# Neural Networks: Backpropagation

Spring 2018

# Review

- Class before break:
  - Neural Network Architecture
  - Training a Neural Network
- Assignments (Canvas):
  - Lab assignment due yesterday
  - Project pre-proposal due next week
- Questions?

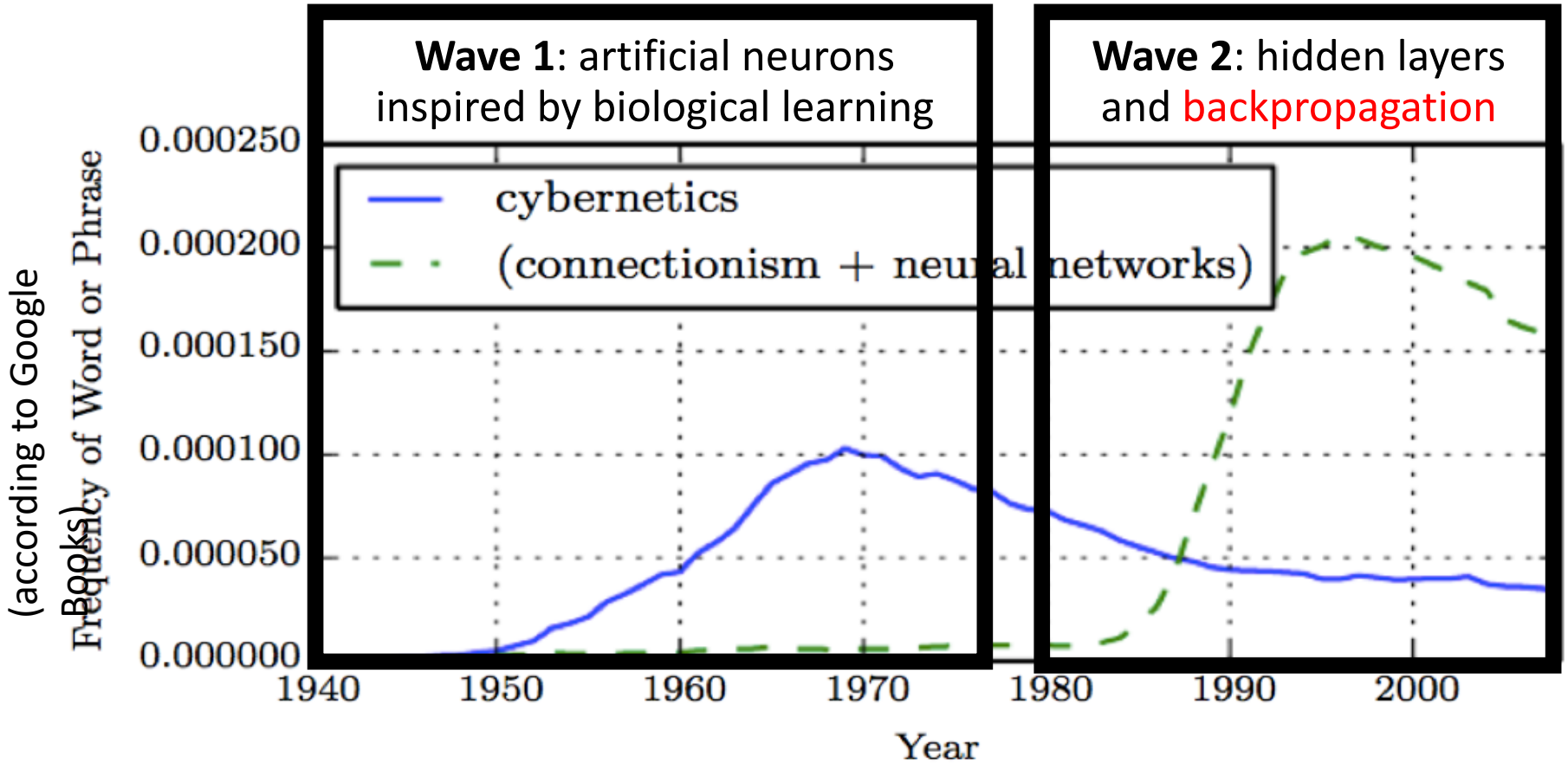
# Today’s Topics

- Neural Network: Gradient Calculation with Backpropagation
- Neural Network: Weight Update Methods
- Neural Network: Training
- Lab

# Today’s Topics

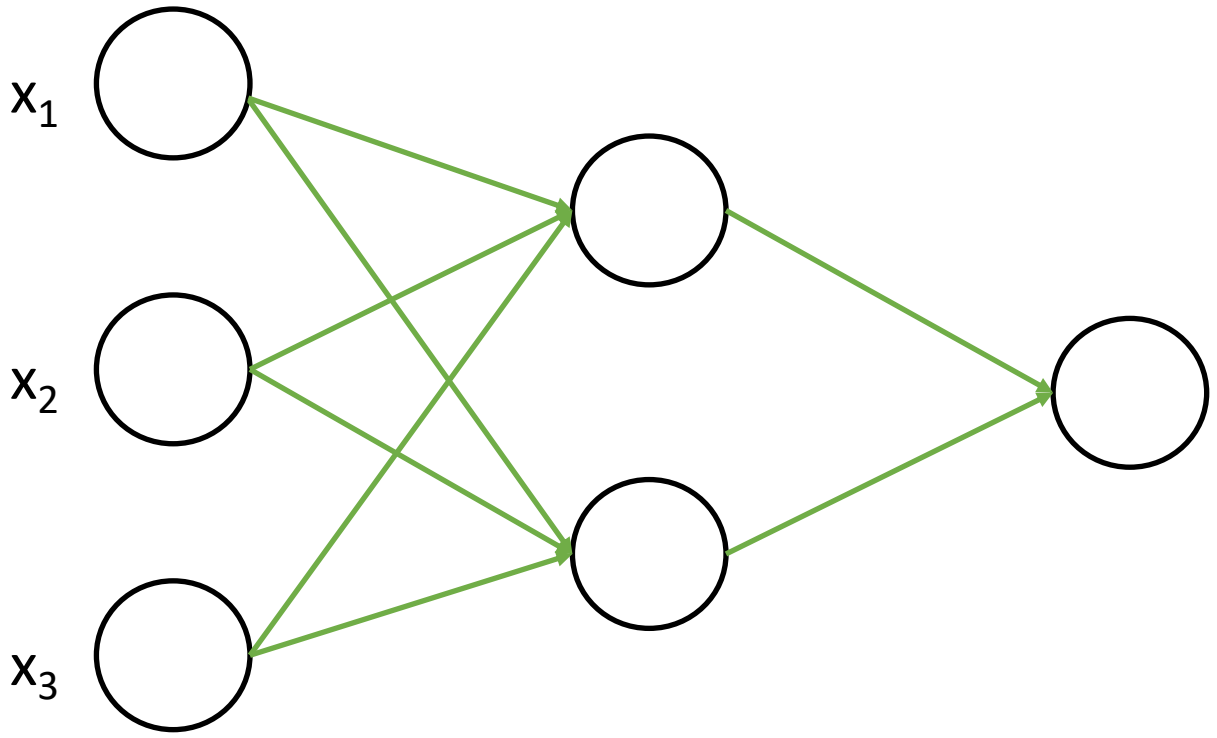
- Neural Network: Gradient Calculation with Backpropagation
- Neural Network: Weight Update Methods
- Neural Network: Training
- Lab

# Recap: History of Neural Networks




Ian Goodfellow, Yoshua Bengio, and Aaron Courville; Deep Learning, 2016.

# Recap: Neural Network Architecture




How many weights are in this model?

- (3x2) +(2x1) = 8

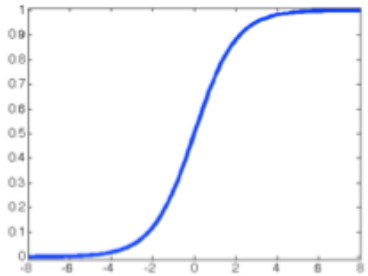
How many model parameters are there?

- 8+3 = 11

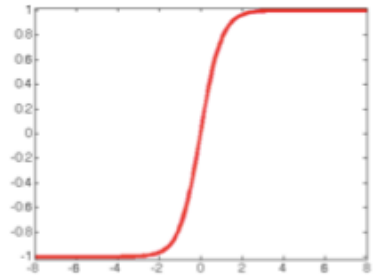
How to create a non-linear model?

- Use activation function at each unit

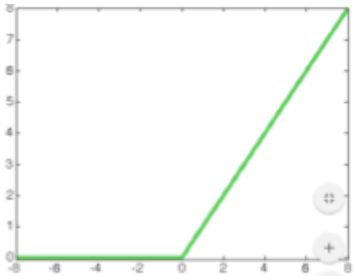
Sigmoid



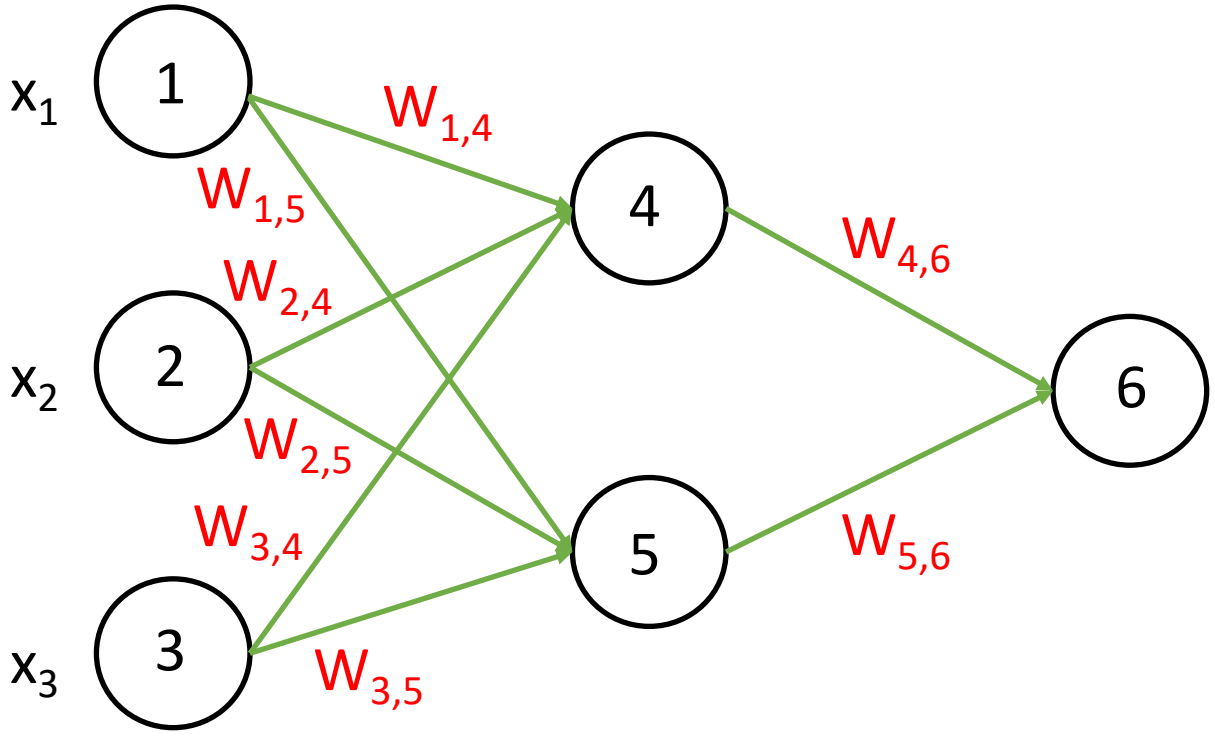
Tanh



ReLU



# Recap: Train Model Using Gradient Descent

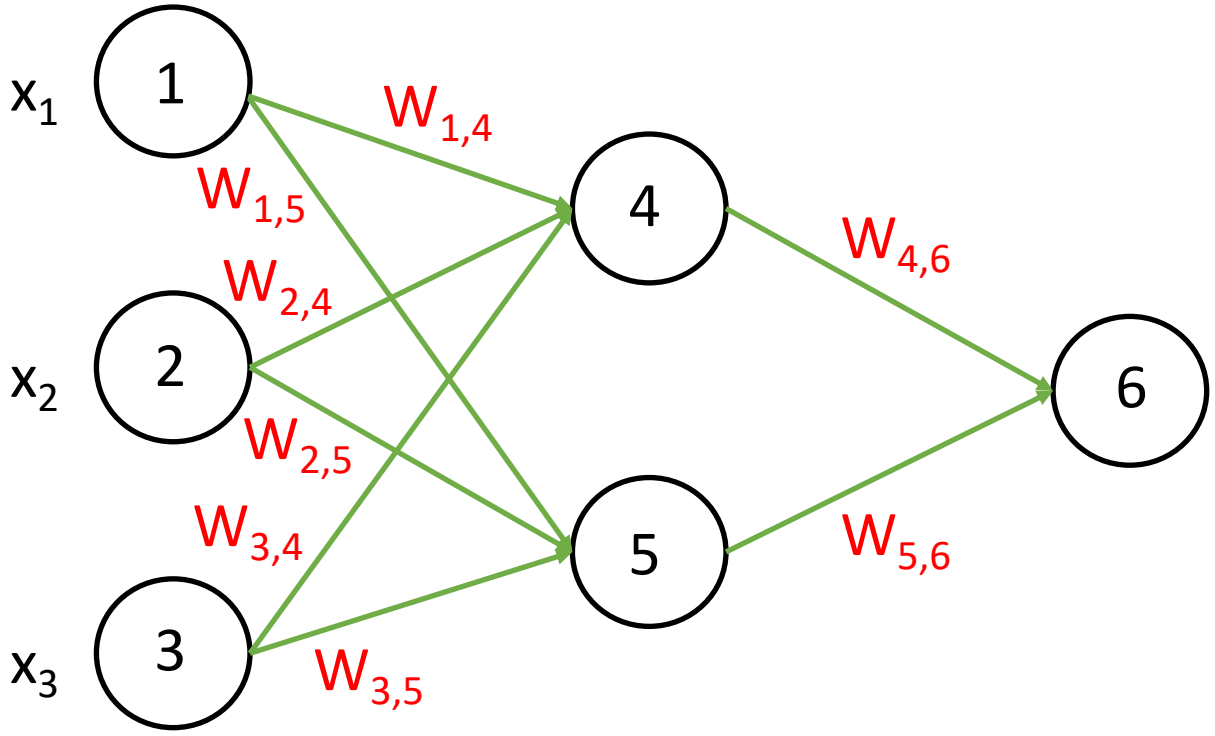



Do until satisfied

- For each training example $d$ in $D$
  1. Compute the gradient $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

Content from: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/mlbook/ch4.pdf

# Challenge: How to Compute Gradient?



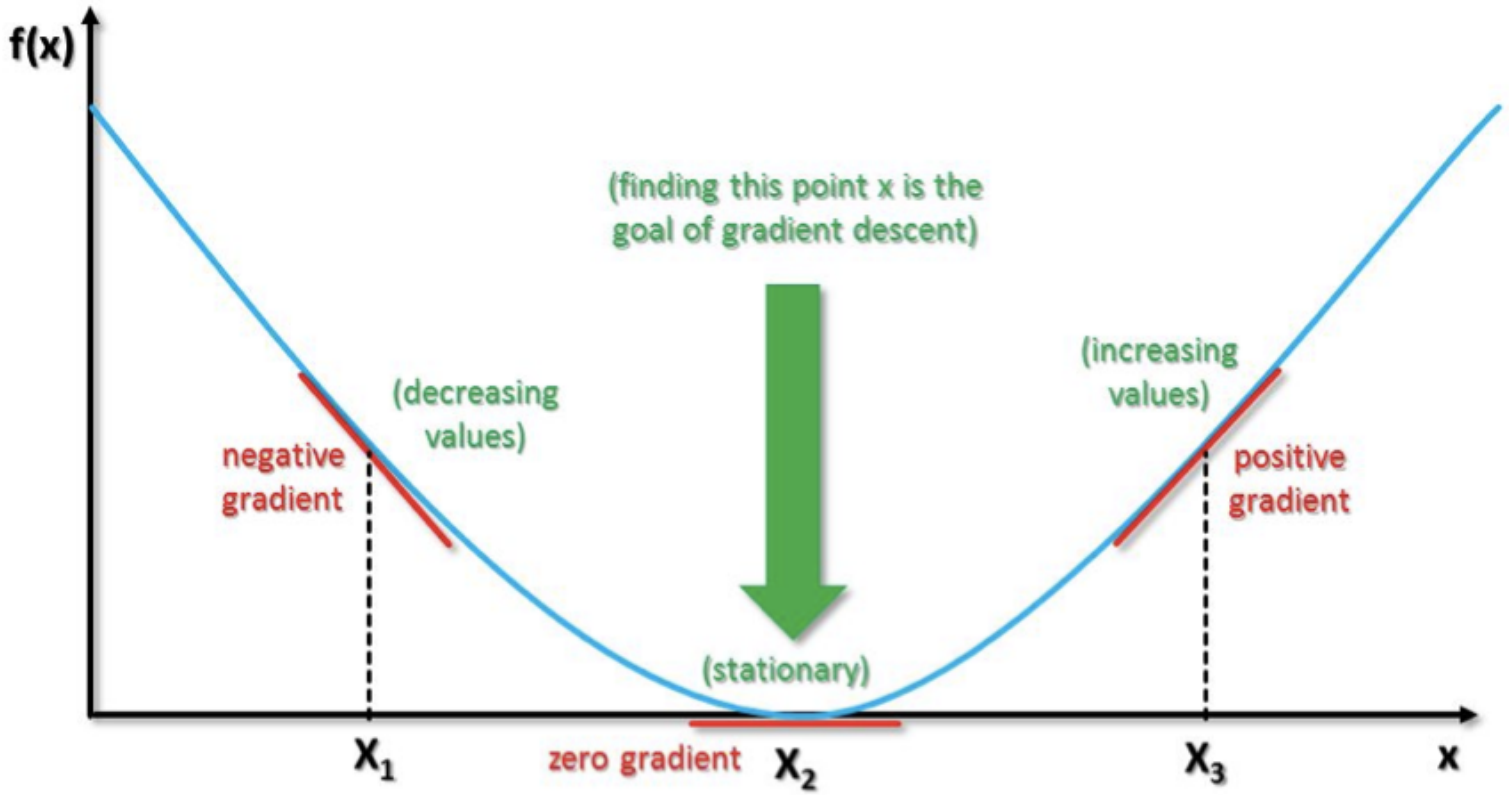

Do until satisfied

- For each training example $d$ in $D$
  1. Compute the gradient $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

Content from: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/mlbook/ch4.pdf

# Gradient Descent in 2D

- e.g.,

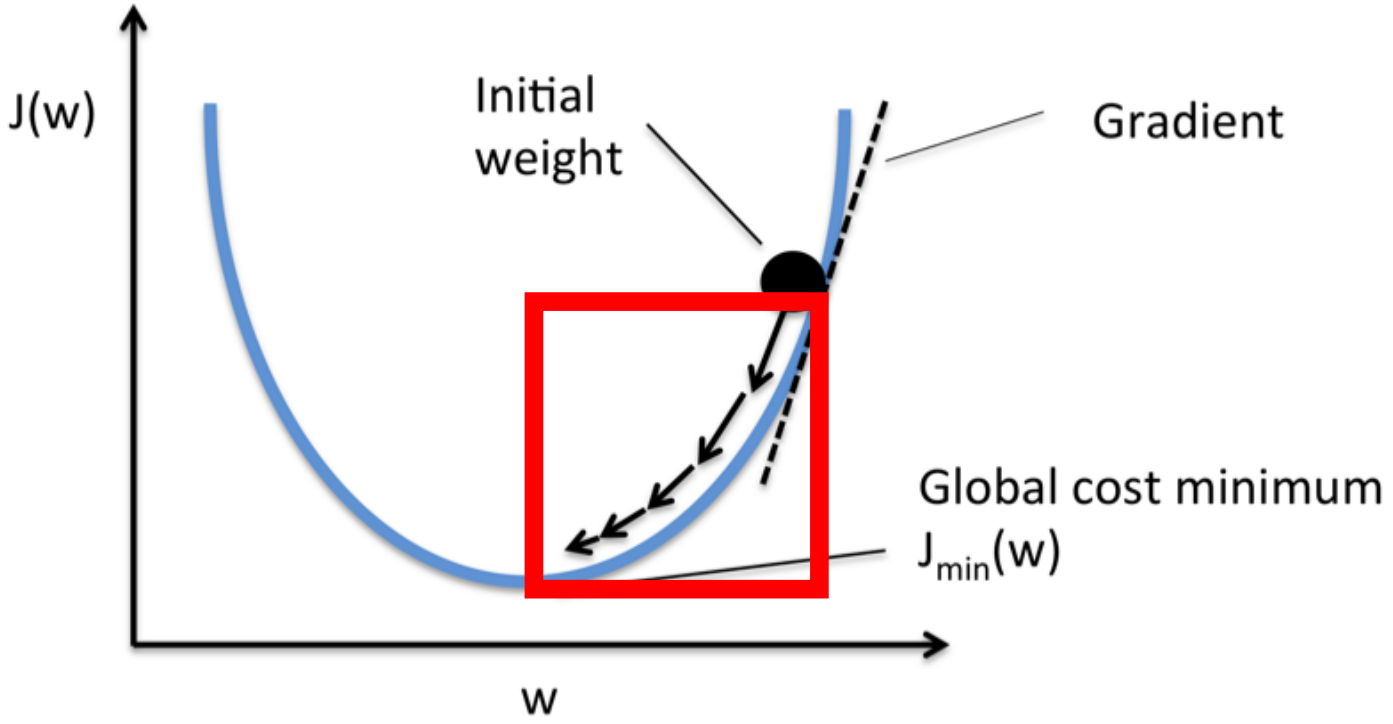



- Goal: find x that minimizes f(x)
  - Where does x minimize f(x)?
- Where is gradient negative?
  - Which direction must you move to reach minimum when gradient is negative?
- Where is gradient positive?
  - Which direction must you move to reach minimum when gradient is positive?

Figure from: http://www.big-data.tips/gradient-descent

# Example: Gradient Descent in 2D

- e.g.,




Note: steps get smaller as gradient gets smaller

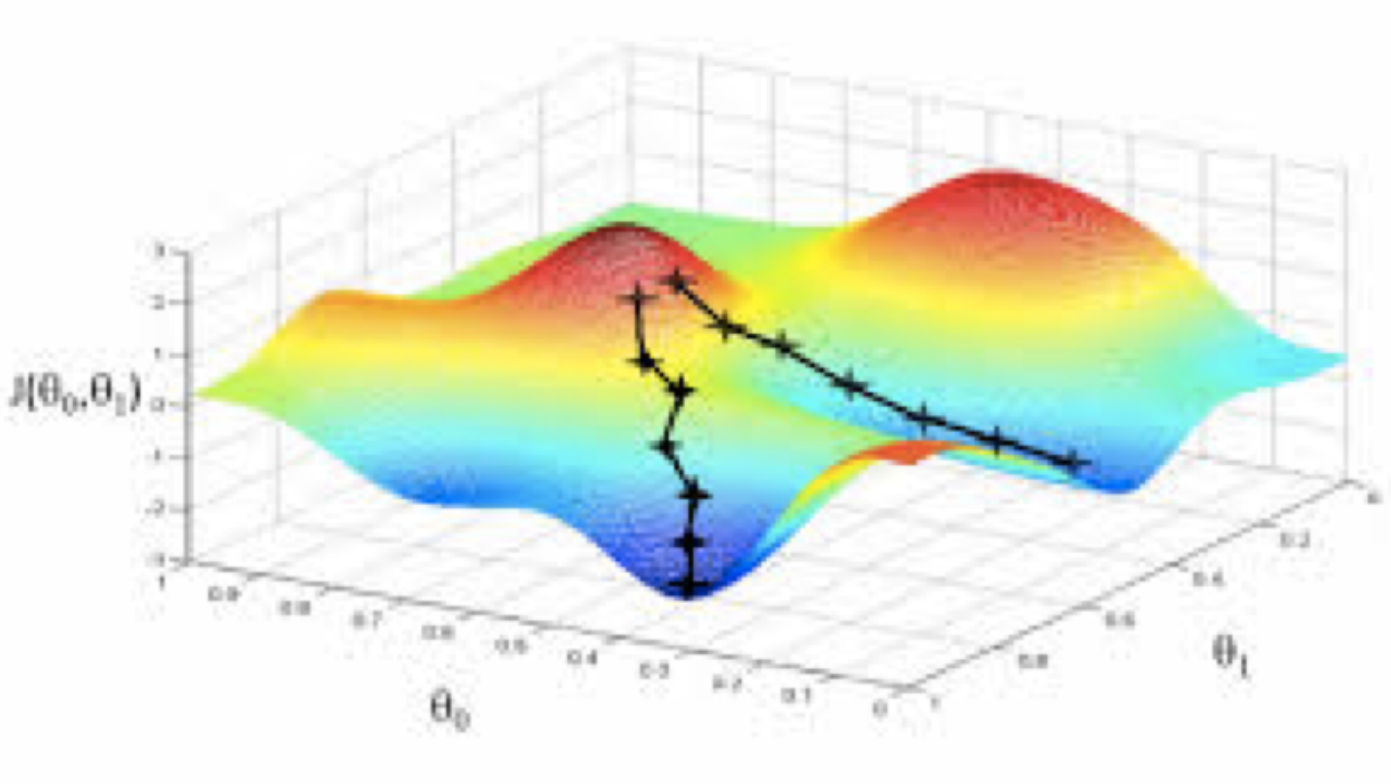
Do until satisfied

- For each training example $d$ in $D$
  1. Compute the gradient $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

Figure from: https://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/

# Gradient Descent in 3D

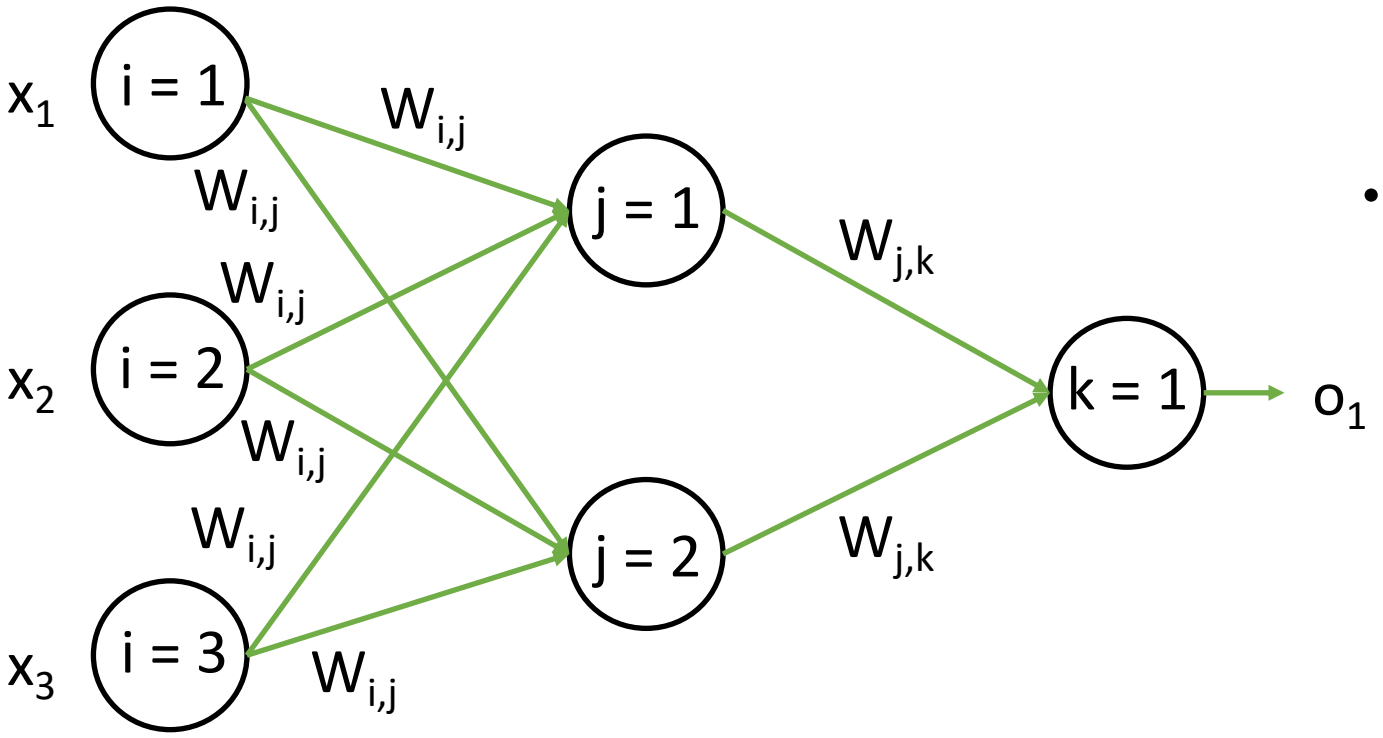
- e.g.,




- Goal: find parameters that minimize J
  - Where do parameters minimize J?
- Analogy: a blindfolded hiker who wishes to reach the bottom of a mountain range

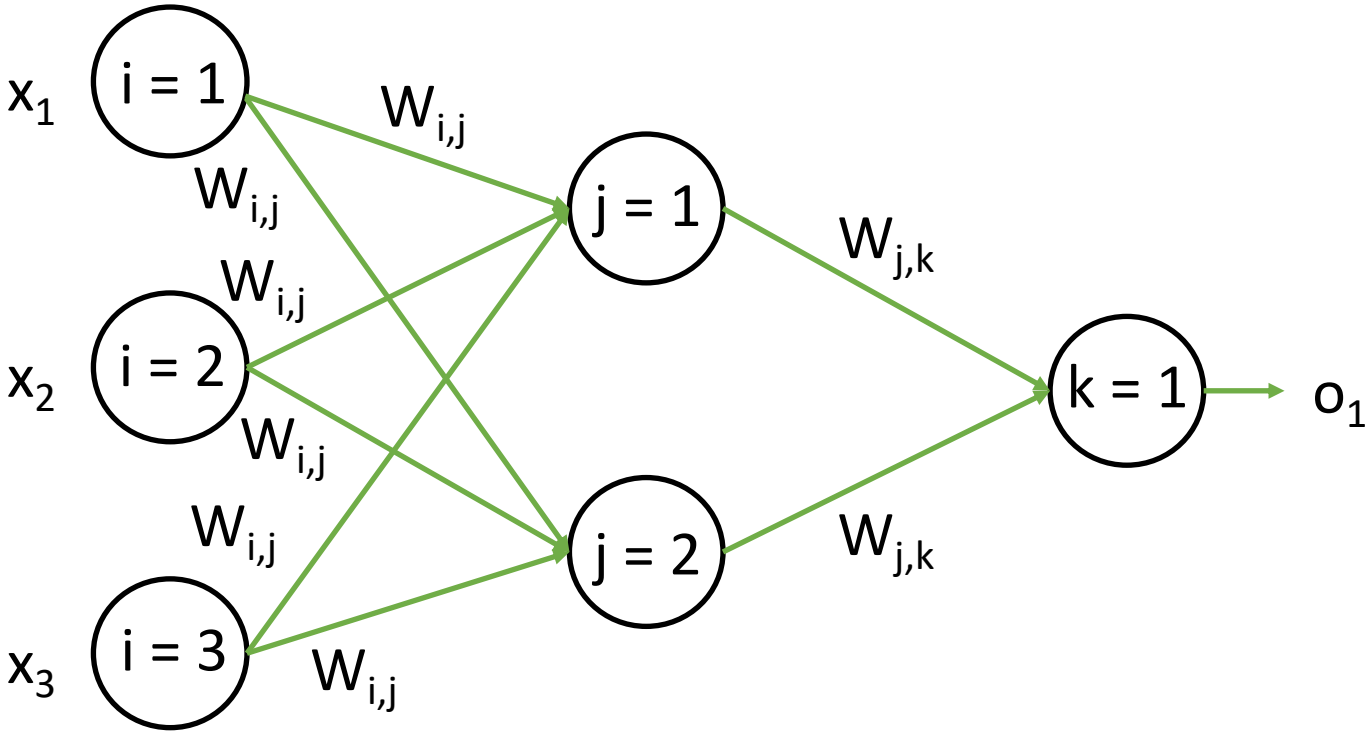
Figure from: https://medium.com/ai-society/hello-gradient-descent-ef74434bdfa5

# Gradient Descent



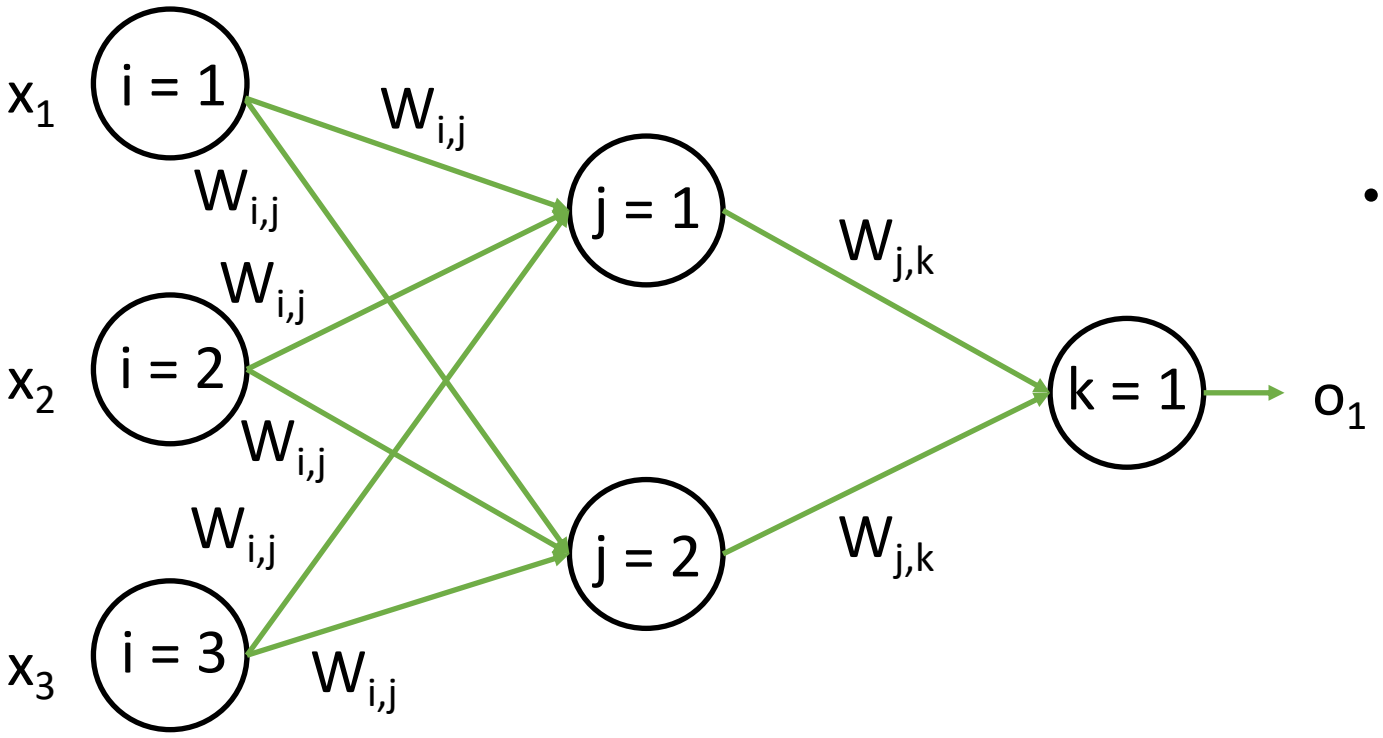

- Goal: find parameters that minimize function
  - What function do we try to minimize for neural networks?
  - How do we optimize the parameters to minimize the neural network function?

# Gradient Descent: Loss Function



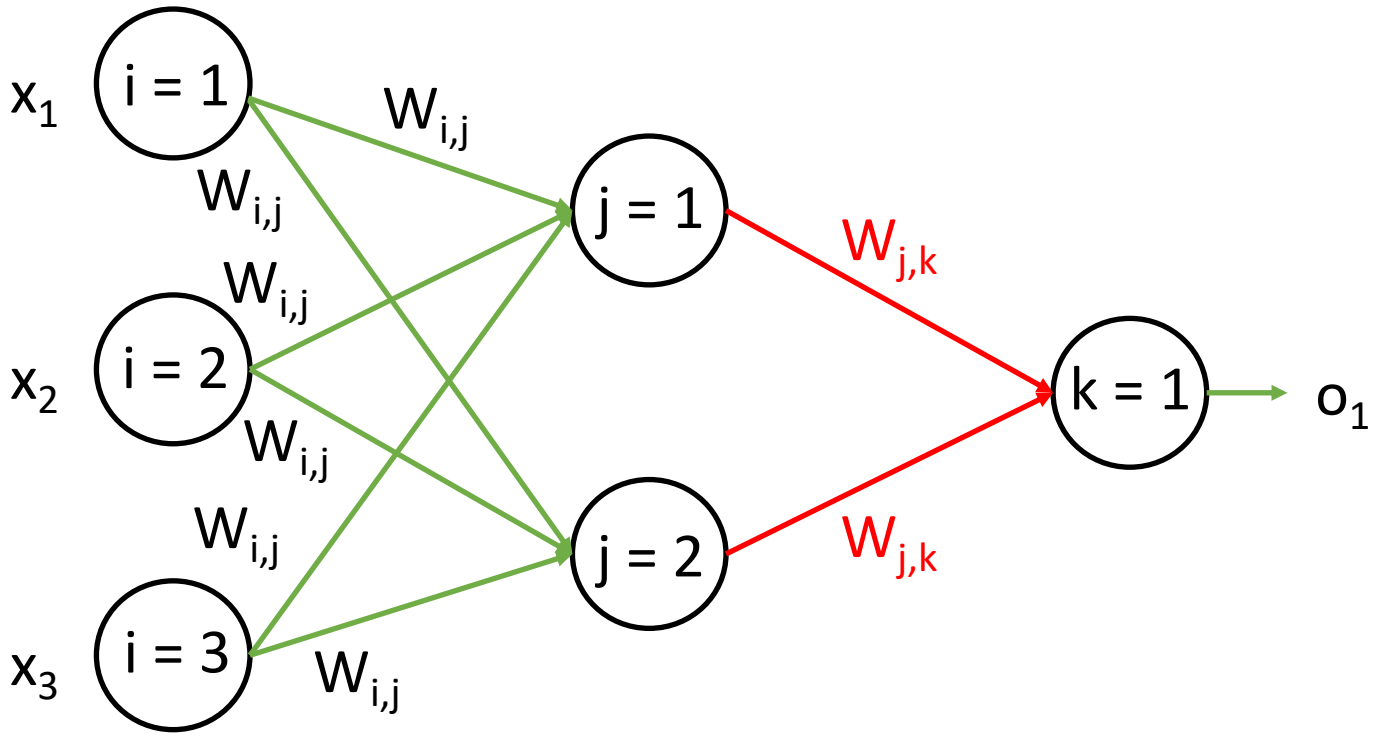

- (Predicted – Actual)?
  - No. Sensitive to sign.
- |Predicted – Actual|?
  - Poor. Discontinuity when prediction is perfect.
- $(\text{Predicted} - \text{Actual})^2$?
  - Yes. Differentiable everywhere.
- Other functions?
  - Sure! Other functions can work.

# Gradient Descent




- Goal: find parameters that minimize function
  - What function do we try to minimize for neural networks?
    - Squared loss: $(t_k - o_k)^2$
  - How do we optimize the parameters to minimize the neural network function?

# Gradient Descent: Gradient Equation




t is a constant value:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} (t_k - o_k)^2$$

# Gradient Descent: Gradient Equation

t is a constant value:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}(t_k - o_k)^2$$

Using the following chain rule :

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial o_k} * \frac{\partial o_k}{\partial w_{jk}}$$

$$\frac{\partial E}{\partial o_k} = -2(t_k - o_k)$$

Sigmoid activation function: $\sigma(x) = \frac{1}{1+e^{-x}}$

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$\frac{d\sigma(x)}{dx} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right)$$

$$\frac{d\sigma(x)}{dx} = (1-\sigma(x))\,\sigma(x)$$

# Gradient Descent: Gradient Equation

t is a constant value:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}(t_k - o_k)^2$$

$$\frac{\partial E}{\partial o_k} = -2(t_k - o_k)$$

Sigmoid activation function: $\sigma(x) = \frac{1}{1+e^{-x}}$

Using the following chain rule :

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial o_k} * \frac{\partial o_k}{\partial w_{jk}}$$

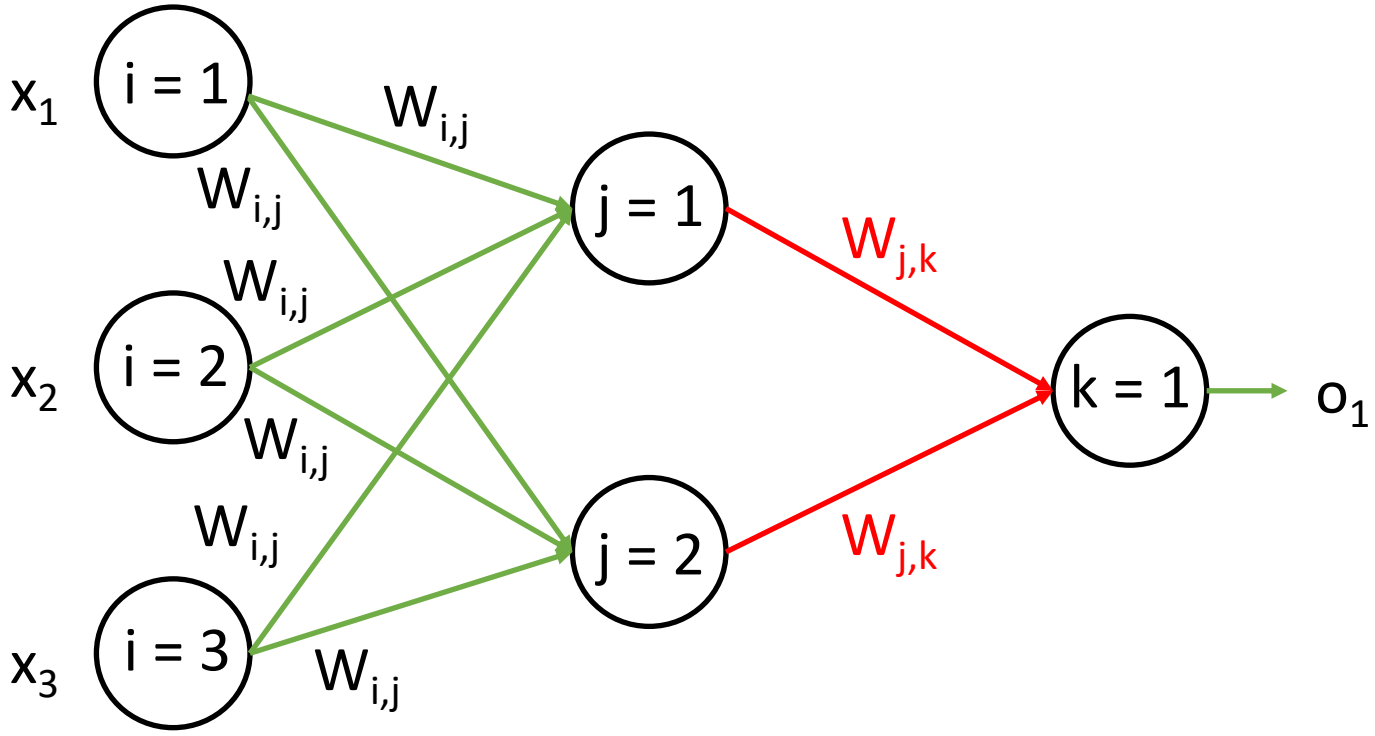
$$\frac{d\sigma(x)}{dx} = (1-\sigma(x))\,\sigma(x)$$

We can rewrite our function as follows:

Efficiency: will remove now and save this task for later

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k) \; sigmoid(\sum_j w_{jk} * o_j) * (1 - sigmoid(\sum_j w_{jk} * o_j)) * o_j$$

# Gradient Descent: Gradient Equation (Output Layer)

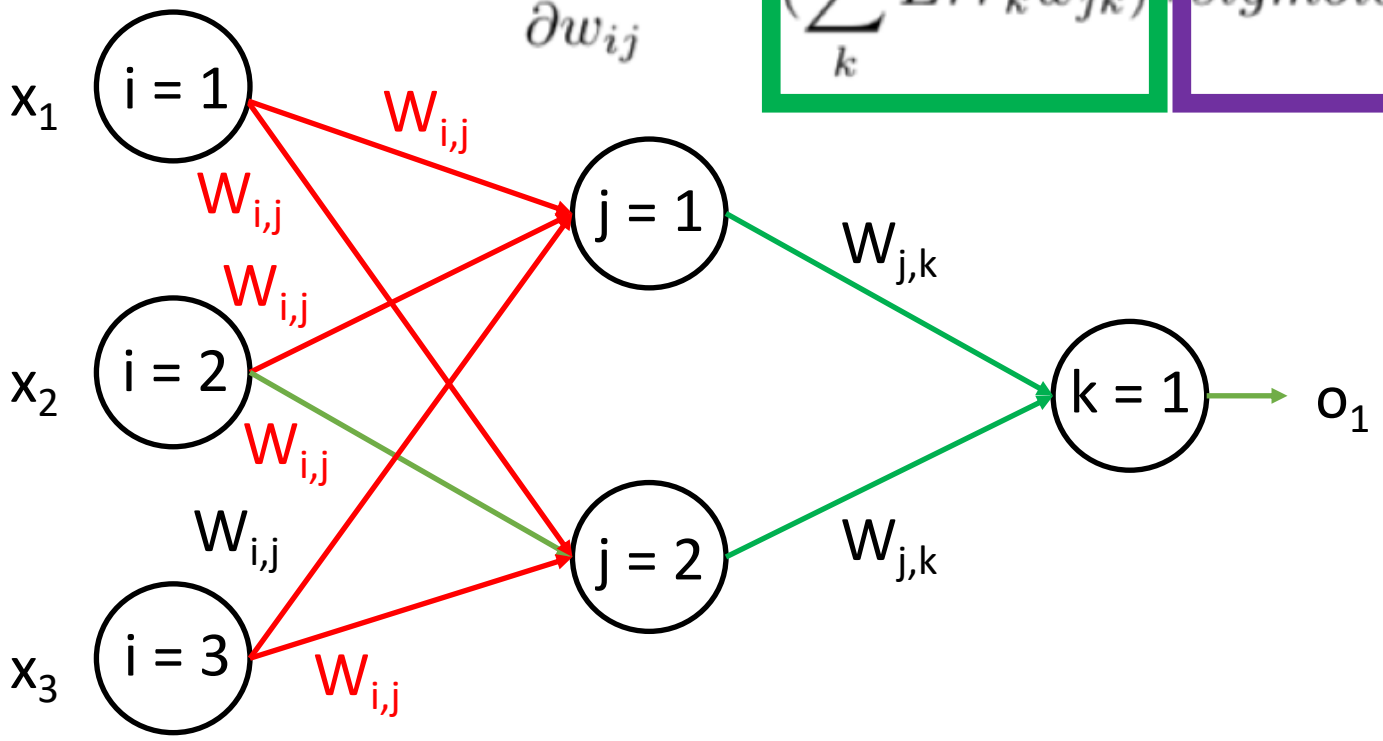



$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}(t_k - o_k)^2$$

$$\frac{\partial E}{\partial w_{jk}} = -(t_k - o_k) * sigmoid(\sum_j w_{jk} * o_j) * (1 - sigmoid(\sum_j w_{jk} * o_j))$$

# Gradient Descent: Gradient Equation (Hidden Layer)

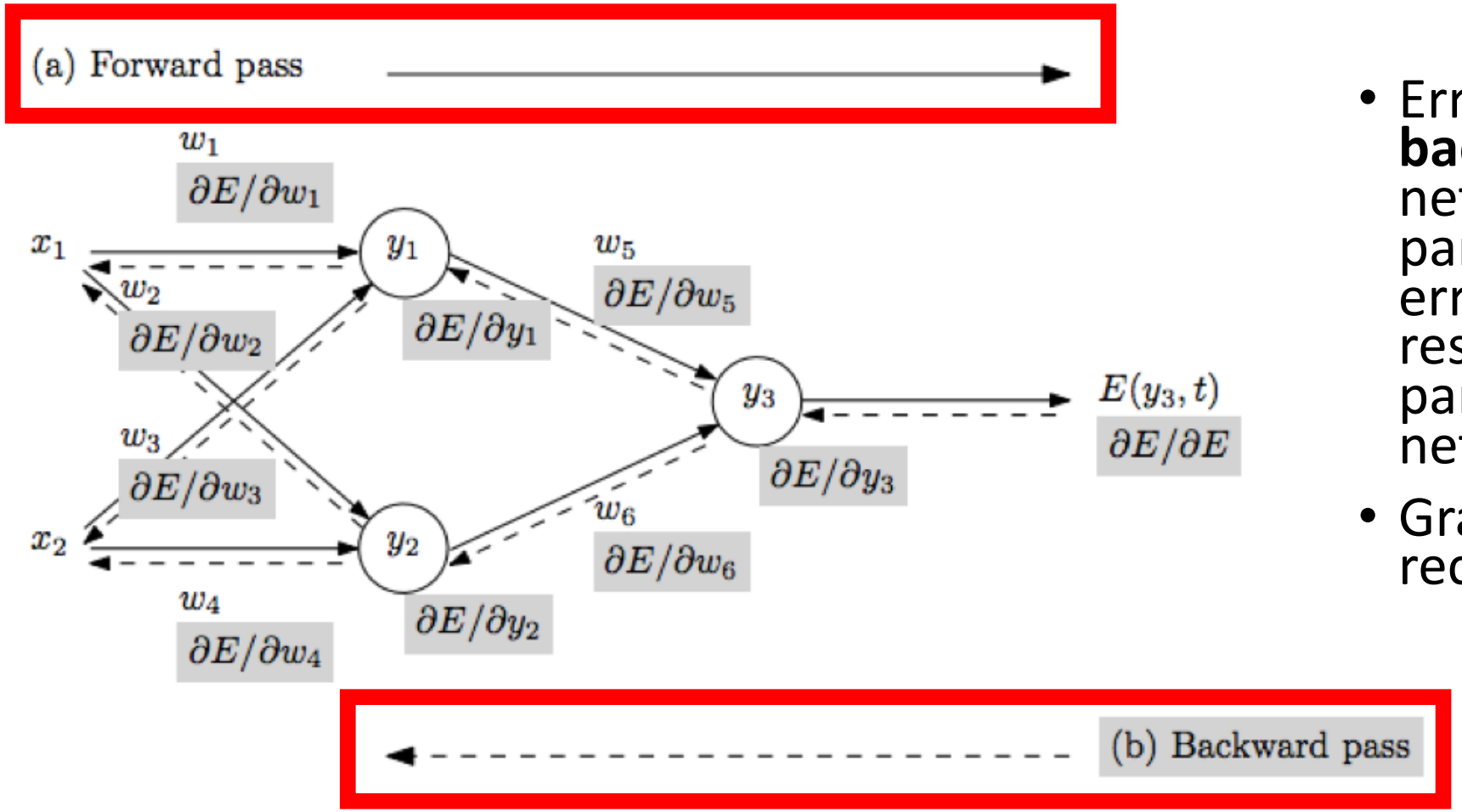
$$\frac{\partial E}{\partial w_{ij}} = -\left(\sum_k Err_k w_{jk}\right) * sigmoid\left(\sum_i w_{ij} * o_i\right) * \left(1 - sigmoid\left(\sum_i w_{ij} * o_i\right)\right)$$




Symmetry in how to solve for parameters in hidden layers

$$\frac{\partial E}{\partial w_{jk}} = -(t_k - o_k) * sigmoid\left(\sum_j w_{jk} * o_j\right) * \left(1 - sigmoid\left(\sum_j w_{jk} * o_j\right)\right)$$

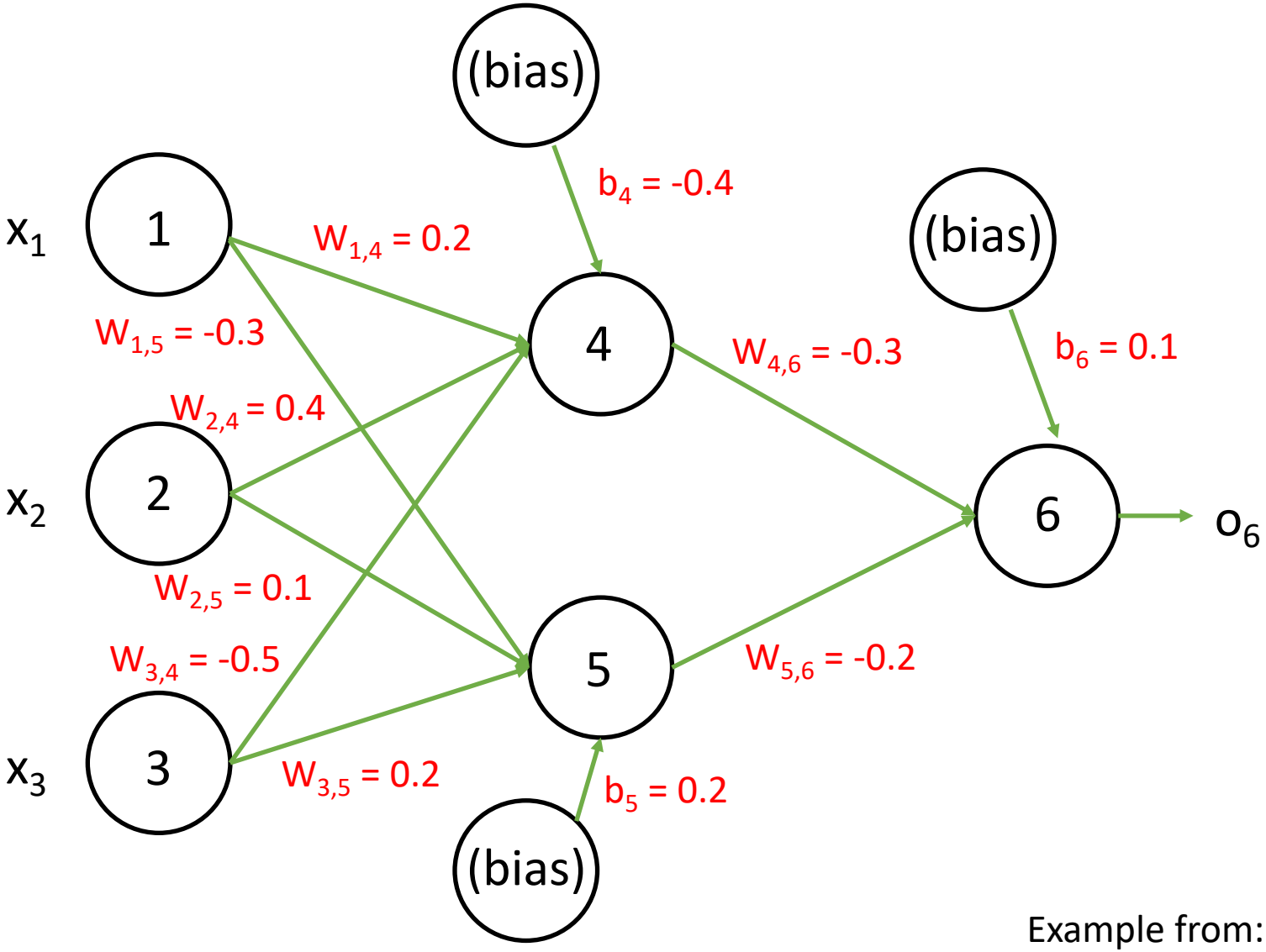
# Backpropagation: Depends on Two Steps




- Error **propagated backwards** from output of network by computing partial derivatives of the error at the output with respect to each model parameter within the network.
- Gradient computed through recursive use of **chain rule.**

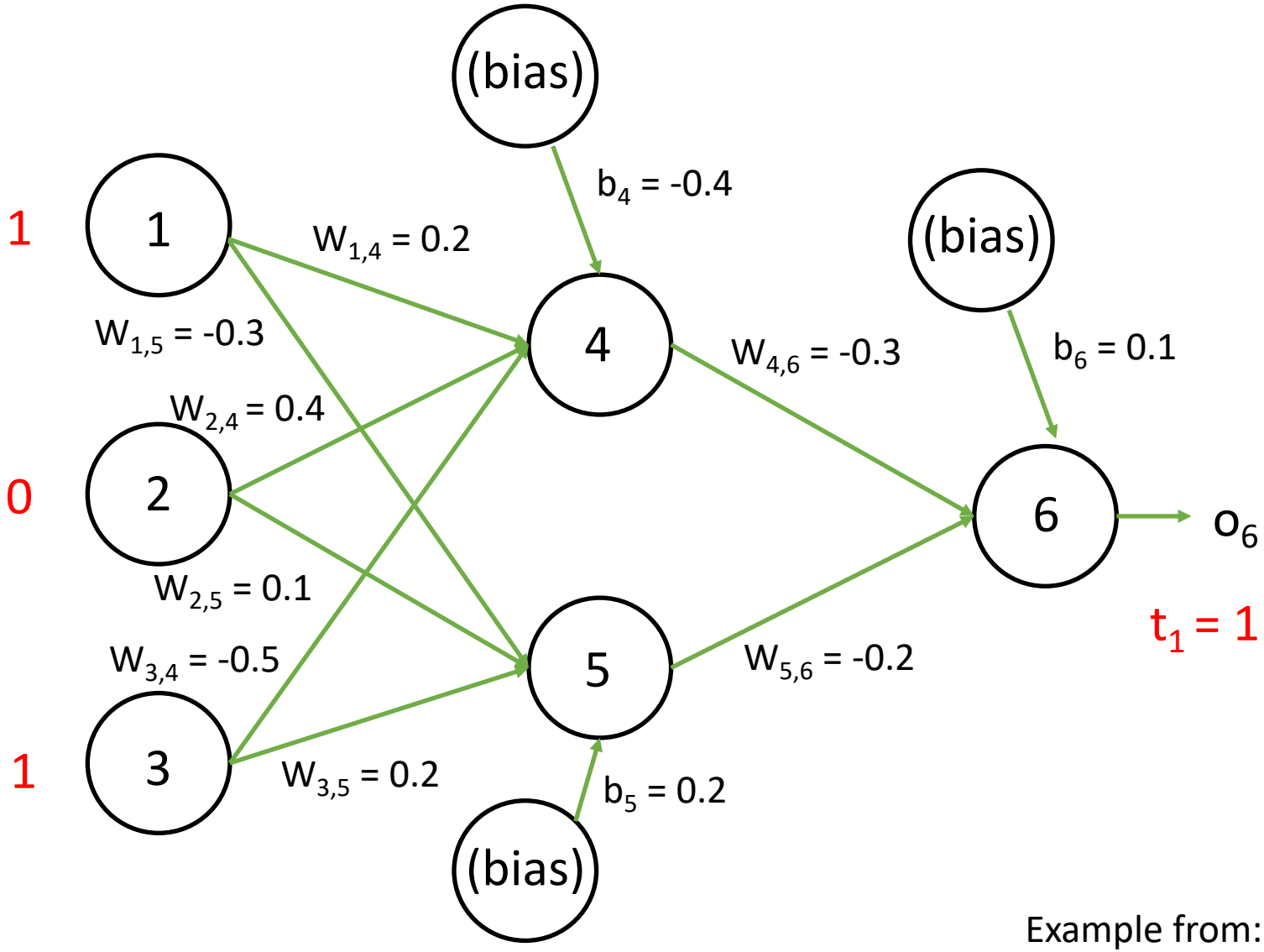
Figure from: Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind; Automatic Differentiation in Machine Learning: a Survey; 2018

# Example: Initialization Values (Weights, Biases)



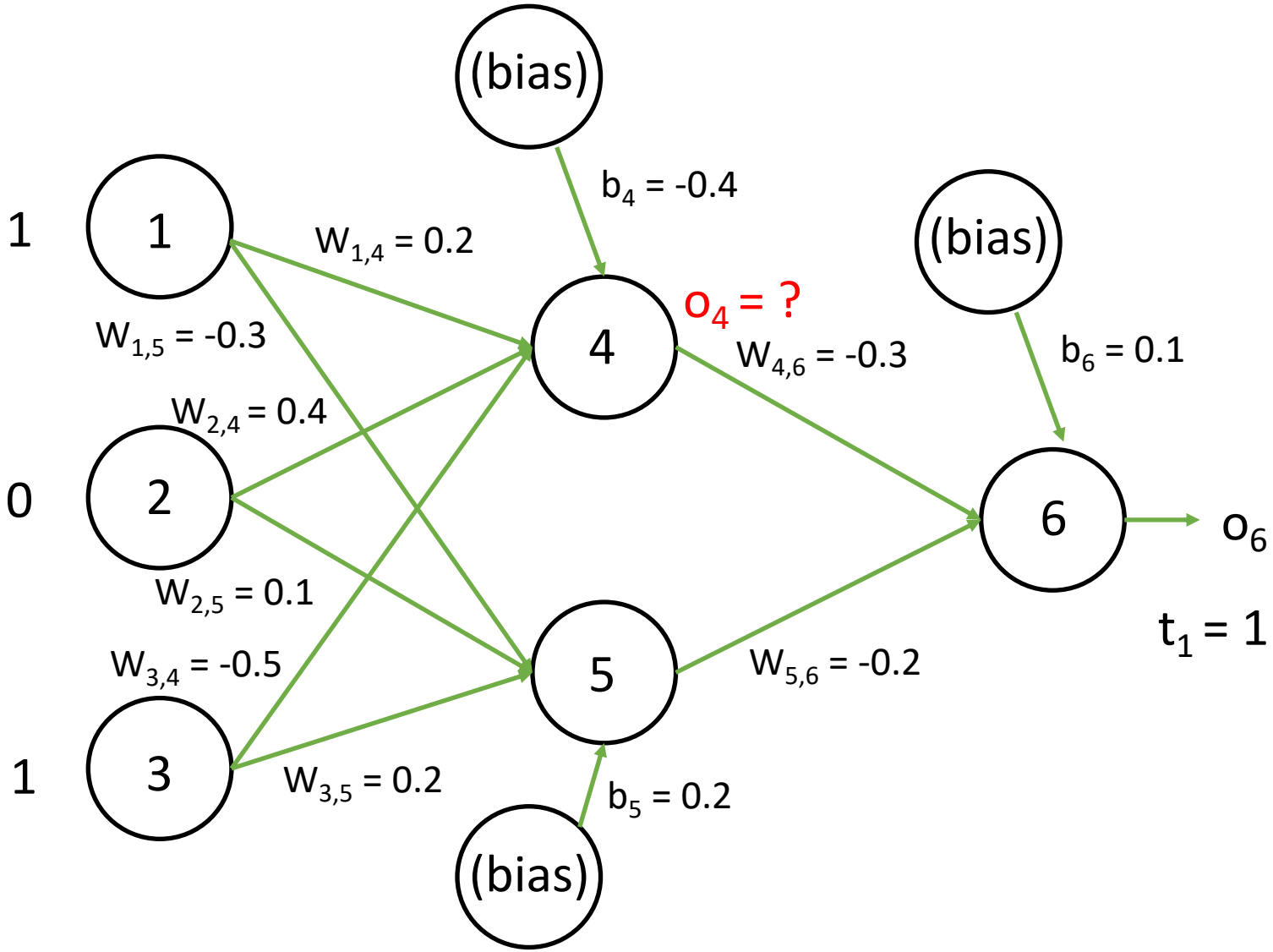

Example from: Jiawei Han and Micheline Kamber; Data Mining.

# Example: Input Training Example




Example from: Jiawei Han and Micheline Kamber; Data Mining.

# Example: Step 1 – Forward Pass




Input to node 4:

$i_4 = (1 \times 0.2 + 0 \times 0.4 + 1 \times -0.5) - 0.4$

$i_4 = -0.7$

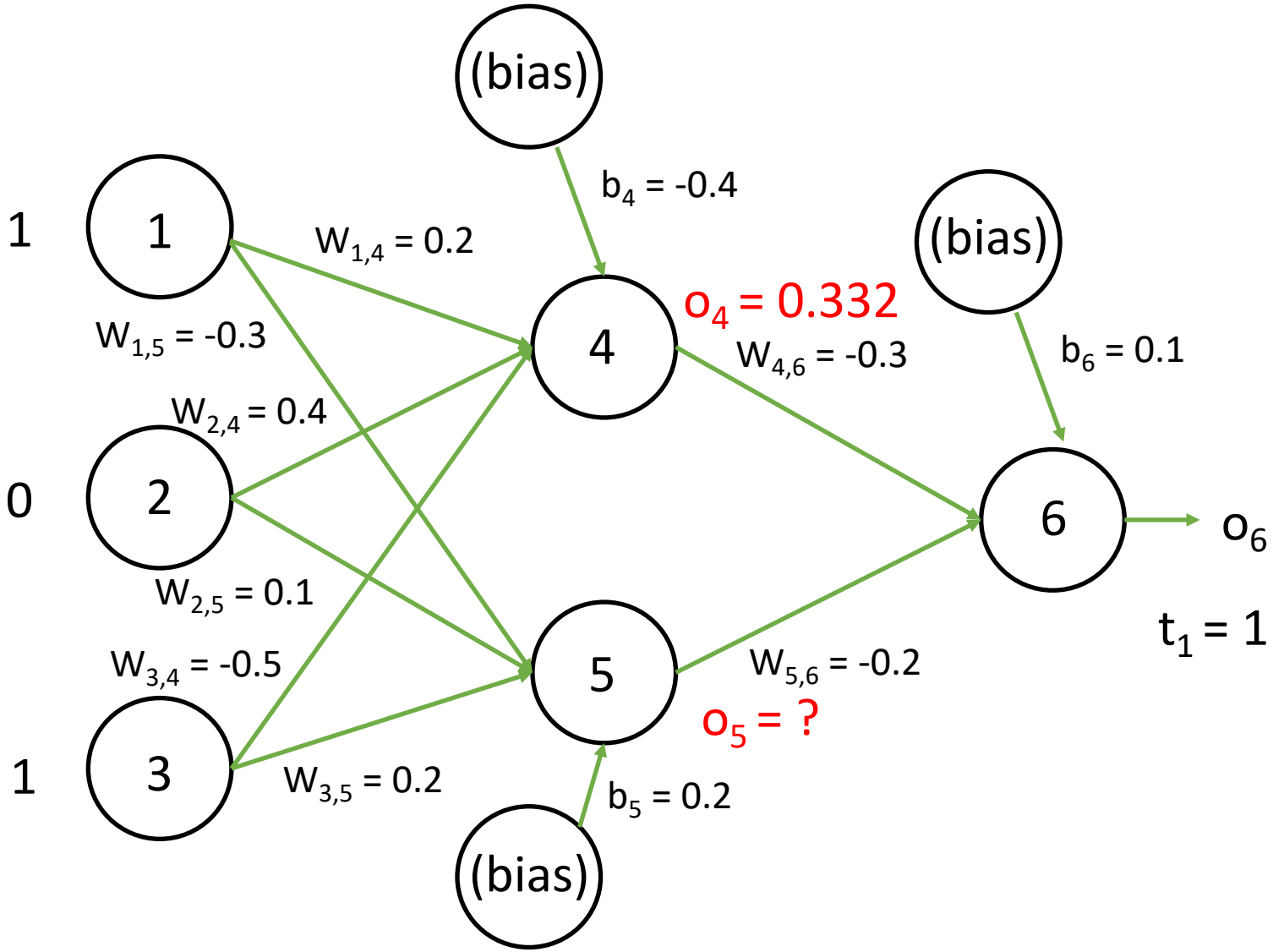
Output of node 4 (sigmoid function):

$o_4 = sigmoid(-0.7)$

$o_4 = 1/(1+e^{-(-0.7)})$

$o_4 = 0.332$

# Example: Step 1 – Forward Pass




Input to node 5:

$i_5 = (1 \times -0.3 + 0 \times 0.1 + 1 \times 0.2) + 0.2$

$i_5 = 0.1$

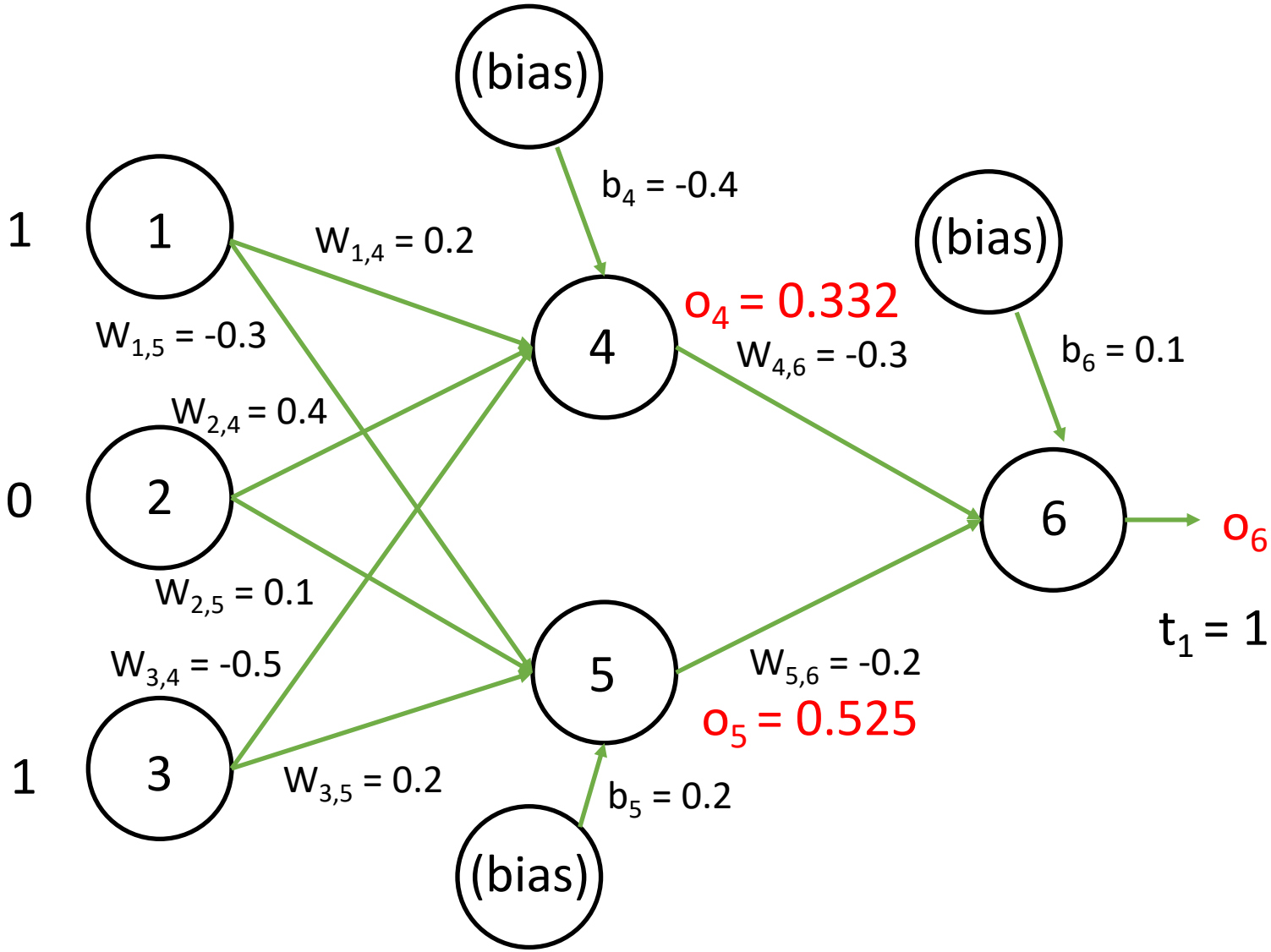
Output of node 5 (sigmoid function):

$o_5 = sigmoid(0.1)$

$o_5 = 1/(1+e^{-0.1})$

$o_5 = 0.525$

# Example: Step 1 – Forward Pass




Input to node 6:

$i_6 = (0.332 \times -0.3 + 0.1 \times -0.2) + 0.1$

$i_6 = -0.105$

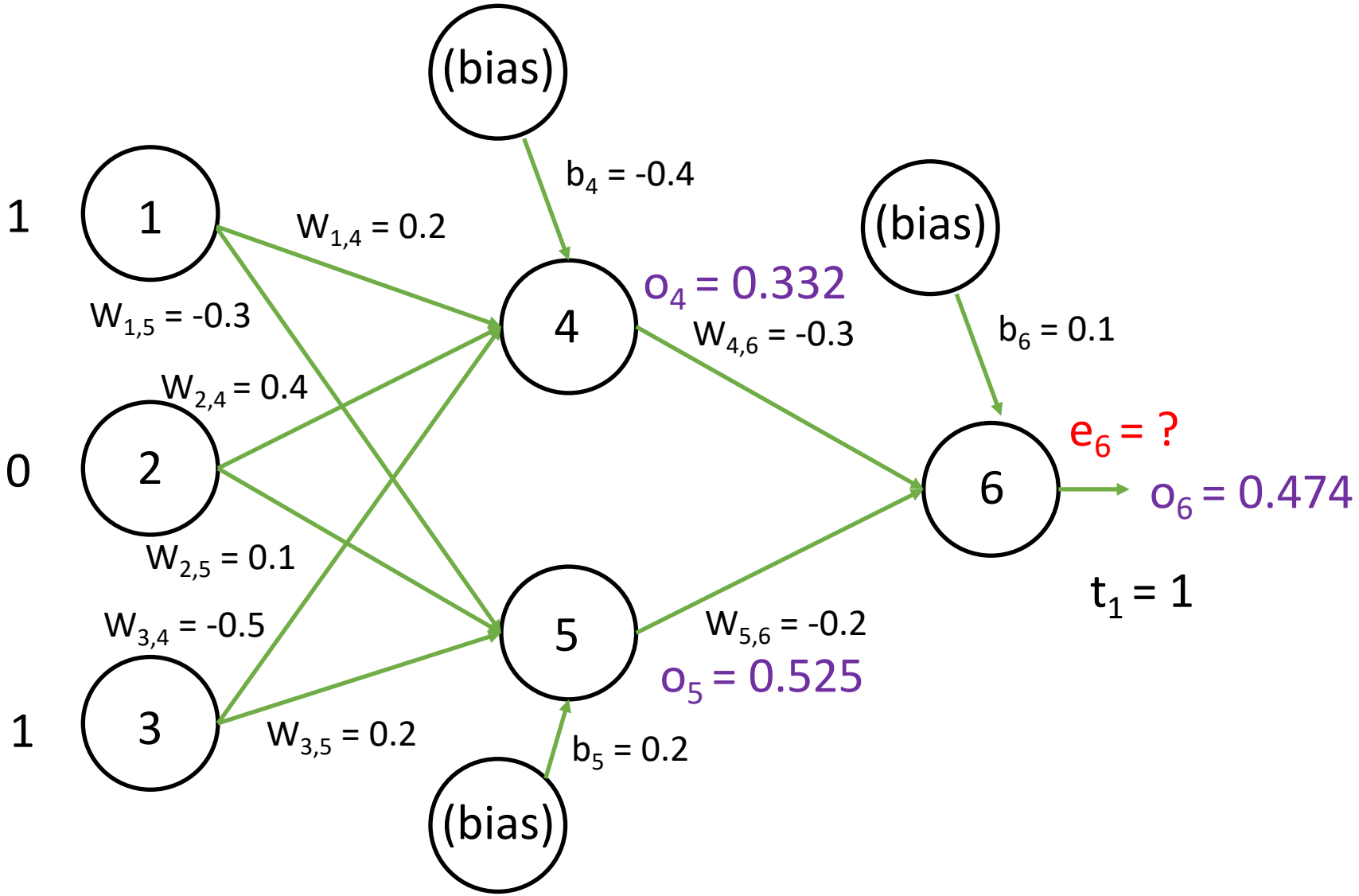
Output of node 6 (sigmoid function):

$o_6 = sigmoid(-0.105)$

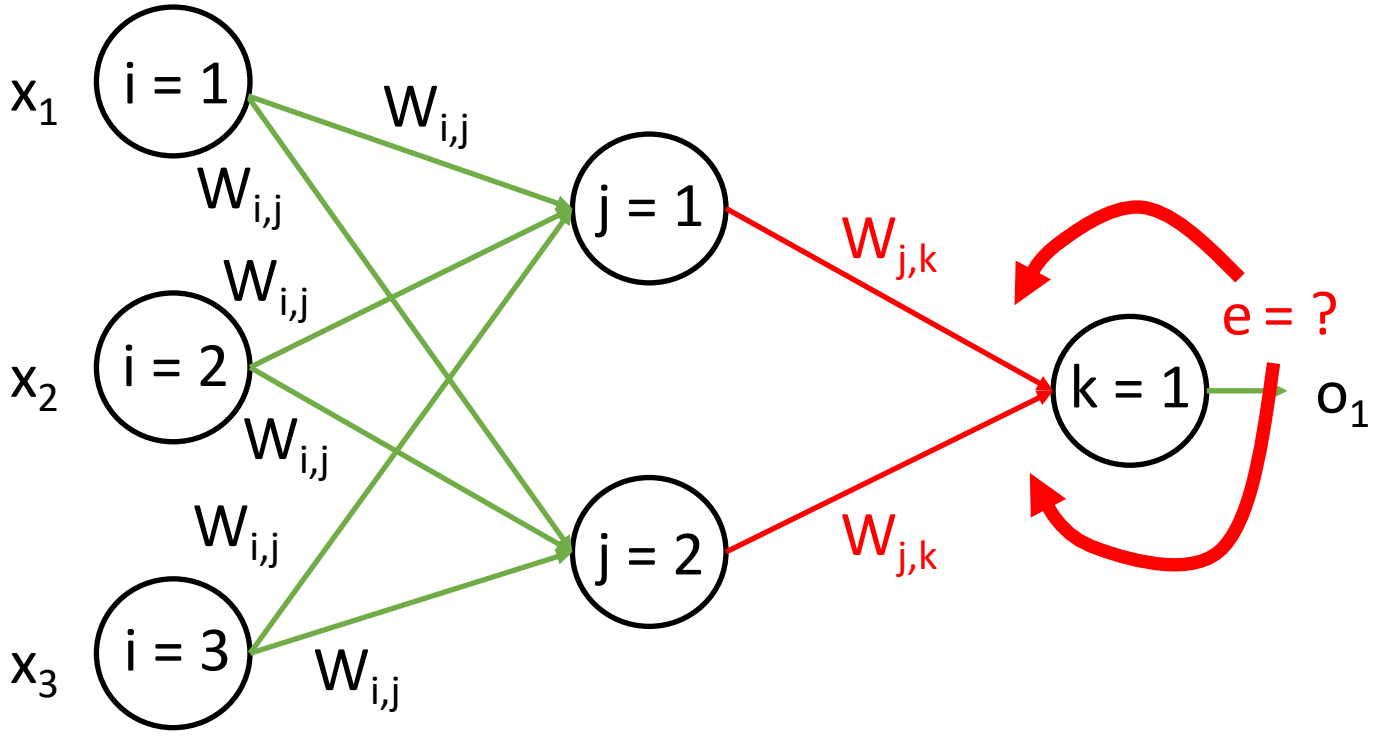
$o_6 = 1/(1+e^{-(-0.105)})$

$o_6 = 0.474$

# Example: Step 2 – Backward Pass

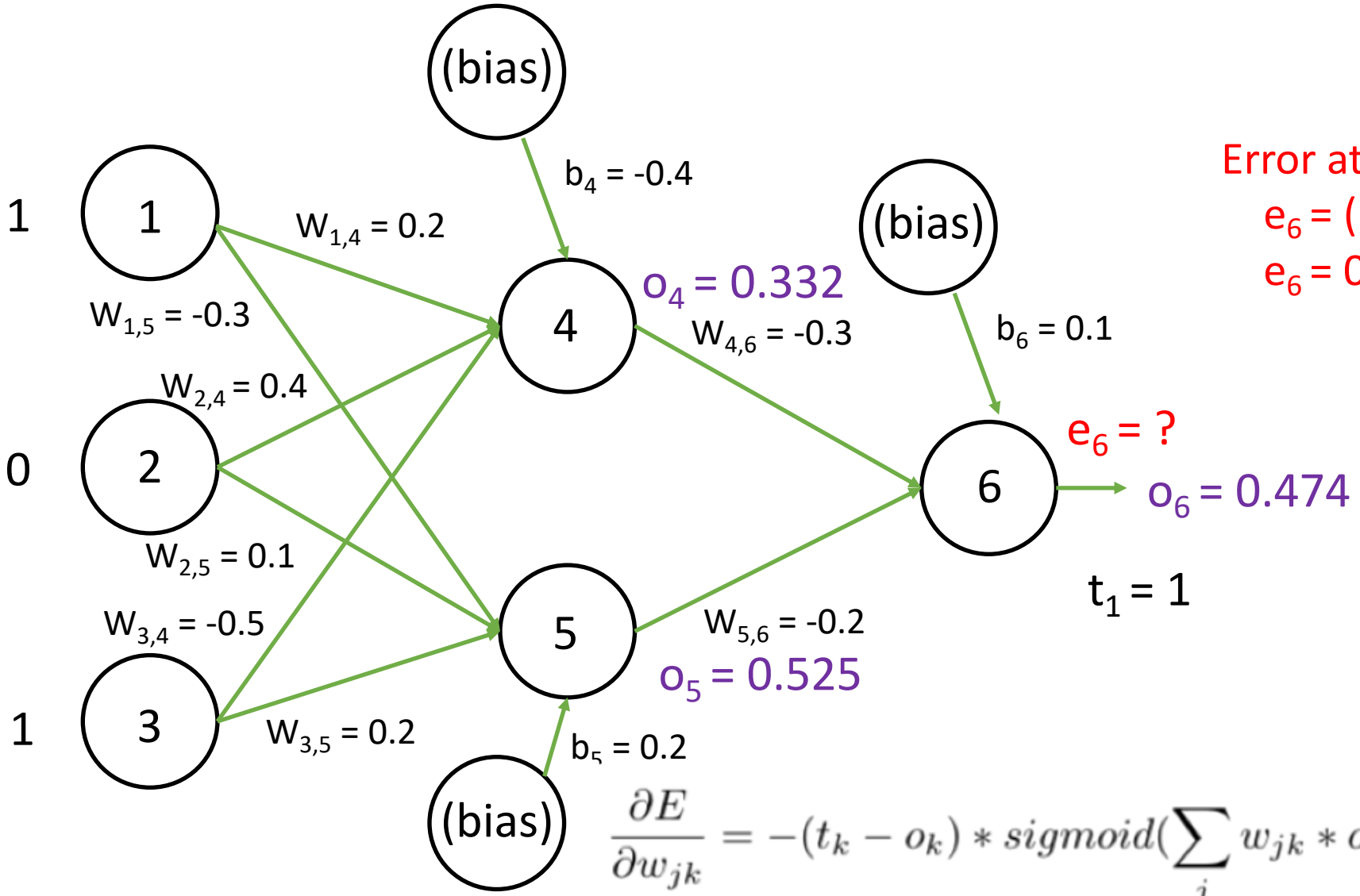


# Recall: Gradient Equation (Output Layer)




$$\frac{\partial E}{\partial w_{jk}} = -(t_k - o_k) * sigmoid(\sum_j w_{jk} * o_j) * (1 - sigmoid(\sum_j w_{jk} * o_j))$$

# Example: Step 2 – Backward Pass



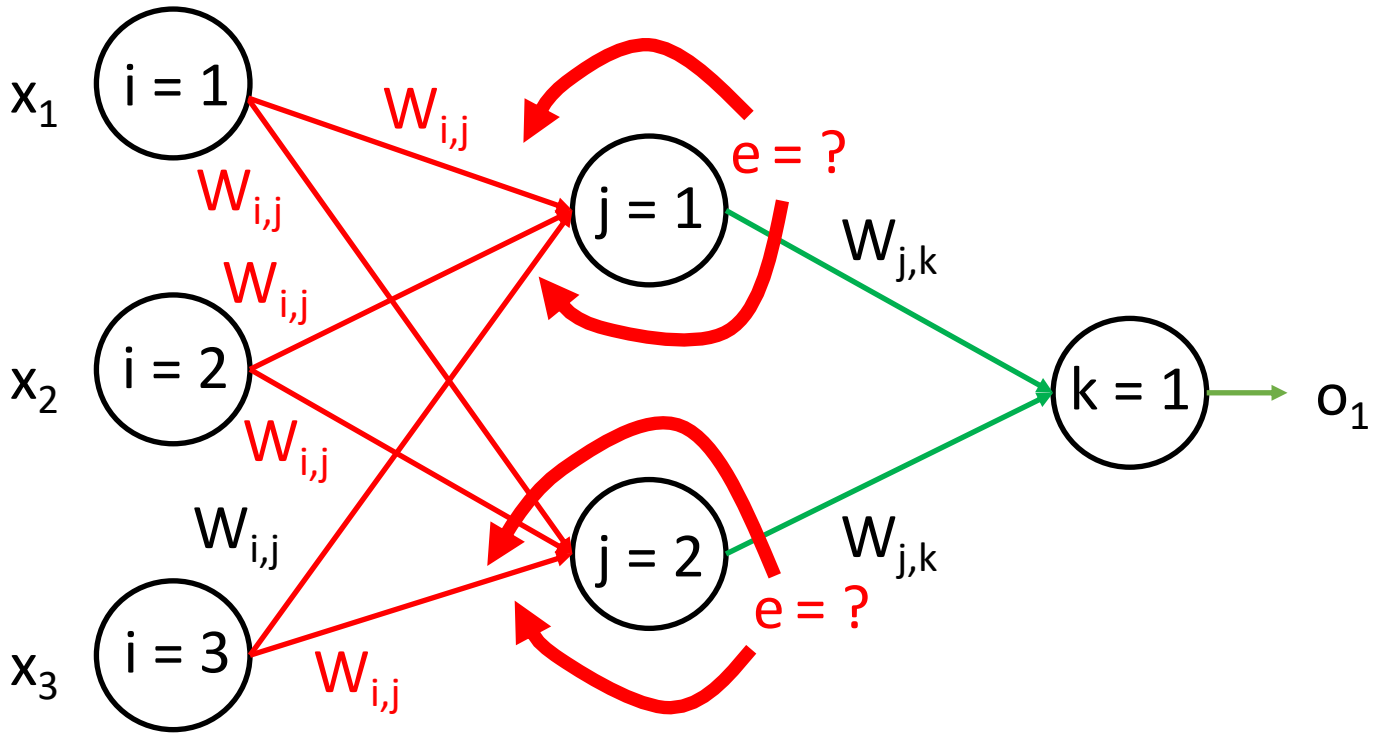

Error at node 6:

$e_6 = (1-0.474)\ (0.474)(1-0.474)$

$e_6 = 0.1311$

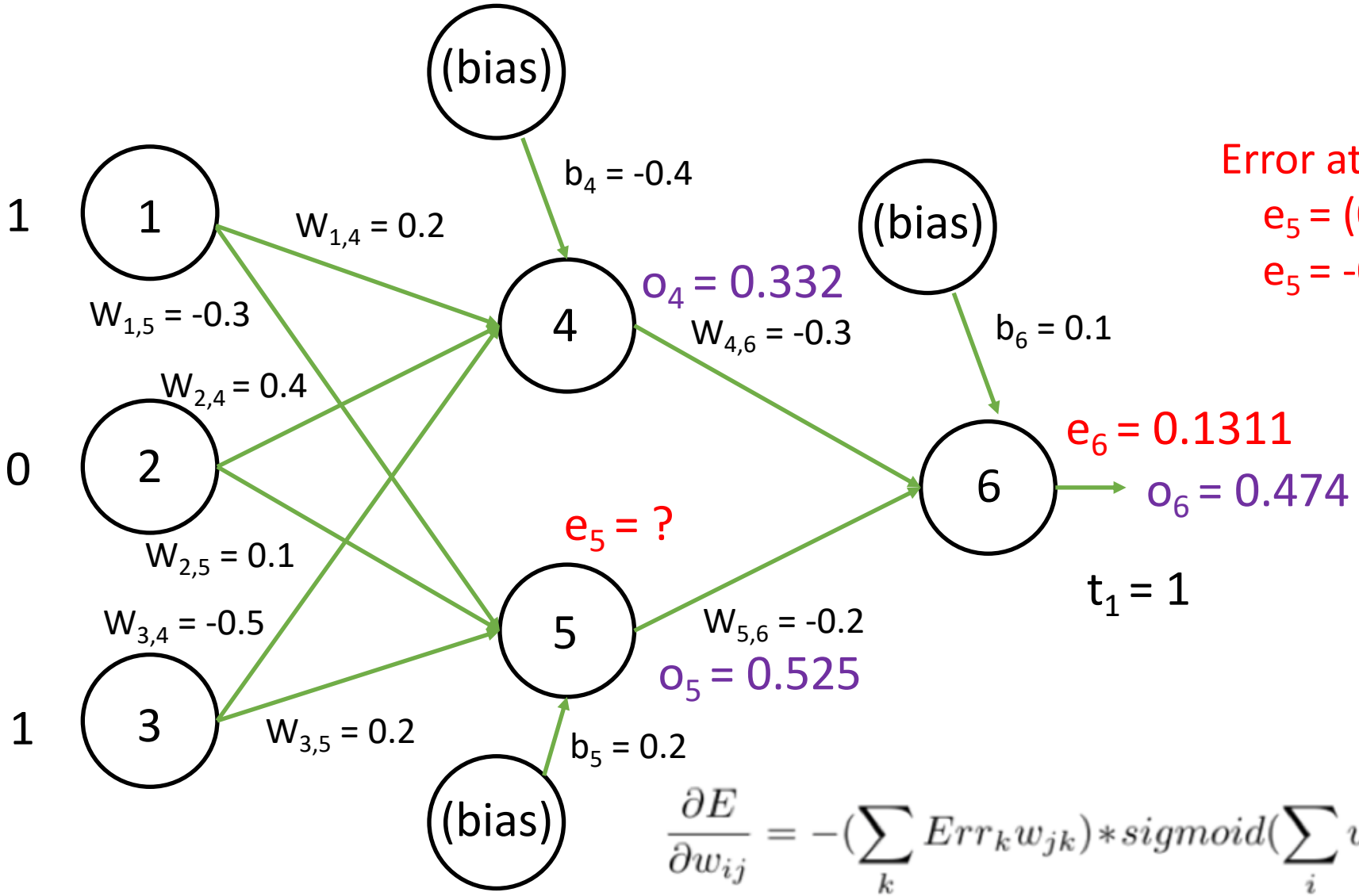
$$\frac{\partial E}{\partial w_{jk}} = -(t_k - o_k) * sigmoid(\sum_j w_{jk} * o_j) * (1 - sigmoid(\sum_j w_{jk} * o_j))$$

# Recall: Gradient Equation (Hidden Layer)




$$\frac{\partial E}{\partial w_{ij}} = -(\sum_k Err_k w_{jk}) * sigmoid(\sum_i w_{ij} * o_i) * (1 - sigmoid(\sum_i w_{ij} * o_i))$$

# Example: Step 2 – Backward Pass



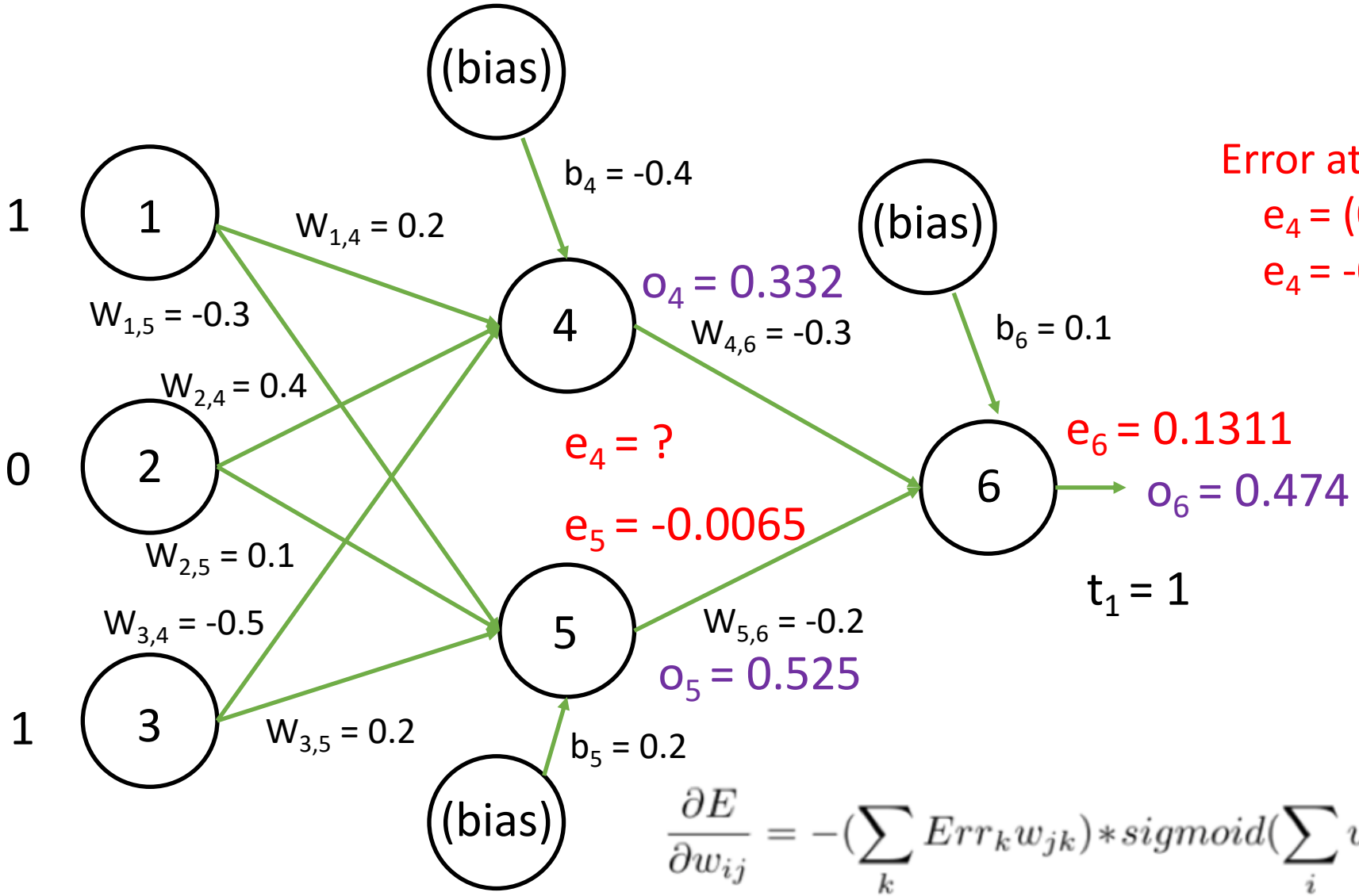

Error at node 5:

$e_5 = (0.1311)(-0.2)(0.525)(1-0.525)$

$e_5 = -0.0065$

$$\frac{\partial E}{\partial w_{ij}} = -(\sum_k Err_k w_{jk}) * sigmoid(\sum_i w_{ij} * o_i) * (1 - sigmoid(\sum_i w_{ij} * o_i))$$

# Example: Step 2 – Backward Pass



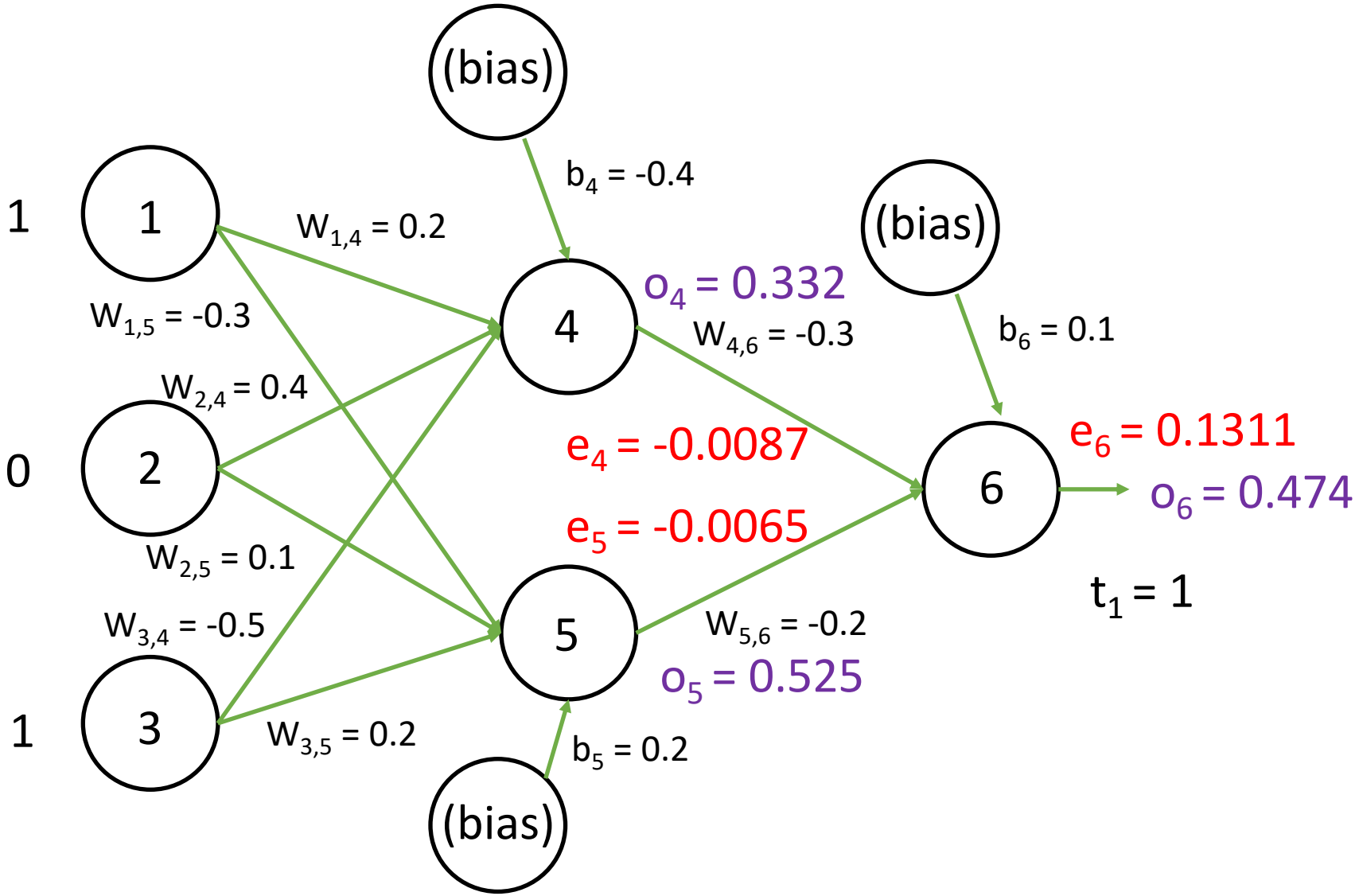

Error at node 4:

$e_4 = (0.1311)(-0.3)(0.332)(1-0.332)$

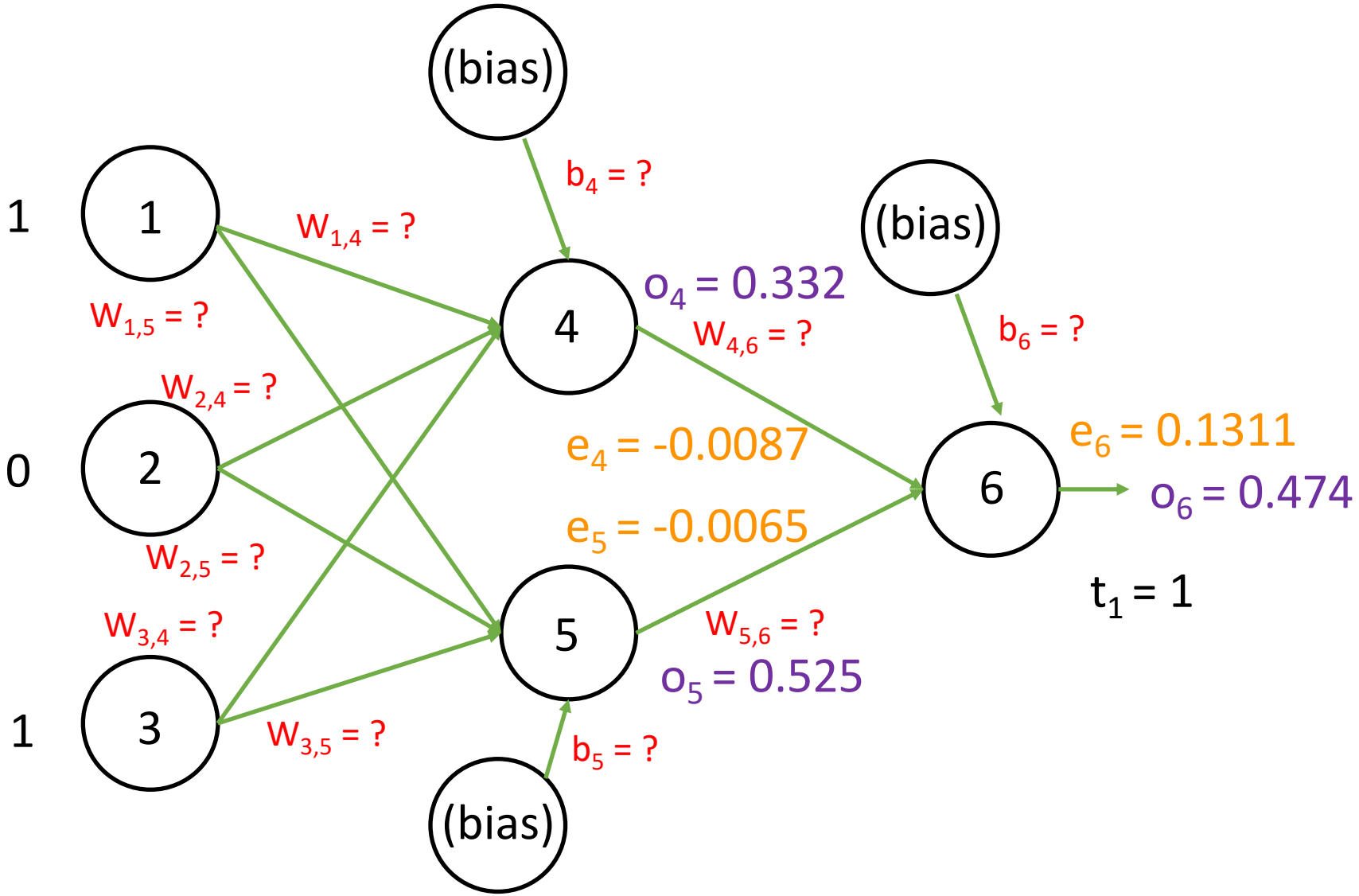
$e_4 = -0.0087$

$$\frac{\partial E}{\partial w_{ij}} = -(\sum_k Err_k w_{jk}) * sigmoid(\sum_i w_{ij} * o_i) * (1 - sigmoid(\sum_i w_{ij} * o_i))$$

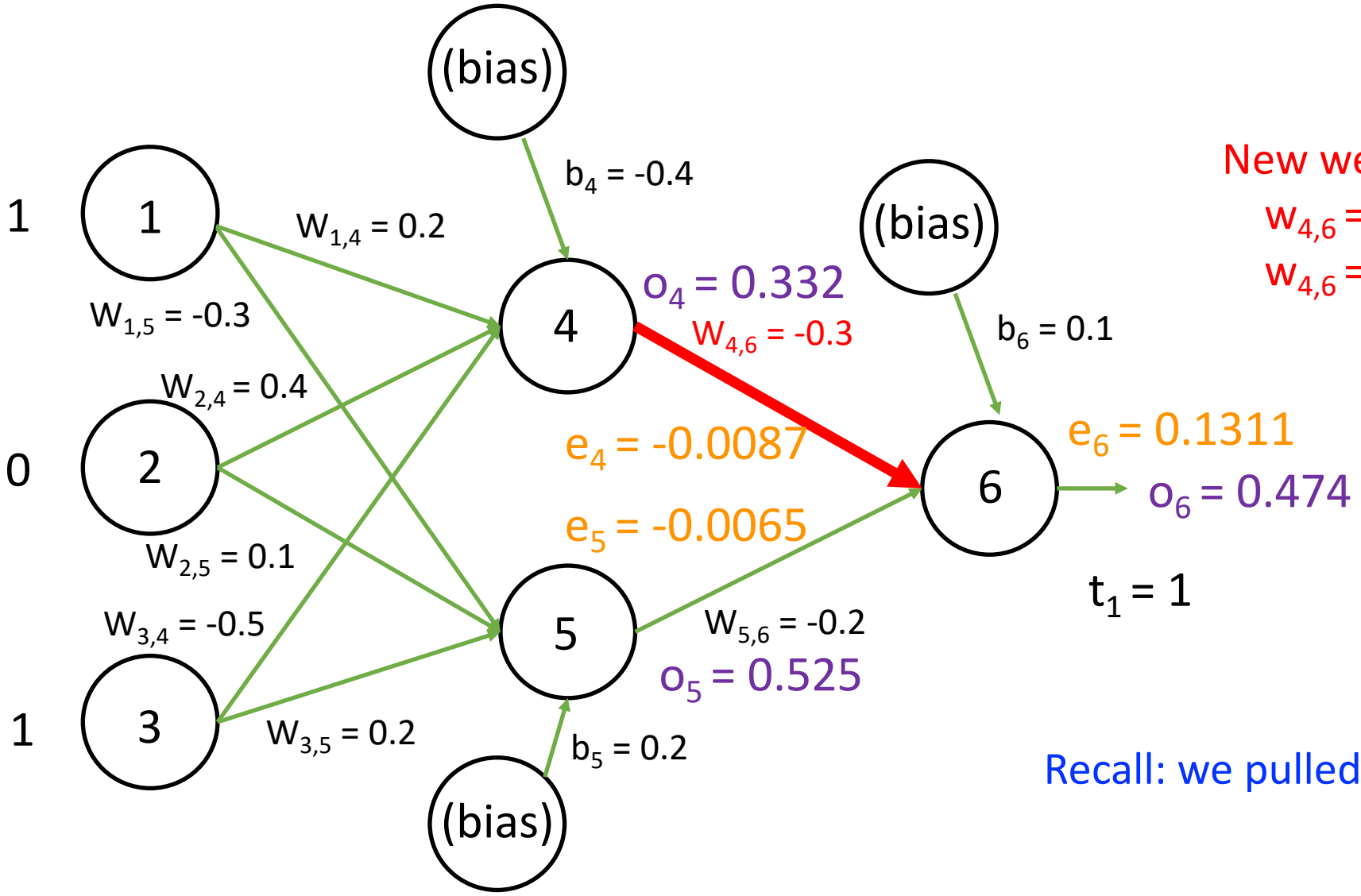
# Example: Step 2 – Backward Pass



# Example: Step 3 – Update Weights



# Example: Step 3 – Update Weights




New weights (learning rate = 0.9):

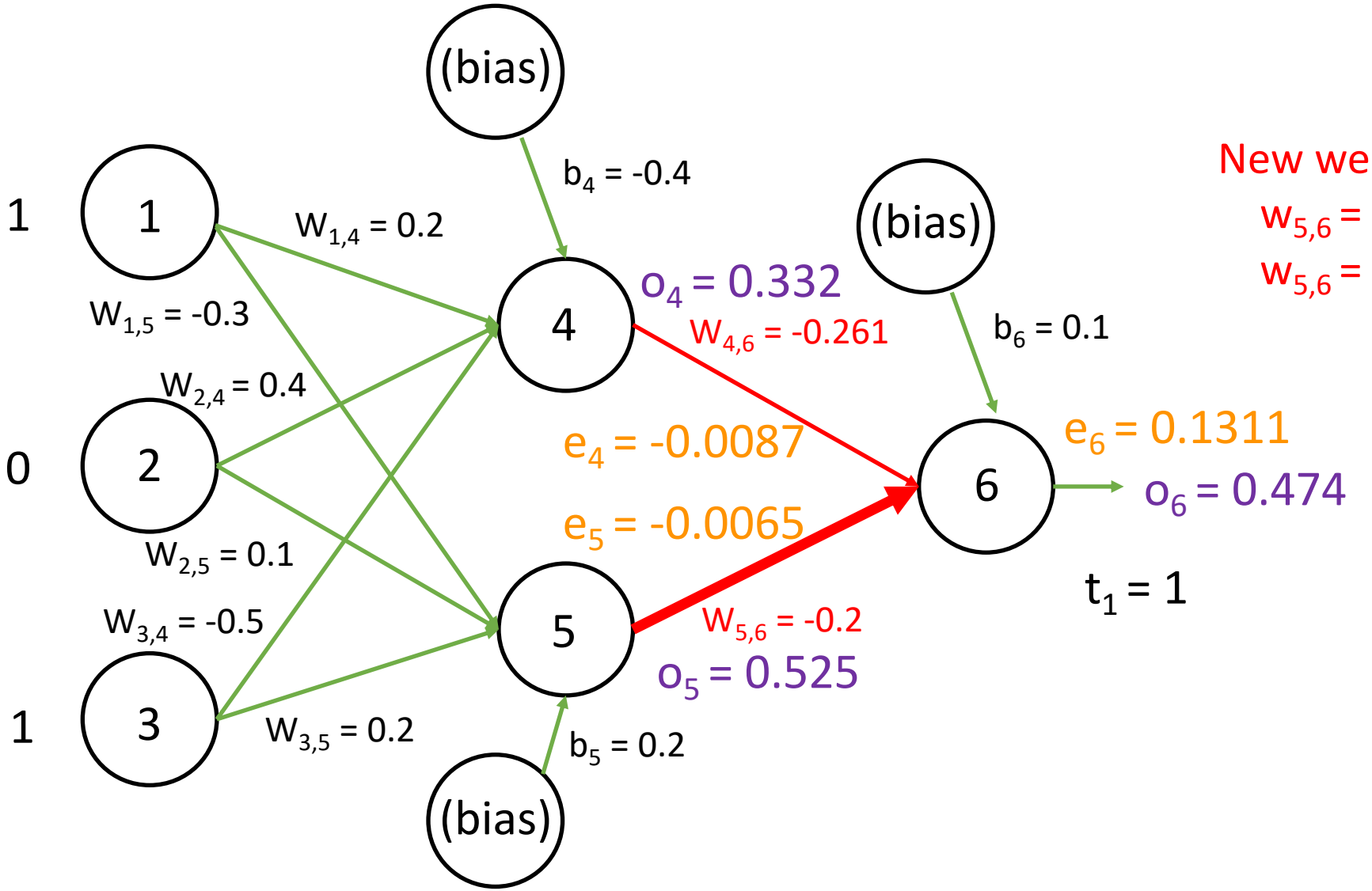
$w_{4,6} = -0.3 + (0.9)(0.1311)(0.332)$

$w_{4,6} = -0.261$

Recall: we pulled this out of the equation earlier

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

# Example: Step 3 – Update Weights



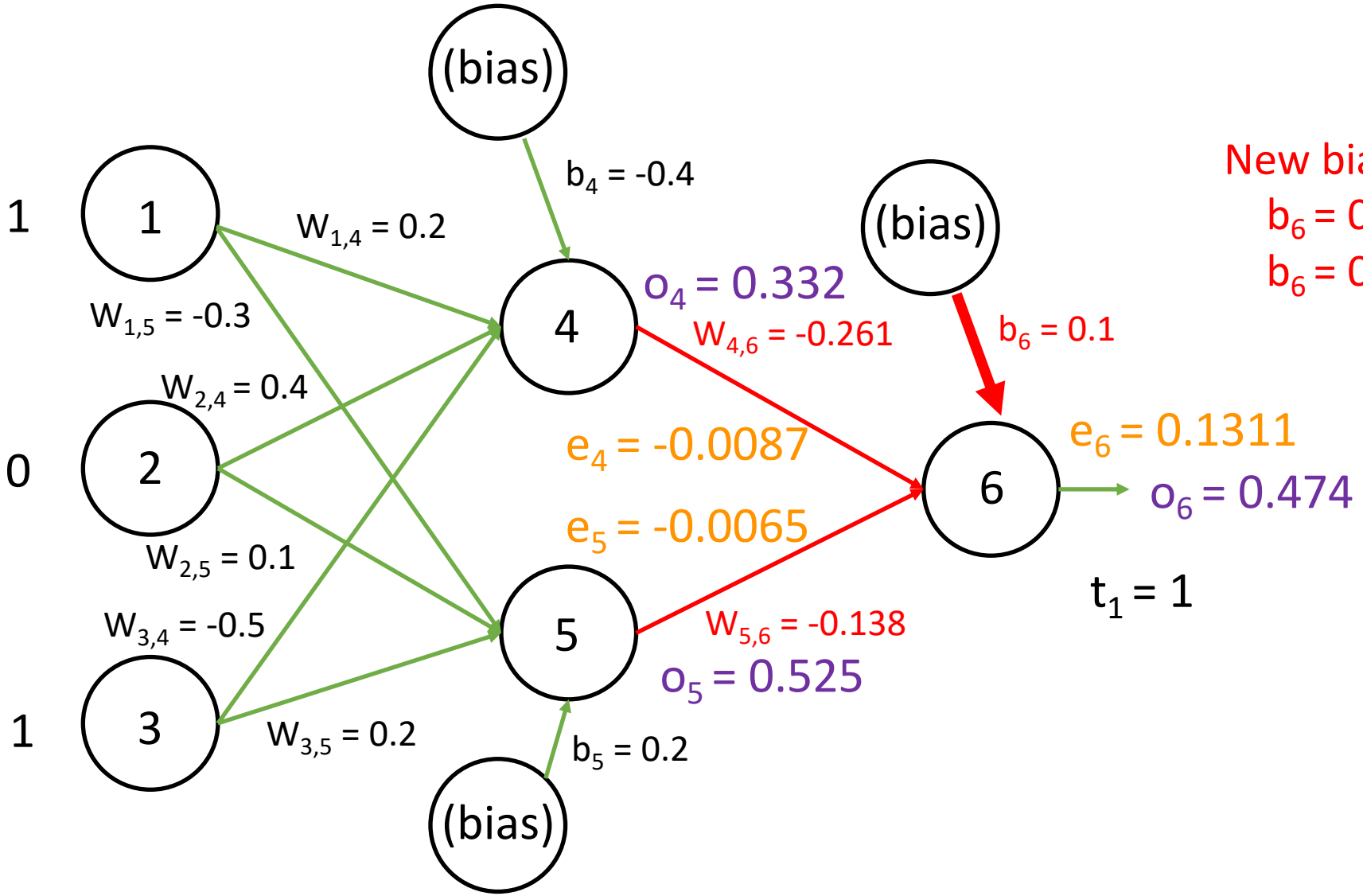

New weights (learning rate = 0.9):

$w_{5,6} = -0.2 + (0.9)(0.1311)(0.525)$

$w_{5,6} = -0.138$

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

# Example: Step 3 – Update Weights



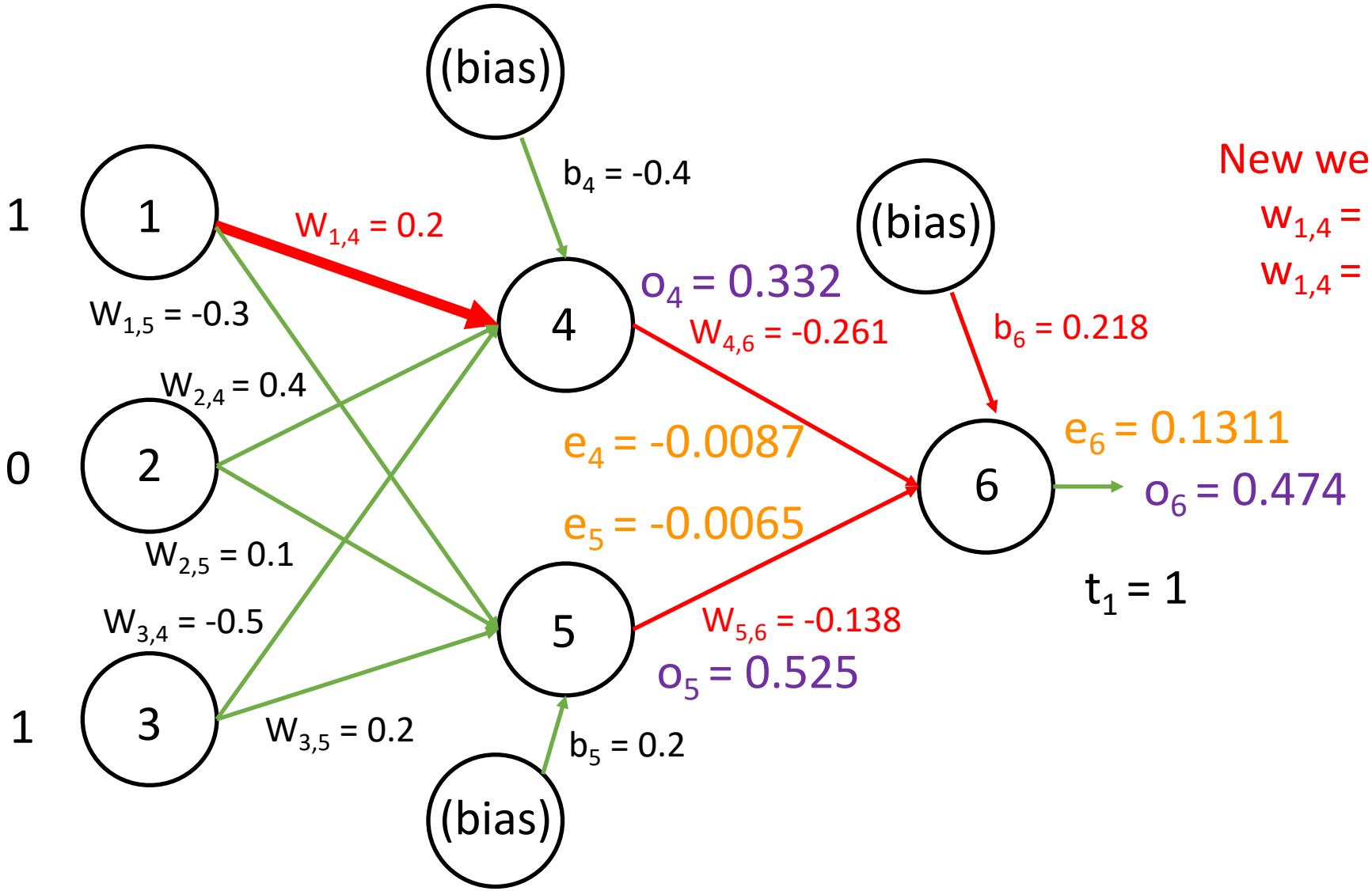

New bias (learning rate = 0.9):

$b_6 = 0.1 + (0.9)(0.1311)$

$b_6 = 0.218$

$$b_k = b_k + \eta Err_k$$

# Example: Step 3 – Update Weights



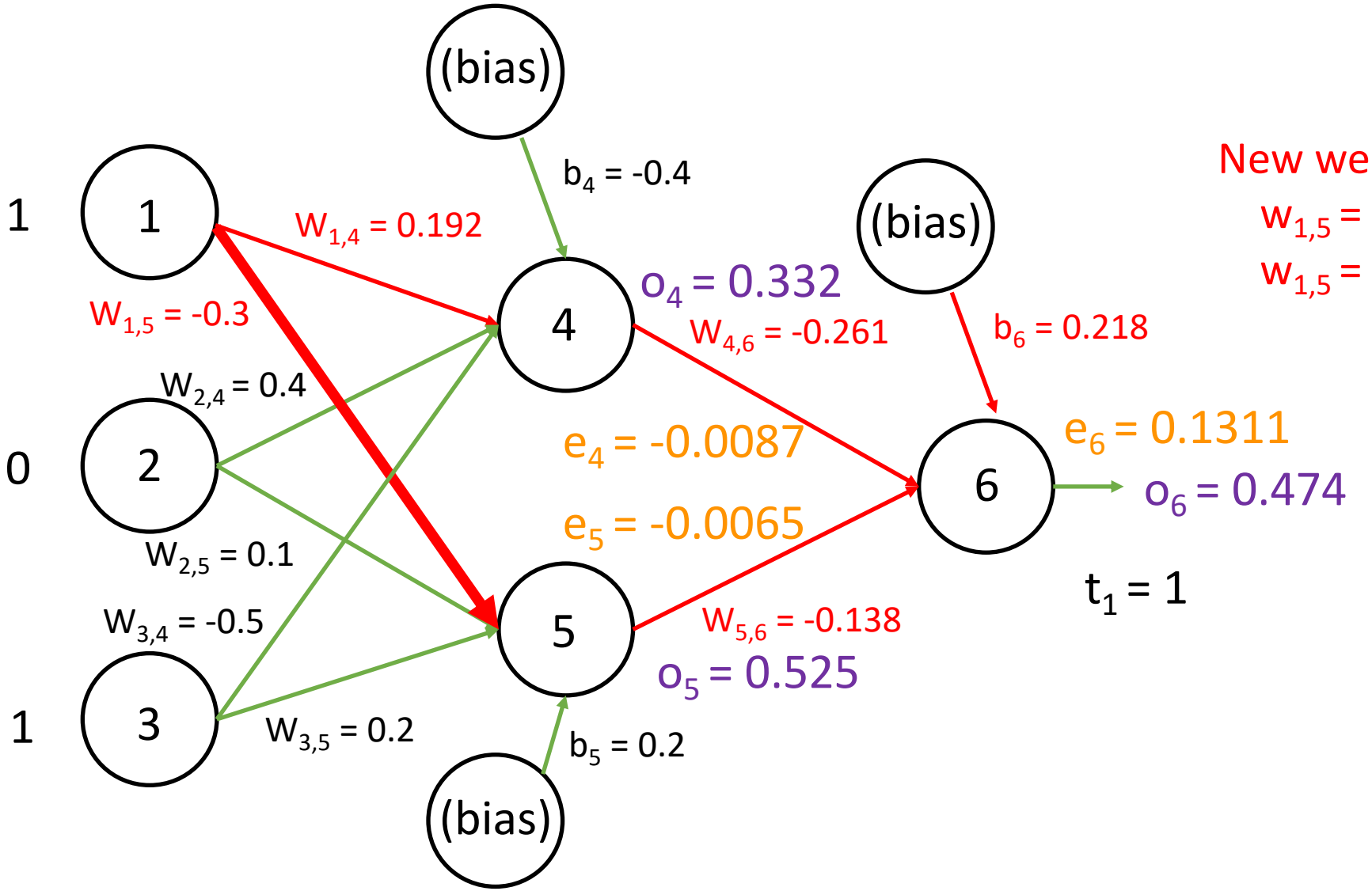

New weights (learning rate = 0.9):

$w_{1,4} = 0.2 + (0.9)(-0.0087)(1)$

$w_{1,4} = 0.192$

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

# Example: Step 3 – Update Weights



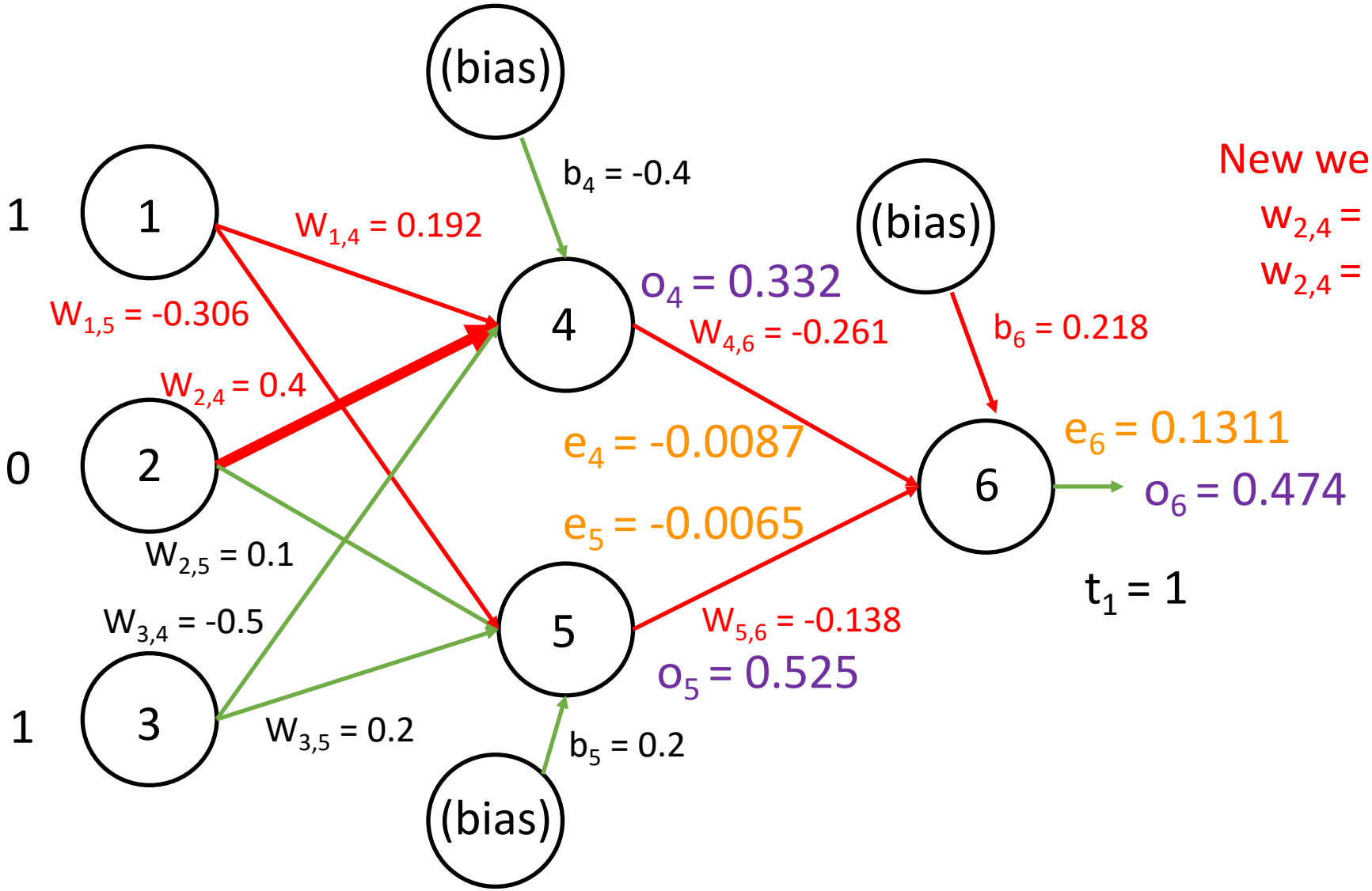

New weights (learning rate = 0.9):

$w_{1,5} = -0.3 + (0.9)(-0.0065)(1)$

$w_{1,5} = -0.306$

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

# Example: Step 3 – Update Weights



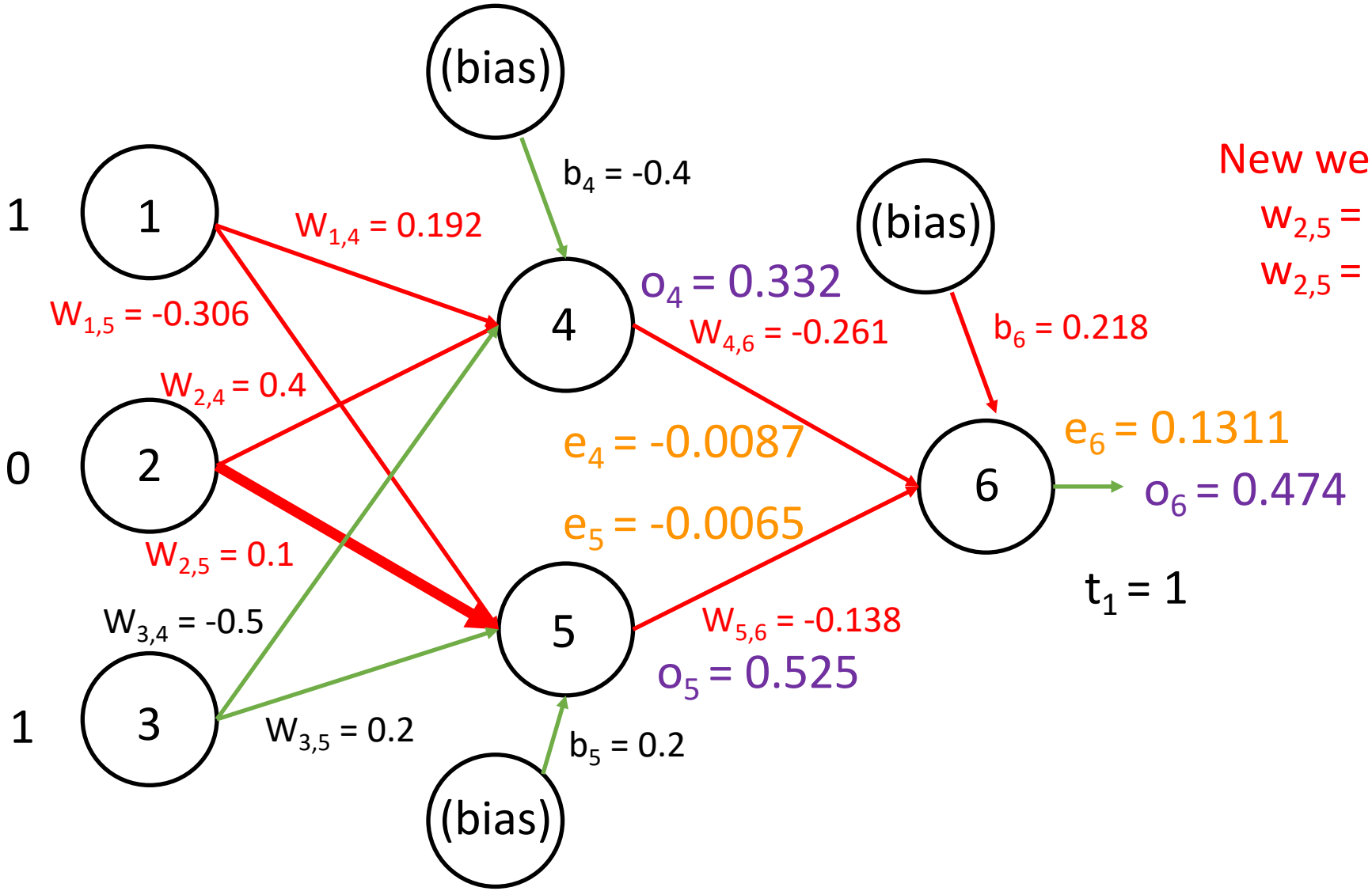

New weights (learning rate = 0.9):

$w_{2,4} = 0.4 + (0.9)(-0.0087)(0)$

$w_{2,4} = 0.4$

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

# Example: Step 3 – Update Weights



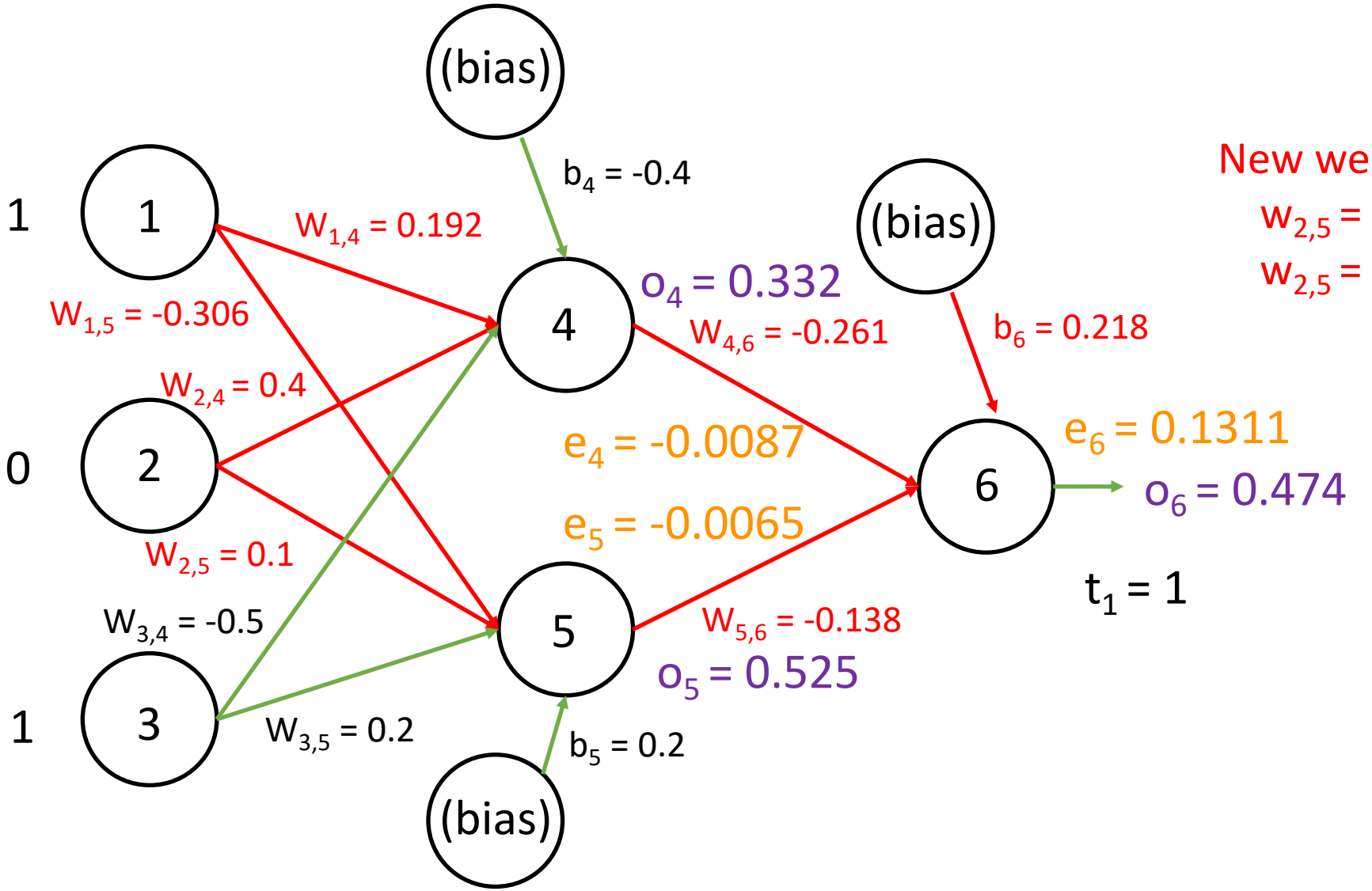

New weights (learning rate = 0.9):

$w_{2,5} = 0.1 + (0.9)(-0.0065)(0)$

$w_{2,5} = 0.1$

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

# Example: Step 3 – Update Weights



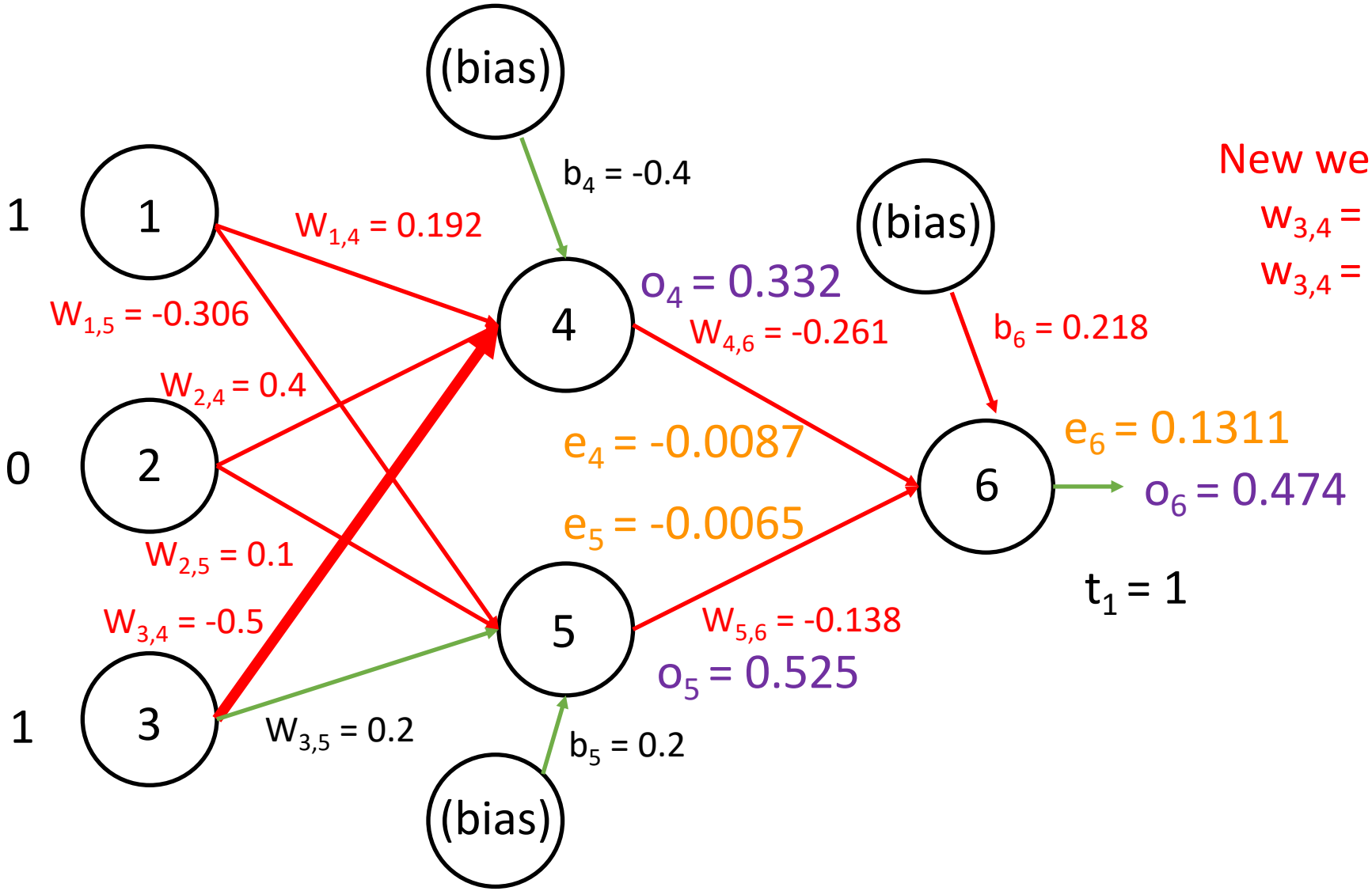

New weights (learning rate = 0.9):

$w_{2,5} = 0.1 + (0.9)(-0.0065)(0)$

$w_{2,5} = 0.1$

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

# Example: Step 3 – Update Weights



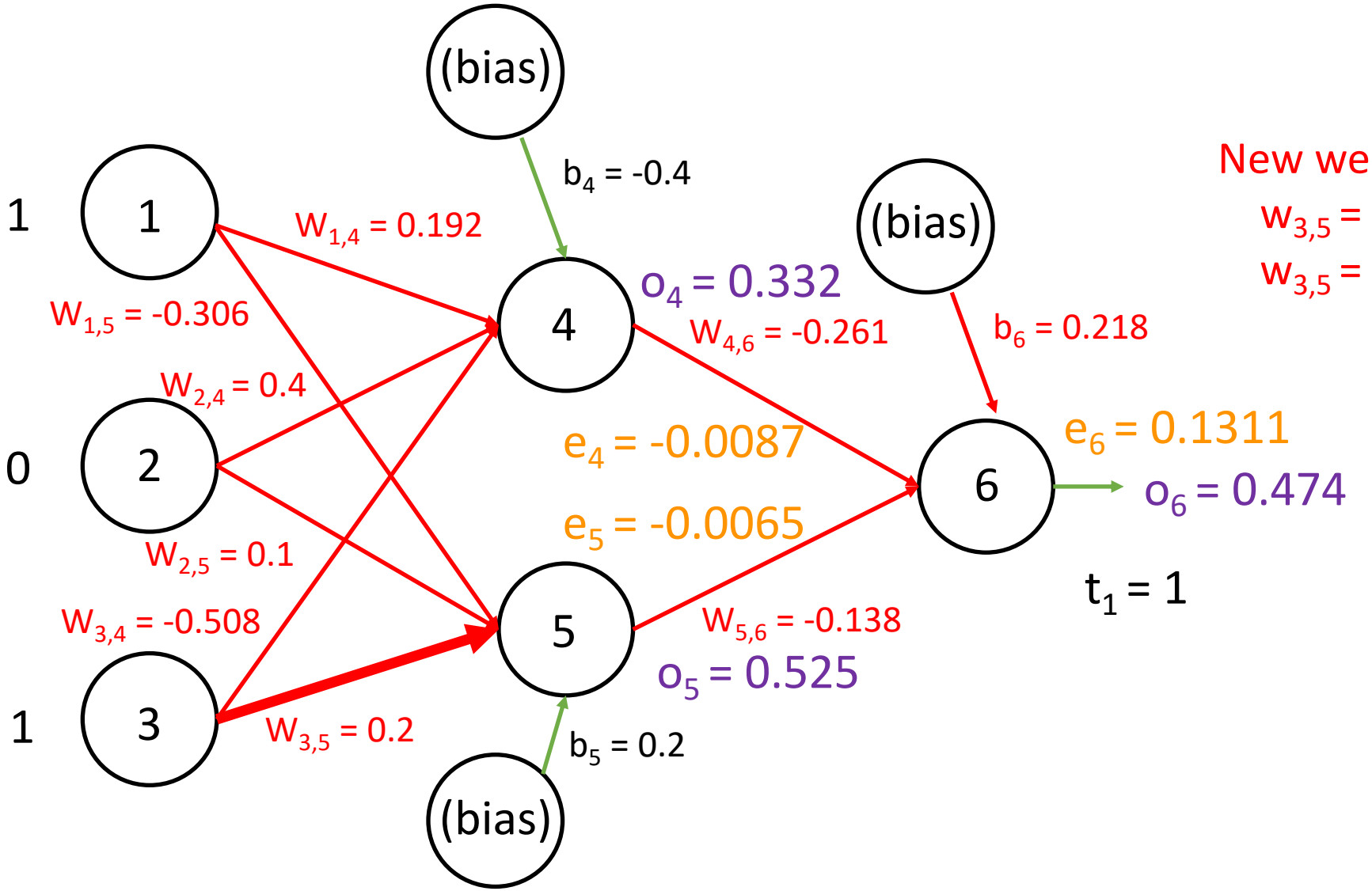

New weights (learning rate = 0.9):

$w_{3,4} = -0.5 + (0.9)(-0.0087)(1)$

$w_{3,4} = -0.508$

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

# Example: Step 3 – Update Weights



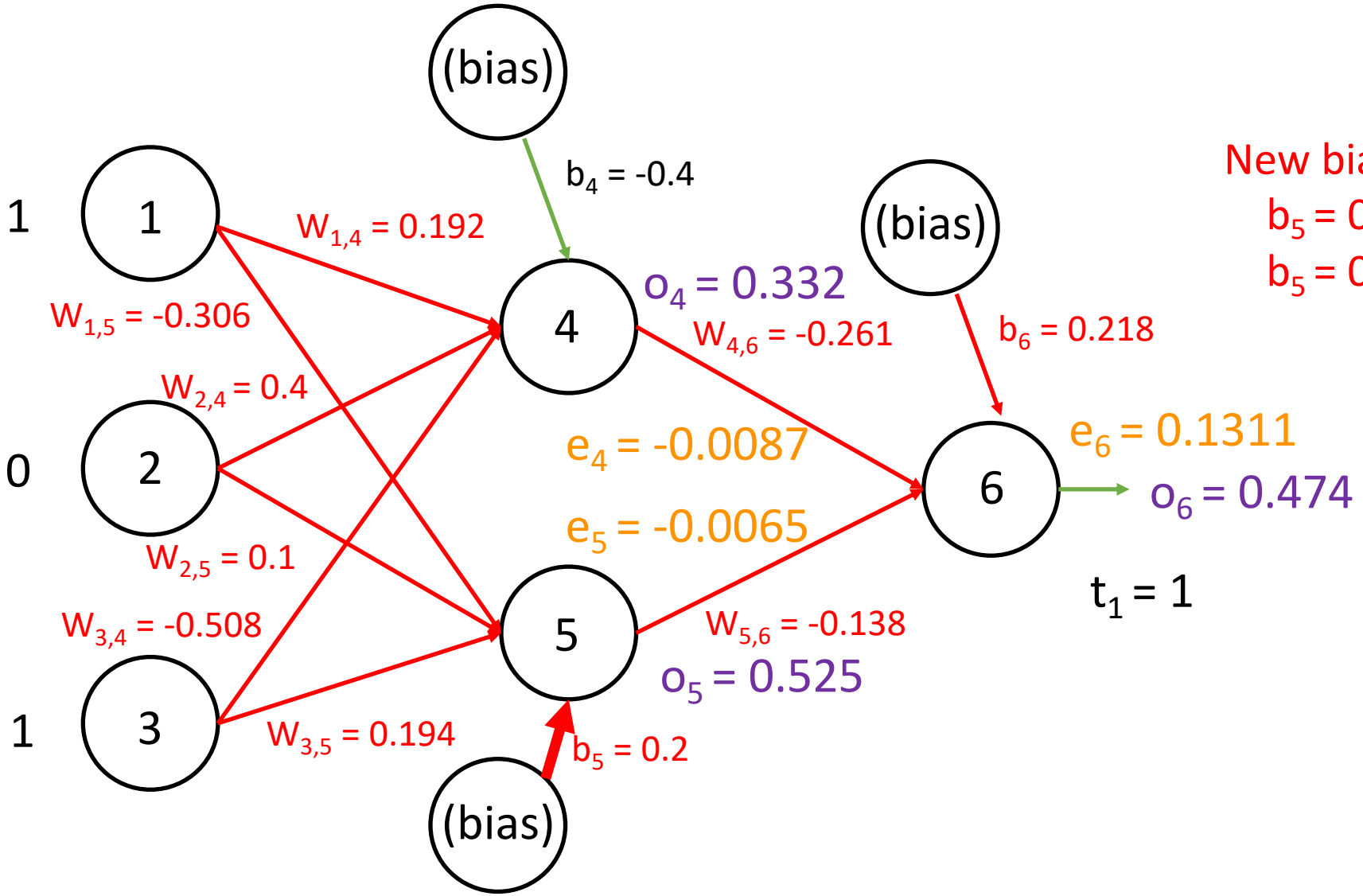

New weights (learning rate = 0.9):

$w_{3,5} = 0.2 + (0.9)(-0.0065)(1)$

$w_{3,5} = 01.94$

$$w_{jk} = w_{jk} + \eta Err_k o_j$$

# Example: Step 3 – Update Weights



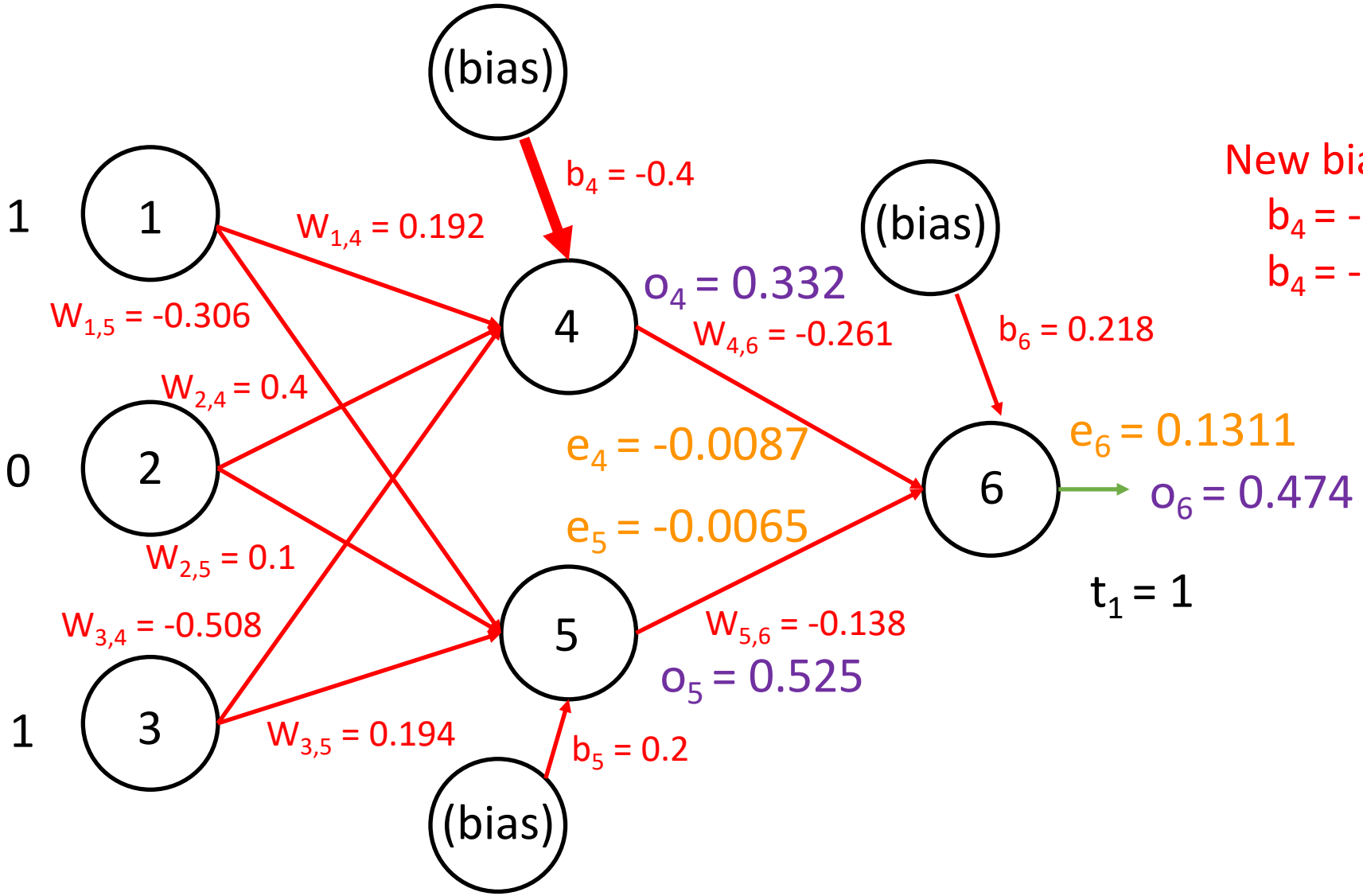

New bias (learning rate = 0.9):

$b_5 = 0.2 + (0.9)(-0.0065)$

$b_5 = 0.194$

$$b_k = b_k + \eta Err_k$$

# Example: Step 3 – Update Weights



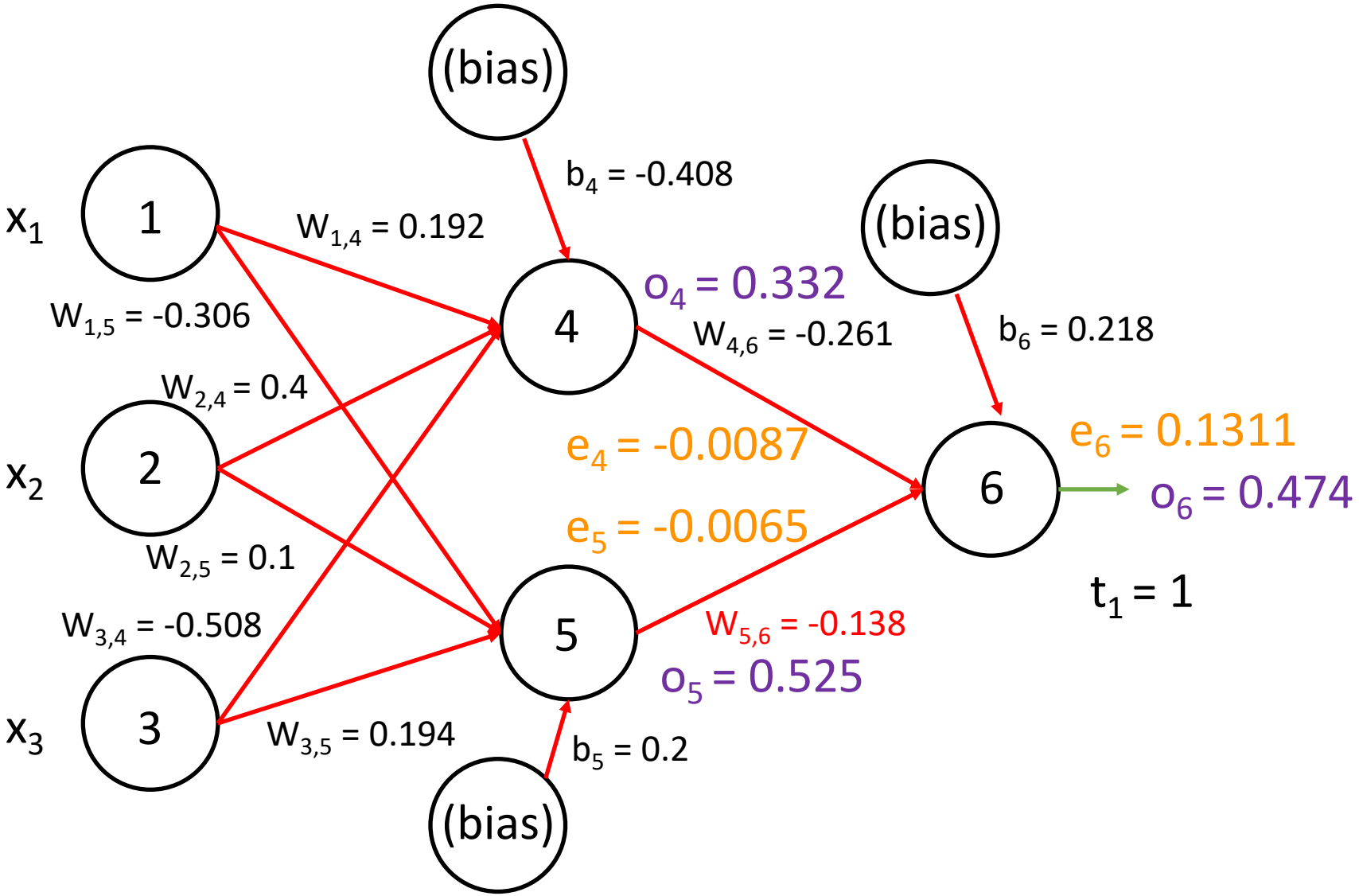

New bias (learning rate = 0.9):

$b_4 = -0.4 + (0.9)(-0.0087)$

$b_4 = -0.408$

$$b_k = b_k + \eta Err_k$$

# Repeat Steps 1-3 With New Examples



# Common Loss Functions

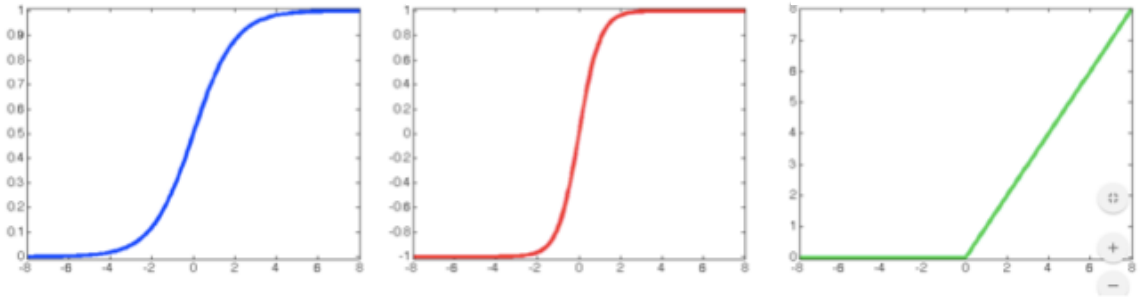
Squared loss: $\sum_k \frac{1}{2}(o_k^{(n)} - t_k^{(n)})^2$

Cross-entropy loss: $-\sum_k t_k^{(n)} \log o_k^{(n)}$

http://www.cs.utoronto.ca/~fidler/teaching/2015/slides/CSC411/10_nn1.pdf

# Other Activation Function's Derivatives

Sigmoid Tanh ReLU



| name | function | derivative |
|---|---|---|
| Sigmoid | $\sigma(z) = \frac{1}{1+\exp(-z)}$ | $\sigma(z) \cdot (1 - \sigma(z))$ |
| Tanh | $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$ | $1/\cosh^2(z)$ |
| ReLU | $\mathrm{ReLU}(z) = \max(0, z)$ | $\begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z \leq 0 \end{cases}$ |

http://www.cs.utoronto.ca/~fidler/teaching/2015/slides/CSC411/10_nn1.pdf

# Stochastic vs Mini-Batch Mode

**Batch mode** Gradient Descent:
Do until satisfied

1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

**Incremental mode** Gradient Descent:
Do until satisfied

- For each training example $d$ in $D$
  1. Compute the gradient $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/mlbook/ch4.pdf

# Today's Topics

- Neural Network: Gradient Calculation with Backpropagation
- Neural Network: Weight Update Methods
- Neural Network: Training
- Lab

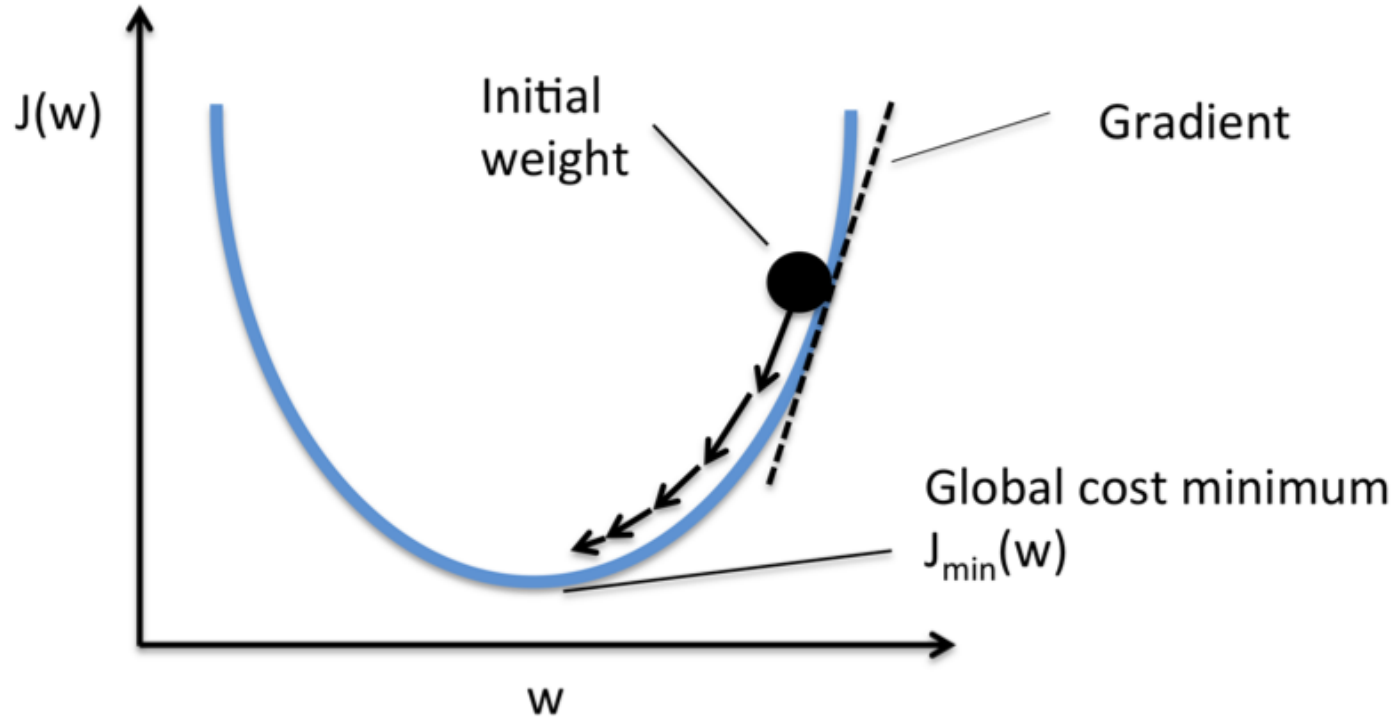
# Motivation: Train Faster!!!

- Can take hours, days, weeks, months, or more to train…

# How to Update Using the Gradient?

- Vanilla Approach: `x += - learning_rate * dx`

Recall: steps get smaller as gradient gets smaller




http://cs231n.github.io/neural-networks-3/#update

Figure from: https://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/

# How to Update Using the Gradient?

- Momentum optimization:
  - Analogy: roll a ball down a hill and it will pick up momentum

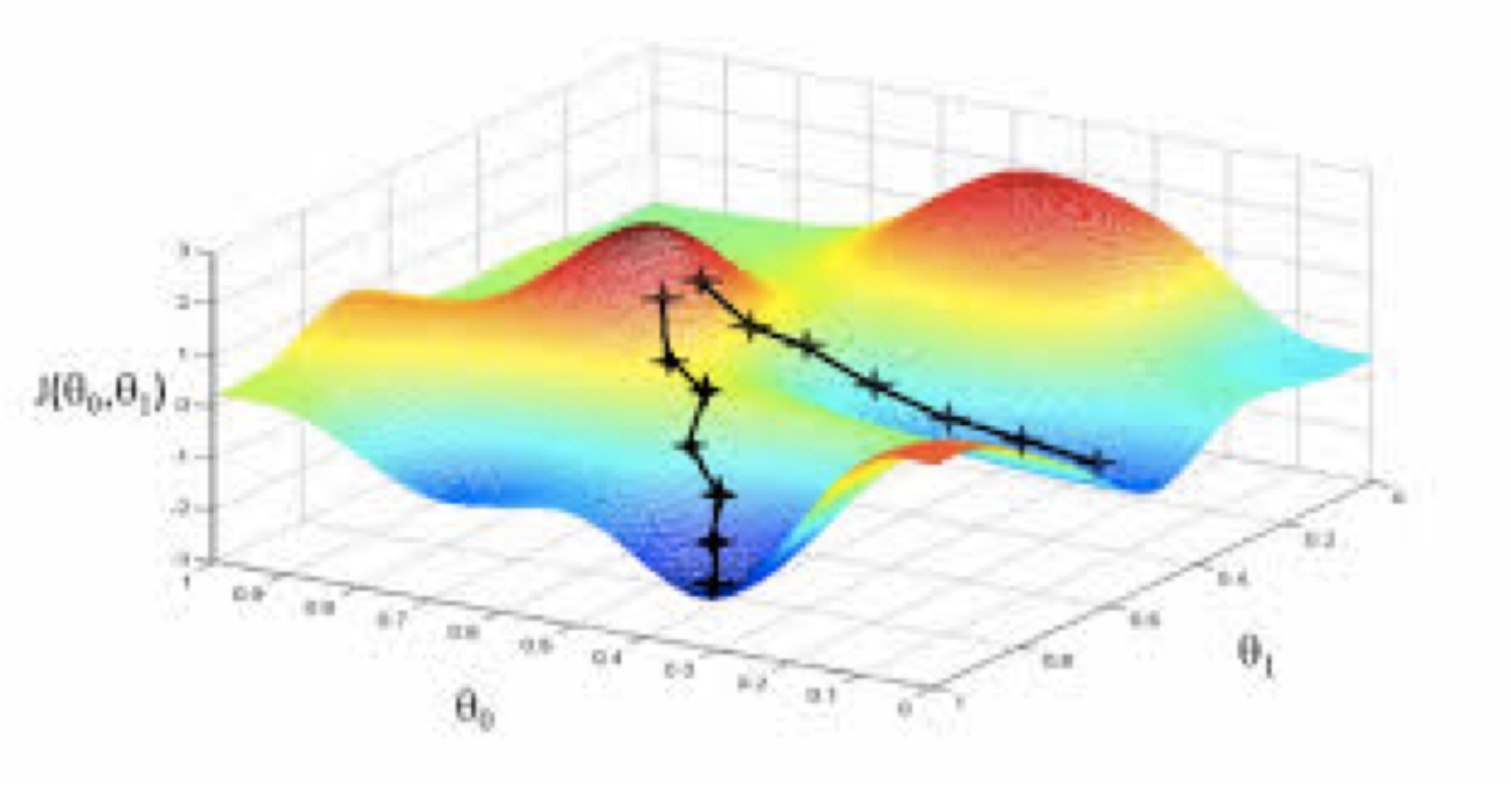



Figure from: https://medium.com/ai-society/hello-gradient-descent-ef74434bdfa5

# How to Update Using the Gradient?

- Momentum optimization:
  - Analogy: roll a ball down a hill and it will pick up momentum

Gradient is used for acceleration rather than speed

Values range from 0 to 1 (larger values mean greater friction)

```
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

http://cs231n.github.io/neural-networks-3/#update

# How to Adapt the Learning Rate?

- **Step decay**:
  - Reduce the learning rate by some factor every few epochs.
- **Exponential decay**
- **1/t decay**

http://cs231n.github.io/neural-networks-3/#update

# How to Adapt the Learning Rate?

- Adapt learning rate per-parameter
- e.g., AdaGrad: decays faster when dimensions are steeper

```
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

- e.g., RMSprop:

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

- e.g., Adam:

```
m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)
x += - learning_rate * m / (np.sqrt(v) + eps)
```

http://cs231n.github.io/neural-networks-3/#update

# Weight Updates: How to Speed Up Training?

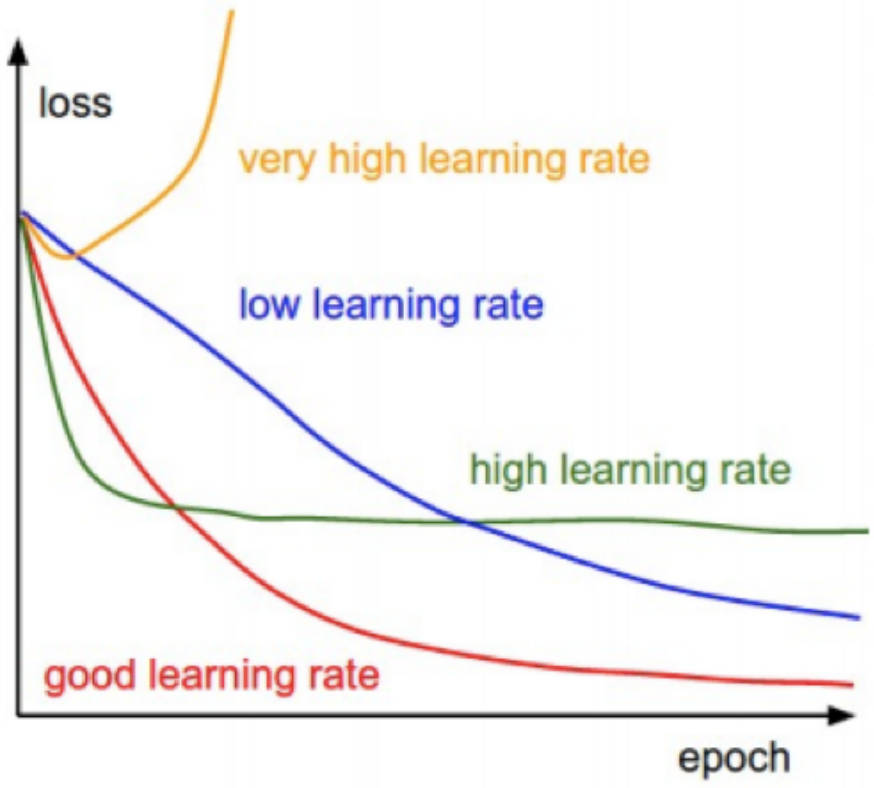
- See demo at http://cs231n.github.io/neural-networks-3/#update

# Today's Topics

- Neural Network: Gradient Calculation with Backpropagation
- Neural Network: Weight Update Methods
- Neural Network: Training
- Lab

# Monitor Loss During Training

- What should happen to the loss function value during training?



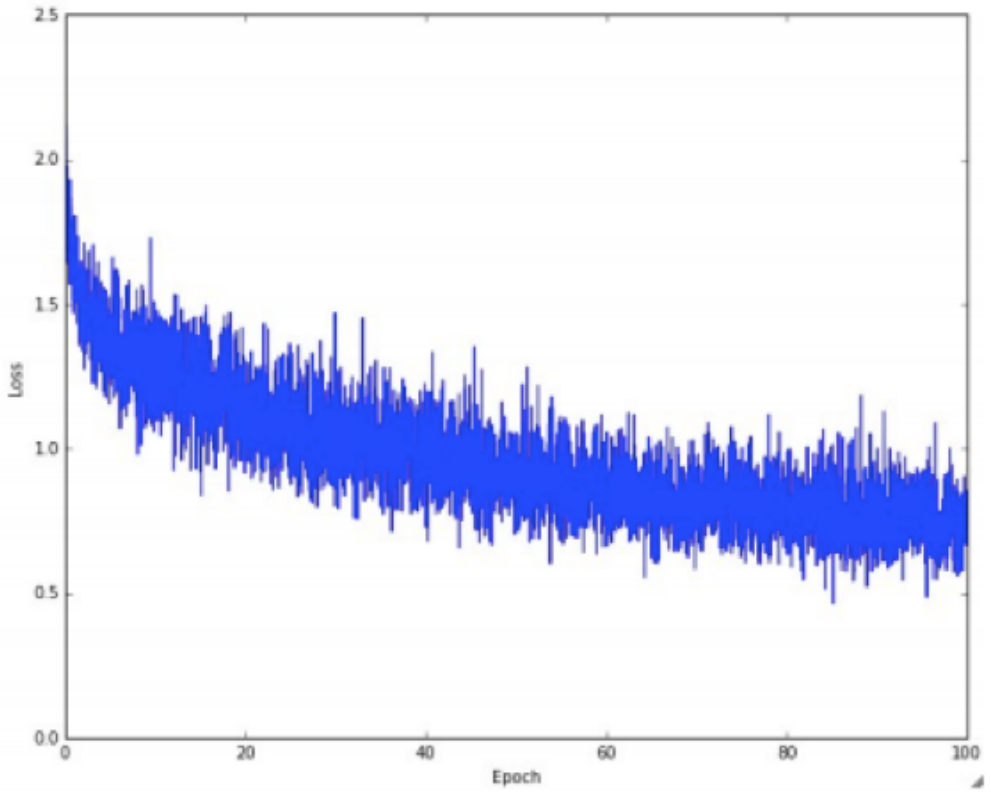
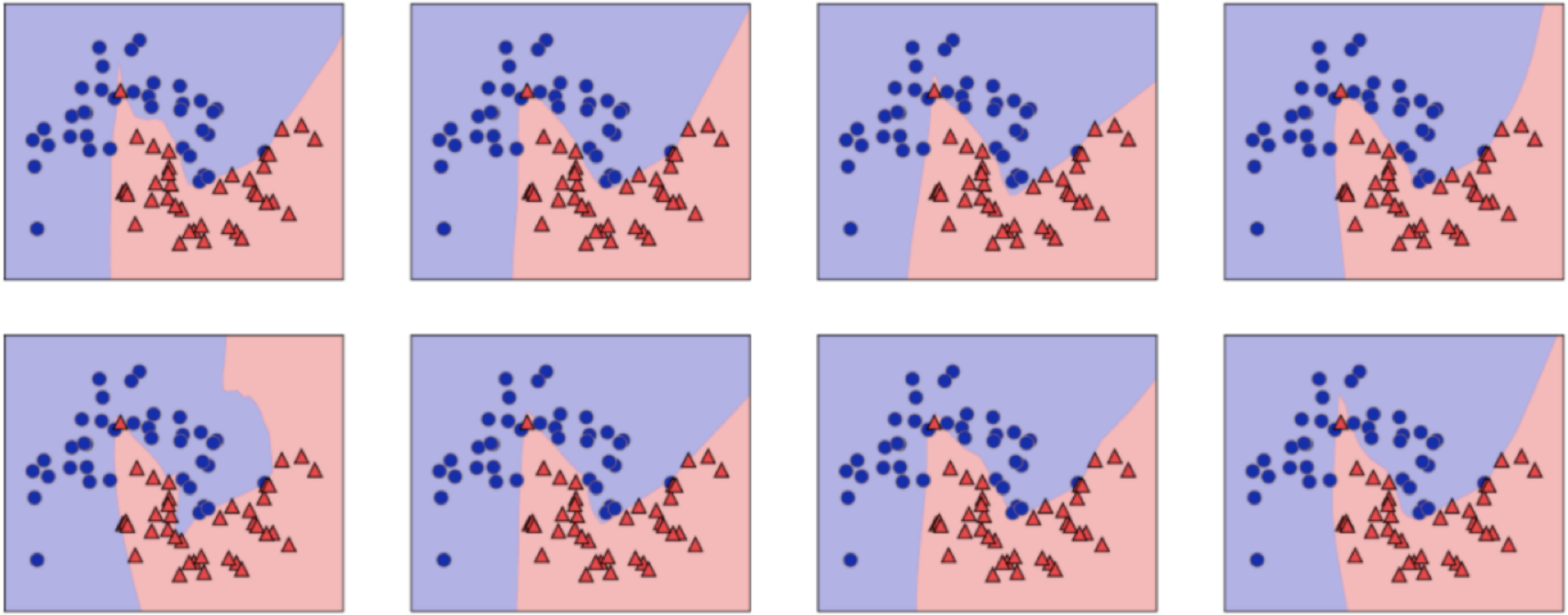



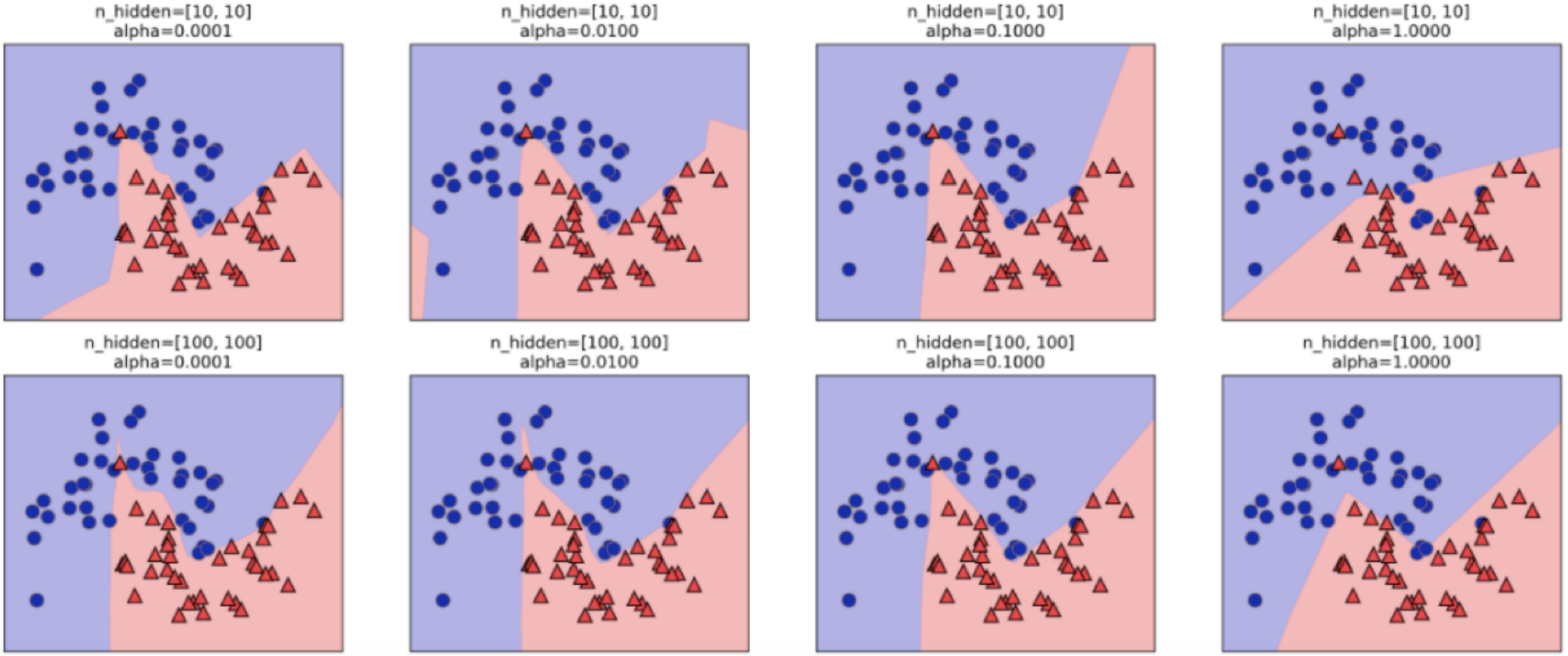

Image credit: http://www.cs.utoronto.ca/~fidler/teaching/2015/slides/CSC411/10_nn1.pdf

# Weight Initialization Matters



https://github.com/amueller/introduction_to_ml_with_python/blob/master/02-supervised-learning.ipynb

# Regularization




https://github.com/amueller/introduction_to_ml_with_python/blob/master/02-supervised-learning.ipynb

# Today’s Topics

- Neural Network: Gradient Calculation with Backpropagation
- Neural Network: Weight Update Methods
- Neural Network: Training
- Lab