THE UNIVERSITY *of* EDINBURGH
informatics

Applied Machine Learning (AML)

Optimisation

Oisin Mac Aodha • Siddharth N.

Optimisation

Core Questions

- What task am I trying to solve?
- How should I model the problem?
- How should I represent my data?
- How can I estimate the parameters of my model?
- How should I measure the performance of my model?

Core Questions

- What task am I trying to solve?
- How should I model the problem?
- How should I represent my data?
- How can I estimate the parameters of my model?
- How should I measure the performance of my model?

Why Optimisation?

Main idea: "learning" $\longrightarrow$ continuous optimisation

Why Optimisation?

Main idea: “learning” $\longrightarrow$ continuous optimisation

- Linear regression
- Logistic regression
- Neural networks

…

Why Optimisation?

Main idea: "learning" $\longrightarrow$ continuous optimisation

- Linear regression
- Logistic regression
- Neural networks

$\dots$

Maximum Likelihood

$$
\begin{aligned}
\ell(\boldsymbol{w}) &= \log p(\boldsymbol{x}_1, y_1, \boldsymbol{x}_2, y_2, \ldots, \boldsymbol{x}_N, y_N | \boldsymbol{w}) \\
&= \log \prod_{i=1}^{N} p(\boldsymbol{x}_i, y_i, | \boldsymbol{w}) \\
&= \sum_{i=1}^{N} \log p(\boldsymbol{x}_i, y_i, | \boldsymbol{w})
\end{aligned}
$$

E.g. $\mathrm{NLL}(\boldsymbol{w}) = -\ell(\boldsymbol{w})$

Why Optimisation?

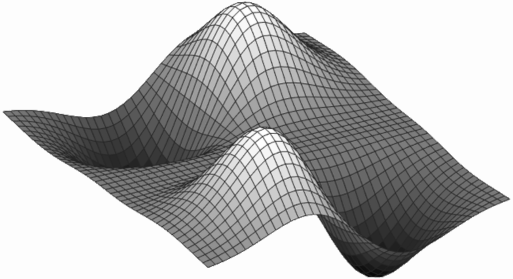
Result: An "error function" $\mathcal{L}(w)$ to minimise

Why Optimisation?

Result: An “error function” $\mathcal{L}(w)$ to minimise

Error function

- For fixed data $\mathcal{D}$, every setting of weights results in some error

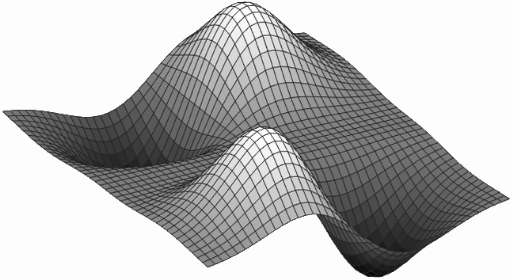


Why Optimisation?

Result: An "error function" $\mathcal{L}(w)$ to minimise

Error function

- For fixed data $\mathcal{D}$, every setting of weights results in some error
- Learning $\equiv$ descending error surface



Why Optimisation?

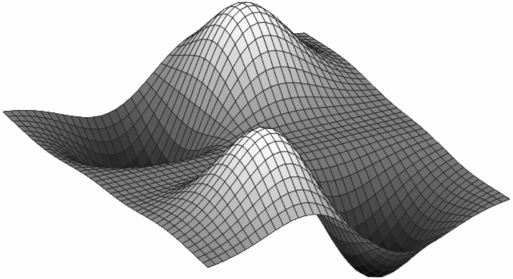
Result: An "error function" $\mathcal{L}(w)$ to minimise

Error function

- For fixed data $\mathcal{D}$, every setting of weights results in some error
- Learning ≡ descending error surface
- When data is iid

$$\mathcal{L}(w) = \sum_i \mathcal{L}^i(w)$$

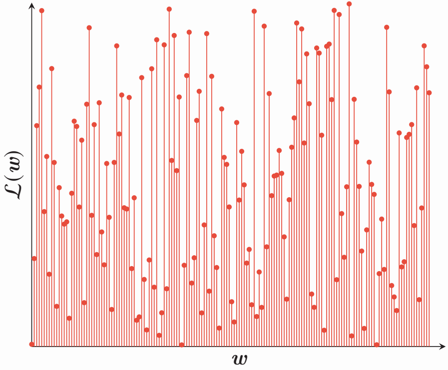
for each data point



Role of Smoothness

Unconstrained

- minimisation impossible
- …can only search through all w



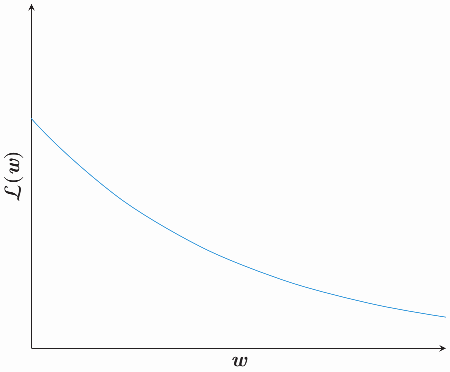
Role of Smoothness

Unconstrained

- minimisation impossible
- …can only search through all w

Constrained/Continuous

- $\mathcal{L}(w)$ provides information about $\mathcal{L}$ at nearby values



Role of Derivatives

Check: perturb w_i keeping all else the same; does error $\uparrow / \downarrow$?

Calculus

- for differentiable $\mathcal{L}$, partial derivatives $\frac{\partial \mathcal{L}}{\partial w_i}$

Role of Derivatives

Check: perturb w_i keeping all else the same; does error $\uparrow / \downarrow$?

Calculus

- for differentiable $\mathcal{L}$, partial derivatives $\frac{\partial \mathcal{L}}{\partial w_i}$
- vector of partial derivatives $\equiv$ gradient of the error

$$\nabla_{\boldsymbol{w}} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \ldots, \frac{\partial \mathcal{L}}{\partial w_N} \right)$$

Role of Derivatives

Check: perturb w_i keeping all else the same; does error $\uparrow / \downarrow$?

Calculus

- for differentiable $\mathcal{L}$, partial derivatives $\frac{\partial \mathcal{L}}{\partial w_i}$
- vector of partial derivatives $\equiv$ gradient of the error

$$\nabla_{\boldsymbol{w}} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \ldots, \frac{\partial \mathcal{L}}{\partial w_N} \right)$$

- direction of steepest error *ascent*

Role of Derivatives

Check: perturb w_i keeping all else the same; does error $\uparrow / \downarrow$?

Calculus

- for differentiable $\mathcal{L}$, partial derivatives $\frac{\partial \mathcal{L}}{\partial w_i}$
- vector of partial derivatives ≡ gradient of the error

$$\nabla_{\boldsymbol{w}} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \ldots, \frac{\partial \mathcal{L}}{\partial w_N} \right)$$

- direction of steepest error *ascent*

  ($-\nabla_{\boldsymbol{w}} \mathcal{L}$ for *descent*)

Role of Derivatives

Check: perturb w_i keeping all else the same; does error ↑ / ↓ ?

Calculus

- for differentiable $\mathcal{L}$, partial derivatives $\frac{\partial \mathcal{L}}{\partial w_i}$
- vector of partial derivatives ≡ gradient of the error

$$\nabla_{\boldsymbol{w}} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \ldots, \frac{\partial \mathcal{L}}{\partial w_N} \right)$$

- direction of steepest error *ascent*

  ($-\nabla_{\boldsymbol{w}} \mathcal{L}$ for *descent*)

Role of Derivatives

Check: perturb w_i keeping all else the same; does error $\uparrow / \downarrow$?

Calculus

- for differentiable $\mathcal{L}$, partial derivatives $\frac{\partial \mathcal{L}}{\partial w_i}$
- vector of partial derivatives ≡ gradient of the error

$$\nabla_w \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \ldots, \frac{\partial \mathcal{L}}{\partial w_N} \right)$$

- direction of steepest error *ascent*

  ($-\nabla_w \mathcal{L}$ for *descent*)

Key Challenges

- Computing $\nabla_w \mathcal{L}$ efficiently

Role of Derivatives

Check: perturb w_i keeping all else the same; does error $\uparrow / \downarrow$?

Calculus

- for differentiable $\mathcal{L}$, partial derivatives $\frac{\partial \mathcal{L}}{\partial w_i}$
- vector of partial derivatives $\equiv$ gradient of the error

$$\nabla_{w}\mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \ldots, \frac{\partial \mathcal{L}}{\partial w_N}\right)$$

- direction of steepest error *ascent*

  ($-\nabla_{w}\mathcal{L}$ for *descent*)

Key Challenges

- Computing $\nabla_{w}\mathcal{L}$ efficiently
- Minimising error with gradient

Role of Derivatives

Check: perturb w_i keeping all else the same; does error $\uparrow$ / $\downarrow$?

Calculus

- for differentiable $\mathcal{L}$, partial derivatives $\frac{\partial \mathcal{L}}{\partial w_i}$
- vector of partial derivatives ≡ gradient of the error

$$\nabla_w \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \ldots, \frac{\partial \mathcal{L}}{\partial w_N} \right)$$

- direction of steepest error *ascent*

  ($-\nabla_w \mathcal{L}$ for *descent*)

Key Challenges

- Computing $\nabla_w \mathcal{L}$ efficiently
- Minimising error with gradient
- Location of minimiser

Optimisation

General Optimisation Problem

Optimisation Algorithms

$$\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$$

Optimisation Algorithms

$$\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$$

Components

- procedure to compute $\mathcal{L}(\boldsymbol{w})$
- procedure to compute $\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$

Optimisation Algorithms

$$\min_{w} \mathcal{L}(w)$$

Components

- procedure to compute $\mathcal{L}(w)$
- procedure to compute $\nabla_w \mathcal{L}(w)$

Aside

- some others don't use gradients
- some use higher-order gradients

…not covered here

Basic Optimisation Algorithm

```
Require: stopping error ϵ, step size η
  w ← initialisation
  while L(w) > ϵ do
      calculate g = ∇_w L(w)
      compute direction d from w, L(w), g
      w ← w − η d
  return w
```

Require: stopping error ϵ, step size η

- $\boldsymbol{w} \leftarrow$ initialisation
- **while** $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**
  - calculate $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$
  - compute direction $\boldsymbol{d}$ from $\boldsymbol{w}$, $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{g}$
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{d}$
- **return** $\boldsymbol{w}$

Basic Optimisation Algorithm

Require: stopping error ϵ, step size η

$\boldsymbol{w}_0,$

$\boldsymbol{w} \leftarrow$ initialisation
while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**
    calculate $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$
    compute direction $\boldsymbol{d}$ from $\boldsymbol{w}$, $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{g}$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{d}$
return $\boldsymbol{w}$

Basic Optimisation Algorithm

Require: stopping error ϵ, step size η

$\boldsymbol{w} \leftarrow$ initialisation

while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**

  calculate $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$

  compute direction $\boldsymbol{d}$ from $\boldsymbol{w}$, $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{g}$

  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{d}$

return $\boldsymbol{w}$

$\boldsymbol{w}_0$,

$\mathcal{L}(\boldsymbol{w}_0)$,

$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_0)$,

Basic Optimisation Algorithm

Require: stopping error ϵ, step size η

$\boldsymbol{w} \leftarrow$ initialisation
while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**
    calculate $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$
    compute direction $\boldsymbol{d}$ from $\boldsymbol{w}$, $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{g}$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{d}$
return $\boldsymbol{w}$

$\boldsymbol{w}_0,$ $\boldsymbol{w}_1,$

$\mathcal{L}(\boldsymbol{w}_0),$

$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_0),$

Basic Optimisation Algorithm

Require: stopping error ϵ, step size η

$\boldsymbol{w} \leftarrow$ initialisation

while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**

  calculate $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$

  compute direction $\boldsymbol{d}$ from $\boldsymbol{w}$, $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{g}$

  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{d}$

return $\boldsymbol{w}$

$$\begin{array}{cc} \boldsymbol{w}_0, & \boldsymbol{w}_1, \\ \mathcal{L}(\boldsymbol{w}_0), & \mathcal{L}(\boldsymbol{w}_1), \\ \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_0), & \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_1), \end{array}$$

Basic Optimisation Algorithm

Require: stopping error ϵ, step size η

$\boldsymbol{w} \leftarrow$ initialisation
while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**
    calculate $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$
    compute direction $\boldsymbol{d}$ from $\boldsymbol{w}$, $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{g}$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{d}$
return $\boldsymbol{w}$

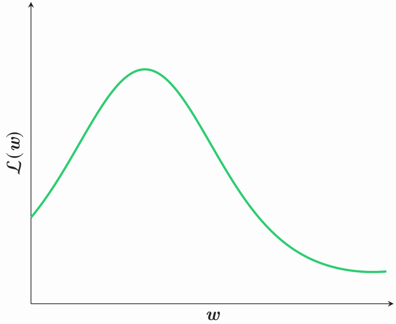
$$\begin{array}{cccc} \boldsymbol{w}_0, & \boldsymbol{w}_1, & \boldsymbol{w}_2, & \ldots \\ \mathcal{L}(\boldsymbol{w}_0), & \mathcal{L}(\boldsymbol{w}_1), & \mathcal{L}(\boldsymbol{w}_2), & \ldots \\ \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_0), & \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_1), & \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_2), & \ldots \end{array}$$

Choosing a direction

Simple choice: $d = \nabla_w \mathcal{L}$ locally steepest direction

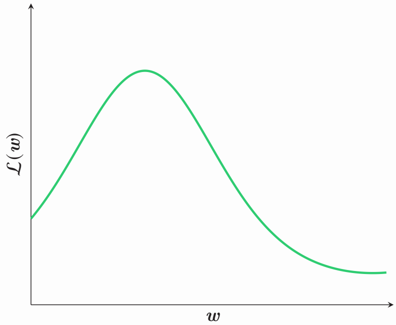
Choosing a direction

Simple choice: $d = \nabla_w \mathcal{L}$ locally steepest direction



Choosing a direction

Simple choice: $d = \nabla_w \mathcal{L}$ locally steepest direction



Recall (multi-variate) Taylor theorem:
for $w \in \mathbb{R}^N$ and perturbation $\delta \in \mathbb{R}^N$ such that $a = w - \delta$

$$\mathcal{L}(w) \approx \mathcal{L}(a) + \nabla_w \mathcal{L}(a)^\top \delta + \frac{1}{2}\delta^\top \nabla_w^2 \mathcal{L}(a)\delta + \ldots$$

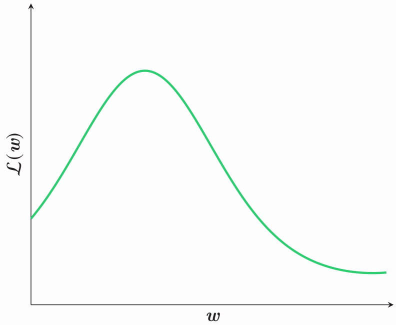
$$\approx \mathcal{L}(a) + \nabla_w \mathcal{L}(a)^\top \delta$$

(dropping higher order terms for small δ)

$$\therefore \mathcal{L}(a + \delta) = \mathcal{L}(a) + \nabla_w \mathcal{L}(a)^\top \delta$$

Choosing a direction

Simple choice: $d = \nabla_w \mathcal{L}$ locally steepest direction



Recall (multi-variate) Taylor theorem:
for $w \in \mathbb{R}^N$ and perturbation $\delta \in \mathbb{R}^N$ such that $a = w - \delta$

$$\mathcal{L}(w) \approx \mathcal{L}(a) + \nabla_w \mathcal{L}(a)^\top \delta + \frac{1}{2}\delta^\top \nabla_w^2 \mathcal{L}(a) \delta + \ldots$$

$$\approx \mathcal{L}(a) + \nabla_w \mathcal{L}(a)^\top \delta$$

(dropping higher order terms for small δ)

$$\therefore \mathcal{L}(a + \delta) = \mathcal{L}(a) + \nabla_w \mathcal{L}(a)^\top \delta$$

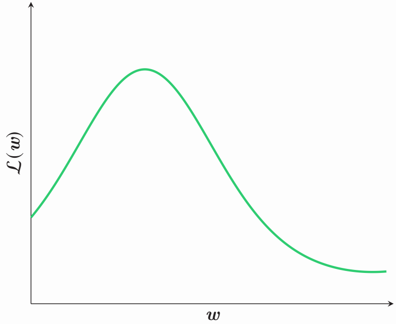
which is minimised at $\delta = -\eta \nabla_w \mathcal{L}(a)$ as

$$\mathcal{L}(a - \eta \nabla_w \mathcal{L}(a)) = \mathcal{L}(a) - \eta \|\nabla_w \mathcal{L}(a)\|^2$$

$$\implies \mathcal{L}(a - \eta \nabla_w \mathcal{L}(a)) \leq \mathcal{L}(a) \qquad \text{(for } \eta > 0\text{)}$$

Choosing a direction

Simple choice: $d = \nabla_w \mathcal{L}$ locally steepest direction



Recall (multi-variate) Taylor theorem:
for $w \in \mathbb{R}^N$ and perturbation $\delta \in \mathbb{R}^N$ such that $a = w - \delta$

$$\mathcal{L}(w) \approx \mathcal{L}(a) + \nabla_w \mathcal{L}(a)^\top \delta + \frac{1}{2}\delta^\top \nabla_w^2 \mathcal{L}(a)\delta + \dots$$

$$\approx \mathcal{L}(a) + \nabla_w \mathcal{L}(a)^\top \delta$$

(dropping higher order terms for small δ)

$$\therefore \mathcal{L}(a + \delta) = \mathcal{L}(a) + \nabla_w \mathcal{L}(a)^\top \delta$$

which is minimised at $\delta = -\eta \nabla_w \mathcal{L}(a)$ as

$$\mathcal{L}(a - \eta \nabla_w \mathcal{L}(a)) = \mathcal{L}(a) - \eta \|\nabla_w \mathcal{L}(a)\|^2$$

$$\implies \mathcal{L}(a - \eta \nabla_w \mathcal{L}(a)) \leq \mathcal{L}(a) \qquad \text{(for } \eta > 0\text{)}$$

Taking a step along δ cannot increase value “locally”

Gradient Descent

Gradient Descent

Gradient Descent

Generic Optimisation

Require: stopping error ϵ, step size η

$\boldsymbol{w} \leftarrow$ initialisation

while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**

  calculate $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$

  compute direction $\boldsymbol{d}$ from $\boldsymbol{w}$, $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{g}$

  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{d}$

return $\boldsymbol{w}$

Gradient Descent

Generic Optimisation

Require: stopping error ϵ, step size η

$\boldsymbol{w} \leftarrow$ initialisation
while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**
    calculate $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$
    compute direction $\boldsymbol{d}$ from $\boldsymbol{w}$, $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{g}$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{d}$
return $\boldsymbol{w}$

Gradient Descent

Require: stopping error ϵ, step size η

$\boldsymbol{w} \leftarrow$ initialisation
while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**
    calculate $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$
    compute direction $\boldsymbol{d} = \boldsymbol{g}$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{g}$
return $\boldsymbol{w}$

Gradient Descent

Generic Optimisation

Require: stopping error ϵ, step size η

$\boldsymbol{w} \leftarrow$ initialisation
while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**
  calculate $\boldsymbol{g} = \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})$
  compute direction $\boldsymbol{d}$ from $\boldsymbol{w}$, $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{g}$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\boldsymbol{d}$
return $\boldsymbol{w}$

Gradient Descent

Require: stopping error ϵ, step size η

$\boldsymbol{w} \leftarrow$ initialisation
while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**
  calculate $\boldsymbol{g} = \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})$
  compute direction $\boldsymbol{d} = \boldsymbol{g}$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\boldsymbol{g}$
return $\boldsymbol{w}$

Effect of Step Size (η)

Step Size (η)

Sometimes called *learning rate*

Effect of Step Size (η)

Step Size (η)

Sometimes called *learning rate*

- $\eta > 0$

Effect of Step Size (η)

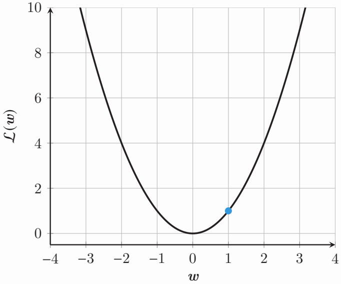
Step Size (η)

Sometimes called *learning rate*

- $\eta > 0$
- η too small $\longrightarrow$ too slow

Example: Minimise $\mathcal{L}(w) = w^2$

Take $\eta = 0.1$



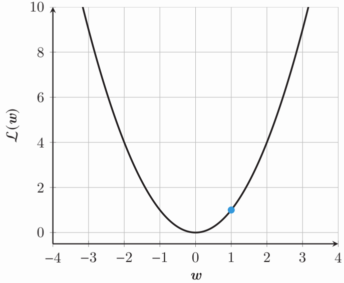
Effect of Step Size (η)

Step Size (η)

Sometimes called *learning rate*

- $\eta > 0$
- η too small $\longrightarrow$ too slow

Example: Minimise $\mathcal{L}(w) = w^2$



Take $\eta = 0.1$

$$w_0 = 1.0$$
$$w_1 = w_0 - 0.1 \cdot 2w_0 = 0.8$$
$$w_2 = w_1 - 0.1 \cdot 2w_0 = 0.64$$
$$\vdots$$
$$w_2 = w_1 - 0.1 \cdot 2w_0 = 0.512$$
$$w_{25} = 0.0047$$

THE UNIVERSITY of EDINBURGH informatics

Effect of Step Size (η)

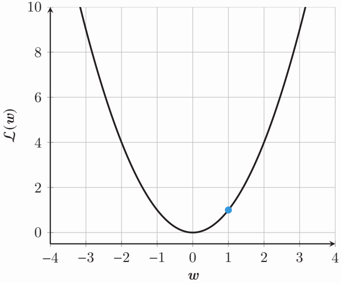
Step Size (η)

Sometimes called *learning rate*

- $\eta > 0$
- η too small $\longrightarrow$ too slow
- η too large $\longrightarrow$ instability

Example: Minimise $\mathcal{L}(w) = w^2$

Take $\eta = 1.1$



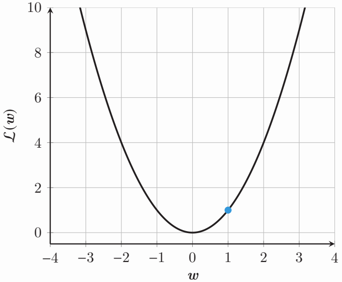
Effect of Step Size (η)

Step Size (η)

Sometimes called *learning rate*

- $\eta > 0$
- η too small $\longrightarrow$ too slow
- η too large $\longrightarrow$ instability

Example: Minimise $\mathcal{L}(w) = w^2$



Take $\eta = 1.1$

$$w_0 = 1.0$$
$$w_1 = w_0 - 1.1 \cdot 2w_0 = -1.2$$
$$w_2 = w_1 - 1.1 \cdot 2w_0 = 1.44$$
$$\vdots$$
$$w_2 = w_1 - 1.1 \cdot 2w_0 = -1.72$$
$$w_{25} = 79.50$$

Heuristic for step size (η)

```
Require: stopping error ϵ, step size η
w ← initialisation
while L(w) > ϵ do
    compute g = ∇_w L(w)
    compute direction d from w, L(w), g

    w ← w − η d

return w
```

Require: stopping error ϵ, step size η

$\boldsymbol{w} \leftarrow$ initialisation

while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**

compute $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$

compute direction $\boldsymbol{d}$ from $\boldsymbol{w}$, $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{g}$

$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{d}$

return $\boldsymbol{w}$

Heuristic for step size (η)

Require: stopping error ϵ, step size η

$\boldsymbol{w} \leftarrow$ initialisation
while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**
    compute $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$
    compute direction $\boldsymbol{d}$ from $\boldsymbol{w}$, $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{g}$
    $\ell_{-} = \mathcal{L}(\boldsymbol{w})$ ▹ error before update
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{d}$
return $\boldsymbol{w}$

Heuristic for step size (η)

Require: stopping error ϵ, step size η

```
w ← initialisation
while L(w) > ε do
    compute g = ∇_w L(w)
    compute direction d from w, L(w), g
    ℓ_- = L(w)                              ▷ error before update
    w ← w − η d
    ℓ_+ = L(w)                              ▷ error after update

return w
```

Heuristic for step size (η)

Require: stopping error ϵ, step size η

$w \leftarrow$ initialisation
while $\mathcal{L}(w) > \epsilon$ **do**
  compute $g = \nabla_w \mathcal{L}(w)$
  compute direction d from w, $\mathcal{L}(w)$, g
  $\ell_- = \mathcal{L}(w)$ ▹ error before update
  $w \leftarrow w - \eta d$
  $\ell_+ = \mathcal{L}(w)$ ▹ error after update
  if $\ell_+ \geq \ell_-$ **then** ▹ if error increases

  else ▹ if error decreases

return w

Heuristic for step size (η)

Require: stopping error ϵ, step size η

$\boldsymbol{w} \leftarrow$ initialisation
while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**
    compute $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$
    compute direction $\boldsymbol{d}$ from $\boldsymbol{w}$, $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{g}$
    $\ell_- = \mathcal{L}(\boldsymbol{w})$ ▹ error before update
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{d}$
    $\ell_+ = \mathcal{L}(\boldsymbol{w})$ ▹ error after update
    if $\ell_+ \geq \ell_-$ **then** ▹ if error increases
        $\eta \leftarrow \eta/2$; **revert** $\boldsymbol{w}$ ▹ reduce step size
    else ▹ if error decreases

return $\boldsymbol{w}$

Heuristic for step size (η)

Require: stopping error ϵ, step size η

$\boldsymbol{w} \leftarrow$ initialisation
while $\mathcal{L}(\boldsymbol{w}) > \epsilon$ **do**
  compute $\boldsymbol{g} = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$
  compute direction $\boldsymbol{d}$ from $\boldsymbol{w}$, $\mathcal{L}(\boldsymbol{w})$, $\boldsymbol{g}$
  $\ell_- = \mathcal{L}(\boldsymbol{w})$ ▹ error before update
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{d}$
  $\ell_+ = \mathcal{L}(\boldsymbol{w})$ ▹ error after update
  if $\ell_+ \geq \ell_-$ **then** ▹ if error increases
    $\eta \leftarrow \eta/2$; **revert** $\boldsymbol{w}$ ▹ reduce step size
  else ▹ if error decreases
    $\eta \leftarrow 1.1\eta$ ▹ speed up *slightly*
return $\boldsymbol{w}$

Gradient Descent

Stochastic Gradient Descent

Gradient Computation

Recall that gradient $\nabla_w \mathcal{L}(w)$ is computed over (iid) data $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$.

Gradient Computation

Recall that gradient $\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})$ is computed over (iid) data $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$.

$$\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) = \nabla_{\boldsymbol{w}}\left[-\frac{1}{N}\sum_{i=1}^{N}\log p(y_i|\boldsymbol{x}_i, \boldsymbol{w})\right] \qquad \text{(e.g. logistic regression)}$$

$$= -\frac{1}{N}\sum_{i=1}^{N}\nabla_{\boldsymbol{w}}\log p(y_i|\boldsymbol{x}_i, \boldsymbol{w}) = -\frac{1}{N}\sum_{i=1}^{N}\nabla_{\boldsymbol{w}}\mathcal{L}^i(\boldsymbol{w})$$

Gradient Computation

Recall that gradient $\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})$ is computed over (iid) data $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$.

$$\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) = \nabla_{\boldsymbol{w}}\left[-\frac{1}{N}\sum_{i=1}^{N}\log p(y_i|\boldsymbol{x}_i, \boldsymbol{w})\right] \qquad \text{(e.g. logistic regression)}$$

$$= -\frac{1}{N}\sum_{i=1}^{N}\nabla_{\boldsymbol{w}}\log p(y_i|\boldsymbol{x}_i, \boldsymbol{w}) = -\frac{1}{N}\sum_{i=1}^{N}\nabla_{\boldsymbol{w}}\mathcal{L}^i(\boldsymbol{w})$$

Challenge

- Estimation requires evaluating gradients at *all* N data points

Gradient Computation

Recall that gradient $\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$ is computed over (iid) data $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$.

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \left[-\frac{1}{N} \sum_{i=1}^{N} \log p(y_i | \boldsymbol{x}_i, \boldsymbol{w}) \right] \qquad \text{(e.g. logistic regression)}$$

$$= -\frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{w}} \log p(y_i | \boldsymbol{x}_i, \boldsymbol{w}) = -\frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{w}} \mathcal{L}^i(\boldsymbol{w})$$

Challenge

- Estimation requires evaluating gradients at *all* N data points
- Fine if N is small, but if N is large? Say millions?

Gradient Computation

Recall that gradient $\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})$ is computed over (iid) data $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$.

$$\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) = \nabla_{\boldsymbol{w}}\left[-\frac{1}{N}\sum_{i=1}^{N}\log p(y_i|\boldsymbol{x}_i, \boldsymbol{w})\right] \qquad \text{(e.g. logistic regression)}$$

$$= -\frac{1}{N}\sum_{i=1}^{N}\nabla_{\boldsymbol{w}}\log p(y_i|\boldsymbol{x}_i, \boldsymbol{w}) = -\frac{1}{N}\sum_{i=1}^{N}\nabla_{\boldsymbol{w}}\mathcal{L}^i(\boldsymbol{w})$$

Challenge

- Estimation requires evaluating gradients at *all* N data points
- Fine if N is small, but if N is large? Say millions?
- Can we get a “good enough” gradient from fewer data points? Maybe one!?

Stochastic Gradient Descent

Idea: Compute update for parameter with just a single instance

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \cdot \nabla_{\boldsymbol{w}} \mathcal{L}^{i}(\boldsymbol{w})$$

Stochastic Gradient Descent

Idea: Compute update for parameter with just a single instance

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \cdot \nabla_{\boldsymbol{w}} \mathcal{L}^{i}(\boldsymbol{w})$$

Indexing

- Choose randomly for $i \in \{1, \ldots, N\}$
- $\mathbb{E}\left[\nabla_{\boldsymbol{w}} \mathcal{L}^{i}(\boldsymbol{w})\right] = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$
- provides an unbiased estimate of the gradient at each step

Stochastic Gradient Descent

Idea: Compute update for parameter with just a single instance

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \cdot \nabla_{\boldsymbol{w}} \mathcal{L}^{i}(\boldsymbol{w})$$

Indexing

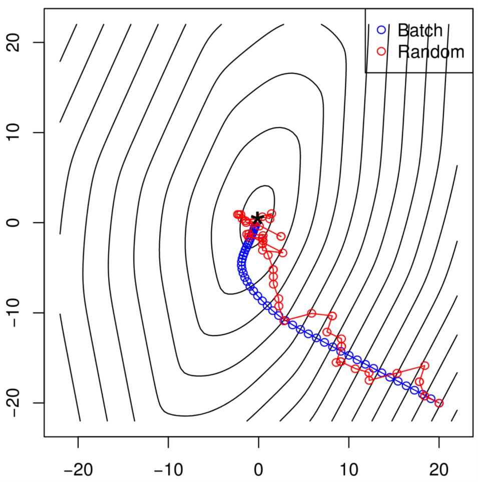
- Choose randomly for $i \in \{1, \ldots, N\}$
- $\mathbb{E}\left[\nabla_{\boldsymbol{w}} \mathcal{L}^{i}(\boldsymbol{w})\right] = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$
- provides an unbiased estimate of the gradient at each step

Features

- Cost per iteration independent of N
- Potential savings in memory usage
- Can be noisy in practice

Stochastic Gradient Descent — Example

Example with $N = 10$, $D = 2$ comparing standard versus stochastic gradient descent



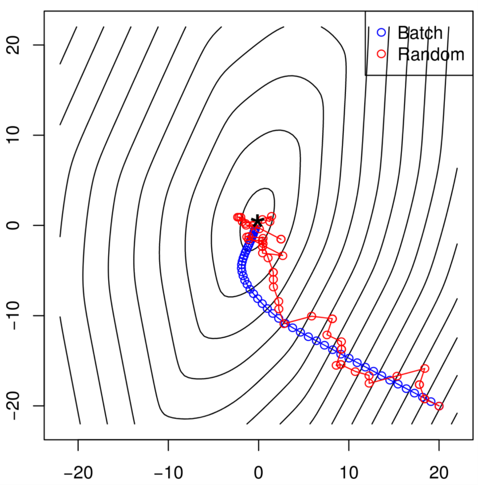
Blue standard steps $O(N \times D)$

Red stochastic steps $O(D)$

Figures: Ryan Tibshirani - Convex Optimisation

Stochastic Gradient Descent — Example

Example with $N = 10$, $D = 2$ comparing standard versus stochastic gradient descent



Blue standard steps $\mathcal{O}(N \times D)$

Red stochastic steps $\mathcal{O}(D)$

Characteristics

- work well far from optimum
- struggle close to optimum

Figures: Ryan Tibshirani - Convex Optimisation

Stochastic Gradient Descent — Mini-batches

Idea: Compute update for parameter with a *few* random instances chosen as $I \subseteq \{1, \ldots, N\}$, such that $|I| = B$, $B \ll N$.

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \cdot \frac{1}{B} \sum_{i \in I} \nabla_{\boldsymbol{w}} \mathcal{L}^i(\boldsymbol{w})$$

Stochastic Gradient Descent — Mini-batches

Idea: Compute update for parameter with a *few* random instances chosen as $I \subseteq \{1, \ldots, N\}$, such that $|I| = B$, $B \ll N$.

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \cdot \frac{1}{B} \sum_{i \in I} \nabla_{\boldsymbol{w}} \mathcal{L}^i(\boldsymbol{w})$$

Again, we are approximating the full gradient using an unbiased estimate

$$\mathbb{E}\left[\frac{1}{B} \sum_{i \in I} \nabla_{\boldsymbol{w}} \mathcal{L}^i(\boldsymbol{w})\right] = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$$

Stochastic Gradient Descent — Mini-batches

Idea: Compute update for parameter with a *few* random instances chosen as $I \subseteq \{1, \ldots, N\}$, such that $|I| = B$, $B \ll N$.

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \cdot \frac{1}{B} \sum_{i \in I} \nabla_{\boldsymbol{w}} \mathcal{L}^i(\boldsymbol{w})$$

Again, we are approximating the full gradient using an unbiased estimate

$$\mathbb{E}\left[\frac{1}{B} \sum_{i \in I} \nabla_{\boldsymbol{w}} \mathcal{L}^i(\boldsymbol{w})\right] = \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$$

Features

- reduces the variance of the gradient estimator by $1/B$
- also B times more expensive to compute $\mathcal{O}(B \times D)$

Gradient Descent

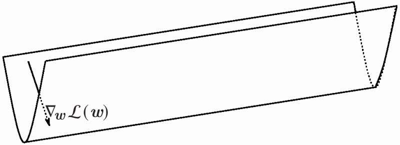
Problems

Problems With Gradient Descent

- Setting the step size η
- Shallow valleys
- Surface curvature
- Local minima

Problems With Gradient Descent

- Setting the step size η
- Shallow valleys
- Surface curvature
- Local minima



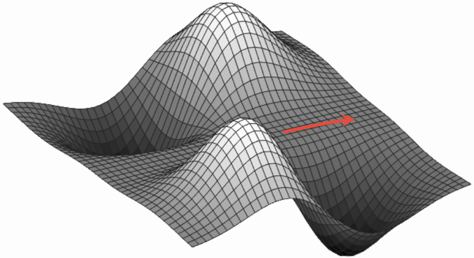
- Gradient descent slows down in shallow valleys
- Incorporate *momentum*

$$\boldsymbol{d} \leftarrow \beta \boldsymbol{d} + (1 - \beta)\eta \cdot \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$$

- Have to choose both η and β

Problems With Gradient Descent

- Setting the step size η
- Shallow valleys
- Surface curvature
- Local minima

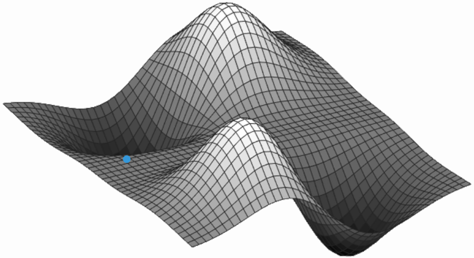


- Gradient need not point towards optimum!

  Note: *locally* steepest direction
- Local curvature is measured by the Hessian: $H = \nabla_{w}^{2} \mathcal{L}(w)$

Problems With Gradient Descent

- Setting the step size η
- Shallow valleys
- Surface curvature
- Local minima



- Gradient at *any* minimum is 0!
- *Convex* functions: local minimum $\equiv$ global minimum

  e.g. squared error, logistic regression likelihood, …
- No standard solution

  best to rerun optimiser from different initialisations

Summary

- Optimisation is a complex field

Summary

- Optimisation is a complex field
  - How and why we convert learning problems into optimization problems

Summary

- Optimisation is a complex field
  - How and why we convert learning problems into optimization problems
  - Gradient Descent / Stochastic Gradient Descent

Summary

- Optimisation is a complex field
  - How and why we convert learning problems into optimization problems
  - Gradient Descent / Stochastic Gradient Descent
  - Issues with Gradient Descent

Summary

- Optimisation is a complex field
    - How and why we convert learning problems into optimization problems
    - Gradient Descent / Stochastic Gradient Descent
    - Issues with Gradient Descent
- Many variants providing better stability and convergence

  E.g. momentum, acceleration, averaging, variance reduction, …

Summary

- Optimisation is a complex field
  - How and why we convert learning problems into optimization problems
  - Gradient Descent / Stochastic Gradient Descent
  - Issues with Gradient Descent
- Many variants providing better stability and convergence
  E.g. momentum, acceleration, averaging, variance reduction, …
- See AdaGrad, Adam, AdaMax, … and many more!