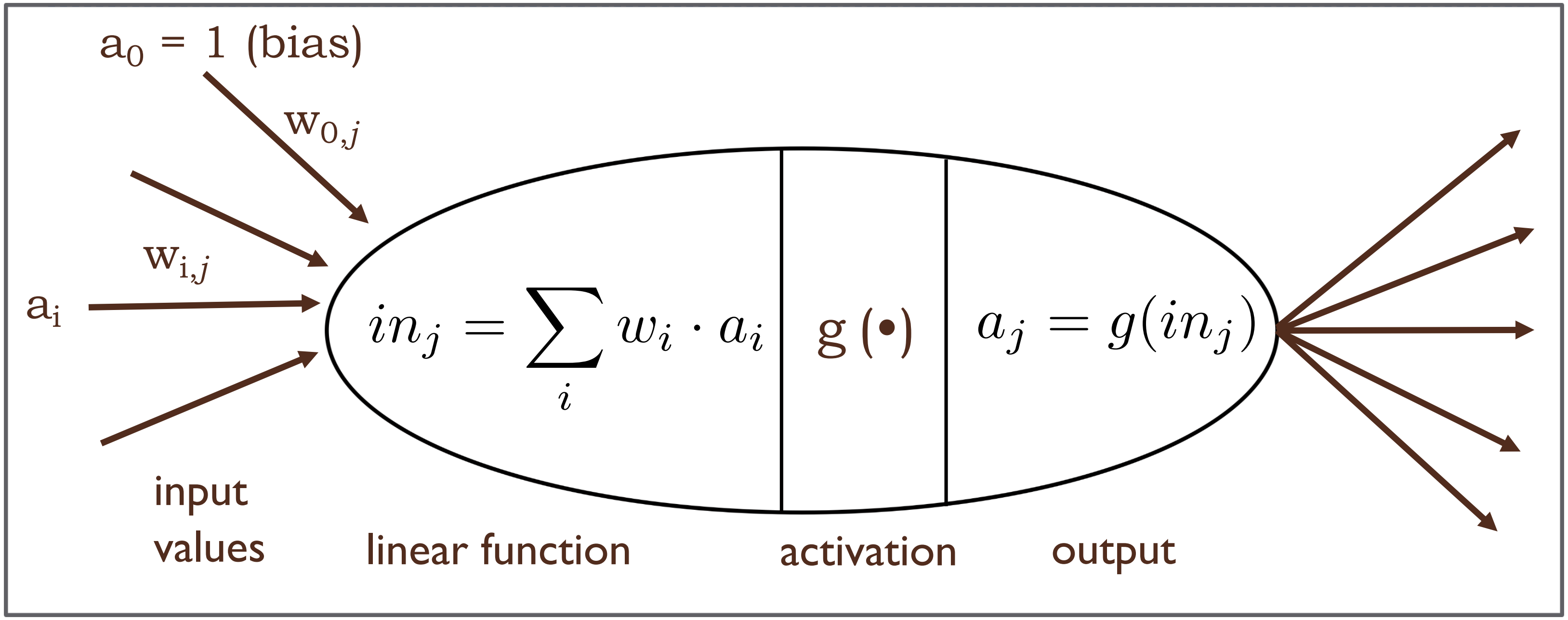
CS135
Introduction to Machine Learning

Lecture 12: Training Neural Networks

Review: Basic Neuron Model



- Each neuron has a set of weights on input connections coming into it
  - Inputs are numbers coming from other neurons, or from original input
- Neuron computes linear weighted sum of the inputs and passes that value through its activation function, g
- Output a is either passed along to other neurons, or is used as final output for the entire network

Activation Functions Everywhere!

- Logistic $f(x) = \frac{1}{1+e^{-x}}$ $\frac{\delta f}{\delta x}(x) = f(x)(1 - f(x))$

- ReLU $f(x) = \max(0, x)$ $\frac{\delta f}{\delta x}(x) = \{0,\ undef,\ 1\}$

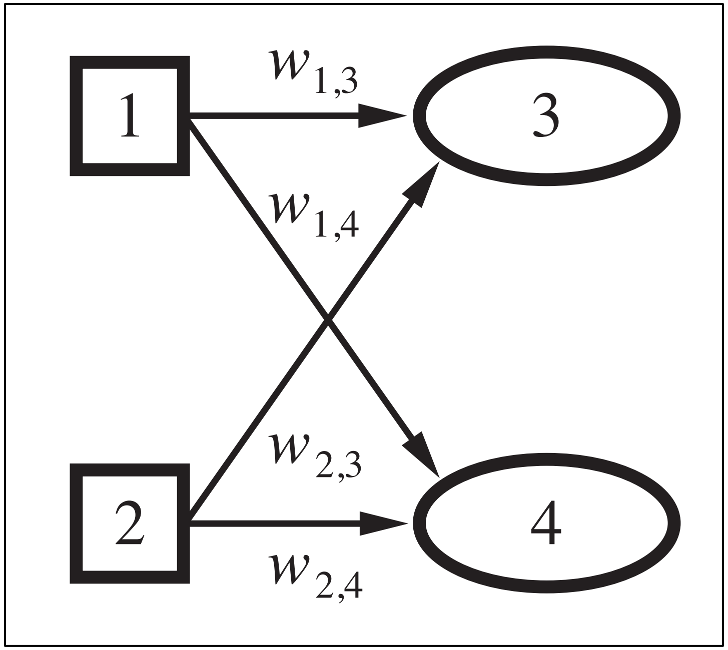
- Softplus $f(x) = \ln(1 + e^{x})$ $\frac{\delta f}{\delta x}(x) = \frac{1}{1+e^{-x}}$

- Hyperbolic Tangent $f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$ $\frac{\delta f}{\delta x}(x) = 1 - f(x)^2$

- Gaussian $f(x) = e^{-\frac{x^2}{2}}$ $\frac{\delta f}{\delta x}(x) = -x\, f(x)$

Tufts

Single-Layer Perceptron Networks



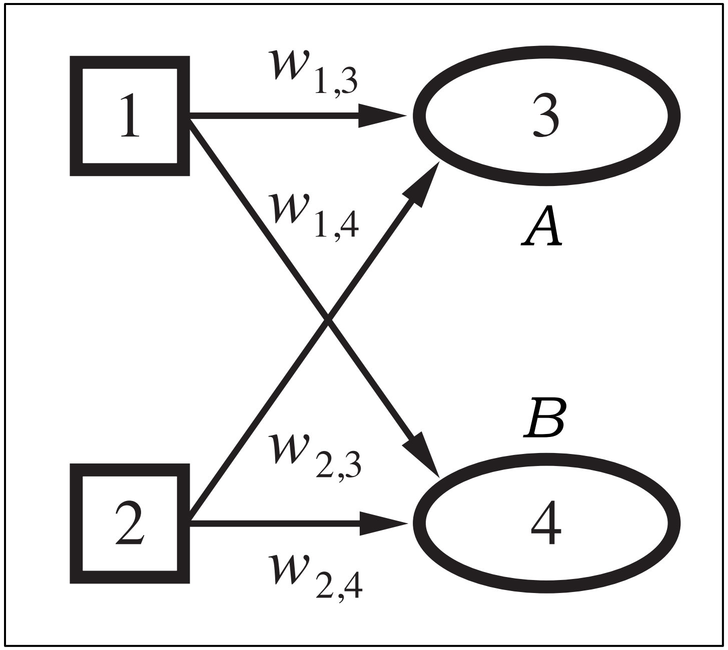
- In such a network, input features (boxes) are connected ***directly*** to a single layer of output neurons, each of which represents one possible data classification
- Each output is the result of the activation function on the weighted input features, e.g.:

$$in_3 = w_{0,3} + w_{1,3}\,x_1 + w_{2,3}\,x_2$$

$$a_3 = g(in_3) = \frac{1}{1 + e^{-in_3}}$$

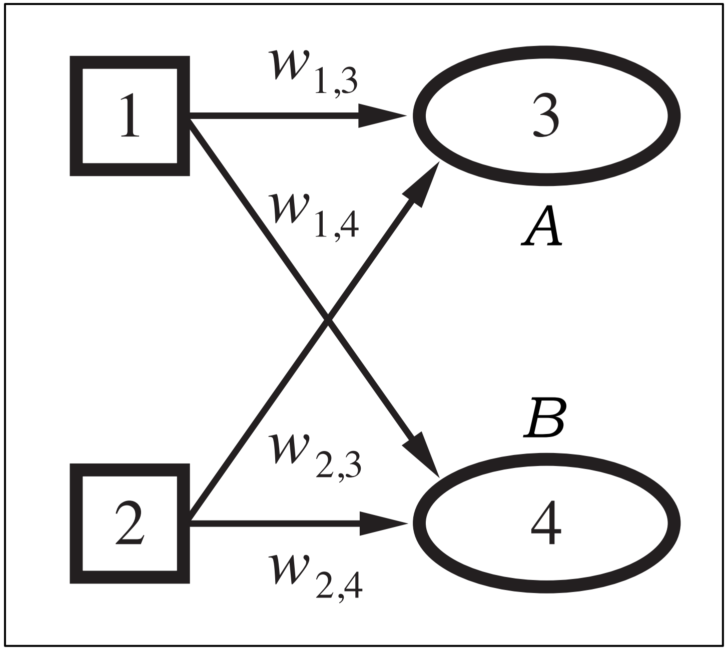
Remember: in these slides, we will use the logistic activation function as an example, but other options are possible.

Gradient Descent in Single-Layer Networks



- A small network like this has two output neurons, corresponding to two classes of data, A and B
- If the input ***is*** of A type, and the network is working properly:
  - The value output by 3 should be (close to) 1
  - The value output by 4 should be (close to) 0
- If input ***is not*** of A type, and the network is working properly:
  - The value output by 3 should be (close to) 0
  - The value output by 4 should be (close to) 1

Gradient Descent in Single-Layer Networks



Remember: neuron for class B will *also* calculate its own output and error.

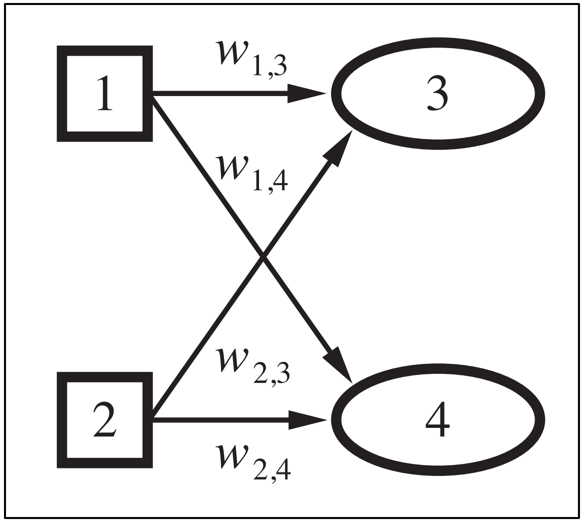
- For each of the output neurons, we can measure its error, comparing the output we get to the classification, y, that we want to see
- E.g., if neuron 3 represents class A, then if the input ***is*** of A type:

$$\begin{aligned} Err_3 &= y - g(in_3) \\ &= 1 - g(in_3) \end{aligned}$$

- If input ***is not*** of A type:

$$\begin{aligned} Err_3 &= y - g(in_3) \\ &= 0 - g(in_3) \end{aligned}$$

Gradient Descent in Single-Layer Networks

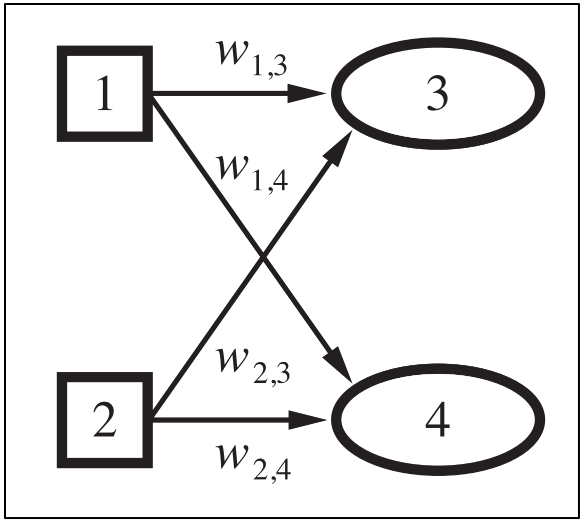


- Once we know the error, we can do a gradient-based update to get new weights for each neuron:

$$w_i \leftarrow w_i + \alpha \, Err_j \times g'(in_j) \times x_i$$

- Where we have:
  - x_i: the value of the input to that connection
  - Err_j: the error on the output (if any)
  - $g'(in_j)$: the derivative of the activation function
  - α: convergence control parameter

Gradient Descent in Single-Layer Networks



- Once we know the error, we can do a gradient-based update to get new weights for each neuron:

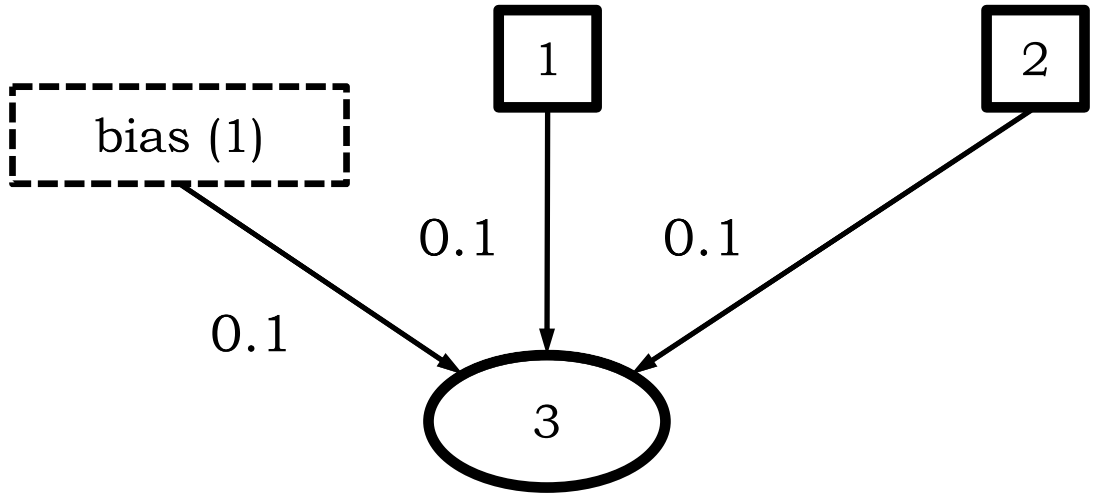
$$w_i \leftarrow w_i + \alpha \, Err_j \times g'(in_j) \times x_i$$

- E.g., neuron 3 and logistic function:

$$w_{0,3} \leftarrow w_{0,3} + \alpha \, (y - g(in_3)) \times g(in_3)(1 - g(in_3))$$
$$w_{1,3} \leftarrow w_{1,3} + \alpha \, (y - g(in_3)) \times g(in_3)(1 - g(in_3)) \times x_1$$
$$w_{2,3} \leftarrow w_{2,3} + \alpha \, (y - g(in_3)) \times g(in_3)(1 - g(in_3)) \times x_2$$

- As in linear regression and other methods, we continue making these updates until the weights ***converge***

Gradient Descent: Details

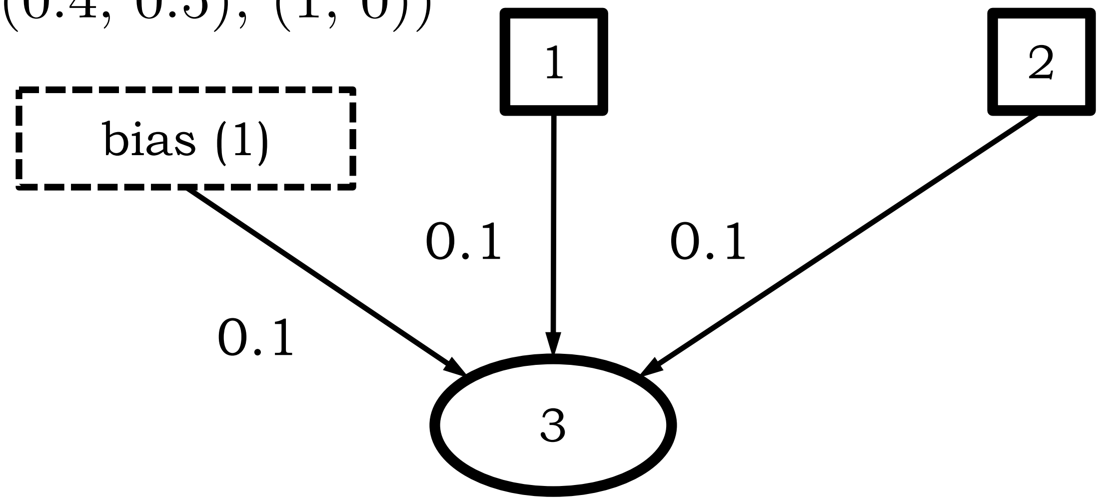


- Suppose we initialize neuron 3 with weight 0.1 on every connection, and we provide it with training example:

$$((x_1,\ x_2),\ \mathbf{y}) = (\underbrace{(0.4,\ 0.5)}_{\text{Inputs (1 \& 2)}},\ \underbrace{(1,\ 0)}_{\text{Outputs (3 \& 4)}})$$

Gradient Descent: Details

$$((x_1,\ x_2),\ \mathbf{y}) = ((0.4,\ 0.5),\ (1,\ 0))$$



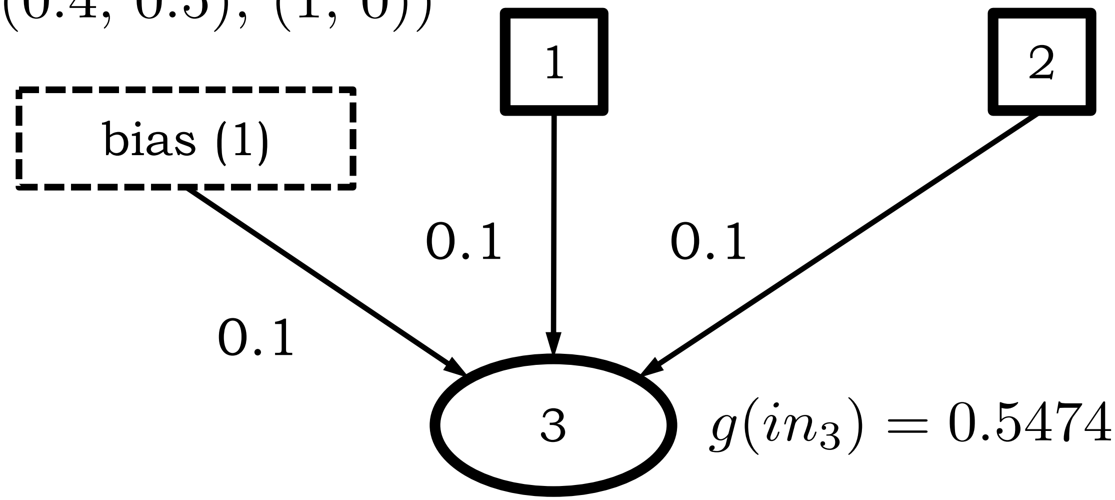
- Given these inputs, and these weights, the neuron outputs:

$$in_3 = 0.1 + (0.1 \times 0.4) + (0.1 \times 0.5) = 0.19$$

$$g(in_3) = \frac{1}{1 + e^{-0.19}} \approx 0.5474$$

Gradient Descent: Details

$$((x_1, x_2), \mathbf{y}) = ((0.4, 0.5), (1, 0))$$



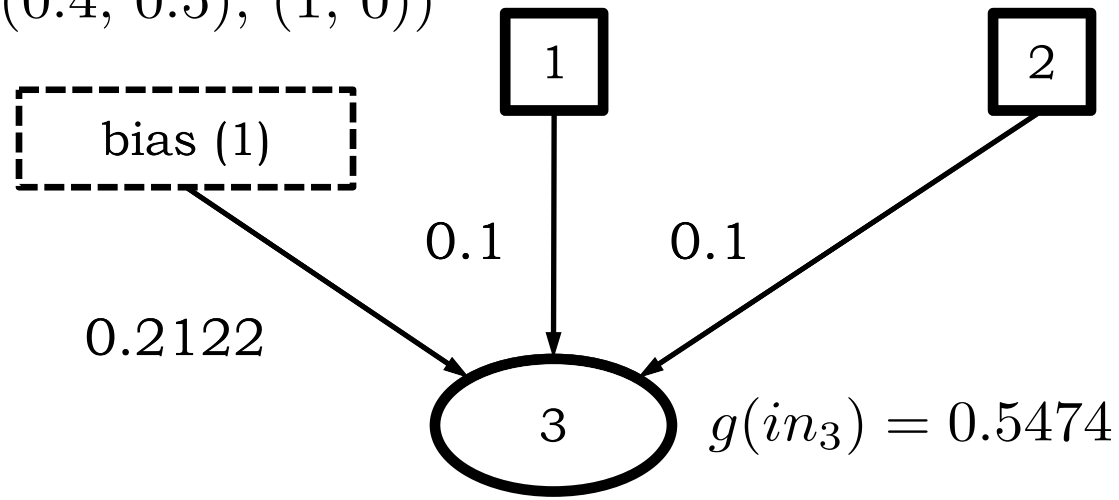
- We can now calculate error and the derivative term:

$$Err_3 = (1 - g(in_3)) = (1 - 0.5474) = 0.4526$$

$$g'(in_3) = g(in_3)(1 - g(in_3)) = (0.5474 \times 0.4526) \approx 0.2478$$

Gradient Descent: Details

$((x_1, x_2), \mathbf{y}) = ((0.4, 0.5), (1, 0))$



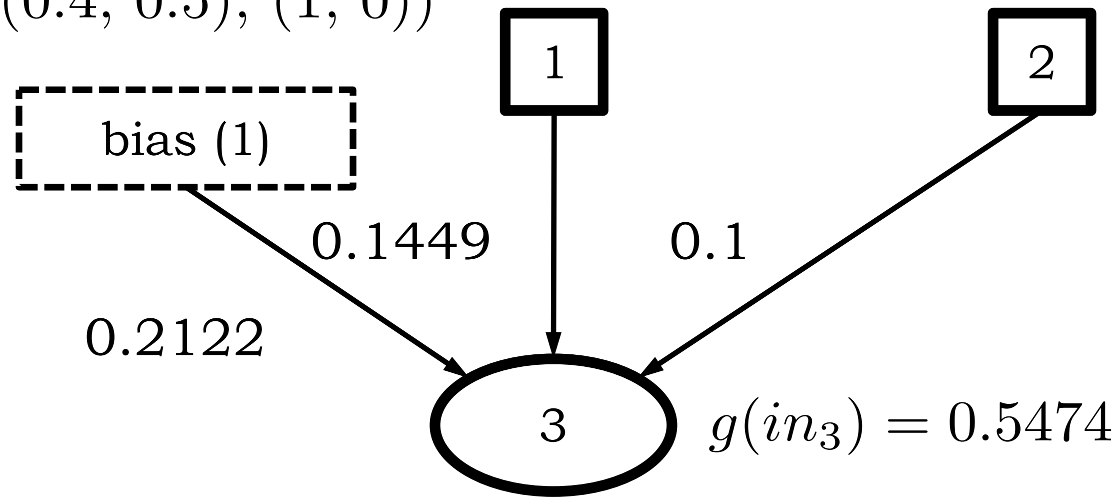
- Finally, we update each weight and get (assuming that for now we have $\alpha = 1$):

$$w_0 \leftarrow w_0 + \alpha \, Err_3 \times g'(in_3) \times x_0$$

$$w_0 \leftarrow 0.1 + (0.4526 \times 0.2478 \times 1) \approx 0.2122$$

Gradient Descent: Details

$((x_1, x_2), \mathbf{y}) = ((0.4, 0.5), (1, 0))$



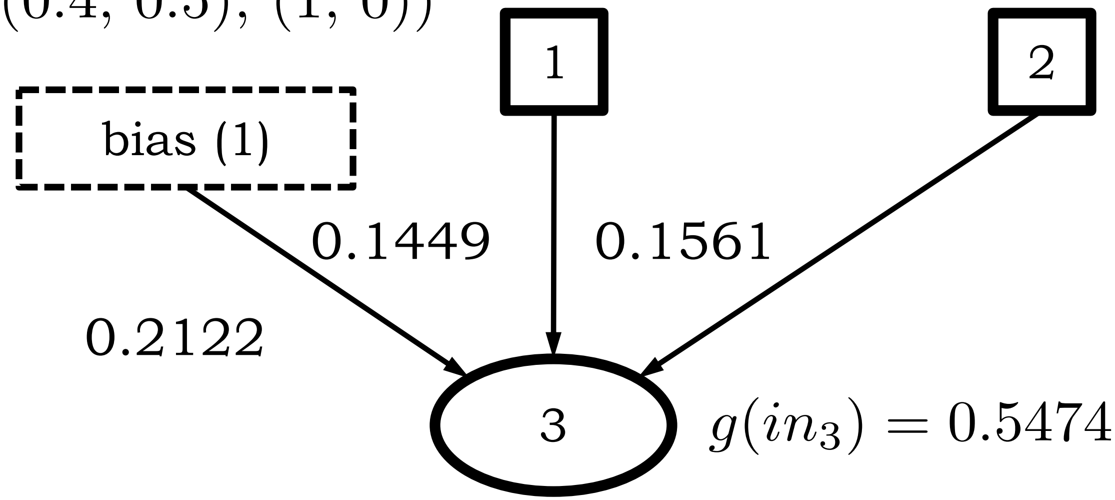
- Finally, we update each weight and get (assuming that for now we have $\alpha = 1$):

$$w_1 \leftarrow w_1 + \alpha\, Err_3 \times g'(in_3) \times x_1$$

$$w_1 \leftarrow 0.1 + (0.4526 \times 0.2478 \times 0.4) \approx 0.1449$$

Gradient Descent: Details

$((x_1, x_2), \mathbf{y}) = ((0.4, 0.5), (1, 0))$



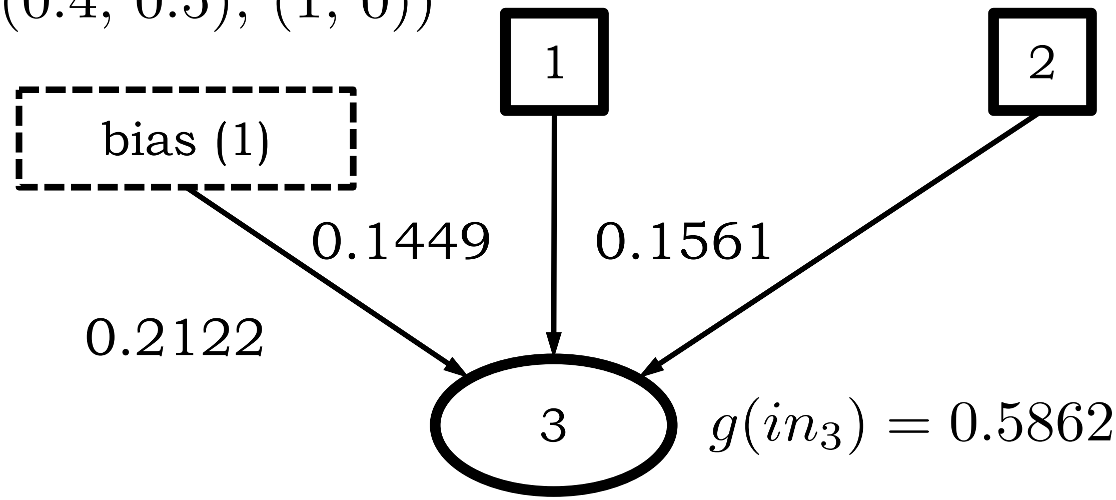
- Finally, we update each weight and get (assuming that for now we have $\alpha = 1$):

$$w_2 \leftarrow w_2 + \alpha \, Err_3 \times g'(in_3) \times x_2$$

$$w_2 \leftarrow 0.1 + (0.4526 \times 0.2478 \times 0.5) \approx 0.1561$$

Gradient Descent: Details

$((x_1, x_2), \mathbf{y}) = ((0.4, 0.5), (1, 0))$



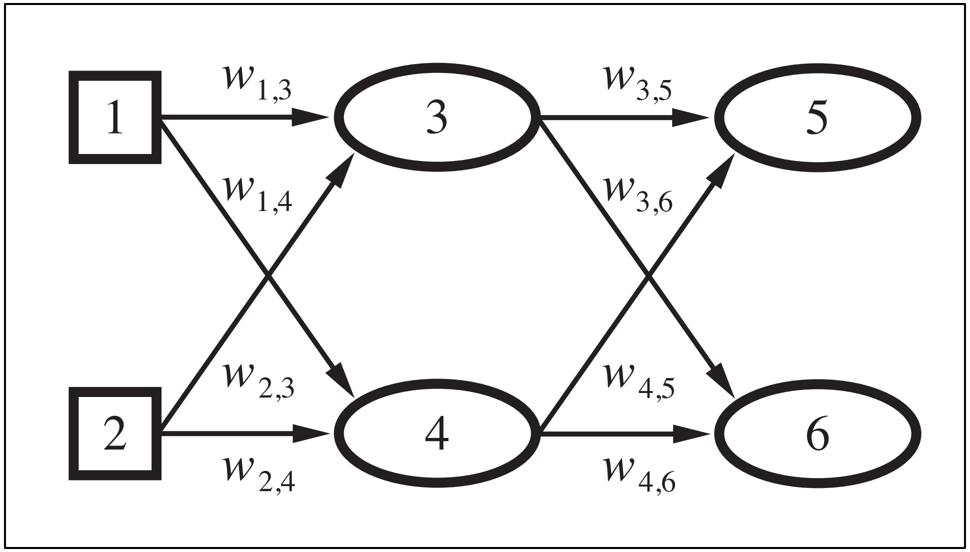
- The result is that each connection has been ***strengthened***, increasing each weight in proportion to both error made and the input to that connection
- As a result, the value computed by the neuron increases as well (reducing, but not eliminating the amount of error the neuron makes):

$$in_3 = 0.2122 + (0.1449 \times 0.4) + (0.1561 \times 0.5) = 0.34821$$

$$g(in_3) = \frac{1}{1 + e^{-0.34821}} \approx 0.5862$$

Tufts

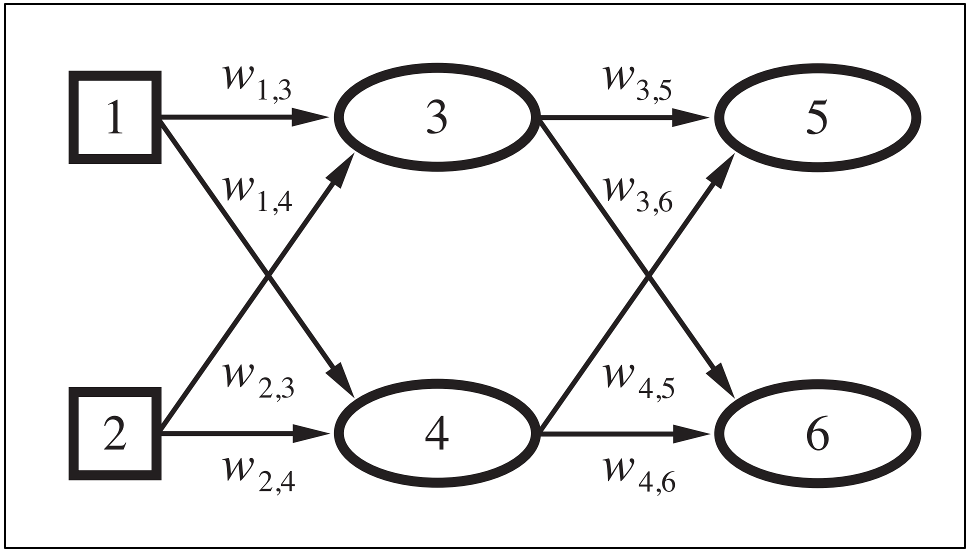
Multi-Layer Feed-Forward Neural Networks



- It was known from the beginning that a single-layer network could not represent all possible functions
- For more complex functions, we will add one or more hidden layers of neurons between input and output layers
- In a feed-forward network, each layer is connected only to next layer in the order (inputs → outputs)

Tufts

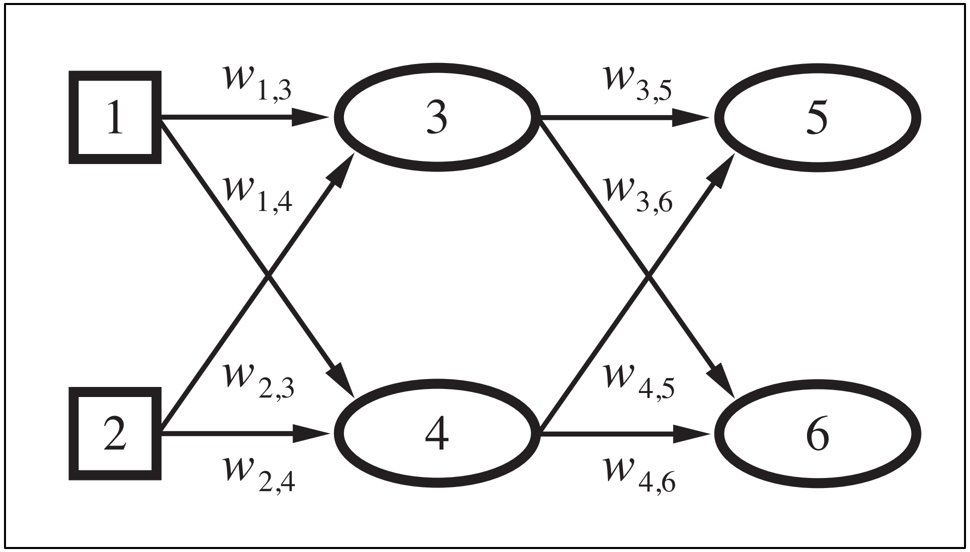
Multi-Layer Feed-Forward Neural Networks



- Output neurons process inputs ***indirectly***, via inputs that come from the hidden layer "above" them, e.g.:

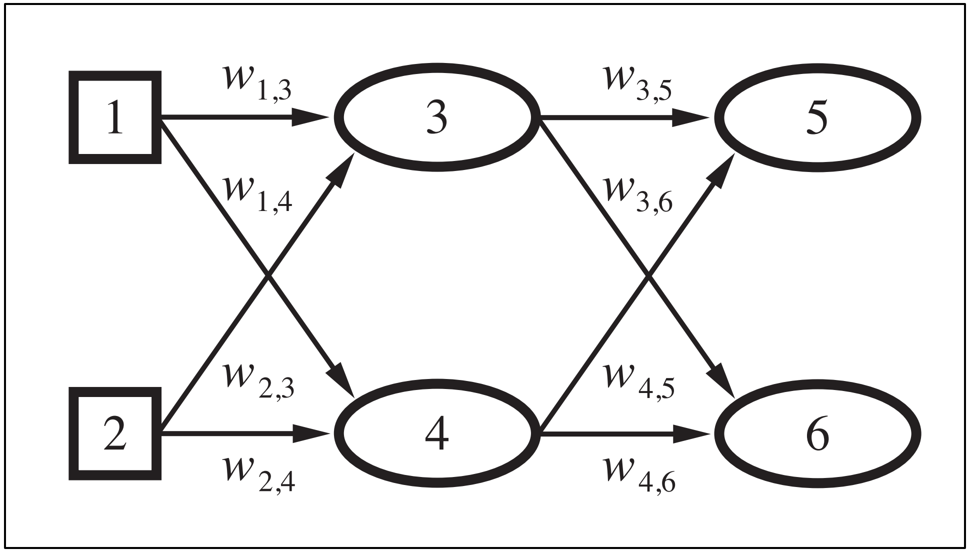
$$\begin{aligned} a_5 = g(in_5) &= g(w_{0,5} + w_{3,5}\, a_3 + w_{4,5}\, a_4) \\ &= g(w_{0,5} + w_{3,5}\, g(w_{0,3} + w_{1,3}\, x_1 + w_{2,3}\, x_2) \\ &\qquad + w_{4,5}\, g(w_{0,4} + w_{1,4}\, x_1 + w_{2,4}\, x_2)) \end{aligned}$$

Learning in Neural Networks



- A neural network can learn a classification function by adjusting its weights to compute different responses
- This process is another version of gradient descent: the algorithm moves through a complex space of partial solutions, always seeking to ***minimize*** overall error

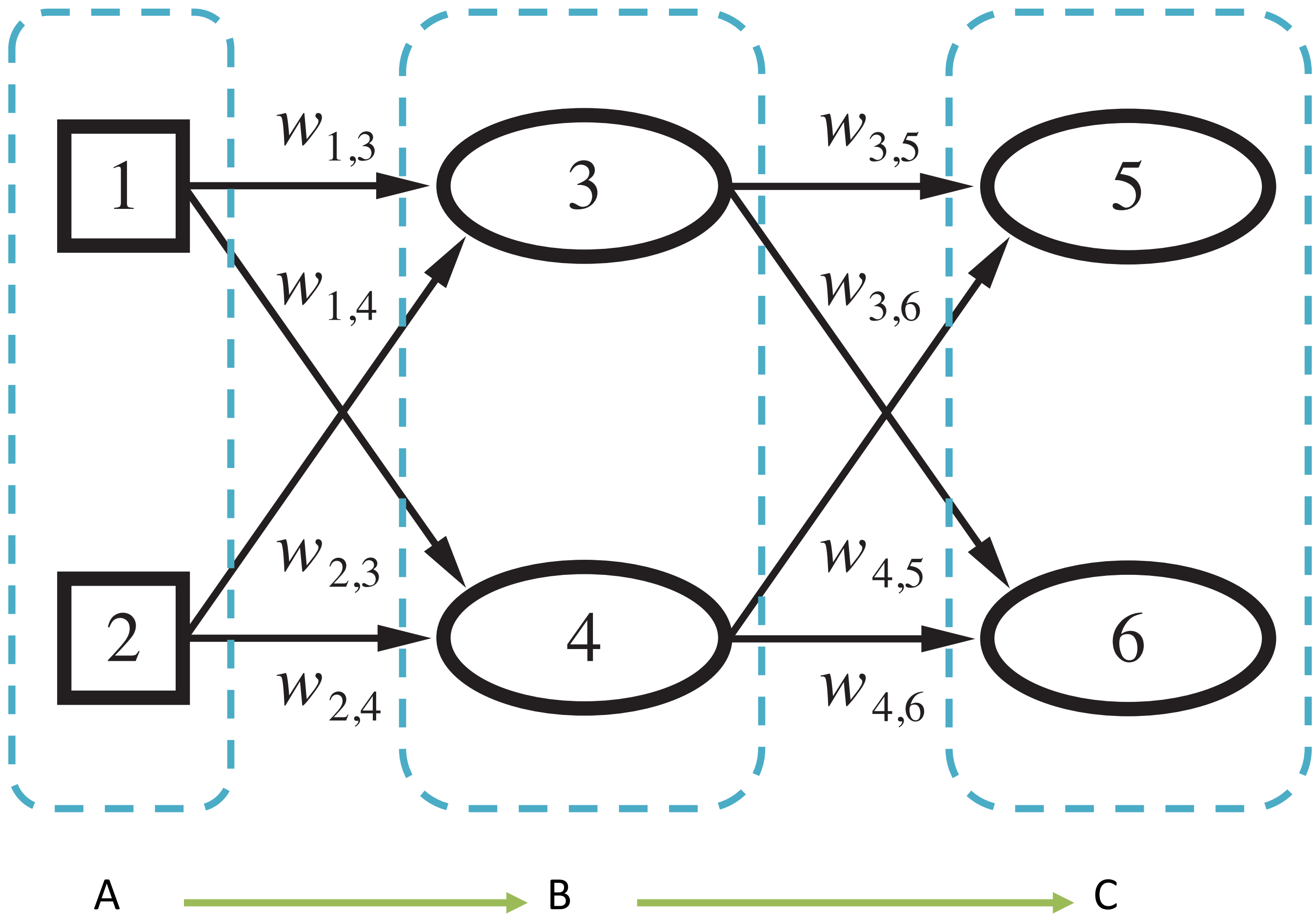
Training Networks by Back-Propagation



- Techniques for training neural networks date to the late 1960's, although it has been "rediscovered" a few times
- In a feed-forward network, we can train by:

1. Pushing input ***forward*** to produce output
2. Pushing error ***backward*** to update weights

Training Networks by Back-Propagation



Training Networks by Back-Propagation