ML and Optimization Lecture 15

- Autoencoders (MLPs)
- Backpropagation (aka chain rule)
- Pytorch autoencoder example

Autoencoders

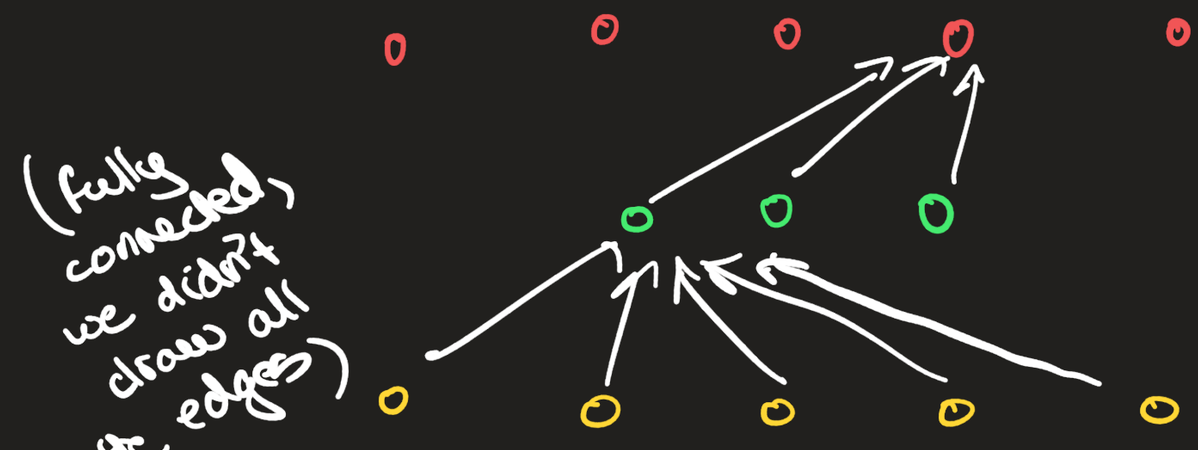
- nonlinear generalization of PCA / SVD
- find "good" low-dimensional encodings of your input signal (encoder)
- measure "good" by how well the signal can be reconstructed (auto)

this is an unsupervised method

Learning problem:

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

simple AE architecture (MLP)



output layer
$n_2 = d$

encoding layer
$n_1 = o(d)$

Input Layer
$n_0 = d$

our encoder layer has output

$$E(x) = \sigma(W^1 x + b^1)$$

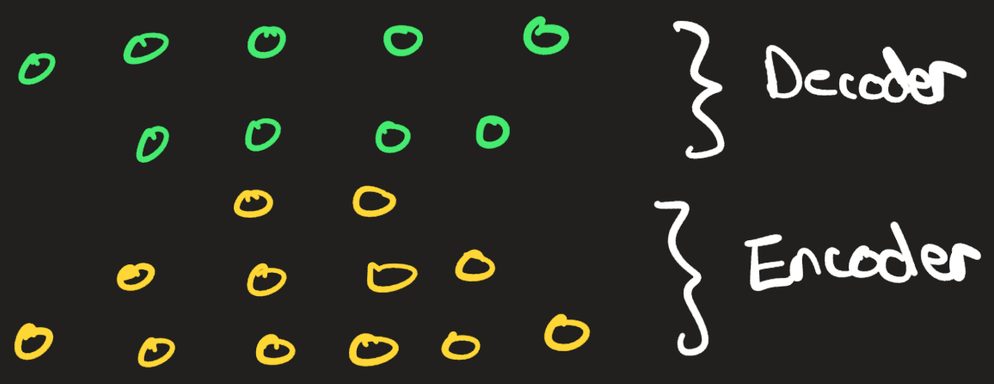
and the decoder layer has output

$$D(z) = \sigma(W^2 z + b^2)$$

The associated RERM problem is

$$\min_{W^1, b^1, W^2, b^2} \frac{1}{n} \sum_{i=1}^{n} \| x_i - \sigma(W^2 \sigma(W^1 x_i + b^1) + b^2) \|_2^2 + \lambda \left[\| W^1 \|_F^2 + \| W^2 \|_F^2 \right]$$

Typically AEs progressively encode/decode so E and D are more complicated/expressive nonlinear functions



Backpropagation

- Chain rule systematically applied 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:

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

then

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

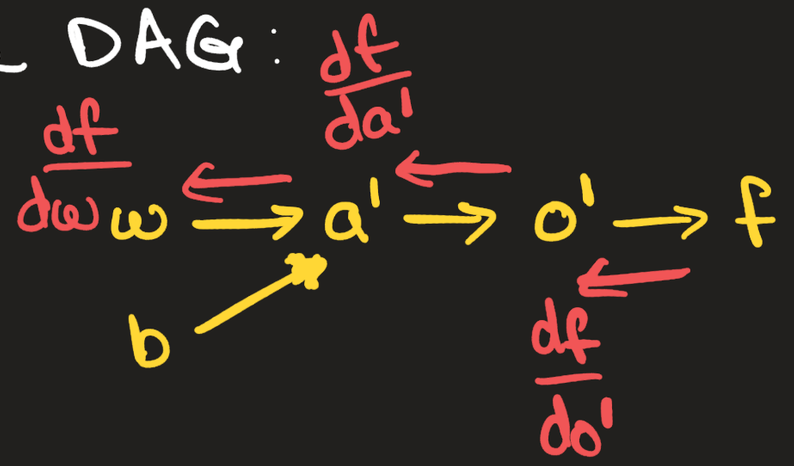
$$\frac{df}{db} = (\sigma(wx+b)-t)\sigma'(wx+b)$$

Consider instead the DAG:

$$a' = wx + b$$

$$o' = \sigma(a')$$

$$f(w,b) = \frac{1}{2}(o' - t)^2$$

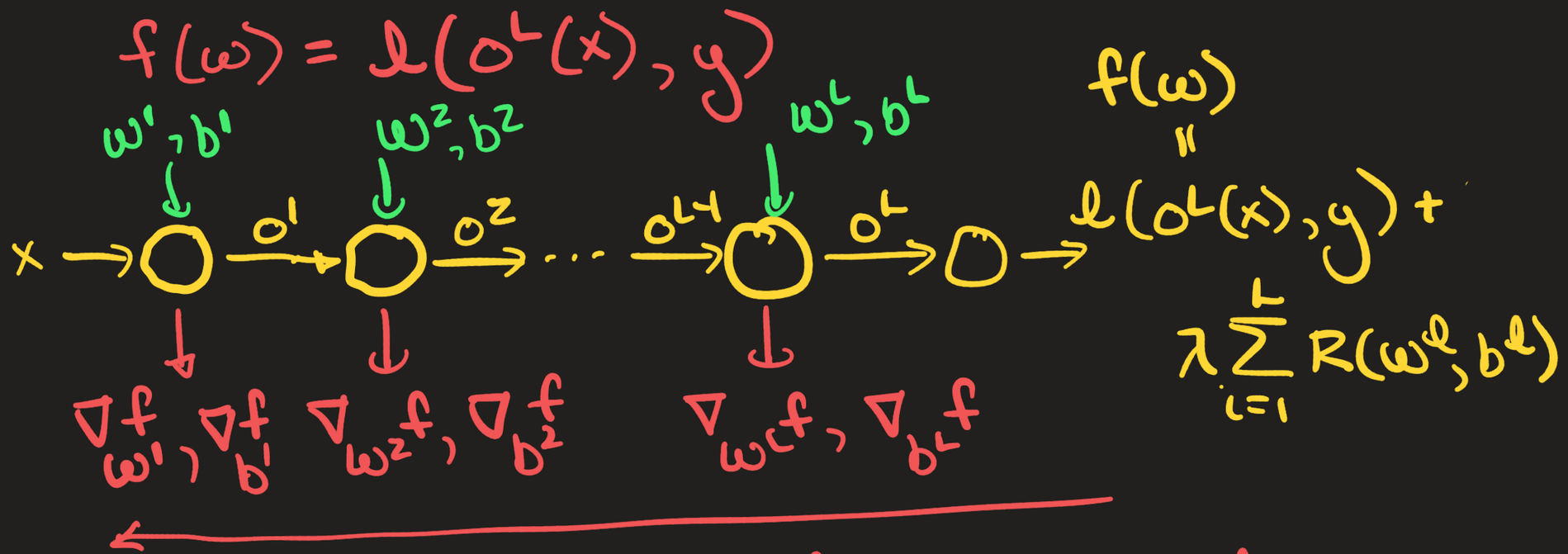


$$\frac{df}{dw} = \frac{df}{do'}\frac{do'}{dw} = \underbrace{\frac{df}{do'}\frac{do'}{da'}}_{\frac{df}{da'}}\frac{da'}{dw}$$

$$\frac{df}{db} = \frac{df}{do'}\frac{do'}{db} = \underbrace{\frac{df}{do'}\frac{do'}{da'}}_{\frac{df}{da'}}\frac{da'}{db}$$

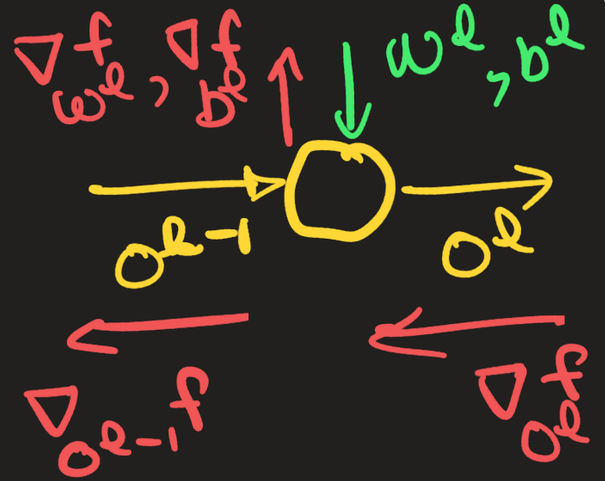
Backpropagation for MLPs

We write the DAG associated with our MLP's output. We assume we have an L-layer MLP and one input point x



compute $\nabla_{w^l} f, \nabla_{b^l} f$ in a backward pass using the chain rule

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



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

By the chain rule, if we know $\nabla_{o^l} f$, then we can compute:

1) $\nabla_{o^{l-1}} f$ — derivatives w.r.t. output of previous layer
2) $\nabla_{w^l} f, \nabla_{b^l} f$ — derivatives w.r.t. parameters of layer l

Chain-rule for vector-valued functions

If f is a scalar function: $f: \mathbb{R}^p \to \mathbb{R}$

$$f(x+h) \simeq f(x) + \langle \nabla f(x), h \rangle$$

Consider instead $f = \begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix} : \mathbb{R}^p \to \mathbb{R}^m$

$$f(x+h) = \begin{bmatrix} f_1(x+h) \\ \vdots \\ f_m(x+h) \end{bmatrix} \simeq f(x) + \begin{bmatrix} \langle \nabla f_1(x), h \rangle \\ \vdots \\ \langle \nabla f_m(x), h \rangle \end{bmatrix}$$

$$= f(x) + J_f(x) \cdot h$$

where $J_f(x) = \begin{bmatrix} \nabla f_1(x)^T \\ \vdots \\ \nabla f_m(x)^T \end{bmatrix} \in \mathbb{R}^{m \times p}$ is the <u>Jacobian</u> of f at x

Multivariate Chain Rule

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

Define

$$f(\omega) = r(h(\omega))$$

The Chain Rule gives

$$\nabla_\omega f(\omega) = J_h(\omega)^T \nabla r(h(\omega))$$

where

$$J_h(\omega) = \left[\frac{\partial h_i(\omega)}{\partial \omega_j} \right]_{i,j}$$ is the Jacobian of h at ω

Return to backward pass

First application of chain rule

Use $f(w^\ell) = f(o^\ell(w^\ell))$

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

then the chain rule allows us to compute

$$\nabla_{w^\ell} f = \underbrace{J_{o^\ell}(w^\ell)^T \nabla_{o^\ell} f}_{\text{gradient of NN output}} + \underbrace{\lambda \nabla_{w^\ell} R(w^\ell)}_{\text{gradient of regularizer}}$$

and similarly

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

Second application of chain rule

Use $f(o^{l-1}) = f(o^l(o^{l-1}))$

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

$$\nabla_{o^{l-1}} f = J_{o^l}(o^{l-1})^T \nabla_{o^l} f$$

Backprop Algorithm (for MCPs)

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

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

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

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

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

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