Backward Propagation Revisited for Deep Neural Networks

Deep Learning

Brad Quinton, Scott Chin

Learning Objectives

- Become able to apply backpropagation on any arbitrarily complex computation graph using a systematic numerical approach
- Become familiar with backprop through common mathematical operations

Brad Quinton, Scott Chin

History of Back Propagation

- "Learning Internal Representations by Error Propagation", Rumelhart, Hinton, Williams,1986, http://www.cs.toronto.edu/~hinton/absps/pdp8.pdf
- Popularized use of backpropagation for training neural networks in 1986 → Before this, people used adhoc rules for parameter search
- Now have a systematic framework to train deep networks
- Reinvigorated area of Neural Networks
- Many advances in ability to train deep networks due to understanding backprop and changing things to work well with it.

Brad Quinton, Scott Chin

Training with Gradient Descent and Back Prop

- Question at each iteration:

Given the current parameter values, how should we change the them to reduce the cost (loss)?

- Answer:

The partial derivative of the cost w.r.t. each parameter tells us how to update each parameter

Brad Quinton, Scott Chin

Why the derivative?

$$a_{103}^{[3]} = g\left(\ldots + w_{103,201}^{[3]} a_{201}^{[2]} + \cdots \right)$$

$$\frac{\partial J}{\partial w_{103,201}^{[3]}} = -3$$

Why the derivative?

$a_{103}^{[3]} = g\left(\ldots + w_{103,201}^{[3]} a_{201}^{[2]} + \cdots \right)$

$$\frac{\partial J}{\partial w_{103,201}^{[3]}} = -3$$

- If we increase the value of $w_{103,201}^{[3]}$, we will **decrease** the value of J
- Changing $w_{103,201}^{[3]}$ by 1 will change J by 3 (magnitude of impact)
- To minimize J in this case, we should increase $w_{103,201}^{[3]}$
- From our Gradient Descent update step:

$$w_{103,201}^{[3]} = w_{103,201}^{[3]} - \propto \frac{\partial J}{\partial w_{103,201}^{[3]}} = w_{103,201}^{[3]} - \propto (-3)$$

→ Increasing

Brad Quinton, Scott Chin

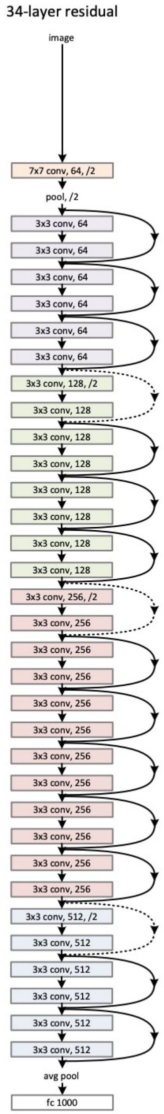
Calculating Closed-form Partial Derivatives

- For a 3-layer neural network:
  - $\hat{y}(x) = \sigma\left(W^{[3]}\tanh\left(W^{[2]}\tanh\left(W^{[1]}x + B^{[1]}\right) + B^{[2]}\right) + B^{[3]}\right)$
- Cost function:
  - $J\left(L\left(g_3\left(f_3\left(g_2\left(f_2\left(g_1\left(f_1\left(x, W^{[1]}, B^{[1]}\right)\right), W^{[2]}, B^{[2]}\right)\right), W^{[3]}, B^{[3]}\right)\right), y\right)\right)$
- Some of the partial derivatives for the weights
  - $\frac{\partial L}{\partial w_{i,j}^{[1]}} = \frac{\partial L}{\partial g_3}\frac{\partial g_3}{\partial f_3}\frac{\partial f_3}{\partial g_2}\frac{\partial g_2}{\partial f_2}\frac{\partial f_2}{\partial g_1}\frac{\partial g_1}{\partial f_1}\frac{\partial f_1}{\partial w_{i,j}^{[1]}}$
  - $\frac{\partial L}{\partial w_{i,j}^{[2]}} = \frac{\partial L}{\partial g_3}\frac{\partial g_3}{\partial f_3}\frac{\partial f_3}{\partial g_2}\frac{\partial g_2}{\partial f_1}\frac{\partial f_2}{\partial w_{i,j}^{[2]}}$

Brad Quinton, Scott Chin

Calculating Closed-form Partial Derivatives

- Becomes infeasible and error prone with deep networks and many parameters
- If you want to try a different loss function or make architectural changes like trying different activation functions, need to derive again
- This is where Backpropagation really shines
- Let’s talk more about computation graphs



ResNet152 CNN Arch
152 Layers
60million parameters

ResNet34 shown here

Deep Residual Learning for Image Recognition https://arxiv.org/pdf/1512.03385.pdf

Don't Frameworks handle this for us?

- High-level frameworks such as Tensorflow and PyTorch implement Automatic Differentiation
- So Yes, in practice, you will probably never have to write any backpropagation code. But it is very important to understand how it works
- It lets you understand how a lot of important things work and why they work (e.g. activation functions)
- Without understanding backprop, you will always only see the neural network training process as a black-box

Brad Quinton, Scott Chin

Backprop on Computation Graphs

Context

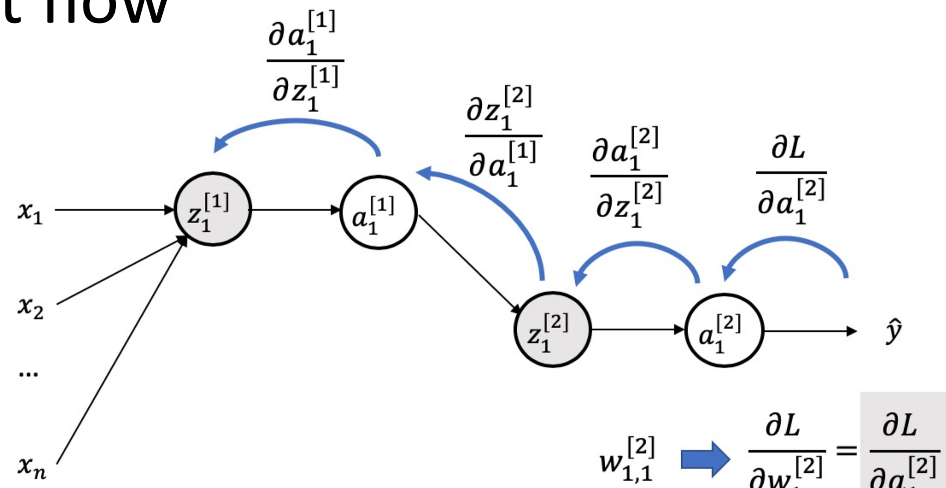
- For the context of this lecture, thinking of a single training sample. i.e. one piece of data
- Therefore, we can talk about Loss and Cost interchangeably
- Next Tuesday, we will talk about Vectorized Backprop and talk about multiple (training) samples

Quick Note on terminology

- We will often use the terms *gradient* and *partial derivative* interchangeably although they are technically not the same
- Let $f(x, y)$ be a multi-variable function
- Partial derivative of f with respect to x means $\frac{\partial f}{\partial x}$
- Gradient of f is a vector of the partial derivatives of all inputs to f
  $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

From Earlier:

- We introduced the concept of backprop using computation graphs
- Now let's take a closer look to investigate some patterns from the perspective of gradient flow



$w_{1,1}^{[2]} \Rightarrow \frac{\partial L}{\partial w_1^{[2]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}}$

$b^{[2]} \Rightarrow \frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial b^{[2]}}$

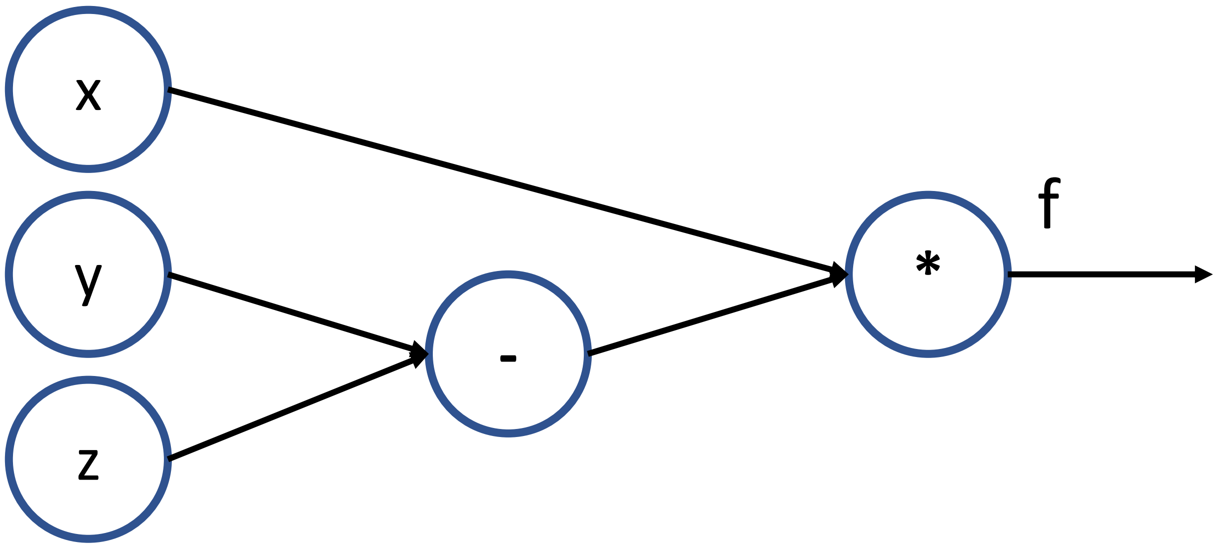
$w_{1,1}^{[1]} \Rightarrow \frac{\partial L}{\partial w_1^{[1]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} \cdot \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \cdot \frac{\partial z_1^{[1]}}{\partial w_1^{[1]}}$

$b^{[1]} \Rightarrow \frac{\partial L}{\partial b^{[1]}} = \frac{\partial L}{\partial a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \cdot \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} \cdot \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \cdot \frac{\partial z_1^{[1]}}{\partial b^{[1]}}$

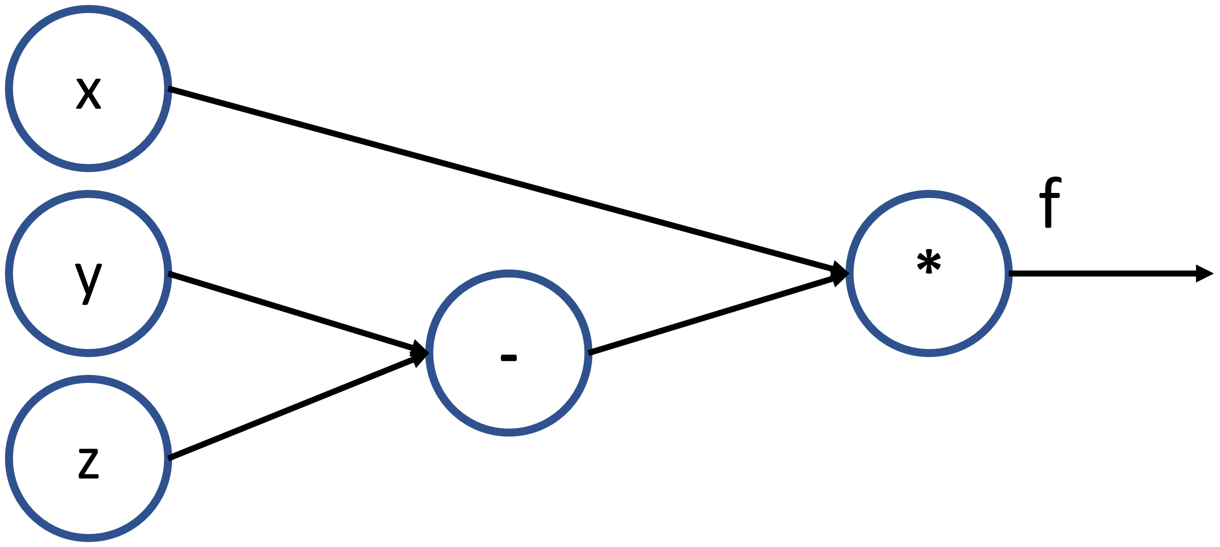
Notice how the same derivatives are re-used "across" and "back" through the NN... Brad Quintc

Recall: Computation Graphs

- Directed graph where nodes are operations or variables
- Edges are flow of data
- Inputs to a node are variables or outputs of other operation nodes



Brad Quinton, Scott Chin



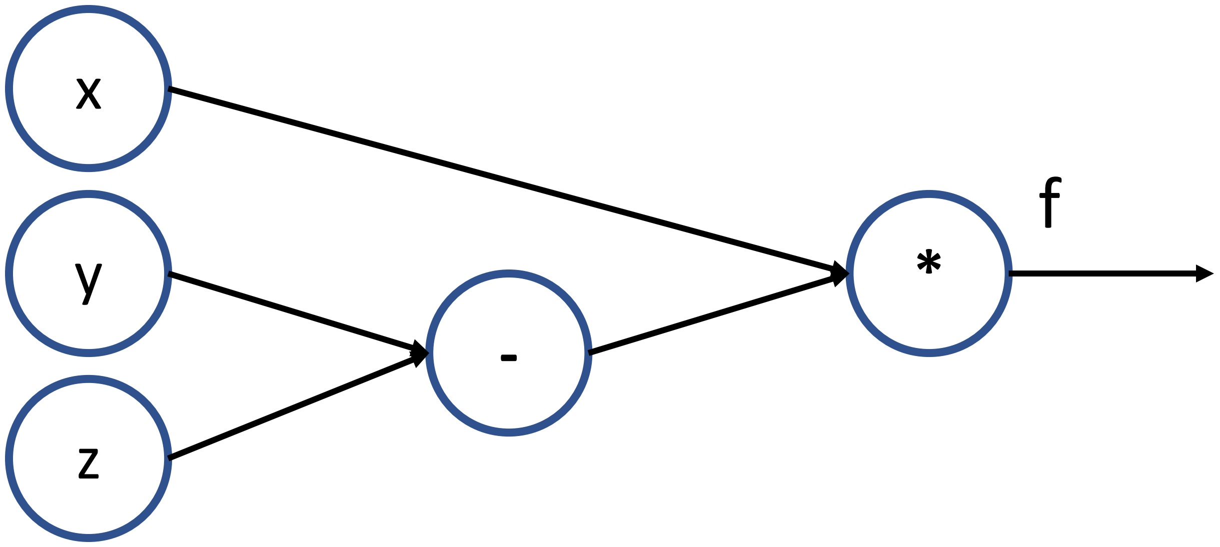
$$f = x * (y - z)$$

Goal

Find the partial derivatives of final output w.r.t. all graph inputs

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

Brad Quinton, Scott Chin



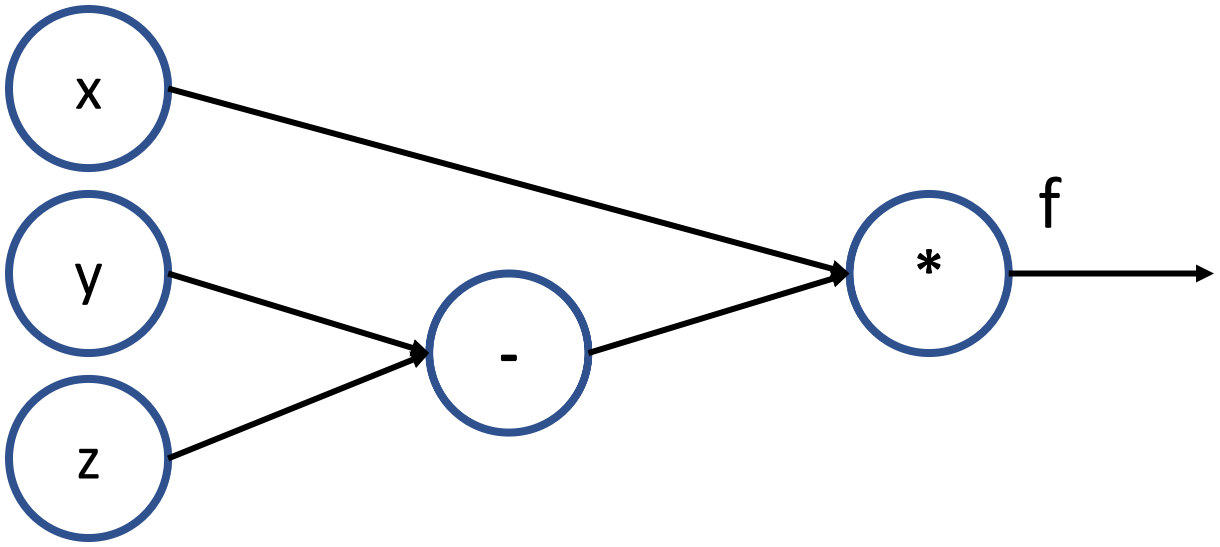
$$f = x * (y - z)$$

Goal

Find the partial derivatives of final output w.r.t. all graph inputs

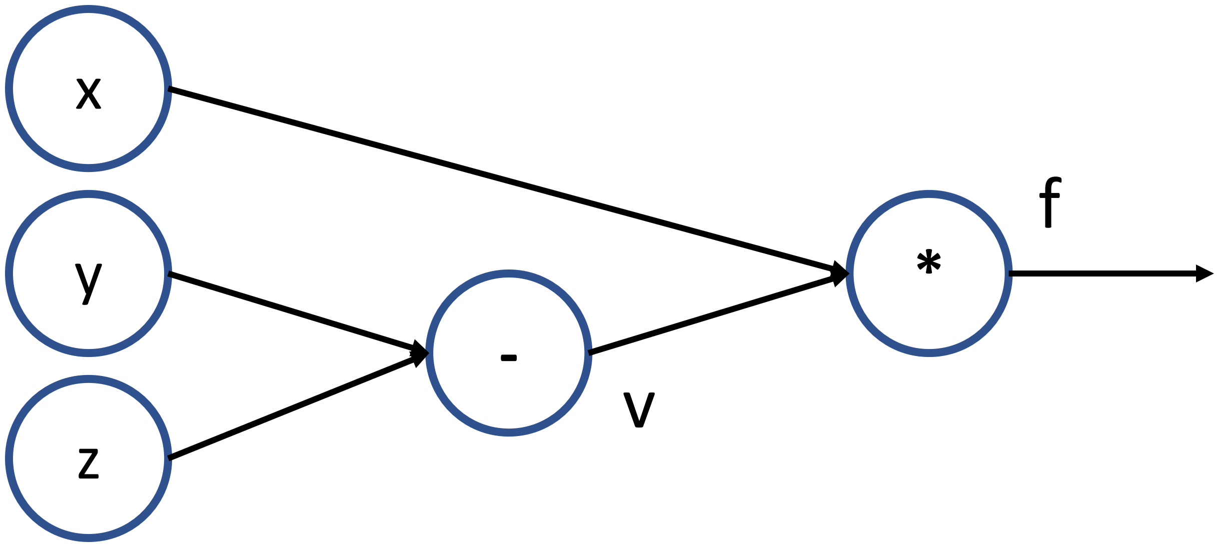
$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

After this example, I will give you another one to try on your own first. If you want to give that one a try, then you may want to pay extra attention here. Seems like a fitting exam question …☺



$$f = x * (y - z)$$

At Graph Construction

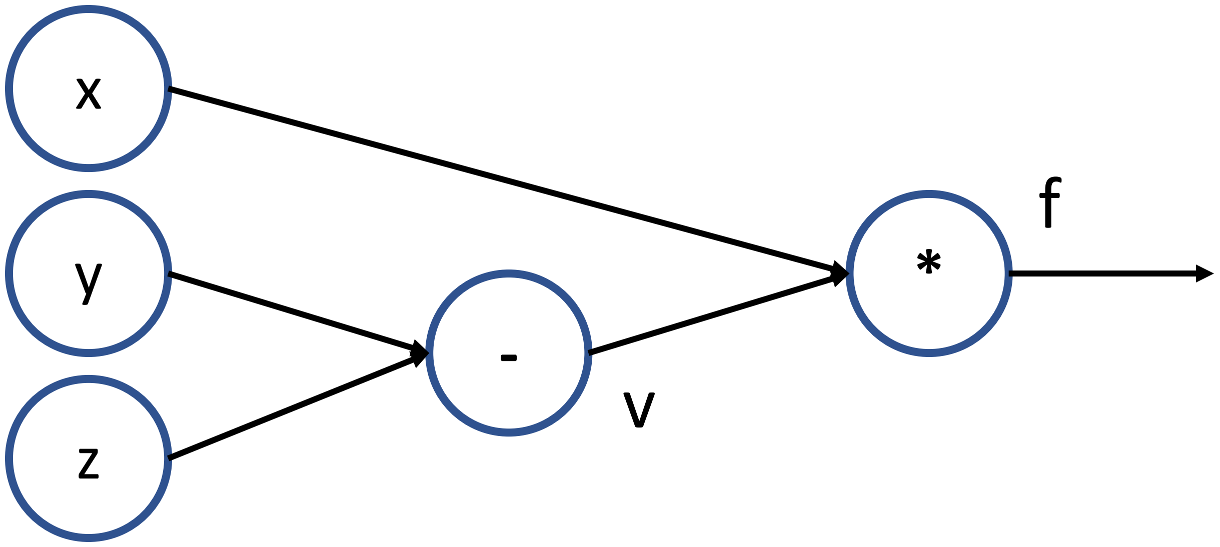


$$f = x * (y - z)$$

At Graph Construction

1. Assign variable names to each intermediate node's output

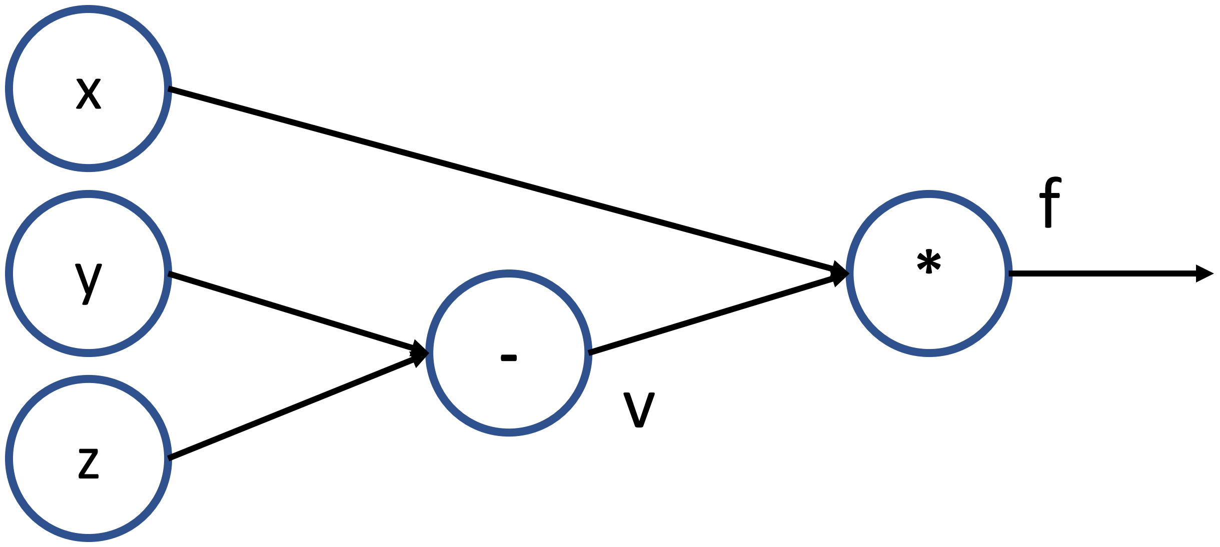
Brad Quinton, Scott Chin



$$f = x * v \qquad\qquad v = y - z$$

At Graph Construction

1. Assign variable names to each intermediate node's output
2. Re-express each node as a function of its immediate inputs



$$f = x * v$$

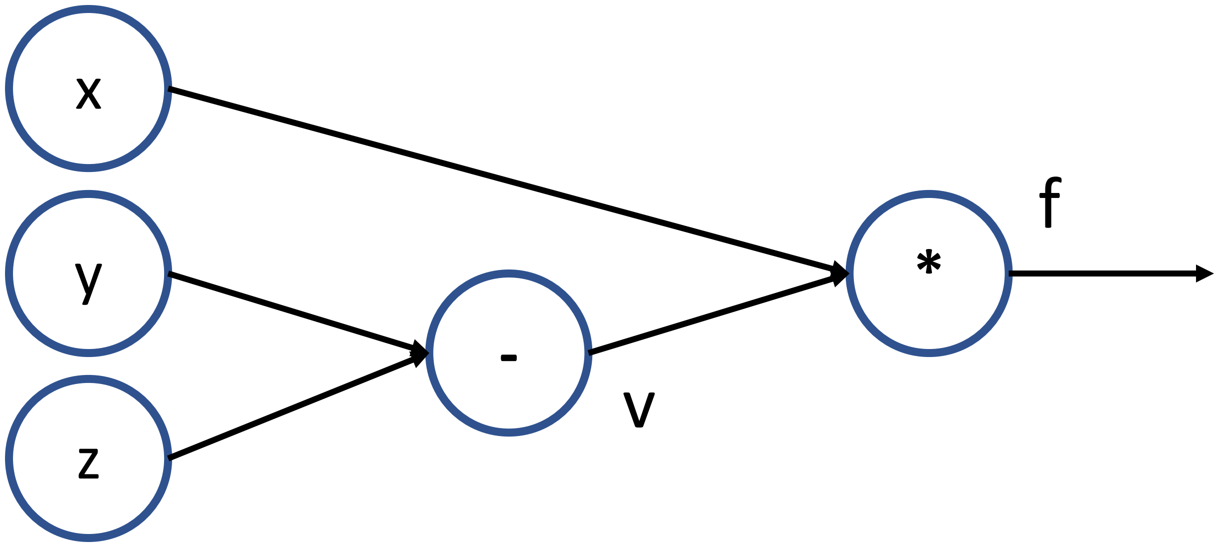
$$\frac{\partial f}{\partial x} = v \, , \frac{\partial f}{\partial v} = x$$

$$v = y - z$$

$$\frac{\partial v}{\partial y} = 1 \, , \frac{\partial v}{\partial z} = -1$$

<u>At Graph Construction</u>

1. Assign variable names to each intermediate node’s output
2. Re-express each node as a function of its immediate inputs
3. Derive ”local” gradients of each node’s output w.r.t. its immediate inputs. These should be simple derivations.



$$f = x * v$$

$$\frac{\partial f}{\partial x} = v \, , \frac{\partial f}{\partial v} = x$$

$$v = y - z$$

$$\frac{\partial v}{\partial y} = 1 \, , \frac{\partial v}{\partial z} = -1$$

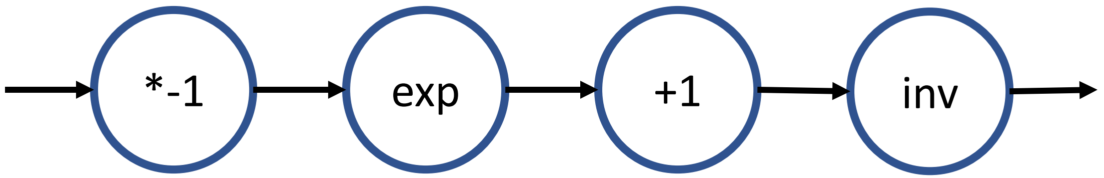
<u>At Graph Construction</u>

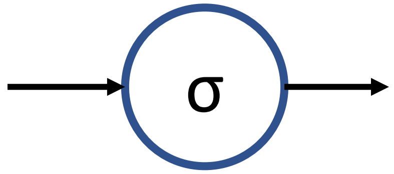
- No data yet.
- We’ve only defined the operations of the network
- Analogous to hardware design, and writing the RTL (i.e. defining the structure of a digital circuit).
- Much of writing TensorFlow/PyTorch code is defining the structure of you’re network.

Aside: Compute Graph Representation

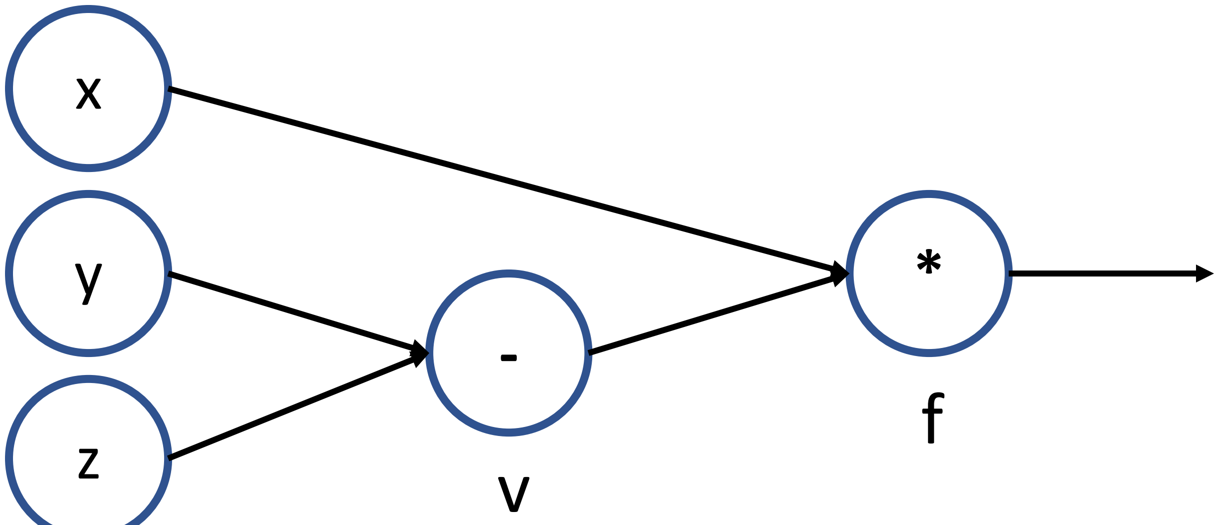
- The same function can be represented in different ways.
- Each operation node can be as simple or complex as you want.
- For backprop, choose a representation where the local gradients of each node are easy to derive and compute

Example: Two representations for Sigmoid function





Brad Quinton, Scott Chin

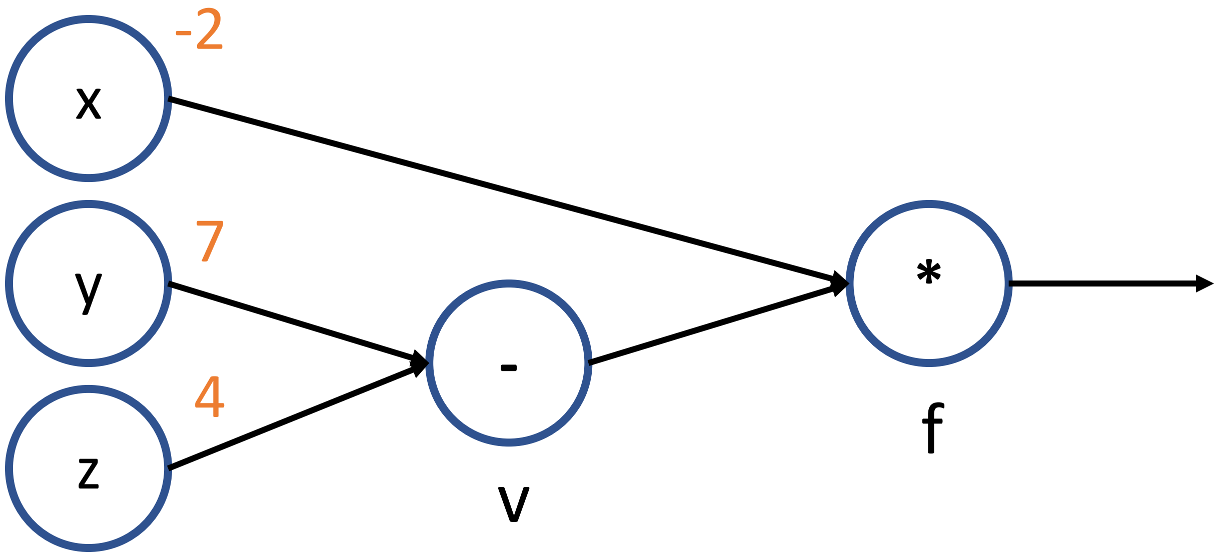


Forward Propagation

$$f = x * v \qquad\qquad v = y - z$$

$$\frac{\partial f}{\partial x} = v \,, \frac{\partial f}{\partial v} = x \qquad\qquad \frac{\partial v}{\partial y} = 1 \,, \frac{\partial v}{\partial z} = -1$$

Brad Quinton, Scott Chin



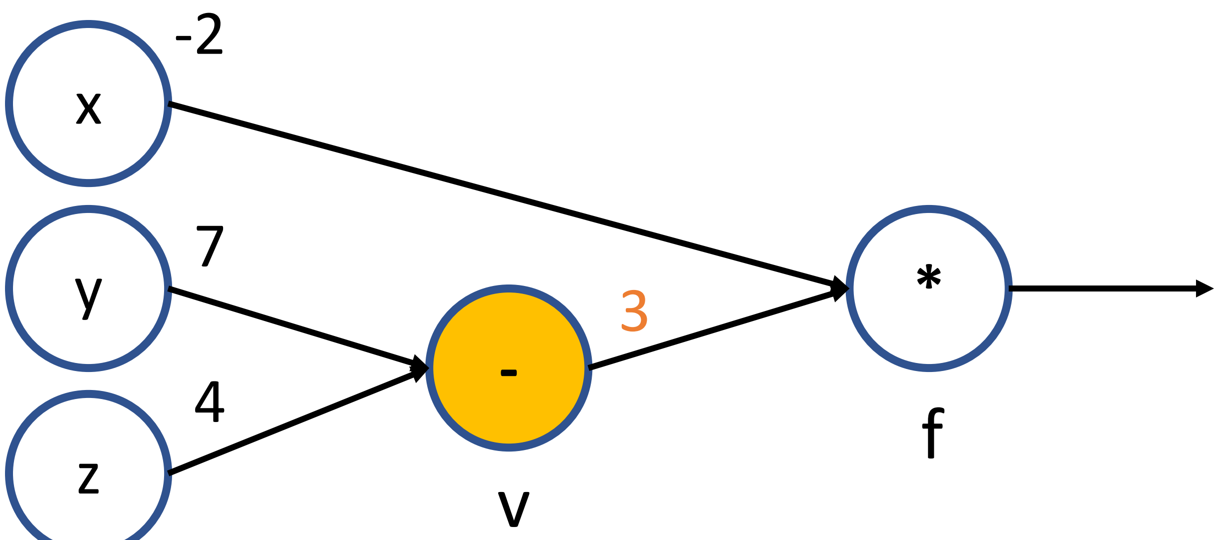
Forward Propagation

1. Values are supplied to input variables

$$f = x * v \qquad v = y - z$$

$$\frac{\partial f}{\partial x} = v \,, \frac{\partial f}{\partial v} = x \qquad \frac{\partial v}{\partial y} = 1 \,, \frac{\partial v}{\partial z} = -1$$

$$\boxed{x = -2, y = 7, z = 4}$$



Forward Propagation

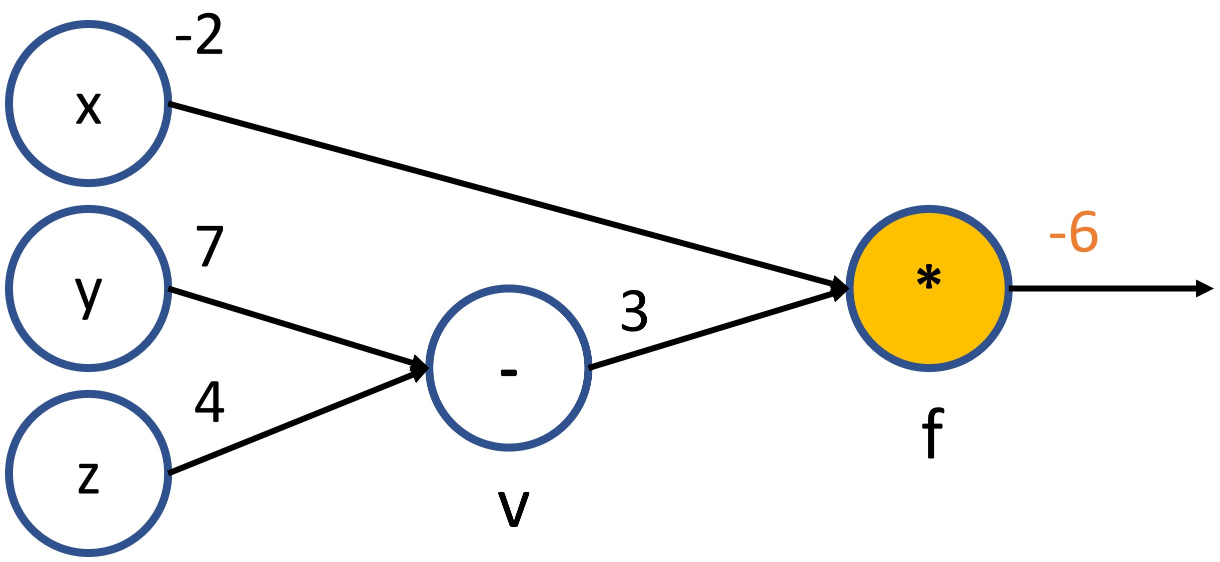
1. Values are supplied to input variables
2. For each node that has values for all of its inputs, compute output and propagate forward

$$f = x * v \qquad v = y - z$$

$$\frac{\partial f}{\partial x} = v, \frac{\partial f}{\partial v} = x \qquad \frac{\partial v}{\partial y} = 1, \frac{\partial v}{\partial z} = -1$$

$$x = -2, y = 7, z = 4$$

$$\boxed{v = 7 - 4 = 3}$$



$$f = x * v \qquad v = y - z$$

$$\frac{\partial f}{\partial x} = v \,, \frac{\partial f}{\partial v} = x \qquad \frac{\partial v}{\partial y} = 1 \,, \frac{\partial v}{\partial z} = -1$$

$$x = -2, y = 7, z = 4$$

$$v = 7 - 4 = 3$$

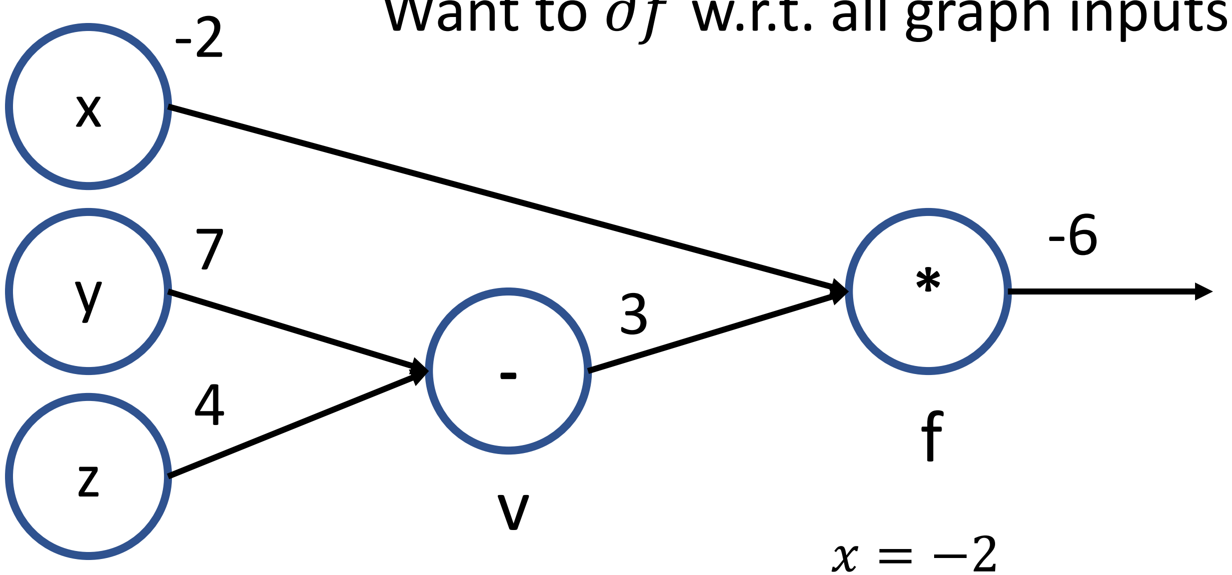
$$\boxed{f = x * v = -2 * 3 = -6}$$

Forward Propagation

1. Values are supplied to input variables
2. For each node that has values for all of its inputs, compute output and propagate forward
3. Repeat until all node outputs computed

Want to ∂f w.r.t. all graph inputs (x, y, z)

Backward Propagation



$x = -2$
$y = 7$
$z = 4$
$v = 3$
$f = -6$

$f = x * v$

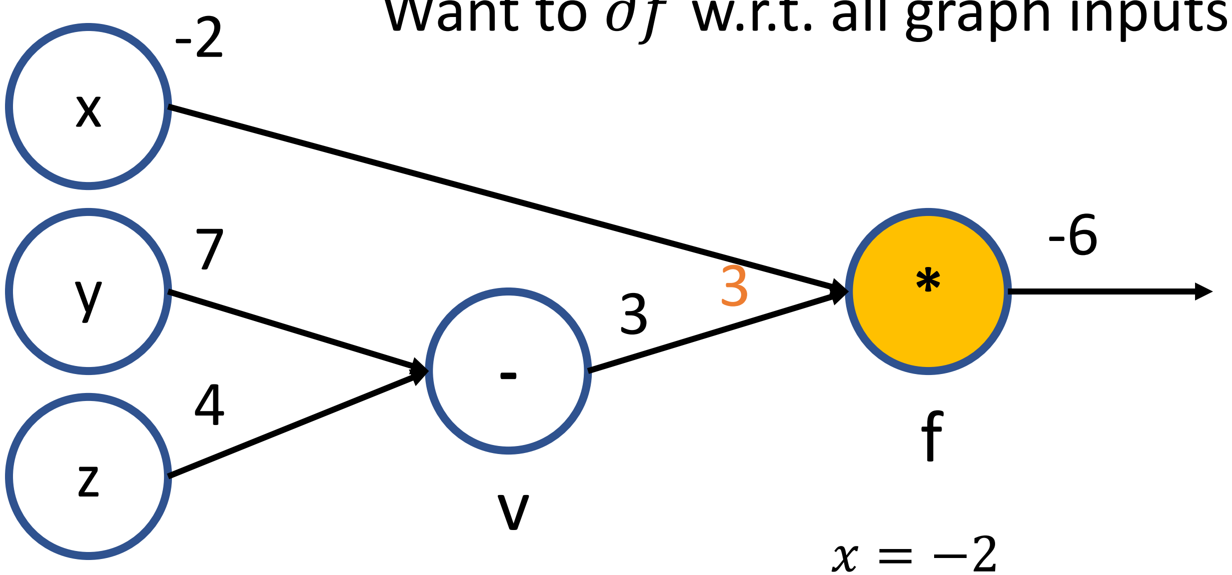
$v = y - z$

$\frac{\partial f}{\partial x} = v, \frac{\partial f}{\partial v} = x$

$\frac{\partial v}{\partial y} = 1, \frac{\partial v}{\partial z} = -1$

Brad Quinton, Scott Chin

Want to ∂f w.r.t. all graph inputs (x, y, z)



$f = x * v$

$v = y - z$

$x = -2$
$y = 7$
$z = 4$
$v = 3$
$f = -6$

$\frac{\partial f}{\partial x} = v, \frac{\partial f}{\partial v} = x$

$\frac{\partial v}{\partial y} = 1, \frac{\partial v}{\partial z} = -1$

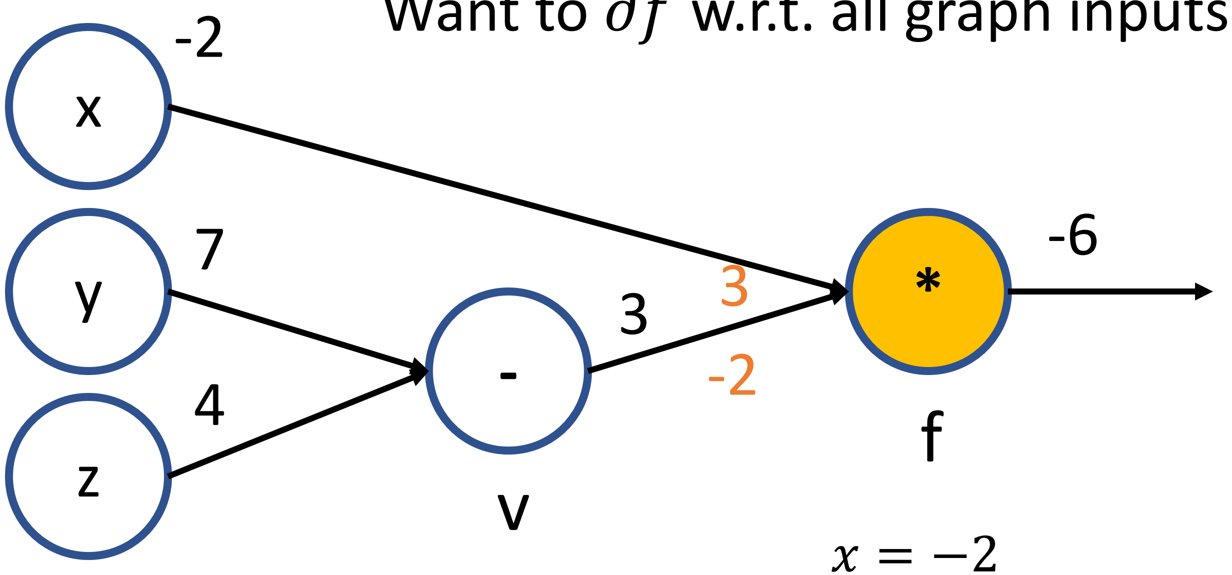
$$\boxed{\frac{\partial f}{\partial x} = v = 3}$$

Backward Propagation

1. Compute input gradient on the output node(s)

Brad Quinton, Scott Chin

Want to ∂f w.r.t. all graph inputs (x, y, z)



Backward Propagation

1. Compute input gradient on the output node(s)

$f = x * v$

$\frac{\partial f}{\partial x} = v, \frac{\partial f}{\partial v} = x$

$v = y - z$

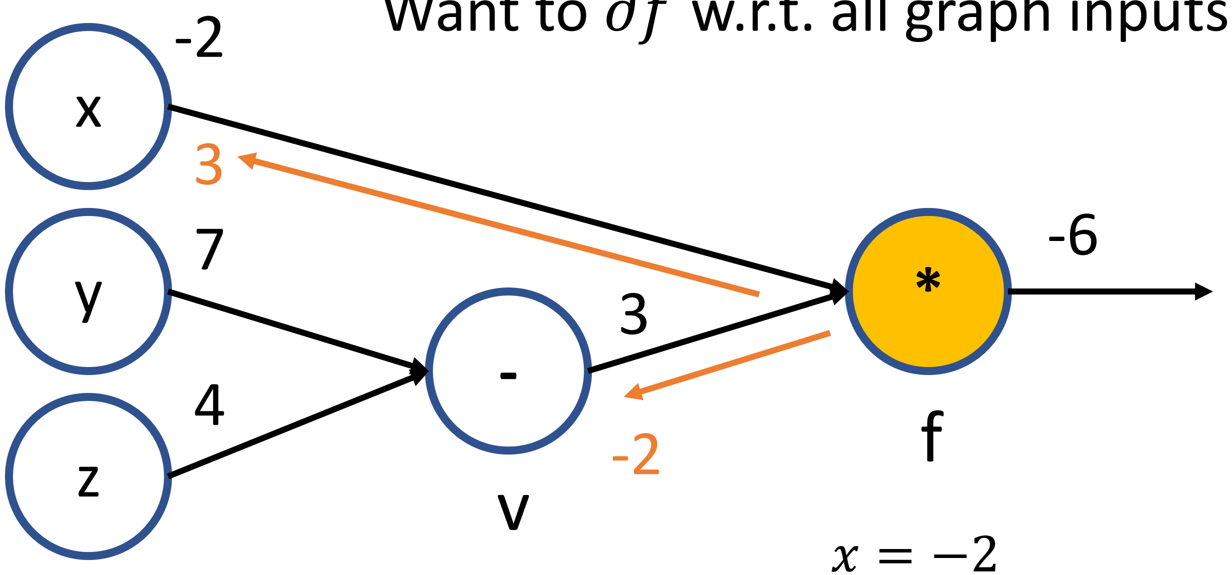
$\frac{\partial v}{\partial y} = 1, \frac{\partial v}{\partial z} = -1$

$x = -2$
$y = 7$
$z = 4$
$v = 3$
$f = -6$

$$\boxed{\begin{aligned}\frac{\partial f}{\partial x} &= v = 3, \\ \frac{\partial f}{\partial v} &= x = -2\end{aligned}}$$

Brad Quinton, Scott Chin

Want to ∂f w.r.t. all graph inputs (x, y, z)



$f = x * v$

$\frac{\partial f}{\partial x} = v, \frac{\partial f}{\partial v} = x$

$v = y - z$

$\frac{\partial v}{\partial y} = 1, \frac{\partial v}{\partial z} = -1$

$x = -2$
$y = 7$
$z = 4$
$v = 3$
$f = -6$

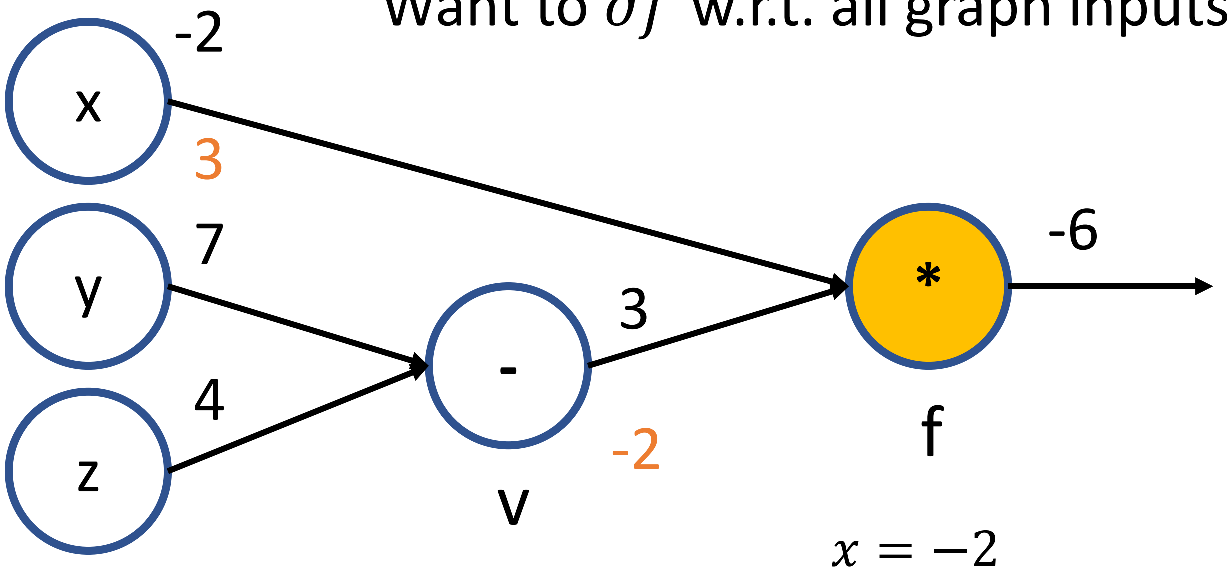
$$\boxed{\frac{\partial f}{\partial x} = v = 3, \quad \frac{\partial f}{\partial v} = x = -2}$$

Backward Propagation

1. Compute input gradient on the output node(s)

Brad Quinton, Scott Chin

Want to ∂f w.r.t. all graph inputs (x, y, z)



Backward Propagation

1. Compute input gradient on the output node(s)

$f = x * v$

$v = y - z$

$\frac{\partial f}{\partial x} = v \, , \frac{\partial f}{\partial v} = x$

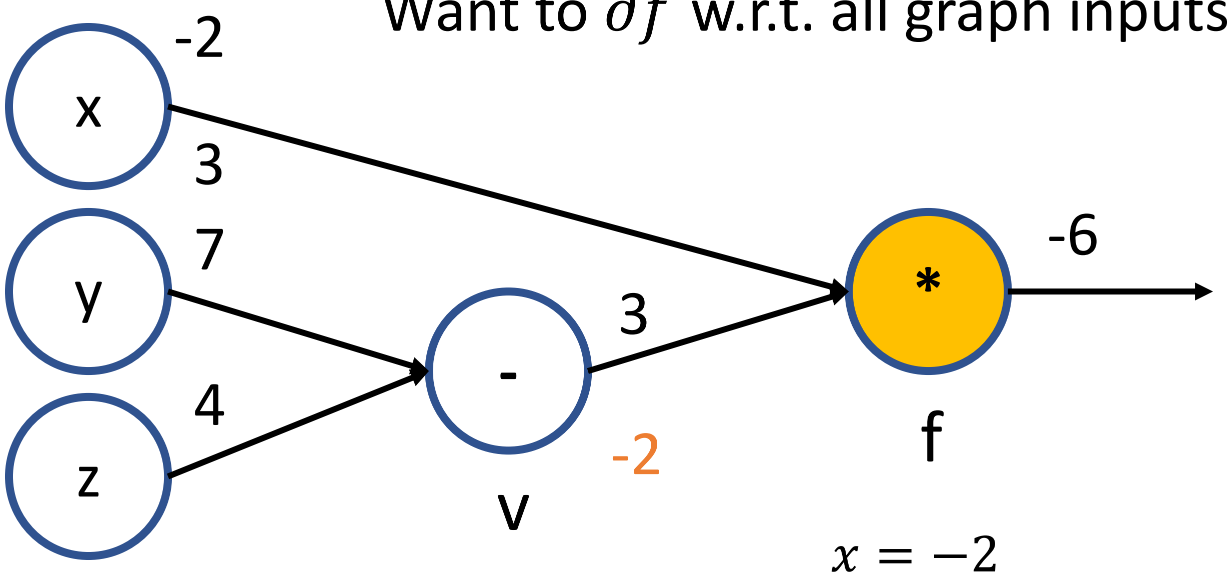
$\frac{\partial v}{\partial y} = 1 \, , \frac{\partial v}{\partial z} = -1$

$x = -2$
$y = 7$
$z = 4$
$v = 3$
$f = -6$

$\boxed{\frac{\partial f}{\partial x} = 3 \, , \frac{\partial f}{\partial v} = -2}$

Brad Quinton, Scott Chin

Want to ∂f w.r.t. all graph inputs (x, y, z)



$$f = x * v \qquad v = y - z$$

$$\frac{\partial f}{\partial x} = v, \frac{\partial f}{\partial v} = x \qquad \frac{\partial v}{\partial y} = 1, \frac{\partial v}{\partial z} = -1$$

$$\boxed{\frac{\partial f}{\partial x} = 3, \frac{\partial f}{\partial v} = -2}$$

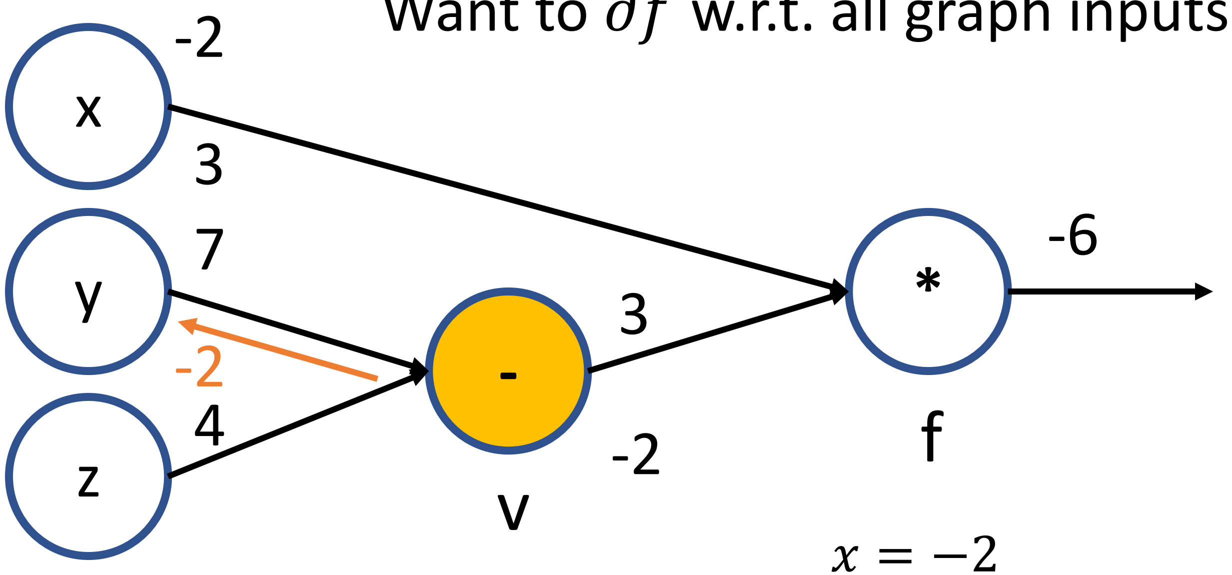
$x = -2$
$y = 7$
$z = 4$
$v = 3$
$f = -6$

Backward Propagation

1. Compute input gradient on the output node(s)
2. For each node that has a value for its output gradient, compute each input gradient using chain rule and propagate backwards

Brad Quinton, Scott Chin

Want to ∂f w.r.t. all graph inputs (x, y, z)



$f = x * v$

$\frac{\partial f}{\partial x} = v, \frac{\partial f}{\partial v} = x$

$v = y - z$

$\frac{\partial v}{\partial y} = 1, \frac{\partial v}{\partial z} = -1$

$\frac{\partial f}{\partial x} = 3, \frac{\partial f}{\partial v} = -2$

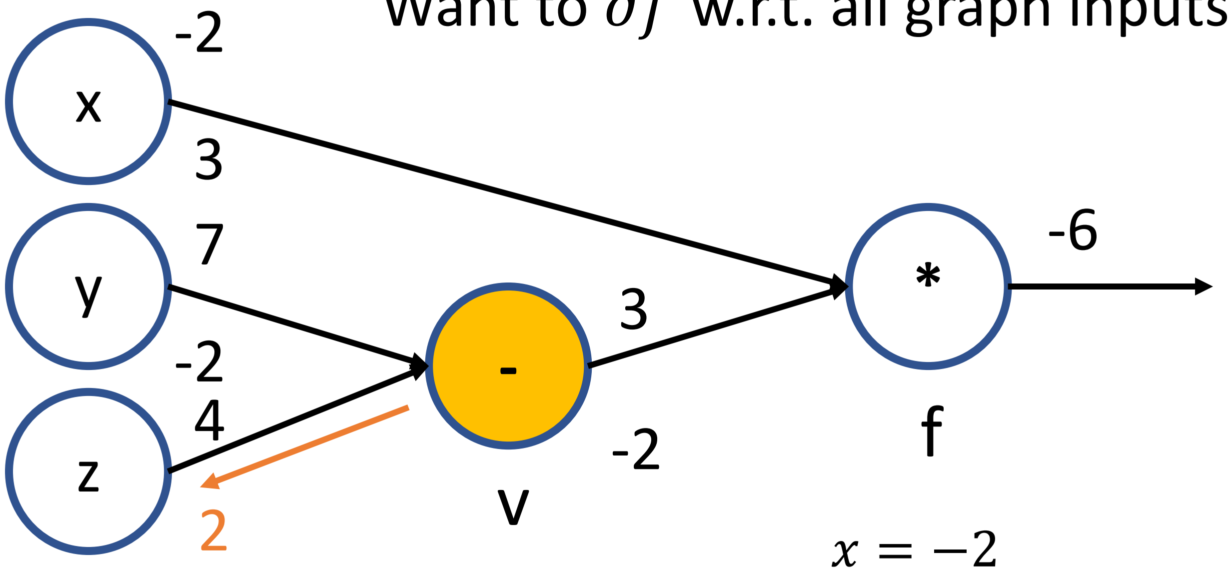
$x = -2$
$y = 7$
$z = 4$
$v = 3$
$f = -6$

$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial f}{\partial v} \cdot \frac{\partial v}{\partial y} = -2 \cdot 1 = -2}$$

Backward Propagation

1. Compute input gradient on the output node(s)
2. For each node that has a value for its output gradient, compute each input gradient using chain rule and propagate backwards

Want to ∂f w.r.t. all graph inputs (x, y, z)



$f = x * v$

$v = y - z$

$x = -2$
$y = 7$
$z = 4$
$v = 3$
$f = -6$

$\frac{\partial f}{\partial x} = v \,, \frac{\partial f}{\partial v} = x$

$\frac{\partial v}{\partial y} = 1 \,, \frac{\partial v}{\partial z} = -1$

$\frac{\partial f}{\partial x} = 3 \,, \frac{\partial f}{\partial v} = -2$

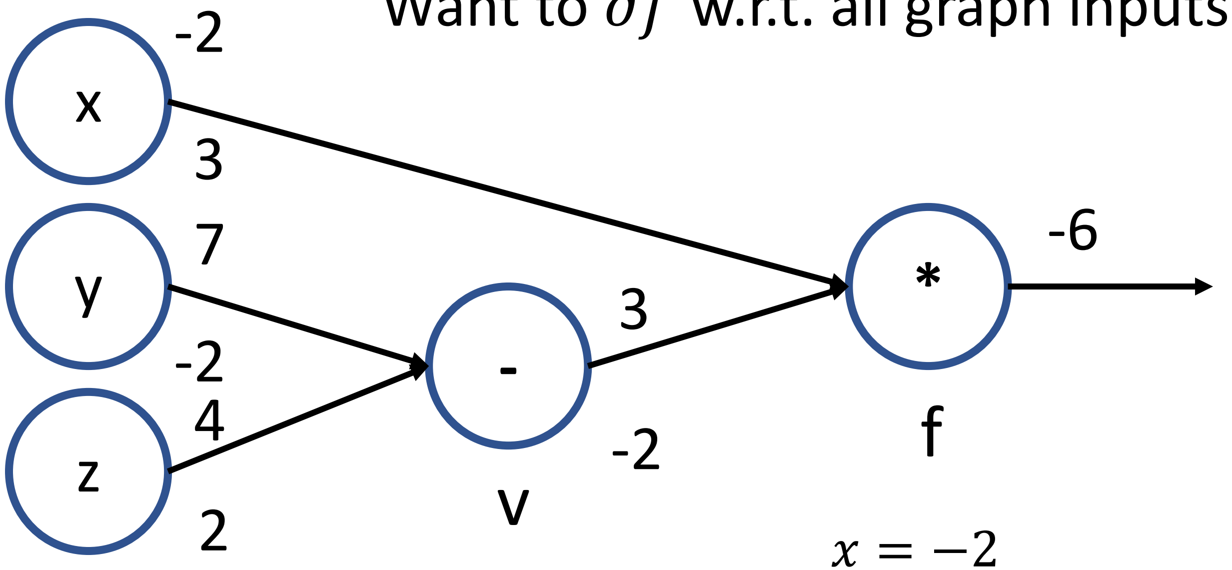
Backward Propagation

1. Compute input gradient on the output node(s)
2. For each node that has a value for its output gradient, compute each input gradient using chain rule and propagate backwards
3. Repeat until all gradients computed

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial v} \cdot \frac{\partial v}{\partial y} = -2 \cdot 1 = -2$$

$$\boxed{\frac{\partial f}{\partial z} = \frac{\partial f}{\partial v} \cdot \frac{\partial v}{\partial z} = -2 \cdot (-1) = 2}$$

Want to ∂f w.r.t. all graph inputs (x, y, z)



$f = x * v$

$\frac{\partial f}{\partial x} = v, \frac{\partial f}{\partial v} = x$

$v = y - z$

$\frac{\partial v}{\partial y} = 1, \frac{\partial v}{\partial z} = -1$

$\frac{\partial f}{\partial x} = 3, \frac{\partial f}{\partial v} = -2$

$\frac{\partial f}{\partial y} = -2, \frac{\partial f}{\partial z} = 2$

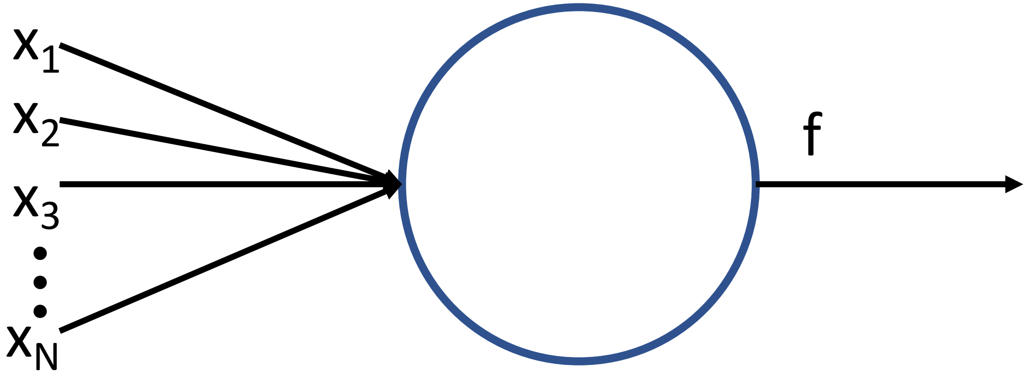
$x = -2$
$y = 7$
$z = 4$
$v = 3$
$f = -6$

Backward Propagation

1. Compute input gradient on the output node(s)
2. For each node that has a value for its output gradient, compute each input gradient using chain rule and propagate backwards
3. Repeat until all gradients computed

Brad Quinton, Scott Chin

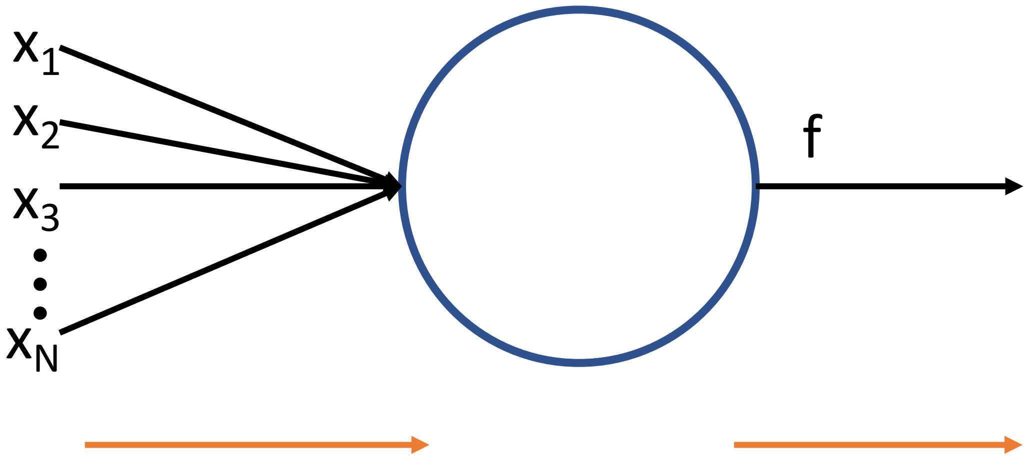
From Each Node's Perspective



Local Gradients:

$$\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_N}$$

From Each Node's Perspective



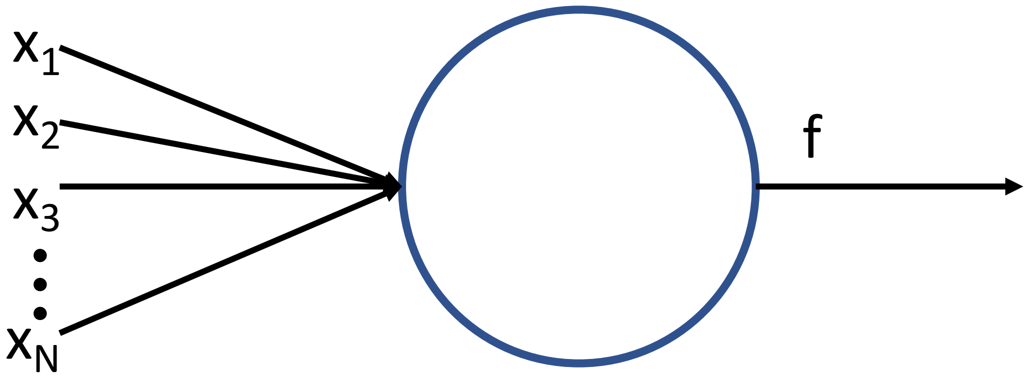
Forward Propagation

- When all input values arrive
  - Compute output value
  - Compute local gradient values

Local Gradients:

$$\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_N}$$

From Each Node's Perspective



$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial x_i}$$

$$\frac{\partial L}{\partial f}$$

Local Gradients:

$$\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_N}$$

Forward Propagation

- When all input values arrive
  - Compute output value
  - Compute local gradient values

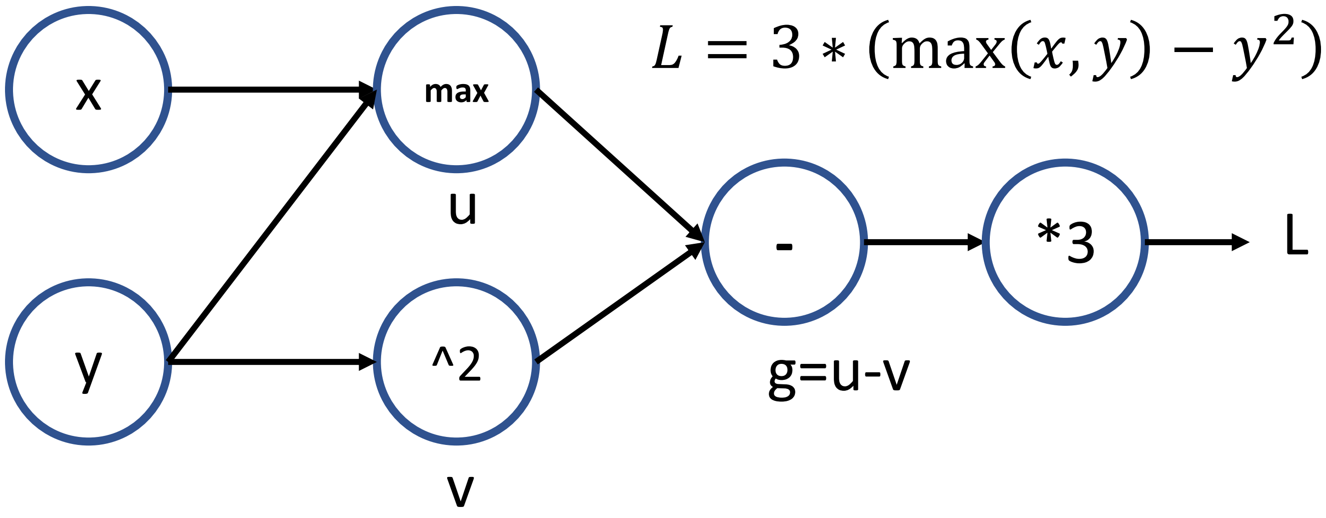
Backward Propagation

- When upstream gradient arrives on output
  - Using Chain Rule, compute downstream gradient on inputs

Key Take-Aways for Compute Graph Backprop

- Backprop is a very local process
- Computations for both forward and backward propagation can be performed on a per-node basis as values arrive
  - On inputs during forward prop
  - On output during backward prop
- Local gradients can be computed during forward propagation!
- Use chain rule to “flow” gradient back through a node
- If it helps, you can think of it as a circuit, message passing, pipes, etc.

Another Example – Take 5 min to try solving

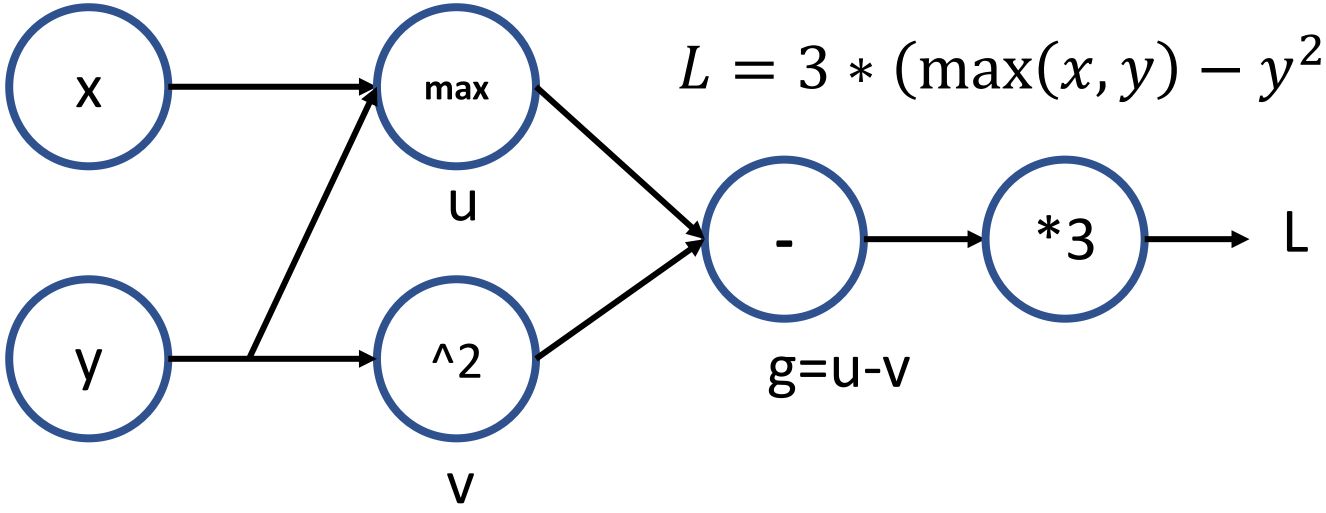


$L = 3 * (\max(x, y) - y^2)$

x=2, y=3

Answer: $\frac{\partial L}{\partial x} = 0\,, \frac{\partial L}{\partial y} = -15$

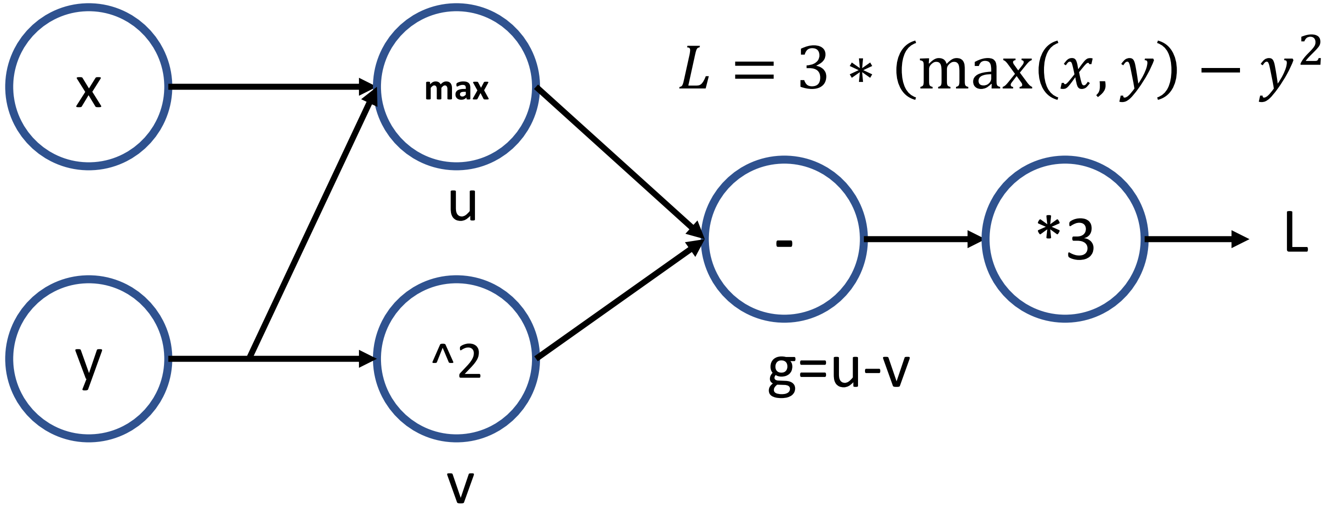
1. Compute the output of each node (forward prop)
2. Compute the local gradient of each node
3. Backprop: Compute input gradient on the output node(s)
4. Chain gradient back by multiplying each local gradient with node's output gradient



$L = 3 * (\max(x, y) - y^2)$

x=2, y=3

Answer: $\frac{\partial L}{\partial x} = 0 , \frac{\partial L}{\partial y} = -15$



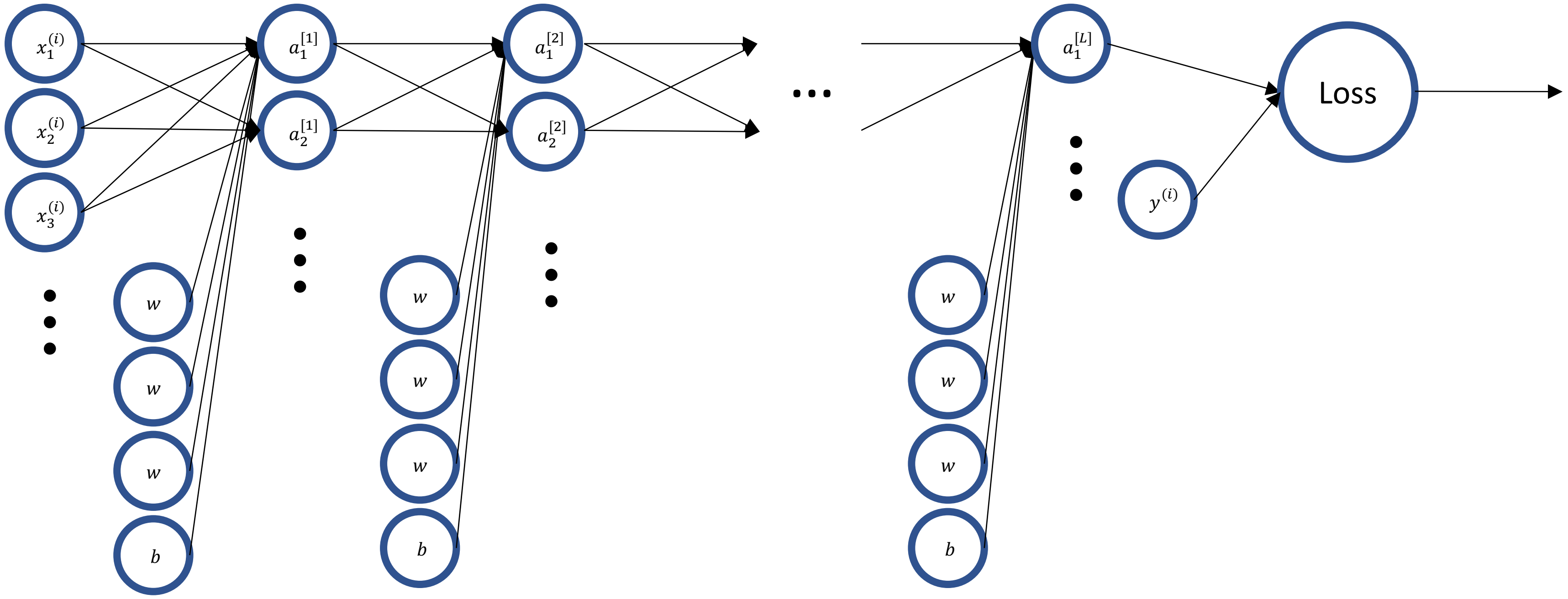
$$L = 3 * (\max(x, y) - y^2)$$

x=2, y=3

Answer: $\frac{\partial L}{\partial x} = 0 \, , \frac{\partial L}{\partial y} = -15$

Working Towards

- Model neural network as a compute graph
- Model loss/cost function as another operation at the end
- Backprop on compute graph to find ∂L w.r.t. each parameter



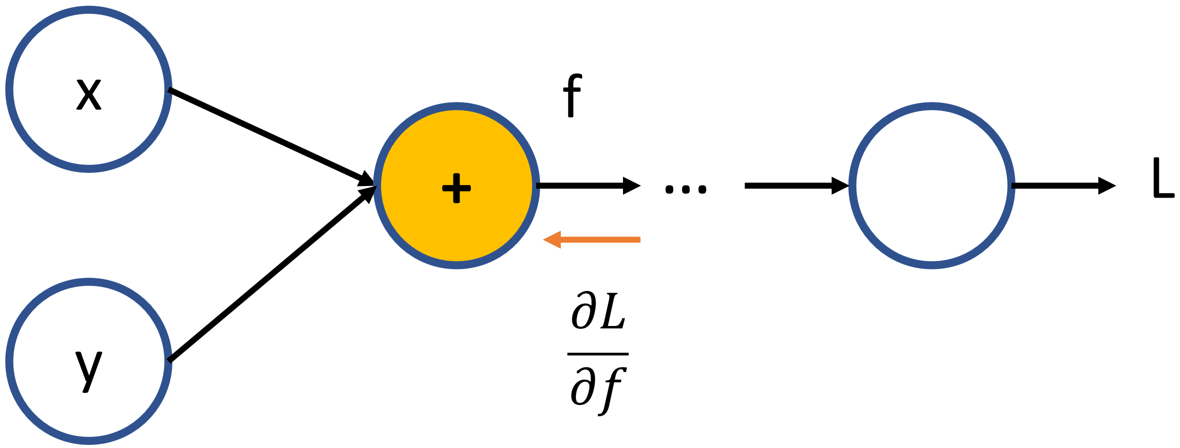
Brad Quinton, Scott Chin

Compute Graph Nodes

- Let’s review the common operation nodes we’ve seen so far and analyze how gradients flow back through them
  - Add
  - Subtract
  - Multiply
  - Equality
  - Branch
  - Max()
  - Sigmoid()
  - Tanh()

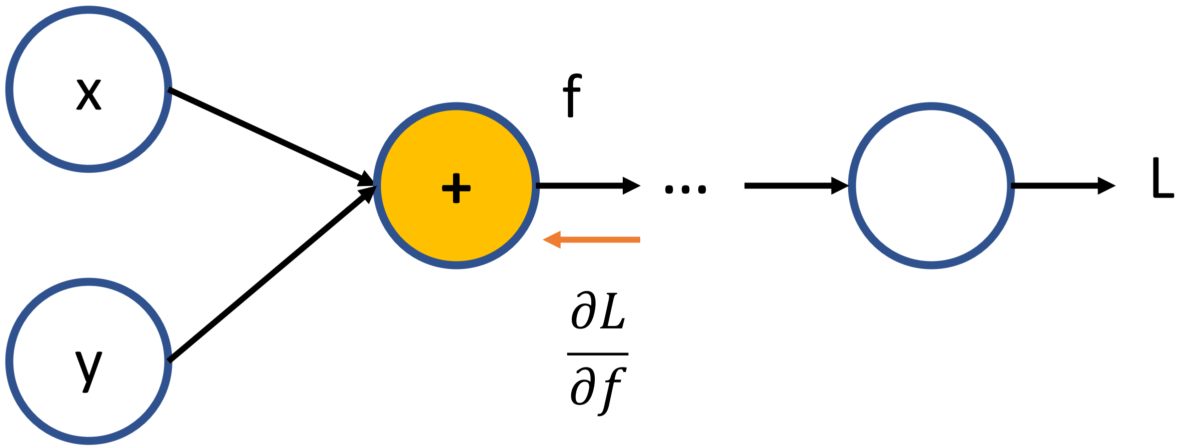
Brad Quinton, Scott Chin

Addition



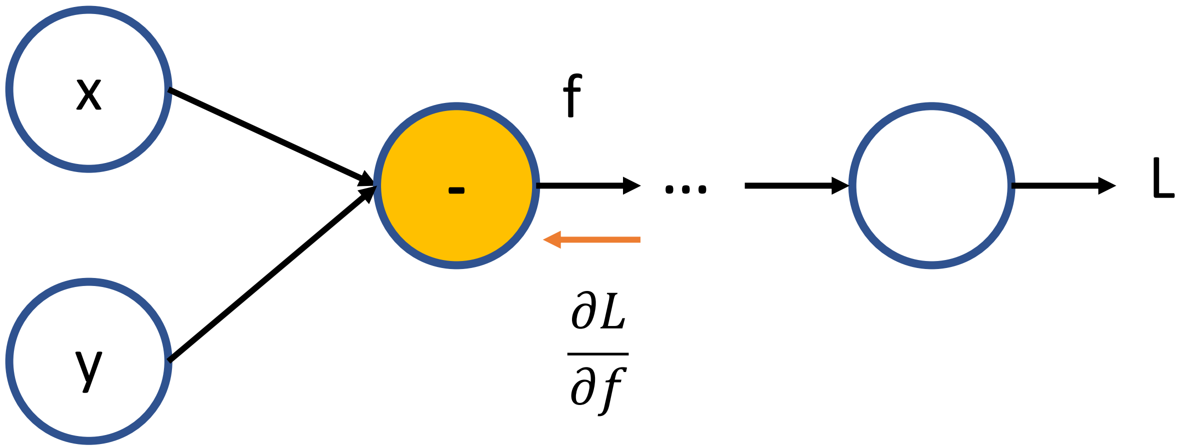
Brad Quinton, Scott Chin

Addition – Annotated



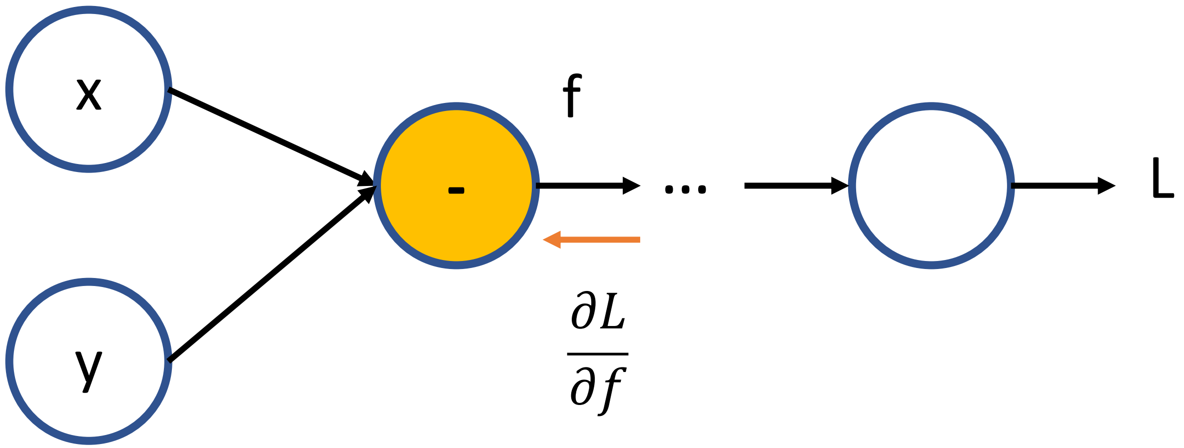
- Upstream gradient is distributed to all inputs
- Intuition: A change on any input independently changes the output

Subtraction



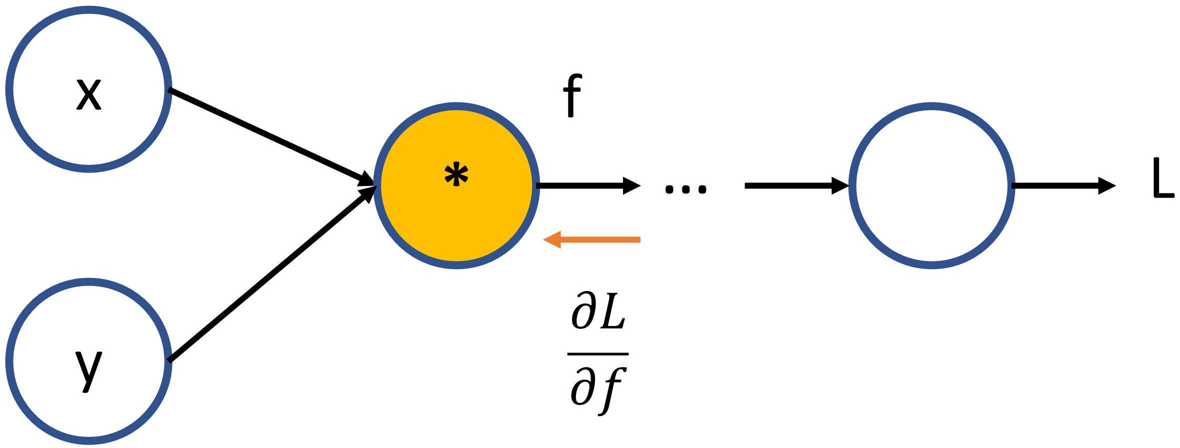
Brad Quinton, Scott Chin

Subtraction – Annotated

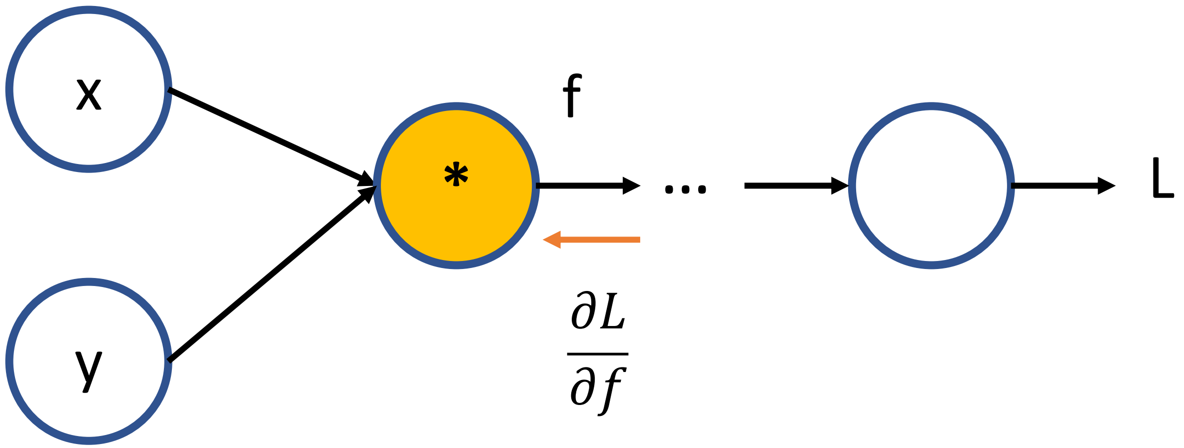


- Upstream gradient passed onto variable being subtracted from
- Negative of upstream gradient passed onto variable being subtracted

Multiplication

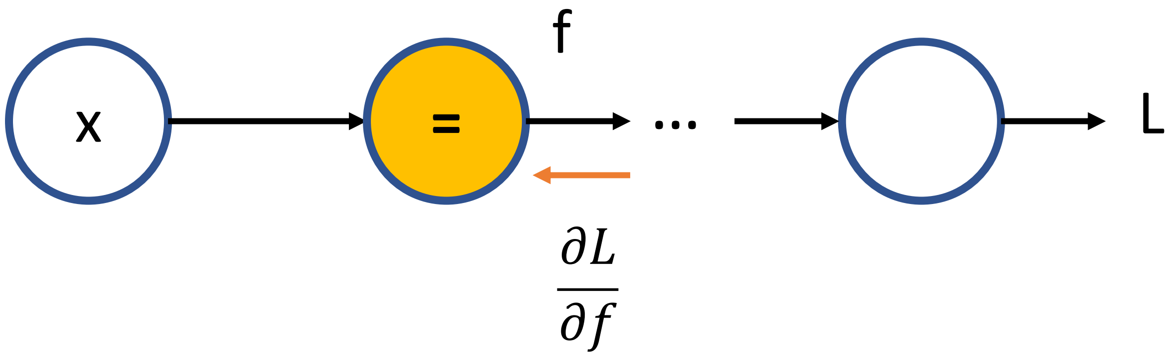


Multiplication – Annotated

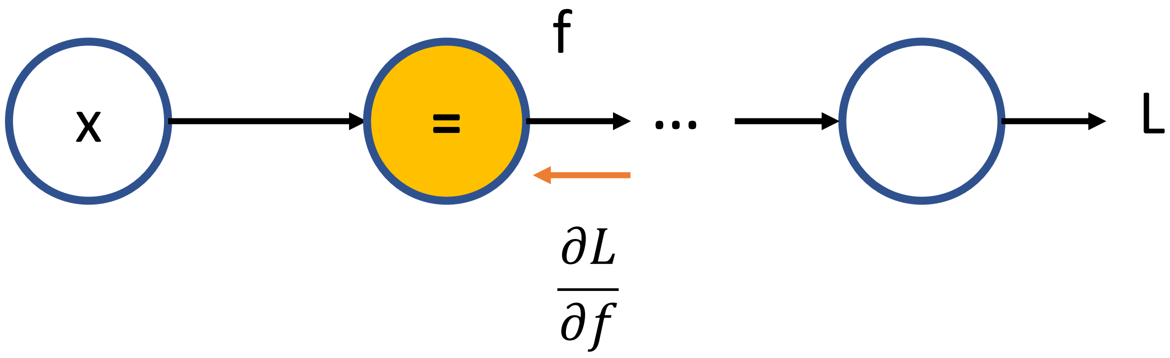


- Upstream multiplied with all other input values
- Intuition: A change on an input is scaled by the value of the other inputs to affect a change in the output

Equality (Linear)



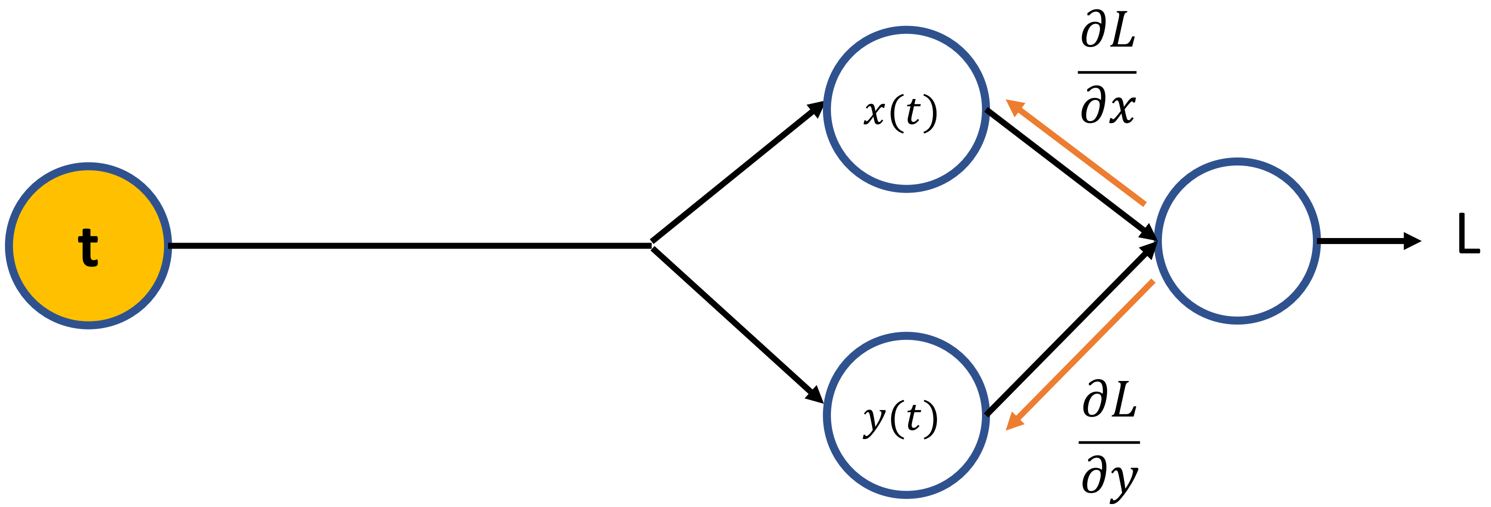
Equality (Linear) – Annotated



- Pass Through

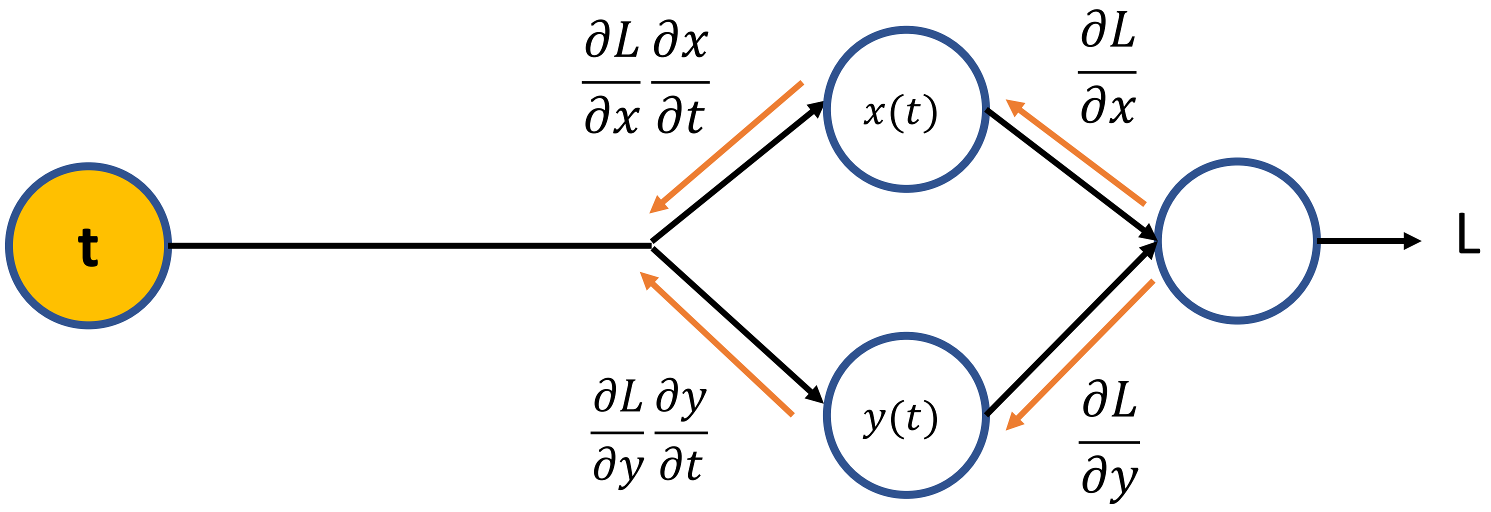
Brad Quinton, Scott Chin

Branch



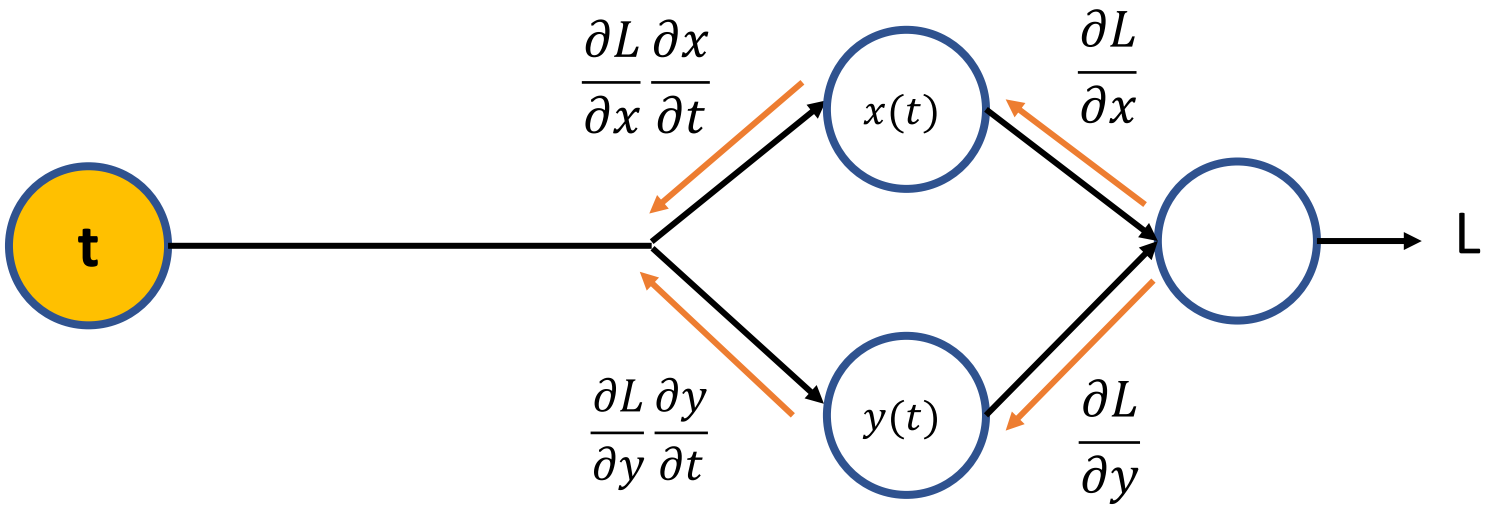
Brad Quinton, Scott Chin

Branch



- What do you think happens here?
- What is $\frac{\partial L}{\partial t}$?

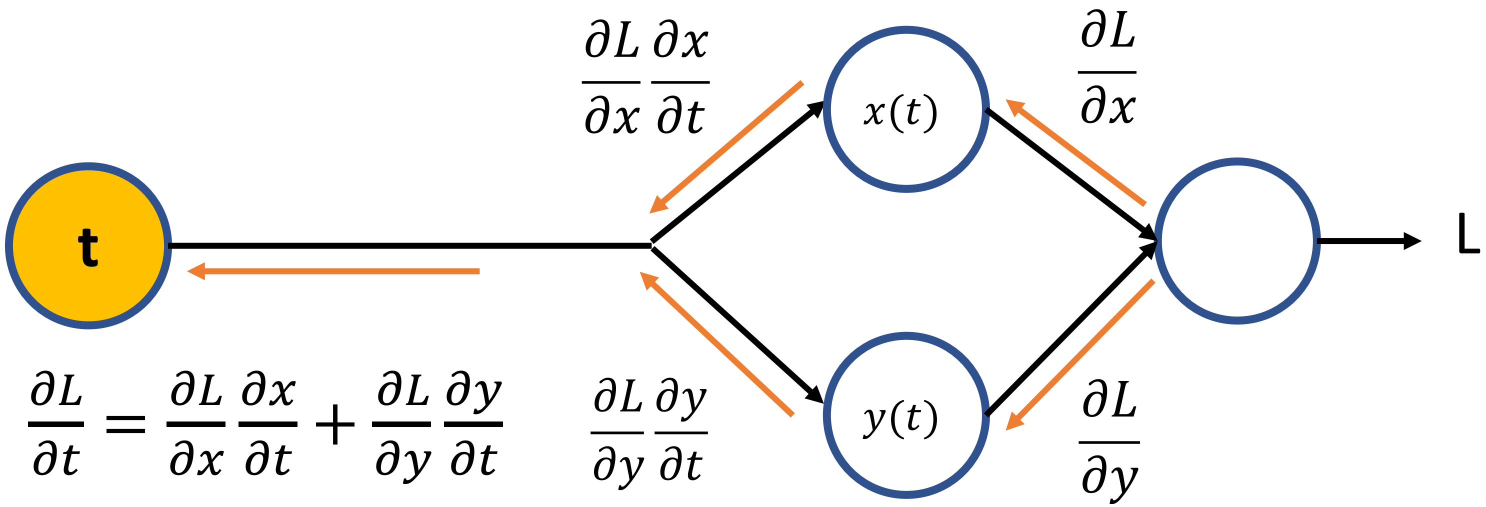
Branch



- Sum gradients from branches

Brad Quinton, Scott Chin

Branch

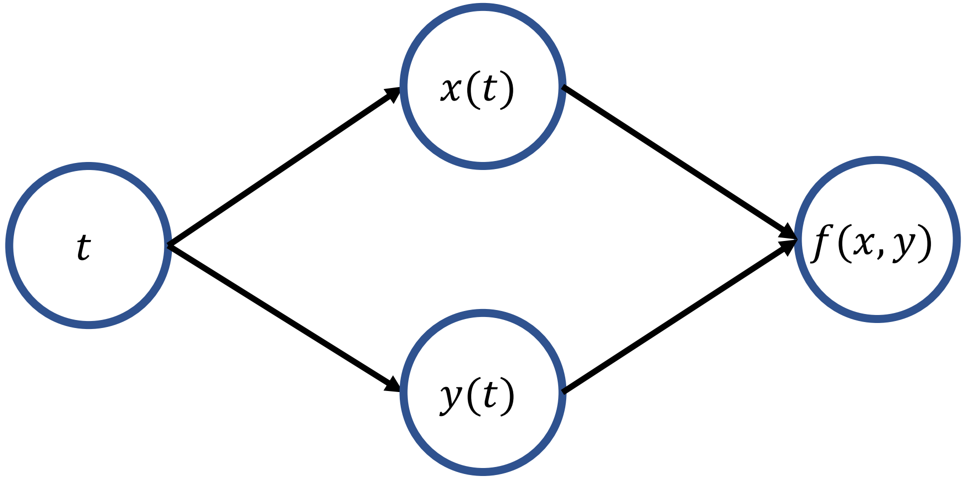


$$\frac{\partial L}{\partial t} = \frac{\partial L}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial L}{\partial y}\frac{\partial y}{\partial t}$$

- Sum gradients from branches
- Mathematically, this is the Multivariable Chain Rule

Multivariable Chain Rule

- Consider a multivariable function $f(x(t), y(t))$

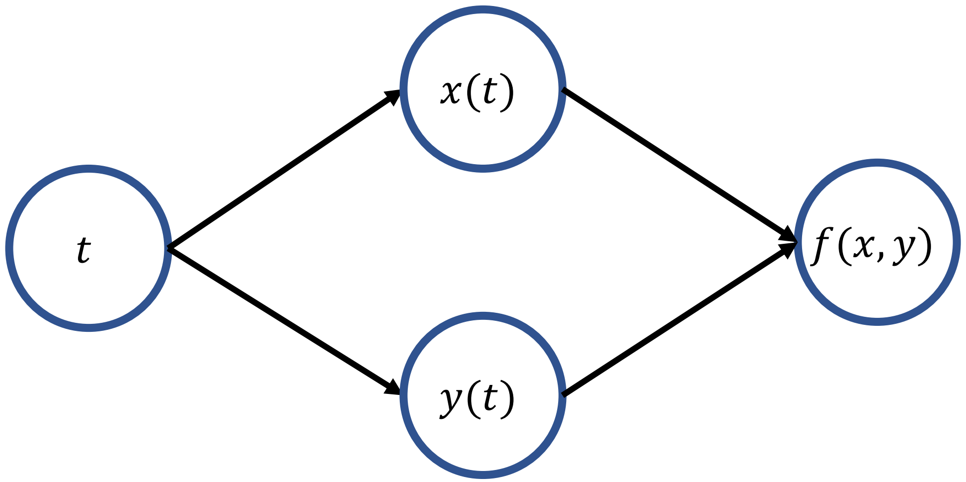


Brad Quinton, Scott Chin

Multivariable Chain Rule

- Consider a multivariable function $f(x(t), y(t))$

$$\frac{d}{dt} f\big(x(t), y(t)\big) = \frac{\partial f}{\partial t}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t}$$



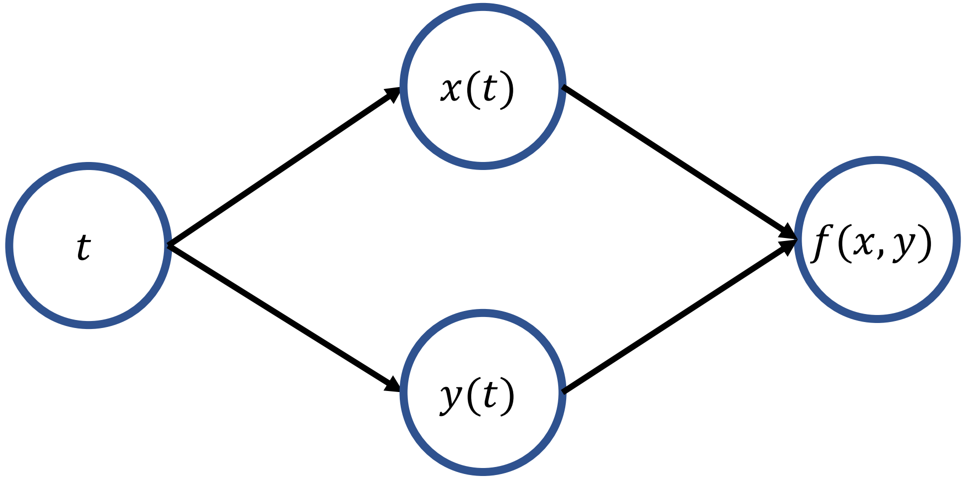
Brad Quinton, Scott Chin

Multivariable Chain Rule

- Consider a multivariable function $f(x(t), y(t))$

$$\frac{d}{dt} f\big(x(t), y(t)\big) = \underbrace{\frac{\partial f}{\partial t}\frac{\partial x}{\partial t}} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t}$$

Change in f due to influence of t on x

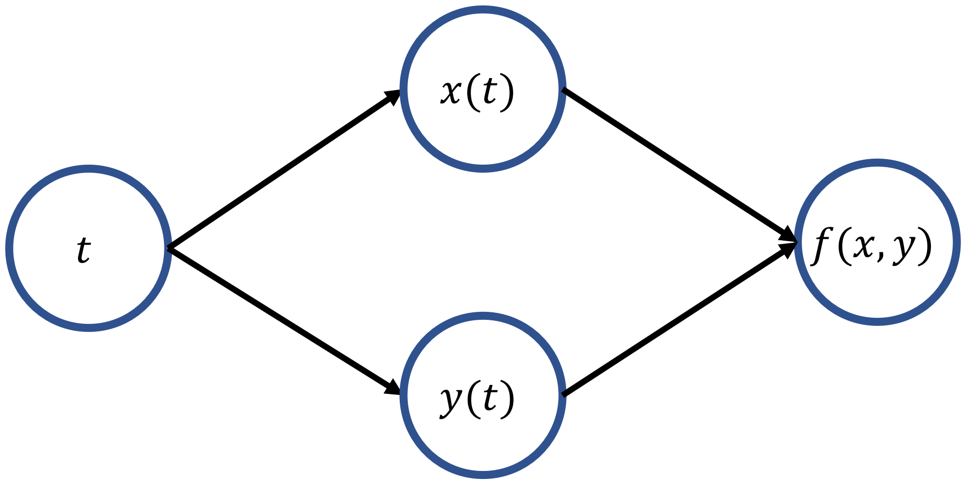


Brad Quinton, Scott Chin

Multivariable Chain Rule

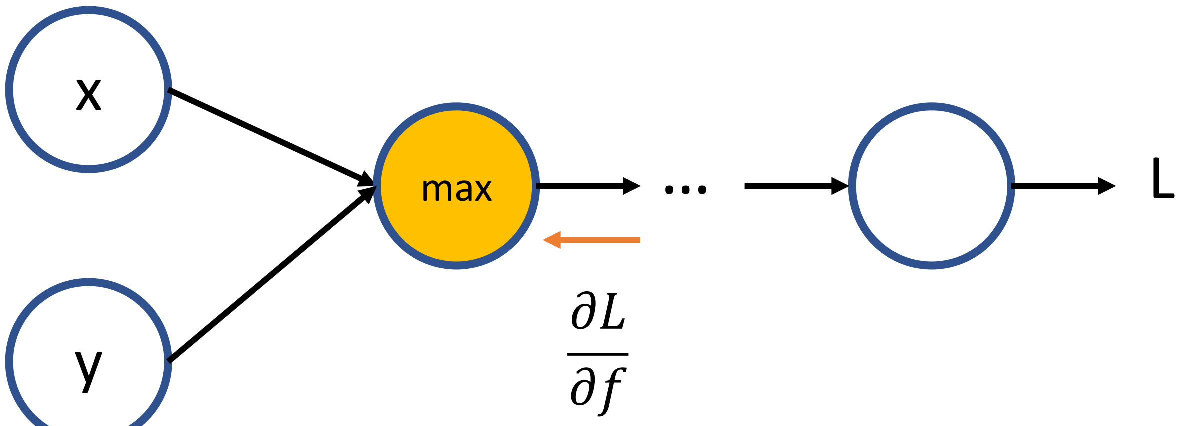
- Consider a multivariable function $f(x(t), y(t))$

$$\frac{d}{dt} f\big(x(t), y(t)\big) = \underbrace{\frac{\partial f}{\partial t}\frac{\partial x}{\partial t}}_{\text{Change in } f \text{ due to influence of } t \text{ on } x} + \underbrace{\frac{\partial f}{\partial y}\frac{\partial y}{\partial t}}_{\text{Change in } f \text{ due to influence of } t \text{ on } y}$$



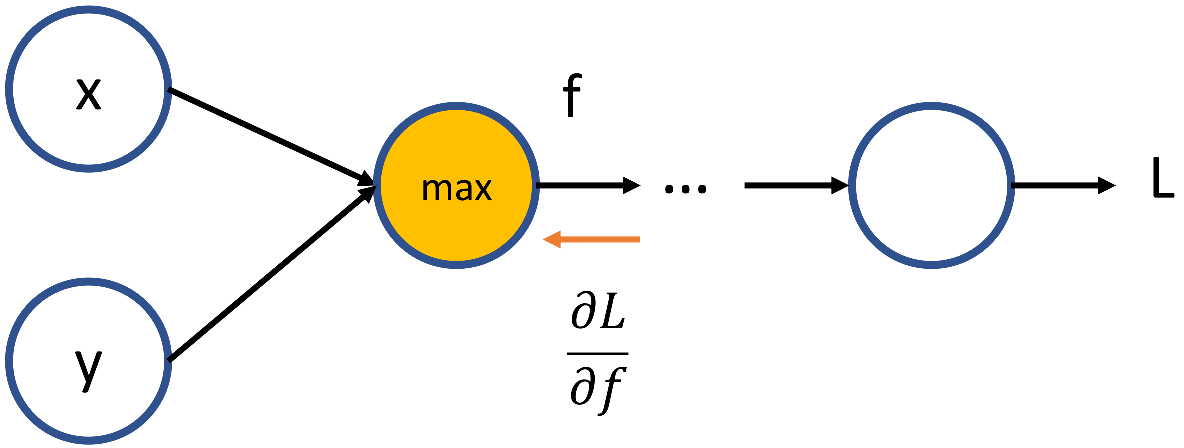
Brad Quinton, Scott Chin

Max



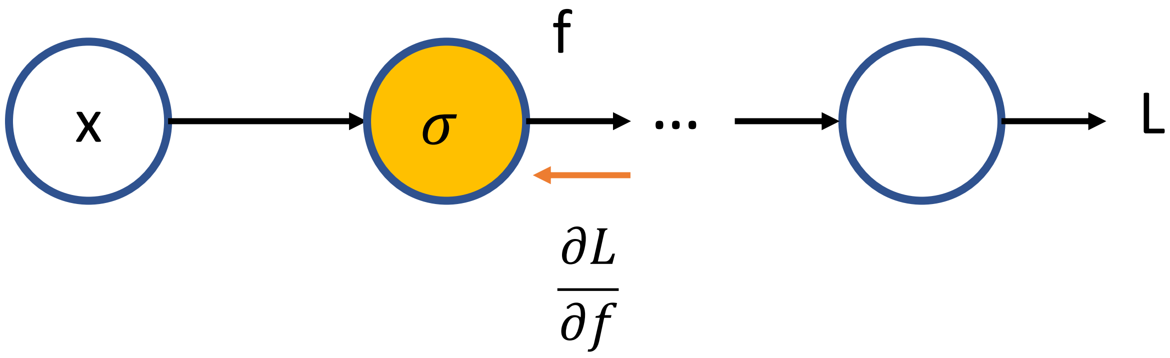
Brad Quinton, Scott Chin

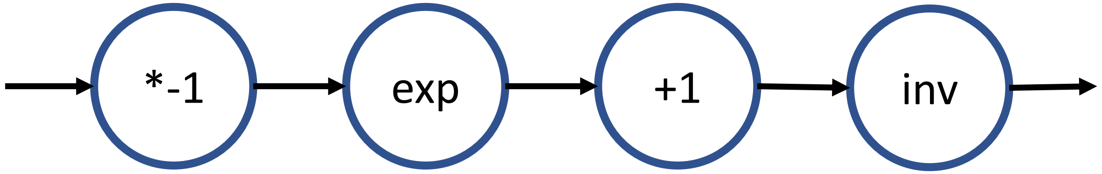
Max



- Upstream gradient is routed to larger variable
- Intuition: Only one input can affect the output at any time

Sigmoid



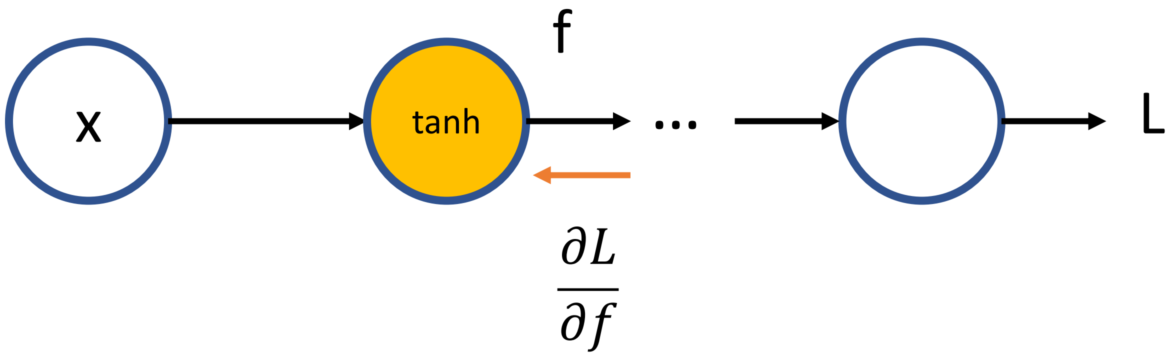


$$f(x) = \frac{1}{1 + e^{-x}}$$

$$\frac{\partial f}{\partial x} = f \cdot (1 - f)$$

Brad Quinton, Scott Chin

Tanh

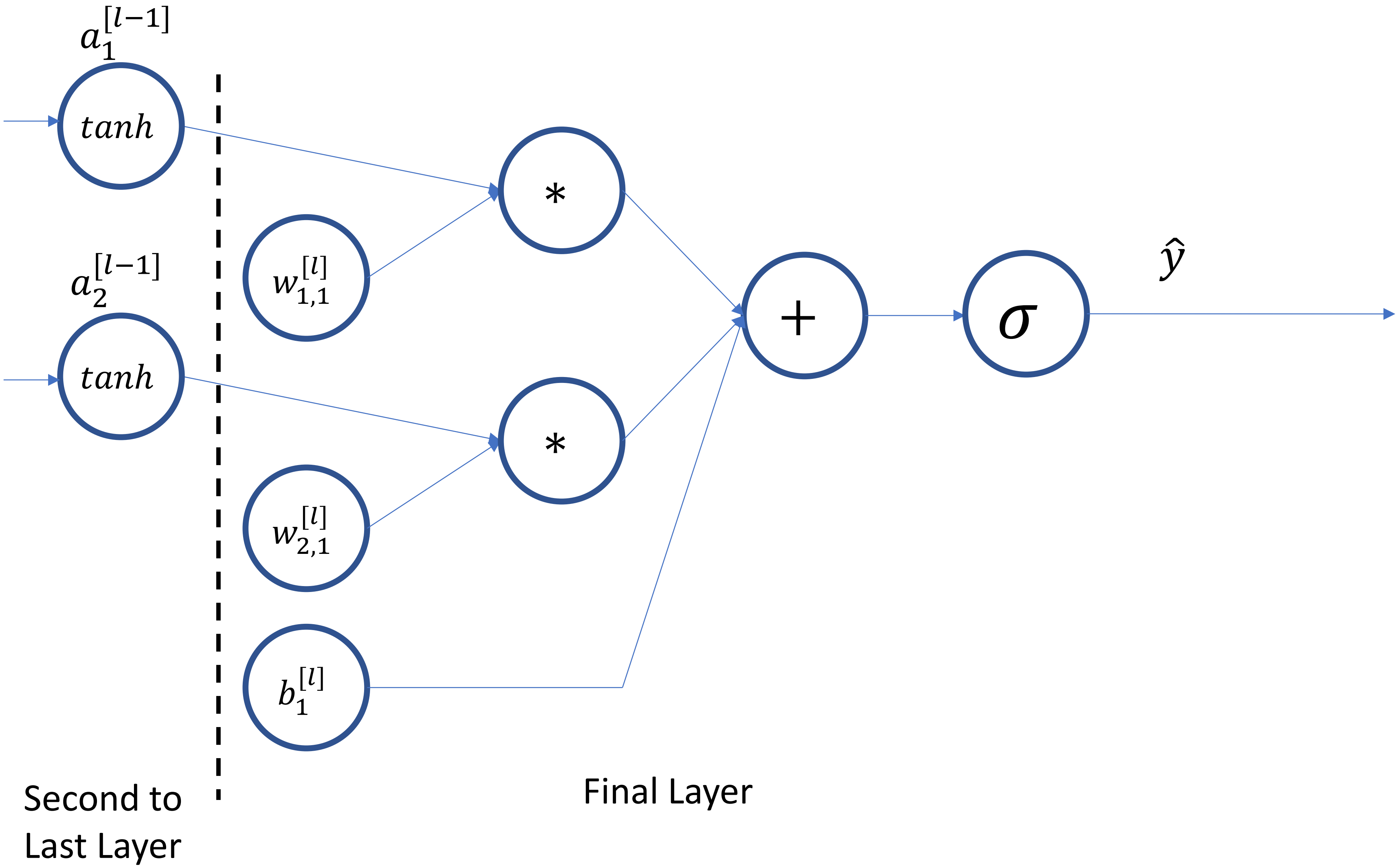


$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\frac{\partial f}{\partial x} = 1 - f^{2}$$

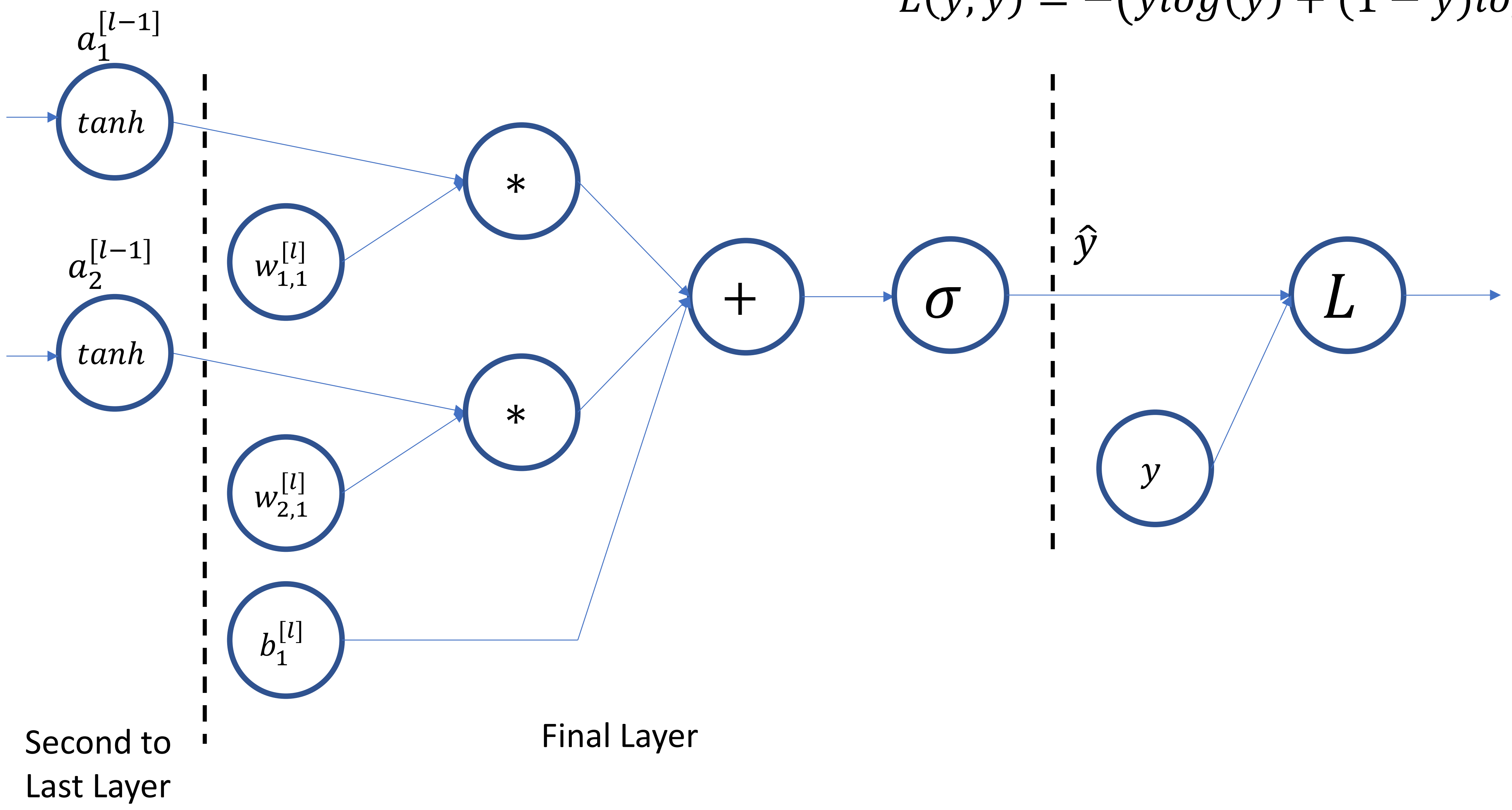
Brad Quinton, Scott Chin

Example with
Training Neural Network



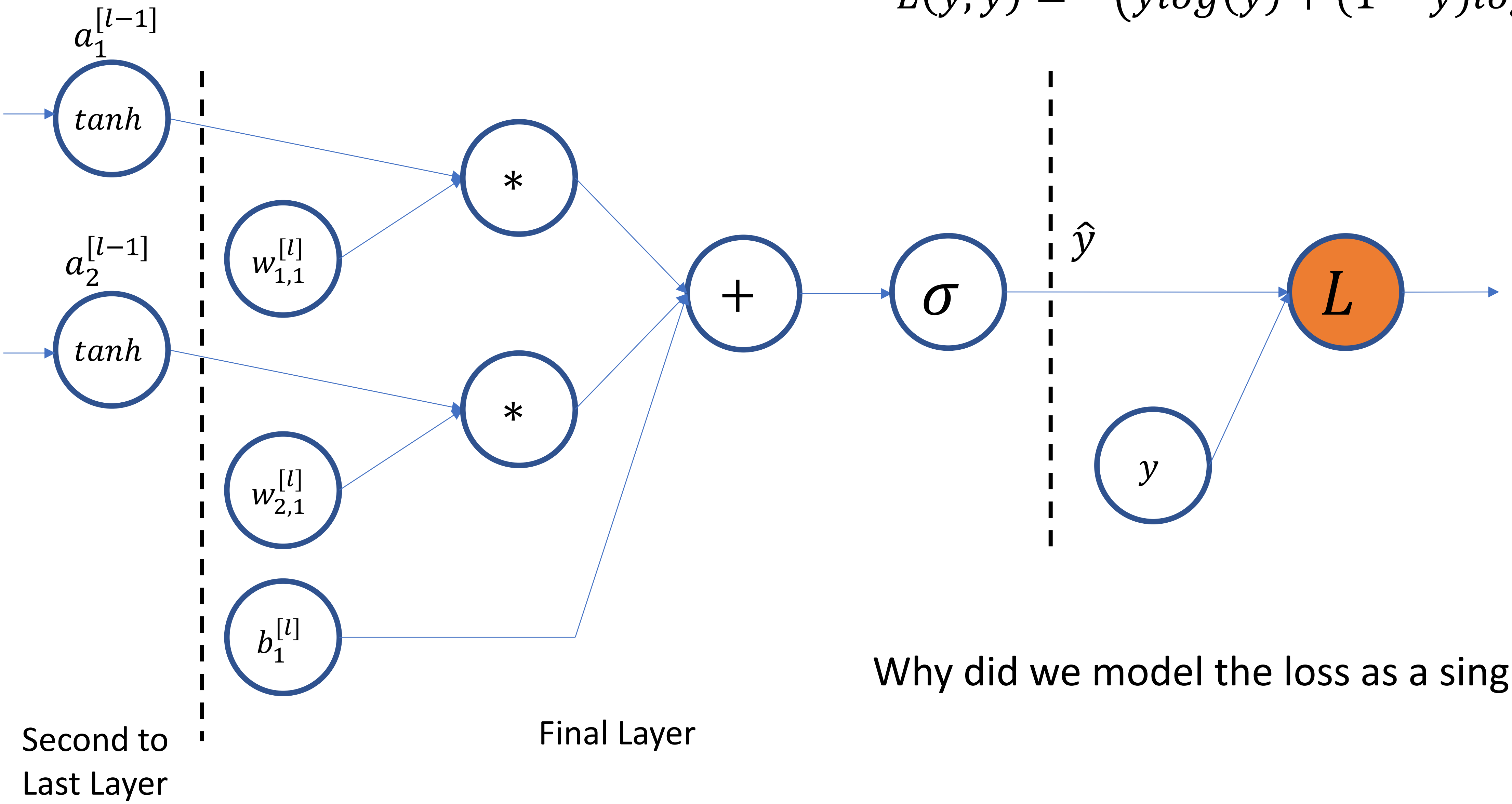
Brad Quinton, Scott Chin

$$L(\hat{y}, y) = -\left(y\log(\hat{y}) + (1-y)\log(1-\hat{y})\right)$$



Brad Quinton, Scott Chin

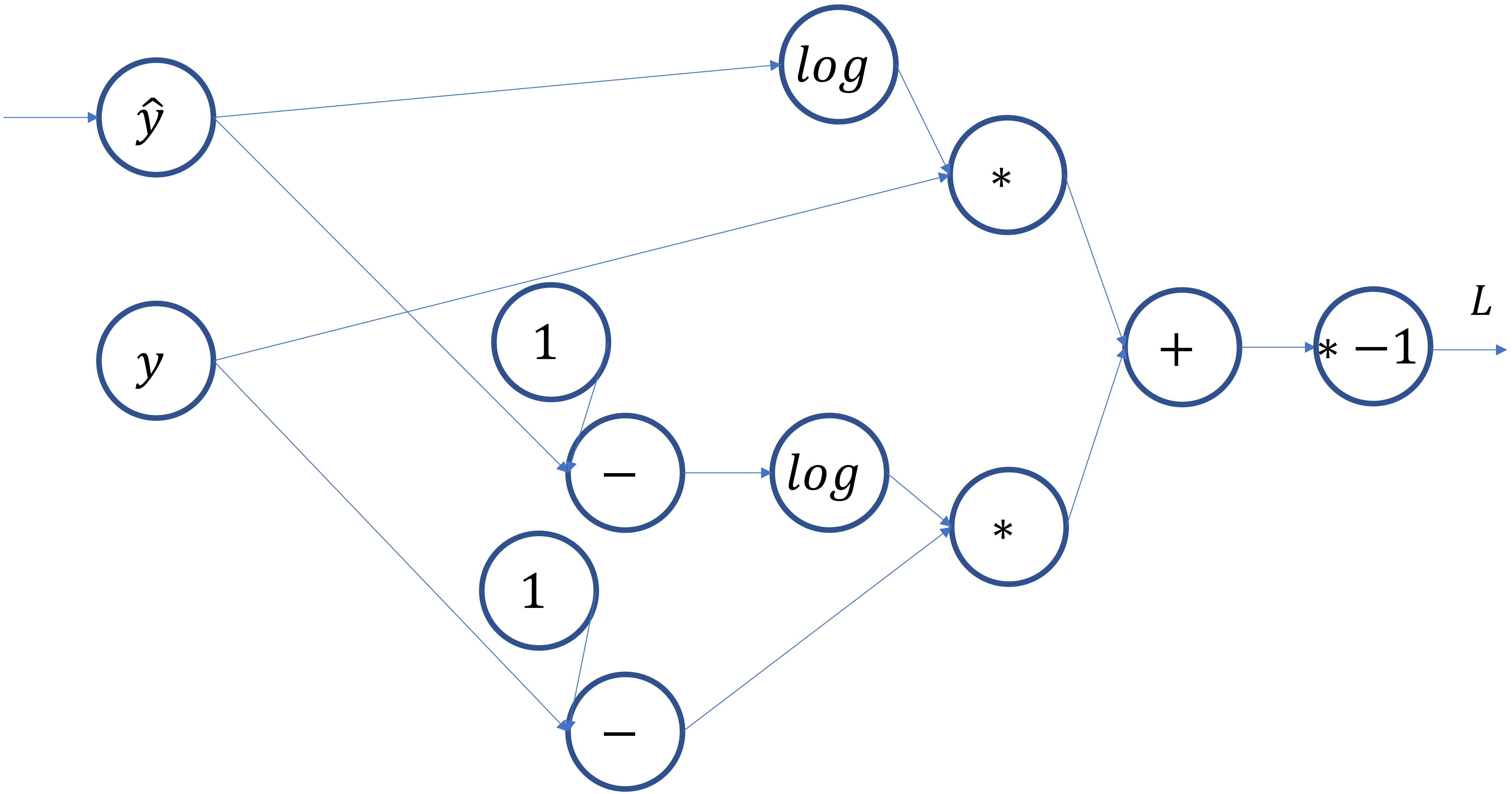
$$L(\hat{y}, y) = -\big(y log(\hat{y}) + (1-y) log(1-\hat{y})\big)$$



Why did we model the loss as a single node?

Brad Quinton, Scott Chin

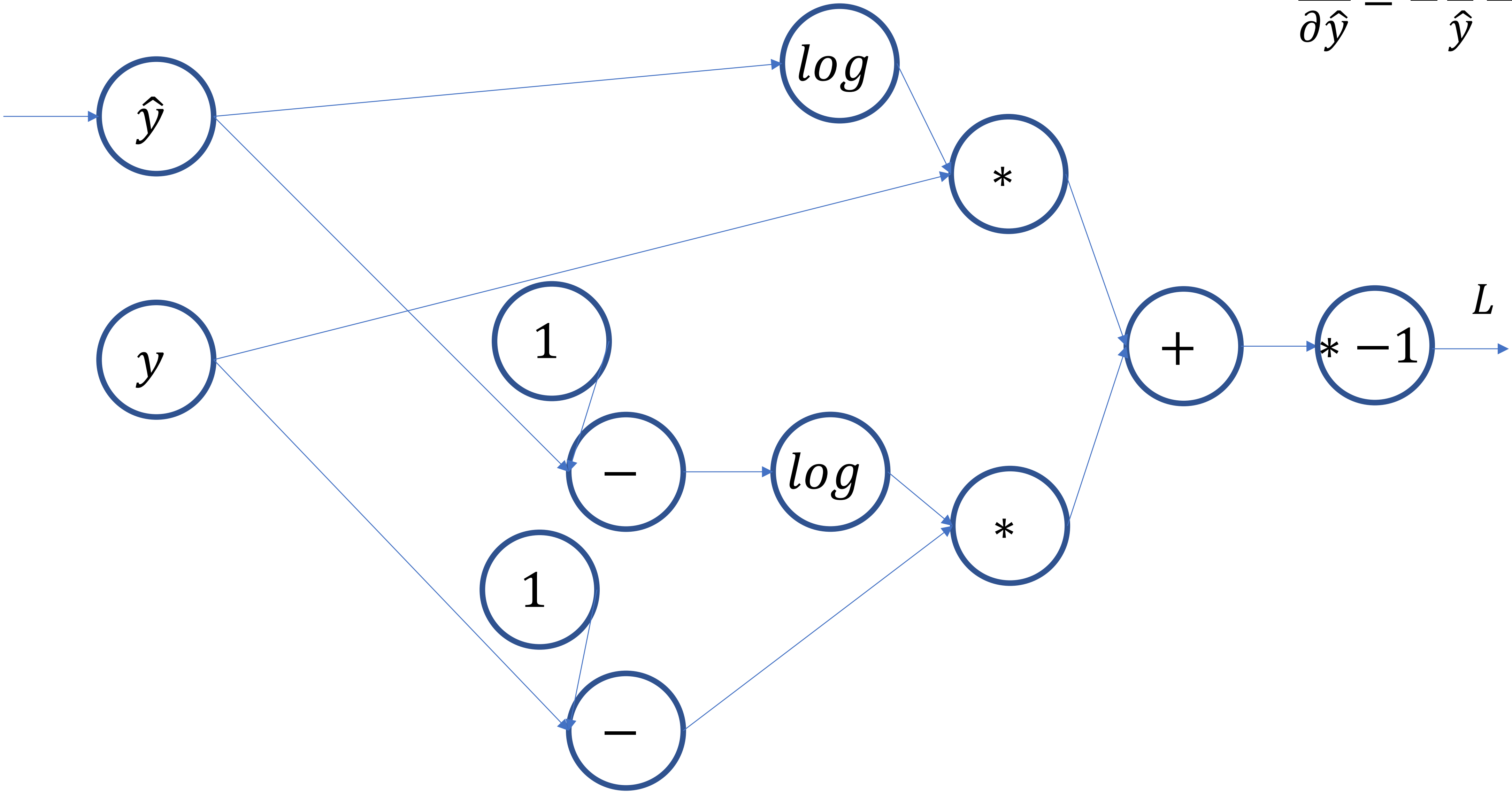
We could have modelled L more explicitly: $L(\hat{y}, y) = -\big(y log(\hat{y}) + (1-y) log(1-\hat{y})\big)$



Brad Quinton, Scott Chin

We could have modelled L more explicitly: $L(\hat{y}, y) = -\big(y log(\hat{y}) + (1-y) log(1-\hat{y})\big)$

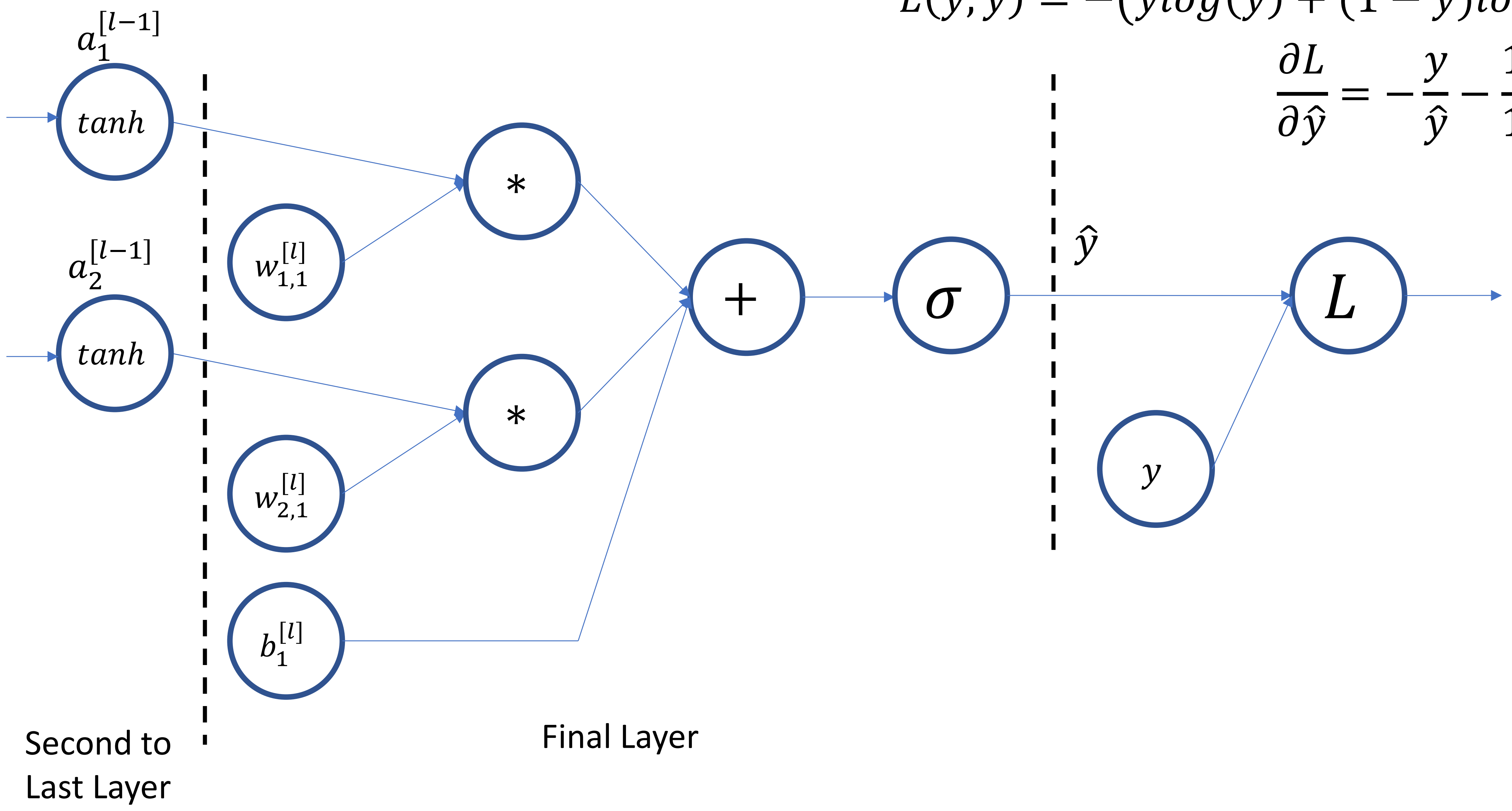
$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}$$



Brad Quinton, Scott Chin

$$L(\hat{y}, y) = -\left(y\log(\hat{y}) + (1-y)\log(1-\hat{y})\right)$$

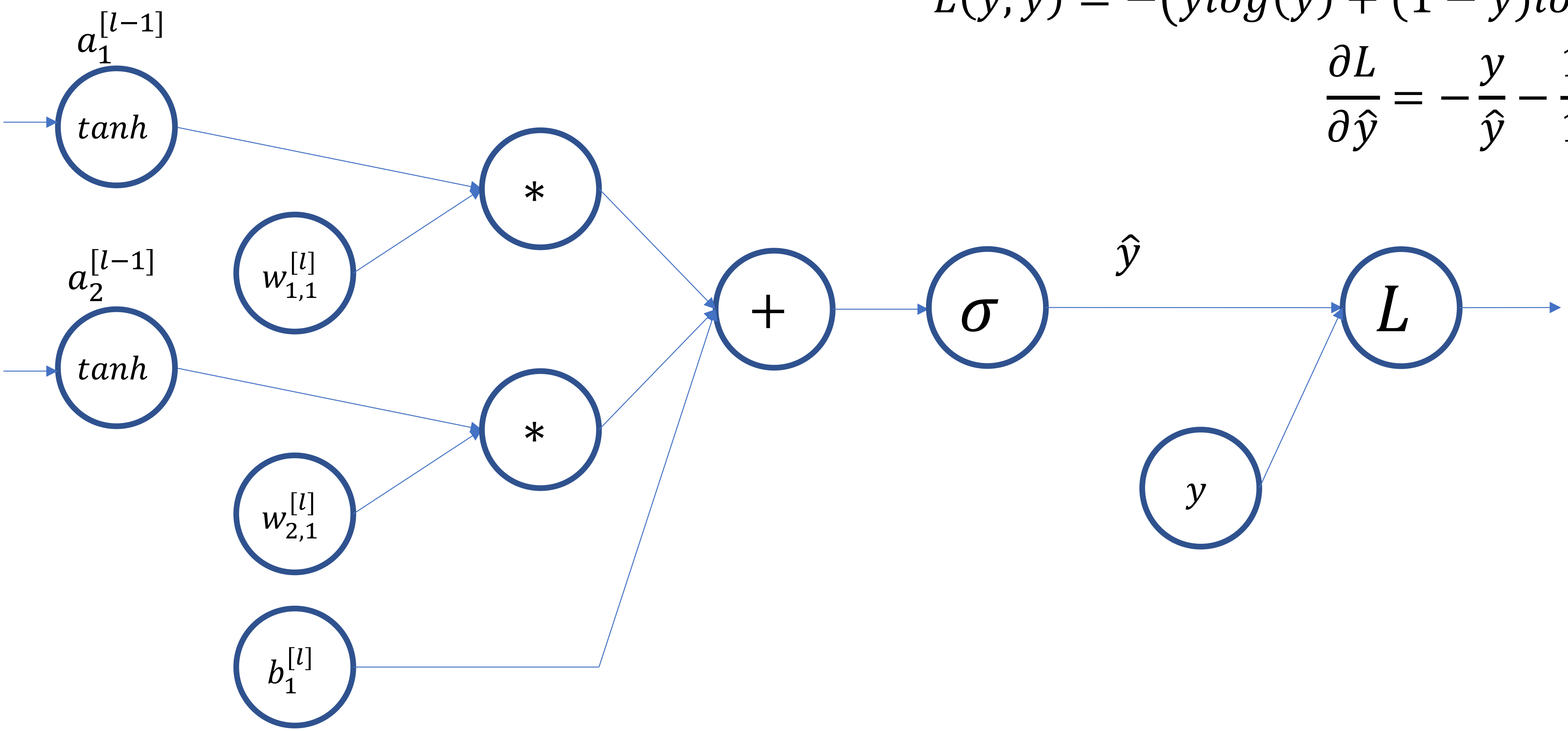
$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}$$



Brad Quinton, Scott Chin

$$L(\hat{y},y) = -\left(y\log(\hat{y}) + (1-y)\log(1-\hat{y})\right)$$

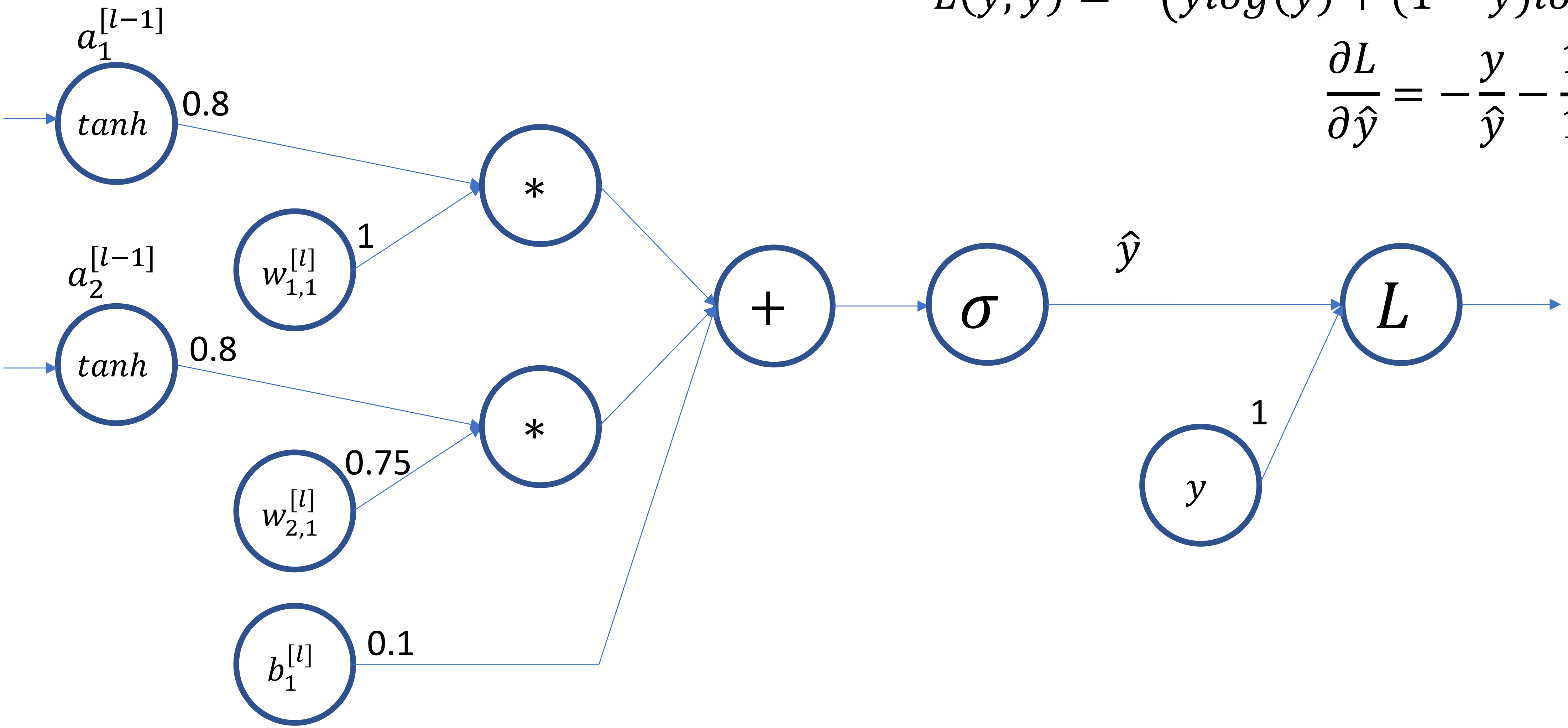
$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}$$



Brad Quinton, Scott Chin

$$L(\hat{y}, y) = -\big(y\log(\hat{y}) + (1-y)\log(1-\hat{y})\big)$$

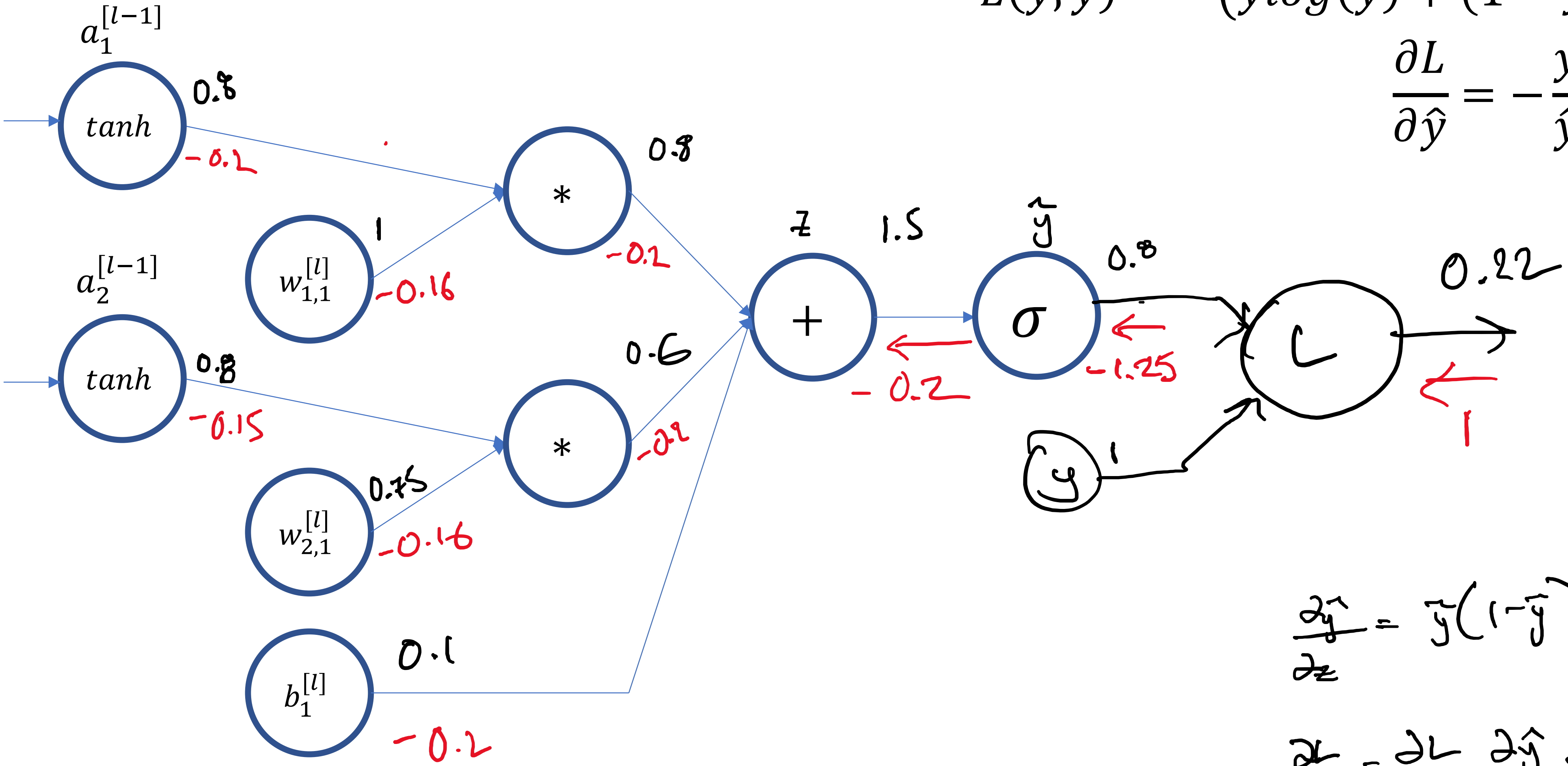
$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}$$



Brad Quinton, Scott Chin

$$L(\hat{y}, y) = -\left(y\log(\hat{y}) + (1-y)\log(1-\hat{y})\right)$$

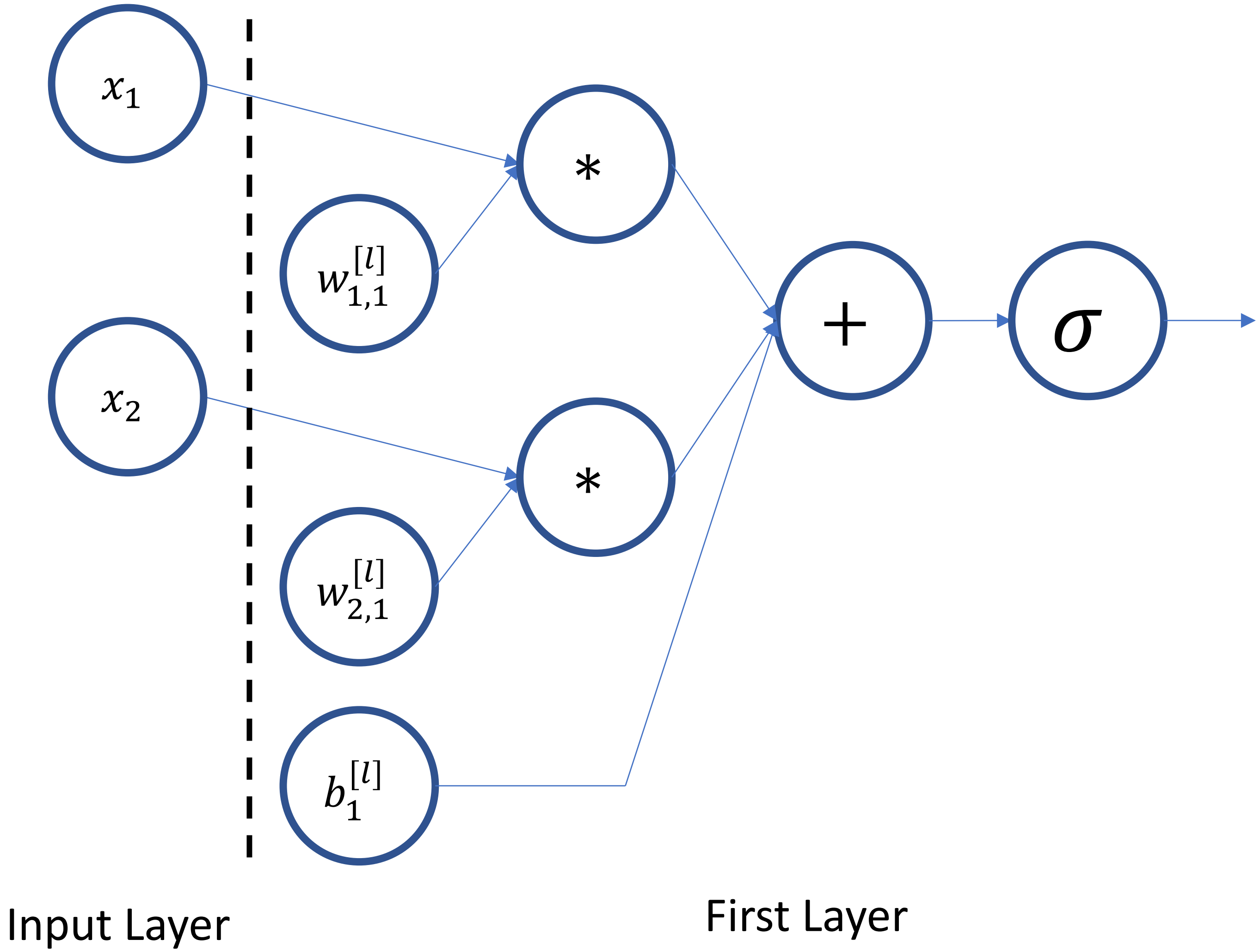
$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}$$



$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y}) = 0.16$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} = -1.25 \cdot 0.16 = 0.2$$

Backprop at Input Layer



- No need to compute $\frac{\partial L}{\partial x_i}$ since we aren't interested in how to change the input data to minimize Loss
- But later, we will look at how this can help visualize what the network has learned

Learning Objectives

- Become able to apply backpropagation on any arbitrarily complex computation graph using a systematic numerical approach
- Become familiar with backprop through common mathematical operations