consideration of previous work because they relied on gradient-based methods for optimization. In Inverse Kinematics where we minimize a non-convex function, SDM achieved significantly better convergence than gradient-based approaches. In Face Alignment, SDM achieved state-of-the-arts results. Moreover, it was extremely computationally efficient, which makes it applicable for many mobile applications. In addition, we provided a unified view of several popular methods including SDM on sequential prediction, and reformulated them as a sequence of function compositions. Finally, we suggested some future research directions on SDM and sequential prediction.

Acknowledgments

The work in this document is in collaboration with my advisor, Fernando De la Torre. My foremost thanks give to him for his adequate support and guidance to enable the thesis to be completed. I would also like to thank Srinivasa Narasimhan, Kris Kitani, and Aleix Martinez for their interests in my work and for serving on my committee. I appreciate the time they set aside for me and am thankful for the invaluable discussions and comments regarding this work.

I am also grateful to the National Science Foundation (NSF) and Federal High Way Administration (FHWA), who supported me through grant RI-1116583, CPS-0931999, and award number, DTFH61-14-C-00001.

I want to thank professor Andrew Bagnell for an insightful discussion during our lunch. I am grateful for Martial Hebert and my Masters advisor Daniel Huber. Without their references, I would not be enrolled into the PhD program. I want to thank Ada Zhang for proofreading my papers and giving each of them the most detailed comments ever. I want to thank all my labmates for all the wonderful time I spent the last four years in the Human Sensing lab.

I want to thank my parents for sending me to study abroad and their unconditional love and support from the beginning of my birth. I want to thank Coach James Li for recruiting me from China to the University of Arizona. I want to thank my girlfriend Jing Zhou for being my soul-mate, accompanying me every single step along the way and offering opinions on every single decision I faced. I also want to thank Yuandong Tian for referring me to FAIR and Jia Li for recruiting me to Snapchat Research. I cannot wait to start another journey.

Last not the least, many thanks to Wen-Sheng Chu, Zehua Huang, Feng Zhou, Supreeth Achar, and Francisco Vicente for being friends of mine whom I can always find to talk to in tough times.

Contents

List of Figures

Notation

$\mathbb{R}$	real numbers
$\mathbb{R}^{n\times m}$	set of $n \times m$ real matrices
$\mathbf{0}$	null vector, null matrix; denoted also as $\mathbf{0}_n$ or $\mathbf{0}_{n\times m}$
$\mathbf{1}_n$	column vector of ones
$\mathbf{X}$	bold capital letters denote a matrix
$\mathbf{x}$	bold lower-case letters denote a column vector
x	all non-bold letters represent scalars
$\mathbf{x}_i$	the i^{th} column of matrix $\mathbf{X}$.
x_{ij}	the scalar in the i^{th} row and j^{th} column of matrix $\mathbf{X}$
x_j	the scalar in the j^{th} element of $\mathbf{x}$.
$\mathbf{I}_n \in \mathbb{R}^{n\times n}$	identity matrix.
$\mathbf{1}_{m\times n} \in \mathbb{R}^{m\times n}$	matrix of all ones
$\mathbf{0}_{m\times n} \in \mathbb{R}^{m\times n}$	matrix of all zeros
$\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^T\mathbf{x}}$	the Euclidean distance.
$\|\mathbf{X}\|_F = \sqrt{\text{tr}(\mathbf{X}^T\mathbf{X})}$	the Frobenious norm of a matrix
$\det(\mathbf{X})$ or $\|\mathbf{X}\|$	determinant of matrix $\mathbf{X}$
$\mathbf{X} \circ \mathbf{Y}$	Hadamard products of matrices
$\mathbf{X} \otimes \mathbf{Y}$	Kronecker products of matrices
$\mathbf{X} \oslash \mathbf{Y}$	element-wise division of two matrices
$\mathbf{X} \succeq 0$	semi-positive definite matrix
$\mathbf{X} \succ 0$	positive definite matrix
$\mathbf{X} \preceq 0$	semi-negative definite matrix
$\mathbf{X} \prec 0$	negative definite matrix
$\text{trace}(\mathbf{X})$	trace of a matrix
$\mathbf{X}(i,j)$	submatrix of $\mathbf{X}$, obtained by deleting row i and column j from $\mathbf{X}$

Chapter 1

Introduction

"The intuitive mind is a sacred gift and the rational mind is a faithful servant. We have created a society that honors the servant and has forgotten the gift."

Albert Einstein

1.1 Motivation

Mathematical optimization plays a fundamental role in solving many problems in computer vision. This is evidenced by the significant number of papers using optimization techniques published in any major computer vision conferences. Many important problems in computer vision, such as structure from motion, image alignment, optical flow, or camera calibration can be posed as nonlinear optimization problems. There are a large number of different approaches for solving these continuous nonlinear optimization problems based on first and second order methods, such as gradient descent [1] for dimensionality reduction, Gauss-Newton for image alignment [12, 32, 59] or Levenberg-Marquardt (LM) [61] for structure from motion [19].

Despite many centuries of history, Newton's method and its variants (*e.g.*, Quasi-Newton methods [11, 18, 20]) are regarded as powerfull optimization tools for finding local minima/maxima of smooth functions when second derivatives are available. Newton's method makes the assumption that a smooth function $f(\mathbf{x})$ can be well approximated by a quadratic function in a neighborhood of the minimum. If the Hessian is positive definite, the minimum can be found by solving a system of linear equations. Given an initial estimate $\mathbf{x}_0 \in \mathbb{R}^{p\times 1}$, Newton's method creates a sequence of updates as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}^{-1}(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k), \tag{1.1}$$

where $\mathbf{H}(\mathbf{x}_k) \in \mathbb{R}^{p\times p}$ and $\mathbf{J}(\mathbf{x}_k) \in \mathbb{R}^{p\times 1}$ are the Hessian matrix and Jacobian matrix evaluated at $\mathbf{x}_k$. In the case of Quasi-Newton methods, $\mathbf{H}^{-1}$ can be approximated by analyzing successive gradient vectors. Newton-type methods have two main advantages over competitors. First, they are guaranteed to converge to a local minima provided that, in the neighborhood of the minimum, the Hessian is invertible and the minimizing function is Lipschitz continuous. Second, the convergence rate is quadratic.

However, when applying Newton's method to computer vision problems, three main problems arise: (1) The Hessian is positive definite at the local minimum, but it may not be positive definite elsewhere; therefore, the Newton steps may not be in the descent direction. LM addressed this issue by adding a damping factor to the Hessian matrix. This increases the robustness of the algorithm but reduces the convergence rate. (2) Newton's method requires the function to be twice differentiable. This is a strong requirement in many computer vision applications. For instance, the popular SIFT [55] or HoG [31] features are non-differentiable image operators. In these cases, we can estimate the gradient or the Hessian numerically, but this is typically computationally expensive. (3) The dimension of the Hessian matrix can be large; inverting the Hessian requires $O(p^3)$ operations and $O(p^2)$ in space, where p is the dimension of the parameter to estimate. Although explicit inversion of the Hessian is not needed using Quasi-Newton methods, it can still be computationally expensive to use these methods in computer vision problems. In order to address previous limitations, we proposed the idea of learning descent directions (and rescaling factors) in a supervised manner with a new method called Supervised Descent Method (SDM).

Consider Fig. 1.1 where the goal is to minimize a nonlinear function $f(\mathbf{x})$, where $\mathbf{x}$ is the vector of parameters to optimize. The z-axis has been reversed for visualization purposes. The left image shows the optimization trajectory following the Newton's method. The traditional Newton update has to compute the Hessian and the Jacobian at each step. The right image il-

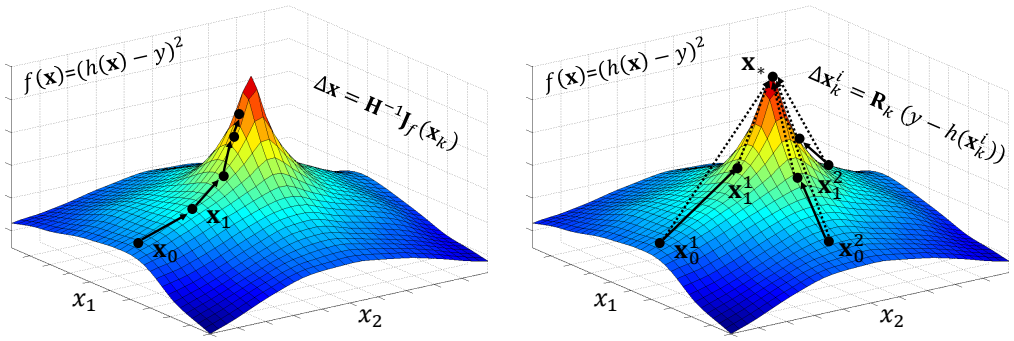


Figure 1.1: a) Newton's method to minimize $f(\mathbf{x})$. The z-axis is reversed for visualization purposes. b) SDM learns a sequence of generic descent maps $\{\mathbf{R}_k\}$ from the optimal optimization trajectories (dotted lines). Each parameter update $\Delta\mathbf{x}^i$ is the product of $\mathbf{R}_k$ and a sample-specific component $(y - h(\mathbf{x}_k^i))$.

lustrates the main idea behind SDM. SDM proposes an offline learning step where it iteratively learns a series of generic Descent Map (DM) from the optimal optimization trajectories (indicated by the dotted lines). In testing, the same DM is used for driving an unseen sample to $\mathbf{x}_*$. As we will argue in this thesis, SDM provides faster and better optimization strategies than traditional gradient-based methods in several computer vision problems, and could have broader applicability to many other problems beyond computer vision.

1.2 Contributions

1. **Theoretical analysis of (G)SDM:** First, we introduced a novel concept, generic DM for minimizing NLS functions and verify this concept using a few analytic functions. Second, we derived the conditions under which SDM will converge. Third, we developed a relaxed and a generalized versions of SDM, which make it applicable for more applications. For functions with multiple local minima, we extended the concept of DM and proved that there existed a finite partition of the function domain such that a separate DM existed within each subset. Fourth, (G)SDM can also be used to provide a better initialization for gradient-based methods. We name this idea as a Hybrid Descent Method. Finally, we established the connection between SDM and Imitation Learning (IL). More explicitly, DM can be interpreted as an optimization policy in the context of IL.
2. **Applications of (G)SDM:** We presented detailed algorithms to address three problems in computer vision and another one in Robotics: rigid tracking, face alignment (frontal and multi-view), 3D pose estimation, and Inverse Kinematics. We compared SDM with several gradient-based methods on minimizing non-convex analytic functions, and showed better convergence results than all the competing methods. In tracking, SDM was able to take advantage of more robust features, such as, HoG and SIFT. Those non-differentiable image features were out of consideration of previous work [6, 58] because they relied on gradient-based methods for optimization. SDM achieved state-of-the-arts results in facial feature detection and tracking. Moreover, it was extremely computationally efficient, which makes it applicable for many mobile applications.

3. **A unified framework for sequential prediction algorithms:** We reviewed seven representative methods on sequential prediction. Also, we discussed the differences between each of them and SDM, and provided a unified view of all methods including SDM as a sequence of function compositions. Additionally, we argued why sequential prediction was preferable to performing inference using a single complex model in solving real-world problems. Finally, we suggested some future research directions on sequential predictions.
4. **Dataset and software:** We built a challenging public dataset and also proposed an evaluation protocol for benchmarking facial feature tracking methods. Both evaluation code and dataset can be downloaded from the link below[1]. To the best of our knowledge, this is the first public dataset for evaluating facial feature tracking on profile-to-profile faces. We integrated the SDM-based tracker into IntraFace[2], a free software package for facial image analysis research. The software has accumulated 4800 downloads in eight months that was active, and it has proven useful for many researchers working on facial image analysis.

1.3 Organization

In this chapter, we discuss the motivation behind my thesis and list our contributions to the research community. In Chapter 2, we present theoretical analysis and algorithms of SDM and a few of its extensions. We establish the connection between SDM and previous sequential prediction methods, and also provide a unified view of them. In Chapter 3, we review previous work of three classical problems in computer vision and present SDM's solutions to all of them. In Chapter 4, we present the experimental results on the above applications using (G)SDM and its comparison with state-of-the-arts methods, followed by a discussion of failure cases. In addition, we compare (G)SDM with several gradient-based methods on minimizing analytic functions and solving the problem of Inverse Kinematics. In Chapter 5, we summarize this thesis and discuss possible extensions and future research directions of SDM.

[1] `www.humansensing.cs.cmu.edu/xxiong`
[2] `www.humansensing.cs.cmu.edu/intraface`

Chapter 2

Theory and Methods

"There is a way to do it better - find it."

Thomas Edison

2.1 Theory of SDM

This section provides the theoretical basis behind SDM. Section 2.1.1 reviews mathematical definitions. Section 2.1.2 illustrates the ideas behind SDM and DM using one-dimensional functions, and Section 2.1.3 extends them to high-dimensional functions. Section 2.1.4 and Section 2.1.5 derive practical algorithms for different situations. Section 2.1.6 extends SDM to handle functions with multiple local minima.

2.1.1 Background

Before deriving SDM, we review two concepts, *Lipschitz continuous* [4] and *monotone operator* [73] and their respective properties.

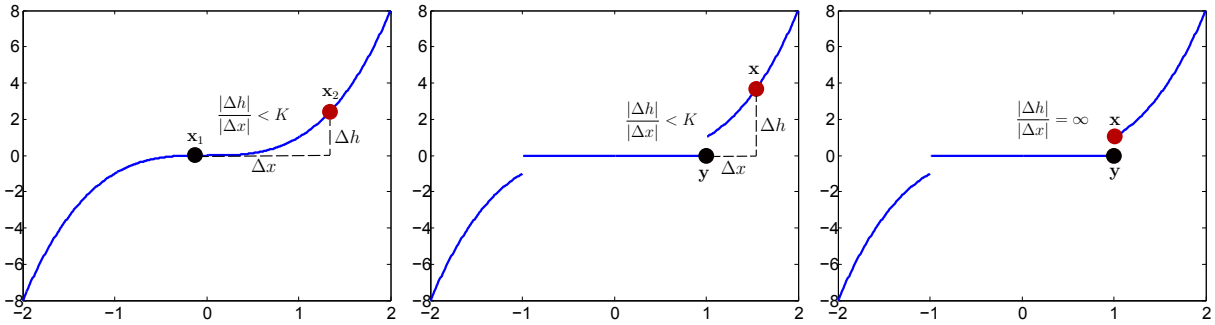
Definition 1. A function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ is called *Lipschitz continuous* if there exists a real constant $K \geq 0$ such that

$$\|\mathbf{f}(\mathbf{x}_1) - \mathbf{f}(\mathbf{x}_2)\|_2 \leq K\|\mathbf{x}_1 - \mathbf{x}_1\|_2, \forall \mathbf{x}_1, \mathbf{x}_2,$$

where the smallest K is referred as the Lipschitz constant. If $K = 1$, the function $\mathbf{f}$ is said to be *nonexpansive*. If $K < 1$, the function is called a *contraction*.

Here we limit the metric space to be L_2 to simplify the upcoming theoretical analysis, but other distance metrics can be used. The following are some well-known properties:

1. Composition of nonexpansive operators is nonexpansive;
2. Composition of nonexpansive operator and contraction is contraction.



(a) (b) (c)

Figure 2.1: a) Function $h(x)$ is Lipschitz continuous in the domain of $[-2, 2]$; b) Function $h(x)$ is not Lipschitz continuous over $[-2, 2]$, but it is at the point $\mathbf{x}$; c) Function $h(x)$ is neither continuous in the domain $[-2, 2]$ nor at the point $\mathbf{x}$.

Definition 2. A function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ is called to have a *Lipschitz constant* K *at a point* $\mathbf{x}$ if there exists a real constant $K \geq 0$ such that

$$\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y})\|_2 \leq K\|\mathbf{x} - \mathbf{y}\|_2, \forall \mathbf{y}.$$

Notice that the previous definition is defined over any two points within a certain domain while this one is defined for one fixed point. Fig. 2.1 provides a better illustration of understanding

the differences between the two definitions. A function being Lipschitz continuous in domain X implies that it is continuous at any point $\mathbf{x} \in X$. In other words, definition 1 is the sufficient condition of definition 2.

Definition 3. A function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is called a *monotone operator* if

$$\langle \mathbf{x}_1 - \mathbf{x}_2, \mathbf{f}(\mathbf{x}_1) - \mathbf{f}(\mathbf{x}_2) \rangle \geq 0, \forall \mathbf{x}_1, \mathbf{x}_2,$$

where $\langle \cdot \rangle$ represents the inner product operator.

Here are some basic properties [15] about monotone operators: if F and G are monotone,

1. $F + G$ is monotone;
2. if $\alpha \geq 0$, then αF is monotone;
3. F^{-1} is monotone;
4. for $T \in \mathbb{R}^{n \times m}$, $T^\top F(Tz)$ is monotone (on $\mathbb{R}^m$).

Some well-known monotone operators [15] are:

1. If f is convex closed proper (CCP) then $F(\mathbf{x}) = \partial f(\mathbf{x})$ is maximal monotone.
2. KKT operator is monotone.

Definition 4. A function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is called a *monotone operator at a point* $\mathbf{x}$ if

$$\langle \mathbf{x} - \mathbf{y}, \mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y}) \rangle \geq 0, \forall \mathbf{y}.$$

Fig. 2.2 illustrates the differences between the above two definitions. A function being a monotone operator in domain X implies that it is a monotone operator at any point $\mathbf{x} \in X$. In other words, definition 3 is the sufficient condition of definition 4.

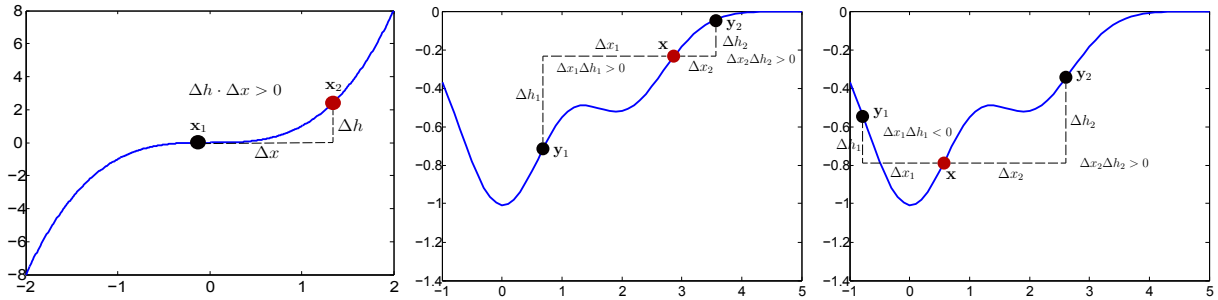


Figure 2.2: a) Function $h(x)$ is monotonically increasing in the domain of $[-2, 2]$; b) Function $h(x)$ is not monotonic in the domain of $[-1, 5]$, but it is a monotone operator at point $\mathbf{x}$; c) Function $h(x)$ is neither monotonic within the domain $[-1, 5]$ nor a monotone operator at point $\mathbf{x}$.

2.1.2 One-dimensional Case

This section derives the theory for SDM in 1D functions. Given a 1D NLS problem,

$$\min_x f(x) = \min_x (h(x) - y)^2, \tag{2.1}$$

where $h(x)$ is a nonlinear function and y is a known scalar. Applying the chain rule to Eq. 2.1, the gradient descent update yields

$$x_k = x_{k-1} - \alpha h'(x_{k-1})(h(x_{k-1}) - y). \tag{2.2}$$

Optimizing Eq. 2.1 using gradient-based methods follows Eq. 2.2, but have different ways to compute the step size α. For example, Newton's method sets $\alpha = \frac{1}{f''(x_{k-1})}$. Computing the step size and gradient direction in high-dimensional spaces is computationally expensive and can be numerically unstable, especially in the case of non-differentiable functions, where finite differences are required to compute estimates of the Hessian and Jacobian. The main idea behind SDM is to avoid explicit computation of the Hessian and Jacobian and learn the "descent directions" ($\alpha h'(x_{k-1})$) from training data. During training, SDM samples many different initial configurations in the parameter space $\{x_0^i\}_i$ and learns a constant $r \sim \alpha h'(x_{k-1})$, which drives all samples towards the optimal solution x_*. We define r more formally below.

Definition 5. A scalar r is called a *generic DM* if there exists a scalar $0 < c < 1$ such that $\forall x \in U(x_*)$, $|x_* - x_k| \leq c|x_* - x_{k-1}|$. x_k is updated using the following equation:

$$x_k = x_{k-1} - r(h(x_{k-1}) - h(x_*)). \tag{2.3}$$

The existence of a generic DM is guaranteed when both of the following conditions hold:

1. $h(x)$ is strictly monotonic at x_*.
2. $h(x)$ has Lipschitz constant K at x_*.

A detailed proof is presented in Appendix (Theorem 1). Interestingly, the above two conditions are closely related to the essence of a generic DM. The update rule (Eq. 2.3) can be split into two terms: (1) $r \sim \alpha h'(x_{k-1})$ (generic DM) that is sample independent, and (2) $(h(x_{k-1}) - y)$ that is sample dependent. r contains only part of the descent direction and needs to be multiplied by $h(x_{k-1}) - y$ to produce a descent direction. Condition 1 ensures that $h'(x)$ does not change signs around x_*, while condition 2 constrains the smoothness of the function, putting an upper bound on step sizes.

$h(x)$	K	r	x_*
$h_1 = \frac{1}{x}$	$-\frac{1}{3}$	-6	3
$h_2 = -e^{-x^2} - \frac{1}{2}e^{-(x-2)^2}$	0.41	4.82	3
$h_3 = \begin{cases} x & x > 2 \\ 0 & 2 \geq x > 0 \\ x - 1 & x \leq 0 \end{cases}$	3	0.67	3

Figure 2.3: Experimental setup

In Fig. 2.4, we illustrate how to minimize three different functions using a generic DM. Fig. 2.3 describes our experimental setup: our choices for three different functions $\{h(x)\}$, the optimal values $\{x_*\}$, Lipschitz constants $\{K\}$, and their corresponding generic DMs $\{r\}$. According to Theorem 1, the DM r is set to be $\text{sign}(h')(\frac{2}{K} - \epsilon)$, where ϵ is a small positive number. The Lipschitz constant K is computed numerically in a neighborhood of x_*. Figs. 2.4c plot the

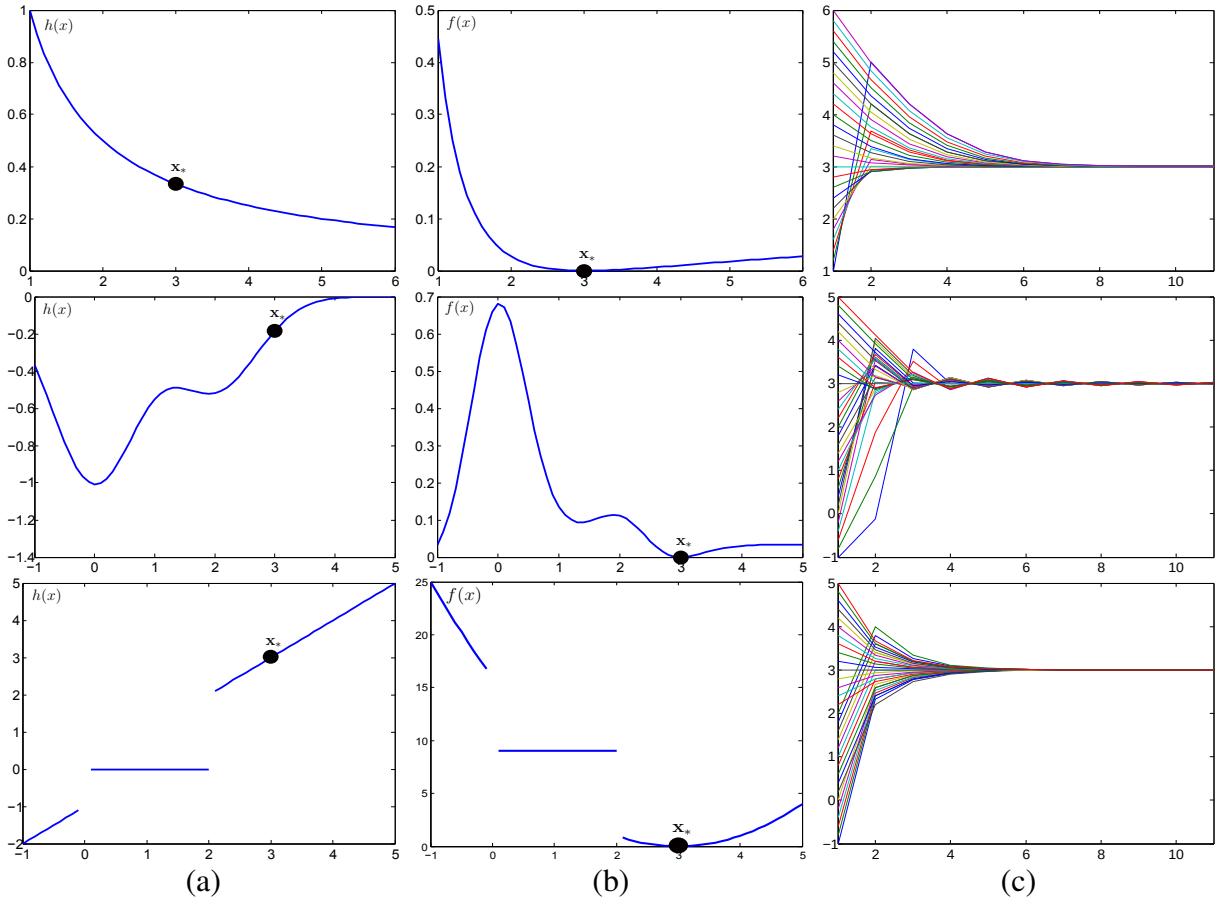


Figure 2.4: a) Functions $\{h_i\}$; b) Objective functions $\{f\}$ and x_* is the solution; c) Traces of optimizing parameter throughout iterations of SDM. y-axis shows the values of x in each iteration and x-axis indicates number of iterations.

traces of the optimizing parameters for each function where x_k is updated following Eq. 2.3. Note that SDM always converges to the optimal value x_*, regardless of its initial value x_0. The second row in Fig. 2.4 shows that SDM finds the minimum in a function with local minima. Given a x_*, for SDM to converge function $h(x)$ is required to be monotonic at x_*, which is a much weaker condition than being monotonic in the whole domain. The third row in Fig. 2.4 shows that SDM can be even used for optimizing non-smooth/non-continuous functions. This is possible because the convergence condition only requires $h(x)$ to be Lipschitz continuous at x_*, not necessarily over the whole domain.

2.1.3 Multi-dimensional Case

This section extends the concept of generic DM to multi-dimensional functions, where $\mathbf{h} : \mathbb{R}^n \rightarrow \mathbb{R}^m$. For multi-dimensional NLS functions, the gradient descent update in Eq. 2.2 becomes

$$\mathbf{x}_k = \mathbf{x}_{k-1} - \alpha \mathbf{A} \mathbf{J}_{\mathbf{h}}^{\top}(\mathbf{x}_{k-1})(\mathbf{h}(\mathbf{x}_{k-1}) - \mathbf{y}) \tag{2.4}$$

where $\mathbf{J_h}(\mathbf{x}) \in \mathbb{R}^{m\times n}$ is the Jacobian matrix, $\mathbf{A}_{n\times n}$ is the identity ($\mathbf{I}_n$) in first order gradient methods, and the inverse Hessian (or an approximation) for second-order methods, α is the step size. A generic DM $\mathbf{R}$ exists if there exists a scalar $0 < c < 1$ such that

$$\|\mathbf{x}_* - \mathbf{x}_k\|_2 \leq c\|\mathbf{x}_* - \mathbf{x}_{k-1}\|_2 . \forall \mathbf{x} \in U(\mathbf{x}_*)$$

The update rule of Eq. 2.3 becomes

$$\mathbf{x}_k = \mathbf{x}_{k-1} - \mathbf{R}(\mathbf{h}(\mathbf{x}_{k-1}) - \mathbf{h}(\mathbf{x}_*)). \tag{2.5}$$

In Appendix(Theorem 3), we prove that the existence of a generic DM if both of the following conditions hold:

1. $\mathbf{Rh}(\mathbf{x})$ is a monotone operator at $\mathbf{x}_*$.
2. $\mathbf{h}(\mathbf{x})$ has Lipschitz constant K at $\mathbf{x}_*$.

We have stated the conditions that ensure the existence of a generic DM. However, three issues arise when applying the above idea to real-world applications:

1. The above two conditions may not hold;
2. Convergence rate may be slow since the most conservative step is taken at every iteration to ensure that all samples are progressing towards $\mathbf{x}_*$;
3. Samples may form very different distributions throughout each iteration, *i.e.*, initially, samples are coming from a uniform distribution, but it is unlikely to stay uniform after one update. See Fig. 2.4c for examples. Samples become more and more concentrated as iteration increases. DM should be re-computed in each iteration.

2.1.4 Relaxed SDM

In this section, we introduce a relaxed version of the generic DM and derive a practical algorithm. Previously, a single matrix $\mathbf{R}$ was used for **all** samples. In this section, we extend SDM to learn a sequence of $\{\mathbf{R}_k\}$ that moves the **expected value** of $\mathbf{x}_k$ towards the optimal solution $\mathbf{x}_*$. The relaxed SDM is a sequential algorithm that learns such DMs from training data.

Let us denote X to be a random variable representing the state of $\mathbf{x}$. In the first iteration ($k = 0$), we assume that X_0 is coming from a known distribution $X_0 \sim P_0$ and use lower case p_0 to represent its probability density function. The first DM $\mathbf{R}_0$ is computed by minimizing the expected loss between the true state and the predicted states, given by

$$\begin{aligned}
&\mathbf{E}\|\mathbf{x}_* - X_1\|_2^2 = \mathbf{E}\|\mathbf{x}_* - X_0 + \mathbf{R}_0(\mathbf{h}(X_0) - \mathbf{h}(\mathbf{x}_*))\|_2^2 \\
&= \int_{\mathbf{x}_0} \|\mathbf{x}_* - \mathbf{x}_0 + \mathbf{R}_0(\mathbf{h}(\mathbf{x}_0^i) - \mathbf{h}(\mathbf{x}_*))\|_2^2 p_0(\mathbf{x}_0) d\mathbf{x}_0 \\
&\approx \sum_i \|\mathbf{x}_* - \mathbf{x}_0^i + \mathbf{R}_0(\mathbf{h}(\mathbf{x}_0^i) - \mathbf{h}(\mathbf{x}_*))\|_2^2.
\end{aligned} \tag{2.6}$$

Here we have used Monte Carlo sampling to approximate the integral, and $\mathbf{x}_0^i$ is drawn from the distribution P_0. $\mathbf{x}_*, \mathbf{h}(\mathbf{x}_*)$ are known in training and minimizing Eq. 2.6 is simply a linear least squares problem, which can be solved in closed-form.

It is unlikely that the first generic DM can achieve the desired minimum in one step for all initial configurations. As in Newton's method, after an update, we recompute a new generic map (*i.e.*, a new Jacobian and Hessian in Newton's method). In iteration k, with $\mathbf{R}_{k-1}$ estimated, each sample $\mathbf{x}_{k-1}^i$ is updated to its new location $\mathbf{x}_k^i$ as follows:

$$\mathbf{x}_k^i = \mathbf{x}_{k-1}^i - \mathbf{R}_{k-1}(\mathbf{h}(\mathbf{x}_{k-1}^i) - \mathbf{h}(\mathbf{x}_*)). \tag{2.7}$$

This is equivalent to drawing samples from the conditional distribution $P(X_k|X_{k-1} = \mathbf{x}_{k-1}^i)$. We can use the samples to approximate the expected loss as follows:

$$\begin{aligned}
&\mathbf{E}\|\mathbf{x}_* - X_{k+1}\|_2^2 = \mathbf{E}\|\mathbf{x}_* - X_k + \mathbf{R}_k(\mathbf{h}(X_k) - \mathbf{h}(\mathbf{x}_*))\|_2^2 \\
&= \int_{\mathbf{x}_k} \|\mathbf{x}_* - \mathbf{x}_k + \mathbf{R}_k(\mathbf{h}(\mathbf{x}_k) - \mathbf{h}(\mathbf{x}_*))\|_2^2 p(\mathbf{x}_k|\mathbf{x}_{k-1}) d\mathbf{x}_k \\
&\approx \sum_i \|\mathbf{x}_* - \mathbf{x}_k^i + \mathbf{R}_k(\mathbf{h}(\mathbf{x}_k^i) - \mathbf{h}(\mathbf{x}_*))\|_2^2.
\end{aligned} \tag{2.8}$$

Minimizing Eq. 2.8 yields the k^{th} DM. During learning, SDM alternates between minimizing Eq. 2.8 and updating Eq. 2.7 until convergence. In testing, $\mathbf{x}_k$ is updated recursively using Eq. 2.7 with learned DMs.

2.1.5 Generalized SDM

Thus far, we have assumed that $\mathbf{y} = \mathbf{h}(\mathbf{x}_*)$ is the same in training as in testing (*e.g.*, template tracking). This section presents generalized SDM, which addresses the case where $\mathbf{y}$ differs in training and testing. In this case, the generic DM $\mathbf{R}$ is not defined for one particular $\mathbf{x}_*$ but defined over a domain X. We say that $\mathbf{R}$ is a generic DM if there exists a scalar $0 < c < 1$ such that

$$\|\mathbf{x}_* - \mathbf{x}_k\|_2 \leq c\|\mathbf{x}_* - \mathbf{x}_{k-1}\|_2. \forall \mathbf{x}_*, \mathbf{x}_0 \in X$$

In Appendix (Theorem 4), we prove that the existence of a generic DM if both of the following conditions hold:

1. $\mathbf{Rh}(\mathbf{x})$ is a monotone operator over domain X.
2. $\mathbf{h}(\mathbf{x})$ has Lipschitz constant K over domain X.

These are stronger conditions than the ones stated in Section 2.1.3. Here is an interesting observation: for DM defined over a particular point $\mathbf{x}_*$, the two sufficient conditions are defined over the same point (see Section 2.1.3). When DM is defined over domain X, two sufficient conditions are defined over the same domain as well. Similar to SDM, a practical algorithm is described below based on the relaxation of the above concept.

The generalized SDM during training starts at the same initial point $\mathbf{x}_0$ and samples different $\{\mathbf{x}_*^i\}$ around it. At the same time, for each sample we compute $\mathbf{y}^i = \mathbf{h}(\mathbf{x}_*^i)$. The training procedure remains the same as stated in the previous section, except we replace $\mathbf{h}(\mathbf{x}_*)$ in Eq. 2.7 with $\mathbf{y}^i$,

$$\mathbf{x}_k^i = \mathbf{x}_{k-1}^i - \mathbf{R}_{k-1}(\mathbf{h}(\mathbf{x}_{k-1}^i) - \mathbf{y}^i)), \tag{2.9}$$

and we replace $\mathbf{x}_*, \mathbf{h}(\mathbf{x}_*)$ in Eq. 2.8 with $\mathbf{x}_*^i, \mathbf{y}^i$,

$$\sum_i \|\mathbf{x}_*^i - \mathbf{x}_k^i + \mathbf{R}_k(\mathbf{h}(\mathbf{x}_k^i) - \mathbf{y}^i))\|_2^2. \tag{2.10}$$

In testing, we start SDM with the same initial value $\mathbf{x}_0$ that we used in training and recursively update parameters $\mathbf{x}$ using Eq. 2.9.

2.1.6 Global SDM

In this section, we extend SDM to deal with multiple local minima. For a function f with a unique minimum, the gradients of $\mathbf{h}$ often share similar directions. Therefore, a weighted average can be learned. When dealing with a function f with several local minima, the gradients of $\mathbf{h}$ are likely to have conflicting directions so averaging them is not adequate and it may cause the SDM training to stall.

We will bypass this problem by learning **not one** but **a set** of DMs. In the Appendix A (Theorem 5), we prove that it is possible to find a partition of domain $\mathbf{x}$, $S = \{S^t\}_1^T$, such that there exists a generic DM $\mathbf{R}^t$ within each subset S^t. The subsets of this partition are defined as *Domains of Homogeneous Descent (DHD)*. We will show an example of how we find DHD for a function $\mathbf{h} : \mathbb{R}^2 \to \mathbb{R}^2$. In this example, a set of four subsets is created. Let $\mathbf{g} = (\mathbf{x}_* - \mathbf{x}) \circ (\mathbf{y}_* - \mathbf{h}(\mathbf{x}))$ where $\mathbf{x}$ is current parameter. Each $\mathbf{x}$ falls in one of the four partitions based on the following criteria:

$$\begin{aligned}
\mathbf{x} &\in S^1 \quad \text{if } g_1 \leq 0 \wedge g_2 \leq 0, \\
\mathbf{x} &\in S^2 \quad \text{if } g_1 \leq 0 \wedge g_2 > 0, \\
\mathbf{x} &\in S^3 \quad \text{if } g_1 > 0 \wedge g_2 \leq 0, \\
\mathbf{x} &\in S^4 \quad \text{if } g_1 > 0 \wedge g_2 > 0.
\end{aligned}$$

In Fig. 2.5 we plot four NLS functions along with their DHD found by following the strategy above. Different subsets within DHD are colored in different grayscales. Interestingly, local minima are located at the intersections of different domains. Note that DHD need to be re-computed in every iteration since they depend on the current estimates of $\mathbf{x}$ and $\mathbf{h}(\mathbf{x})$.

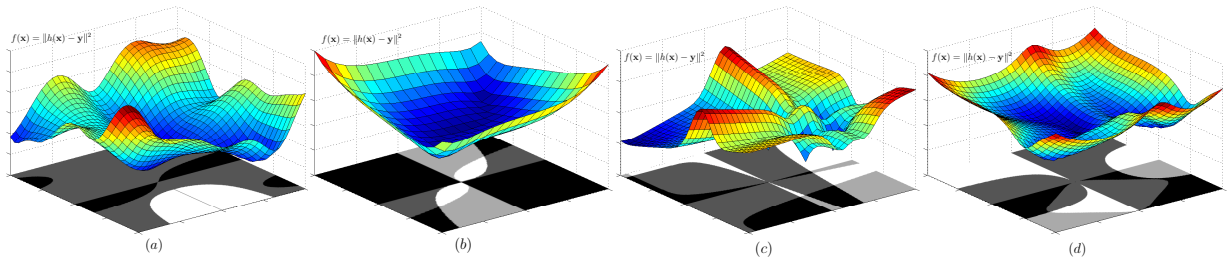


Figure 2.5: Illustration of DHD on four NLS functions where $\mathbf{h}(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}^2$. Different domains are colored in different grayscales.

DHD is a pure conceptual idea. It exists but cannot be computed during testing time since we do not know $\mathbf{x}_*$. Therefore, we developed two approximation strategies. The first one is to

approximate $(\mathbf{x}_* - \mathbf{x})$ by $(\mathbf{x}_k - \mathbf{x}_{k-1})$ where $\mathbf{x}_{k-1}$ and $\mathbf{x}_k$ are the consecutive evaluations of $\mathbf{x}$. The details of how we apply this strategy to solve analytic functions can be found in Section 4.1. In the second strategy, we focus on tracking application where we use the previous estimate as a rough estimate for $\mathbf{x}_*$. In Section 3.3.2. we derive a practical algorithm with an application to track faces from profile to profile.

2.2 Connection with Previous Work

In this section, we review a handful of representative methods on sequential prediction. Also, we will discuss the differences between each of them and SDM, and provide a unified view of all methods including SDM as a sequence of functions compositions.

2.2.1 SDM Revisited

Let us revisit the update formula 2.7 of SDM and redefine it using a function $\mathbf{f}$, parametrized by $\mathbf{W} = \{\mathbf{R}, \mathbf{y}_*\}$:

$$\mathbf{f}_{\mathbf{W}}(\mathbf{x}) = \mathbf{f}(\mathbf{x}; \mathbf{R}, \mathbf{y}_*) = \mathbf{x} + \mathbf{R}(\mathbf{y}_* - \mathbf{h}(\mathbf{x})). \tag{2.11}$$

Assuming SDM converges to $\mathbf{x}_*$ in k iterations, below we reformulate SDM's inference as a sequence of function compositions:

$$\begin{array}{rclcll}
\mathbf{x}_1 & = & \mathbf{x}_0 & + & \mathbf{R}_0(\mathbf{y}_* - \mathbf{h}(\mathbf{x}_0)) & = \mathbf{f}_{\mathbf{W}_0}(\mathbf{x}_0) \\
\mathbf{x}_2 & = & \mathbf{x}_1 & + & \mathbf{R}_1(\mathbf{y}_* - \mathbf{h}(\mathbf{x}_1)) & = \mathbf{f}_{\mathbf{W}_1}(\mathbf{x}_1) \\
 & & & & & = \mathbf{f}_{\mathbf{W}_1} \circ \mathbf{f}_{\mathbf{W}_0}(\mathbf{x}_0) \\
\mathbf{x}_3 & = & \mathbf{x}_2 & + & \mathbf{R}_2(\mathbf{y}_* - \mathbf{h}(\mathbf{x}_2)) & = \mathbf{f}_{\mathbf{W}_2}(\mathbf{x}_2) \\
 & & & & & = \mathbf{f}_{\mathbf{W}_2} \circ \mathbf{f}_{\mathbf{W}_1} \circ \mathbf{f}_{\mathbf{W}_0}(\mathbf{x}_0) \\
 & \vdots & & \vdots & & \vdots \\
\mathbf{x}_k & = & \mathbf{x}_{k-1} & + & \mathbf{R}_{k-1}(\mathbf{y}_* - \mathbf{h}(\mathbf{x}_{k-1})) & = \mathbf{f}_{\mathbf{W}_{k-1}}(\mathbf{x}_{k-1}) \\
 & & & & & = \mathbf{f}_{\mathbf{W}_{k-1}} \circ \cdots \circ \mathbf{f}_{\mathbf{W}_1} \circ \mathbf{f}_{\mathbf{W}_0}(\mathbf{x}_0)
\end{array}$$

Above, we simply replace $\mathbf{x}_i$ with the previous iteration's result. In SDM, the function parameters $\mathbf{W}$s are learned greedily to minimize between the predicted $\Delta\mathbf{x}$ and the true one.

2.2.2 Gradient Boosting

Gradient Boosting (GB), first invented by J. H. Friedman [38], is a regression method to approximate continuous functions. Like other boosting methods, GB combines weak learners into a single strong learner, in an iterative fashion. Given a training set $\{X, Y\} = \{(x^i, y^i)\}$, GB iteratively fit a weak regressor to the residual by minimizing a loss function L. Then, the multiplier γ is obtained by solving a one-dimensional optimization problem:

$$Y \leftarrow Y - \gamma h(X) \tag{2.12}$$

$$h \leftarrow \arg\min_h L(Y, h(X)) \tag{2.13}$$

$$\gamma \leftarrow \arg\min_\gamma L(Y, \gamma h(X)). \tag{2.14}$$

The above three steps repeats until the residual is small or the maximum iteration is reached. The first weak learner h_0 is often taken to be a constant (*e.g.*, the average of all training samples). The final learner is the sum of all weak regressors,

$$f(x) = \sum_{i=1}^{k} \gamma_i h_i(x) + \text{const.} \tag{2.15}$$

Note that all h_is could be the same parametric regressor but with different parameters. The parameters is implicitly hidden in this notation.

Function additions can be represented as function compositions. Let us define a new set of functions:

$$f_i(y) = \begin{cases} y + \gamma_i h_i(x) & \text{if } i > 1 \\ \text{const} & \text{if } i = 1 \end{cases}$$

Note that x is not a variable but a known input. Below, we illustrate how Eq. 2.15 can be rewritten as a sequence of compositions of f_is:

$$\begin{aligned} h_0(x) &= \text{const} && = f_0(x) \\ h_0(x) + \gamma_1 h_1(x) &= f_0(x) + \gamma_1 h_1(x) && = f_1(f_0(x)) \\ h_0(x) + \gamma_1 h_1(x) + \gamma_2 h_2(x) &= f_1(f_0(x)) + \gamma_2 h_2(x) && = f_2(f_1(f_0(x))) \\ \vdots \quad &\vdots && \vdots \\ h_0(x) + \textstyle\sum_{i=1}^{k} \gamma_i h_i(x) &= f_{k-1} \cdots (f_1(f_0(x)) + \gamma_k h_k(x) && = f_k \cdots (f_1(f_0(x)) \end{aligned}$$

Next, we will reinterpret SDM as a Boosting method and discuss the differences between SDM and GB. Given an input $y_* = h(x_*)$, let us run SDM:

$$\begin{array}{lllll} x_1 & \leftarrow & x_0 & + & r_0(y_* - h(x_0)) \\ x_2 & \leftarrow & x_1 & + & r_1(y_* - h(x_1)) \\ x_3 & \leftarrow & x_2 & + & r_2(y_* - h(x_2)) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_k & \leftarrow & x_{k-1} & + & r_{k-1}(y_* - h(x_{k-1})). \end{array}$$

We can recursively substitute x_i with x_{i-1} until x_0 is reached, which gives us

$$x_k = x_0 + \sum_{i=0}^{k-1} r_i(y_* - h(x_i)). \tag{2.16}$$

We know x_k is the SDM's approximation of x_*. Assuming that h is invertible SDM can be interpreted as an algorithm to approximate inverse function h^{-1}:

$$h^{-1}(y_*) = x_* \approx x_0 + \sum_{i=0}^{k-1} r_i(y_* - h(x_i)). \tag{2.17}$$

Recall that x_0 is set to be a constant in SDM. This makes Eq. 2.17 very similar to Eq. 2.15. Both algorithms approximate a function by a linear combination of base learners. However, there exists some crucial differences between the two methods:

1. Both methods are trained in a sequential manner. In testing, GB does not have to operate in an iterative manner. For example, one can compute all $h_i(x)$ in parallel and combine them in any order. SDM has to wait for x_{k-1} before being able to compute x_k.
2. In SDM, the inverse function is approximated by a linear combination of the *same function* $h(x_i)$ evaluated at *different points* $\{x_i\}$. In Gradient Boosting, the function is approximated by a weighted sum of *different functions* $\{h_i(x)\}$ evaluated at the *same point* x.
3. The choice of the weak learner is arbitrary in GB while in SDM the inverse function is used as the base learner and it does not have to be weak.

2.2.3 Cascaded Pose Regression

Cascaded Pose Regression (CPR) [34] is designed for computing 2D object pose in images. The pose of an object is often defined by a set of landmarks that describes the shape of the object. CPR progressively refines an initial guess of object's pose where each refinement is carried out through a different regressor. CPR extends GB by incorporating the "pose-indexed features". A pose-indexed feature is simply a function that takes the input image as well as the pose of the object and outputs a vector of real values, and we denote it as $\mathbf{h}$. Given an image I and an initial guess of the object pose $\mathbf{x}$, CPR learns a sequence of regressors that move $\mathbf{x}$ towards the human labels $\mathbf{x}_*$ by repeating the following steps:

$$\phi \leftarrow \mathbf{h}(I, \mathbf{x}) \tag{2.18}$$

$$\mathbf{g} \leftarrow \arg\min_{\mathbf{g}} L(\mathbf{x}_* - \mathbf{x}, \mathbf{g}(\phi)) \tag{2.19}$$

$$\mathbf{x} \leftarrow \mathbf{x} + \mathbf{g}(\phi). \tag{2.20}$$

The regressor $\mathbf{g}$ maps the pose-indexed features ϕ to the pose update, and it is learned by minimizing a loss function L between the true pose update and the predicted one. As in SDM/GB, each regressor in CPR is learned greedily, *e.g.*, without considering the potential effects of the future steps. In testing, CPR updates object's pose by applying a sequence of regressors in the same order they are learned.

Next, let us define a new set of functions:

$$\mathbf{f}_i(\mathbf{x}) = \mathbf{x} + \mathbf{g}_i(\mathbf{h}(I, \mathbf{x})). \tag{2.21}$$

We will rewrite CPR as a sequence of compositions of functions $\mathbf{f}_i$s:

$$\begin{aligned}
\phi_0 &= \mathbf{h}(I, \mathbf{x}_0) \\
\mathbf{x}_1 &= \mathbf{x}_0 + \mathbf{g}_0(\phi_0) = \mathbf{x}_0 + \mathbf{g}_0(\mathbf{h}(I, \mathbf{x}_0)) = \mathbf{f}_0(\mathbf{x}_0) \\
\phi_1 &= \mathbf{h}(I, \mathbf{x}_1) \\
\mathbf{x}_2 &= \mathbf{x}_1 + \mathbf{g}_1(\phi_1) = \mathbf{x}_1 + \mathbf{g}_1(\mathbf{h}(I, \mathbf{x}_1)) = \mathbf{f}_1(\mathbf{x}_1) = \mathbf{f}_1 \circ \mathbf{f}_0(\mathbf{x}_0) \\
&\vdots \\
\phi_{k-1} &= \mathbf{h}(I, \mathbf{x}_{k-1}) \\
\mathbf{x}_k &= \mathbf{x}_{k-1} + \mathbf{g}_{k-1}(\phi_{k-1}) = \mathbf{x}_{k-1} + \mathbf{g}_{k-1}(\mathbf{h}(I, \mathbf{x}_{k-1})) = \mathbf{f}_{k-1}(\mathbf{x}_{k-1}) = \mathbf{f}_{k-1} \cdots \circ \mathbf{f}_1 \circ \mathbf{f}_0(\mathbf{x}_0)
\end{aligned}$$

Despite their different origins (CPR is an extension of GB while SDM is inspired by Newton's method) and different choices of regressors (CPR adopts random fern regressor [66] while SDM uses linear regression), the underlying algorithms of SDM and CPR are the same.

2.2.4 Data Driven Descent

Data Driven Descent (DDD) [88, 89], developed by Tian and Narasimhan, is a method for estimating nonrigid deformation of a novel scene according to a given template image. In their problems, they assume that a deformation model $\mathbf{h}(I, \mathbf{x})$ is known, controlled by the deformation parameters $\mathbf{x}$. Given a template image I_t, the method starts by generating a set of training

images $\{I_{tr}\}$ from the template image deformed under a set of random perturbations of $\mathbf{x}$. The training image I_{tr}^i and its parameter $\mathbf{x}_{tr}^i$ are related by the deformation model, $I_{tr}^i = \mathbf{h}(I_t, \mathbf{x}_{tr}^i)$. Then, DDD gradually deforms the test image I_0 to match the template according to the nearest neighbor in the training set. Explicitly, DDD repeats the following two steps until the difference between the deformed test image and the template is small:

$$\begin{aligned}\mathbf{x} &\leftarrow \mathbf{x} + \texttt{knn}_{\{I_{tr},\mathbf{x}_{tr}\}}(I_0)\\ I_0 &\leftarrow \mathbf{h}(I_0, \mathbf{x}).\end{aligned}$$

Function $\texttt{knn}_{\{I_{tr},\mathbf{x}_{tr}\}}(I)$ implements the KNN regressor that takes an input image I and returns the deformation parameter whose corresponding image is the closest neighbor to I. The initial parameters $\mathbf{x}_0$ is set to be $\mathbf{0}$. In addition to DDD, they also proved the conditions under which the method will converge and the sample complexity to guarantee an ϵ error.

We will define a new function to represent the parameter update in DDD:

$$\mathbf{f}(\mathbf{x}; I_0, \{I_{tr}, \mathbf{x}_{tr}\}) = \mathbf{x} + \texttt{knn}_{\{I_{tr},\mathbf{x}_{tr}\}}(\mathbf{h}(I_0, \mathbf{x})). \tag{2.22}$$

Function $\mathbf{f}$ is parametrized by the test image I_0 and training set $\{(I_{tr}, \mathbf{x}_{tr})\}$. In the following derivation, we will omit this parametrization since they remain unchanged given a particular test image and a template during the execution of DDD. Below, we derive how DDD can be rewritten as a sequence of compositions of function $\mathbf{f}$:

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{x}_0 + \texttt{knn}_{\{I_{tr},\mathbf{x}_{tr}\}}(I_0) = \mathbf{f}(\mathbf{x}_0)\\ I_1 &= \mathbf{h}(I_0, \mathbf{x}_1)\\ \mathbf{x}_2 &= \mathbf{x}_1 + \texttt{knn}_{\{I_{tr},\mathbf{x}_{tr}\}}(I_1)\\ &= \mathbf{x}_1 + \texttt{knn}_{\{I_{tr},\mathbf{x}_{tr}\}}(\mathbf{h}(I_0, \mathbf{x}_1)) = \mathbf{f}(\mathbf{x}_1) = \mathbf{f} \circ \mathbf{f}(\mathbf{x}_0)\\ I_2 &= \mathbf{h}(I_1, \mathbf{x}_2)\\ &\vdots\\ \mathbf{x}_k &= \mathbf{x}_{k-1} + \texttt{knn}_{\{I_{tr},\mathbf{x}_{tr}\}}(I_{k-1})\\ &= \mathbf{x}_{k-1} + \texttt{knn}_{\{I_{tr},\mathbf{x}_{tr}\}}(\mathbf{h}(I_0, \mathbf{x}_{k-1})) = \mathbf{f}(\mathbf{x}_{k-1}) = \mathbf{f} \cdots \circ \mathbf{f} \circ \mathbf{f}(\mathbf{x}_0)\end{aligned}$$

CPR and SDM can be seen as the parametric counterparts of DDD. Despite their different choices of regressors, the essence of those algorithms is the same.

2.2.5 Stacking

As one of the ensemble learning schemes, Stacked Generalization (Stacking) [95] was proposed to combine different learning models so the final prediction would outperform each individual learner. First, several base learners are trained using the available data. Then, a secondary learner is trained to make a final prediction using all the predictions of the base learners as additional inputs. In this section, we will focus on some extended work on Stacking for tackling structured prediction problems where the data is highly structured and it is no longer i.i.d. In the

statistical learning community, it is often known as MAP inference whose goal is to find most likely assignment of each individual in the given data.

Here is a brief description on how to use Stacking to solve MAP inference. Given a labeled training set $\{X, Y\}$ with X being the data and Y being its corresponding label set, we first train a classifier h_1 over the entire training set. Using h_1, we can classify X to generate predictions $\hat{Y} = h_1(X)$ from which to derive contextual cues, and then train a new classifier h_2. Specifically, we use $\hat{Y}$ to create a new feature set with contextual cues $X' = \phi(X, \hat{Y})$ and train a new classifier h_2 based on the new data set $\{X', Y\}$. The process can be repeated for multiple rounds until no improvement is observed. As iterations goes on, one expects to see improvement over h_1 if there exists consistent relation among data points. If such assumption fails, it would make no difference than predicting each data point individually using h_1. The relation among data points is captured through a contextual feature function ϕ. For example, one contextual feature function could be simply concatenating the original feature X with previous prediction $\hat{Y}$ of the "neighboring points". Different applications may have their own definitions of neighbors. For instance, in webpages the neighbors are defined by links, the neighbors in academic papers are defined by citations, whether two image pixels are neighbors may depend on how close they are in RGB space, and in documents whether two words are neighbors depend on how far they appear in a sentence.

Below, we will show how the Stacking inference procedures can be rewritten as a sequence of function compositions. We define a new set of functions:

$$\begin{cases} f_1(X) = h_1(X) \\ f_i(Y) = h_i(\phi(X, Y)) \quad \text{if } i > 1. \end{cases}$$

Supposing that the inference runs k iterations we have:

$$\begin{aligned}
&\hat{Y}_1 = h_1(X) = f_1(X) \\
&X' = \phi(X, \hat{Y}_1) \\
&\hat{Y}_2 = h_2(X') = h_2(\phi(X', \hat{Y}_1)) = f_2(\hat{Y}_1) = f_2 \circ f_1(X) \\
&X' = \phi(X, \hat{Y}_2) \\
&\hat{Y}_3 = h_3(X') = h_3(\phi(X, \hat{Y}_2)) = f_3(\hat{Y}_2) = f_3 \circ f_2 \circ f_1(X) \\
&\quad \vdots \\
&\hat{Y}_k = h_k(X') = h_k(\phi(X, \hat{Y}_{k-1})) = f_k(\hat{Y}_{k-1}) = f_k \circ \cdots \circ f_2 \circ f_1(X).
\end{aligned}$$

Above, we showed that the final output can be obtained by a sequence of function compositions by simply recursively replacing $\hat{Y}$ with the results from previous iteration. $\hat{Y}$ tends to be over optimistic if h is trained and test on the same data. To avoid overfitting, $\hat{Y}$ is obtained from temporary classifiers that are trained on subsets of available data set and test on the rest.

In spite of its simplicity, Stacking often outperforms Probabilistic Graphical Model (PGM) based approaches on structural prediction problems, such as, point cloud classification [99], image segmentation [64], email signature recognition [24], web-page classification and name entity extraction [50]. Due to its intractability, when applying PGM one may limit the interactions

among the nodes, *e.g.*, Ising model [13], pairwise MRF [48]. This limits one's ability to leverage complex contextual relation that exists in real applications.

2.2.6 Imitation Learning

Imitation Learning (IL) or Learning from Demonstration (LfD) (also known as Programming by Demonstration (PbD) and Apprenticeship Learning) can be seen as a subset of Supervised Learning. In Supervised Learning, the agent is presented with labeled training data and learns an approximation to the function that produced the data. Within IL, this training dataset is composed of example executions of the task by a demonstration teacher. The IL problem can be formally constructed as follows. The world consists of states S and actions A. In real-world applications, the state is often not fully observable and the learner instead has access to an observed state Z. We are given a demonstration $d_j \in D$ represented as pairs of observations and actions: $d_j = \{(\mathbf{z}_j, \mathbf{a}_j)\}, \mathbf{z}_j \in Z, \mathbf{a}_j \in A$. The goal is to learn a policy $\pi : Z \rightarrow A$ that selects actions based on observations made in the current state. World can be seen as the environment where the prediction is made. An illustration of policy derivation and execution can be found in Fig. 2.6. A comprehensive survey of LfD can be found in [3].

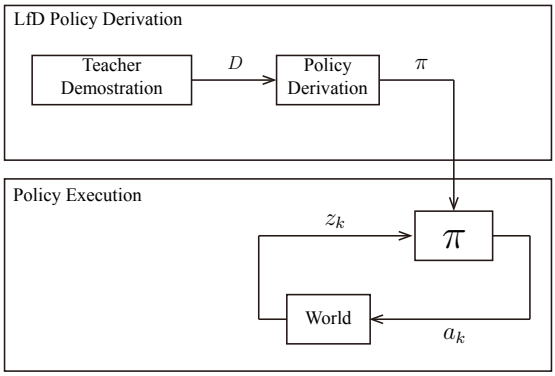


Figure 2.6: Policy derivation and execution. This picture is copied from [3]

Within this framework, we have the state variable $\mathbf{x}$, a state transition function $\mathbf{u} : S \times A \rightarrow S'$, and an observation function $\mathbf{o} : S \rightarrow Z$. We omit World since we assume that it does not change throughout the execution. Let us define a new function $\mathbf{f}$ to be:

$$\mathbf{f}(\mathbf{x}) = \mathbf{u}(\pi(\mathbf{o}(\mathbf{x})), \mathbf{x}). \tag{2.23}$$

Assume that we reach the final state after k iterations of policy execution, which can be rewritten

as a sequence of function compositions:

$$
\begin{aligned}
\mathbf{a}_0 &= \pi(\mathbf{o}(\mathbf{x}_0)) \\
\mathbf{x}_1 &= \mathbf{u}(\mathbf{a}_0, \mathbf{x}_0) = \mathbf{u}(\pi(\mathbf{o}(\mathbf{x}_0)), \mathbf{x}_0) = \mathbf{f}(\mathbf{x}_0) \\
\mathbf{a}_1 &= \pi(\mathbf{o}(\mathbf{x}_1)) \\
\mathbf{x}_2 &= \mathbf{u}(\mathbf{a}_1, \mathbf{x}_1) = \mathbf{u}(\pi(\mathbf{o}(\mathbf{x}_1)), \mathbf{x}_1) = \mathbf{f}(\mathbf{x}_1) = \mathbf{f} \circ \mathbf{f}(\mathbf{x}_0) \\
&\vdots \\
\mathbf{a}_{k-1} &= \pi(\mathbf{o}(\mathbf{x}_{k-1})) \\
\mathbf{x}_k &= \mathbf{u}(\mathbf{a}_{k-1}, \mathbf{x}_{k-1}) = \mathbf{u}(\pi(\mathbf{o}(\mathbf{x}_{k-1})), \mathbf{x}_{k-1}) = \mathbf{f}(\mathbf{x}_{k-1}) = \mathbf{f} \circ \cdots \circ \mathbf{f} \circ \mathbf{f}(\mathbf{x}_0)
\end{aligned}
$$

In the above example, we stick with one policy and one observation function. In practice, different observation functions and different policies may be used at different stages of the execution.

Finally, we will show how SDM can be explained in the IL framework. In the context of minimizing a NLS function, $\mathbf{x}_*$ is regarded as the desired state and the objective is to find the action $\Delta\mathbf{x}$ that moves from the initial state $\mathbf{x}_0$ to the desired state $\mathbf{x}_*$. The nonlinear function $\mathbf{h}$ is the observation function that partially represents the state. The demonstration data contains a set of observation and action pairs. The observed states are represented by a set $Z = \{\mathbf{h}(\mathbf{x}^i) - \mathbf{y}^i\}$ of errors (misalignments) between the known vectors $\{\mathbf{y}^i\}$ and the function evaluations at the current parameter estimates $\{\mathbf{h}(\mathbf{x}^i)\}$. The action set will correspond to the parameter updates $A = \{\Delta\mathbf{x}^i\}$, and the policy maps misalignments to parameter updates. In SDM, the policy is derived as a sequence of linear mapping functions between states and actions. Within this context, the teacher is always available for giving feedback. More specifically, since the ground truth solutions $\{\mathbf{x}_*^i\}$ are available throughout training, the teacher can always give the perfect action based on the state observation. SDM takes advantage of this fact by learning not one but a sequence of policies so the latter ones correct mistakes made from previous iterations after the teacher's feedbacks.

2.2.7 Neural Network

Neural Network is one of the learning models inspired by the brain. Despite its recent tremendous success in Computer Vision [51], the underlying model is very simple. Neural Network approximates a nonlinear function through a sequence of compositions of an activation function, *e.g.*, the outputs of each layer are fed into next layer as the inputs (See Fig. 2.7 for an example). Below, we list the inference steps according to the network depicted in Fig. 2.7:

$$\mathbf{a}_1 = \mathbf{f}(\mathbf{W}_1^\top \mathbf{x}) \tag{2.24}$$

$$\mathbf{a}_2 = \mathbf{f}(\mathbf{W}_2^\top \mathbf{a}_1) \tag{2.25}$$

$$\mathbf{a}_3 = \mathbf{f}(\mathbf{W}_3^\top \mathbf{a}_2) \tag{2.26}$$

$$\hat{\mathbf{y}} = \mathbf{f}(\mathbf{W}_4^\top \mathbf{a}_3) \tag{2.27}$$

This network contains four layers including one input layer, two hidden layers, and one output layer. We use $\mathbf{f}$ to denote the activation function, $\mathbf{W}_k$ to represent the parameters in the k^{th} layer,

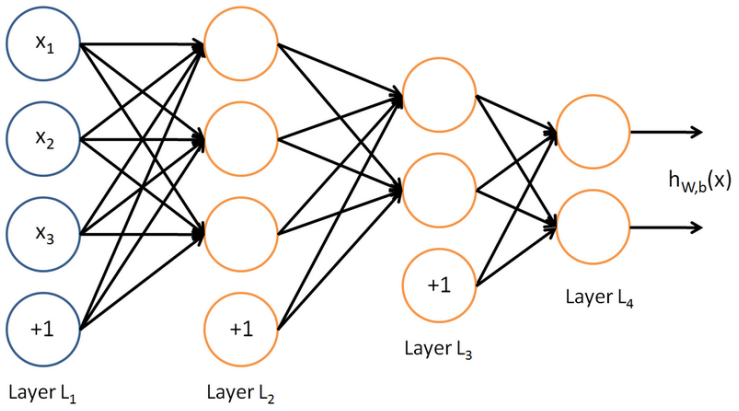


Figure 2.7: A demonstration of a classical Neural Network architecture. This picture is copied from [65].

and $\hat{\mathbf{y}}$ to stand for the estimated outputs. The activation function has to be nonlinear because the composition of linear functions is still linear. Some popular choices of $\mathbf{f}$ include: sigmoid function, hyperbolic tangent, and more recently Rectified Linear Unit (ReLU) [51]. In the case of sigmoid function, the i^{th} element of $\mathbf{f}$ is,

$$f_i(\mathbf{z}) = \frac{1}{1 + \exp(-z_i)}$$

where $\mathbf{z} = \mathbf{W}^\top \mathbf{x}$. Combining Eqs. 2.24 to 2.27 and extending the network to k layers yield the following equation for estimating $\hat{\mathbf{y}}$:

$$\hat{\mathbf{y}} = \mathbf{f}(\cdots(\mathbf{f}(\mathbf{f}(\mathbf{x}, \mathbf{W}_0), \mathbf{W}_1), \mathbf{W}_2), \cdots, \mathbf{W}_{k-1}) \tag{2.28}$$

$$= \mathbf{f}_{\mathbf{W}_{k-1}} \circ \cdots \circ \mathbf{f}_{\mathbf{W}_1} \circ \mathbf{f}_{\mathbf{W}_0}(\mathbf{x}). \tag{2.29}$$

We use $\mathbf{f}_{\mathbf{W}}$ to denote $\mathbf{f}$ parametrized by $\mathbf{W}$. From Eq. 2.29, it is clear that Neural Network can be represented as a sequence of compositions of functions.

Despite their identical inference formulation, there are two important differences between SDM and Neural Network:

1. The parameters in SDM trained greedily layer by layer while Neural Network optimizes all layers as a whole. Treating the parameters in each layer as a whole provides the network more flexibility and raise more difficulty for training. Training a Neural Network greedily would be difficult because optimal activation $\mathbf{a}_*$ in each layer is unknown. In SDM, the optimal parameters are available throughout the training process.
2. The task of the two are totally different. SDM learns a policy that drives the initial state $\mathbf{x}_0$ toward $\mathbf{x}_*$. Neural Network tries to approximate a nonlinear function that reproduces input label $\mathbf{y}$.

In Computer Vision, for large images Neural Network is often used with additional convolution layers and max pooling layers to reduce the dimensionality before feeding into the fully connected layer. A slight modification to Eq. 2.29 can be made to incorporate this change:

$$\hat{\mathbf{y}} = \mathbf{f}^{k-1}_{\mathbf{W}_{k-1}} \circ \cdots \circ \mathbf{f}^{1}_{\mathbf{W}_1} \circ \mathbf{f}^{0}_{\mathbf{W}_0}(\mathbf{x}). \tag{2.30}$$

Now, $\mathbf{f}^i$ indicates a different function at each layer.

2.2.8 Summary

In this section, we have reviewed seven representative work in sequential prediction, discussed their similarities and differences, showed that all their inference procedures can be reformulated as a sequence of function compositions. According to their tasks, they can be grouped into two categories:

1. Given an initial configuration $\mathbf{x}_0$, the goal is to learn a policy that moves it to the optimal state $\mathbf{x}_*$.
2. Given a data set $\{X, Y\}$, the goal is to approximate a function that takes X as the input and tries to reproduce Y.

The above two categories cover the majority of areas in Supervised Learning. SDM, CPR, DDD, and IL belong to the first category and GB, Stacking, and Neural Network fall into the second one. A comparison of all the seven methods can be found in Fig. 2.8.

Algorithm	Task	Training Strategy	Base Function
SDM	$\mathbf{x}_0 \rightarrow \mathbf{x}_*$	Greedy	Parametric
GB	$X \rightarrow Y$	Greedy	Parametric
CPR	$\mathbf{x}_0 \rightarrow \mathbf{x}_*$	Greedy	Parametric
DDD	$\mathbf{x}_0 \rightarrow \mathbf{x}_*$	N/A	Non-parametric
Stacking	$X \rightarrow Y$	Greedy	Parametric
IL	$\mathbf{x}_0 \rightarrow \mathbf{x}_*$	Various	Both
Neural Network	$X \rightarrow Y$	Optimal	Parametric

Figure 2.8: A comparison of all seven sequential prediction methods.

Chapter 3

Applications

"Well, it may be all right in practice, but it will never work in theory."

Warren Buffett

"Beware of bugs in the above code; I have only proved it correct, not tried it."

Donald Knuth

3.1 Rigid Tracking

This section illustrates how to apply SDM to the problem of tracking rigid objects. In particular, we show how we can extend the classical Lucas-Kanade (LK) method [58] to efficiently operate in HoG [31] space. To the best of our knowledge, this is the first algorithm to perform alignment in HoG space.

3.1.1 Previous Work (Lucas-Kanade)

The Lucas-Kanade (LK) tracker is one of the earliest and most popular computer vision trackers due to its efficiency and simplicity. It formulates image alignment as a NLS problem. Alignment is achieved by finding the motion parameter $\mathbf{p}$ that minimizes

$$\min_{\mathbf{p}} \ ||\mathbf{d}(\mathbf{f}(\mathbf{x},\mathbf{p})) - \mathbf{t}(\mathbf{x})||_2^2, \tag{3.1}$$

where $\mathbf{t}(\mathbf{x})$ is the template, $\mathbf{x} = [x_1, y_1, ...x_l, y_l]^\top$ is a vector containing the coordinates of the pixels to detect/track, and $\mathbf{f}(\mathbf{x},\mathbf{p})$ is a vector with entries $[u_1, v_1, ..., u_l, v_l]^\top$ representing a geometric transformation. In this section, we limit the transformation to be affine. That is, (u_i, v_i) relates to (x_i, y_i) by

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} p_1 & p_2 \\ p_4 & p_5 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} p_3 \\ p_6 \end{bmatrix}.$$

The i^{th} entry of $\mathbf{d}(\mathbf{f}(\mathbf{x},\mathbf{p}))$ is the pixel intensity of the image $\mathbf{d}$ at (u_i, v_i). Minimizing Eq. 3.1 is a NLS problem because the motion parameters are nonlinearly related to the pixel values.

Given a template (often the initial frame), the LK method uses Gauss-Newton to minimize Eq. 3.1 by linearizing the motion parameters around an initial estimate $\mathbf{p}_0$:

$$\min_{\Delta\mathbf{p}} \ ||\mathbf{d}(\mathbf{f}(\mathbf{x},\mathbf{p}_k)) + \frac{\partial \mathbf{d}}{\partial \mathbf{f}} \frac{\partial \mathbf{f}}{\partial \mathbf{p}_k} \Delta\mathbf{p} - \mathbf{t}(\mathbf{x})||_2^2, \tag{3.2}$$

where $\frac{\partial \mathbf{d}}{\partial \mathbf{p}_k}$ is the Jacobian of the image over motion parameter. The Jacobian is decomposed into $\frac{\partial \mathbf{d}}{\partial \mathbf{f}} \frac{\partial \mathbf{f}}{\partial \mathbf{p}_k}$ using the chain rule. $\frac{\partial \mathbf{d}}{\partial \mathbf{f}}$ is the image gradient evaluated under the affine transformation $\mathbf{f}(\mathbf{x},\mathbf{p}_k)$ and $\frac{\partial \mathbf{f}}{\partial \mathbf{p}_k}$ is the Jacobian of the geometric transformation evaluated at the current $\mathbf{p}_k$. Differentiating Eq. 3.2 over $\Delta\mathbf{p}$ and setting it to zero gives us the LK update,

$$\Delta\mathbf{p} = \mathbf{H}_k^{-1} \left(\frac{\partial \mathbf{d}}{\partial \mathbf{f}} \frac{\partial \mathbf{f}}{\partial \mathbf{p}_k} \right)^\top \big(\mathbf{t}(\mathbf{x}) - \mathbf{d}(\mathbf{f}(\mathbf{x},\mathbf{p}_k)) \big),$$

where $\mathbf{H}_k = (\frac{\partial \mathbf{d}}{\partial \mathbf{f}} \frac{\partial \mathbf{f}}{\partial \mathbf{p}_k})^\top (\frac{\partial \mathbf{d}}{\partial \mathbf{f}} \frac{\partial \mathbf{f}}{\partial \mathbf{p}_k})$ is the Gauss-Newton approximation of the Hessian. The motion parameter is then updated as $\mathbf{p}_{k+1} = \mathbf{p}_k + \Delta\mathbf{p}$.

3.1.2 An SDM Solution

The LK method employs Gauss-Newton for minimization. For gradient-based methods to converge, the objective function has to be convex. However, this is not the case in affine tracking

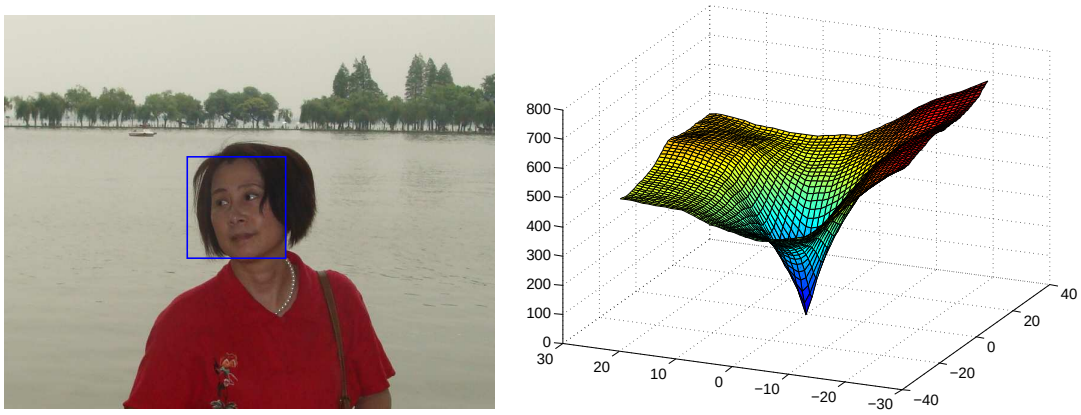


Figure 3.1: a) an image of the world's best mom; b) objective function.

(even when only translation is involved). As shown in Fig. 3.1, the objective function is not convex. The template is the bounding box region and the objective mesh is created by computing the average pixel difference between the template and the ones generated from shifting the bounding box around it. Recall the SDM's convergence properties, which do not require the objective function to be convex. Another problem with LK is that it is not robust to illumination changes. Robustness can be achieved by aligning images w.r.t. more robust image descriptors instead of pixel intensities, *i.e.*,

$$\min_{\mathbf{p}} \; ||\mathbf{h}(\mathbf{d}(\mathbf{f}(\mathbf{x},\mathbf{p}))) - \mathbf{h}(\mathbf{t}(\mathbf{x}))||_2^2, \tag{3.3}$$

where $\mathbf{h}$ is some image descriptor function (HoG, in our case). The LK (Gauss-Newton) update for minimizing 3.3 can be derived as follows:

$$\Delta\mathbf{p} = \mathbf{H}_k^{-1}\left(\frac{\partial\mathbf{h}}{\partial\mathbf{p}_k}\right)^\top \big(\mathbf{h}(\mathbf{t}(\mathbf{x})) - \mathbf{h}(\mathbf{d}(\mathbf{f}(\mathbf{x},\mathbf{p}_k)))\big).$$

However, the update can no longer be computed efficiently: HoG is a non-differentiable image operator, and thus the Jacobian ($\frac{\partial\mathbf{h}}{\partial\mathbf{p}_k}$) has to be estimated numerically at each iteration.

In contrast, SDM minimizes Eq. 3.3 by replacing the computationally expensive term $\mathbf{H}_k^{-1}(\frac{\partial\mathbf{h}}{\partial\mathbf{p}_k})^\top$ with a pre-trained DM $\mathbf{R}$, and gives the following update:

$$\Delta\mathbf{p} = \mathbf{R}_k(\mathbf{h}(\mathbf{t}(\mathbf{x})) - \mathbf{h}(\mathbf{d}(\mathbf{f}(\mathbf{x},\mathbf{p}_k)))). \tag{3.4}$$

Each update step in SDM is very efficient, mainly consisting of one affine image warping and one HoG descriptor computation on the warped image. One may notice the inconsistency between Eq. 3.4 and the SDM update we previously introduced in Eq. 2.7. In the following, we will show the equivalence of the two.

The template can be seen as the HoG descriptors extracted from the image under an identity transformation, $\mathbf{t}(\mathbf{x}) = \mathbf{t}(\mathbf{f}(\mathbf{x},\mathbf{p}_*))$, where

$$\mathbf{p}_* = \begin{bmatrix}1 & 0 & 0 & 1 & 0 & 0\end{bmatrix}^\top.$$

Under the assumption that only affine deformation is involved, the image $\mathbf{d}$ on which we perform tracking can be interpreted as the template image under an unknown affine transformation $\tilde{\mathbf{p}}$:

$$\mathbf{d}(\mathbf{x}) = \mathbf{t}(\mathbf{f}(\mathbf{x}, \tilde{\mathbf{p}})).$$

In Eq. 3.4, image $\mathbf{d}$ warped under the current parameter $\mathbf{p}_k$ can be re-written as

$$\mathbf{d}(\mathbf{f}(\mathbf{x}, \mathbf{p}_k)) = \mathbf{t}(\mathbf{f}(\mathbf{f}(\mathbf{x}, \tilde{\mathbf{p}}), \mathbf{p}_k)). \tag{3.5}$$

The composition of two affine transformations remains affine, so Eq. 3.5 becomes

$$\mathbf{d}(\mathbf{f}(\mathbf{x}, \mathbf{p}_k)) = \mathbf{t}(\mathbf{f}(\mathbf{x}, \widehat{\mathbf{p}})),$$

where $\widehat{\mathbf{p}}$ an unknown affine parameter that differs from $\tilde{\mathbf{p}}$. Therefore, we can re-write Eq. 3.4 as

$$\Delta\mathbf{p} = \mathbf{R}(\mathbf{h}(\mathbf{t}(\mathbf{f}(\mathbf{x}, \mathbf{p}_*))) - \mathbf{h}(\mathbf{t}(\mathbf{f}(\mathbf{x}, \widehat{\mathbf{p}})))). \tag{3.6}$$

Eq. 3.6 can be further simplified to follow the same form of Eq. 2.7:

$$\Delta\mathbf{p} = \mathbf{R}(\mathbf{g}(\mathbf{p}_*) - \mathbf{g}(\widehat{\mathbf{p}})),$$

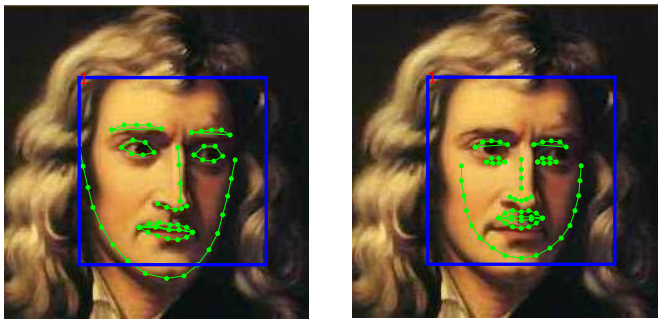
where $\mathbf{g} = \mathbf{h} \circ \mathbf{t} \circ \mathbf{f}$.

In our implementation, we use Eq. 3.4 as the update rule instead of Eq. 3.6 because $\mathbf{p}_k$ is a known parameter w.r.t image $\mathbf{d}$ and $\widehat{\mathbf{p}}$ is unknown w.r.t the template image $\mathbf{t}$. The descent maps are learned in the neighborhood of $\mathbf{p}_*$, but as tracking continues, the motion parameter may deviate greatly from $\mathbf{p}_*$. When tracking a new frame, before applying SDM the image is first warped back using the motion parameter estimated in the previous frame so that the optimization happens within a neighborhood of $\mathbf{p}_*$. Therefore, after SDM finishes, we warp back the estimated $\Delta\mathbf{p}$ using the same parameter.

The training of SDM involves sampling initial motion parameters and solving a sequence of linear systems (detailed in section 2.1.4). We sample $\{\mathbf{p}_0^i\}_i$ around $\mathbf{p}_*$ and those samples are used for approximating the expectation expressed in Eq. 2.8. The details of how we generate initial samples are described in section 4.3.

3.2 Face Alignment

In the previous section, we showed how SDM can be used for aligning regions of images that undergo an affine motion. This section extends SDM to detect and track nonrigid objects. In particular, we will show how SDM achieves state-of-the-art performance in facial feature detection and tracking.



(a) $\mathbf{x}_*$ (b) $\mathbf{x}_0$

Figure 3.2: a) Manually labeled image with 66 landmarks. Blue outline indicates face detector. b) Mean landmarks, $\mathbf{x}_0$, initialized using the face detector.

3.2.1 Previous Work

This section reviews existing work on face alignment.

Parameterized Appearance Models (PAMs), such as Active Appearance Models [6, 26, 32], Morphable Models [14, 47], Eigentracking [12], and template tracking [59, 92] build an object appearance and shape representation by performing Principal Component Analysis (PCA) on a set of manually labeled data. Fig. 3.2a illustrates an image labeled with p landmarks ($p = 66$ in this case). After the images are aligned with Procrustes [35], a shape model is learned by performing PCA on the registered shapes. A linear combination of k_s shape bases $\mathbf{U}^s \in \mathbb{R}^{2p \times k_s}$ can reconstruct (approximately) any aligned shape in the training set. Similarly, an appearance model $\mathbf{U}^a \in \mathbb{R}^{m \times k_a}$ is built by performing PCA on the texture. Alignment is achieved by finding the motion parameter $\mathbf{p}$ and appearance coefficients $\mathbf{c}^a$ that best align the image w.r.t. the subspace $\mathbf{U}^a$,

$$\min_{\mathbf{c}^a, \mathbf{p}} ||\mathbf{d}(\mathbf{f}(\mathbf{x}, \mathbf{p})) - \mathbf{U}^a \mathbf{c}^a||_2^2. \tag{3.7}$$

In the case of the LK tracker, $\mathbf{c}^a$ is fixed to be $\mathbf{1}_{k_a}$ and $\mathbf{U}^a$ is a subspace that contains a single vector, the reference template. The notation follows that of Section 3.1.1; $\mathbf{x} = [x_1, y_1, \ldots, x_l, y_l]^\top$ contains the coordinates of the pixels to track, and $\mathbf{f}(\mathbf{x}, \mathbf{p})$ is a vector denoted by $[u_1, v_1, ..., u_l, v_l]^\top$ that now includes both affine and nonrigid transformation. That is, (u_i, v_i) relates to (x_i, y_i) by

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ a_4 & a_5 \end{bmatrix} \begin{bmatrix} x_i^s \\ y_i^s \end{bmatrix} + \begin{bmatrix} a_3 \\ a_6 \end{bmatrix}.$$

Here

$$[x_1^s, y_1^s, ...x_l^s, y_l^s]^\top = \overline{\mathbf{x}} + \mathbf{U}^s \mathbf{c}^s,$$

where $\overline{\mathbf{x}}$ is the mean shape face, $\mathbf{a}, \mathbf{c}^s$ are the affine and nonrigid motion parameters respectively, and $\mathbf{p} = [\mathbf{a}; \mathbf{c}^s]$. Similar to the LK method, PAMs algorithms [6, 12, 26, 32] optimize Eq. 3.7 using the Gauss-Newton method. A more robust formulation of (3.7) can be achieved by either replacing the L_2 norm with a robust error function [8, 12] or by performing matching on robust features, such as gradient orientation [93].

Discriminative approaches learn a mapping from image features to motion parameters or landmarks. Cootes *et al.* [26] proposed to fit AAMs by learning a linear regression between the increment of motion parameters $\Delta\mathbf{p}$ and the appearance differences $\Delta\mathbf{d}$. The linear regressor is a numerical approximation of the Jacobian [26]. Following this idea, several discriminative methods that learn a mapping from $\mathbf{d}$ to $\Delta\mathbf{p}$ have been proposed. Gradient Boosting, first introduced by Friedman [38], has become one of the most popular regressors in face alignment because of its efficiency and ability to model nonlinearities. Saragih and Göcke [80] and Tresadern *et al.* [91] showed that using boosted regression for AAM discriminative fitting significantly improved over the original linear formulation. Dollár *et al.* [34] incorporated "pose indexed features" to the boosting framework, where features are re-computed at the latest estimate of the landmark locations in addition to learning a new weak regressor at each iteration. Beyond gradient boosting, Rivera and Martinez [71] explored kernel regression to map from image features directly to landmark locations, achieving surprising results for low-resolution images. Recently, Cootes *et al.* [25] investigated Random Forest regressors in the context of face alignment. At the same time, Sánchez *et al.* [77] proposed to learn a regression model in the continuous domain to efficiently and uniformly sample the motion space. In the context of tracking, Zimmermann *et al.* [110] learned a set of independent linear predictors for different local motions and then chose a subset of them during tracking. Unlike PAMs, a major problem of discriminative approaches is that the cost function being minimizing is unclear, making these algorithms difficult to analyze theoretically. This paper is the first to formulate a precise cost function for discriminative approaches.

Part-based deformable models perform alignment by maximizing the posterior likelihood of part locations given an image. The objective function is composed of the local likelihood of each part times a global shape prior. Different methods typically vary the optimization methods or the shape prior. Constrained Local Models (CLM) [30] model this prior similarly as AAMs, assuming all faces lie in a linear subspace spanned by PCA bases. Saragih *et al.* [79] proposed a nonparametric representation to model the posterior likelihood and the resulting optimization method is reminiscent of mean-shift. In [10], the shape prior was modeled nonparametrically from training data. Recently, Saragih [78] derived a sample specific prior to constrain the output space providing significant improvements over the original PCA prior. Instead of using a global model, Huang et al. [46] proposed to build separate Gaussian models for each part (*e.g.*, mouth, eyes) to preserve more detailed local shape deformations. Zhu and Ramanan [108] assumed that the face shape is a tree structure (for fast inference), and used a part-based model for face detection, pose estimation, and facial feature detection.

3.2.2 An SDM Solution

Similar to rigid tracking in section 3.1.2, we perform face alignment in the HoG space. Given an image $\mathbf{d} \in \mathbb{R}^{m\times 1}$ of m pixels, $\mathbf{d}(\mathbf{x}) \in \mathbb{R}^{p\times 1}$ indexes p landmarks in the image. $\mathbf{h}$ is a nonlinear

feature extraction function and $\mathbf{h}(\mathbf{d}(\mathbf{x})) \in \mathbb{R}^{128p\times 1}$ in the case of extracting HoG features. In this setting, face alignment can be framed as minimizing the following NLS function over landmark coordinates $\mathbf{x}$:

$$f(\mathbf{x}) = \|\mathbf{h}(\mathbf{d}(\mathbf{x})) - \mathbf{y}_*\|_2^2, \tag{3.8}$$

where $\mathbf{y}_* = \mathbf{h}(\mathbf{d}(\mathbf{x}_*))$ represents the HoG values computed on the local patches extracted from the manually labeled landmarks. During training, we assume that the correct p landmarks (in our case $p = 66$) are known, and we will refer to them as $\mathbf{x}_*$ (see Fig. 3.2a). Also, to simulate the testing scenario, we run the face detector on the training images to provide an initial configuration of the landmarks ($\mathbf{x}_0$), which corresponds to an average shape (see Fig. 3.2b).

Eq. 3.8 has several fundamental differences with previous work on PAMs (Eq. 3.7). First, in Eq. 3.8, we do not learn any model of shape or appearance beforehand from training data. Instead, we align the image w.r.t. a template $\mathbf{y}_*$. For the shape, we optimize the landmark locations $\mathbf{x} \in \mathbb{R}^{2p\times 1}$ directly. Recall that in traditional PAMs, nonrigid motion is modeled as a linear combination of shape bases learned by performing PCA on a training set. Our shape formulation is able to generalize better to untrained situations (*e.g.*, asymmetric facial gestures). Second, we use HoG features extracted from patches around the landmarks to achieve a representation robust to illumination changes.

In face alignment, the testing template $\mathbf{y}_*$ is unknown and different from those used for training (*i.e.*, the test subject is not one of the training subjects). Therefore, SDM learns an additional bias term $\mathbf{b}_k$ to represent an average template during training. Furthermore, the function $\mathbf{h}$ is parametrized not only by $\mathbf{x}$, but also by the images (*i.e.*, different subjects or different conditions of subjects). The training step modifies Eq. 2.7 to minimize the expected loss over all initializations and images, where the expected loss is given by

$$\sum_{i,j} \|\mathbf{x}_* - \mathbf{x}_k^{i,j} + \mathbf{R}_k \mathbf{h}(\mathbf{d}^i(\mathbf{x}_k^{i,j})) - \mathbf{b}_k\|_2^2. \tag{3.9}$$

We use i to index images and j to index initializations. The update of Eq. 2.7 is thus modified to be

$$\mathbf{x}_k^{i,j} = \mathbf{x}_{k-1}^{i,j} - \mathbf{R}_{k-1}\mathbf{h}(\mathbf{d}^i(\mathbf{x}_{k-1}^{i,j})) + \mathbf{b}_{k-1}. \tag{3.10}$$

Despite the modification, minimizing Eq. 3.10 is still a linear least squares problem. Note that we do not use $\mathbf{y}_*^i$ in training, although they are available. In testing, given an unseen image $\tilde{\mathbf{d}}$ and an initial guess of $\tilde{\mathbf{x}}_0$, $\tilde{\mathbf{x}}_k$ is updated recursively using Eq. 3.10. If $\tilde{\mathbf{d}}, \tilde{\mathbf{x}}_0$ are drawn from the same distribution that produces the training data, each iteration is guaranteed to decrease the expected loss between $\tilde{\mathbf{x}}_k$ and $\mathbf{x}_*$.

3.2.3 Online SDM

SDM may have poor performance on an unseen sample that is dramatically different from those in the training set. It would be desirable to incorporate this new sample into the existing model without re-training. This section describes such a procedure that updates an existing SDM model in an online fashion.

Assume that one is given a trained SDM model, represented by $\{\mathbf{R}_k, \mathbf{b}_k, \mathbf{\Sigma}_k^{-1}\}$, where $\mathbf{\Sigma}_k = \mathbf{\Phi}_k\mathbf{\Phi}_k^\top$ and $\mathbf{\Phi}_k$ is the data matrix used in training the k^{th} descent map. For a given new face image

$\mathbf{d}$ and labeled landmarks $\mathbf{x}_*$, one can compute the initial landmark perturbation $\Delta\mathbf{x}_0 = \mathbf{x}_* - \mathbf{x}_0$ and the feature vector extracted at $\mathbf{x}_0$, $\boldsymbol{\phi}_0 = \mathbf{h}(\mathbf{d}(\mathbf{x}_0))$. Using the well known recursive least squares algorithm [42], SDM can be re-trained by iterating the following three steps:

1. Update inverse covariance matrix $\boldsymbol{\Sigma}_k^{-1}$:

$$\boldsymbol{\Sigma}_k^{-1} \leftarrow \boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\phi}_k(w^{-1} + \boldsymbol{\phi}_k^\top\boldsymbol{\Sigma}_k^{-1}\boldsymbol{\phi}_k)^{-1}\boldsymbol{\phi}_k^\top\boldsymbol{\Sigma}_k^{-1}. \tag{3.11}$$

2. Update the generic descent direction $\mathbf{R}_k$:

$$\mathbf{R}_k \leftarrow \mathbf{R}_k + (\Delta\mathbf{x}_k - \mathbf{R}_k\boldsymbol{\phi}_k)w\boldsymbol{\phi}_k^\top\boldsymbol{\Sigma}_k^{-1}.$$

3. Generate a new sample pair $(\Delta\mathbf{x}_{k+1}, \boldsymbol{\phi}_{k+1})$ for re-training in the next iteration:

$$\begin{aligned}\Delta\mathbf{x}_{k+1} &\leftarrow \Delta\mathbf{x}_k - \mathbf{R}_k\boldsymbol{\phi}_k \\ \boldsymbol{\phi}_{k+1} &\leftarrow \mathbf{h}(\mathbf{d}(\mathbf{x}_* + \Delta\mathbf{x}_{k+1})).\end{aligned}$$

Setting the weight to be $w = 1$ treats every sample equally. For different applications, one may want the model to emphasize the more recent samples. For example, SDM with exponential forgetting can be implemented with a small modification of Eq. 3.11:

$$\boldsymbol{\Sigma}_k^{-1} \leftarrow \lambda^{-1}[\boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\phi}_k(\lambda + \boldsymbol{\phi}_k^\top\boldsymbol{\Sigma}_k^{-1}\boldsymbol{\phi}_k)^{-1}\boldsymbol{\phi}_k^\top\boldsymbol{\Sigma}_k^{-1}],$$

where $0 < \lambda < 1$ is a discount parameter. Assuming n data points come in order, the weight on the i^{th} sample is λ^{n-i}. Above, we do not explain the update formula for the bias term $\mathbf{b}_k$, since it is often incorporated into $\mathbf{R}_k$ by augmenting the feature vector with 1. Note that in Eq. 3.11, the term in parentheses is a scalar. Since no matrices need to be inverted, our re-training scheme is very efficient, consisting of only a few matrix multiplications and feature extractions.

3.2.4 Extensions of SDM on Face Alignment

Since its original publications [97, 98], many extensions of SDM have been proposed. They mainly involve experimenting with different patch descriptors, better shape initialization strategies, and various regression methods. Some of the notable works are listed as follows:

- Ren *et al.* [70] replaced SIFT descriptors with binary features boosting the computational speed to 3000 FPS on a modern CPU without visible loss of alignment accuracy.
- Zhang *et al.* [103] replaced the SIFT descriptors and linear regressor by a Stacked Auto-encoder Network(SAN) [43], which acts as a nonlinear regressor that directly maps pixel values to landmark displacements. The initial face shape was predicted by a separate SAN from a low-resolution face image. They showed improvement over SDM especially in the case where the face detector's results are unreliable.
- Qu *et al.* [68] introduced an additional step to remove similarity transformation of the face image so alignment can be operated on a normalized image. Also, each iteration descriptors are extracted at patches with decreasing sizes. They compared six different image descriptors (HoG, SIFT, LBP and their variants measured in Helliger distance) and concluded that RootSIFT [2] was the best among the six. Additionally, they adopted Iteratively Reweighted Least Squares (IRLS) as the regressor to remove the effects of outliers.

- Yang *et al.* [101] proposed a novel initialization strategy for SDM based on the head pose estimated through a Convolutional Neural Network (CNN) [53]. Instead of using a mean face shape, SDM is initialized by either projecting a 3D mean face shape using the estimated head pose or searching the nearest neighbor from a training set according to estimated head pose distance.
- Yang *et al.* [100] proposed random subspace SDM, which is reminiscent of an ensemble learning method, tree bagging [17]. Instead of learning one linear regressor, they learned a set of linear regressors that are trained on random subsets of the training data. This reduces the variance of estimator. In testing, the average prediction of all the regressors is used.
- Zhang *et al.* [104] proposed a simple strategy to better handle images in different resolutions. During training, they built nine SDM models, one for a different resolution of face images. In testing, a super resolution technique is used if necessary and the best of the nine model is selected based on the resolution of the detected face.
- Zhu *et al.* [107] showed that SDM can be extended to automatically transfer landmark annotations across datasets. Their method allows integration of different face alignment datasets with different annotation protocols so they can be combined to train a single model. With their method, one can also enrich landmark annotations from a sparsely labelled images.
- Zhu *et al.* [109] applied SDM to fit a 3D morphable model based on the appearance of a face. Here, the 2D face shape is controlled by a 3D PCA face model and a weak perspective projection model. In each iteration, SDM learns a mapping from image features to the model parameters (weak perspective camera parameters and 3D shape deformation coefficients).

3.3 Multi-view Face Alignment

SDM provides an efficient and accurate solution to track facial features in near-frontal faces, but it fails at tracking faces with large head rotations. This section presents a solution to address those limitations based on GSDM. First, let us review some previous work on multi-view face alignment.

3.3.1 Previous Work

Previous work on multi-view facial feature tracking can be grouped into two categories based on whether a 2D or 3D face model is used.

Let us first review some of the 2D model based approaches. The shape of a deformable object can be modeled by a probability density function. A multi-modal 2D face model can be represented either in a non-parametric way *e.g.*, kernel density estimation [83] or in a parametric way, *e.g.*, a mixture of Gaussians [27]. Therefore, there are two common strategies to extend traditional frontal face alignment methods to multi-view tracking. The first one is to build separate models according to the head pose. Some of the examples are multi-view Active Appearance Model (AAM) [29], view-based Active Wavelet Networks [44], and multi-view Direct

Appearance Models [54]. The other is to use kernel methods. For example, Romdhani *et al.* [74] extended Active Shape Model [28] to track profile-to-profile faces. They used kernel PCA [81] to model the shape and appearance to address the nonlinearity introduced by large pose changes. However, kernel-based density estimation is slow and its complexity increases with number of training samples. Another interesting work [106] treated the shape parameter and pose as hidden variables and framed the alignment problem into a Bayesian framework. However, the inference is intractable so the EM algorithm (local minima prone) is used to approximate the solution. Beyond the two common strategies, another way to address the multi-view problem would be online tracking. Ellis *et al.* [36] proposed an efficient online tracker using adaptive appearance models, and one could extend this approach to track faces and other nonrigid objects.

Next, let us take a look at 3D model based approaches. Matthews *et al.* [60] provided a detailed comparison between 2D and 3D face models in three different aspects, fitting speed, representational power, and construction. They concluded that 2D face model may be too "powerful" that can represent invalid faces. Xiao *et al.* [96] extended the AAM fitting algorithm to impose additional shape constraints introduced by a 3D model that are lacked in the 2D model. Baltrusaitis *et al.* [9] extended Constrained Local Models [30] for RGBD data streams and show better alignment performance than its original. However, the training data is difficult to collect. Gu and Kanade [41] formulated multi-view face alignment as a Bayesian inference problem with missing data, whose task is to solve 3D shape and 3D pose from the noisy and incomplete 2D shape observation. Recently, Cao *et al.* [22] extended an earlier 2D regression-based framework [23] with a 3D face model, but only near-frontal face results are shown in the experiments. Other interesting work [82, 84, 108] have been proposed for detecting facial landmarks in the profile-to-profile faces but they are not suitable for tracking applications. Note that most 3D based methods still reply on head pose to build separate models to address the multi-view problem.

Our work differs from existing approaches in several ways. First, our approach do not prebuild any shape or appearance model and we directly optimize over landmark coordinates. This has been shown to provide superior performance for facial feature tracking [97]. Second, our method provides a mathematically sound manner to partition the parameter space for facial feature tracking. Existing approaches typically find heuristic partition of the head pose angles. Finally, our method is general and can be applied to other problems, such as extrinsic camera calibration (see Section 4.6).

3.3.2 A Global SDM Solution

SDM provides an efficient and accurate solution to track facial features in near-frontal faces, but it fails at tracking faces with large head rotations. When tracking profile-to-profile faces the shape parameter space is enlarged so it is unlikely to find a single valid DM (See section 2.1.3 and recall the two conditions for DM to exist). In section 2.1.6, we introduced the idea of DHD, which refers to a partition on the parameter space such that there exists a DM within each subset. The problem of multi-view face alignment is reduced to finding DHD. Given a finite set of samples, finding the optimal DHD $S = \{S^t\}_1^T$ and its corresponding DMs $R = \{\mathbf{R}^t\}_1^T$ can be

formulated as the following constrained optimization problem,

$$\min_{S,R} \sum_{t=1}^{T} \sum_{i \in S^t} \|\Delta \mathbf{x}_*^i - \mathbf{R}^t \Delta \boldsymbol{\phi}^{i,t}\|^2 \tag{3.12}$$

$$\text{s. t. } \Delta {\mathbf{x}_*^i}^\top \mathbf{R}^t \Delta \boldsymbol{\phi}^{i,t} > 0, \forall t, i \in S^t. \tag{3.13}$$

One can use a predefined T or choose the best T using a validation set. We denote $\overline{\boldsymbol{\phi}}_*^t - \boldsymbol{\phi}^i$ by $\Delta \boldsymbol{\phi}^{i,t}$, where $\overline{\boldsymbol{\phi}}_*^t$ is the template averaged over all image in the t^{th} subset. The constraints stated in (3.13) guarantee that $\mathbf{R}^t \mathbf{h}(\mathbf{x})$ is a monotone operator around $\mathbf{x}_*$, which is one condition ensuring that $\mathbf{R}^t$ is a generic DM within the t^{th} subset.

Minimizing (3.12) is NP-hard. We develop a deterministic approach to approximate the solution of (3.12). First, let us ignore the constraints and solve the unconstrained optimization problem in (3.12). If $\mathbf{R}^t$ is a local minimizer, one necessary condition is that the partial derivative of (3.12) against $\mathbf{R}^t$ is zero. Setting the this derivative to zero gives:

$$\mathbf{R}^t = \Delta \mathbf{X}_*^t \Delta \Phi^{t\top} (\Delta \Phi^t \Delta \Phi^{t\top})^{-1}. \tag{3.14}$$

$\Delta \mathbf{X}_*^t$ and Φ^t are matrices whose columns are $\Delta \mathbf{x}_*^i$ and $\boldsymbol{\phi}^i$ from the t^{th} subset. Plugging Eq. 3.14 into the constraints in (3.13) yields,

$${\Delta \mathbf{x}_*^i}^\top \Delta \mathbf{X}_*^t \Delta \Phi^{t\top} (\Delta \Phi^t \Delta \Phi^{t\top})^{-1} \Delta \boldsymbol{\phi}^{i,t} > 0, \forall t, i \in S^t. \tag{3.15}$$

The sufficient conditions for (3.15) are:

$${\Delta \mathbf{x}_*^i}^\top \Delta \mathbf{X}_*^t > \mathbf{0}, \forall t, i \in S^t \tag{3.16}$$

$$\Delta \Phi^{t\top} (\Delta \Phi^t \Delta \Phi^{t\top})^{-1} \Delta \boldsymbol{\phi}^{i,t} > \mathbf{0}, \forall t, i \in S^t \tag{3.17}$$

From the fact that any two vectors within the same hyperoctant (the generalization of quadrant) have a positive dot product, we design a partition such that each subset occupies a hyperoctant in the parameter space. This partition satisfies the inequalities in (3.16). We can apply the same strategy to further partition each subset according to the hyperoctants in feature space, which yields the following inequalities

$$\Delta \Phi^{t\top} \Delta \boldsymbol{\phi}^{i,t} > \mathbf{0}, \forall t, i \in S^t \tag{3.18}$$

The covariance matrix $\Phi \Phi^\top$ is positive-definite (if not, a diagonal matrix can be added). The inverse of a positive definite matrix is also positive definite. This fact along with (3.18) suffice to show the inequalities in (3.17). However, this partition is impractical leading to exponential number of DMs so we propose the following approximation.

In the case of human faces, $\Delta \mathbf{x}$ and $\Delta \boldsymbol{\phi}$ are embedded in a lower dimensional manifold. We perform dimension reduction (PCA) on the whole training set $\Delta \mathbf{X}$ and project the data onto the subspace expanded by the first two most dominant directions. This gives us a partition in $\mathbb{R}^2$ where each subset occupies a quadrant. Each subset inside this partition is further partitioned into two halves based on the first principle component learned from $\Delta \Phi$. This partition strategy

gives us eight subsets so eight DMs are learned in each iteration of the algorithm. The PCA bases are saved and used to determine which DM to use in testing time. The training of GSDM converges in four iterations. In testing $\mathbf{x}_*$ is unknown and assuming that the movement between two consecutive frames is small the prediction of the previous frame is used to approximate $\Delta\mathbf{x}_*$. We only used two PCA bases, although one can increase the number of bases to create more subsets in the partition. The approximation would be more accurate at the same time more training data will be needed to learn a reliable DM. One can also use nonlinear dimension reduction techniques [67]. This simple partition strategy has been validated in our experiments and yields promising results.

3.4 3D Pose Estimation

In the two applications we have shown thus far, the optimization parameters lie in $\mathbb{R}^n$ space. In this section, we will show how SDM can also be used to optimize parameters such as a rotation matrix, which belongs to the $SO(3)$ group.

The problem of 3D pose estimation can be described as follows. Given the 3D model of an object represented as $3D$ points $\mathbf{M} \in \mathbb{R}^{3\times n}$, its projection $\mathbf{U} \in \mathbb{R}^{2\times n}$ on an image, and the intrinsic camera parameters $\mathbf{K} \in \mathbb{R}^{3\times 3}$, the goal is to estimate the object pose (3D rotation $\mathbf{Q} \in \mathbb{R}^{3\times 3}$ and translation $\mathbf{t} \in \mathbb{R}^{3\times 1}$).[1] This is also known as extrinsic camera parameter calibration.

3.4.1 Previous Work

Object pose estimation is a well-studied problem. The general approaches can be grouped into two categories: iterative and non-iterative.

Let us first review the non-iterative approaches. If the model and image projection points are perfectly measured, this problem can be solved in closed-form by finding the perspective projection matrix using Direct Linear Transformation (DLT) [72, 85]. The projection matrix maps the 3D model points to image points in homogeneous coordinates. Since it has 11 unknowns, at least six nonplanar correspondences are required. However, these approaches are very fragile to noise. Fischler and Bolles [37] used the fact that relative distance between two points is preserved under rigid transformation to derive a fourth order polynomial in the unknown points depth for every triplets of points. Four solutions are obtained in general so a fourth point is needed to disambiguate. For arbitrary n, there are $\frac{n(n-1)}{2}$ fourth order polynomials one can stack them in a matrix form and solve it using SVD [69]. This solution requires $O(n^5)$ operations. Moreno-Nogu *et al.* [63] provided a linear solution by introducing four virtual control points and any 3D point was written as a weighted sum of them. They reduced PnP problem to estimating the coordinates of these control points in the camera referential.

Now let us switch to iterative methods. Lowe [56] and Yuan [102] improved the robustness of the estimates by minimizing the reprojection error. Since the projection function is nonlinear, they used Newton-Raphson method to optimize it. However, both algorithms require good initial values to converge and for both algorithms, each iteration is an $O(n^3)$ operation (requiring the pseudo-inverse of the Jacobian). Lu *et al.* [57] formulated the pose estimation as minimizing object-space collinearity error, from which they derived a fast iterative algorithm that was robust to outliers. DeMenthon and Davis proposed an accurate and efficient POSIT algorithm [33] that iteratively finds object pose by assuming a scaled orthographic projection.

[1] $\mathbf{Q}$ is used for rotation matrix to avoid conflict with DM $\mathbf{R}_k$

3.4.2 A SDM Solution

The 3D pose estimation problem can also be formulated as a constrained NLS problem that minimizes the reprojection error w.r.t. $\mathbf{Q}$ and $\mathbf{t}$:

$$\begin{aligned} \underset{\mathbf{Q},\mathbf{t}}{\text{minimize}} \quad & \|\mathbf{h}(\mathbf{Q},\mathbf{t},\mathbf{M}) - \mathbf{U}\|_F \\ \text{subject to} \quad & \mathbf{Q}^\top\mathbf{Q} = \mathbf{I}_3 \text{ and } \det(\mathbf{Q}) = 1. \end{aligned}$$

$\mathbf{h} = \mathbf{g}_2 \circ \mathbf{g}_1$ can be seen as composition of two functions $\mathbf{g}_1$ and $\mathbf{g}_2$, which can be written in closed-form as follows:

$$\begin{aligned} \mathbf{g}_1(\mathbf{Q},\mathbf{t},\mathbf{X}) &= \mathbf{K}(\mathbf{Q}\mathbf{X} + \mathbf{1}_n^\top \otimes \mathbf{t}), \\ \mathbf{g}_2(\mathbf{X}) &= \begin{bmatrix} \mathbf{x}_1^\top \oslash \mathbf{x}_3^\top \\ \mathbf{x}_2^\top \oslash \mathbf{x}_3^\top \end{bmatrix}, \end{aligned}$$

where $\otimes$ represents the Kronecker product, $\oslash$ denotes element-wise division, and $\mathbf{x}_1^\top$ is the first row vector of $\mathbf{X}$. We parameterize the rotation matrix as a function of the Euler-angles $\boldsymbol{\theta}$. Then, the objective function can be simplified into the following unconstrained optimization problem:

$$\min_{\mathbf{p}} \|\mathbf{h}(\mathbf{p},\mathbf{M}) - \mathbf{U}\|_F,$$

where $\mathbf{p} = [\boldsymbol{\theta};\mathbf{t}]$. We minimize the above function using reversed SDM introduced in Section 2.1.5. For training SDM, we sample a set of poses $\{\mathbf{p}_*^i\}$ and compute the image projections $\{\mathbf{U}^i\}$ under each pose. Recall that the training of reversed SDM alternates between minimization of Eq. 2.10 and updating of Eq. 2.9. We rewrite these equations in the context of pose estimation:

$$\begin{aligned} \mathbf{p}_k^i &= \mathbf{p}_{k-1}^i - \mathbf{R}_{k-1}(\mathbf{h}(\mathbf{p}_{k-1}^i,\mathbf{M}) - \mathbf{U}^i), \qquad (3.19) \\ \sum_i &\|\mathbf{p}_*^i - \mathbf{p}_k^i + \mathbf{R}_k(\mathbf{h}(\mathbf{p}_k^i,\mathbf{M}) - \mathbf{U}^i)\|_2^2. \end{aligned}$$

In testing, given an unseen $\tilde{\mathbf{U}}$, SDM recursively applies the update given by Eq. 3.19.

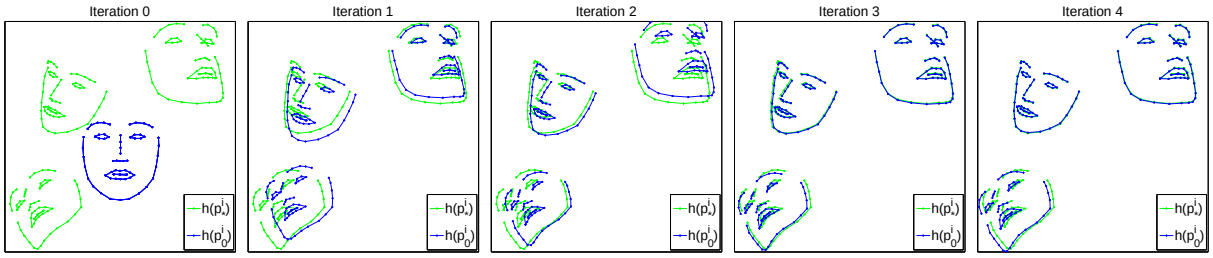


Figure 3.3: Three examples of SDM minimizing the reprojection errors through each step. Blue outlines represent the image projections $\mathbf{h}(\mathbf{p}_k^i)$ under the current parameter estimates $\mathbf{p}_k^i$. Green outlines arethe given inputs the algorithm trying to match.

Fig. 3.3 shows three examples of how the reprojection errors are decreased through each SDM update when performing head pose estimation. In these cases, the SDM always starts at $\mathbf{p}_0$ (see iteration 0 in Fig. 3.3) and quickly converges to the optimal solutions. More results can be found in section 4.6 as well as a comparison with the POSIT algorithm.

Chapter 4

Experimental Results and Discussions

"Success is the ability to go from one failure to another with no loss of enthusiasm."

Winston Churchill

4.1 Analytic Functions

In this section, we compare the performance of SDM and GSDM against gradient-based methods on minimizing analytic functions. The three competing methods are: Steepest Descent, Newton's method (with user provided Hessian), and LBFGS. For gradient-based methods, we used the existing implementation from the `minFunc` software package provided by Mark Schmidt[1]. Steepest Descent is implemented with backtracking line search [5, 62]. In the implementation of Newton's method, in the case of Hessian being a negative definite matrix, a damping factor is added. For all functions chosen in this experiment, their first and second derivatives can be derived analytically (See Fig. 1 in Appendix B). We supplied those to the `minFunc` optimizer so no numerical approximation is used for gradient or Hessian computation. For differentiable functions, SDM did not demonstrate much speed gain over gradient-based methods. In this experiment, we are particularly interested in how different optimization methods perform when minimizing non-convex functions and when they are initialized far away from the true values. Below, we start with scalar functions followed by multivariate functions.

4.1.1 Scalar Functions

The functions that we are optimizing share the same formula:

$$\min_x f(x) = (h(x) - y_*)^2,$$

where $h(x)$ is a nonlinear function (see Fig. 4.1) and y_* is a given constant. We have chosen only invertible functions (all functions are either monotonically increasing or decreasing within the selected domain). Otherwise, for a given y_* multiple solutions may be obtained. For all selected $h(x)$, the corresponding functions $f(x)$ are not convex but have a unique local/global minimum. All methods start from a fixed initial point, $x_0 = c$. The training data $\mathbf{x} = \{x^i\}$ are sampled uniformly. The test data $\mathbf{y}_* = \{y_*^i\}$ are generated by evaluating $h(x)$ at different values of x than those used in training. Gradient-based methods do not require a training step so they are directly evaluated on test data. The sampling details are shown in Fig. 4.1 following the Matlab notation.

Function $h(x)$	Training Set x^i	Test Set x_*^i	K
$\sin(x)$	$[-\frac{\pi}{2}{:}0.0031{:}\frac{\pi}{2}]$	$[-\frac{\pi}{2}{:}0.01{:}\frac{\pi}{2}]$	1
x^3	$[-2{:}0.031{:}4.1]$	$[-2{:}0.1{:}4]$	48
e^x	$[-10{:}0.031{:}10]$	$[-10{:}0.1{:}9]$	e^9
x^{-1}	$[1{:}0.031{:}10.1]$	$[1{:}0.1{:}10]$	1

Figure 4.1: Experimental setup for 1D analytic functions.

The training and testing algorithms of SDM and GSDM for minimizing scalar functions are presented in Appendix C (Fig. 2 and Fig. 3). Function `find` returns a set of indices that satisfy

[1] `http://www.cs.ubc.ca/~schmidtm/`

the boolean condition. Function `dhd_1d` partitions the training set into two according to the signs of $\Delta x^i \Delta y^i_*$. In practice, we specify a spill parameter that allows some overlap between the two sets. The spill parameter is measured in cosine similarity. The training of (G)SDM is finished if the training error is below some threshold or stops decreasing. The same number of iterations is then used in testing. For gradient-based methods, the optimization stops if one of the following conditions holds: function value is changed by less than a threshold, the magnitude of gradient is close to zero, step size is small, or the max number of iterations is reached.

Fig. 4.3 presents the convergence results of all competing methods. We plot the absolute deviation between the optimized results and the true values at different x_*s. All gradient-based methods behave similarly: they fail to converge for some range of x_* because the optimizing functions are not convex. GSDM always converges given enough iterations and it provides much faster convergence than SDM. Function $h(x) = \frac{1}{x}$ is monotonic over $(-\infty, 0)$ or $(0, \infty)$ but is not monotonic over $(-\infty, \infty)$. In our implementation of SDM, we add a clipping step to prevent optimizing parameter going out of bounds. Without this clipping step, SDM's training stalls after a few iterations. However, GSDM is not influenced in this special case and it converges with/without the clipping step.

From Fig. 4.3bc, we observed that when $h(x) = e^x$ both SDM and GSDM take the most steps to converge while minimizing $f(x) = (\sin(x) - y)^2$ takes the least number of iterations. Theorem 2 states that the norm of descent map is inverse proportional to K. That is, the bigger the K the more iterations it takes for SDM to converge. Fig. 4.1 also listed the Lipschitz constants K computed for every function in the range of test data. $K = e^9$ when $h(x) = e^x$ while $K = 1$ for $h(x) = \sin(x)$, which explains why one converges faster than another.

If f is quadratic (*e.g.*, h is linear function of x), SDM will converge in one iteration because the average gradient evaluated at different locations will be the same for linear functions.This coincides with a well-known fact that Newton's method converges in one iteration for quadratic functions.

4.1.2 Multi-dimensional Functions

This section extends the above experiments to multivariate functions:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \min_{\mathbf{x}} \|\mathbf{h}(\mathbf{x}) - \mathbf{y}_*\|^2,$$

where $\mathbf{h}(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}^2$ is a nonlinear function (see Fig. 4.2) and $\mathbf{y}_*$ is a given constant vector. We follow the same training and testing protocols used in the previous section. Now, the data are sampled uniformly on a 2D grid shown in Fig. 4.2. According to the inverse function theorem, $\mathbf{h}(\mathbf{x})$ is invertible locally if the determinant of its Jacobian is non-zero at $\mathbf{x}$. In this previous section, to guarantee a unique solution we select only monotonic functions. Similarly, now we select functions and domains such that the determinants of their Jacobians within the selected domains are greater than zero.

Fig. 4.4 presents the convergence results of the gradient-based methods. Each dot represents the optimal solution of a test data. If it is colored green, that indicates the algorithm has converged to the correct $\mathbf{x}_*$, otherwise, red. The plots of the same column are generated by the same method

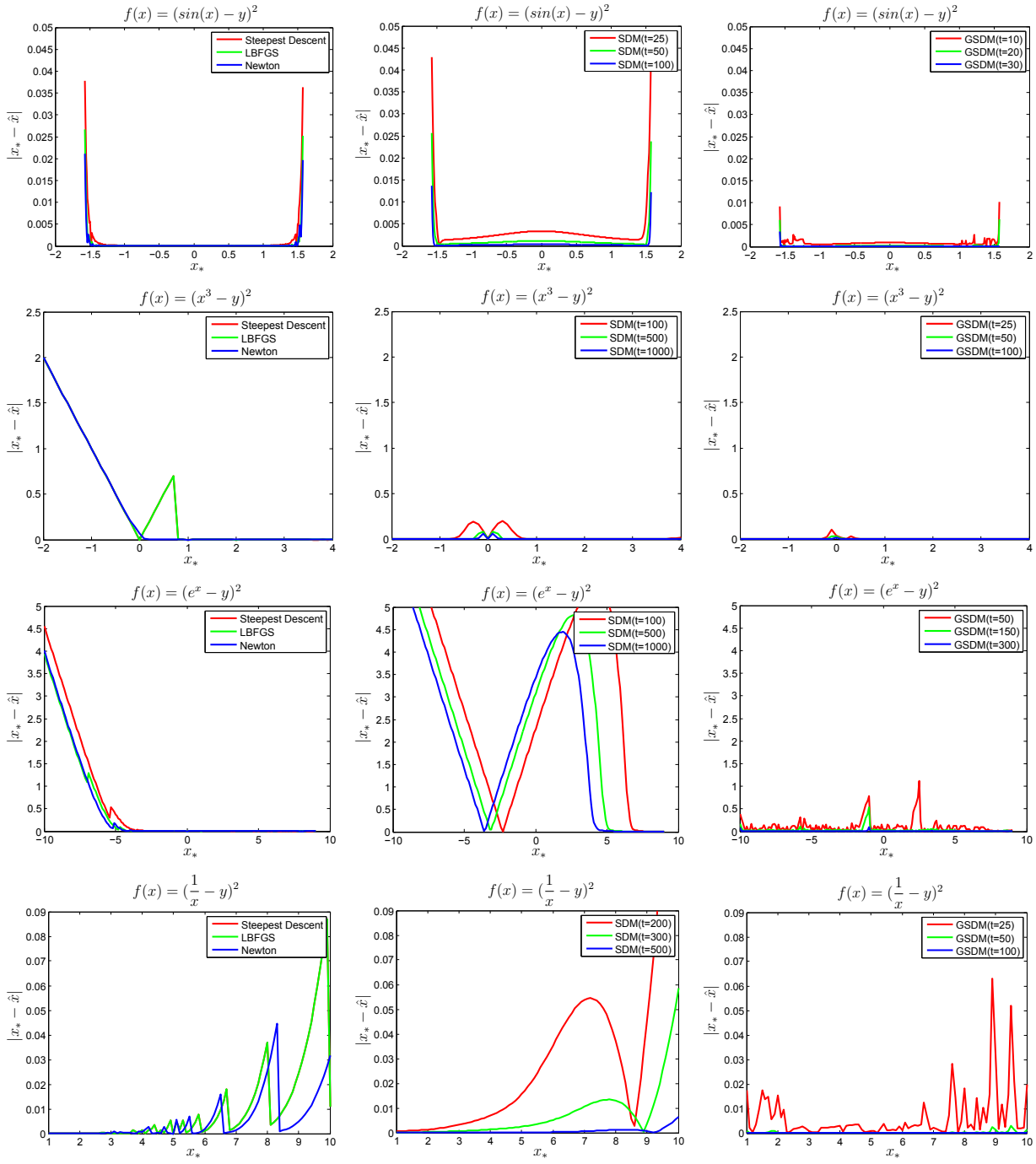
Function $\mathbf{h}(\mathbf{x})$	Training Set x_1^i	x_2^i	Test Set x_{*1}^i	x_{*2}^i	K
$\begin{bmatrix}\cos(x_1)x_2 \\ \sin(x_1)x_2\end{bmatrix}$	$[-\pi{:}0.07{:}\pi]$	[2:0.07:5.1]	$[-\pi{:}0.05{:}\pi]$	[2:0.05:5]	48
$\begin{bmatrix}2x_1x_2 \\ 5x_1 - x_2^2\end{bmatrix}$	[0.5:0.031:3.1]	$[-3{:}0.031{:}3.1]$	[0.5:0.05:3]	$[-3{:}0.05{:}3]$	1
$\begin{bmatrix}e^{x_1} + x_2^2 \\ x_1^2x_2\end{bmatrix}$	$[-2{:}0.031{:}-0.4]$	$[-3{:}0.031{:}3.1]$	[-3:0.05:-0.5]	$[-3{:}0.05{:}3]$	e^9

Figure 4.2: Experimental setup for 2D analytic functions.

and each row ties to a particular function. From top to bottom, the functions are:

$$f_1(\mathbf{x}) = \frac{1}{2}\left\|\begin{bmatrix}\cos(x_1)x_2 \\ \sin(x_1)x_2\end{bmatrix} - \mathbf{y}\right\|^2, f_2(\mathbf{x}) = \frac{1}{2}\left\|\begin{bmatrix}2x_1x_2 \\ 5x_1 - x_2^2\end{bmatrix} - \mathbf{y}\right\|^2, f_3(\mathbf{x}) = \frac{1}{2}\left\|\begin{bmatrix}e^{x_1} + x_2^2 \\ x_1^2x_2\end{bmatrix} - \mathbf{y}\right\|^2.$$

Different methods exhibit different convergence patterns and it is difficult to say which one is the best in general because it is function dependent. None of the three methods is perfect. Gradient-based methods are likely to converge to the wrong local minima. Fig. 4.5 shows three examples thereof. In those examples, the true solution is actually closer (measured in L_2 distance) from the initial starting position. Fig. 4.6 shows the convergence results of GSDM on the same three functions. Given enough iterations, GSDM always converge to the correction solution.



(a) (b) (c)

Figure 4.3: Convergence results on non-convex analytic functions. Each row represents a different function. a) Gradient-based methods; b) SDM; c) GSDM. In a) different colors represent different methods while in b) and c) different colors indicate different number of iterations.

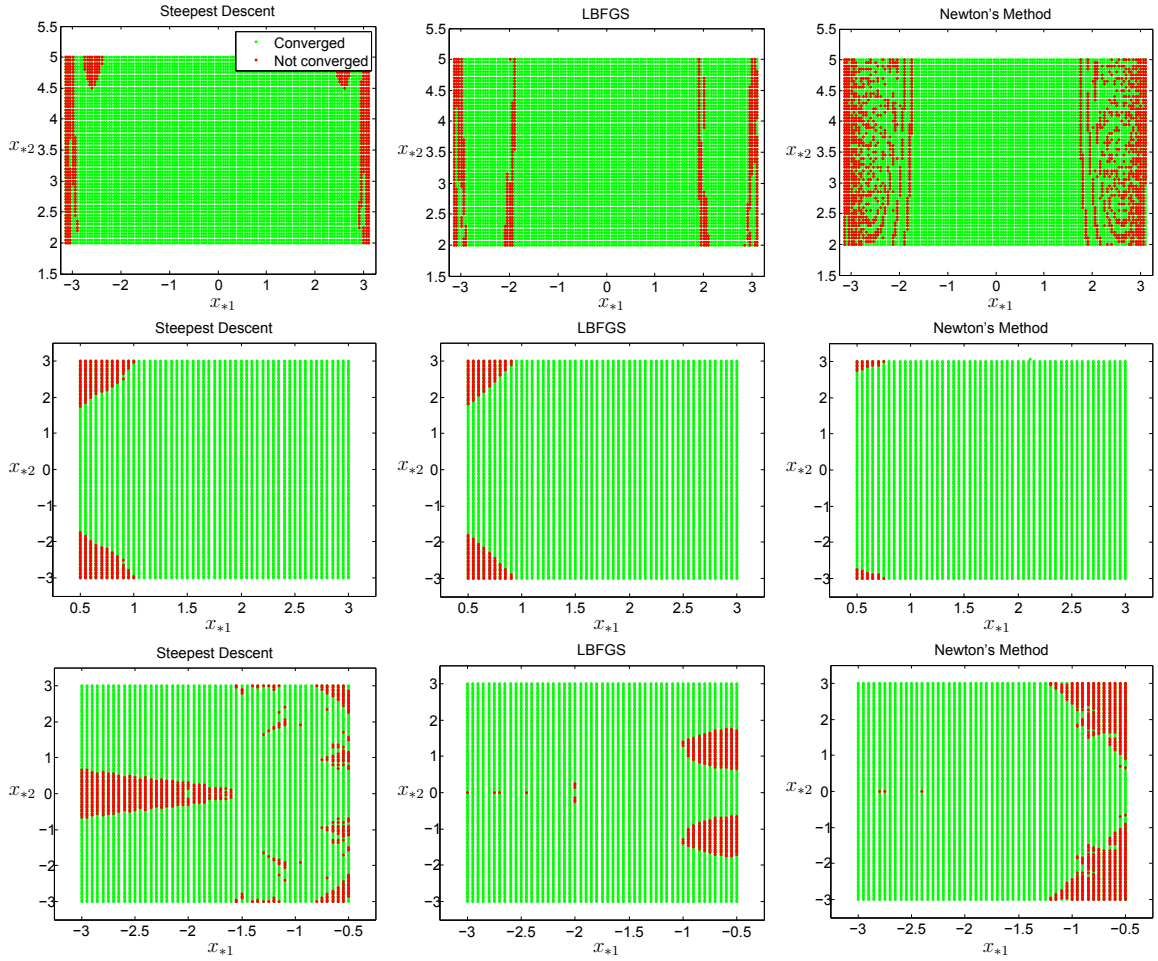


Figure 4.4: Convergence results for gradient-based methods on optimizing 2D analytic functions. a) Steepest Descent; b) LBFGS; c) Newton's method. Each dot represents the optimal solution of a test data. Green dots indicate that algorithm has converged to the correct solution. Red dots indicate failure.

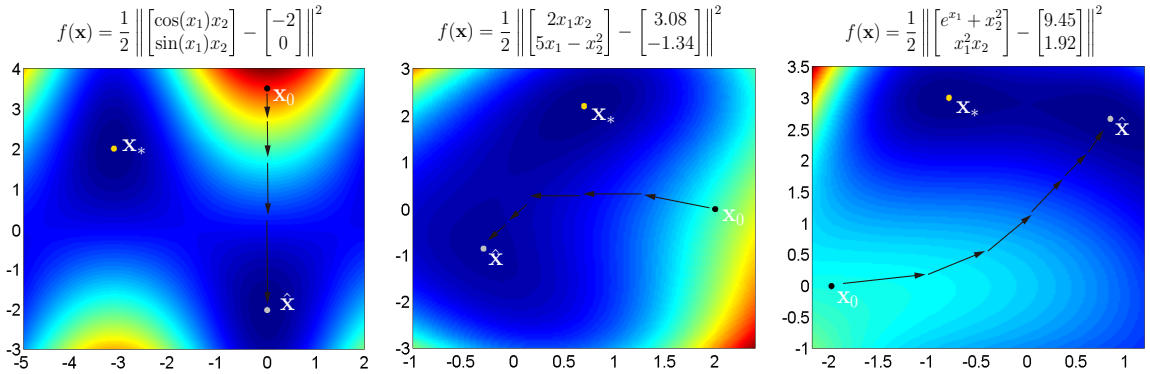


Figure 4.5: Failure cases of gradient-based methods. Gradient-based methods often converge to the wrong local minima. $\mathbf{x}_0$ is the initial starting point, $\mathbf{x}_*$ is the true solution, and $\hat{\mathbf{x}}$ is the solution by following the gradient direction.

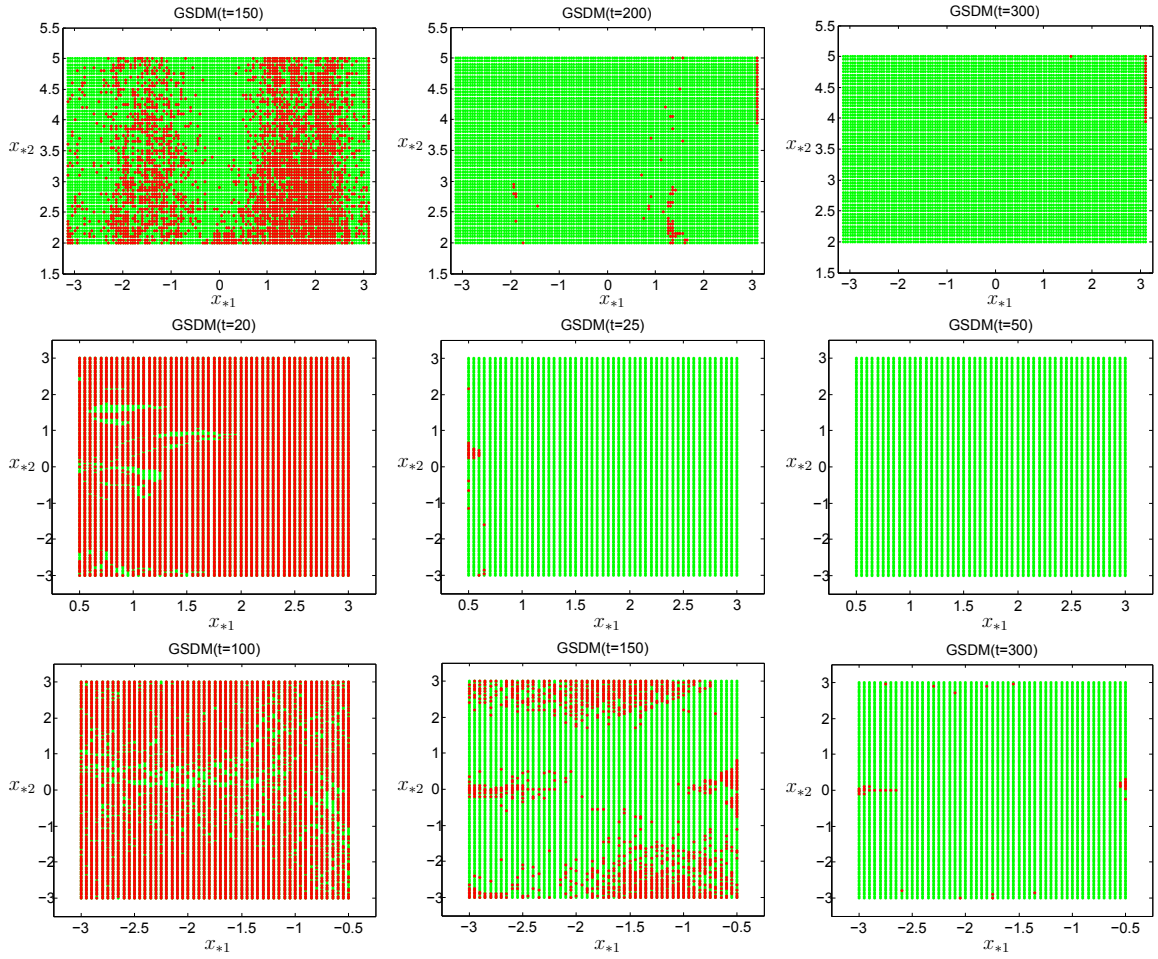


Figure 4.6: Convergence results for GSDM on optimizing 2D analytic functions. Each row shows the progress of GSDM on a different function as iteration increases. The number of iterations is given on top of each subplot. Each dot represents the optimal solution of a test data. Green dots indicate that algorithm has converged to the correct solution. Red dots indicate failure.

4.2 Inverse Kinematics

The Inverse Kinematics problem tries to determine the joint configurations $\boldsymbol{\theta}$ of a robot that provide a desired position $\mathbf{y}$ of the end-effector. It can be formualted as a NLS problem:

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \frac{1}{2}\|\mathbf{h}(\boldsymbol{\theta}) - \mathbf{y}\|^2, \tag{4.1}$$

where $\mathbf{h}$ is the forward kinematics function. This problem poses an interesting challenge to (G)SDM due to the presence of multiple global optima.

4.2.1 Two-arm Robot

We start with a toy example: a two-arm robot living in a 2D world (See Fig. 4.7). One end-point of the robot is fixed at the origin $(0, 0)$ and the position of the other end-point is controlled by the two joint angles (θ_1, θ_2). Depending on the desired position, zero, one, or two solutions may be obtained. In the example shown in Fig 4.7, the second solution is simply the reflection of the first one over the line connecting the two end-points. In this experiment, we assume that the provided position can be reached so we eliminate the case of zero solution.

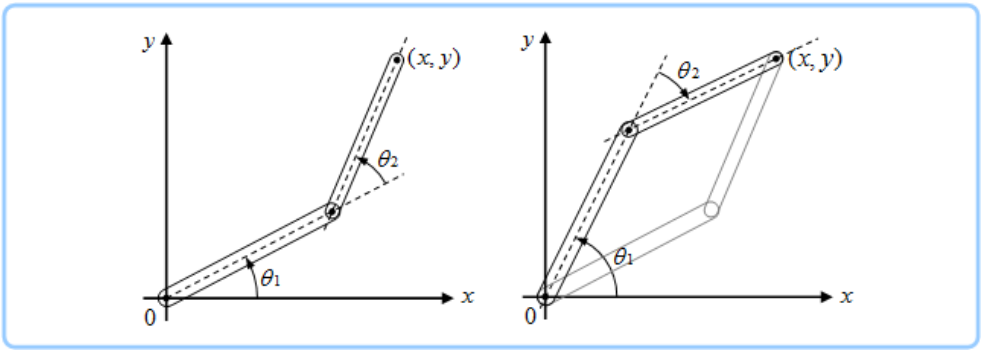


Figure 4.7: An example of a two-arm robot. One end-point of the robot is fixed at $(0, 0)$ and is controlled by the two joint angles (θ_1, θ_2). Multiple solutions can be obtained for an end point (x, y).

For the robot in Fig. 4.7, the corresponding forward kinematics function is given as follows:

$$\mathbf{h}(\boldsymbol{\theta}) = \begin{bmatrix} l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \\ l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \end{bmatrix}.$$

Its Jacobian matrix is:

$$\frac{\partial \mathbf{h}}{\partial \boldsymbol{\theta}} = \begin{bmatrix} -l_1 \sin(\theta_1) - l_2 \sin(\theta_1 + \theta_2) & -l_2 \sin(\theta_1 + \theta_2) \\ l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) & l_2 \cos(\theta_1 + \theta_2) \end{bmatrix}.$$

The determinant of the Jacobian is:

$$\left| \frac{\partial \mathbf{h}}{\partial \boldsymbol{\theta}} \right| = \sin(\theta_2).$$

Without loss of generality, we assume $\sin(\theta_2) > 0$, which imples $0 \leq \theta_2 \leq \pi$. To guarantee a unique solution, we reformulate (4.1) into a constrained optimization problem:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}(\mathbf{h}(\boldsymbol{\theta}) - \mathbf{y})^\top (\mathbf{h}(\boldsymbol{\theta}) - \mathbf{y}) \\ \text{subject to} & -\pi \leq \theta_1 \leq \pi \\ & 0 \leq \theta_2 \leq \pi. \end{array}$$

The legal region is a convex set, a rectangular box in $\mathbb{R}^2$ in particular.

We compare GSDM with Interior Point Method (IPM) [21, 94], implemented as `fmincon` function in the Matlab optimization toolbox. The training and testing algorithms of GSDM remains the same as the ones described in Appendix C (Fig. 4). The test data consists of a set of $\{(\mathbf{y}^i_{test}, \boldsymbol{\theta}^i_{test})\}$, where $\{\boldsymbol{\theta}^i_{test}\}$ are uniformly sampled from a grid inside the legal region with increments of 0.1 on both dimensions. $\mathbf{y}^i_{test}$ is computed using the forward kinematics formula given the input joint angles $\boldsymbol{\theta}^i_{test}$. $\{\boldsymbol{\theta}^i_{test}\}$ is used as ground truth to evaluate the performance of each method. The training data $\{\boldsymbol{\theta}^i_{train}\}$ for GSDM is sampled in a similar fashion with increments of 0.031. During the execution of GSDM, the optimizing parameters may go out of the legal region. In this case, we project them back to the legal region by simply clipping the exceeding values. Note that this is viable because the legal region is a rectangle. For non-convex constraint sets, the best strategy to incorporate those into GSDM is still unclear. It remains as one of the future research directions.

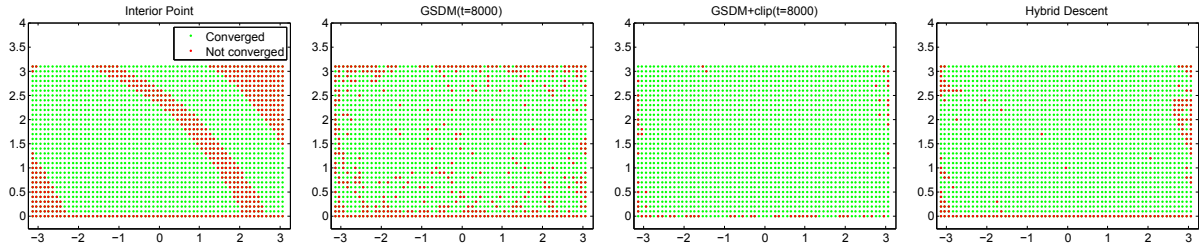


Figure 4.8: Each dot represent the true solution of a test data. Green dots indicate the algorithm converges to the correct values, otherwise red.

Fig. 4.8 presents the convergence results of all competing methods. With the "clipping" step, GSDM provides better convergence. Although GSDM (with clipping) outperforms all other methods, it took 8000 iterations while IPM takes around 20 to converge (sometimes to the wrong solutions). Number of iterations may not be a good indication of the run time of each method because in GSDM the amount of computation within each iteration is far less than those in the IPM. For example, given 2016 test samples GSDM take around 12 seconds to finish while IPM takes 58 seconds.

Within the first 10 iterations, GSDM reduces the residual by a half. From this observation, we propose a *Hybrid Descent Method* where we use the results from GSDM (first 10 iterations) as an initialization for the IPM. Hybrid Descent provides much better convergence results than using IPM alone, yet significantly reduces the number of iterations to a total of 30 iterations (10 iterations of GSDM plus 20 iterations of IPM).

IPM is gradient-based and sometimes converges to a wrong solution. In Fig. 4.9, we select two examples of such mistakes and plot the heat maps of their corresponding objective functions.

The selected samples are circled by a blue outline. In the top example, IPM converges to the futher local minima while in the bottom one it converges to the closer local minima. Both cases are incorrect, which demonstrates the limitations of gradient-based approaches. Fig. 4.9c shows why Hybrid Descent performs better than using IPM alone. The white lines show the traces of the first ten iterations of GSDM where $\mathbf{x}$ is taken close enough to the true solution $\mathbf{x}_*$. Thus, IPM is able to converge to the correct minima given a good initialization.

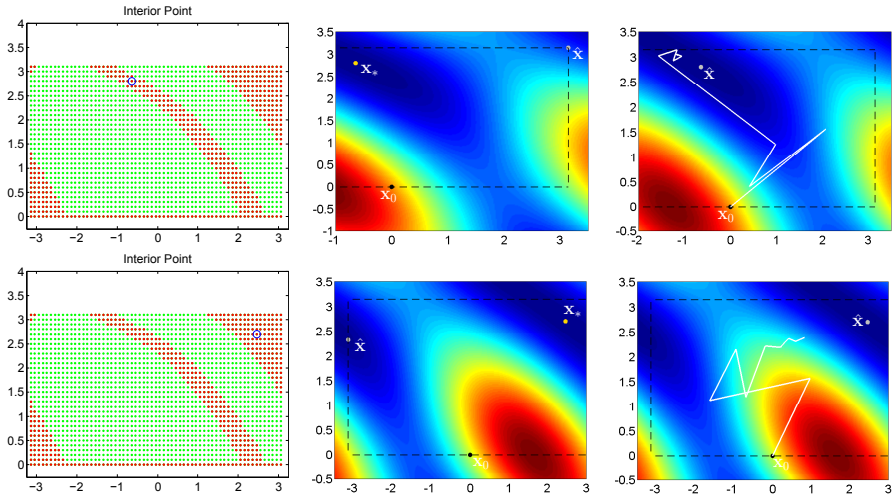


Figure 4.9: a) the test data is circled by a blue outline. b) two examples of IPM converging to the wrong solution. Each heat map describes the values of an objective function for a particular test sample. The test sample is circled by a blue outline. $\mathbf{x}_0$ is the initial position, $\mathbf{x}_*$ is the optimal solution, and $\hat{\mathbf{x}}$ is where the algorithm converges to. Black dashed lines are the borders of the legal region. c) white lines show the traces from the first ten iterations of GSDM. $\hat{\mathbf{x}}$ is the solution to Hybrid Descent.

4.2.2 Three-arm Robot

In this section, we add one additional arm to the above robot, and this makes the problem significantly more complicated. Here we assume that all arms share the same length l. In our three-arm robot example, the forward kinematics function becomes:

$$\mathbf{h}(\boldsymbol{\theta}) = \begin{bmatrix} l\cos(\theta_1) + l\cos(\theta_1+\theta_2) + l\cos(\theta_1+\theta_2+\theta_3) \\ l\sin(\theta_1) + l\sin(\theta_1+\theta_2) + l\sin(\theta_1+\theta_2+\theta_3) \end{bmatrix}.$$

Its Jacobian $\mathbf{J_h} \in \mathbb{R}^{2\times 3}$ is not a square matrix. To guarantee a unique solution, we compute the determinants of all three 2×2 minors and focus on the case where all of them are greater than zero:

$$\det(\mathbf{J_h}(\emptyset, 1) = \sin(\theta_3) \tag{4.2}$$
$$\det(\mathbf{J_h}(\emptyset, 2) = l^2\sin(\theta_3) + l^2\sin(\theta_2+\theta_3) \tag{4.3}$$
$$\det(\mathbf{J_h}(\emptyset, 3) = l^2\sin(\theta_2) + l^2\sin(\theta_2+\theta_3). \tag{4.4}$$

$\mathbf{J_h}(i,j)$ denote a submatrix matrix, obtained by deleting row i and column j. Rewriting those constraints gives us the following constrained optimization problem:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}(\mathbf{h}(\boldsymbol{\theta}) - \mathbf{y})^\top (\mathbf{h}(\boldsymbol{\theta}) - \mathbf{y}) \\ \text{subject to} & -\pi \leq \theta_1 \leq \pi \\ & 0 \leq \theta_3 \leq \pi \\ & -\theta_3 - 2\theta_2 \leq 0 \\ & \theta_3 + \frac{1}{2}\theta_2 \leq \pi \\ & \theta_3 + 2\theta_2 \leq 2\pi. \end{array}$$

The constraint set is convex because it is formed by the intersection of seven half spaces.

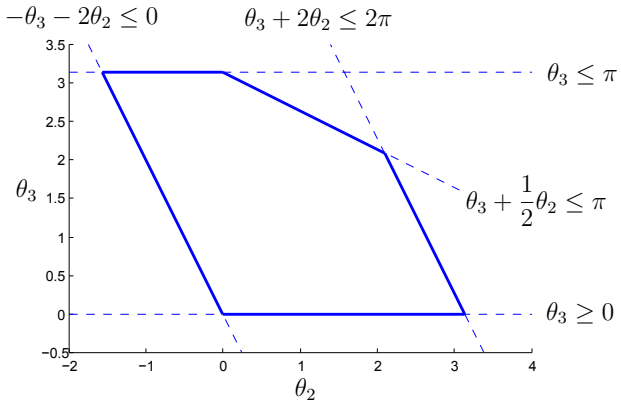


Figure 4.10: A visualization for the constraints on parameters θ_2 and θ_3. The dashed lines half spaces and the solid lines are borders of the legal region.

Fig. 4.10 illustrates the convex set formed by the constraints posed on θ_2 and θ_3. We omit the constraints on θ_1 because they are relatively simple. The overall shape of the convex set can be imagined as a 3D cube where each half space carves out part of the cube.

We compare GSDM with IPM and Hybrid Descent. The training and testing procesures of GSDM remain similar as in the previous experiment but with one slight modification. In function `dhd_2d`, four additional subsets are created based on the signs of $\Delta\theta_3$. The training and test data are generated by uniformly sampling $\boldsymbol{\theta}$ in $\mathbb{R}^3$, and then rejecting the samples that fall out of the legal region. Fig. 4.11 presents the results from three competing method. GSDM is able to achieve above 99% convergence rate with 20K iterations. Hybrid Descent improves upon Interior Point method by 8% by only using 10 iterations of GSDM. Given 7474 test samples, GSDM (with 20K iterations) finish around 87 seconds while IPM takes 481 seconds.

Interior Point	GSDM			Hybrid Descent
	$t = 10^4$	$t = 1.5 \cdot 10^4$	$t = 2 \cdot 10^4$	
83.3	54.6	95.9	99.3	91.1

Figure 4.11: Convergence rates of the three-arm robot experiment. The table presents the percentage of the test data that has converged to the correct solutions.

4.3 Rigid Tracking

This section presents the tracking results comparing LK and SDM using an affine transformation. We used a publicly available implementation of the LK method [7] for tracking a single template. The experiments are conducted on a public dataset[2] published by [39]. The dataset features six different planar textures: mansion, sunset, Paris, wood, building, and bricks. Each texture includes 16 videos, each of which corresponds to a different camera motion path or changes illumination condition. In our experiment, we chose five of the 16 motions, giving us a total of 30 videos. The five motions correspond to translation, dynamic lighting, in-plane rotation, out-plane rotation, and scaling.

Both trackers (SDM and LK) used the same template, which was extracted at a local region on the texture in the first frame. In our implementation of SDM, the motion parameters were sampled from an isotropic Gaussian distribution with zero mean. The standard deviations were set to be $[0.05, 0.05, 0.05, 0.05, 8, 8]^\top$. We used 300 samples and four iterations to train SDM. The tracker was considered lost if there was more than 30% difference between the template and the back-warp image. Note that this difference was computed in HoG space.

		mansion	sunset	Paris	wood	building	bricks
translation (87)	SDM	**87**	87	**86**	**87**	**87**	**87**
	LK	3	87	18	3	3	5
in-plane rtn (49)	SDM	32	**22**	49	**35**	20	**39**
	LK	32	18	49	26	20	24
lighting (99)	SDM	99	**99**	99	**99**	99	99
	LK	99	16	99	24	99	99
out-plane rtn (49)	SDM	42	35	**41**	34	37	**37**
	LK	42	35	37	34	**39**	35
scaling (49)	SDM	**49**	**49**	**49**	**49**	43	**49**
	LK	11	13	28	29	**49**	21

Figure 4.12: Comparison between SDM and LK on rigid tracking experiments. Each entry in the table states the number of frames successfully tracked by each algorithm. The total number of frames is given by number in the parentheses from the first column.

Fig. 4.12 shows the number of frames successfully tracked by the LK tracker and SDM. SDM performs better than or as well as the LK tracker in 28 out of the 30 sequences. We observe that SDM performs much better than LK in translation. One possible explanation is that HoG features are more robust to motion blur. Not surprisingly, SDM performs perfectly in the presence of dynamic lighting because HoG is robust to illumination changes. In-plane rotation tends to be the most challenging motion for SDM, but even in this case, it is very similar to LK.

[2] http://ilab.cs.ucsb.edu/tracking_dataset_ijcv/

4.4 Facial Feature Detection

This section reports experiments on facial feature detection in two "face-in-the-wild" datasets, and compares SDM with state-of-the-art methods. The two face databases are the LFPW dataset[3] [10] and the LFW-A&C dataset [78].

The experimental setup is as follows. First, the face is detected using the OpenCV face detector [16]. The evaluation is performed on those images in which a face can be detected. The face detection rates are 96.7% on LFPW and 98.7% on LFW-A&C, respectively. The initial shape estimate is given by centering the mean face at the normalized square. The translational and scaling differences between the initial and true landmark locations are also computed, and their means and variances are used for generating Monte Carlo samples in Eq. 2.6. We generated 10 perturbed samples for each training image. HoG descriptors are computed on 32×32 local patches around each landmark. To reduce the dimensionality of the data, we performed PCA, preserving 98% of the energy on the image features.

LFPW dataset contains images downloaded from the web that exhibit large variations in pose, illumination, and facial expression. Unfortunately, only image URLs are given and some are no longer valid. We downloaded 884 of the 1132 training images and 245 of the 300 test images. We followed the evaluation metric used in [10], where the error is measured as the average Euclidean distance between the 29 labeled and predicted landmarks. The error is then normalized by the inter-ocular distance.

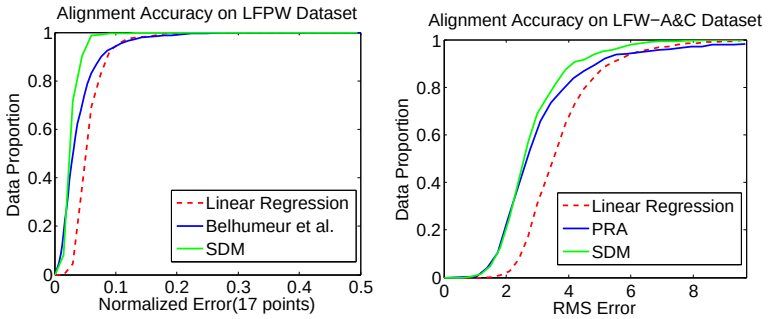


Figure 4.13: CED curves from LFPW and LFW-A&C datasets.

We compared our approach with two recently proposed methods [10, 23]. Fig. 4.13 shows the Cumulative Error Distribution (CED) curves of SDM, Belhumeur *et al.* [10], and our method trained with only one linear regression. Note that SDM is different from the AAM trained in a discriminative manner with linear regression [26] because we do not learn a shape or appearance model. Note that such curves are computed from 17 of the 29 points defined in [30], following the convention used in [10]. Clearly, SDM outperforms [10] and linear regression. It is also important to notice that a completely fair comparison is not possible since [10] was trained and tested with some images that were no longer available. However, the average is in favor of our method. The recently proposed method in [23] is based on boosted regression with pose-indexed features. To the best of our knowledge this paper reported the state-of-the-art results on LFPW

[3] `http://www.kbvt.com/LFPW/`

dataset. In [23], no CED curve was given and they reported a mean error ($\times 10^{-2}$) of 3.43. SDM shows comparable performance with a average of 3.47.

Figure 4.14: Example facial feature detection results from SDM on LFPW dataset. The first two rows show faces with strong changes in pose and illumination, and faces partially occluded. The last row shows the 10 **worst** images measured by normalized mean error.

The first two rows of Fig. 4.14 show our results on faces with large variations in poses and illumination as well as ones that are partially occluded. The last row displays the *worst* 10 results measured by the normalized mean error. Most errors were caused by the gradient feature's inability to distinguish between similar facial parts and occluding objects (*e.g.*, glasses frame and eye brows).

LFW-A&C is a subset of the LFW dataset[4], consisting of 1116 images of people whose names begin with an 'A' or 'C'. Each image is annotated with the same 66 landmarks shown in Fig. 3.2. We compared our method with the Principle Regression Analysis (PRA) method [78], which proposes a sample-specific prior to constrain the regression output. This method achieves the state-of-the-art results on this dataset. Following [78], those whose name started with 'A' were used for training, giving us a total of 604 images. The remaining images were used for testing. Root mean squared error (RMSE) was used to measure the alignment accuracy. Each image has a fixed size of 250×250 and the error was not normalized. PRA reported a median alignment error of 2.8 on the test set while ours averages 2.7. The comparison of CED curves can be found in Fig. 4.13b and our method outperforms both PRA and Linear Regression. Qualitative results from SDM on the more challenging samples are plotted in Fig. 4.15.

4.5 Facial Feature Tracking

Over the past few years, researchers in the face alignment field have made rapid progress on improving the landmark accuracy and speed of the algorithms. Such progress is made possible by the availability of larger and more challenging datasets *e.g.*, LFPW [10], Helen [52], AFLW [49], AFW [76], IBUG [75]. However, there is a lack of datasets for evaluation of face tracking from profile to profile as well as a standard protocol for evaluating tracking performance. To fill the

[4] `http://vis-www.cs.umass.edu/lfw/`

Figure 4.15: Example facial feature detection results from SDM on LFW-A&C dataset.

void, we build two challenging datasets, Distracted Driver Face(DDF) and Naturalistic Driving Study(NDS), and propose a standard evaluation protocol for facial feature tracking. Both the evaluation protocol code and NDS dataset are made available for the research community[5].

The **DDF dataset** contains 15 video sequences, a total of 10,882 frames. Each sequence captures a single subject performing distracted driving in a stationary vehicle or an indoor environment. 12 out of 15 videos are recorded with subjects sitting inside of a vehicle. Five of them are recorded in the night under infrared (IR) light and the others are recorded during the daytime under natural lighting. The remaining three are recorded indoors. The top three rows in Fig. 4.17 shows one subject from each category.

The **NDS dataset** [90] contains 20 sequences of driver faces recorded during a drive conducted between the Blacksburg, VA and Washington, DC areas. Each sequence consists of a one-minute video recorded at 15 fps with a resolution of 360×240. For both datasets, we labeled one in every ten frames and each labeled frame consists of either 49 landmarks (near-frontal faces) or 31 landmarks (profile faces). Both datasets consist of many faces with extreme pose ($\pm 90^\circ$ yaw, $\pm 50^\circ$ pitch) and many under extreme lighting condition (*e.g.*, IR). NDS is the more challenging one due to the low spatial and temporal resolution.

Evaluation protocol: A popular evaluation metric for facial feature detection is the cumulative error curve. However, this curve cannot take into account the frames that are lost during tracking. We propose the Cumulative Error Histogram (CEH) as the evaluation metric. The idea of CEH is to quantize the tracking error at different scales. The histogram will have k bins, where the i^{th} bin counts the fraction of frames (number of frames over total number of frames) with errors less than the i^{th} error scale. For the frames where the tracker is lost or the landmark error is larger than the last error scale, we add them to the last bin. For the successfully tracked frames, the error is measured using the normalized root mean square (RMS) metric. In previous work, normalization is often done by using the inter-ocular distance. However, for a profile face such distance tends to go to zero so we use the face length as a reference approximated by the distance

[5] `http://humansensing.cs.cmu.edu/xxiong`

between the lower lip point and the inner eyebrow point. The mean of all bins in a CEH can be used as single-value score to compare among different tracking methods. The CEH score has the value between 1 and $\frac{1}{k}$ with higher value indicating better performance. In the worst case, *e.g.*, no face is tracked in a sequence, all bins except the last one equal to zero yielding a score of $\frac{1}{k}$. On the other hand, the score equals one if all frames fall in the first bin.

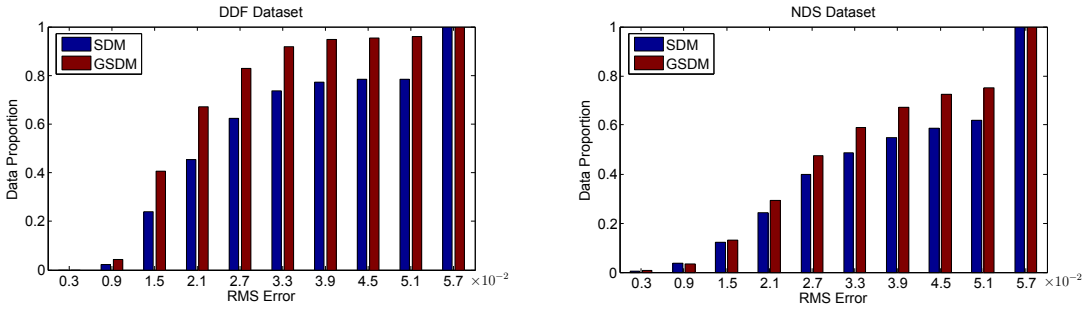


Figure 4.16: Performance comparison between SDM and GSDM in terms of CEH on DDF dataset (left) and NDS dataset (right).

In the experiments, both the SDM and GSDM algorithms are trained on MPIE [40] and a subset of LFW [45]. We use CEH to measure the performance of each tracker, and $k = 10$ and the max error is set to be 0.06. A face detector (OpenCV [16] in our case) is called once the tracker is lost and the tracker is not re-initialized until a valid face is detected. No manual effort is involved to re-initialize both trackers. Fig. 4.16 shows CEHs between SDM and GSDM in both datasets. GSDM is able to track more frames and provides more accurate landmark prediction than SDM. Both algorithms have significant performance drop-off in NDS dataset because of the noisy, low resolution images and heavy occlusion introduced by the sunglasses. Additionally, images in NDS dataset are significantly different than the ones in our training set. Example results can be found in Fig. 4.17 or from the link below[6]. Our C++ implementation averages around 8ms per frame, tested with an Intel i7 3752M processor.

4.6 3D Pose Estimation

This section reports the experimental results on extrinsic camera calibration using GSDM and a comparison with SDM and the widely popular POSIT method [33].

The experiment is set up as follows. We selected three different meshes of 3D objects: a cube, a face, and a human body[7] (see Fig. 4.18). In the training of GSDM, we follow a similar partition strategy introduced in Section 3.3.2. Each dimension in the parameter space is independent of each other so no dimension reduction is needed. DHD are found by splitting the parameter space according to three rotation angles. Each domain within DHD occupies an octant in $\mathbb{R}^3$. It gives us eight DMs to learn in every iteration and the number of iterations for both SDM and GSDM are set to be 10. In testing, unlike in the tracking application where we can use the previous

[6] `http://goo.gl/EGiUFV`
[7] `www.robots.ox.ac.uk/~wmayol/3D/nancy_matlab.html`

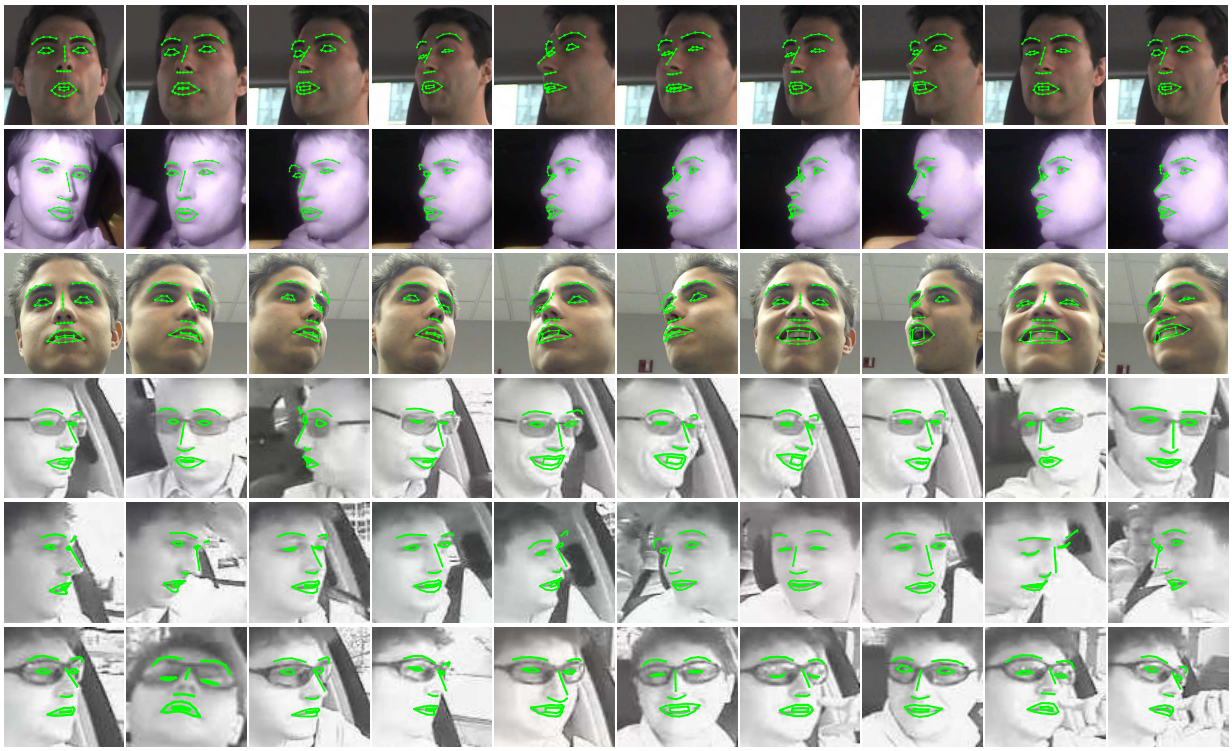


Figure 4.17: Tracking results from GSDM on the DDF dataset (top three rows) and NDS dataset (bottom three rows).

frame information as an approximation of $\mathbf{x}_*$, we iterate through all DMs and uses the one that returns the minimum reprojection error.

We placed a virtual camera at the origin of the world coordinates. In this experiment, we set the focal length (in terms of pixels) to be $f_x = f_y = 1000$ and principle point to be $[u_0, v_0] = [500, 500]$. The skew coefficient was set to be zero. The training and testing data were generated by placing a 3D object at $[0, 0, 2000]$, perturbed with different 3D translations and rotations. The POSIT algorithm does not require labeled data. Three rotation angles were uniformly sampled from -60° to 60° with increments of 10° in training and 7° in testing. Three translation values were uniformly sampled from -400mm to 400mm with increments of 200mm in training and 170mm in testing. Then, for each combination of the six values, we computed the object's image projection using the above virtual camera and used it as the input for both algorithms. White noise ($\sigma^2 = 4$) was added to the projected points. In our implementation of SDM, to ensure numerical stability, the image coordinates $[u, v]$ of the projection were normalized as follows: $\begin{bmatrix} \hat{u} \\ \hat{v} \end{bmatrix} = \begin{bmatrix} (u - u_0)/f_x \\ (v - v_0)/f_y \end{bmatrix}$.

Fig. 4.19 shows the mean errors and standard deviations of the estimated rotations (in degree) and translations (in mm) for three algorithms. SDM performs the worst among the three because the parameter space is so large that there not exists a single DM. GSDM overcomes this problem by partitioning the large space into eight subsets and learning eight DMs. Both GSDM and POSIT achieve around 1° accuracy for rotation estimation, but GSDM is much more accurate

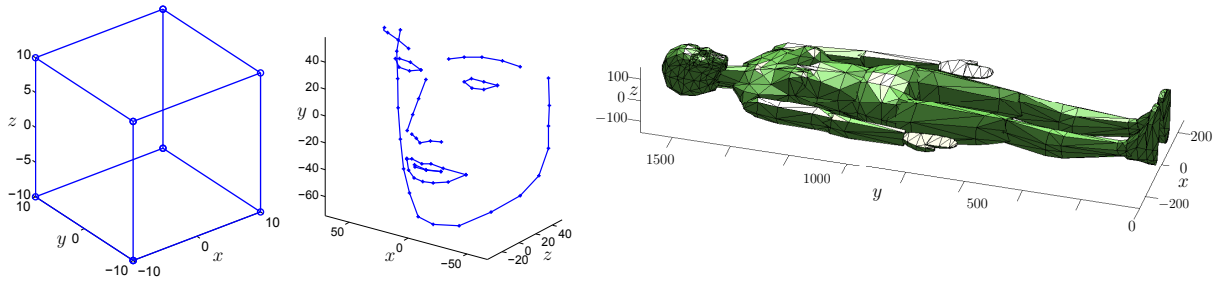


Figure 4.18: 3D objects used in pose estimation experiments. Units are in millimeters (mm).

		θ_x	θ_y	θ_z	t_x	t_y	t_z
Cube	SDM	5.2 ± 5.2	8.9 ± 8.4	10.5 ± 8.9	18.0 ± 14.4	18.4 ± 14.2	129.9 ± 120.9
	GSDM	$\mathbf{0.7 \pm 0.7}$	0.9 ± 0.9	0.7 ± 0.7	$\mathbf{2.4 \pm 3.1}$	$\mathbf{2.4 \pm 3.1}$	$\mathbf{17.1 \pm 17.3}$
	POSIT	0.8 ± 0.7	$\mathbf{0.9 \pm 0.8}$	$\mathbf{0.7 \pm 0.6}$	66.5 ± 48.1	66.3 ± 51.3	69.4 ± 50.3
Face	SDM	5.8 ± 6.2	10.3 ± 10.4	10.9 ± 12.4	14.6 ± 18.1	14.3 ± 18.7	123.4 ± 132.9
	GSDM	$\mathbf{0.8 \pm 1.0}$	$\mathbf{1.1 \pm 1.2}$	$\mathbf{0.9 \pm 1.0}$	$\mathbf{2.5 \pm 8.3}$	$\mathbf{2.4 \pm 7.8}$	$\mathbf{18.9 \pm 19.8}$
	POSIT	1.5 ± 1.3	1.9 ± 1.7	1.5 ± 1.3	28.6 ± 24.4	32.5 ± 22.8	47.3 ± 36.8
Body	SDM	3.0 ± 4.4	4.4 ± 6.1	4.7 ± 6.9	12.1 ± 20.5	12.3 ± 20.7	101.3 ± 134.2
	GSDM	$\mathbf{0.6 \pm 0.1}$	$\mathbf{0.7 \pm 0.1}$	$\mathbf{0.8 \pm 0.1}$	$\mathbf{0.3 \pm 0.8}$	$\mathbf{0.3 \pm 0.9}$	$\mathbf{1.5 \pm 3.9}$
	POSIT	0.6 ± 0.6	2.5 ± 2.6	1.1 ± 0.9	38.0 ± 22.1	28.5 ± 27.2	37.8 ± 30.9

Figure 4.19: Performance comparison among GSDM, SDM, and POSIT algorithms on estimating 3D object pose. Rotation (in degree) and translation (in mm) errors and their standard deviations.

for translation. This is because POSIT assumes a scaled orthographic projection, while the true image points are generated by a perspective projection.

Chapter 5

Conclusions and Future Directions

> "A paper is accepted or rejected before it is ever submitted."
>
> *X. X.*

5.1 Conclusions

In this dissertation, we focus on solving NLS problems using a supervised approach. In particular, we developed a Supervised Descent Method, performed thorough theoretical analysis of this approach, and demonstrated its effectiveness on optimizing analytic functions and solving four other real-world applications, Inverse Kinematics, Rigid Tracking, Face Alignment (frontal and multi-view), 3D Object Pose Estimation. In the following, we conclude this thesis with a summary of our contributions and the method's limitations.

5.1.1 Contributions

Below, we list our contributions:

- **Theoretical analysis of (G)SDM:**
  We introduced and validated a novel concept, generic DM, from which we developed a practical algorithm SDM for minimizing NLS functions. Later, we derived the conditions under which SDM will converge. For functions with multiple local minima, we extended the concept of DM and proved that there existed a finite partition of the function domain such that a separate DM existed within each subset. Finally, we established the connection between SDM and Imitation Learning (IL). More explicitly, DM can be interpreted as an optimization policy in the context of IL.
- **Applications of (G)SDM:**
  We focused on the applications that can be formulated into a NLS problem. They can be grouped into three categories depending on the information on $\mathbf{y}_*$.

  In Rigid Tracking, the template $\mathbf{y}_*$ is given in testing and the same one is used for training SDM. SDM was able to take advantage of more robust features, such as, HoG and SIFT. Those non-differentiable image features were out of consideration of previous work [6, 58] because they relied on gradient-based methods for optimization.

  In Inverse Kinematics, the desired position $\mathbf{y}_*$ is given in testing but different from those we have used for training SDM. In Section 2.1.5, we developed a generalized version of SDM to address this case and achieved significantly better convergence than gradient-based approaches. The problem of Object Pose Estimation falls in the same category.

  In Face Alignment, the template $\mathbf{y}_*$ is unknown in testing. SDM introduced an additional bias term to learn an average template that replaces $\mathbf{y}_*$. SDM achieved state-of-the-arts results in facial feature detection and tracking. Moreover, it was extremely computationally efficient, which makes it applicable for many mobile applications. In addition, GSDM was developed to handle the multi-view case where large pose variations are expected.
- **A unified framework for sequential prediction algorithms:**
  We reviewed seven representative methods on sequential prediction and discussed the differences between each of them and SDM. Then, we provided a unified view of all methods including SDM as a sequence of function compositions.
- **Dataset and software:**
  We built a challenging public dataset and also proposed an evaluation protocol for bench-

marking facial feature tracking methods. To the best of our knowledge, this was the first public dataset for evaluating facial feature tracking on profile-to-profile faces. We integrated the SDM-based tracker into IntraFace, a free software package for facial image analysis research. The software had accumulated 4800 downloads in eight months that was active, and it had proven useful for many researchers working on facial image analysis.

5.1.2 Limitations

Here is a list of SDM's limitations:

- The convergence of (G)SDM may be slow for certain functions. In our experiments of Inverse Kinematics, GSDM took thousands of iterations to converge. In this case, we proposed a Hybrid Descent Method where the first few iterations of (G)SDM were used to provide a better initialization for gradient-based methods.
- The optimal partition strategy for GSDM is unclear. In Section 2.1.6, we derived a simple partition strategy that guaranteed the convergence of GSDM. However, it was not practical due to the following two problems: $\mathbf{x}_*$ was unknown in testing time and the number of DMs needed was exponential to the dimension of the optimizing parameters. We developed two approximations to address those issues. For example, for high-dimensional functions we first projected the input variables onto their first few principal components. In tracking applications, we approximated $\mathbf{x}_*$ using the prediction from the previous frame. The optimality of these two approximations remains unclear.
- For functions with multiple global optima, gradient-based methods may be a better choice as they will converge to one of local minima because they are local methods. GSDM addresses this issue by creating a partition where a unique solution is guaranteed within each subset. However, as stated above the optimal partition strategy remains unclear.
- The learned DMs are object-specific. For example, if SDM is trained on face images it cannot be used to align non-face objects, even in the case where two objects may share the same number of landmarks.

5.2 Future Directions

In this section, we discuss possible potential extensions and future research directions on SDM and other sequential prediction methods.

5.2.1 SDM

Here are some of the unanswered algorithmic questions on SDM:

- *How to handle non-convex constraints?*
  In Section 4.2, we showed that a simple clipping step can be added to SDM to incorporate convex constraints. It is unclear what extra steps are needed if the constraints are non-convex. One can still project the violating samples back to the legal region but projecting onto a non-convex set is not straightforward.

- *How to generate training samples from a constrained set?*
  In Sections 4.1 and 4.2, the training data was generated from a region where a unique solution was guaranteed. For parametric functions, such guarantee is encoded in the signs of the Jacobian determinant (if the Jacobian is not a square matrix, we use the determinants of all its minors instead). For non-differentiable functions, it is unclear how we can generate training samples to satisfy such guarantee.
- *What is next for face alignment?*
  In our current system, most failure cases are caused by occlusion. This may be fixed by learning a robust feature representation through a CNN. However, tremendous amount of the training labels are required for training a complex network. Also, for face alignment the labeling task is significantly more time consuming than others. The current public datasets are in the order of thousands. One may investigate learning a mixture of other tasks combined with face alignment [105], such as, facial attributes recognition, face recognition.

We think the following theoretical questions are worth of investigating:

- *Can we prove that learning a sequence of DMs will converge?*
  In this thesis, we derived the conditions under which SDM will converge using a single DM. In the proof, we assumed that the initial samples were uniformly distributed in the neighborhood of $\mathbf{x}_*$ and the DM was found at the beginning by minimizing the maximum error of all samples. That is, we took the most conservative step in each iteration. In practice, we observed that samples were likely to form a very different distribution than the one in the previous iteration. Relaxed SDM learned a separate DM in each iteration to take advantage of this observation. This strategy yielded faster convergence in practice.
- *Can we provide a probability bound on the convergence of an unseen sample using SDM?*
  In Supervised Learning, the test error can be bounded by the training error using statistical analysis. One can adapt a similar proof from the area of regression to bound the convergence in SDM. The difference is that SDM involves a sequence of regressions where the current learner depends on the previous outputs. This may increase the difficulty of theoretical analysis.

5.2.2 Sequential Prediction

In Section 2.2, we reviewed seven different methods on sequential prediction and showed that all of them could be reformulated as a sequence of function compositions.

Sequential prediction goes beyond any particular application. The classical computer vision pipeline can be interpreted as a sequence of function compositions: from image preprocessing, to feature extraction, to dimension reduction, and finally to classification/regression. Each submodule is an individual function that takes the outputs from the previous step. However, each step within this pipeline is independent from each other and it is not optimized for the final goal. Consider the following example of image recognition:

$$\mathbf{f}_{SVM} \circ \mathbf{f}_{PCA} \circ \mathbf{f}_{SIFT} \circ \mathbf{f}_{hist\_eq}(I) \rightarrow \mathbf{y}.$$

The image I is first preprocessed with histogram equalization, next SIFT descriptors are extracted from the normalized image, then PCA is performed to reduce the dimensionality of the SIFT

features before finally fed into a SVM classifier. Histogram equalization tries to increase the global contrast of an input image by effectively spreading out the most frequent intensity values. SIFT, a carefully engineered descriptor, is designed for finding and matching interest points, and later often used for object detection and recognition. PCA is a popular algorithm for reducing the dimensionality of the data by finding a linear transformation of the data that preserves its variability as much as possible. It is arguable whether any of these objectives has any correlation with the ultimate goal (*e.g.*, classification error). A better approach would be treating each step as a parametric function with parameters to be learned. The parameters should be learned by directly minimizing the loss on your ultimate task. Convolutional Neural Network (CNN) [51, 53, 86, 87] is an successful example of the above idea. Not surprisingly, it outperformed the traditional pipeline by a large margin.

Sequential prediction is preferable to inference on a single complex model. Instead of modeling your problem with a single function with millions of parameters (infinite in the case of nonparametric methods), function composition provides a powerful tool to represent the same complexity but with potentially less parameters. The number of possible combinations grows exponentially with the number of layers (iterations) and the choices of base functions. Future directions of research may include:

1. Analyze compositional behaviors of functions. For example, what is the representational power if only logistic function is used?
2. How we design complimentary base functions such that their compositions can be more powerful?
3. Should one prefer more stages or more powerful base functions?
4. There are only 115 different types of atom in the universe. Think each atom type as a different base function and all the matter surrounding us as functions can be modeled. How do we find those "atom" functions?

Appendix

Appendix A: Proofs

Theorem 1. If the function $h(x)$ satisfies the following two conditions:

1. $h(x)$ is monotonic at the minimum x_*,
2. $h(x)$ has Lipschitz constant K at x_*,

then there exists a generic DM.

Proof. Without loss of generality, we assume that $h(x)$ is monotonically increasing, and that $h(x_k) \neq h(x_*)$. Otherwise, the optimization has reached the minimum. x_* is an arbitrary point in the monotonic region and $h(x_*)$ is a known scalar. In the following, we use Δx_k to denote $x_* - x_k$ and Δh_k to denote $h(x_*) - h(x_k)$. We want to find a r such that

$$\frac{|\Delta x_k|}{|\Delta x_{k-1}|} < 1, \text{if } x_* \neq x_{k-1}. \tag{1}$$

We replace x_k with x_{k-1} using Eq. 2.3 and the left side of Eq. 1 becomes

$$\begin{aligned}\frac{|\Delta x_k|}{|\Delta x_{k-1}|} &= \frac{|\Delta x_{k-1} - r\Delta h_{k-1}|}{|\Delta x_{k-1}|} = \frac{|\Delta x_{k-1}(1 - r\frac{\Delta h_{k-1}}{\Delta x_{k-1}})|}{|\Delta x_{k-1}|} \\ &= \frac{|\Delta x_{k-1}||1 - r\frac{\Delta h_{k-1}}{\Delta x_{k-1}}|}{|\Delta x_{k-1}|} = \left|1 - r\frac{\Delta h_{k-1}}{\Delta x_{k-1}}\right| = \left|1 - r\frac{|\Delta h_{k-1}|}{|\Delta x_{k-1}|}\right|. \end{aligned} \tag{2}$$

The last step is derived from condition 1. Denoting $\frac{|\Delta h_{k-1}|}{|\Delta x_{k-1}|}$ as K_{k-1} and setting Eq. $2 < 1$ gives us

$$\begin{aligned} -1 < 1 - rK_{k-1} &< 1 \\ \Rightarrow 0 < \quad r \quad &< \frac{2}{K_{k-1}}. \end{aligned} \tag{3}$$

From condition 2, we know that $K^i_{k-1} \leq K$. Any $0 < r < \frac{2}{K}$ will satisfy the inequalities in Eq. 3, and therefore, there exists a generic DM. Similarly, we can show $0 > r > -\frac{2}{K}$ is a generic DM when $h(x)$ is a monotonically decreasing. □

Theorem 2. If the function $h(x)$ satisfies the following two conditions:

1. $h(x)$ is monotonic over the domain X,
2. $h(x)$ has Lipschitz constant K over X,

then there exists a generic DM over X.

Proof. The theorem can be derived following the same proof above. □

Theorem 3. If function $\mathbf{h}(\mathbf{x})$ satisfies the following two conditions:

1. $\mathbf{g}(\mathbf{x}) = \mathbf{R}\mathbf{h}(\mathbf{x})$ is a monotone operator at the minimum $\mathbf{x}_*$,
2. $\mathbf{h}(\mathbf{x})$ has Lipschitz constant K at $\mathbf{x}_*$,

then $\mathbf{R}$ is a generic DM.

Proof. To simplify the notation, we denote $\mathbf{x}_* - \mathbf{x}$ as $\Delta\mathbf{x}$, $\mathbf{h}(\mathbf{x}_*) - \mathbf{h}(\mathbf{x})$ as $\Delta\mathbf{h}$, and use $\|\mathbf{x}\|$ to represent the L2 norm. We want to show that there exists $\mathbf{R}$ such that

$$\frac{\|\mathbf{x}_* - \mathbf{x}_k\|}{\|\mathbf{x}_* - \mathbf{x}_{k-1}\|} < 1, \text{if } \mathbf{x}_* \neq \mathbf{x}_{k-1}. \tag{4}$$

We replace $\mathbf{x}_k$ with $\mathbf{x}_{k-1}$ using Eq. 6 and squaring the left side of Eq. 4 gives us

$$\begin{aligned}\frac{\|\Delta\mathbf{x}_k\|^2}{\|\Delta\mathbf{x}_{k-1}\|^2} &= \frac{\|\Delta\mathbf{x}_{k-1} - \mathbf{R}\Delta\mathbf{h}_{k-1}\|^2}{\|\Delta\mathbf{x}_{k-1}\|^2} = \frac{\|\Delta\mathbf{x}_{k-1}\|^2}{\|\Delta\mathbf{x}_{k-1}\|^2} + \frac{\|\mathbf{R}\Delta\mathbf{h}_{k-1}\|^2}{\|\Delta\mathbf{x}_{k-1}\|^2} - 2\frac{\Delta\mathbf{x}_{k-1}^\top \mathbf{R}\Delta\mathbf{h}_{k-1}}{\|\Delta\mathbf{x}_{k-1}\|^2} \\ &= 1 + \frac{\|\mathbf{R}\Delta\mathbf{h}_{k-1}\|}{\|\Delta\mathbf{x}_{k-1}\|^2}\left(\|\mathbf{R}\Delta\mathbf{h}_{k-1}\| - 2\Delta\mathbf{x}_{k-1}^{i\top}\frac{\mathbf{R}\Delta\mathbf{h}_{k-1}}{\|\mathbf{R}\Delta\mathbf{h}_{k-1}\|}\right).\end{aligned} \tag{5}$$

Setting Eq. 5 < 1 gives us,

$$\|\mathbf{R}\Delta\mathbf{h}_{k-1}\| \leq 2\Delta\mathbf{x}_{k-1}^\top \frac{\mathbf{R}\Delta\mathbf{h}_{k-1}}{\|\mathbf{R}\Delta\mathbf{h}_{k-1}\|} \tag{6}$$

Condition 1 ensures that $\Delta\mathbf{x}_{k-1}^\top \mathbf{R}\Delta\mathbf{h}_{k-1} > 0$. From the geometric definition of dot product, we can rewrite the right side of the inequality 6 as,

$$2\Delta\mathbf{x}_{k-1}^\top \frac{\mathbf{R}\Delta\mathbf{h}_{k-1}}{\|\mathbf{R}\Delta\mathbf{h}_{k-1}\|} = 2\|\Delta\mathbf{x}_{k-1}\|\cos\theta,$$

where θ is the angle between vectors $\Delta\mathbf{x}_{k-1}$ and $\mathbf{R}\Delta\mathbf{h}_{k-1}$. Using condition 2 we have

$$2\|\Delta\mathbf{x}_{k-1}\|\cos\theta \geq \frac{2}{K}\|\Delta\mathbf{h}_{k-1}\|\cos\theta \tag{7}$$

From the Cauchy-Schwartz inequality,

$$\|\mathbf{R}\Delta\mathbf{h}_{k-1}\| \leq \|\mathbf{R}\|_F\|\Delta\mathbf{h}_{k-1}\|. \tag{8}$$

Given the inequalities in Eqs. 7 and 8, the condition that makes Eq. 6 hold is,

$$\|\mathbf{R}\|_F \leq \frac{2}{K}\cos\theta.$$

Any $\mathbf{R}$ whose $\|\mathbf{R}\|_F < \frac{2}{K}\min\cos\theta$ gaurantees the inequality stated in Eq. 4. Therefore, there exists a generic DM. □

Theorem 4. If function $\mathbf{h}(\mathbf{x})$ satisfies the following two conditions:

1. $\mathbf{g}(\mathbf{x}) = \mathbf{R}\mathbf{h}(\mathbf{x})$ is a monotone operator over a domain X,
2. $\mathbf{h}(\mathbf{x})$ has Lipschitz constant K over X,

then $\mathbf{R}$ is a generic DM over X.

Proof. The theorem can be derived following the same proof above. □

Theorem 5. If $\mathbf{h}(\mathbf{x})$ has Lipschitz constant K at the minimum $\mathbf{x}_*$, there exists a finite partition of domain $\mathbf{x}$, $S = \{S^t\}_1^T$, such that $\forall\mathbf{x} \in S^t$, there exists a generic DM $\mathbf{R}^t$.

Proof. To simplify the notation, we denote $\mathbf{x}_* - \mathbf{x}$ as $\Delta\mathbf{x}$ and $\mathbf{h}(\mathbf{x}_*) - \mathbf{h}(\mathbf{x})$ as $\Delta\mathbf{h}$. We will prove the above theorem by finding a specific partition with its corresponding DMs. Let us consider a partition strategy based on the signs of $\Delta x_j \Delta h_j$. Each sign can take on two values ± 1 and j ranges from 1 to $\min(n, m)$. Each subset of this partition contains $\mathbf{x}$ that satisfy one of the $2^{\min(n,m)}$ unique conditions. Without loss of generality, let us derive the DM for the subset S^0 where $\forall j, sign(\Delta x_j \Delta h_j) = 1$. We want to show that there exists a $\mathbf{R}$ such that

$$\frac{\|\mathbf{x}_* - \mathbf{x}_k\|}{\|\mathbf{x}_* - \mathbf{x}_{k-1}\|} < 1, \text{if } \mathbf{x}_* \neq \mathbf{x}_{k-1}. \tag{9}$$

We replace $\mathbf{x}_k$ with $\mathbf{x}_{k-1}$ using Eq. 2.5 and squaring the left side of Eq. 9 gives us,

$$\begin{aligned}\frac{\|\Delta\mathbf{x}_k\|^2}{\|\Delta\mathbf{x}_{k-1}\|^2} =& \frac{\|\Delta\mathbf{x}_{k-1} - \mathbf{R}\Delta\mathbf{h}_{k-1}\|^2}{\|\Delta\mathbf{x}_{k-1}\|^2} = \frac{\|\Delta\mathbf{x}_{k-1}\|^2}{\|\Delta\mathbf{x}_{k-1}\|^2} + \frac{\|\mathbf{R}\Delta\mathbf{h}_{k-1}\|^2}{\|\Delta\mathbf{x}_{k-1}\|^2} - 2\frac{\Delta\mathbf{x}_{k-1}^{i\top}\mathbf{R}\Delta\mathbf{h}_{k-1}}{\|\Delta\mathbf{x}_{k-1}\|^2} \\ =& 1 + \frac{\|\mathbf{R}\Delta\mathbf{h}_{k-1}\|}{\|\Delta\mathbf{x}_{k-1}\|^2}\left(\|\mathbf{R}\Delta\mathbf{h}_{k-1}\| - 2\Delta\mathbf{x}_{k-1}^{i\top}\frac{\mathbf{R}\Delta\mathbf{h}_{k-1}}{\|\mathbf{R}\Delta\mathbf{h}_{k-1}\|}\right).\end{aligned} \tag{10}$$

Setting Eq. $10 < 1$ gives us,

$$\|\mathbf{R}\Delta\mathbf{h}_{k-1}\| \leq 2\Delta\mathbf{x}_{k-1}^{i\top}\frac{\mathbf{R}\Delta\mathbf{h}_{k-1}}{\|\mathbf{R}\Delta\mathbf{h}_{k-1}\|}. \tag{11}$$

The choice of $\mathbf{R}$ needs to guarantee that the right side of Eq. 11 is greater than zero. Remember that in subset $S^{(0)}$ $sign(\Delta x_j \Delta h_j) = 1, \forall j$. A trivial $\mathbf{R}$ would be $c\mathbf{D}$, where $c > 0$ and $\mathbf{D}$ is a rectangular diagonal matrix with the diagonal elements equal to 1. From the geometric definition of dot product, we can rewrite the right side of the inequality 11 as,

$$2\Delta\mathbf{x}_{k-1}^{i\top}\frac{\mathbf{R}\Delta\mathbf{h}_{k-1}}{\|\mathbf{R}\Delta\mathbf{h}_{k-1}\|} = 2\|\Delta\mathbf{x}_{k-1}\|\cos\theta^i,$$

where θ^i is the angle between vectors $\Delta\mathbf{x}_{k-1}$ and $\mathbf{R}\Delta\mathbf{h}_{k-1}$. Using the condition that $\mathbf{h}(\mathbf{x})$ has a Lipschitz constant K at $\mathbf{x}_*$,we have

$$2\|\Delta\mathbf{x}_{k-1}\|\cos\theta^i \geq \frac{2}{K}\|\Delta\mathbf{h}_{k-1}\|\cos\theta^i. \tag{12}$$

From the Cauchy-Schwartz inequality,

$$\|\mathbf{R}\Delta\mathbf{h}_{k-1}\| \leq \|\mathbf{R}\|_F\|\Delta\mathbf{h}_{k-1}\|. \tag{13}$$

Given the inequalities in Eqs. 12 and 13, the condition that makes Eq. 11 hold is,

$$\|\mathbf{R}\|_F = \sqrt{c}\|\mathbf{D}\|_F = \sqrt{c} \leq \frac{2}{K}\cos\theta^i. \tag{14}$$

Any $\mathbf{R} = c\mathbf{D}$ where $\sqrt{c} < \frac{2}{K}\min_i \cos\theta^i$ guarantees the inequality stated in Eq. 9. Therefore, there exists a generic DM for subset $S^{(0)}$. For other subsets in the partition a general choice of $\mathbf{D}$ has the entries

$$d_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ sign(\Delta x^i_{j,k-1}\Delta h^i_{j,k-1}) & \text{Otherwise.} \end{cases}$$

Following the same proof we can easily show DM exist for other subsets in the partition. □

Appendix B: Derivatives on Analytic Functions

$f(x)$	f'	f''
$\frac{1}{2}(\sin(x)-y)^2$	$(\sin(x)-y)\cos(x)$	$\cos^2(x)-\sin^2(x)+\sin(x)y$
$\frac{1}{2}(x^3-y)^2$	$(x^3-y)3x^2$	$15x^4-6xy$
$\frac{1}{2}(e^x-y)^2$	$(e^x-y)e^x$	$2e^{2x}-e^xy$
$\frac{1}{2}(x^{-1}-y)^2$	$-(x^{-1}-y)x^{-2}$	$3x^{-4}-2x^{-3}y$

$$
\begin{aligned}
\mathbf{h}(\mathbf{x}) &= \begin{bmatrix} 2x_1x_2 \\ 5x_1 - x_2^2 \end{bmatrix} \\
f(\mathbf{x}) &= \tfrac{1}{2}\left\| \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \right\|^2 \\
\mathbf{J}_f(\mathbf{x}) &= \begin{bmatrix} 2x_2(h_1-y_1)+5(h_2-y_2) \\ 2x_1(h_1-y_1)-2x_2(h_2-y_2) \end{bmatrix}^\top \\
\mathbf{H}_f(\mathbf{x}) &= \begin{bmatrix} 4x_2^2+25 & 8x_1x_2-2y_1-10x_2 \\ 8x_1x_2-2y_1-10x_2 & 4x_1^2+6x_2^2-10x_1+2y_2 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{h}(\mathbf{x}) &= \begin{bmatrix} \cos(x_1)x_2 \\ \sin(x_1)x_2 \end{bmatrix} \\
f(\mathbf{x}) &= \tfrac{1}{2}\left\| \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \right\|^2 \\
\mathbf{J}_f(\mathbf{x}) &= \begin{bmatrix} -h_1x_2\sin(x_1)+h_2x_2\cos(x_1) \\ h_1\cos(x_1)+h_2\sin(x_1) \end{bmatrix}^\top \\
\mathbf{H}_f(\mathbf{x}) &= \begin{bmatrix} x_2y_1\cos(x_1)+x_2y_2\sin(x_1) & y_1\sin(x_1)-y_2\cos(x_1) \\ y_1\sin(x_1)-y_2\cos(x_1) & 1 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{h}(\mathbf{x}) &= \begin{bmatrix} e^{x_1}+x_2^2 \\ x_1^2x_2 \end{bmatrix} \\
f(\mathbf{x}) &= \tfrac{1}{2}\left\| \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \right\|^2 \\
\mathbf{J}_f(\mathbf{x}) &= \begin{bmatrix} e^{x_1}(h_1-y_1)+2x_1x_2(h_2-y_2) \\ 2x_2(h_1-y_1)+x_1^2(h_2-y_2) \end{bmatrix}^\top \\
\mathbf{H}_f(\mathbf{x}) &= \begin{bmatrix} e^{x_1}(2e^{x_1}+x_2^2-y_1)+6x_1^2x_2^2-2x_2y_2 & 2e^{x_1}x_2+2x_1(2x_1^2x_2-y_2) \\ 2e^{x_1}x_2+2x_1(2x_1^2x_2-y_2) & 2e^{x_1}+6x_2^2-2y_1+x_1^4 \end{bmatrix}
\end{aligned}
$$

Figure 1: The first and second derivatives of analytic functions used in experiments 4.1.

Appendix C: Algorithms

Algorithm 1: SDM training

input : $maxIter, \mathbf{x}_*, \mathbf{x}_0, \lambda$
output: $\{r_k\}$

$k \leftarrow 1$;
$\mathbf{y}_* \leftarrow h(\mathbf{x}_*)$;
while $k < maxIter$ **do**
  $\Delta\mathbf{x}_* \leftarrow \mathbf{x}_* - \mathbf{x}_k$;
  $\Delta\mathbf{y}_* \leftarrow \mathbf{y}_* - h(\mathbf{x}_k)$;
  $c \leftarrow \Delta\mathbf{y}_* \cdot \Delta\mathbf{y}_*$;
  $\lambda' \leftarrow c\lambda$;
  $r_k \leftarrow \frac{\Delta\mathbf{y}_* \cdot \Delta\mathbf{x}_*}{c+\lambda'}$;
  $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + r_k\Delta\mathbf{y}_*$;
  $k \leftarrow k + 1$;
end

Algorithm 2: SDM testing

input : $maxIter, \mathbf{y}_*, \mathbf{x}_0$
output: $\mathbf{x}_k$

$k \leftarrow 1$;
while $k < maxIter$ **do**
  $\Delta\mathbf{x} \leftarrow \mathbf{x}_k - \mathbf{x}_{k-1}$;
  $\Delta\mathbf{y}_* \leftarrow \mathbf{y}_* - h(\mathbf{x}_k)$;
  $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + r_k\Delta\mathbf{y}_*$;
  $k \leftarrow k + 1$;
end

Figure 2: Training and testing algorithms of SDM for minimizing 1D analytic functions.

Algorithm 3: GSDM training

input : $maxIter, \mathbf{x}_*, \mathbf{x}_0, \lambda$
output: $\{r_k\}$

$k \leftarrow 1$;
$\mathbf{y}_* \leftarrow h(\mathbf{x}_*)$;
while $k < maxIter$ **do**
  $\Delta\mathbf{x} \leftarrow \mathbf{x}_k - \mathbf{x}_{k-1}$;
  $\Delta\mathbf{x}_* \leftarrow \mathbf{x}_* - \mathbf{x}_k$;
  $\Delta\mathbf{y}_* \leftarrow \mathbf{y}_* - h(\mathbf{x}_k)$;
  $[S_1, S_2] \leftarrow \texttt{dhd\_1d}(\Delta\mathbf{x}, \Delta\mathbf{y}_*)$;
  for $i \leftarrow 1$ **to** 2 **do**
    $c \leftarrow \Delta\mathbf{y}_*^{S_i} \cdot \Delta\mathbf{y}_*^{S_i}$;
    $\lambda' \leftarrow c\lambda$;
    $r_k^i \leftarrow \frac{\Delta\mathbf{y}_*^{S_i} \cdot \Delta\mathbf{x}_*^{S_i}}{c+\lambda'}$;
    $\mathbf{x}_k^{S_i} \leftarrow \mathbf{x}_{k-1}^{S_i} + r_k^i\Delta\mathbf{y}_*^{S_i}$;
  end
  $k \leftarrow k + 1$;
end

Algorithm 4: GSDM testing

input : $maxIter, \mathbf{y}_*, \mathbf{x}_0$
output: $\mathbf{x}_k$

$k \leftarrow 1$;
while $k < maxIter$ **do**
  $\Delta\mathbf{x} \leftarrow \mathbf{x}_k - \mathbf{x}_{k-1}$;
  $\Delta\mathbf{y}_* \leftarrow \mathbf{y}_* - h(\mathbf{x}_k)$;
  $[S_1, S_2] \leftarrow \texttt{dhd\_1d}(\Delta\mathbf{x}, \Delta\mathbf{y}_*)$;
  for $i \leftarrow 1$ **to** 2 **do**
    $\mathbf{x}_k^{S_i} \leftarrow \mathbf{x}_{k-1}^{S_i} + r_k^i\Delta\mathbf{y}_*^{S_i}$;
  end
  $k \leftarrow k + 1$;
end

Algorithm 5: function `dhd_2d`

input : $\Delta\mathbf{y}_*, \Delta\mathbf{x}$
output: $\{S_1, S_2\}$

$S_1 \leftarrow \texttt{find}(\Delta\mathbf{x} \circ \Delta\mathbf{y}_* > 0)$;
$S_2 \leftarrow \texttt{find}(\Delta\mathbf{x} \circ \Delta\mathbf{y}_* \leq 0)$;

Figure 3: Training and testing algorithms of GSDM for minimizing 1D analytic functions.

Algorithm 6: GSDM training

input : $maxIter, \mathbf{X}_*, \mathbf{X}_0, \lambda$
output: $\{\mathbf{R}_k\}$

$k \leftarrow 1$;
$\mathbf{Y}_* \leftarrow \mathbf{h}(\mathbf{X}_*)$;
while $k < maxIter$ **do**
  $\Delta\mathbf{X} \leftarrow \mathbf{X}_k - \mathbf{X}_{k-1}$;
  $\Delta\mathbf{X}_* \leftarrow \mathbf{X}_* - \mathbf{X}_k$;
  $\Delta\mathbf{Y}_* \leftarrow \mathbf{Y}_* - \mathbf{h}(\mathbf{X}_k)$;
  $\{S_i\}_{i=1}^4 \leftarrow \texttt{dhd\_2d}(\Delta\mathbf{X}, \Delta\mathbf{Y}_*)$;
  for $i \leftarrow 1$ **to** 4 **do**
    $\Sigma \leftarrow \Delta\mathbf{Y}_*^{S_i} {\Delta\mathbf{Y}_*^{S_i}}^\top$
    $c \leftarrow \text{trace}(\Sigma)$;
    $\lambda' \leftarrow c\lambda$;
    $r_k^i \leftarrow \Delta\mathbf{Y}_*^{S_i} {\Delta\mathbf{X}_*^{S_i}}^\top (\Sigma + \lambda'\mathbf{I})^{-1}$;
    $\mathbf{X}_k^{S_i} \leftarrow \mathbf{X}_{k-1}^{S_i} + \mathbf{R}_k^i \Delta\mathbf{Y}_*^{S_i}$;
  end
  $k \leftarrow k + 1$;
end

Algorithm 7: GSDM testing

input : $maxIter, \mathbf{Y}_*, \mathbf{X}_0$
output: $\mathbf{x}_k$

$k \leftarrow 1$;
while $k < maxIter$ **do**
  $\Delta\mathbf{X} \leftarrow \mathbf{X}_k - \mathbf{X}_{k-1}$;
  $\Delta\mathbf{Y}_* \leftarrow \mathbf{Y}_* - \mathbf{h}(\mathbf{X}_k)$;
  $\{S_i\}_{i=1}^4 \leftarrow \texttt{dhd\_2d}(\Delta\mathbf{X}, \Delta\mathbf{Y}_*)$;
  for $i \leftarrow 1$ **to** 4 **do**
    $\mathbf{X}_k^{S_i} \leftarrow \mathbf{X}_{k-1}^{S_i} + \mathbf{R}_k^i \Delta\mathbf{Y}_*^{S_i}$;
  end
  $k \leftarrow k + 1$;
end

Algorithm 8: `dhd_2d` function.

input : $\Delta\mathbf{Y}_*, \Delta\mathbf{X}$
output: $\{S_i\}_{i=1}^4$

for $i \leftarrow 1$ **to** 2 **do**
  $\mathbf{w}_i \leftarrow \Delta\mathbf{x}_i \circ \Delta\mathbf{y}_{*i}$;
end
$S_1 \leftarrow \texttt{find}(\mathbf{w}_1 < 0 \wedge \mathbf{w}_2 < 0)$;
$S_2 \leftarrow \texttt{find}(\mathbf{w}_1 < 0 \wedge \mathbf{w}_2 \geq 0)$;
$S_3 \leftarrow \texttt{find}(\mathbf{w}_1 \geq 0 \wedge \mathbf{w}_2 < 0)$;
$S_4 \leftarrow \texttt{find}(\mathbf{w}_1 \geq 0 \wedge \mathbf{w}_2 \geq 0)$;

Figure 4: Training and testing algorithms of GSDM for minimizing 2D analytic functions. The same algorithms are also used in Section 4.2 for Inverse Kinematics.

Bibliography

[1] Karim T. Abou-Moustafa, Fernando De la Torre, and Frank P. Ferrie. Pareto discriminant analysis. In *CVPR*, 2010. 1.1

[2] Relja Arandjelović and Andrew Zisserman. Three things everyone should know to improve object retrieval. In *Computer Vision and Pattern Recognition (CVPR)*, pages 2911–2918, 2012. 3.2.4

[3] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009. (document), 2.2.6, 2.6

[4] Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific J. Math.*, 16(1):1–3, 1966. 2.1.1

[5] Larry Armijo et al. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966. 4.1

[6] S. Baker and I. Matthews. Lucas-Kanade 20 years on: A unifying framework. *IJCV*, 56 (3):221 – 255, March 2004. 2, 3.2.1, 3.2.1, 5.1.1

[7] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *IJCV*, 56:221–255, 2004. 4.3

[8] Simon Baker, Ralph Gross, Iain Matthews, and Takahiro Ishikawa. Lucas-kanade 20 years on: A unifying framework: Part 2. Technical Report CMU-RI-TR-03-01, Robotics Institute, 2003. 3.2.1

[9] Tadas Baltrusaitis, Peter Robinson, and L Morency. 3d constrained local model for rigid and non-rigid facial tracking. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2610–2617. IEEE, 2012. 3.3.1

[10] P. N. Belhumeur, D. W. Jacobs, D. J. Kriegman, and N. Kumar. Localizing parts of faces using a consensus of exemplars. In *CVPR*, 2011. 3.2.1, 4.4, 4.4, 4.5

[11] E. K. Berndt, B. H. Hall, R. E. Hall, and Jerry A. Hausman. Estimation and inference in nonlinear structural models. *Annals of Economic and Social Measurement*, 3(4):653–665, 1974. 1.1

[12] M. J. Black and A. D. Jepson. Eigentracking: Robust matching and tracking of objects using view-based representation. *IJCV*, 26(1):63–84, 1998. 1.1, 3.2.1, 3.2.1

[13] Andrew Blake, Carsten Rother, Matthew Brown, Patrick Perez, and Philip Torr. Interactive image segmentation using an adaptive gmmrf model. In *ECCV*, pages 428–441, 2004.

2.2.5

[14] V. Blanz and T. Vetter. A morphable model for the synthesis of 3D faces. In *SIGGRAPH*, 1999. 3.2.1

[15] Stephen Boyd. Monotone operators. 2014. URL `http://stanford.edu/class/ee364b/lectures/monotone_slides.pdf`. 2.1.1, 2.1.1

[16] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 4.4, 4.5

[17] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996. 3.2.4

[18] C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19(92):577–593, 1965. 1.1

[19] A.M. Buchanan and A. W. Fitzgibbon. Damped newton algorithms for matrix factorization with missing data. In *CVPR*, 2005. 1.1

[20] R. H. Byrd, P. Lu, and J. Nocedal. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing*, 16(5):1190–1208, 1995. 1.1

[21] Richard H Byrd, Jean Charles Gilbert, and Jorge Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89 (1):149–185, 2000. 4.2.1

[22] Chen Cao, Qiming Hou, and Kun Zhou. Displaced dynamic expression regression for real-time facial tracking and animation. *ACM Transactions on Graphics (TOG)*, 33(4):43, 2014. 3.3.1

[23] X. Cao, Y. Wei, F. Wen, and J. Sun. Face alignment by explicit shape regression. In *CVPR*, 2012. 3.3.1, 4.4

[24] William W. Cohen. Stacked sequential learning. In *International Joint Conference on Artificial Intelligence*, pages 671–676, 2005. 2.2.5

[25] T. F. Cootes, M. C. Ionita, C. Lindner, and P. Sauer. Robust and accurate shape model fitting using random forest regression voting. In *ECCV*, 2012. 3.2.1

[26] T.F. Cootes, G.J. Edwards, and C.J. Taylor. Active appearance models. *TPAMI*, 23(6): 681–685, 2001. 3.2.1, 3.2.1, 4.4

[27] Timothy F Cootes and Christopher J Taylor. A mixture model for representing shape variation. *Image and Vision Computing*, 17(8):567–573, 1999. 3.3.1

[28] Timothy F Cootes, Christopher J Taylor, David H Cooper, and Jim Graham. Active shape models-their training and application. *Computer vision and image understanding*, 61(1): 38–59, 1995. 3.3.1

[29] Timothy F Cootes, Gavin V Wheeler, Kevin N Walker, and Christopher J Taylor. View-based active appearance models. *Image and vision computing*, 20(9):657–664, 2002. 3.3.1

[30] D. Cristinacce and T. Cootes. Automatic feature localisation with constrained local models. *Journal of Pattern Recognition*, 41(10):3054–3067, 2008. 3.2.1, 3.3.1, 4.4

[31] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, volume 1,

pages 886–893, 2005. 1.1, 3.1

[32] F. De la Torre and Minh Hoai Nguyen. Parameterized kernel principal component analysis: Theory and applications to supervised and unsupervised image alignment. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2008. 1.1, 3.2.1, 3.2.1

[33] Daniel F. DeMenthon and Larry S. Davis. Model-based object pose in 25 lines of code. *IJCV*, 15:123–141, 1995. 3.4.1, 4.6

[34] P. Dollár, P. Welinder, and P. Perona. Cascaded pose regression. In *CVPR*, 2010. 2.2.3, 3.2.1

[35] Ian L. Dryden and Kanti V. Mardia. *Statistical shape analysis*. Wiley, Chichester, 1998. 3.2.1

[36] Liam Ellis, Nicholas Dowson, Jiri Matas, and Richard Bowden. Linear regression and adaptive appearance models for fast simultaneous modelling and tracking. *International journal of computer vision*, 95(2):154–179, 2011. 3.3.1

[37] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 3.4.1

[38] J. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001. 2.2.2, 3.2.1

[39] Steffen Gauglitz, Tobias Höllerer, and Matthew Turk. Evaluation of interest point detectors and feature descriptors for visual tracking. *IJCV*, 94(3):335–360, 2011. doi: 10.1007/s11263-011-0431-5. 4.3

[40] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker. Multi-pie. In *AFGR*, 2007. 4.5

[41] Lie Gu and Takeo Kanade. 3d alignment of face in a single image. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 1305–1312. IEEE, 2006. 3.3.1

[42] M. H. Hayes. *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons, 1996. 3.2.3

[43] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. 3.2.4

[44] Changbo Hu, Rogerio Feris, and Matthew Turk. Real-time view-based face alignment using active wavelet networks. In *Analysis and Modeling of Faces and Gestures, 2003. AMFG 2003. IEEE International Workshop on*, pages 215–221. IEEE, 2003. 3.3.1

[45] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007. 4.5

[46] Yuchi Huang, Qingshan Liu, and Dimitris N. Metaxas. A component-based framework for generalized face alignment. *IEEE Transactions on Systems, Man, and Cybernetics*, 41 (1):287–298, 2011. 3.2.1

[47] M. J. Jones and T. Poggio. Multidimensional morphable models. In *ICCV*, 1998. 3.2.1

[48] Jon Kleinberg and Eva Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. *Journal of the ACM (JACM)*, 49(5):616–639, 2002. 2.2.5

[49] Martin Kostinger, Paul Wohlhart, Peter M Roth, and Horst Bischof. Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 2144–2151. IEEE, 2011. 4.5

[50] Zhenzhen Kou and William W Cohen. Stacked graphical models for efficient inference in markov random fields. In *SDM*, pages 533–538. SIAM, 2007. 2.2.5

[51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 2.2.7, 2.2.7, 5.2.2

[52] Vuong Le, Jonathan Brandt, Zhe Lin, Lubomir Bourdev, and Thomas S Huang. Interactive facial feature localization. In *Computer Vision–ECCV 2012*, pages 679–692. Springer, 2012. 4.5

[53] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3.2.4, 5.2.2

[54] Stan Z Li, HongJiang Zhang, Qiansheng Cheng, et al. Multi-view face alignment using direct appearance models. In *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, pages 324–329. IEEE, 2002. 3.3.1

[55] David Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2): 91–110, 2004. 1.1

[56] David G. Lowe. Fitting parameterized three-dimensional models to images. *PAMI*, 13: 441–450, 1991. 3.4.1

[57] C-P Lu, Gregory D Hager, and Eric Mjolsness. Fast and globally convergent pose estimation from video images. *Pattern Analysis and Machine Intelligence*, 22(6):610–622, 2000. 3.4.1

[58] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the Internaional Joint Conference on Artificial Intelligence*, 1981. 2, 3.1, 5.1.1

[59] Bruce Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of Imaging Understanding Workshop*, 1981. 1.1, 3.2.1

[60] Iain Matthews, Jing Xiao, and Simon Baker. 2d vs. 3d deformable face models: Representational power, construction, and real-time fitting. *International journal of computer vision*, 75(1):93–113, 2007. 3.3.1

[61] Jorge J. Moré. The Levenberg-Marquardt algorithm: Implementation and theory. In G. A. Watson, editor, *Numerical Analysis*, pages 105–116. Springer, Berlin, 1977. 1.1

[62] Jorge J Moré and David J Thuente. Line search algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software (TOMS)*, 20(3):286–307, 1994. 4.1

[63] Francesc Moreno-Noguer, Vincent Lepetit, and Pascal Fua. Accurate non-iterative O(n) solution to the pnp problem. In *ICCV*, pages 1–8, 2007. 3.4.1

[64] Daniel Munoz, J Andrew Bagnell, and Martial Hebert. Stacked hierarchical labeling. In *Computer Vision–ECCV 2010*, pages 57–70. 2010. 2.2.5

[65] Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, and Caroline Suen. UFLDL tutorial. URL `http://deeplearning.stanford.edu/wiki/index.php/Neural_Networks`. (document), 2.7

[66] Mustafa Özuysal, Michael Calonder, Vincent Lepetit, and Pascal Fua. Fast keypoint recognition using random ferns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(3):448–461, 2010. 2.2.3

[67] Robert Pless and Richard Souvenir. A survey of manifold learning for images. *IPSJ Transactions on Computer Vision and Applications*, 1:83–94, 2009. 3.3.2

[68] Chengchao Qu, Hua Gao, Eduardo Monari, Jürgen Beyerer, and Jean-Philippe Thiran. Towards robust cascaded regression for face alignment in the wild. 2015. 3.2.4

[69] Long Quan and Zhongdan Lan. Linear n-point camera pose determination. *Pattern Analysis and Machine Intelligence*, 21(8):774–780, 1999. 3.4.1

[70] Shaoqing Ren, Xudong Cao, Yichen Wei, and Jian Sun. Face alignment at 3000 fps via regressing local binary features. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1685–1692, 2014. 3.2.4

[71] S. Rivera and A. M. Martinez. Learning deformable shape manifolds. *Pattern Recognition*, 45(4):1792–1801, 2012. 3.2.1

[72] Lawrence G. Roberts. *Machine Perception of Three-Dimensional Solids*. Outstanding Dissertations in the Computer Sciences. Garland Publishing, New York, 1963. ISBN 0-8240-4427-4. 3.4.1

[73] R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(5):877–898, 1976. 2.1.1

[74] Sami Romdhani, Shaogang Gong, Ahaogang Psarrou, et al. A multi-view nonlinear active shape model using kernel pca. In *BMVC*, volume 10, pages 483–492, 1999. 3.3.1

[75] Christos Sagonas, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. 300 faces in-the-wild challenge: The first facial landmark localization challenge. In *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*, pages 397–403. IEEE, 2013. 4.5

[76] Christos Sagonas, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. A semi-automatic methodology for facial landmark annotation. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on*, pages 896–903. IEEE, 2013. 4.5

[77] E. Sanchez, F. De la Torre, and D. Gonzalez. Continuous regression for non-rigid image alignment. In *ECCV*, 2012. 3.2.1

[78] J. Saragih. Principal regression analysis. In *CVPR*, 2011. 3.2.1, 4.4, 4.4

[79] J. Saragih, S. Lucey, and J. Cohn. Face alignment through subspace constrained mean-shifts. In *ICCV*, 2009. 3.2.1

[80] Jason Saragih and Roland Goecke. A nonlinear discriminative approach to AAM fitting. In *ICCV*, 2007. 3.2.1

[81] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *Artificial Neural NetworksICANN'97*, pages 583–588. Springer, 1997. 3.3.1

[82] Xiaohui Shen, Zhe Lin, Jonathan Brandt, and Ying Wu. Detecting and aligning faces by image retrieval. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3460–3467. IEEE, 2013. 3.3.1

[83] Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986. 3.3.1

[84] Brandon M Smith, Jonathan Brandt, Zhe Lin, and Li Zhang. Nonparametric context modeling of local appearance for pose-and expression-robust facial landmark localization. In *CVPR*, 2014. 3.3.1

[85] Ivan E. Sutherland. Three-dimensional data input by tablet. *Proceedings of The IEEE*, 62 (4):453–461, 1974. 3.4.1

[86] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014. 5.2.2

[87] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lars Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1701–1708. IEEE, 2014. 5.2.2

[88] Yuandong Tian and Srinivasa G Narasimhan. Globally optimal estimation of nonrigid image distortion. *International journal of computer vision*, 98(3):279–302, 2012. 2.2.4

[89] Yuandong Tian and Srinivasa G Narasimhan. Theory and practice of hierarchical data-driven descent for optimal deformation estimation. *International Journal of Computer Vision*, pages 1–24, 2015. 2.2.4

[90] Transportation Research Board of the National Academies of Science. The 2nd strategic highway research program naturalistic driving study dataset. Available from the SHRP 2 NDS InSight Data Dissemination web site: `https://insight.shrp2nds.us/`, 2013. 4.5

[91] P. Tresadern, P. Sauer, and T. F. Cootes. Additive update predictors in active appearance models. In *BMVC*, 2010. 3.2.1

[92] G. Tzimiropoulos, S. Zafeiriou, and M. Pantic. Robust and efficient parametric face alignment. In *ICCV*, 2011. 3.2.1

[93] G. Tzimiropoulos, S. Zafeiriou, and M. Pantic. Subspace learning from image gradient orientations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12): 2454–2466, 2012. 3.2.1

[94] Richard A Waltz, José Luis Morales, Jorge Nocedal, and Dominique Orban. An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming*, 107(3):391–408, 2006. 4.2.1

[95] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992. 2.2.5

[96] Jing Xiao, Simon Baker, Iain Matthews, and Takeo Kanade. Real-time combined 2d+ 3d active appearance models. In *CVPR*, pages 535–542, 2004. 3.3.1

[97] Xuehan Xiong and Fernando De la Torre. Supervised descent method and its applications to face alignment. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 532–539. IEEE, 2013. 3.2.4, 3.3.1

[98] Xuehan Xiong and Fernando De la Torre. Supervised descent method for solving nonlinear least squares problems in computer vision. *arXiv preprint arXiv:1405.0601*, 2014. 3.2.4

[99] Xuehan Xiong, Daniel Munoz, J Andrew Bagnell, and Martial Hebert. 3-d scene analysis via sequenced predictions over points and regions. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2609–2616, 2011. 2.2.5

[100] Heng Yang, Xuhui Jia, Ioannis Patras, and Kwok-Ping Chan. Random subspace supervised descent method for regression problems in computer vision. *Signal processing letters*, 22(10):1816–1820, 2015. 3.2.4

[101] Heng Yang, Wenxuan Mou, Yichi Zhang, Ioannis Patras, Hatice Gunes, and Peter Robinson. Face alignment assisted by head pose estimation. *arXiv preprint arXiv:1507.03148*, 2015. 3.2.4

[102] Joseph S.-C. Yuan. A general photogrammetric method for determining object position and orientation. *IEEE Trans. on Robotics and Automation*, 5:129–142, 1989. 3.4.1

[103] Jie Zhang, Shiguang Shan, Meina Kan, and Xilin Chen. Coarse-to-fine auto-encoder networks (cfan) for real-time face alignment. In *Computer Vision–ECCV 2014*, pages 1–16. Springer, 2014. 3.2.4

[104] Lu Zhang, Jan Allebach, Qian Lin, and Xianwang Wang. Resolution-adaptive face alignment with head pose correction. In *IS&T/SPIE Electronic Imaging*, 2015. 3.2.4

[105] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *ECCV 2014*, pages 94–108, 2014. 5.2.1

[106] Yi Zhou, Wei Zhang, Xiaoou Tang, and Harry Shum. A bayesian mixture model for multi-view face alignment. In *CVPR*, 2005. 3.3.1

[107] Shizhan Zhu, Cheng Li, Chen Change Loy, and Xiaoou Tang. Transferring landmark annotations for cross-dataset face alignment. *arXiv preprint arXiv:1409.0602*, 2014. 3.2.4

[108] X. Zhu and D. Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *CVPR*, 2012. 3.2.1, 3.3.1

[109] Xiangyu Zhu, Junjie Yan, Dong Yi, Zhen Lei, and Stan Z Li. Discriminative 3d morphable model fitting. In *Automatic Face and Gesture Recognition (FG)*, pages 1–8, 2015. 3.2.4

[110] Karel Zimmermann, Jiri Matas, and Tomás Svoboda. Tracking by an optimal sequence of linear predictors. *TPAMI*, 31(4):677–692, 2009. 3.2.1