Machine Learning

CMPT 726

Mo Chen
SFU School of Computing Science
2025-10-17

Neural Networks

Features / Basis Functions

Recall: By replacing raw data with features, we can make the prediction depend *non-linearly* on the raw data.

$\hat{y} = \vec{w}^{\top} \phi(\vec{x})$

The feature map ϕ is fixed. Can we *learn* the features?

We can replace the feature map with another model and find the optimal parameters of that model (often referred to as “learning” the parameters of the model).

This is the idea behind neural networks.

Neural Networks

We start with the multiple output linear regression model on features:

$\hat{\vec{y}} = W\phi(\vec{x})$

Let's rename W as W_L and use $\vec{h}_L$ to denote $\phi(\vec{x})$. So, $\hat{\vec{y}} = W_L\vec{h}_L$.

Let's replace the feature map $\vec{x} \mapsto \phi(\vec{x})$ with $\vec{x} \mapsto \psi(W_{L-1}\vec{x})$, where $\psi(\cdot)$ is some known function. We can then learn both the features $\vec{h}_L$ (by learning W_{L-1}) and the parameters W_L .

So, now the model becomes

$\hat{\vec{y}} = W_L\vec{h}_L$, where $\vec{h}_L = \psi(W_{L-1}\vec{x})$.

The model parameters consist of both W_L and W_{L-1} .

Neural Networks

Let's replace the raw data with features again. The model becomes

$\hat{\vec{y}} = W_L \vec{h}_L$, where $\vec{h}_L = \psi(W_{L-1}\phi(\vec{x}))$.

We use $\vec{h}_{L-1}$ to denote $\phi(\vec{x})$.The model can be then written as

$\hat{\vec{y}} = W_L \vec{h}_L$, where $\vec{h}_L = \psi(W_{L-1}\vec{h}_{L-1})$.

Let's replace the feature map $\vec{x} \mapsto \phi(\vec{x})$ with $\vec{x} \mapsto \psi(W_{L-2}\vec{x})$, where $\psi(\cdot)$ is some known function. We can then learn both the features $\vec{h}_{L-1}$ (by learning W_{L-2}) and the parameters W_L and W_{L-1}.

So, now the model becomes:

$\hat{\vec{y}} = W_L \vec{h}_L$, where $\vec{h}_L = \psi(W_{L-1}\vec{h}_{L-1})$ and $\vec{h}_{L-1} = \psi(W_{L-2}\vec{x})$.

The model parameters consist of W_L ,W_{L-1} and W_{L-2}.

Neural Networks

Applying this idea recursively gives us the following model:

$\hat{\vec{y}} = W_L \vec{h}_L$,where $\vec{h}_L = \psi(W_{L-1}\vec{h}_{L-1})$ and $\vec{h}_{L-1} = \psi(W_{L-2}\vec{h}_{L-2})$, ..., and $\vec{h}_1 = \psi(W_0\vec{x})$.

The model parameters consist of W_L ,W_{L-1} ,…. ,and W_0 .

We can write this more compactly:

$$\hat{\vec{y}} = W_L\psi\left(W_{L-1}\psi\left(W_{L-2}\cdots\psi(W_0\vec{x})\right)\right)$$

This model is the simplest type of neural network, known as a **multi-layer perceptron** (MLP).

Neural networks are also known as **neural nets** or **artificial neural networks**.

Weights and Biases

Multi-layer perceptron model:

$\hat{\vec{y}} = W_L \vec{h}_L$,where $\vec{h}_L = \psi(W_{L-1}\vec{h}_{L-1})$ and $\vec{h}_{L-1} = \psi(W_{L-2}\vec{h}_{L-2})$,… ,and $\vec{h}_1 = \psi(W_0\vec{x})$.

Recall: In linear regression, $\hat{y} = \vec{w}^\top \vec{x}$, where $\vec{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_{n-1} \\ b \end{pmatrix}$ and $\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_{n-1} \\ 1 \end{pmatrix}$

Here, $\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_{n_0-1} \\ 1 \end{pmatrix}$ and $\vec{h}_l = \psi(\vec{z}_l) = \begin{pmatrix} g(z_1) \\ \vdots \\ g(z_{n_l-1}) \\ 1 \end{pmatrix}, W_l = \begin{pmatrix} w_{1,1} & \cdots & w_{1,n_l-1} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ w_{n_{l+1}-1,1} & \cdots & w_{n_{l+1}-1,n_l-1} & b_{n_{l+1}-1} \end{pmatrix}$

$w_{i,j}$ are known as **weights**, b_i are known as **biases**, and $g(\cdot)$ is known as an **activation function**.

Terminology

Multi-layer perceptron model:

$\hat{\vec{y}} = W_L \vec{h}_L$, where $\vec{h}_L = \psi(W_{L-1}\vec{h}_{L-1})$ and $\vec{h}_{L-1}$
$= \psi(W_{L-2}\vec{h}_{L-2})$,… ,and $\vec{h}_1 = \psi(W_0 \vec{x})$.

$\hat{\vec{y}}$: Output layer / last layer
$\vec{h}_l$: lth hidden layer / $(l+1)$th layer / features / (post-)activations
$\vec{x}$: Input layer / first layer
$\vec{z}_l := W_l \vec{h}_l$: Pre-activations

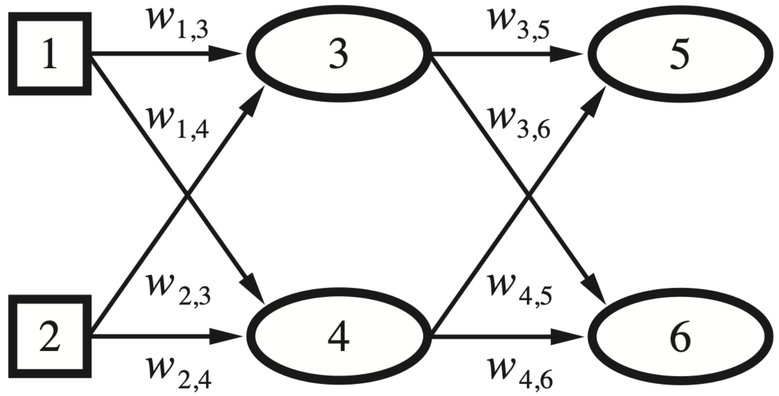
Terminology

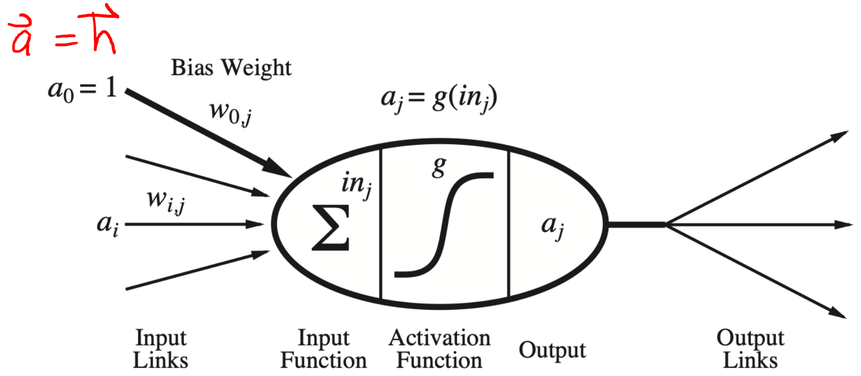
Multi-layer perceptron model:

$\hat{\vec{y}} = W_L \vec{h}_L$,where $\vec{h}_L = \psi(W_{L-1}\vec{h}_{L-1})$ and $\vec{h}_{L-1} = \psi(W_{L-2}\vec{h}_{L-2})$,… ,and $\vec{h}_1 = \psi(W_0\vec{x})$.

- An element of $\hat{\vec{y}}$, $\vec{h}_l$ and $\vec{x}$ is known as a neuron / a unit.
- Input neuron / input unit: an element of $\vec{x}$
- Hidden neuron / hidden unit: an element of $\vec{h}_l$
- Output neuron / output unit: an element of $\hat{\vec{y}}$

Model can be interpreted as a network





Credit: Russel & Norvig

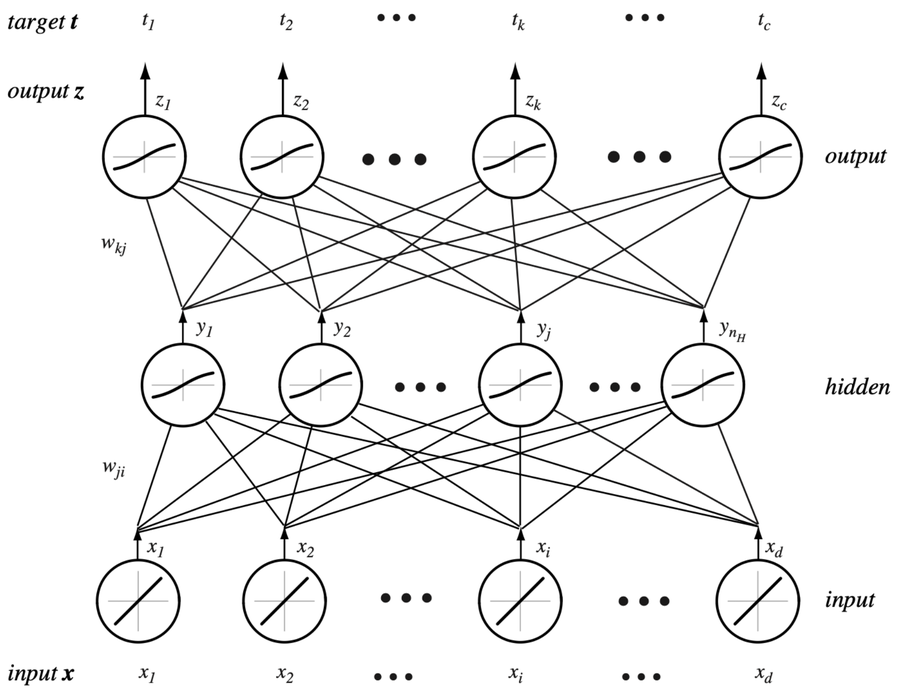
Terminology

Multi-layer perceptron model:

$\hat{\vec{y}} = W_L \vec{h}_L$,where $\vec{h}_L = \psi(W_{L-1}\vec{h}_{L-1})$ and $\vec{h}_{L-1} = \psi(W_{L-2}\vec{h}_{L-2})$,… ,and $\vec{h}_1 = \psi(W_0\vec{x})$.

- An element of $\hat{\vec{y}}$, $\vec{h}_l$ and $\vec{x}$ is known as a neuron / a unit.
- Input neuron / input unit: an element of $\vec{x}$
- Hidden neuron / hidden unit: an element of $\vec{h}_l$
- Output neuron / output unit: an element of $\hat{\vec{y}}$

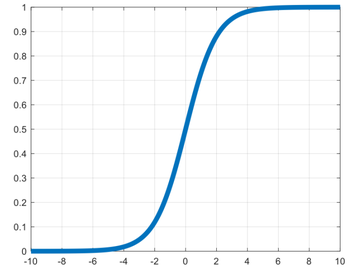
Represents a network of neurons



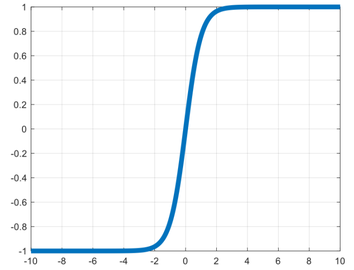
Credit: Russel & Norvig

Activation Functions

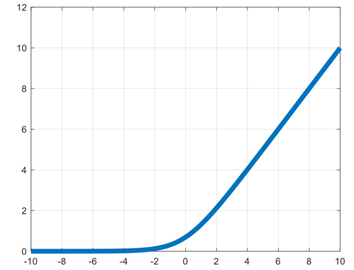
Sigmoid: $g(z) = \frac{1}{1+\exp(-z)} := \sigma(z)$



Hyperbolic tangent (Tanh): $g(z) = \tanh(z)$

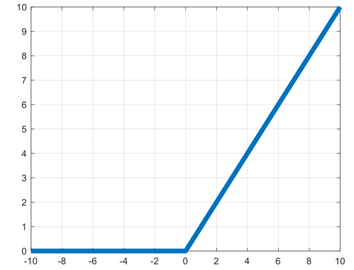


Softplus: $g(z) = \log(1 + \exp(z))$



Most common:

ReLU (stands for rectified linear unit): $g(z) = \max(0, z)$



Activation Functions

Linear: $g(z) = z$

But this is not a common choice of activation function. Why?

$\hat{\vec{y}} = W_L \vec{h}_L$, where $\vec{h}_L = \begin{pmatrix} W_{L-1}\vec{h}_{L-1} \\ 1 \end{pmatrix}$ and $\vec{h}_{L-1} = \begin{pmatrix} W_{L-2}\vec{h}_{L-2} \\ 1 \end{pmatrix}$,… ,and $\vec{h}_1 = \begin{pmatrix} W_0\vec{x} \\ 1 \end{pmatrix}$.

Let $\widetilde{W}_l$ denote the first n_l columns of W_l and $\vec{b}_l$ denote the last column of W_l:

$$\hat{\vec{y}} = \widetilde{W}_L \widetilde{W}_{L-1} \cdots \widetilde{W}_0 \vec{x} + \left(\vec{b}_L + \widetilde{W}_L \vec{b}_{L-1} + \widetilde{W}_L \cdots \widetilde{W}_1 \vec{b}_0\right)$$

Define $\widetilde{W} = \widetilde{W}_L \widetilde{W}_{L-1} \cdots \widetilde{W}_0$ and $\vec{b} = \vec{b}_L + \widetilde{W}_L \vec{b}_{L-1} + \widetilde{W}_L \cdots \widetilde{W}_1 \vec{b}_0$.

This becomes $\hat{\vec{y}} = \widetilde{W}\vec{x} + \vec{b}$, which is just a linear regression model.

As a result, the activation function is also commonly known as the non-linearity.

Model Summary

Model: $\hat{\vec{y}} := f\left(\vec{x}; \{W_l\}_{l=0}^{L}\right) = W_L \psi\left(W_{L-1}\psi\left(W_{L-2} \cdots \psi(W_0\vec{x})\right)\right)$

Parameters: $W_L, W_{L-1}, \ldots,$ and W_0

Loss function:

(For now let's consider the multiple output regression loss)

$$L\left(\{W_l\}_{l=0}^{L}\right) = \sum_{i=1}^{N} \left\|\vec{y}_i - \hat{\vec{y}}_i\right\|_2^2 = \sum_{i=1}^{N} \left\|\vec{y}_i - f\left(\vec{x}_i; \{W_l\}_{l=0}^{L}\right)\right\|_2^2$$

How can we find the optimal parameters?

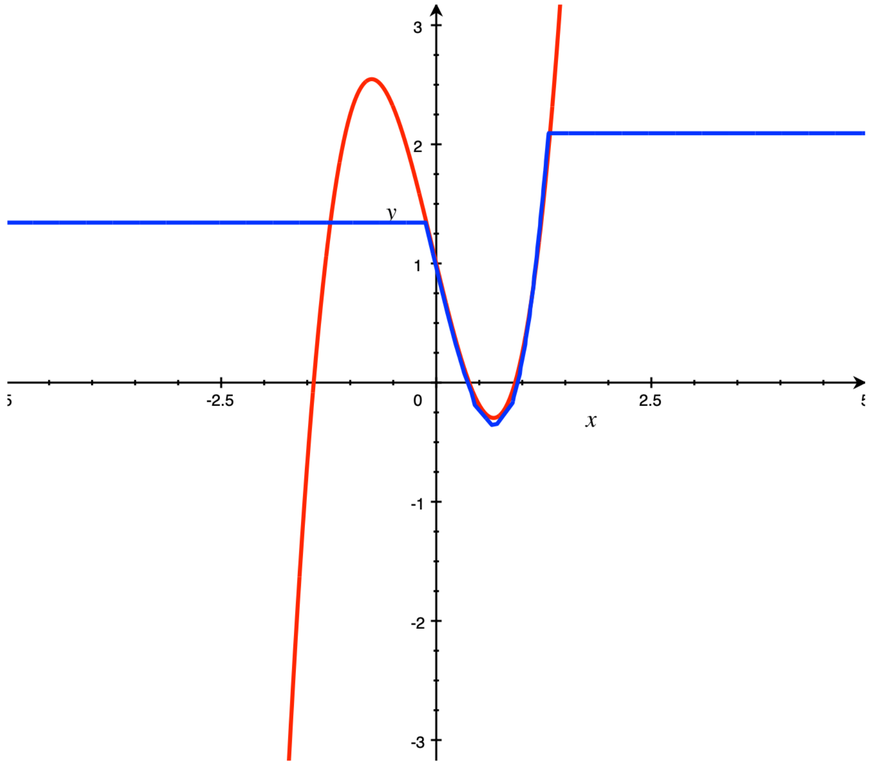
As we saw with non-linear least squares, because the model is non-linear in the parameters, we won't be able to solve for the optimal parameters analytically. Need to use iterative optimization algorithms.

Universal Function Approximation

For *any* continuous function on a closed and bounded set, a multi-layer perceptron with one hidden layer and sufficiently many hidden units on that layer can approximate it to any arbitrary precision, for most choices of activation functions*.

*must be nonlinear, bounded, nondecreasing, and continuous

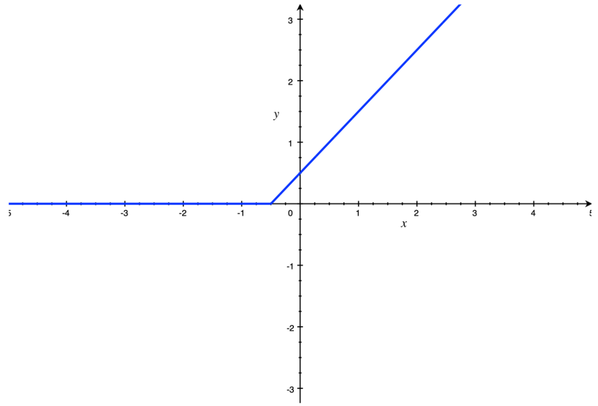
Because of this property, neural networks are **universal function approximators**.



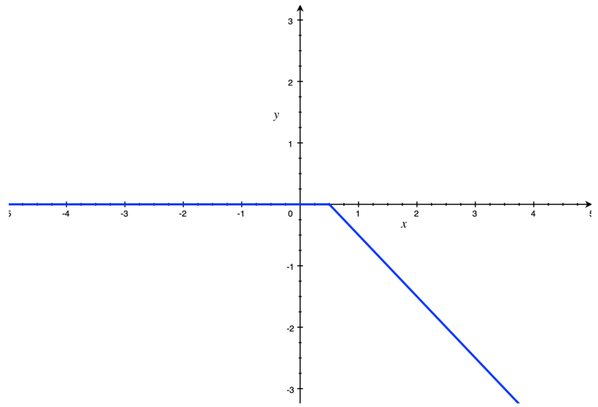
Universal Function Approximation

Example with ReLU activation function: $g(z) = \max(0, z)$

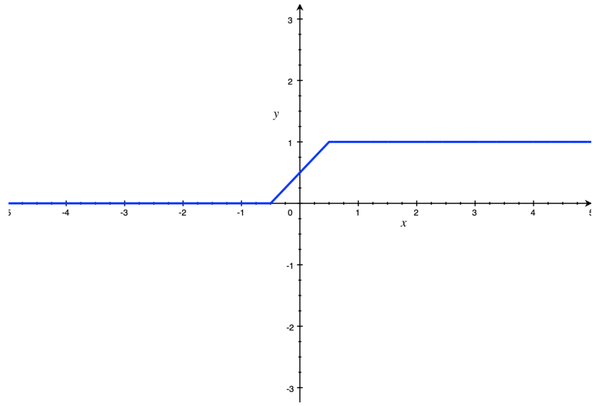
$g\left(z + \frac{1}{2}\right)$:



$-g\left(z - \frac{1}{2}\right)$:

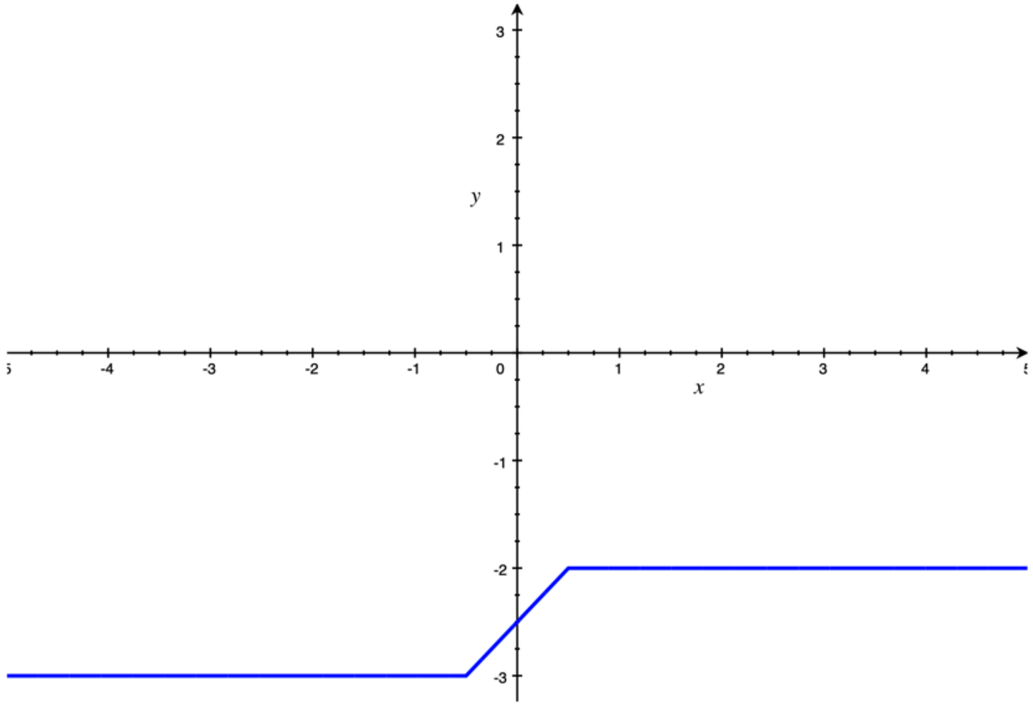


$g\left(z + \frac{1}{2}\right) - g\left(z - \frac{1}{2}\right)$:

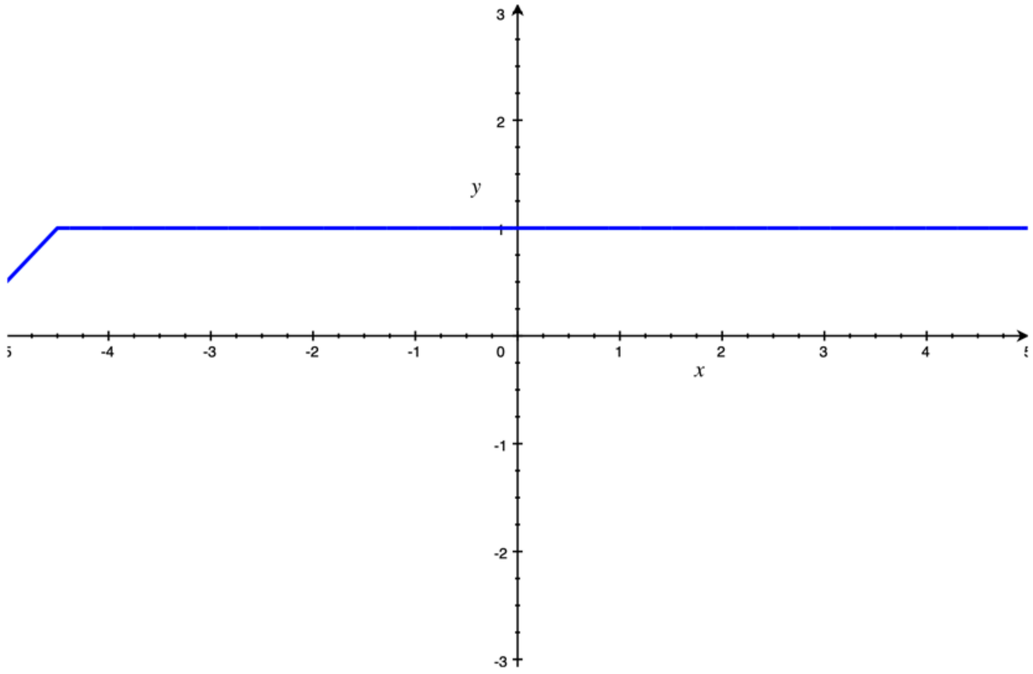


Universal Function Approximation

$g\left(z+\frac{1}{2}\right)-g\left(z-\frac{1}{2}\right)+a$:

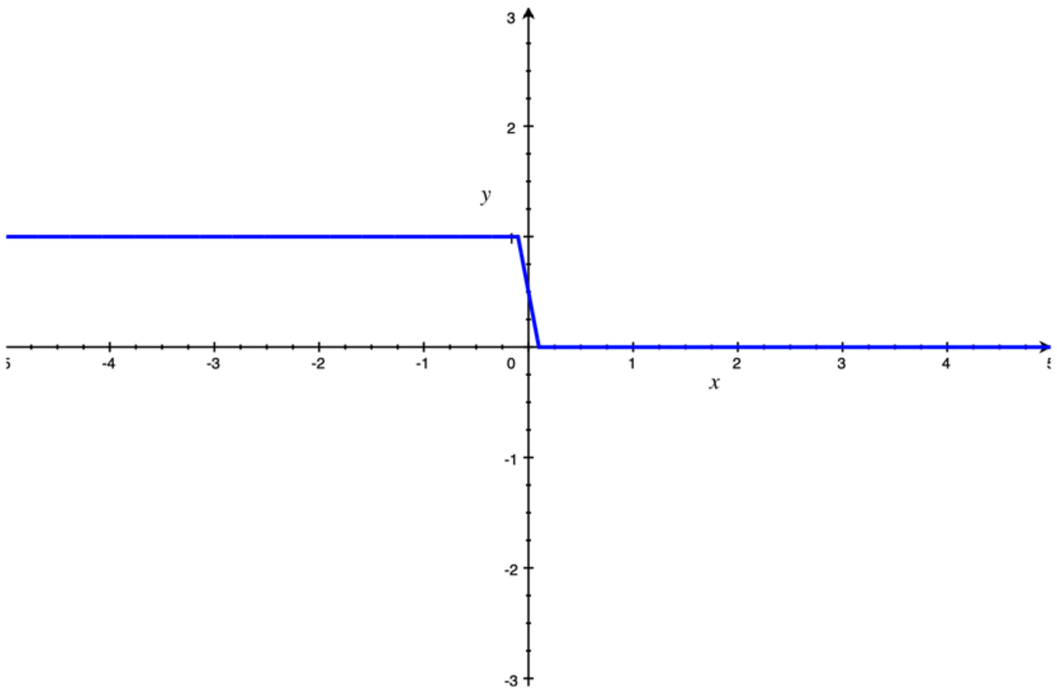


$g\left((z-b)+\frac{1}{2}\right)-g\left((z-b)-\frac{1}{2}\right)$:

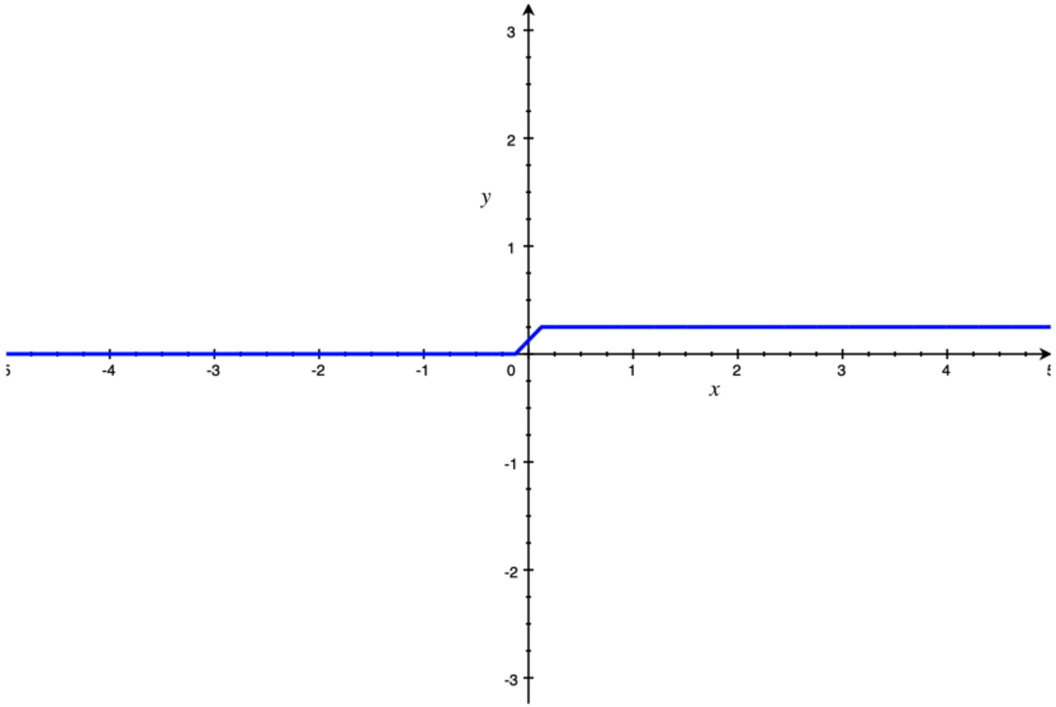


Universal Function Approximation

$g\left(cz+\frac{1}{2}\right)-g\left(cz-\frac{1}{2}\right)$:

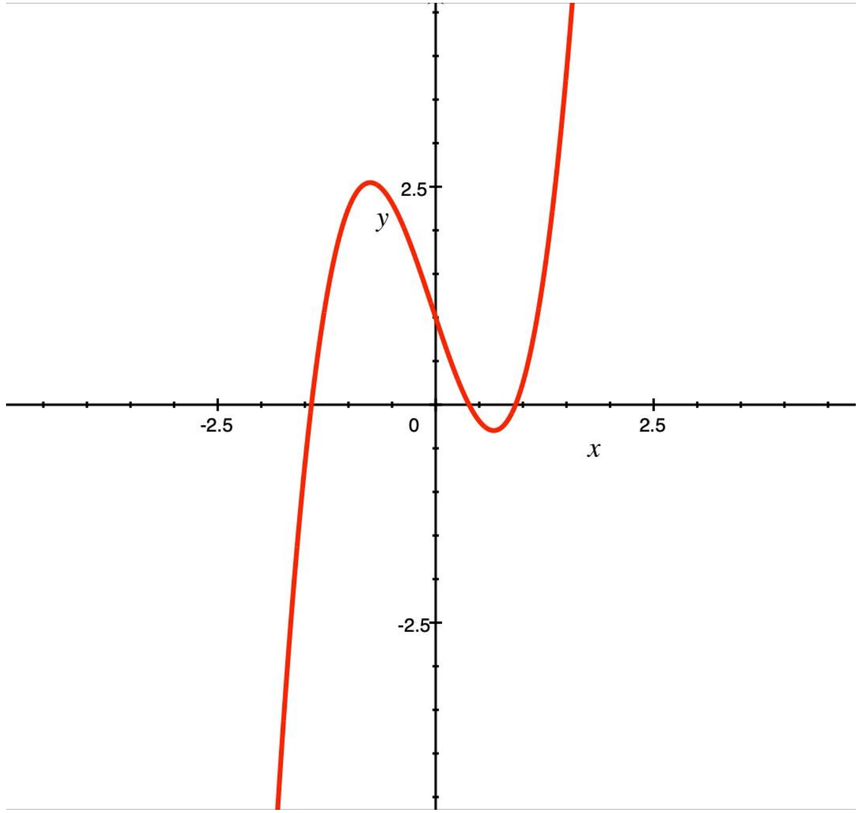


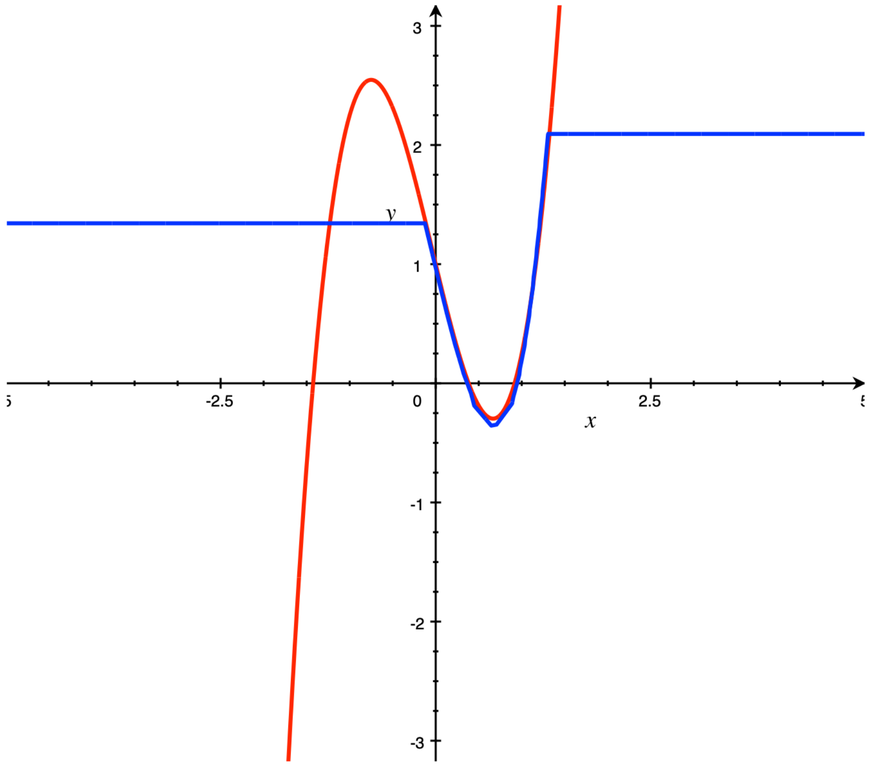
$g\left(z+\frac{1}{2d}\right)-g\left(z-\frac{1}{2d}\right)$:



Universal Function Approximation

$$g\left(c(z-b)+\frac{1}{2d}\right)-g\left(c(z-b)-\frac{1}{2d}\right)+a\text{:}\quad\left(\sum_i g\left(c_i(z-b_i)+\frac{1}{2d}\right)-g\left(c_i(z-b_i)-\frac{1}{2d}\right)+a_i\right)+\alpha\text{:}$$





Universal Function Approximation

$$f(z) = \left(\sum_i g\left(c_i(z - b_i) + \frac{1}{2d} \right) - g\left(c_i(z - b_i) - \frac{1}{2d} \right) + a_i \right) + \alpha$$

$$= \begin{pmatrix} 1 & -1 & \cdots & 1 & -1 & \left(\sum_i a_i \right) + \alpha \end{pmatrix} \begin{bmatrix} g \\ \varphi \left(\begin{pmatrix} c_1 & -b_1 c_1 + \frac{1}{2d} \\ c_1 & -b_1 c_1 - \frac{1}{2d} \\ \vdots & \vdots \\ c_n & -b_n c_n + \frac{1}{2d} \\ c_n & -b_n c_n - \frac{1}{2d} \end{pmatrix} \begin{pmatrix} z \\ 1 \end{pmatrix} \right) \\ 1 \end{bmatrix}$$

So this is indeed a multi-layer perceptron with one hidden layer.

Training Neural Networks

Flatten the parameters $\{W_l\}_{l=0}^{L}$ and the gradients $\left\{\frac{\partial L}{\partial W_l}\right\}_{l=0}^{L}$ into vectors $\vec{\theta}$ and $\frac{\partial L}{\partial \vec{\theta}}$,and then apply an iterative optimization algorithm like gradient descent.

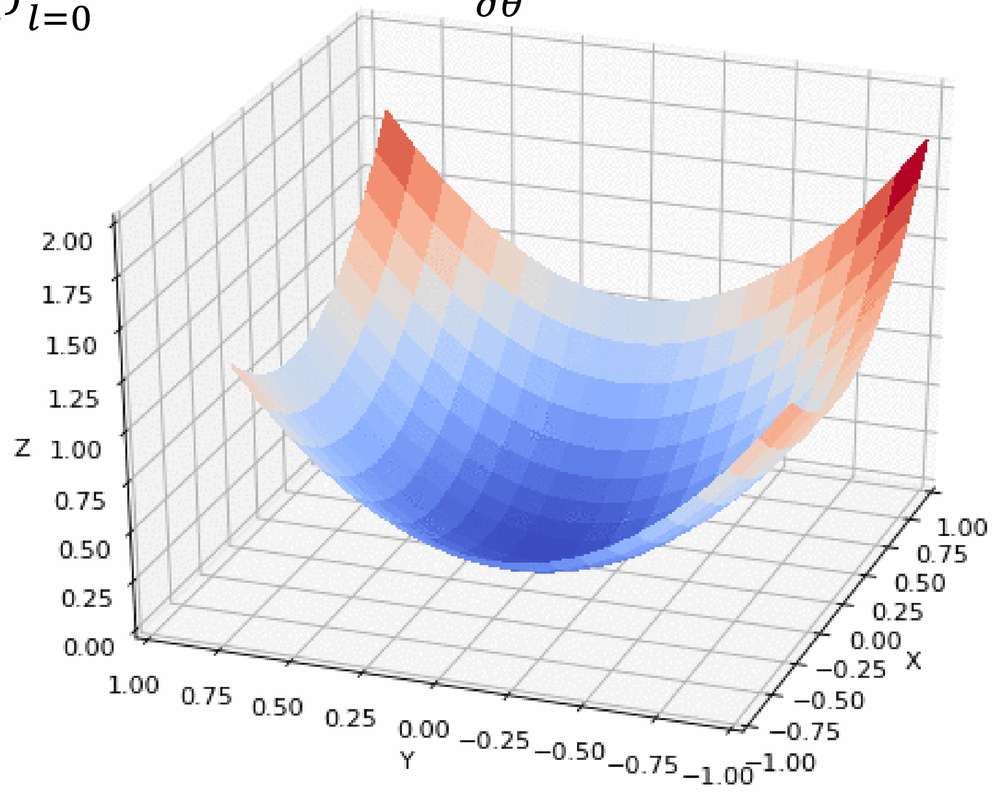
Gradient Descent:

$\vec{\theta}^{(0)} \leftarrow$ random vector
for $t = 1,2,3, \ldots$

$$\vec{\theta}^{(t)} \leftarrow \vec{\theta}^{(t-1)} - \gamma_t \frac{\partial L}{\partial \vec{\theta}}\left(\vec{\theta}^{(t-1)}\right)$$

if algorithm has converged
return $\vec{\theta}^{(t)}$

How to compute the gradients $\left\{\frac{\partial L}{\partial W_l}\right\}_{l=0}^{L}$?



Credit: Pierre Vigier

Training Neural Networks

Multivariate Chain Rule:

If $u(v, w)$ is differentiable in v and w, $v(x, y)$ and $w(x, y)$ are both differentiable in x and y,

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial v}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial w}\frac{\partial w}{\partial x}$$

So,

$$\frac{\partial L}{\partial W_l} = \frac{\partial}{\partial W_l}\sum_{i=1}^{N}\left\|\vec{y}_i - \hat{\vec{y}}_i\right\|_2^2 = \sum_{i=1}^{N}\frac{\partial}{\partial W_l}\left\|\vec{y}_i - \hat{\vec{y}}_i\right\|_2^2 = -\sum_{i=1}^{N}\sum_{j=1}^{n_{L+1}} 2(y_{i,j} - \hat{y}_{i,j})\frac{\partial \hat{y}_{i,j}}{\partial W_l}$$

Backpropagation

Backpropagation is an efficient algorithm for computing gradients.

Simple example:

Let $\vec{x} = [x_1 \quad x_2 \quad x_3 \quad x_4]^\top$. Consider

$$y = \underbrace{(x_1 + x_2)}_{z_1}\underbrace{(x_3 + x_4)}_{z_2}.$$

Naïve approach: Compute partial derivatives of the output w.r.t. the input variables and evaluate any unknown partial derivatives that arise.

(Also known as forward mode differentiation)

$$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial x_1} = \frac{\partial y}{\partial z_1} \cdot 1 + \frac{\partial y}{\partial z_2} \cdot 0$$

$$\frac{\partial y}{\partial z_1} = z_2 = x_3 + x_4$$

$$\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial x_2} + \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial x_2} = \frac{\partial y}{\partial z_1} \cdot 1 + \frac{\partial y}{\partial z_2} \cdot 0$$

$$\frac{\partial y}{\partial z_1} = z_2 = x_3 + x_4$$

Backpropagation

Backpropagation is an efficient algorithm for computing gradients.

Simple example:

Let $\vec{x} = [x_1 \quad x_2 \quad x_3 \quad x_4]^\top$. Consider

$$y = \underbrace{(x_1 + x_2)}_{z_1}\underbrace{(x_3 + x_4)}_{z_2}.$$

Naïve approach: Compute partial derivatives of the output w.r.t. the input variables and evaluate any unknown partial derivatives that arise.

(Also known as forward mode differentiation)

$$\frac{\partial y}{\partial x_3} = \frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial x_3} + \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial x_3} = \frac{\partial y}{\partial z_1} \cdot 0 + \frac{\partial y}{\partial z_2} \cdot 1$$

$$\frac{\partial y}{\partial z_2} = z_1 = x_1 + x_2$$

$$\frac{\partial y}{\partial x_4} = \frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial x_4} + \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial x_4} = \frac{\partial y}{\partial z_1} \cdot 0 + \frac{\partial y}{\partial z_2} \cdot 1$$

$$\frac{\partial y}{\partial z_2} = z_1 = x_1 + x_2$$

Need to perform 4 additions

Backpropagation

Backpropagation is an efficient algorithm for computing gradients.

Simple example:

Let $\vec{x} = [x_1 \quad x_2 \quad x_3 \quad x_4]^\top$. Consider

$$y = \underbrace{(x_1 + x_2)}_{z_1}\underbrace{(x_3 + x_4)}_{z_2}.$$

Backpropagation: Compute partial derivatives of the output w.r.t. intermediate variables before computing the partial derivatives of the output w.r.t. the input variables. This is a form of dynamic programming.

(Also known as reverse mode differentiation)

$$\frac{\partial y}{\partial z_1} = z_2 = x_3 + x_4, \frac{\partial y}{\partial z_2} = z_1 = x_1 + x_2$$

$$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial x_1} = z_2 \cdot 1 + z_1 \cdot 0 = z_2$$

$$\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial x_2} + \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial x_2} = z_2 \cdot 1 + z_1 \cdot 0 = z_2$$

$$\frac{\partial y}{\partial x_3} = \frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial x_3} + \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial x_3} = z_2 \cdot 0 + z_1 \cdot 1 = z_1$$

$$\frac{\partial y}{\partial x_4} = \frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial x_4} + \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial x_4} = z_2 \cdot 0 + z_1 \cdot 1 = z_1$$

Need to perform 2 additions

Computation Graph

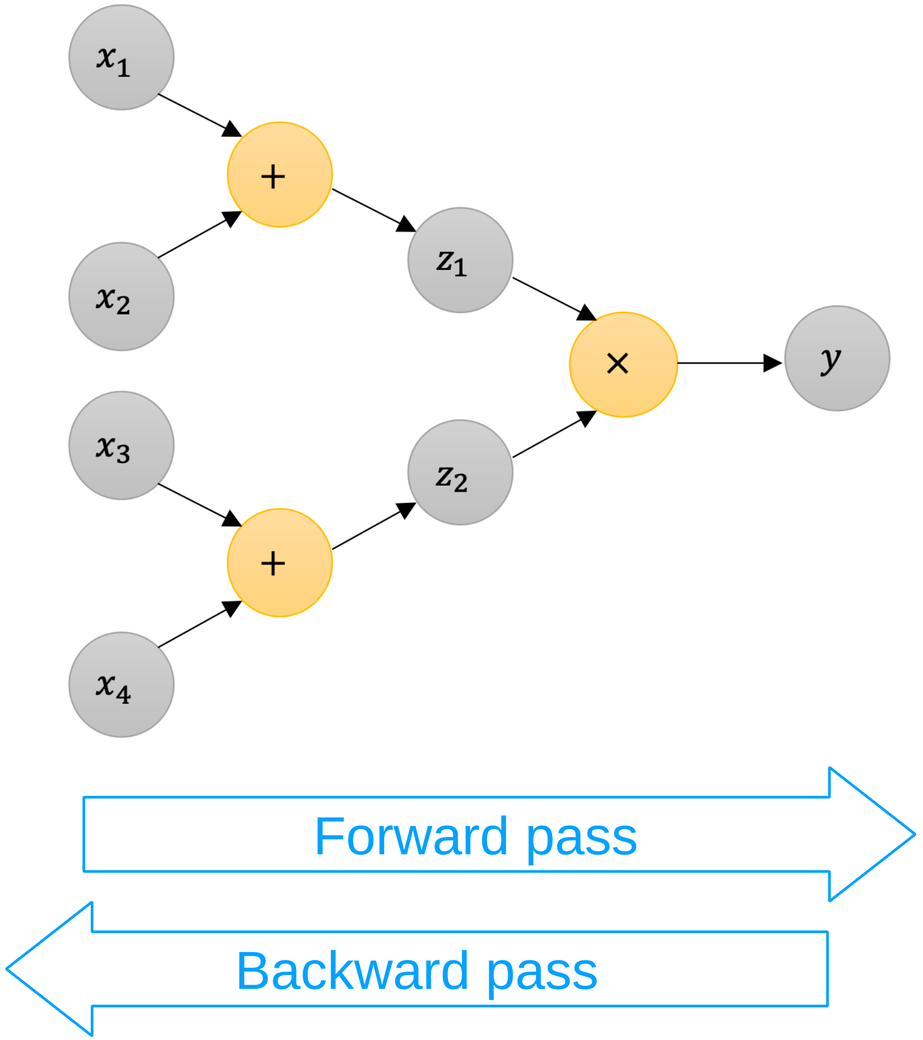
We can represent the function as a computation graph, where each intermediate variable and operator is represented as a node:

E.g.: $y = z_1 z_2$, where $z_1 = x_1 + x_2$ and $z_2 = x_3 + x_4$.

Backpropagation consists of two stages:

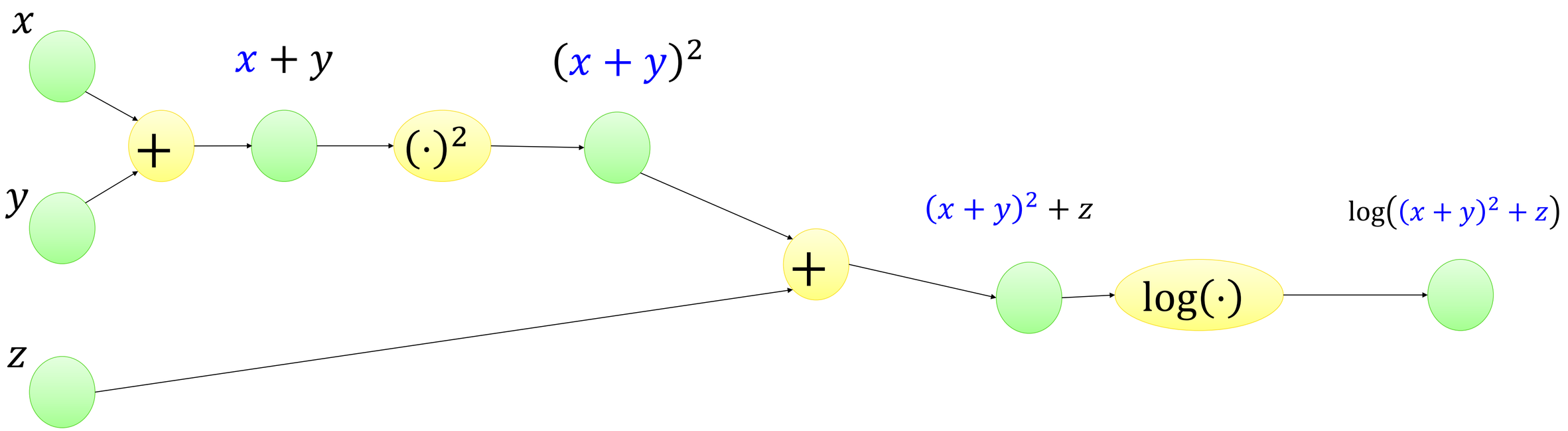
Forward pass: Computing the values of all variables, starting from the children and ending at the ancestor.

Backward pass: Computing the partial derivatives of the ancestor w.r.t. each variable, starting from the ancestor and ending at the children.



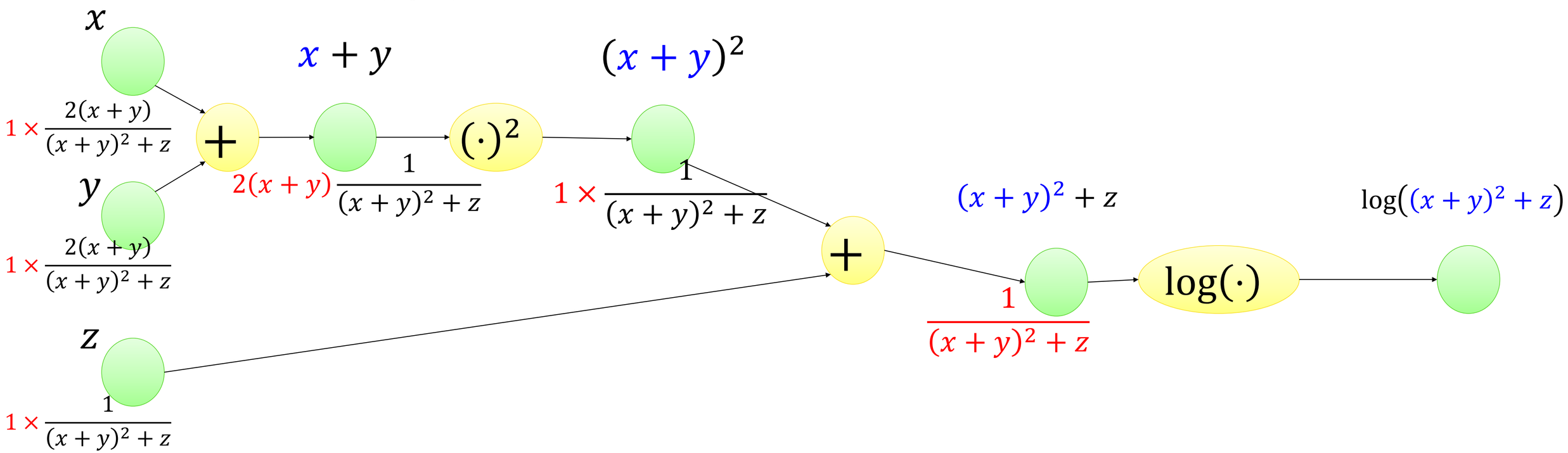
Another Example

- Consider the function $f(x, y, z) = \log\big((x + y)^2 + z\big)$



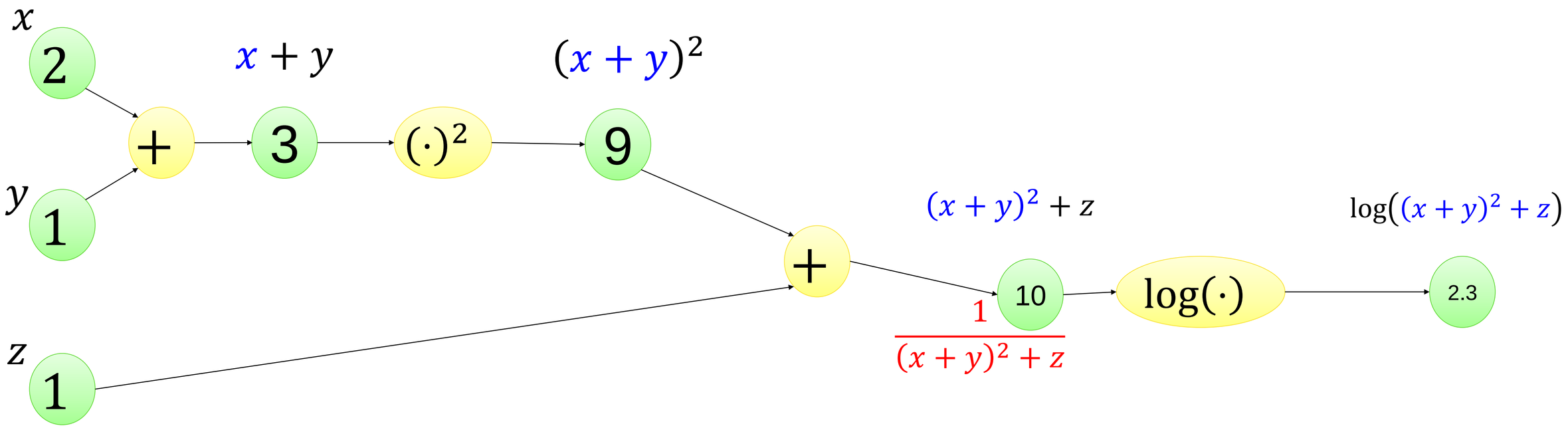
Another Example

- Consider the function $f(x, y, z) = \log\left((x + y)^2 + z\right)$
  - Compute $\frac{\partial f}{\partial (x,y,z)}$



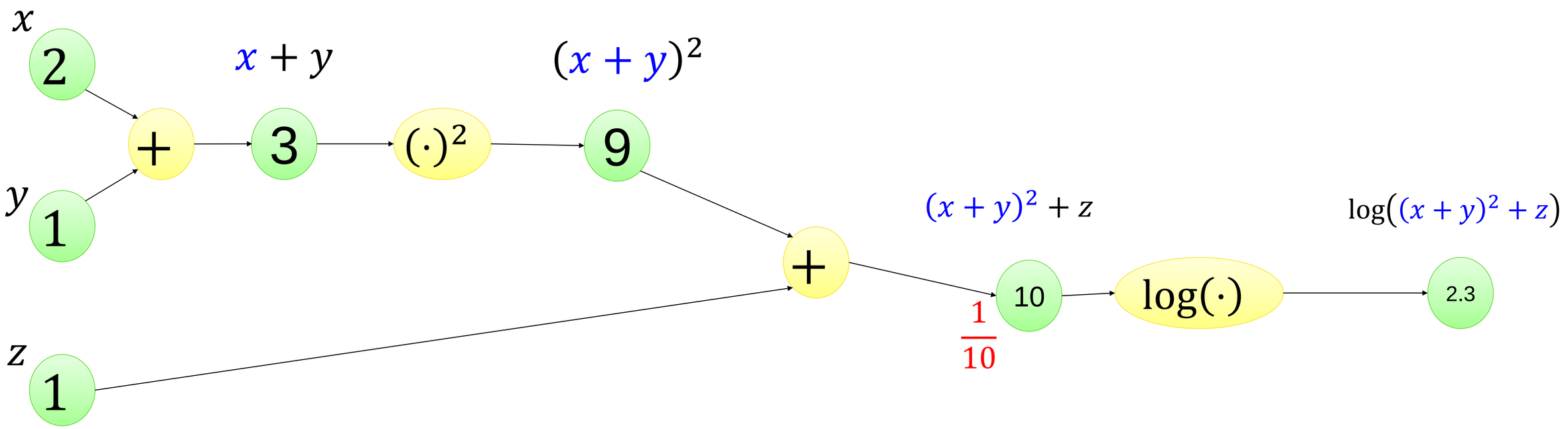
Another Example

- Consider the function $f(x, y, z) = \log\left((x + y)^2 + z\right)$
  - Compute $\frac{\partial f}{\partial x}(2,1,1)$



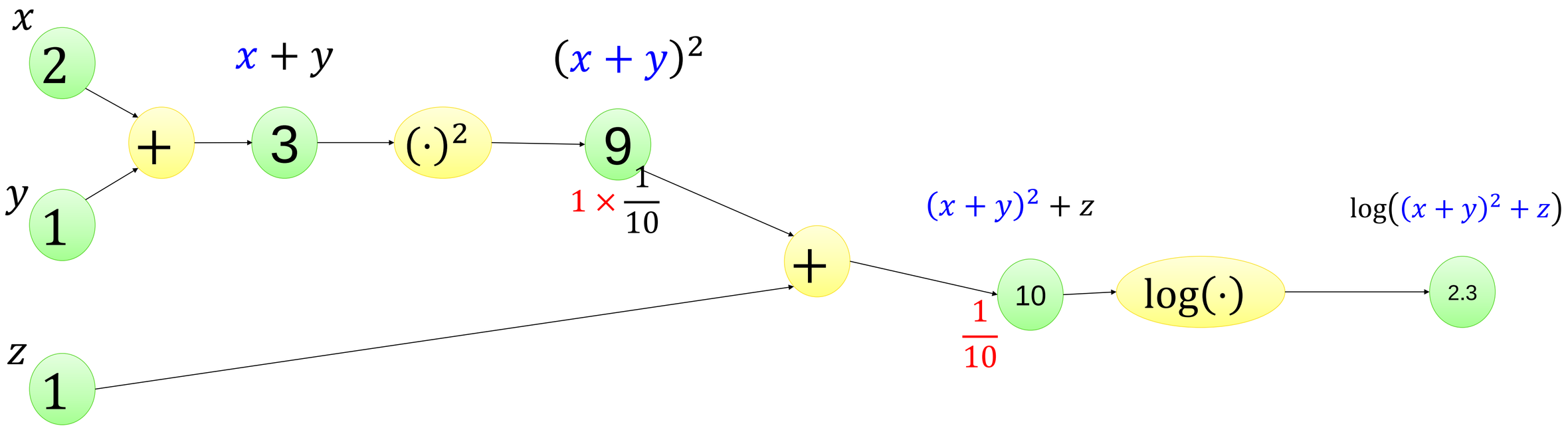
Another Example

- Consider the function $f(x, y, z) = \log\left((x + y)^2 + z\right)$
  - Compute $\frac{\partial f}{\partial x}(2,1,1)$



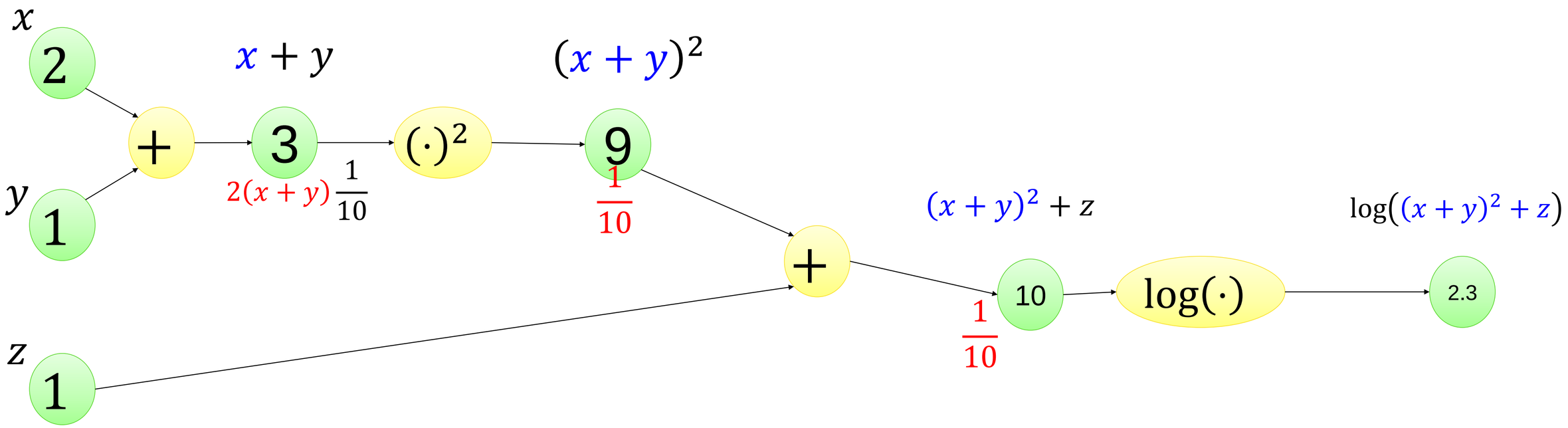
Another Example

- Consider the function $f(x, y, z) = \log\left((x + y)^2 + z\right)$
  - Compute $\frac{\partial f}{\partial x}(2,1,1)$



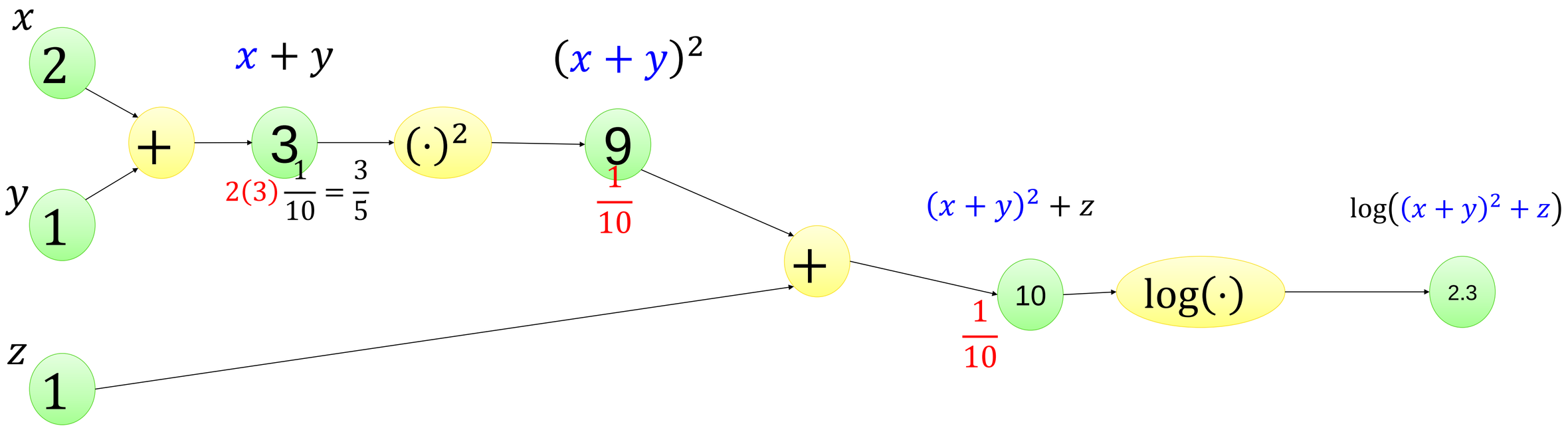
Another Example

- Consider the function $f(x, y, z) = \log\left((x + y)^2 + z\right)$
  - Compute $\frac{\partial f}{\partial x}(2,1,1)$



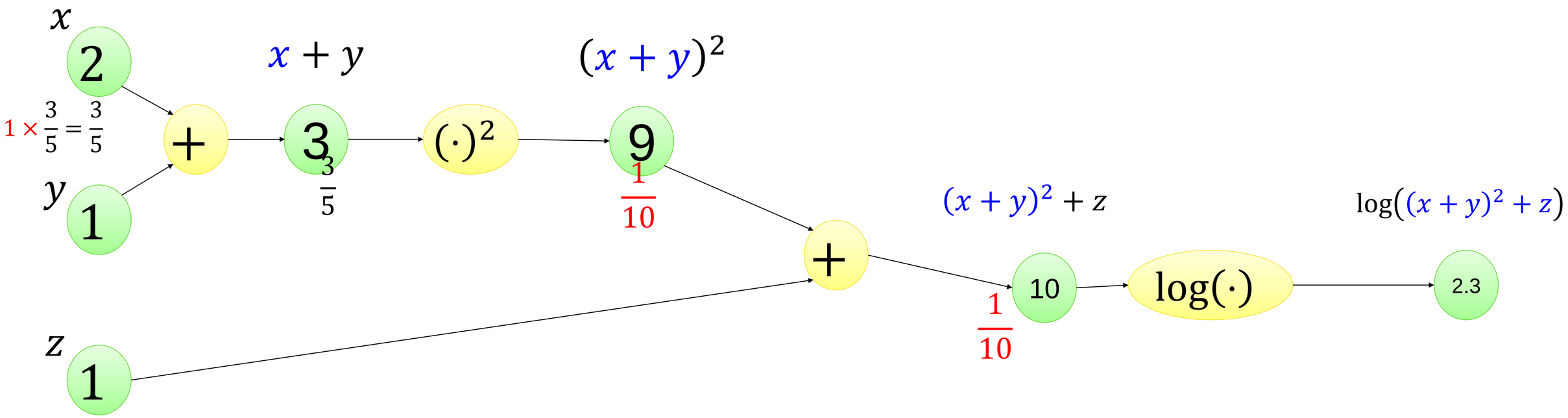
Another Example

- Consider the function $f(x, y, z) = \log\left((x + y)^2 + z\right)$
  - Compute $\frac{\partial f}{\partial x}(2,1,1)$



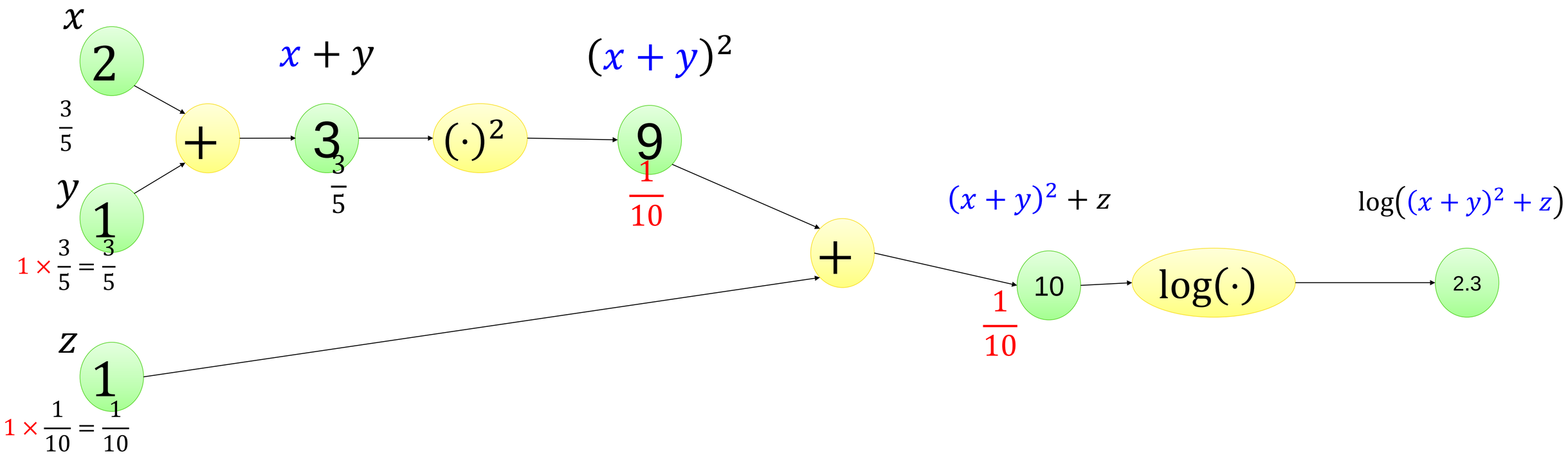
Another Example

- Consider the function $f(x, y, z) = \log\left((x + y)^2 + z\right)$
  - Compute $\frac{\partial f}{\partial x}(2,1,1)$



Another Example

- Consider the function $f(x, y, z) = \log\left((x + y)^2 + z\right)$
  - Compute $\frac{\partial f}{\partial y}(2,1,1)$, $\frac{\partial f}{\partial z}(2,1,1)$



Backpropagation in Neural Networks

Recall: Multi-layer perceptron model

$\hat{\vec{y}} = W_L \vec{h}_L$, where $\vec{h}_L = \psi(\boldsymbol{W_{L-1}\vec{h}_{L-1}})$ and $\vec{h}_{L-1}$ $= \psi(W_{L-2}\vec{h}_{L-2}), \ldots,$ and $\vec{h}_1 = \psi(W_0\vec{x})$.

Let $L(\hat{\vec{y}})$ be an arbitrary loss function.

Forward pass: Compute the values of all intermediate variables (i.e. pre- and post-activations at each hidden layer).

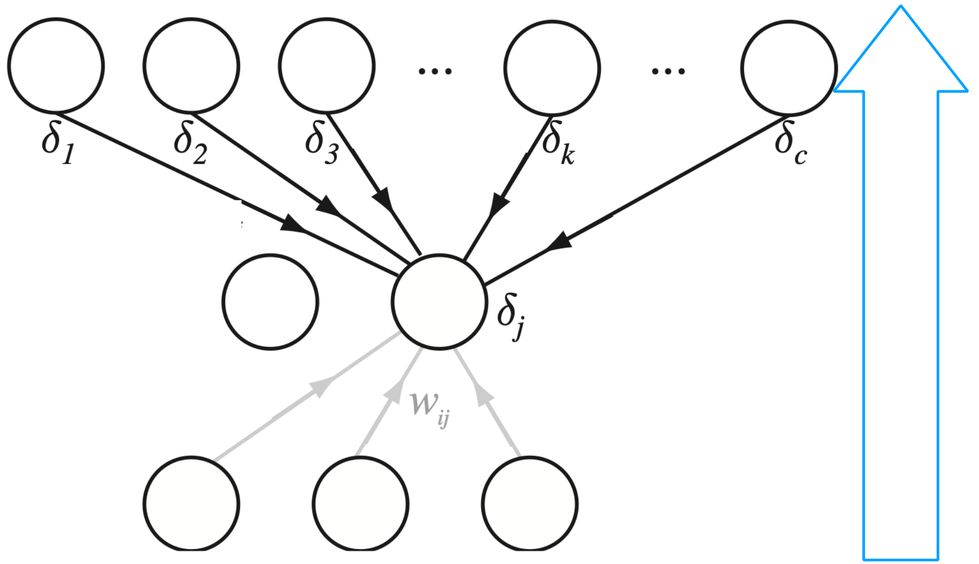
$\vec{z}_1 = W_0\vec{x}$

$\vec{h}_1 = \psi(\vec{z}_1)$

$\vec{z}_2 = W_1\vec{h}_1$

$\vec{h}_2 = \psi(\vec{z}_2)$

…



Backpropagation in Neural Networks

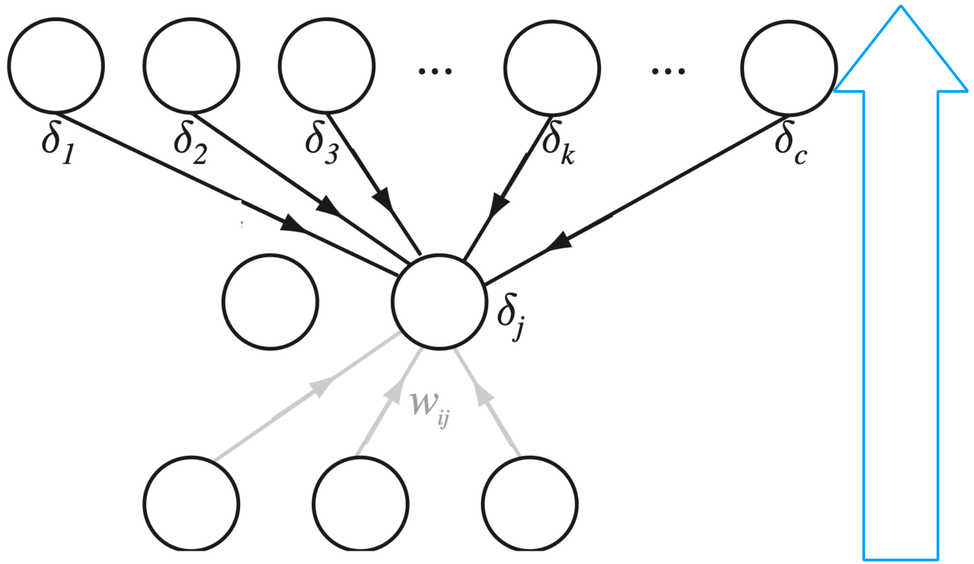
Recall: Multi-layer perceptron model

$\hat{\vec{y}} = W_L \vec{h}_L$, where $\vec{h}_L = \psi(\boldsymbol{W_{L-1}} \boldsymbol{\vec{h}_{L-1}})$ and $\vec{h}_{L-1}$ $= \psi(W_{L-2} \vec{h}_{L-2}), \ldots,$ and $\vec{h}_1 = \psi(W_0 \vec{x})$.

Let $L(\hat{\vec{y}})$ be an arbitrary loss function.

Forward pass: Compute the values of all intermediate variables (i.e. pre- and post-activations at each hidden layer).

…
$\vec{z}_L = W_{L-1} \vec{h}_{L-1}$
$\vec{h}_L = \psi(\vec{z}_L)$
$\hat{\vec{y}} = W_L \vec{h}_L$



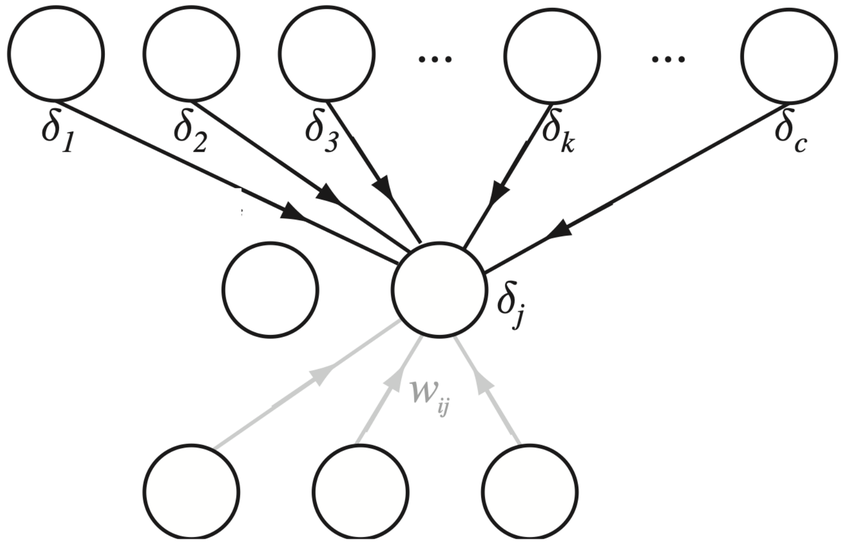
Backpropagation in Neural Networks

Recall: Multi-layer perceptron model

$\hat{\vec{y}} = \boldsymbol{W_L}\vec{\boldsymbol{h}}_{\boldsymbol{L}}$, where $\vec{h}_L = \psi(W_{L-1}\vec{h}_{L-1})$ and $\vec{h}_{L-1}$ $= \psi(W_{L-2}\vec{h}_{L-2}), \ldots,$ and $\vec{h}_1 = \psi(W_0\vec{x})$.

Let $\boldsymbol{L}(\hat{\vec{y}})$ be an arbitrary loss function.

Backward pass: Eventual goal is to compute gradient of the loss function w.r.t. first layer weights and biases. We do so by computing gradients of the loss function w.r.t. all later layers first, starting from the last layer.



Partial derivatives of an **output neuron** w.r.t. **post-activations of the last hidden layer**:

$$\hat{\vec{y}} = W_L\vec{h}_L$$

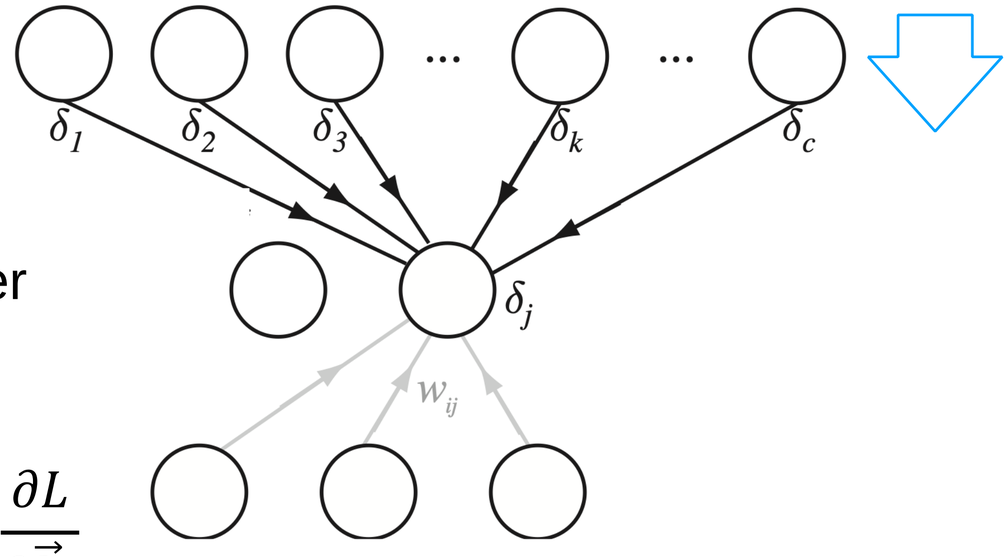
$$\Rightarrow \frac{\partial L}{\partial \vec{h}_L} = \frac{\boldsymbol{\partial}\hat{\vec{y}}}{\boldsymbol{\partial}\vec{\boldsymbol{h}}_{\boldsymbol{L}}}\frac{\partial L}{\partial \hat{\vec{y}}} = W_L^{\top}\frac{\partial L}{\partial \hat{\vec{y}}}$$

Backpropagation in Neural Networks

Recall: Multi-layer perceptron model

$\hat{\vec{y}} = W_L \vec{h}_L$, where $\vec{\boldsymbol{h}}_{\boldsymbol{L}} = \psi(\overbrace{W_{L-1}\vec{h}_{L-1}}^{\vec{\boldsymbol{z}}_{\boldsymbol{L}}})$ and $\vec{\boldsymbol{h}}_{\boldsymbol{L-1}}$ $= \psi(\underbrace{W_{L-2}\vec{h}_{L-2}}_{\vec{\boldsymbol{z}}_{\boldsymbol{L-1}}}), \ldots,$ and $\vec{\boldsymbol{h}}_{\boldsymbol{1}} = \psi(\underbrace{W_0\vec{x}}_{\vec{\boldsymbol{z}}_{\boldsymbol{1}}})$.

Let $L(\hat{\vec{y}})$ be an arbitrary loss function.



Partial derivatives of the **post-activations** of a hidden layer w.r.t. **pre-activations**:

$$\frac{\partial L}{\partial \vec{z}_l} = \frac{\partial \vec{\boldsymbol{h}}_{\boldsymbol{l}}}{\partial \vec{\boldsymbol{z}}_{\boldsymbol{l}}} \frac{\partial L}{\partial \vec{h}_l} = \begin{pmatrix} \boldsymbol{g'(z_{l,1})} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{g'(z_{l,2})} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{g'(z_{l,n_l-1})} & \boldsymbol{0} \end{pmatrix} \frac{\partial L}{\partial \vec{h}_l}$$

Backpropagation in Neural Networks

Recall: Multi-layer perceptron model

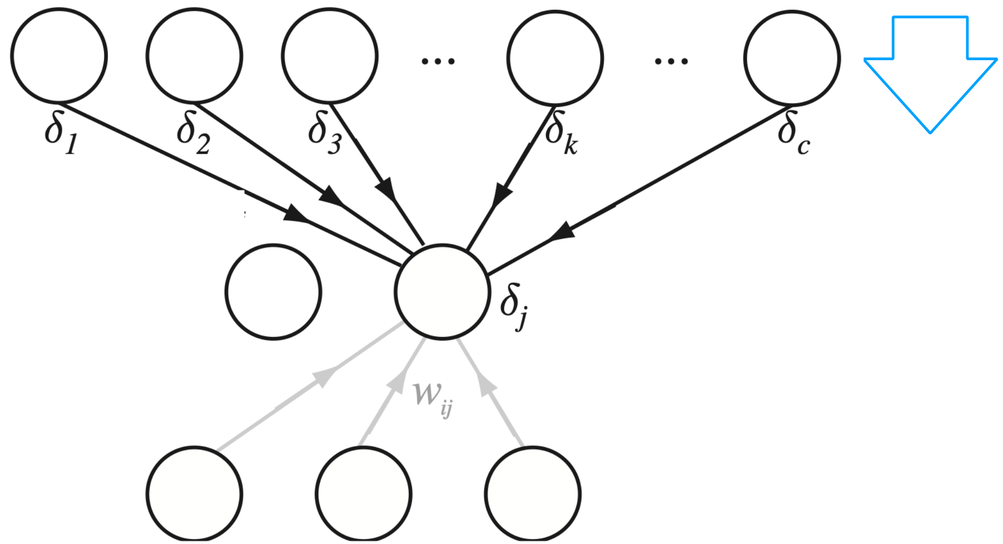
$\hat{\vec{y}} = W_L \vec{h}_L$, where $\vec{h}_L = \psi(\overbrace{W_{L-1}\vec{\boldsymbol{h}}_{\boldsymbol{L-1}}}^{\vec{\boldsymbol{z}}_{\boldsymbol{L}}})$ and $\vec{h}_{L-1}$ $= \psi(\underbrace{W_{L-2}\vec{\boldsymbol{h}}_{\boldsymbol{L-2}}}_{\vec{\boldsymbol{z}}_{\boldsymbol{L-1}}}), \ldots,$ and $\vec{h}_1 = \psi(\underbrace{W_0\vec{\boldsymbol{x}}}_{\vec{\boldsymbol{z}}_{\boldsymbol{1}}})$.

Let $L(\hat{\vec{y}})$ be an arbitrary loss function.

Partial derivatives of the **pre-activation** of a hidden neuron w.r.t. **post-activations** of the preceding layer:

$$\vec{z}_{l+1} = W_l \vec{h}_l$$

$$\Rightarrow \frac{\partial L}{\partial \vec{h}_l} = \frac{\boldsymbol{\partial \vec{z}_{l+1}}}{\boldsymbol{\partial \vec{h}_l}} \frac{\partial L}{\partial \vec{z}_{l+1}} = \boldsymbol{W_l^\top} \frac{\partial L}{\partial \vec{z}_{l+1}}$$



Backpropagation in Neural Networks

Recall: Multi-layer perceptron model

$\hat{\vec{y}} = W_L \vec{h}_L$, where $\vec{h}_L = \psi(\boldsymbol{W_{L-1}\vec{h}_{L-1}})$ and $\vec{h}_{L-1}$ $= \psi(W_{L-2}\vec{h}_{L-2}), \ldots,$ and $\vec{h}_1 = \psi(W_0\vec{x})$.

Let $L(\hat{\vec{y}})$ be an arbitrary loss function.

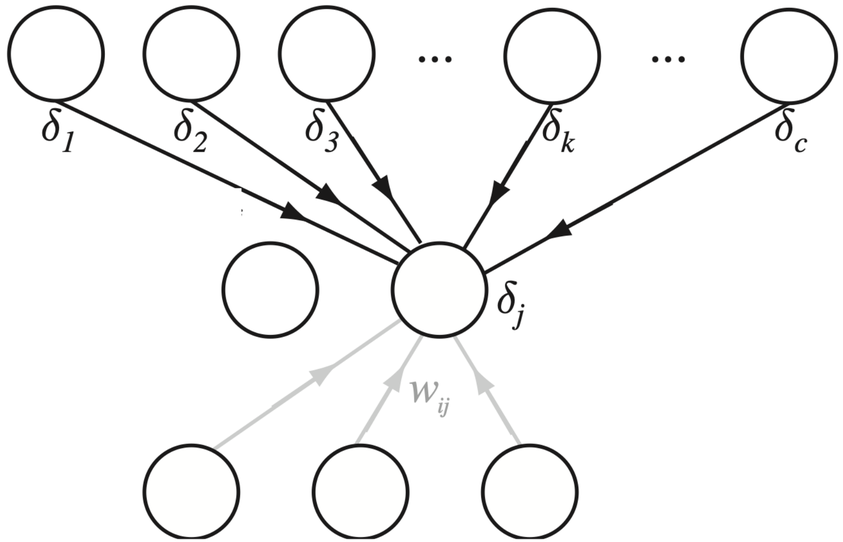
Partial derivatives of the **pre-activation** of a hidden neuron w.r.t. the **weights** between that layer and the preceding layer:

$$\frac{\partial L}{\partial \vec{w}_{l,j,\cdot}^{\top}} = \frac{\boldsymbol{\partial \vec{z}_{l+1}}}{\boldsymbol{\partial \vec{w}_{l,j,\cdot}^{\top}}} \frac{\partial L}{\partial \vec{z}_{l+1}}$$

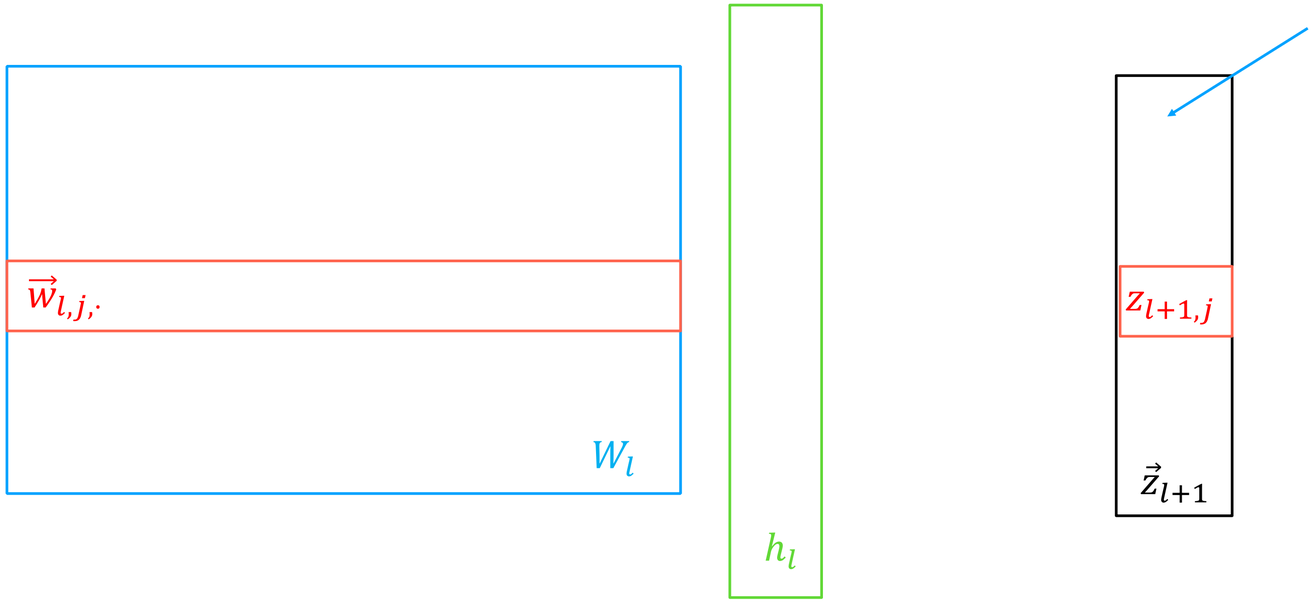
$$z_{l+1,j} = \vec{w}_{l,j,\cdot}^{\top} \vec{h}_l \quad \Rightarrow \quad \frac{\partial z_{l+1,j}}{\partial \vec{w}_{l,j,\cdot}^{\top}} = \vec{h}_l, \qquad \frac{\partial z_{l+1,j'}}{\partial \vec{w}_{l,j,\cdot}^{\top}} = \vec{0} \;\; \forall j' \neq j$$

$$\frac{\partial \vec{z}_{l+1}}{\partial \vec{w}_{l,j,\cdot}^{\top}} = \big(\vec{0} \quad \cdots \quad \underset{j\text{th column}}{\vec{0}} \quad \vec{h}_l \quad \vec{0} \quad \cdots \quad \vec{0}\big) \in \mathbb{R}^{n_l \times (n_{l+1}-1)}$$

$$\frac{\partial L}{\partial \vec{w}_{l,j,\cdot}^{\top}} = \frac{\partial \vec{z}_{l+1}}{\partial \vec{w}_{l,j,\cdot}^{\top}} \frac{\partial L}{\partial \vec{z}_{l+1}} = \big(\vec{0} \quad \cdots \quad \underset{j\text{th column}}{\vec{0}} \quad \vec{h}_l \quad \vec{0} \quad \cdots \quad \vec{0}\big) \frac{\partial L}{\partial \vec{z}_{l+1}}$$



Visualization



A Neural Network Playground

- https://playground.tensorflow.org/