Stochastic Gradient Descent Optimization

Kaiwen Cai

1. Non Linear Optimization

For a least square minimsation problem:

$$\min_{\boldsymbol{x}} F(\boldsymbol{x}) = \frac{1}{2}\|f(\boldsymbol{x})\|_2^2$$

where $\boldsymbol{x} \in \mathbb{R}^n$. A direct method is to calculate the first order derivtive:

$$\frac{dF(\boldsymbol{x})}{d\boldsymbol{x}} = 0 \tag{1}$$

Then the optimum $\boldsymbol{x}$ is obtained. However, Solving the equation 1 requires knowledge of the global characterics, which are ususally intracable. Thus, we resort to a stochastic alternative:

1. starting from an initial value $\boldsymbol{x}_0$.
2. in the k_{th} iteration, find an increment $\Delta\boldsymbol{x}_k$, such that $F(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k)$ is the minimum in the local region.
3. if $\Delta\boldsymbol{x}_k$ is smaller than a predefined creterion, then we stop the iteration.
4. update: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$, go to step (2).

1.1. Newton Method

Based on the stochastic method, we will try to linearize $F(\boldsymbol{x})$ in each iteration:

$$F(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k) \approx F(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)^T\Delta\boldsymbol{x}_k + \frac{1}{2}\Delta\boldsymbol{x}_k^T\boldsymbol{H}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k \tag{2}$$

where $\boldsymbol{J}(\boldsymbol{x}_k)$ and $\boldsymbol{H}(\boldsymbol{x}_k)$ is the first order and second order derivative functions, i.e., Jacobian and Hessian matrix. If we ignore the second order item, the optimum $\Delta\boldsymbol{x}_k^*$ would be:

$$\Delta\boldsymbol{x}_k^* = -\boldsymbol{J}(\boldsymbol{x}_k)$$

Otherwise if we consider the second order item, the cost function would be

$$\Delta\boldsymbol{x}_k^* = \min_{\boldsymbol{\Delta}x_k} \|F(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)^T\Delta\boldsymbol{x}_k + \frac{1}{2}\Delta\boldsymbol{x}_k^T\boldsymbol{H}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k\|_2^2$$

We calculate the first order derivative of the RHS with respect to $\Delta\boldsymbol{x}_k$, and let it equal zero. This would give us:

$$\boldsymbol{J}(\boldsymbol{x}_k) + \boldsymbol{H}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k = \boldsymbol{0} \Rightarrow \boldsymbol{H}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k = -\boldsymbol{J}(\boldsymbol{x}_k). \tag{3}$$

The solution of equation 3 is $\Delta\boldsymbol{x}_k^*$.

1.2. Gaussian-Newton Method

$$F(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k) = \frac{1}{2}\|f(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k)\|_2^2 \approx \frac{1}{2}\|f(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)^T\Delta\boldsymbol{x}_k\|_2^2 \tag{4}$$

$$\Delta\boldsymbol{x}_k^* = \min_{\boldsymbol{\Delta}x_k} \frac{1}{2}\|f(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)^T\Delta\boldsymbol{x}_k\|_2^2$$

We calculate the first order derivative of the RHS with respect to $\Delta\boldsymbol{x}_k$, and let it equal zero. This would give us:

$$\boldsymbol{J}(\boldsymbol{x}_k)f(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)\boldsymbol{J}(\boldsymbol{x}_k)^T\Delta\boldsymbol{x}_k = 0$$

i.e.,

$$\underbrace{\boldsymbol{J}(\boldsymbol{x}_k)\boldsymbol{J}(\boldsymbol{x}_k)^T}_{\boldsymbol{H}(\boldsymbol{x}_k)}\Delta\boldsymbol{x}_k = -\boldsymbol{J}(\boldsymbol{x}_k)f(\boldsymbol{x}_k) \tag{5}$$

Thus, an optimisation pipeline is:

Gaussian Newton Method:

1. starting from an initial value $\boldsymbol{x}_0$.
2. in the k_{th} iteration, calculate $\boldsymbol{J}(\boldsymbol{x}_k)$ and $f(\boldsymbol{x}_k)$.
3. obatain $\Delta\boldsymbol{x}_k^*$ by solving equation 5.
4. if $\Delta\boldsymbol{x}_k^*$ is smaller than a predefined creterion, then we stop the iteration.
5. update: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k^*$, and go to step (2).

Here we practice the Gaussian-Newton method on a simple curve fitting task. For example, given a batch of samples $\{x_i, y_i | i = 1, 2, \cdots, N\}$, each of which can be roughly parametrized by:

$$y_i = \exp(\hat{a}x_i^2 + \hat{b}x_i + \hat{c})$$

The ground truth parameters $\hat{a}, \hat{b}$ and $\hat{c}$ are unknown. The task is to find the optimum parameters a^*, b^* and c^* that best fits the samples. We denote the variable to be estimated as $\boldsymbol{p} = [a, b, c]^T$. Thus, the cost function is:

$$f_i(\boldsymbol{p}) = y_i - \exp(ax_i^2 + bx_i + c)$$

$$F(\boldsymbol{p}) = \sum_i^N \|f_i(\boldsymbol{p})\|_2^2$$

In each iteration, the cost function we are going to minimise is:

$$F(\boldsymbol{p} + \Delta\boldsymbol{p}) = \sum_i^N \|f_i(\boldsymbol{p} + \Delta\boldsymbol{p})\|_2^2 = \sum_i^N \|f_i(\boldsymbol{p}) + \boldsymbol{J}_i(\boldsymbol{p})^T\Delta\boldsymbol{p}\|_2^2$$

$$\Delta\boldsymbol{p}_k^* = \min_{\boldsymbol{\Delta}p_k} \sum_i^N \frac{1}{2}\|f_i(\boldsymbol{p}_k) + \boldsymbol{J}_i(\boldsymbol{p}_k)^T\Delta\boldsymbol{p}_k\|_2^2$$

We calculate the first order derivative of the RHS with respect to $\Delta\boldsymbol{p}_k$, and let it equal zero. This would give us:

$$\sum_i^N \boldsymbol{J}(\boldsymbol{p}_k)f_i(\boldsymbol{p}_k) + \sum_i^N \boldsymbol{J}_i(\boldsymbol{p}_k)\boldsymbol{J}_i(\boldsymbol{p}_k)^T\Delta\boldsymbol{p}_k = 0$$

i.e.,

$$\sum_i^N \boldsymbol{J}_i(\boldsymbol{p}_k)\boldsymbol{J}_i(\boldsymbol{p}_k)^T\Delta\boldsymbol{p}_k = -\sum_i^N \boldsymbol{J}(\boldsymbol{p}_k)f_i(\boldsymbol{p}_k)$$

The Jacobian matrix is calculated as:

$$\boldsymbol{J}_i(\boldsymbol{p}_k) = [\frac{\partial f_i(\boldsymbol{p}_k)}{\partial a}, \frac{\partial f_i(\boldsymbol{p}_k)}{\partial b}, \frac{\partial f_i(\boldsymbol{p}_k)}{\partial c}]^T$$

$$\begin{cases} \frac{\partial f_i(\boldsymbol{p}_k)}{\partial a} = -x_i^2 \exp(ax_i^2 + bx_i + c) \\ \frac{\partial f_i(\boldsymbol{p}_k)}{\partial b} = -x_i \exp(ax_i^2 + bx_i + c) \\ \frac{\partial f_i(\boldsymbol{p}_k)}{\partial c} = -\exp(ax_i^2 + bx_i + c) \end{cases}$$

Then we follow the optimisation steps to iteratively calculate each $\boldsymbol{p}_k$.

1.3. Gaussian Newton Method with Information Matrix

If the samples are corrupted by a known noise, e.g., Gaussian noise $w \sim (0, \sigma^2)$, then the sample model can be regarded as:

$$y_i = \exp(\hat{a}x_i^2 + \hat{b}x_i + \hat{c}) + w_i$$

then

$$f_i(\boldsymbol{p}) \sim (y_i - \exp(ax_i^2 + bx_i + c), w_i)$$

the cost function considering Gaussian noise is:

$$\Delta\boldsymbol{p}_k^* = \min_{\boldsymbol{\Delta}p_k} \sum_i^N \frac{1}{2}\frac{1}{w_i^2}\|f_i(\boldsymbol{p}_k) + \boldsymbol{J}_i(\boldsymbol{p}_k)^T\Delta\boldsymbol{p}_k\|_2^2$$

$$\sum_i^N \frac{1}{w_i^2}\boldsymbol{J}(\boldsymbol{p}_k)f_i(\boldsymbol{p}_k) + \sum_i^N \frac{1}{w_i^2}\boldsymbol{J}_i(\boldsymbol{p}_k)\boldsymbol{J}_i(\boldsymbol{p}_k)^T\Delta\boldsymbol{p}_k = 0$$

i.e.,

$$\sum_i^N \frac{1}{w_i^2}\boldsymbol{J}_i(\boldsymbol{p}_k)\boldsymbol{J}_i(\boldsymbol{p}_k)^T\Delta\boldsymbol{p}_k = -\sum_i^N \frac{1}{w_i^2}\boldsymbol{J}(\boldsymbol{p}_k)f_i(\boldsymbol{p}_k)$$

See python code for detailed comparison experiment on the impact of the variance. Experiment conclusion: Taking sample variance into consideration will significantly improve parameters estimation accuracy.

1.4. Levenberg-Marquardt Method

$$\rho = \frac{f(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k) - f(\boldsymbol{x}_k)}{\boldsymbol{J}(\boldsymbol{x}_k)^T\Delta\boldsymbol{x}_k} \tag{6}$$

ρ indicates how well the approximation is. A robust optimization pipeline is:

Levenberg-Marquardt Method:

1. starting from an initial value $\boldsymbol{x}_0$.
2. in the k_{th} iteration, we solve:

$$\Delta\boldsymbol{x}_k^* = \min_{\boldsymbol{\Delta}x_k} \frac{1}{2}\|f(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)^T\Delta\boldsymbol{x}_k\|_2^2, \qquad s.t.\|\boldsymbol{D}\Delta\boldsymbol{x}_k\| \leqslant \mu$$

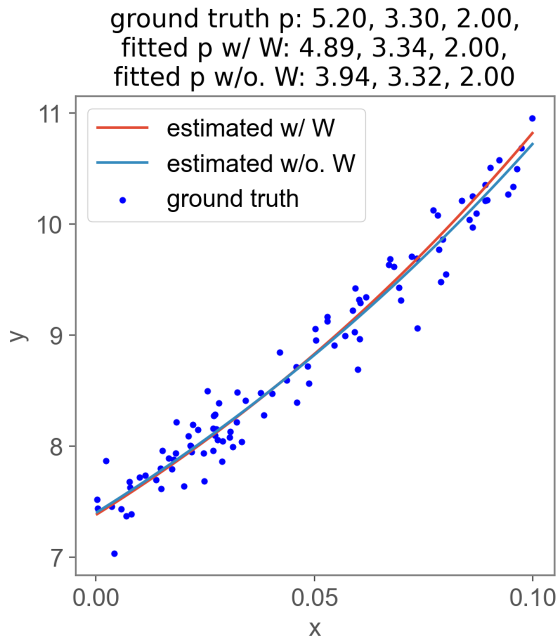

Figure 1. Curve fitting result using Gaussian Newton Method w/ and w/o. information matrix.

3. if $\Delta \boldsymbol{x}_k^*$ is smaller than a predefined creterion, then we stop the iteration.
4. calculate ρ by equation 6. If $\rho > \frac{3}{4}$, then $\mu_{k+1} = 2\mu_k$; else if $\rho < \frac{1}{4}$, then $\mu_{k+1} = 0.5\mu_k$.
5. if ρ is greater than a predefined threhold, then update: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \boldsymbol{x}_k^*$. Go to step 2.

2. Lucas-Kanade Algorithm

2.1. Additive Forward Algorithm

Given a template image $T(\boldsymbol{x})$, we apply an affine transformation $\boldsymbol{W}(\boldsymbol{x}; \boldsymbol{p}_{gt})$ to it and obtain a tranformed image $I(\boldsymbol{x}) = T(\boldsymbol{W}(\boldsymbol{x}; \boldsymbol{p}_{gt}))$. Suppose the affine transformation is hidden and we want to align an input image $I(\boldsymbol{x})$ with a template image $T(\boldsymbol{x})$, i.e., estimate the affine transformation $\boldsymbol{W}(\boldsymbol{x}; \boldsymbol{p})$. Formally,

$$\boldsymbol{x} = [x, y]^T$$
$$\boldsymbol{p} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \end{bmatrix}^T$$
$$\boldsymbol{W}(\boldsymbol{x}; \boldsymbol{p}) = \begin{bmatrix} 1 + p_1 & p_3 & p_5 \\ p_2 & 1 + p_4 & p_6 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

By saying 'alignment' we are actually trying to minimise a cost function:

$$F(\boldsymbol{p}) = \sum_{\boldsymbol{x}} \|f(\boldsymbol{p})\|_2^2 = \sum_{\boldsymbol{x}} \|T(\boldsymbol{x}) - I(\boldsymbol{W}(\boldsymbol{x}; \boldsymbol{p}))\|_2^2$$

A straight-foward solution is to find the zero point of the first order derivative, i.e., $\frac{\partial F(\boldsymbol{p})}{\boldsymbol{p}} = 0$. But this is impossible because while $\boldsymbol{W}(\boldsymbol{x}; \boldsymbol{p})$ is a linear function w.r.t $\boldsymbol{p}$, $I(\boldsymbol{x})$ is a non-linear function w.r.t $\boldsymbol{x}$. Thus, We resort to optimise the cost function in a local region rather than a global region:

$$\begin{aligned}
\Delta \boldsymbol{p}^* &= \arg\min_{\Delta \boldsymbol{p}} \sum_{\boldsymbol{x}} \| I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}+\Delta \boldsymbol{p})) - T(\boldsymbol{x}) \|_2^2 \\
&= \arg\min_{\Delta \boldsymbol{p}} \sum_{\boldsymbol{x}} \| I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})) + \frac{\partial I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))}{\partial \boldsymbol{p}} \Delta \boldsymbol{p} - T(\boldsymbol{x}) \|_2^2 \\
&= \arg\min_{\Delta \boldsymbol{p}} \sum_{\boldsymbol{x}} \| I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})) + \underbrace{\frac{\partial I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))}{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})}}_{1} \underbrace{\frac{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})}{\partial \boldsymbol{p}} \Delta \boldsymbol{p})}_{2} - T(\boldsymbol{x}) \|_2^2 \\
&= \arg\min_{\Delta \boldsymbol{p}} \sum_{\boldsymbol{x}} \| I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})) + \underbrace{\nabla I \frac{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})}{\partial \boldsymbol{p}}}_{\boldsymbol{J}} \Delta \boldsymbol{p}) - T(\boldsymbol{x}) \|_2^2
\end{aligned}$$

where ∇I is image gradient, and $\frac{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})}{\partial \boldsymbol{p}} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}$. In each iteration, the optimum increment is calculated by solving the following equation:

$$\boldsymbol{J}(\boldsymbol{p})\boldsymbol{J}(\boldsymbol{p})^T \Delta \boldsymbol{p}^* = -\boldsymbol{J}(\boldsymbol{p}) f(\boldsymbol{p})$$

then update:

$$\boldsymbol{p} \leftarrow \boldsymbol{p} + \Delta \boldsymbol{p}$$

Since $\boldsymbol{J}(\boldsymbol{p})$ is depent on $\boldsymbol{p}$, $\boldsymbol{J}(\boldsymbol{p})$ needs to be re-calculated in each itereation.

Puzzles:

1. $\frac{\partial I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))}{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})}$ should be evaluated at $\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})$, right? [1] does an operation: warp the gradient ∇I with $\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})$.

LK additive forward algorithm:

1. starting from an intial guess $\boldsymbol{p}_k = \boldsymbol{p}_0$, pixel region $\mathcal{X}$
2. calculate the error $f(\boldsymbol{p}_k) = I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}_k)) - T(\boldsymbol{x})$ for each $\boldsymbol{x} \in \mathcal{X}$.
3. calculate $\frac{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})}{\partial \boldsymbol{p}}|_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{p}=\boldsymbol{p}_k}$.
4. calculate the gradient of image I and warp it with $\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}_k)$ and we get ∇I.
5. $\boldsymbol{J}(\boldsymbol{p}_k) = \nabla I \frac{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})}{\partial \boldsymbol{p}}|_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{p}=\boldsymbol{p}_k}$
6. $\Delta \boldsymbol{p}_k^* = -\sum_{x\in\mathcal{X}} [\boldsymbol{J}(\boldsymbol{p}_k)\boldsymbol{J}(\boldsymbol{p}_k)^T]^{-1} \boldsymbol{J}(\boldsymbol{p}_k) f(\boldsymbol{p}_k)$
7. check if stop the iteration, otherwise update $\boldsymbol{p}_{k+1} = \boldsymbol{p}_k + \Delta \boldsymbol{p}_k^*$.
8. go to step 2.

2.2. Compositional Algorithm

Compositional algorithm decomposes the warpping as:

$$\begin{aligned}
k+1 &\leftarrow k \\
\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}+\Delta \boldsymbol{p}) &\leftarrow \boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}) \qquad \text{: Additive Forward Algorithm update} \\
\boldsymbol{W}(\boldsymbol{W}(\boldsymbol{x};\Delta \boldsymbol{p});\boldsymbol{p}) &\leftarrow \boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}) \qquad \text{: Compositional Algorithm update}
\end{aligned}$$

[1]This term means gradient image evaluated at the warpped pixels.
[2]Note this term is evaluated at the $\boldsymbol{x}$ and $\boldsymbol{p}$.

$$
\begin{aligned}
\Delta \boldsymbol{p}^* &= \arg\min_{\Delta \boldsymbol{p}} \sum_{\boldsymbol{x}} \|I(\boldsymbol{W}(\boldsymbol{W}(\boldsymbol{x};\Delta\boldsymbol{p});\boldsymbol{p})) - T(\boldsymbol{x})\|_2^2 \\
&= \arg\min_{\Delta \boldsymbol{p}} \sum_{\boldsymbol{x}} \|I(\boldsymbol{W}(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{0}),\boldsymbol{p})) + \underbrace{\frac{\partial I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))}{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{0})}}_{3} \underbrace{\frac{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{0})}{\partial \boldsymbol{p}}}_{4} \Delta\boldsymbol{p} - T(\boldsymbol{x})\|_2^2 \\
&= \arg\min_{\Delta \boldsymbol{p}} \sum_{\boldsymbol{x}} \|I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})) + \underbrace{\nabla I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))\frac{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{0})}{\partial \boldsymbol{p}}}_{\boldsymbol{J}} \Delta\boldsymbol{p} - T(\boldsymbol{x})\|_2^2
\end{aligned}
$$

$\nabla I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))$ is easily obatined since we will have to calculate the first term $I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))$ anyway. $\frac{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{0})}{\partial \boldsymbol{p}}$ is not depent on $\boldsymbol{p}$, thus it needs to be calculated only once. Since we are using compositional warpping, update of $\boldsymbol{p}$ cannot be done with simple addition. Instead:

$$
\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}_{k+1}) = \boldsymbol{W}(\boldsymbol{W}(\boldsymbol{x};\Delta\boldsymbol{p}_k);\boldsymbol{p}_k) = \boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}_k)\cdot \boldsymbol{W}(\boldsymbol{x};\boldsymbol{\Delta} p_k)
$$

$$
\begin{aligned}
\begin{bmatrix} 1+p_1^{k+1} & p_3^{k+1} & p_5^{k+1} \\ p_2^{k+1} & 1+p_4^{k+1} & p_6^{k+1} \\ 0 & 0 & 1 \end{bmatrix} &= \begin{bmatrix} 1+p_1^k & p_3^k & p_5^k \\ p_2^k & 1+p_4^k & p_6^k \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1+\Delta p_1^k & \Delta p_3^k & \Delta p_5^k \\ \Delta p_2^k & 1+\Delta p_4^k & \Delta p_6^k \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} (1+p_1^k)(1+\Delta p_1^k) + p_3^k \Delta p_2^k & (1+p_1^k)\Delta p_3^k + p_3^k(1+\Delta p_4^k) & (1+p_1^k)\Delta p_5^k + p_3^k\Delta p_6^k + p_5^k \\ p_2^k(1+\Delta p_1^k) + (1+p_4^k)\Delta p_2^k & p_2^k\Delta p_3^k + (1+p_4^k)(1+\Delta p_4^k) & p_2^k \Delta p_5^k + (1+p_4^k)\Delta p_6^k + p_6^k \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

Solving the above equation will give us the update equation:

$$
\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{bmatrix} = \begin{bmatrix} p_1 + \Delta p_1 + p_1\Delta p_1 + p_3 \Delta p_2 \\ p_2 + \Delta p_2 + p_1\Delta p_1 + p_4 \Delta p_2 \\ p_3 + \Delta p_3 + p_1\Delta p_3 + p_3 \Delta p_4 \\ p_4 + \Delta p_4 + p_1\Delta p_3 + p_4 \Delta p_4 \\ p_5 + \Delta p_5 + p_1\Delta p_5 + p_3 \Delta p_6 \\ p_6 + \Delta p_6 + p_1\Delta p_5 + p_4 \Delta p_6 \end{bmatrix} \tag{7}
$$

LK Compositional Algorithm:

1. calculate $\frac{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{0})}{\partial \boldsymbol{p}}|_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{p}=\boldsymbol{0}}$.
2. starting from an intial guess $\boldsymbol{p}_k = \boldsymbol{p}_0$, pixel region $\mathcal{X}$
3. calculate the error $f(\boldsymbol{p}_k) = I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}_k)) - T(\boldsymbol{x})$ for each $\boldsymbol{x}\in\mathcal{X}$.
4. calculate the gradient of image $I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}_k))$ and we get $\nabla I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))$.
5. $\boldsymbol{J}(\boldsymbol{p}_k) = \frac{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{0})}{\partial \boldsymbol{p}}|_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{p}=\boldsymbol{0}} \cdot \nabla I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))$
6. $\Delta \boldsymbol{p}_k^* = -\sum_{x\in\mathcal{X}}[\boldsymbol{J}(\boldsymbol{p}_k)\boldsymbol{J}(\boldsymbol{p}_k)^T]^{-1}\boldsymbol{J}(\boldsymbol{p}_k)f(\boldsymbol{p}_k)$
7. check if stop the iteration, otherwise update $\boldsymbol{p}_{k+1}$ using Eq. 7.
8. go to step 3.

[3]This term means gradient of the warpped image evaluated at the original pixels.
[4]Note this term is evaluated at the $\boldsymbol{x}$ and $\boldsymbol{0}$.

2.3. Inverse Compositional Algorithm

Similarly, ICA is also aimed to solve this optimisatin problem from a local region. The difference is that the cost function can be written from another perspective:

$$
\begin{aligned}
\Delta\boldsymbol{p}^* &= \arg\min_{\Delta\boldsymbol{p}} \sum_{\boldsymbol{x}} \|T(\boldsymbol{W}(\boldsymbol{x};\Delta\boldsymbol{p})) - I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))\|_2^2 \\
&= \arg\min_{\Delta\boldsymbol{p}} \sum_{\boldsymbol{x}} \|T(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{0})) + \frac{\partial T(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{0}))}{\partial \boldsymbol{p}}\Delta\boldsymbol{p} - I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))\|_2^2 \\
&= \arg\min_{\Delta\boldsymbol{p}} \sum_{\boldsymbol{x}} \|T(\boldsymbol{x}) + \underbrace{\frac{\partial T(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{0}))}{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{0})}}_{5} \underbrace{\frac{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{0})}{\partial \boldsymbol{p}}}_{6} \Delta\boldsymbol{p} - I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))\|_2^2 \\
&= \arg\min_{\Delta\boldsymbol{p}} \sum_{\boldsymbol{x}} \|T(\boldsymbol{x}) + \underbrace{\nabla T \frac{\partial T(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{0}))}{\partial \boldsymbol{p}}}_{\boldsymbol{J}} \Delta\boldsymbol{p} - I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))\|_2^2
\end{aligned}
$$

update equation is:

$$T(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{0})) - I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}_{k+1})) = T(\boldsymbol{W}(\boldsymbol{x};\Delta\boldsymbol{p}_k)) - I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}_k)) \implies \boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}_{k+1}) = \boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}_k)\cdot \boldsymbol{W}(\boldsymbol{x};\boldsymbol{\Delta} p_k)^{-1}$$

We want to update using

$$\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}_{k+1}) = \boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}_k)\cdot \boldsymbol{W}(\boldsymbol{x};\boldsymbol{\Delta} p'_k)$$

therefore, solving the below equations

$$
\begin{aligned}
\boldsymbol{W}(\boldsymbol{x};\boldsymbol{\Delta} p_k)^{-1} &= \begin{bmatrix} 1+\Delta p_1 & \Delta p_3 & \Delta p_5 \\ \Delta p_2 & 1+\Delta p_4 & \Delta p_6 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \\
&= \frac{1}{(1+\Delta p_1)(1+\Delta p_4) - \Delta p_2 \Delta p_3} \begin{bmatrix} 1+\Delta p_4 & -\Delta p_3 & -\Delta p_5 - \Delta p_4\Delta p_5 + \Delta p_3 \Delta p_6 \\ -\Delta p_2 & 1+\Delta p_1 & -\Delta p_6 - \Delta p_1 \Delta p_6 + \Delta p_2\Delta p_5 \\ 0 & 0 & 1 \end{bmatrix} \\
\boldsymbol{W}(\boldsymbol{x};\Delta p') &= \begin{bmatrix} 1+\Delta p'_1 & \Delta p'_3 & \Delta p'_5 \\ \Delta p'_2 & 1+\Delta p'_4 & \Delta p'_6 \\ 0 & 0 & 1 \end{bmatrix} \\
\boldsymbol{W}(\boldsymbol{x};\boldsymbol{\Delta} p_k)^{-1} &= \boldsymbol{W}(\boldsymbol{x};\Delta p')
\end{aligned} \tag{8}
$$

will result in:

$$\begin{bmatrix} \Delta p'_1 \\ \Delta p'_2 \\ \Delta p'_3 \\ \Delta p'_4 \\ \Delta p'_5 \\ \Delta p'_6 \end{bmatrix} = \frac{1}{(1+\Delta p_1)(1+\Delta p_4) - \Delta p_2\Delta p_3} \cdot \begin{bmatrix} -\Delta p_1 - \Delta p_1 \Delta p_4 + \Delta p_2 \Delta p_3 \\ -\Delta p_2 \\ -\Delta p_3 \\ -\Delta p_4 - \Delta p_1\Delta p_4 + \Delta p_2 \Delta p_3 \\ -\Delta p_5 - \Delta p_4 \Delta p_5 + \Delta p_3 \Delta p_6 \\ -\Delta p_6 - \Delta p_1 \Delta p_6 + \Delta p_2 \Delta p_5 \end{bmatrix} \tag{9}$$

Now we can update using

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{bmatrix} \leftarrow \begin{bmatrix} p_1 + \Delta p'_1 + p_1 \Delta p'_1 + p_3 \Delta p'_2 \\ p_2 + \Delta p'_2 + p_1 \Delta p'_1 + p_4 \Delta p'_2 \\ p_3 + \Delta p'_3 + p_1 \Delta p'_3 + p_3 \Delta p'_4 \\ p_4 + \Delta p'_4 + p_1 \Delta p'_3 + p_4 \Delta p'_4 \\ p_5 + \Delta p'_5 + p_1 \Delta p'_5 + p_3 \Delta p'_6 \\ p_6 + \Delta p'_6 + p_1 \Delta p'_5 + p_4 \Delta p'_6 \end{bmatrix} \tag{10}$$

Note that $\boldsymbol{J}$ is indepent on $\boldsymbol{p}$, this wonderful property makes it possible that we calculate the Jacobian once, and then use it over and over again.

[5]This term means gradient of the image evaluated at the original pixels.

[6]Note this term is evaluated at the $\boldsymbol{x}$ and $\boldsymbol{0}$.

Figure 2. Left to right: template image T , transformed template image I , recovered template image $\tilde{T}$ using ICA.

LK Inverse Compositional Algorithm:

1. calculate $\frac{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{0})}{\partial \boldsymbol{p}}|_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{p}=\boldsymbol{0}}$, ∇T and $\boldsymbol{J} = \frac{\partial \boldsymbol{W}(\boldsymbol{x};\boldsymbol{0})}{\partial \boldsymbol{p}}|_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{p}=\boldsymbol{0}} \cdot \nabla T$.
2. starting from an intial guess $\boldsymbol{p}_k = \boldsymbol{p}_0$, pixel region $\boldsymbol{x} \in \mathcal{X}$
3. calculate the error $f(\boldsymbol{p}_k) = T(\boldsymbol{x}) - I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}_k))$ for each $\boldsymbol{x} \in \mathcal{X}$.
4. $\Delta \boldsymbol{p}_k^* = -\sum_{x\in\mathcal{X}}[\boldsymbol{J}(\boldsymbol{p}_k)\boldsymbol{J}(\boldsymbol{p}_k)^T]^{-1}\boldsymbol{J}(\boldsymbol{p}_k)f(\boldsymbol{p}_k)$
5. check if stop the iteration, otherwise update $\boldsymbol{p}_{k+1}$ using the equation 9 and Eq. 10.
6. go to step 3.

References

[1] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, 56(3):221–255, 2004.