Backup plans:

If this Zoom meeting ends prematurely, five-minute break, check Piazza
General Case: Abstract Representation of a Neuron

\[
a(x) = g(w \cdot \bar{x})
\]

Diagram:

- "activation" \( a \)
- "bias" \( w_0 \)
- \( x_1, \ldots, x_n \)
- \( w_1, \ldots, w_n \)
General Case: Abstract Representation of a Neuron

\[ a(\bar{x}) = g(\bar{w} \cdot \bar{x}) \]

\[ w_j \leftarrow w_j + \alpha g'(\bar{w} \cdot \bar{x}) (y_i - g(\bar{w} \cdot \bar{x})) x_{ij} \]
Multi-Layer Neural Network

\[ W_{0n+1} \rightarrow o_1 \quad \cdots \quad W_{0k} \rightarrow o_j \]

\[ w_{n+1o1} \rightarrow n+1 \quad \cdots \quad w_{ko1} \rightarrow k \]

\[ W_{1n+1} \rightarrow 1 \quad \cdots \quad W_{nk} \rightarrow n \]
Multi-Layer Neural Network

Can have arbitrary number of layers
Multi-Layer Neural Network

• Increases our “representational power”

• Learning the weights requires something different

  Can compute the derivative of $\frac{\partial}{\partial w_{ij}} \text{Error}_D(\bar{w})$ for each $w_{ij}$

  Computing $\frac{\partial}{\partial w_{ij}} \text{Error}_D(\bar{w})$ is more complex for lower levels
Multi-Layer Neural Network

• Increases our “representational power”

• Learning the weights requires something different

  Can compute the derivative of $\frac{\partial}{\partial w_{ij}} \text{Error}_D (\bar{W})$ for each $w_{ij}$

  Computing $\frac{\partial}{\partial w_{ij}} \text{Error}_D (\bar{W})$ is more complex for lower levels

  Harness the chain rule
Using Chain Rule for Learning

\[ f(x) = g(h(x)) \]
\[ f'(x) = g'(h(x))h'(x) \]
Using Chain Rule for Learning

\[ f(x) = g(h(x)) \]

\[ f'(x) = g'(h(x))h'(x) \]

Neural net version: neuron \( z \) with \( k \) input neurons \( a_1(w_{ij}), \ldots, a_k(w_{ij}) \)

\[ a_z(w_{ij}) = g\left(\sum_{l=1}^{k} w_{lz}a_l(w_{ij})\right) \]
Explanatory note given question from lecture:
I’m writing these as a function of $w_{ij}$ rather than of $\bar{w} \cdot \bar{x}$ because we’re taking the partial derivative with respect to $w_{ij}$ and, rather than writing $\frac{\partial}{\partial w_{ij}}$ everywhere, I’m showing that this partial derivative treats $w_{ij}$ as the only variable and treats the other weights in $\bar{w} \cdot \bar{x}$ as constants. This lets us write derivatives as, for example, $a_z'(w_{ij})$ rather than $\frac{\partial}{\partial w_{ij}} (\bar{w} \cdot \bar{x})$.

Neural net version: neuron $z$ with $k$ input neurons $a_1(w_{ij}), ..., a_k(w_{ij})$

$$a_z(w_{ij}) = g \left( \sum_{l=1}^{k} w_{lz} a_l(w_{ij}) \right)$$
Using Chain Rule for Learning

\[ f(x) = g(h(x)) \]

\[ f'(x) = g'(h(x))h'(x) \]

Neural net version: neuron \( z \) with \( k \) input neurons \( a_1(w_{ij}), ..., a_k(w_{ij}) \)

\[ a_z(w_{ij}) = g \left( \sum_{l=1}^{k} w_{lz}a_l(w_{ij}) \right) \]

\[ a_z'(w_{ij}) = g' \left( \sum_{l=1}^{k} w_{lz}a_l(w_{ij}) \right) \left( \sum_{l=1}^{k} w_{lz}a_l'(w_{ij}) \right) \]
Using Chain Rule for Learning

\[ a'_z(w_{ij}) = g' \left( \sum_{l=1}^{k} w_{lz} a_l(w_{ij}) \right) \left( \sum_{l=1}^{k} w_{lz} a'_l(w_{ij}) \right) \]
Using Chain Rule for Learning

\[ a_z'(w_{ij}) = g'(\sum_{l=1}^{k} w_{lz} a_l(w_{ij})) \left( \sum_{l=1}^{k} w_{lz} a_l'(w_{ij}) \right) \]

Computed in order to generate top-level output
Using Chain Rule for Learning

\[ a_z'(w_{ij}) = g' \left( \sum_{l=1}^{k} w_{lz} a_l(w_{ij}) \right) \left( \sum_{l=1}^{k} w_{lz} a_l'(w_{ij}) \right) \]

If \( g'(z) = g(z)(1 - g(z)) \) or \( g'(z) = (1 - g^2(z)) \)

\( g(z) \) is computed in order to generate top-level output
Using Chain Rule for Learning

$$a_z'(w_{ij}) = g'\left(\sum_{l=1}^{k} w_{lz}a_l(w_{ij})\right)\left(\sum_{l=1}^{k} w_{lz}a_l'(w_{ij})\right)$$

Same is true recursively
Backpropagation Algorithm Outline

1. Compute the network’s outputs starting from inputs and moving upward.

2. Compute the weight updates starting from the outputs and moving downwards.
Backprop: Notation

- $k$ layers
- Each neuron is labeled by 1, 2, ...
- Neurons in layer $l$ denoted by $L_l$
- Input layer: Layer 1, $L_1 = \{1, \ldots, n\}$, and $a_j = x_j$
- Output layer: Layer $k$, $L_k$
- Weight between neuron $i$ and $j$: $w_{ij}$
- Bias weight for neuron $j$: $w_{0j}$
- Matrix of weights: $W$ (only a subset are really weights)
- $\Delta_i$: Corresponds to $(y_i - g(\bar{w} \cdot \bar{x}_i))$ for internal neurons
Backprop: High Level

Backprop(D,W):

Initialize all weights $w_{ij}$ to small random numbers

Repeat

For each example $(\bar{x}, \bar{y})$

Feedforward:

For m=2 to k

Compute the activations for neurons at level m from those at level m-1

Backpropagation:

For m=k down to 2

Compute new weights for neurons between levels m-1 and m

using “gradient descent”

Until <stopping criterion>
Backprop: High Level

Backprop(D,W):
  Initialize all weights $w_{ij}$ to small random numbers
  Repeat
    For each example $(\vec{x}, \vec{y})$
      Feedforward:
        For m=2 to k
          Compute the activations for neurons at level m from those at level m-1
      Backpropagation:
        For m=k down to 2
          Compute new weights for neurons between levels m-1 and m
            using “gradient descent”
    Until <stopping criterion>
Backprop: Feedfoward

For $m = 2$ to $k$

For each node $j$ in $L_m$

\[
\text{in}_j \leftarrow \sum_{\text{node } i \text{ into } j } w_{ij} a_i
\]

\[
a_j \leftarrow g(\text{in}_j)
\]
Backprop: High Level

Backprop(D,W):

Initialize all weights \( w_{ij} \) to small random numbers

Repeat

For each example \((\bar{x}, \bar{y})\)

Feedforward:

For m=2 to k

Compute the activations for neurons at level m from those at level m-1

Backpropagation:

For m=k down to 2

Compute new weights for neurons between levels m-1 and m using “gradient descent”

Until <stopping criterion>
Backprop: Backpropagation

For each node $j$ in $L_k$

$$\Delta_j \leftarrow g'(in_j)(y_j - a_j)$$

For $m = k-1$ down to 2

For each node $i$ in $L_m$

$$\Delta_i \leftarrow g'(in_i) \sum_{\text{node } j \text{ out of } i} w_{ij} \Delta_j$$

For each weight $w_{ij}$

$$w_{ij} \leftarrow w_{ij} + \alpha a_i \Delta_j$$
Backprop: High Level

Backprop(D,W):
  Initialize all weights $w_{ij}$ to small random numbers
  Repeat
    For each example $(\bar{x}, y)$
      Feedforward:
        For m=2 to k
          Compute the activations for neurons at level m from those at level m-1
      Backpropagation:
        For m=k down to 2
          Compute new weights for neurons between levels m-1 and m using “gradient descent”
  Until <stopping criterion>