Dynamic Programming for Total Variation Denoising, Explained

Ryan J. Tibshirani

In this note, we will explain Nick Johnson's dynamic programming algorithm for univariate total variation denoising, also called the univariate fused lasso signal approximator. We consider the weighted problem:

$$\underset{\beta \in \mathbb{R}^n}{\text{minimize}} \; \frac{1}{2} \sum_{i=1}^n w_i (y_i - \theta_i)^2 + \lambda \sum_{i=1}^{n-1} |\theta_i - \theta_{i+1}|, \tag{1}$$

for a fixed regularization parameter $\lambda \geq 0$, and strictly positive observation weights $w_i > 0$, $i = 1, \ldots n$.

1 Standard representation, $O(n^2)$ complexity

Define

$$f_1(\theta_1) = \frac{1}{2} w_1 (y_1 - \theta_1)^2,$$

and then define, for all $k = 1, \ldots n-1$,

$$\begin{aligned}
b_k(\theta_{k+1}) &= \underset{\theta_k \in \mathbb{R}}{\text{argmin}} \; f_k(\theta_k) + \lambda |\theta_k - \theta_{k+1}| \\
g_k(\theta_{k+1}) &= \min_{\theta_k \in \mathbb{R}} \; f_k(\theta_k) + \lambda |\theta_k - \theta_{k+1}| \\
&= f_k(b_k(\theta_{k+1})) + \lambda |b_k(\theta_{k+1}) - \theta_{k+1}| \\
f_{k+1}(\theta_{k+1}) &= g_k(\theta_{k+1}) + \frac{1}{2} w_{k+1} (y_{k+1} - \theta_{k+1})^2.
\end{aligned}$$

The following result is key.

Lemma 1. *For all $k = 1, \ldots n-1$, the functions f_k, g_k are piecewise quadratic, convex, and differentiable. Further, there exists points t_k^-, t_k^+ such that*

$$f_k'(t_k^-) = -\lambda, \quad f_k'(t_k^+) = \lambda,$$

and

$$b_k(\theta_{k+1}) = T_{t_k^-, t_k^+}(\theta_{k+1}) = \begin{cases} t_k^+ & \theta_{k+1} > t_k^+ \\ \theta_{k+1} & \theta_{k+1} \in [t_k^-, t_k^+] \\ t_k^- & \theta_{k+1} < t_k^-. \end{cases}$$

Proof. We first prove all claims by induction, considering first the base case $k = 1$. The function f_1 is clearly a convex, differentiable quadratic. Furthermore, by subgradient optimality, θ_1 minimizes $f_1(\theta_1) + \lambda |\theta_1 - \theta_2|$ if and only if

$$f_1'(\theta_1) + \lambda s = 0,$$

where s is a subgradient of $|\theta_1 - \theta_2|$ with respect to θ_1, i.e., $s = \text{sign}(\theta_1 - \theta_2)$ when $\theta_1 - \theta_2 \neq 0$, and $s \in [-1, 1]$ otherwise. This optimality condition is satisfied at $\theta_1 = b_1(\theta_2) = T_{t_1^-, t_1^+}(\theta_2)$, where $f_1'(t_1^-) = -\lambda$, $f_1'(t_1^+) = \lambda$. To see this, check each case:

- when $\theta_2 > t_1^+$, we have $b_1(\theta_2) - \theta_2 < 0$, so $s = -1$, and the optimality condition reads

$$\lambda + \lambda \cdot (-1) = 0;$$

- when $\theta_2 \in [t_1^-, t_1^+]$, we have $f_1'(\theta_2) \in [-\lambda, \lambda]$ by definition of t_1^-, t_1^+ (and monotonicity of the derivative), and $b_1(\theta_2) - \theta_2 = 0$, so we can choose the subgradient to be any value in $[-1, 1]$, and for $s = -f_1'(\theta_2)/\lambda$, we have
$$f_1'(\theta_2) + \lambda \cdot \big(- f_1'(\theta_2)/\lambda \big) = 0;$$
- when $\theta_2 < t_1^-$, the argument is similar to the first case.

(We remark that we can explicitly compute $t_1^- = -\lambda/w_1 + y_1$ and $t_1^+ = \lambda/w_1 + y_1$.) Finally, as for g_1, note that $f_1(\theta_1) + \lambda|\theta_1 - \theta_2|$ is jointly convex in θ_1, θ_2, and therefore g_1, which is given by partial minimization over θ_1, is convex in its argument θ_2. It is piecewise quadratic with knots at t_1^-, t_1^+, and in fact we can write it explicitly as
$$g_1(\theta_2) = \begin{cases} \frac{1}{2} w_1 (y_1 - t_1^+)^2 + \lambda(\theta_2 - t_1^+) & \theta_2 > t_1^+ \\ \frac{1}{2} w_1 (y_1 - \theta_2)^2 & \theta_2 \in (t_1^-, t_1^+) \\ \frac{1}{2} w_1 (y_1 - t_1^-)^2 + \lambda(t_1^- - \theta_2) & \theta_2 < t_1^-. \end{cases}$$
From this we can see that it is differentiable because its left and right derivatives match at t_1^-, t_1^+ (they are equal to $-\lambda, \lambda$ respectively). This completes the base case.

Now assume all statements are true at $k-1$. Then $f_k(\theta_k) = g_{k-1}(\theta_k) + \frac{1}{2} w_k (y_k - \theta_k)^2$ is piecewise quadratic, convex, and differentiable, because $g_{k-1}(\theta_k)$ is. By subgradient optimality, θ_k minimizes $f_k(\theta_k) + \lambda|\theta_k - \theta_{k+1}|$ if and only if
$$f_k'(\theta_k) + \lambda s = 0,$$
where s is a subgradient of $|\theta_k - \theta_{k+1}|$ with respect to θ_k. By the same arguments as in the base case, at optimality we have $\theta_k = b_k(\theta_{k+1}) = T_{t_k^-, t_k^+}(\theta_{k+1})$, where t_k^-, t_k^+ are characterized by $f_k'(t_k^-) = -\lambda$, $f_k'(t_k^+) = \lambda$. (Why do such points exist? Note that we can see directly from its definition that $g_{k-1}(\theta_k)$ is linear outside of $[t_{k-1}^-, t_{k-1}^+]$, and hence $f_k(\theta_k) = g_{k-1}(\theta_k) + \frac{1}{2} w_k (y_k - \theta_k)^2$ has derivatives approaching $\pm\infty$ as θ_k approaches $\pm\infty$.) Finally, the function $f_k(\theta_k) + \lambda|\theta_k - \theta_{k+1}|$ is jointly convex in θ_k, θ_{k+1}, and so g_k, which is given by partial minimization of this function over θ_k, is convex in θ_{k+1}. Expressing g_k as
$$\begin{aligned} g_k(\theta_{k+1}) &= f_k(b_k(\theta_{k+1})) + \lambda|b_k(\theta_{k+1}) - \theta_{k+1}| \\ &= \begin{cases} f_k(t_k^+) + \lambda(\theta_{k+1} - t_k^+) & \theta_{k+1} > t_k^+ \\ f_k(\theta_{k+1}) & \theta_{k+1} \in [t_k^-, t_k^+] \\ f_k(t_k^-) + \lambda(t_k^- - \theta_{k+1}) & \theta_{k+1} < t_k^-, \end{cases} \end{aligned}$$
we see that $g_k(\theta_{k+1})$ is piecewise quadratic, with additional knots at t_k^-, t_k^+. It is differentiable on the interior of $[t_k^-, t_k^+]$ because f_k is, and it is differentiable at t_k^-, t_k^+ by construction: the left and right derivatives match at these points (with values $-\lambda, \lambda$ respectively). This completes the proof. □

This lemma and the recursive definition of f_k, g_k suggests that we could perform one "forward pass" and end up with a single piecewise quadratic, convex, differentiable function $f_n(\theta_n)$. This function encapsulates partial minimization of the total variation denoising criterion over all $\theta_1, \ldots \theta_{n-1}$, so minimizing f_n delivers the nth component of the solution, $\hat{\theta}_n$. Once we have this value, we could then perform a "backward pass", by evaluating $\hat{\theta}_k = b_k(\hat{\theta}_{k+1})$ for $k = n-1, \ldots 1$. (The functions b_k will be called "back pointers".) We would have then have fully enumerated the total variation denoising solution, and can terminate.

The backward pass is operationally clear. The forward pass requires computation of t_k^-, t_k^+, the points at which the derivative of f_k are equal to $-\lambda, \lambda$, respectively, for each $k = 1, \ldots n-1$. How do we do this? The trick is to keep track of the knots of f_k, and the derivatives of f_k at the knots; recalling that f_k is convex and differentiable, its derivative must increase as we move from left to right through the knot points. Hence, for determination of t_k^-, for example, note that
$$-\lambda \in [f_k'(x_1), f_k'(x_2)] \implies t_k^- \in [x_1, x_2].$$
Therefore if we find the knot points x_1, x_2 such that the derivative of f_k at these points straddles $-\lambda$, then we know that t_k^- must lie between x_1 and x_2. The next result further describes how to calculate t_k^- from $x_1, x_2, f_k'(x_1), f_k'(x_2)$ (and it applies just the same to calculating t_k^+).

Lemma 2. *Suppose that q is a quadratic function, and we seek x_0 such that $q'(x_0) = d$. Then, given any x_1, x_2 such that $q'(x_1) \neq q'(x_2)$, we can compute x_0 according to*

$$x_0 = \frac{dx_1 - dx_2 + q'(x_1)x_2 - q'(x_2)x_1}{q'(x_1) - q'(x_2)}.$$

Proof. Write the derivative as $q'(x) = ax + b$. We have

$$\begin{aligned} q'(x_1) &= ax_1 + b \\ q'(x_2) &= ax_2 + b, \end{aligned}$$

and solving for a and b gives

$$\begin{aligned} a &= \frac{q'(x_1) - q'(x_2)}{x_1 - x_2} \\ b &= \frac{q'(x_2)x_1 - q'(x_1)x_2}{x_1 - x_2}. \end{aligned}$$

Hence, to have $q'(x_0) = d$, we need $x_0 = (d - b)/a$, and plugging in a, b gives the result. □

Putting this together, a summary of the dynamic programming algorithm is as follows.

Dynamic programming for total variation denoising, standard form.

1. Begin with $k = 1$. As $f_1(x) = \frac{1}{2}w_1(y_1 - x)^2$ (a single quadratic), initialize the knot set $X = \emptyset$, and derivative set $D = \emptyset$. Initialize the back pointers $t_1^- = -\lambda/w_1 + y_1$, $t_1^+ = \lambda/w_1 + y_1$.
2. For $k = 2, \ldots n$:
   (a) As $g_{k-1}(x) = f_{k-1}(b_{k-1}(x)) + \lambda|b_{k-1}(x) - x|$, and $f_k(x) = g_{k-1}(x) + \frac{1}{2}w_k(y_k - x)^2$. Append t_{k-1}^-, t_{k-1}^+ to the knot set X, and correspondingly append derivatives $-\lambda, \lambda$ to D. Delete any knots outside of these two points (any knots to the left of t_{k-1}^-, and to the right of t_{k-1}^+).
   (b) For all knots $x_i \in K$, update its derivative $d_i \in D$ according to $d_i \leftarrow d_i + w_k(x_i - y_k)$.
   (c) Find the location i^- such that $-\lambda \in [d_{i^- - 1}, d_{i^-}]$; compute t_k^- using Lemma 2.
   (d) Find the location i^+ such that $\lambda \in [d_{i^+ - 1}, d_{i^+}]$; compute t_k^+ using Lemma 2.
3. Find the location i_0 such that $0 \in [d_{i_0 - 1}, d_{i_0}]$; compute $\hat{\theta}_n$ using Lemma 2.
4. For $k = n - 1, \ldots 1$:
   (a) Let $\hat{\theta}_k = b_k(\hat{\theta}_{k+1}) = T_{t_k^-, t_k^+}(\hat{\theta}_{k+1})$.
5. Return $\hat{\theta}_1, \ldots \hat{\theta}_n$.

This algorithm is clean and simple, but what is its running time? Unfortunately the number of operations needed is $O(n^2)$. One can see this from the forward pass, in Step 2(b), where we must update the derivatives at all knots. In the worse case, the number of knots is $O(k)$ at iteration k, which makes the whole algorithm $O(n^2)$, quadratic in the number of observations.

2 Tail representation, $O(n)$ complexity

We now describe Nick's (very clever) tail representation for the derivatives f_k', which guarantees linear-time worst-case compexity for the dynamic programming approach.

Lemma 3. *For each $k = 1, \ldots n$, the derivative of the (piecewise quadratic, convex, differentiable) function f_k can be represented as*

$$f_k'(x) = a_k^{\text{first}} x + b_k^{\text{first}} + \sum_{i=1}^{m_k} 1\{x \geq x_{k,i}\}(a_{k,i}x + b_{k,i}),$$

or equivalently

$$f_k'(x) = -a_k^{\text{last}} x - b_k^{\text{last}} - \sum_{i=1}^{m_k} 1\{x < x_{k,i}\}(a_{k,i}x + b_{k,i}),$$

where m_k denotes the number of knots, $x_{k,1} < \ldots < x_{k,m_k}$ are the sorted knot points. The coefficients satisfy

$$a_k^{\text{first}} + \sum_{i=1}^{j} a_{k,j} = -a_k^{\text{last}} - \sum_{i=j+1}^{m_k} a_{k,j}, \quad b_k^{\text{first}} + \sum_{i=1}^{j} b_{k,j} = -b_k^{\text{last}} - \sum_{i=j+1}^{m_k} b_{k,j},$$

for all $j = 0, \ldots m_k$ (where the empty sum is interpreted to be 0).

This lemma suggests an elegant implementation of the dynamic programming algorithm. It can also be proved by induction, which we omit; however we give the details of the implementation next (and comment on the proof afterward.)

Dynamic programming for the 1d fused lasso, tail form.

1. Begin with $k = 1$, and $m_1 = 0$ (no knots). Set

$$a_1^{\text{first}} = w_1, \quad b_1^{\text{first}} = -w_1 y_1,$$
$$a_1^{\text{last}} = -w_1, \quad b_1^{\text{last}} = w_1 y_1.$$

2. Compute $t_1^- = -\lambda/w_1 + y_1$, $t_1^+ = \lambda/w_1 + y_1$. In preparation for iteration $k = 2$, set $m_2 = 2$, and set the knots $x_{2,1} = t_1^-$, $x_{2,2} = t_1^+$. Also set the coefficients

$$a_2^{\text{first}} = w_2, \quad b_2^{\text{first}} = -w_2 y_2 - \lambda,$$
$$a_{2,1} = a_1^{\text{first}}, \quad b_{2,1} = b_1^{\text{first}} + \lambda,$$
$$a_{2,2} = a_1^{\text{last}}, \quad b_{2,2} = b_1^{\text{last}} + \lambda,$$
$$a_2^{\text{last}} = -w_2, \quad b_2^{\text{last}} = w_2 y_2 - \lambda.$$

3. For $k = 2, \ldots n-1$:

   (a) Compute

$$\ell = \min\Big\{j \in \{1, \ldots m_k + 1\} : \Big(a_k^{\text{first}} + \sum_{i=1}^{j-1} a_{k,i}\Big)x_{k,j} + \Big(b_k^{\text{first}} + \sum_{i=1}^{j-1} b_{k,i}\Big) > -\lambda\Big\},$$
$$u = \max\Big\{j \in \{\ell - 1, \ldots m_k\} : \Big(-a_k^{\text{last}} - \sum_{i=j+1}^{m_k} a_{k,i}\Big)x_{k,j} + \Big(-b_k^{\text{first}} - \sum_{i=j+1}^{m_k} b_{k,i}\Big) < \lambda\Big\},$$

   (where we interpret an empty sum to be 0, and $x_{k,m_k+1} = \infty$), and

$$t_k^- = \frac{-\lambda - b_k^{\text{first}} - \sum_{i=1}^{\ell-1} b_{k,i}}{a_k^{\text{first}} + \sum_{i=1}^{\ell-1} a_{k,i}}, \quad t_k^+ = \frac{\lambda + b_k^{\text{last}} + \sum_{i=u+1}^{m_k} b_{k,i}}{-a_k^{\text{last}} - \sum_{i=u+1}^{m_k} a_{k,i}}.$$

   (b) In preparation for iteration $k+1$, set $m_{k+1} = u - \ell + 3$, and set the knots as

$$x_{k+1,1} = t_k^-,$$
$$x_{k+1,i} = x_{k,\ell+i-2} \quad \text{for } i = 2, \ldots u - \ell + 2,$$
$$x_{k+1,m_{k+1}} = t_k^+.$$

   (In words, we are appending t_k^-, t_k^+ as knots, and deleting all knots numbered $1, \ldots \ell - 1$ and $u+1, \ldots m_k$.)

Also set

$$a_{k+1}^{\text{first}} = w_{k+1}, \quad b_{k+1}^{\text{first}} = -w_{k+1}y_{k+1} - \lambda,$$
$$a_{k+1,1} = a_k^{\text{first}} + \sum_{i=1}^{\ell-1} a_{k,i}, \quad b_{k+1,1} = b_k^{\text{first}} + \sum_{i=1}^{\ell-1} b_{k,i} + \lambda,$$
$$a_{k+1,i} = a_{k,\ell+i-2}, \quad b_{k+1,i} = b_{k,\ell+i-2}, \quad \text{for } i = 2, \dots u - \ell + 2,$$
$$a_{k+1,m_{k+1}} = a_k^{\text{last}} + \sum_{i=u+1}^{m_k} a_{k,i}, \quad b_{k+1,m_{k+1}} = b_k^{\text{last}} + \sum_{i=u+1}^{m_k} b_{k,i} + \lambda,$$
$$a_{k+1}^{\text{last}} = -w_{k+1}, \quad b_{k+1}^{\text{last}} = w_{k+1}y_{k+1} - \lambda.$$

4. Compute

$$i_0 = \min\left\{j \in \{1, \dots m_n + 1\} : \Big(a_n^{\text{first}} + \sum_{i=1}^{j-1} a_{n,i}\Big)x_{k,j} + \Big(b_n^{\text{first}} + \sum_{i=1}^{j-1} b_{n,i}\Big) > 0\right\},$$

and

$$\hat{\theta}_n = \frac{a_n^{\text{first}} + \sum_{i=1}^{i_0-1} a_{n,i}}{b_n^{\text{first}} + \sum_{i=1}^{i_0-1} b_{n,i}}.$$

5. For $k = n-1, \dots 1$:
   (a) Let $\hat{\theta}_k = b_k(\hat{\theta}_{k+1}) = T_{t_k^-, t_k^+}(\hat{\theta}_{k+1})$.
6. Return $\hat{\theta}_1, \dots \hat{\theta}_n$.

The inductive proof for Lemma 3 essentially just follows Steps 1–3 laid out in the above algorithm; the key point is that only the coefficients $a_{k+1,1}, a_{k+1,m_{k+1}}$ and $b_{k+1,1}, b_{k+1,m_{k+1}}$ need to be modified in preparation for iteration $k+1$ (see Step 3(b)), and all other coefficients remain the same. Overall, the work required in the kth iteration is dominated by the computation of ℓ, u, and the quantities

$$\sum_{i=1}^{\ell-1} a_{k,i}, \quad \sum_{i=1}^{\ell-1} b_{k,i}, \quad \sum_{i=u+1}^{m_k} a_{k,i}, \quad \sum_{i=u+1}^{m_k} b_{k,i}.$$

This requires a number of operations equal to the number of knots deleted at iteration k, which is $\ell + (m_k - u)$. There are exactly $2n$ knots in total, and each knot can be deleted at most once, which means that the total time for forming these quantities across all iterations is $O(n)$. The rest of the algorithm is clearly linear-time, and so the dynamic programming implementationgiven above is worst-case $O(n)$.