Neural networks
Training neural networks - backpropagation algorithm
Topics: stochastic gradient descent (SGD)

• Algorithm that performs updates after each example
  ‣ initialize $\theta$ ( $\theta \equiv \{W^{(1)}, b^{(1)}, \ldots, W^{(L+1)}, b^{(L+1)}\}$
  ‣ for N iterations
    - for each training example $(x^{(t)}, y^{(t)})$
      \[
      \begin{align*}
      \Delta &= -\nabla_\theta l(f(x^{(t)}; \theta), y^{(t)}) - \lambda \nabla_\theta \Omega(\theta) \\
      \theta &\leftarrow \theta + \alpha \Delta
      \end{align*}
      \]
    \} \text{ training epoch} \}
      \text{ iteration over all examples}

• To apply this algorithm to neural network training, we need
  ‣ the loss function $l(f(x^{(t)}; \theta), y^{(t)})$
  ‣ a procedure to compute the parameter gradients $\nabla_\theta l(f(x^{(t)}; \theta), y^{(t)})$
  ‣ the regularizer $\Omega(\theta)$ (and the gradient $\nabla_\theta \Omega(\theta)$)
  ‣ initialization method
**BACKPROPAGATION**

**Topics:** backpropagation algorithm

- This assumes a forward propagation has been made before
  - compute output gradient (before activation)
    \[ \nabla a^{(L+1)}(x) - \log f(x)_y \iff -(e(y) - f(x)) \]
  - for \( k \) from \( L+1 \) to \( 1 \)
    - compute gradients of hidden layer parameter
      \[ \nabla w^{(k)} - \log f(x)_y \iff (\nabla a^{(k)}(x) - \log f(x)_y) \ h^{(k-1)}(x)^\top \]
      \[ \nabla b^{(k)} - \log f(x)_y \iff \nabla a^{(k)}(x) - \log f(x)_y \]
    - compute gradient of hidden layer below
      \[ \nabla h^{(k-1)}(x) - \log f(x)_y \iff W^{(k)}\top (\nabla a^{(k)}(x) - \log f(x)_y) \]
    - compute gradient of hidden layer below (before activation)
      \[ \nabla a^{(k-1)}(x) - \log f(x)_y \iff (\nabla h^{(k-1)}(x) - \log f(x)_y) \odot [\ldots, g'(a^{(k-1)}(x)_j), \ldots] \]
Topics: flow graph

- Forward propagation can be represented as an acyclic flow graph
- It's a nice way of implementing forward propagation in a modular way
  - each box could be an object with an fprop method, that computes the value of the box given its children
  - calling the fprop method of each box in the right order yield forward propagation
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each children
  - fprop depends on the fprop of a box’s children, while bprop depends the bprop of a box’s parents

- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
Topics: automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each children
  - fprop depends on the fprop of a box’s children, while bprop depends the bprop of a box’s parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each children
  - fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each children
  - fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents

- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
Topics: automatic differentiation

• Each object also has a bprop method
  ‣ it computes the gradient of the loss with respect to each children
  ‣ fprop depends on the fprop of a box’s children, while bprop depends the bprop of a box’s parents

• By calling bprop in the reverse order, we get backpropagation
  ‣ only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each children
  - fprop depends on the fprop of a box’s children, while bprop depends the bprop of a box’s parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each children
  - fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents

- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
Topics: automatic differentiation
• Each object also has a bprop method
  ‣ it computes the gradient of the loss with respect to each children
  ‣ fprop depends on the fprop of a box’s children, while bprop depends the bprop of a box’s parents
• By calling bprop in the reverse order, we get backpropagation
  ‣ only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each children
  - fprop depends on the fprop of a box’s children, while bprop depends the bprop of a box’s parents

- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
Topics: finite difference approximation

- To debug your implementation of fprop/bprop, you can compare with a finite-difference approximation of the gradient

\[
\frac{\partial f(x)}{\partial x} \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}
\]

- \( f(x) \) would be the loss
- \( x \) would be a parameter
- \( f(x + \epsilon) \) would be the loss if you add \( \epsilon \) to the parameter
- \( f(x - \epsilon) \) would be the loss if you subtract \( \epsilon \) to the parameter