

Optimization in General - (i.e, not just Deep Learning)

A/Prof Richard Yi Da Xu
<http://richardxu.com>

University of Technology Sydney (UTS)

October 21, 2018

Gradient Descend: what is directional derivative

- ▶ **Your aim** to find:

$$\arg \min_{\mathbf{x}} (f(\mathbf{x}))$$

- ▶ How? Solve $\nabla f(\mathbf{x}_n) = 0$! But in many scenarios, this isn't easy!
- ▶ The rate of change of $f(x, y)$ in the direction of the unit vector $u = (a, b)$ is called the directional derivative $d_u f(x, y)$. The definition of the directional derivative is:

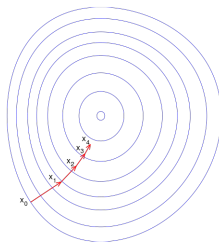
$$d_u f(x, y) = \lim_{h \rightarrow 0} \frac{f(x + ah, y + bh) - f(x, y)}{h}$$

- ▶ **Theorem** the minimum directional derivative of a differentiable function f at (x_0, y_0) is $-|\nabla f(x_0, y_0)|$ and occurs for u with the opposite direction as $\nabla f(x_0, y_0)$

Gradient Descend

Here is where **Gradient Descend** algorithm may help. The iterative algorithm looks something like:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha_n \nabla f(\mathbf{x}_n), \quad n \geq 0$$



Moral of the story, you must know how to compute the objective function's **derivative**.

- Taylor expansion of $f(\mathbf{x})$ around \mathbf{x}_n in 1-D:

$$f(x_n + \Delta x) \approx f(x_n) + f'(x_n)\Delta x + \frac{1}{2}f''(x_n)\Delta x^2$$

- we need to find what is the “right” value of Δx that minimises $f(\cdot)$:

$$\frac{df(x_n + \Delta x)}{d\Delta x} = \frac{d}{d\Delta x} \left(f(x_n) + f'(x_n)\Delta x + \frac{1}{2}f''(x_n)\Delta x^2 \right) = f'(x_n) + f''(x_n)\Delta x$$

$$f'(x_n) + f''(x_n)\Delta x = 0 \implies \Delta x = \frac{-f'(x_n)}{f''(x_n)}$$

$$\begin{aligned} x_{n+1} &= x_n + \Delta x \\ &= x_n - (f''(x_n))^{-1} f'(x_n) \end{aligned}$$

- Taylor expansion of $f(\mathbf{x})$ around \mathbf{x}_n in higher dimension:

$$\implies \mathbf{x}_{n+1} = \mathbf{x}_n - \underbrace{(\nabla^2 f(\mathbf{x}_n))^{-1}}_{\alpha_n} \nabla f(\mathbf{x}_n)$$

- ▶ knowing,

$$\begin{aligned}\frac{df(\mathbf{x}_n + \Delta \mathbf{x})}{d\Delta \mathbf{x}} &= \mathbf{0} \\ \implies \mathbf{x}_{n+1} &= \mathbf{x}_n - \underbrace{\left(\nabla^2 f(\mathbf{x}_n)\right)^{-1}}_{\alpha_n} \nabla f(\mathbf{x}_n)\end{aligned}$$

- ▶ $\nabla^2 f(\mathbf{x}_n)$ is called Hessian matrix.
- ▶ **bigger** steps in low-curvature (where $\nabla^2 f(\mathbf{x}_n)$ is small)
- ▶ **smaller** steps in high-curvature scenarios.

Conjugate Gradient Descent - why need conjugate?

- ▶ we have a 2-d function $f(x_1, x_2)$:
- ▶ suppose step k occurred along x_1 -axis, and led to position \mathbf{x}^{k+1}
- ▶ at \mathbf{x}^{k+1} , $f(\mathbf{x}^{k+1})$ is minimized in its x_1 component:

$$\frac{\partial f(\mathbf{x}^{k+1})}{\partial x_1} = 0$$

- ▶ next step is along x_2 -axis: that step leads to a position \mathbf{x}^{k+2} : we find the appropriate step, such that:

$$\frac{\partial f(\mathbf{x}^{k+2})}{\partial x_2} = 0$$

- ▶ we know $\frac{\partial^2 f(\mathbf{x}^{k+2})}{\partial x_1 \partial x_2} = \frac{\partial}{\partial x_2} \left(\frac{\partial f(\mathbf{x}^{k+2})}{\partial x_1} \right)$, then:

$$\frac{\partial^2 f(\mathbf{x}^{k+2})}{\partial x_2 \partial x_1} \neq 0 \implies \frac{\partial f(\mathbf{x}^{k+2})}{\partial x_1} \neq 0$$

- ▶ in words, it says if \mathbf{x}^{k+2} is **not** overall stationery/saddle point, and we also know \mathbf{x}^{k+2} **is** stationery point in x_2 direction; then it **mustn't** be stationery point in x_1 direction
- ▶ we want to move along direction other than x_2 -axis, such that $\frac{\partial f(\mathbf{x}^{k+2})}{\partial x_1}$ remains zero

- ▶ the next four slides are heavily referenced using http://www.cs.cmu.edu/~pradeepr/convexopt/Lecture_Slides/conjugate_direction_methods.pdf
- ▶ we need to search for new non-axis directions:
- ▶ $\{d_1, d_2, \dots, d_n\}$ are said to be Q-conjugate, such that,

$$d_j^\top Q d_k = 0 \quad j \neq k$$

- ▶ when Q is also symmetric, $\{\lambda_k, v_k\}$ are eigen-(value,vector) pair, we know all eigen-vectors are orthogonal:

$$\begin{aligned} Q v_k &= \lambda_k v_k \\ \implies v_j^\top Q v_k &= \lambda_k v_j^\top v_k = 0 \quad j \neq k \end{aligned}$$

- ▶ so eigen-vectors $\{v_1, \dots, v_n\}$ of symmetric matrix can be thought as special case of Q-conjugate vectors, where these vectors are ortho-normal without Q

- ▶ let Q be **positive definite**, then all its Q -conjugate vectors $\{d_1, d_2, \dots, d_n\}$ are linearly independent
- ▶ **proof by contradiction**, i.e., suppose one of its vector say d_k can be written in linear combination of d_1, \dots, d_{k-1} :

$$\begin{aligned}d_k &= \alpha_1 d_1 + \dots + \alpha_{k-1} d_{k-1} \\ \implies d_k^\top Q d_k &= d_k^\top Q (\alpha_1 d_1 + \dots + \alpha_{k-1} d_{k-1}) \\ &= d_k^\top Q \alpha_1 d_1 + \dots + d_k^\top Q \alpha_{k-1} d_{k-1} \\ &= 0\end{aligned}$$

contradiction part is, by definition of positive definiteness: $d_k^\top Q d_k > 0 \ \forall d_k \neq 0$!

compute α_k independently

- ▶ if we are to minimize a **quadratic** problem:

$$\min_{\mathbf{x} \in \mathbb{R}^n} \frac{1}{2} \mathbf{x}^\top Q \mathbf{x} - b^\top \mathbf{x} + c$$

- ▶ if matrix $Q \in \mathbb{R}^{n \times n}$ is positive definite, then minimal value \mathbf{x}^* is:

$$Q\mathbf{x}^* = b$$

- ▶ let $\{d_0, d_1, \dots, d_{n-1}\}$ be arbitrary Q-conjugate set

$$\begin{aligned} \mathbf{x}^* &= \alpha_0 d_0 + \dots + \alpha_{n-1} d_{n-1} && \text{linearly-independent basis} \\ \Rightarrow d_k^\top Q \mathbf{x}^* &= d_k^\top Q (\alpha_0 d_0 + \dots + \alpha_{n-1} d_{n-1}) && \times \text{ by arbitrary } k^{\text{th}} \\ &= \alpha_k d_k^\top Q d_k \\ \Rightarrow \alpha_k &= \frac{d_k^\top Q \mathbf{x}^*}{d_k^\top Q d_k} = \frac{d_k^\top b}{d_k^\top Q d_k} \end{aligned}$$

- ▶ **beauty** is that we don't need to know \mathbf{x}^* to compute α_k , only Q-conjugacy is required

Conjugate Direction

$$\begin{aligned}\mathbf{x}^* &= \alpha_0 \mathbf{d}_0 + \cdots + \alpha_{n-1} \mathbf{d}_{n-1} \\ &= \sum_{k=0}^{d-1} \frac{\mathbf{d}_k^\top \mathbf{b}}{\mathbf{d}_k^\top \mathbf{Q} \mathbf{d}_k} \mathbf{d}_k \quad \text{substitute } \alpha_k = \frac{\mathbf{d}_k^\top \mathbf{b}}{\mathbf{d}_k^\top \mathbf{Q} \mathbf{d}_k}\end{aligned}$$

- ▶ the above can be achieved in parallel, where each \mathbf{d}_k does **not** minimizing anything
- ▶ also it is **not** an algorithm, it simply decomposes \mathbf{x}^*
- ▶ instead, we try to solve along a **path**, with an initial point \mathbf{x}^0 :

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{x}_0 + \alpha_0 \mathbf{d}_0 \\ &\dots \\ \mathbf{x}_k &= \mathbf{x}_0 + \alpha_0 \mathbf{d}_0 + \cdots + \alpha_{k-1} \mathbf{d}_{k-1} \\ &\dots \\ \mathbf{x}^* &= \mathbf{x}_0 + \alpha_0 \mathbf{d}_0 + \cdots + \alpha_{n-1} \mathbf{d}_{n-1}\end{aligned}$$

- ▶ what about the new α_k to match with this path?

now we have \mathbf{x}_0

- ▶ $\mathbf{x}_0 \in \mathbb{R}^n$ be an arbitrary starting point:
- ▶ so instead of writing $\mathbf{x}^* = \sum_{k=0}^{d-1} \alpha_k d_k$
- ▶ we also know $g_k \equiv \nabla f(\mathbf{x}_k) = Q\mathbf{x}_k - b = Q\mathbf{x}_k - Q\mathbf{x}^* = Q(\mathbf{x}_k - \mathbf{x}^*)$
- ▶ instead of decompose \mathbf{x}^* , let's now try to decompose $\mathbf{x}^* - \mathbf{x}_0$:

$$\mathbf{x}_1 - \mathbf{x}_0 = \underbrace{\mathbf{x}_0 + \alpha_0 d_0}_{\mathbf{x}_1} - \mathbf{x}_0$$

$$\mathbf{x}_k - \mathbf{x}_0 = \underbrace{\mathbf{x}_0 + \alpha_0 d_0 + \cdots + \alpha_k d_{k-1}}_{\mathbf{x}_k} - \mathbf{x}_0 = \alpha_0 d_0 + \cdots + \alpha_{k-1} d_{k-1}$$

$$\mathbf{x}^* - \mathbf{x}_0 = \underbrace{\mathbf{x}_0 + \alpha_0 d_0 + \cdots + \alpha_{n-1} d_{n-1}}_{\mathbf{x}^*} - \mathbf{x}_0 = \alpha_0 d_0 + \cdots + \alpha_{n-1} d_{n-1}$$

$$\implies d_k^\top Q(\mathbf{x}^* - \mathbf{x}_0) = d_k^\top Q(\alpha_0 d_0 + \cdots + \alpha_{n-1} d_{n-1})$$

$$= d_k^\top Q \alpha_k d_k$$

$$\implies \alpha_k = \frac{d_k^\top Q(\mathbf{x}^* - \mathbf{x}_0)}{d_k^\top Q d_k}$$

$$= -\frac{d_k^\top g_0}{d_k^\top Q d_k}$$

- **recap**, for $\mathbf{x}^* = \alpha_0 d_0 + \cdots + \alpha_{n-1} d_{n-1}$:

$$\mathbf{x}^* = \sum_{k=0}^{d-1} \underbrace{\frac{d_k^\top b}{d_k^\top Q d_k}}_{\alpha_k} d_k$$

- **recap**, for $\mathbf{x}^* = \alpha_0 d_0 + \cdots + \alpha_{n-1} d_{n-1} + \mathbf{x}_0$:

$$\begin{aligned} \mathbf{x}^* - \mathbf{x}_0 &= \sum_{k=0}^{d-1} \underbrace{-\frac{d_k^\top Q(\mathbf{x}^* - \mathbf{x}_0)}{d_k^\top Q d_k}}_{\alpha_k} d_k \\ \mathbf{x}^* &= \sum_{k=0}^{d-1} \underbrace{-\frac{d_k^\top Q(\mathbf{x}^* - \mathbf{x}_0)}{d_k^\top Q d_k}}_{\alpha_k} d_k + \mathbf{x}_0 \end{aligned}$$

- we will see that to write α_k in terms of $Q(\mathbf{x}^* - \mathbf{x}_0)$ may **not** be as useful as to write in terms of \mathbf{x}_k

Expanding subspace theorem

- ▶ looking at:

$$\begin{aligned}d_k^\top Q(\mathbf{x}^* - \mathbf{x}_0) &= d_k^\top Q(\mathbf{x}^* - \mathbf{x}_k + \mathbf{x}_k - \mathbf{x}_0) = d_k^\top Q(\mathbf{x}^* - \mathbf{x}_k) + d_k^\top Q(\mathbf{x}_k - Q\mathbf{x}_0) \\&= d_k^\top Q(\mathbf{x}^* - \mathbf{x}_k) + d_k^\top Q(\alpha_0 d_0 + \dots + \alpha_{n-1} d_{n-1}) \\&= d_k^\top Q(\mathbf{x}^* - \mathbf{x}_k)\end{aligned}$$

- ▶ noted that $d_k^\top Q(\mathbf{x}^* - \mathbf{x}_0) = d_k^\top Q(\mathbf{x}^* - \mathbf{x}_k) \not\Rightarrow Q(\mathbf{x}^* - \mathbf{x}_0) = Q(\mathbf{x}^* - \mathbf{x}_k)$
- ▶ think about the case:

$$\begin{bmatrix} 1 & 1 \end{bmatrix} v_1 = \begin{bmatrix} 1 & 1 \end{bmatrix} v_2 = 5 \quad \text{but} \quad v_1 = \begin{bmatrix} 4 & 1 \end{bmatrix} \text{ and } v_2 = \begin{bmatrix} 1 & 4 \end{bmatrix} \text{ satisfy}$$

- ▶ therefore:

$$\alpha_k = \frac{d_k^\top Q(\mathbf{x}^* - \mathbf{x}_0)}{d_k^\top Q d_k} = -\frac{d_k^\top g_0}{d_k^\top Q d_k} = \frac{d_k^\top Q(\mathbf{x}^* - \mathbf{x}_k)}{d_k^\top Q d_k} = -\frac{d_k^\top g_k}{d_k^\top Q d_k}$$

- ▶ **recap:** we move from \mathbf{x}_0 by adding Q-conjugate directions $\{d_1, \dots, d_n\}$, each time by

$$\alpha_k = -\frac{d_k^\top g_k}{d_k^\top Q d_k} \text{ amount}$$

- ▶ we need to prove why this movement is getting “better”, i.e., each k step **minimizes all previous directions**

Looking at the algorithm closely

- ▶ to know if \mathbf{x}_k is minimizing dimensions along its path using step size $\alpha_k = -\frac{d_k^\top g_k}{d_k^\top Q d_k}$:

$$\mathbf{x}_k \xrightarrow{\alpha_k \times d_k} \mathbf{x}_{k+1} \qquad \mathbf{x}_{k+1} \xrightarrow{\alpha_{k+1} \times d_{k+1}} \mathbf{x}_{k+2}$$

where each \mathbf{x}_k is used to compute its corresponding $g_k \equiv \nabla f(\mathbf{x}_k)$

- ▶ starting in the first step, given arbitrary point \mathbf{x}_0 :

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{x}_0 + \alpha_0 d_0 \\ g_0 &= Q\mathbf{x}_0 - b\end{aligned}$$

- ▶ obviously, we hope \mathbf{x}_1 to minimize the **line** (direction) $\mathbf{x}_0 + \alpha_0 d_0$
- ▶ this is equivalently saying, $g_1 \equiv \nabla f(\mathbf{x}_1) \perp (\mathbf{x}_0 + \alpha_0 d_0)$
- ▶ think this way, we now have changed the coordinates from one ortho-normal basis to another: $[x_1, x_2] \rightarrow [u, v]$ let:

$$(u = (\mathbf{x}_0 + \alpha_0 d_0) \quad \text{and} \quad v \perp u) \implies \left[\frac{\partial f}{\partial u}, \frac{\partial f}{\partial v} \right] = \left[0, \frac{\partial f}{\partial v} \right]$$

Looking at the algorithm closely

- ▶ we have,

$$\begin{aligned}g_1 &= \nabla f(\mathbf{x}_1) = Q\mathbf{x}_1 - b \\&= Q(\mathbf{x}_0 + \alpha_0 d_0) - b = (Q\mathbf{x}_0 - b) + \alpha_0 Qd_0 \\&= g_0 + \alpha_0 Qd_0\end{aligned}$$

- ▶ $g_1 \not\perp d_0$ in general, but we can show a particular choice α_0 makes it do, i.e., \mathbf{x}_1 minimizes the line $\mathbf{x}_0 + \alpha_0 d_0$

$$\begin{aligned}d_0^\top g_1 &= d_0^\top g_0 + d_0^\top \alpha_0 Qd_0 && \times d_0^\top \text{ on each side} \\&= d_0^\top g_0 + \alpha_0 d_0^\top Qd_0 \\&= d_0^\top g_0 - \frac{d_0^\top g_0}{d_0^\top Qd_0} d_0^\top Qd_0 && \text{sub } \alpha_0 = -\frac{d_0^\top g_0}{d_0^\top Qd_0} \\&= d_0^\top g_0 - d_0^\top g_0 = 0 \\&\implies d_0 \perp g_1\end{aligned}$$

- ▶ above shows the choice d_0 is also somewhat arbitrary
- ▶ **to understand** by choose a different \mathbf{x}_0 , results a different g_0 , having an arbitrary (g_0, d_0) pair results a unique $\alpha_0 = -\frac{d_0^\top g_0}{d_0^\top Qd_0}$ making \mathbf{x}_1 the minimum of the **line** $\mathbf{x}_0 + \alpha_0 d_0$
- ▶ however, a sensible choice is $d_0 = -\nabla f(\mathbf{x}_0) = -g_0$

Expanding Subspace Theorem

- ▶ knowing $g_1 \perp d_0$, we also can prove similarly that:

$$g_k \perp \underbrace{\text{span}(d_0, \dots, d_{k-1})}_{k \text{ terms}}$$

for example, if $\mathbf{x}_2 \perp (\mathbf{x}_0 + \alpha_0 d_0)$ and $\mathbf{x}_2 \perp (\mathbf{x}_1 + \alpha_1 d_1)$, we know that $\mathbf{x}_2 \perp$ a **surface** span of the two perpendicular lines d_0 and d_1 , we write this as:

$$g_2 \perp \underbrace{\text{span}(d_0, d_1)}_{2 \text{ terms}}$$

we can drop \mathbf{x}_0 and \mathbf{x}_1

- ▶ we can see that \mathbf{x}_k minimizes f over $\{\mathbf{x}_0 + \text{span}(d_0, \dots, d_{k-1})\}$
- ▶ therefore, it's obvious"

$$\mathbf{x}_n = \arg \min_{\mathbf{x} \in \{\mathbf{x}_0 + \text{span}(d_0, \dots, d_{n-1})\}} \frac{1}{2} \mathbf{x}^\top Q \mathbf{x} - b^\top \mathbf{x}$$

- ▶ one more thing missing, we know it works well for any arbitrary Q-conjugate vectors $\{d_0, \dots, d_n\}$:
- ▶ a sensible guess of d_1 *would be* (we already used $d_0 = -\nabla f(\mathbf{x}_0) = -g_0$):

$$d_1 = -\nabla f(\mathbf{x}_1) + \beta_0 d_0 = -g_1 + \beta_0 d_0$$

- ▶ use definition of conjugacy:

$$\begin{aligned}d_1^\top Q d_0 &= 0 \\ \implies (-g_1 + \beta_0 d_0)^\top d_0 &= 0 \\ -g_1^\top Q d_0 + \beta_0 d_0^\top Q d_0 &= 0 \\ \beta_0 &= \frac{g_1^\top Q d_0}{d_0^\top Q d_0}\end{aligned}$$

Conjugate Gradient Algorithm

1. let f be a quadratic function:

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^\top Q \mathbf{x} + \mathbf{b}^\top \mathbf{x} + c$$

2. **initialize:** Let $i = 0$ and $\mathbf{x}_i = \mathbf{x}_0$, $\mathbf{d}_i = \mathbf{d}_0 = \nabla f(\mathbf{x}_0)$
3. compute α_0 to minimize the function $f(\mathbf{x}_i + \alpha \mathbf{d}_i)$:

$$\begin{aligned} \alpha_k &= - \frac{\mathbf{d}_k^\top (Q \mathbf{x}_k + \mathbf{b})}{\mathbf{d}_k^\top Q \mathbf{d}_k} \\ &= - \frac{\mathbf{d}_k^\top \mathbf{g}_k}{\mathbf{d}_k^\top Q \mathbf{d}_k} \end{aligned}$$

4. update

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{d}_k$$

5. update the direction:

$$\beta_k = \frac{\mathbf{g}_{k+1}^\top Q \mathbf{d}_k}{\mathbf{d}_k^\top Q \mathbf{d}_k}$$

6. Repeat steps 2-4 until we have looked in n directions, where $\mathbf{x} \in \mathbb{R}^n$

A quick demo to show Stochastic Gradient Descent (1)

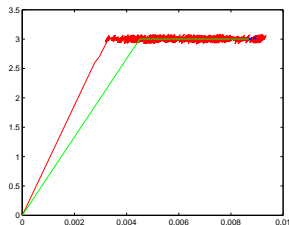
A simple example:

$$F(\theta) = \|\mathbf{x}^T \theta - \mathbf{y}\|^2 = \sum_{i=1}^N (x_i^T \theta - y_i)^2$$

$$\begin{aligned}\nabla F(\theta) &= 2\mathbf{x}^T (\mathbf{x}\theta - \mathbf{y}) \\ &\propto \mathbf{x}\theta - \mathbf{y} \\ &= \sum_{i=1}^N x_i^T \theta - y_i\end{aligned}$$

- ▶ Traditional gradient descent approach: $\theta_{n+1} = \theta_n - \alpha_n \left(\sum_{i=1}^N x_i^T \theta - y_i \right)$
- ▶ However, think about what if N is 1,000,000, which happens often in the BIG DATA era.
- ▶ Stochastic Gradient Descent HELPS!

A quick demo to illustrate Stochastic Gradient Descent (2)



Idea, instead of

$$\theta_{n+1} = \theta_n - \alpha_n \left(\sum_{i=1}^N x_i^T \theta - y_i \right)$$

Each iteration, we select randomly a data point pair (x_j, y_j) , and do:

$$\theta_{n+1} = \theta_n - \alpha_n (x_j^T \theta - y_j) \quad j \sim U(1, \dots, N)$$

It surprisingly works quite well in many settings. See demo

- The objective function:

$$E_D(\mathbf{w}) + \alpha E_W(\mathbf{w})$$

- Example:

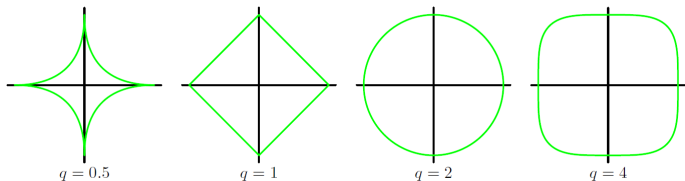
$$\frac{1}{2} \sum_{n=1}^N \left(t_n - \mathbf{w}^T \phi(\mathbf{x}_n) \right)^2 + \frac{\alpha}{2} \mathbf{w}^T \mathbf{w} \implies \mathbf{w}_{\text{ML}} = \left(\alpha \mathbf{I} + \Phi^T \Phi \right)^{-1} \Phi^T \mathbf{t}$$

- A generalised example:

$$\frac{1}{2} \sum_{n=1}^N \left(t_n - \mathbf{w}^T \phi(\mathbf{x}_n) \right)^2 + \frac{\alpha}{2} \sum_{j=1}^M |w_j|^q \implies \mathbf{w}_{\text{ML}} \text{ not so easy to obtain}$$

Diagrams of ϕ_j and struggle between $E_D(\mathbf{w})$ and $\alpha E_W(\mathbf{w})$

Plot of various norm functions: q -norm $\|\mathbf{w}\|_q := \left(\sum_{i=1}^n |w_i|^q \right)^{1/q} = 1$:



minimise $E_D(\mathbf{w}) + \alpha E_W(\mathbf{w})$ becomes the “tradeoff” between the two:

