Machine Learning 2.02: Hyperparameter Optimisation

Tom S. F. Haines
T.S.F.Haines@bath.ac.uk

Optimisation

- Optimisation of ML model

$$\underset{\theta}{\operatorname{argmin}} \left(f(\theta) \right)$$

where

- θ – Parameters, e.g. weights of linear regression
- $f(\theta)$ – Error between model and *training set*, given parameters θ

No gradient

- Hyperparameter optimisation:

$$\underset{\Theta}{\operatorname{argmin}}\,(F(\Theta))$$

where

- Θ – Hyperparameters, e.g. maximum tree depth of a decision tree
- $F(\Theta)$ – Error on a *validation set*, having trained the model with θ

No gradient

- Hyperparameter optimisation:

$$\underset{\Theta}{\operatorname{argmin}}\,(F(\Theta))$$

  where
  - Θ – Hyperparameters, e.g. maximum tree depth of a decision tree
  - $F(\Theta)$ – Error on a *validation set*, having trained the model with θ

- "Just" optimisation, but
  - No gradient
  - Slow (train model for every evaluation – sometimes *days*)
  - Continuous and discrete parameters

UNIVERSITY OF BATH

Everything

- This lecture:
  - Optimisation without a gradient
  - Optimisation when evaluation is slow
  - Optimisation with both discrete and continuous values

- These approaches work for **every** optimisation problem...
  ...but are **worse** than a specialist approach,
  e.g. one that uses the gradient (in which case use gradient descent, or better)

UNIVERSITY OF BATH

Meat sack optimisation

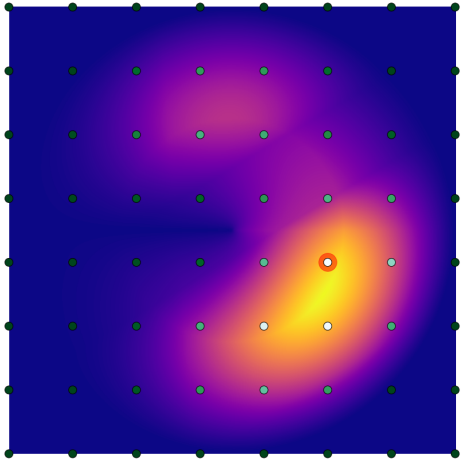
In research: "Optimisation by grad student"

- Meat sack = you!
  – trying values manually and seeing what works
    (noting results in text file/spreadsheet)
- Popular, and yet
  - Wastes your time
  - Boring
  - Computer will outperform you
- Avoid!

UNIVERSITY OF BATH

Grid search

- Exactly as it sounds!
  (nested loop)
- Explodes in high dimensions
  e.g. 8 per dimension
  - 1D $\implies$ 8
  - 2D $\implies$ $8^2 = 64$
  - 3D $\implies$ $8^3 = 512$
  - 4D $\implies$ $8^4 = 4096$
  - $\vdots$
- Also called *gridding*
- Used in decision tree for splits

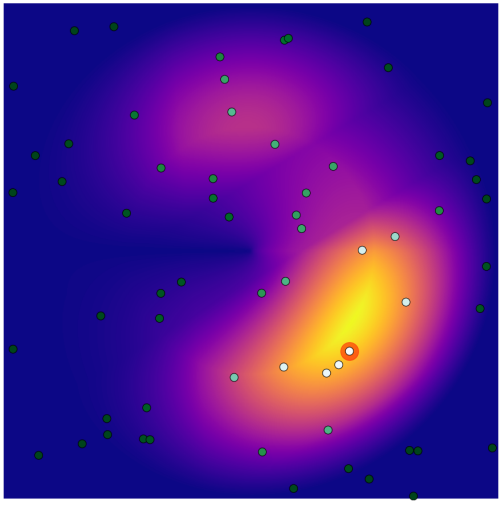


UNIVERSITY OF BATH

Random search

- Try many random parameters
- Choose a distribution for each (often uniform)

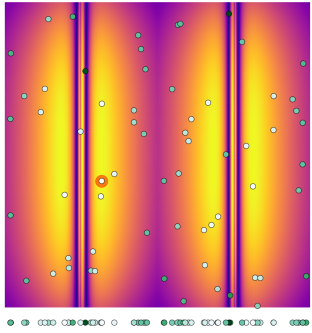
- Works in high dimensions
- Can bias search with distribution



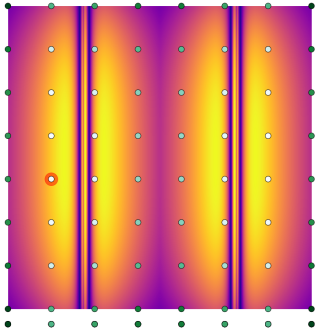
UNIVERSITY OF BATH

Be random

- Random better than gridding
- Dimensions often (locally) irrelevant ∴ gridding wastes computation
- Dense gridding with few dimensions ok



99.7% of maximum

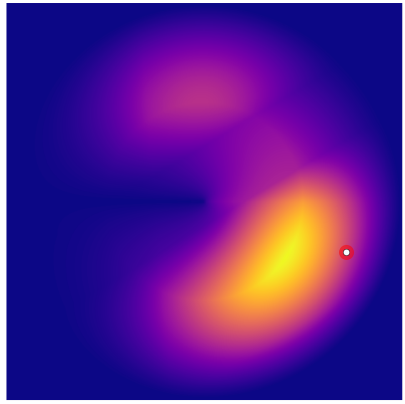


90.5% of maximum

Hill climbing

- Gradient ascent without knowing gradient!
- Try random steps, accept if improvement
- Random step typically offset by Gaussian draw
  (hyperparameter optimiser now has a parameter!)

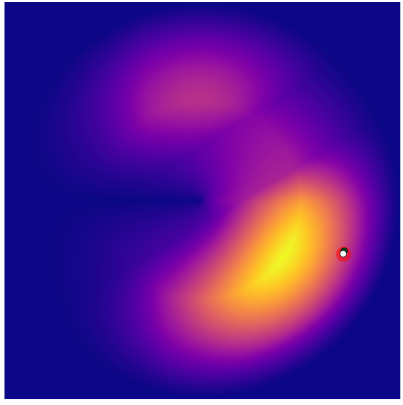
Random initialisation



Hill climbing

- Gradient ascent without knowing gradient!
- Try random steps, accept if improvement
- Random step typically offset by Gaussian draw
  (hyperparameter optimiser now has a parameter!)

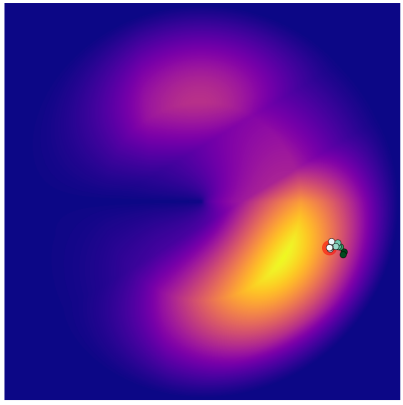
Three random steps before first accept



Hill climbing

- Gradient ascent without knowing gradient!
- Try random steps, accept if improvement
- Random step typically offset by Gaussian draw
  (hyperparameter optimiser now has a parameter!)

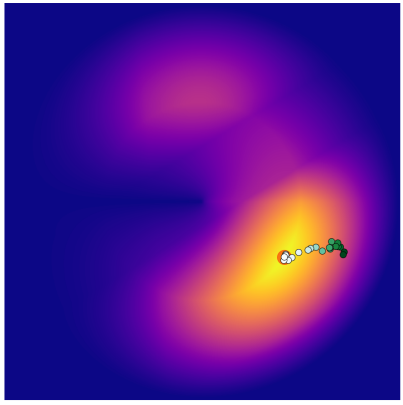
20 random steps for 10 accepts



Hill climbing

- Gradient ascent without knowing gradient!
- Try random steps, accept if improvement
- Random step typically offset by Gaussian draw
  (hyperparameter optimiser now has a parameter!)

62 random steps for 21 accepts

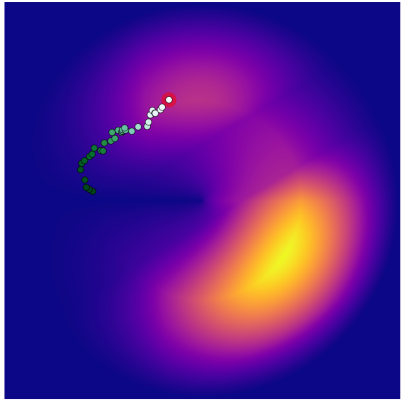


UNIVERSITY OF BATH

Shotgun hill climbing

- Same failure as gradient ascent (/descent)

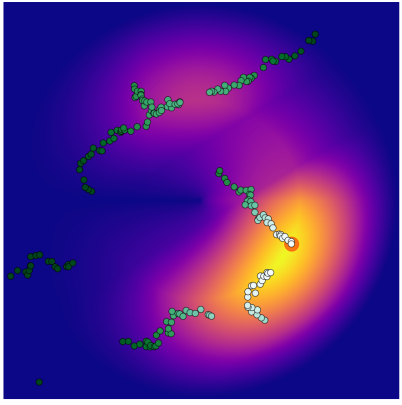
Unfortunate start



Shotgun hill climbing

- Same failure as gradient ascent (/descent)
- Random restarts!
  – Random search / hill climbing hybrid

8 restarts



UNIVERSITY OF BATH

Genetic algorithms I

- Motivation:
  - Survival of the fittest!
  - Strongest survive and breed!
  - Next generation stronger!
  - Evolution!

Genetic algorithms II

Silly language:

- Collection of “genes”
- Each gene has a “fitness”
- They “mutate”
- They “breed”
- Survival of the fittest

UNIVERSITY OF BATH

Genetic algorithms II

Silly language:

- Collection of "genes" $\Rightarrow$
- Each gene has a "fitness" $\Rightarrow$
- They "mutate" $\Rightarrow$
- They "breed" $\Rightarrow$
- Survival of the fittest $\Rightarrow$

Sensible language:

- Gene = Parameters (Θ)
- Fitness = Function being optimised ($F(\Theta)$)
- Mutate = Hill climbing
- Breeding = Something new
- Keeping best parameters

Genetic algorithms III

- Steps:
  1. Initialise
  2. Repeat until convergence:
     - 2.1 Crossover (more common name for breeding)
     - 2.2 Mutation
     - 2.3 Selection

- Will cover 2 backwards...

UNIVERSITY OF BATH

GA 1: Initialise

- Choose population size, n
  (typically 100+; memory is cheap)
- Define distribution over parameters, P
  (same as for random search, often uniform)
- Draw $\Theta_i \sim P$ for $i \in \{1, 2, \ldots, n\}$

GA 2.3: Selection

- Each iteration:
  1. Generate m new parameter vectors (Θ_i) using crossover and mutation
  2. Cull $n + m$ back to n

- Options:
  - Keep n with highest $F(\Theta_i)$
  - Draw without replacement with probability proportional to $F(\Theta_i)$
    (usually hard code keeping best; often better)

UNIVERSITY OF BATH

GA 2.2: Mutation

- Hill climbing!
- Add some noise, approaches:
  - Offset with Gaussian draw
  - Select random parameter and replace (draw from initialisation distribution)
- Normal to only mutate subset

UNIVERSITY OF BATH

GA 2.1: Crossover

- Select two parameter vectors (Θ_i, Θ_j) at random
  (uniform probability or proportional to $F(\Theta_i)$)
- Combine them somehow!
  (domain dependent)
- Approaches:
  - Per-parameter random selection (most common)

$$\Theta_{\text{new}}[p] = (\Theta_i[p])^{1-z_p} + (\Theta_j[p])^{z_p}, \quad z_p \sim \text{Bernoulli}\left(\frac{1}{2}\right)$$

  - Interpolate (continuous only)

$$\Theta_{\text{new}} = (1-t)\Theta_i + t\Theta_j, \quad t \sim \text{Uniform}()$$

  - Some combination of above
- Searches space between local maxima

UNIVERSITY OF BATH

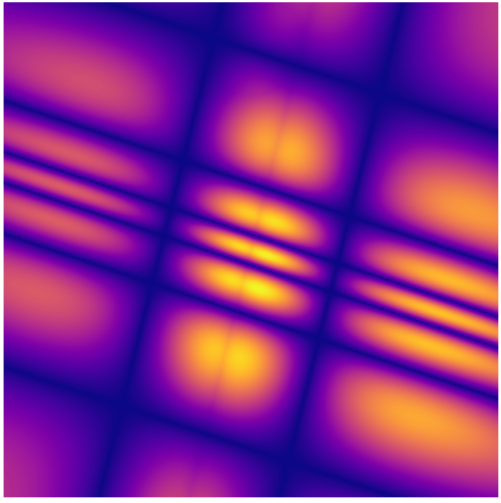
GA 2: Convergence

- Stopping conditions:
  - No improvement for k iterations
  - Time limit
  - Good enough
- *Anytime algorithm* = Can always return answer
- Live system: Never stop!

UNIVERSITY OF BATH

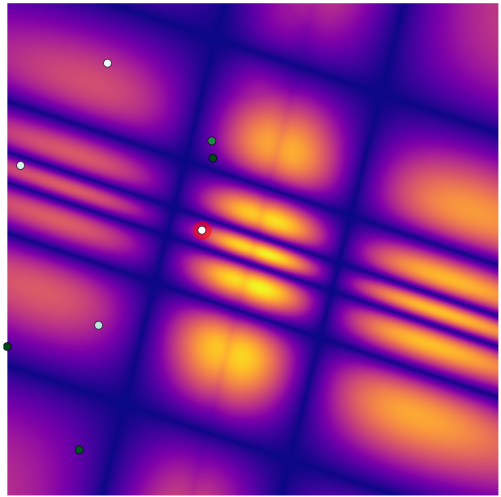
Demo

- Target function



Demo

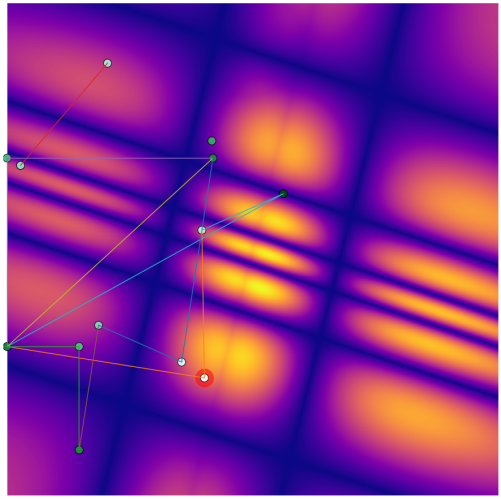
- Random initialisation – best = 0.463



UNIVERSITY OF BATH

Demo

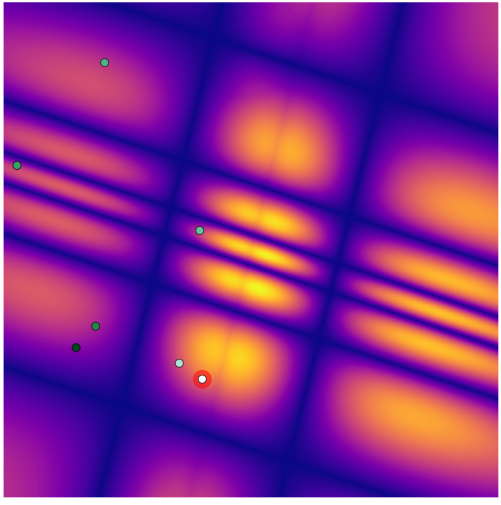
- Random initialisation – best $= 0.463$
- Extended population
  (lines show parents)



UNIVERSITY OF BATH

Demo

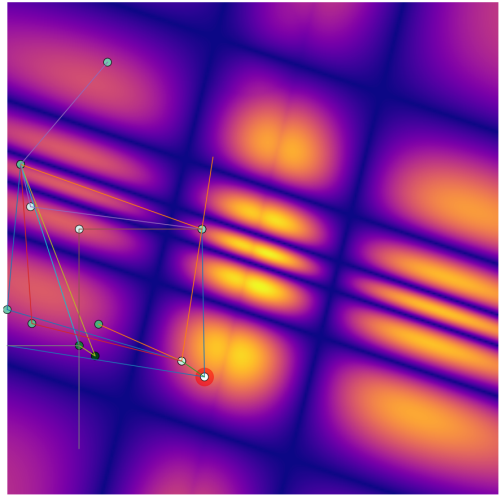
- Random initialisation – best = 0.463
- Iteration 1 best = 0.643



UNIVERSITY OF BATH

Demo

- Random initialisation – best = 0.463
- Iteration 1 best = 0.643

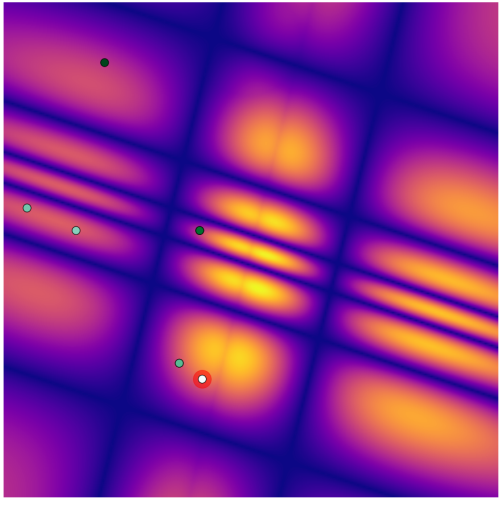


UNIVERSITY OF BATH

Demo

- Random initialisation – best = 0.463

- Iteration 1 best = 0.643
- Iteration 2 best = 0.643 (no improvement)

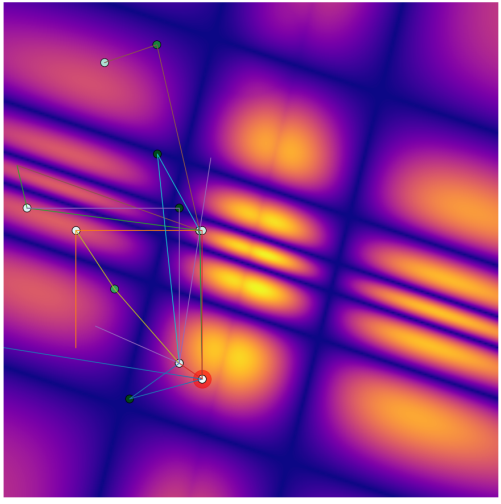


UNIVERSITY OF BATH

Demo

- Random initialisation – best = 0.463

- Iteration 1 best = 0.643
- Iteration 2 best = 0.643 (no improvement)

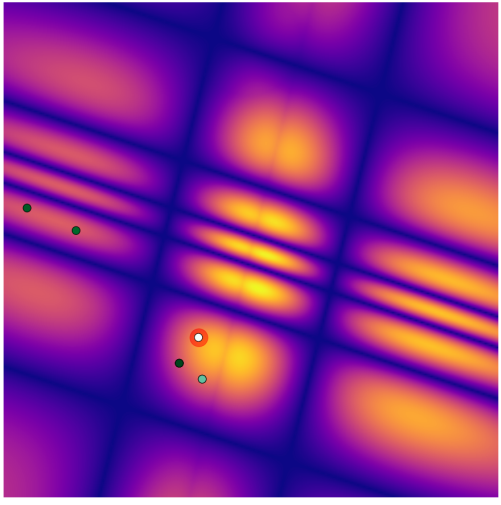


UNIVERSITY OF BATH

Demo

- Random initialisation – best = 0.463

- Iteration 1 best = 0.643
- Iteration 2 best = 0.643 (no improvement)
- Iteration 4 best = 0.760 (skipping...)

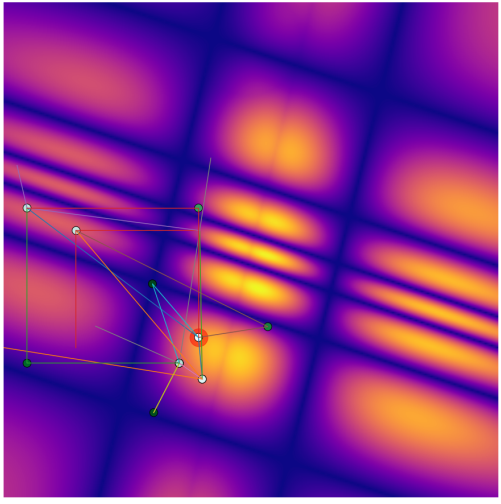


UNIVERSITY OF BATH

Demo

- Random initialisation – best = 0.463

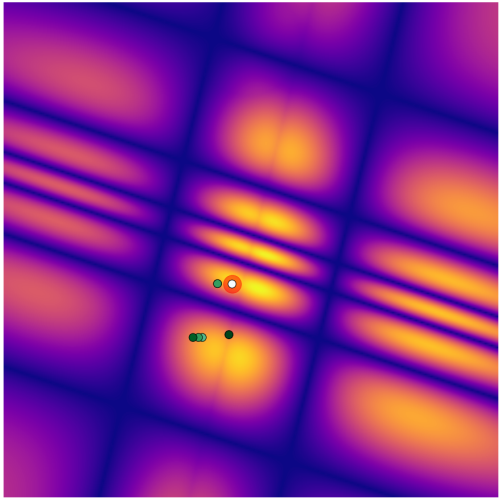
- Iteration 1 best = 0.643
- Iteration 2 best = 0.643 (no improvement)
- Iteration 4 best = 0.760 (skipping...)



Demo

- Random initialisation – best = 0.463

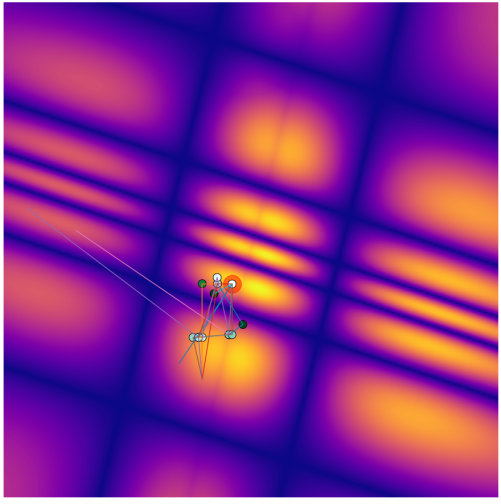
- Iteration 1 best = 0.643
- Iteration 2 best = 0.643 (no improvement)
- Iteration 4 best = 0.760 (skipping...)
- Iteration 8 best = 0.912



Demo

- Random initialisation – best = 0.463

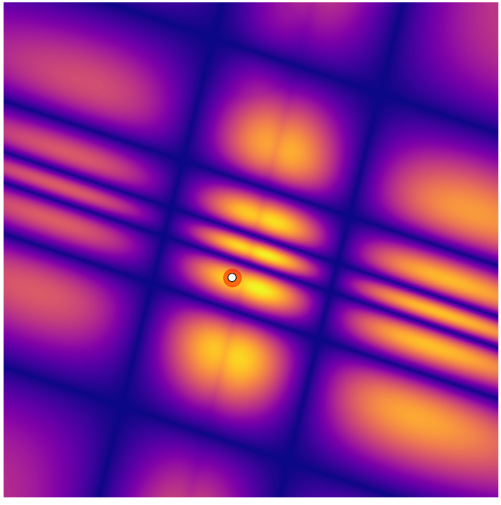
- Iteration 1 best = 0.643
- Iteration 2 best = 0.643 (no improvement)
- Iteration 4 best = 0.760 (skipping...)
- Iteration 8 best = 0.912



Demo

- Random initialisation – best = 0.463

- Iteration 1 best = 0.643
- Iteration 2 best = 0.643 (no improvement)
- Iteration 4 best = 0.760 (skipping...)
- Iteration 8 best = 0.912
- Iteration 16 best = 0.932



Extensions

- Many!
- e.g. Evaluation of $F(\Theta)$ slow enough $\therefore$.
  - Keep database of all evaluations: $n = \infty$
  - Cancel evaluation if parameters too close to previous (shrink "too close" in later stages)

(use multiple computers if possible)

Extensions

- Many!
- e.g. Evaluation of $F(\Theta)$ slow enough $\therefore$.
  - Keep database of all evaluations: $n = \infty$
  - Cancel evaluation if parameters too close to previous
    (shrink "too close" in later stages)

  (use multiple computers if possible)
- Be careful:
  Ambiguity + "magic" of evolution + easy code = Feeling of progress when there is none

Further approaches

- Particle swarm
- Simulated annealing
- Nelder-Mead
  (`scipy.optimize.minimize(method='Nelder-Mead')`
- Bayesian optimisation
  (in week 6)

Starling murmuration looks like a particle swarm

Which approach?

- Depends on **speed** and **dimensionality** of $F(\Theta)$

- Slower $\implies$ more sophisticated algorithm
- More dimensions $\implies$ more sophisticated algorithm

- Trade-off between time spent on the algorithm vs function evaluation
- Odds of finding a good answer

UNIVERSITY OF BATH

Differencing

- Or... use gradient descent!
- Estimate gradient:
  - Forward differencing: (can also go backwards)

$$\frac{df(x)}{dx} \approx \frac{f(x+h) - f(x)}{h}$$

  - Central differencing:

$$\frac{df(x)}{dx} \approx \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

(h depends on numerical precision, but $h = 1e-8$ is a good start for *doubles*)

Differencing

- Or... use gradient descent!
- Estimate gradient:
  - Forward differencing: (can also go backwards)

$$\frac{df(x)}{dx} \approx \frac{f(x+h) - f(x)}{h}$$

  - Central differencing:

$$\frac{df(x)}{dx} \approx \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

(h depends on numerical precision, but $h = 1e-8$ is a good start for *doubles*)

- Function evaluations required:

Dimensions	Forward	Central
1	2	2
2	3	4
3	4	6
n	n+1	2n

- For fast/low dimension functions this may be best

Summary

- No gradient, no problem!
  (a few approaches)
- If nothing else, avoid manual!
- Generally prefer the more sophisticated algorithm!
- Next lecture: Advanced gradient descent

UNIVERSITY OF BATH

Reading list

- Much deeper analysis of random vs gridding:
  "*Random Search for Hyper-Parameter Optimization*"
  by Bergstra and Bengio, 2012

- Mathematically rigorous genetic algorithm:
  "*Sex as Gibbs Sampling: a probability model of evolution*"
  by Watkins and Buttkewitz, 2014

- Advanced particle swarm optimisation technique:
  "*Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation*"
  by Hansen and Ostermeier, 1996