

Back Propagation and the Chain Rule

David S. Rosenberg

Bloomberg ML EDU

December 19, 2017

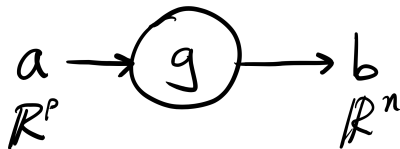
Learning with Back-Propagation

- Back-propagation is an **algorithm** for computing the gradient
- With lots of chain rule, you could also work out the gradient by hand.
- Back-propagation is
 - a clean way to organize the computation of the gradient
 - an efficient way to compute the gradient

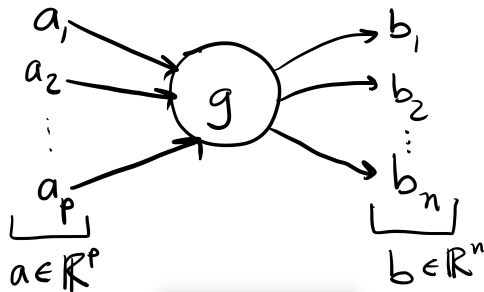
Partial Derivatives

- Consider a function $g: \mathbb{R}^p \rightarrow \mathbb{R}^n$.

- Typical computation graph:

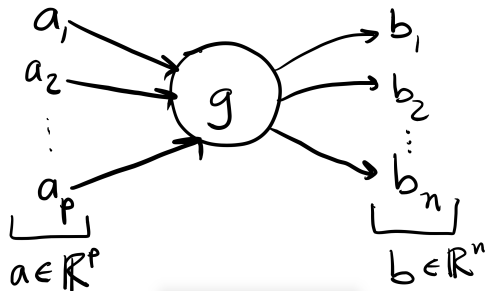


- Broken out into components:



Partial Derivatives

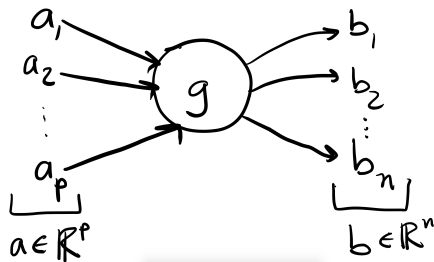
- Consider a function $g : \mathbb{R}^p \rightarrow \mathbb{R}^n$.



- Partial derivative $\frac{\partial b_i}{\partial a_j}$ is the instantaneous rate of change of b_i as we change a_j .
- If we change a_j slightly to $a_j + \delta$,
- Then (for small δ), b_i changes to approximately $b_i + \frac{\partial b_i}{\partial a_j} \delta$.

Partial Derivatives of an Affine Function

- Define the affine function $g(x) = Mx + c$, for $M \in \mathbf{R}^{n \times p}$ and $c \in \mathbf{R}$.



- If we let $b = g(a)$, then what is b_i ?
- b_i depends on the i th row of M :

$$b_i = \sum_{k=1}^p M_{ik} a_k + c_i$$

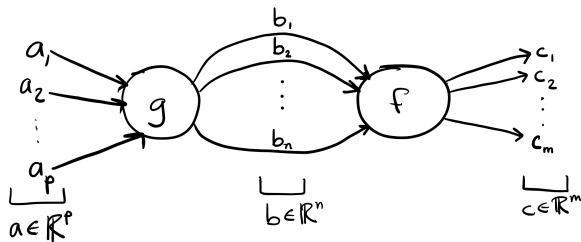
and

$$\frac{\partial b_i}{\partial a_j} = M_{ij}.$$

- So for an affine mapping, entries of matrix M directly tell us the rates of change.

Chain Rule (in terms of partial derivatives)

- $g: \mathbb{R}^p \rightarrow \mathbb{R}^n$ and $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$. Let $b = g(a)$. Let $c = f(b)$.



- Chain rule says that

$$\frac{\partial c_i}{\partial a_j} = \sum_{k=1}^n \frac{\partial c_i}{\partial b_k} \frac{\partial b_k}{\partial a_j}.$$

- Change in a_j may change each of b_1, \dots, b_n .
- Changes in b_1, \dots, b_n may each effect c_i .
- Chain rule tells us that, to first order, the net change in c_i is
 - the sum of the changes induced along each path from a_j to c_i .

Example: Least Squares Regression

Review: Linear least squares

- Hypothesis space $\{f(x) = w^T x + b \mid w \in \mathbf{R}^d, b \in \mathbf{R}\}$.
- Data set $(x_1, y_1), \dots, (x_n, y_n) \in \mathbf{R}^d \times \mathbf{R}$.
- Define

$$\ell_i(w, b) = [(w^T x_i + b) - y_i]^2.$$

- In SGD, in each round we'd choose a random index $i \in 1, \dots, n$ and take a gradient step

$$\begin{aligned} w_j &\leftarrow w_j - \eta \frac{\partial \ell_i(w, b)}{\partial w_j}, \text{ for } j = 1, \dots, d \\ b &\leftarrow b - \eta \frac{\partial \ell_i(w, b)}{\partial b}, \end{aligned}$$

for some step size $\eta > 0$.

- Let's revisit how to calculate these partial derivatives...

Computation Graph and Intermediate Variables

- For a generic training point (x, y) , denote the loss by

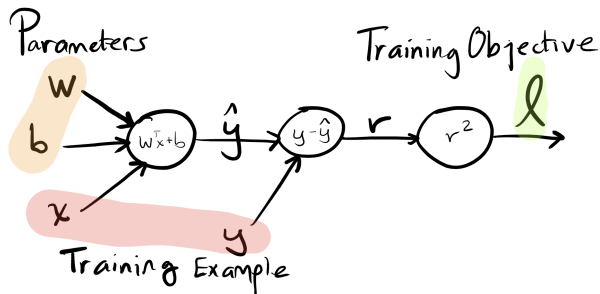
$$\ell(w, b) = [(w^T x + b) - y]^2.$$

- Let's break this down into some intermediate computations:

$$\text{(prediction)} \quad \hat{y} = \sum_{j=1}^d w_j x_j + b$$

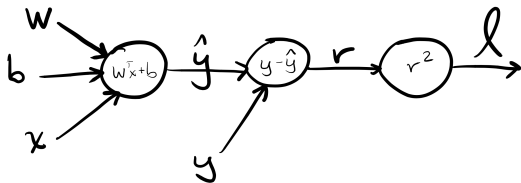
$$\text{(residual)} \quad r = y - \hat{y}$$

$$\text{(loss)} \quad \ell = r^2$$



Partial Derivatives on Computation Graph

- We'll work our way from graph output ℓ back to the parameters w and b :



$$\begin{aligned}\frac{\partial \ell}{\partial r} &= 2r \\ \frac{\partial \ell}{\partial \hat{y}} &= \frac{\partial \ell}{\partial r} \frac{\partial r}{\partial \hat{y}} = (2r)(-1) = -2r \\ \frac{\partial \ell}{\partial b} &= \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b} = (-2r)(1) = -2r \\ \frac{\partial \ell}{\partial w_j} &= \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_j} = (-2r)x_j = -2rx_j\end{aligned}$$

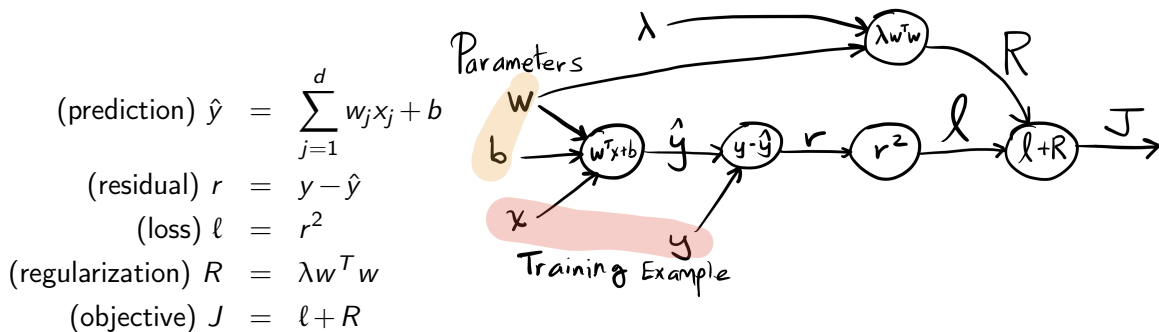
Example: Ridge Regression

Ridge Regression: Computation Graph and Intermediate Variables

- For training point (x, y) , the ℓ_2 -regularized objective function is

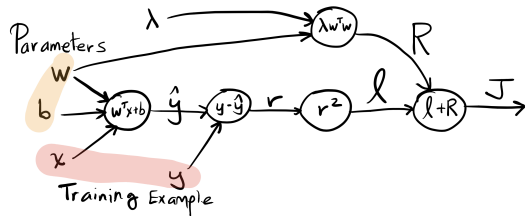
$$J(w, b) = [(w^T x + b) - y]^2 + \lambda w^T w.$$

- Let's break this down into some intermediate computations:



Partial Derivatives on Computation Graph

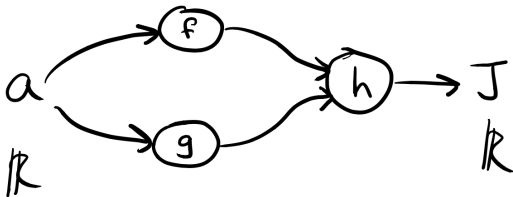
- We'll work our way from graph output ℓ back to the parameters w and b :



$$\begin{aligned}\frac{\partial J}{\partial \ell} &= \frac{\partial J}{\partial R} = 1 \\ \frac{\partial J}{\partial \hat{y}} &= \frac{\partial J}{\partial \ell} \frac{\partial \ell}{\partial r} \frac{\partial r}{\partial \hat{y}} = (1)(2r)(-1) = -2r \\ \frac{\partial J}{\partial b} &= \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b} = (-2r)(1) = -2r \\ \frac{\partial J}{\partial w_j} &= ?\end{aligned}$$

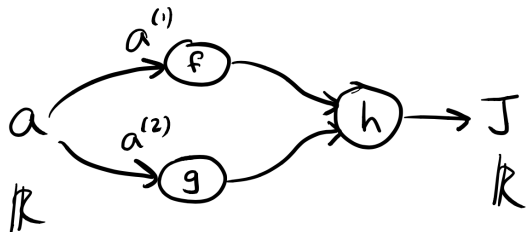
Handling Nodes with Multiple Children

- Consider $a \mapsto J = h(f(a), g(a))$.



- It's helpful to think about having two independent copies of a , call them $a^{(1)}$ and $a^{(2)}$...

Handling Nodes with Multiple Children

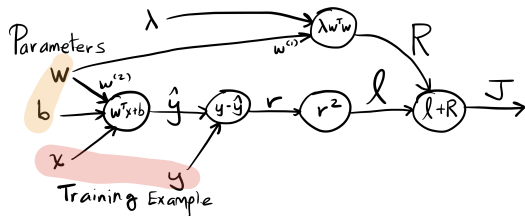


$$\begin{aligned}\frac{\partial J}{\partial a} &= \frac{\partial J}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial a} + \frac{\partial J}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial a} \\ &= \frac{\partial J}{\partial a^{(1)}} + \frac{\partial J}{\partial a^{(2)}}\end{aligned}$$

- Derivative w.r.t. a is the sum of derivatives w.r.t. each copy of a .

Partial Derivatives on Computation Graph

- We'll work our way from graph output ℓ back to the parameters w and b :



$$\frac{\partial J}{\partial \hat{y}} = \frac{\partial J}{\partial \ell} \frac{\partial \ell}{\partial r} \frac{\partial r}{\partial \hat{y}} = (1)(2r)(-1) = -2r$$

$$\frac{\partial J}{\partial w_j^{(2)}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_j^{(2)}} = \frac{\partial J}{\partial \hat{y}} x_j$$

$$\frac{\partial J}{\partial w_j^{(1)}} = \frac{\partial J}{\partial R} \frac{\partial R}{\partial w_j^{(1)}} = (1)(2\lambda w_j^{(1)})$$

$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial w_j^{(1)}} + \frac{\partial J}{\partial w_j^{(2)}}$$

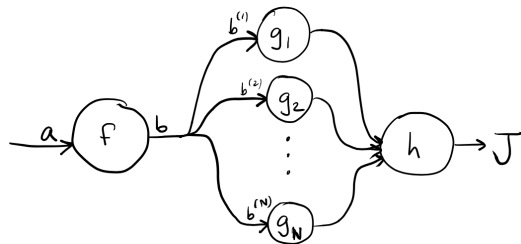
General Backpropagation

Backpropagation: Overview

- Backpropagation is a specific way to evaluate the partial derivatives of a computation graph output J w.r.t. the inputs and outputs of all nodes.
- Backpropagation works node-by-node.
- To run a “backward” step at a node f , we assume
 - we’ve already run “backward” for all of f ’s children.
- **Backward** at node $f : a \mapsto b$ returns
 - Partial of objective value J w.r.t. f ’s output: $\frac{\partial J}{\partial b}$
 - Partial of objective value J w.r.t f ’s input: $\frac{\partial J}{\partial a}$

Backpropagation: Simple Case

- Simple case: all nodes take a single scalar as input and have a single scalar output.

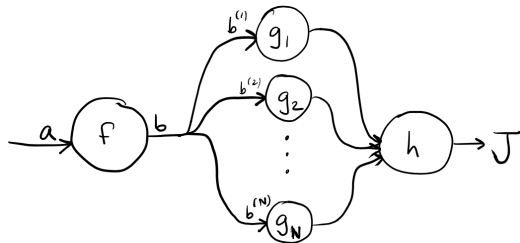


- Backprop for node f :
- Input:** $\frac{\partial J}{\partial b^{(1)}}, \dots, \frac{\partial J}{\partial b^{(N)}}$
(Partials w.r.t. inputs to all children)
- Output:**

$$\frac{\partial J}{\partial b} = \sum_{k=1}^N \frac{\partial J}{\partial b^{(k)}}$$
$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial b} \frac{\partial b}{\partial a}$$

Backpropagation (General case)

- More generally, consider $f : \mathbf{R}^d \rightarrow \mathbf{R}^n$.



- **Input:** $\frac{\partial J}{\partial b_j^{(i)}}$, $i = 1, \dots, N$, $j = 1, \dots, n$
- **Output:**

$$\frac{\partial J}{\partial b_j} = \sum_{k=1}^N \frac{\partial J}{\partial b_j^{(k)}}$$
$$\frac{\partial J}{\partial a_i} = \sum_{j=1}^n \frac{\partial J}{\partial b_j} \frac{\partial b_j}{\partial a_i}$$

Running Backpropagation

- If we run “backward” on every node in our graph,
 - we’ll have the gradients of J w.r.t. all our parameters.
- To run backward on a particular node,
 - we assumed we already ran it on all children.
- A **topological sort** of the nodes in a directed [acyclic] graph
 - is an ordering which every node appears before its children.
- So we’ll evaluate backward on nodes in a **reverse topological ordering**.