# Training Neural Networks with Gradient Descent

**Danna Gurari**
University of Colorado Boulder
Spring 2025


https://dannagurari.colorado.edu/course/neural-networks-and-deep-learning-spring-2025/

# Review

- Last lecture:
  - Motivation for neural networks: need non-linear models
  - Neural networks’ basic ingredients: hidden layers and activation units
  - Neural networks’ support for diverse problems: output units
  - Objective function: what a model should learn

- Assignments (Canvas):
  - Problem set 1 due Tuesday

- Questions?

# Today’s Topics

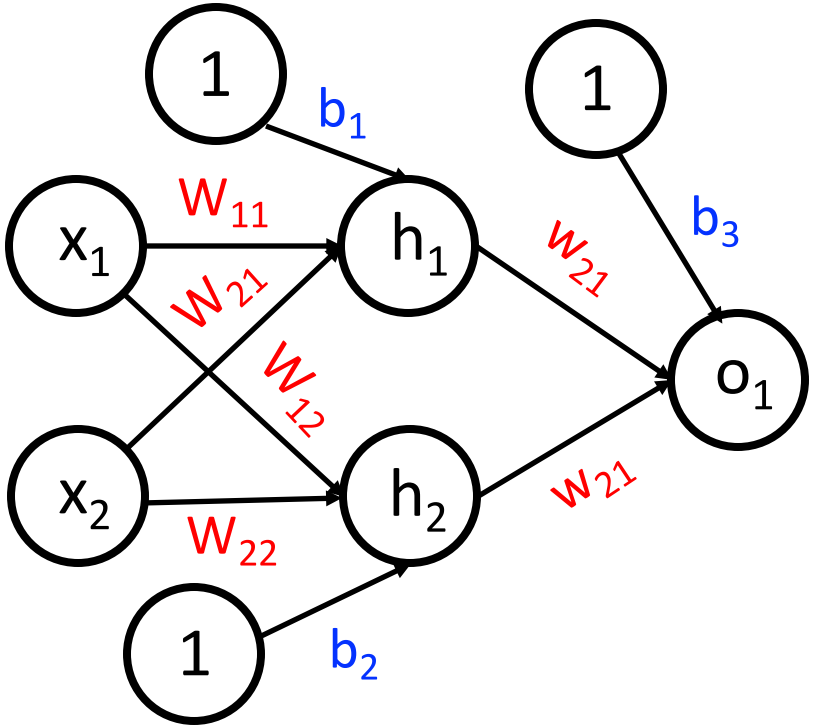
- Gradient descent: how neural networks learn
- Mathematical foundation of gradient descent: derivatives
- Applying gradient descent to train neural networks
- Training example

# Today’s Topics

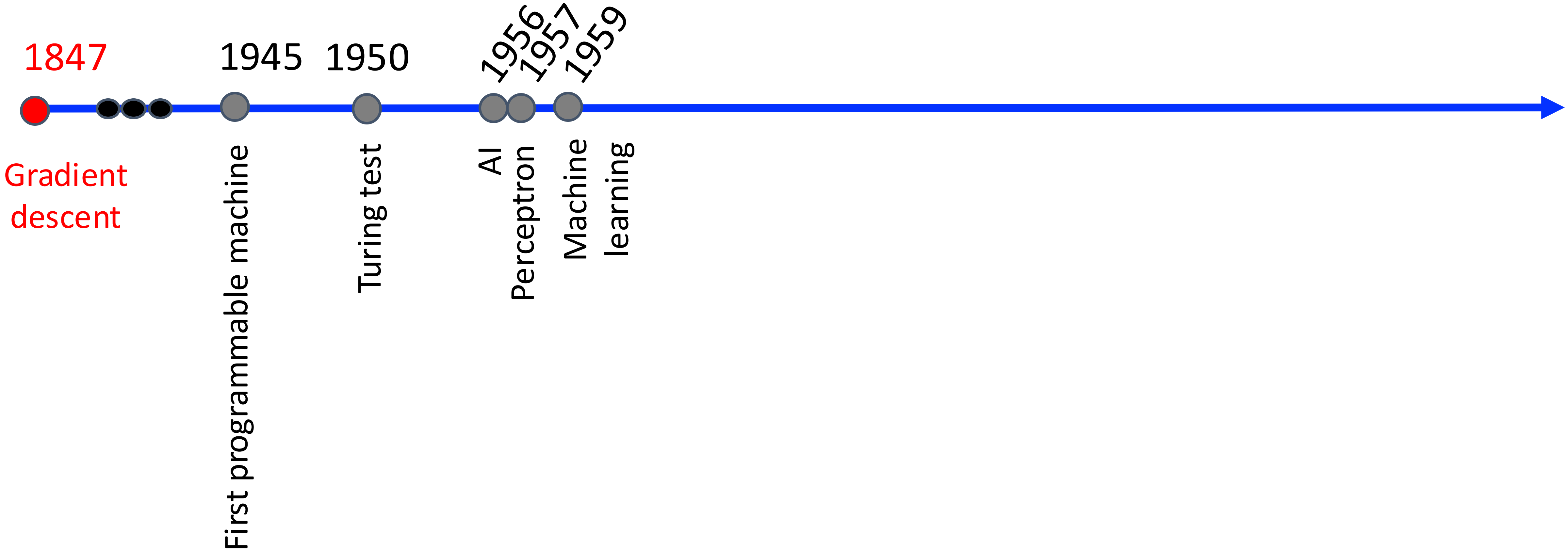
- Gradient descent: how neural networks learn
- Mathematical foundation of gradient descent: derivatives
- Applying gradient descent to train neural networks
- Training example

# Recall Goal: Train Neural Network to Minimize an Objective Function

- Learn model parameters that minimize an objective/loss function using gradient descent; e.g., (weights, biases)



# Approach: Gradient Descent




Louis Augustin Cauchy: Compte Rendu `a l'Acad´emie des Sciences of October 18, 1847

# Gradient Descent: Intuition

- Iteratively searching for better model parameters (e.g., weights and biases) that lead to smaller “losses” in the objective function
- Analogy: hike from mountains to Boulder blind or blindfolded!

# Gradient Descent: Intuition

- Repeat:
  1. Guess
  2. Calculate error
- e.g., learn linear model for converting kilometers to miles when only observing the input “miles” and output “kilometers”


Miles → Kilometers = miles x constant → Kilometers

# Gradient Descent: Intuition

- Repeat:
  1. $\boxed{\text{Guess}}$
  2. Calculate error
- e.g., learn constant multiplier to convert US dollars to Israeli shekels

$10 → Shekels = dollars x constant

# Gradient Descent: Intuition

- Repeat:
  1. Guess
  2. Calculate error
- e.g., learn constant multiplier to convert US dollars to Israeli shekels

$10 → Shekels = dollars x constant → Error = Guess - Correct

# Gradient Descent: Intuition

- Repeat:
  1. $\boxed{\text{Guess}}$
  2. Calculate error
- e.g., learn constant multiplier to convert US dollars to Israeli shekels

$10 → Shekels = dollars x constant

# Gradient Descent: Intuition

- Repeat:
  1. Guess
  2. Calculate error
- e.g., learn constant multiplier to convert US dollars to Israeli shekels

$10 → Shekels = dollars x constant → Error = Guess - Correct

# Gradient Descent: Intuition

- Repeat:
  1. $\boxed{\text{Guess}}$
  2. Calculate error
- e.g., learn constant multiplier to convert US dollars to Israeli shekels

$10 → Shekels = dollars x constant

# Gradient Descent: Intuition

- Repeat:
  1. Guess
  2. Calculate error
- e.g., learn constant multiplier to convert US dollars to Israeli shekels

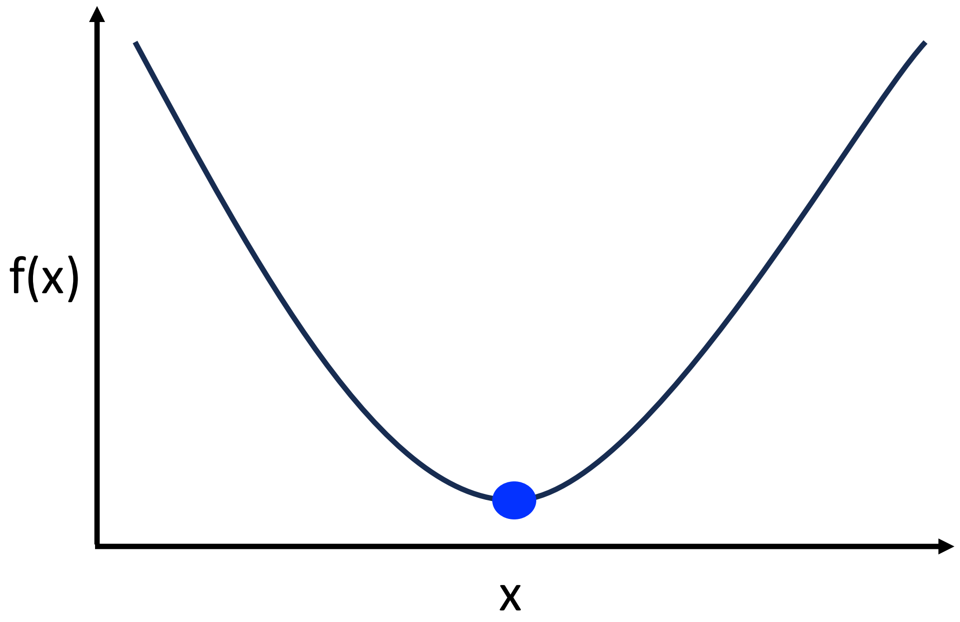
$10 → Shekels = dollars x constant → Error = Guess - Correct

- Idea: iteratively adjust constant (i.e., model parameter) to reduce error

# Gradient Descent: Possible Scenarios

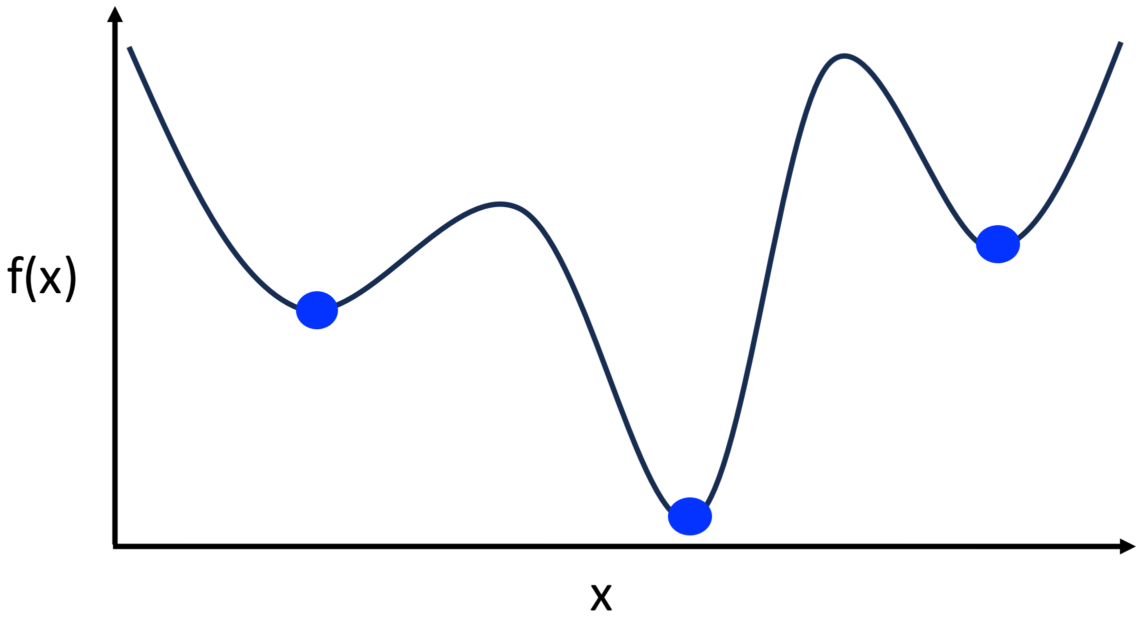
(simple 1-dimensional plots)

Convex Functions (one minimum)




Currency conversion example

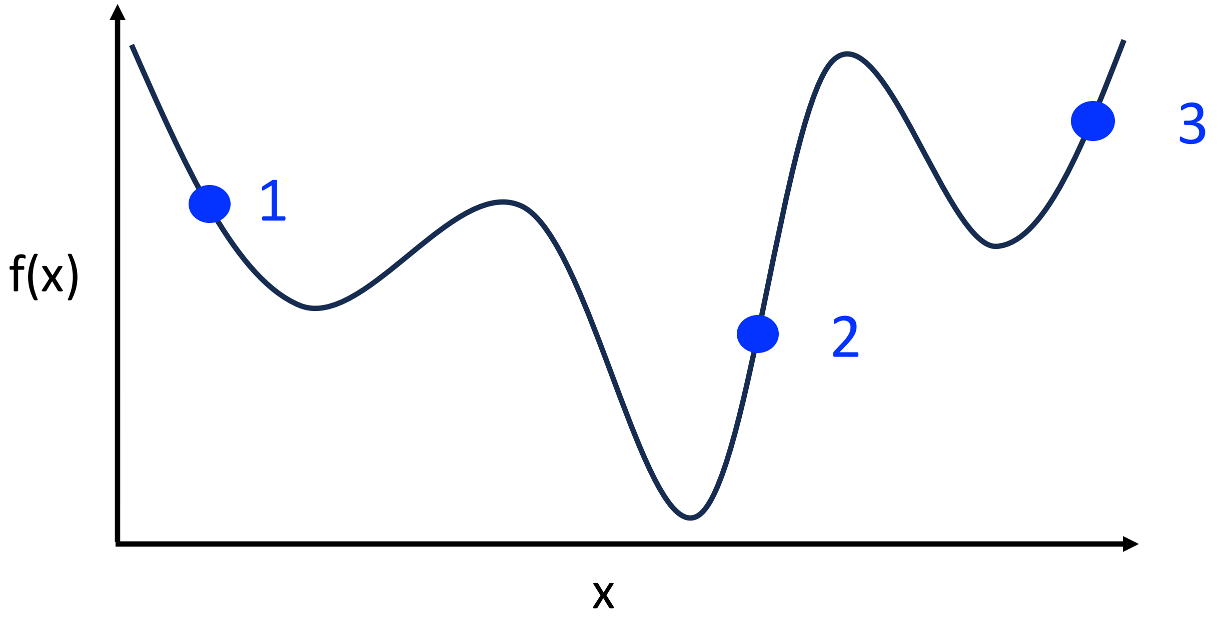
Non-Convex Functions (multiple minima)




Our focus: deep learning

# Gradient Descent: Definitions

- **Gradient:** a vector indicating how a slight change to each function variable in ***x*** increases the output ***f(x)*** (i.e., partial derivatives for multiple variables)
- Recall, a derivative indicates the slope (rise/run) of the function at any point
- **Gradient descent**: to ***minimize*** the function, iteratively step in opposite direction of gradient (i.e., descent rather than ascent)

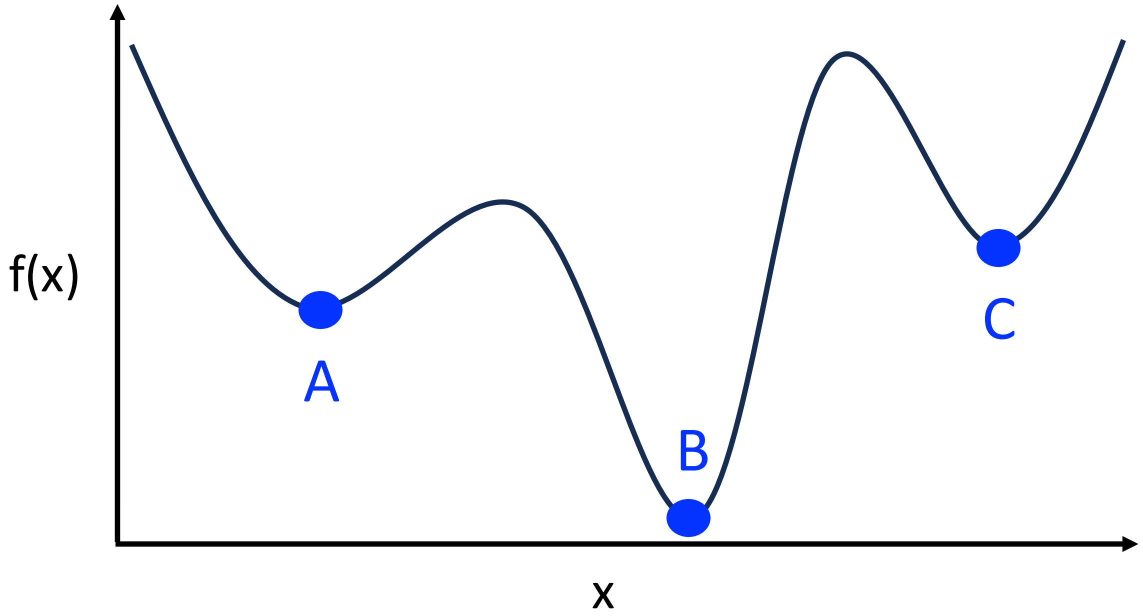



- What does the derivative look like at position 1? (Class volunteer)
- Is position 1’s slope positive or negative?
- From position 1, should follow opposite the gradient direction to minimize the function

- What does the derivative look like at position 2? (Class volunteer)
- Is position 2’s slope positive or negative?
- From position 2, should follow opposite the gradient direction to minimize the function

# Gradient Descent: Definitions

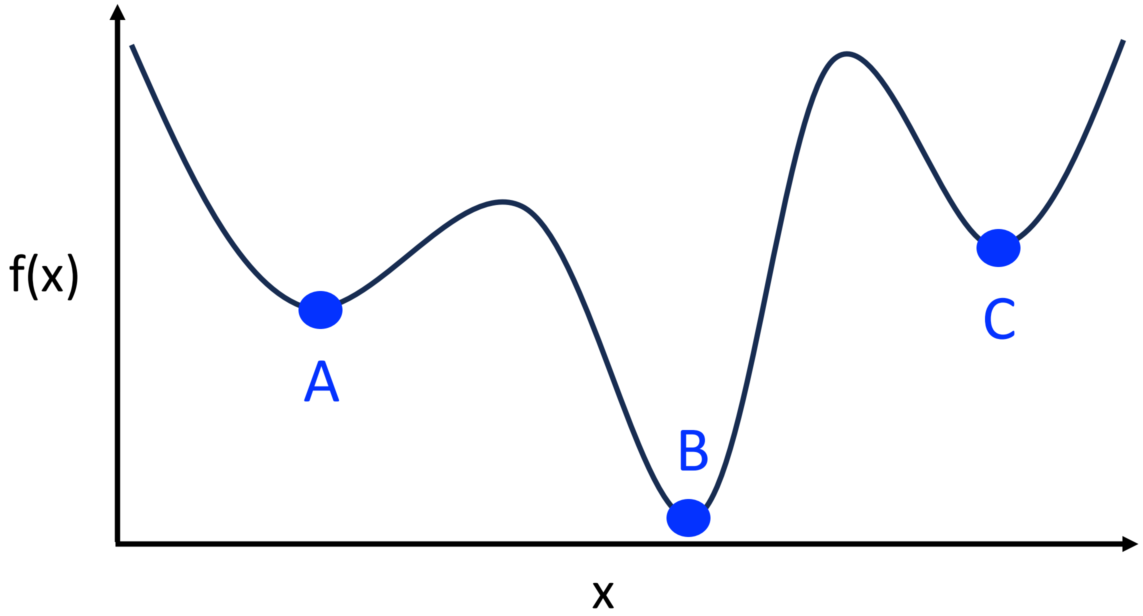
- **Gradient:** a vector indicating how a slight change to each function variable in ***x*** increases the output ***f(x)*** (i.e., partial derivatives for multiple variables)
- Recall, a derivative indicates the slope (rise/run) of the function at any point
- **Gradient descent**: to ***minimize*** the function, iteratively step in opposite direction of gradient (i.e., descent rather than ascent)




Which letter(s) show global minima?

Which letter(s) show local minima?

# Gradient descent can arrive at a local minimum rather than global minimum, a possibility that initially deterred its adoption for neural networks




Global minima: B

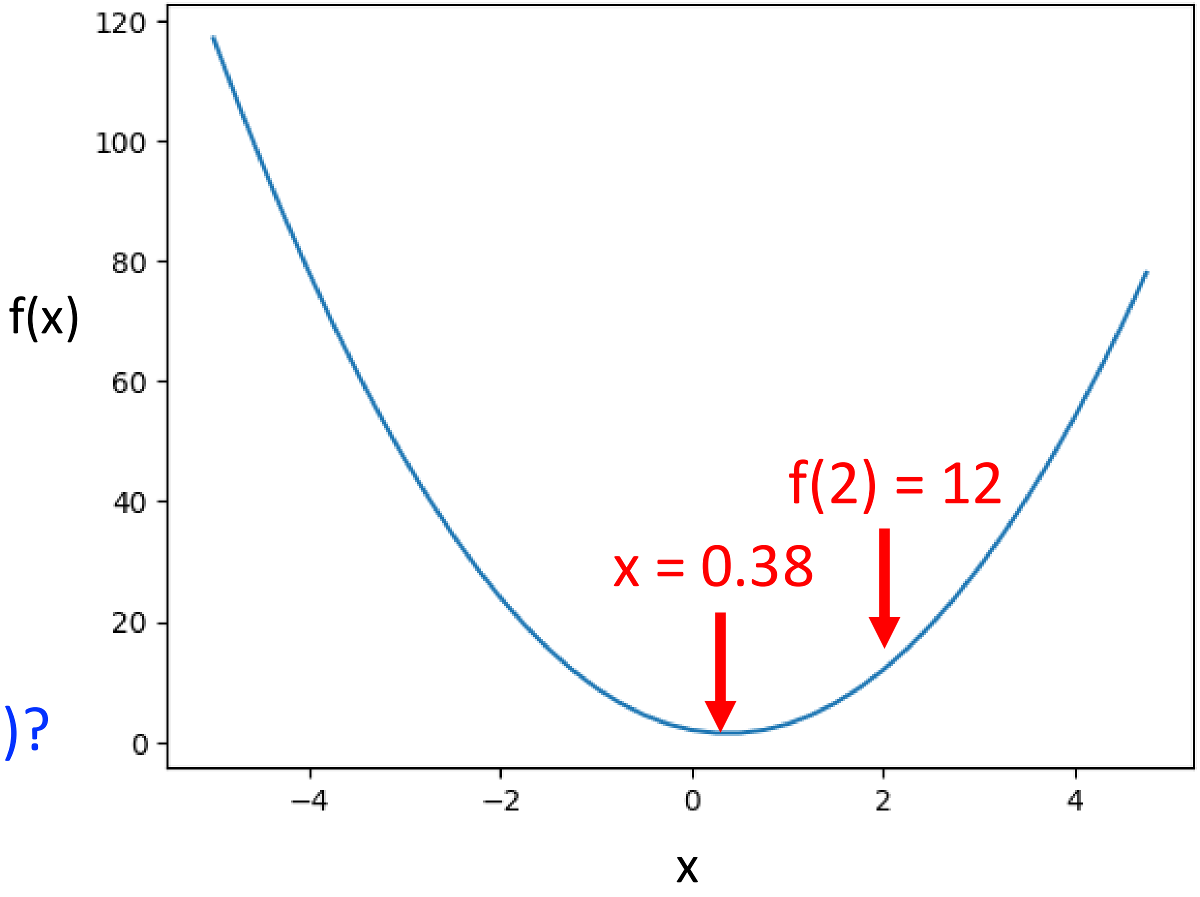
Local minima: A, B, C

# Today’s Topics

- Gradient descent: how neural networks learn
- Mathematical foundation of gradient descent: derivatives
- Applying gradient descent to train neural networks
- Training example

# Reviewing Fundamentals: Toy Example 1

$f(x) = 2 - 3x + 4x^2$




Is this function convex or non-convex?

What is the value of the function at x =2?

What is the value of x at the minimum of f(x)?

# Reviewing Fundamentals: Toy Example 1

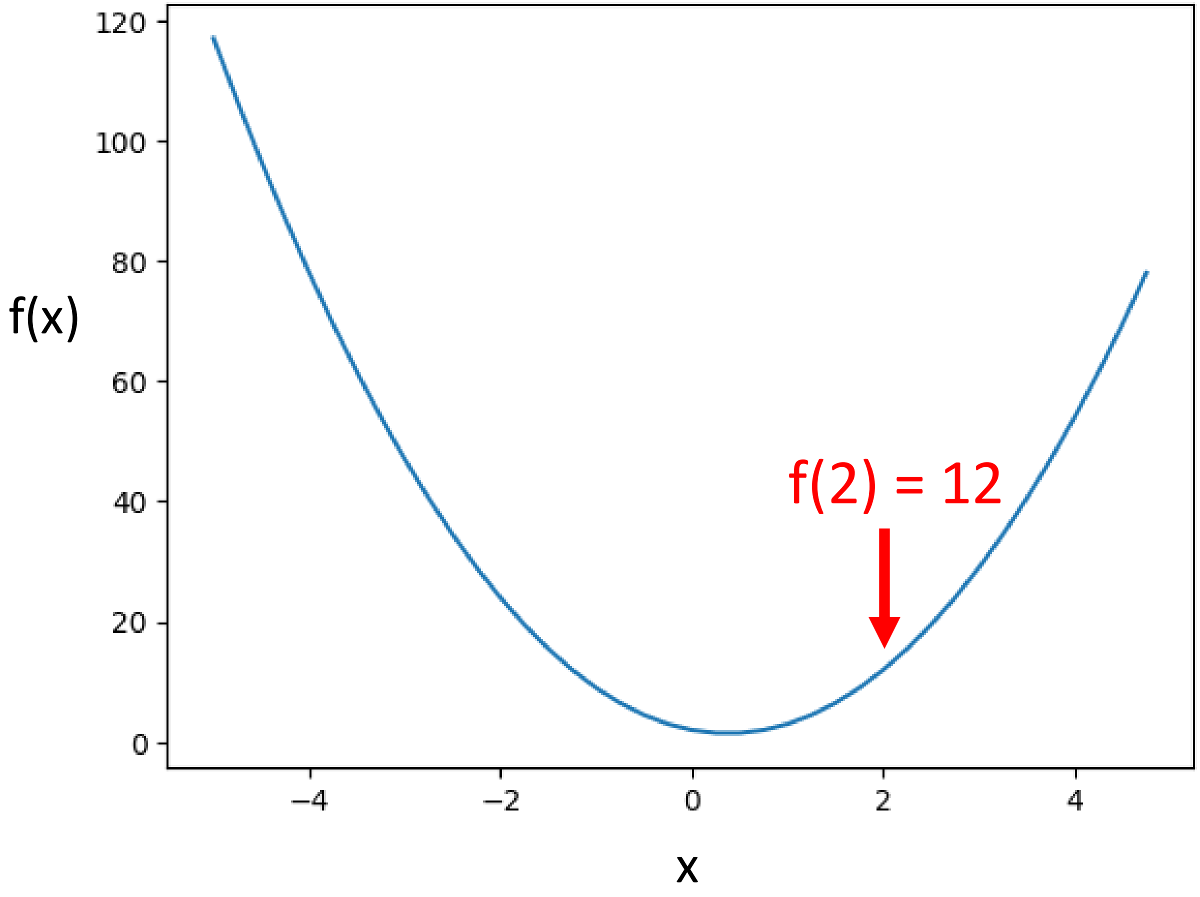
$f(x) = 2 - 3x + 4x^2$

Derivative indicates how a slight change (i.e., h) to every x will change the output:

$$\text{slope} = \frac{\text{rise}}{\text{run}} = \frac{f(x+h) - f(x)}{h}$$

If *h* is slightly positive, at x=2, will *f(x+h)* be greater than or less than 12?

By how much?



# Reviewing Fundamentals: Toy Example 1

$f(x) = 2 - 3x + 4x^2$

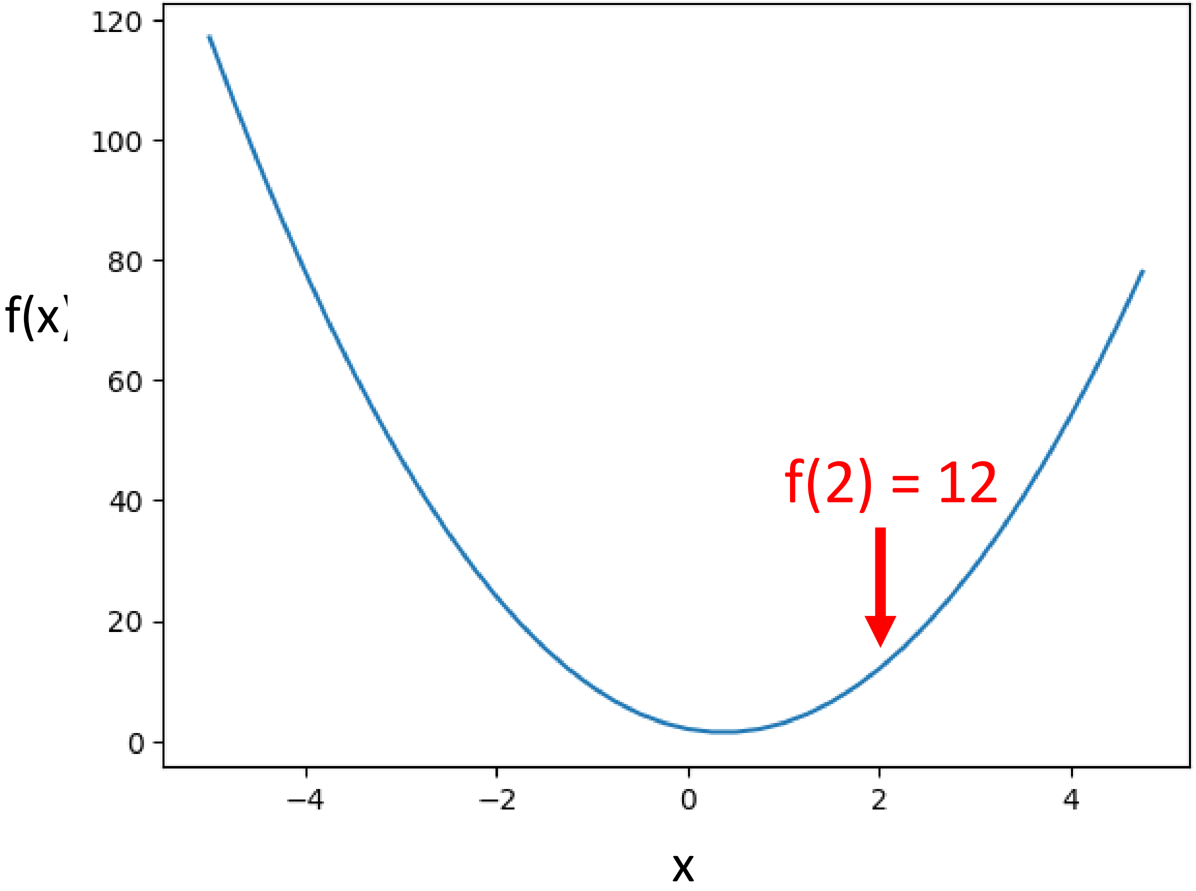
Derivative indicates how a slight change (i.e., h) to every x will change the output:

$$\text{slope} = \frac{\text{rise}}{\text{run}} = \frac{f(x+h) - f(x)}{h}$$

e.g., at $x = 2$

- for $h = 0.01$: $f(x+h) = 12.1304$, slope $= 13.04$

Increasing x by h=0.01 results in an increase of ~13.04 times that amount (13.04 * .01)

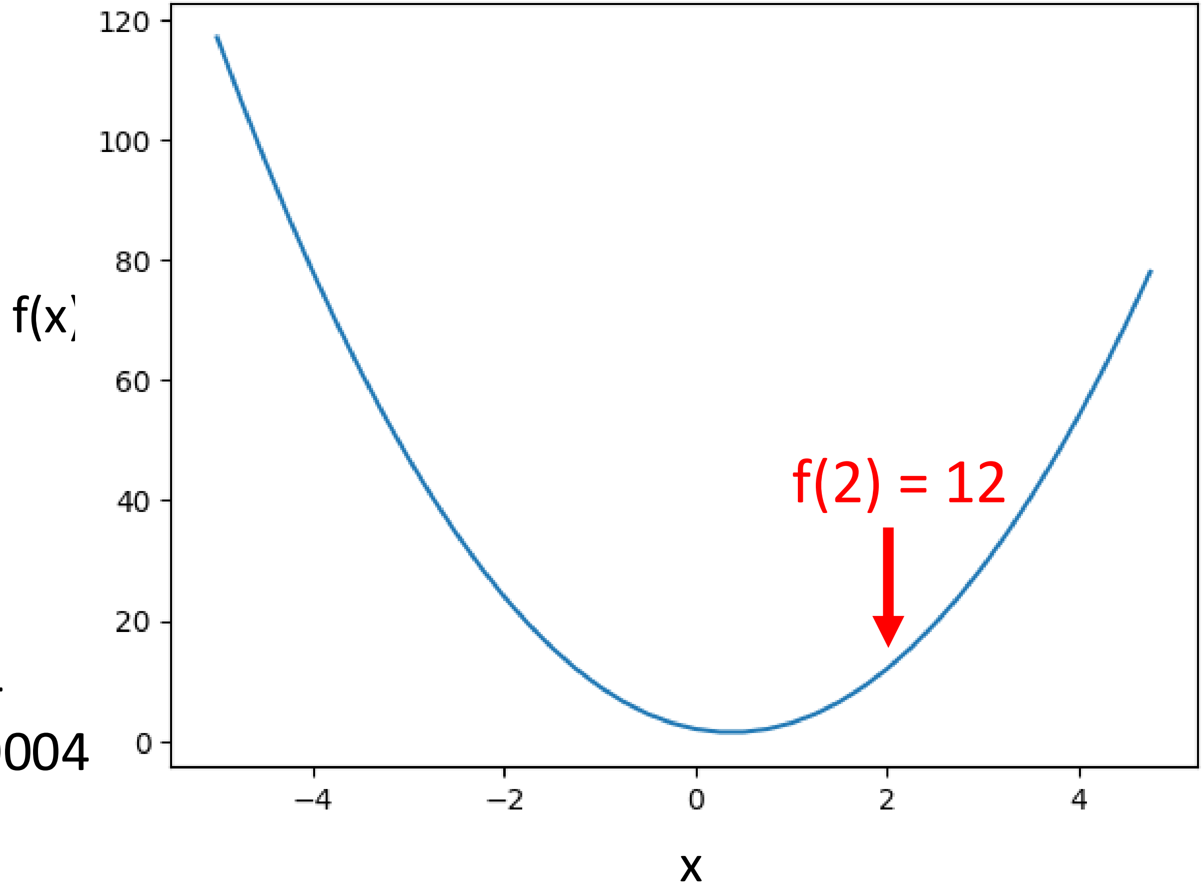


# Reviewing Fundamentals: Toy Example 1

$f(x) = 2 - 3x + 4x^2$

Derivative indicates how a slight change (i.e., h) to every x will change the output:

$$\text{slope} = \frac{\text{rise}}{\text{run}} = \frac{f(x+h) - f(x)}{h}$$




e.g., at x = 2

- for h = 0.01: $f(x+h) = 12.1304$, slope = 13.04
- for h = 0.001: $f(x+h) = 12.013004$ & slope = 13.004
- for h = 0.0001: $f(x+h) = 12.0013004$ & slope = 13.0004

As h gets closer to 0, slope approaches what?

Mathematically, $f'(x) = -3 + 8x$; $f'(2) = -3 + 16 = 13$

# Reviewing Fundamentals: Toy Example 1

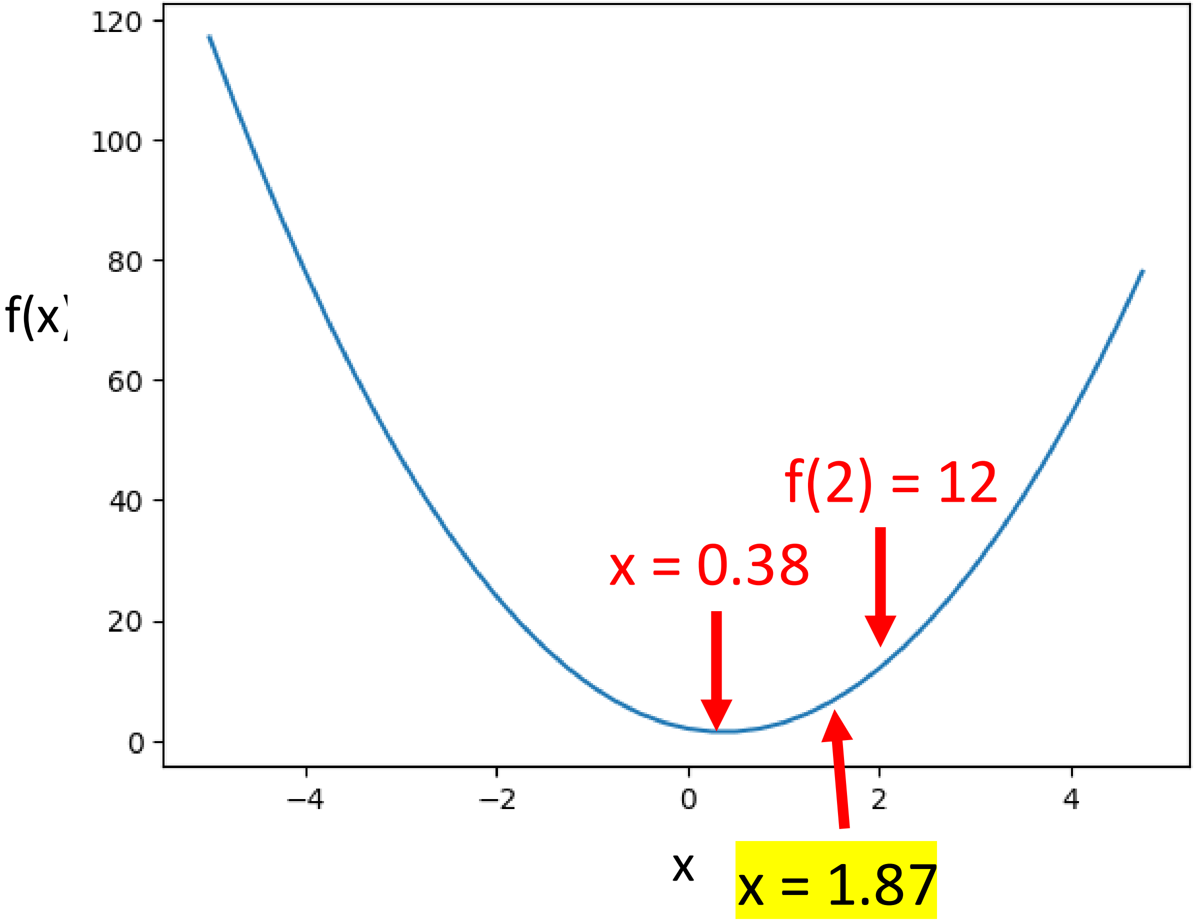
$f(x) = 2 - 3x + 4x^2$

Derivative indicates how a slight change (i.e., h) to every x will change the output:

$$\text{slope} = \frac{\text{rise}}{\text{run}} = \frac{f(x+h) - f(x)}{h}$$

e.g., at x = 2 and df/dx = 13
Approach minimum point by stepping in opposite direction of derivative (e.g., learning rate of 0.01); what should be new value for x?

$2 - (0.01 * 13) = 1.87$

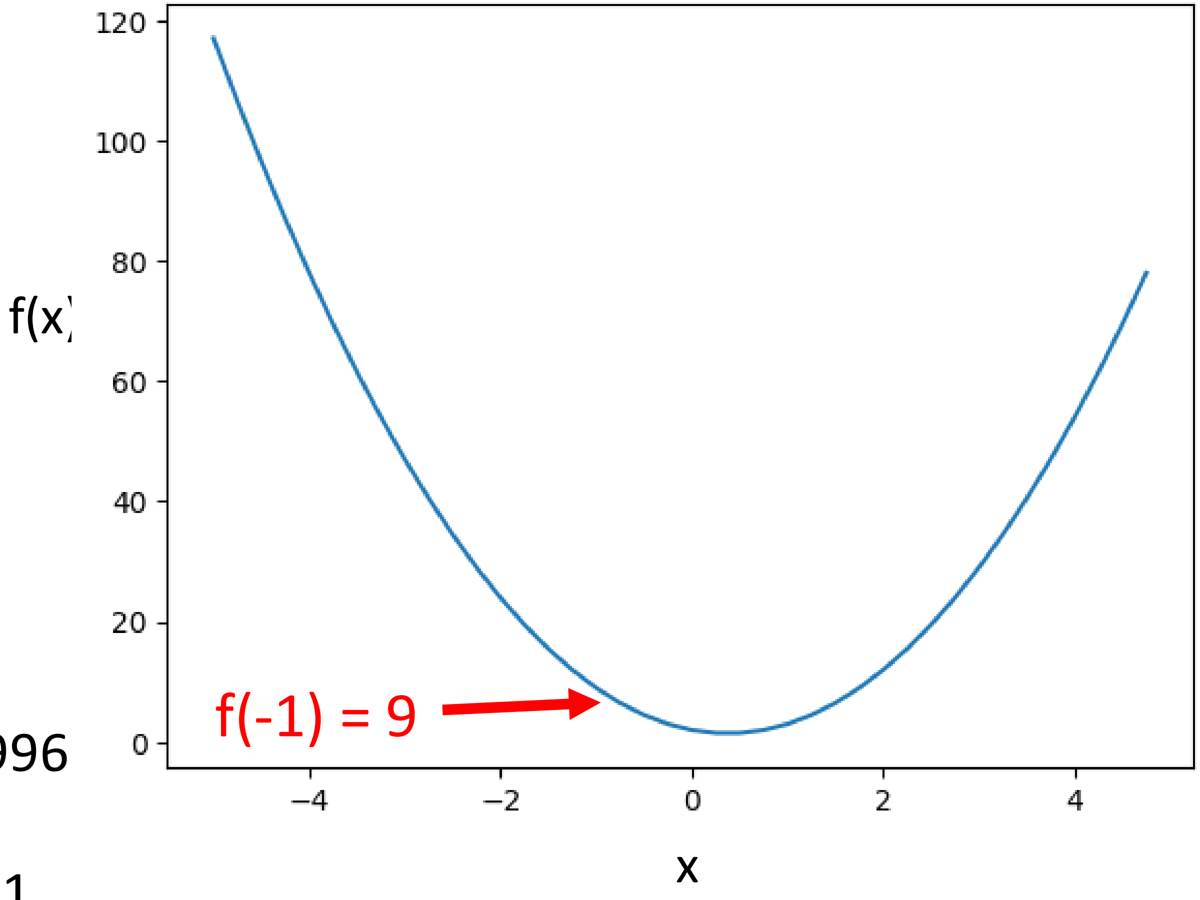


# Reviewing Fundamentals: Toy Example 1

$f(x) = 2 - 3x + 4x^2$

Derivative indicates how a slight change (i.e., h) to every x will change the output:

$$\text{slope} = \frac{\text{rise}}{\text{run}} = \frac{f(x+h) - f(x)}{h}$$




e.g., at x = -1
- for h = 0.01: f(x+h) = 8.8904, slope = -10.96
- for h = 0.001: f(x+h) = 8.989004, slope = -10.996
- for h = 0.0001: f(x+h) = 8.99890004, slope = -10.9996

As h gets closer to 0, slope approaches what?

Mathematically, $f'(x) = -3 + 8x$; $f'(-1) = -3 + 8*-1 = -11$

# Reviewing Fundamentals: Toy Example 1

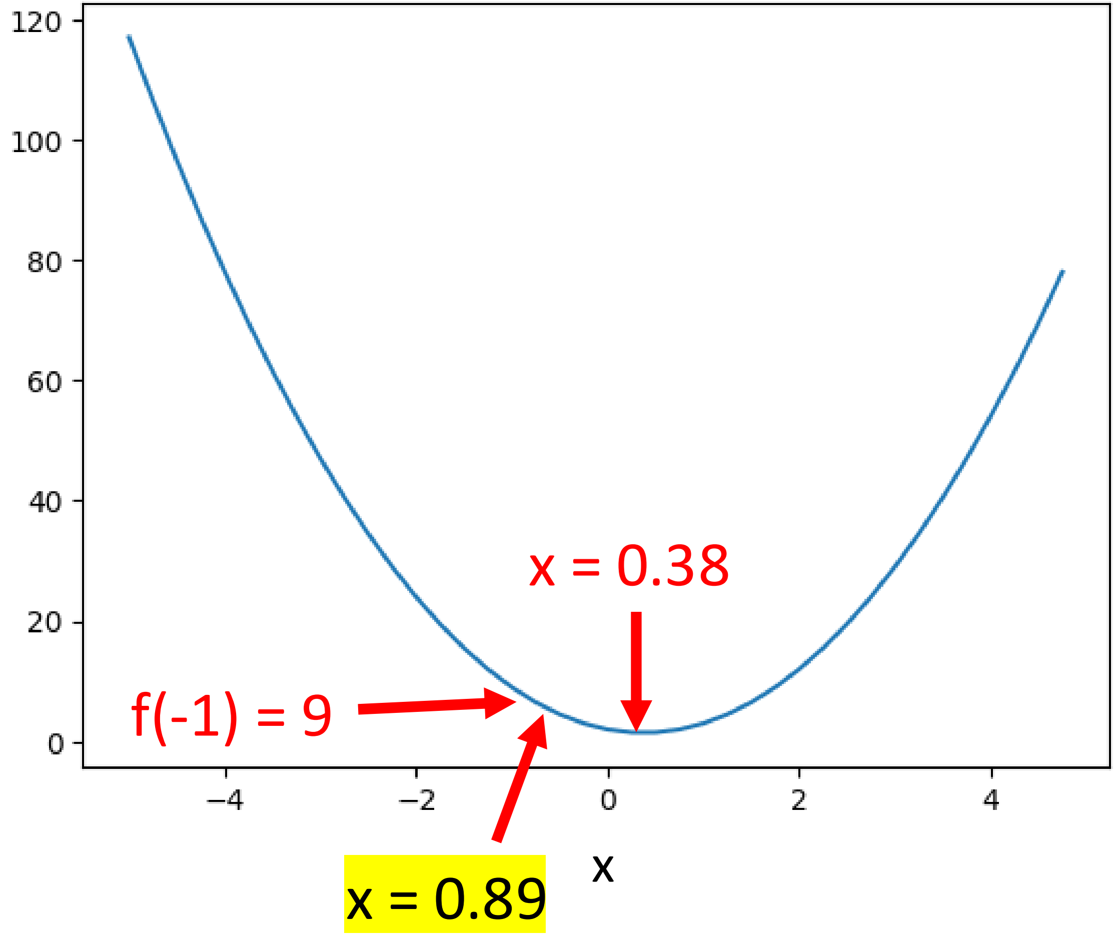
$f(x) = 2 - 3x + 4x^2$

Derivative indicates how a slight change (i.e., h) to every x will change the output:

$$\text{slope} = \frac{\text{rise}}{\text{run}} = \frac{f(x+h) - f(x)}{h}$$

e.g., at $x = -1$ and $df/dx = -11$
Approach minimum point by stepping in opposite direction of derivative (e.g., learning rate of 0.01); what should be new value for x?

$-1 - (0.01 * -11) = -0.89$



# Reviewing Fundamentals: Toy Example 2

f(**x**) = a + bc; where **x** is a vector with all parameters

Gradient indicates how a slight change (i.e., h) to any **input** will change the output:

$$\text{slope} = \frac{\text{rise}}{\text{run}} = \frac{f(\mathbf{x} + h) - f(\mathbf{x})}{h}$$

What is the value of the function when a = 1, b = -2, and c = 3?

If we only change *a* with a slightly positive *h*, will f(**x**+h) be greater than or less than -5?

By how much?

# Reviewing Fundamentals: Toy Example 2

f(**x**) = a + bc; where **x** is a vector with all parameters

Gradient indicates how a slight change (i.e., h) to any **input** will change the output:

$$\text{slope} = \frac{\text{rise}}{\text{run}} = \frac{f(\mathbf{x} + h) - f(\mathbf{x})}{h}$$

e.g., at a = 1, b = -2, and c = 3 when only changing the value of a

- for h = 0.01: f(**x**+h) = (1+.01) + -2*3 = -4.99, slope = 1.000
- for h = 0.001: f(**x**+h) = (1+.001) + -2*3 = -4.999, slope = 1.000

As h gets closer to 0, slope approaches 1

Mathematically, df/da = 1

# Reviewing Fundamentals: Toy Example 2

f(**x**) = a + bc; where **x** is a vector with all parameters

Gradient indicates how a slight change (i.e., h) to any **input** will change the output:

$$\text{slope} = \frac{\text{rise}}{\text{run}} = \frac{f(\mathbf{x} + h) - f(\mathbf{x})}{h}$$

Recall the value of the function when a = 1, b = -2, and c = 3 is -5

If we change only *b* with a slightly positive *h*, will f(**x**+h) be greater than or less than -5?

By how much?

# Reviewing Fundamentals: Toy Example 2

f(**x**) = a + bc; where **x** is a vector with all parameters

Gradient indicates how a slight change (i.e., h) to any **input** will change the output:

$$\text{slope} = \frac{\text{rise}}{\text{run}} = \frac{f(\mathbf{x} + h) - f(\mathbf{x})}{h}$$

e.g., at a = 1, b = -2, and c = 3 when only changing the value of b (rounding to 3 decimals)
- for h = 0.01: f(**x**+h) = 1 + (-2 +.01)*3 = -4.970, slope = 3.000
- for h = 0.001: f(**x**+h) = 1 + (-2 +.001)*3 = -4.997, slope = 3.000

As h gets closer to 0, slope approaches 3

Mathematically, df/db = c (which we set to 3)

# Reviewing Fundamentals: Toy Example 2

f(**x**) = a + bc; where **x** is a vector with all parameters

Gradient indicates how a slight change (i.e., h) to any **input** will change the output:

$$\text{slope} = \frac{\text{rise}}{\text{run}} = \frac{f(\mathbf{x} + h) - f(\mathbf{x})}{h}$$

Recall the value of the function when a = 1, b = -2, and c = 3 is -5

If we change only *c* with a slightly positive *h*, will f(**x**+h) be greater than or less than -5?

By how much?

# Reviewing Fundamentals: Toy Example 2

f(**x**) = a + bc; where **x** is a vector with all parameters

Gradient indicates how a slight change (i.e., h) to any **input** will change the output:

$$\text{slope} = \frac{\text{rise}}{\text{run}} = \frac{f(\mathbf{x} + h) - f(\mathbf{x})}{h}$$

e.g., at  a = 1, b = -2, and c = 3 when only changing the value of c (rounding to 3 decimals)
- for h = 0.01: f(**x**+h) = 1 + -2*(3 +.01) = -5. 02, slope = -2.000
- for h = 0.001: f(**x**+h) = 1 + -2*(3 +.001) = -5.002, slope = -2.000

As h gets closer to 0, slope approaches -2

Mathematically, df/dc = b (which we set to -2)

# Reviewing Fundamentals: Toy Example 2

f(**x**) = a + bc; where **x** is a vector with all parameters

Gradient indicates how a slight change (i.e., h) to any **input** will change the output:

$$\text{slope} = \frac{\text{rise}}{\text{run}} = \frac{f(\mathbf{x} + h) - f(\mathbf{x})}{h}$$

In summary, gradient for any input can be computed with these equations:

df/da = 1; df/db = c; df/dc =b

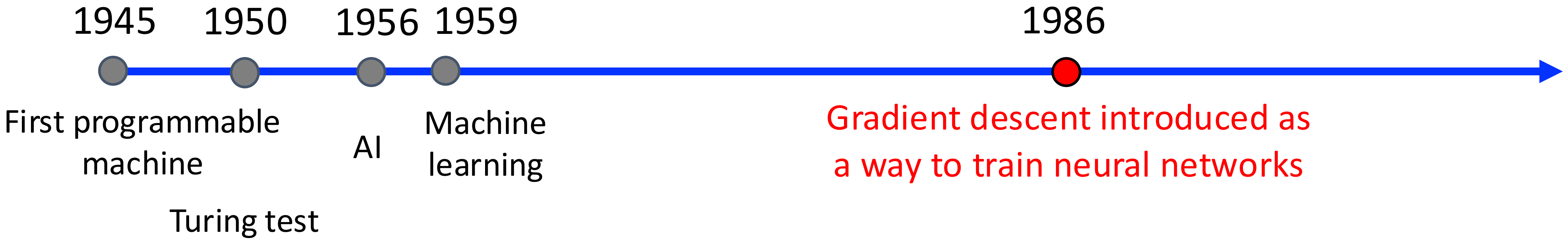
Can minimize objective function by stepping (e.g., learning rate of 0.01) in opposite direction of derivative; e.g., at a = 1, b = -2, and c = 3

a - 0.01*1 = 0.99; b - 0.01*c = -2.03; c - 0.01*b = 3.02 f(x) = -5.14

# Today’s Topics

- Gradient descent: how neural networks learn
- Mathematical foundation of gradient descent: derivatives
- Applying gradient descent to train neural networks
- Training example

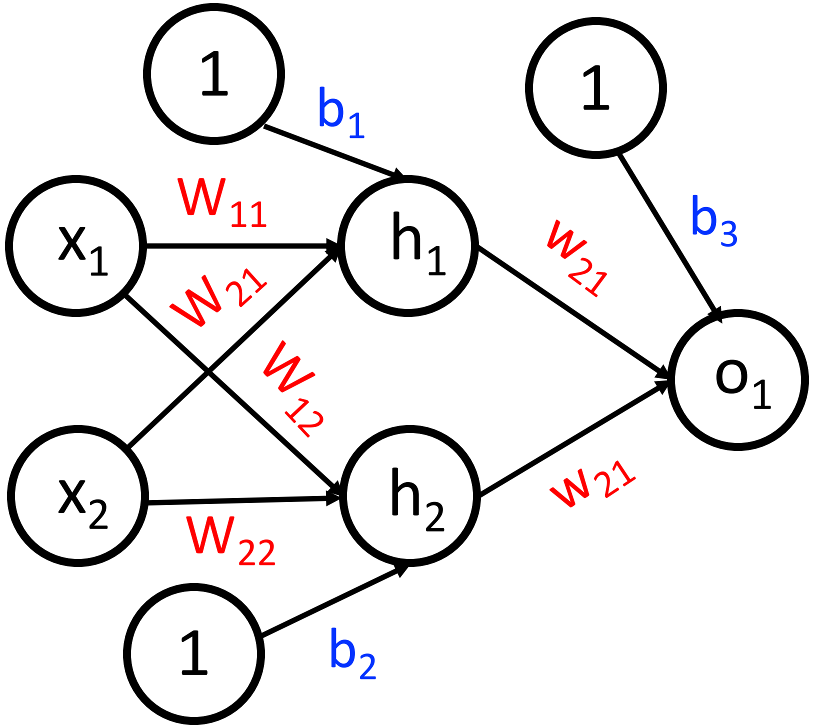
# Training Neural Networks

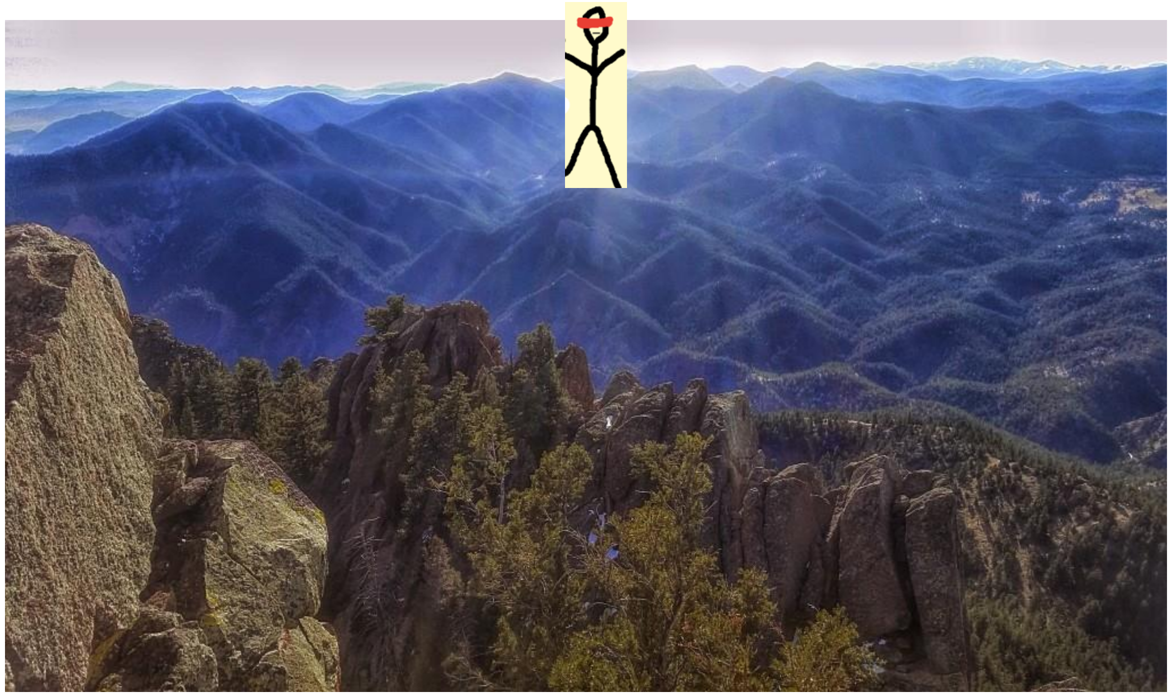



D. Rulhart, G. Hinton, and R. Williams, Learning Internal Representations by Error Propagation, 1986.

# Training Neural Networks

- Descend the loss function with gradient descent; e.g., (weights, biases)



# Gradient Descent: Implementation

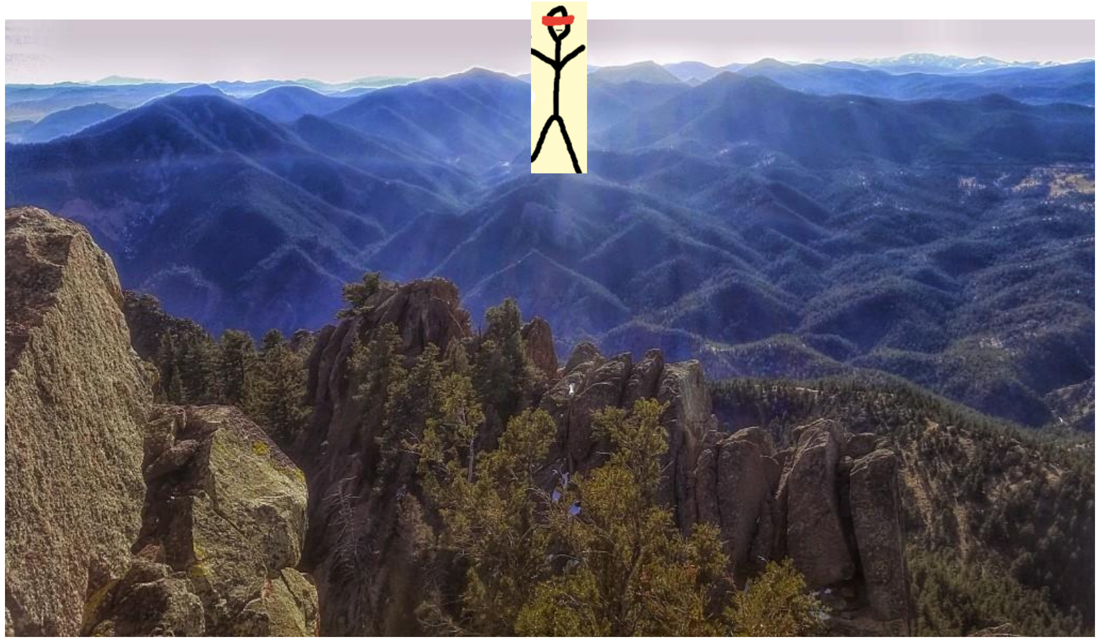
- Repeat until stopping criterion met:

1. Propagate training data through model to make predictions
2. Measure error of the model's predictions on training data using a loss function
3. $\boxed{\text{Calculate gradients to determine how each model parameter contributed to model error}}$
4. Update each parameter using calculated gradients

Breakthrough in 1986: backpropagation used to compute the gradient

# Breakthrough: Backpropagation

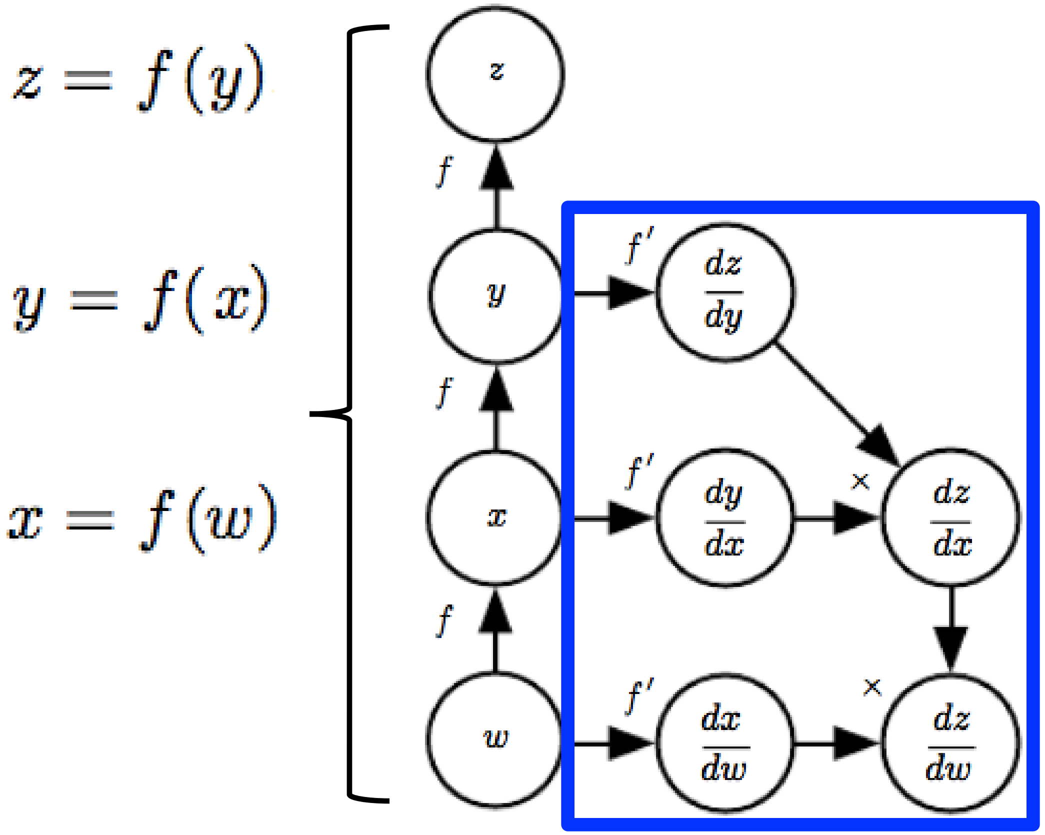
- **Key observation**: loss function is a mathematical function, that takes as input training data and model parameters (e.g., weights, biases)



Using backpropagation, we can know how to tweak all of a function's variables to minimize an error/loss function

# Breakthrough: Backpropagation

- **Key observation**: loss function is a mathematical function connected in a chain
- **What we want:** partial derivatives for function variables to inform how to minimize the loss function
- **Backpropagation**: achieves this with chain rule of calculus, which relies only on local derivatives; e.g.,




Indicates how much a small change to each variable changes the function's output

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

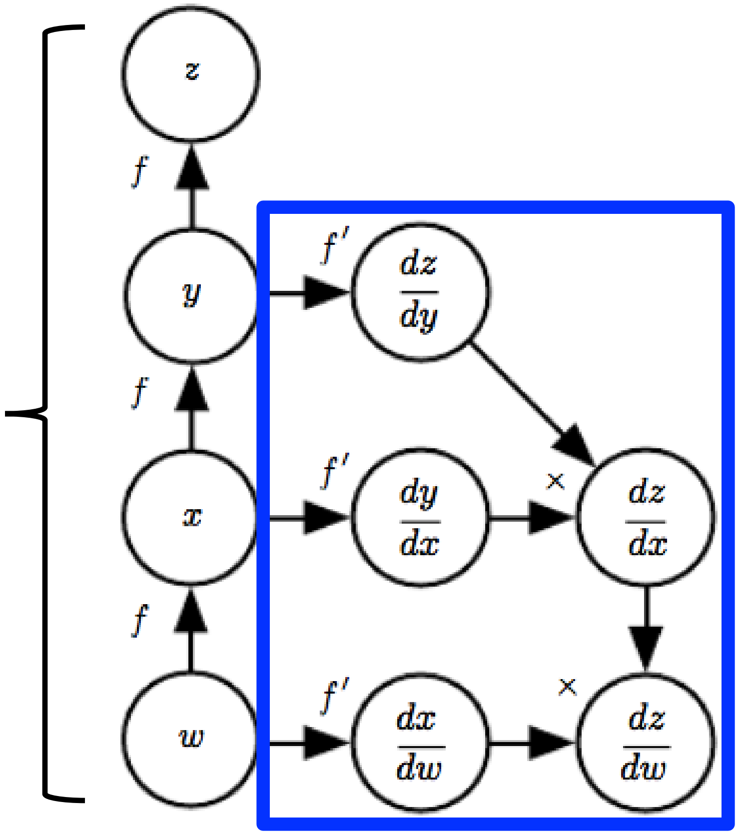
# Breakthrough: Backpropagation

- **Key observation**: loss function is a mathematical function connected in a chain
- **What we want:** partial derivatives for function variables to inform how to minimize the loss function
- **Backpropagation**: achieves this with chain rule of calculus, which relies only on local derivatives; e.g.,

$$z = f(y)$$

$$y = f(x)$$

$$x = f(w)$$




What is *dz/dx*? $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$ (how changes in x will change the output, z)

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

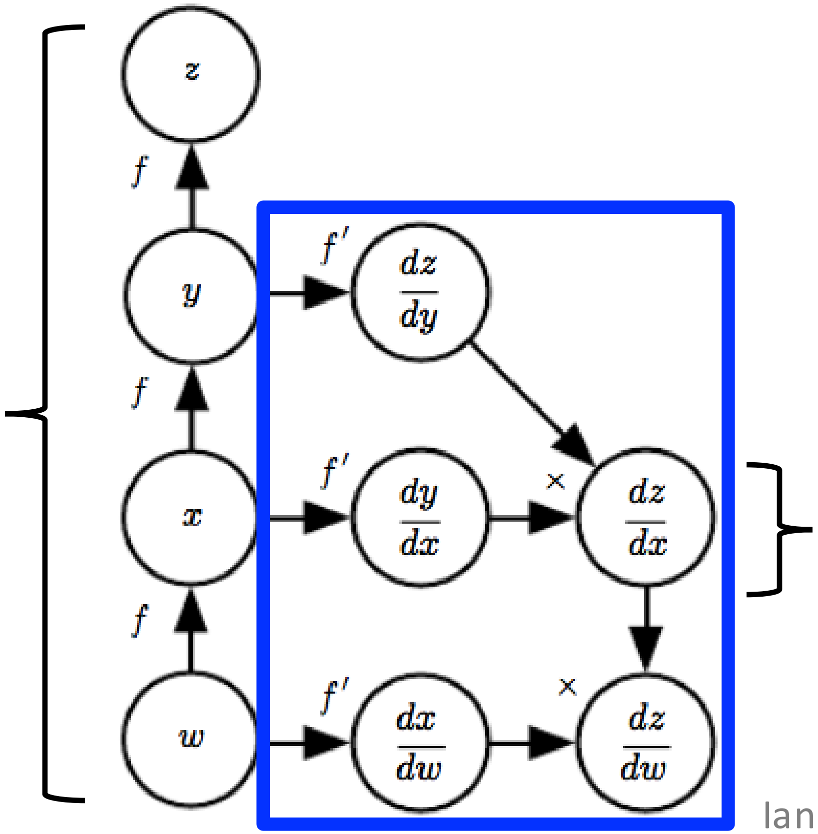
# Breakthrough: Backpropagation

- **Key observation**: loss function is a mathematical function connected in a chain
- **What we want:** partial derivatives for function variables to inform how to minimize the loss function
- **Backpropagation**: achieves this with chain rule of calculus, which relies only on local derivatives; e.g.,

$$z = f(y)$$

$$y = f(x)$$

$$x = f(w)$$




What is *dz/dx*?

Intuitive example: how much faster is my husband compared to my daughter? (*dz/dx*)

*dz/dy:* husband is 2x faster than son

*dy/dx:* son is 3x faster than daughter

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

# Gradient Descent: Implementation

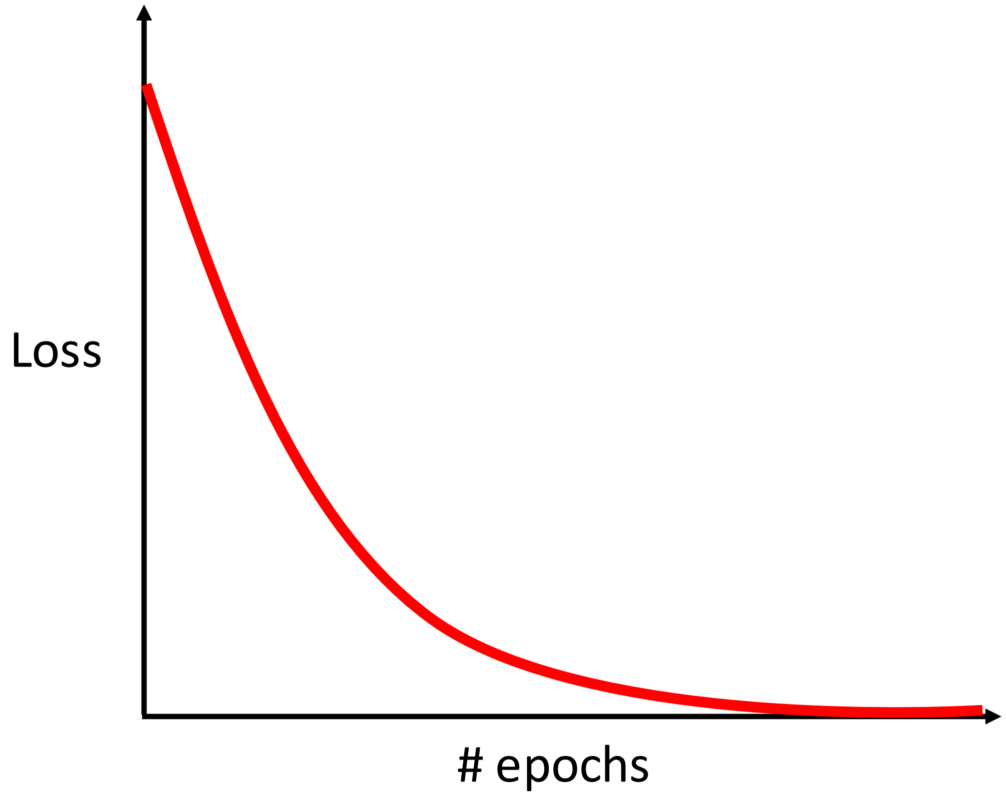
- Repeat until stopping criterion met:

1. Propagate training data through model to make predictions
2. Measure error of the model's predictions on training data using a loss function
3. Calculate gradients to determine how each model parameter contributed to model error
4. Update each parameter using calculated gradients

We babysit this step to see how learning is going

# Loss Curves Signal How Well Training is Going

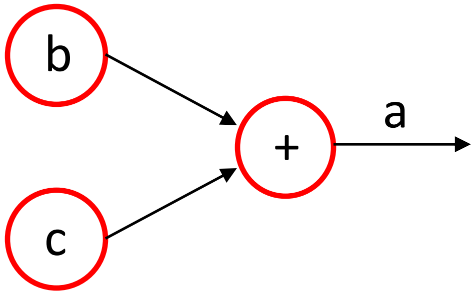
What we want to
see when training:

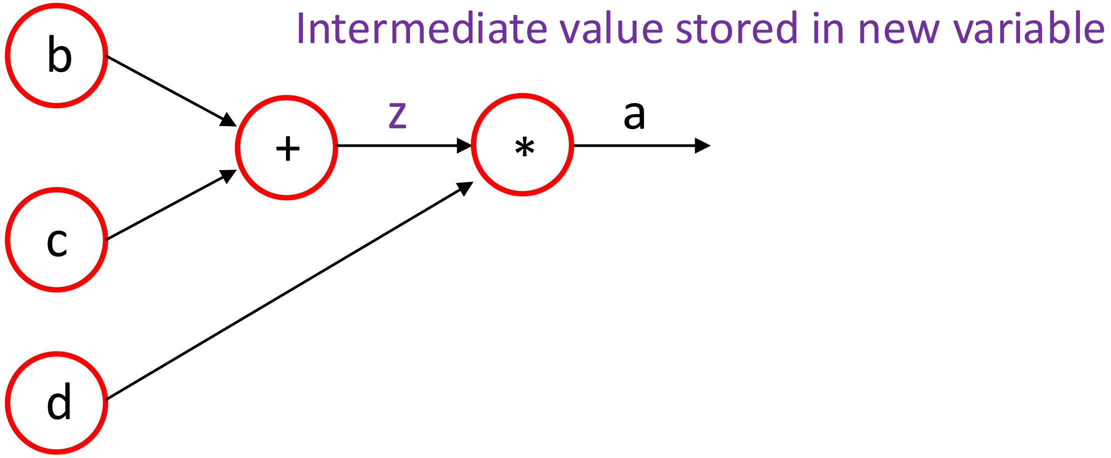


# Abstraction for Training Neural Networks: Computational Graphs

- A directed graph expressing a mathematical expression
  - Nodes: either variables (e.g., input values) or functions (applied to the in-flowing edges)
  - Edges: values output from functions

- For example: a = b + c OR a = d*(b + c)

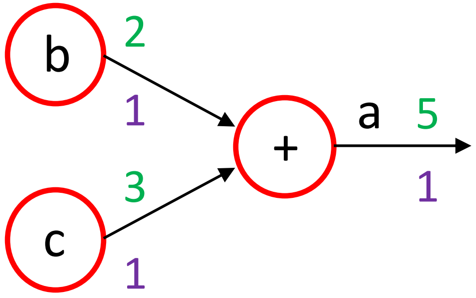


# How to Use a Computational Graph?

- Forward pass: to evaluate an expression, set input variables and then perform all sequential computational steps in the graph
- Backward pass: compute local gradient at each edge and then combine all relevant gradients for each function variable
- For example, using b=2 and c=3:

f(b,c) = b + c



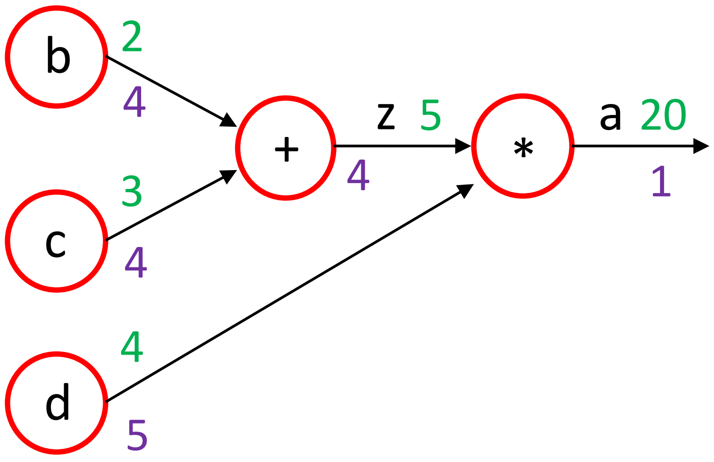

From calculus, we know:

*- df/df = 1*   *- df/db = 1*   *- df/dc = 1*

# How to Use a Computational Graph?

- Forward pass: to evaluate an expression, set input variables and then perform all sequential computational steps in the graph
- Backward pass: compute local gradient at each edge and then combine all relevant gradients for each function variable
- For example, using b=2, c=3, and d=4:

f(b,c,d) = d*(b + c)




From calculus, we know:

*- df/df = 1* *- df/dz = d* *- df/dd = z*

*- df/db = df/dz * dz/db* *- dz/db = 1*

*- df/dc = df/dz * dz/dc* *- df/dc = 1*

Multiplication passes the other value through and addition passes the incoming gradient

# Motivation for Computational Graphs

- Many deep learning frameworks rely on computational graphs; why?
  - **Efficient representation**: modular, primitive functions can be reused and combined to build complex models
  - **Efficient computation:** derivatives can be calculated based on "local" derivatives of the primitive functions in the graph, which enables them to be computed in parallel and stored for later reuse
  - **Interpretability:** simplifies understanding, debugging, and identifying bottlenecks in the network

# Today’s Topics

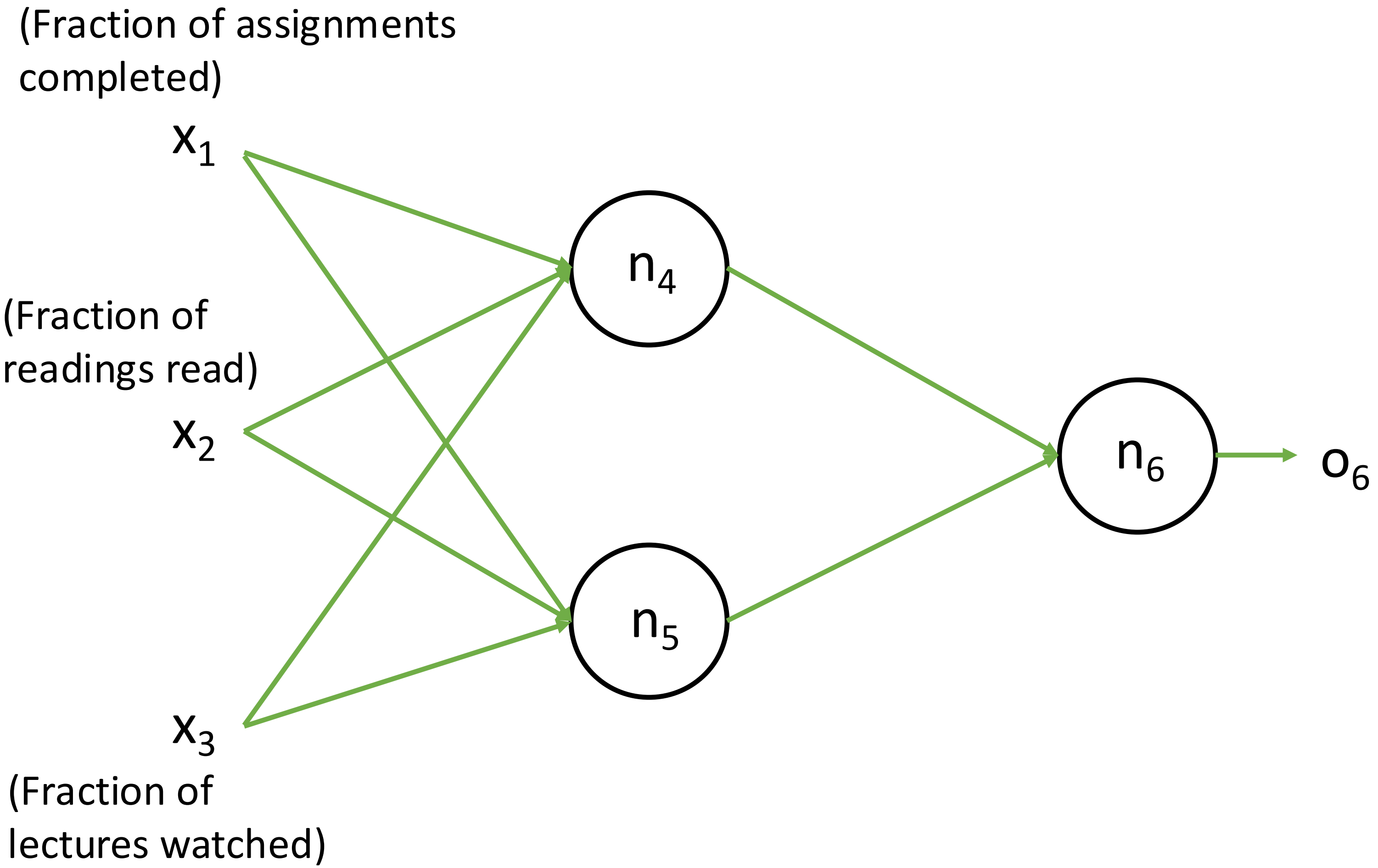
- Gradient descent: how neural networks learn
- Mathematical foundation of gradient descent: derivatives
- Applying gradient descent to train neural networks
- Training example

# Example

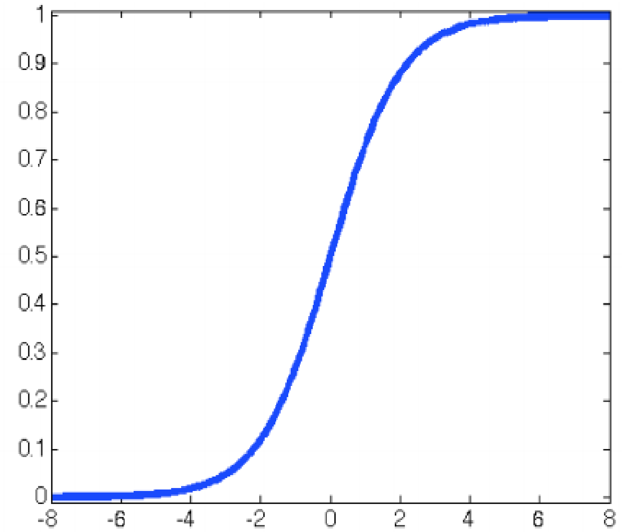
- Binary classification: predict if a student will get a B or better (minimum required for CS graduate student in this course)

- Inputs:
    - Fraction of assignments completed
    - Fraction of readings read
    - Fraction of lectures watched

# Example: Choose Neural Network Architecture



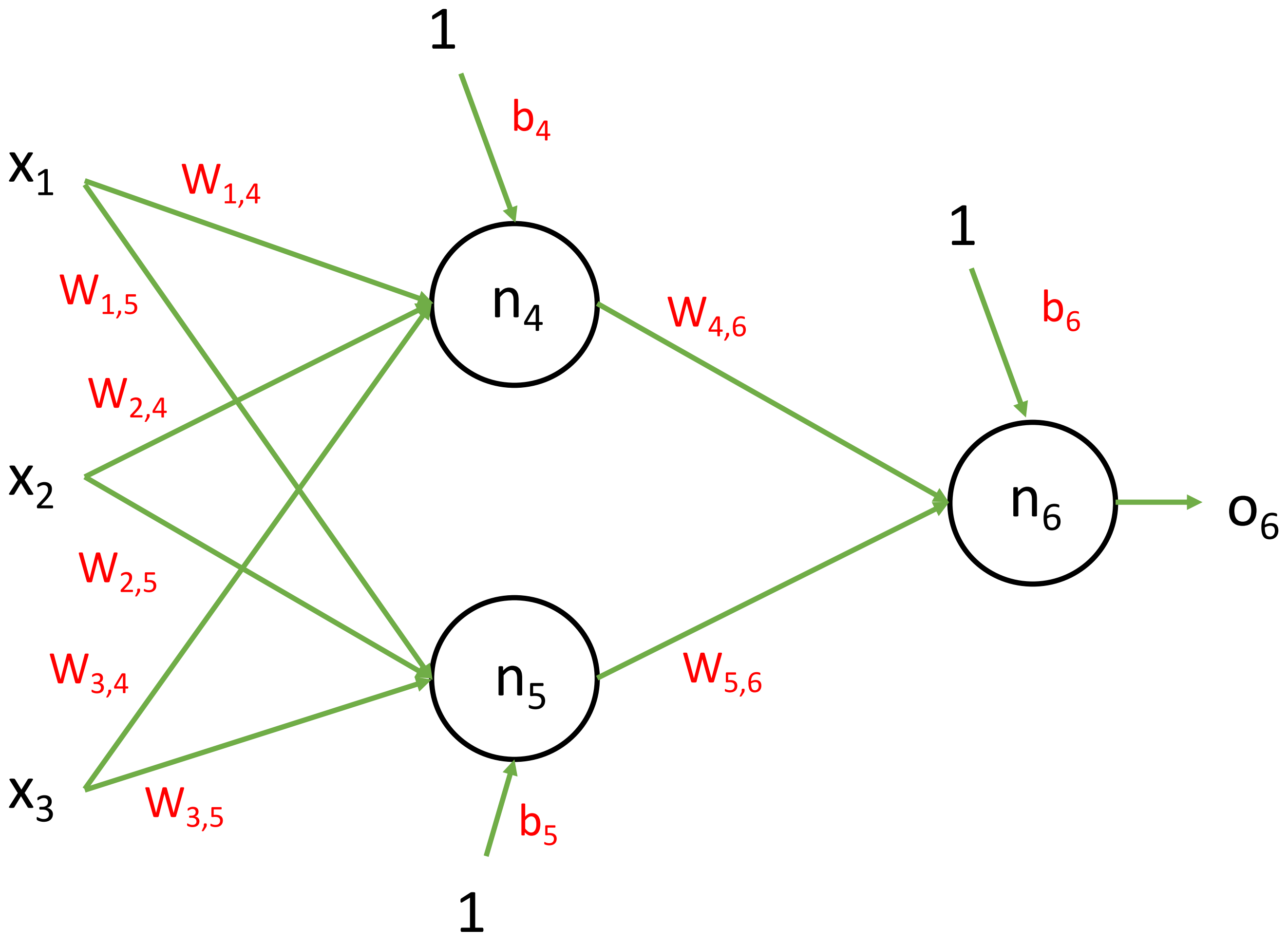

Sigmoid Activation Function




$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

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

# Example: Choose Neural Network Architecture




2-layer neural network

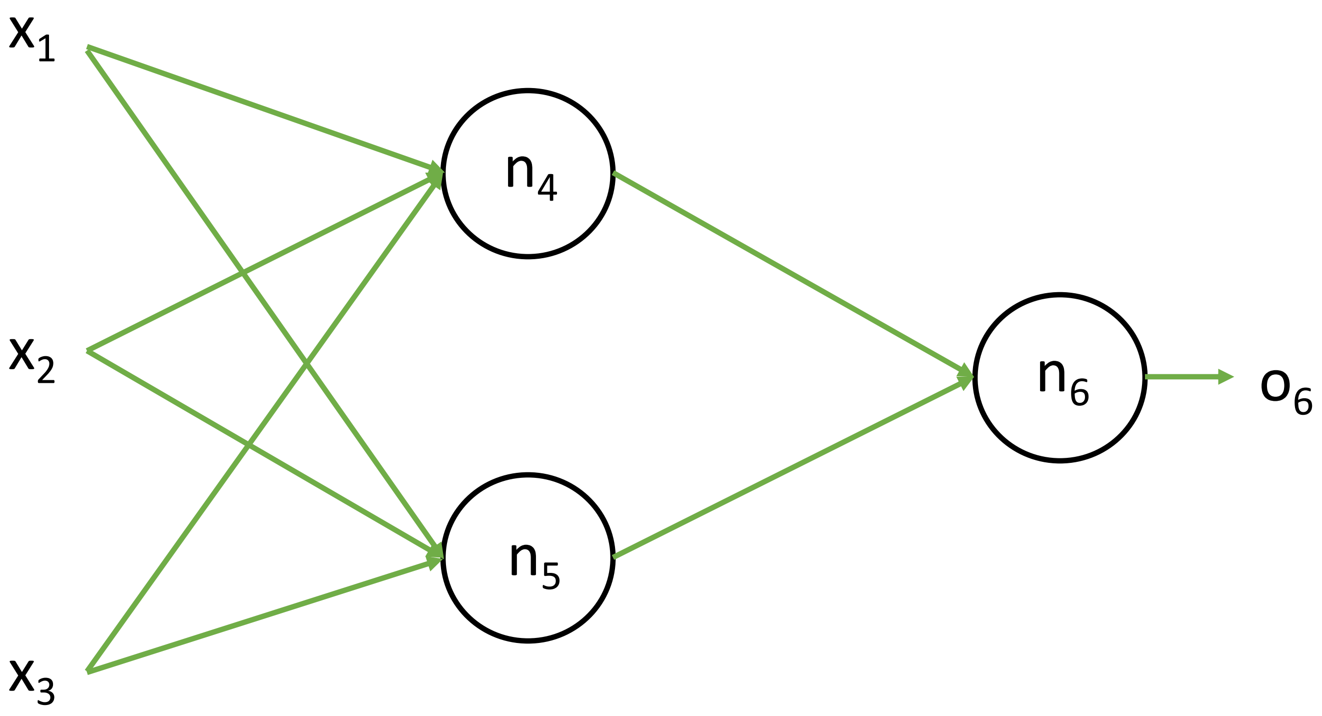
8 weights:

- Input to Hidden Layer 1:
  - 3x2 = 6
- Hidden Layer 1 to Hidden Layer 2:
  - 2x1 = 2

- 3 bias terms

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

# Example: Choose Loss Function




Squared Error

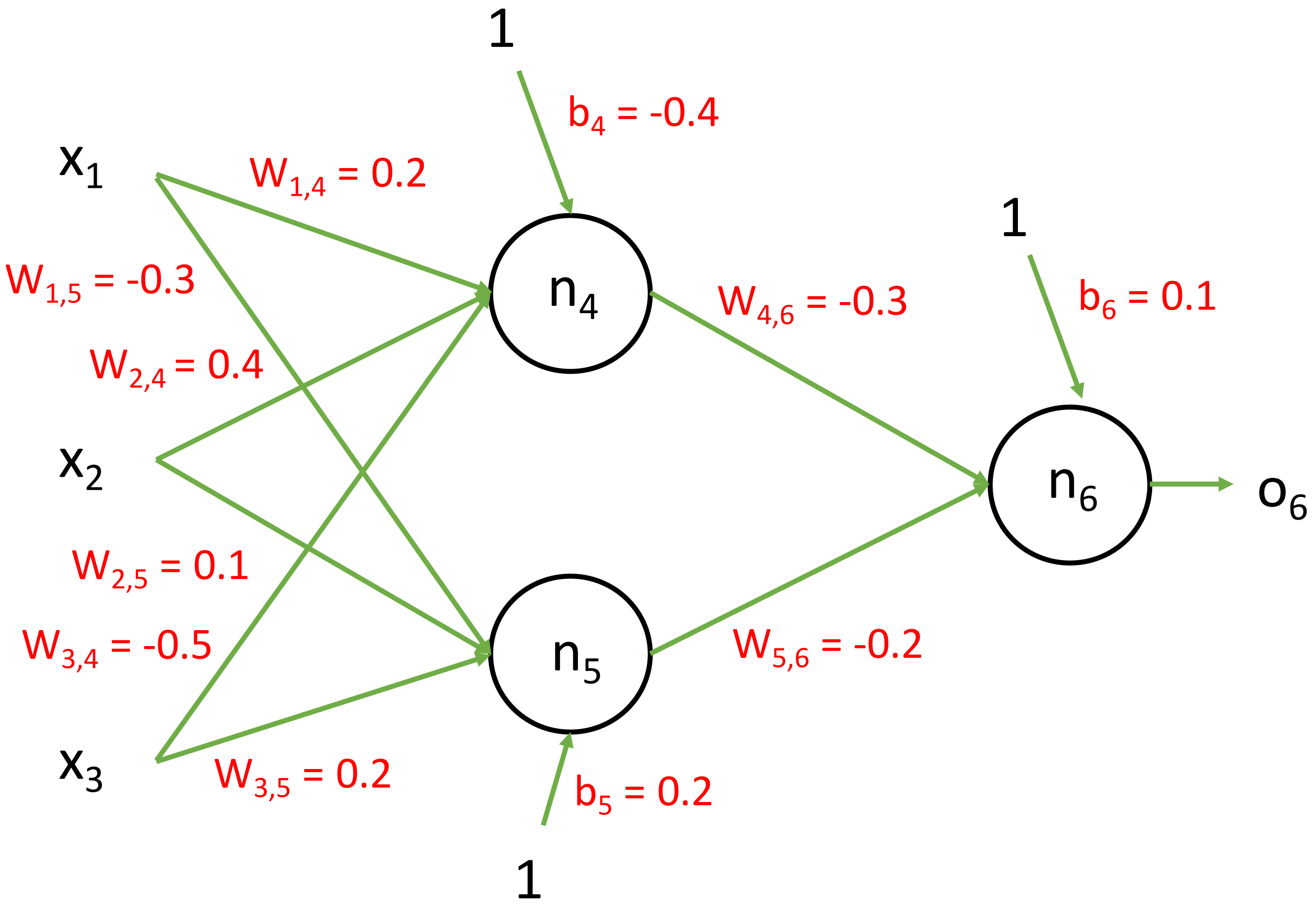
$$(Y_i - \hat{Y}_i)^2$$

True value

Predicted value

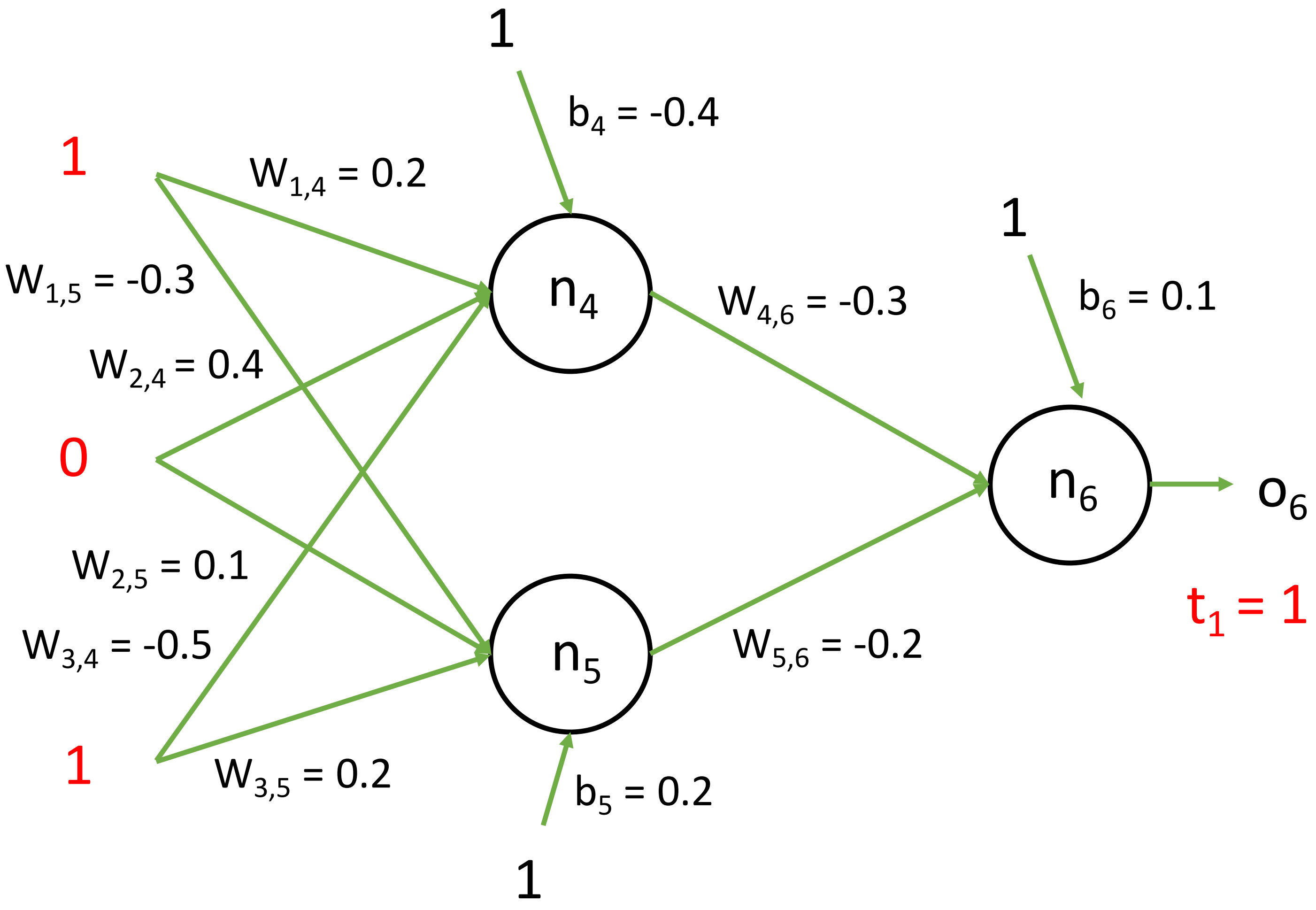
Example from: Jiawei Han and Micheline Kamber; Data Mining.

# Example: Initialize Model Parameters



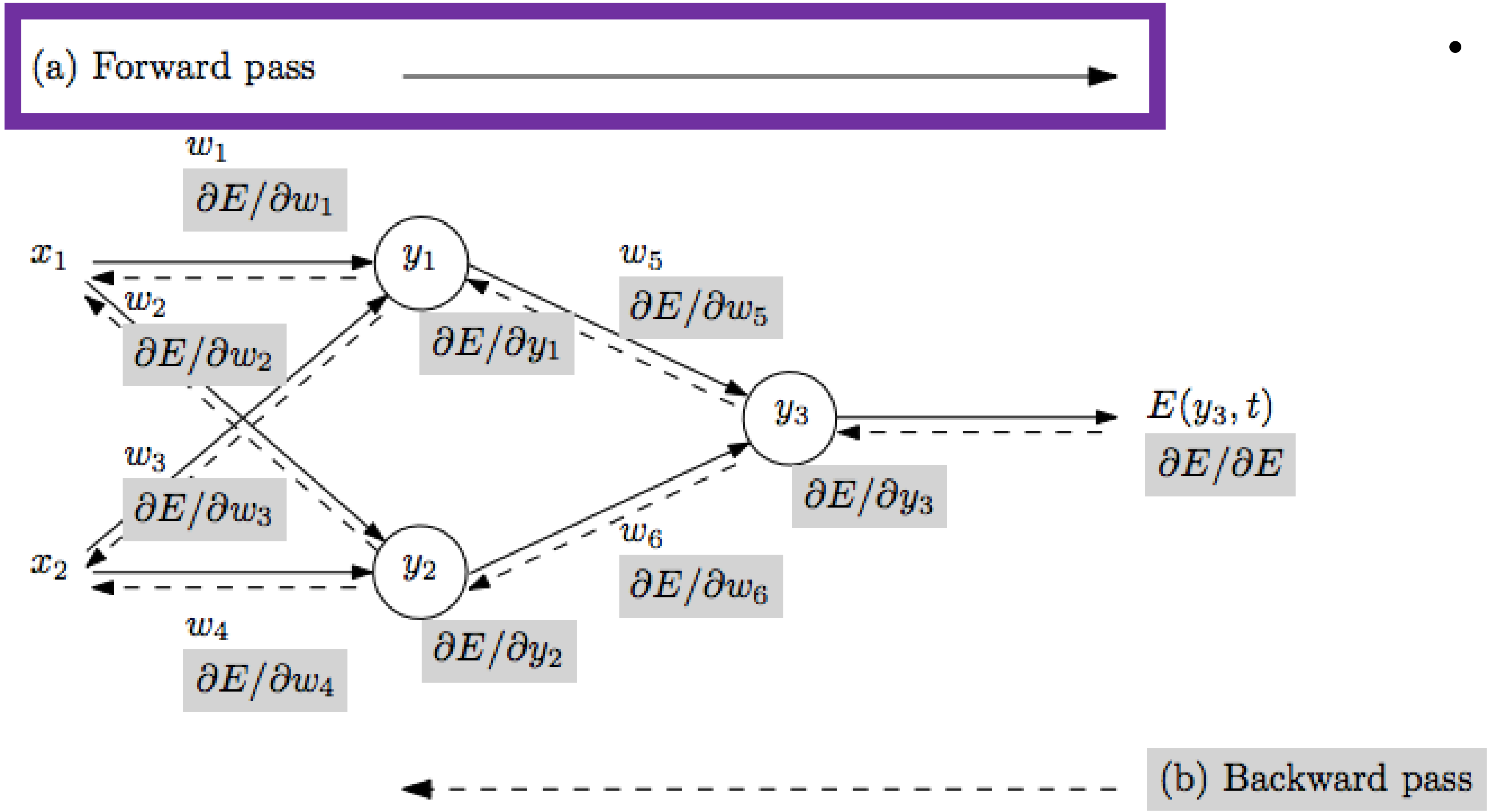

Example from: Jiawei Han and Micheline Kamber; Data Mining.

# Example: Input Training Example




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

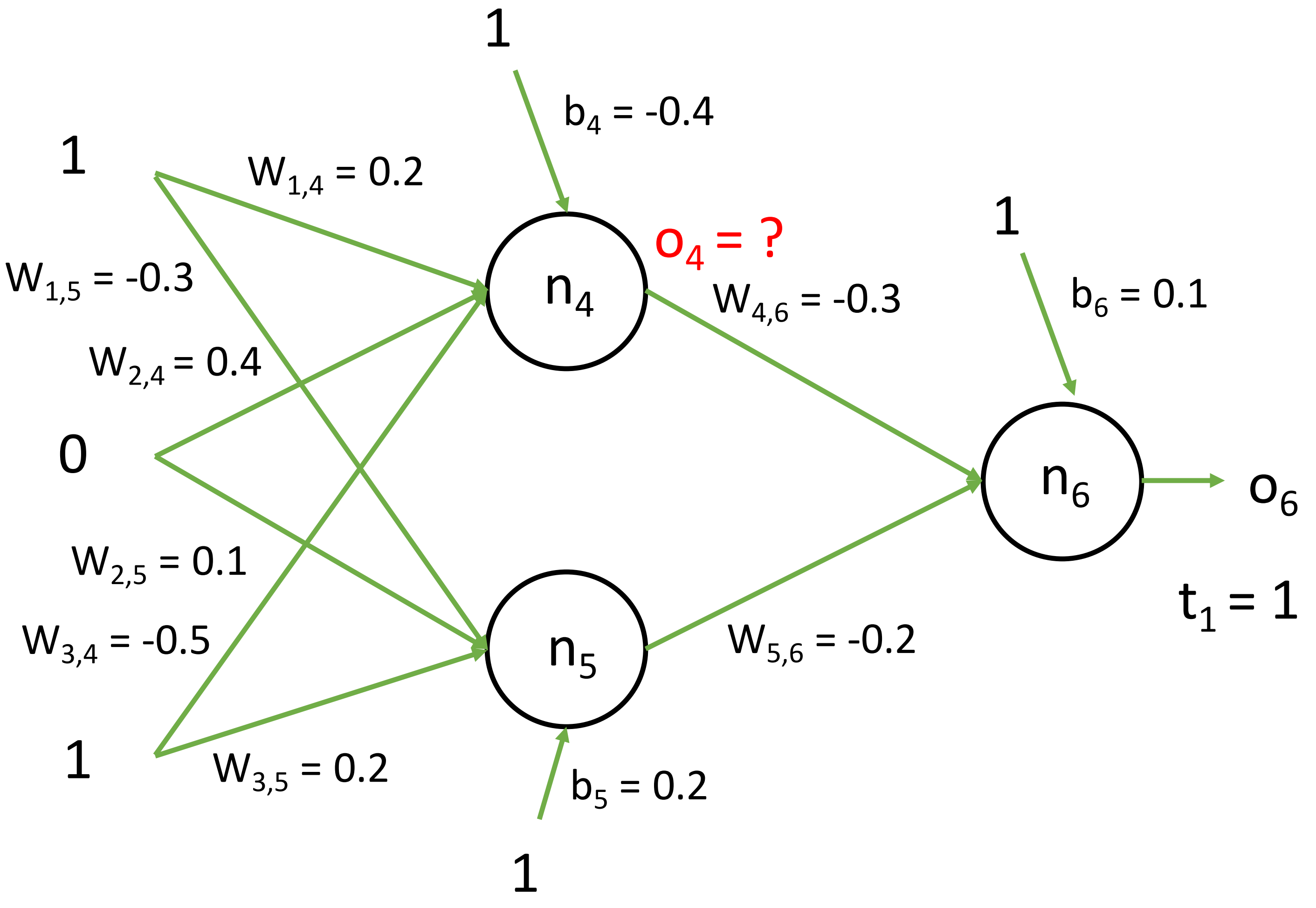
# Example: Step 1 – Forward Pass




- Repeat until stopping criterion met:
    1. **Forward pass**: propagate training data through model to make predictions

Baydin et al. Automatic Differentiation in Machine Learning: a Survey. 2018

# Example: Step 1 – Forward Pass




Weighted input sum to node 4:

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

$i_4 = -0.7$

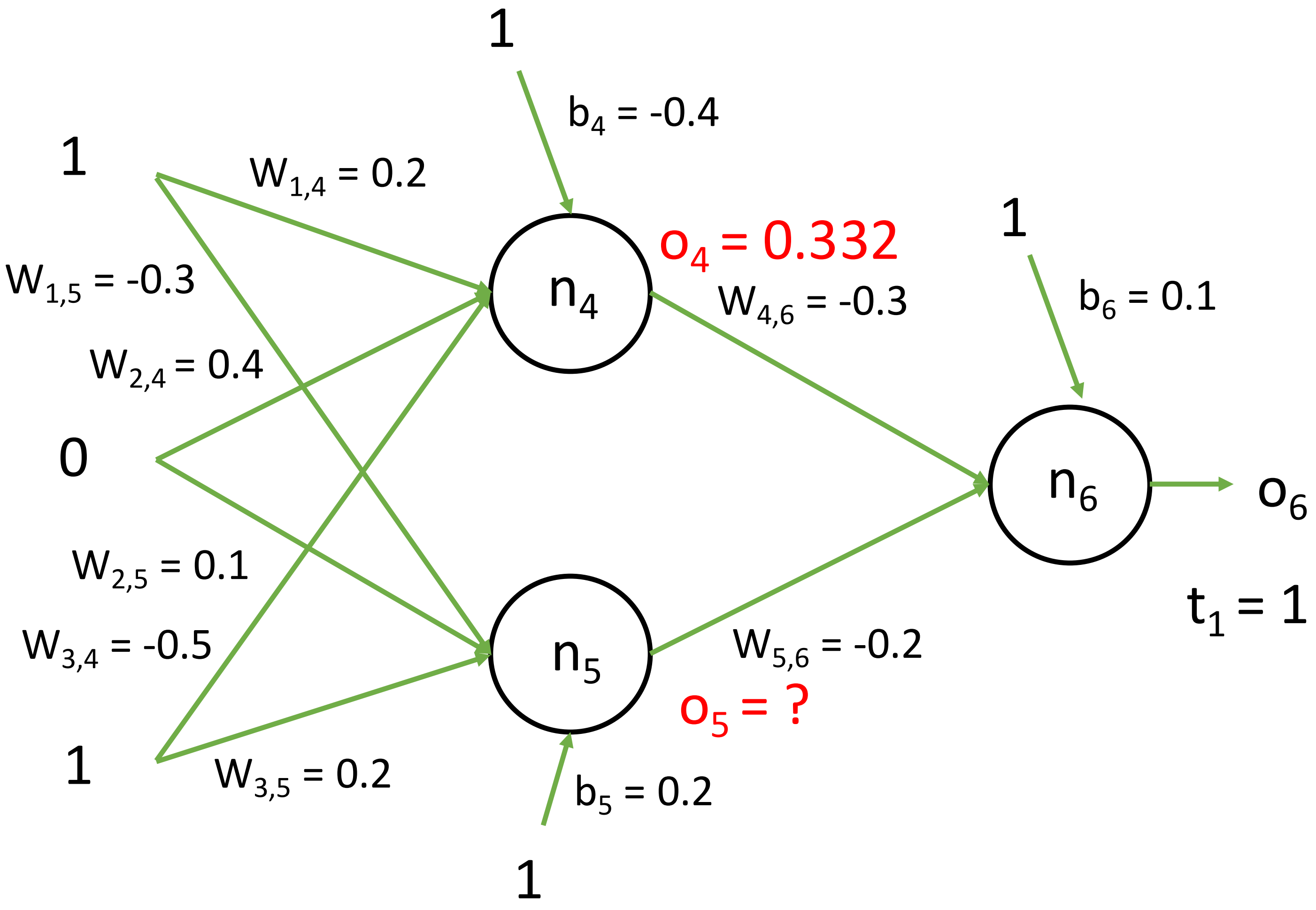
Activation result from node 4:

$o_4 = \textit{sigmoid}(-0.7)$

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

$o_4 = 0.332$

# Example: Step 1 – Forward Pass




Weighted input sum to node 5:

$i_5 = (1 \text{ x } -0.3 + 0 \text{ x } 0.1 + 1 \text{ x } 0.2) + 0.2$

$i_5 = 0.1$

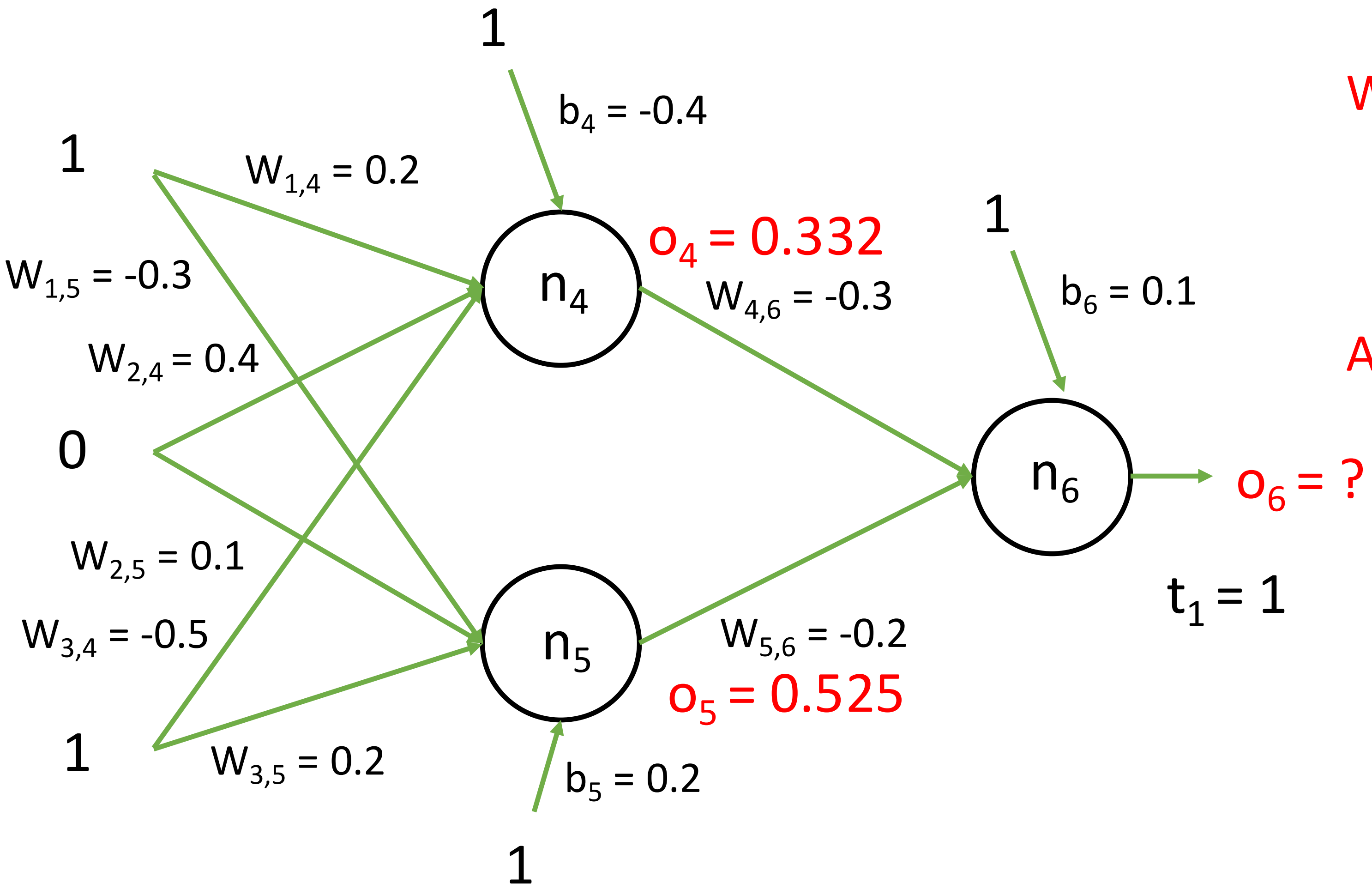
Activation result from node 5:

$o_5 = \textit{sigmoid}(0.1)$

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

$o_5 = 0.525$

# Example: Step 1 – Forward Pass




Weighted input sum to node 6:

$i_6 = (0.332 \times -0.3 + 0.525 \times -0.2) + 0.1$

$i_6 = -0.105$

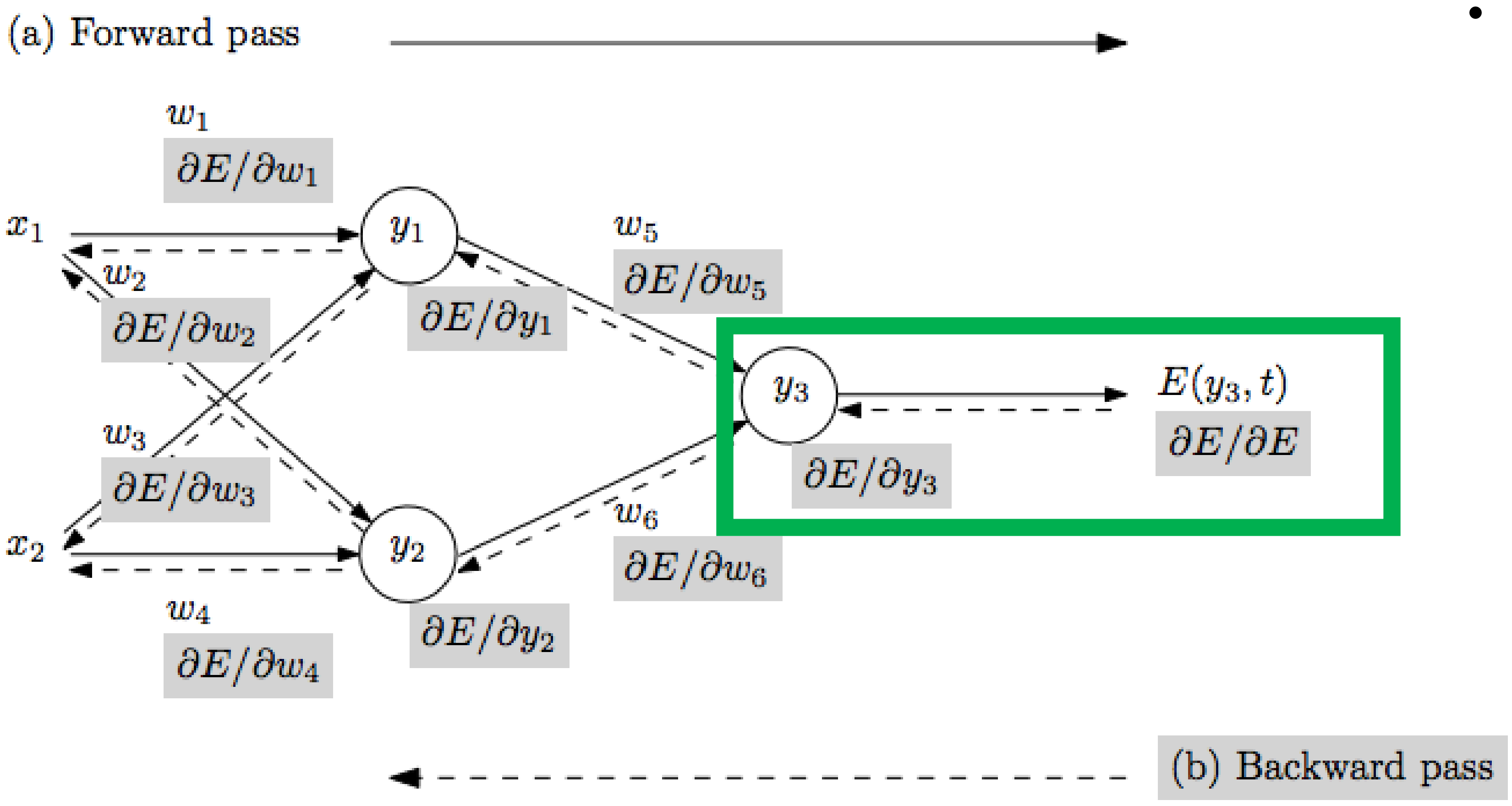
Activation result from node 6:

$o_6 = \textit{sigmoid}(-0.105)$

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

$o_6 = 0.474$

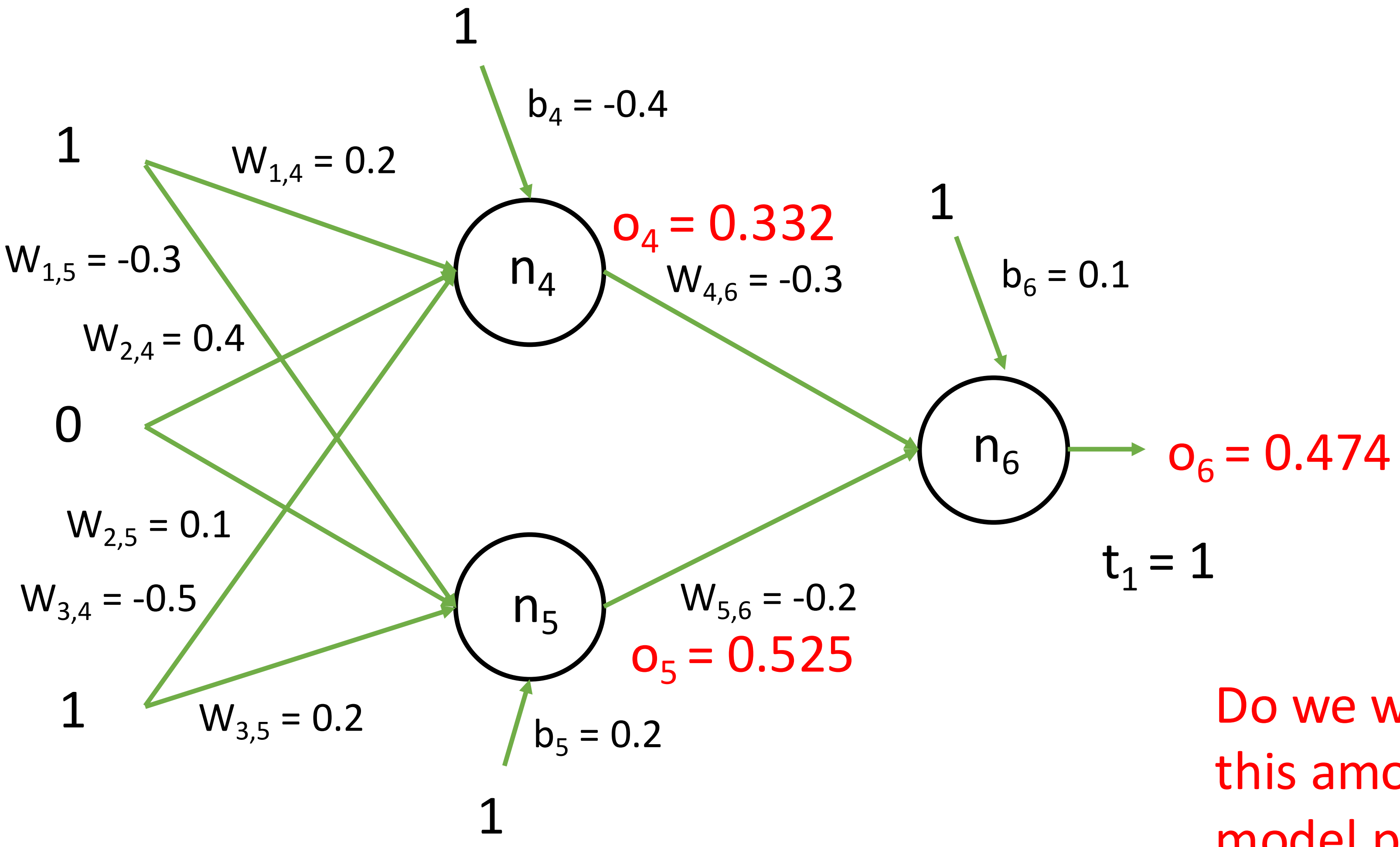
# Example: Step 2 – Calculate Error (Loss)




- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make predictions
  2. **Error quantification**: measure error of the model's predictions on training data using a loss function

Baydin et al. Automatic Differentiation in Machine Learning: a Survey. 2018

# Example: Step 2 – Calculate Error (Loss)



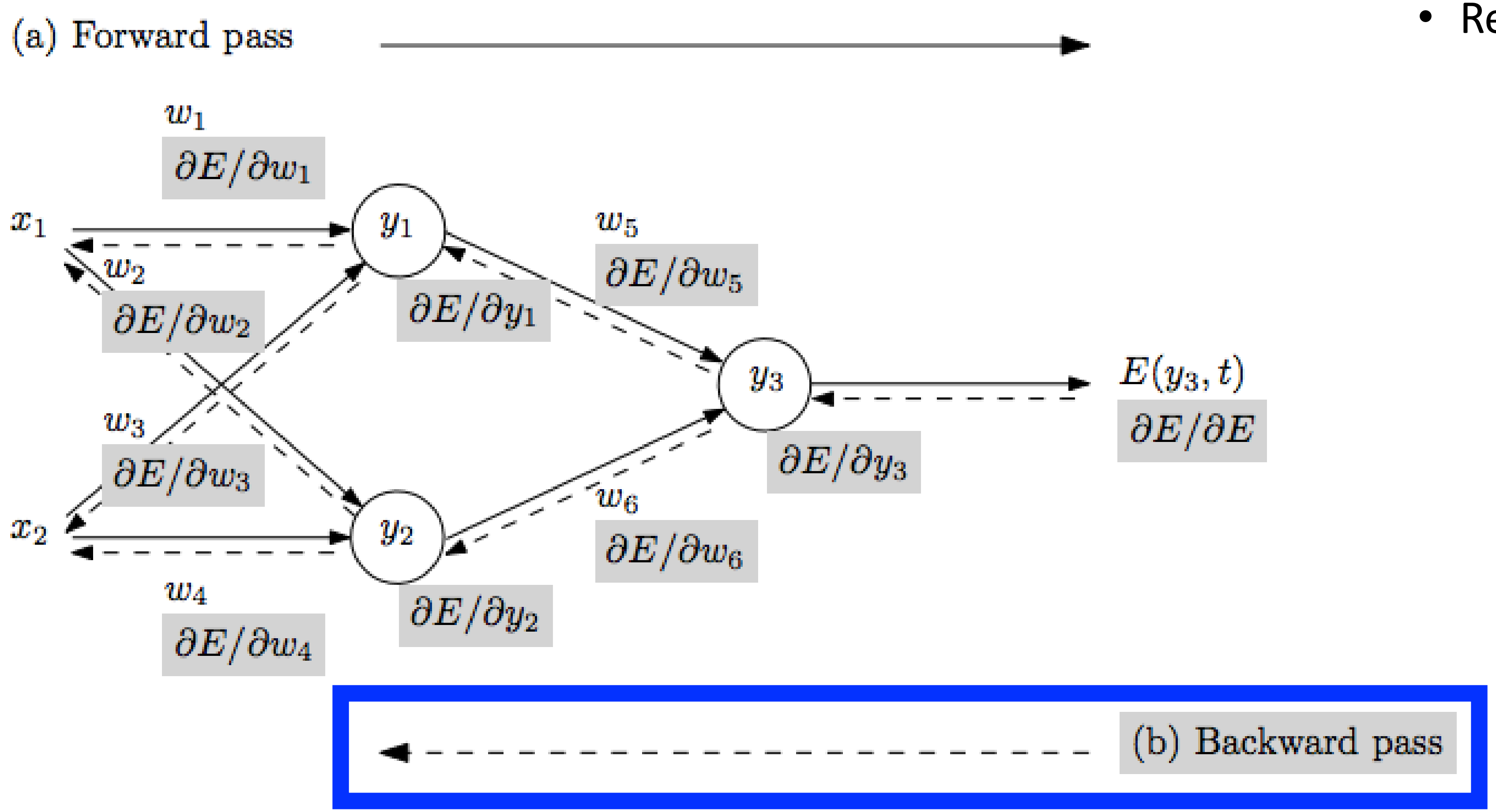

Squared Error

$= (1-0.474)^2$

$= 0.277$

Do we want to increase or reduce this amount when modifying the model parameters during training?

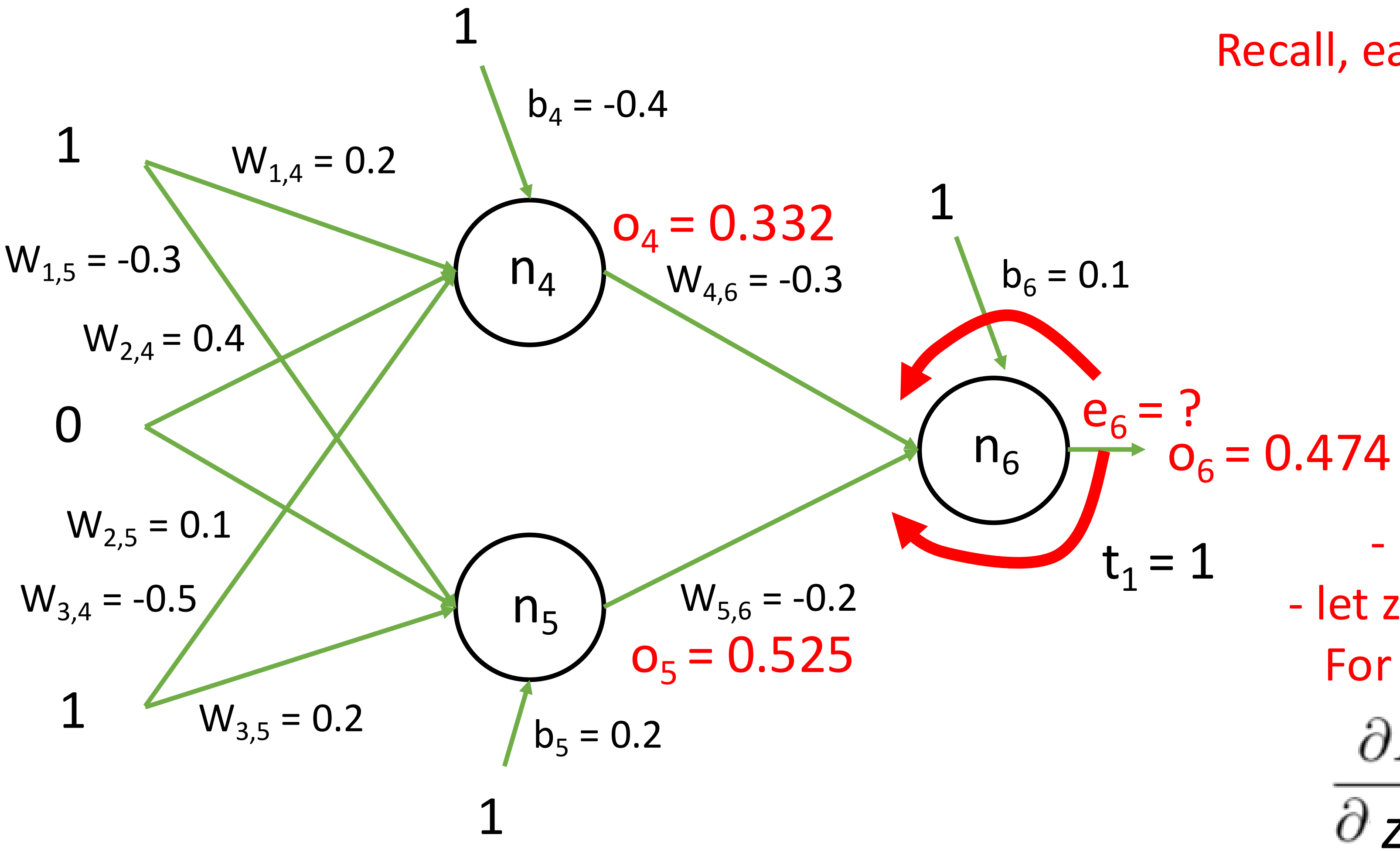
# Example: Step 3 – Backward Pass



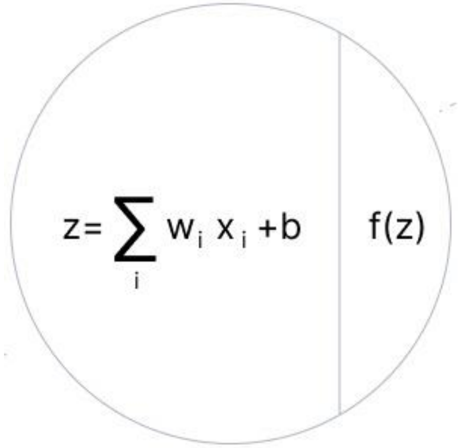

- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make predictions
  2. **Error quantification**: measure error of the model's predictions on training data using a loss function
  3. **Backward pass**: calculate gradients to determine how each model parameter contributed to model error

Baydin et al. Automatic Differentiation in Machine Learning: a Survey. 2018

# Example: Step 3 – Backward Pass




Recall, each node contains 2 functions:




- let $o_k$ represent the activation
- let $z_k$ represent the weighted sum

For node 6, we will first compute:

$$\frac{\partial E}{\partial z_6} = \frac{\partial E}{\partial o_6} * \frac{\partial o_6}{\partial z_6}$$

Image source: https://www.v7labs.com/blog/neural-networks-activation-functions

# Example: Step 3 – Backward Pass

Loss function:

constant value

$$\frac{\partial E}{\partial o_6} = \frac{\partial}{\partial o_6}(t_k - o_6)^2 = \boxed{-2(t_6 - o_6)}$$

Activation function (sigmoid):

$$\frac{\partial o_6}{\partial z_6} = \frac{\partial}{\partial z_6}\frac{1}{1+e^{-z_6}} = (1-\sigma(z_6))\,\sigma(z_6)$$

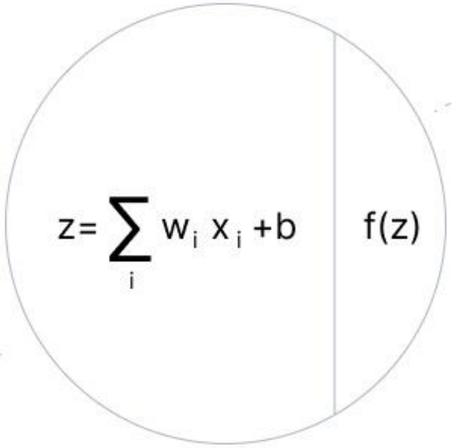
$$= \boxed{(1 - o_6)\,o_6}$$

Putting them together:

$$\frac{\partial E}{\partial z_6} = \boxed{-\cancel{2}(t_6 - o_6)}\ \boxed{(1 - o_6)\,o_6}$$

Can drop constant (does not affect learning)

Recall, each node contains 2 functions:




- let $o_k$ represent the activation
- let $z_k$ represent the weighted sum

For node 6, we will first compute:

$$\boxed{\frac{\partial E}{\partial z_6}} = \boxed{\frac{\partial E}{\partial o_6}} * \boxed{\frac{\partial o_6}{\partial z_6}}$$

Image source: https://www.v7labs.com/blog/neural-networks-activation-functions

# Example: Step 3 – Backward Pass

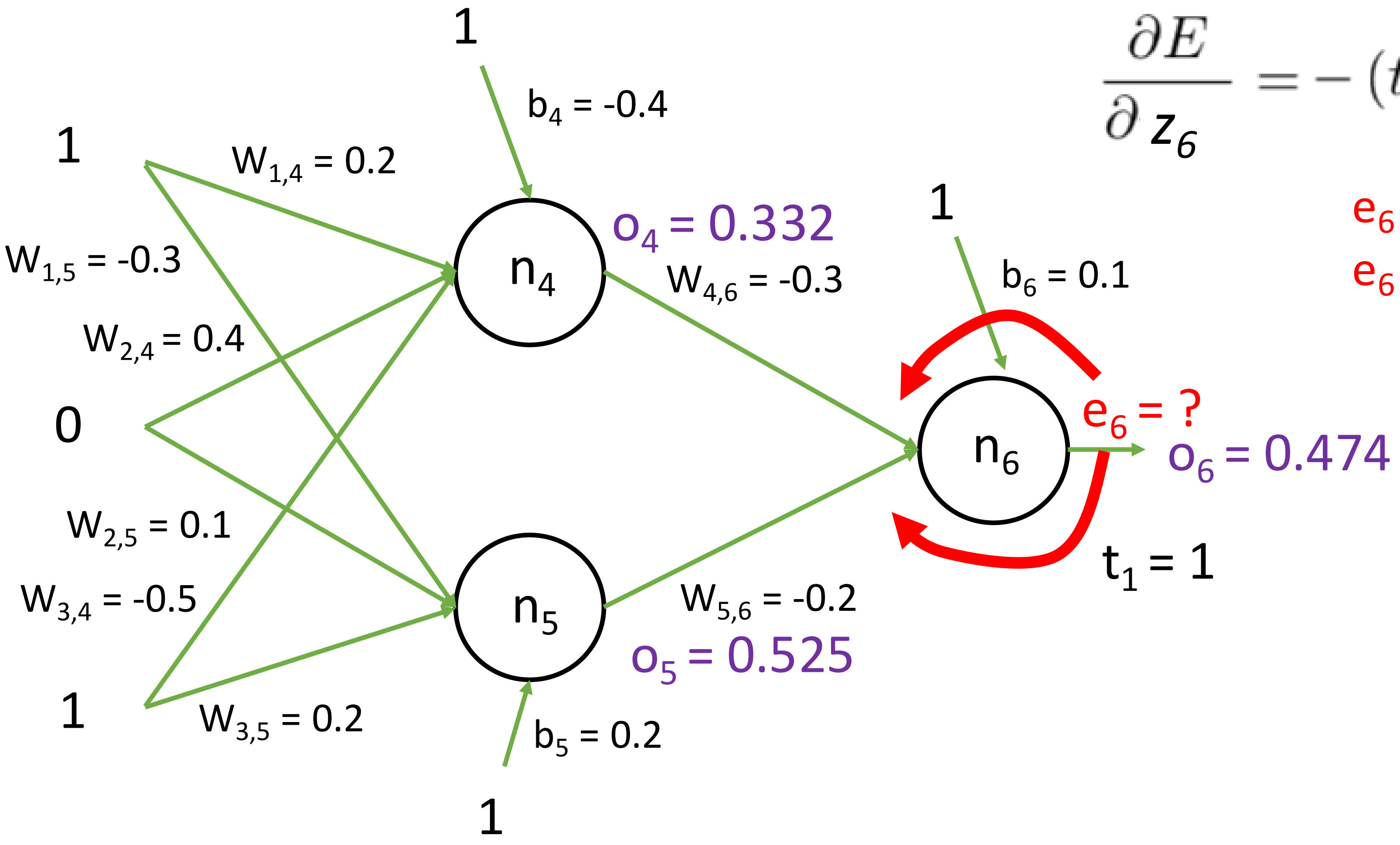
Key Observation: Can compute this because both the activation function and loss function are **differentiable**!!!

$$\frac{\partial E}{\partial z_6} = \boxed{-\cancel{2}(t_6 - o_6)}\ \boxed{(1 - o_6)\ )\ o_6}$$

Can drop constant (does not affect learning)

$$\boxed{\frac{\partial E}{\partial z_6}} = \boxed{\frac{\partial E}{\partial o_6}} * \boxed{\frac{\partial o_6}{\partial z_6}}$$

# Example: Step 3 – Backward Pass

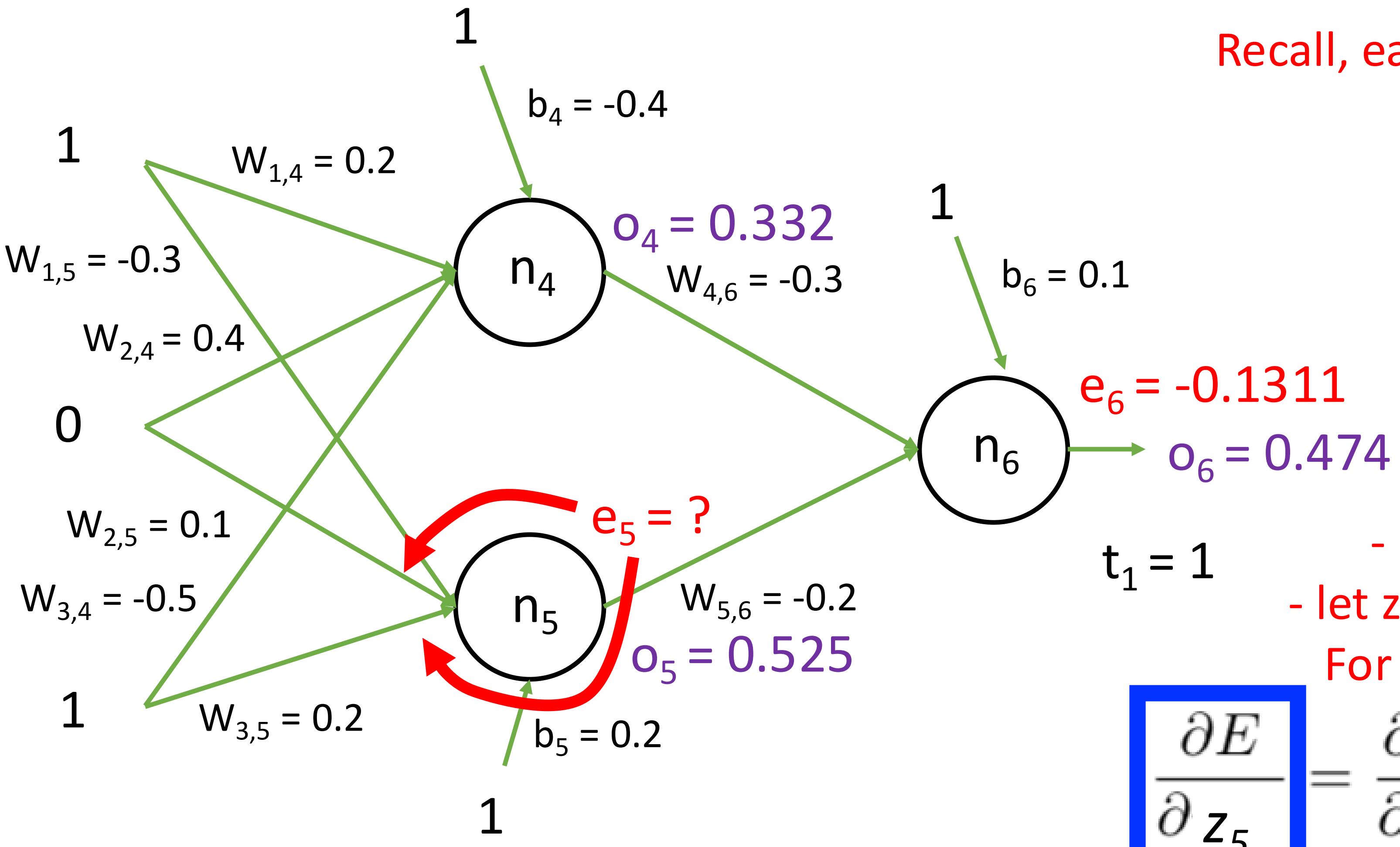



$$\frac{\partial E}{\partial z_6} = -(t_6 - o_6)(1 - o_6)\, o_6$$

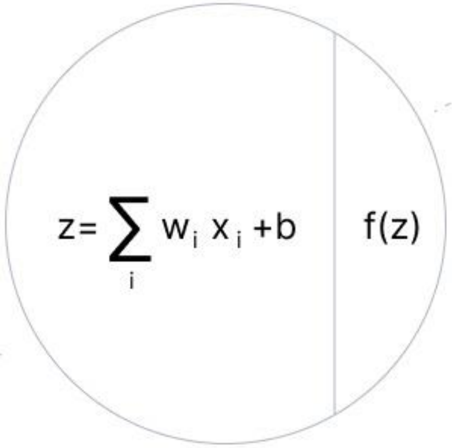
$e_6$ = -(1-0.474)(1-0.474)(0.474)

$e_6$ = -0.1311

# Example: Step 3 – Backward Pass




Recall, each node contains 2 functions:




- let $o_k$ represent the activation
- let $z_k$ represent the weighted sum

For node 5, we will first compute:

$$\boxed{\frac{\partial E}{\partial z_5}} = \frac{\partial E}{\partial o_6} * \frac{\partial o_6}{\partial z_6} * \boxed{\frac{\partial z_6}{\partial o_5}} * \boxed{\frac{\partial o_5}{\partial z_5}}$$

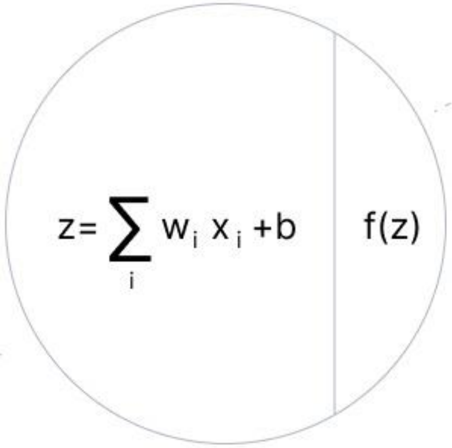
Image source: https://www.v7labs.com/blog/neural-networks-activation-functions

# Example: Step 3 – Backward Pass

Parent node:

$$\frac{\partial z_6}{\partial o_5} = \frac{\partial}{\partial o_5}(o_4 w_{4,6} + o_5 w_{5,6} + b_6) = \boxed{w_{5,6}}$$

Recall, each node contains 2 functions:




Activation function (sigmoid):

$$\frac{\partial o_5}{\partial z_5} = \frac{\partial}{\partial z_5}\frac{1}{1+e^{-z_5}} = (1-\sigma(z_5))\,\sigma(z_5)$$
$$= \boxed{(1-o_5)\,o_5}$$

- let $o_k$ represent the activation
- let $z_k$ represent the weighted sum

For node 5, we will first compute:

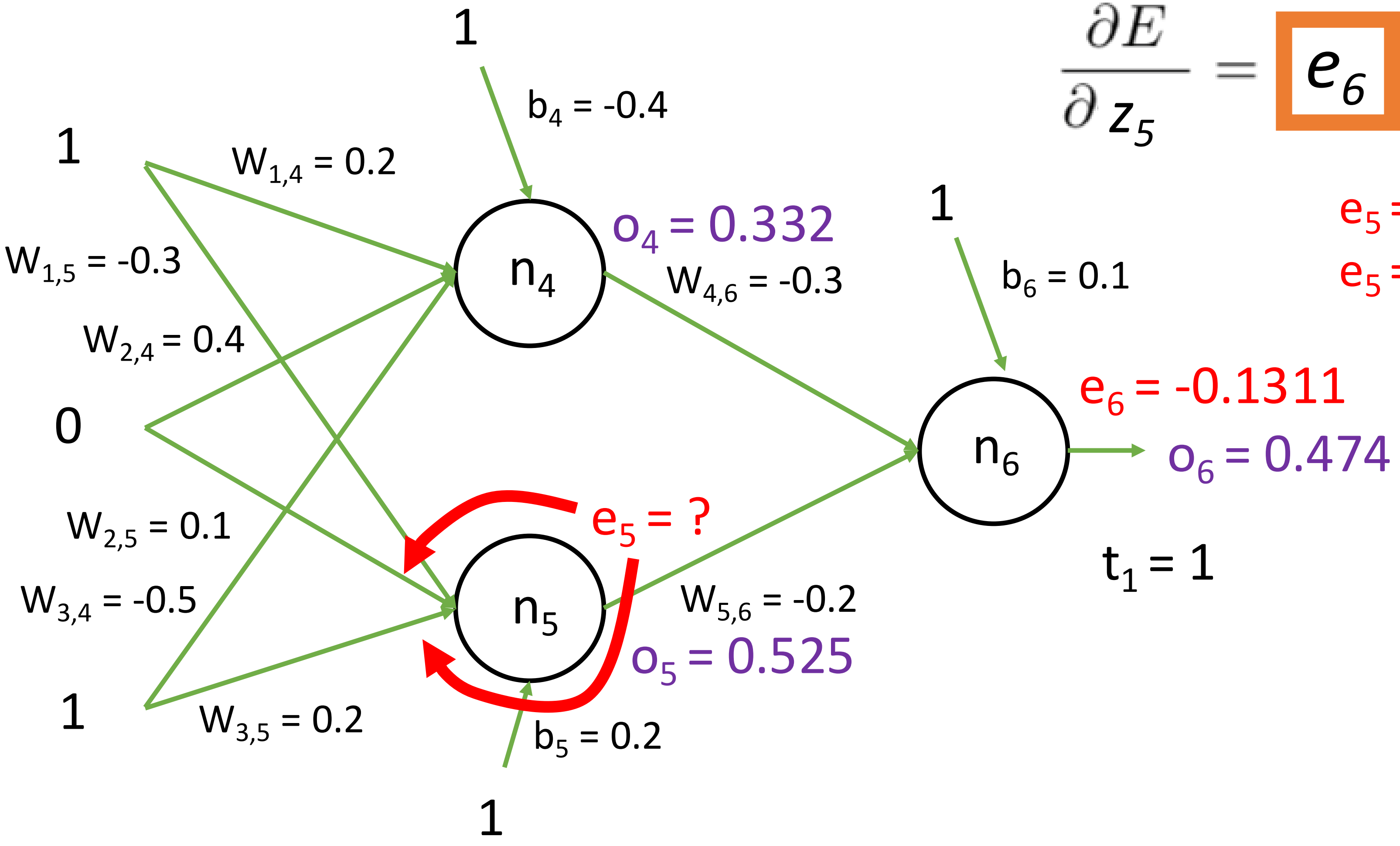
$$\boxed{\frac{\partial E}{\partial z_5}} = \boxed{\frac{\partial E}{\partial o_6} * \frac{\partial o_6}{\partial z_6}} * \boxed{\frac{\partial z_6}{\partial o_5}} * \boxed{\frac{\partial o_5}{\partial z_5}}$$

Putting them together:

$$\frac{\partial E}{\partial z_5} = \boxed{e_6}\,\boxed{w_{5,6}}\,\boxed{(1-o_5)\,o_5}$$

Image source: https://www.v7labs.com/blog/neural-networks-activation-functions

# Example: Step 3 – Backward Pass

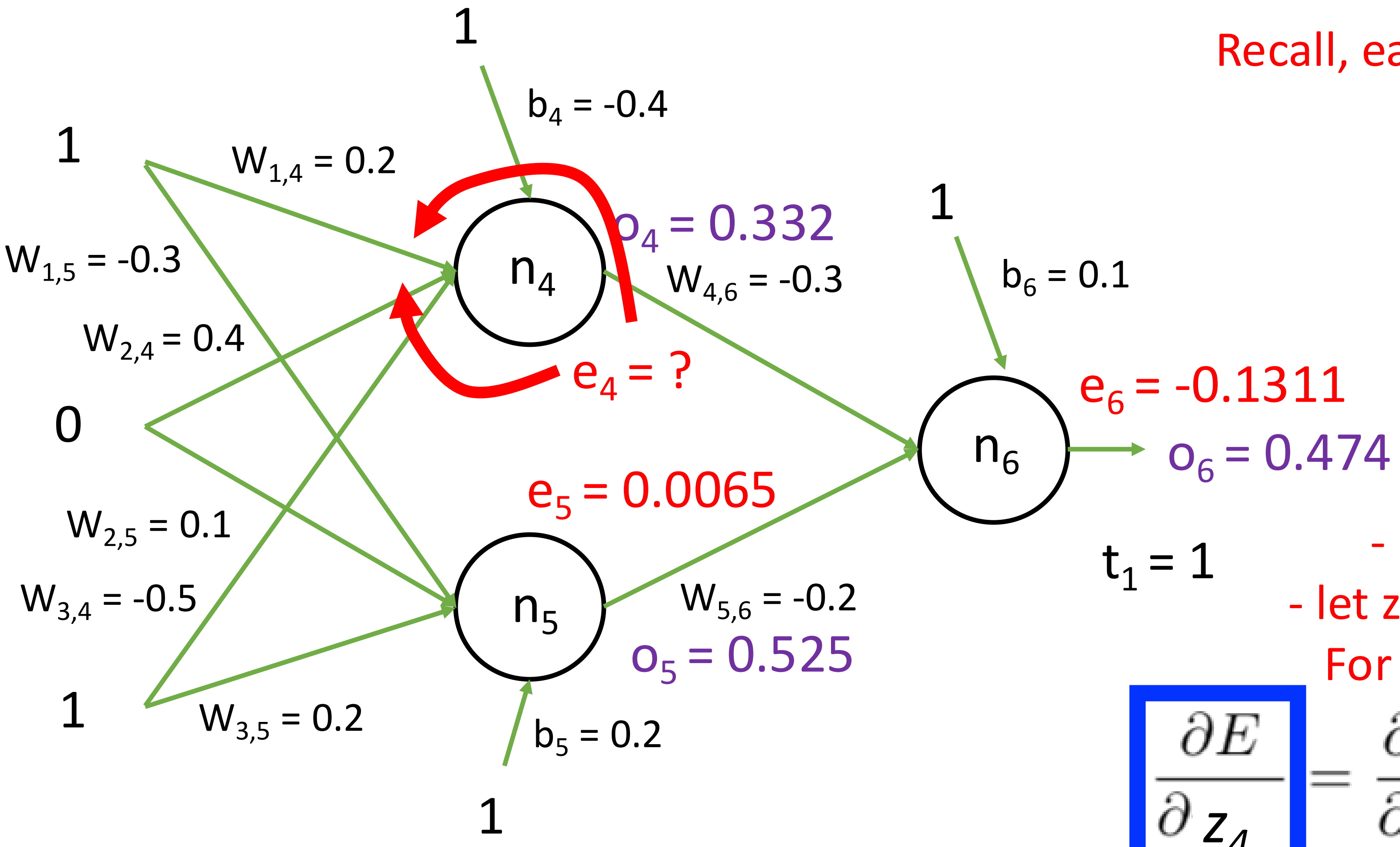



$$\frac{\partial E}{\partial z_5} = \boxed{e_6}\,\boxed{w_{5,6}}\,\boxed{(1 - o_5))\ o_5}$$

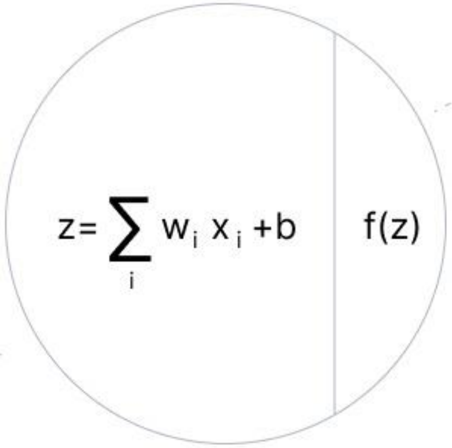
$e_5 = -0.1311 * -0.2 * (1-0.525) * 0.525$

$e_5 = 0.0065$

# Example: Step 3 – Backward Pass




Recall, each node contains 2 functions:




- let $o_k$ represent the activation
- let $z_k$ represent the weighted sum

For node 5, we will first compute:

$$\boxed{\frac{\partial E}{\partial z_4}} = \frac{\partial E}{\partial o_6} * \frac{\partial o_6}{\partial z_6} * \boxed{\frac{\partial z_6}{\partial o_4}} * \boxed{\frac{\partial o_4}{\partial z_4}}$$

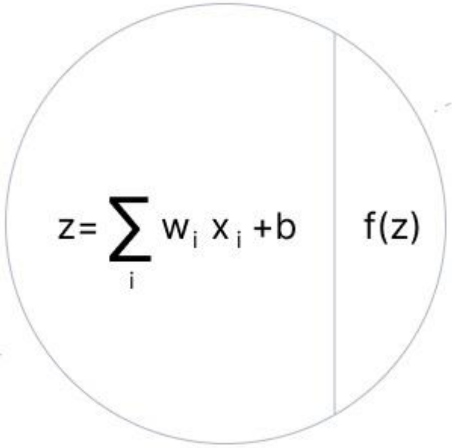
Image source: https://www.v7labs.com/blog/neural-networks-activation-functions

# Example: Step 3 – Backward Pass

Parent node:

$$\frac{\partial z_6}{\partial o_4} = \frac{\partial}{\partial o_4}\left(o_4\, w_{4,6} + o_5\, w_{5,6} + b_6\right) = \boxed{w_{4,6}}$$

Recall, each node contains 2 functions:




Activation function (sigmoid):

$$\frac{\partial o_4}{\partial z_4} = \frac{\partial}{\partial z_4}\,\frac{1}{1+e^{-z_4}} = (1-\sigma(z_4))\,\sigma(z_4)$$

$$= \boxed{(1 - o_4\;))\; o_4}$$

- let $o_k$ represent the activation
- let $z_k$ represent the weighted sum

Putting them together:

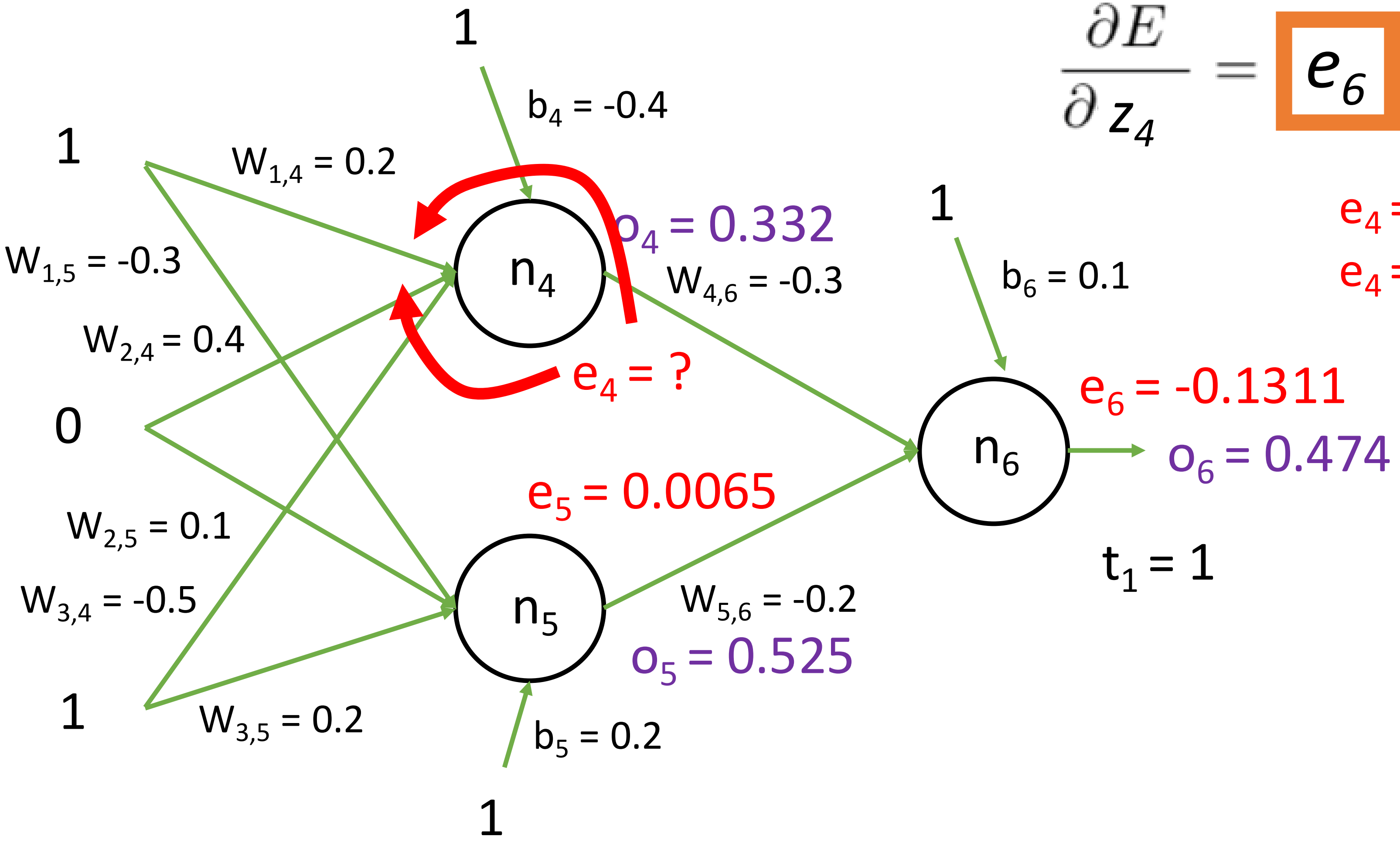
$$\frac{\partial E}{\partial z_4} = \boxed{e_6}\,\boxed{w_{4,6}}\,\boxed{(1 - o_4\;))\; o_4}$$

For node 5, we will first compute:

$$\boxed{\frac{\partial E}{\partial z_4}} = \boxed{\frac{\partial E}{\partial o_6} * \frac{\partial o_6}{\partial z_6}} * \boxed{\frac{\partial z_6}{\partial o_4}} * \boxed{\frac{\partial o_4}{\partial z_4}}$$

Image source: https://www.v7labs.com/blog/neural-networks-activation-functions

# Example: Step 3 – Backward Pass

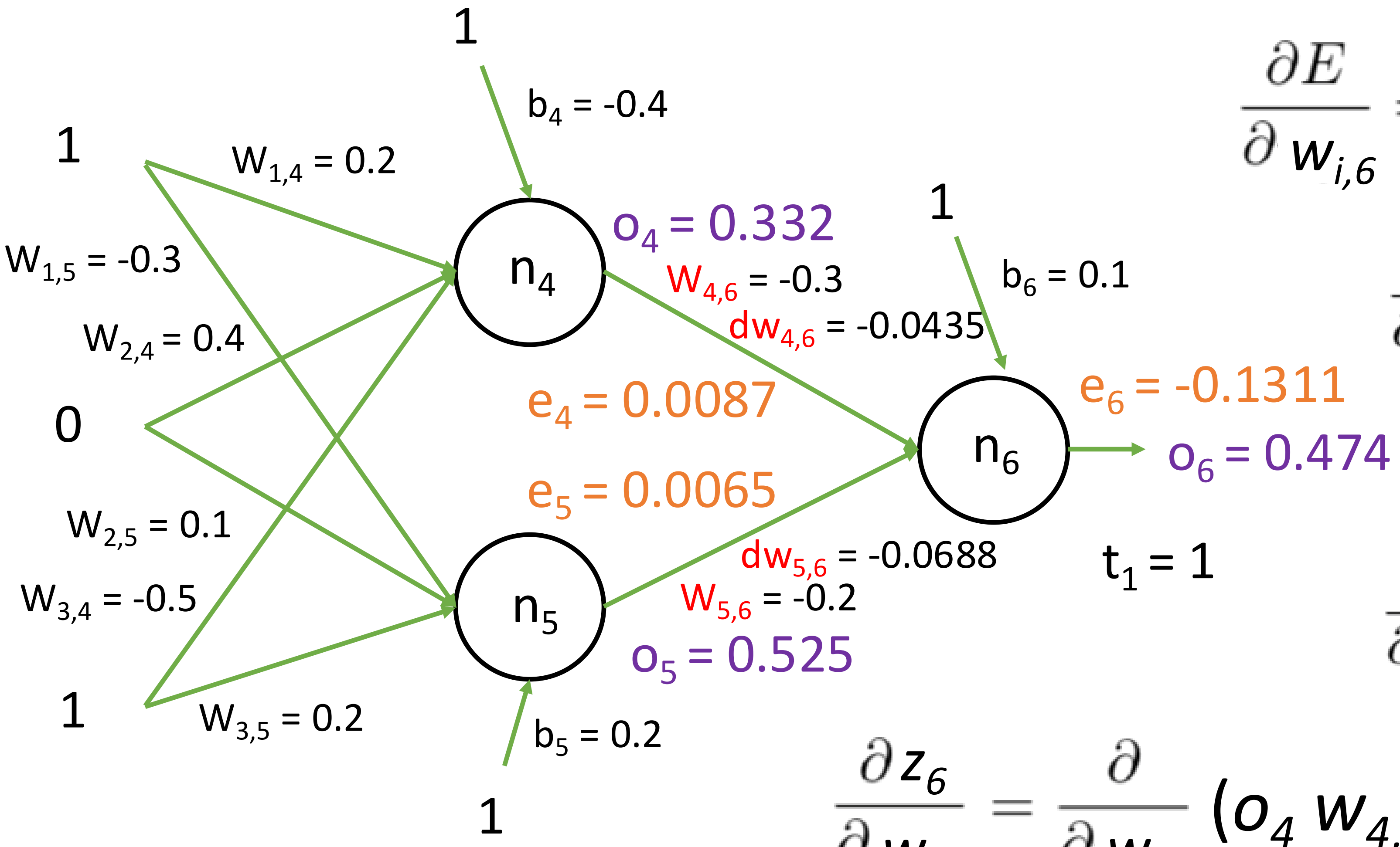



$$\frac{\partial E}{\partial z_4} = \boxed{e_6}\,\boxed{w_{4,6}}\,\boxed{(1 - o_4)\,o_4}$$

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

$e_4 = 0.0087$

# Example: Step 3 – Backward Pass




$$\frac{\partial E}{\partial w_{i,6}} = \frac{\partial E}{\partial o_6} * \frac{\partial o_6}{\partial z_6} * \boxed{\frac{\partial z_6}{\partial w_{i,6}}}$$

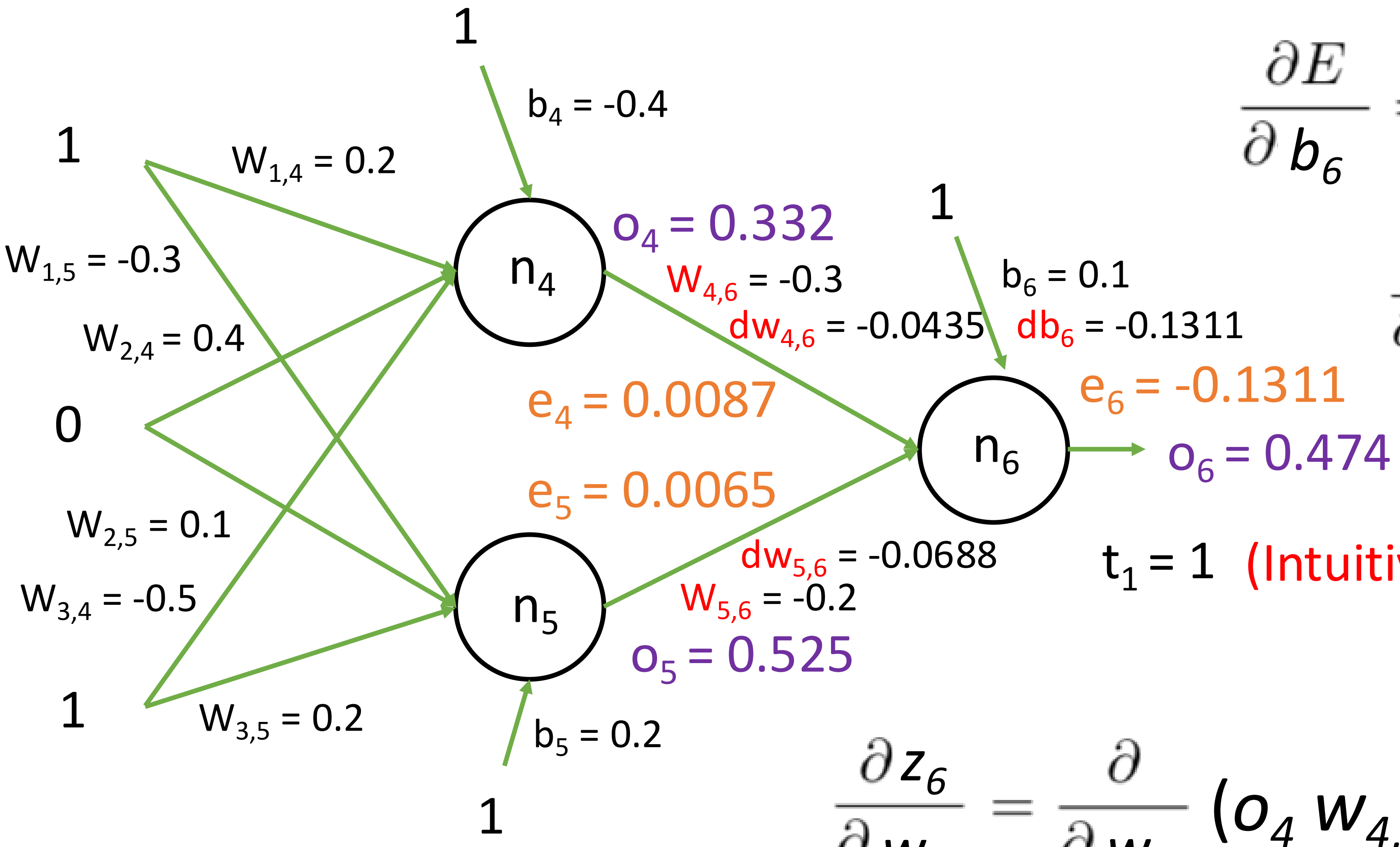
$$\frac{\partial E}{\partial w_{4,6}} = e_6 * o_4 = -0.0435$$

*Recall, gradient points to direction of larger value!*

$$\frac{\partial E}{\partial w_{5,6}} = e_6 * o_5 = -0.0688$$

$$\frac{\partial z_6}{\partial w_{i,6}} = \frac{\partial}{\partial w_{i,6}}(o_4 w_{4,6} + o_5 w_{5,6} + b_6) = \boxed{o_i}$$

# Example: Step 3 – Backward Pass



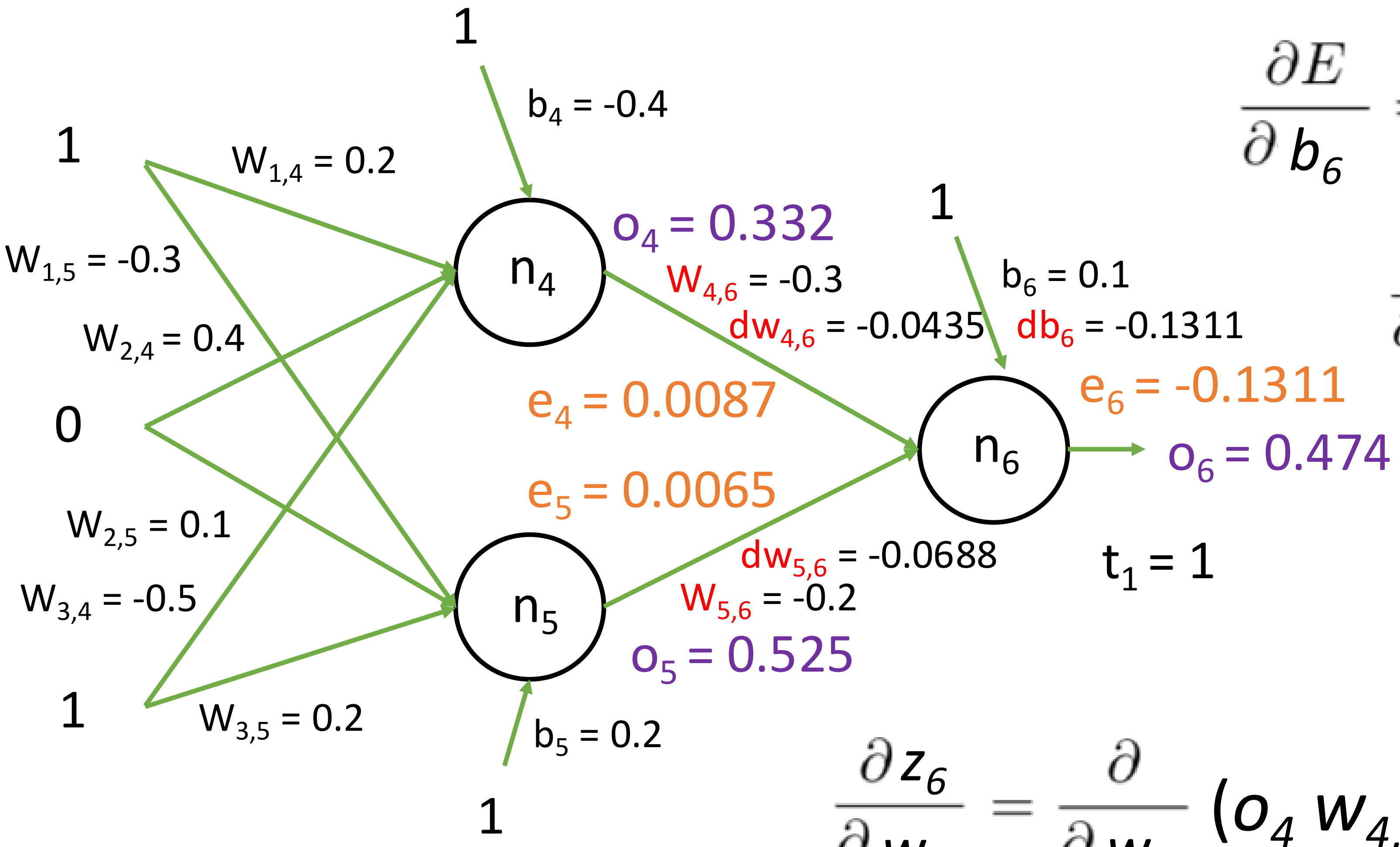

$$\frac{\partial E}{\partial b_6} = \frac{\partial E}{\partial o_6} * \frac{\partial o_6}{\partial z_6} * \frac{\partial z_6}{\partial b_6}$$

$$\frac{\partial E}{\partial b_6} = e_6 * 1 = -0.1311$$

(Intuitively, just passes error along)

$$\frac{\partial z_6}{\partial w_{i,6}} = \frac{\partial}{\partial w_{i,6}}(o_4 w_{4,6} + o_5 w_{5,6} + b_6) = 1$$

# Example: Step 3 – Backward Pass



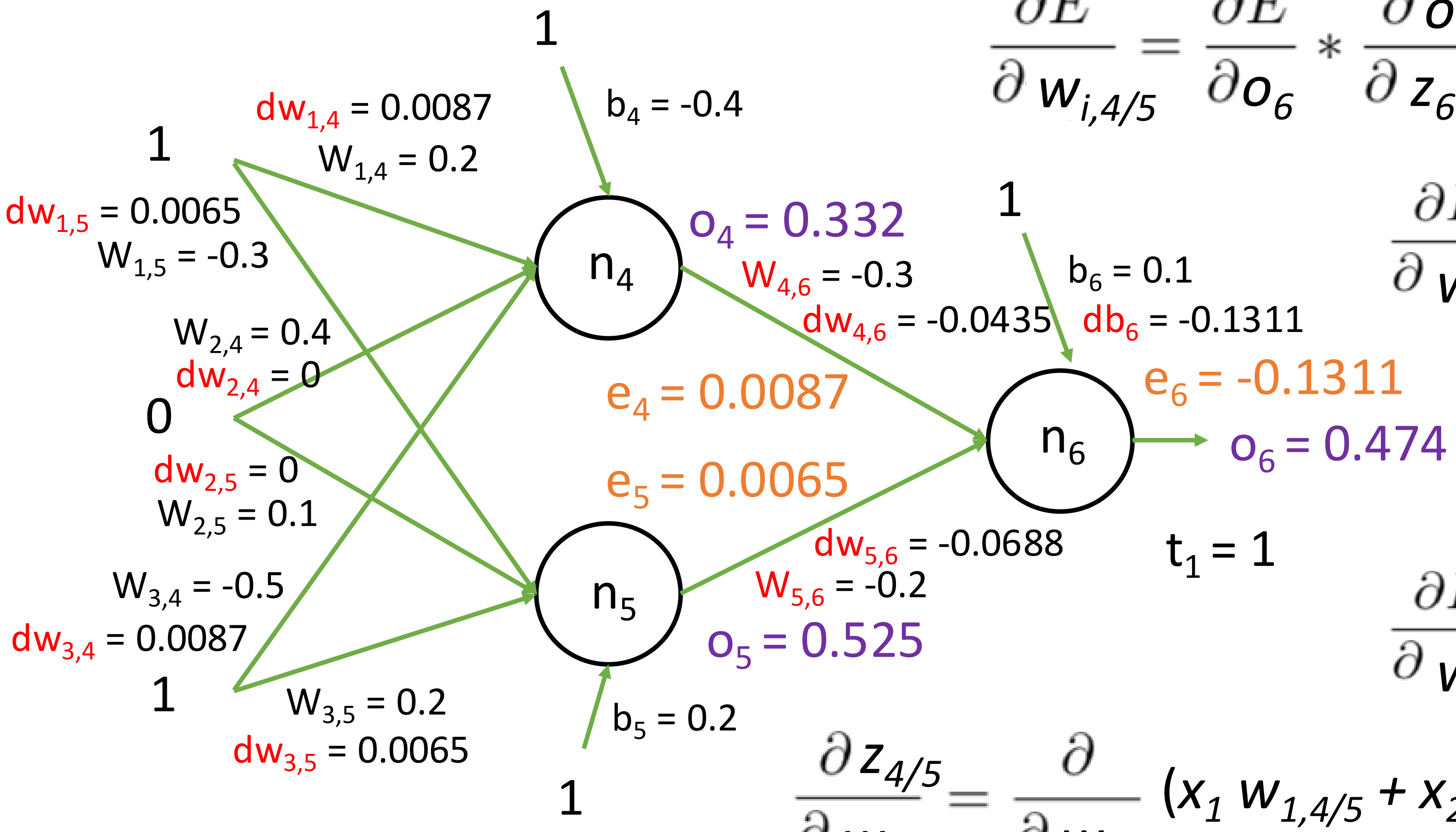

$$\frac{\partial E}{\partial b_6} = \frac{\partial E}{\partial o_6} * \frac{\partial o_6}{\partial z_6} * \frac{\partial z_6}{\partial b_6}$$

$$\frac{\partial E}{\partial b_6} = e_6 * 1 = -0.1311$$

*Recall, gradient points to direction of larger value!*

$$\frac{\partial z_6}{\partial w_{i,6}} = \frac{\partial}{\partial w_{i,6}}(o_4 w_{4,6} + o_5 w_{5,6} + b_6) = 1$$

# Example: Step 3 – Backward Pass




$$\frac{\partial E}{\partial w_{i,4/5}} = \frac{\partial E}{\partial o_6} * \frac{\partial o_6}{\partial z_6} * \frac{\partial z_6}{\partial o_{4/5}} * \frac{\partial o_{4/5}}{\partial z_{4/5}} * \frac{\partial z_{4/5}}{\partial w_{i,4/5}}$$

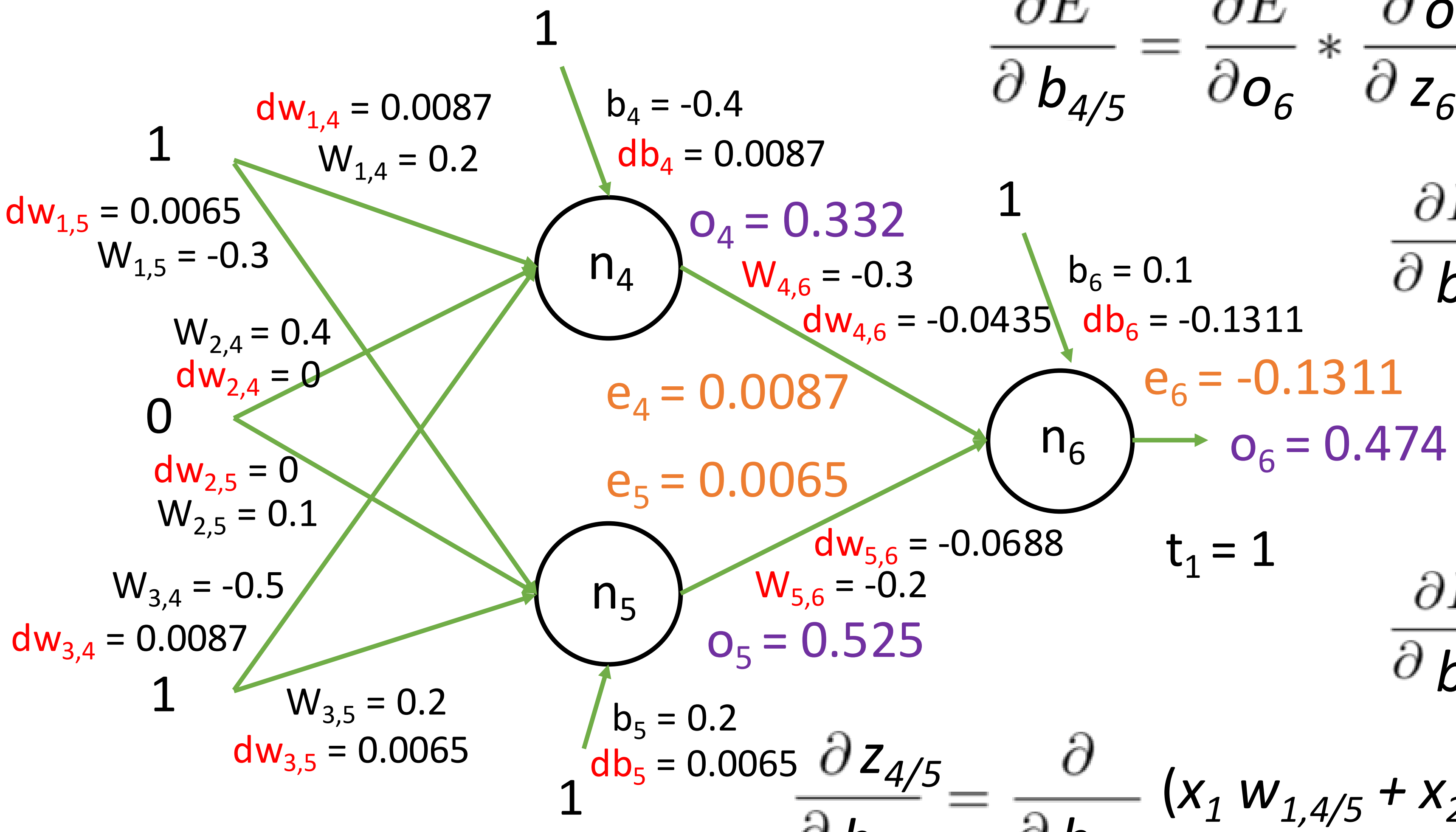
$$\frac{\partial E}{\partial w_{1,4}} = e_4 * x_1 = 0.0087$$

*Recall, gradient points to direction of larger value!*

$$\frac{\partial E}{\partial w_{3,5}} = e_5 * x_3 = 0.0065$$

$$\frac{\partial z_{4/5}}{\partial w_{i,4/5}} = \frac{\partial}{\partial w_{i,4/5}} (x_1 w_{1,4/5} + x_2 w_{2,4/5} + x_3 w_{3,4/5} + b_{4/5}) = x_i$$

# Example: Step 3 – Backward Pass




$$\frac{\partial E}{\partial b_{4/5}} = \frac{\partial E}{\partial o_6} * \frac{\partial o_6}{\partial z_6} * \frac{\partial z_6}{\partial o_{4/5}} * \frac{\partial o_{4/5}}{\partial z_{4/5}} * \frac{\partial z_{4/5}}{\partial b_{4/5}}$$

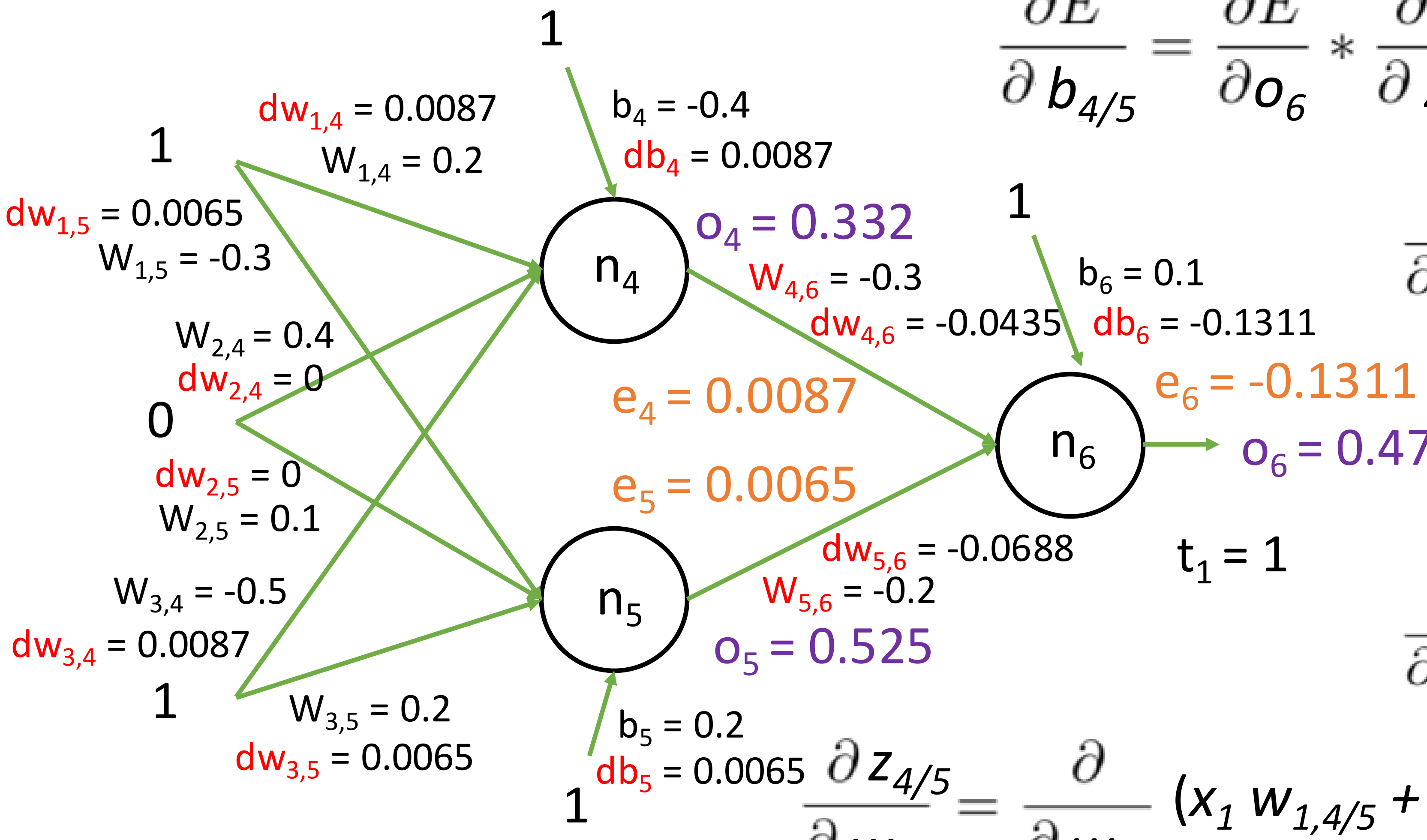
$$\frac{\partial E}{\partial b_4} = e_4 * 1 = 0.0087$$

(Just passes error along)

$$\frac{\partial E}{\partial b_5} = e_5 * 1 = 0.0065$$

$$\frac{\partial z_{4/5}}{\partial b_{4/5}} = \frac{\partial}{\partial b_{4/5}} (x_1 w_{1,4/5} + x_2 w_{2,4/5} + x_3 w_{3,4/5} + b_{4/5}) = 1$$

# Example: Step 3 – Backward Pass




$$\frac{\partial E}{\partial b_{4/5}} = \frac{\partial E}{\partial o_6} * \frac{\partial o_6}{\partial z_6} * \frac{\partial z_6}{\partial o_{4/5}} * \frac{\partial o_{4/5}}{\partial z_{4/5}} * \frac{\partial z_{4/5}}{\partial b_{4/5}}$$

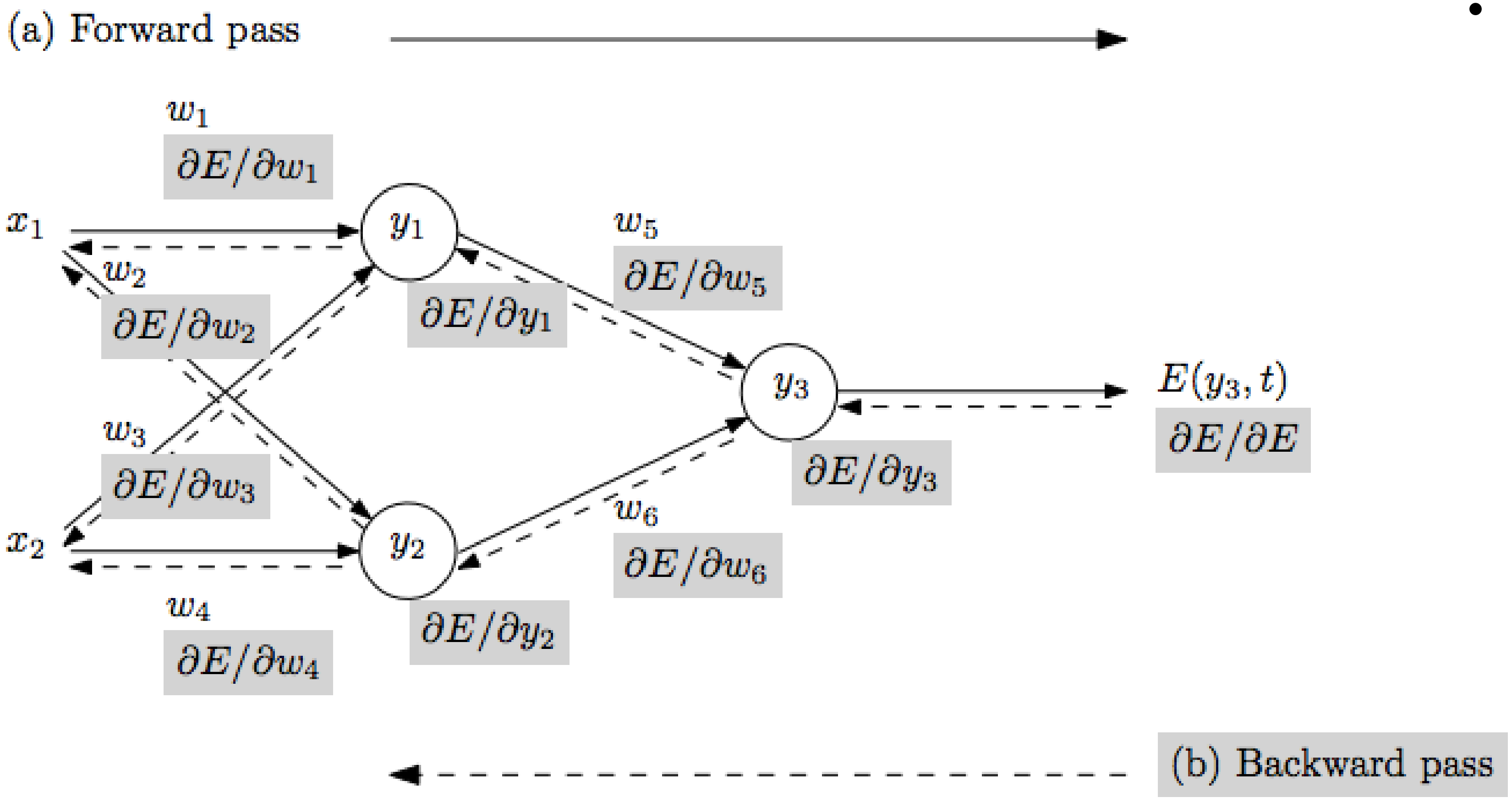
$$\frac{\partial E}{\partial b_4} = e_4 * 1 = 0.0087$$

*Recall, gradient points to direction of larger value!*

$$\frac{\partial E}{\partial b_5} = e_5 * 1 = 0.0065$$

$$\frac{\partial z_{4/5}}{\partial w_{i,4/5}} = \frac{\partial}{\partial w_{i,4/5}}(x_1 w_{1,4/5} + x_2 w_{2,4/5} + x_3 w_{3,4/5} + b_{4/5}) = 1$$

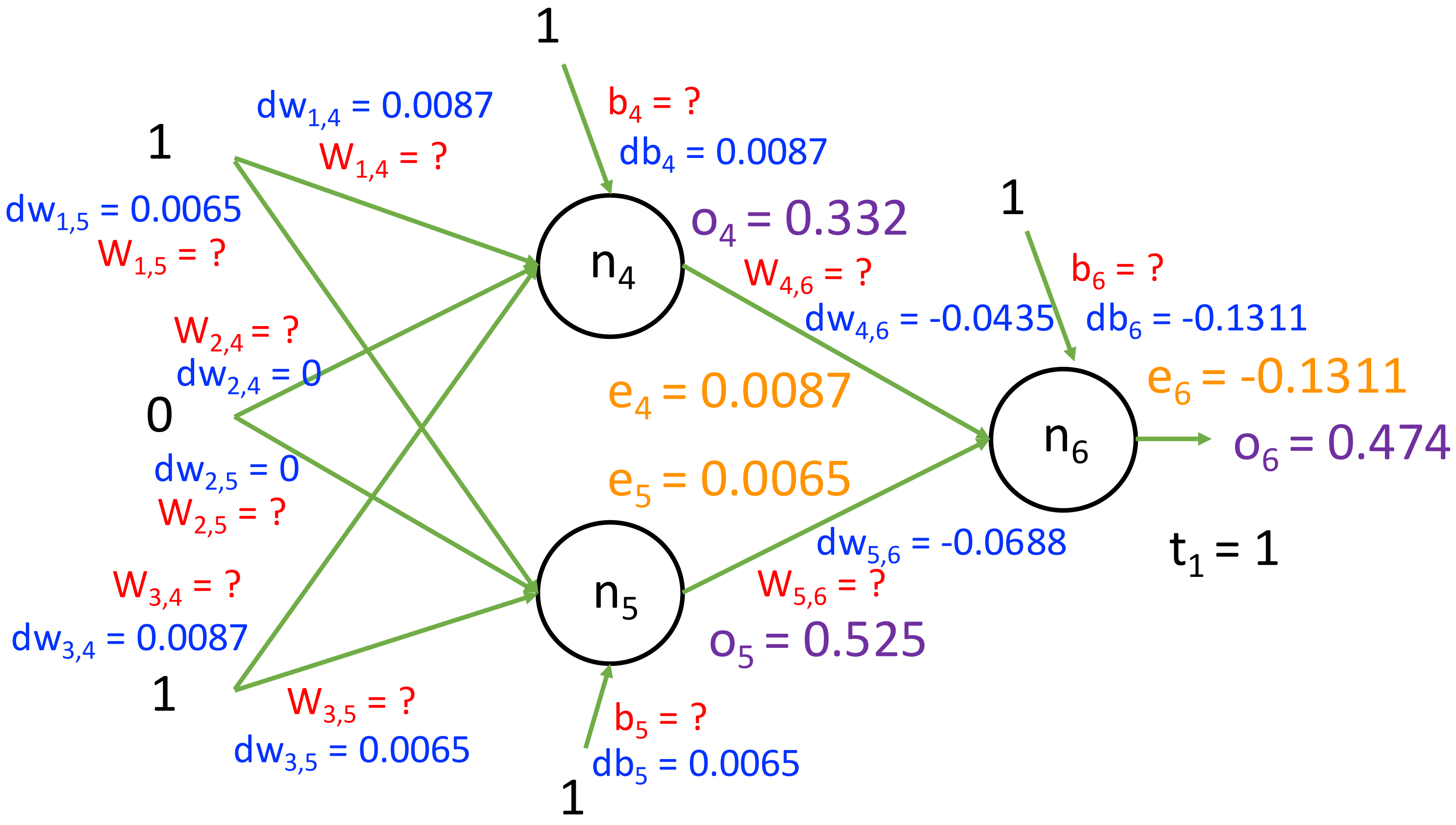
# Example: Step 4 – Update Weights



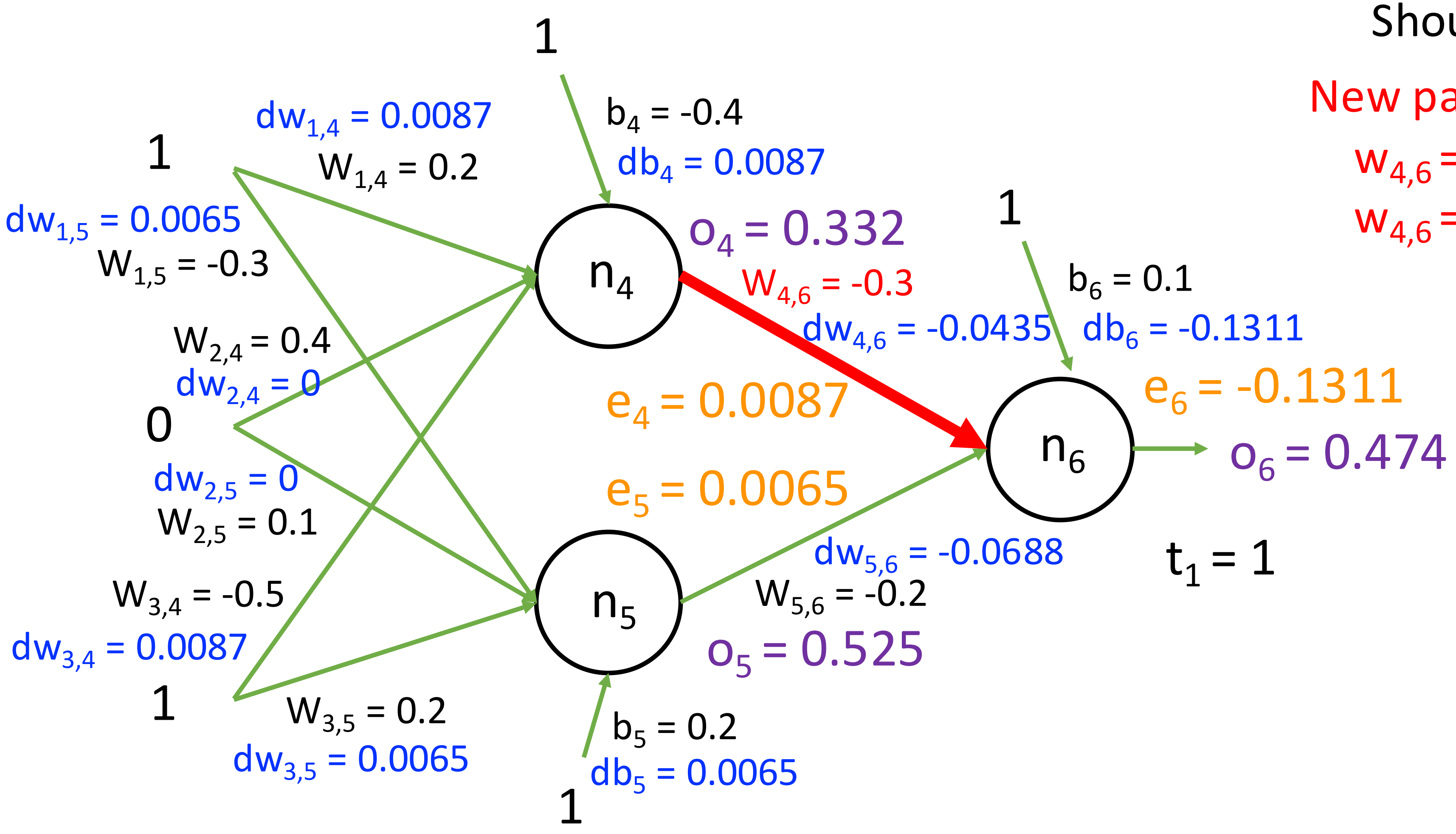

- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make predictions
  2. **Error quantification**: measure error of the model's predictions on training data using a loss function
  3. **Backward pass**: calculate gradients to determine how each model parameter contributed to model error
  4. Update each parameter using calculated gradients

Baydin et al. Automatic Differentiation in Machine Learning: a Survey. 2018

# Example: Step 4 – Update Weights



# Example: Step 4 – Update Weights



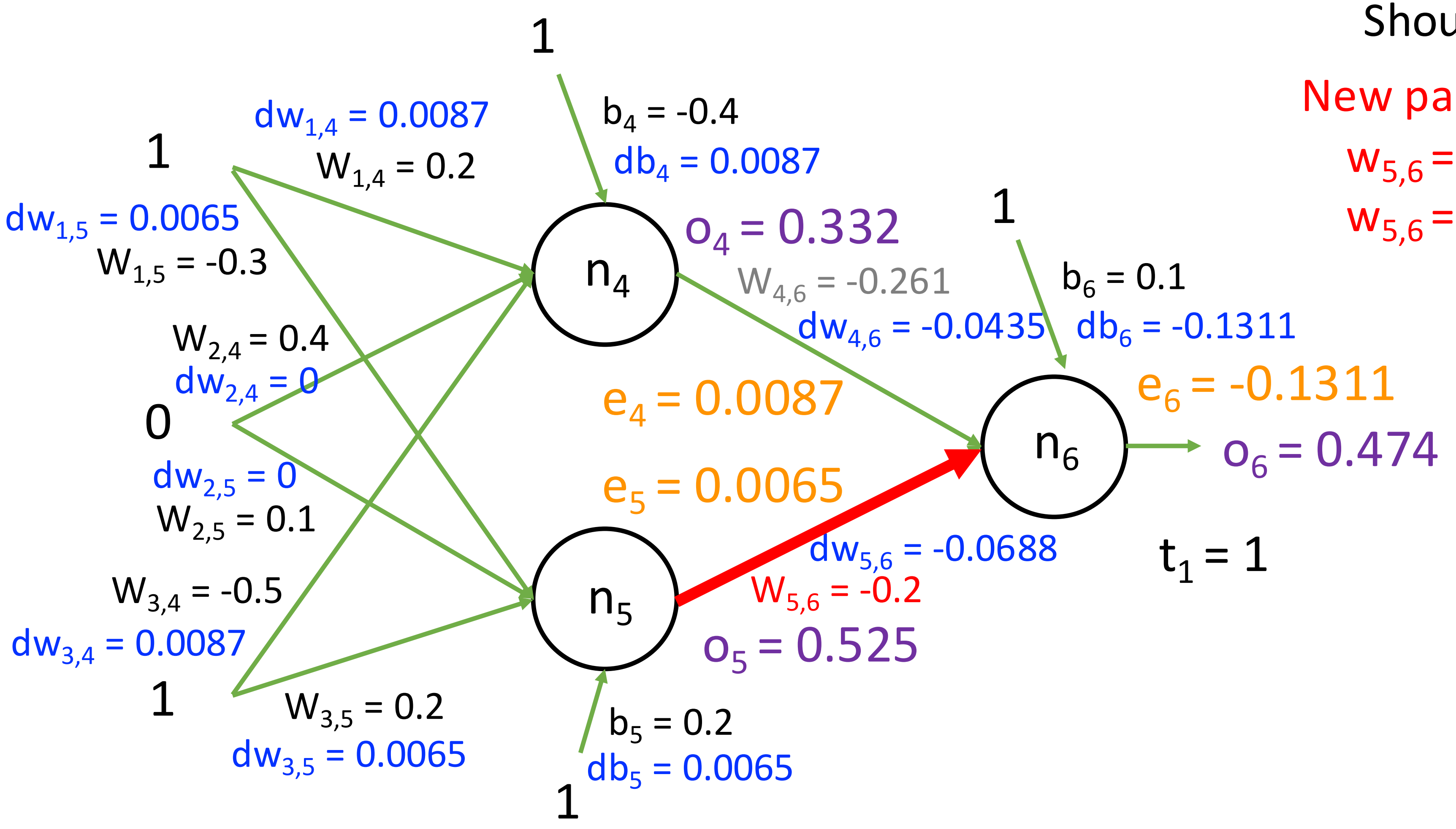

Should $w_{4,6}$ increase or decrease?

New parameters (learning rate = 0.9):

$w_{4,6} = -0.3 - 0.9 * -0.0435$

$w_{4,6} = -0.261$

# Example: Step 4 – Update Weights



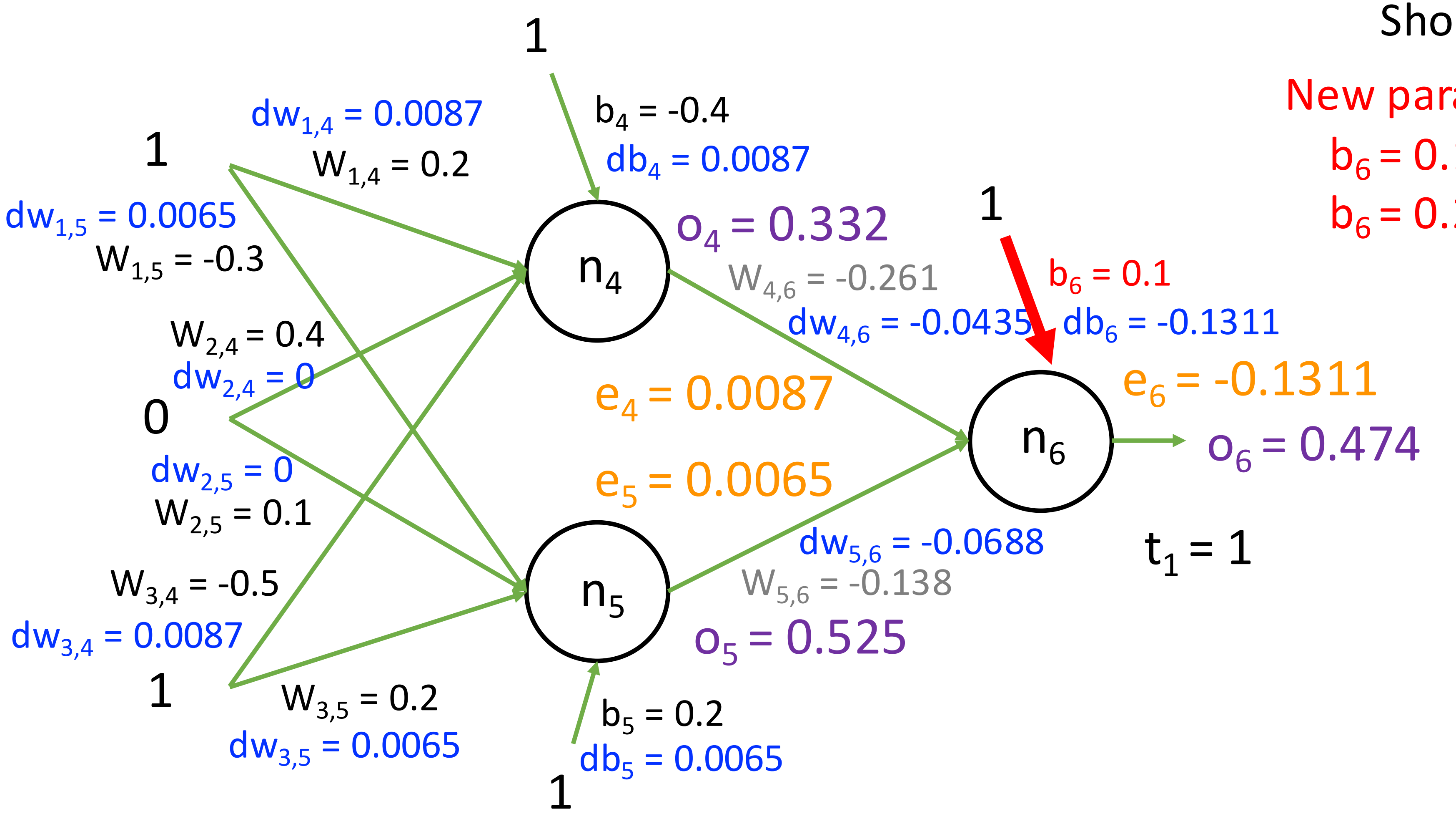

Should $w_{5,6}$ increase or decrease?

New parameters (learning rate = 0.9):

$w_{5,6} = -0.2 - 0.9 * -0.0688$

$w_{5,6} = -0.138$

# Example: Step 4 – Update Weights



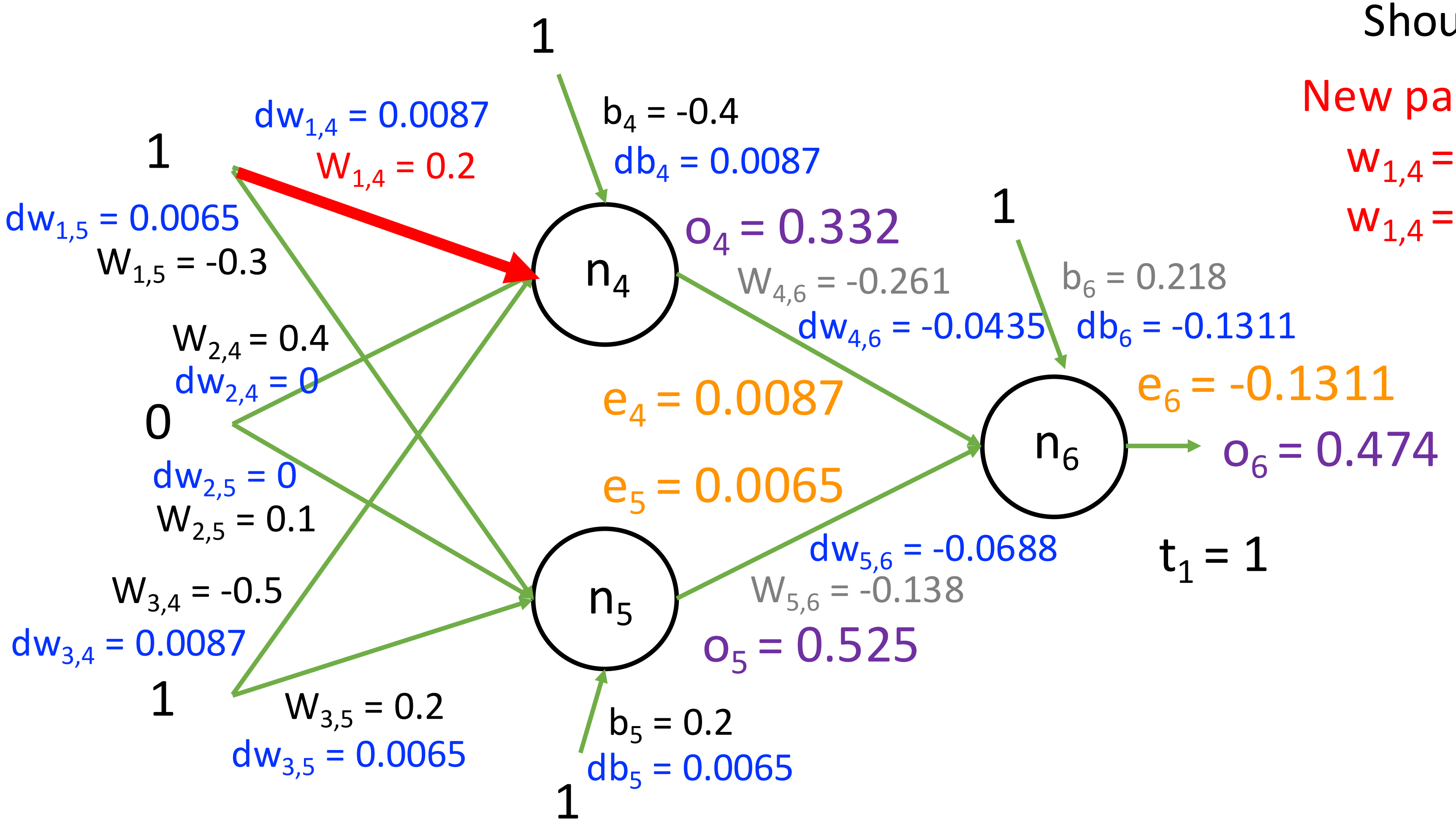

Should $b_6$ increase or decrease?

New parameters (learning rate = 0.9):

$b_6 = 0.1 - 0.9 * -0.1311$

$b_6 = 0.218$

# Example: Step 4 – Update Weights



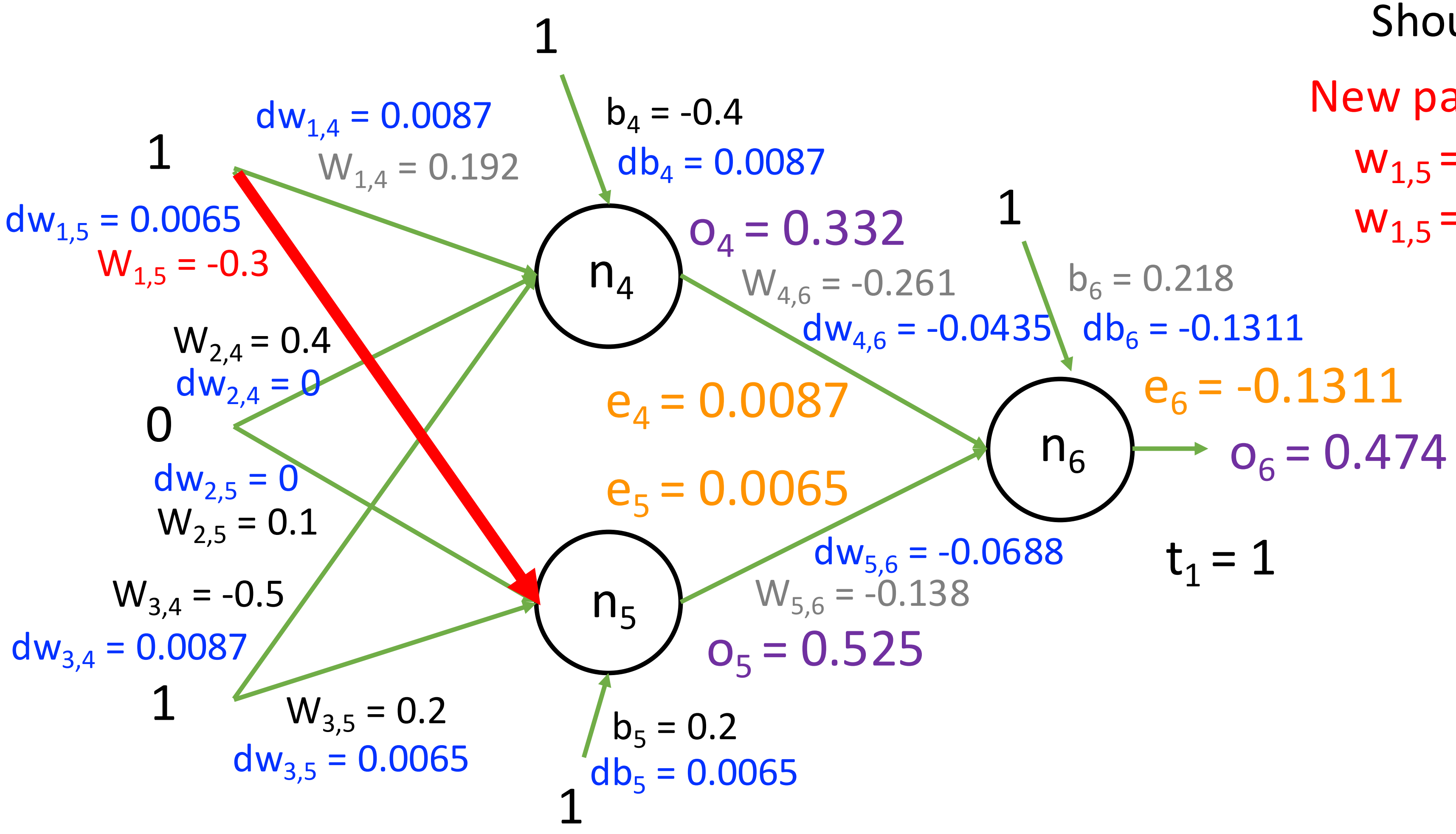

Should $w_{1,4}$ increase or decrease?

New parameters (learning rate = 0.9):

$w_{1,4} = 0.2 - 0.9 * 0.0087$

$w_{1,4} = 0.192$

# Example: Step 4 – Update Weights



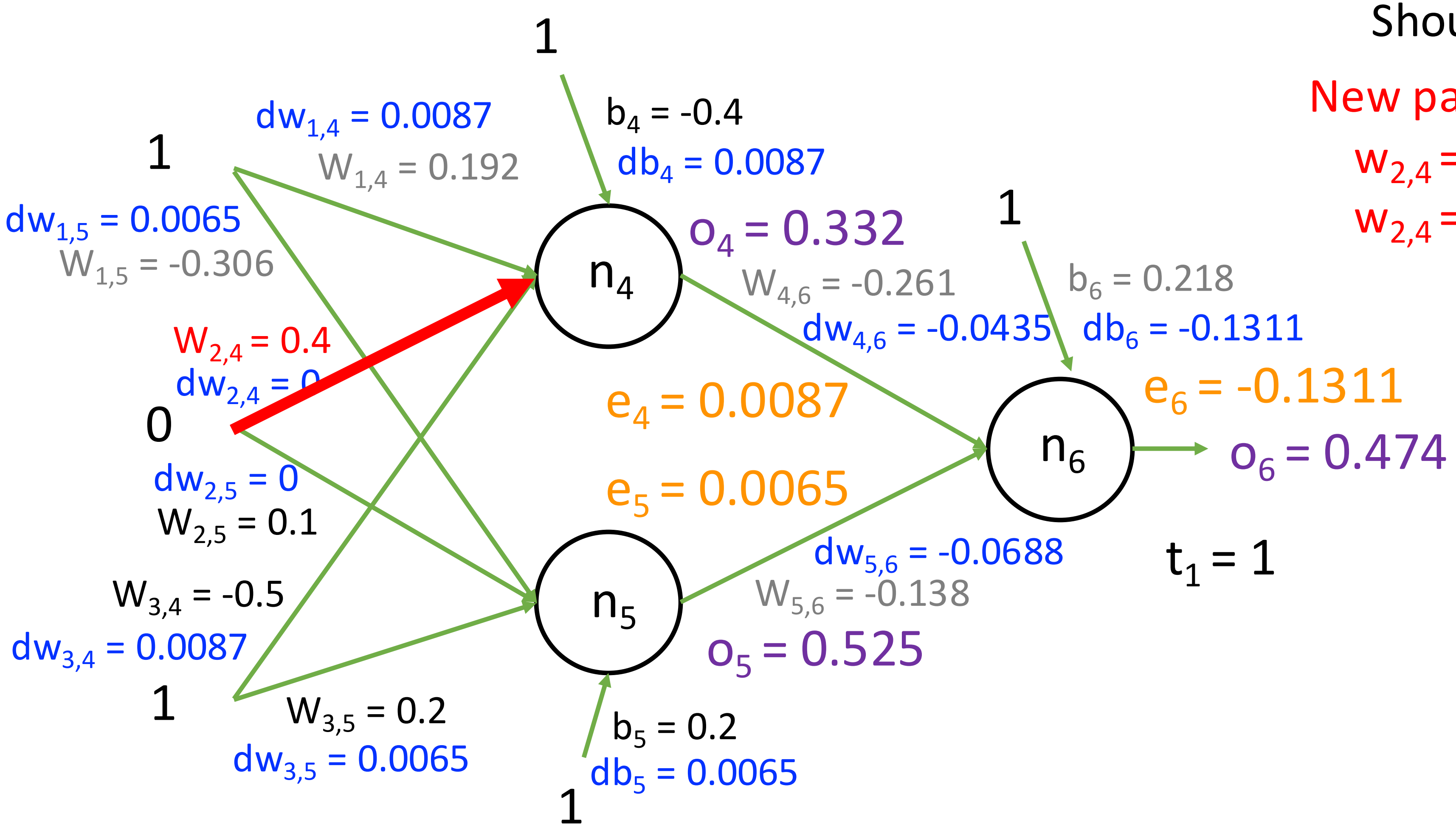

Should $w_{1,5}$ increase or decrease?

New parameters (learning rate = 0.9):

$w_{1,5}$ = -0.3 - 0.9 * 0.0065

$w_{1,5}$ = -0.306

# Example: Step 4 – Update Weights



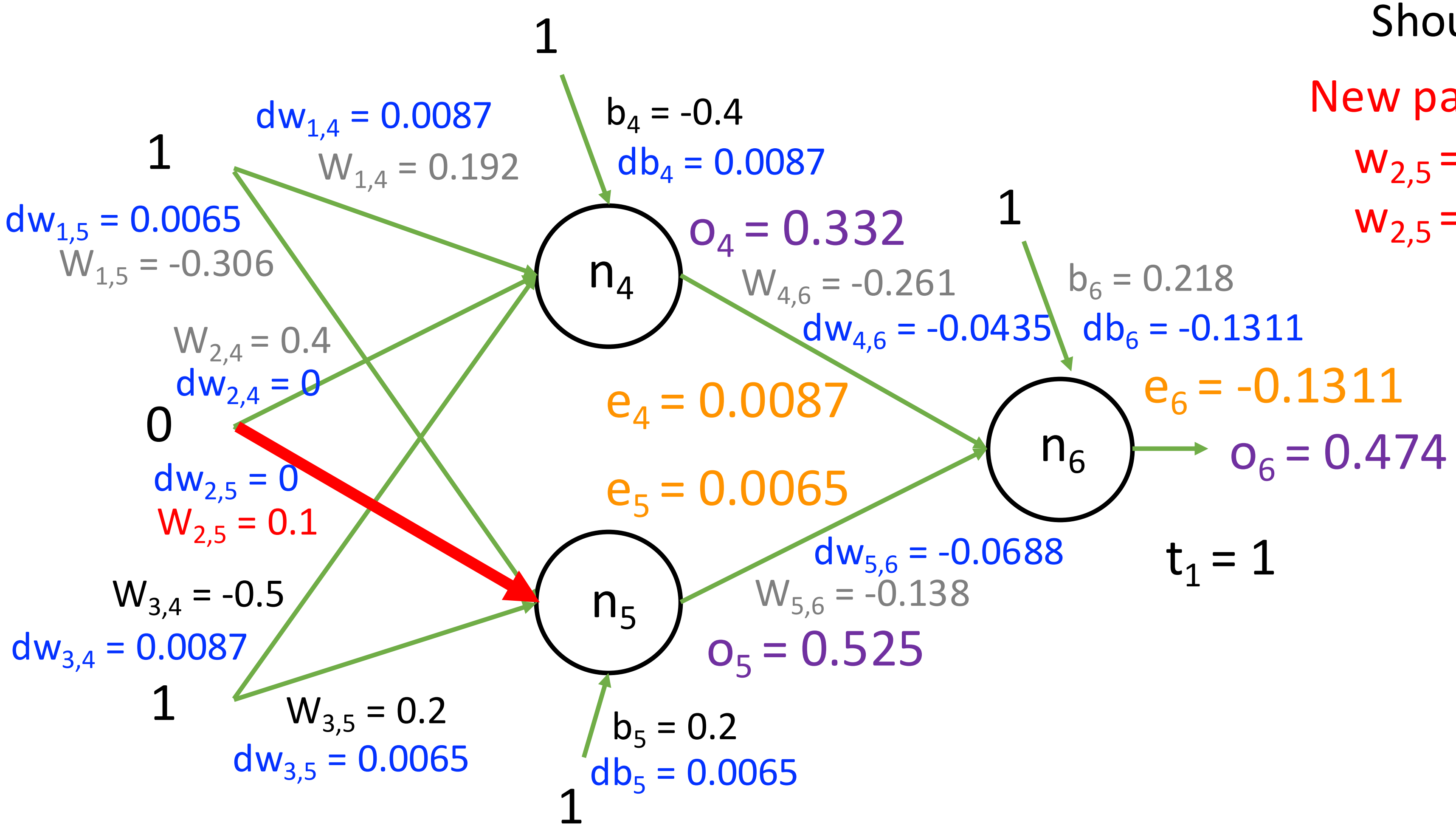

Should $w_{2,4}$ increase or decrease?

New parameters (learning rate = 0.9):

$w_{2,4} = 0.4 - 0.9 * 0$

$w_{2,4} = 0.4$

# Example: Step 4 – Update Weights



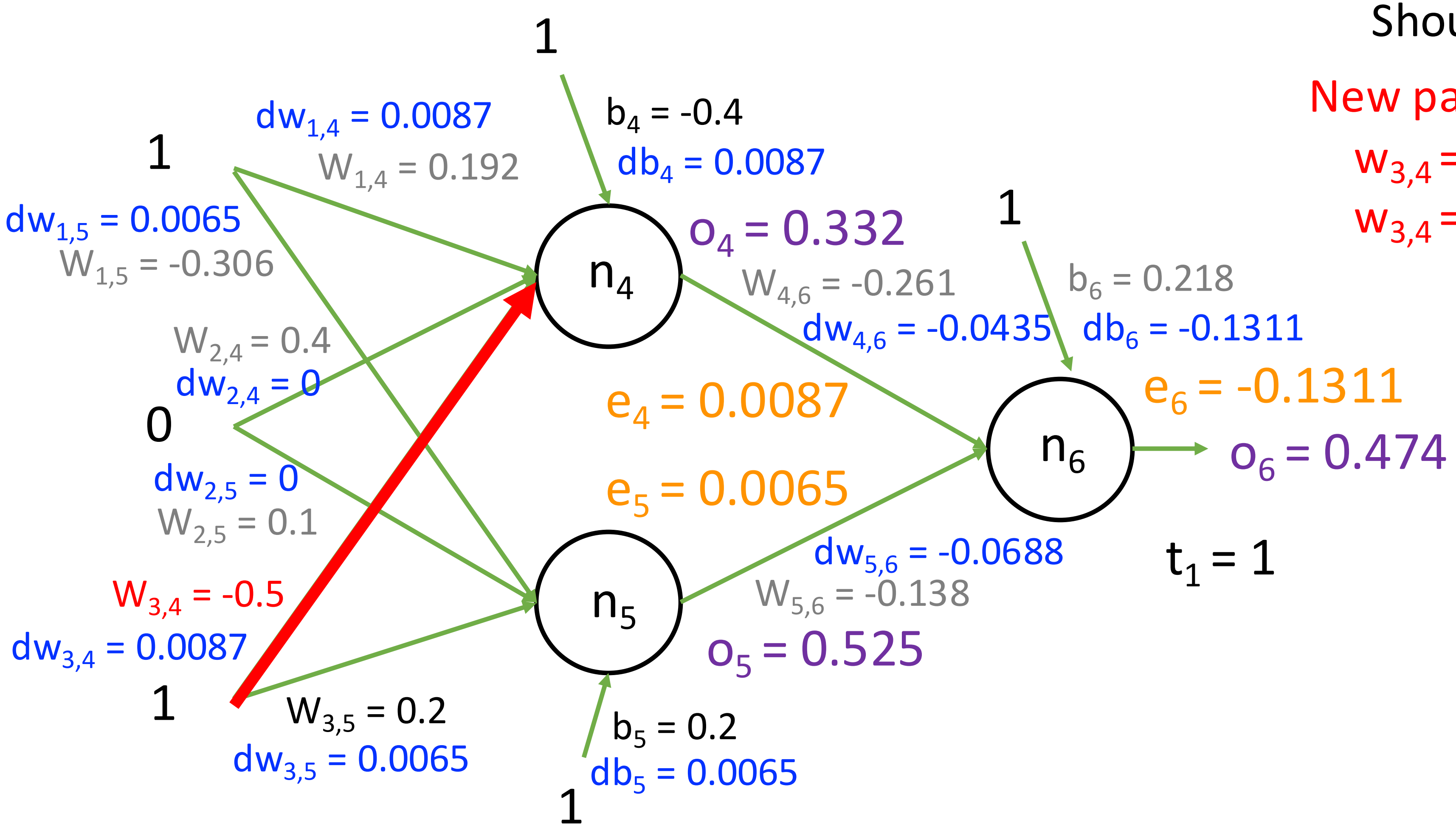

Should $w_{2,5}$ increase or decrease?

New parameters (learning rate = 0.9):

$w_{2,5} = 0.1 - 0.9 * 0$

$w_{2,5} = 0.1$

# Example: Step 4 – Update Weights



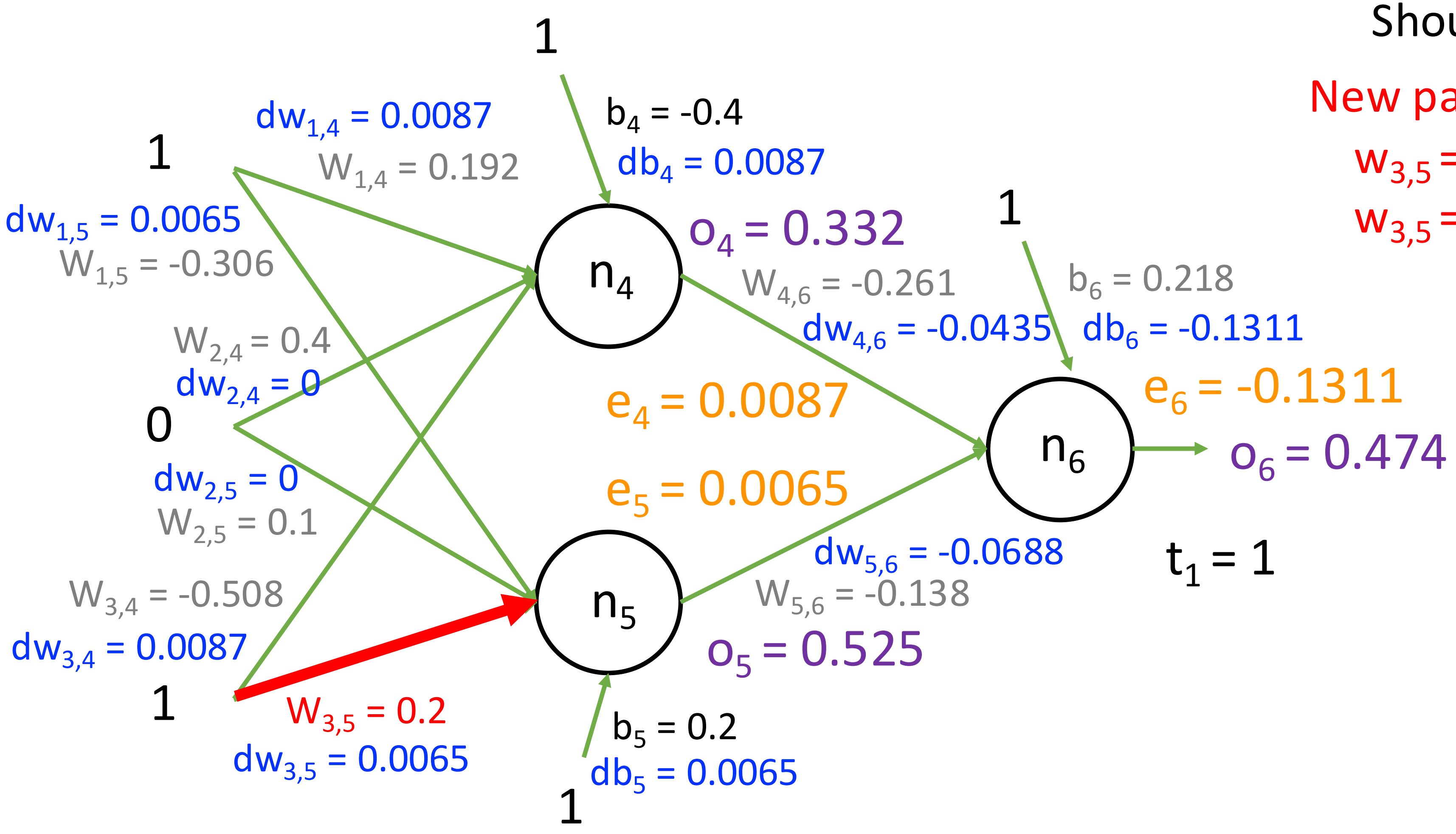

Should $w_{3,4}$ increase or decrease?

New parameters (learning rate = 0.9):

$w_{3,4}$ = -0.5 - 0.9 * 0.0087

$w_{3,4}$ = -0.508

# Example: Step 4 – Update Weights



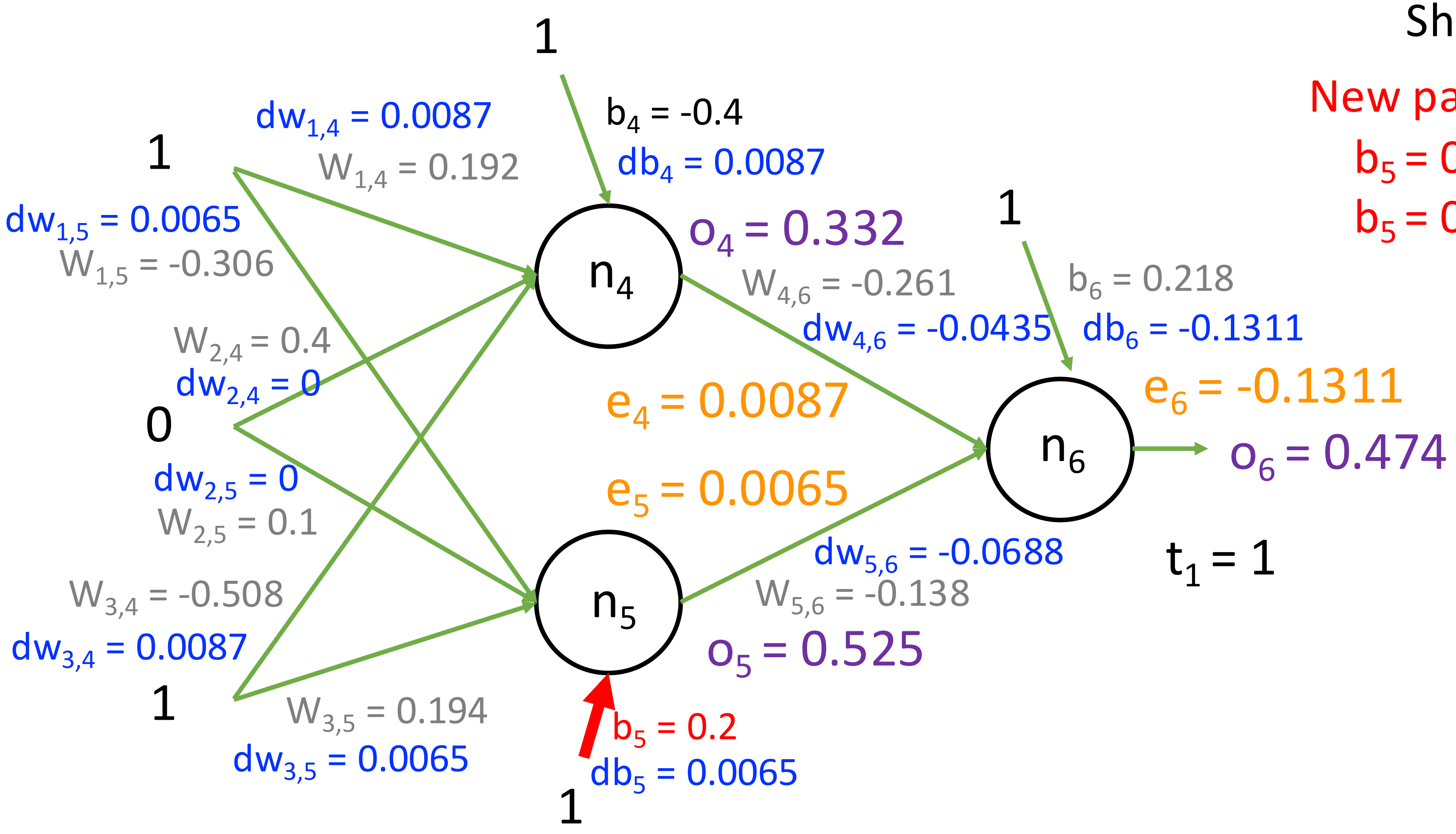

Should $w_{3,5}$ increase or decrease?

New parameters (learning rate = 0.9):

$w_{3,5} = 0.2 - 0.9 * 0.0065$

$w_{3,5} = 0.194$

# Example: Step 4 – Update Weights



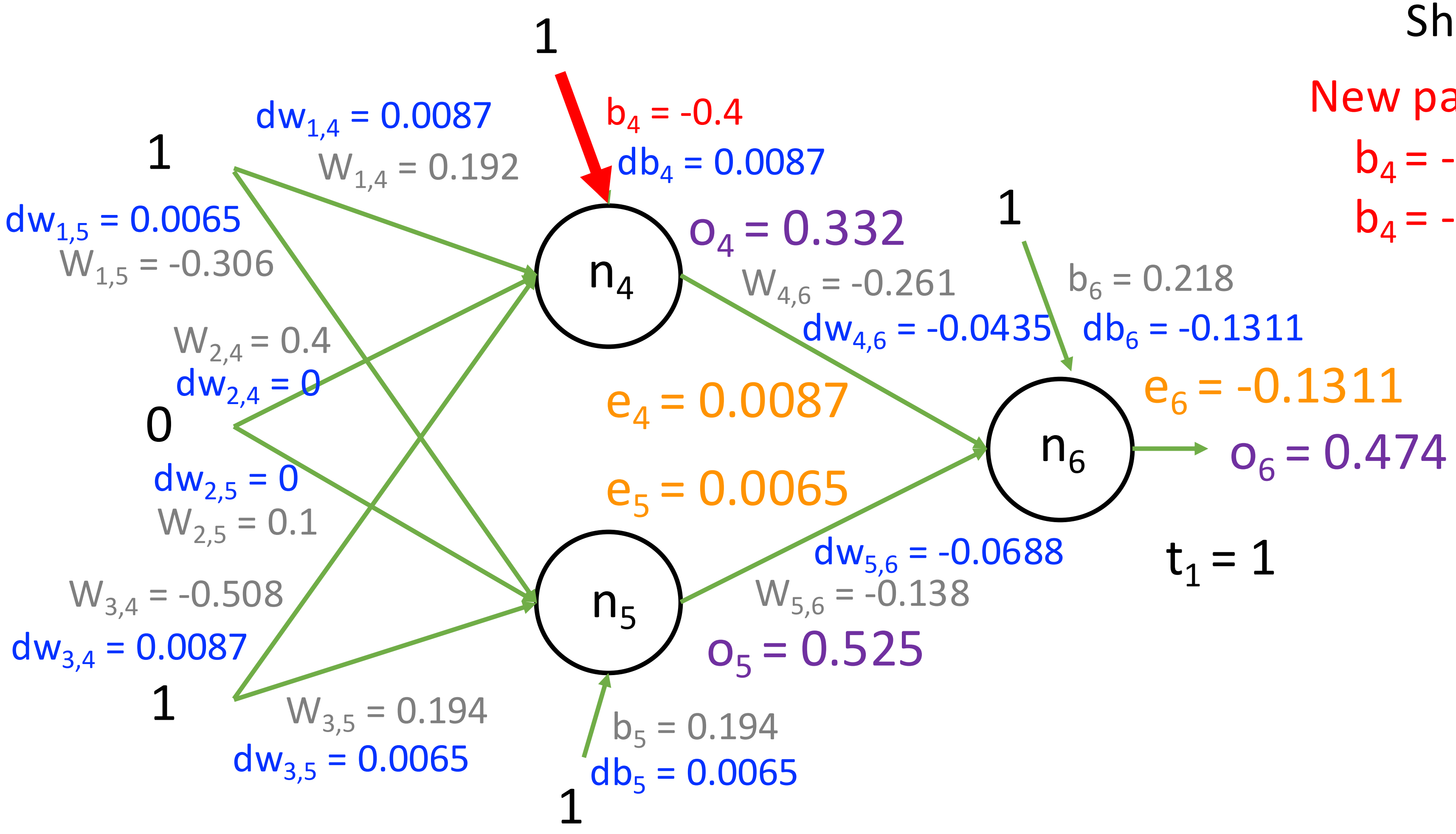

Should $b_5$ increase or decrease?

New parameters (learning rate = 0.9):

$b_5 = 0.2 - 0.9 * 0.0065$

$b_5 = 0.194$

# Example: Step 4 – Update Weights



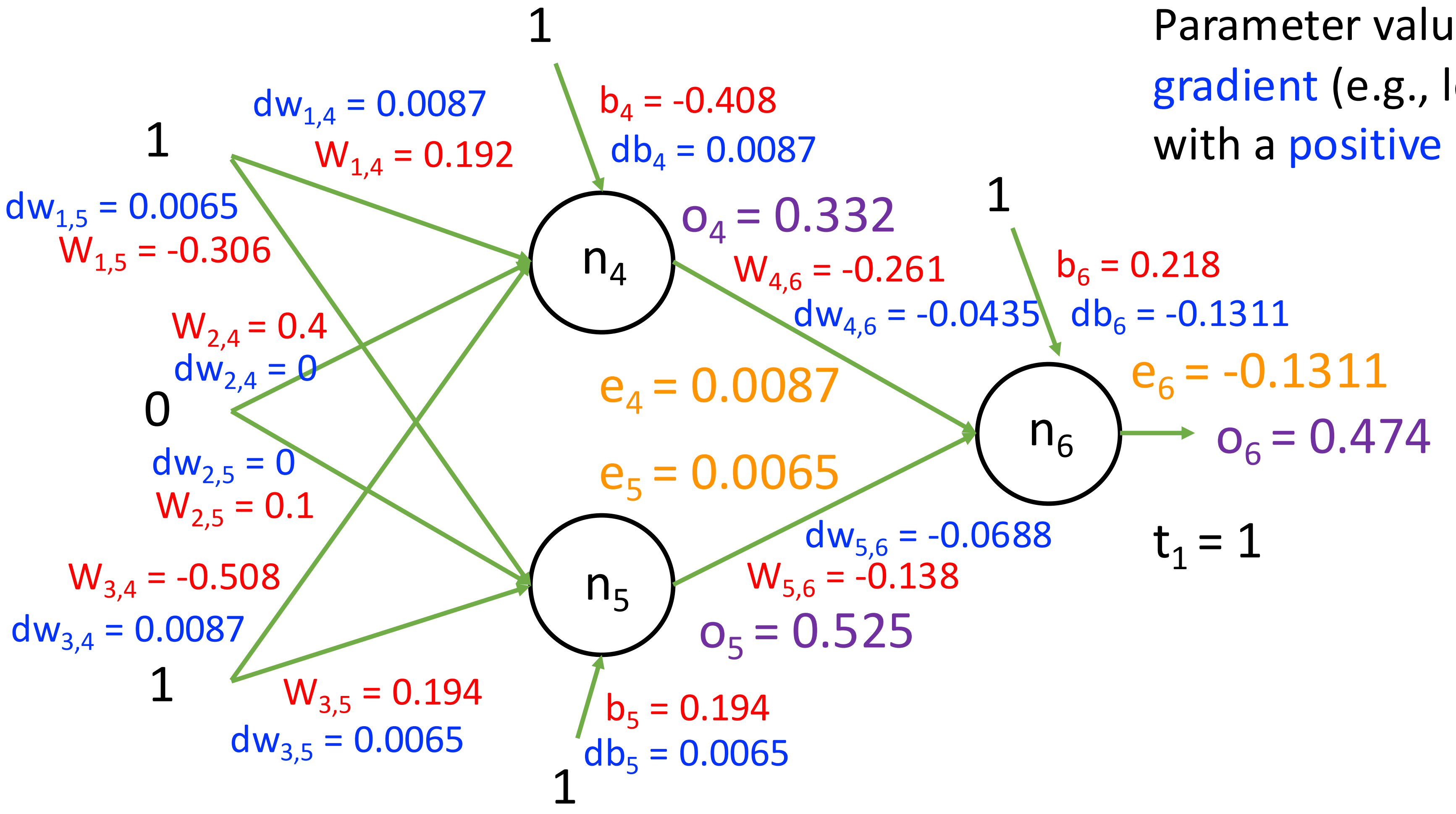

Should $b_4$ increase or decrease?

New parameters (learning rate = 0.9):

$b_4 = -0.4 - 0.9 * 0.0087$

$b_4 = -0.408$

# Example: Repeat Steps 1-4 With New Samples



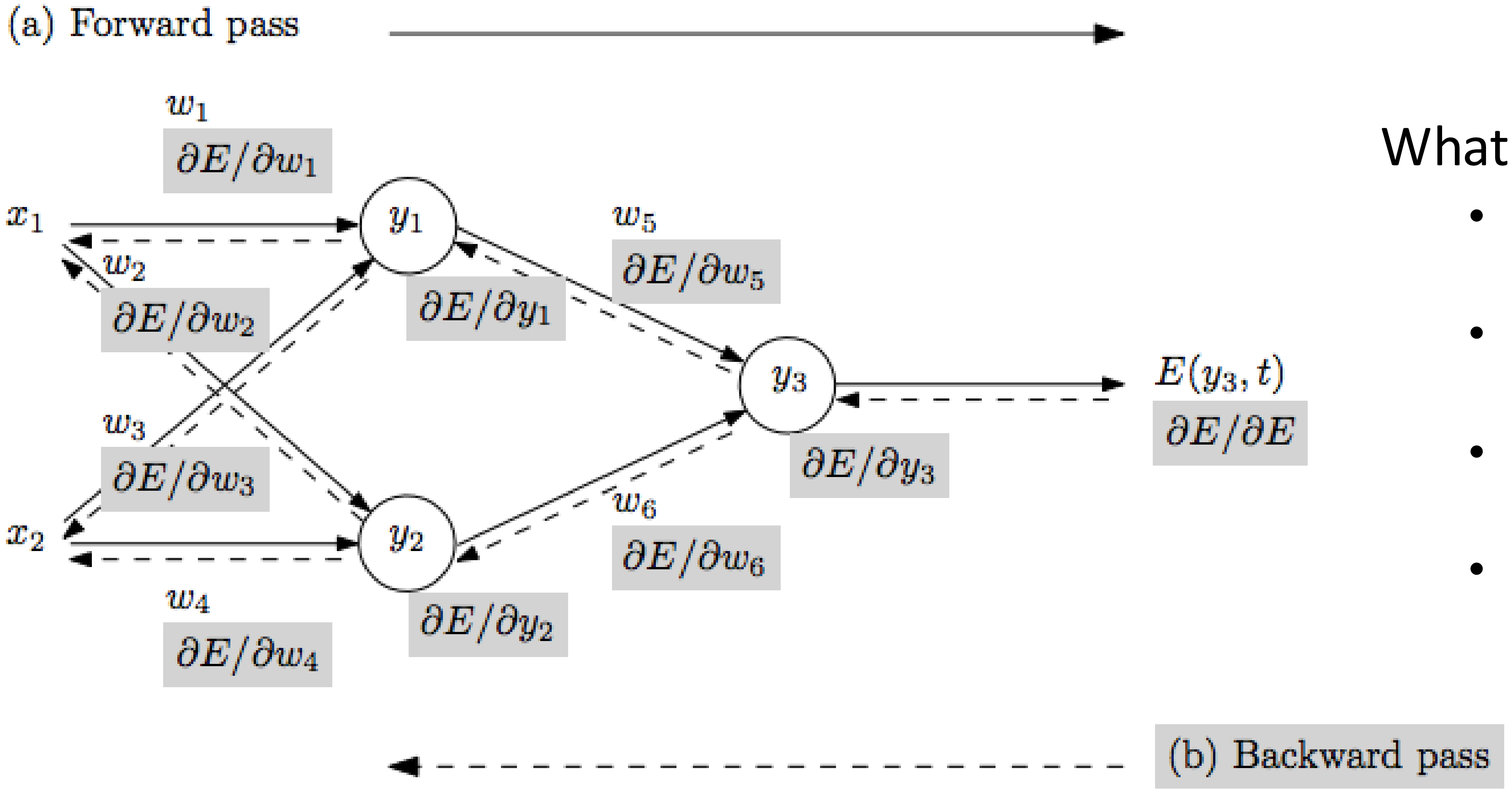

Parameter values **increase** with a negative gradient (e.g., less negative) and **decrease** with a positive gradient (e.g., more negative)

# Example: Completed Training Example For…

- Binary classification: predict if a student will get a B or better (minimum required for CS graduate student in this course)

- Inputs:
    - Fraction of assignments completed
    - Fraction of readings read
    - Fraction of lectures watched

# When to Stop Training Neural Networks?




What stopping criterion to use?

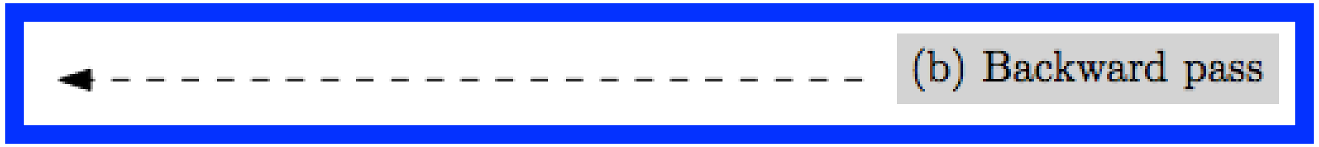
- Weight changes are incredibly small
- Finished a pre-specified number of epochs
- Percentage of misclassified example is below some threshold
- …

# Training Summary: How Neural Networks Learn

Calculating gradients depends on:

1) Objective/loss function

2) Activation functions




- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through model to make predictions
  2. **Error quantification**: measure error of the model's predictions on training data using a loss function
  3. **Backward pass**: calculate gradients to determine how each model parameter contributed to model error
  4. Update each parameter using calculated gradients

Baydin et al. Automatic Differentiation in Machine Learning: a Survey. 2018

# Today’s Topics

- Gradient descent: how neural networks learn
- Mathematical foundation of gradient descent: derivatives
- Applying gradient descent to train neural networks
- Training example

The End