Section 8

Admin

- Office Hours: Wednesday 1-2PM!
- Homework 3: **March 2nd**
- Course and Section Feedback?

Agenda

- Backpropagation (50 min!)
  - Questions 1, 2, 3

Backpropagation

Prelude: Chain Rule

A quick, very related aside...

Jacobian Matrices:

- Definition:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^T f_1 \\ \vdots \\ \nabla^T f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

- For some intuition think of $f\colon \mathbb{R}^n \to \mathbb{R}^m$ as a family of m functions that maps the input space to the output $f_1, \ldots, f_m$. The Jacobian matrix is just the transposed gradient of these functions.

m functions over
n inputs

Chain Rule (1a)

Let $f : \mathbb{R}^n \to \mathbb{R}^m$, $g : \mathbb{R}^\ell \to \mathbb{R}^n$. Write the Jacobian of $f \circ g$ in terms of the Jacobian $\frac{\partial f}{\partial y}$ of f and the Jacobian $\frac{\partial g}{\partial x}$ of g. Make sure the matrix dimensions line up. What conditions must hold in order for this formula to make sense?

Jacobian Matrix:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^T f_1 \\ \vdots \\ \nabla^T f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Chain Rule (1a)

$f(x)$ $x = [x_1, \ldots, x_n]$

$\frac{\partial f}{\partial x}$, think: 1D

Let $f : \mathbb{R}^n \to \mathbb{R}^m$, $g : \mathbb{R}^\ell \to \mathbb{R}^n$. Write the Jacobian of $f \circ g$ in terms of the Jacobian $\frac{\partial f}{\partial y}$ of f and the Jacobian $\frac{\partial g}{\partial x}$ of g. Make sure the matrix dimensions line up. What conditions must hold in order for this formula to make sense?

The Chain Rule theorem states that:

$$\frac{\partial (f \circ g)}{\partial x}(x) = \frac{\partial f}{\partial y}(g(x)) \cdot \frac{\partial g}{\partial x}(x)$$

$f(x,y)$

In order for the dimensions to line up for matrix multiplication, we must have $\frac{\partial f}{\partial y} \in \mathbb{R}^{m \times n}$ and $\frac{\partial g}{\partial x} \in \mathbb{R}^{n \times \ell}$. Note that by this convention, the gradient of a scalar function is a *row* vector:

$$\frac{\partial f}{\partial y}(y) = \begin{bmatrix} \frac{\partial f_1}{\partial y_1}(y) & \cdots & \frac{\partial f_1}{\partial y_n}(y) \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial y_1}(y) & \cdots & \frac{\partial f_m}{\partial y_n}(y) \end{bmatrix}$$

In order to apply the chain rule, f must be differentiable at $g(x)$ and g must be differentiable at x.

Chain Rule (1b)

What if instead the input of g is a matrix $W \in \mathbb{R}^{p \times q}$? Can we still write the derivative $\frac{\partial g}{\partial W}$ of g as a matrix?

Jacobian Matrix:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^T f_1 \\ \vdots \\ \nabla^T f_m \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Chain Rule (1b)

What if instead the input of g is a matrix $W \in \mathbb{R}^{p \times q}$? Can we still write the derivative $\frac{\partial g}{\partial W}$ of g as a matrix?

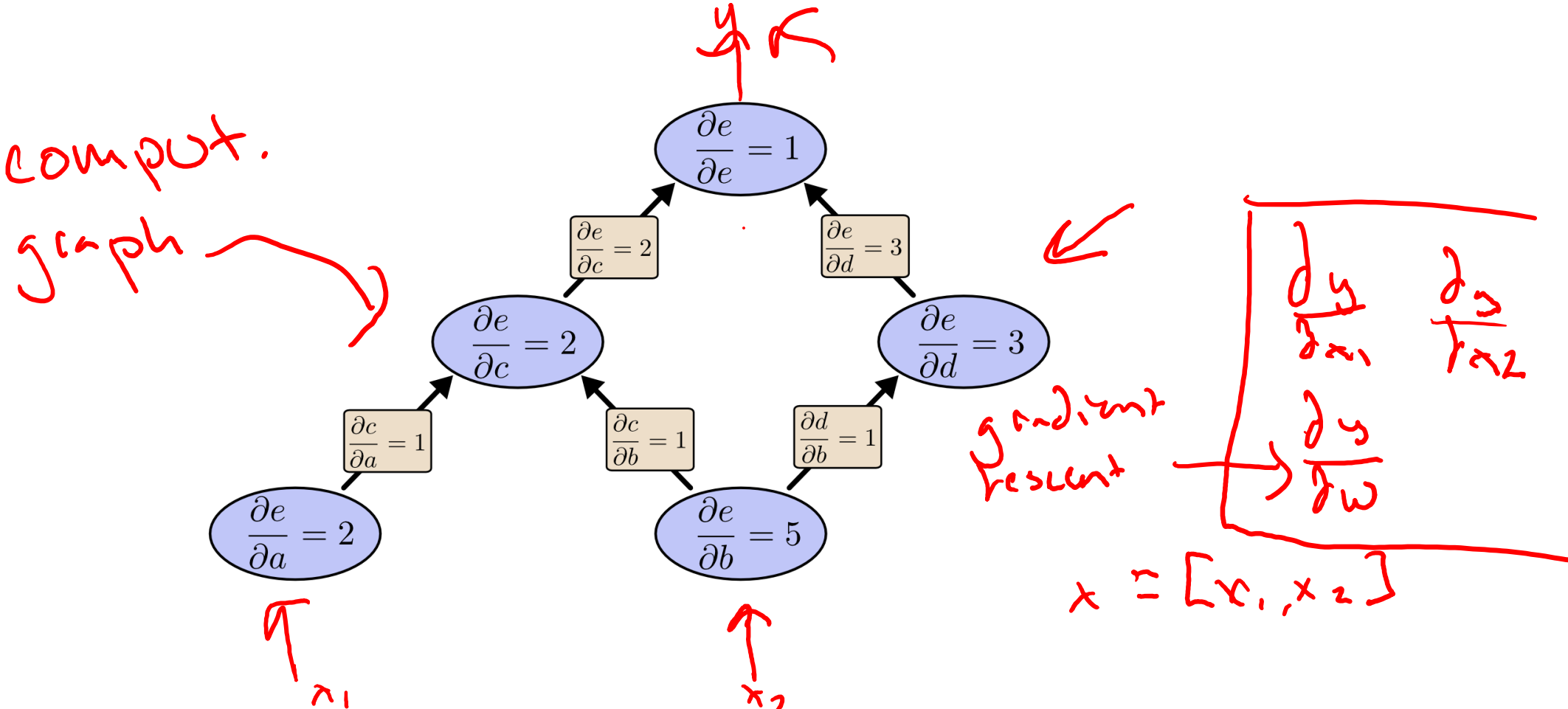
No, we cannot. The derivative of $g : \mathbb{R}^{p \times q} \to \mathbb{R}^n$ would be a three-dimensional $n \times p \times q$ tensor. In practice, people often *flatten* the input matrix W to a vector $vec(W) \in \mathbb{R}^{pq}$. Then we can write the derivative of g as a Jacobian matrix, $\frac{\partial g}{\partial vec(W)} \in \mathbb{R}^{n \times pq}$. Then we must remember to un-flatten the derivative later when we update the matrix W.

↳ Chain rule works "pretty much" as expected in the multidim. case

Back to Backpropagation

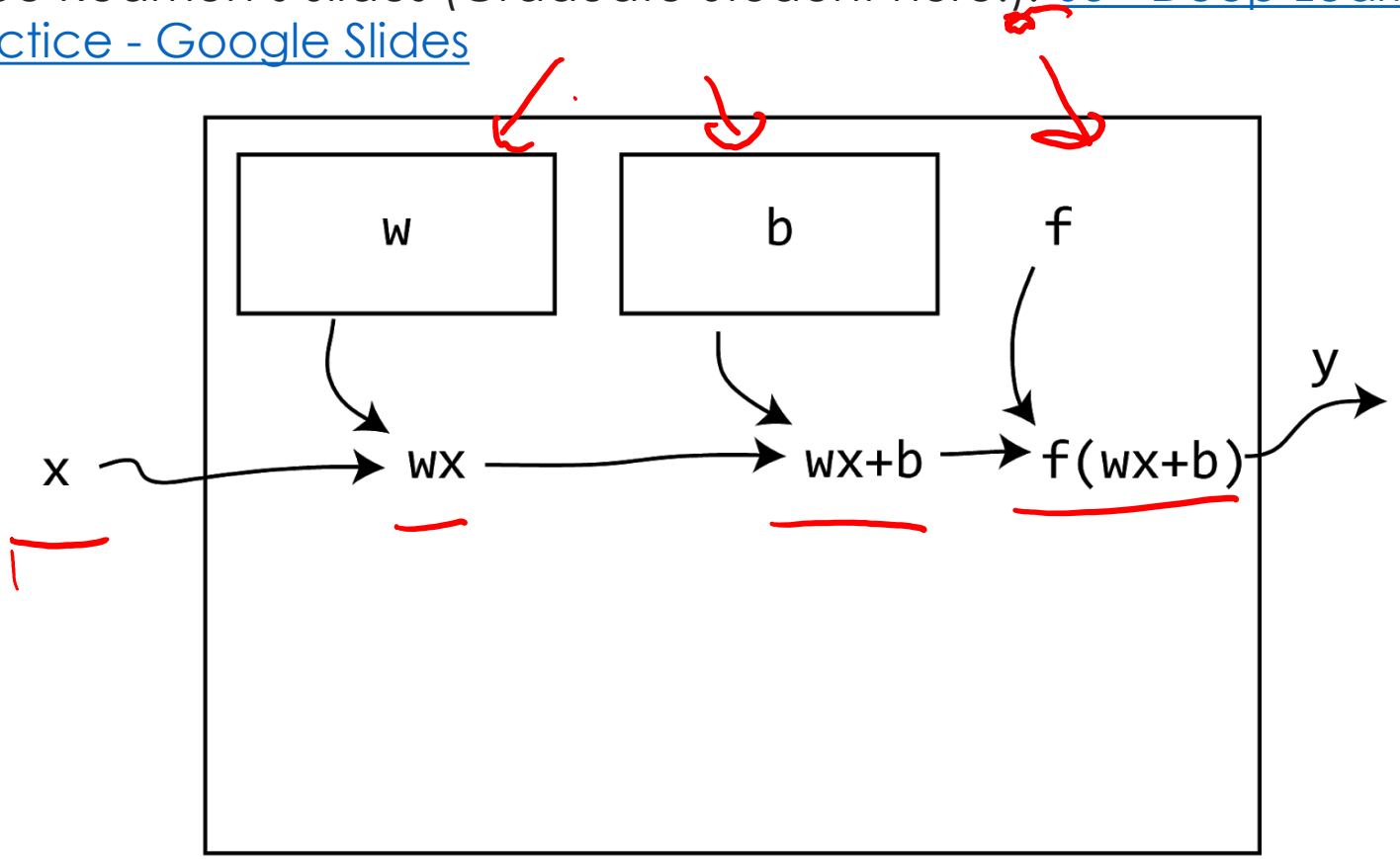
Backpropagation – Intuition

- "Backpropagation is the key algorithm that makes training deep model computationally tractable" ([Calculus on Computational Graphs: Backpropagation -- colah's blog](#))

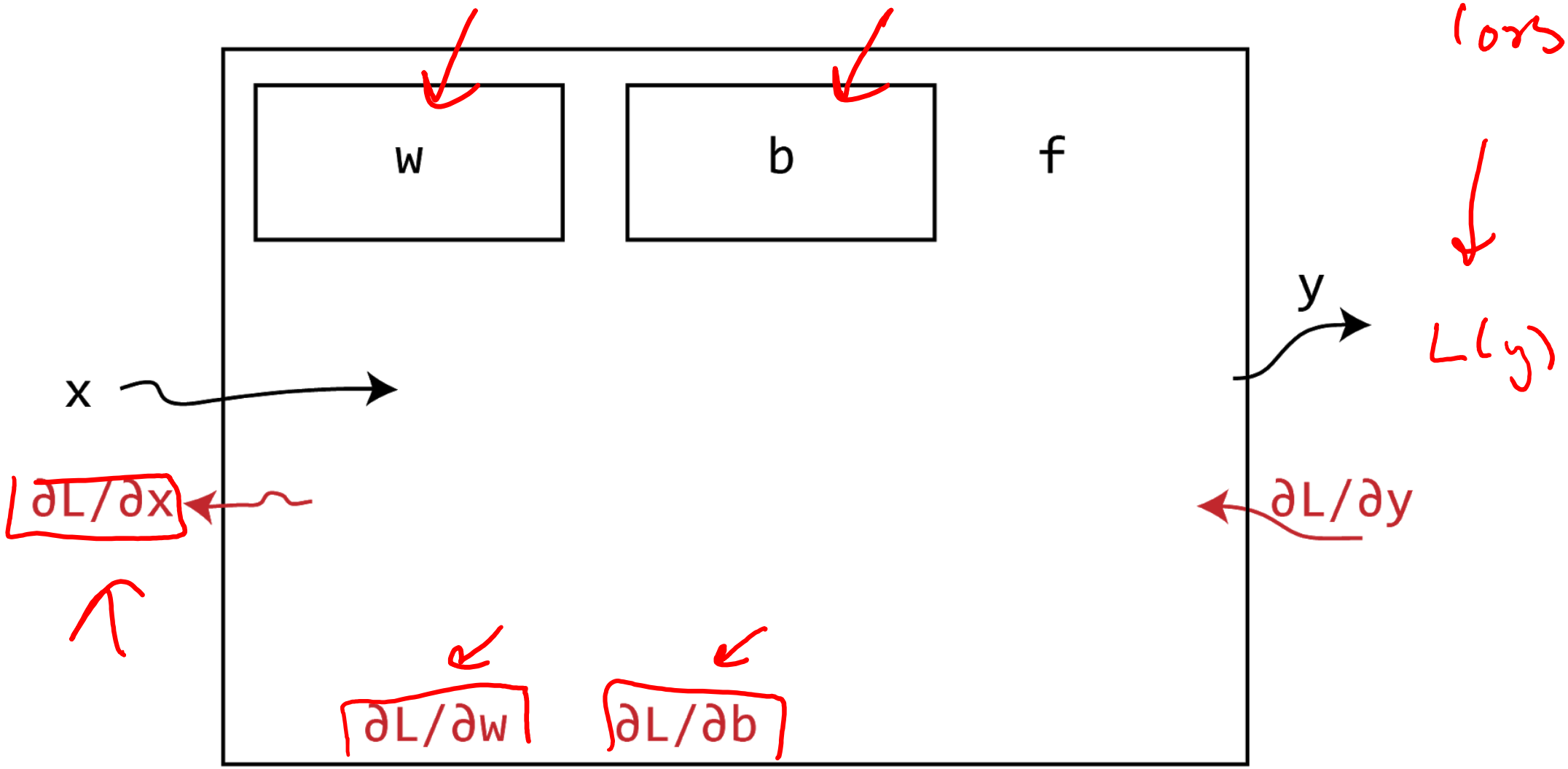


Backpropagation – Forward Prop.

Stealing from Joe Redmon's Slides (Graduate Student here!): 03 - Deep Learning: Neural Networks in Practice - Google Slides

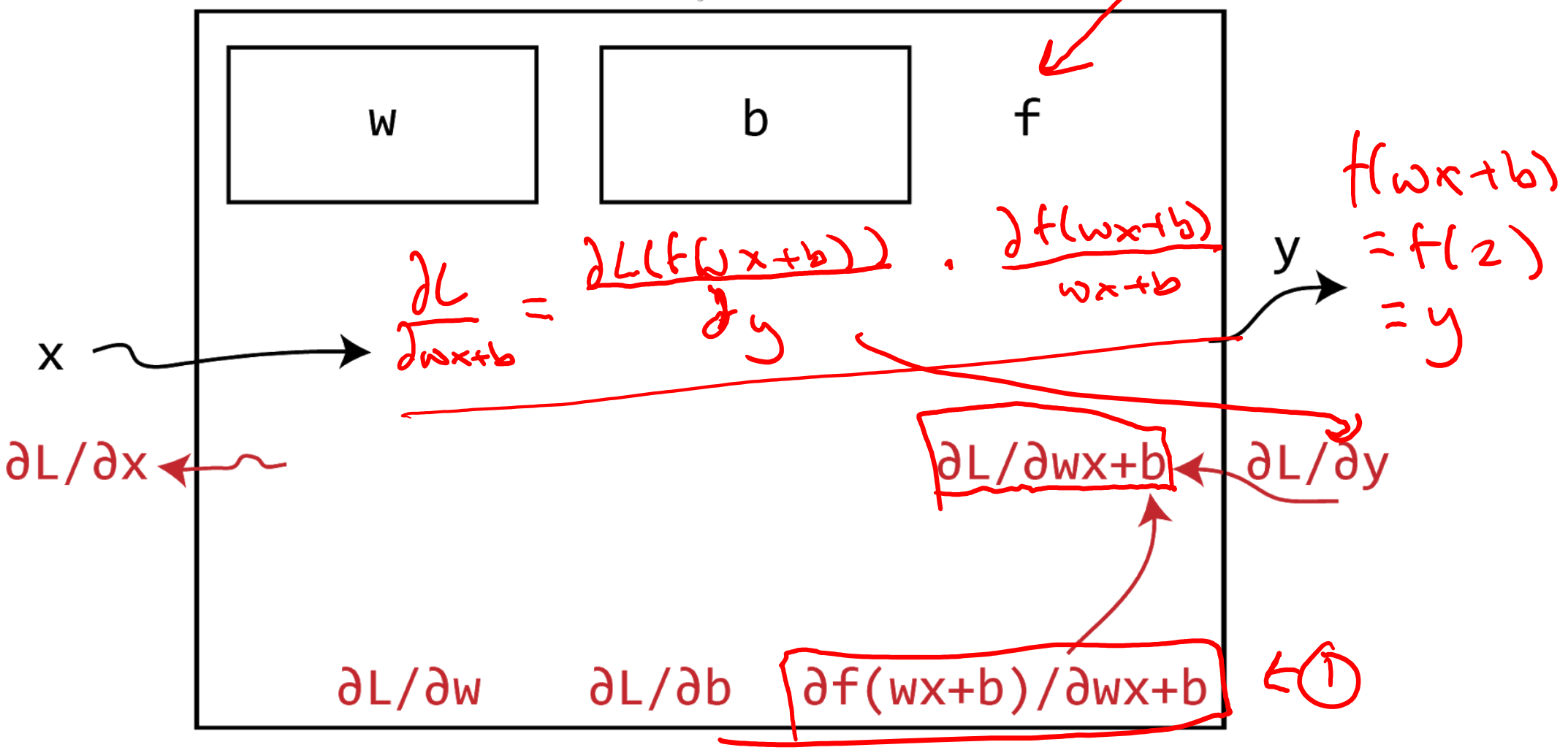


Backpropagation – Intuition

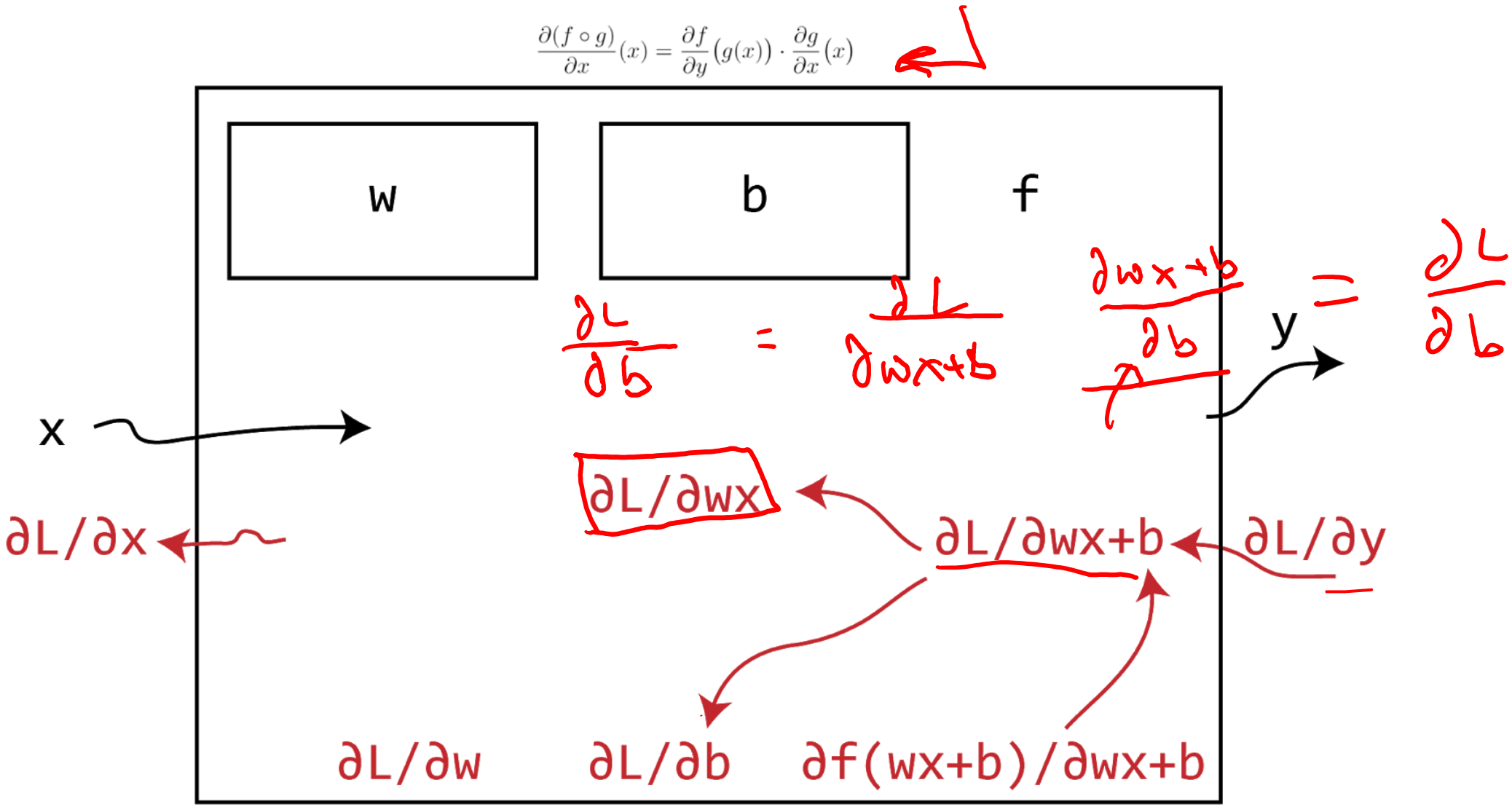


Backpropagation – Intuition

$$\frac{\partial(f \circ g)}{\partial x}(x) = \frac{\partial f}{\partial y}(g(x)) \cdot \frac{\partial g}{\partial x}(x)$$

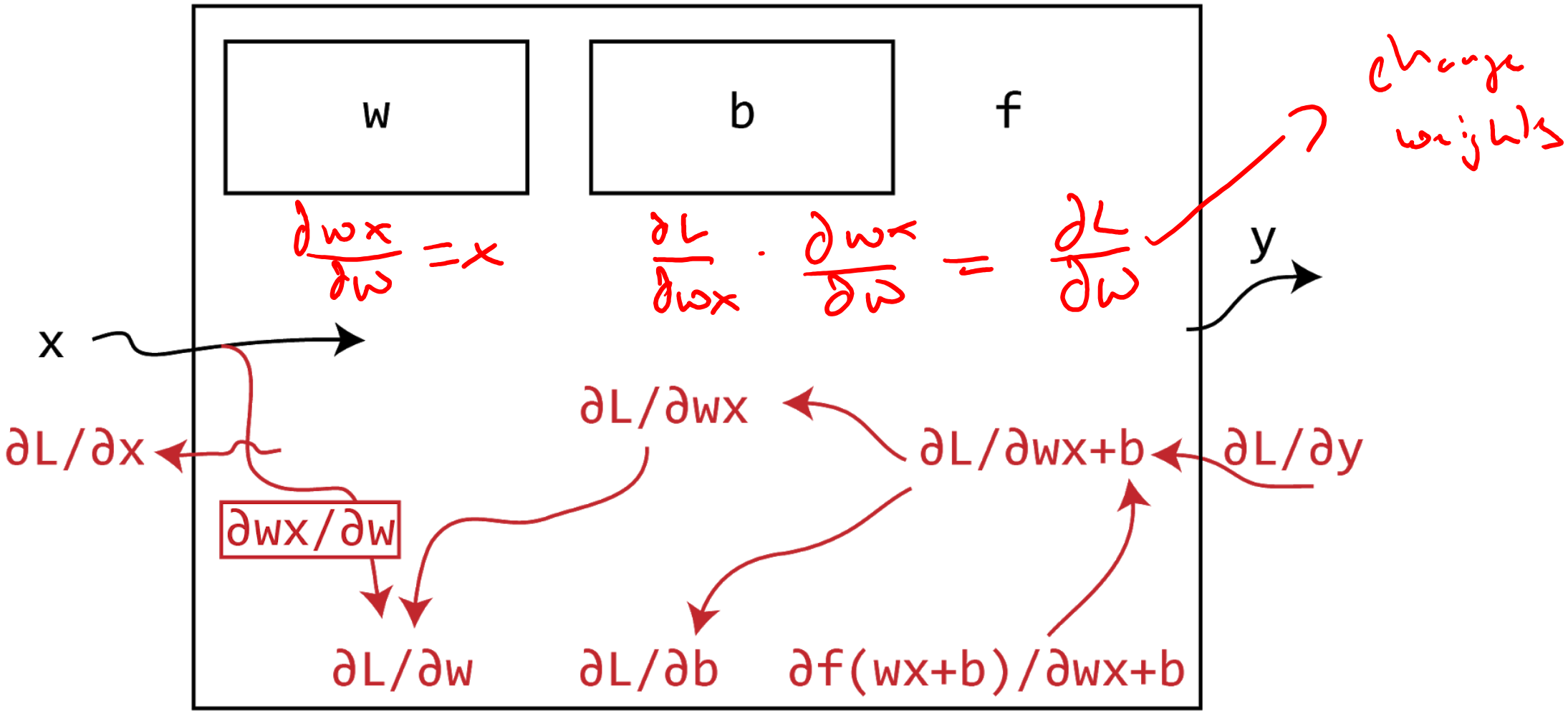


Backpropagation – Intuition



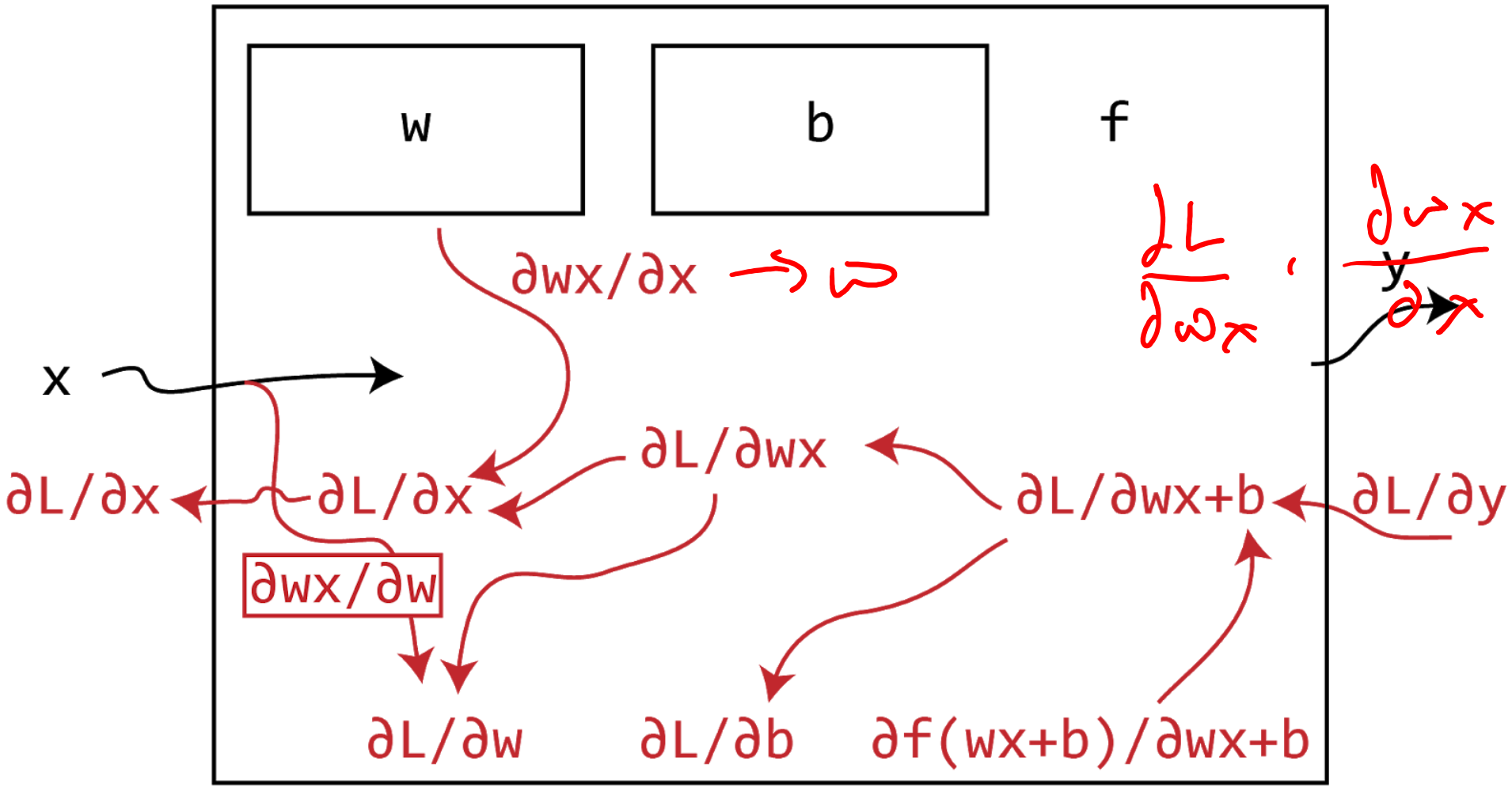
Backpropagation – Intuition

$$\frac{\partial(f \circ g)}{\partial x}(x) = \frac{\partial f}{\partial y}(g(x)) \cdot \frac{\partial g}{\partial x}(x)$$



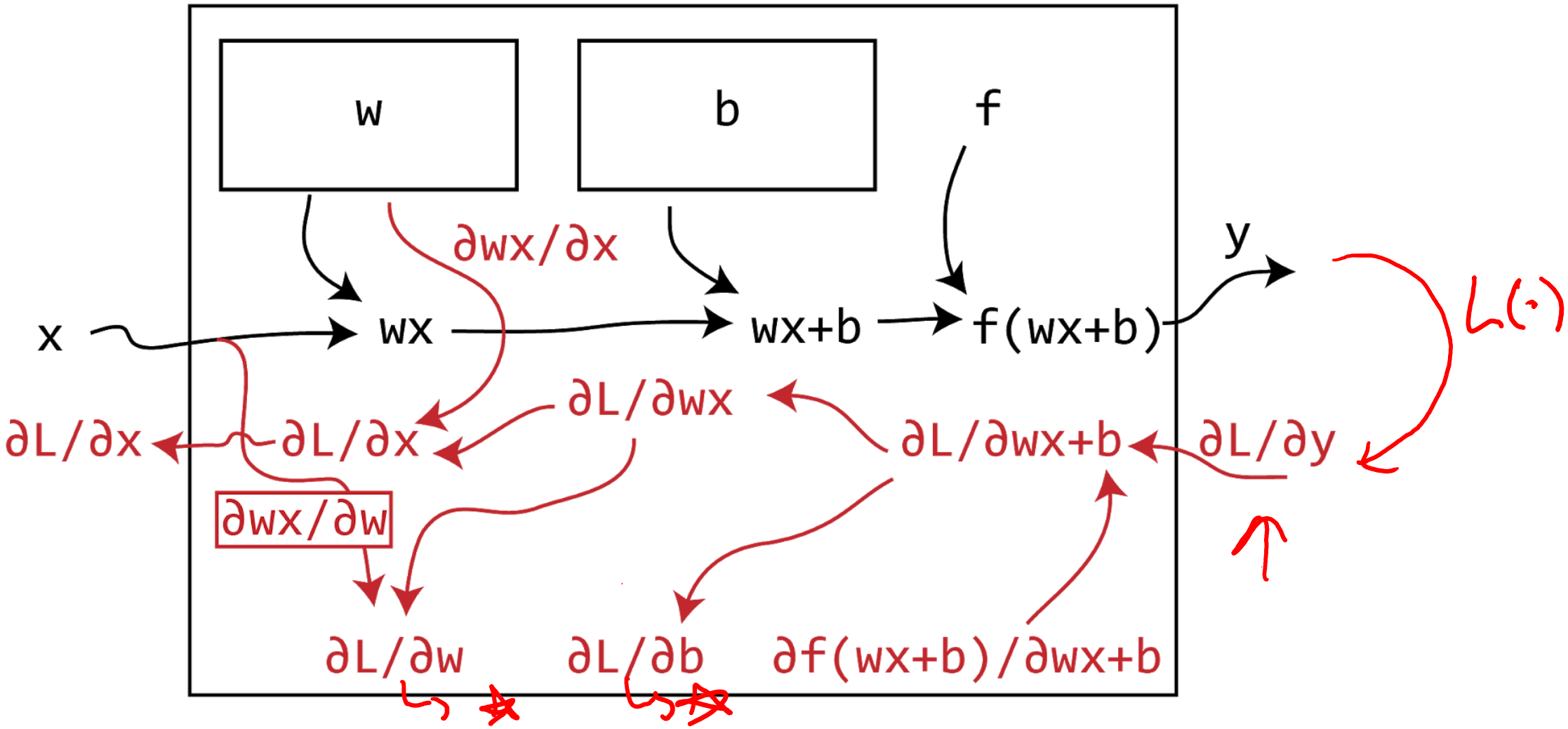
Backpropagation – Intuition

$$\frac{\partial(f \circ g)}{\partial x}(x) = \frac{\partial f}{\partial y}(g(x)) \cdot \frac{\partial g}{\partial x}(x)$$



Backpropagation – Intuition

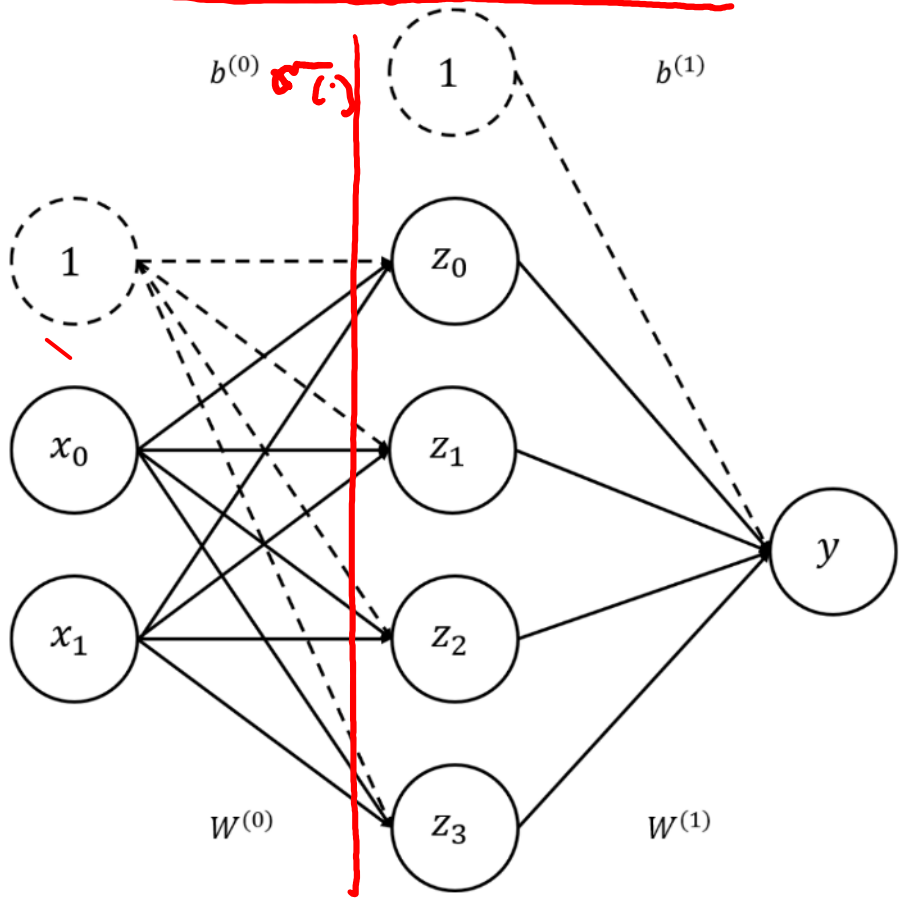
$$\frac{\partial(f \circ g)}{\partial x}(x) = \frac{\partial f}{\partial y}(g(x)) \cdot \frac{\partial g}{\partial x}(x)$$



gradient vanishing ⟶ Ω

Backpropagation – Neural Nets (3.1)

Consider a 1-hidden-layer neural network with a single output unit. Formally the network can be defined by the parameters $W^{(0)} \in \mathbb{R}^{h \times d}$, $b^{(0)} \in \mathbb{R}^h$, $W^{(1)} \in \mathbb{R}^{1 \times h}$ and $b^{(1)} \in \mathbb{R}$. The input is given by $x \in \mathbb{R}^d$. We will use sigmoid activation for the first hidden layer z and no activation for the output y. Below is a visualization of such a neural network with $d = 2$ and $h = 4$.

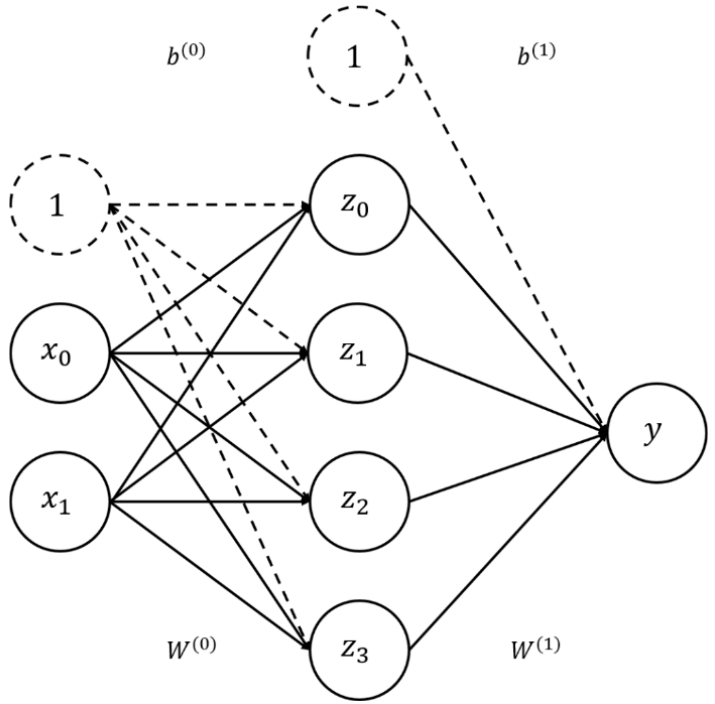


$z = \sigma(W^{(0)} x + b^{(0)})$

Backpropagation – Neural Nets (3.1a)

Write out the forward pass for the network using $x, W^{(0)}, b^{(0)}, z, W^{(1)}, b^{(1)}, \sigma$ and y.

Hint: Write $z = \ldots$ and $y = \ldots$

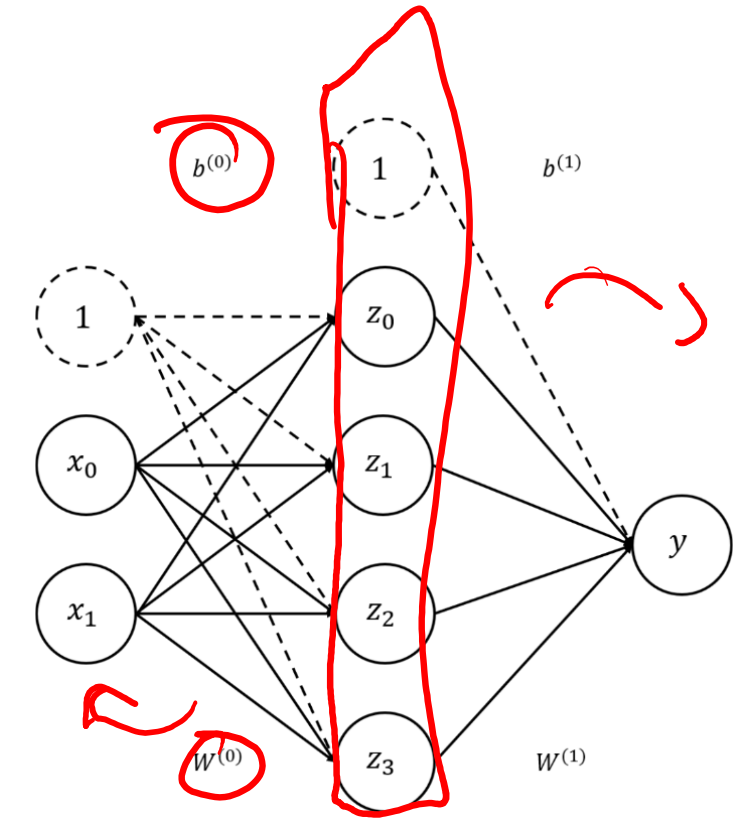


Backpropagation – Neural Nets (3.1a)

Write out the forward pass for the network using $x, W^{(0)}, b^{(0)}, z, W^{(1)}, b^{(1)}, \sigma$ and y.

Hint: Write $z = \ldots$ and $y = \ldots$

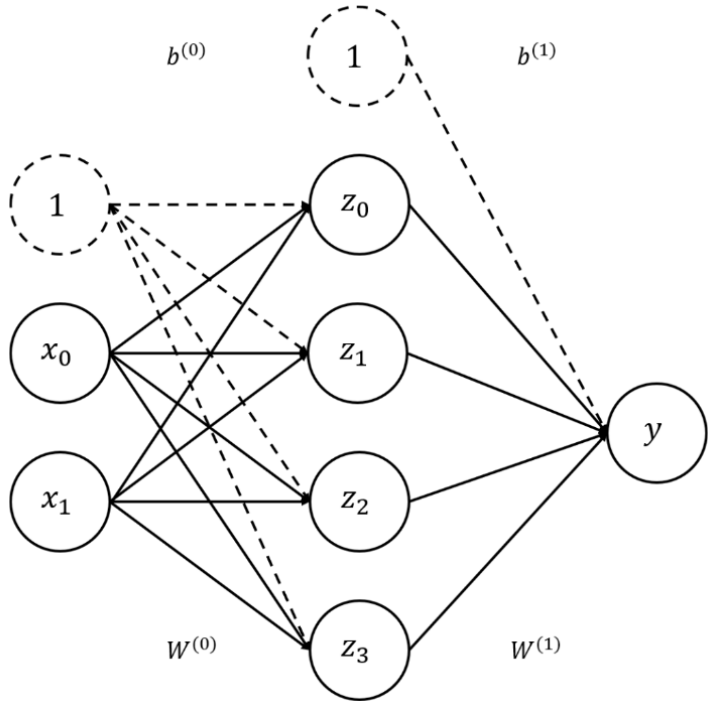
$$z = \sigma(W^{(0)}x + b^{(0)})$$
$$y = W^{(1)}z + b^{(1)}$$



$z = w^0 x + b^0$

Backpropagation – Neural Nets (3.1b)

Find the partial derivatives of the output with respect $W^{(1)}$ and $b^{(1)}$, namely $\frac{\partial y}{\partial W^{(1)}}$ and $\frac{\partial y}{\partial b^{(1)}}$.



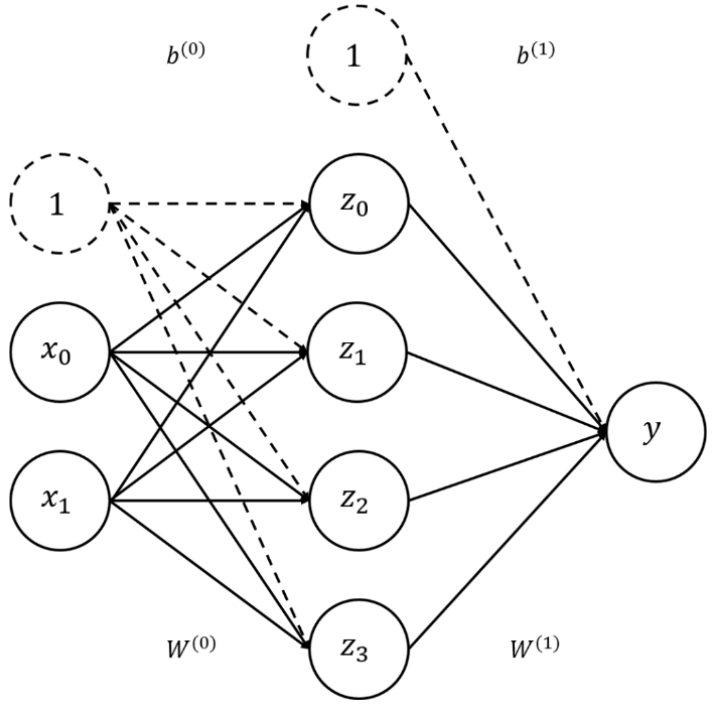
Backpropagation – Neural Nets (3.1b)

Find the partial derivatives of the output with respect $W^{(1)}$ and $b^{(1)}$, namely $\frac{\partial y}{\partial W^{(1)}}$ and $\frac{\partial y}{\partial b^{(1)}}$.

$y = W^{(1)} z + b^{(1)}$

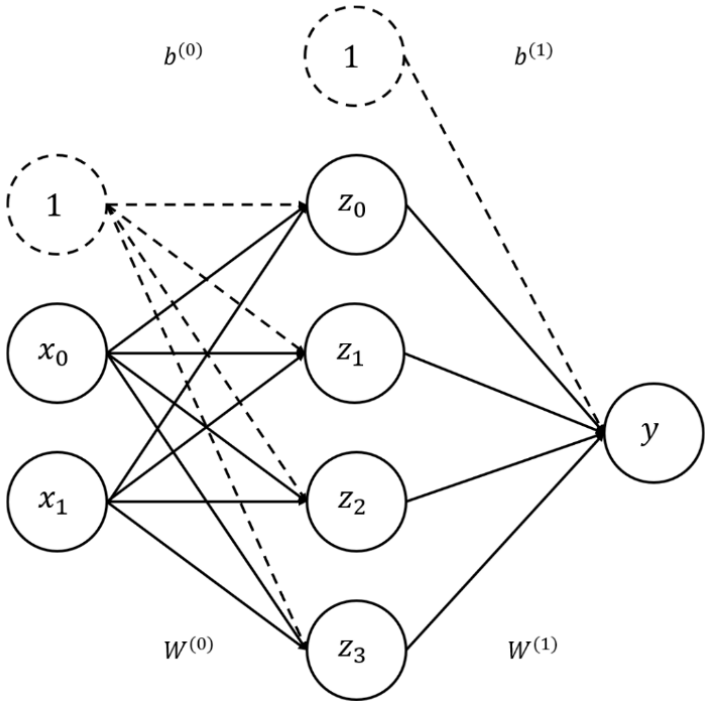
$$\frac{\partial y}{\partial W^{(1)}} = z$$

$$\frac{\partial y}{\partial b^{(1)}} = 1$$



Backpropagation – Neural Nets (3.1c)

Now find the partial derivative of the output with respect to the output of the hidden layer z, that is $\frac{\partial y}{\partial z}$

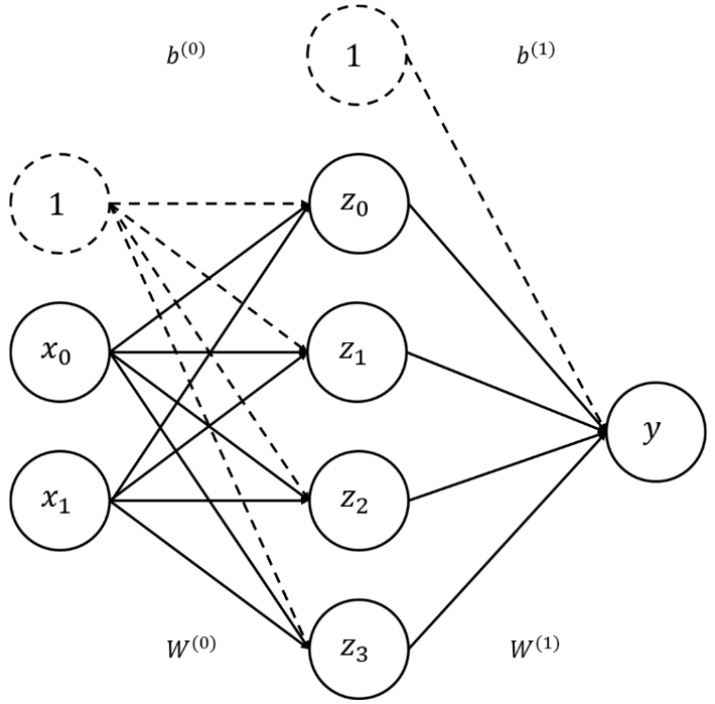


Backpropagation – Neural Nets (3.1c)

Now find the partial derivative of the output with respect to the output of the hidden layer z, that is $\frac{\partial y}{\partial z}$

$$y = W^{(1)} z + b^{(1)}$$

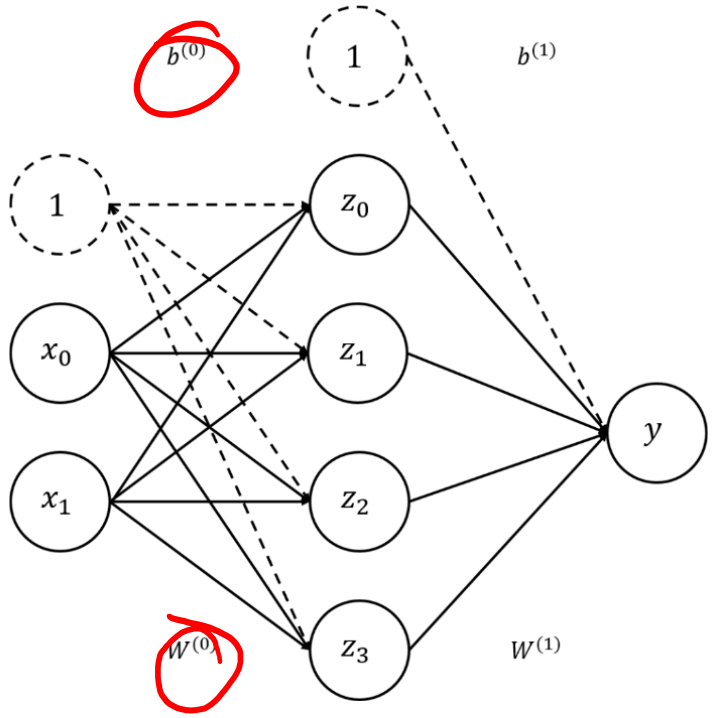
$$\frac{\partial y}{\partial z} = W^{(1)}$$



Backpropagation – Neural Nets (3.1d)

Finally find the partial derivatives of the output with respect to $W^{(0)}$ and $b^{(0)}$, that is $\frac{\partial y}{\partial W^{(0)}}$ and $\frac{\partial y}{\partial b^{(0)}}$.

Hint: First find $\frac{\partial z_i}{\partial W_i^{(0)}}$ and $\frac{\partial z_i}{\partial b_i^{(0)}}$. Then note that $\frac{\partial y}{\partial W_i^{(0)}} = \frac{\partial y}{\partial z_i}\frac{\partial z_i}{\partial W_i^{(0)}}$ and $\frac{\partial y}{\partial b_i^{(0)}} = \frac{\partial y}{\partial z_i}\frac{\partial z_i}{\partial b_i^{(0)}}$



$$= \frac{\partial(w_1^{(0)}x + b)}{\partial(w_1^{(0)}x)}$$

$$= \sigma'(w_1^{(0)}x + b)^2$$

$$= (w_1^{(0)}x + b)(1 - w_1^{(0)}x + b)$$

$$= z(1 - z)$$

Backpropagation – Neural Nets (3.1d)

Finally find the partial derivatives of the output with respect to $W^{(0)}$ and $b^{(0)}$, that is $\frac{\partial y}{\partial W^{(0)}}$ and $\frac{\partial y}{\partial b^{(0)}}$.

Hint: First find $\frac{\partial z_i}{\partial W_i^{(0)}}$ and $\frac{\partial z_i}{\partial b_i^{(0)}}$. Then note that $\frac{\partial y}{\partial W_i^{(0)}} = \frac{\partial y}{\partial z_i}\frac{\partial z_i}{\partial W_i^{(0)}}$ and $\frac{\partial y}{\partial b_i^{(0)}} = \frac{\partial y}{\partial z_i}\frac{\partial z_i}{\partial b_i^{(0)}}$

$$\frac{\partial z_i}{\partial W_i^{(0)}} = z_i(1-z_i)x^T$$

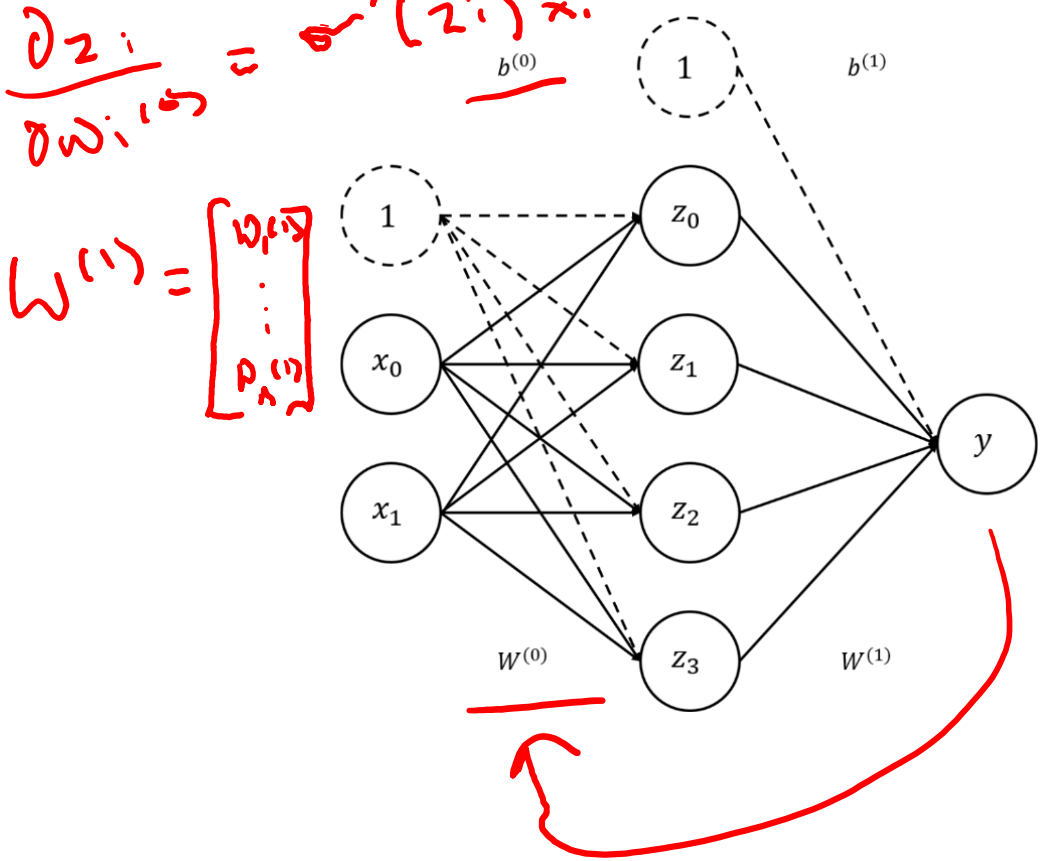
$$\frac{\partial y}{\partial W_i^{(0)}} = \frac{\partial y}{\partial z_i}\frac{\partial z_i}{\partial W_i^{(0)}} = W_i^{(1)} \cdot z_i(1-z_i)x^T$$

$$\frac{\partial y}{\partial W^{(0)}} = \left[W^{(1)} \circ z \circ (1-z)\right] x^T$$

$$\frac{\partial z_i}{\partial b_i^{(0)}} = z_i(1-z_i)$$

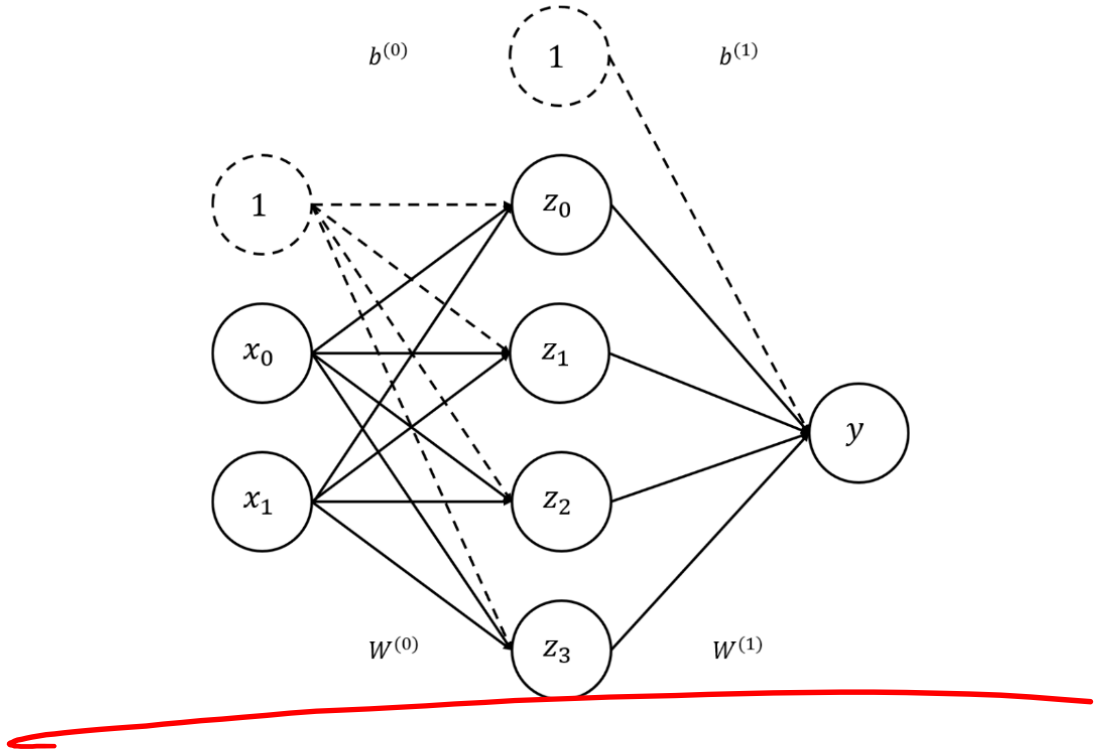
$$\frac{\partial y}{\partial b_i^{(0)}} = \frac{\partial y}{\partial z_i}\frac{\partial z_i}{\partial b_i^{(0)}} = W_i^{(1)} \cdot z_i(1-z_i)$$

$$\frac{\partial y}{\partial b^{(0)}} = W^{(1)} \circ z \circ (1-z)$$



Backpropagation – Weights (3.2)

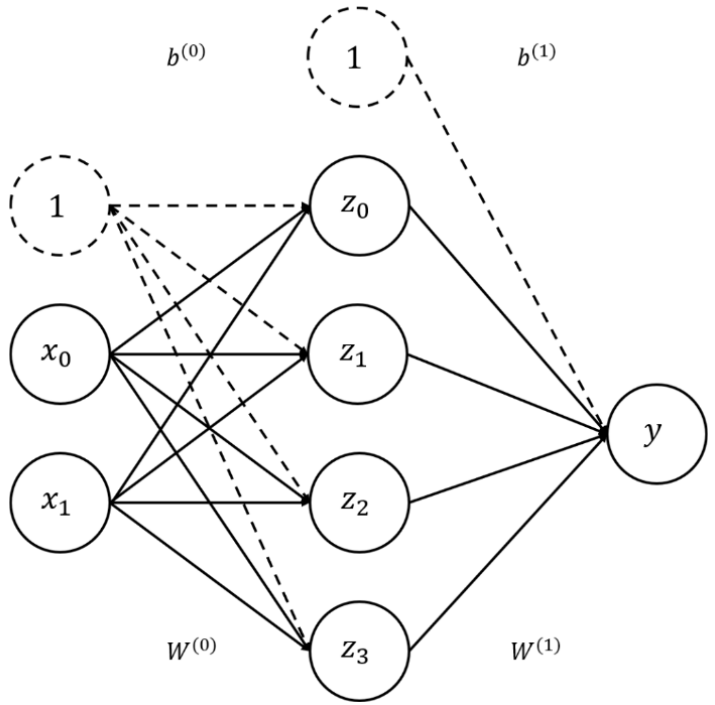
Suppose we initialize all weights and biases in the network to 0 before performing gradient descent.



Backpropagation – Weights (3.2a)

Suppose we initialize all weights and biases in the network to 0 before performing gradient descent.

For all $x \in \mathbb{R}^d$, find z and y after the forward pass.



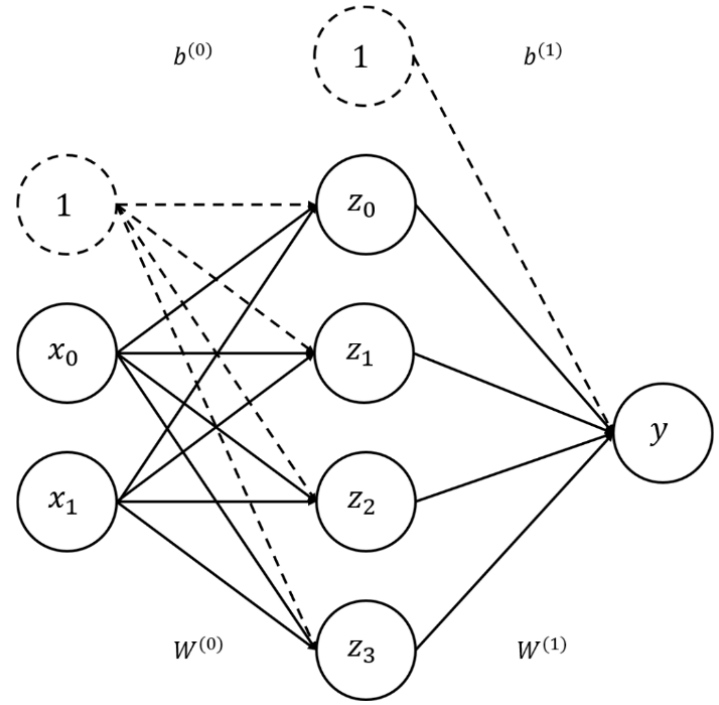
Backpropagation – Weights (3.2a)

Suppose we initialize all weights and biases in the network to 0 before performing gradient descent.

For all $x \in \mathbb{R}^d$, find z and y after the forward pass.

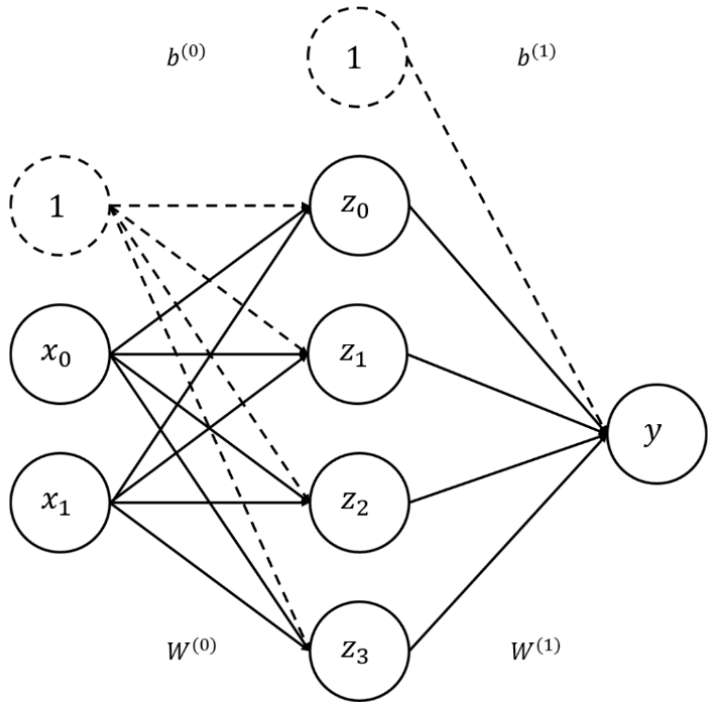
$$z_i = \sigma(W_i^{(0)} x + b^{(0)_i}) = \sigma(\mathbf{0}x + 0) = \sigma(0) = \frac{1}{2}$$

$$y = W^{(1)} z + b^{(1)} = \mathbf{0} \cdot \frac{\mathbf{1}}{\mathbf{2}} + 0 = 0$$



Backpropagation – Weights (3.2b)

Suppose we initialize all weights and biases in the network to 0 before performing gradient descent. Now find the values of the gradients $\frac{\partial y}{\partial W^{(1)}}$, $\frac{\partial y}{\partial b^{(1)}}$, $\frac{\partial y}{\partial W^{(0)}}$ and $\frac{\partial y}{\partial b^{(0)}}$. Note that some of the gradients will be in terms of x.



Backpropagation – Weights (3.2b)

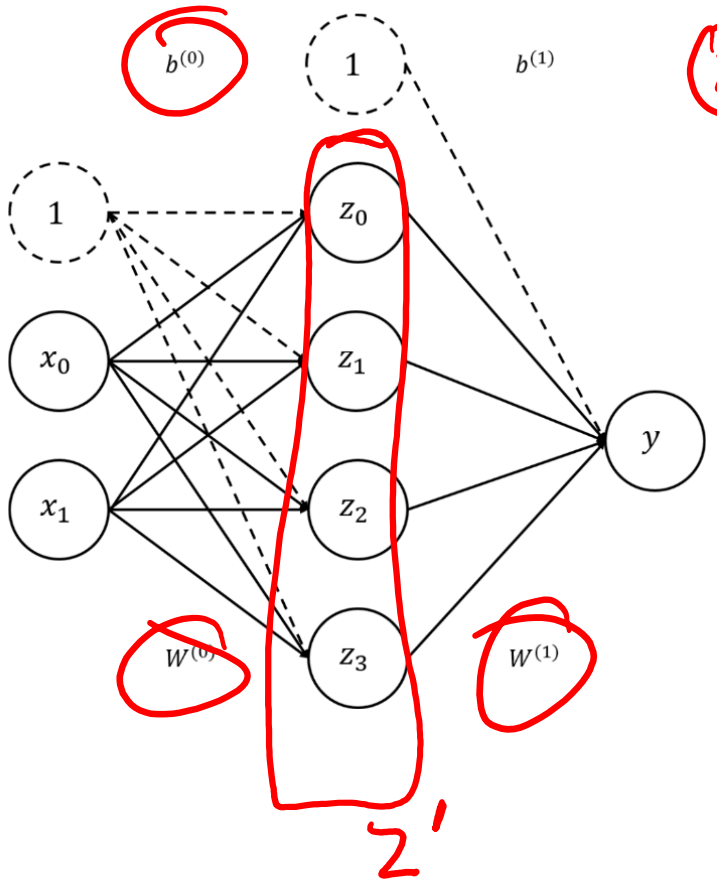
Suppose we initialize all weights and biases in the network to 0 before performing gradient descent. Now find the values of the gradients $\frac{\partial y}{\partial W^{(1)}}$, $\frac{\partial y}{\partial b^{(1)}}$, $\frac{\partial y}{\partial W^{(0)}}$ and $\frac{\partial y}{\partial b^{(0)}}$. Note that some of the gradients will be in terms of x.

$$\frac{\partial y}{\partial W^{(1)}} = z = \mathbf{1}$$

$$\frac{\partial y}{\partial b^{(1)}} = 1$$

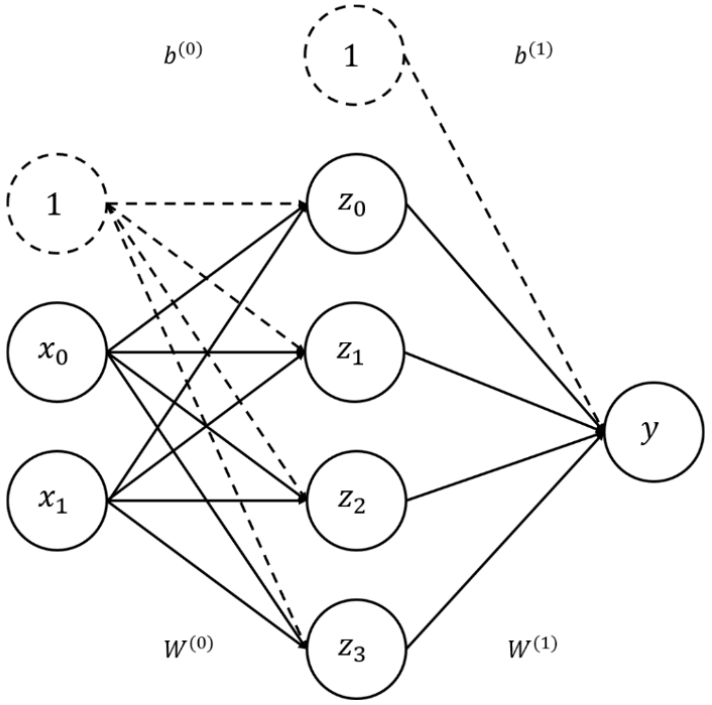
$$\frac{\partial y}{\partial W^{(0)}} = \left[W^{(1)} \circ z \circ (1-z)\right] x^T = (\mathbf{0} \circ \frac{\mathbf{1}}{\mathbf{2}} \circ \frac{\mathbf{1}}{\mathbf{2}}) x^T = \mathbf{0}$$

$$\frac{\partial y}{\partial b^{(0)}} = W^{(1)} \circ z \circ (1-z) = \mathbf{0} \circ \frac{\mathbf{1}}{\mathbf{2}} \circ \frac{\mathbf{1}}{\mathbf{2}} = \mathbf{0}$$



Backpropagation – Weights (3.2c)

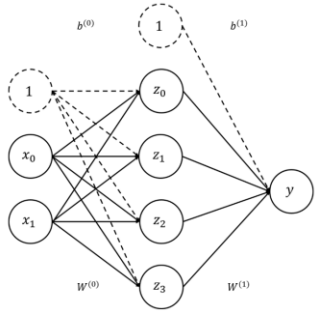
Suppose we initialize all weights and biases in the network to 0 before performing gradient descent. Observe the values of each z_i and observe each $\frac{\partial y}{\partial W_i^{(l)}}$ and $\frac{\partial y}{\partial b_i^{(l)}}$. What do you notice? And what does this imply for the expressiveness of the network? (Note that there is nothing special about the value 0 here, it just simplifies the calculations. The same can be shown for initialization with any constant c)



Backpropagation – Weights (3.2c)

Suppose we initialize all weights and biases in the network to 0 before performing gradient descent. Observe the values of each z_i and observe each $\frac{\partial y}{\partial W_i^{(l)}}$ and $\frac{\partial y}{\partial b_i^{(l)}}$. What do you notice? And what does this imply for the expressiveness of the network? (Note that there is nothing special about the value 0 here, it just simplifies the calculations. The same can be shown for initialization with any constant c)

The key insight is that if we initialize the weights to all have the same value, all z_i are the same. Similarly all $W_i^{(l)}$ and $b_i^{(l)}$ are the same too and so the output y could be expressed with just a single z_i instead of h. Thus the neural network boils down to just having a single hidden unit. The same holds for the gradients, so during a step of gradient descent, $W_i^{(l)}$ and $b_i^{(l)}$ are updated in the same way. Thus after a step of gradient descent, all $W_i^{(l)}$ and $b_i^{(l)}$ are still the same. By induction, the same holds after an arbitrary number of steps of gradient descent.



Backpropagation – Other Ways (3.3)

There are at least three other ways of computing the gradient of a complicated function $f(x) : \mathbb{R}^n \to \mathbb{R}$. For each of these, think about one possible disadvantage that might explain why people tend to prefer reverse-mode automatic differentiation (i.e. backprop) in practice.

Backpropagation – Other Ways (3.3a)

Finite differencing. We can approximate the gradient as $\frac{\partial f}{\partial x_i} \approx \frac{1}{\varepsilon}\big(f(x + \varepsilon e_i) - f(x)\big)$, where ε is some small positive number and e_i is the ith standard basis vector.

Backpropagation – Other Ways (3.3a)

Finite differencing. We can approximate the gradient as $\frac{\partial f}{\partial x_i} \approx \frac{1}{\varepsilon}\big(f(x + \varepsilon e_i) - f(x)\big)$, where ε is some small positive number and e_i is the ith standard basis vector.

There are at least two possible disadvantages of this approach. First, it is an approximation: we must be careful how we choose ε. Second, this requires forward-propagating through f at least n times (once for each standard basis vector), which is computationally expensive. That said, this method is very simple to implement, and it is often used to check that other methods of differentiation have been implemented correctly.

Backpropagation – Other Ways (3.3b)

Analytic differentiation. If we can write down a mathematical formula for f, we could derive a formula for ∇f using pencil and paper, then write code to compute it directly.

Backpropagation – Other Ways (3.3b)

Analytic differentiation. If we can write down a mathematical formula for f, we could derive a formula for ∇f using pencil and paper, then write code to compute it directly.

One disadvantage of this approach is that it's error-prone: humans make mistakes. It might also be computationally inefficient. For example, if the gradients of f with respect to variables x and y share a term, we would want to compute that shared value only once and re-use it as necessary. In fact, if we implement this properly, we've essentially re-invented and manually implemented automatic differentiation over a computational graph.

$\nabla L(w) =$

Backpropagation – Other Ways (3.3c)

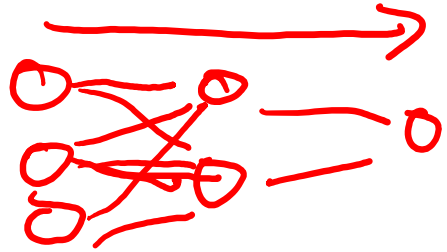
Forward-mode automatic differentiation. You may be surprised to learn that there is there is an automatic differentiation technique that proceeds *forward* through the graph and does not require saving any intermediate results. In the example above, let $v = W^{(0)}x + b^{(0)}$ be the pre-activation output of the hidden layer. Above, we computed

$$\frac{\partial y}{\partial W^{(0)}}(x, W^{(0)}, b^{(0)}, W^{(1)}, b^{(1)}) = \left(\frac{\partial y}{\partial z}(z, W^{(1)}, b^{(1)}) \cdot \frac{\partial z}{\partial v}(v)\right) \cdot \frac{\partial v}{\partial W^{(0)}}(W^{(0)}, x, b^{(0)})$$

(Note that $\frac{\partial v}{\partial W^{(0)}}$ is one of those cases where we need to flatten the matrix $W^{(0)}$ in order for the notation to make sense.) We could instead have computed this gradient by multiplying in the other direction:

$$\frac{\partial y}{\partial W^{(0)}}(x, W^{(0)}, b^{(0)}, W^{(1)}, b^{(1)}) = \frac{\partial y}{\partial z}(z, W^{(1)}, b^{(1)}) \cdot \left(\frac{\partial z}{\partial v}(v) \cdot \frac{\partial v}{\partial W^{(0)}}(W^{(0)}, x, b^{(0)})\right)$$

This second method is called forward-mode automatic differentiation. Why do you think people usually use reverse-mode autodiff instead?



Backpropagation – Other Ways (3.3c)

This second method is called forward-mode automatic differentiation. Why do you think people usually use reverse-mode autodiff instead?

Note that $\frac{\partial y}{\partial z}$ is a vector, while $\frac{\partial z}{\partial v}$ and $\frac{\partial v}{\partial W^{(0)}}$ are both matrices. It is usually the case that forward-mode autodiff requires a matrix-matrix multiplication at each step, which takes more time than the vector-matrix multiplication used in reverse-mode autodiff. Forward-mode autodiff does usually require less memory than reverse-mode (since we don't have to save the results of the forward pass), but for deep learning workflows, memory is usually less important than computation time.