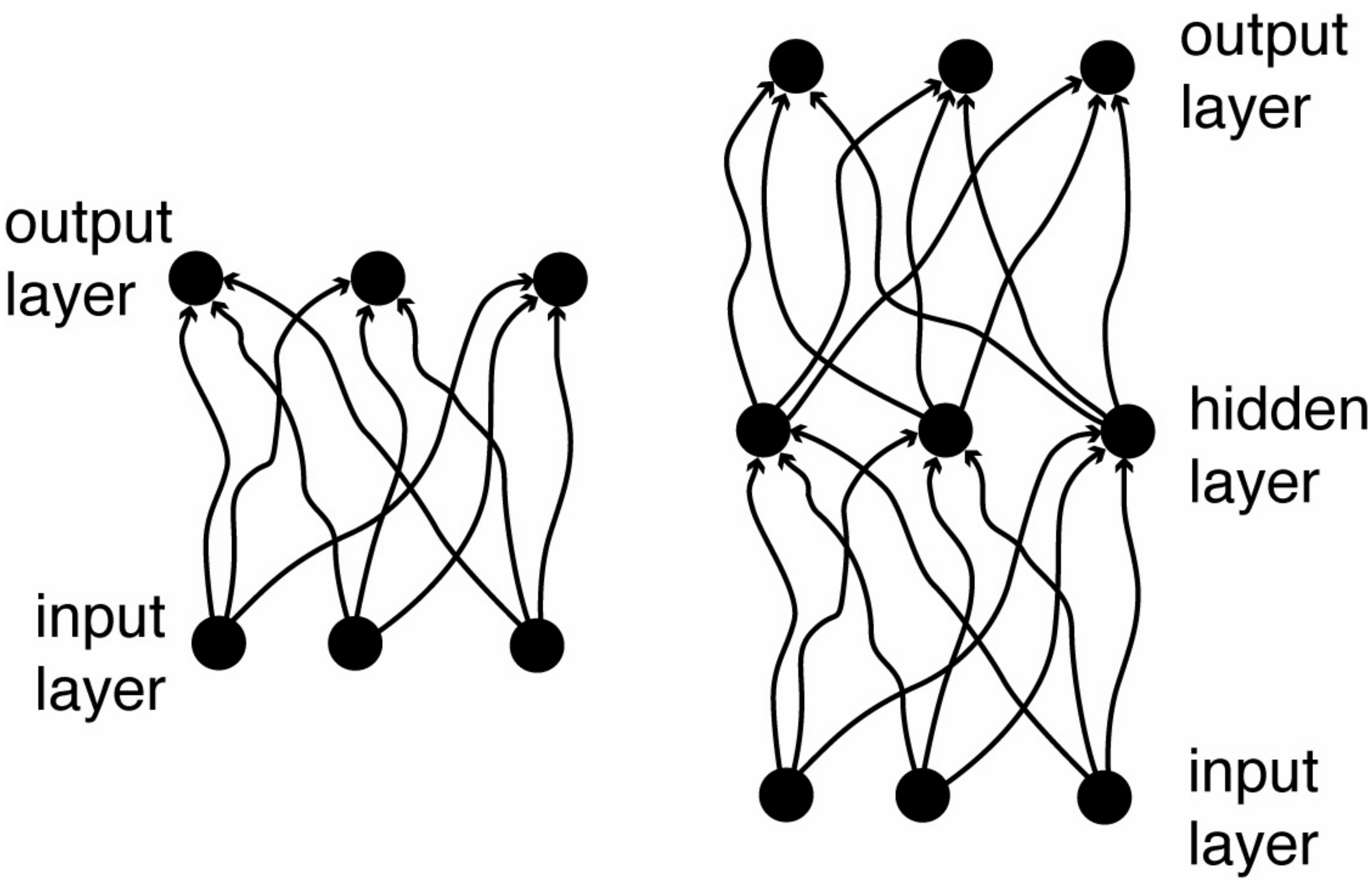
MIT Department of Brain and Cognitive Sciences
9.641J, Spring 2005 - Introduction to Neural Networks
Instructor: Professor Sebastian Seung

Backpropagation learning

Simple vs. multilayer perceptron



Hidden layer problem

- Radical change for the supervised learning problem.
- No desired values for the hidden layer.
- The network must find its own hidden layer activations.

Generalized delta rule

- Delta rule only works for the output layer.
- Backpropagation, or the generalized delta rule, is a way of creating desired values for hidden layers

Outline

- The algorithm
- Derivation as a gradient algoritihm
- Sensitivity lemma

Multilayer perceptron

- L layers of weights and biases
- L+1 layers of neurons

$$x^0 \xrightarrow{W^1, b^1} x^1 \xrightarrow{W^2, b^2} \cdots \xrightarrow{W^L, b^L} x^L$$

$$x_i{}^l = f\left(\sum_{j=1}^{n_{l-1}} W_{ij}{}^l x_j{}^{l-1} + b_i{}^l \right)$$

Reward function

- Depends on activity of the output layer only.

$$R\left(x^{L}\right)$$

- Maximize reward with respect to weights and biases.

Example: squared error

- Square of desired minus actual output, with minus sign.

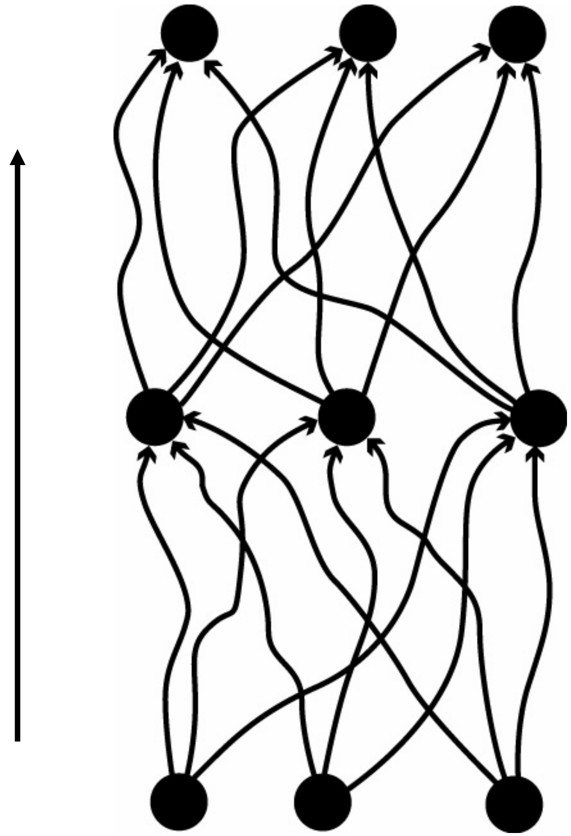
$$R\left(x^{L}\right)=-\frac{1}{2}\sum_{i=1}^{n_L}\left(d_i-x_i^{\,L}\right)^2$$

Forward pass

For $l = 1$ to L,

$$u_i^{\,l} = \sum_{j=1}^{n_{l-1}} W_{ij}^{\,l} x_j^{\,l-1} + b_i^{\,l}$$

$$x_i^{\,l} = f\left(u_i^{\,l}\right)$$



Sensitivity computation

- The sensitivity is also called “delta.”

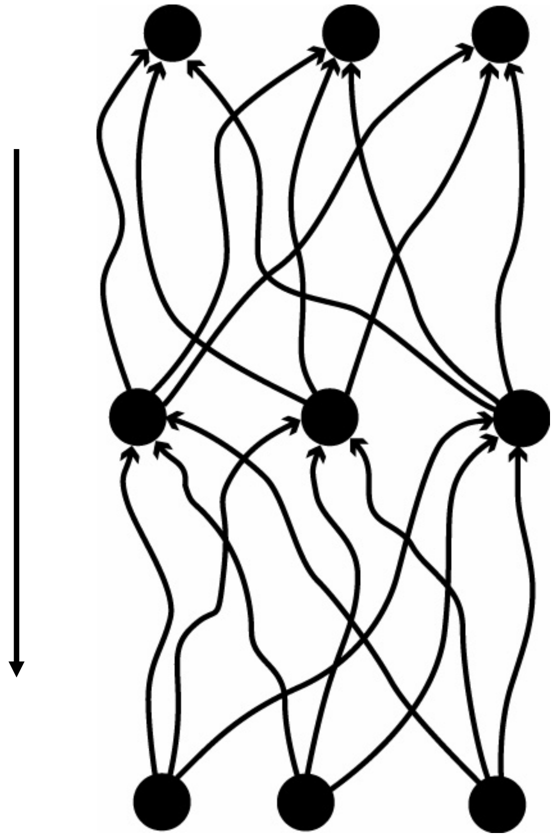
$$s_i^L = f'\left(u_i^L\right)\frac{\partial R}{\partial x_i^L}$$

$$= f'\left(u_i^L\right)\left(d_i - x_i^L\right)$$

Backward pass

$$\text{for } l = L \text{ to } 2$$

$$s_j^{\,l-1} = f'\left(u_j^{\,l-1}\right)\sum_{i=1}^{n_l} s_i^{\,l} W_{ij}^{\,l}$$



Learning update

- In any order

$$\Delta W_{ij}{}^{l} = \eta s_i{}^{l} x_j{}^{l-1}$$

$$\Delta b_i{}^{l} = \eta s_i{}^{l}$$

Backprop is a gradient update

- Consider R as function of weights and biases.

$$\frac{\partial R}{\partial W_{ij}{}^{l}} = s_i{}^{l} x_j{}^{l-1} \qquad \Delta W_{ij}{}^{l} = \eta \frac{\partial R}{\partial W_{ij}{}^{l}}$$

$$\frac{\partial R}{\partial b_i{}^{l}} = s_i{}^{l} \qquad \Delta b_i{}^{l} = \eta \frac{\partial R}{\partial b_i{}^{l}}$$

Sensitivity lemma

- Sensitivity matrix = outer product
  – sensitivity vector
  – activity vector

$$\frac{\partial R}{\partial W_{ij}^{l}} = \frac{\partial R}{\partial b_i^{l}} x_j^{l-1}$$

- The sensitivity vector is sufficient.
- Generalization of “delta.”

Coordinate transformation

$$u_i{}^l = \sum_{j=1}^{n_{l-1}} W_{ij}{}^l f\left(u_j{}^{l-1}\right) + b_i{}^l$$

$$\frac{\partial u_i{}^l}{\partial u_j{}^{l-1}} = W_{ij}{}^l f'\left(u_j{}^{l-1}\right)$$

$$\frac{\partial R}{\partial u_i{}^l} = \frac{\partial R}{\partial b_i{}^l}$$

Output layer

$$x_i^{\ L} = f\left(u_i^{\ L}\right)$$

$$u_i^{\ L} = \sum_j W_{ij}^{\ L} x_j^{\ L-1} + b_i^{\ L}$$

$$\frac{\partial R}{\partial b_i^{\ L}} = f'\left(u_i^{\ L}\right)\frac{\partial R}{\partial x_i^{\ L}}$$

Chain rule

- composition of two functions

$$u_{l-1} \to R \qquad\qquad u_{l-1} \to u_l \to R$$

$$\frac{\partial R}{\partial u_j^{\,l-1}} = \sum_i \frac{\partial R}{\partial u_i^{\,l}} \frac{\partial u_i^{\,l}}{\partial u_j^{\,l-1}}$$

$$\frac{\partial R}{\partial b_j^{\,l-1}} = \sum_i \frac{\partial R}{\partial b_i^{\,l}} W_{ij}^{\,l} f'\left(u_j^{\,l-1}\right)$$

Computational complexity

- Naïve estimate
  - network output: order N
  - each component of the gradient: order N
  - N components: order N^2
- With backprop: order N

Biological plausibility

- Local: pre- and postsynaptic variables

$$x_j^{l-1} \xrightarrow{W_{ij}^{\,l}} x_i^{\,l}, \quad s_j^{l-1} \xleftarrow{W_{ij}^{\,l}} s_i^{\,l}$$

- Forward and backward passes use same weights
- Extra set of variables

Backprop for brain modeling

- Backprop may not be a plausible account of learning in the brain.
- But perhaps the networks created by it are similar to biological neural networks.
- Zipser and Andersen:
  - train network
  - compare hidden neurons with those found in the brain.

LeNet

- Weight-sharing
- Sigmoidal neurons
- Learn binary outputs

Machine learning revolution

- Gradient following
  - or other hill-climbing methods
- Empirical error minimization