ML and Optimization Lecture 16

- Autoencoders (MLPs)
- Backpropagation (aka chain rule)
- Pytorch MLP example on Fashion MNIST
Autoencoders

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

This is an unsupervised method

Learning problem:

$$\min_{E,D} \mathbb{E}_{x \sim P} \| x - D(E(x)) \|^2 + \lambda R(E,D)$$
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 RERN problem is

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

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:

\[ f(\omega, b) = \frac{1}{2} (\sigma(\omega x + b) - t)^2 \]

then

\[
\frac{df}{d\omega} = (\sigma(\omega x + b) - t) \sigma'(\omega x + b) x \\
\frac{df}{db} = (\sigma(\omega x + b) - t) \sigma'(\omega x + b)
\]
Consider instead the DAG:

\[ a' = wx + b \]
\[ o' = \sigma(a') \]
\[ f(w_o) = \frac{1}{2}(o' - t)^2 \]

\[
\frac{df}{dw} = \frac{df}{do'} \frac{do'}{dw} = \frac{df}{do'} \frac{do'}{da'} \frac{da'}{dw} \]
\[
\frac{df}{da'}
\]

\[
\frac{df}{db} = \frac{df}{do'} \frac{do'}{db} = \frac{df}{do'} \frac{do'}{da'} \frac{da'}{db} \]
\[
\frac{df}{da'}
\]
Backpropagation for MLPs

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

$$f(\omega) = l(o^L(x), y)$$

- $\omega^1, b^1$
- $\omega^2, b^2$
- $\omega^L, b^L$

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

- $\nabla f_{\omega^i}, \nabla f_{b^i}$
- $\nabla w^i f, \nabla b^i f$
- $\nabla w^L f, \nabla b^L f$

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

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

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

2) $\nabla_{\mathbf{w}^l}, \nabla_{\mathbf{b}^l}$ — derivatives w.r.t. parameters of layer $l$
Chain-rule for vector-valued functions

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

\[
f(x+h) \approx f(x) + \langle \nabla f(x), h \rangle
\]

Consider instead \( f = \begin{bmatrix} f_1 & \cdots & f_m \end{bmatrix} : \mathbb{R}^p \to \mathbb{R}^m \)

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

\[
= f(x) + \mathbf{J}_f(x) \cdot h
\]

where \( \mathbf{J}_f(x) = \begin{bmatrix} \nabla f_1(x)^T \\ \vdots \\ \nabla f_m(x)^T \end{bmatrix} \in \mathbb{R}^{m \times p} \) is the Jacobian 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 \( \omega \)
Return to backward pass

First application of chain rule

If we know \( \nabla_{\text{o}_0} f = \left[ \frac{\partial f}{\partial (\text{o}_i)} \right]_{i=1}^{n} \)

then the chain rule allows us to compute

\[
\nabla_{\text{w}} f = J_{\text{o}_0} (\text{w}_0) ^T \nabla_{\text{o}_0} f + \lambda \nabla_{\text{w}_0} R(\text{w}_0) \]

and similarly

\[
\nabla_{\text{b}} f = J_{\text{o}_0} (\text{b}_0) ^T \nabla_{\text{o}_0} f + \lambda \nabla_{\text{b}_0} R(\text{b}_0) \]
Second application of chain rule

Since \( t \) depends on \( \alpha^{t-1} \) only through \( \alpha \),
we have by the chain rule that

\[
\nabla_{\alpha^{t-1}} f = J_{\alpha^{t-1}} (\alpha^{t-1})^T \nabla_{\alpha} f
\]
Backprop Algorithm (for MCPs)

\[ \nabla_{\theta_L} f = J_{o_L} (b_L) (b_L)^T \nabla_{o_L} f + \lambda \nabla_{b_L} R(b_L) \]
\[ \nabla_{w_L} f = J_{o_L} (w_L) (w_L)^T \nabla_{o_L} f + \lambda \nabla_{w_L} R(w_L) \]

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

\[ \nabla_{o_L} f \leftarrow J_{o_{L+1}} (o_{L+1}) (o_{L+1})^T \nabla_{o_{L+1}} f \]
\[ \nabla_{b_L} f \leftarrow J_{o_L} (b_L) (b_L)^T \nabla_{o_L} f + \lambda \nabla_{b_L} R(b_L) \]
\[ \nabla_{w_L} f \leftarrow J_{o_L} (w_L) (w_L)^T \nabla_{o_L} f + \lambda \nabla_{w_L} R(w_L) \]