ML & Opt Lec 17

Autoencoders
Pytorch Implementation of autoencoders
Backprop

Autoencoders

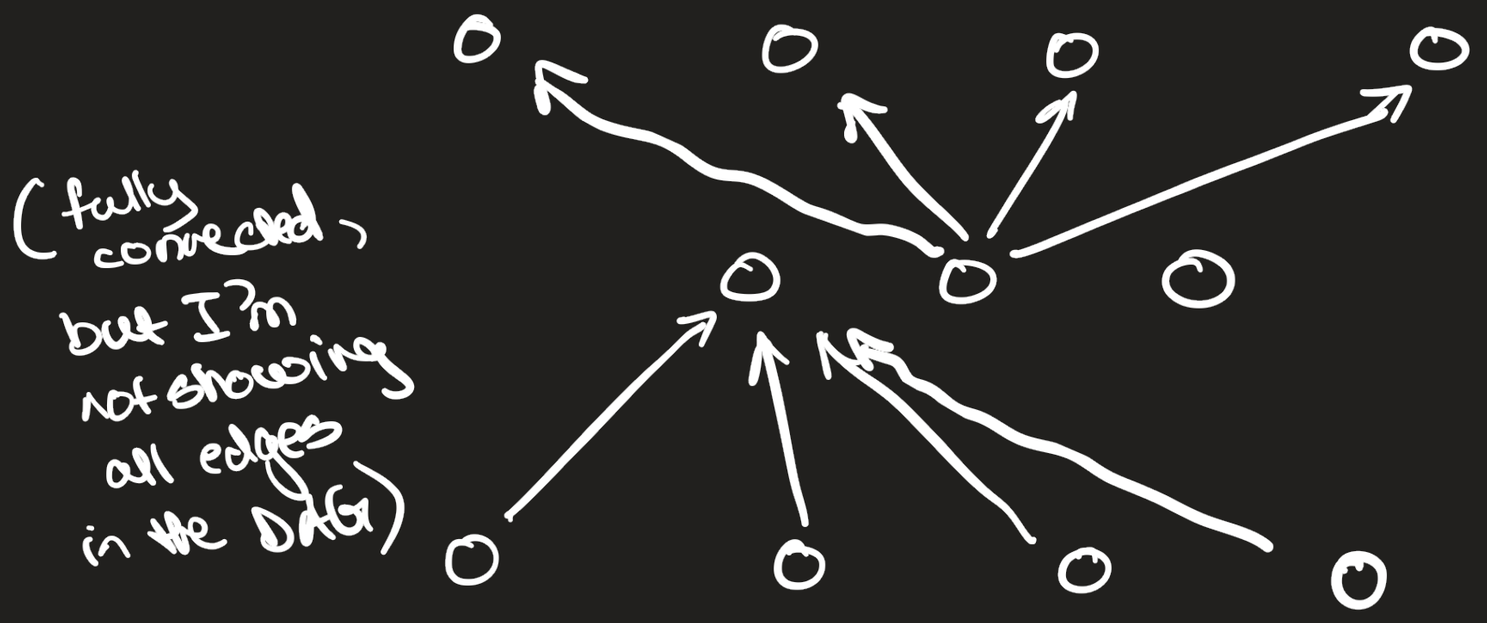
- nonlinear generalizations of PCA & SVD
- find "good" low dimensional encodings of your input signal (auto encoder)
- measure "good"ness by how well the signal can be reconstructed (auto encoder)

Learning problem:

$$\min_{E,D} \mathbb{E}_{x \sim p} \|x - D(E(x))\|_2^2 + \lambda R(E,D)$$

in practice, we throw away D and just use E

Simple AE architecture (FC FFN)



our encoder looks like

$$E(x) = \sigma(W'x + b') \quad \text{where } W' \in \mathbb{R}^{n_1 \times n_0} \text{ and } b' \in \mathbb{R}^{n_1}$$

and decoder is

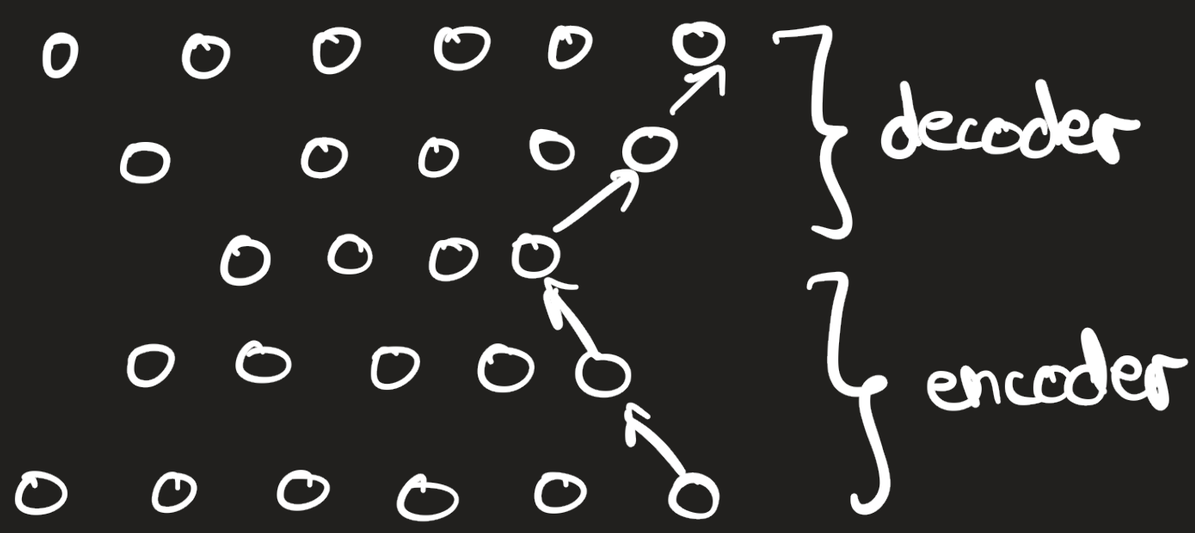
$$D(y) = \sigma(W^2 y + b^2) \quad \text{where } W^2 \in \mathbb{R}^{n_0 \times n_1}$$

The associated RERM problem

$$\underset{\substack{\omega_E \to \omega_D \\ \| \\ \{\omega^1, b^1\} \quad \{\omega^2, b^2\}}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \| x_i - \sigma(\omega^2 \sigma(\omega^1 x + b^1) + b^2) \|_2^2 + \lambda \left[\|\omega^1\|_F^2 + \|\omega^2\|_F^2 + \|b^1\|_2^2 + \|b^2\|_2^2 \right]$$

Typical AE:

- progressively encode/decode so E and D are more sophisticated/expressive nonlinear functions



Backpropagation

- Chain rule applied systematically to compute gradients with respect to NN parameters
- Idea: specify the forward pass as a DAG, then we can use the chain rule to do a backward pass on the DAG to compute the required derivatives

Ex:

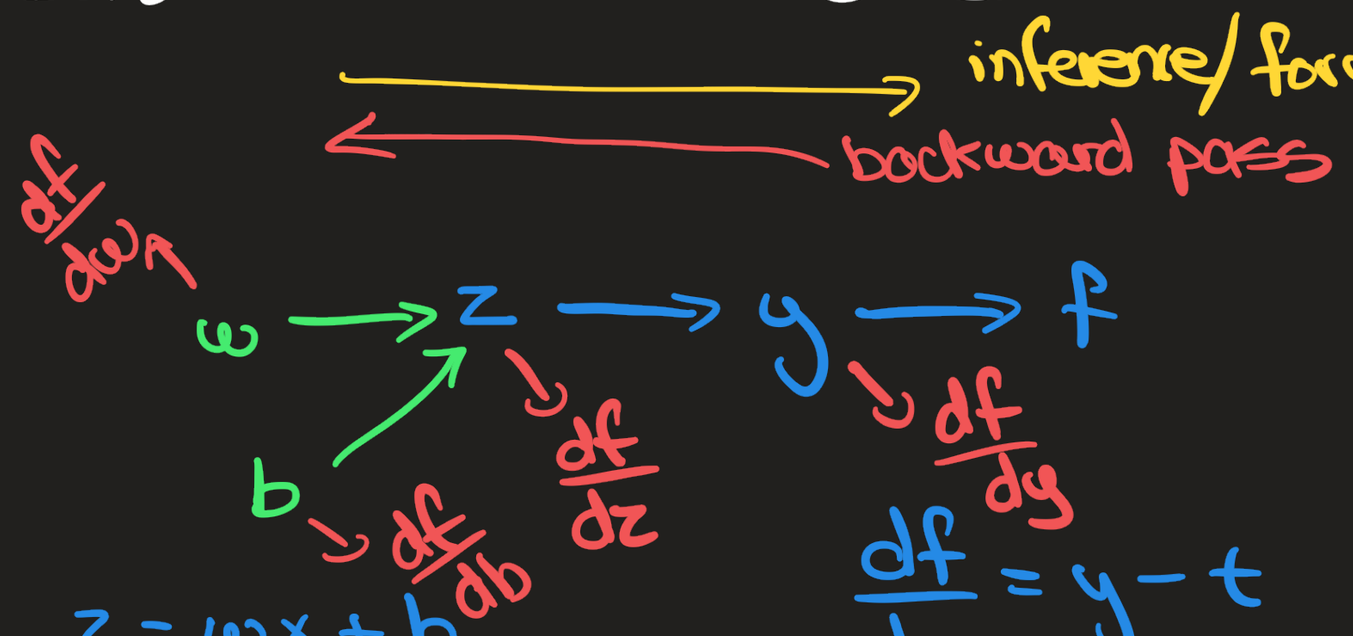
$$f(w,b) = \frac{1}{2}\left(\sigma(wx+b)-t\right)^2$$

then

$$\frac{df}{dw} = \left(\sigma(wx+b)-t\right)\sigma'(wx+b)x$$

$$\frac{df}{db} = \left(\sigma(wx+b)-t\right)\sigma'(wx+b)$$

Consider instead the DAG



$$z = wx + b$$
$$y = \sigma(z)$$
$$f = \frac{1}{2}(y-t)^2$$

$$\frac{df}{dy} = y - t$$

$$\frac{dy}{dz} = \sigma'(z)$$

$$\frac{df}{dz} = \frac{df}{dy}\frac{dy}{dz} = (y-t)\sigma'(z)$$

$$\frac{dz}{dw} = x$$

$$\frac{dz}{db} = 1$$

$$\frac{df}{dw} = \frac{df}{dz}\frac{dz}{dw} = (y-t)\sigma'(z)x$$

$$\frac{df}{db} = \frac{df}{dz}\frac{dz}{db} = (y-t)\sigma'(z)$$

Backpropagation for MLPs

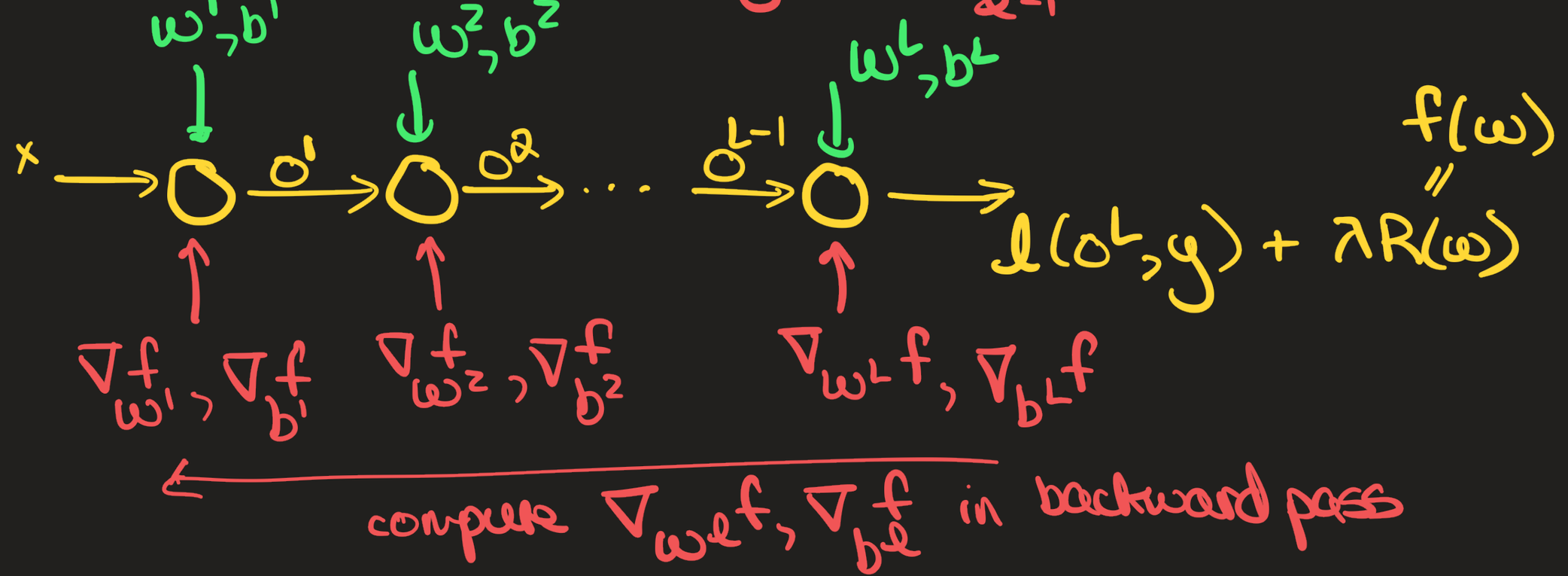
We write the DAG associated with our MLP's output

Have an L-layer MLP and

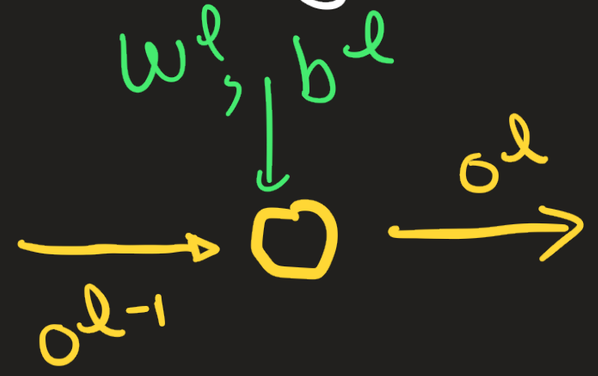
$$f(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(o^L(x_i), y_i) + \lambda \sum_{l=1}^{L} R(w^l, b^l)$$

WLOG consider just one data point

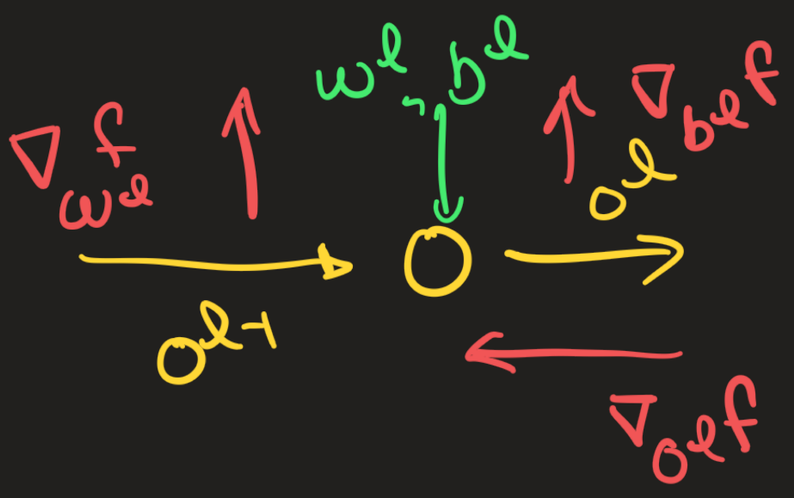
$$f(w) = \ell(o^L(x), y) + \lambda \sum_{l=1}^{L} R(w^l, b^l)$$



To compute the gradients in the backward pass, consider the l^{th} layer of neurons



If we know how f depends on the output from layer l, $\nabla_{o^l} f$, then we can compute how it depends on w^l and b^l by using the chain rule.

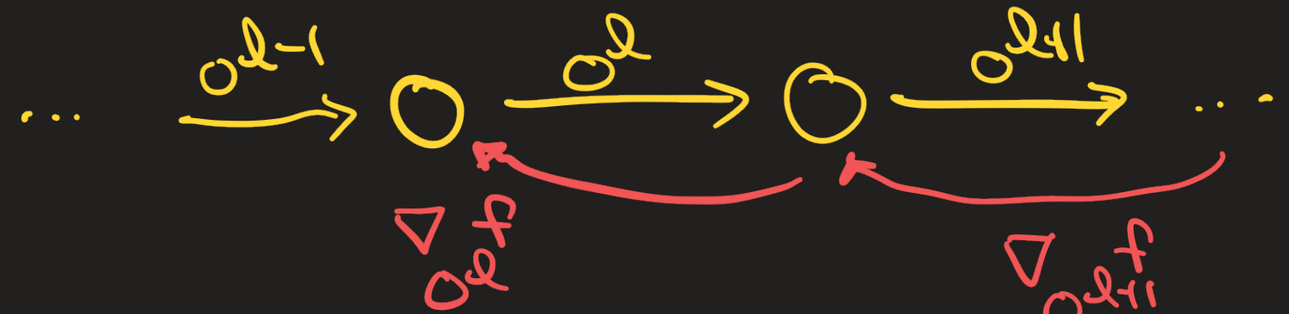


r_l is the function computed by all the layers after layer l; it takes the output of layer l as input

$$f = r_l(o^l)$$

$$o^l = \sigma(w^l o^{l-1} + b^l)$$

Similarly, if we know how f depends on o^{l+1}, $\nabla_{o^{l+1}} f$, then we can compute how it depends on o^l, $\nabla_{o^l} f$



Again this follows from the chain rule

$$f = r_{l+1}(o^{l+1})$$

$$o^{l+1} = \sigma(w^{l+1} o^l + b^{l+1})$$

r_{l+1} is the function computed by all the layers after layer $l+1$; they take layer $(l+1)$'s output as input

Consider these two computations

forward pass:

$$f = r_l(o^l)$$

$$o^l = \sigma(w^l o^{l-1} + b^l)$$

in backward pass want: $\nabla_{w^l} f$, $\nabla_{b^l} f$

forward pass:

$$f = r_{l+1}(o^{l+1})$$

$$o^{l+1} = \sigma(w^{l+1} o^l + b^{l+1})$$

in backward pass want: $\nabla_{o^l} f$

In both cases we have a system of the form

$$f = r(h(z)) \quad \text{where} \quad r: \mathbb{R}^n \to \mathbb{R}$$

$$h: \mathbb{R}^p \to \mathbb{R}^n$$

and want to compute $\nabla_z f = \nabla_z r(h(z))$

e.g.

$$r = r_l : \mathbb{R}^{n_l} \to \mathbb{R}$$

$$h : w^l \mapsto o^l = \sigma(w^l o^{l-1} + b^l)$$

$$h: \mathbb{R}^{n_l \times n_{l-1}} \to \mathbb{R}^{n_l}$$

when we want to compute $\nabla_{w^l} f$

Multivariate Chain Rule

Consider $h: \mathbb{R}^p \to \mathbb{R}^n$ and $r: \mathbb{R}^n \to \mathbb{R}$

Define $f(z) = r(h(z))$

The Chain Rule gives

$$\nabla_z f(z) = J_h(z)^T (\nabla r)(h(z))$$

where

$$J_h(z) = \left[\frac{\partial h_i}{\partial z_j} \right]_{\substack{i=1,\dots,n \\ j=1,\dots,p}} \in \mathbb{R}^{n \times p}$$

is the Jacobian of h at z.

Return to backward pass

First key observation If we can compute

$$\nabla_{o^\ell} f = \left[\frac{\partial f}{\partial (o^\ell)_i}\right]_{i=1}^{n_\ell}$$

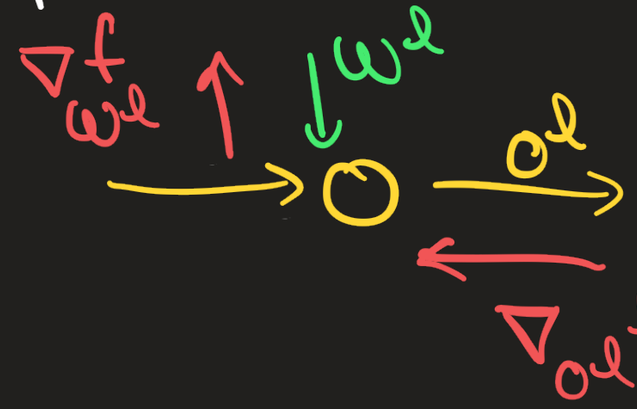
then the chain rule allows us to compute

$$\nabla_{w^\ell} f = J_{o^\ell}(w^\ell)^T \nabla_{o^\ell} f + \lambda \nabla_{w^\ell} R_\ell$$

this part is from the chain rule

this part is just the deriv of the regularizer

This computes the backward pass



and similarly the backward pass to determine $\nabla_{b^\ell} f$ is

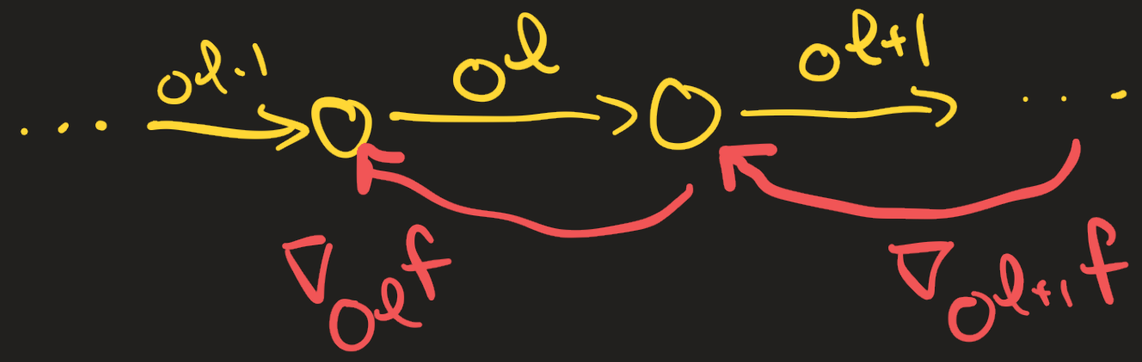
$$\nabla_{b^\ell} f = J_{o^\ell}(b^\ell)^T \nabla_{o^\ell} f + \lambda \nabla_{b^\ell} R_\ell$$

Second key observation

Since f depends on o^ℓ only through $o^{\ell+1}$, we have by the chain rule that

$$\nabla_{o^\ell} f = J_{o^{\ell+1}}(o^\ell)^T \nabla_{o^{\ell+1}} f$$

This computes the backward pass



Putting these pieces together, we see that we can recursively compute $\nabla_{W_\ell} f$ and $\nabla_{b_\ell} f$ as follows in a single backward pass

Backprop Algorithm for MLPs

$$\nabla_{b_L} f = J_{O_L}(b^L)^T \nabla_{O_L} f + \lambda \nabla_{b_L} R_L$$

$$\nabla_{W_L} f = J_{O_L}(W^L)^T \nabla_{O_L} f + \lambda \nabla_{W_L} R_L$$

for $\ell = L-1, \ldots, 1$:

$$\nabla_{O_\ell} f \leftarrow J_{O_{\ell+1}}(O^\ell)^T \nabla_{O_{\ell+1}} f$$

$$\nabla_{b_\ell} f \leftarrow J_{O_\ell}(b^\ell)^T \nabla_{O_\ell} f + \lambda \nabla_{b_\ell} R_\ell$$

$$\nabla_{W_\ell} f \leftarrow J_{O_\ell}(W^\ell)^T \nabla_{O_\ell} f + \lambda \nabla_{W_\ell} R_\ell$$