Differentiate your Objective

VLC = ∫∫∫ Vision Learning Control

Differentiable Programming

How does pre-university calculus relate to AI and the future of computer programming?

Jonathon Hare

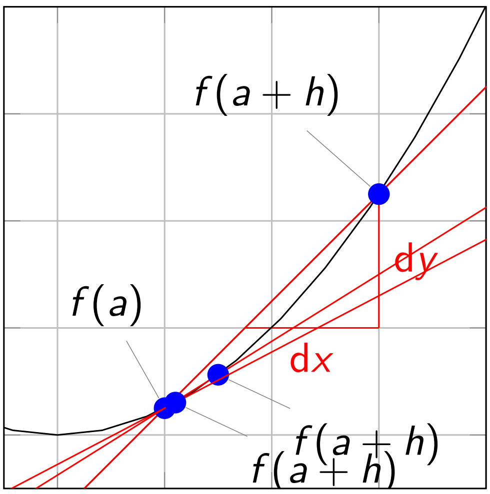
Vision, Learning and Control
University of Southampton

Differentiation

Recap: what is the derivative of a function of one variable?

The derivative in 1D

- Recall that the gradient of a straight line is $\frac{\mathrm{d}y}{\mathrm{d}x}$.
- For an arbitrary real-valued function, $f(a)$, we can approximate the derivative, $f'(a)$ using the gradient of the *secant line* defined by $(a, f(a))$ and a point a small distance, h, away $(a+h, f(a+h))$: $f'(a) \approx \frac{f(a+h)-f(a)}{h}$.
  - This expression is *Newton's Quotient*.
  - As h becomes smaller, the approximated derivative becomes more accurate.
  - If we take the limit as $h \to 0$, then we have an exact expression for the derivative: $\frac{df}{da} = f'(a) = \lim_{h\to 0} \frac{f(a+h)-f(a)}{h}$.



Recap: what are derivatives and how do we find them?

The derivative of $y = x^2$ from first principles

$$
\begin{aligned}
y &= x^2 \\
\frac{dy}{dx} &= \lim_{h\to 0} \frac{(x+h)^2 - x^2}{h} \\
\frac{dy}{dx} &= \lim_{h\to 0} \frac{x^2 + h^2 + 2hx - x^2}{h} \\
\frac{dy}{dx} &= \lim_{h\to 0} \frac{h^2 + 2hx}{h} \\
\frac{dy}{dx} &= \lim_{h\to 0} (h + 2x) \\
\frac{dy}{dx} &= 2x
\end{aligned}
$$

Intuition: What does the gradient dy/dx tell us

- The 'rate of change' of y with respect to x.
- By how much does y change if I make a small change to the x.

Why should we care?

Solving a simple problem with differentiation

- At what angle should a javelin be thrown to maximise the distance travelled?
- Assume initial velocity $u = 28\,\mathrm{m\,s^{-1}}$ and $g = 9.8\,\mathrm{m\,s^{-2}}$
- Choose to ignore launch height as it is negligable compared to distance travelled.
- Kinematics equations:

$$x = ut\cos(\theta) = 28t\cos(\theta)$$
$$y = ut\sin(\theta) - 0.5gt^2 = 28t\sin(\theta) - 4.9t^2$$

Why should we care?

Solving a simple problem with differentiation

$$x = 28t\cos(\theta)$$
$$y = 28t\sin(\theta) - 4.9t^2$$

- Javelin hits ground when $y = 0$ and we only care about $t > 0$:

$$0 = 28t\sin(\theta) - 4.9t^2$$
$$\implies t = \frac{28}{4.9}\sin(\theta)$$

- Substituting into the horizontal component:

$$x = 28\frac{28}{4.9}\sin(\theta)\cos(\theta) = 80\sin(2\theta)$$

Why should we care?

Solving a simple problem with differentiation

$$\max_{\theta} \quad 80\sin(2\theta)$$
$$\text{s.t.} \quad 0 \leq \theta \leq \frac{\pi}{2}$$

Compute derivative w.r.t θ and set to zero:

$$\begin{aligned} 0 &= \frac{\mathrm{d}(80\sin(2\theta))}{\mathrm{d}\theta} \\ &= 160\cos(2\theta) \\ \implies \theta &= \frac{1}{2}\cos^{-1}(0) = \frac{\pi}{4} \end{aligned}$$

Irrespective of the initial velocity maximum distance is acheived at 45°.

Abstraction: Solving problems by minimising an objective

- To compute the parameter (angle) for the javelin example we *maximised* the equation for distance travelled.
- We can solve all kinds of problems if we can:
  - **formulate** a *loss* or *cost* function.
  - **minimise** the loss with respect to the parameter(s)[1].
- Problems:
  - The loss must be differentiable (or rather you must be able to compute or estimate its gradient somehow).
  - Some loss functions might have many minima; you might have to settle for finding a sub-optimal one (or a saddle-point).
  - The loss function could be arbitrarily complex... you might not be able to analytically compute the solution (or the gradient).

[1] Note: maximising a distance is the same as minimising a negative distance

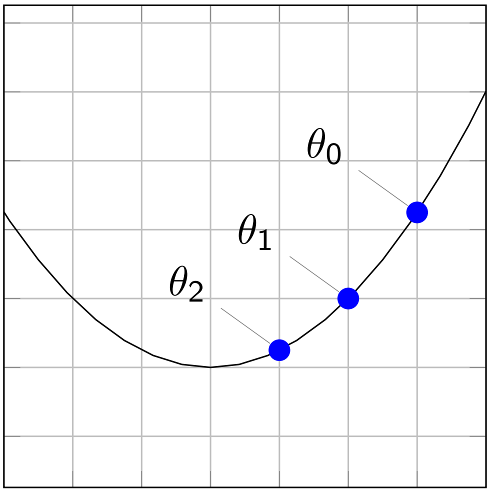
A simple algorithm for minimising a function

Gradient Decent

- How can you numerically estimate the value of the parameter θ that minimises a loss function, $\ell(\theta)$?
- Really intuitive idea: starting from an initial guess, θ_0, take small steps in the direction of the negative gradient.

Gradient Descent:

$\theta_{i+1} = \theta_i - \gamma \frac{d\ell}{d\theta}$ where γ is the *learning rate*



Javelin throwing again, but with Python code

Derivatives of more general functions

- Almost all complex functions can be broken into simpler parts (often with very simple derivatives).
- You can add (or subtract) sub-functions, multiply (or divide) sub-functions and make functions of functions.
  - The sum rule, product rule and chain rule tell you how to differentiate these.
- If you break down functions into their constituent parts computing the derivative becomes very easy
- Example: the sin function can be written in terms of exponentials (Euler's formula) and the derivative of an exponential e^x is just e^x...

Derivatives of more general functions

- Most interesting functions that we might want to work with have more than one parameter that we might want to optimise.
  - In many real applications it can be *millions* of parameters.
- Partial derivatives $\frac{\partial f}{\partial x_i}$ let us compute the gradient of the i-th parameter by holding the other parameters constant.

Back to programming

Programming is really just function composition and control statements

- At the end of the day computer programs are just compositions of really simple functions that computer processors can compute: arithmetic operations (add, multiply, divide, ...), logical operations (and, or, not, comparisons...), operations that move data, etc.
- Many of these primitive operations have *well defined* gradients with respect to their operands.
- The chain rule tells us how to compute gradients of composite functions.

So, in principle we can find the optimal "parameters" of a computer program designed to solve a specific task by following the gradients to optimise it.

Differentiating Branches

Code - *if-else* statement

```
if a > 0.5:
    b = 0
else:
    b = 2 * a
```

Math

$$b(a) = \begin{cases} 0 & \text{if } a > 0.5 \\ 2a & \text{if } a \leq 0.5 \end{cases}$$

$$\frac{\partial b}{\partial a} = \begin{cases} 0 & \text{if } a > 0.5 \\ 2 & \text{if } a \leq 0.5 \end{cases}$$

Differentiating Loops

Code - *for loop* statement

```
b = 1
for i in range(3):
    b = b + b * a
```

Math

$$b_0 = 1$$

$$b_1 = b_0 + b_0 a = 1 + a$$

$$b_2 = b_1 + b_1 a = 1 + 2a + a^2$$

$$b_3 = b_2 + b_2 a = 1 + 3a + 3a^2 + a^3$$

$$\frac{\partial b}{\partial a} = 3 + 6a + 3a^2$$

Can all programs be differentiable?

- We can differentiate through lots of types of programs and algorithms (even the Gradient Decent algorithm is itself differentiable!), but...
- not every operation or function has *useful* gradients
  - discontinuities, large areas of zero-gradient, ...
- Computer science researchers are actively developing mathematical 'tricks' to circumvent many of these problems.
  - *Relaxations* of functions that behave almost the same, but have well defined gradients.
  - *Reparameterisations* of functions involving randomness.
  - *Approximations* of useable gradients for functions that have ill-posed gradients.

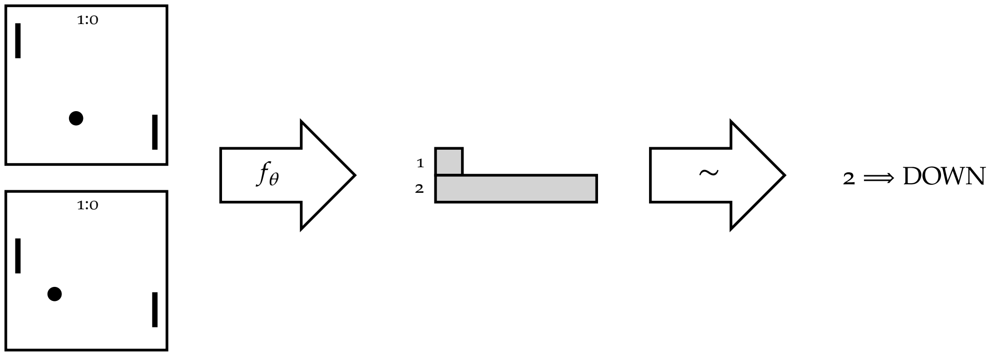
What kinds of functional building blocks are common?

- Today, the most common operations with parameters are:
  - *Vector addition*: the input vector to a function is added to a vector of weights.
  - *Vector-Matrix multiplication*: the input vector to the function is multiplied with a matrix of weights.
  - *Convolution*: the input vector (or matrix...) is 'convolved' with a set of weights.
  - (in all these cases 'weights' are the parameters which are learned)
- The above operations are *linear*, so they are often combined with element-wise nonlinearities; e.g.:
  - $\max(0, x)$ aka ReLU.
  - $\tanh(x)$.
  - $\frac{1}{1+e^{-x}}$ aka *sigmoid* or the *logistic* function.

Real Examples of Differentiable Programming

Playing Games

- You can use differentiable programming to write (and train) 'agents' that can play games.
- It can be hard to get a gradient from a single game involving many moves, but there is a clever trick which allows good estimates of gradients to be created over the average of *many* games.
- This is broadly the area of what is called *reinforcement learning*.

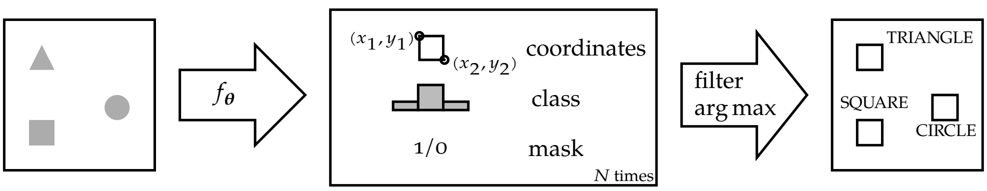


Playing Games

Demo: AlphaStar

Object detection

- Consider a function that takes an image as input and produces an array of *bounding boxes* and corresponding *labels*.
- With enough *training data* we can learn the parameters required to detect objects in images.



Object detection

Demo

Drawing

- We could envisage a differentiable function that takes in a set of line coordinates and turns them into an image...
- With such a function we can optimise the line coordinates so they e.g. match a photograph, thus automatically creating a *sketch*.

Drawing
Demo

Drawing
Demo

Where is this all going?

Software 2.0

There is a revolution happening and you're going to be part of it

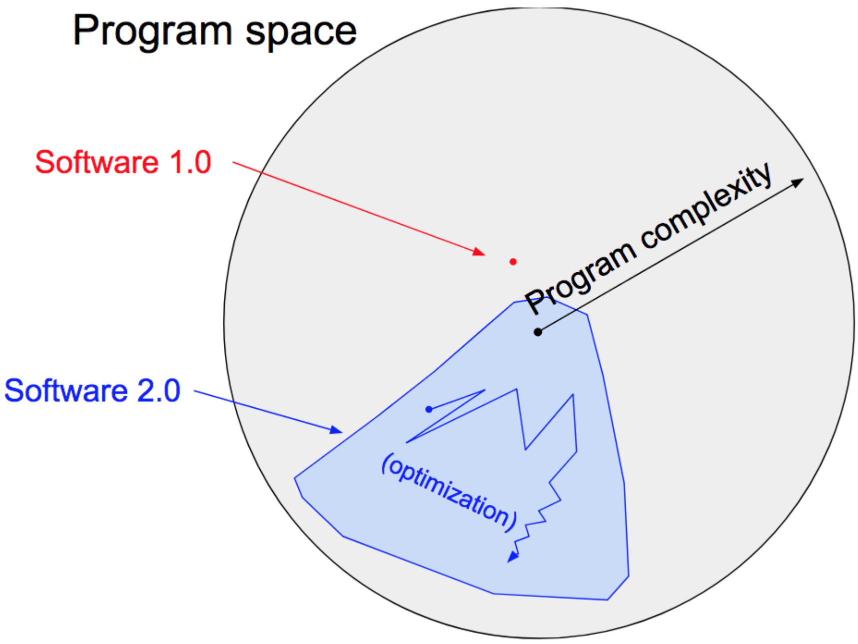


Image credit: Andrei Karpathy
https://karpathy.medium.com/software-2-0-a64152b37c35

Any Questions?