Deep Learning:

Backpropagation & Symbolic Derivatives

Robert Legenstein

Institute of Machine Learning and Neural Computation

robert.legenstein@tugraz.at

Deep Learning VO - WS 25/26

Lecture 4

Recap: Gradient Descent

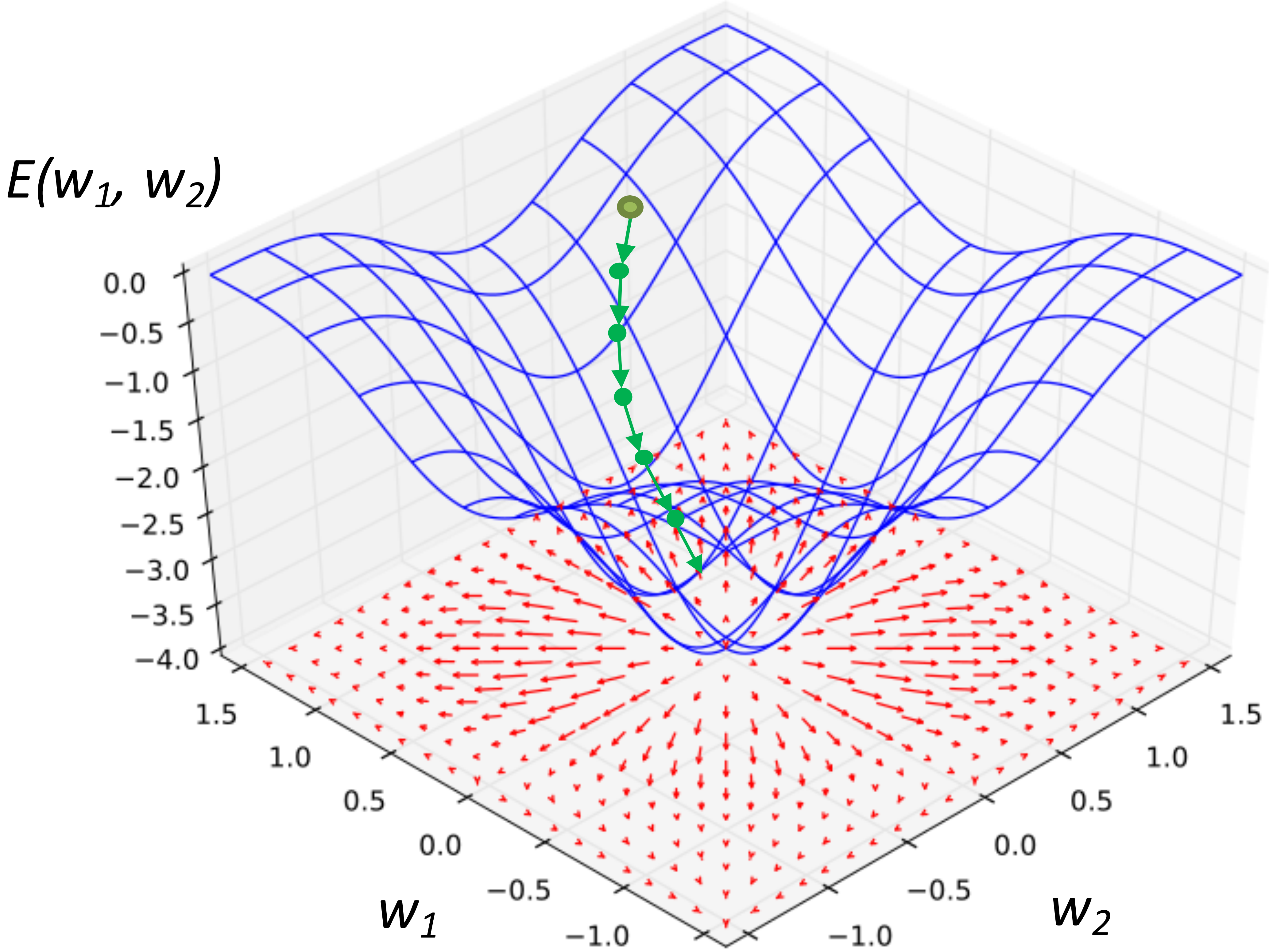
- The error is a **scalar field:**

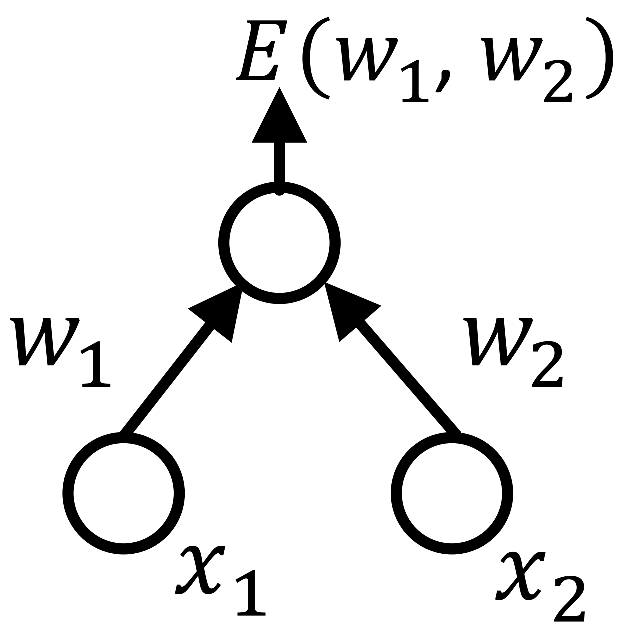
$$E : \mathbb{R}^D \rightarrow \mathbb{R}$$

- The gradient is a **vector field:**

$$\nabla_{\mathbf{w}} E : \mathbb{R}^D \rightarrow \mathbb{R}^D$$

$$\nabla_{\mathbf{w}} E = \begin{pmatrix} \frac{\partial E}{\partial w_1} \\ \vdots \\ \frac{\partial E}{\partial w_D} \end{pmatrix}$$





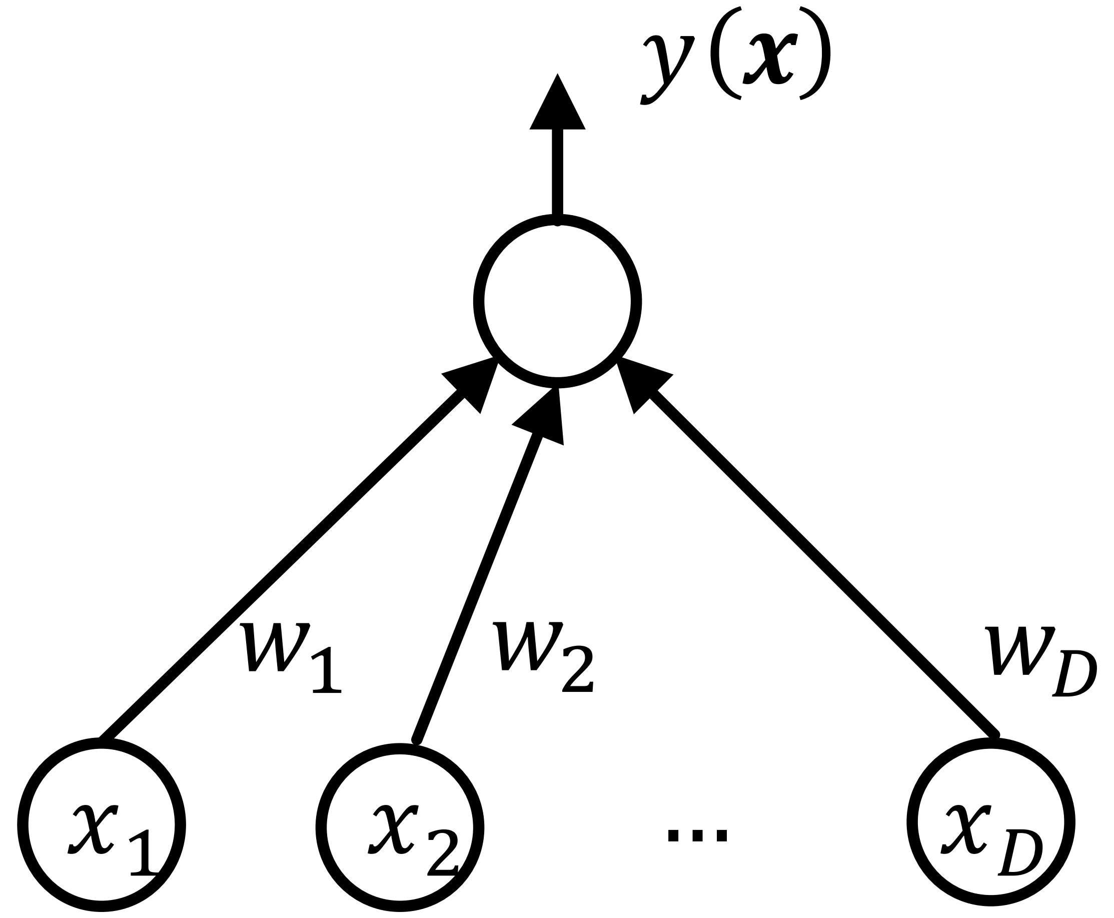
- It points in the direction of the steepest increase of the error.

- We slightly change parameters to reduce error.

$$\mathbf{w}_{new} = \mathbf{w}_{old} - \eta \nabla_{\mathbf{w}} E(\mathbf{w}_{old})$$

$\eta > 0$ is the **learning rate.**

Example: Single neuron regression with Gradient Descent



Weight vector: $\mathbf{w} \in \mathbb{R}^D$

Input vector: $\mathbf{x} \in \mathbb{R}^D$

$$y(\mathbf{x}) = h(\mathbf{w}^T\mathbf{x})$$

Given training examples: $\langle \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)} \rangle$ with targets $\mathbf{t} = (t^{(1)}, \ldots, t^{(M)})^T$.

$$a^{(m)} = \mathbf{w}^T\mathbf{x}^{(m)}$$

$$y^{(m)} = h(a^{(m)})$$

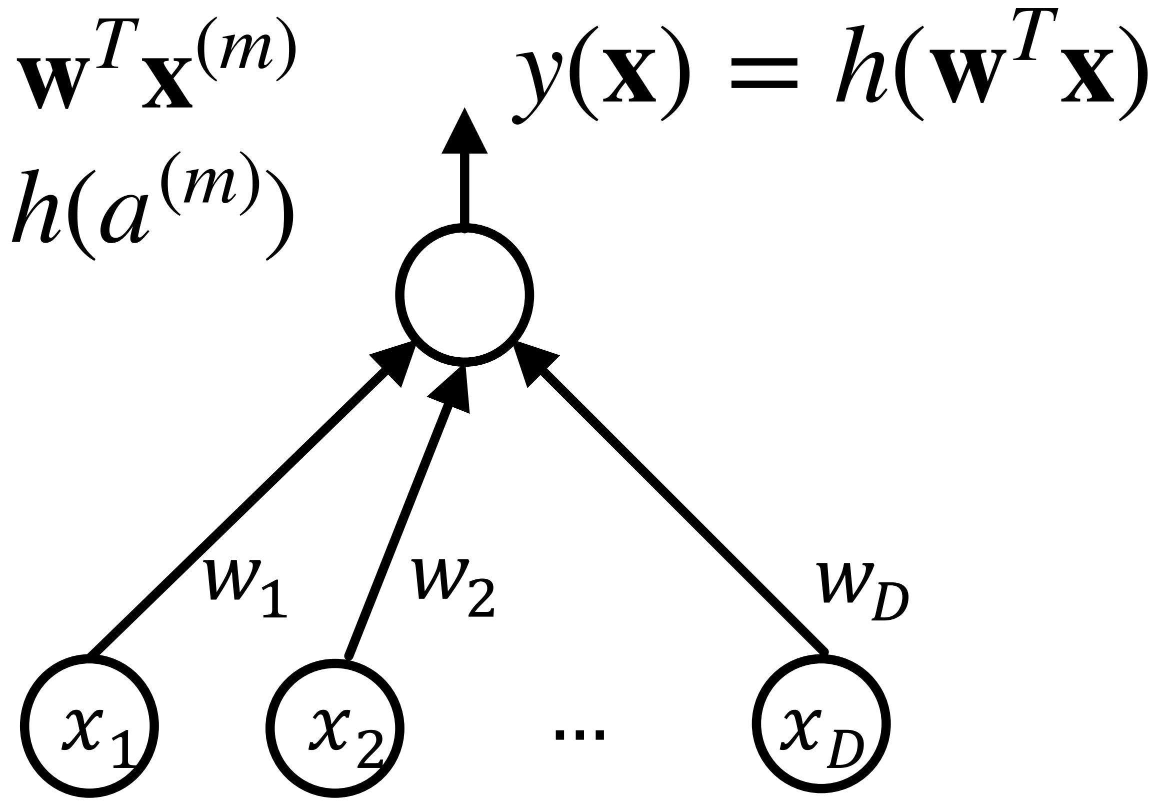
Error function:

$$E = \frac{1}{2}\sum_{m=1}^{M}\left(y^{(m)} - t^{(m)}\right)^2 = \sum_{m=1}^{M} E^{(m)} \quad \text{where} \quad E^{(m)} = \frac{1}{2}\left(y^{(m)} - t^{(m)}\right)^2$$

Example: Single neuron regression with Gradient Descent

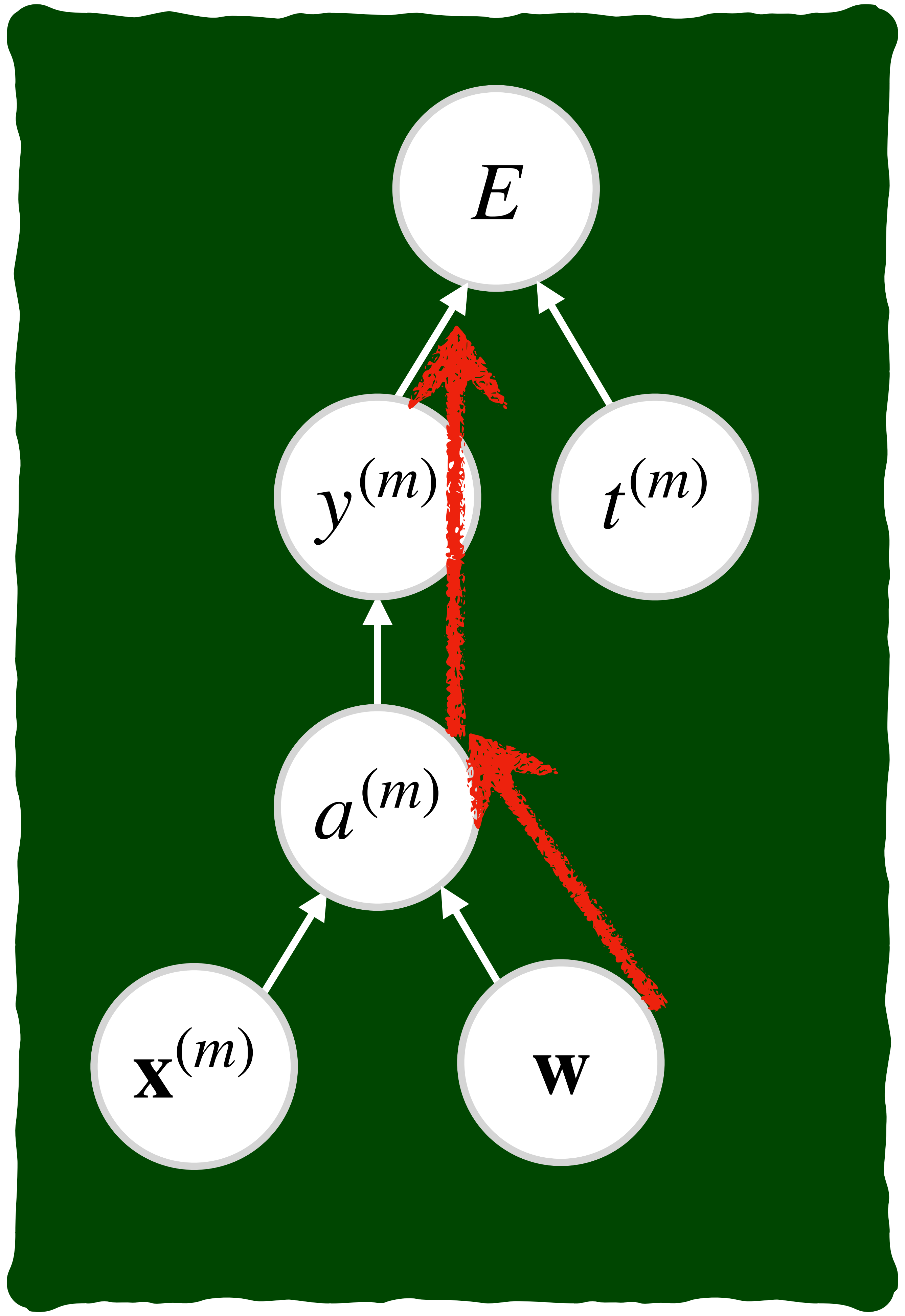
$$E = \sum_{m=1}^{M} E^{(m)} \quad \text{where} \quad E^{(m)} = \frac{1}{2}\left(y^{(m)} - t^{(m)}\right)^2$$

$$a^{(m)} = \mathbf{w}^T \mathbf{x}^{(m)}$$
$$y^{(m)} = h(a^{(m)})$$



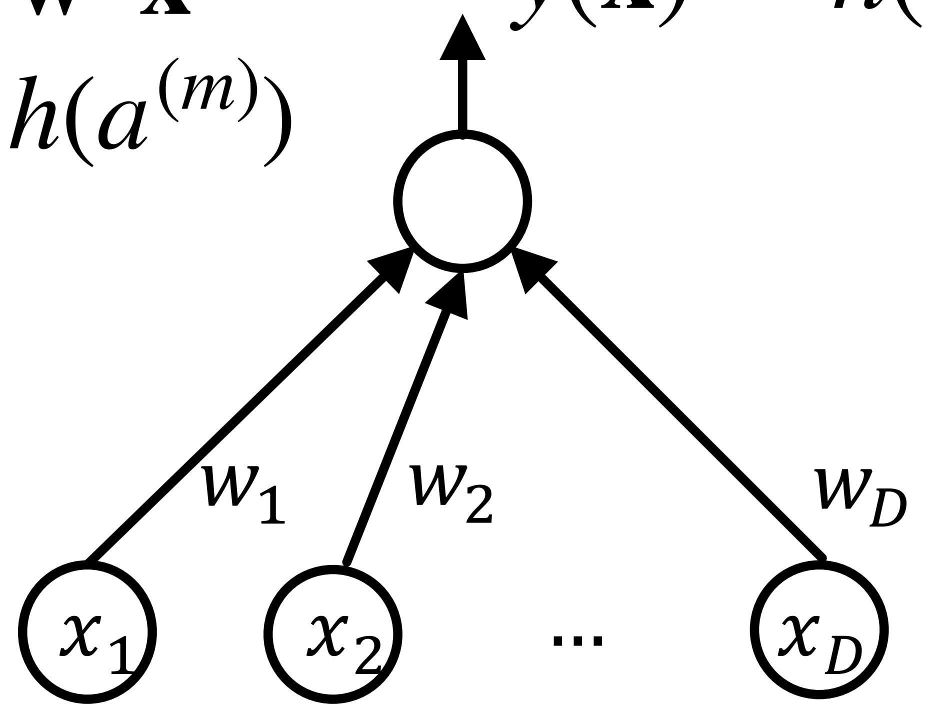
$$\nabla_{\mathbf{w}} E = \sum_{m} \nabla_{\mathbf{w}} E^{(m)}$$

$$\nabla_{\mathbf{w}} E^{(m)} = \frac{\partial E^{(m)}}{\partial a^{(m)}} \cdot \nabla_{\mathbf{w}} a^{(m)}$$



Example: Single neuron regression with Gradient Descent

$$a^{(m)} = \mathbf{w}^T\mathbf{x}^{(m)} \qquad y(\mathbf{x}) = h(\mathbf{w}^T\mathbf{x})$$
$$y^{(m)} = h(a^{(m)})$$



$$E = \sum_{m=1}^{M} E^{(m)} \quad \text{where} \quad E^{(m)} = \frac{1}{2}\left(y^{(m)} - t^{(m)}\right)^2$$

$$\nabla_{\mathbf{w}} E = \sum_{m} \nabla_{\mathbf{w}} E^{(m)}$$

$$\nabla_{\mathbf{w}} E^{(m)} = \frac{\partial E^{(m)}}{\partial a^{(m)}} \cdot \nabla_{\mathbf{w}} a^{(m)}$$

$$\nabla_{\mathbf{w}} a^{(m)} = \nabla_{\mathbf{w}}(\mathbf{w}^T\mathbf{x}^{(m)}) = \mathbf{x}^{(m)}$$

$$\frac{\partial E^{(m)}}{\partial a^{(m)}} = (y^{(m)} - t^{(m)}) \cdot \frac{\partial y^{(m)}}{\partial a^{(m)}} = (y^{(m)} - t^{(m)}) \cdot h'(a^{(m)})$$

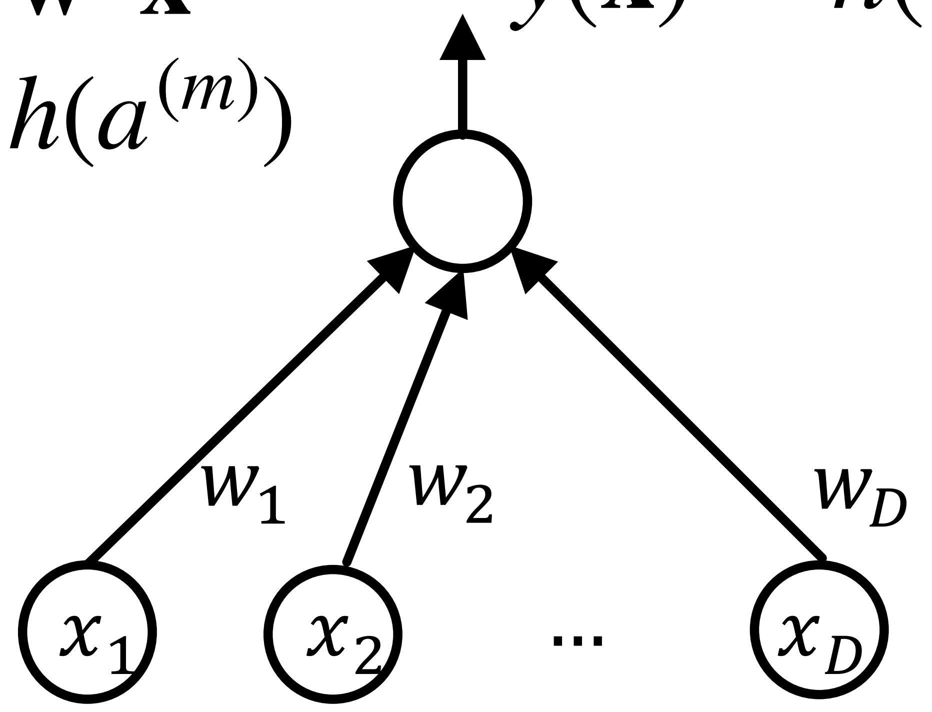
$$\nabla_{\mathbf{w}} E^{(m)} = (y^{(m)} - t^{(m)}) \cdot h'(a^{(m)}) \cdot \mathbf{x}^{(m)}$$

Example: Single neuron regression with Gradient Descent

$$\nabla_{\mathbf{w}} E^{(m)} = (y^{(m)} - t^{(m)}) \cdot h'(a^{(m)}) \cdot \mathbf{x}^{(m)}$$

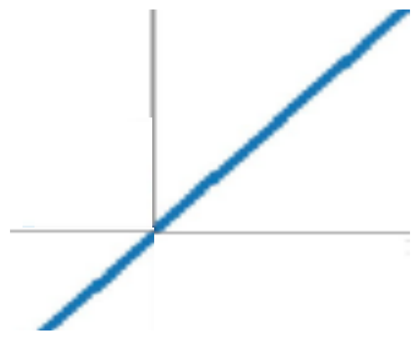
$$a^{(m)} = \mathbf{w}^T \mathbf{x}^{(m)}$$
$$y^{(m)} = h(a^{(m)})$$

$$y(\mathbf{x}) = h(\mathbf{w}^T \mathbf{x})$$



For a linear neuron:

$$h(a^{(m)}) = a^{(m)}$$
$$h'(a^{(m)}) = 1$$



$$\nabla_{\mathbf{w}} E^{(m)} = (y^{(m)} - t^{(m)}) \cdot \mathbf{x}^{(m)}$$

Batch GD:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \sum_{m=1}^{M} (t^{(m)} - y^{(m)}) \cdot \mathbf{x}^{(m)}$$

Stochastic GD:

batch size = 1

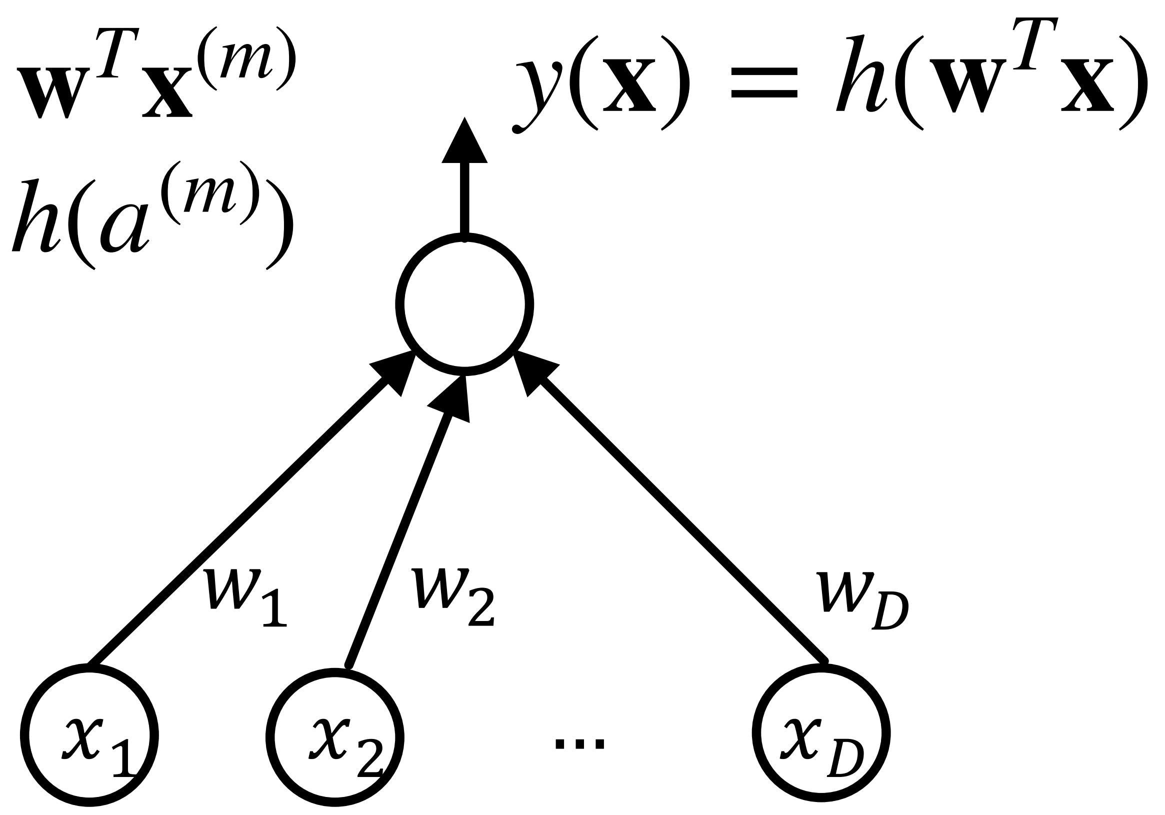
while not converged
  for $m \leftarrow 1$ to M
    $\mathbf{w} \leftarrow \mathbf{w} + \eta(t^{(m)} - y^{(m)}) \cdot \mathbf{x}^{(m)}$

Gradient Descent for standard linear regression

Example: Single neuron regression with Gradient Descent

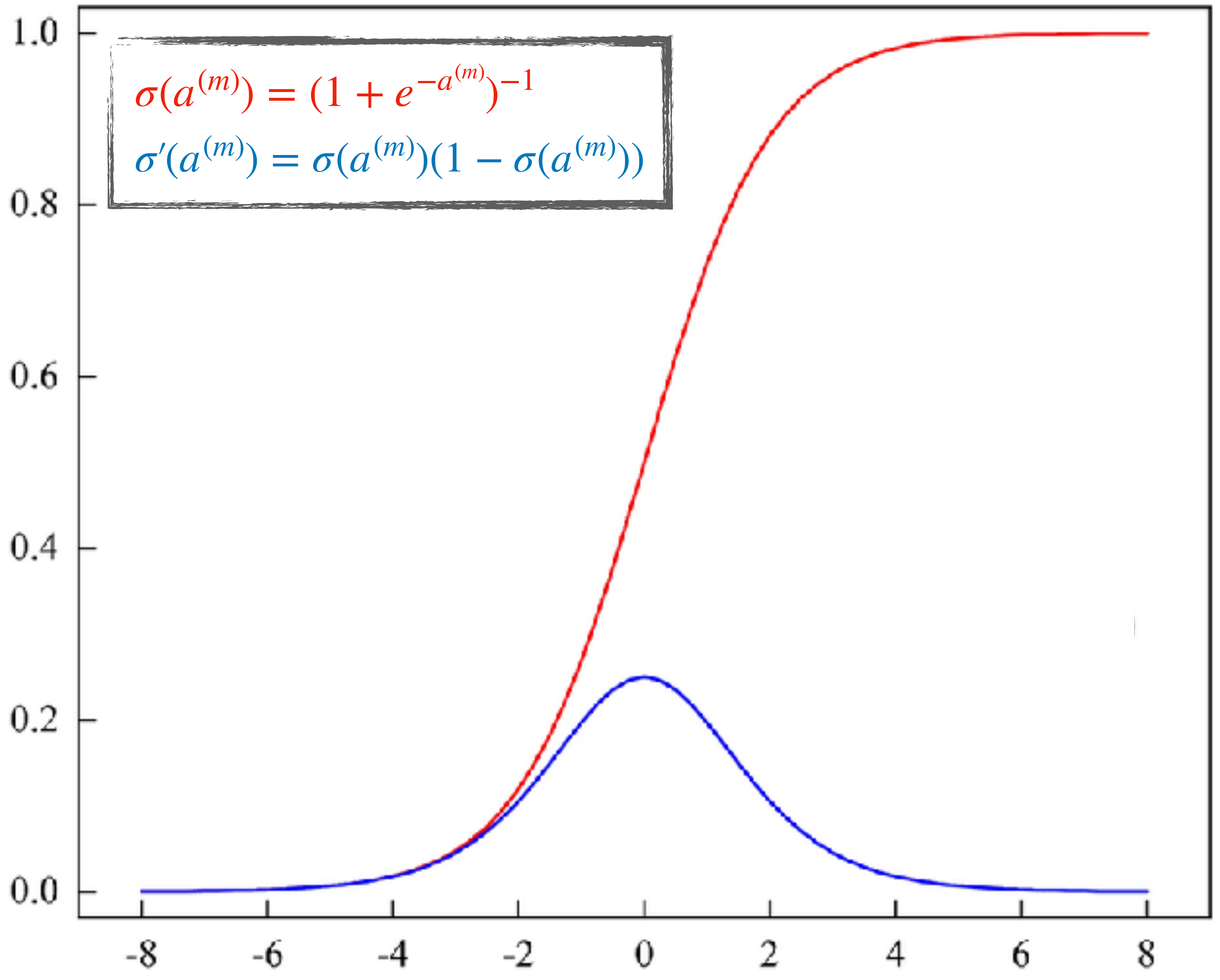
$$\nabla_{\mathbf{w}} E^{(m)} = (y^{(m)} - t^{(m)}) \cdot h'(a^{(m)}) \cdot \mathbf{x}^{(m)}$$

$$a^{(m)} = \mathbf{w}^T \mathbf{x}^{(m)}$$
$$y^{(m)} = h(a^{(m)})$$



For a nonlinear neuron with logsig nonlinearity:

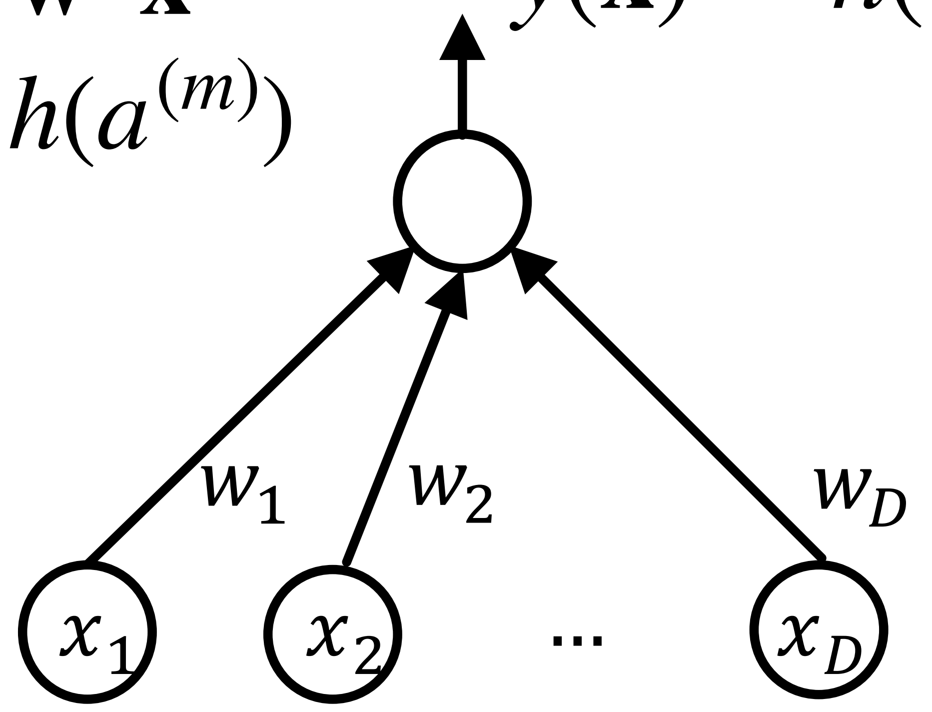
$$h(a^{(m)}) = \sigma(a^{(m)})$$
$$h'(a^{(m)}) = \sigma(a^{(m)})(1 - \sigma(a^{(m)}))$$



Example: Single neuron regression with Gradient Descent

$$a^{(m)} = \mathbf{w}^T \mathbf{x}^{(m)}$$
$$y^{(m)} = h(a^{(m)})$$

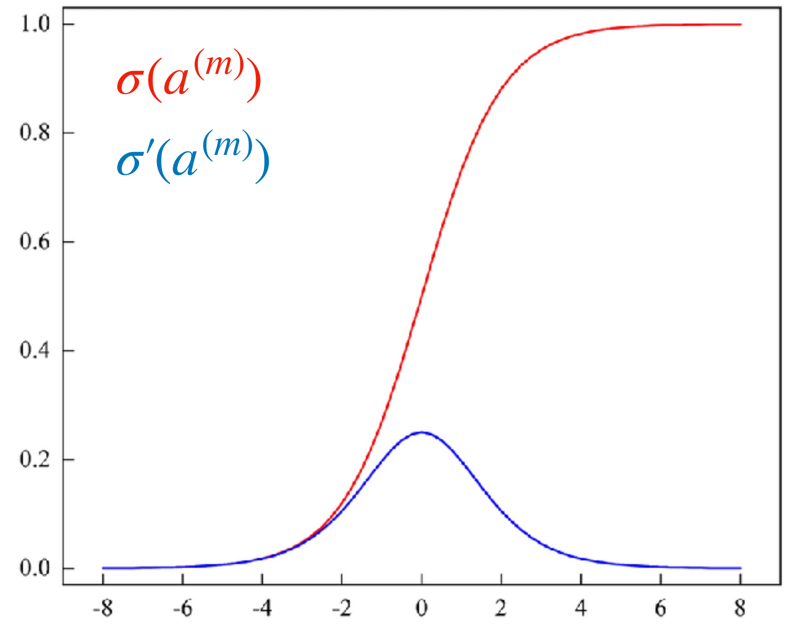
$$y(\mathbf{x}) = h(\mathbf{w}^T \mathbf{x})$$



$$\nabla_{\mathbf{w}} E^{(m)} = (y^{(m)} - t^{(m)}) \cdot h'(a^{(m)}) \cdot \mathbf{x}^{(m)}$$

<u>For a nonlinear neuron with logsig nonlinearity:</u>

$$h(a^{(m)}) = \sigma(a^{(m)})$$
$$h'(a^{(m)}) = \sigma(a^{(m)})(1 - \sigma(a^{(m)}))$$



Batch GD:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \sum_{m=1}^{M} (t^{(m)} - y^{(m)}) \cdot \mathbf{x}^{(m)} \cdot \sigma(a^{(m)})(1 - \sigma(a^{(m)}))$$

Stochastic GD:

batch size = 1

while not converged

for $m \leftarrow 1$ to M

$$\mathbf{w} \leftarrow \mathbf{w} + \eta (t^{(m)} - y^{(m)}) \cdot \mathbf{x}^{(m)} \cdot \sigma(a^{(m)})(1 - \sigma(a^{(m)}))$$

Today

- ☐ Neural Network Training
  - ☑ Error (Loss) Functions
  - ☑ Gradient Descent
  - ☐ Backpropagation
  - ☐ Symbolic Derivatives

Paul Werbos: "Beyond regression: New tools for prediction and analysis in the behavioral sciences." PhD Thesis. Harvard University 1974.

David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams: "Learning representations by back-propagating errors.", Nature, 1986.

Layered Network Forward Pass

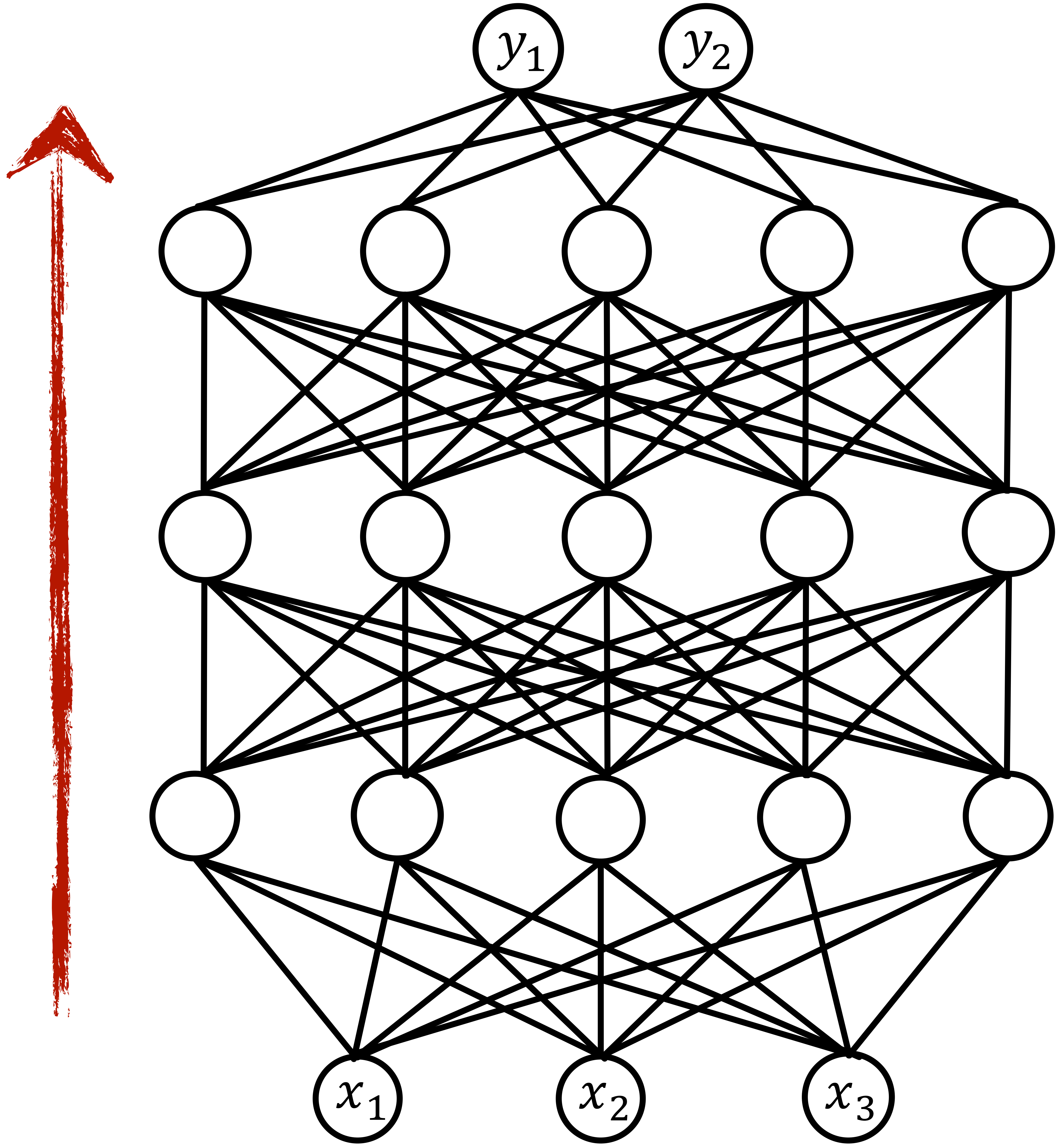
$$\mathbf{y}(\mathbf{x}) = \mathbf{z}^{(L)}(\mathbf{x})$$

$$\mathbf{z}^{(l)}(\mathbf{x}) = h^{(l)}\left(\mathbf{a}^{(l)}(\mathbf{x})\right)$$

$$\mathbf{a}^{(l)}(\mathbf{x}) = W^{(l)}\mathbf{z}^{(l-1)}(\mathbf{x})$$

(we include the bias in the weight matrix for simplicity)

$$\mathbf{z}^{(0)}(\mathbf{x}) = \mathbf{x}$$



Backpropagation

An algorithm to efficiently compute gradients.

E → Error for training sample $\mathbf{x}, \mathbf{t}$.

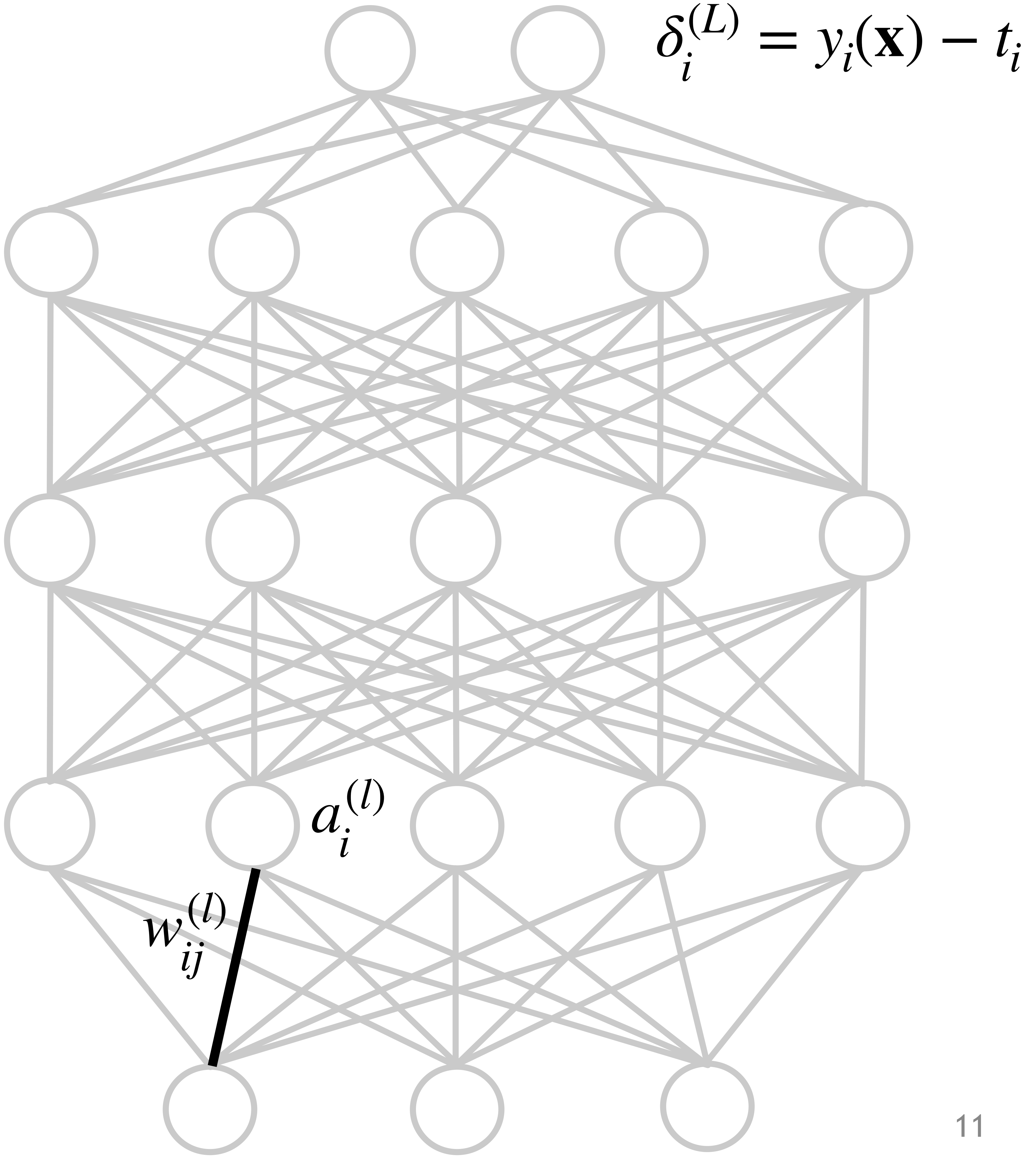
We want: $\dfrac{\partial E}{\partial w_{ij}^{(l)}}$

"Error" of neuron i in layer l: $\delta_i^{(l)} \stackrel{\text{def}}{=} \dfrac{\partial E}{\partial a_i^{(l)}}$

For our usual (error function & output-activation) combinations, we obtain:

$$\delta_i^{(L)} = y_i(\mathbf{x}) - t_i$$

$$\delta^{(L)} = \mathbf{y}(\mathbf{x}) - \mathbf{t}$$



Backpropagation

An algorithm to efficiently compute gradients.

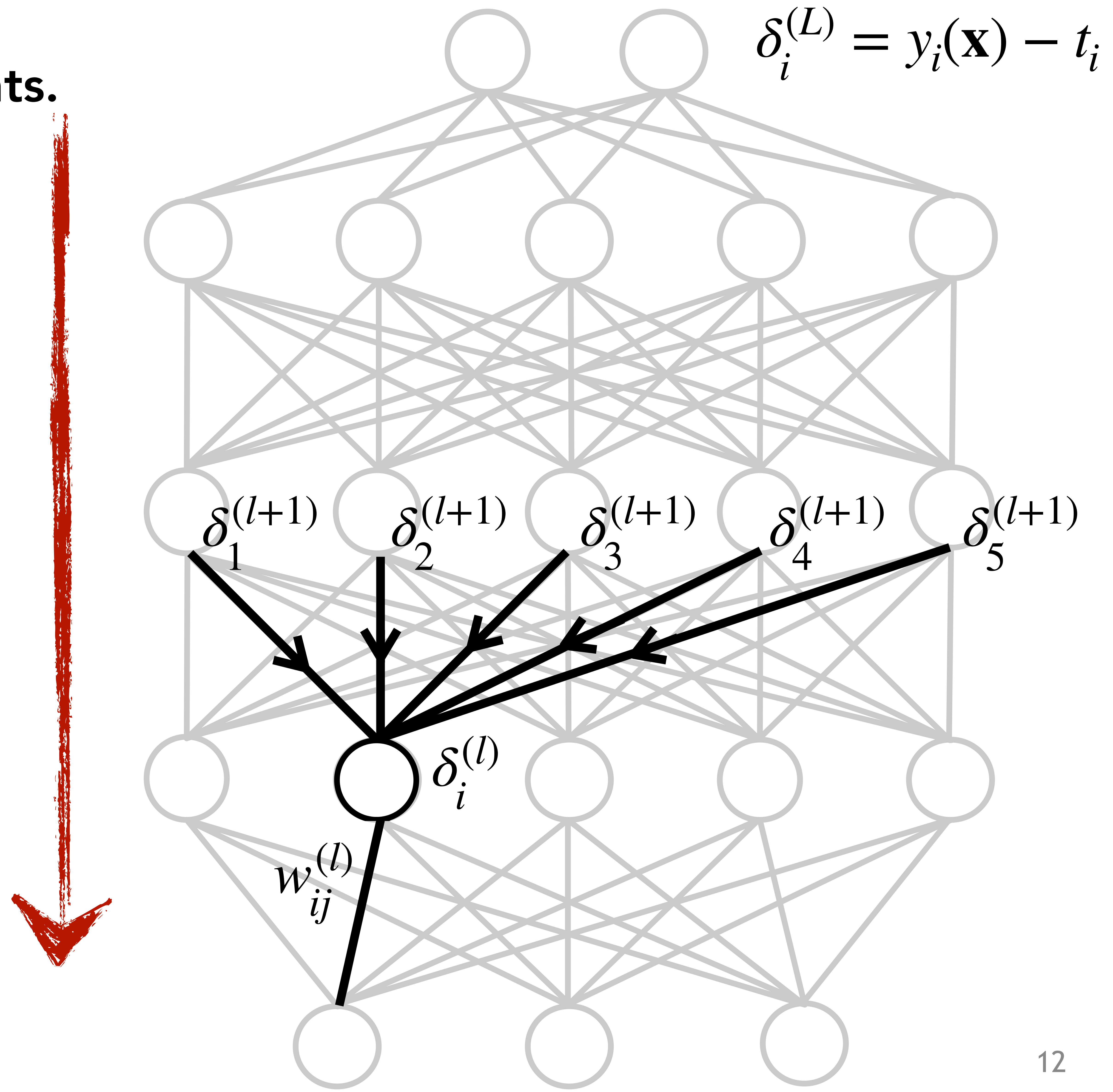
"credit assignment"

$$\delta_i^{(l)} = h^{(l)'}\left(a_i^{(l)}\right) \sum_j w_{ji}^{(l+1)} \delta_j^{(l+1)}$$

$$\delta^{(l)} = h^{(l)'}\left(\mathbf{a}^{(l)}\right) \odot \left(W^{(l+1)^T} \delta^{(l+1)}\right)$$

$h^{(l)'}$ → derivative of activation function

$\odot$ → component-wise product



Backpropagation

An algorithm to efficiently compute gradients.

"credit assignment"

$$\delta_i^{(l)} = h^{(l)'}\left(a_i^{(l)}\right) \sum_j w_{ji}^{(l+1)} \delta_j^{(l+1)}$$

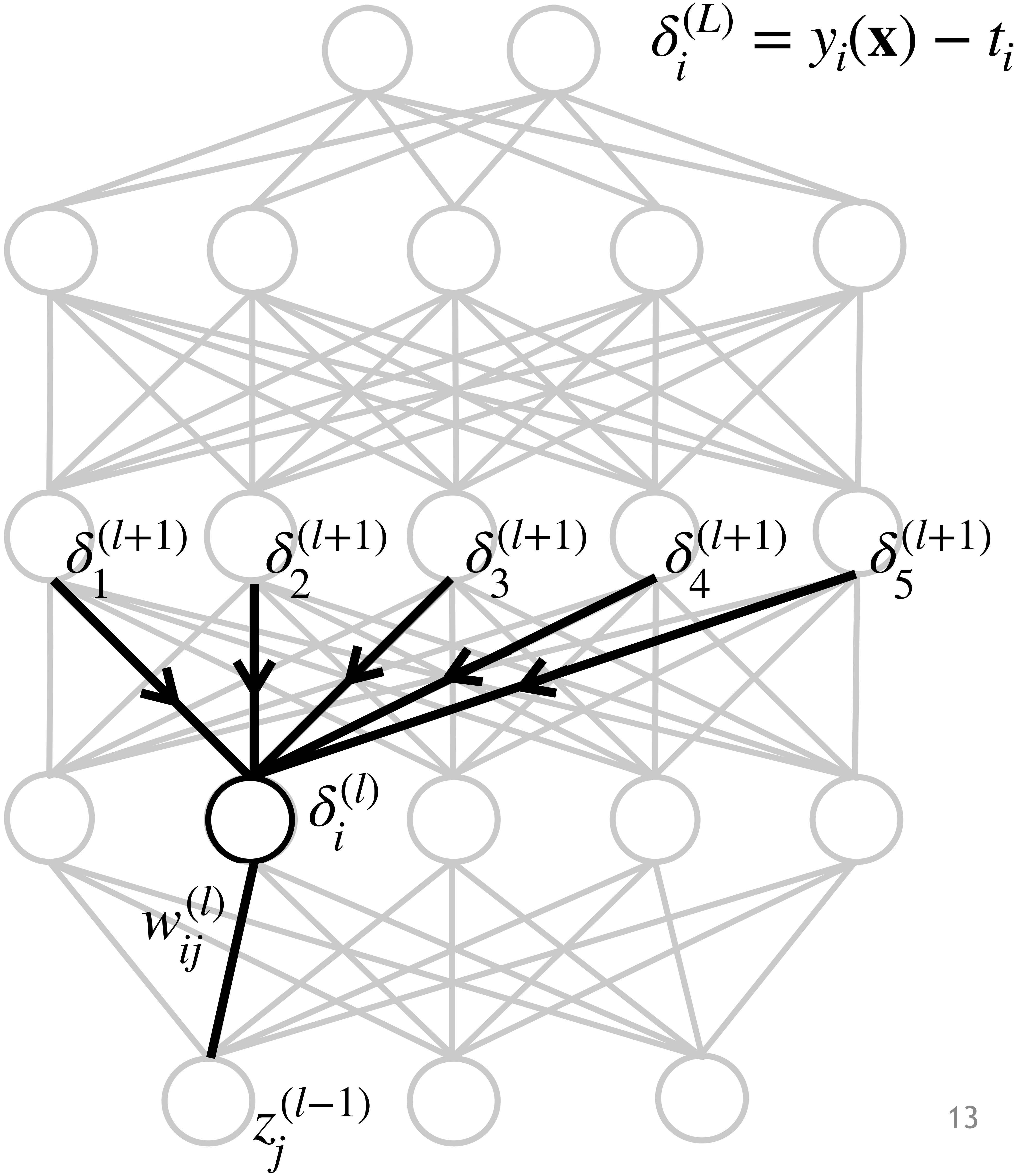
$$\delta^{(l)} = h^{(l)'}\left(\mathbf{a}^{(l)}\right) \odot \left(W^{(l+1)^T} \delta^{(l+1)}\right)$$

Parameter gradients:

$$\boxed{\frac{\partial E}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} z_j^{(l-1)}}$$

$$\nabla_{W^{(l)}} E = \delta^{(l)} \mathbf{z}^{(l-1)^T} \qquad \nabla_{W^{(l)}} E \stackrel{\text{def}}{=} \left[\frac{\partial E}{\partial w_{ij}^{(l)}}\right]_{ij}$$

$$\delta_i^{(L)} = y_i(\mathbf{x}) - t_i$$



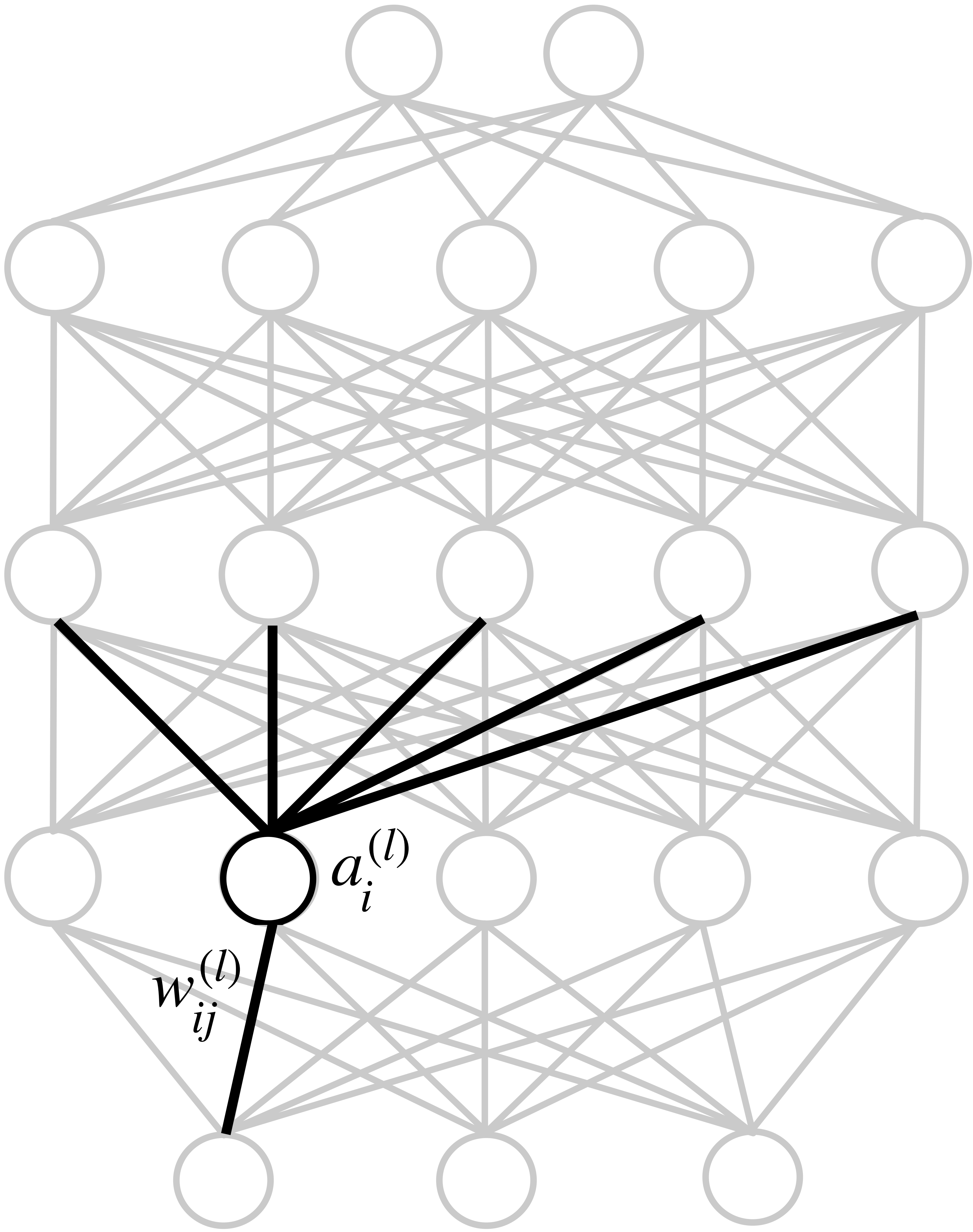
Backpropagation: Derivation

$$\delta_i^{(l)} \stackrel{\text{def}}{=} \frac{\partial E}{\partial a_i^{(l)}}$$

$$a_i^{(l)} = \sum_k w_{ik}^{(l)} z_k^{(l-1)}$$

Parameter gradients:

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{\partial E}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} z_j^{(l-1)}$$



Backpropagation: Derivation

The δ's for output neurons:

$$\delta_i^{(L)} = \frac{\partial E}{\partial a_i^{(L)}} = y_i(\mathbf{x}) - t_i$$

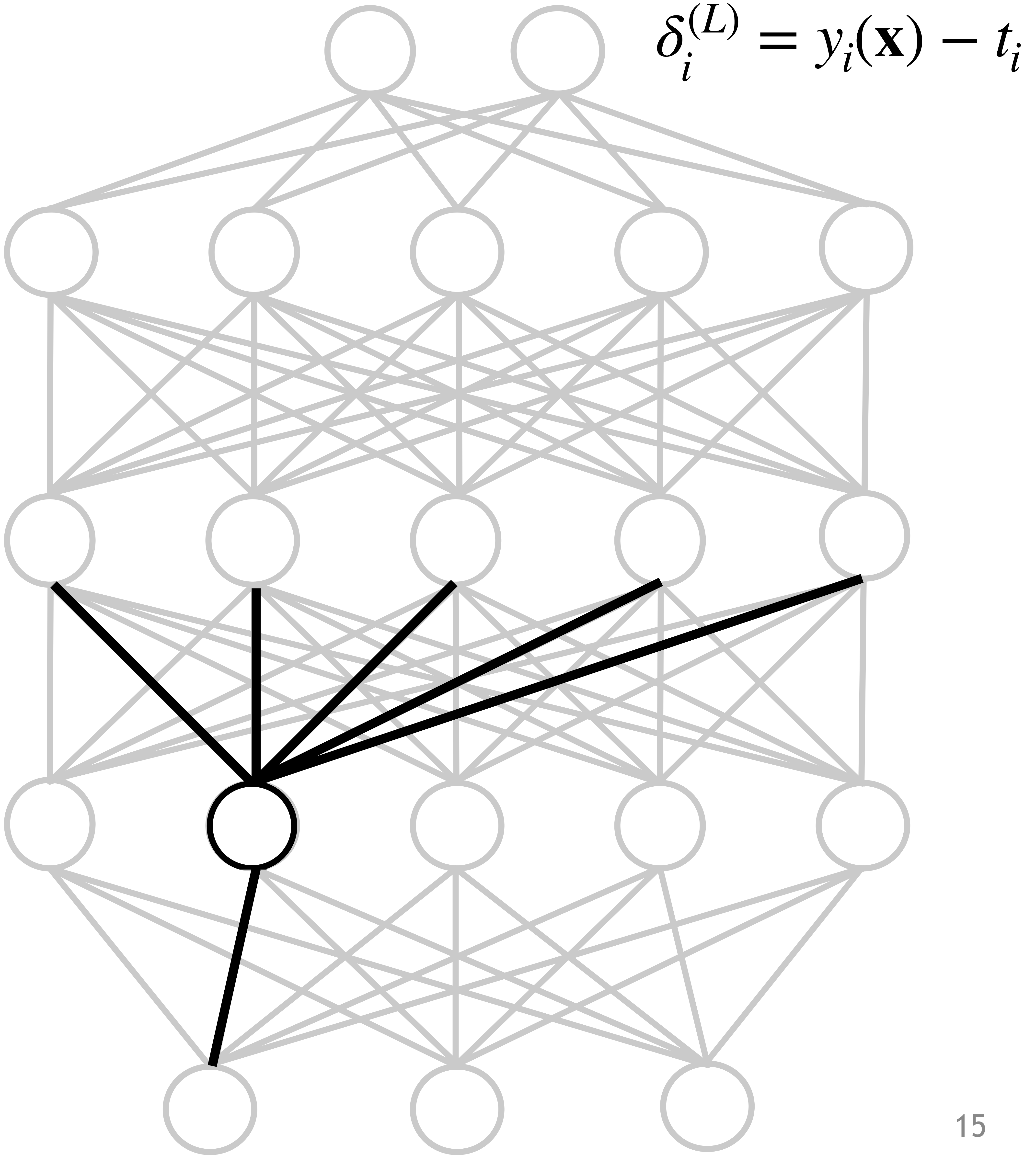
for

- Linear outputs with sum squared error.
- LogSig outputs with cross-entropy error.
- Softmax outputs with cross-entropy error.

Example: Linear outputs with sum squared error

$$E = \frac{1}{2}\sum_k (y_k - t_k)^2 \qquad y_k = a_k^{(L)}$$

$$\frac{\partial E}{\partial a_i^{(L)}} = \frac{1}{2}\frac{\partial}{\partial a_i^{(L)}}\left(a_i^{(L)} - t_i\right)^2 = \left(a_i^{(L)} - t_i\right) = y_i - t_i$$



Backpropagation: Derivation

The δ's for hidden neurons:

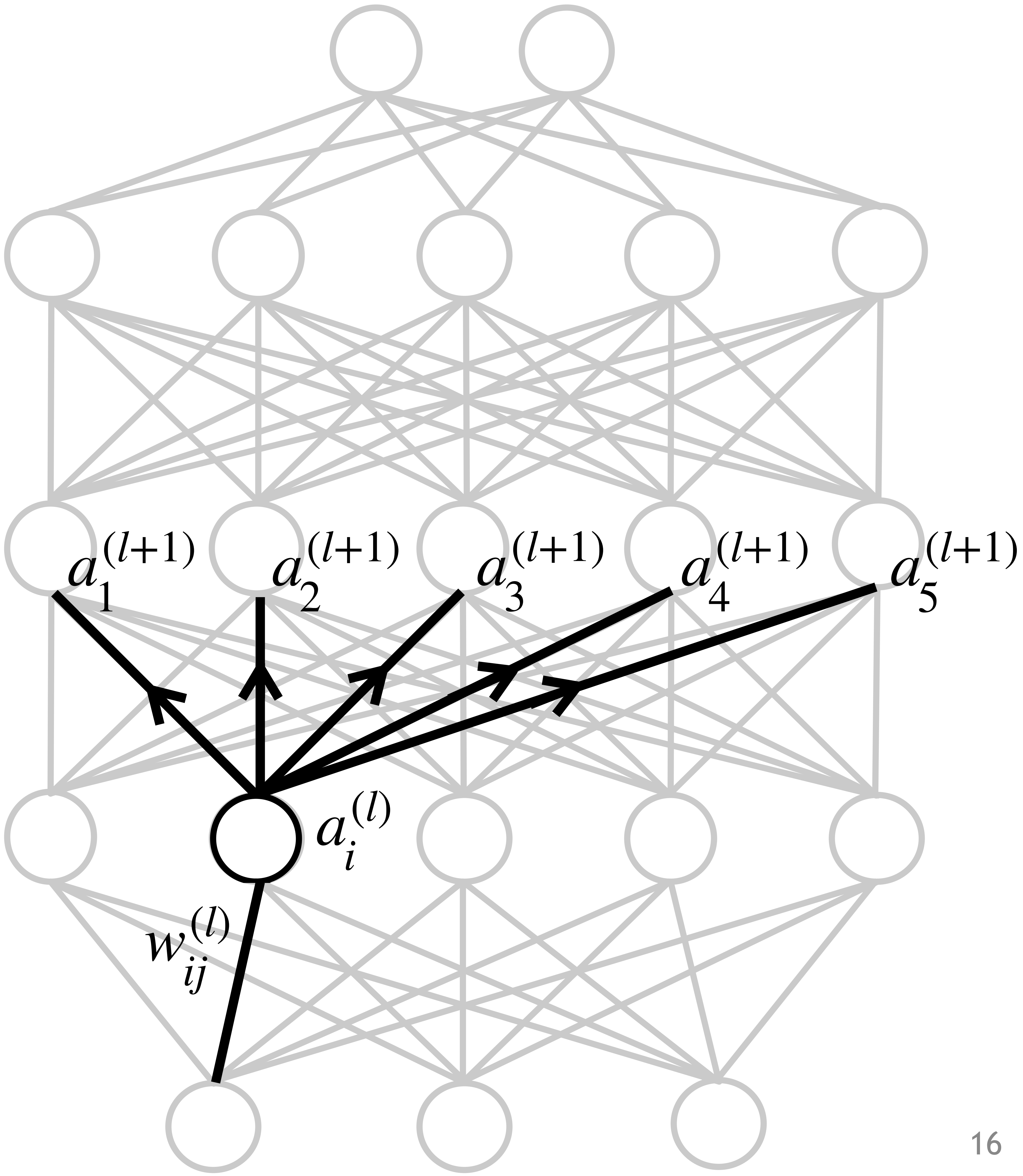
We can use the Chain Rule:

$$\frac{\partial E}{\partial a_i^{(l)}} = \sum_j \frac{\partial E}{\partial a_j^{(l+1)}} \frac{\partial a_j^{(l+1)}}{\partial a_i^{(l)}}$$

since the Error is some function where:

$$E = f(a_1^{(l+1)}, a_2^{(l+1)}, \ldots, a_{N_{l+1}}^{(l+1)})$$

with $\quad a_j^{(l+1)} = f_j(a_i^{(l)})$



Backpropagation: Derivation

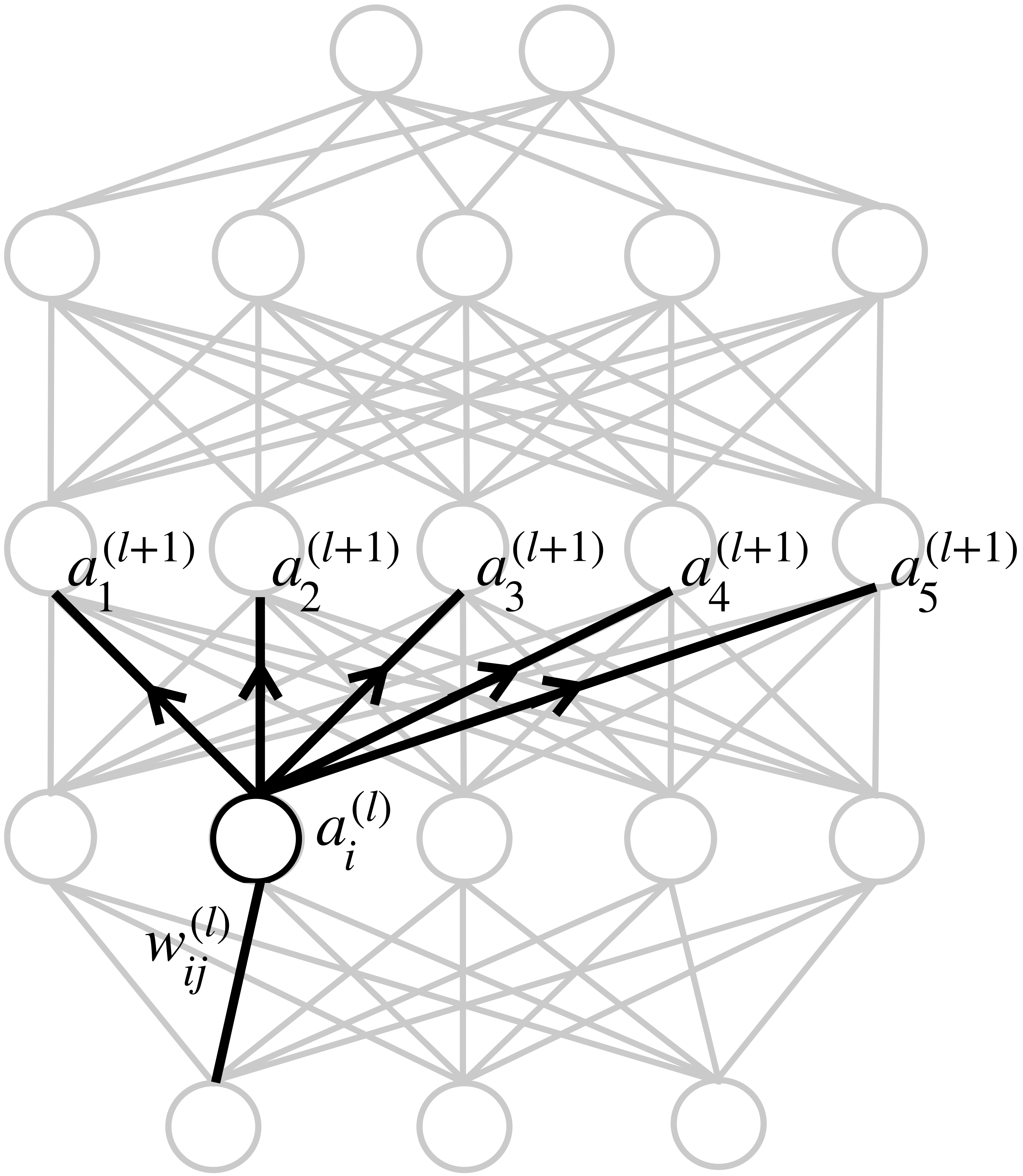
The δ's for hidden neurons:

We use the Chain Rule:

$$\frac{\partial E}{\partial a_i^{(l)}} = \sum_j \frac{\partial E}{\partial a_j^{(l+1)}} \frac{\partial a_j^{(l+1)}}{\partial a_i^{(l)}}$$

$$\frac{\partial E}{\partial a_i^{(l)}} = \sum_j \frac{\partial E}{\partial a_j^{(l+1)}} \frac{\partial a_j^{(l+1)}}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial a_i^{(l)}}$$

$$\delta_i^{(l)} = \sum_j \delta_j^{(l+1)} \cdot w_{ji}^{(l+1)} \cdot h^{(l)\prime}\left(a_i^{(l)}\right)$$



Recall: $a_i^{(l)} = \sum_k w_{ik}^{(l)} z_k^{(l-1)}$ and $z_i^{(l)} = h^{(l)}\left(a_i^{(l)}\right)$

Summary: Backpropagation Algorithm

The Backprop algorithm for layered networks:

(1) **Forward pass:**

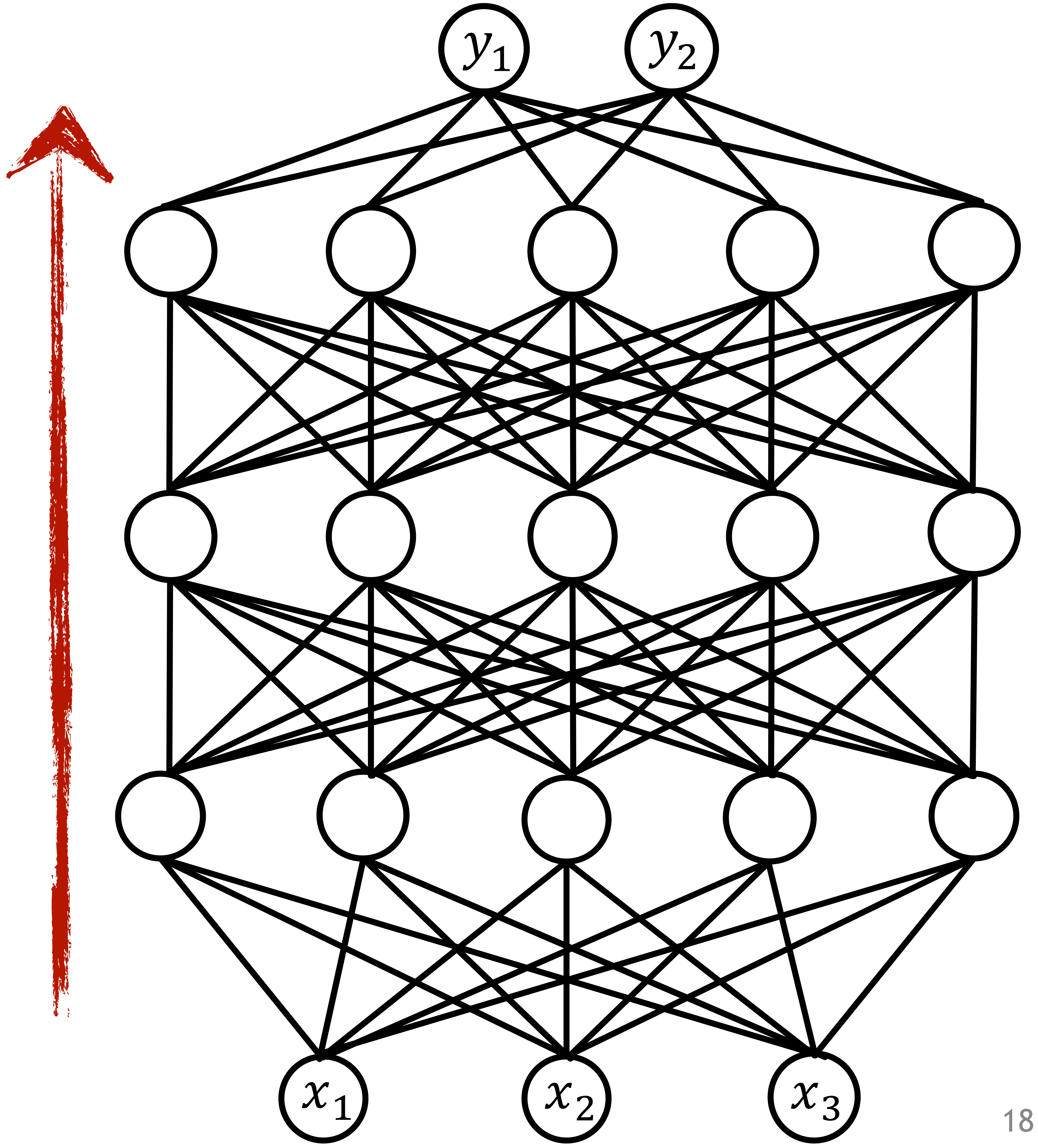
Compute neuron activations and outputs.

$$\mathbf{z}^{(0)} = \mathbf{x}$$

$$\mathbf{a}^{(l)} = W^{(l)}\mathbf{z}^{(l-1)}$$

$$\mathbf{z}^{(l)} = h^{(l)}\left(\mathbf{a}^{(l)}\right)$$

$$\mathbf{y} = \mathbf{z}^{(L)}$$



Summary: Backpropagation Algorithm

The Backprop algorithm for layered networks:

(1) **Forward pass:**

Compute neuron activations and outputs.

$$\mathbf{z}^{(0)} = \mathbf{x}$$

$$\mathbf{a}^{(l)} = W^{(l)}\mathbf{z}^{(l-1)}$$

$$\mathbf{z}^{(l)} = h^{(l)}\left(\mathbf{a}^{(l)}\right)$$

$$\mathbf{y} = \mathbf{z}^{(L)}$$

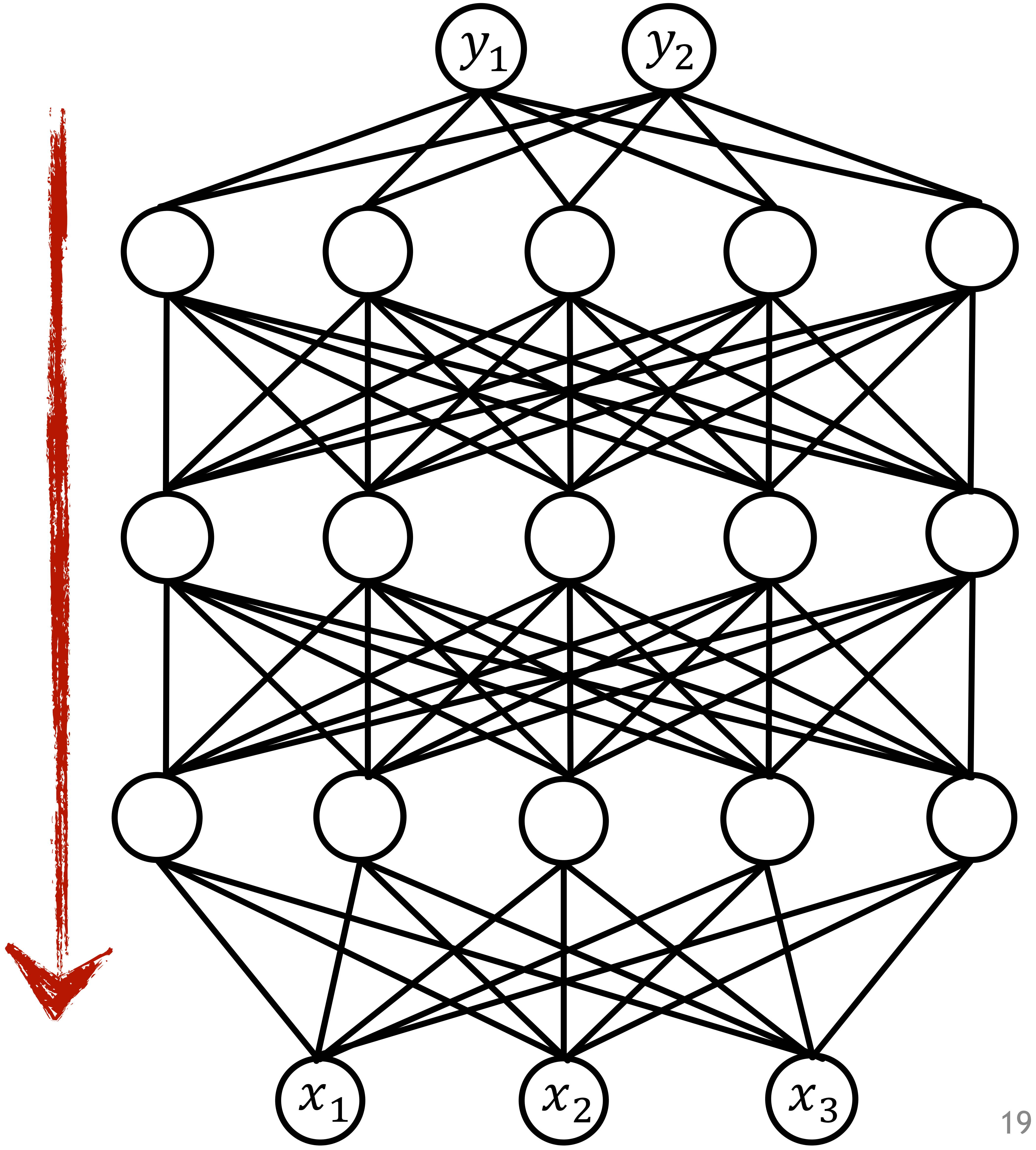
(2) **Backward pass:**

Compute errors and parameter gradients.

$$\delta^{(L)} = \mathbf{y}(\mathbf{x}) - \mathbf{t}$$

$$\delta^{(l)} = h^{(l)\prime}\left(\mathbf{a}^{(l)}\right) \odot \left(W^{(l+1)^T}\delta^{(l+1)}\right)$$

$$\nabla_{W^{(l)}} E = \delta^{(l)}\mathbf{z}^{(l-1)^T}$$



Summary: Backpropagation Algorithm

The Backprop algorithm for layered networks:

(1) **Forward pass:**

Compute neuron activations and outputs.

$$\mathbf{z}^{(0)} = \mathbf{x}$$

$$\mathbf{a}^{(l)} = W^{(l)}\mathbf{z}^{(l-1)}$$

$$\mathbf{z}^{(l)} = h^{(l)}\left(\mathbf{a}^{(l)}\right)$$

$$\mathbf{y} = \mathbf{z}^{(L)}$$

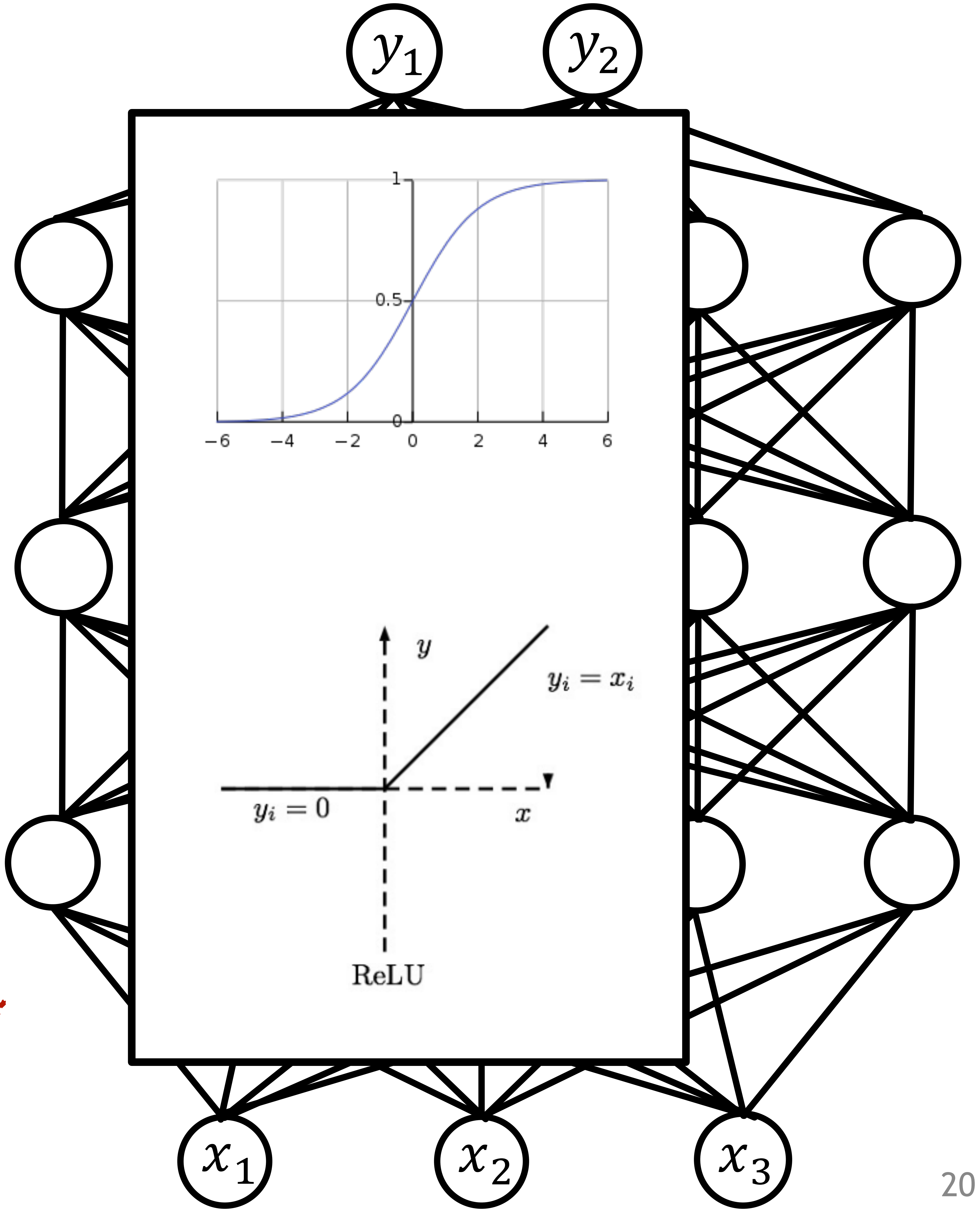
(2) **Backward pass:**

Compute errors and parameter gradients.

$$\delta^{(L)} = \mathbf{y}(\mathbf{x}) - \mathbf{t}$$

$$\delta^{(l)} = h^{(l)'}\left(\mathbf{a}^{(l)}\right) \odot \left(W^{(l+1)^T}\delta^{(l+1)}\right)$$

$$\nabla_{W^{(l)}}E = \delta^{(l)}\mathbf{z}^{(l-1)^T}$$



Summary: Backpropagation Algorithm

The Backprop algorithm for general feed-forward networks:

(1) **Forward pass:** Compute neuron activations and outputs.

$$z_i = x_i \quad \text{for} \;\; i = 1,\ldots,D$$

$$a_i = \sum_{j \in pre(i)} w_{ij}\, z_j$$

$$z_i = h_i(a_i)$$

$$y_k = z_{out_k}$$

(2) **Backward pass:**

Compute errors and parameter gradients:

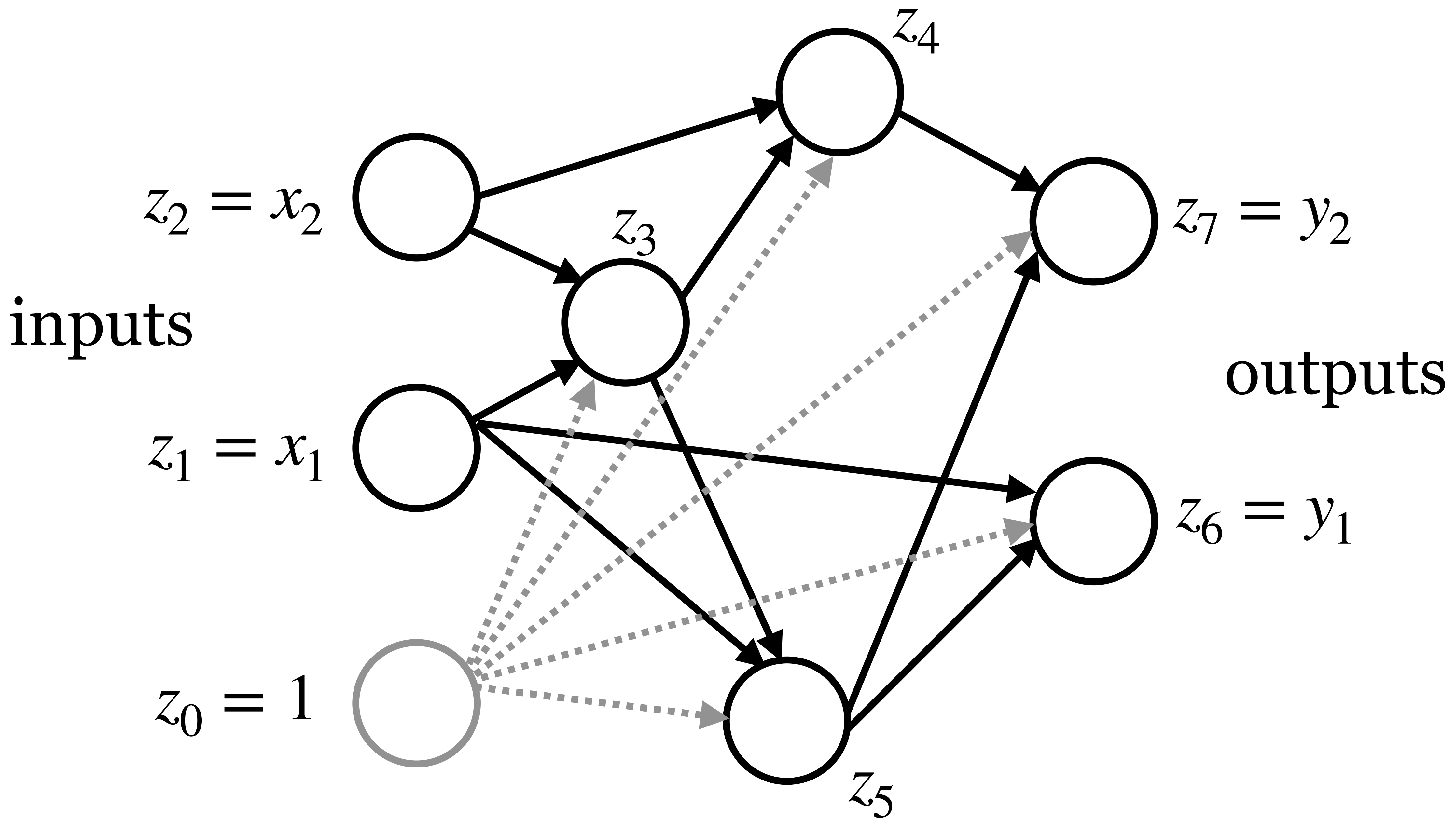
$$\delta_{out_k} = y_k - t_k$$

Backpropagate errors:

$$\delta_i = h_i'(a_i) \sum_{k \in post(i)} w_{ki}\, \delta_k$$

Evaluate derivatives:

$$\frac{\partial E}{\partial w_{ij}} = \delta_i\, z_j$$



Today

- [ ] Neural Network Training
    - [x] Error (Loss) Functions
    - [x] Gradient Descent
    - [x] Backpropagation
    - [ ] Symbolic Derivatives

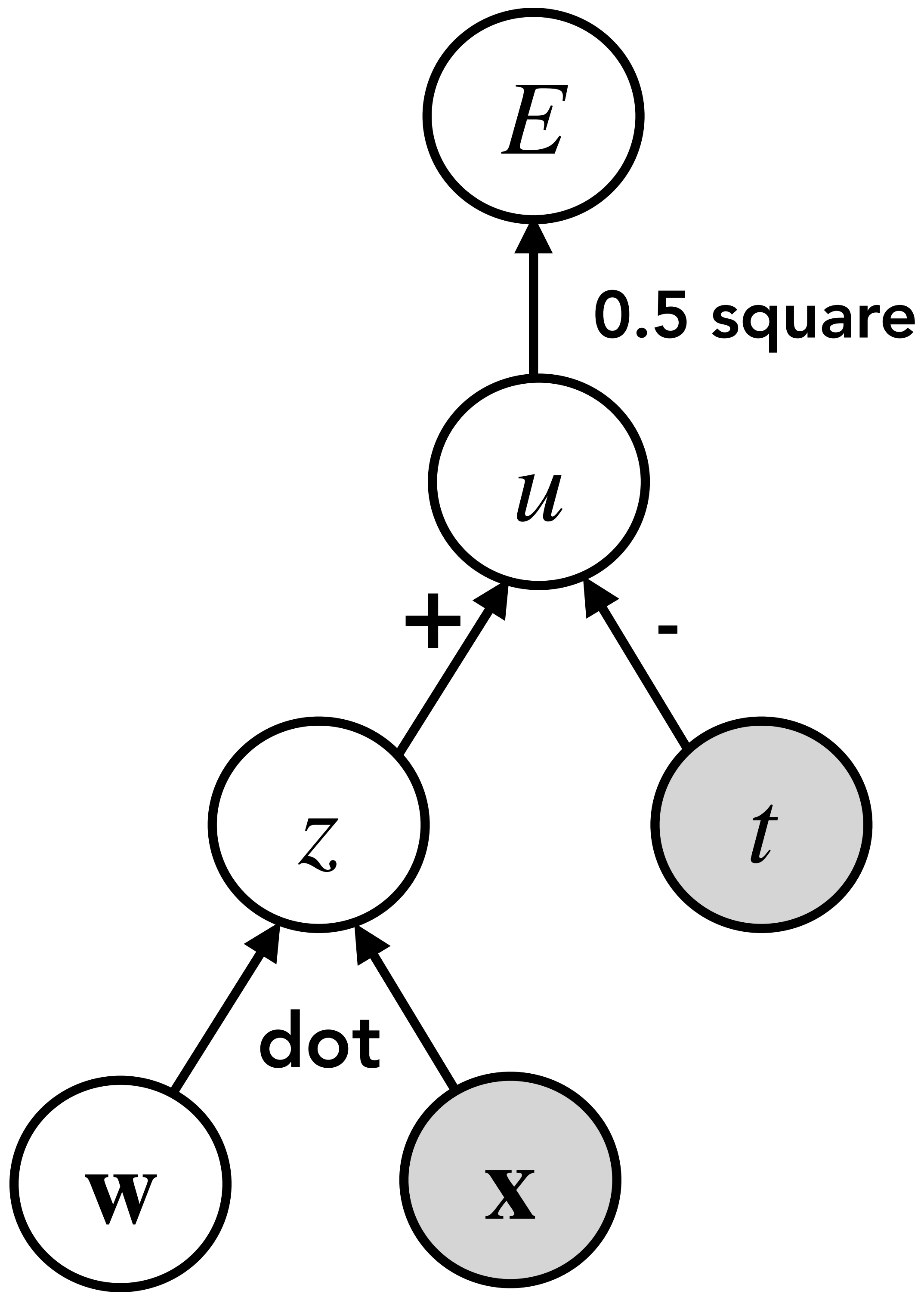
Computational Graphs

Direct implementation of backpropagation in complex models is tedious and error-prone.

- Modern software provide tools that compute symbolic derivatives (e.g. TensorFlow).
- The principle is based on Computational Graphs and the chain rule.

A **computational graph** represents a computation as a graph where:

- Each node represents a variable,
- An operation is a simple function of one or more variables,
- If a variable y is computed by applying an operation to a variable x, then we draw a directed edge from x to y,
- We sometimes annotate the output node with the name of the operation.



Computational graph for linear regression including error computation.

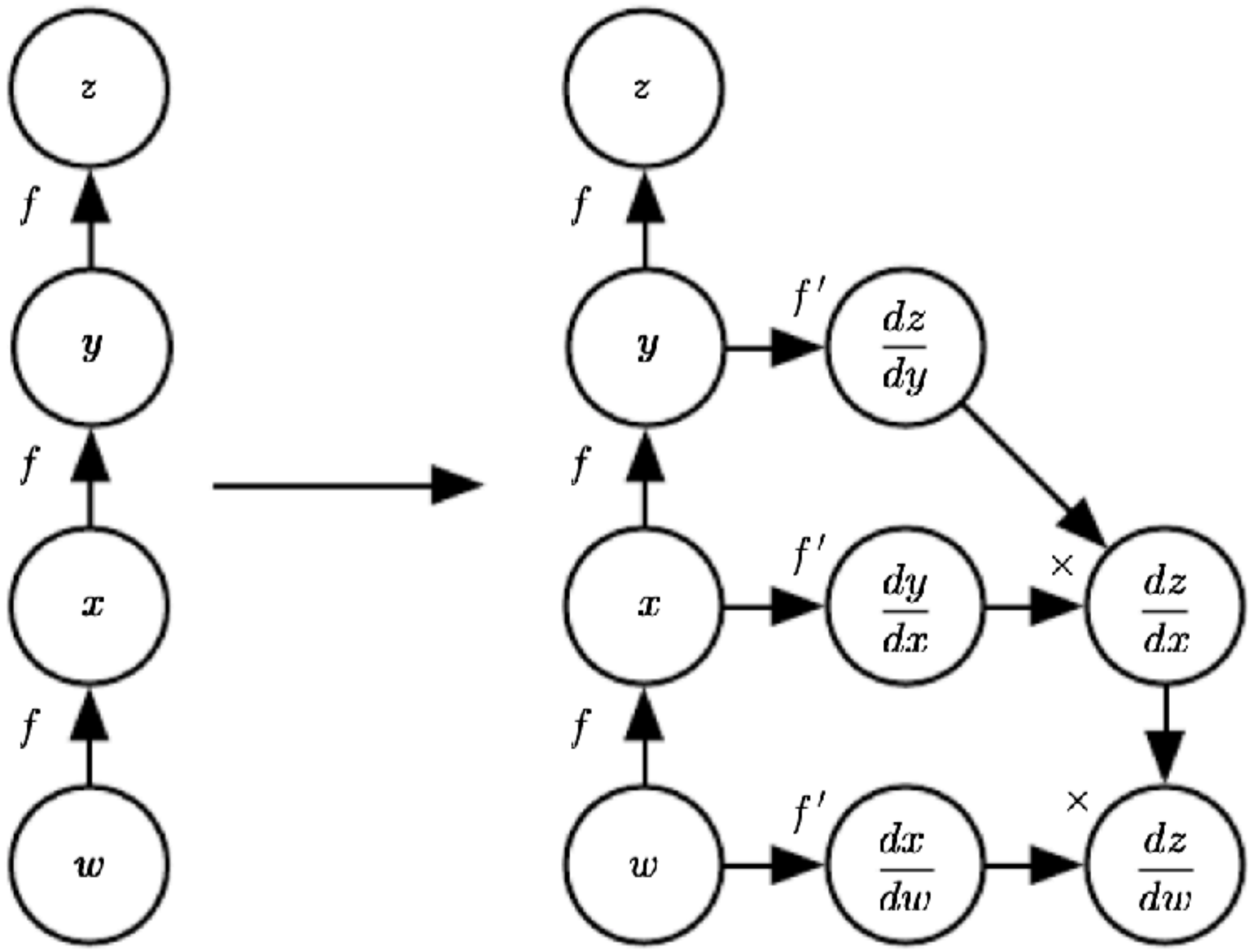
Symbol-to-Symbol Derivatives

The algorithm that computes the derivatives gets as **input:**

- The computational graph,
- The scalar variable z for which the derivative should be computed,
- The set of variables w.r.t. which the derivatives should be computed.

and returns as **output:**

- The appended computational graph that also computes the derivatives.



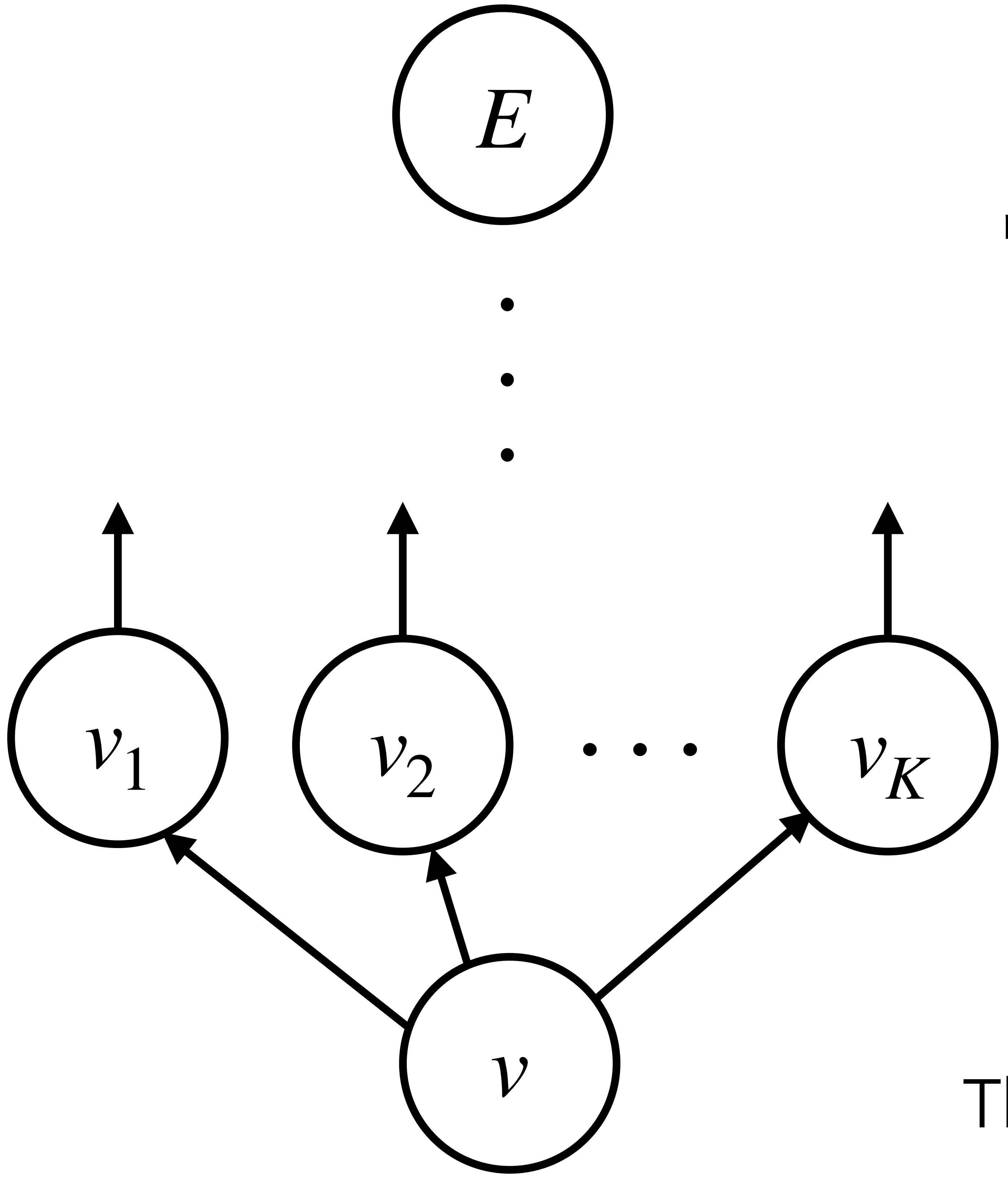
Algorithm to construct computational graph

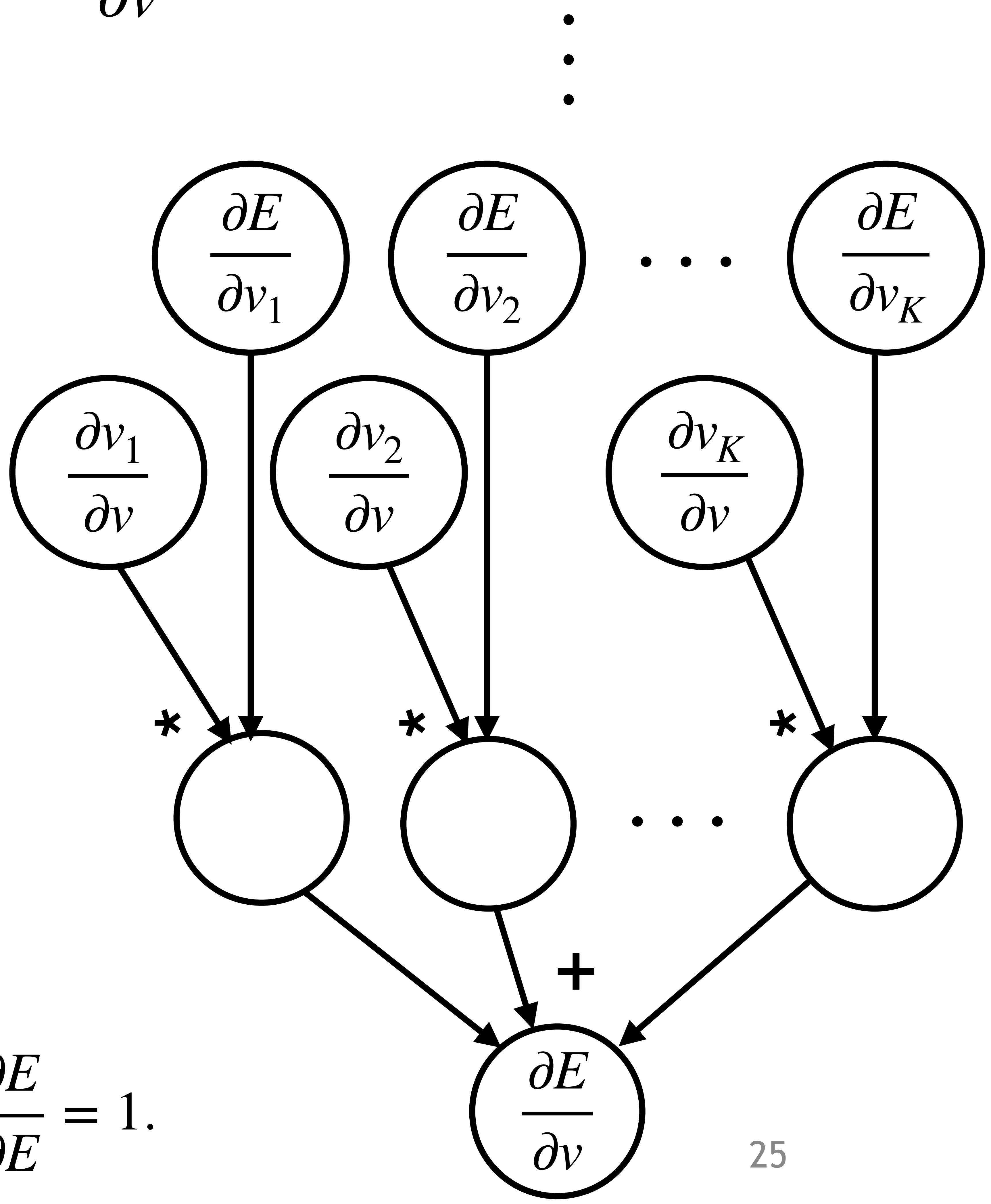
A graph containing a variable v with several consumers v_k. Compute $\dfrac{\partial E}{\partial v}$.

The graph has to be appended by $\dfrac{\partial E}{\partial v} = \sum_i \dfrac{\partial E}{\partial v_i} \dfrac{\partial v_i}{\partial v}$

For this we insert a new node. The graph for it is constructed recursively.

This can be directly computed.

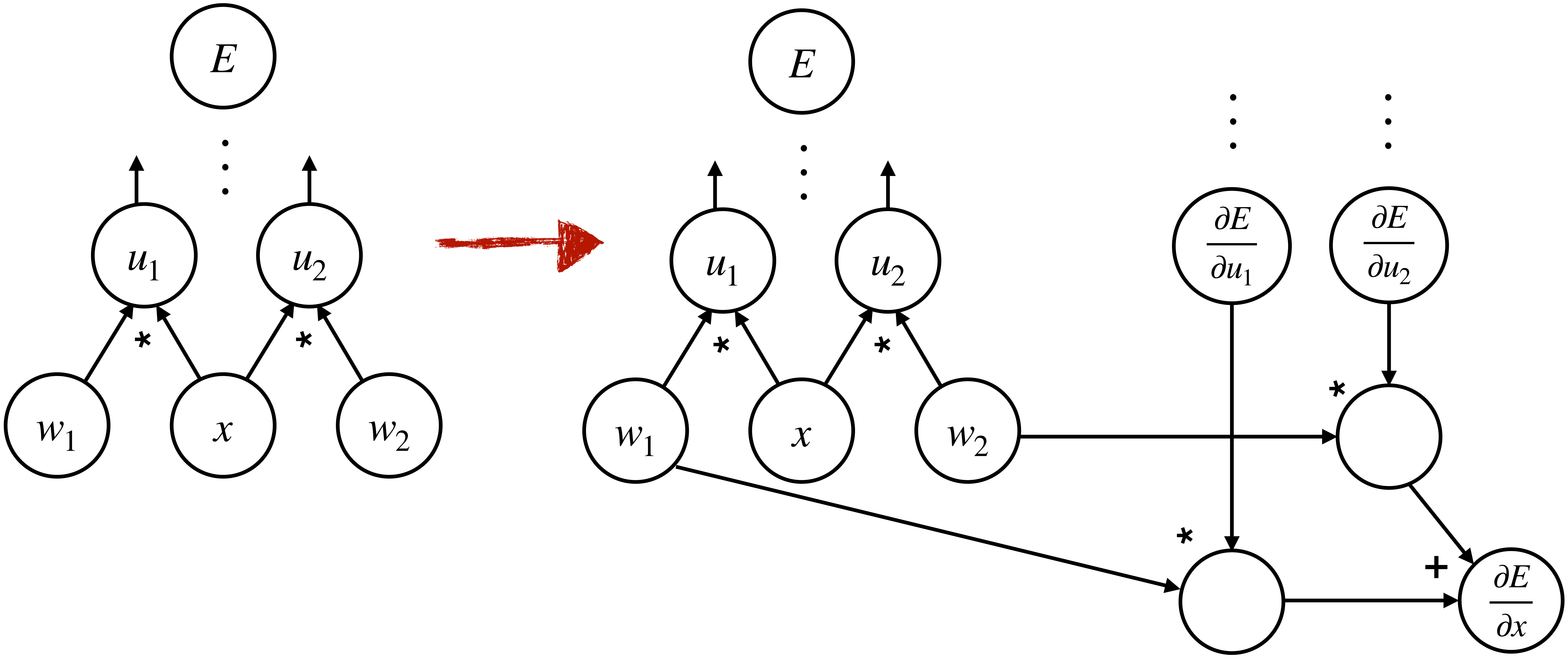




The recursion ends when we arrive at $\dfrac{\partial E}{\partial E} = 1$.

An illustrative example

Compute $\frac{\partial E}{\partial x}$. The graph has to be appended by $\frac{\partial E}{\partial x} = \sum_i \frac{\partial E}{\partial u_i} \frac{\partial u_i}{\partial x}$

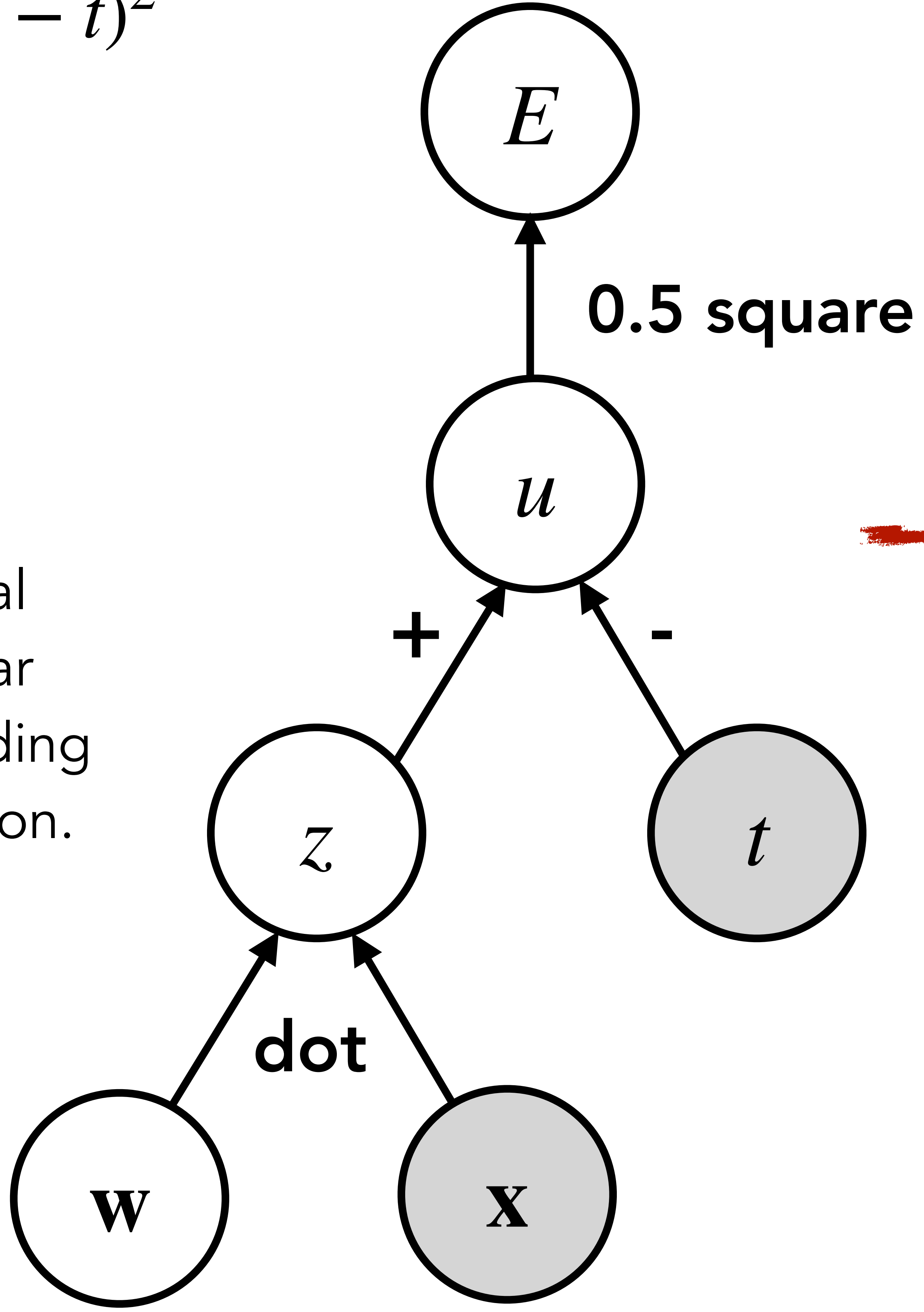


An example for the gradient of a regression model

$$\frac{\partial E}{\partial v} = \sum_i \frac{\partial E}{\partial v_i}\frac{\partial v_i}{\partial v}$$

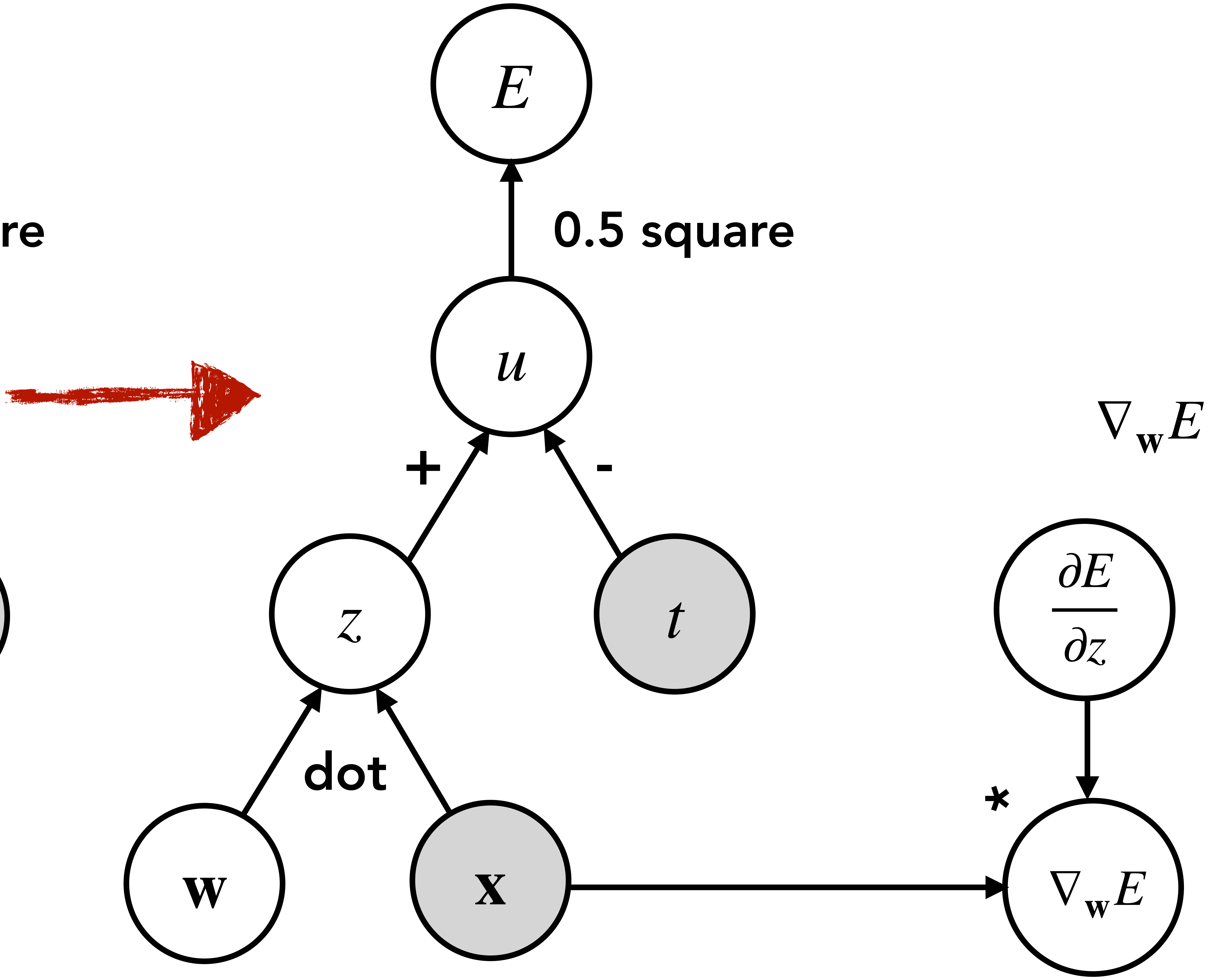
Gradient of a regression model for $\nabla_{\mathbf{w}} E$

$$E = \frac{1}{2}(\mathbf{w}^T\mathbf{x} - t)^2$$



Computational graph for linear regression including error computation.

$$\nabla_{\mathbf{w}} E = \frac{\partial E}{\partial z}\nabla_{\mathbf{w}} z = \frac{\partial E}{\partial z}\mathbf{x}$$



The graph including computation of the derivative.

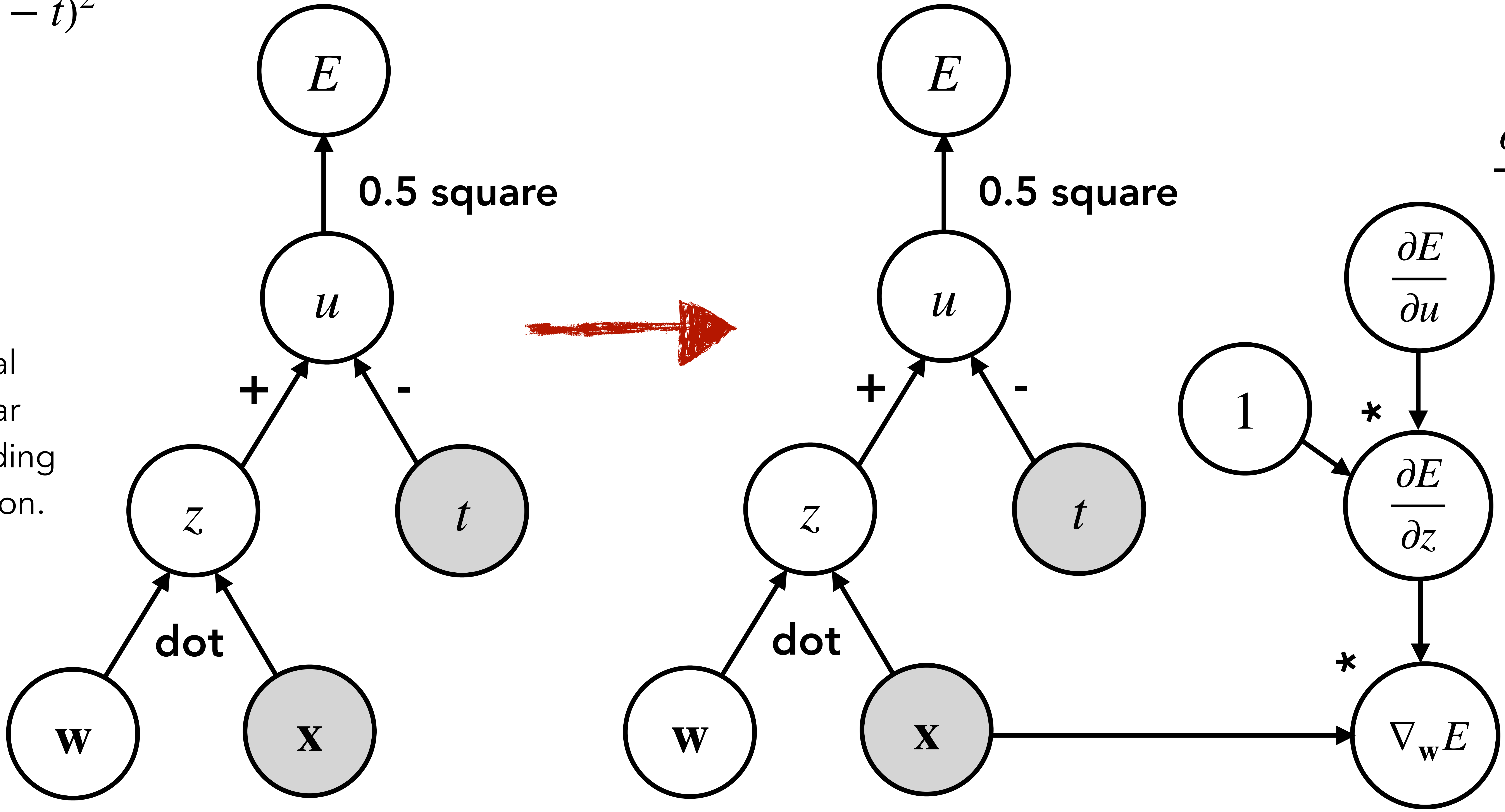
An example for the gradient of a regression model

$$\frac{\partial E}{\partial v} = \sum_i \frac{\partial E}{\partial v_i} \frac{\partial v_i}{\partial v}$$

Gradient of a regression model for $\nabla_{\mathbf{w}} E$

$$E = \frac{1}{2}(\mathbf{w}^T \mathbf{x} - t)^2$$

$$\frac{\partial E}{\partial z} = \frac{\partial E}{\partial u} \frac{\partial u}{\partial z} = \frac{\partial E}{\partial u} 1$$



Computational graph for linear regression including error computation.

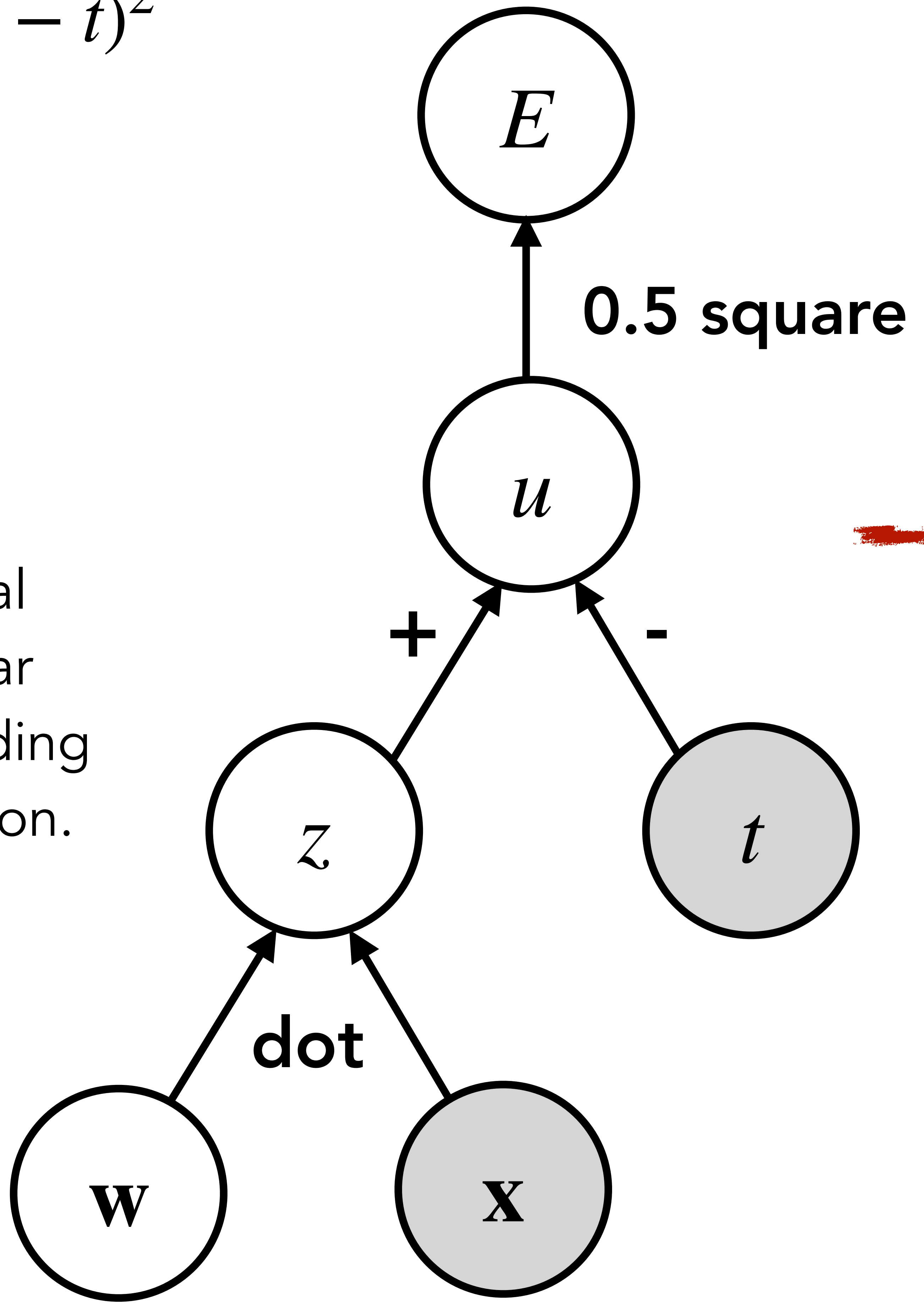
The graph including computation of the derivative.

An example for the gradient of a regression model

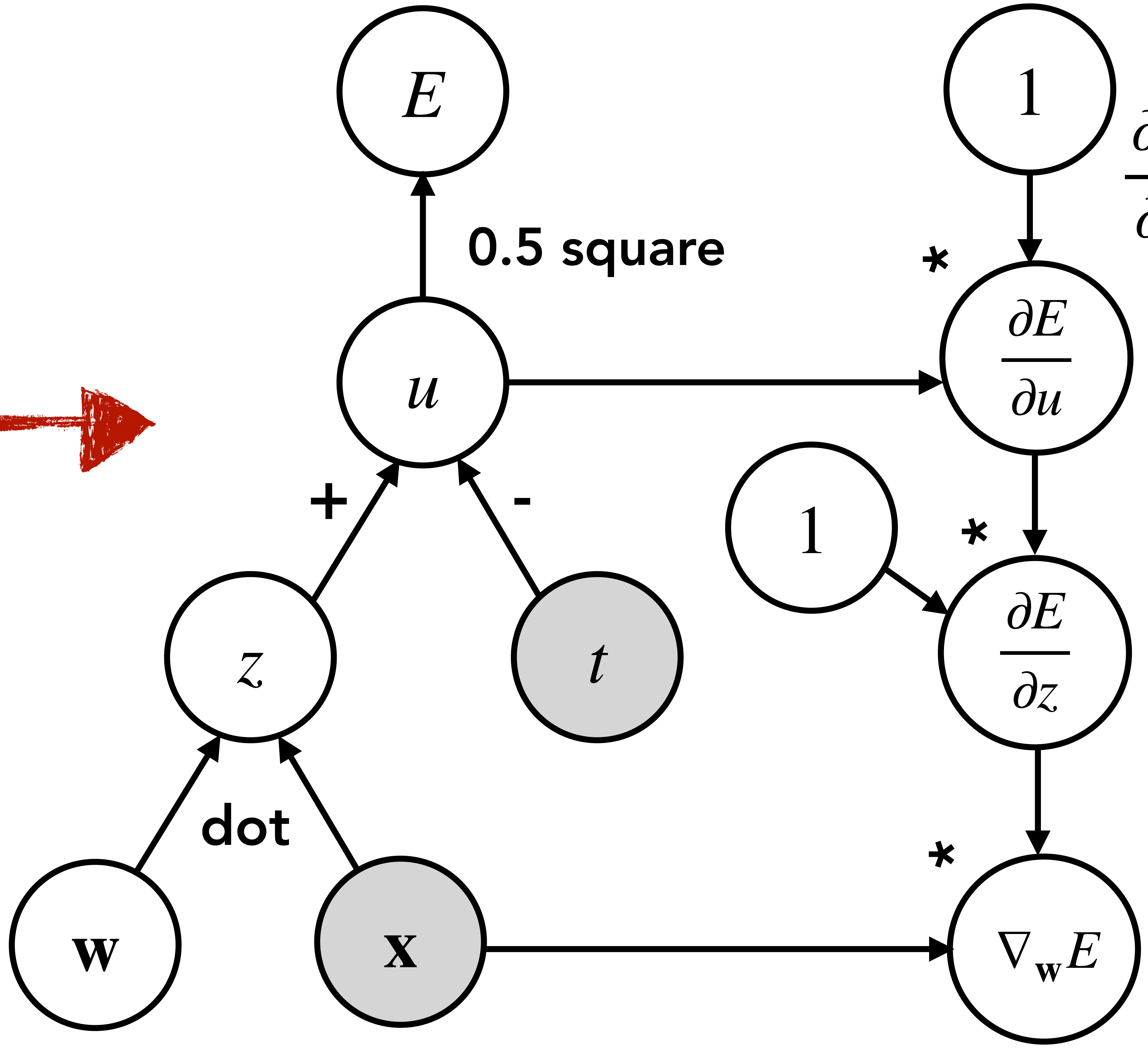
$$\frac{\partial E}{\partial v} = \sum_i \frac{\partial E}{\partial v_i}\frac{\partial v_i}{\partial v}$$

Gradient of a regression model for $\nabla_{\mathbf{w}} E$

$$E = \frac{1}{2}(\mathbf{w}^T\mathbf{x} - t)^2$$



Computational graph for linear regression including error computation.



$$\frac{\partial E}{\partial u} = \frac{\partial E}{\partial E}\frac{\partial E}{\partial u} = 1 * u$$

The graph including computation of the derivative.

Today

- ☑ Neural Network Training
  - ☑ Error (Loss) Functions
  - ☑ Gradient Descent
  - ☑ Backpropagation
  - ☑ Symbolic Derivatives

Questions?