Convex Optimization

Hu SiXing, Hakki Can Karaimer, Pan An, Philipp Keck

National University of Singapore

January 27, 2016

Linear Regression Example

There should be a picture here.

Ordinary Least Squares

Input: points (x_i, y_i)

Regression line: $y = mx + b$

Objective:

$\min_{m,b} \blacksquare_i (y_i - mx_i - b)^2$

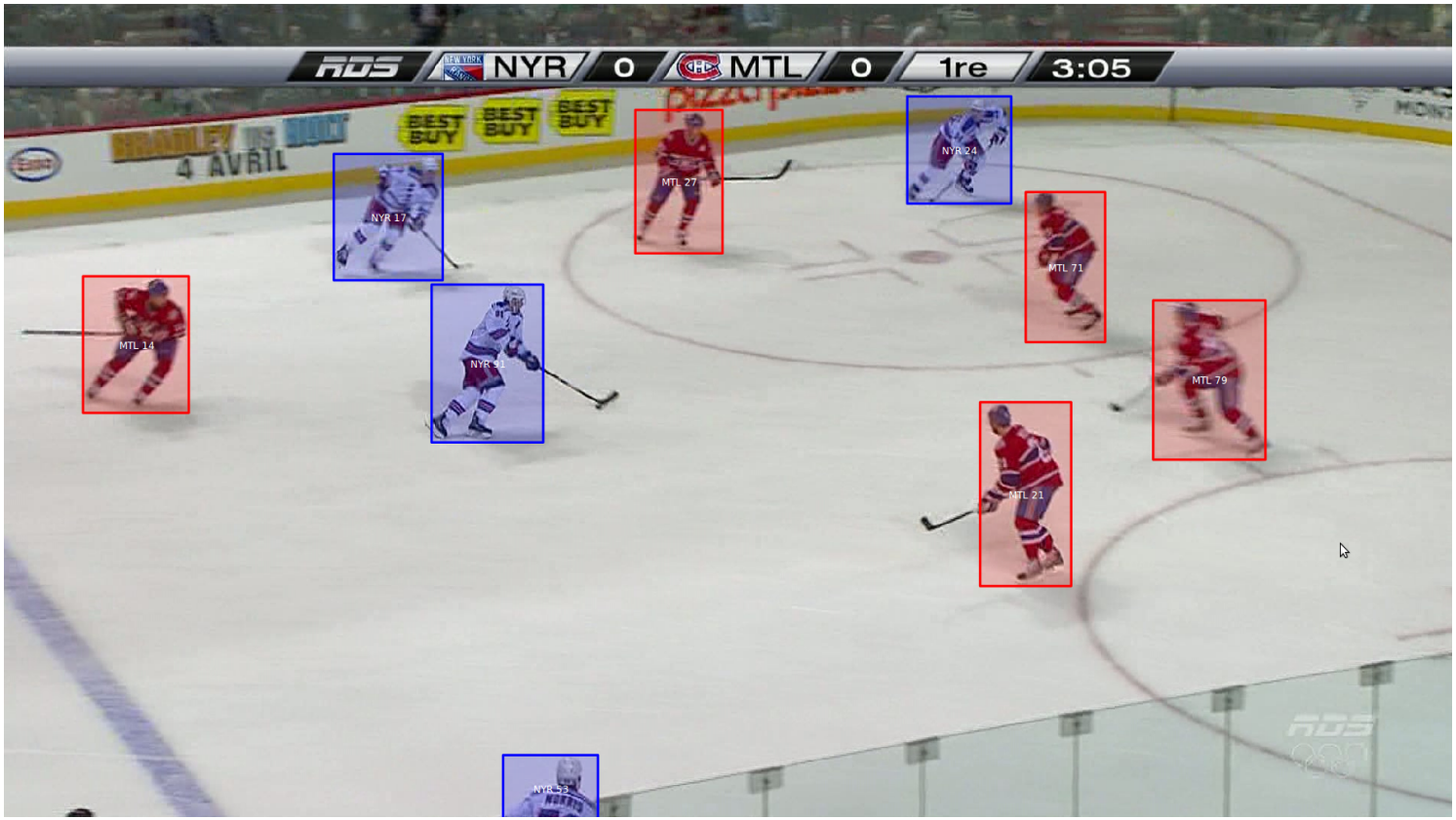
$(\vec{x}_i, y_i)$

$y = \vec{w} \cdot \vec{x} + b$

$\min_{\vec{w}} \blacksquare_i (y_i - \vec{w} \cdot \vec{x}_i - b)^2$

Image Processing — Lucas-Kanade

Classic examples are optical flow techniques like Lucas-Kanade (VideoTracking), Horn-Schunck.



Lucas-Kanade

Goal of Lucas-Kanade

Minimize the sum of squared error between two images.

Assumption

The displacement of the image contents between two nearby instants (frames) is small and approximately constant within a neighborhood of the point p under consideration.

Lucas-Kanade

Optical Flow Equation (2 Dementional)

For a pixel location (x, y, t), the intensity has moved by $\blacksquare x, \blacksquare y, \blacksquare t$, the basic assumption can be represented as:

$$I(x, y, t) = I(x + \blacksquare x, y + \blacksquare y, t + \blacksquare t)$$

Lucas-Kanade

Optical Flow Effect:

For all pixels within a window centered at p:

$$I_x(q_i)V_x + I_y(q_i)V_y = -I_t(q_i)$$

Where $i = 1, 2, 3...n$.

Abbreviations:

$$A = [I_x(q_i)^T, I_y(qi)^T]$$

$$V = [v_x, v_y]^T$$

$$b = [-I_t(q_i)]^T$$

Lucas-Kanade

Lucas-Kanade Method Abstraction:

LK method tries to solve 2×2 system:

$$A^T A V = A^T b$$

A.K.A:

$$V = (A^T A)^{-1} A^T b$$

Notice:

$V = [v_x, v_y]^T$ is variable. Which means that the system does not know the actual velocity of the system.

Lucas-Kanade

Goal of Lucas-Kanade Method:

To minimize $||A^T V - b||^2$.

Basic LK Derivation for Models(Stuff to be Tracked):

$$E[v_x, v_y] = \blacksquare[I(x + v_x, y + v_y) - T(x, y)]^2$$

Where v_x, v_y is the hypothesized location of the model(s) to be tracked, and $T(x, y)$ model.

Lucas-Kanade

Key Step for Implementation of GD (Step 1):

Generalizing LK approach by introducing warp function W:

$$E[v_x, v_y] = \blacksquare[I(W(x, y); P) - T(x, y)]^2$$

Generalizing is used to solve the problem where the constant flow of larger picture frames for a long time is a total waste of calculation power. Warp function examples are Affine and Projective.
The warping function are the convergence factor for steepest descent algorithm.

Lucas-Kanade

Key Step for Implementation of GD (Step 2):

The key to the derivation is Taylor series approximation:

$$I(W(x,y);P+\blacksquare P) \approx I(W([x,y];P)) + \nabla I \frac{\partial W}{\partial P} \blacksquare P$$

- The approximation equation is actually the abstract of the basic assumption of optical flow described in the slides before.
- Derivation of this equation can be discussed in forum (Too long for slides).

Lucas-Kanade

Some Explainations:

- Gradient image ∇I
- Image error $I_E = T(x, y) - I(W[x, y]; P)$
- Jacobian matrix $\frac{\partial W}{\partial P}$
- Steepest image $I_S = \nabla I \frac{\partial W}{\partial P}$
- Hessian Matrix $\blacksquare(\nabla I \frac{\partial W}{\partial P})^T (\nabla I \frac{\partial W}{\partial P})$
- Iteration step $\blacksquare P = \blacksquare I_S^T I_E$

Lucas-Kanade

Algorithms:

- Warp image and get $I(W[x, y]; P)$;
- Get image error I_E;
- Warp gradient image ∇I;
- Evaluate Jacobian;
- Compute steepest descent image $I_S = \nabla I \frac{\partial W}{\partial P}$;
- Compute Hessian matrix $\blacksquare I_S^T I_S$;
- Get warping step $\blacksquare P = I_S I_E$;
- Update warping parameter $P = P + \blacksquare P$;
- Repeat until $\blacksquare P$ is negligible.

APPLICATIONS – MACHINE LEARNING

Generalized Utilization of Convex: Delta Rule

- The delta rule is derived by attempting to minimize the error in the output of the neural network through gradient descent.
- Gradient Descent optimization is the most basic principle for training neurons even with different activation functions.
- Delta rule, can also be modified, if possible, with steepest descent method.

APPLICATIONS – MACHINE LEARNING

Delta Rule:

$$\blacksquare w_{ji} = \alpha(t_j - y_j)g'(h_j)x_j$$

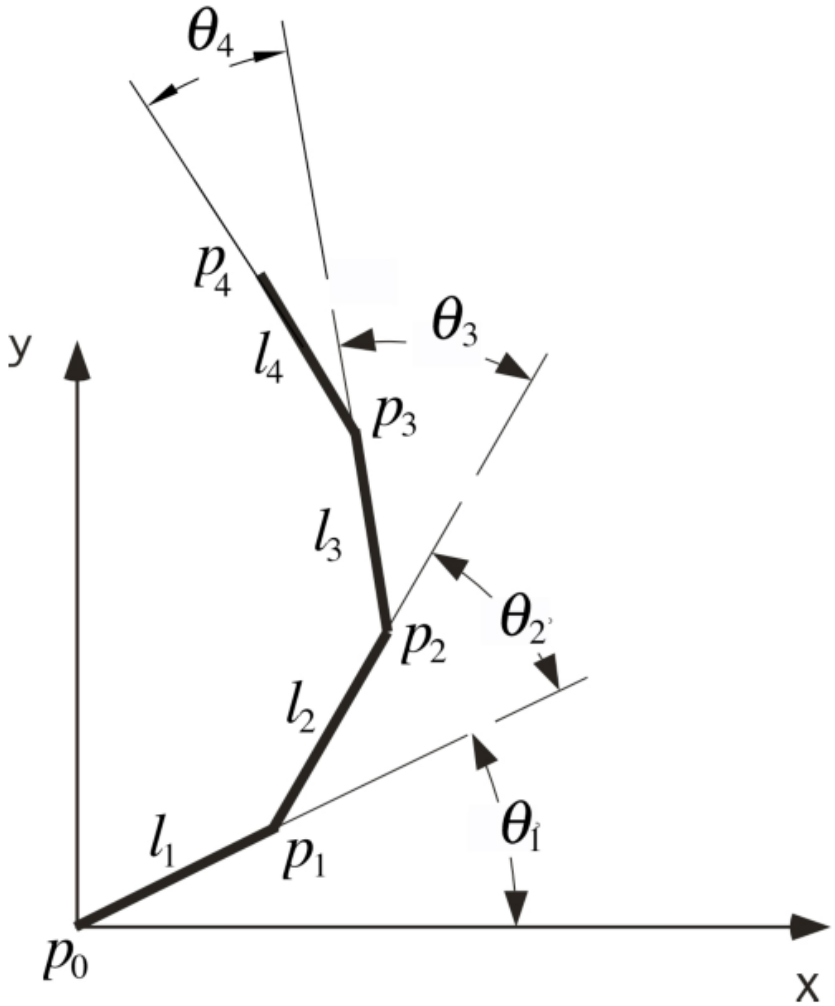
Where α is the learning rate, $g(x)$ is the neuron's activation function. t_j and y_j is the target and actual output of the neuron. h_j is the weighted sum of the neuron's inputs. And x_i is the i_{th} input.

The above equation holds the following:

$$h_j = \blacksquare x_i w_{ji}$$

$$y_j = g(h_j)$$

APPLICATIONS – INVERSE KINEMATICS



APPLICATIONS – INVERSE KINEMATICS

Goal of Inverse Kinematics

Given a position in the space, calculate a way for a robot hand to reach a place.

Problem Abstract:

$$\vec{e} = R_1 T_1 R_2 T_2 R_3 T_3 R_4 T_4 \vec{e_0}$$

Where T_i is a series of translation transformation and R_i is a series of rotation translation.

APPLICATIONS – INVERSE KINEMATICS

Abstraction for Convex Optimization:

$$\blacksquare\vec{\theta} = \alpha J^T \vec{e}$$

.

The target for the optimization is to achieve $|\vec{e_p} - \vec{e_t}| = 0$, where $\vec{e_p}$ is th original position of the tip of the robotic arm and $\vec{e_t}$ is the target position. J is the jacobian matrix in terms of $\vec{\theta}$, which is the vector of all the spatial angles of all joints. α is the convergence rate and $\vec{e}$ is the position derivation (step size).

APPLICATIONS – INVERSE KINEMATICS

About Inverse Kinematics

- Jacobian transpose is the implementation of gradient descent in the real physical world.
- It can actually achieve near linear solution for robotic arms with a fast convergence rate.