

Follow the Gradient

VLC = $\int \int \int$ Vision
Learning & Control

The power of differentiation

Jonathon Hare

Vision, Learning and Control
University of Southampton

- The big idea: optimisation by following gradients
- Recap: what are gradients and how do we find them?
- Recap: Singular Value Decomposition and its applications
- Example: Computing SVD using gradients - The Netflix Challenge

The big idea: optimisation by following gradients

- Fundamentally, we're interested in machines that we train by optimising parameters
 - How do we select those parameters?
- In deep learning/differentiable programming we typically define an objective function that we *minimise* (or *maximise*) with respect to those parameters
- This implies that we're looking for points at which the gradient of the objective function is zero w.r.t the parameters

The big idea: optimisation by following gradients

- Gradient based optimisation is a *big* field!
 - First order methods, second order methods, subgradient methods...
- With deep learning we're primarily interested in first-order methods¹.
 - Primarily using variants of gradient descent: a function $F(\mathbf{x})$ has a minima² (or a saddle-point) at a point $\mathbf{x} = \mathbf{a}$ where \mathbf{a} is given by applying $\mathbf{a}_{n+1} = \mathbf{a}_n - \alpha \nabla F(\mathbf{a}_n)$ until convergence from some initial point \mathbf{a}_0 .

¹Second order gradient optimisers are potentially better, but for systems with many variables are currently impractical as they require computing the Hessian.

²not necessarily global or unique

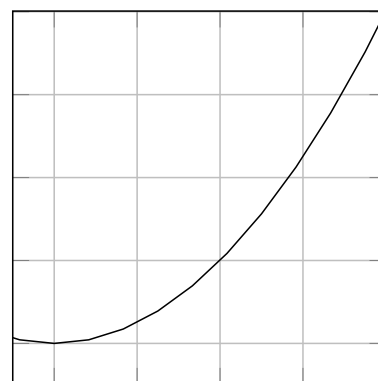
Recap: what are gradients and how do we find them?

The derivative in 1D

- Recall that the gradient of a straight line is $\frac{\Delta y}{\Delta x}$.

- 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' or 'Fermat's Difference Quotient'.
- As h becomes smaller, the approximated derivative becomes more accurate.
- If we take the limit as $h \rightarrow 0$, then we have an exact expression for the derivative:
$$\frac{df}{da} = f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h)-f(a)}{h}.$$



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

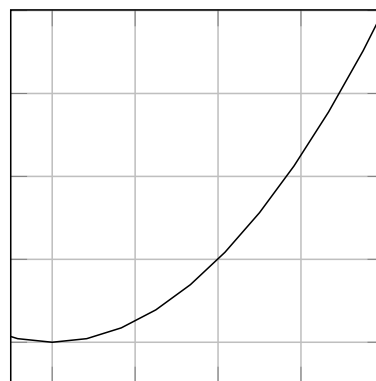
The derivative of $y = x^2$ from first principles

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

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

Aside: numerical approximation of the derivative

- For numerical computation of derivatives it is better to use a “centralised” definition of the derivative:
 - $f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a-h)}{2h}$
 - The bit inside the limit is known as the *symmetric difference quotient*
 - For small values of h this has less error than the standard one-sided difference quotient.



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

Aside: numerical approximation of the derivative

- If you are going to use difference quotients to estimate derivatives you need to be aware of potential rounding errors due to floating point representations.
 - Calculating derivatives this way using less than 64-bit precision is rarely going to be useful. (Numbers are not represented exactly, so even if h is represented exactly, $x + h$ will probably not be)
 - You need to pick an appropriate h - too small and the subtraction will have a large rounding error!

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

Derivatives of deeper functions

- Deep learning is all about optimising deeper functions; functions that are compositions of other functions
 - e.g. $z = f \circ g(x) = f(g(x))$
- The chain rule of calculus tells us how to differentiate compositions of functions:
 - $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$

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

Example: differentiating $z = x^4$

Note that this is a silly example that just serves to demonstrate the principle!

$$z = x^4$$

$$z = (x^2)^2 = y^2 \quad \text{where} \quad y = x^2$$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} = (2y)(2x) = (2x^2)(2x) = 4x^3$$

Equivalently, from first principles:

$$z = x^4$$

$$\frac{dz}{dx} = \lim_{h \rightarrow 0} \frac{(x+h)^4 - x^4}{h}$$

$$\frac{dz}{dx} = \lim_{h \rightarrow 0} \frac{h^4 + 4h^3x + 6h^2x^2 + 4hx^3 + x^4 - x^4}{h}$$

$$\frac{dz}{dx} = \lim_{h \rightarrow 0} h^3 + 4h^2x + 6hx^2 + 4x^3 = 4x^3$$

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

Vector functions

- What if we're dealing with a *vector* function, $\mathbf{y}(t)$?
 - This can be split into its constituent coordinate functions:
 $\mathbf{y}(t) = (y_1(t), \dots, y_n(t))$.
 - Thus the derivative is a vector (the 'tangent vector'),
 $\mathbf{y}'(t) = (y_1'(t), \dots, y_n'(t))$, which consists of the derivatives of the coordinate functions.
 - Equivalently, $\mathbf{y}'(t) = \lim_{h \rightarrow 0} \frac{\mathbf{y}(t+h) - \mathbf{y}(t)}{h}$ if the limit exists.

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

Functions of multiple variables: partial differentiation

- What if the function we're trying to deal with has multiple variables³ (e.g. $f(x, y) = x^2 + xy + y^2$)?
 - This expression has a pair of *partial derivatives*, $\frac{\partial f}{\partial x} = 2x + y$ and $\frac{\partial f}{\partial y} = x + 2y$, computed by differentiating with respect to each variable x and y whilst holding the other(s) constant.
- In general, the partial derivative of a function $f(x_1, \dots, x_n)$ at a point (a_1, \dots, a_n) is given by:
$$\frac{\partial f}{\partial x_i}(a_1, \dots, a_n) = \lim_{h \rightarrow 0} \frac{f(a_1, \dots, a_i + h, \dots, a_n) - f(a_1, \dots, a_i, \dots, a_n)}{h}.$$
- The vector of partial derivatives of a scalar-value multivariate function, $f(x_1, \dots, x_n)$ at a point (a_1, \dots, a_n) , can be arranged into a vector: $\nabla f(a_1, \dots, a_n) = (\frac{\partial f}{\partial x_1}(a_1, \dots, a_n), \dots, \frac{\partial f}{\partial x_n}(a_1, \dots, a_n))$.
 - This is the **gradient** of f at a .
- In the case of a vector-valued multivariate function, the partial derivatives form a matrix called the **Jacobian**.

³A multivariate function

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

Functions of vectors and matrices: partial differentiation

- For the kinds of functions (and programs) that we'll look at *optimising* in this course have a number of typical properties:
 - They are scalar-valued
 - We'll look at programs with *multiple losses*, but ultimately we can just consider optimising with respect to the *sum* of the losses.
 - They involve multiple variables, which are often wrapped up in the form of vectors or matrices, and more generally tensors.
 - **How will we find the gradients of these?**

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

The chain rule for vectors

Suppose that $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$, g maps from \mathbb{R}^m to \mathbb{R}^n and f maps from \mathbb{R}^n to \mathbb{R} .

If $\mathbf{y} = g(\mathbf{x})$ and $z = f(\mathbf{y})$, then

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}.$$

Equivalently, in vector notation:

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^\top \nabla_{\mathbf{y}} z$$

where $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is the $n \times m$ Jacobian matrix of g .

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

The chain rule for Tensors

- Conceptually, the simplest way to think about gradients of tensors is to imagine flattening them into vectors, computing the vector-valued gradient and then reshaping the gradient back into a tensor.
 - In this way we're still just multiplying Jacobians by gradients.
- More formally, consider the gradient of a scalar z with respect to a tensor \mathbf{X} to be denoted as $\nabla_{\mathbf{X}} z$.
 - Indices into \mathbf{X} now have multiple coordinates, but we can generalise by using a single variable i to represent the complete tuple of indices.
 - For all index tuples i , $(\nabla_{\mathbf{X}} z)_i$ gives $\frac{\partial z}{\partial x_i}$.
 - Thus, if $\mathbf{Y} = g(\mathbf{X})$ and $z = f(\mathbf{Y})$ then $\nabla_{\mathbf{X}} z = \sum_j (\nabla_{\mathbf{X}} Y_j) \frac{\partial z}{\partial Y_j}$.

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

Example: $\nabla_{\mathbf{W}} f(\mathbf{XW})$

- Let $\mathbf{D} = \mathbf{XW}$ where the rows of $\mathbf{X} \in \mathbb{R}^{n \times m}$ contain some fixed *features*, and $\mathbf{W} \in \mathbb{R}^{m \times h}$ is a matrix of weights.
- Also let $\mathcal{L} = f(\mathbf{D})$ be some scalar function of \mathbf{D} that we wish to minimise.
- What are the derivatives of \mathcal{L} with respect to the weights \mathbf{W} ?

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

Example: $\nabla_{\mathbf{W}} f(\mathbf{XW})$

- Start by considering a specific weight, W_{uv} : $\frac{\partial \mathcal{L}}{\partial W_{uv}} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial D_{ij}} \frac{\partial D_{ij}}{\partial W_{uv}}$.
- We know that $\frac{\partial D_{ij}}{\partial W_{uv}} = 0$ if $j \neq v$ because D_{ij} is the dot product of row i of \mathbf{X} and column j of \mathbf{W} .
- Therefore, we can simplify the summation to only consider cases where $j = v$: $\sum_{i,j} \frac{\partial \mathcal{L}}{\partial D_{ij}} \frac{\partial D_{ij}}{\partial W_{uv}} = \sum_i \frac{\partial \mathcal{L}}{\partial D_{iv}} \frac{\partial D_{iv}}{\partial W_{uv}}$.
- What is $\frac{\partial D_{iv}}{\partial W_{uv}}$?

$$\begin{aligned} D_{iv} &= \sum_{k=1}^m X_{ik} W_{kv} \\ \frac{\partial D_{iv}}{\partial W_{uv}} &= \frac{\partial}{\partial W_{uv}} \sum_{k=1}^m X_{ik} W_{kv} = \sum_{k=1}^m \frac{\partial}{\partial W_{uv}} X_{ik} W_{kv} \\ \therefore \frac{\partial D_{iv}}{\partial W_{uv}} &= X_{iu} \end{aligned}$$

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

Example: $\nabla_{\mathbf{W}} f(\mathbf{X}\mathbf{W})$

- Putting every together, we have: $\frac{\partial \mathcal{L}}{\partial W_{uv}} = \sum_i \frac{\partial \mathcal{L}}{\partial D_{iv}} X_{iu}$.
- As we're summing over multiplications of scalars, we can change the order: $\frac{\partial \mathcal{L}}{\partial W_{uv}} = \sum_i X_{iu} \frac{\partial \mathcal{L}}{\partial D_{iv}}$.
- and note that the sum over i is doing a dot product with row u and column v if we transpose X_{iu} to X_{ui}^\top : $\frac{\partial \mathcal{L}}{\partial W_{uv}} = \sum_i X_{ui}^\top \frac{\partial \mathcal{L}}{\partial D_{iv}}$.
- We can then see that if we want this for all values of \mathbf{W} it simply generalises to: $\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \mathbf{X}^\top \frac{\partial \mathcal{L}}{\partial \mathbf{D}}$.

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

STOP! What does a gradient actually mean?

- In your early calculus lessons you likely had it hammered into you that gradients represent rates of change of functions.
- This is of course totally true...
- But, it isn't a particularly useful way to think about the gradients of a loss with respect to the weights of a parameterised function.
 - **The gradient of the loss with respect to a parameter tells you how much the loss will change with a small perturbation to that parameter.**

Recap: Singular Value Decomposition and its applications

Let's now change direction — we're going to look at an early success story resulting from using some differentiation and the Singular Value Decomposition (SVD).

For complex \mathbf{A} :

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$$

where \mathbf{V}^* is the *conjugate transpose* of \mathbf{V} .

For real \mathbf{A} :

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

Recap: Singular Value Decomposition and its applications

- SVD has many uses:
 - Computing the Eigendecomposition:
 - Eigenvectors of $\mathbf{A}\mathbf{A}^\top$ are columns of \mathbf{U} ,
 - Eigenvectors of $\mathbf{A}^\top\mathbf{A}$ are columns of \mathbf{V} ,
 - and the non-zero values of $\mathbf{\Sigma}$ are the square roots of the non-zero eigenvalues of both $\mathbf{A}\mathbf{A}^\top$ and $\mathbf{A}^\top\mathbf{A}$.
 - Dimensionality reduction
 - ...use to compute PCA
 - Computing the Moore-Penrose Pseudoinverse
 - for real \mathbf{A} : $\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^\top$ where $\mathbf{\Sigma}^+$ is formed by taking the reciprocal of every non-zero diagonal element and transposing the result.
 - Low-rank approximation and matrix completion
 - if you take the ρ columns of \mathbf{U} , and the ρ rows of \mathbf{V}^\top corresponding to the ρ largest singular values, you can form the matrix $\mathbf{A}_\rho = \mathbf{U}_\rho\mathbf{\Sigma}_\rho\mathbf{V}_\rho^\top$ which will be the *best* rank- ρ approximation of the original \mathbf{A} in terms of the Frobenius norm.

Example: Computing SVD using gradients - The Netflix Challenge

- There are many standard ways of computing the SVD:
 - e.g. 'Power iteration', or 'Arnoldi iteration' or 'Lanczos algorithm' coupled with the 'Gram-Schmidt process' for orthonormalisation
- but, these don't necessarily scale up to really big problems
 - e.g. computing the SVD of a sparse matrix with 17770 rows, 480189 columns and 100480507 non-zero entries!
 - this corresponds to the data provided by Netflix when they launched the *Netflix Challenge* in 2006.
- OK, so what can you do?
 - The 'Simon Funk' solution: realise that there is a really simple (and quick) way to compute the SVD by following gradients...

Example: Computing SVD using gradients - The Netflix Challenge

Deriving a gradient-descent solution to SVD

- One of the definitions of rank- ρ SVD of a matrix \mathbf{A} is that it minimises reconstruction error in terms of the Frobenius norm.
- Without loss of generality we can write SVD as a 2-matrix decomposition $\mathbf{A} = \hat{\mathbf{U}}\hat{\mathbf{V}}^T$ by rolling in the square roots of Σ to both $\hat{\mathbf{U}}$ and $\hat{\mathbf{V}}$: $\hat{\mathbf{U}} = \mathbf{U}\Sigma^{0.5}$ and $\hat{\mathbf{V}}^T = \Sigma^{0.5}\mathbf{V}^T$.
- Then we can define the decomposition as finding $\min_{\hat{\mathbf{U}}, \hat{\mathbf{V}}} (\|\mathbf{A} - \hat{\mathbf{U}}\hat{\mathbf{V}}^T\|_F^2)$

Example: Computing SVD using gradients - The Netflix Challenge

Deriving a gradient-descent solution to SVD

Start by expanding our optimisation problem:

$$\begin{aligned}\min_{\hat{\mathbf{U}}, \hat{\mathbf{V}}}(\|\mathbf{A} - \hat{\mathbf{U}}\hat{\mathbf{V}}^\top\|_F^2) &= \min_{\hat{\mathbf{U}}, \hat{\mathbf{V}}}(\sum_r \sum_c (A_{rc} - \hat{U}_r \hat{V}_{:,c}^\top)^2) \\ &= \min_{\hat{\mathbf{U}}, \hat{\mathbf{V}}}(\sum_r \sum_c (A_{rc} - \sum_{p=1}^{\rho} \hat{U}_{rp} \hat{V}_{cp})^2)\end{aligned}$$

Let $e_{rc} = A_{rc} - \sum_{p=1}^{\rho} \hat{U}_{rp} \hat{V}_{cp}$ denote the error. Then, our problem becomes:

$$\text{Minimise } J = \sum_r \sum_c e_{rc}^2$$

We can then differentiate with respect to specific variables \hat{U}_{rq} and \hat{V}_{cq}

Example: Computing SVD using gradients - The Netflix Challenge

Deriving a gradient-descent solution to SVD

We can then differentiate with respect to specific variables \hat{U}_{rq} and \hat{V}_{cq} :

$$\begin{aligned}\frac{\partial J}{\partial \hat{U}_{rq}} &= \sum_r \sum_c 2e_{rc} \frac{\partial e}{\partial \hat{U}_{rq}} = -2 \sum_r \sum_c \hat{V}_{cq} e_{rc} \\ \frac{\partial J}{\partial \hat{V}_{cq}} &= \sum_r \sum_c 2e_{rc} \frac{\partial e}{\partial \hat{V}_{cq}} = -2 \sum_r \sum_c \hat{U}_{rq} e_{rc}\end{aligned}$$

and use this as the basis for a gradient descent algorithm:

$$\begin{aligned}\hat{U}_{rq} &\Leftarrow \hat{U}_{rq} + \lambda \sum_r \sum_c \hat{V}_{cq} e_{rc} \\ \hat{V}_{cq} &\Leftarrow \hat{V}_{cq} + \lambda \sum_r \sum_c \hat{U}_{rq} e_{rc}\end{aligned}$$

Example: Computing SVD using gradients - The Netflix Challenge

Deriving a gradient-descent solution to SVD

- A stochastic version of this algorithm (updates on one single item of \mathbf{A} at a time) helped win the Netflix Challenge competition in 2009.
- It was both *fast* and *memory efficient*