# CS 4700: Foundations of Artificial Intelligence

ONLINE

Spring 2020
Prof. Haym Hirsh

Lecture 31
April 27, 2020

# Mock Grades Out

1. Based on grades thus far
2. Extrapolated with median in each category if currently below median

This is not a contract, it's to help you with S/U

(Think +/- ~half a grade)

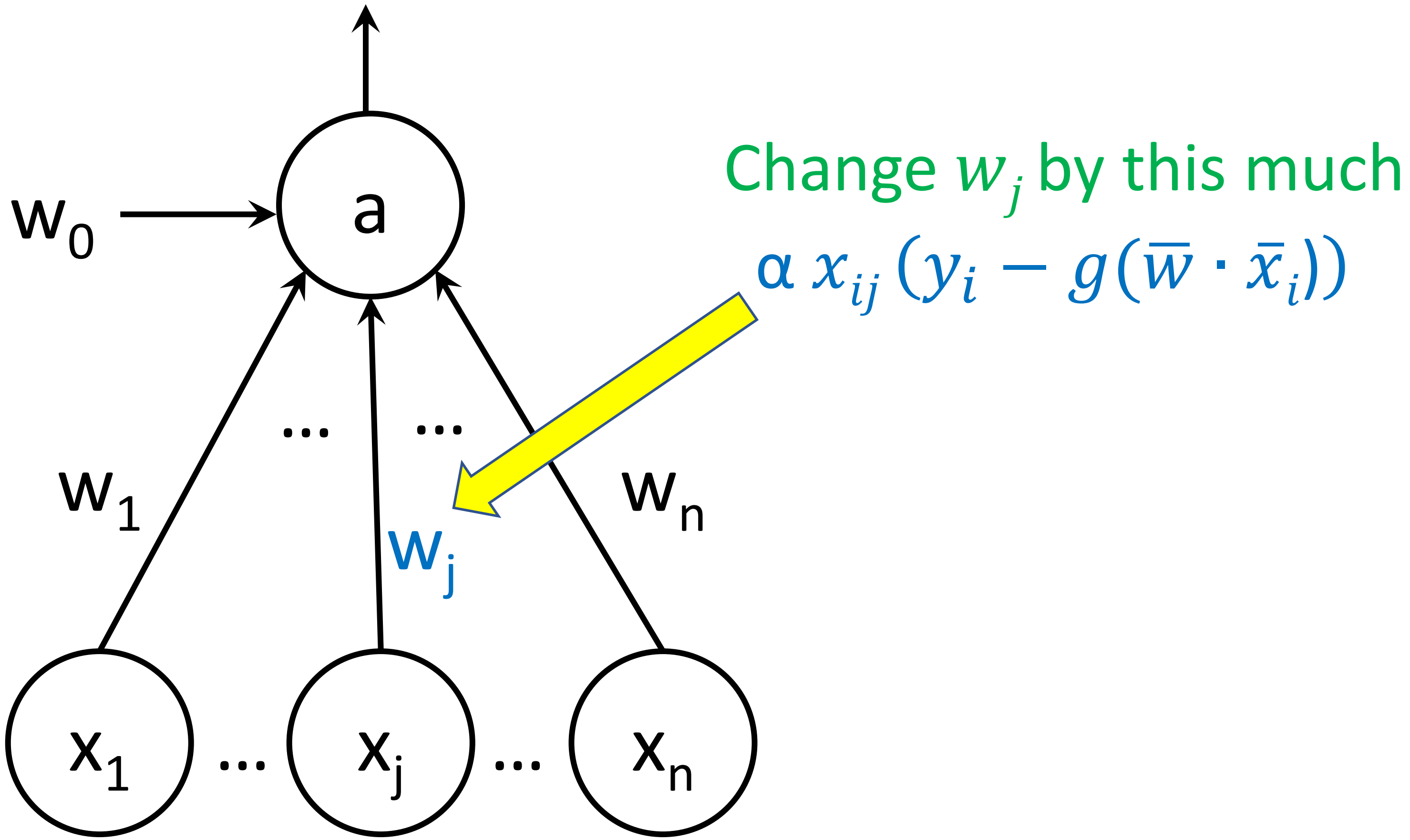
Only time

# COVID-19

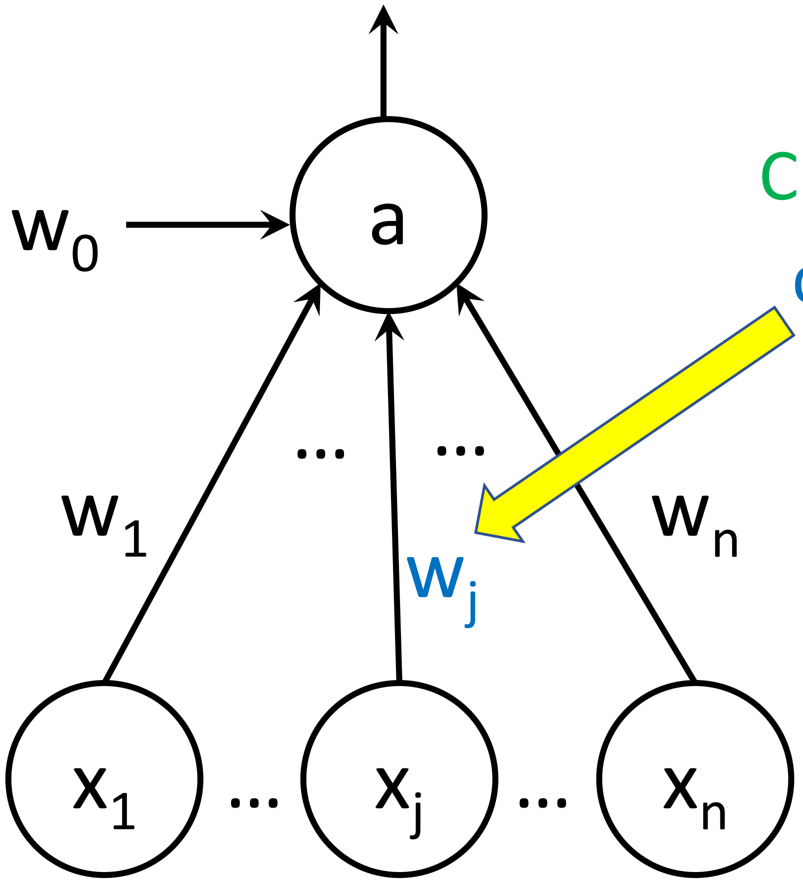
Self-assess your status

Please get in touch if life is intruding in your studies

# Weight $w_j$ Update for a Perceptron



# Weight $w_j$ Update for a Perceptron

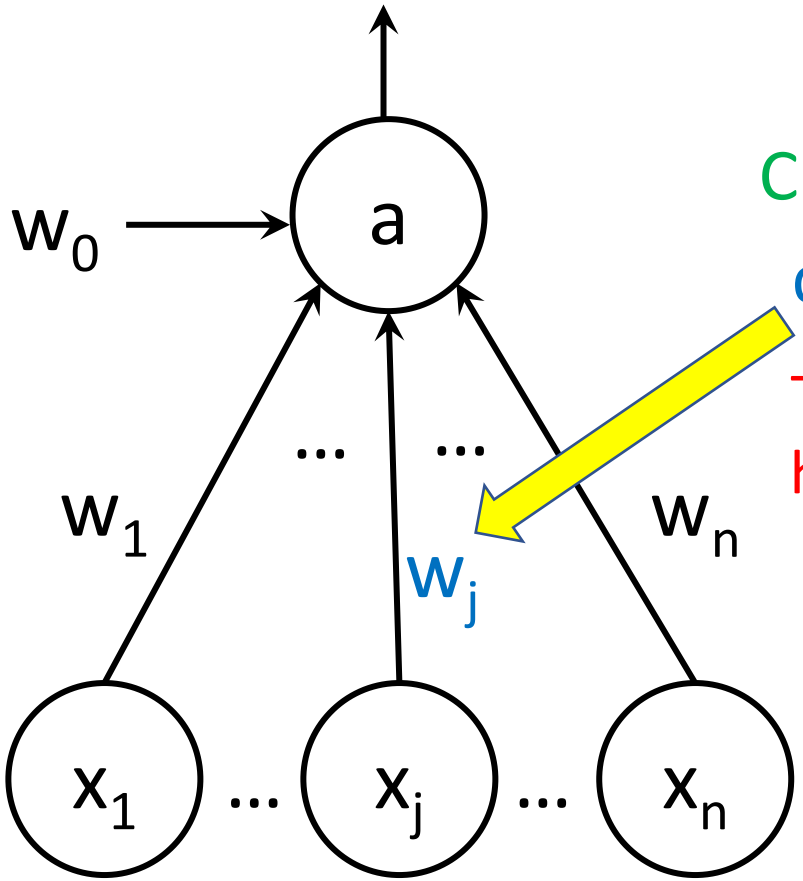



Change $w_j$ by this much

$\alpha\, x_{ij} \boxed{(y_i - g(\bar{w} \cdot \bar{x}_i))}$

Think of this as how much the output is off by – it doesn't depend on $w_j$

# Weight $w_j$ Update for a Perceptron

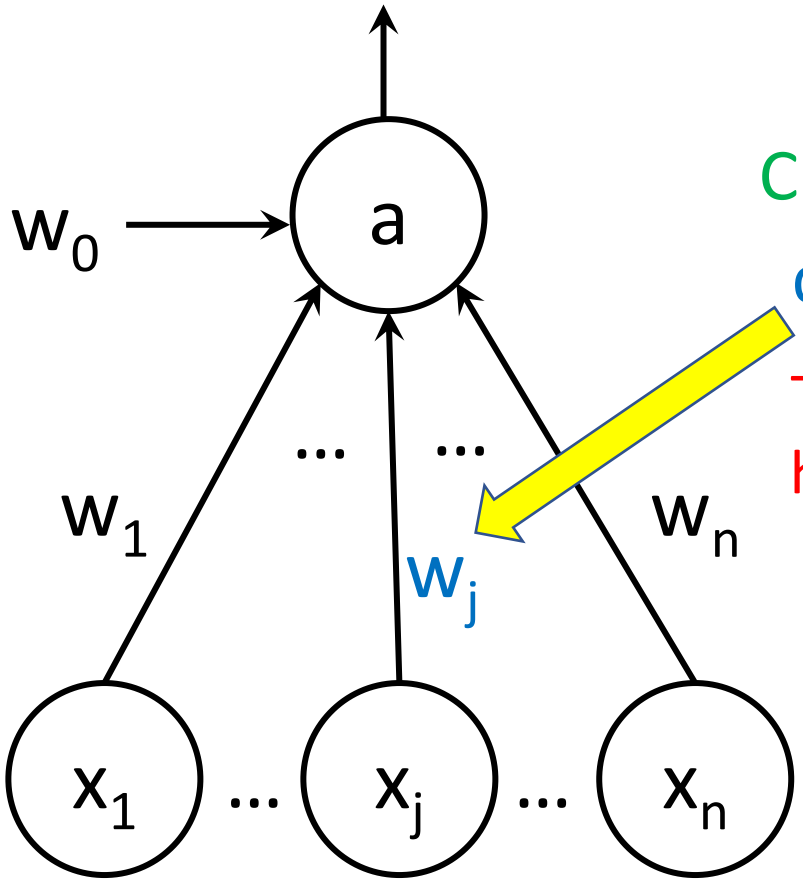



Change $w_j$ by this much

$\alpha\ x_{ij} \boxed{(y_i - g(\bar{w} \cdot \bar{x}_i))}$

Think of this as a sort of error: how much the output is off by

# Weight $w_j$ Update for a Perceptron

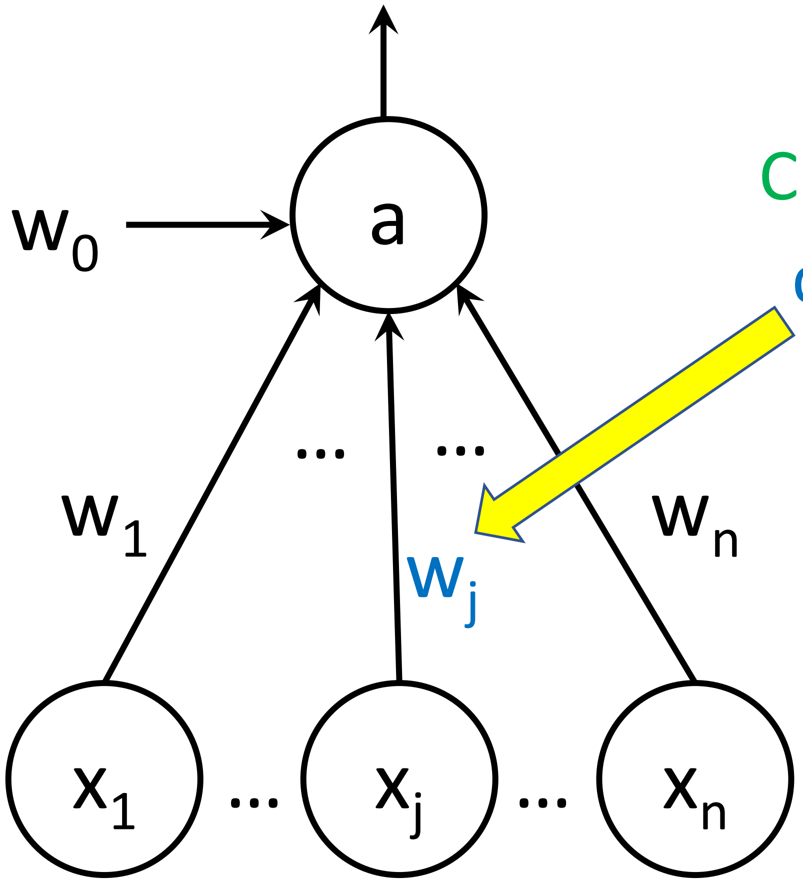



Change $w_j$ by this much

$\alpha\, x_{ij} \boxed{(y_i - g(\bar{w} \cdot \bar{x}_i))}$

Think of this as a sort of error: how much the output is off by – it's the same for all of the weight updates $w_1, \dots, w_n$

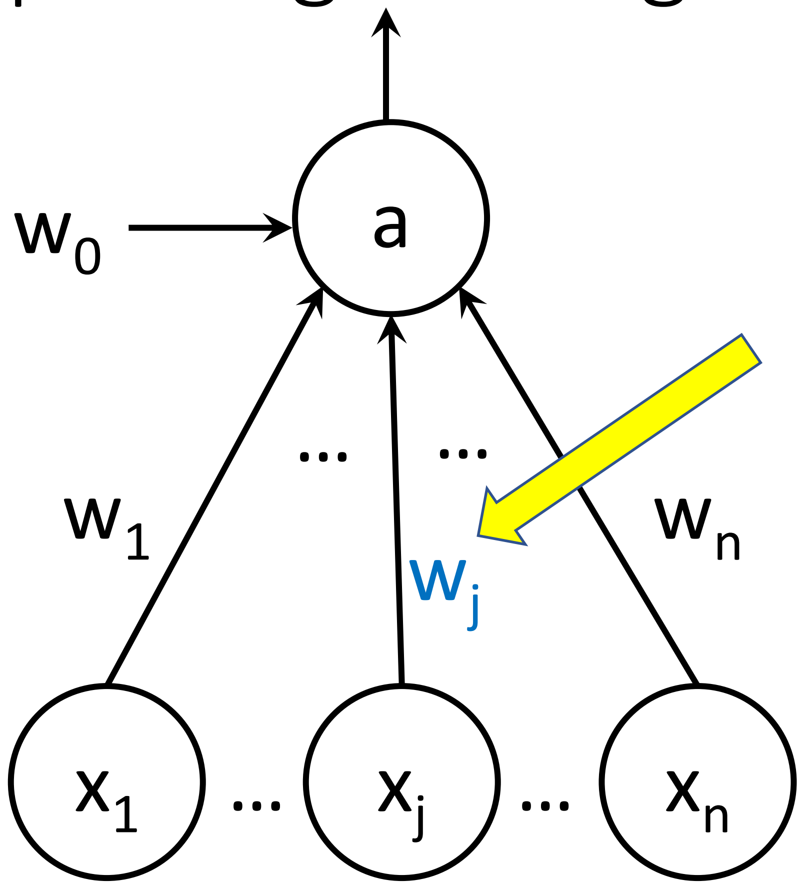
# Weight $w_j$ Update for a Perceptron



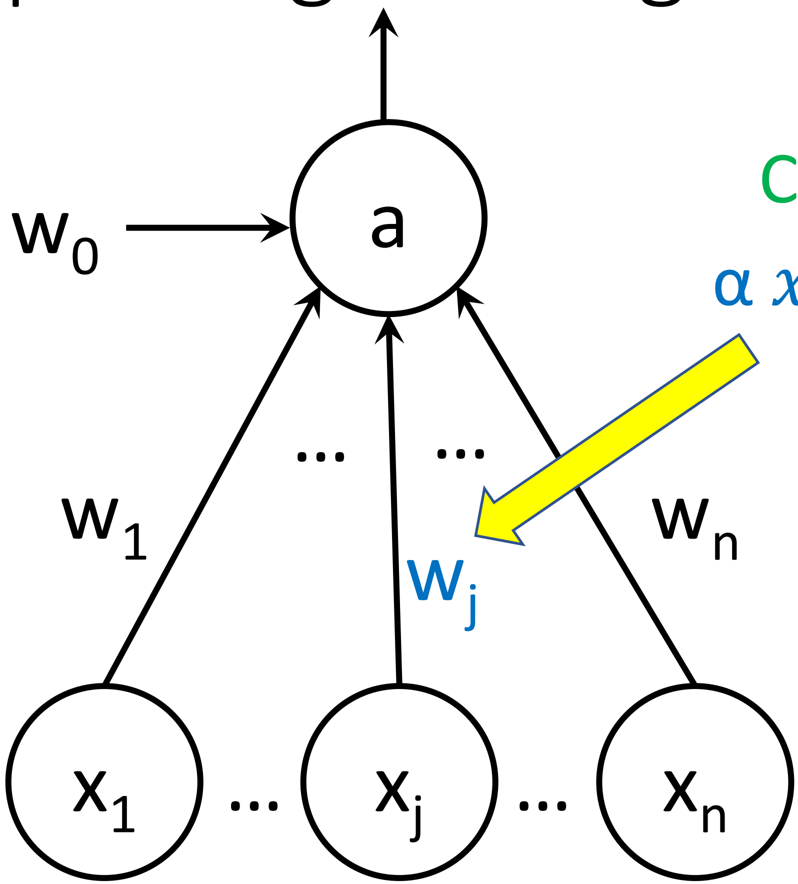

Change $w_j$ by this much

$\alpha\, x_{ij}\,(y_i - g(\bar{w} \cdot \bar{x}_i))$

# Weight $w_j$ Update for Gradient Descent (Single Neuron) [Example: Logistic Regression]



# Weight $w_j$ Update for Gradient Descent (Single Neuron) [Example: Logistic Regression]

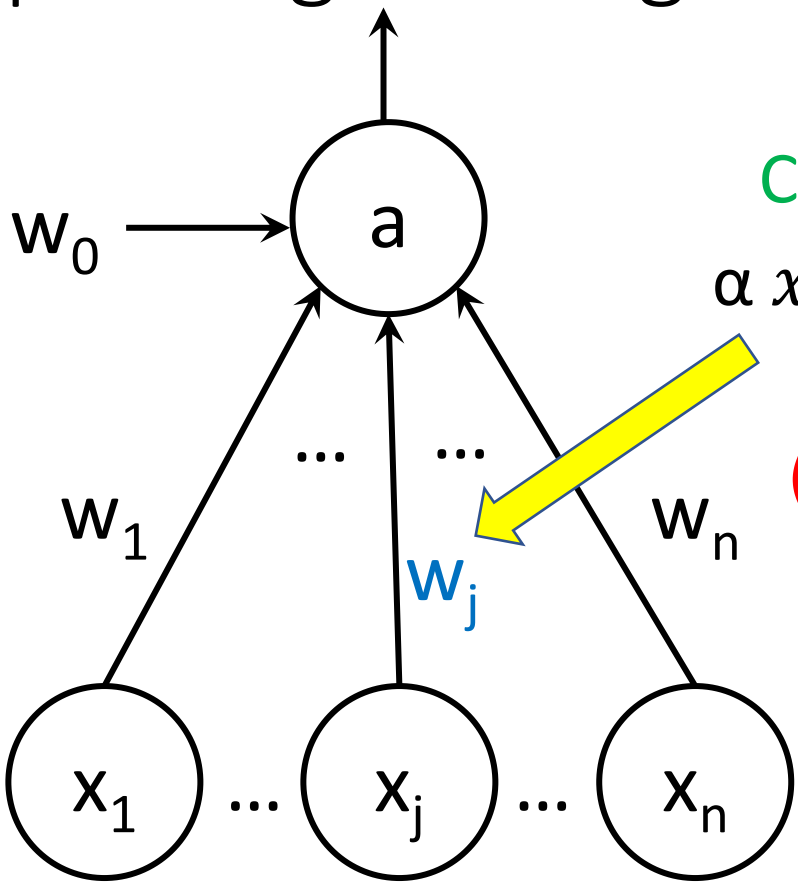



Change $w_j$ by this much

$$\alpha\, x_{ij}\, g'(\bar{w}\cdot\bar{x}_i)(y_i - g(\bar{w}\cdot\bar{x}_i))$$

The only thing that's different

# Weight $w_j$ Update for Gradient Descent (Single Neuron) [Example: Logistic Regression]

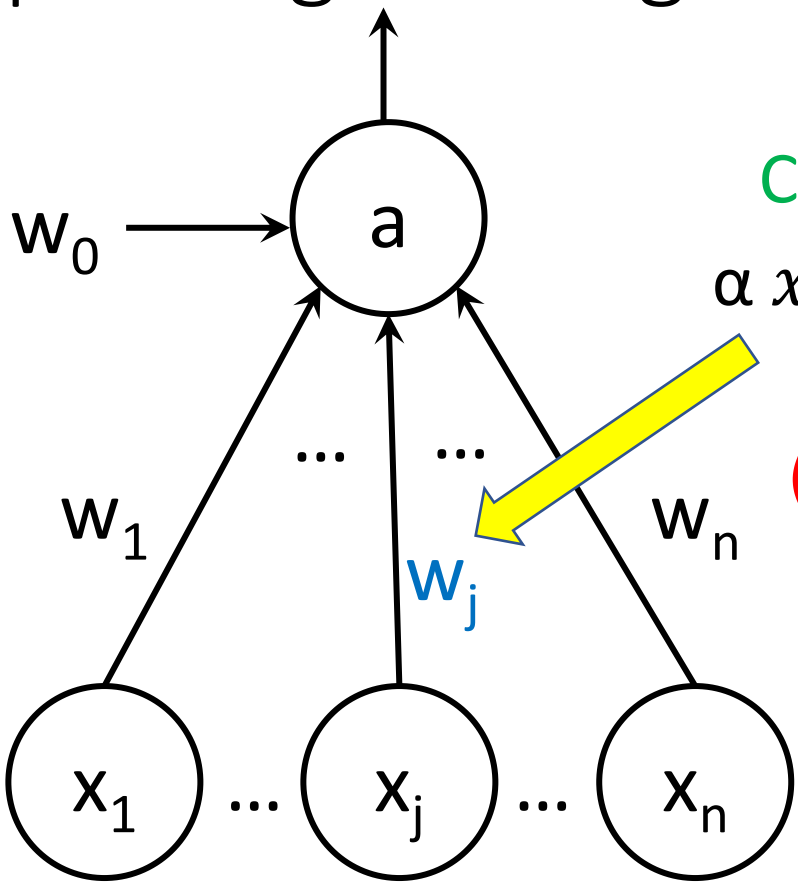



Change $w_j$ by this much

$$\alpha\, x_{ij} \boxed{g'(\overset{(2)}{\overline{w}} \cdot \bar{x}_i)(y_i - g(\overset{(1)}{\overline{w}} \cdot \bar{x}_i))}$$

Think of this as a sort of error:
(1) how much the output is off by (same as before), and
(2) punishes you for having a large $g'(\overline{w} \cdot \bar{x}_i)$

# Weight $w_j$ Update for Gradient Descent (Single Neuron) [Example: Logistic Regression]



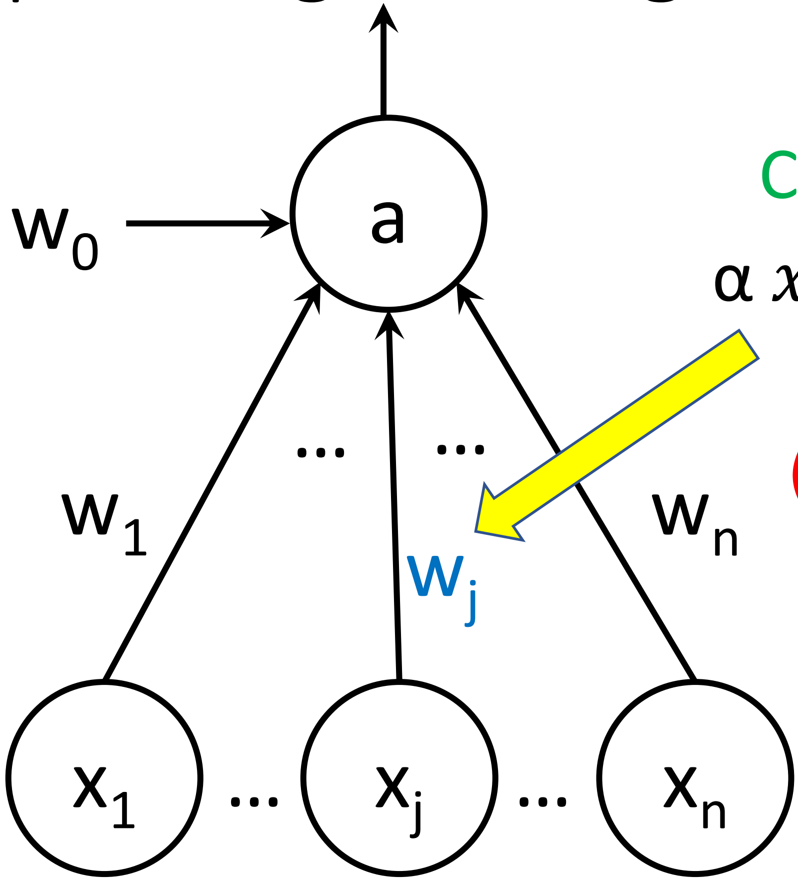

Change $w_j$ by this much

$\alpha\, x_{ij} \boxed{g'(\overset{(2)}{\bar{w} \cdot \bar{x}_i})(y_i - g(\overset{(1)}{\bar{w} \cdot \bar{x}_i}))}$

Think of this as a sort of error:
(1) how much the output is off by (same as before), and
(2) punishes you for having a large $g'(\bar{w} \cdot \bar{x}_i)$

We want similar values of $\bar{x}$ to have similar $g(\bar{w} \cdot \bar{x})$ values

# Weight $w_j$ Update for Gradient Descent (Single Neuron) [Example: Logistic Regression]



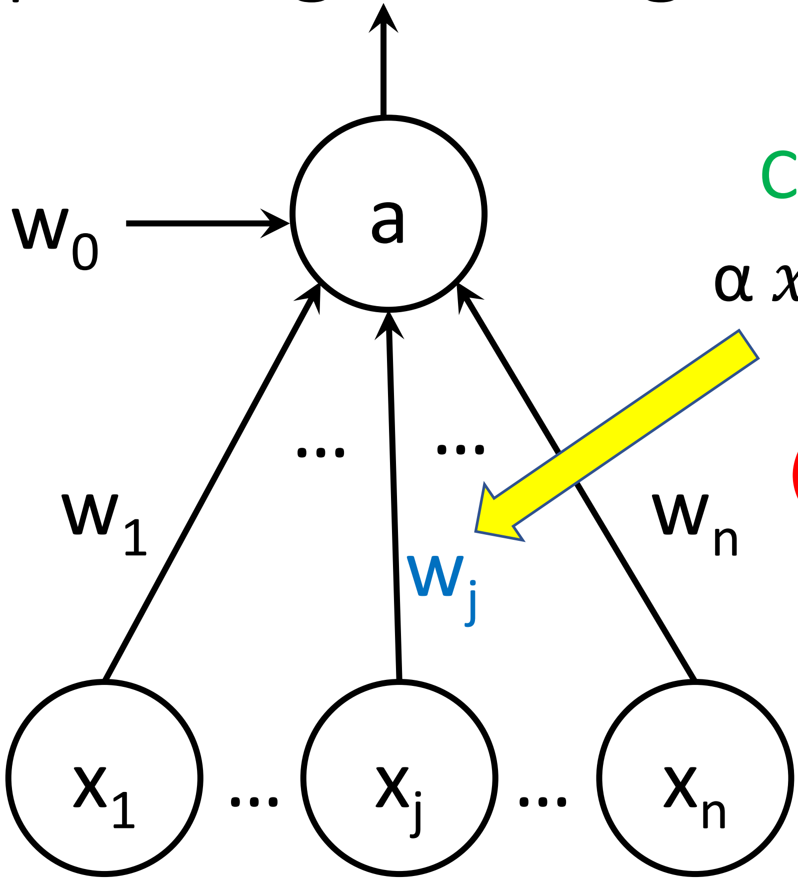

Change $w_j$ by this much

$$\alpha\, x_{ij} \boxed{g'(\overset{(2)}{\bar{w} \cdot \bar{x}_i})(y_i - g(\overset{(1)}{\bar{w} \cdot \bar{x}_i}))}$$

Think of this as a sort of error:
(1) how much the output is off by (same as before), and
(2) punishes you for having a large $g'(\bar{w} \cdot \bar{x}_i)$

We want similar values of $\bar{x}$ to have similar $g(\bar{w} \cdot \bar{x})$ values
(because two similar things should likely have the same label)

# Weight $w_j$ Update for Gradient Descent (Single Neuron) [Example: Logistic Regression]



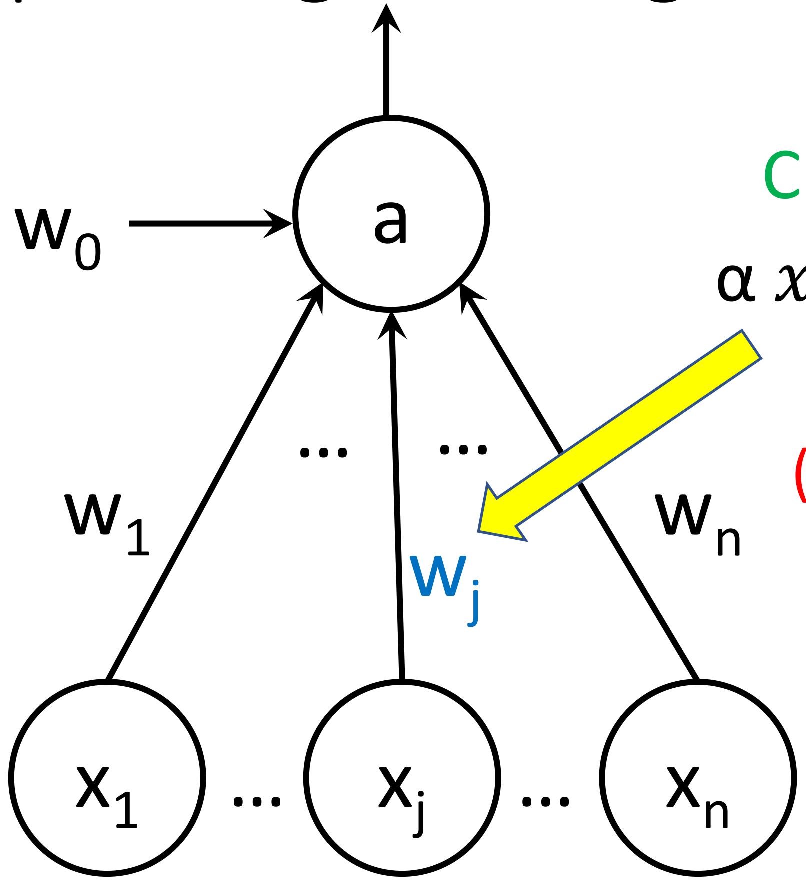

Change $w_j$ by this much

$$\alpha\, x_{ij} \boxed{g'(\overset{(2)}{\overline{w} \cdot \bar{x}_i})(y_i - g(\overset{(1)}{\overline{w} \cdot \bar{x}_i}))}$$

Think of this as a sort of error:
(1) how much the output is off by (same as before), and
(2) punishes you for having a large $g'(\overline{w} \cdot \bar{x}_i)$

We want similar values of $\bar{x}$ to have similar $g(\overline{w} \cdot \bar{x})$ values

# Weight $w_j$ Update for Gradient Descent (Single Neuron) [Example: Logistic Regression]



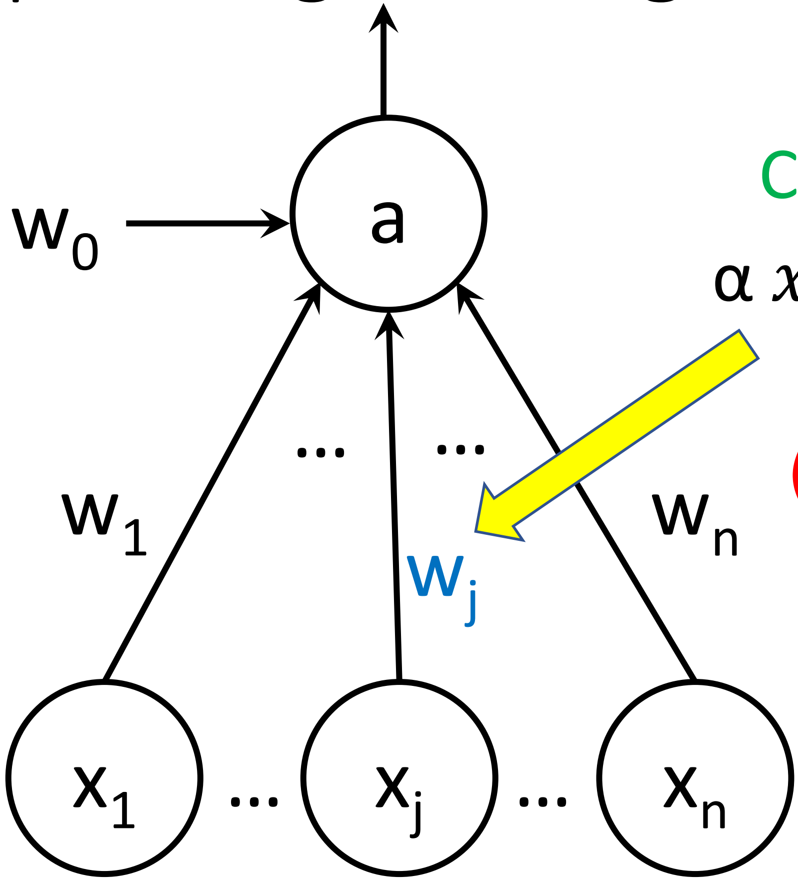

Change $w_j$ by this much

$$\alpha\, x_{ij} \boxed{\overset{(2)}{g'(\bar{w}\cdot\bar{x}_i)}\,\overset{(1)}{(y_i - g(\bar{w}\cdot\bar{x}_i))}}$$

Think of this as a sort of error:
(1) how much the output is off by (same as before), and
(2) punishes you for having a large $g'(\bar{w}\cdot\bar{x}_i)$

We want similar values of $\bar{x}$ to have similar $g(\bar{w}\cdot\bar{x})$ values
$\Rightarrow$ want $g(\bar{w}\cdot\bar{x}_i)$ to be locally flat

# Weight $w_j$ Update for Gradient Descent (Single Neuron) [Example: Logistic Regression]



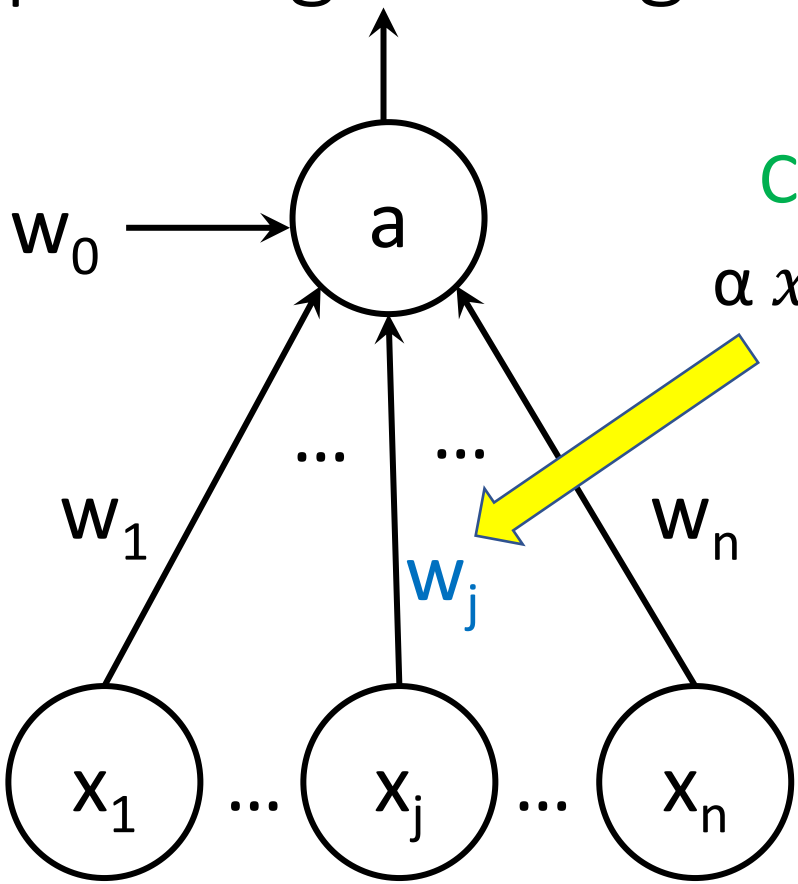

Change $w_j$ by this much

$$\alpha\, x_{ij}\,\boxed{\overset{(2)}{g'(\bar{w}\cdot\bar{x}_i)}\,(y_i - \overset{(1)}{g(\bar{w}\cdot\bar{x}_i)})}$$

Think of this as a sort of error:
(1) how much the output is off by (same as before), and
(2) punishes you for having a large $g'(\bar{w}\cdot\bar{x}_i)$

We want similar values of $\bar{x}$ to have similar $g(\bar{w}\cdot\bar{x})$ values
$\Rightarrow$ want $g(\bar{w}\cdot\bar{x}_i)$ to be locally flat
$\Rightarrow$ want $g'(\bar{w}\cdot\bar{x}_i)$ to be small

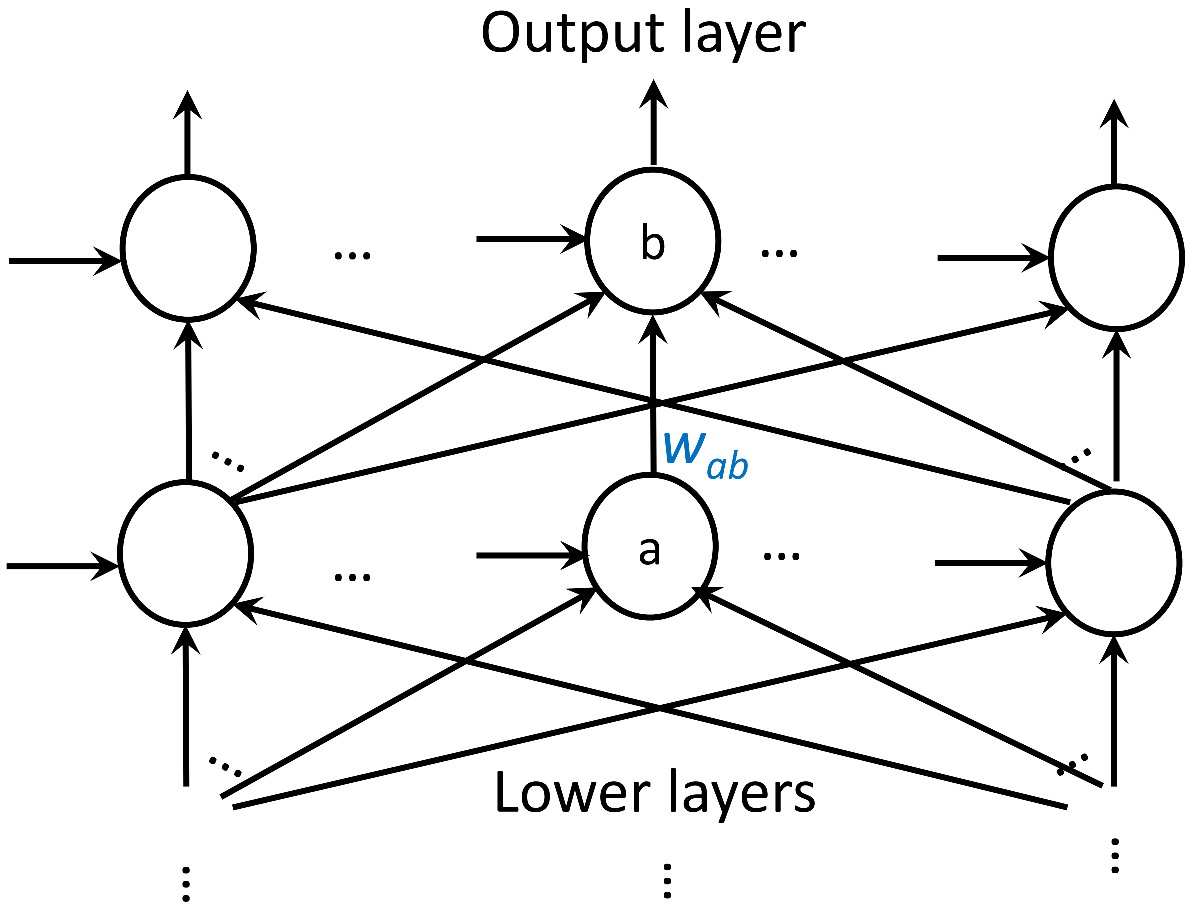
# Weight $w_j$ Update for Gradient Descent (Single Neuron) [Example: Logistic Regression]



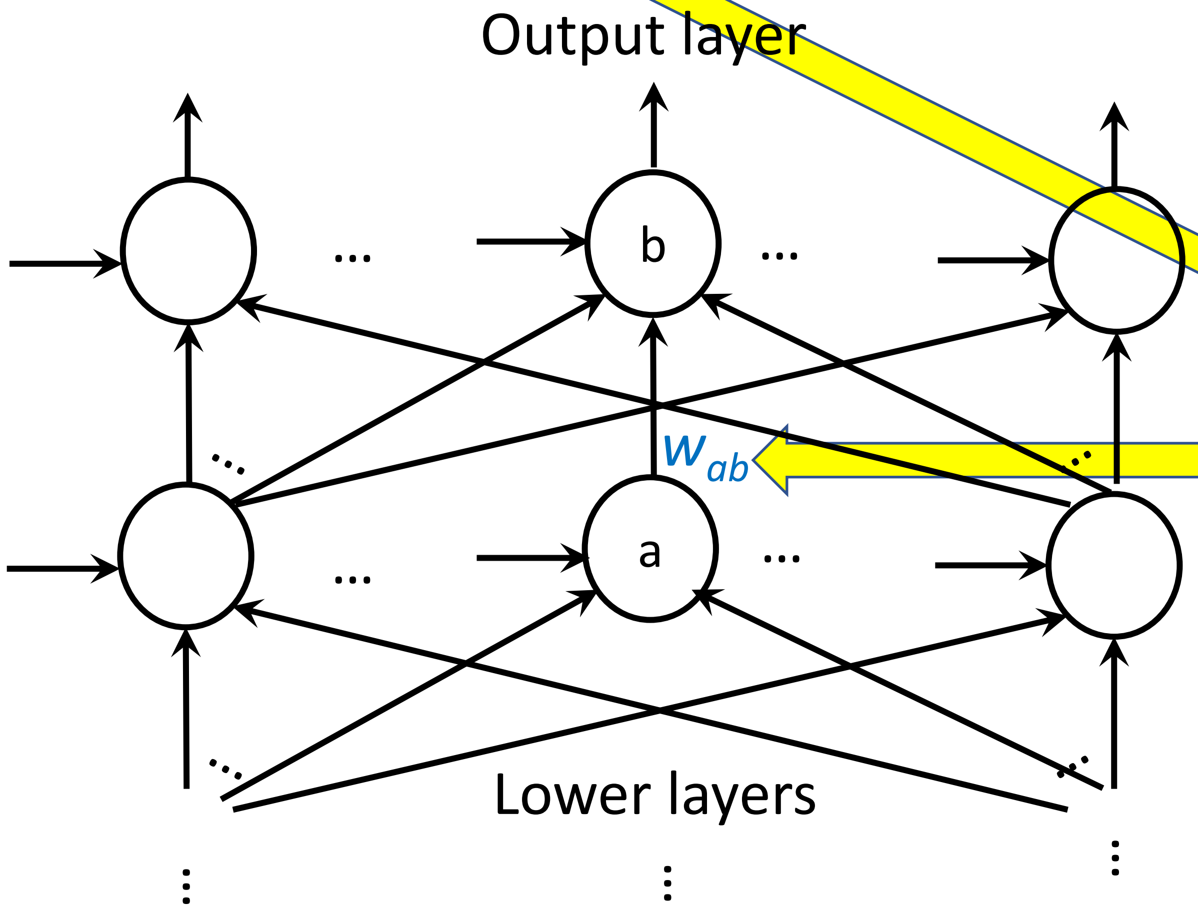

Change $w_j$ by this much

$$\alpha\, x_{ij}\, g'(\bar{w} \cdot \bar{x}_i)(y_i - g(\bar{w} \cdot \bar{x}_i))$$

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer)
## Output Layer Weights

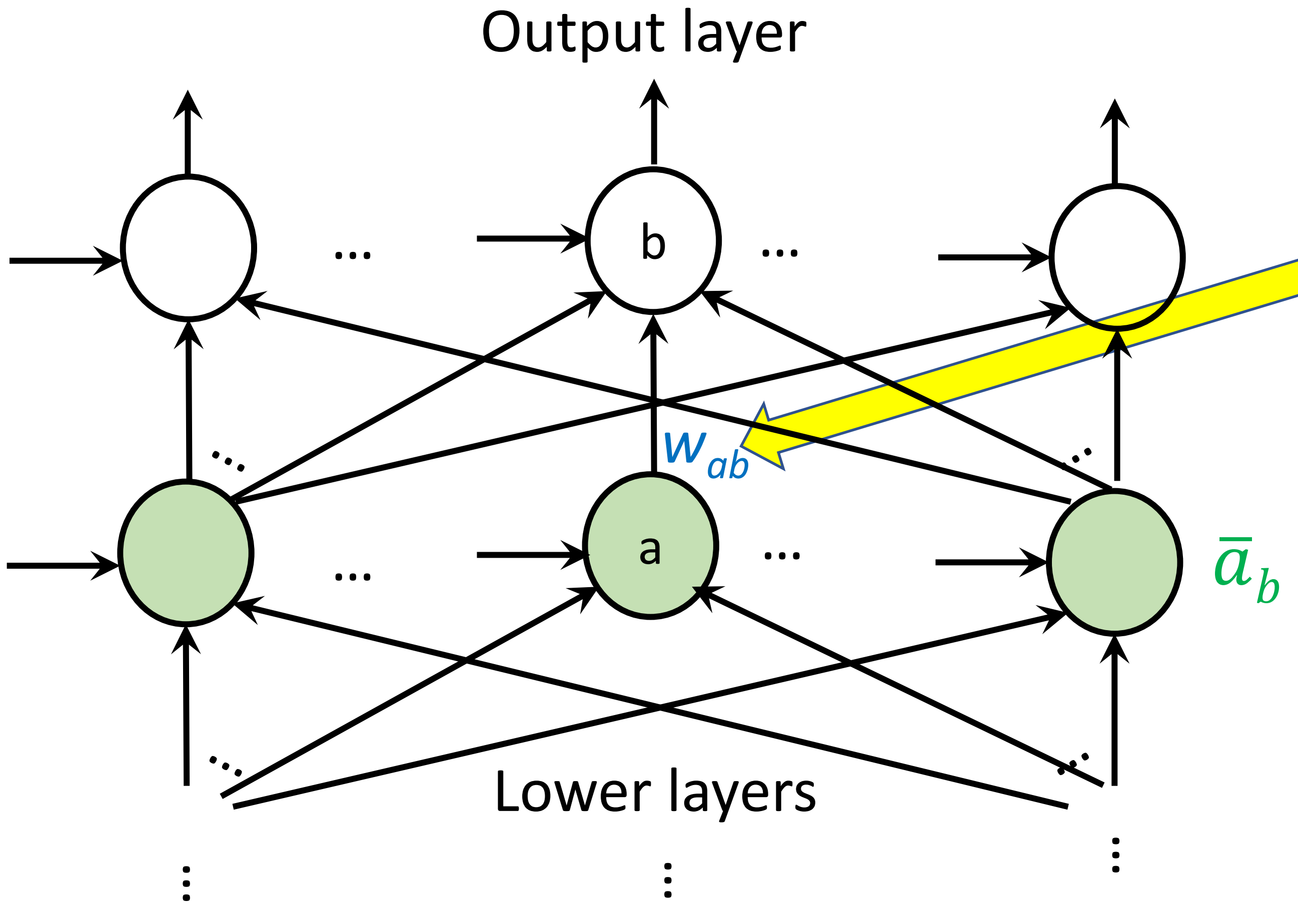


# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer) Output Layer Weights




Intentionally using a and b so that we don't get confused by the use of i's and j's in different places for different things

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer) Output Layer Weights

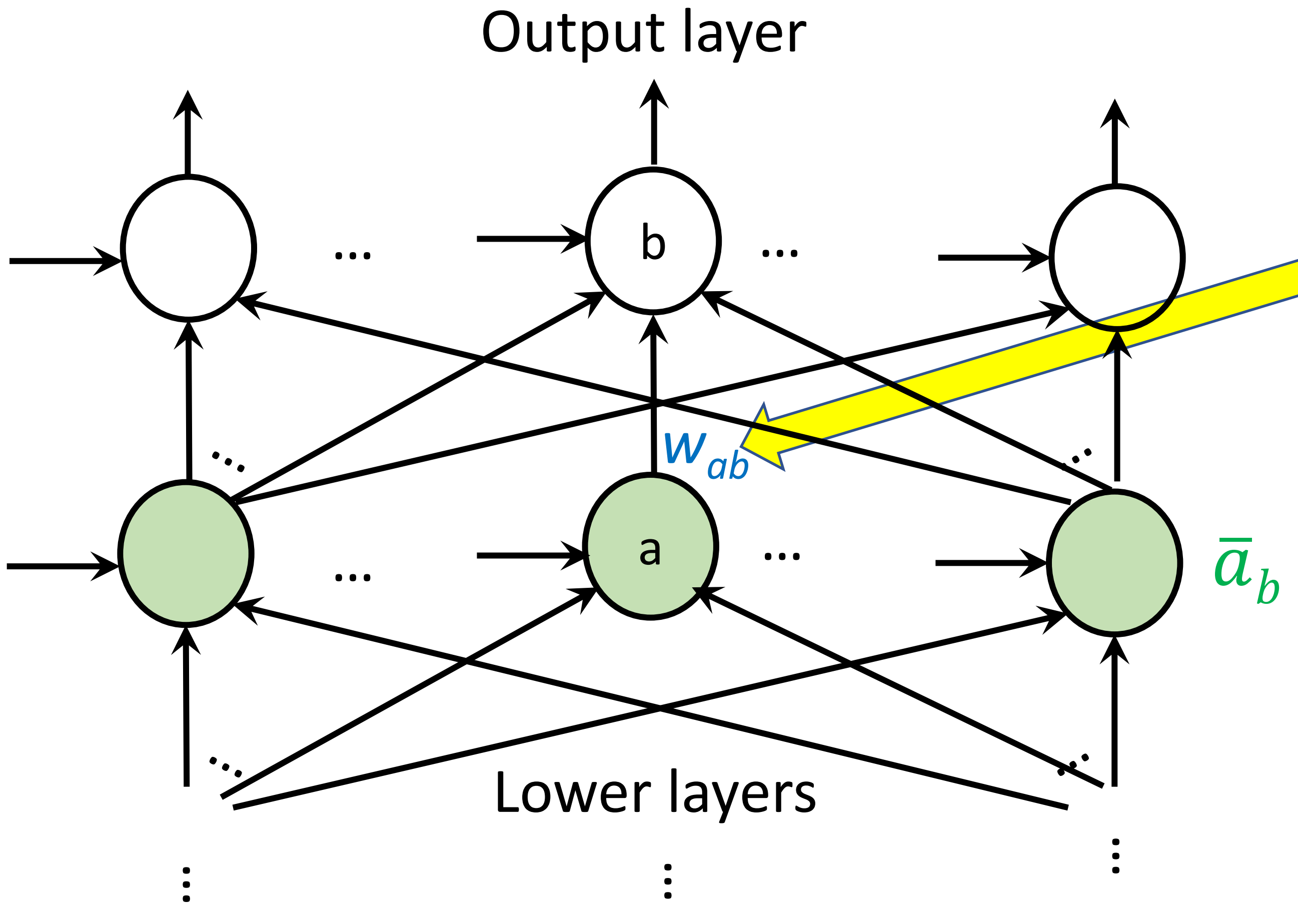



Change $w_{ab}$ by this much

$$\alpha\, a_a\, g'(\bar{w}_b \cdot \bar{a}_b)(y_b - g(\bar{w}_b \cdot \bar{a}_b))$$

where $\bar{a}_b$ is the vector of activations going to $b$

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer)
# Output Layer Weights



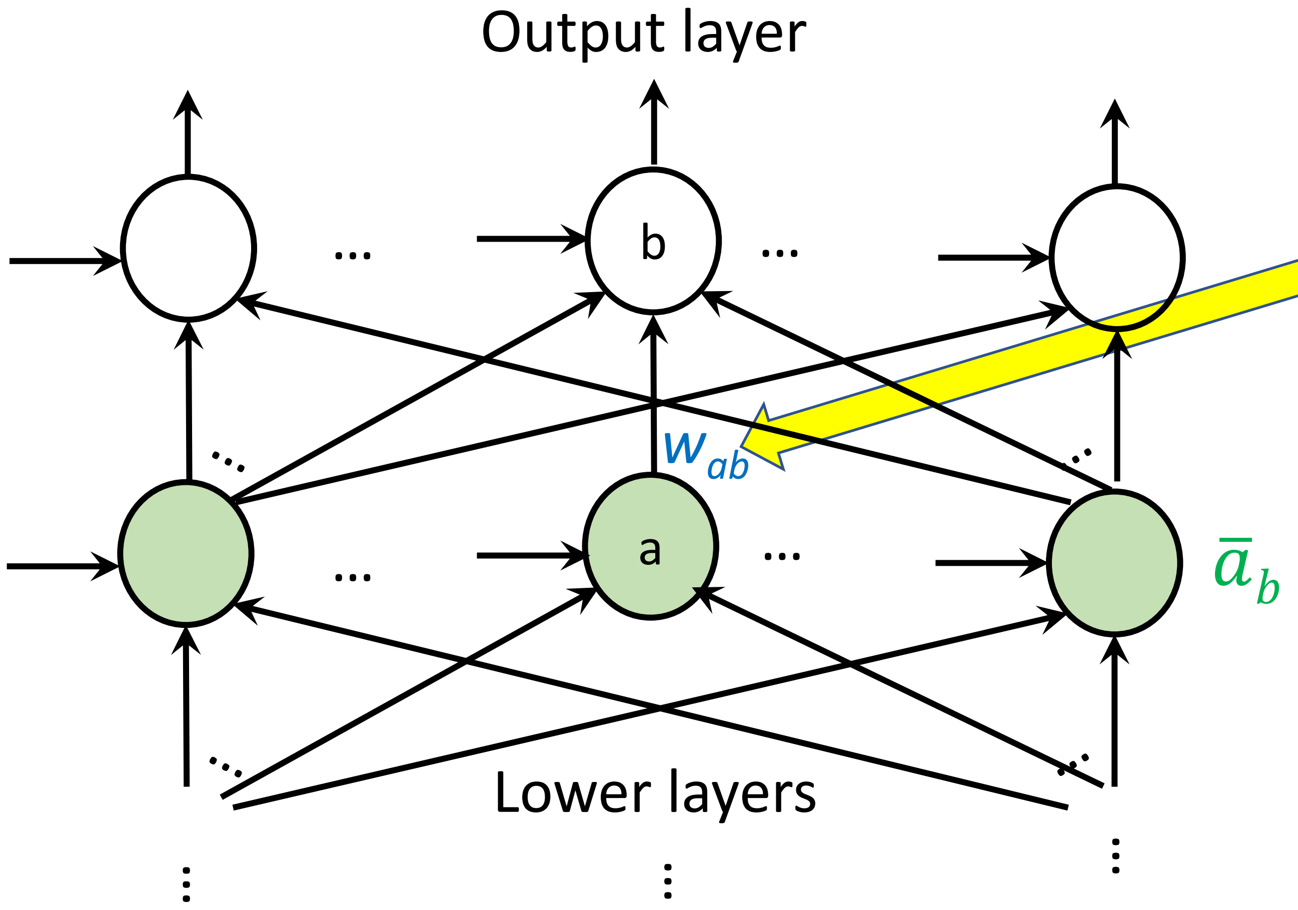

Change $w_{ab}$ by this much

$$\alpha\, a_a\, g'(\bar{w}_b \cdot \bar{a}_b)(y_b - g(\bar{w}_b \cdot \bar{a}_b))$$

where $\bar{a}_b$ is the vector of activations going to $b$

- this is the same as before for a single neuron, but the inputs are the incoming activations rather than the $\bar{x}_i$

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer) Output Layer Weights




Change $w_{ab}$ by this much

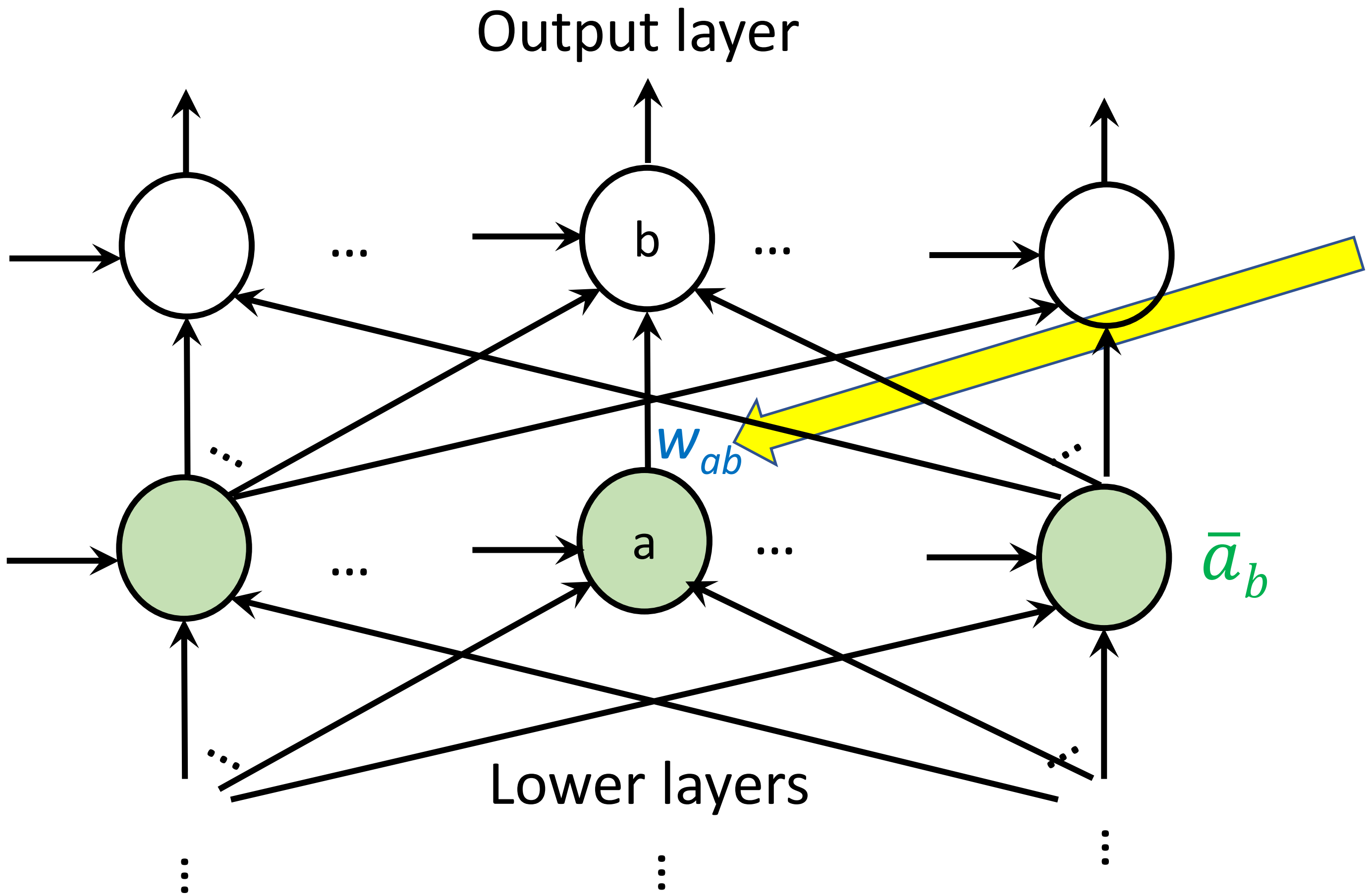
$$\alpha\, a_a\, g'(\bar{w}_b \cdot \bar{a}_b)(y_b - g(\bar{w}_b \cdot \bar{a}_b))$$

where $\bar{a}_b$ is the vector of activations going to $b$

- this is the same as before for a single neuron, but the inputs are the incoming activations rather than the $\bar{x}_i$

Up until now we've included an index *i* referring to the example we're currently doing an update for

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer) Output Layer Weights




Change $w_{ab}$ by this much

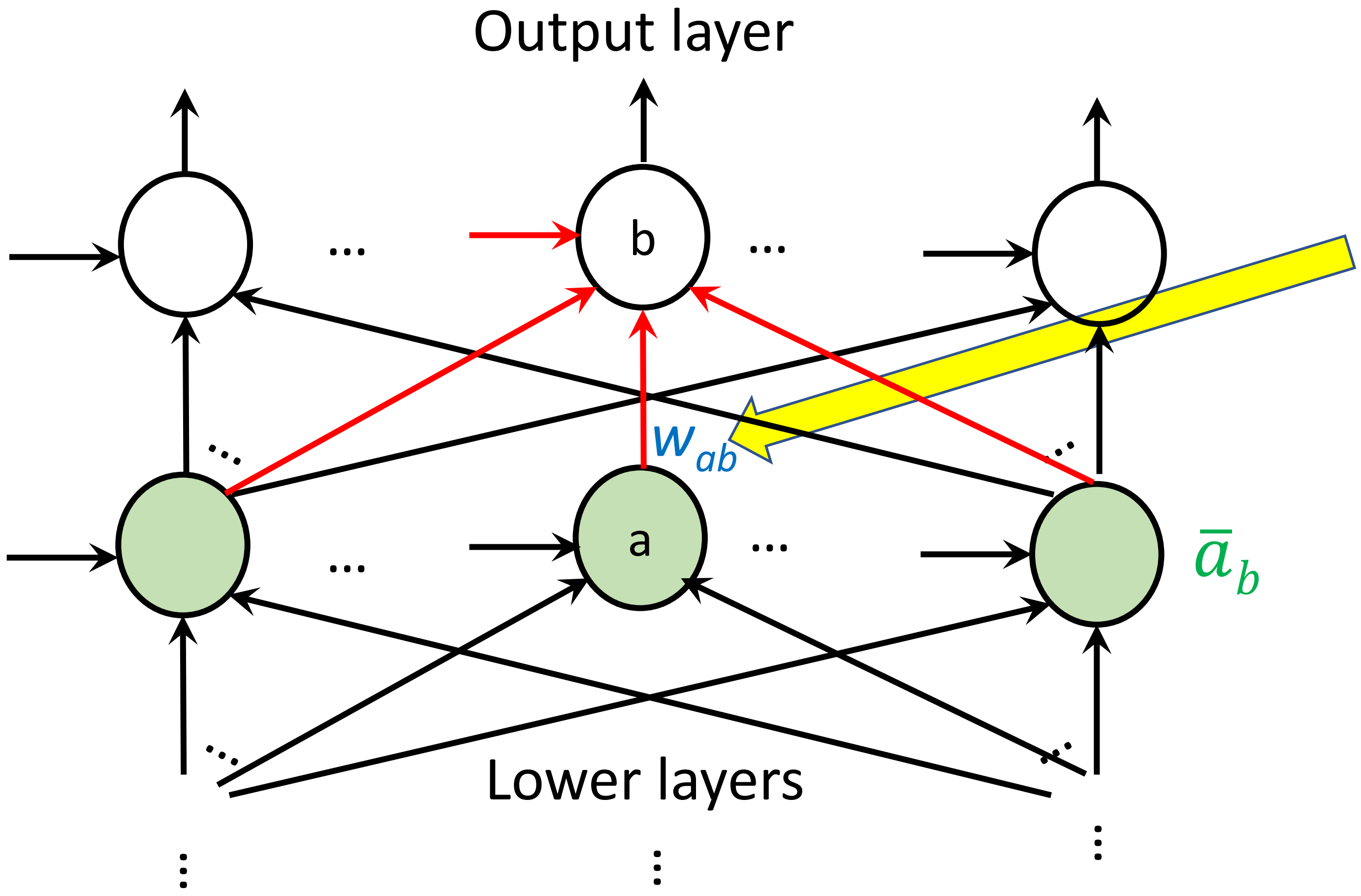
$$\alpha\, a_a\, g'(\bar{w}_b \cdot \bar{a}_b)(y_b - g(\bar{w}_b \cdot \bar{a}_b))$$

where $\bar{a}_b$ is the vector of activations going to $b$

- this is the same as before for a single neuron, but the inputs are the incoming activations rather than the $\bar{x}_i$

Up until now we've included an index *i* referring to the example we're currently doing an update for
Moving ahead I'm leaving that implicit, otherwise we'd have another index, i, on all the $\bar{a}_b$, $a_a$, and $y_b$, etc.

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer) Output Layer Weights

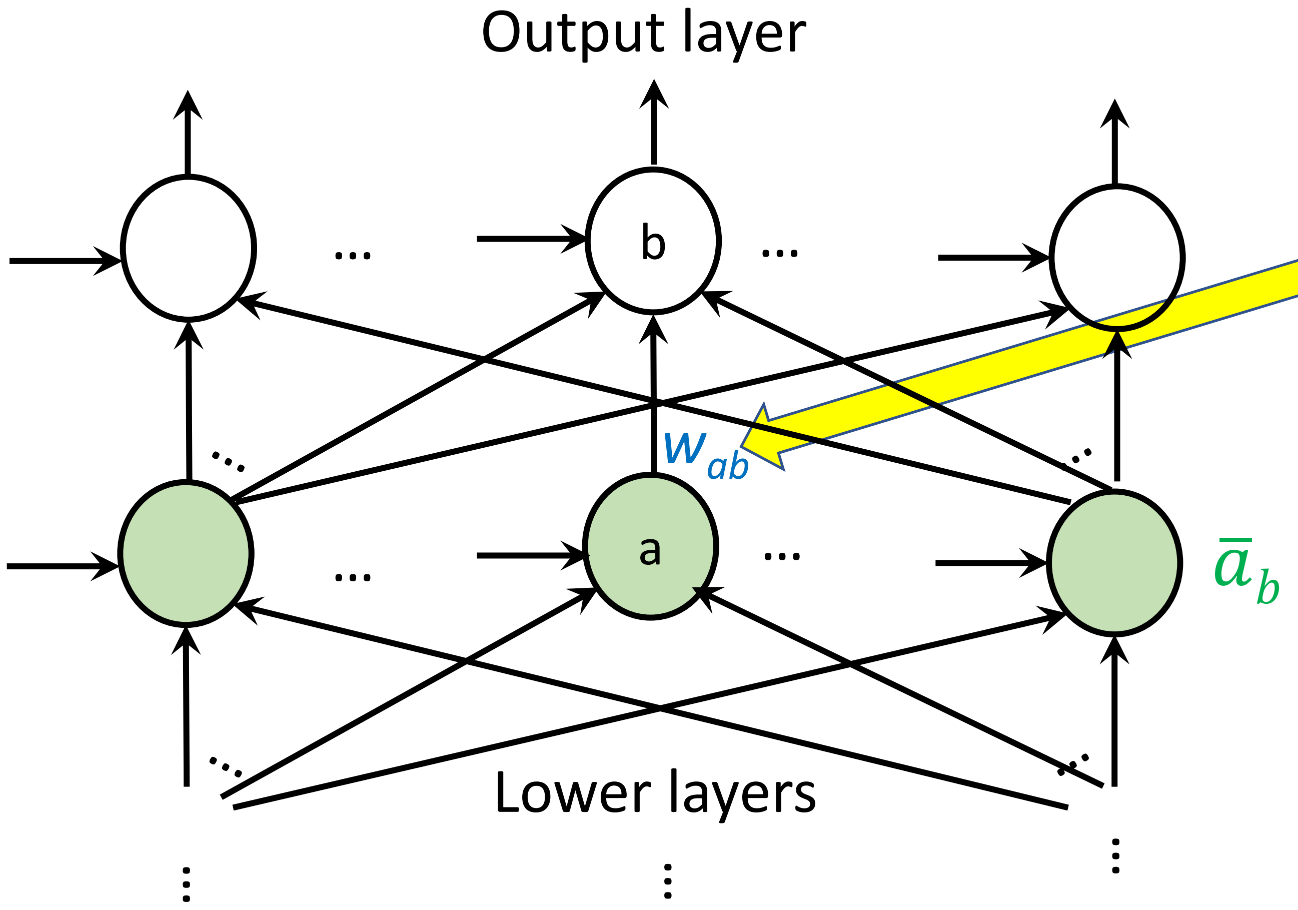



Change $w_{ab}$ by this much

$$\alpha\, a_a\, g'(\bar{w}_b \cdot \bar{a}_b)(y_b - g(\bar{w}_b \cdot \bar{a}_b))$$

Similarly $\bar{w}_b$ is the vector of weights going to $b$

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer) Output Layer Weights

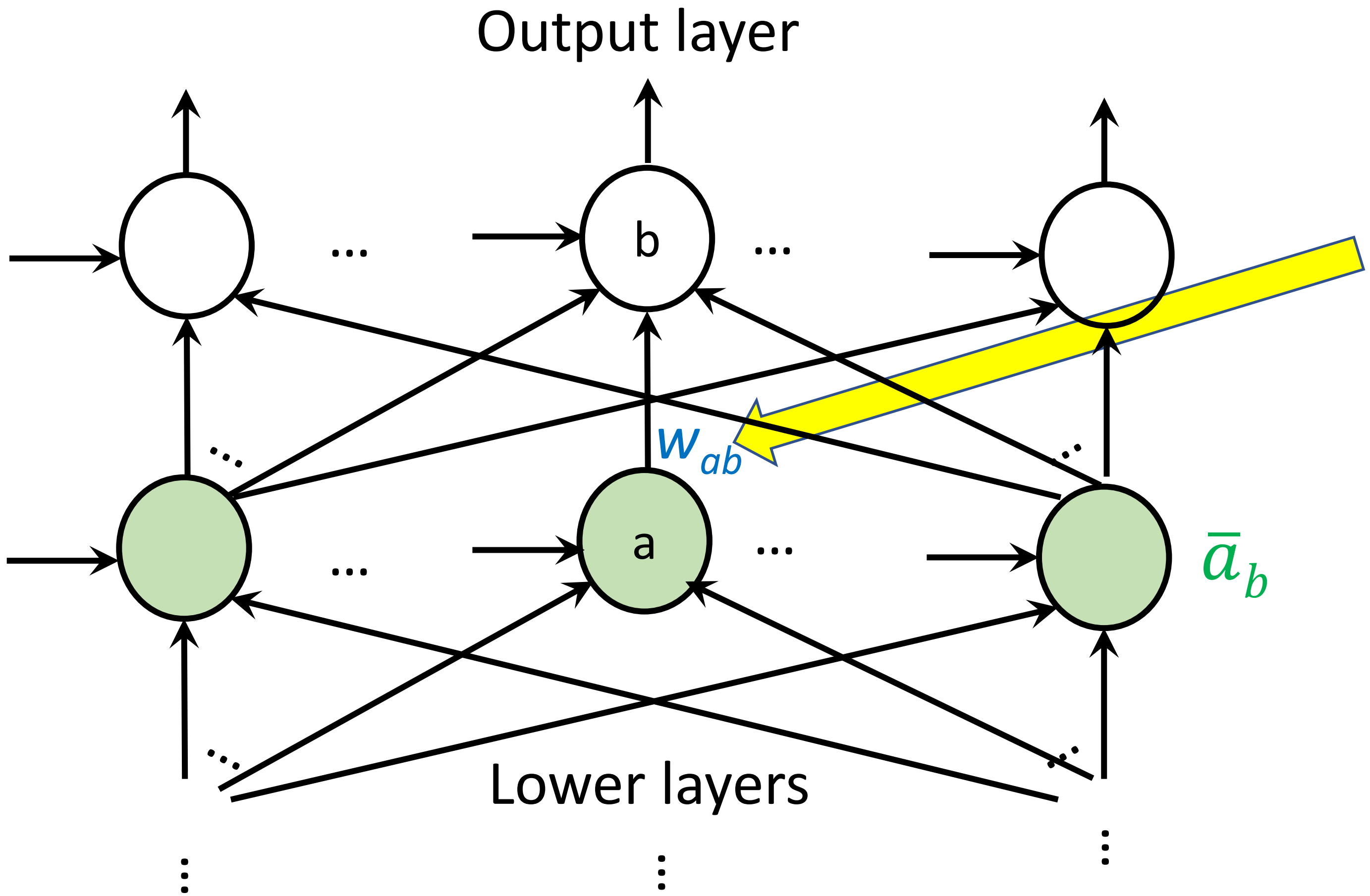



Change $w_{ab}$ by this much

$$\alpha\, a_a \boxed{g'(\bar{w}_b \cdot \bar{a}_b)(y_b - g(\bar{w}_b \cdot \bar{a}_b))}$$

This is the same as before

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer)
# Output Layer Weights

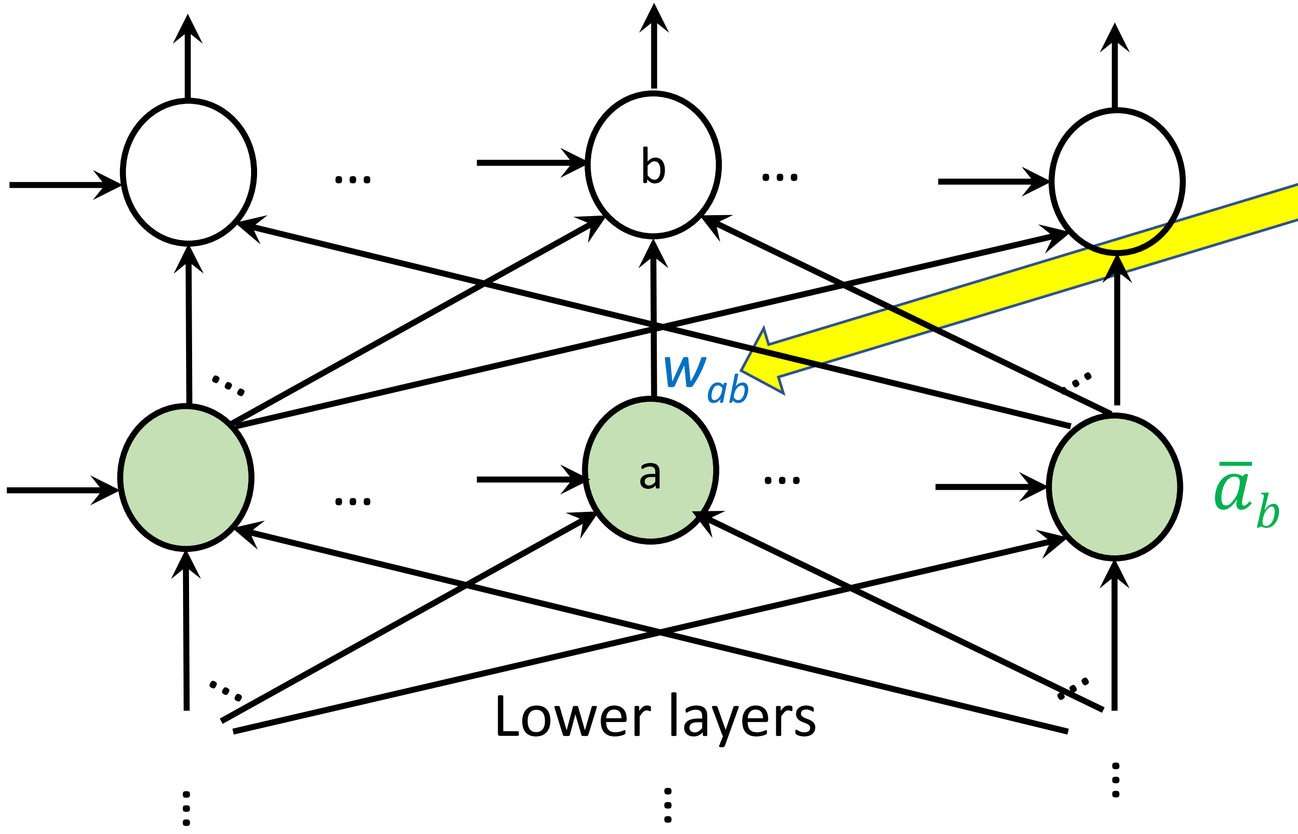



Change $w_{ab}$ by this much

$$\alpha\, a_a \boxed{g'(\bar{w}_b \cdot \bar{a}_b)(y_b - g(\bar{w}_b \cdot \bar{a}_b))}$$

Call it $\Delta_b$

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer) Output Layer Weights



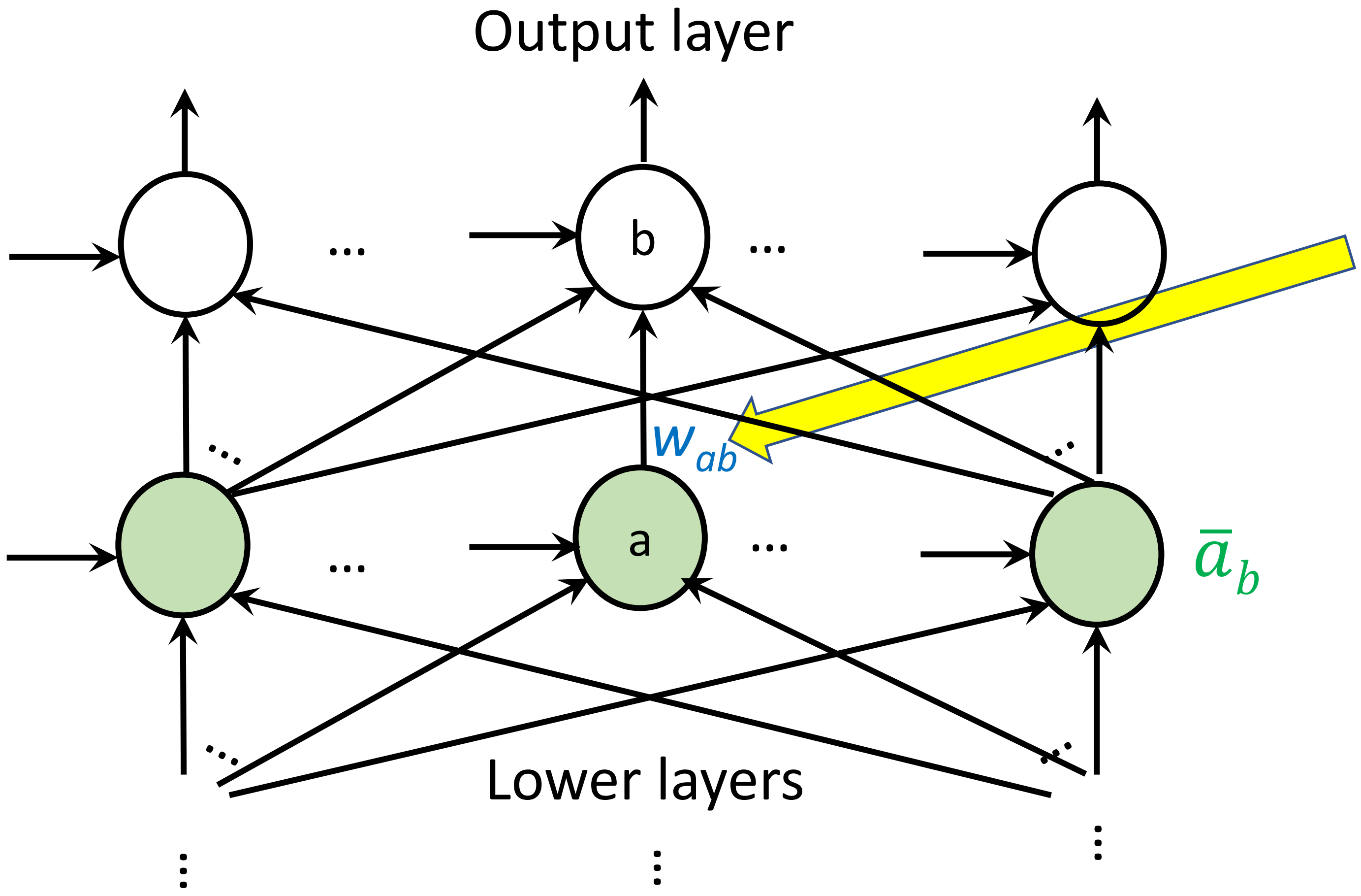

Change $w_{ab}$ by this much

$$\alpha\, a_a \boxed{g'(\bar{w}_b \cdot \bar{a}_b)(y_b - g(\bar{w}_b \cdot \bar{a}_b))}$$

Call it $\Delta_b$

Let $\mathrm{in}_b = \bar{w}_b \cdot \bar{a}_b$

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer) Output Layer Weights




Rewrite as

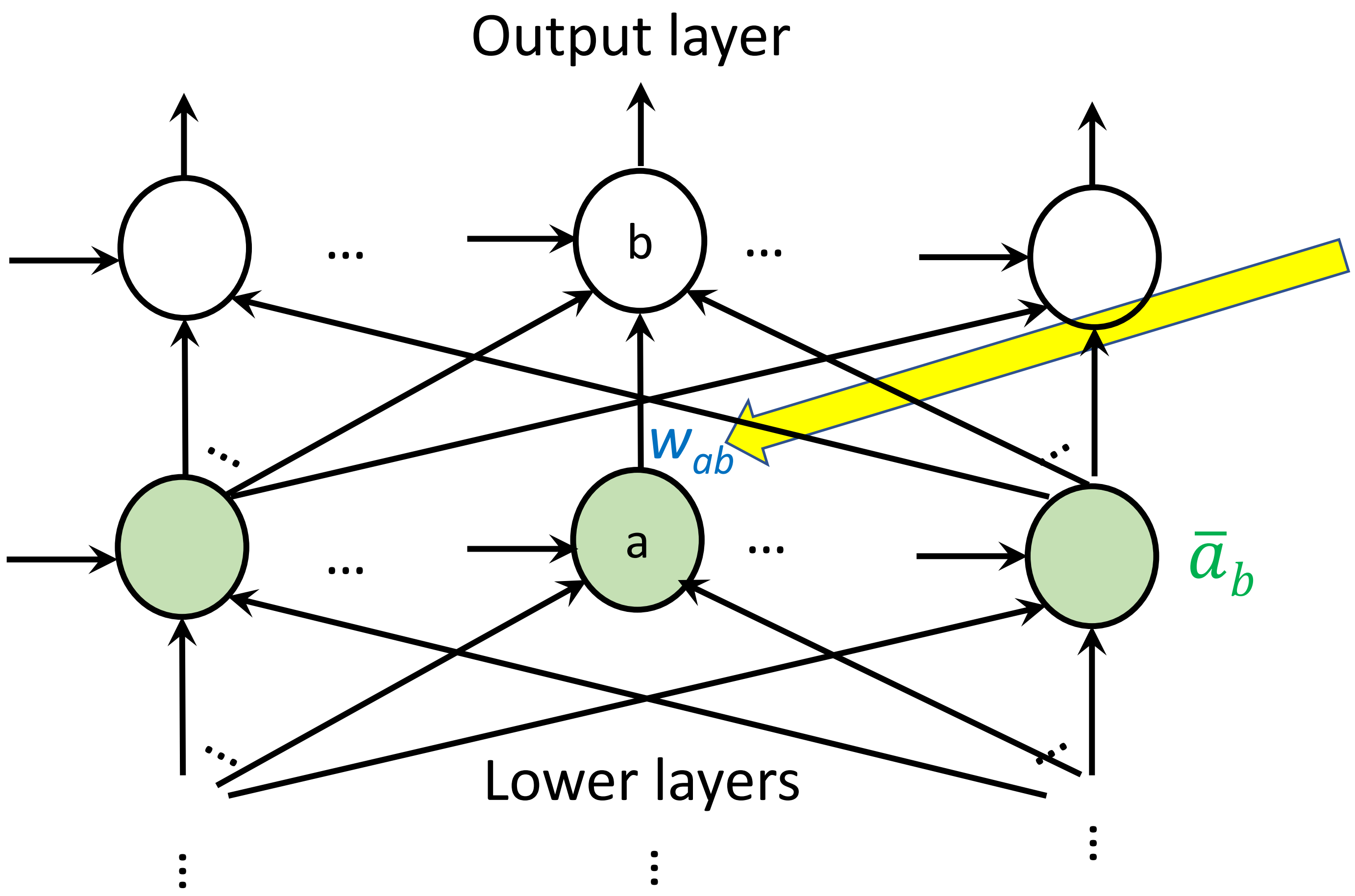
Change $w_{ab}$ by this much

$$\alpha\, a_a\, \Delta_b$$

$$\Delta_b = g'(\text{in}_b)(y_b - g(\text{in}_b))$$

The first step of error backpropagation just computes $\Delta_b$ for the output layer

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer) Output Layer Weights




Rewrite as

Change $w_{ab}$ by this much

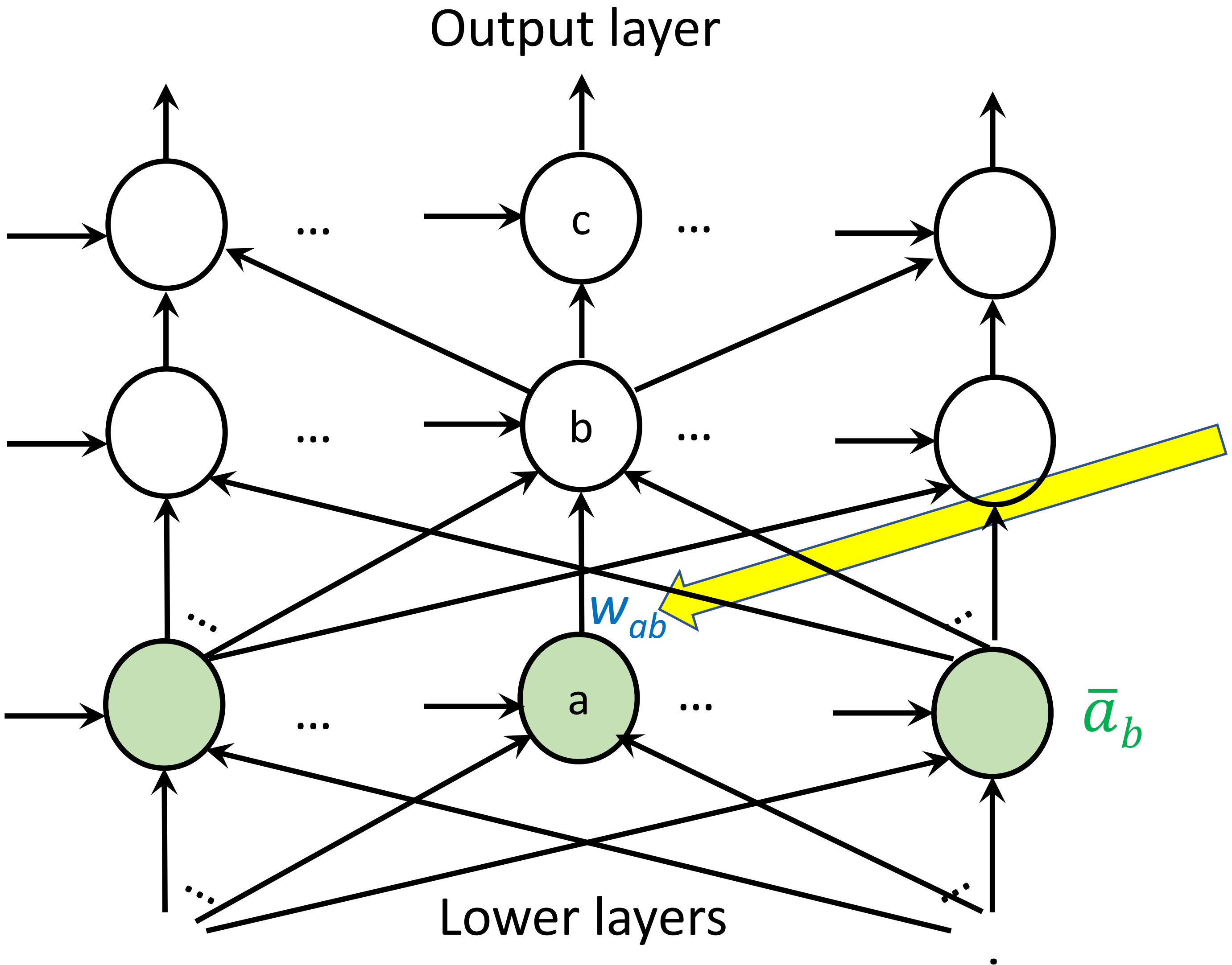
$\alpha\, a_a\, \Delta_b$

$\Delta_b = g'(\mathrm{in}_b)(y_b - g(\mathrm{in}_b))$

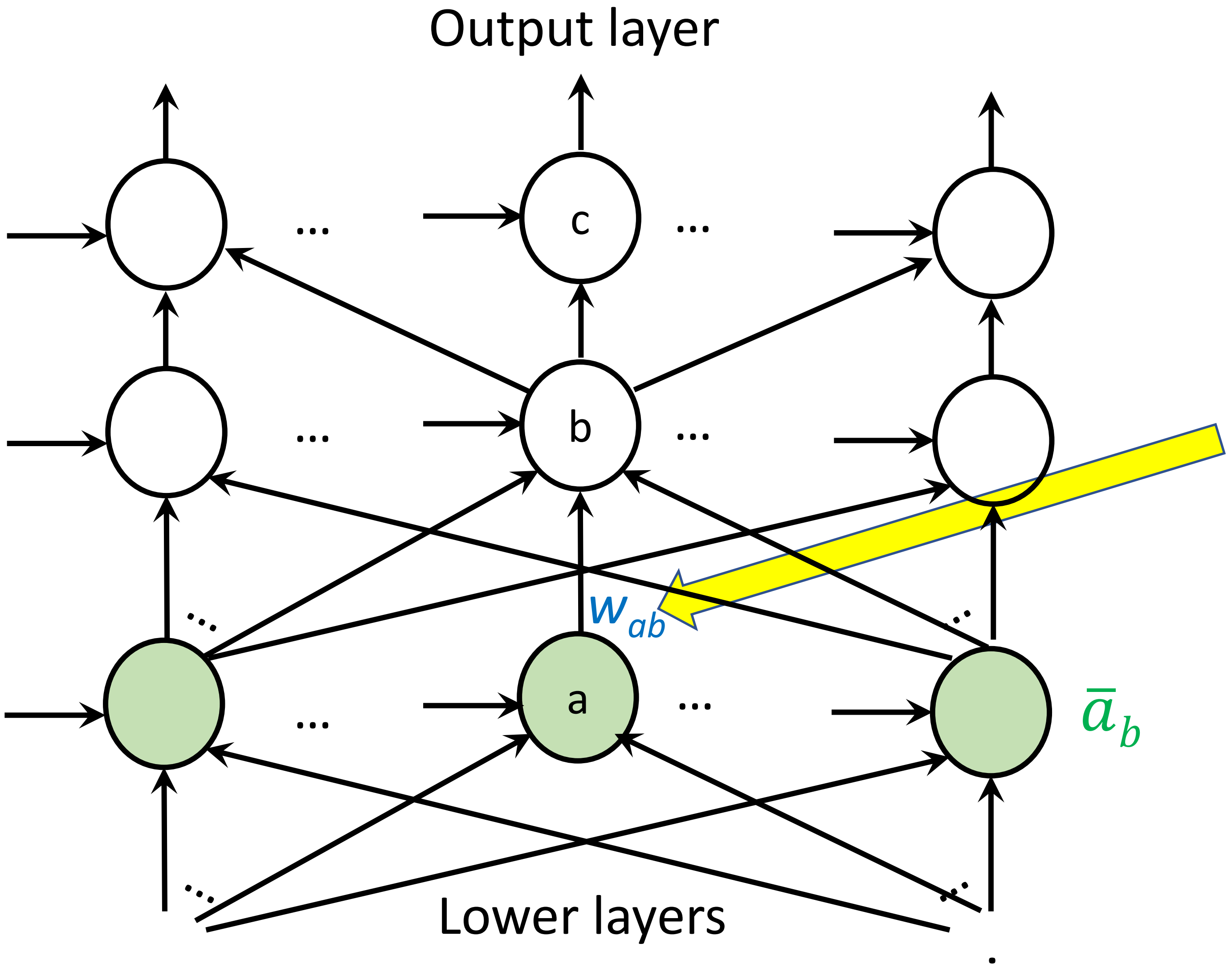
The first step of error backpropagation just computes $\Delta_b$ for the output layer

($\mathrm{in}_b$ and $g(\mathrm{in}_b)$ are computed during the feedforward step)

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer) Hidden Layer Weights



# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer) Hidden Layer Weights



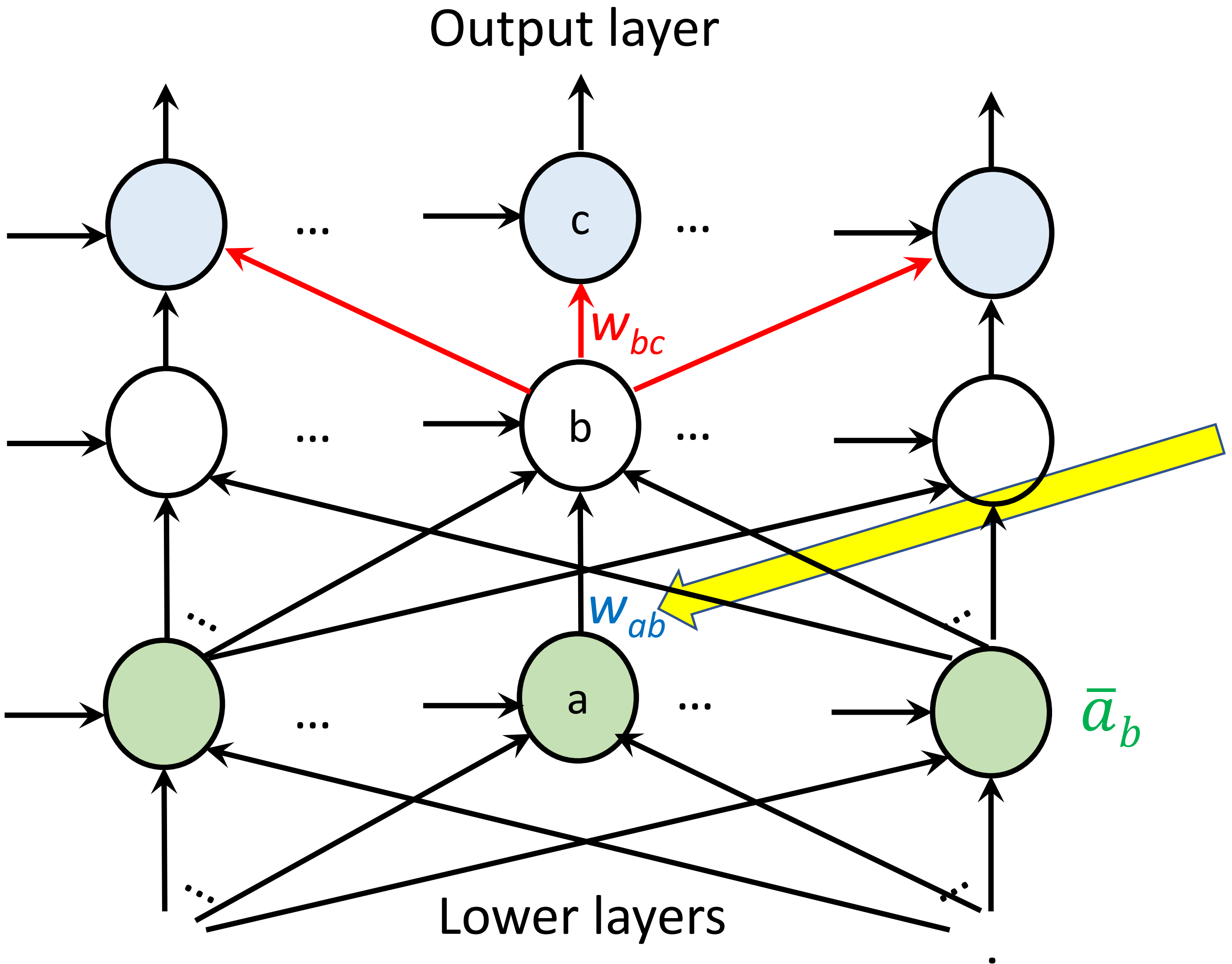

Keep this part the same

Change $w_{ab}$ by this much

$\alpha \, a_a \, \Delta_b$

but change how we calculate $\Delta_b$

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer) Hidden Layer Weights




Keep this part the same

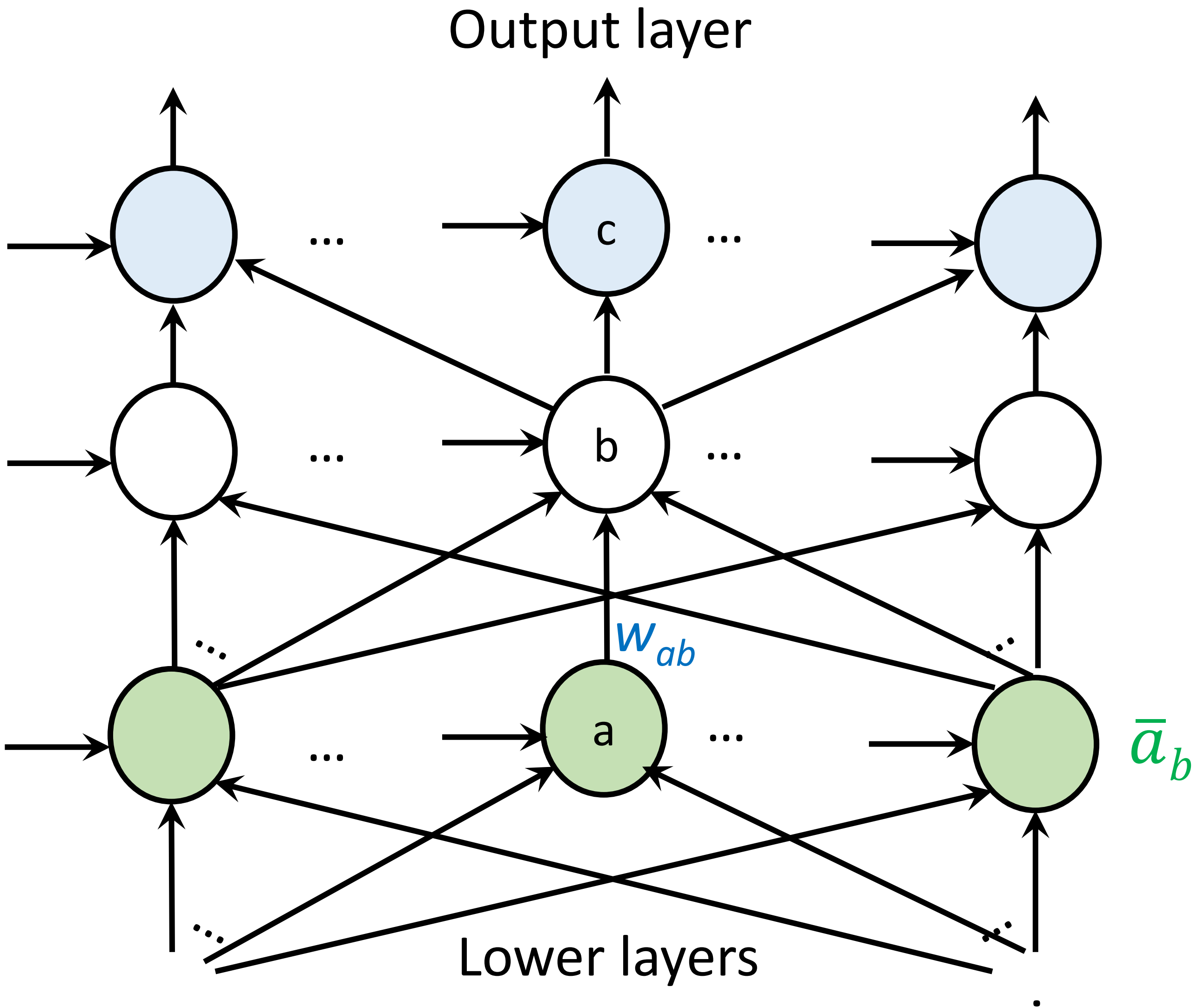
Change $w_{ab}$ by this much

$\alpha\ a_a\ \Delta_b$

but change how we calculate $\Delta_b$

It will be based on the $w_{bc}$'s and $\Delta_c$'s for every node $c$ that $b$ goes to

# Weight $w_{ab}$ Update for Gradient Descent (Multi-Layer) Hidden Layer Weights




Change $w_{ab}$ by this much

$\alpha \; a_a \; \Delta_b$

Once you have $\Delta_b$ for every node you can change each $w_{ab}$ by $\alpha \; a_a \Delta_b$ in one final loop

# Backprop Algorithm

Backprop(D,W):

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

Repeat

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

Feedforward:

For layers m=2 to k

For each node *j* in layer *m*: $in_j \leftarrow \sum_{\text{edge }(i,j)} w_{ij} a_i \qquad a_j \leftarrow g(in_j)$

Backpropagation:

For each node *j* in output layer $L_k$: $\Delta_j \leftarrow g'(in_j)(y_j - a_j)$

For layers m=k-1 down to 2

For each node $i$ in $L_m$: $\Delta_i \leftarrow g'(in_i) \sum_{\text{edge}(i,j)} w_{ij} \Delta_j$

For each weight $w_{ij}$: $w_{ij} \leftarrow w_{ij} + \alpha\, a_i\, \Delta_j$

Reorder the data

Until <stopping criterion>

# Additional Ideas in Practice

- "Mini-batch" updates:
  - Divide data into "batches", compute changes to $w_{ij}$ across each batch
- Other activation functions:
  - Rectifier Linear Unit (ReLU) activation function: g(z) = max(0,z)
  - Softmax: $$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$
  - …

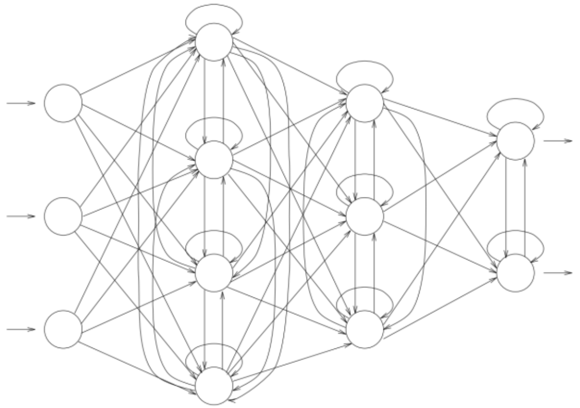
# Additional Ideas in Practice

How many layers, how many neurons in each level, how connected, ….

Evolving repertoire of approaches, for example:

- Recurrent neural networks: common for problems with sequential data, time series data, etc.

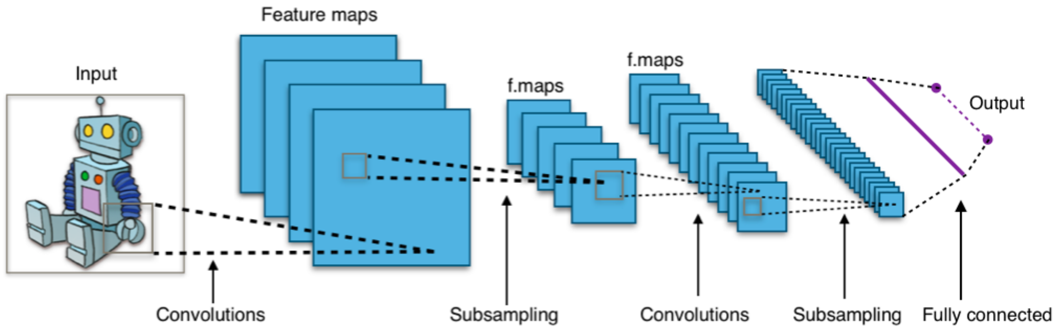
Example:



Tutschku, K., 1995. Recurrent multilayer perceptrons for identification and control: The road to applications.

- Convolutional neural network: common for problems involving image analysis

Example:




https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png

# Supervised Learning: Naïve Bayes

Probabilistic approach to supervised learning

# Supervised Learning: Naïve Bayes

- Basic concept:
  - Problem:
    - Given data D = { ($\bar{x}_i$,$y_i$) } for 1 ≤ i ≤ N
    - Label new item $\bar{x}_{test}$
  - Solution:
    - Assign label

$$\operatorname*{argmax}_{c \in C} P(c | x_{test,1} = v_{1,l_1}, x_{test,2} = v_{2,l_{2,}}, \dots, x_{test,n} = v_{n,l_n})$$

[this says: whichever c is most probable for $\bar{x}_{test}$]

    - When clear from context, will write:

$$\operatorname*{argmax}_{c \in C} P(c | x_{test,1}, \dots, x_{test,n})$$