More Backprop and Activation Functions

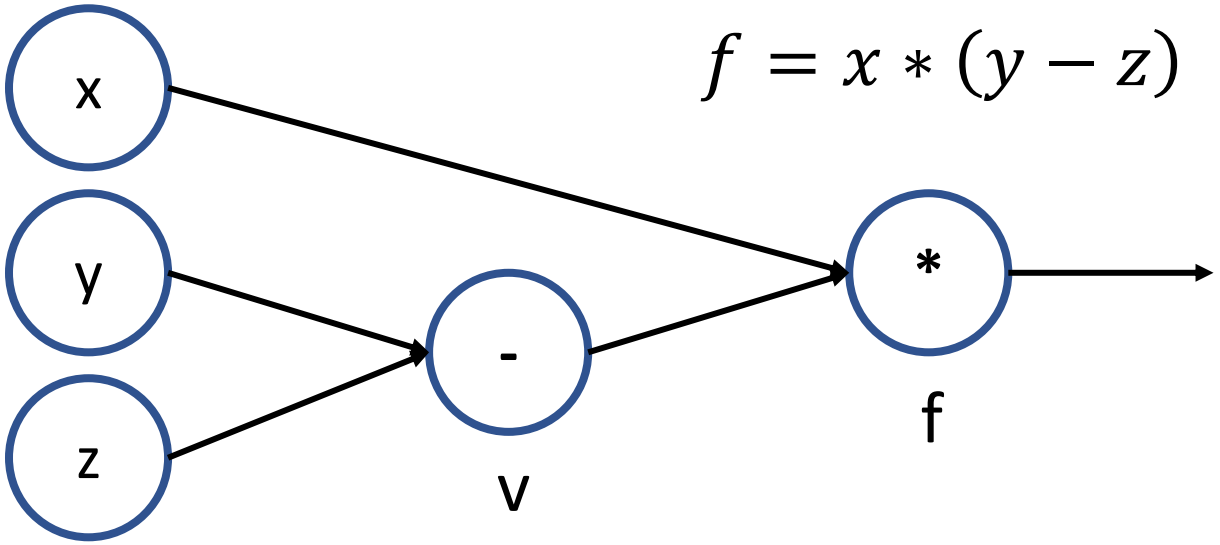
Deep Learning

Brad Quinton, Scott Chin

Learning Objectives

- Gain more experience with how backprop works on computation graphs through examples
  Be able to analyze why certain activation functions are better than other in the context of training and backpropagation

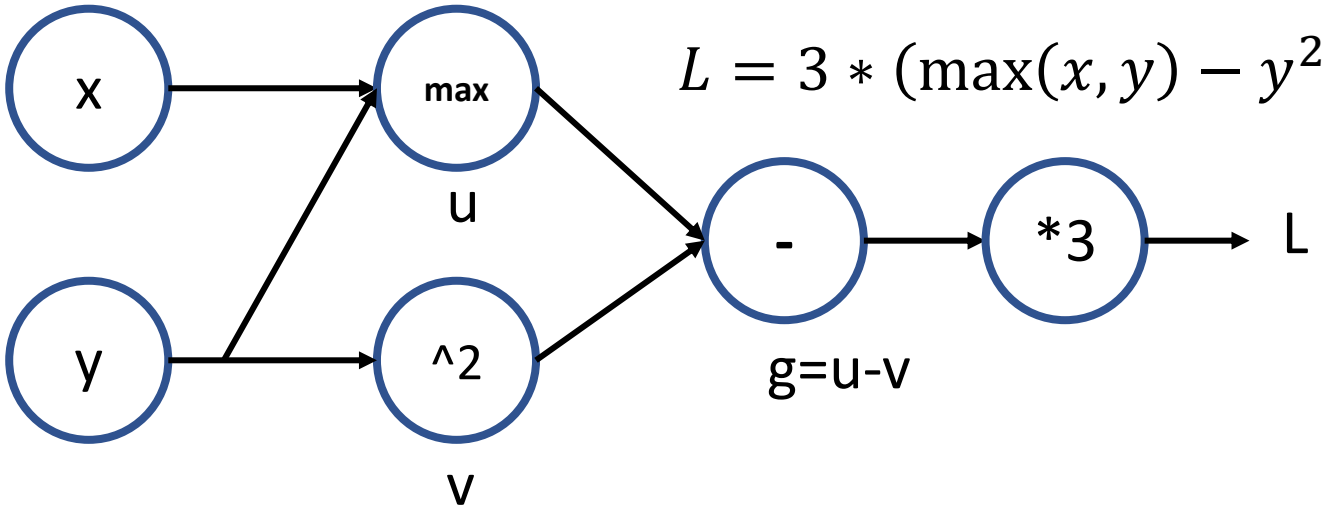
From Last Lecture



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

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

Answer: $\frac{\partial f}{\partial x} = 3$, $\frac{\partial f}{\partial y} = -2$, $\frac{\partial f}{\partial z} = 2$

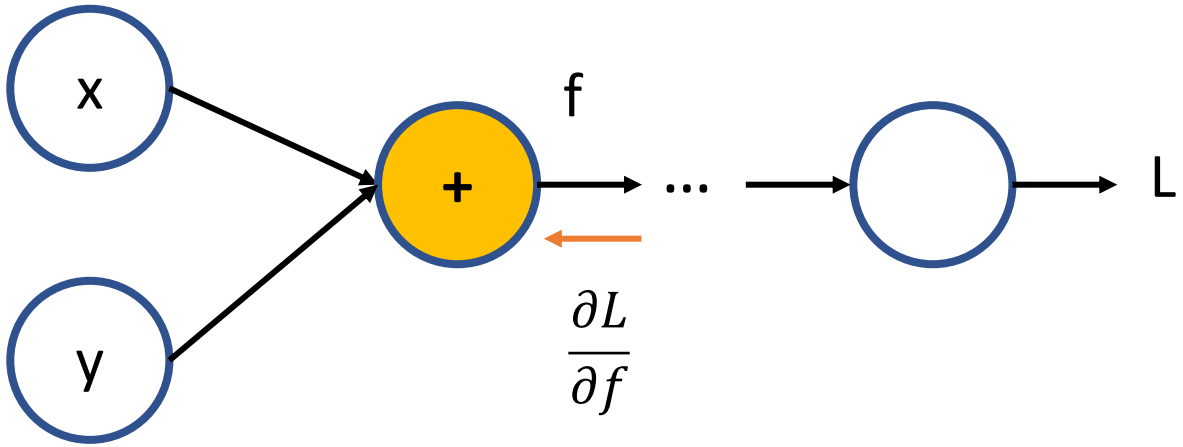


$L = 3 * (\max(x, y) - y^2)$

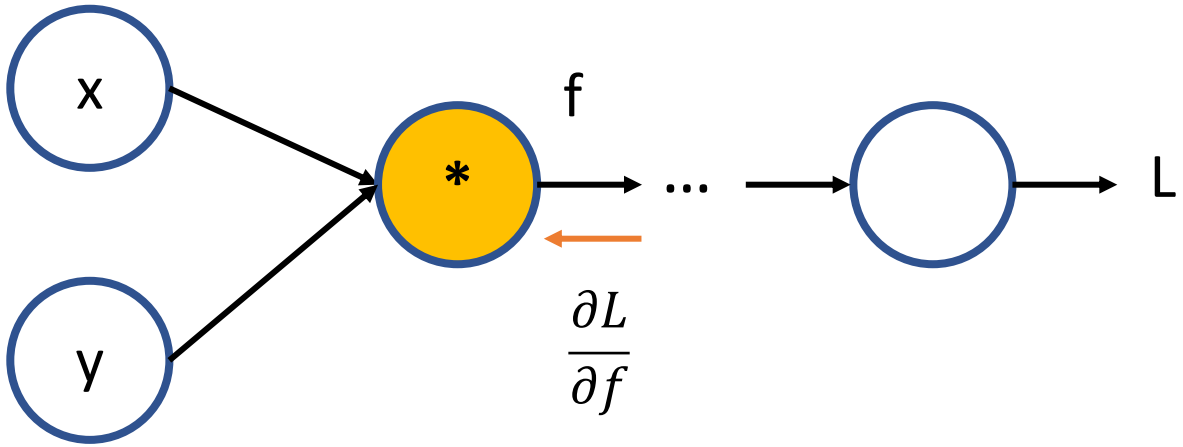
x=2, y=3

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

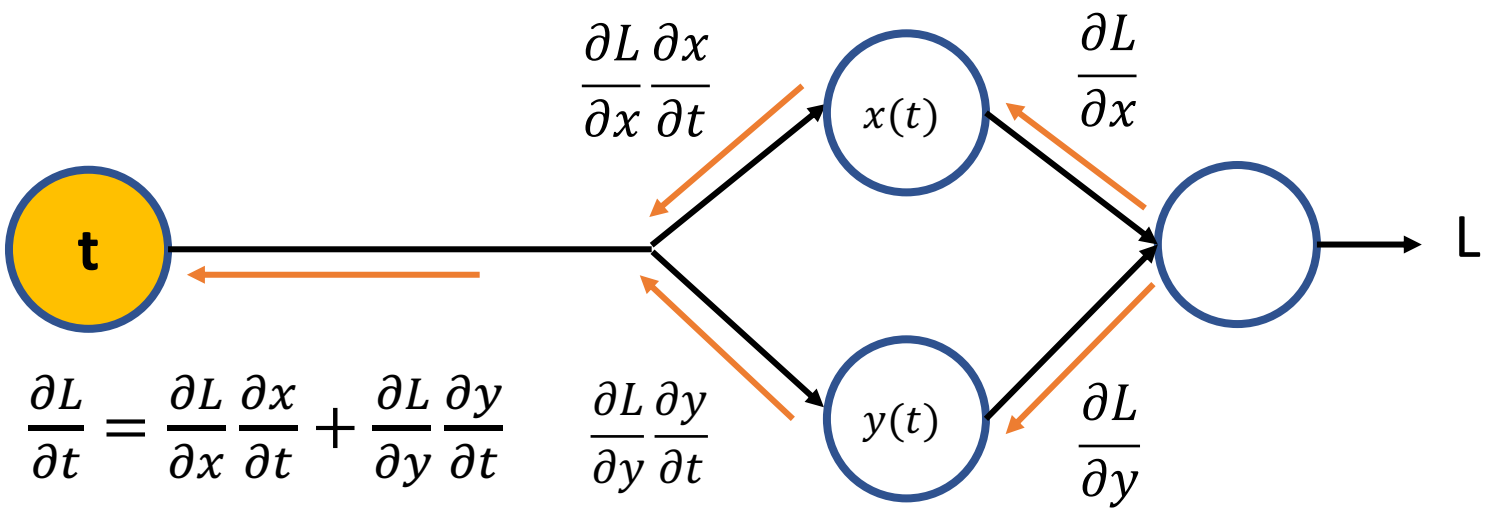
Addition



Multiplication



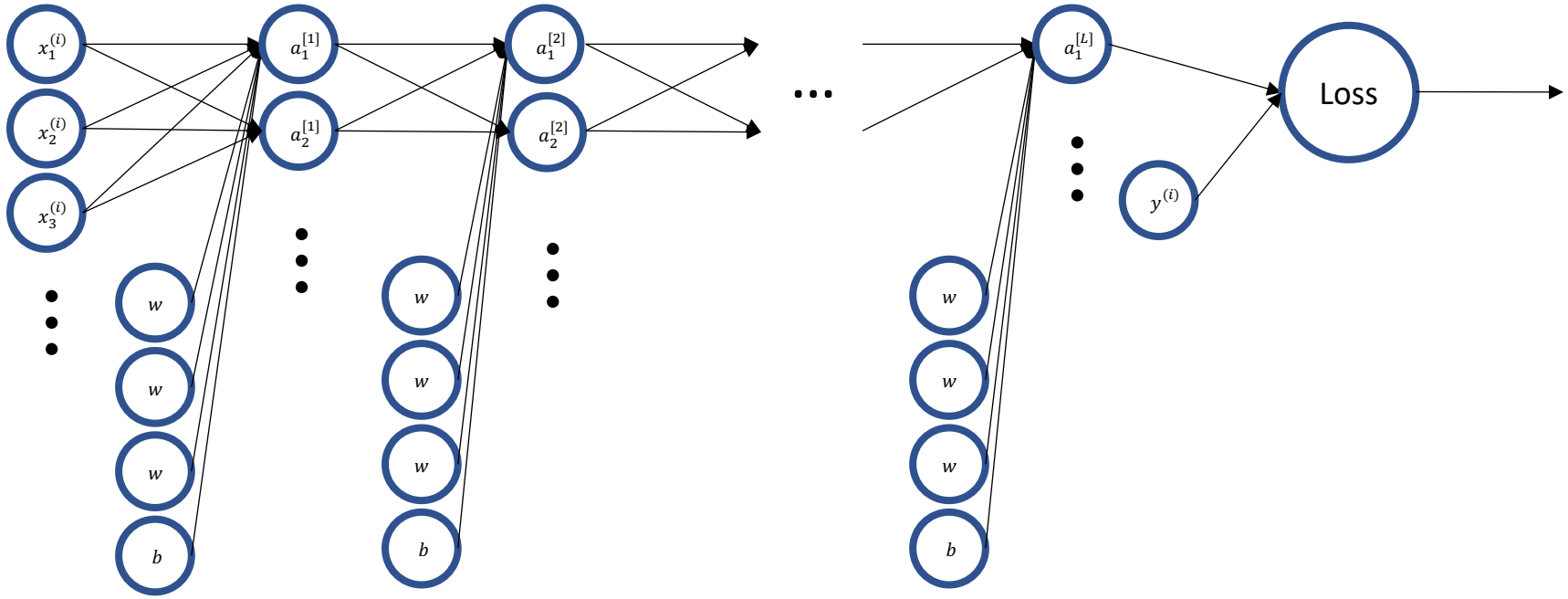
Branch



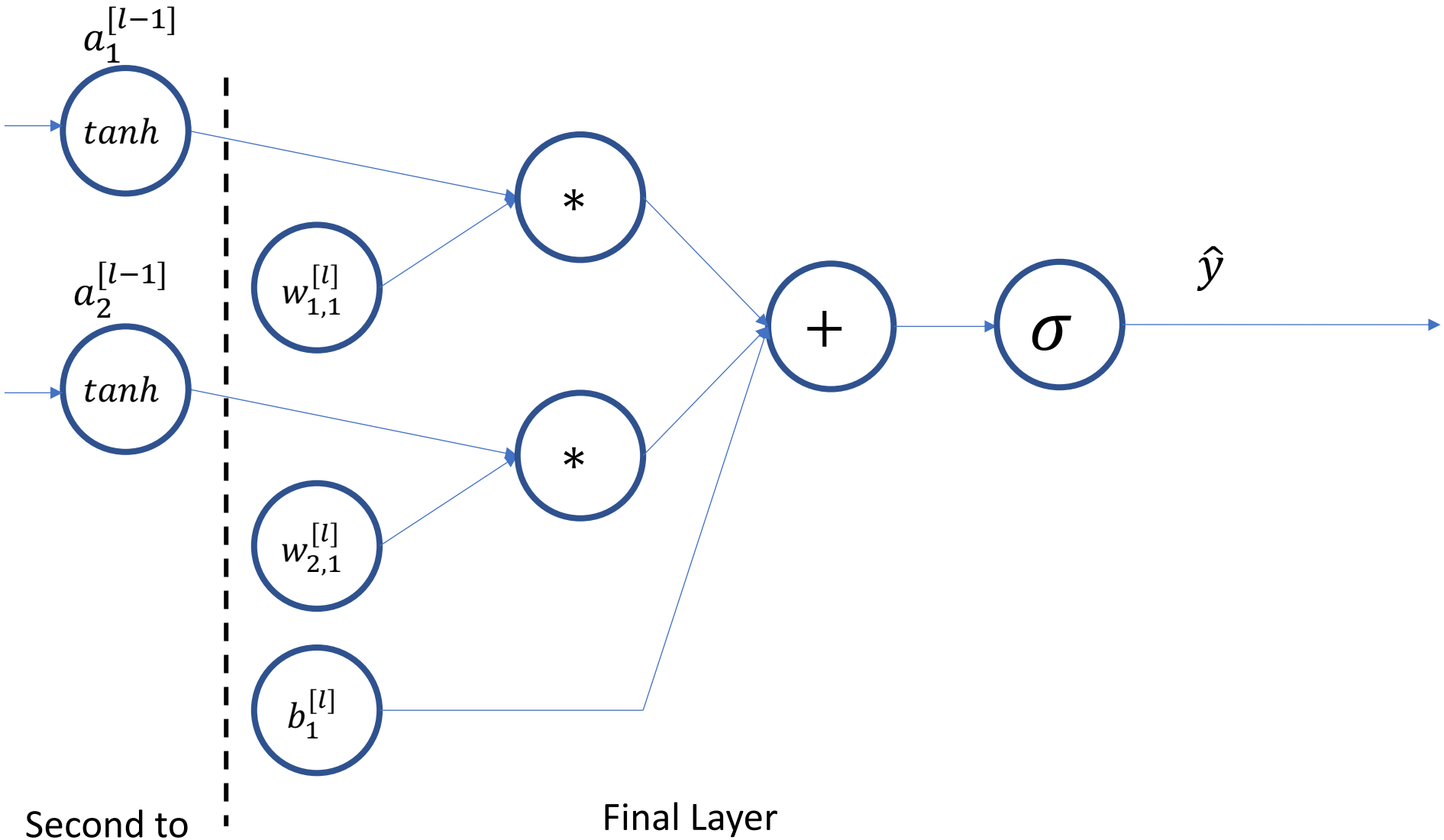
- Sum gradients from branches
- Mathematically, this is the Multivariable Chain Rule

Working Towards

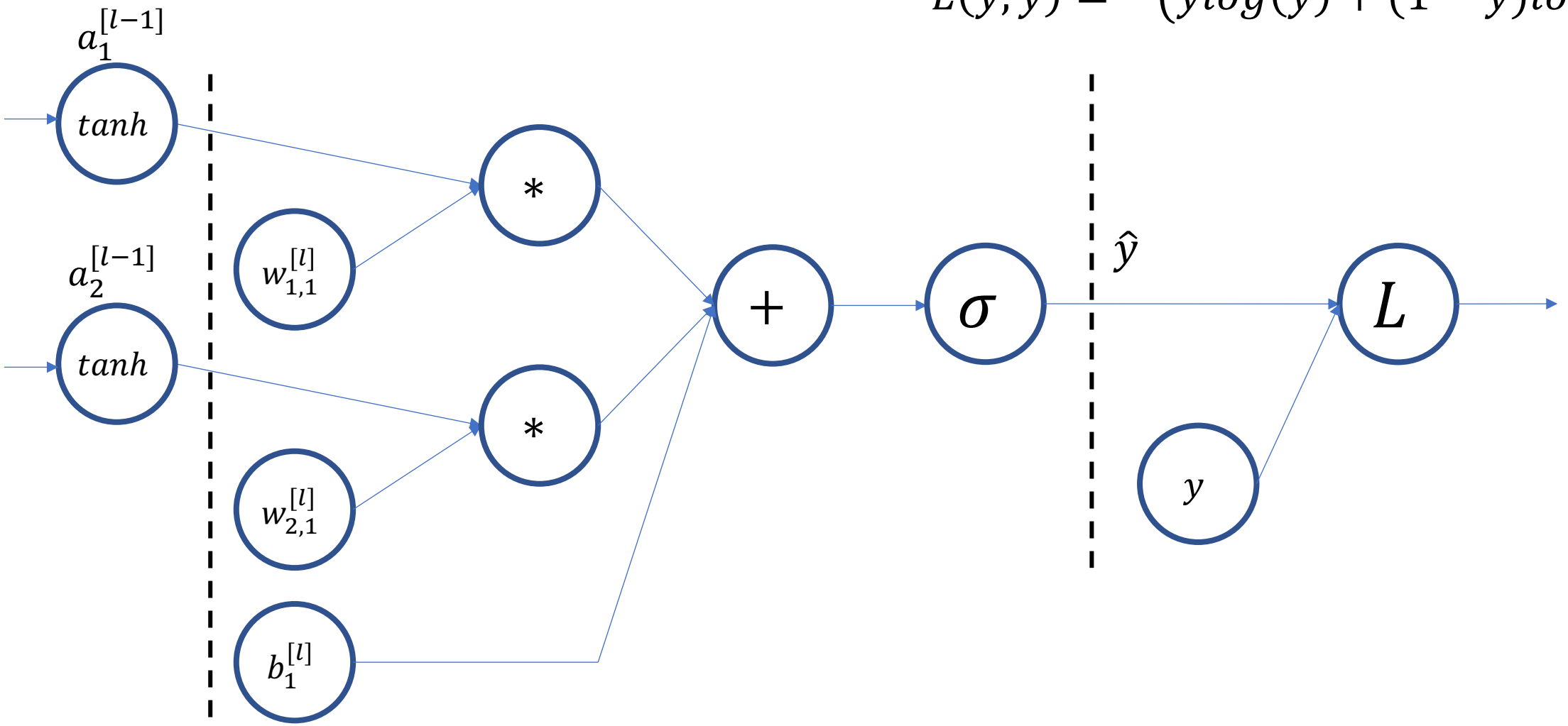
- Model neural network as a compute graph
- Model loss/cost function as another operation at the end
- Backprop on compute graph to find ∂L w.r.t. each parameter



Example with Training Neural Network

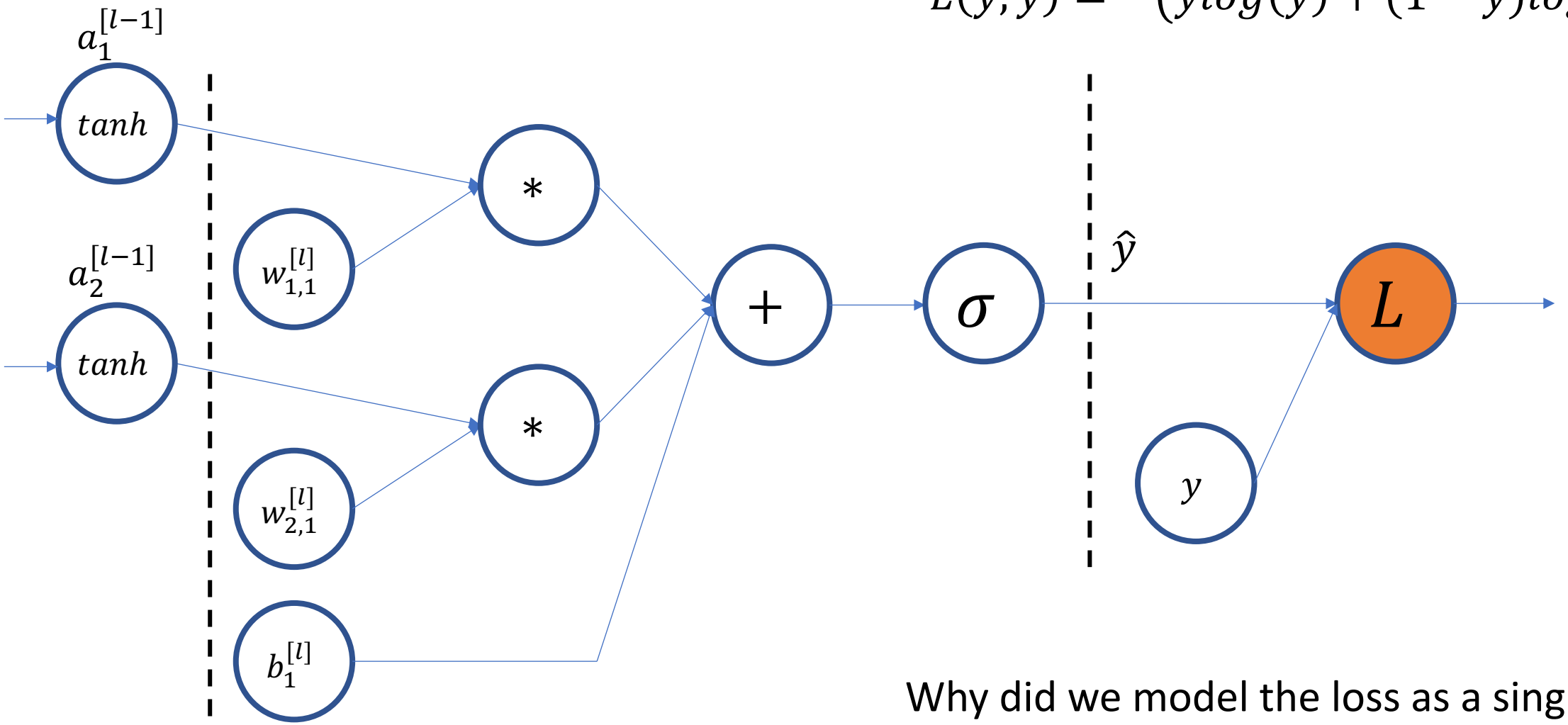

$a_1^{[l-1]}$
$tanh$
$a_2^{[l-1]}$
$tanh$
$w_{1,1}^{[l]}$
$w_{2,1}^{[l]}$
$b_1^{[l]}$
*
*
+
σ
$\hat{y}$
Second to Last Layer
Final Layer

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



Brad Quinton, Scott Chin

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

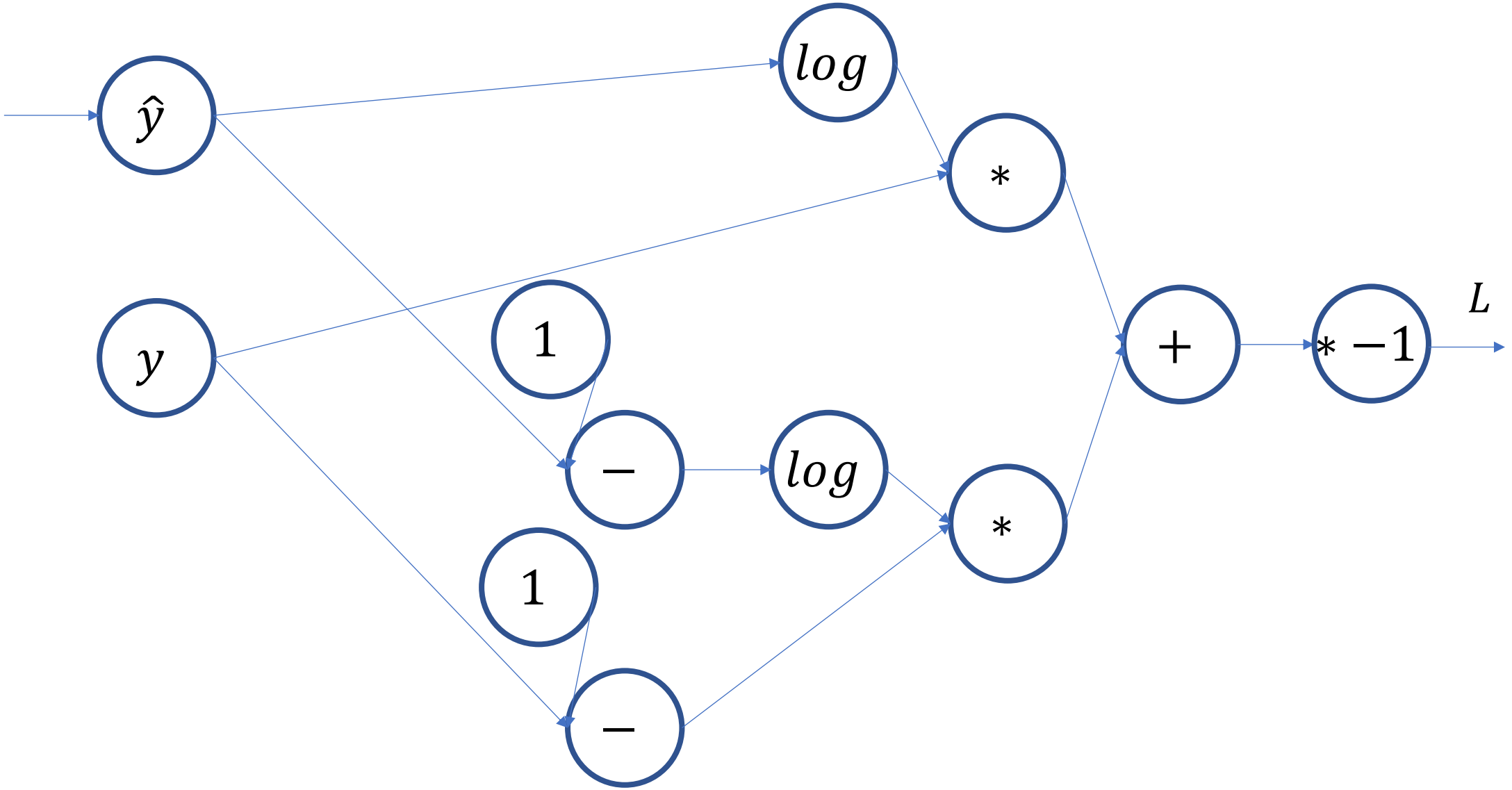


Second to Last Layer

Final Layer

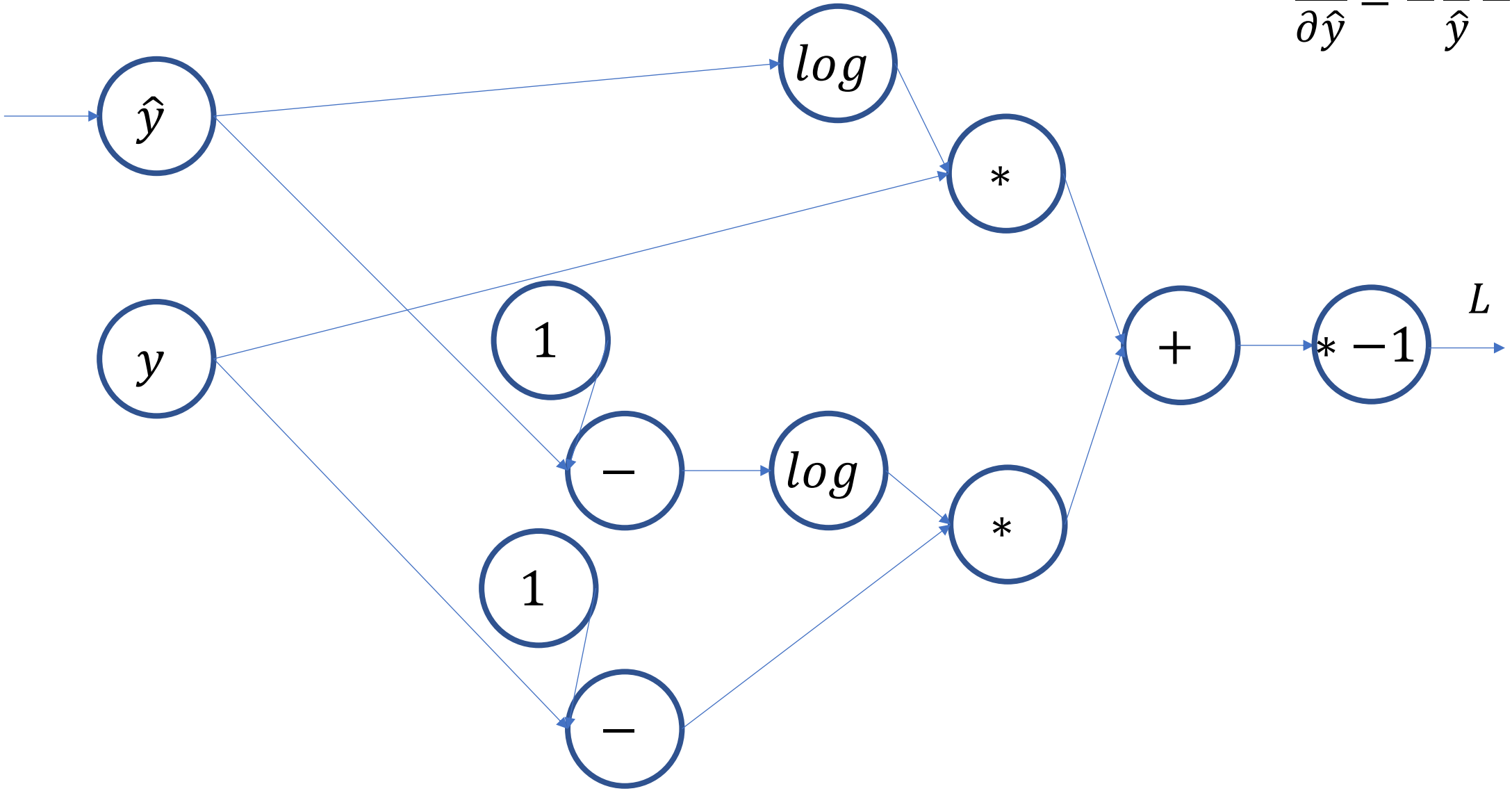
Why did we model the loss as a single node?

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



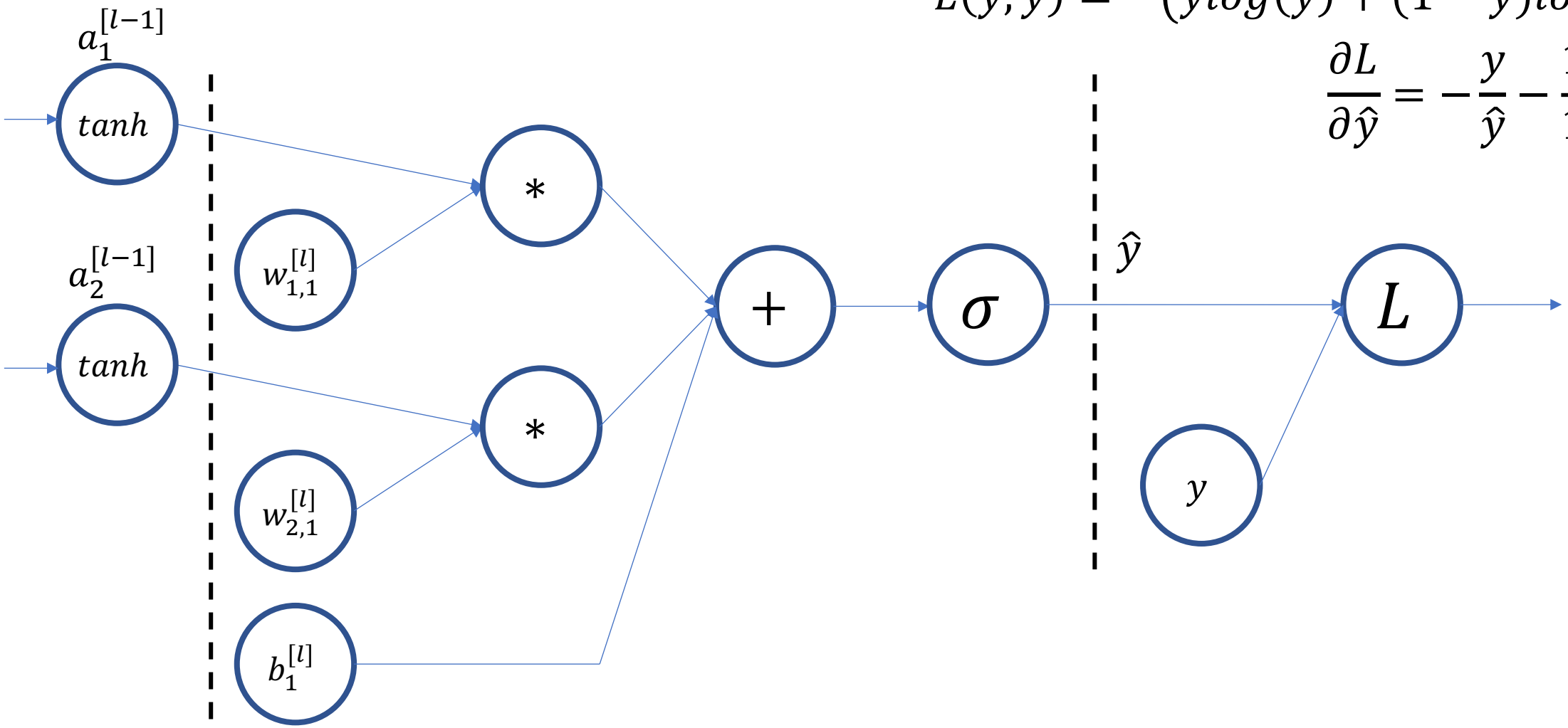
We could have modelled L more explicitly: $L(\hat{y}, y) = -\big(ylog(\hat{y}) + (1-y)log(1-\hat{y})\big)$

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



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

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



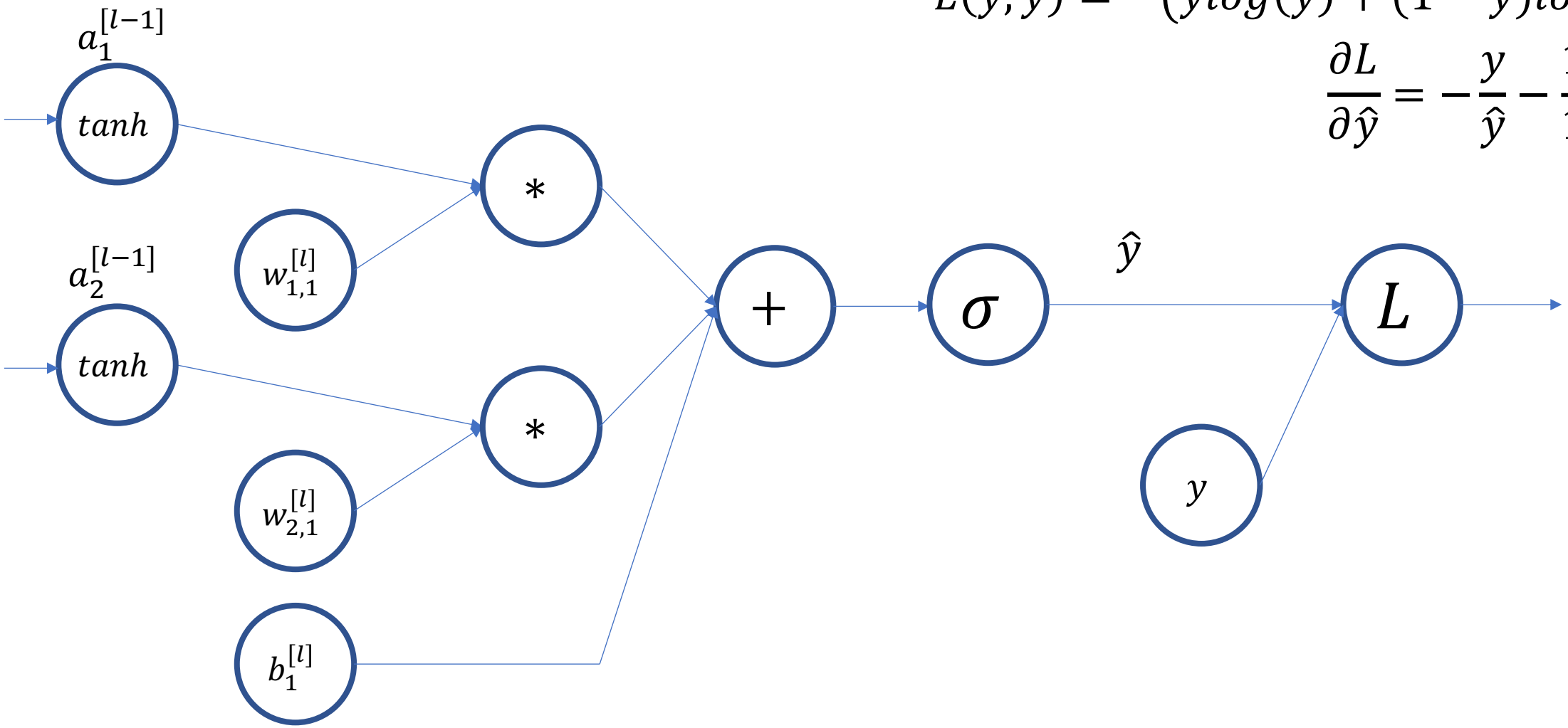
Second to Last Layer

Final Layer

Brad Quinton, Scott Chin

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

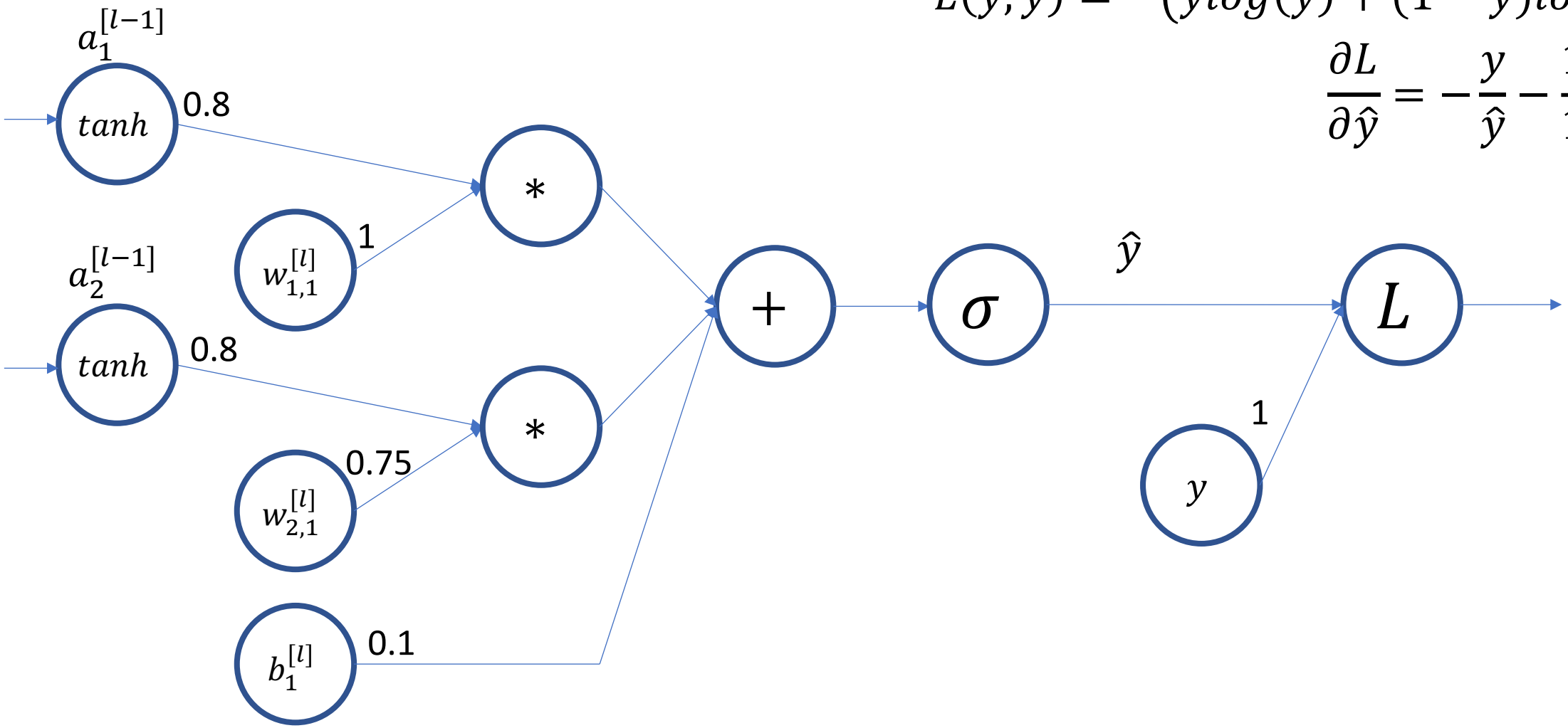
$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}$$



Brad Quinton, Scott Chin

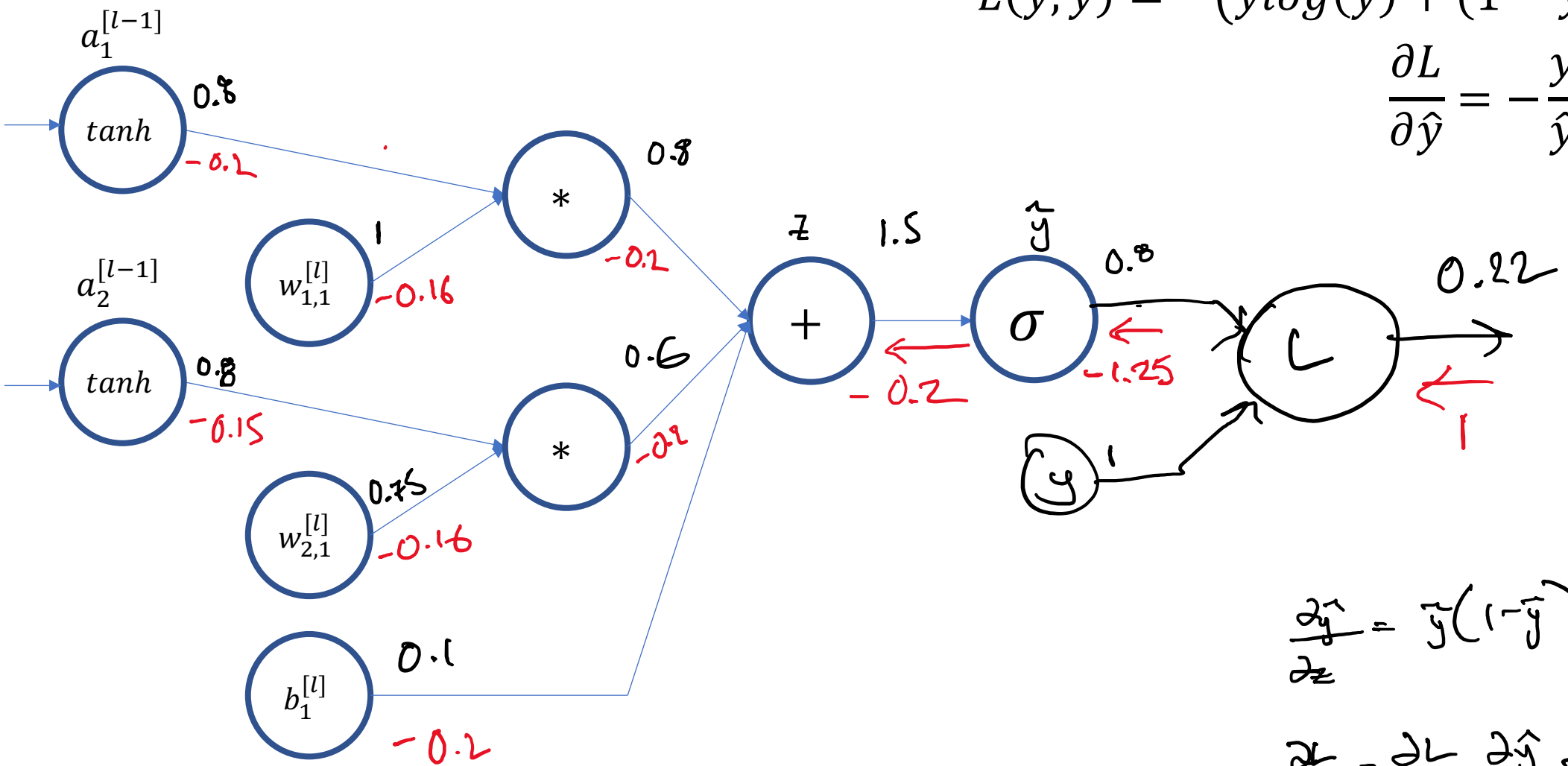
$$L(\hat{y}, y) = -\left(y log(\hat{y}) + (1-y) log(1-\hat{y})\right)$$

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



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

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

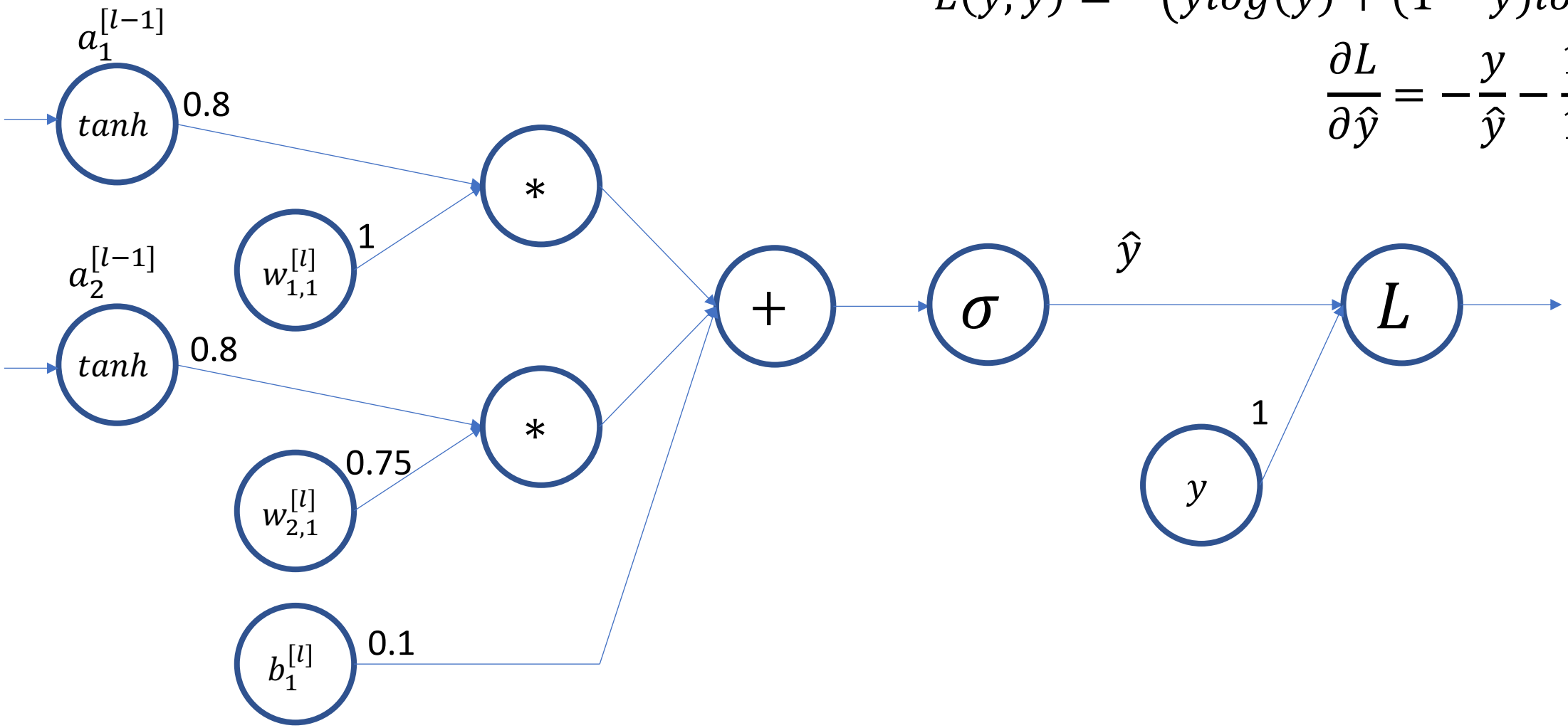


$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y}) = 0.16$$

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

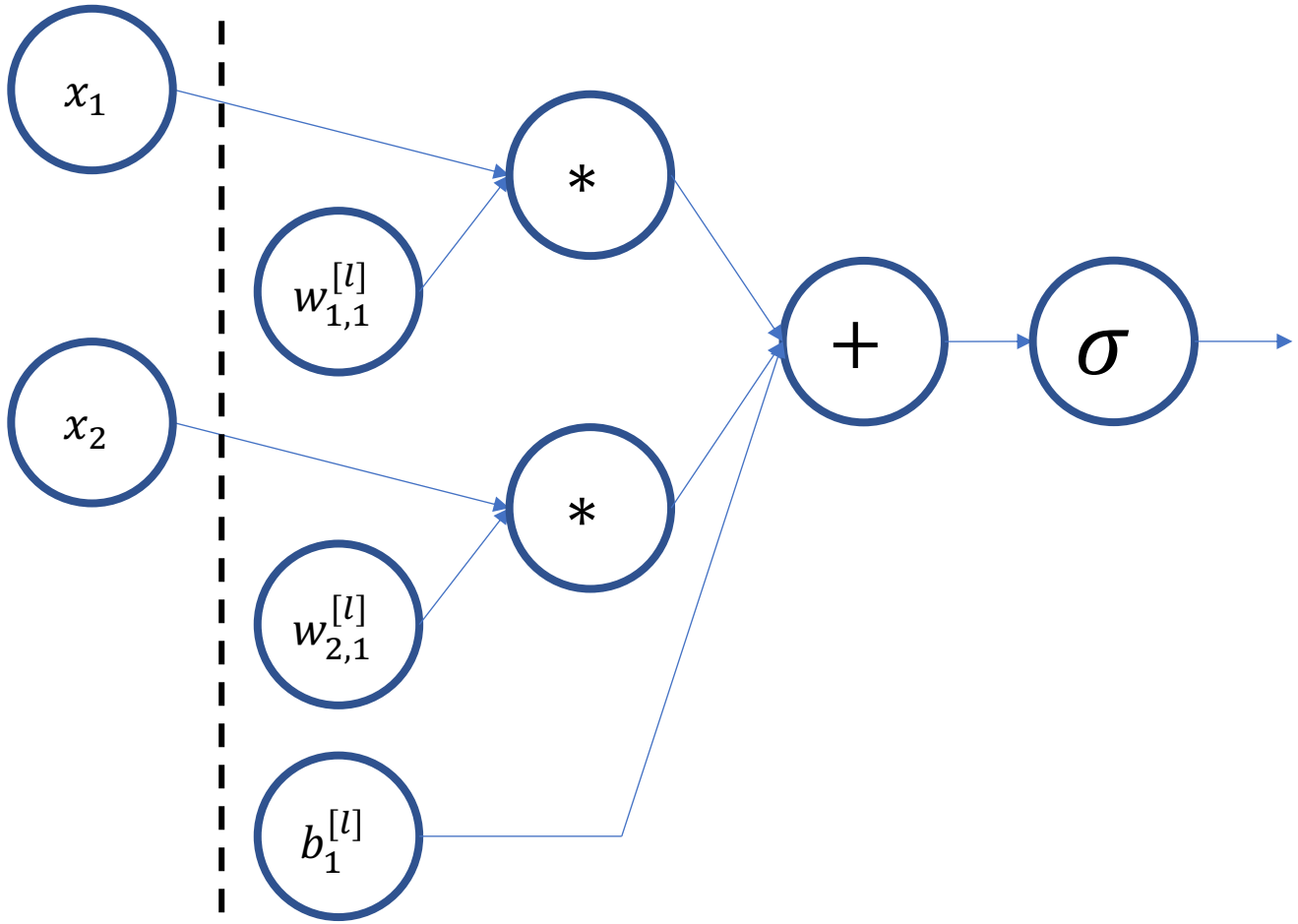
$$L(\hat{y}, y) = -\big(y log(\hat{y}) + (1-y) log(1-\hat{y})\big)$$

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



Practice Exercise: Replace Sigmoid and Loss node with equivalent basic math operators and see if you get the same answer

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

Example - Backprop through Softmax layer revisited

Consider $n_c = 3$

$$L(\hat{y}, y) = -\sum_{j=1}^{n_c} y_j log(\hat{y}_j) = -y_1 log(\hat{y}_1) - y_2 log(\hat{y}_2) - y_3 log(\hat{y}_3)$$

$$\frac{\partial L(\hat{y}, y)}{\partial z_1}, \frac{\partial L(\hat{y}, y)}{\partial z_2}, \frac{\partial L(\hat{y}, y)}{\partial z_3}$$

$$\frac{\partial L(\hat{y}, y)}{\partial z_1} = -y_1 \frac{1}{\hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1} - y_2 \frac{1}{\hat{y}_2}\frac{\partial \hat{y}_2}{\partial z_1} - y_3 \frac{1}{\hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_1}$$

$$\hat{y}_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}} \qquad \hat{y}_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}} \qquad \hat{y}_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$\frac{\partial \hat{y}_1}{\partial z_1} = \hat{y}_1(1 - \hat{y}_1) \qquad \frac{\partial \hat{y}_2}{\partial z_1} = -\hat{y}_2\hat{y}_1$$

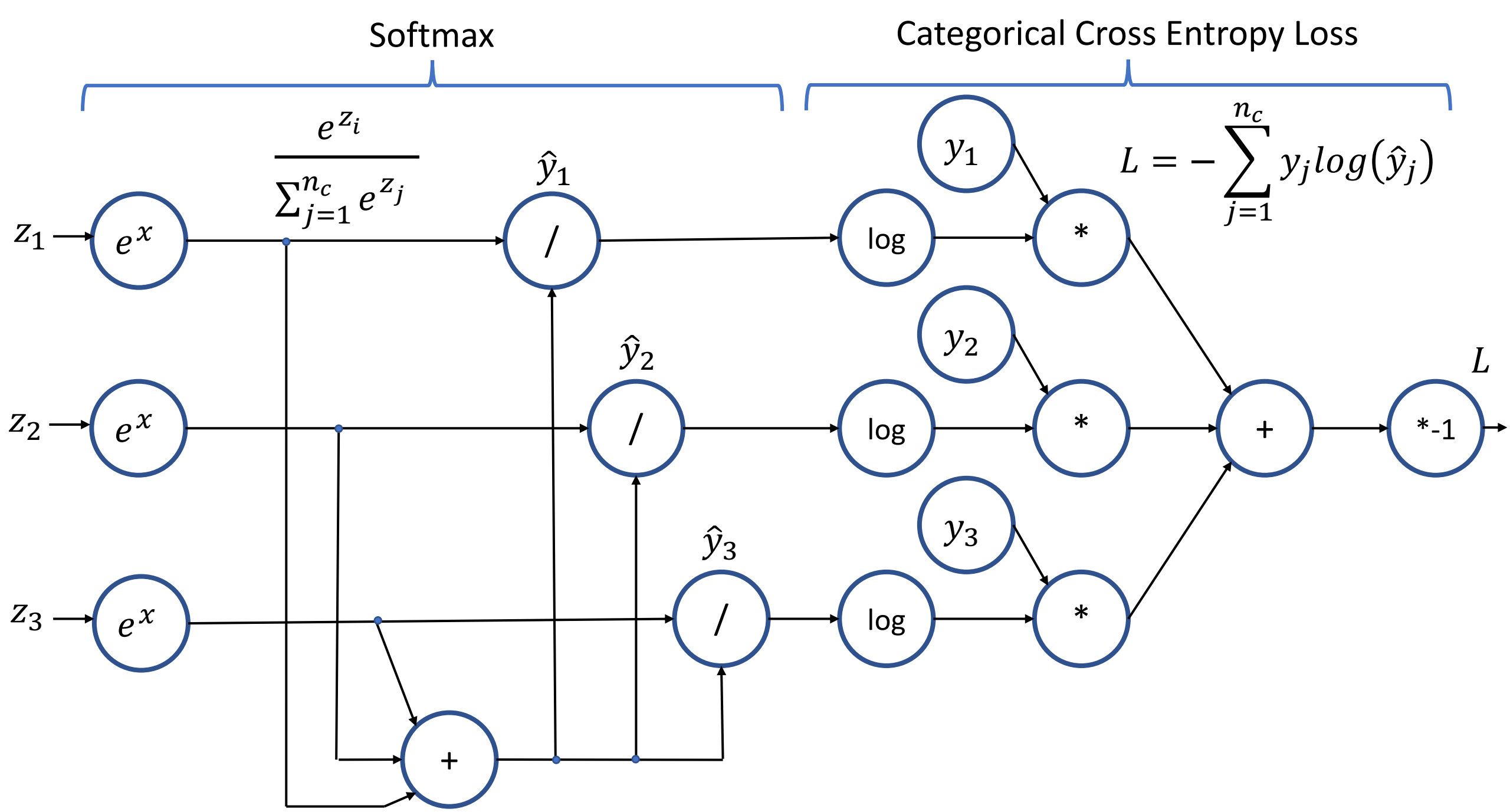
Recall our closed form symbolic derivation

$$\frac{\partial L(\hat{y}, y)}{\partial z_1} = -y_1 \frac{1}{\hat{y}_1}\hat{y}_1(1 - \hat{y}_1) - y_2 \frac{1}{\hat{y}_2}(-\hat{y}_2\hat{y}_1) - y_3 \frac{1}{\hat{y}_3}(-\hat{y}_3\hat{y}_1)$$

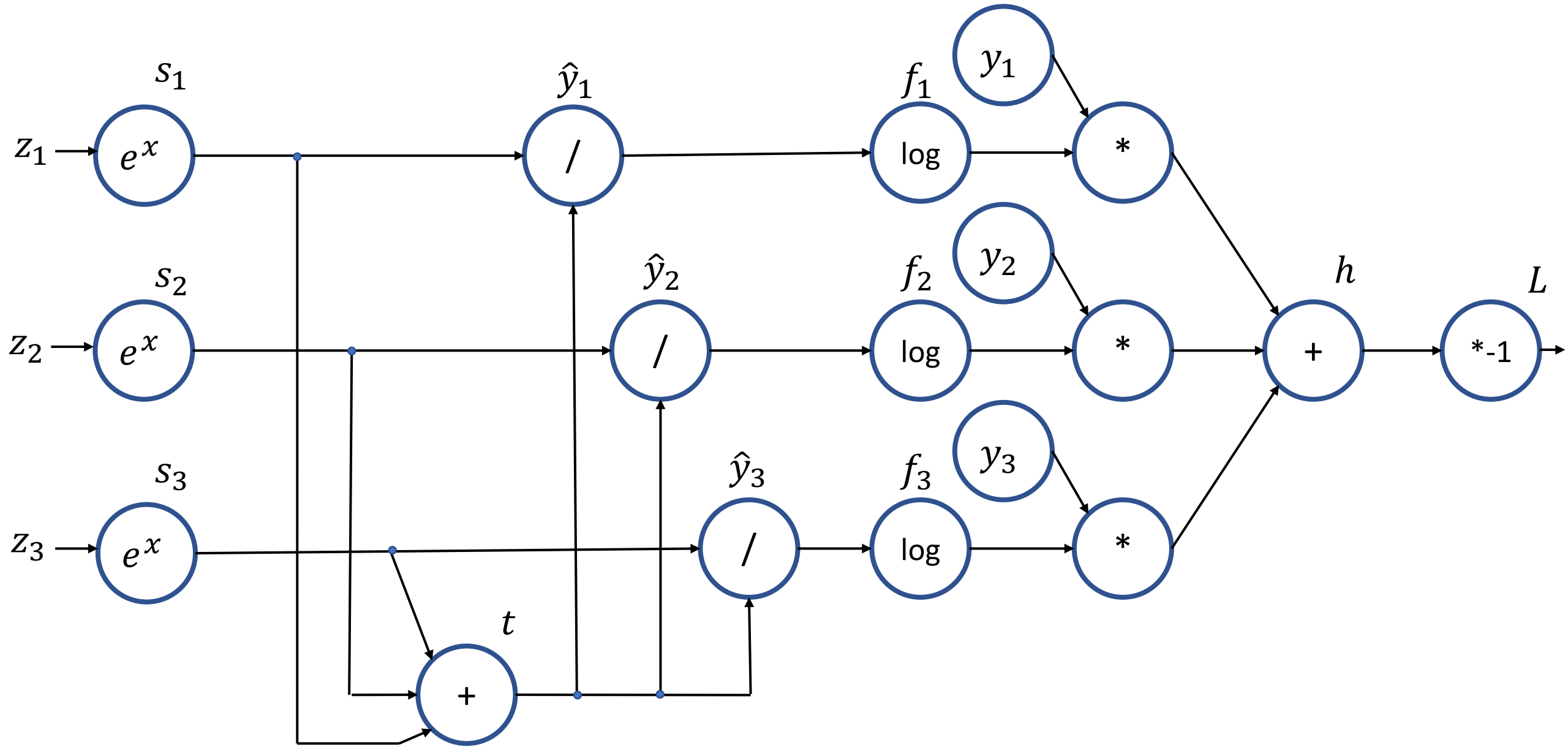
$$= -y_1 \frac{1}{\hat{y}_1}\hat{y}_1(1 - \hat{y}_1) - y_2 \frac{1}{\hat{y}_2}(-\hat{y}_2\hat{y}_1) - y_3 \frac{1}{\hat{y}_3}(-\hat{y}_3\hat{y}_1)$$

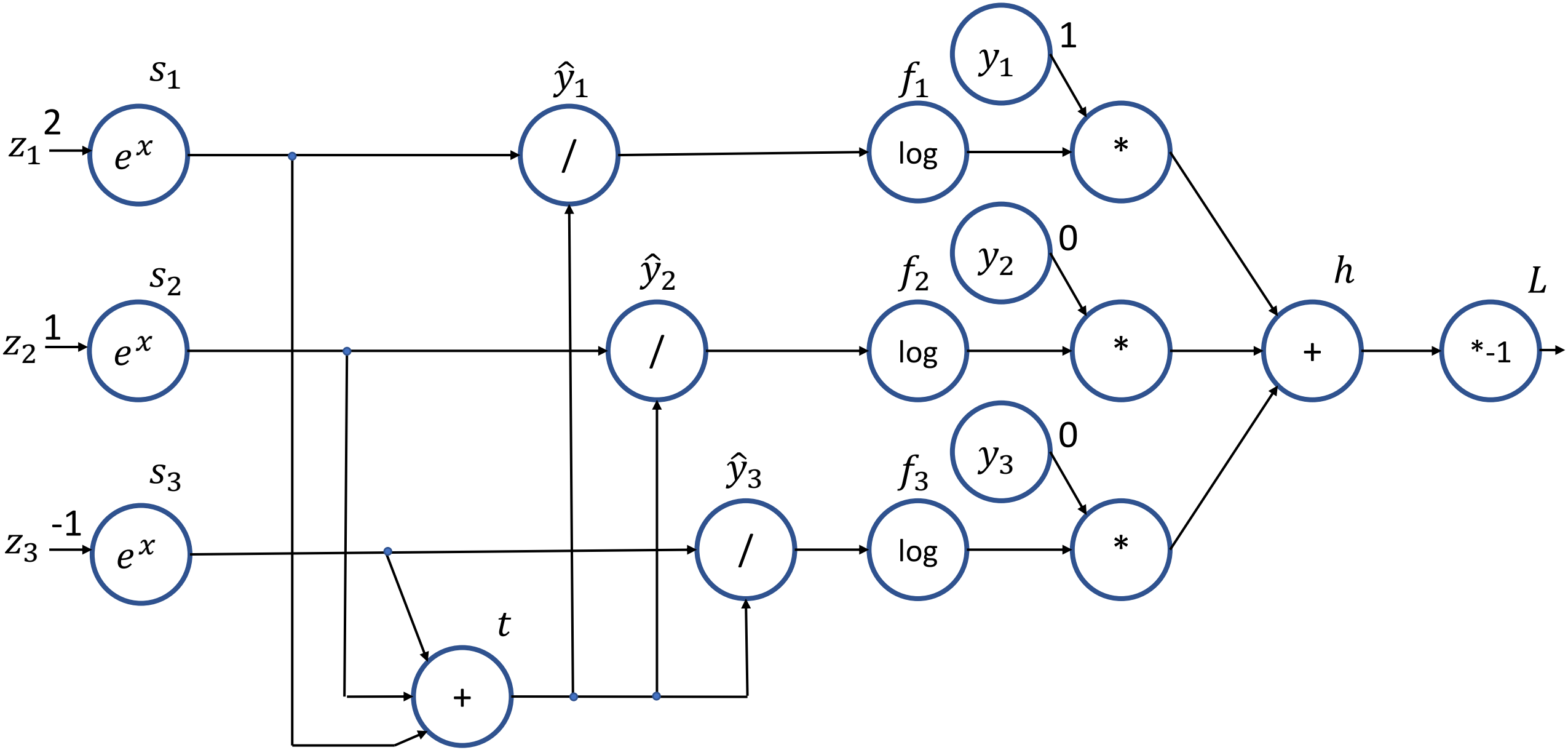
$$= \hat{y}_1(y_1 + y_2 + y_3) - y_1 \qquad y_1 + y_2 + y_3 = 1$$

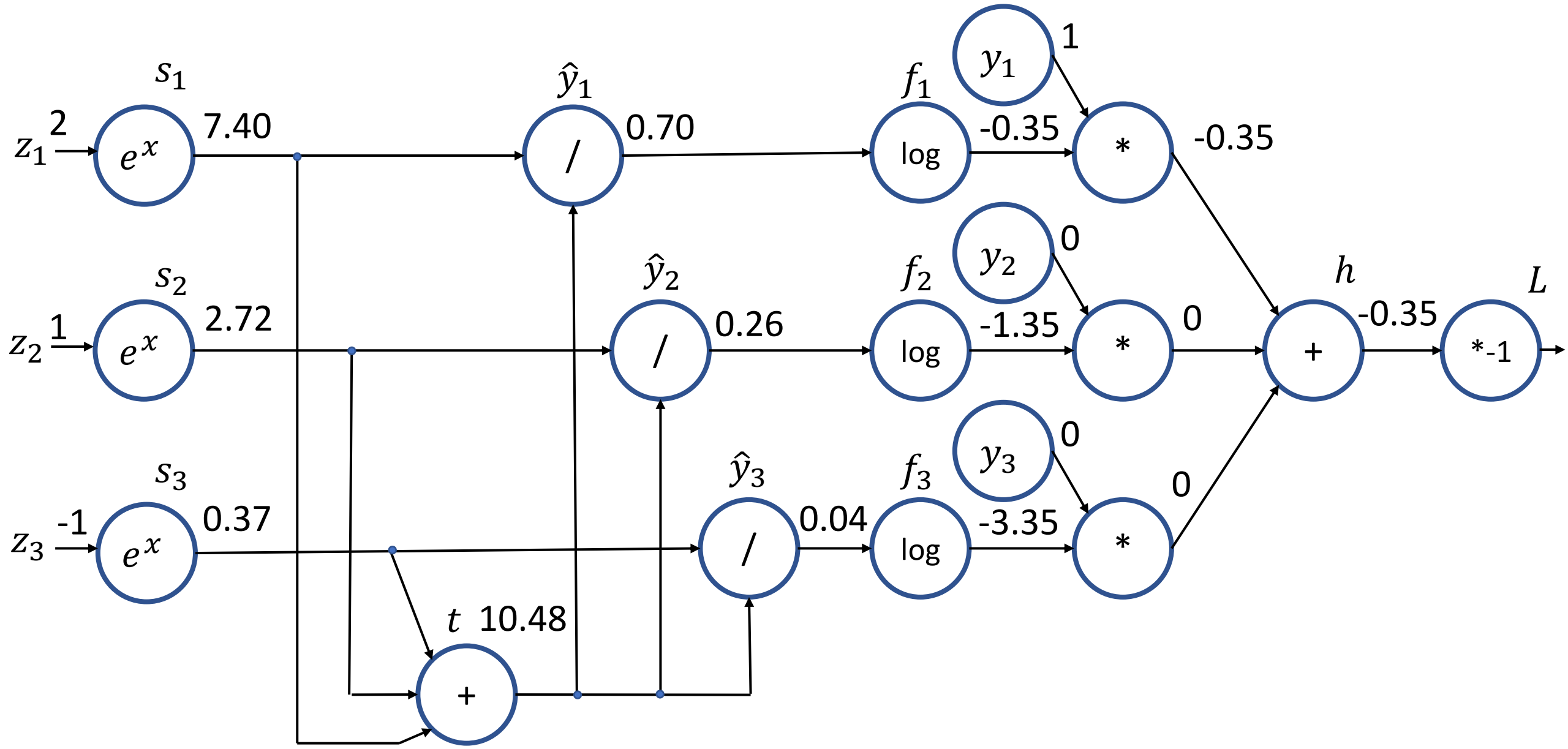
$$= \hat{y}_1 - y_1$$



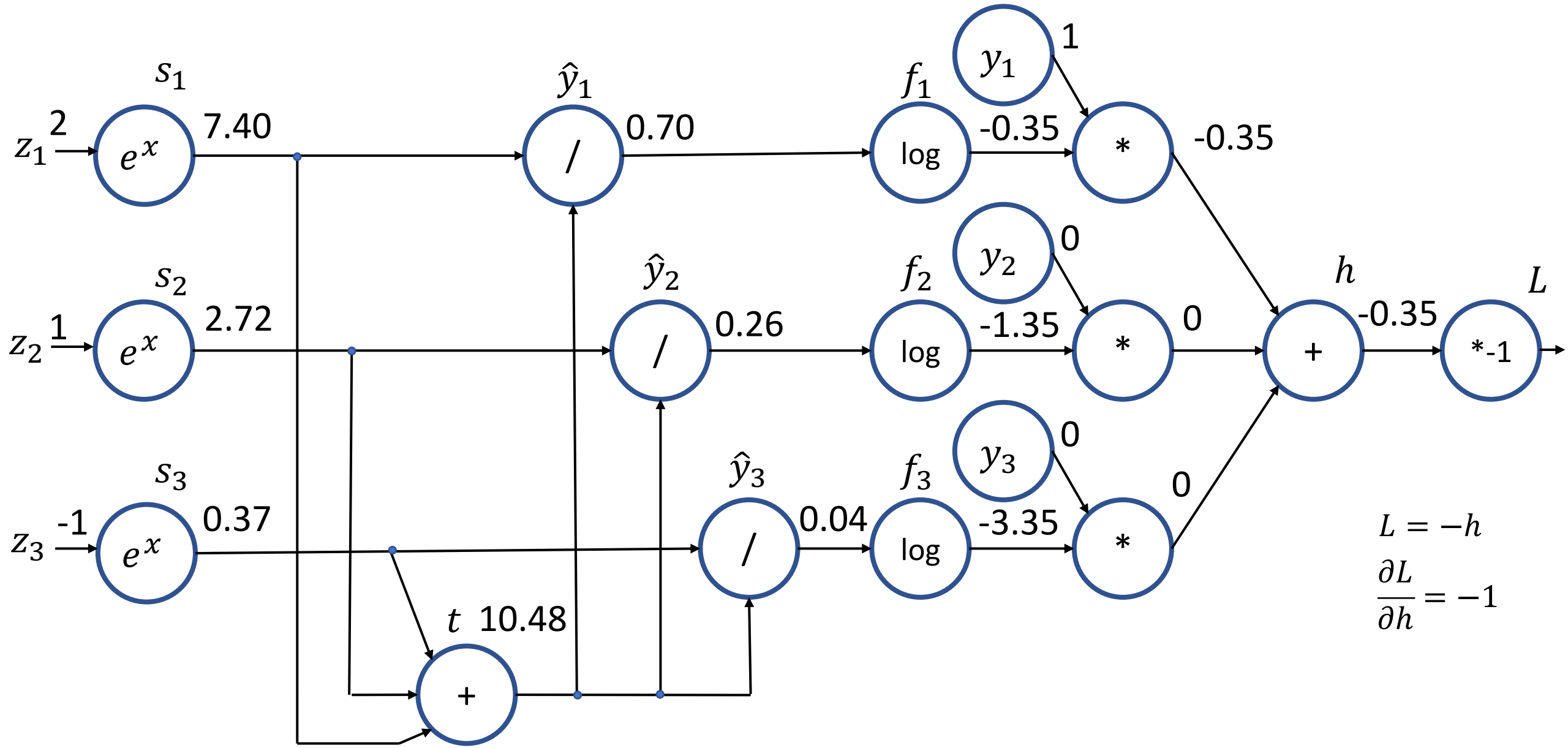
Brad Quinton, Scott Chin

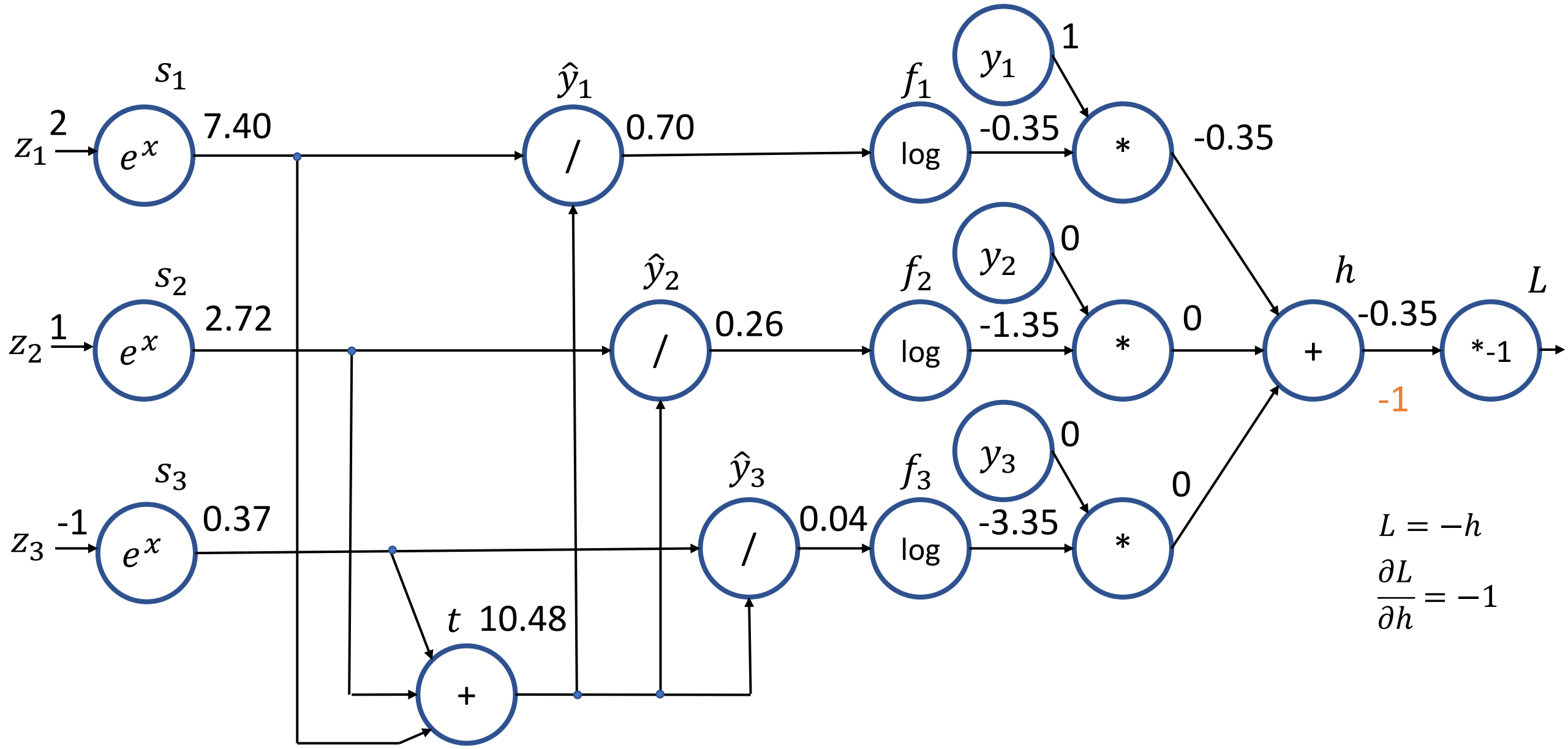

z_1
z_2
z_3
e^x
e^x
e^x
s_1
s_2
s_3
t
+
/
/
/
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
log
log
log
f_1
f_2
f_3
y_1
y_2
y_3
*
*
*
+
h
*-1
L

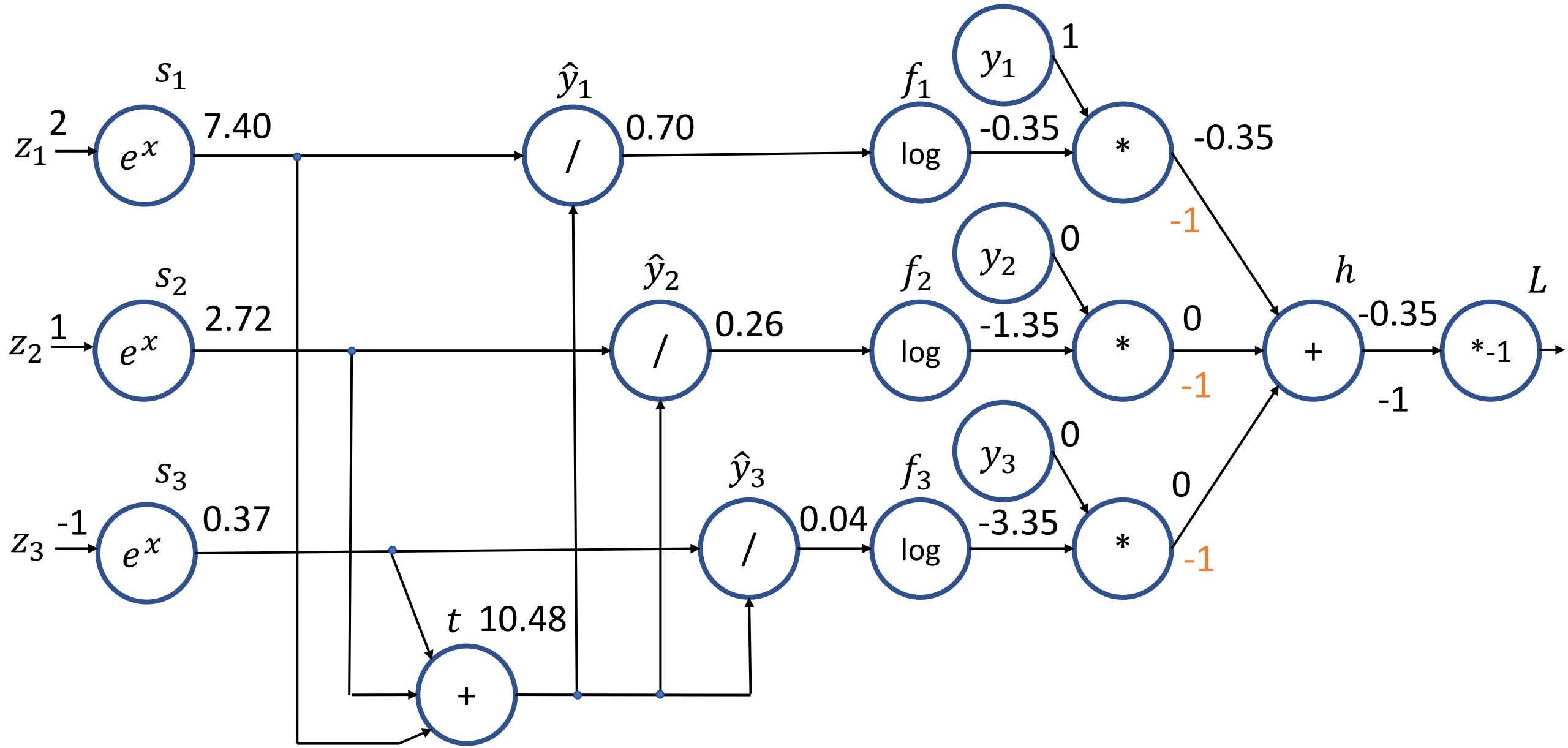

z_1
2
e^x
s_1
/
$\hat{y}_1$
log
f_1
y_1
1
*
z_2
1
e^x
s_2
/
$\hat{y}_2$
log
f_2
y_2
0
*
z_3
-1
e^x
s_3
/
$\hat{y}_3$
log
f_3
y_3
0
*
+
t
+
h
*-1
L

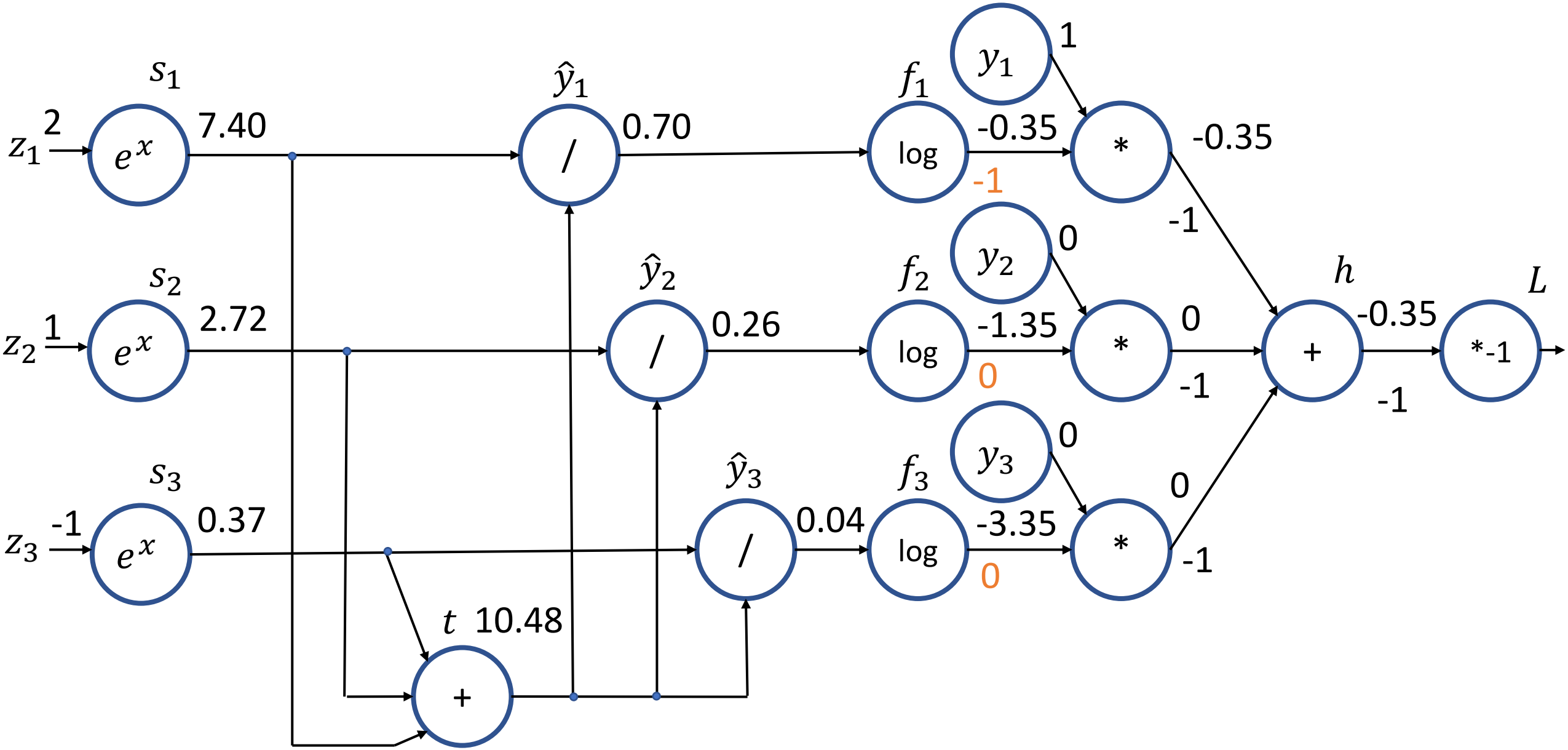


Brad Quinton, Scott Chin


s_1
z_1 2
e^x
7.40
$\hat{y}_1$
/
0.70
f_1
log
y_1 1
-0.35
*
-0.35
s_2
z_2 1
e^x
2.72
$\hat{y}_2$
/
0.26
f_2
log
y_2 0
-1.35
*
0
h
+
-0.35
L
*-1
s_3
z_3 -1
e^x
0.37
$\hat{y}_3$
/
0.04
f_3
log
y_3 0
-3.35
*
0
t 10.48
+
$L = -h$
$\frac{\partial L}{\partial h} = -1$

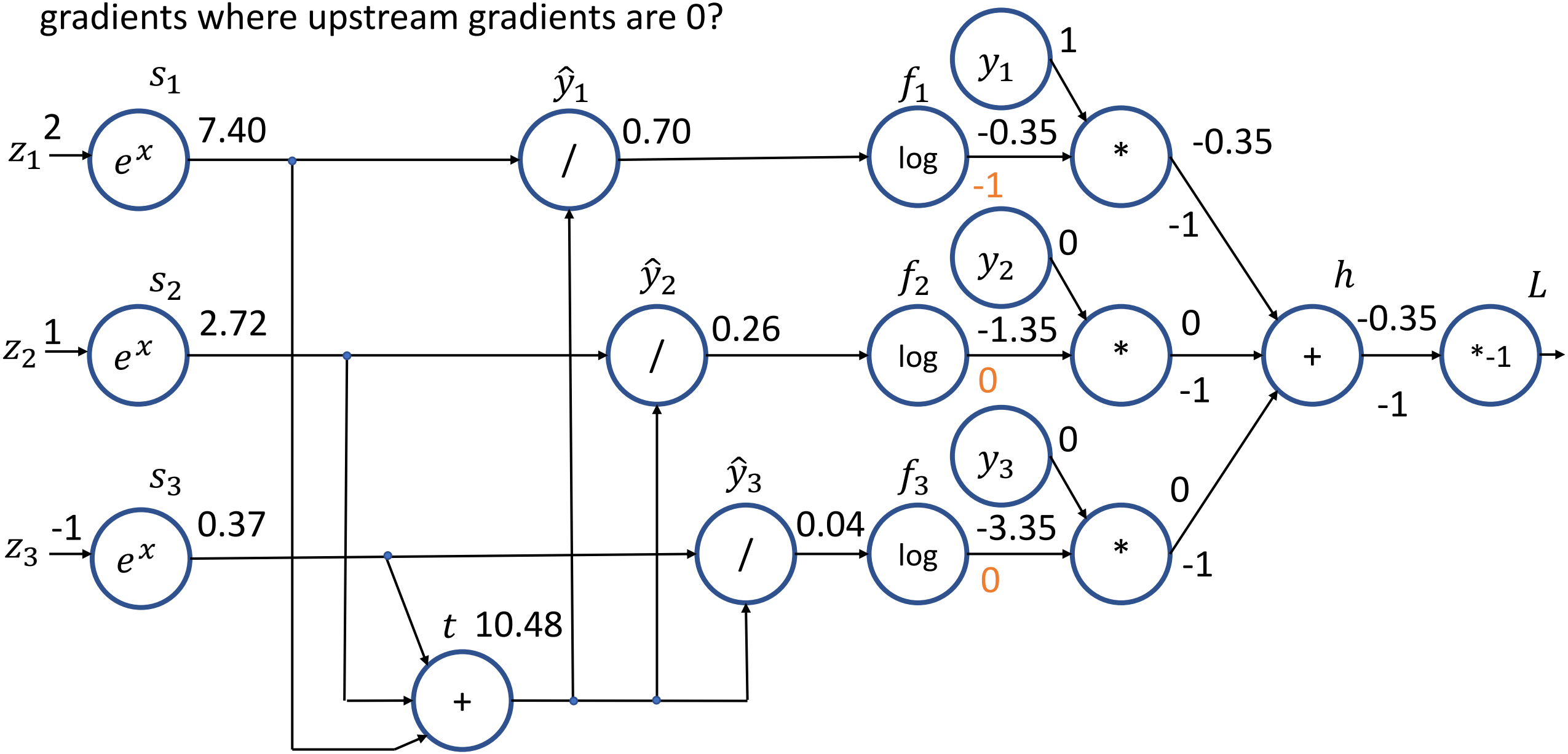

s_1
$\hat{y}_1$
f_1
y_1
1
z_1
2
e^x
7.40
/
0.70
log
-0.35
*
-0.35
s_2
$\hat{y}_2$
f_2
y_2
0
z_2
1
e^x
2.72
/
0.26
log
-1.35
*
0
h
L
-0.35
+
*-1
-1
s_3
$\hat{y}_3$
f_3
y_3
0
z_3
-1
e^x
0.37
/
0.04
log
-3.35
*
0
t 10.48
+
$L = -h$
$\frac{\partial L}{\partial h} = -1$


s_1
z_1 2
e^x
7.40
$\hat{y}_1$
/
0.70
f_1
log
y_1 1
-0.35
*
-0.35
-1
s_2
z_2 1
e^x
2.72
$\hat{y}_2$
/
0.26
f_2
log
y_2 0
-1.35
*
0
-1
h
+
-0.35
L
*-1
-1
s_3
z_3 -1
e^x
0.37
$\hat{y}_3$
/
0.04
f_3
log
y_3 0
-3.35
*
0
-1
t 10.48
+

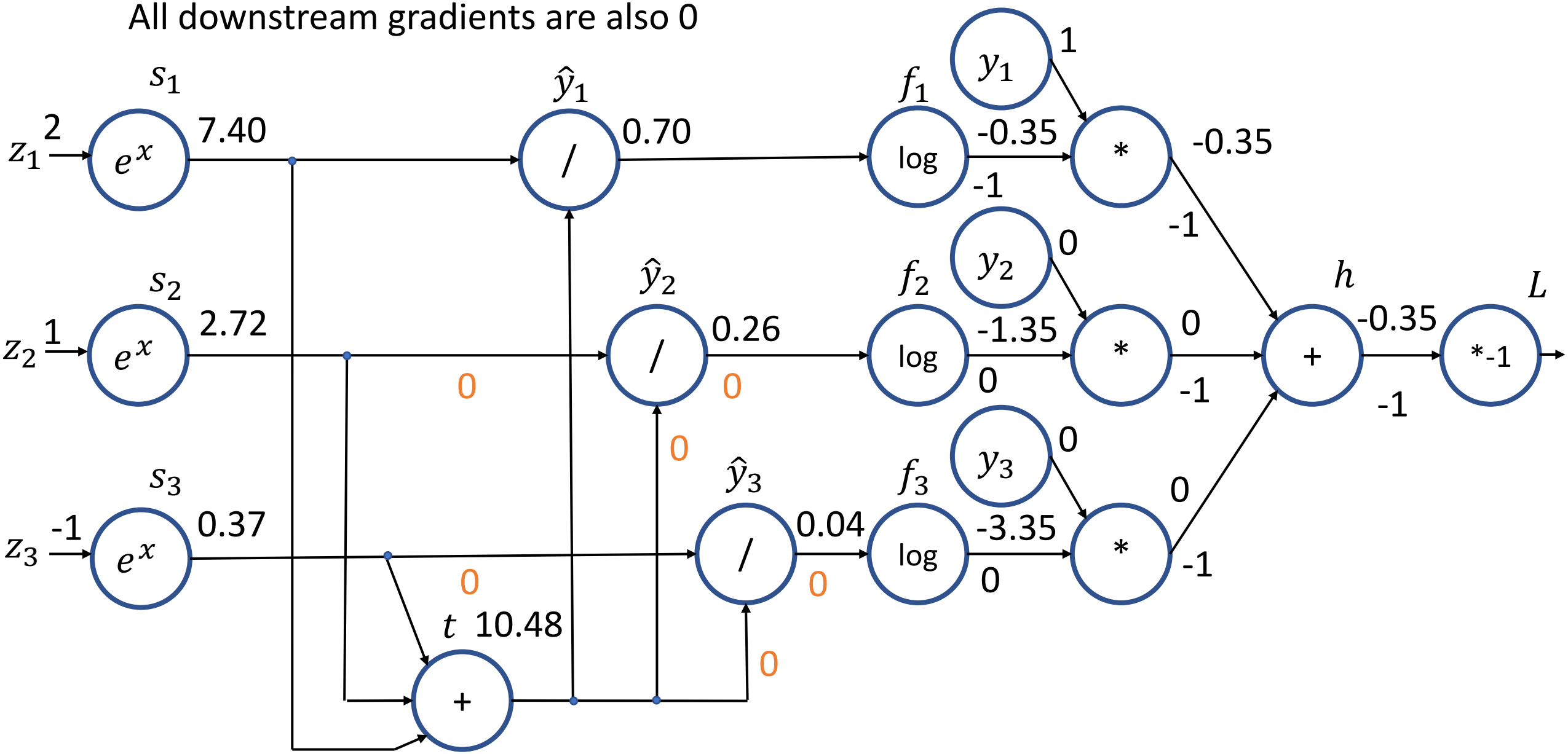


Brad Quinton, Scott Chin

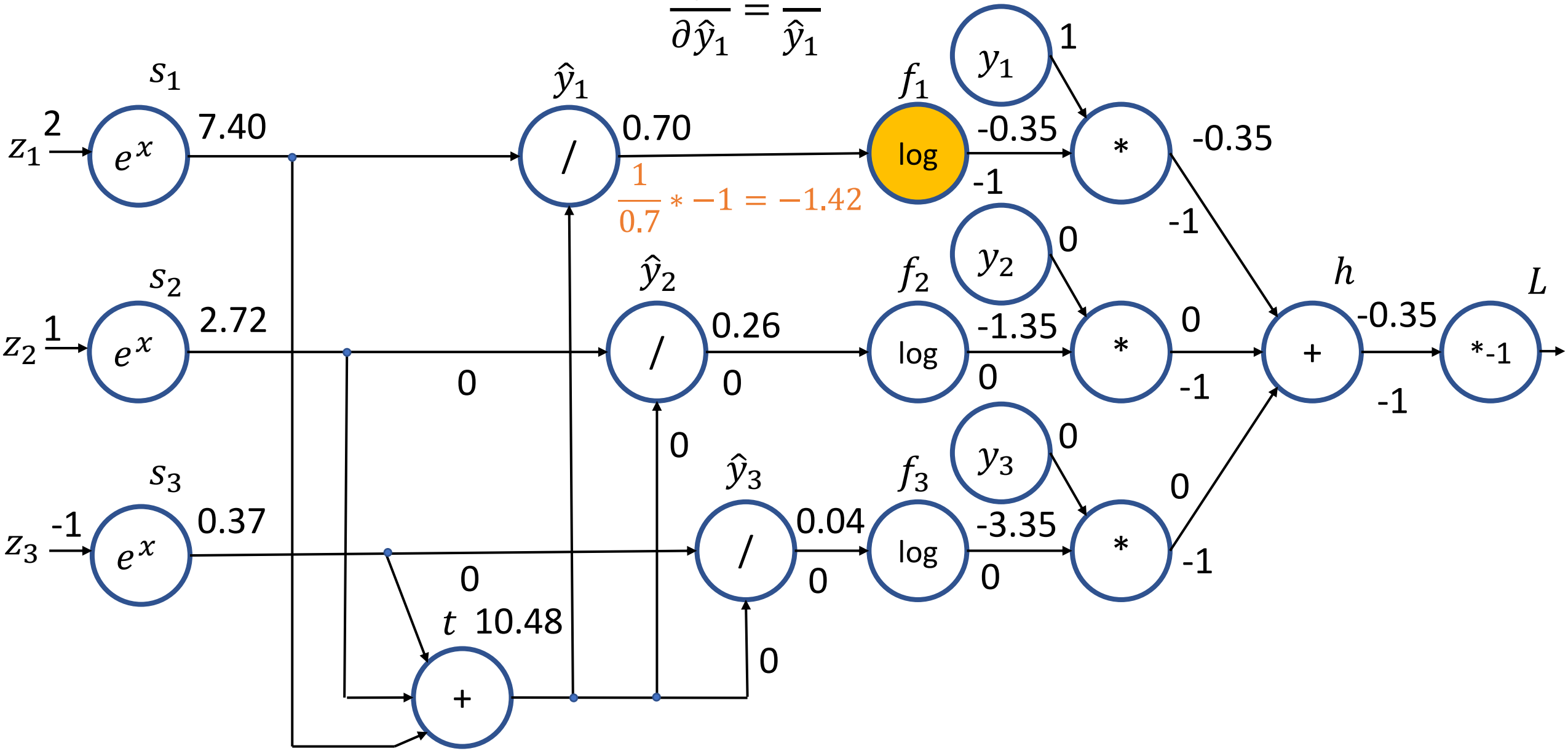
Key Observation: What can we say about downstream gradients where upstream gradients are 0?



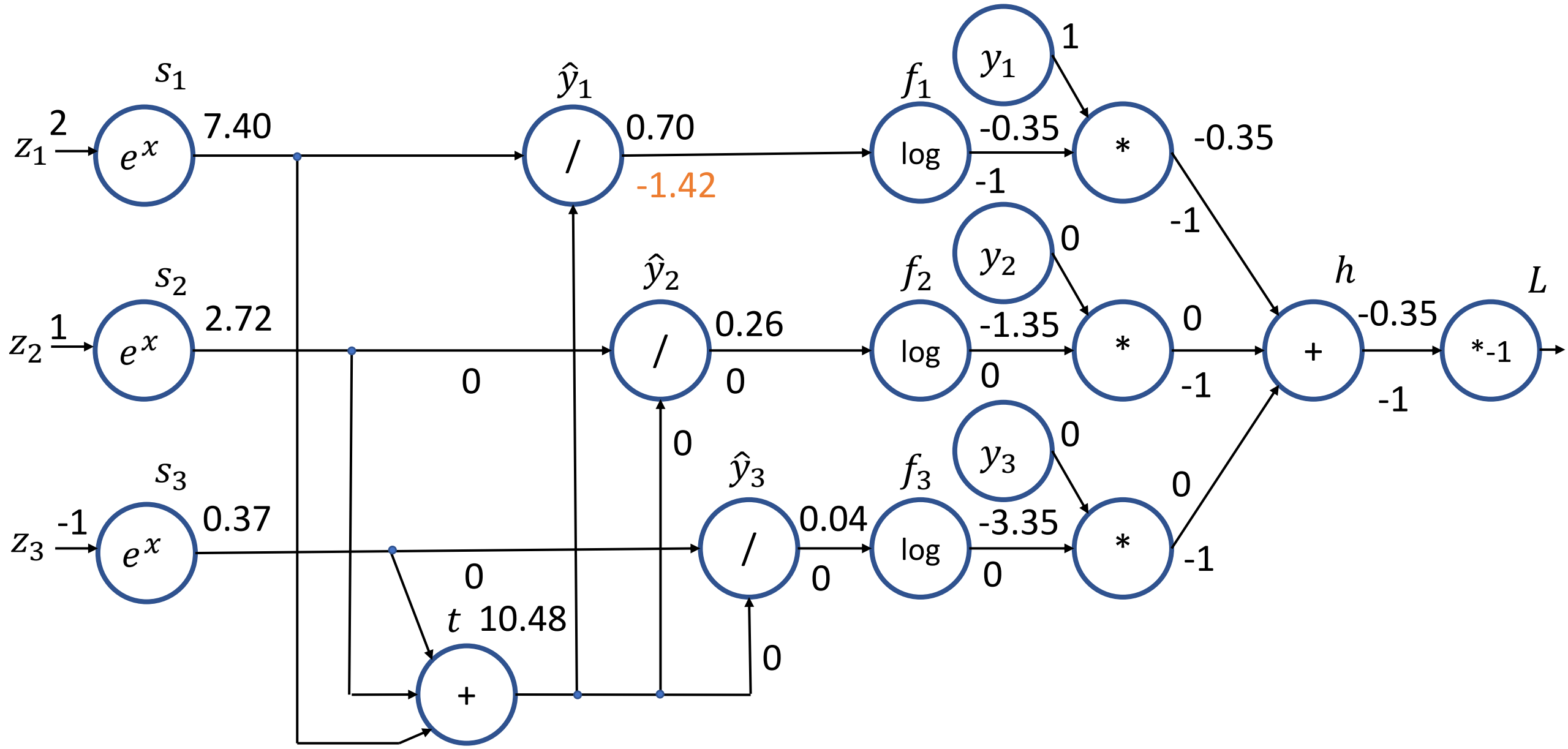
Key Observation: Once upstream gradient is 0,
All downstream gradients are also 0



$$\frac{\partial f_1}{\partial \hat{y}_1} = \frac{1}{\hat{y}_1}$$

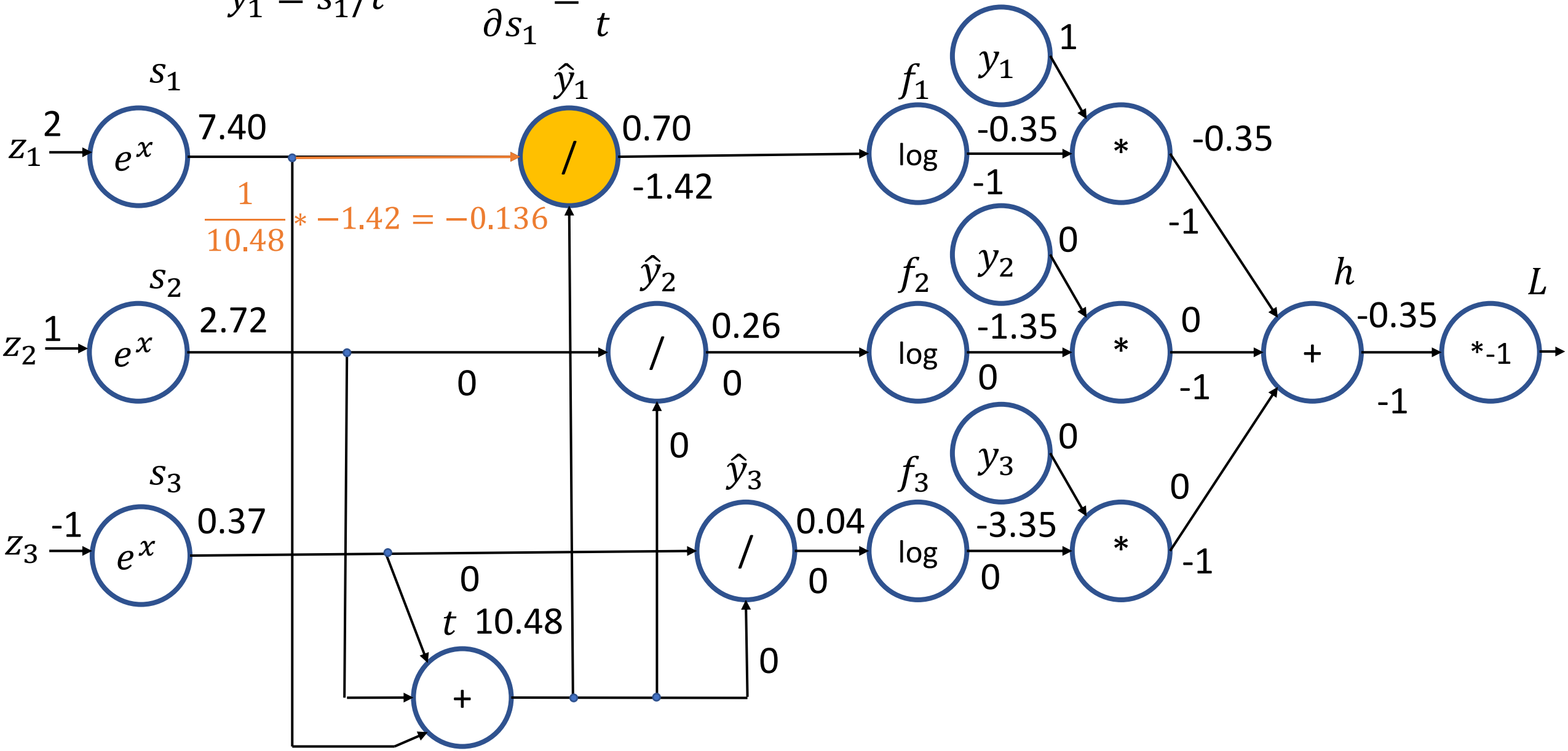


Brad Quinton, Scott Chin



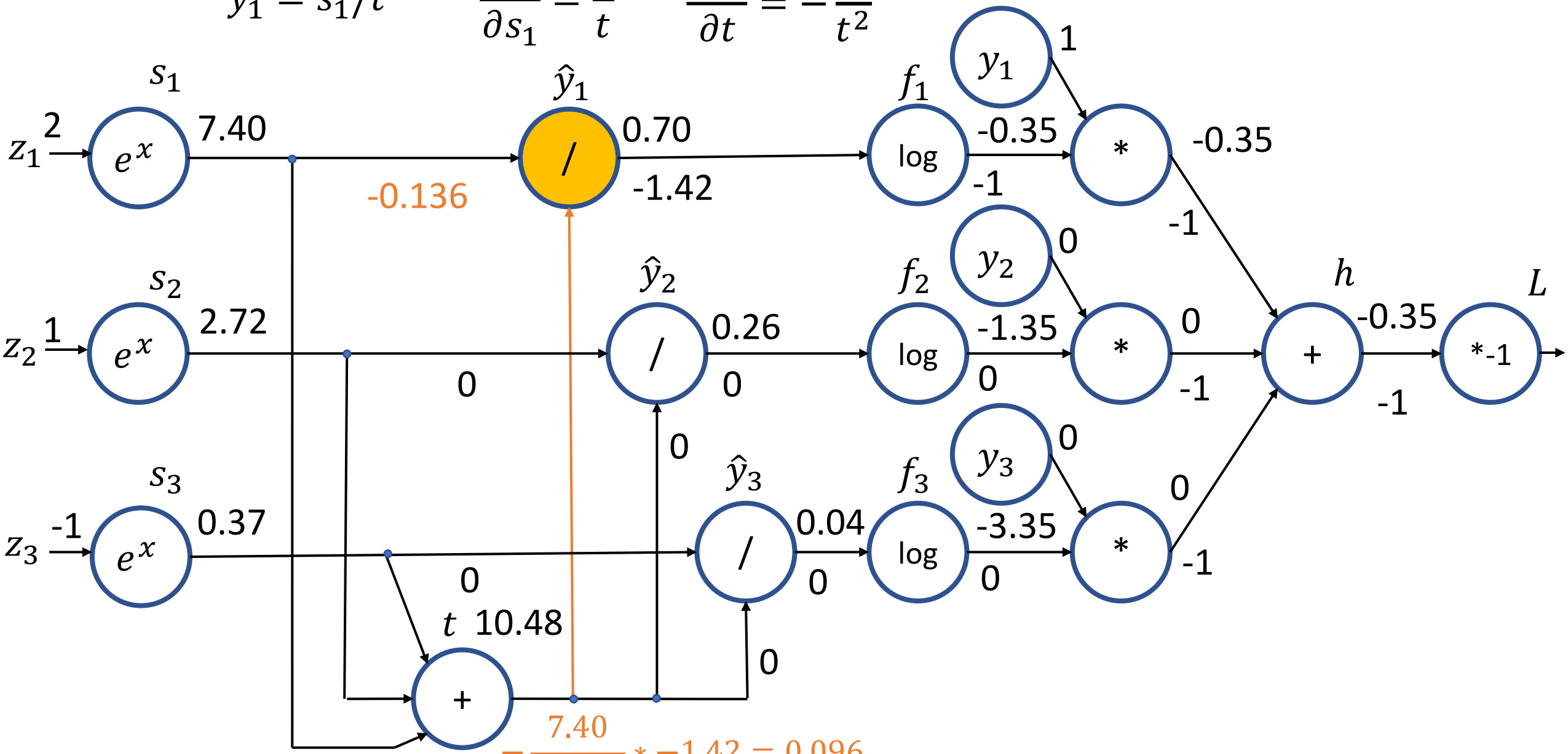
Brad Quinton, Scott Chin

$$\hat{y}_1 = s_1/t \qquad \frac{\partial \hat{y}_1}{\partial s_1} = \frac{1}{t}$$

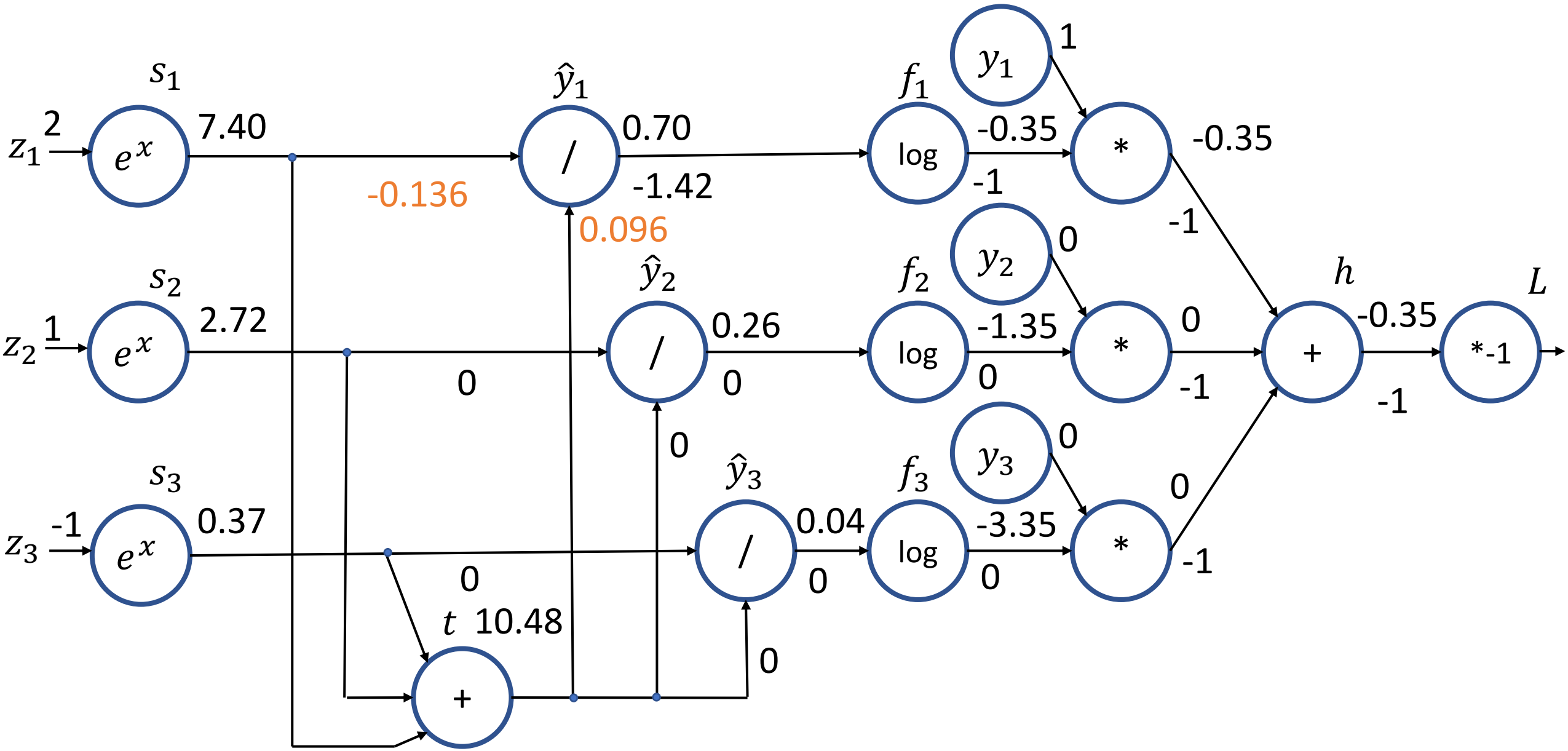


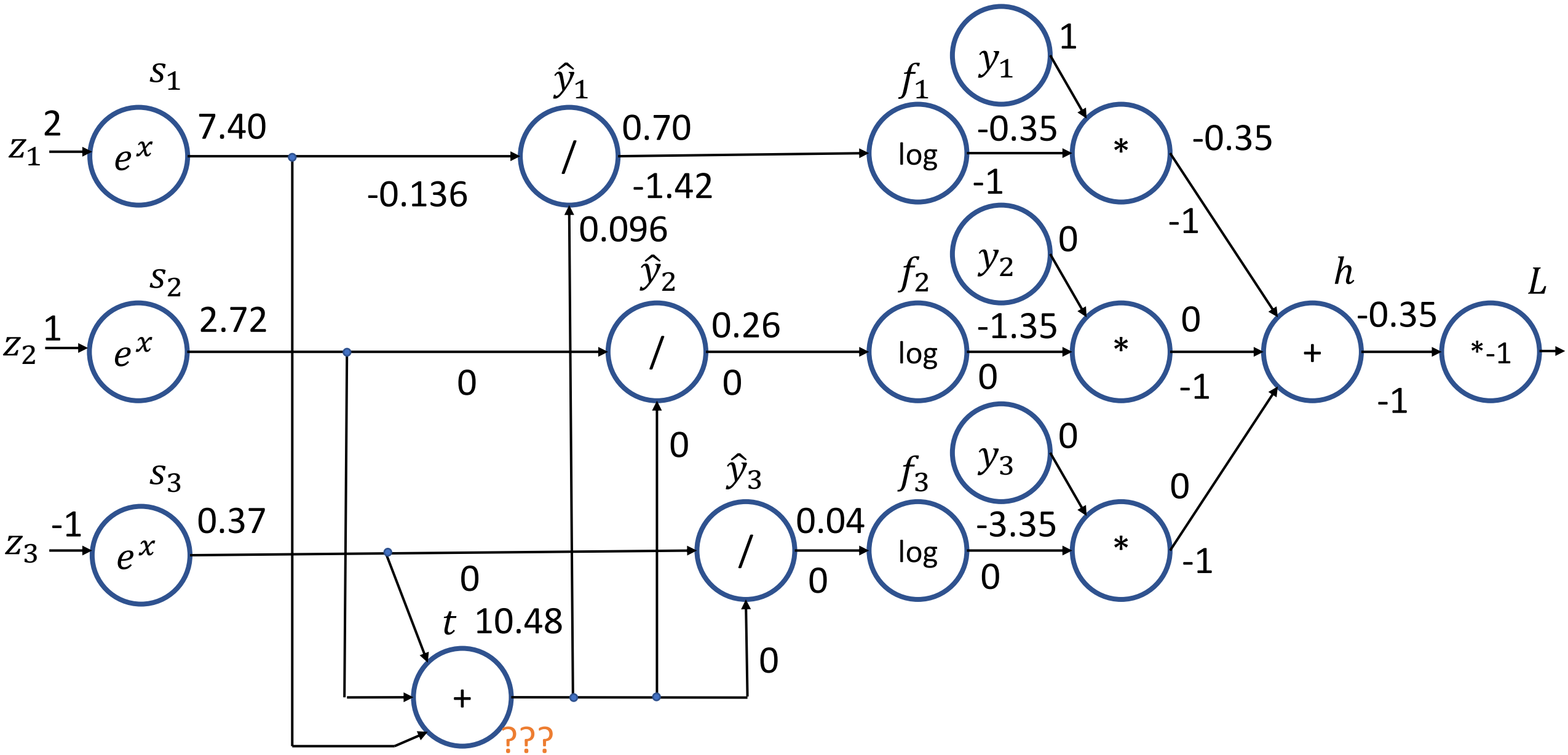
Brad Quinton, Scott Chin

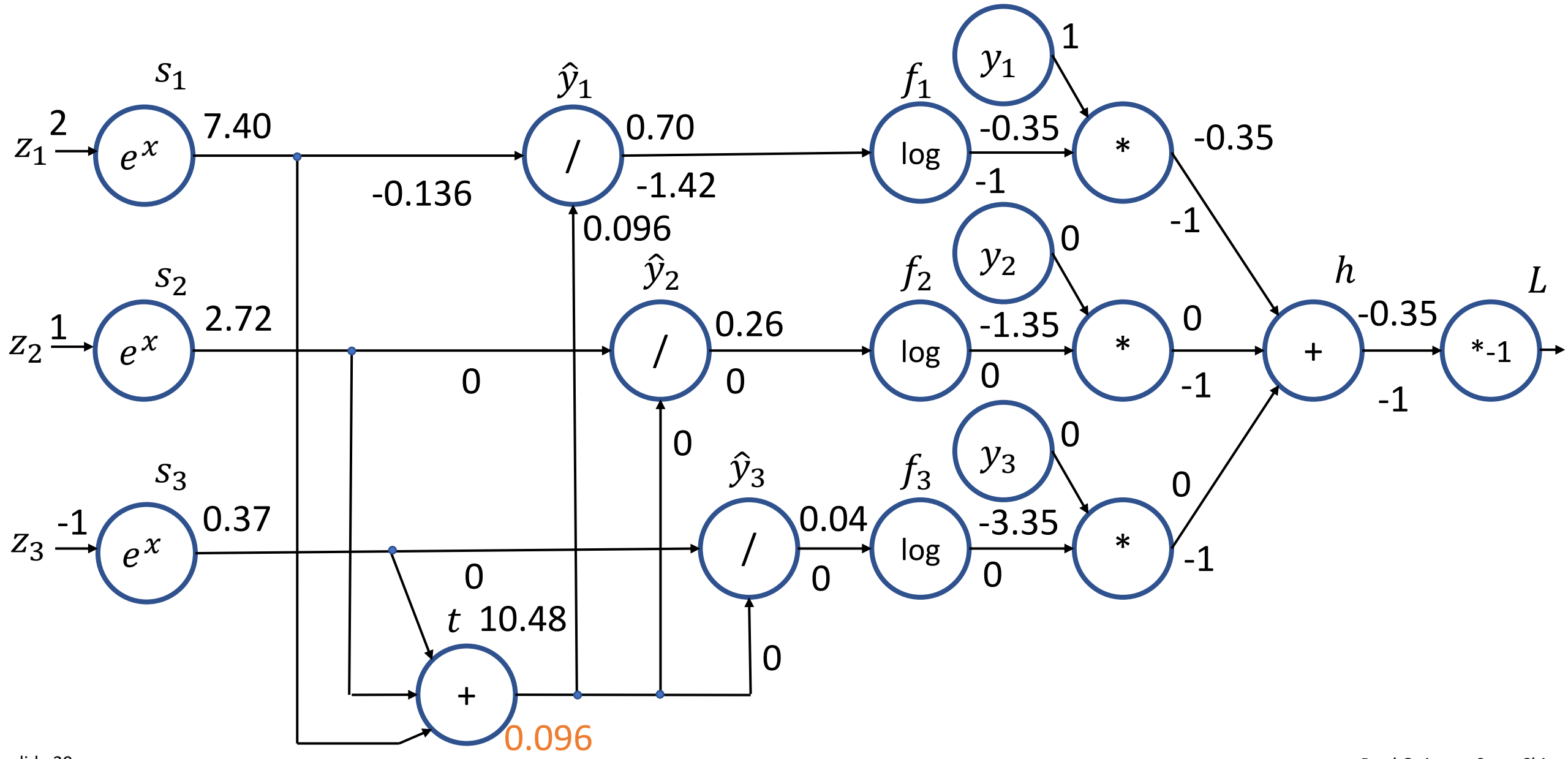
$$\hat{y}_1 = s_1/t \qquad \frac{\partial \hat{y}_1}{\partial s_1} = \frac{1}{t} \qquad \frac{\partial \hat{y}_1}{\partial t} = -\frac{s_1}{t^2}$$



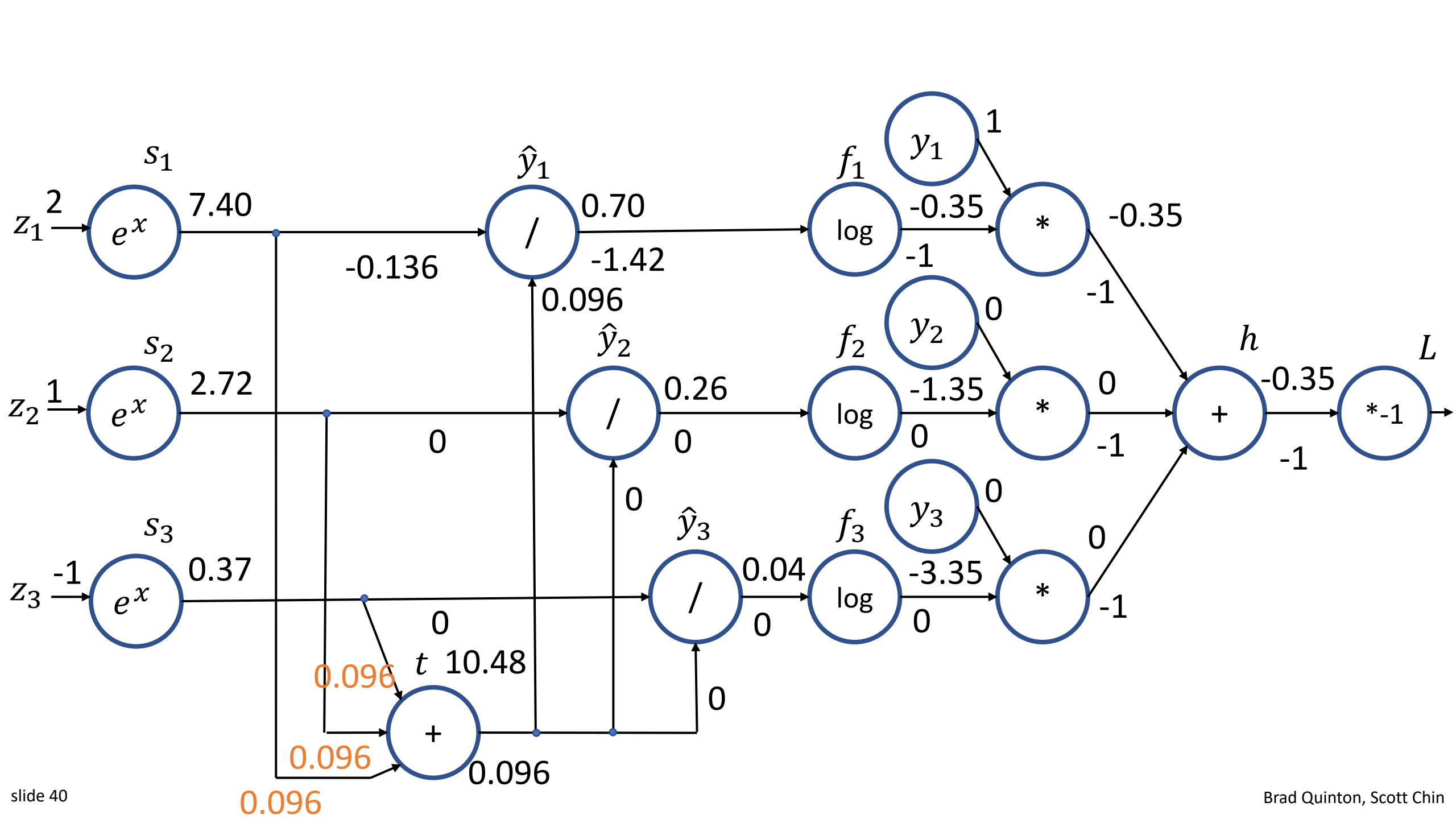
Brad Quinton, Scott Chin

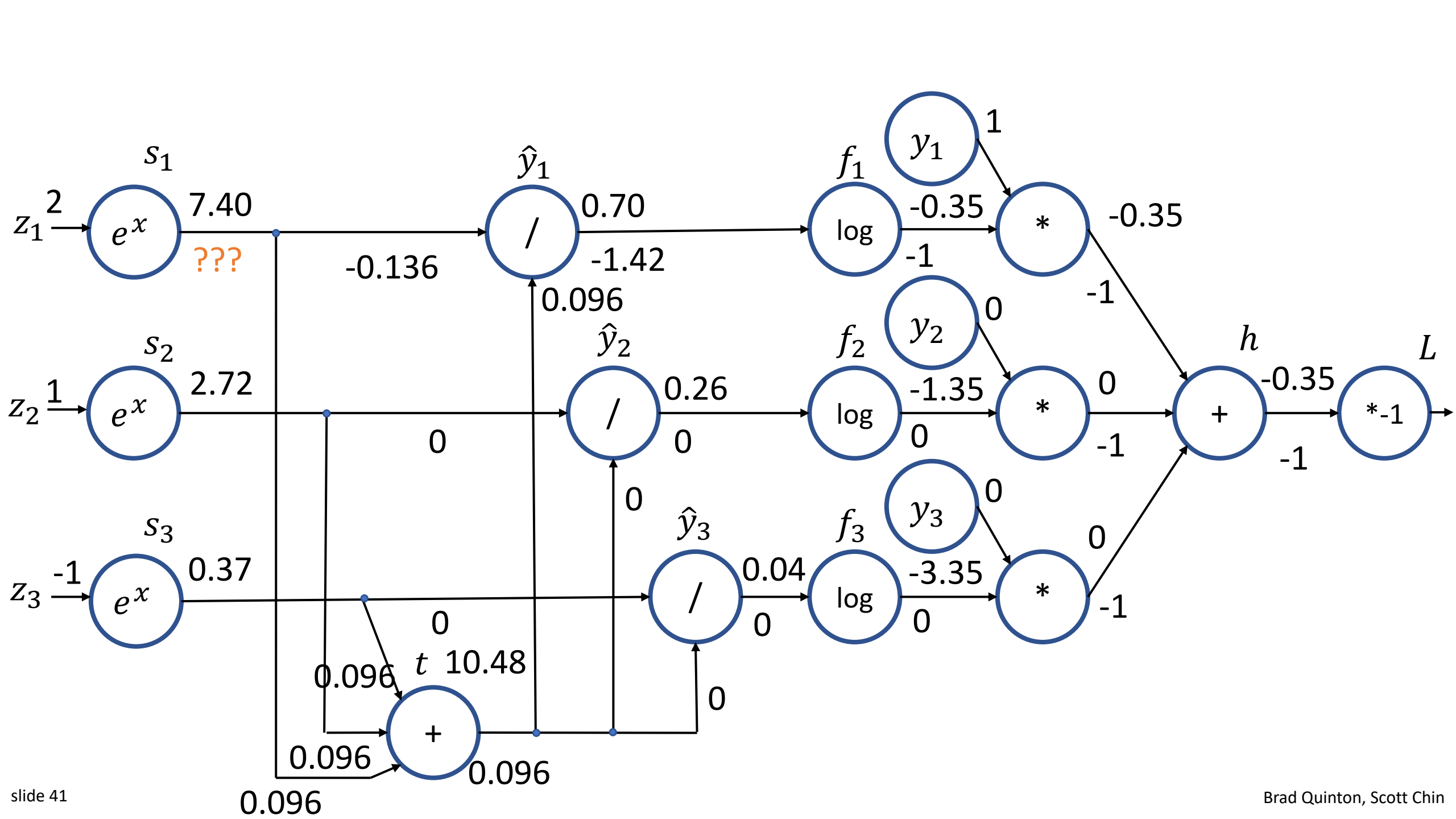

z_1
2
e^x
s_1
7.40
/
$\hat{y}_1$
0.70
-1.42
log
f_1
y_1
1
-0.35
-1
*
-0.35
-1
-0.136
0.096
z_2
1
e^x
s_2
2.72
/
$\hat{y}_2$
0.26
0
log
f_2
y_2
0
-1.35
0
*
0
-1
h
+
-0.35
-1
*-1
L
0
0
z_3
-1
e^x
s_3
0.37
/
$\hat{y}_3$
0.04
0
log
f_3
y_3
0
-3.35
0
*
0
-1
0
0
t 10.48
+
0


s_1
z_1
2
e^x
7.40
$\hat{y}_1$
/
0.70
-0.136
-1.42
0.096
f_1
log
y_1
1
-0.35
-1
*
-0.35
-1
s_2
z_2
1
e^x
2.72
$\hat{y}_2$
/
0.26
0
0
f_2
log
y_2
0
-1.35
0
*
0
-1
h
+
-0.35
-1
L
*-1
s_3
z_3
-1
e^x
0.37
$\hat{y}_3$
/
0.04
0
0
f_3
log
y_3
0
-3.35
0
*
0
-1
0
t 10.48
+
???

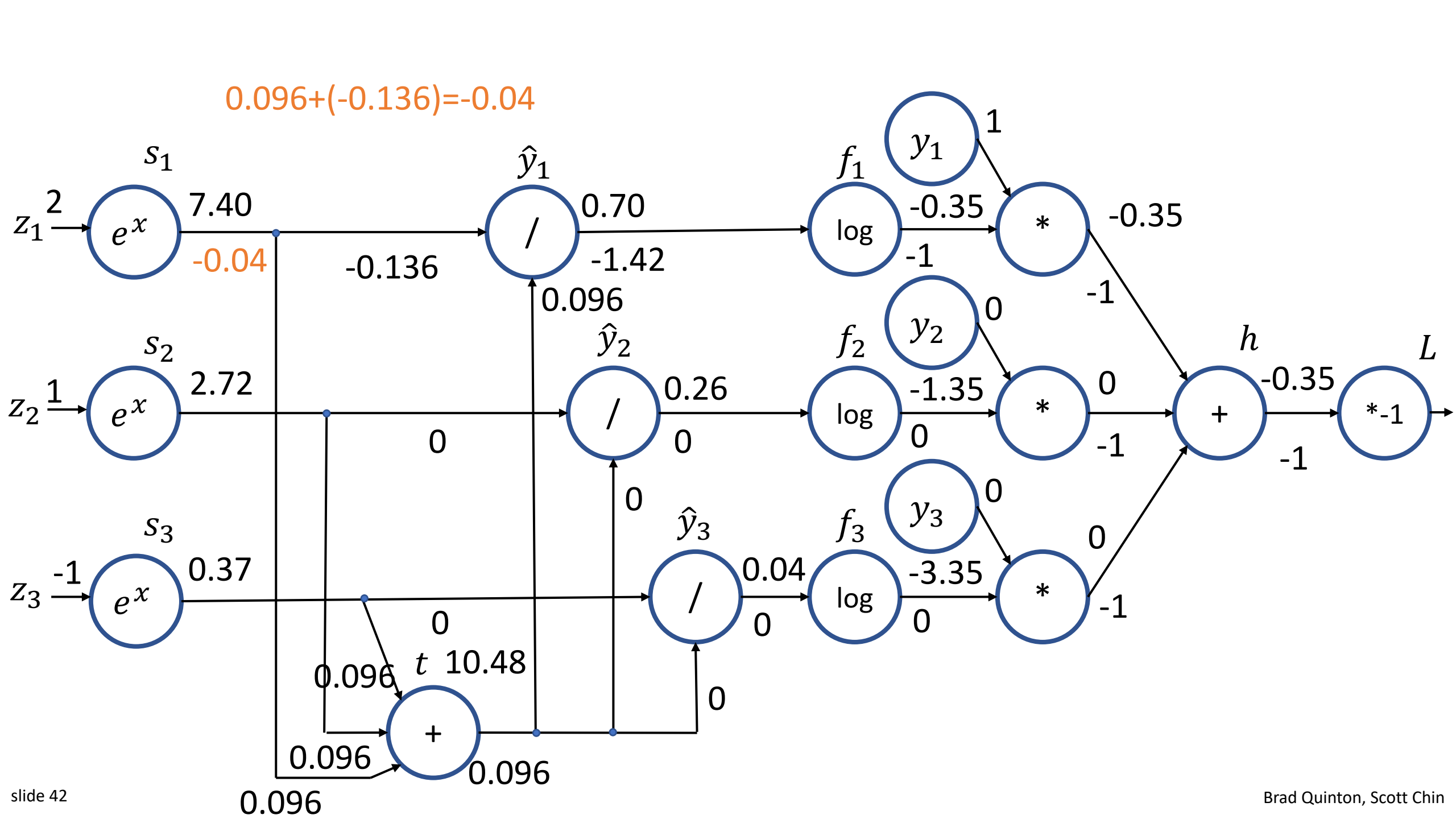


Brad Quinton, Scott Chin

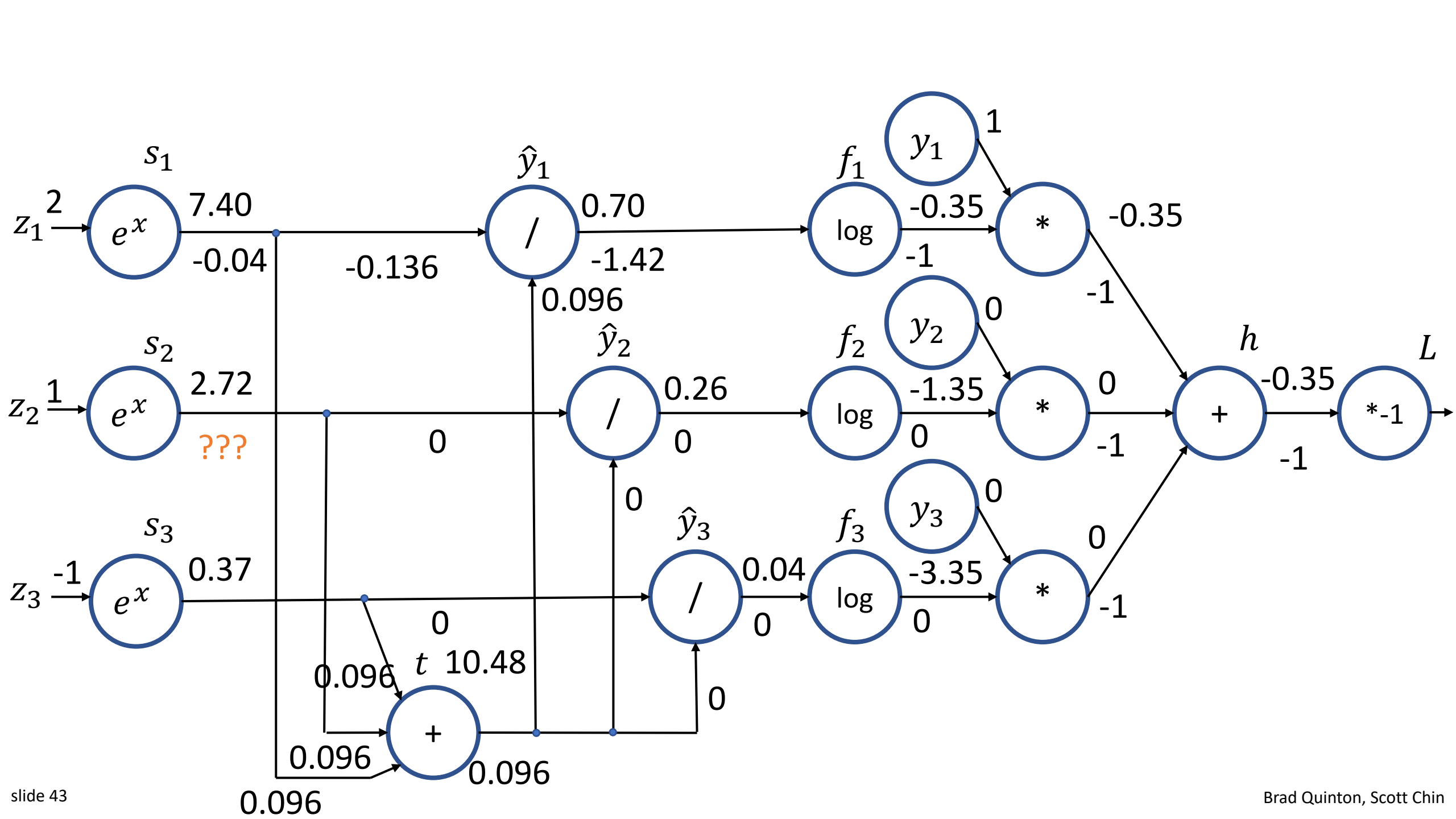

s_1
z_1
2
e^x
7.40
$\hat{y}_1$
/
0.70
-0.136
-1.42
0.096
f_1
log
y_1
1
-0.35
-1
*
-0.35
-1
s_2
z_2
1
e^x
2.72
$\hat{y}_2$
/
0.26
0
0
f_2
log
y_2
0
-1.35
0
*
0
-1
h
+
-0.35
-1
L
*-1
s_3
z_3
-1
e^x
0.37
$\hat{y}_3$
/
0.04
0
0
f_3
log
y_3
0
-3.35
0
*
0
-1
0
t 10.48
+
0.096
0.096
0.096
0.096



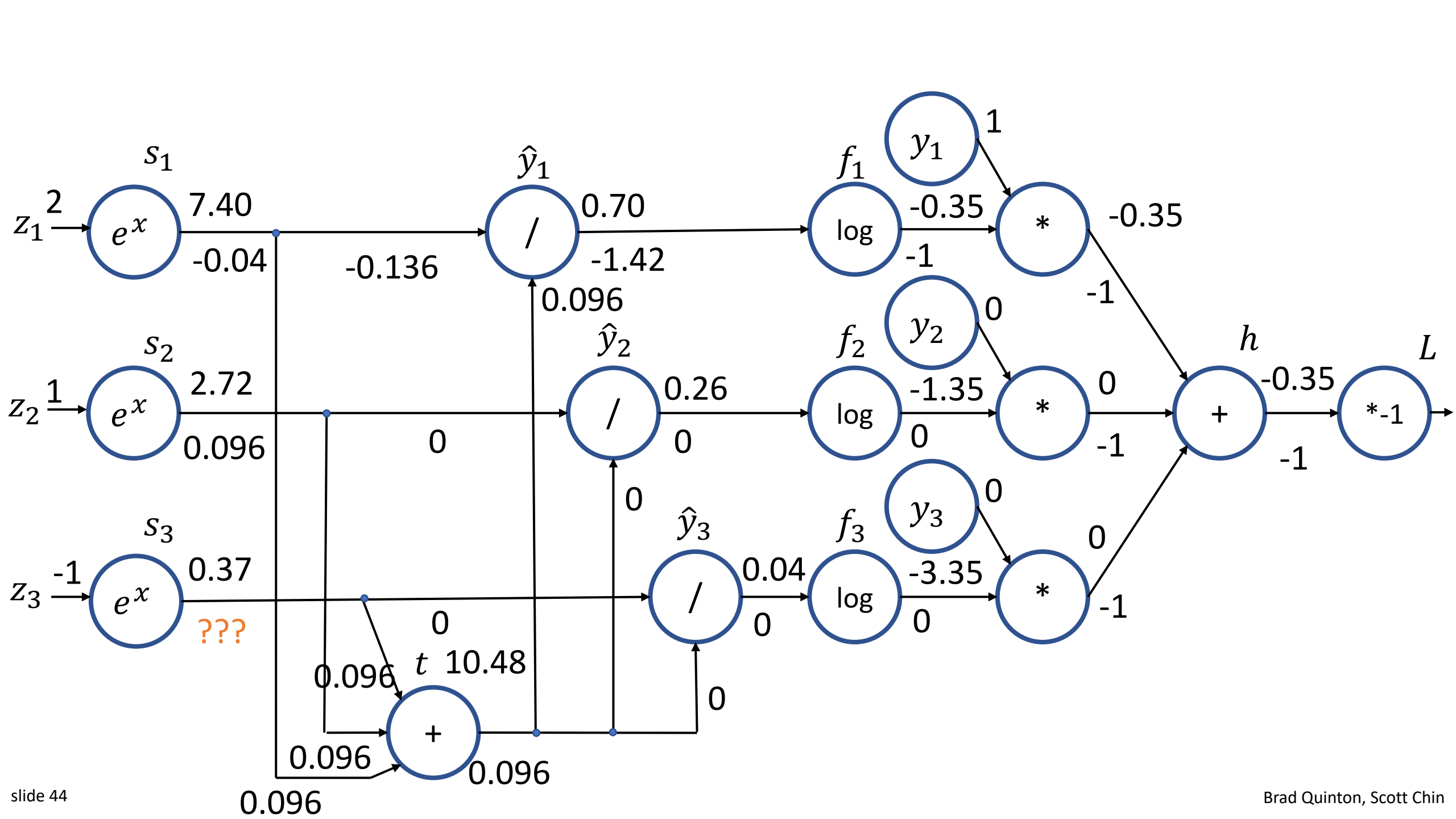
Brad Quinton, Scott Chin



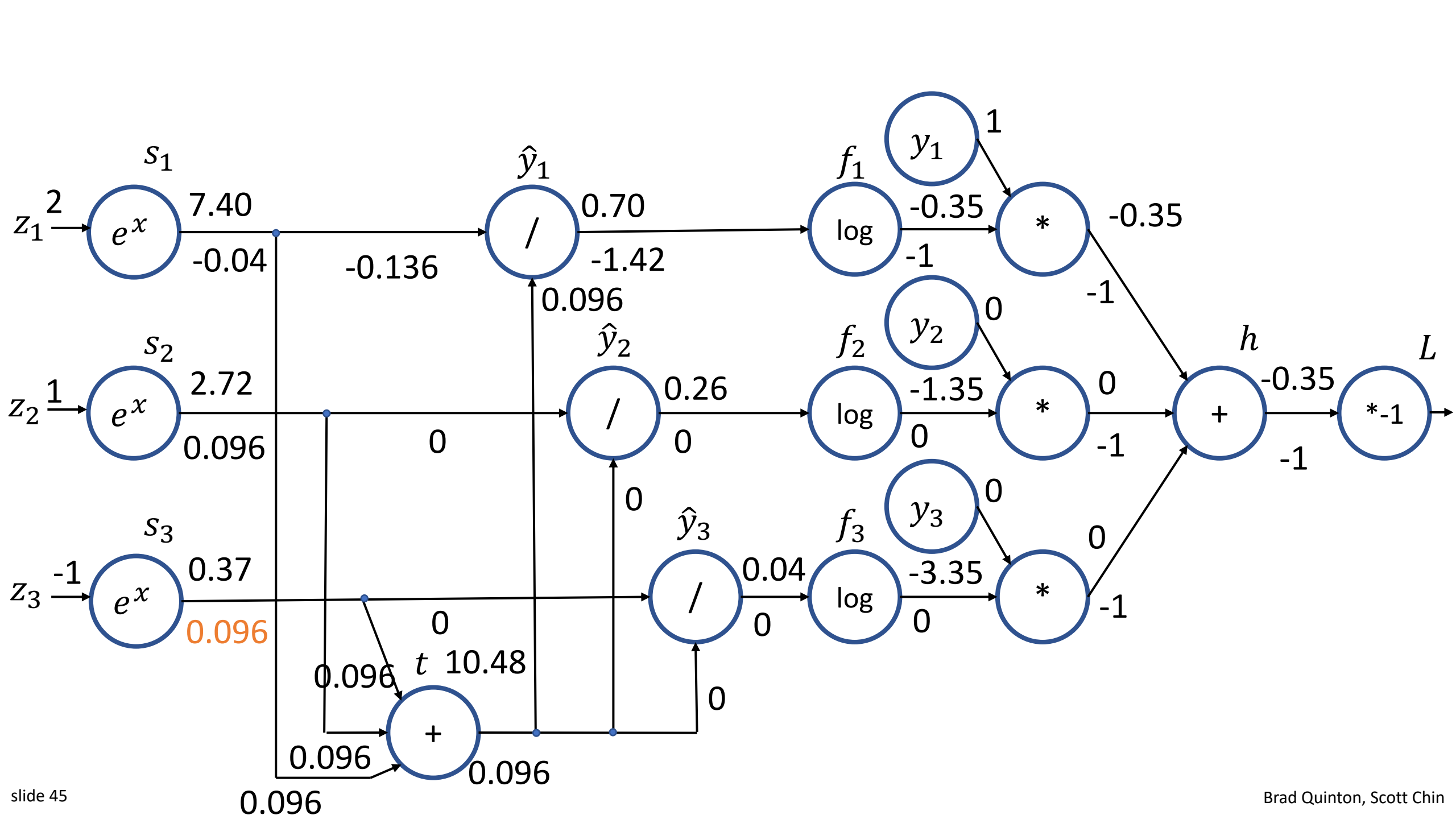
Brad Quinton, Scott Chin



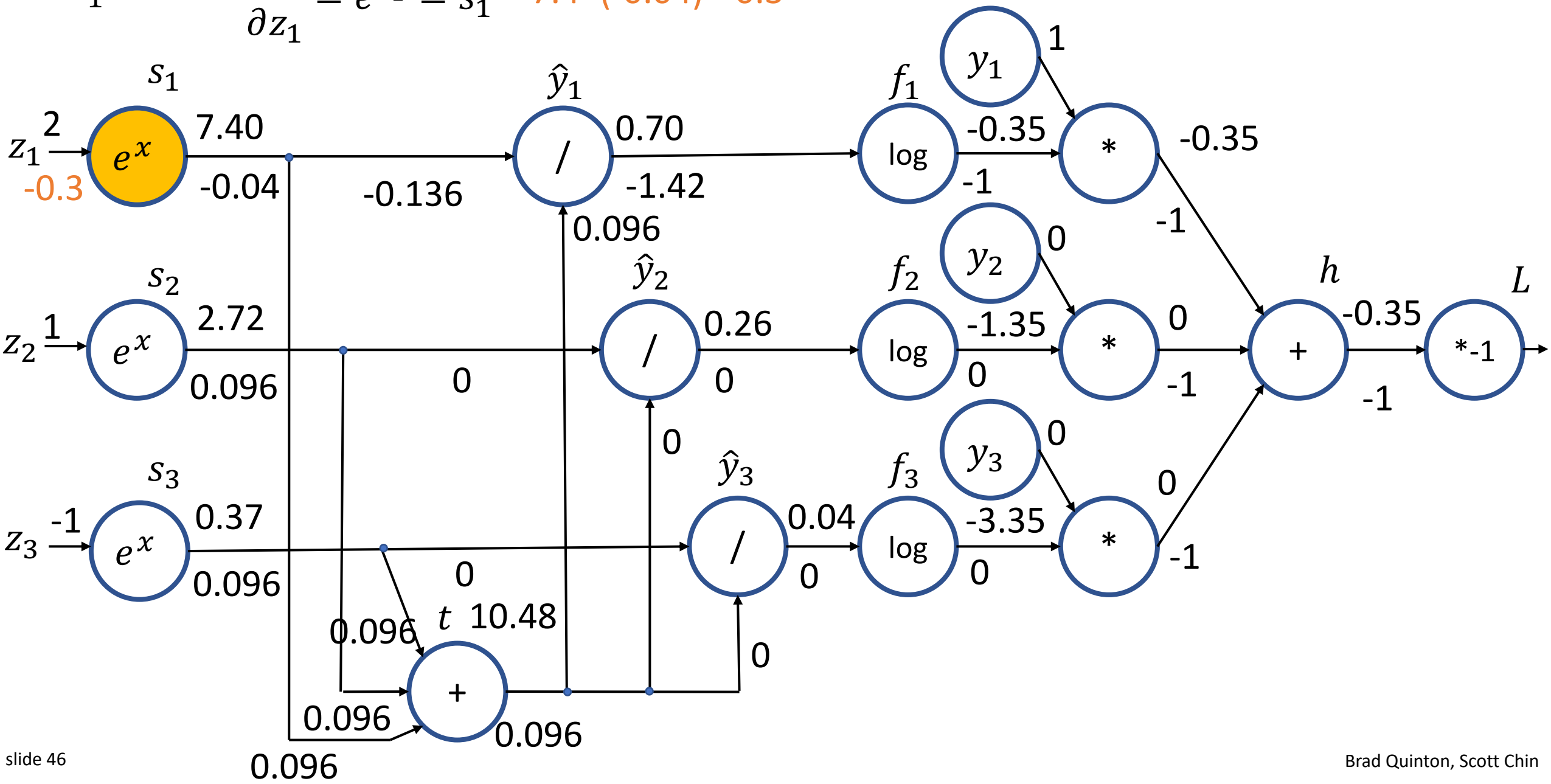
Brad Quinton, Scott Chin



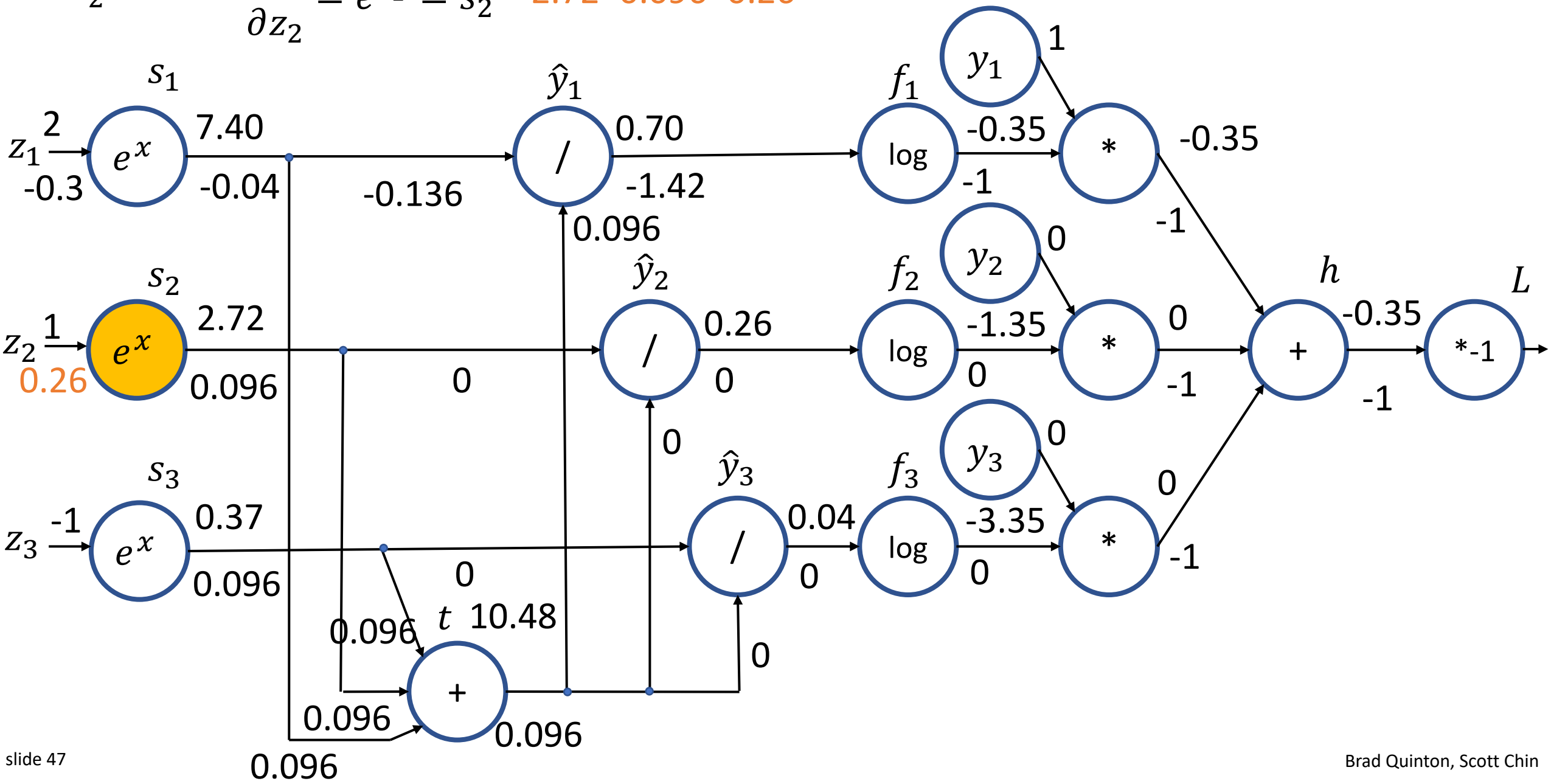
Brad Quinton, Scott Chin


s_1
s_2
s_3
z_1
z_2
z_3
2
1
-1
e^x
7.40
2.72
0.37
-0.04
0.096
0.096
-0.136
0
0
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
/
0.70
0.26
0.04
-1.42
0
0
0.096
0
0
f_1
f_2
f_3
log
y_1
y_2
y_3
1
0
0
-0.35
-1.35
-3.35
-1
0
0
*
-0.35
0
0
-1
-1
-1
h
+
-0.35
-1
L
*-1
t 10.48
+
0.096
0.096
0.096
0.096
0
0
0.096
slide 45
Brad Quinton, Scott Chin

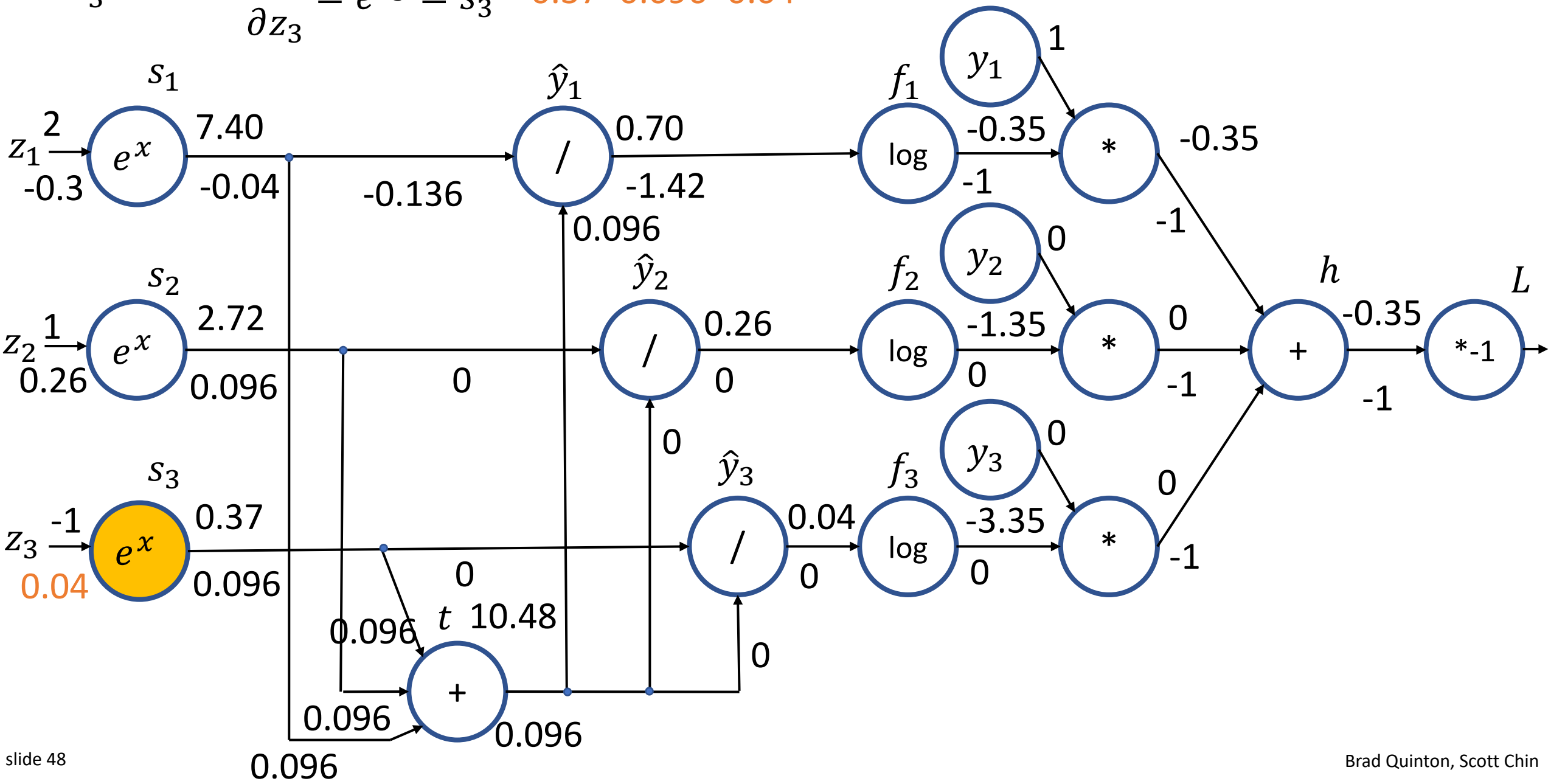
$s_1 = e^{z_1}$ $\frac{\partial s_1}{\partial z_1} = e^{z_1} = s_1$ 7.4*(-0.04)=-0.3



$s_2 = e^{z_2}$ $\frac{\partial s_2}{\partial z_2} = e^{z_2} = s_2$ 2.72*0.096=0.26

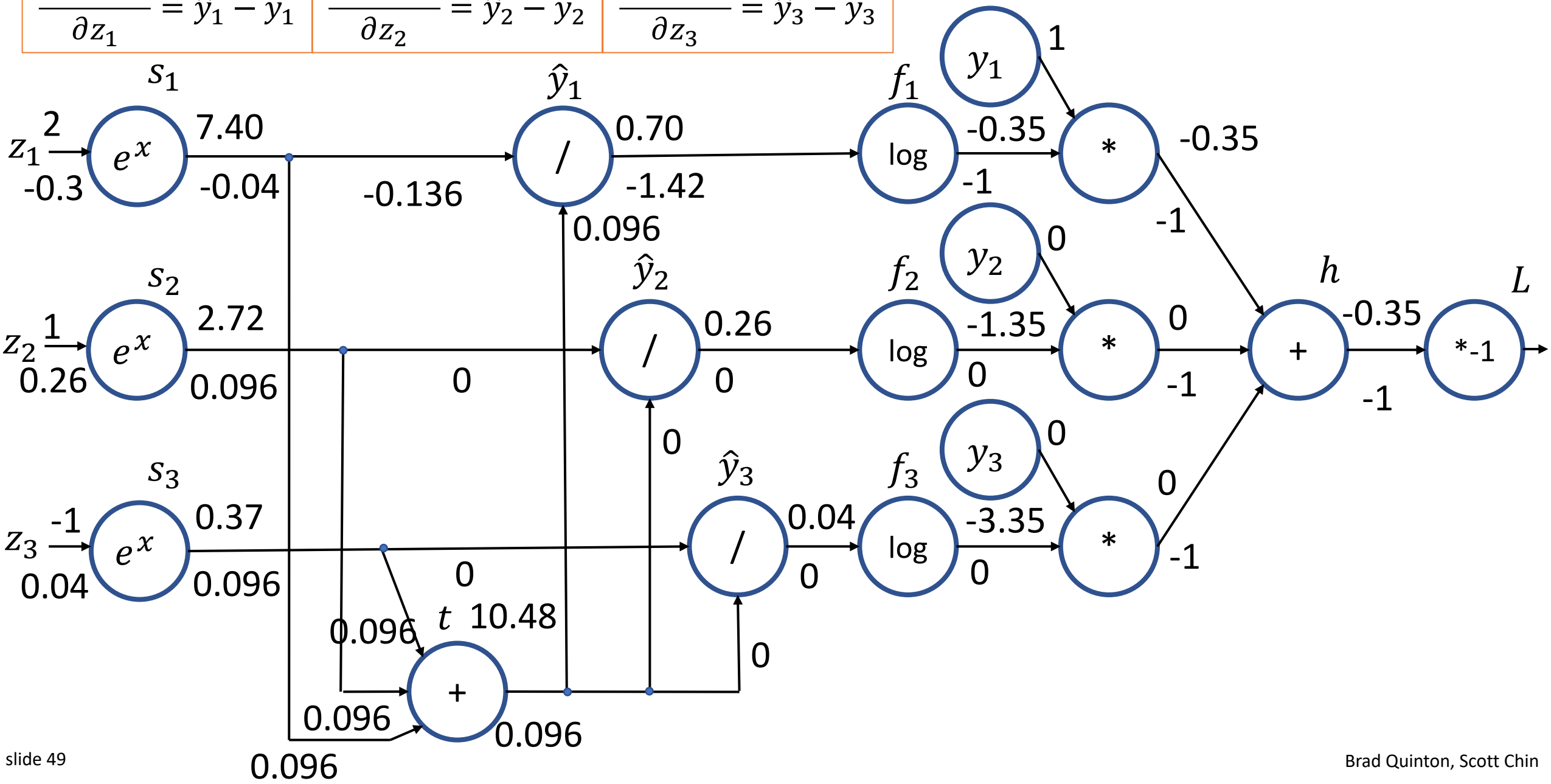


$s_3 = e^{z_3}$ $\frac{\partial s_3}{\partial z_3} = e^{z_3} = s_3$ 0.37*0.096=0.04



Check: Analytical Formula from earlier lecture

$\frac{\partial L(\hat{y}, y)}{\partial z_1} = \hat{y}_1 - y_1$	$\frac{\partial L(\hat{y}, y)}{\partial z_2} = \hat{y}_2 - y_2$	$\frac{\partial L(\hat{y}, y)}{\partial z_3} = \hat{y}_3 - y_3$

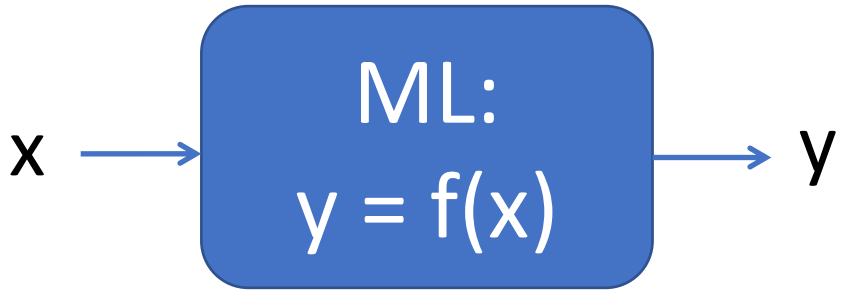


So far …

- Backpropagation sends a “signal” back throughout the network telling us how to change each parameter
- But backpropagation doesn’t magically make any neural network architectures trainable. You need to know how it can go wrong.
- Understanding the “dangers” that backprop can present will help you debug your deep learning systems
- So how can things go wrong?
- Let’s start with the things related to activation functions

Activation Functions

- Recall that we need nonlinear activation functions to learn complex non-linear mappings from input to output of the model
- Any nonlinear functions would satisfy this requirement so why are some better than others?
- Understanding backpropagation lets us answer this question



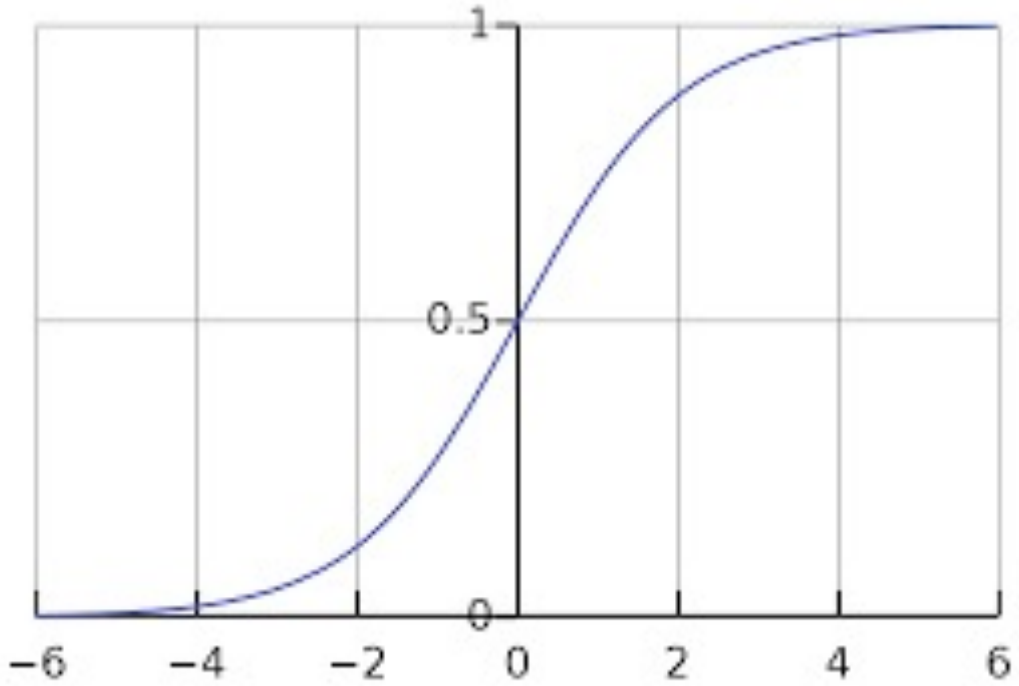
Activation Functions

- Sigmoid
- tanh
- ReLU
- ReLU variants

Sigmoid Activation Function

- Maps input to values between 0 and 1
- Inspired by probability theory

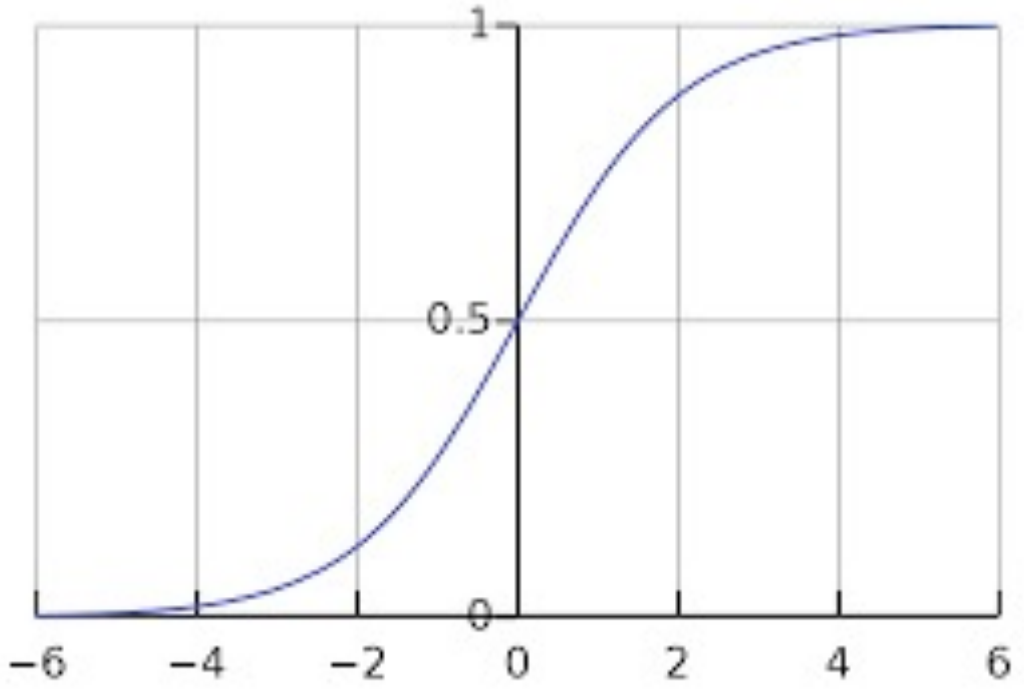
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Sigmoid Activation Function

- Maps input to values between 0 and 1
- Inspired by probability theory

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

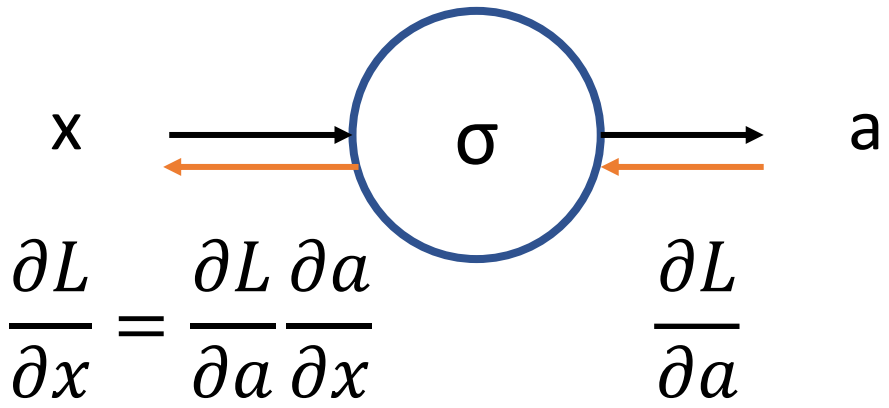


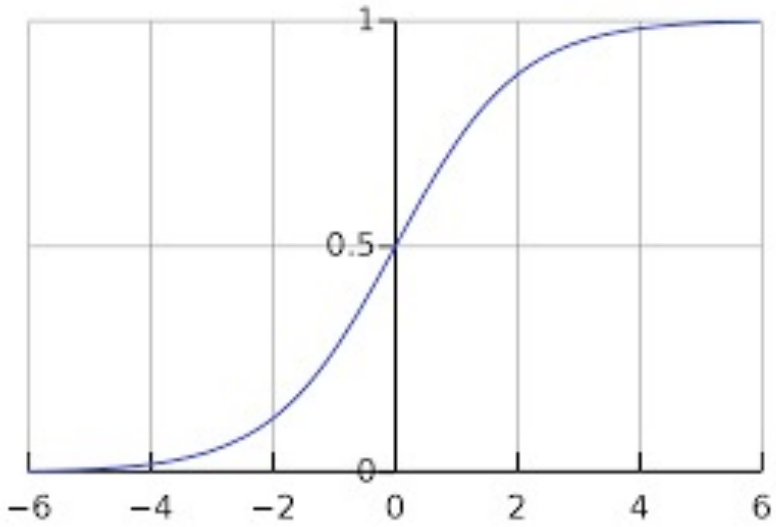
Biggest Problem with this function:

- Think about the derivative and backpropagation

Backprop through Sigmoid Function

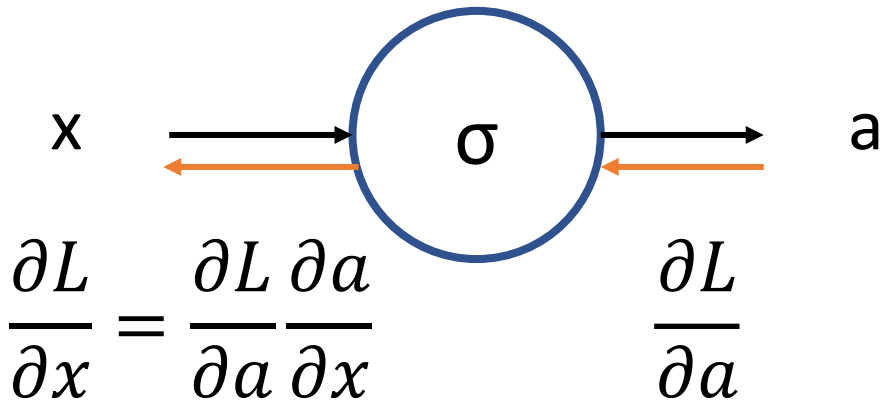
- What happens during backprop when x <-6 or x > 6?

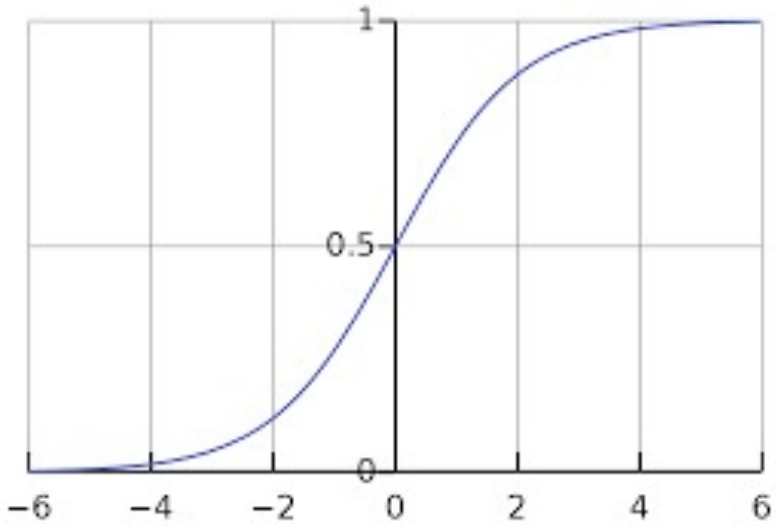




Backprop through Sigmoid Function

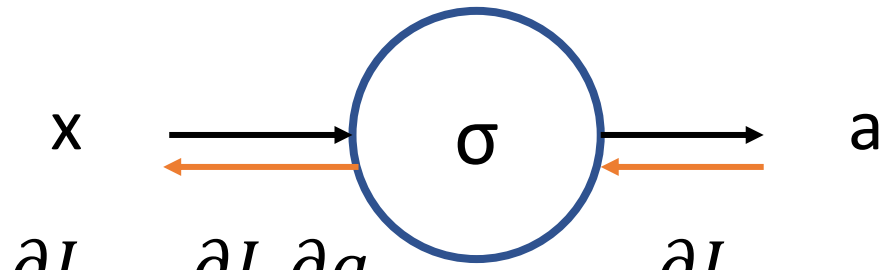
- What happens during backprop when x <-6 or x > 6?
- Local gradient $\frac{\partial a}{\partial x}$ is practically 0. How does that affect backprop?





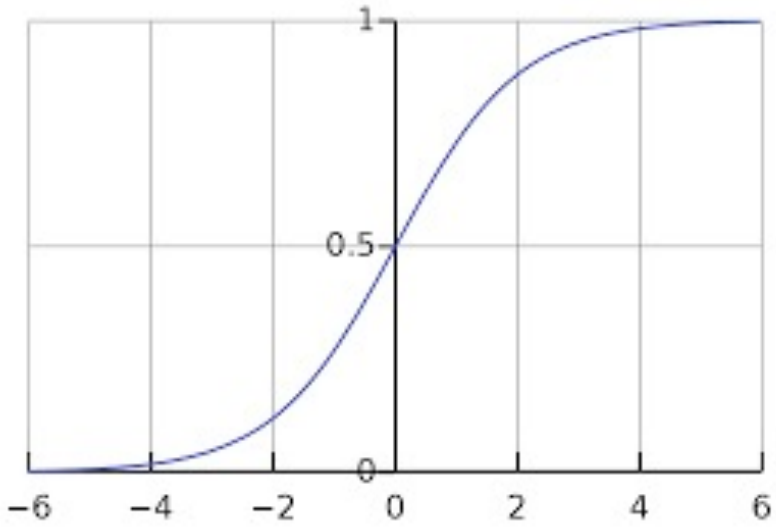
Backprop through Sigmoid Function

- What happens during backprop when x <-6 or x > 6?
- Local gradient $\frac{\partial a}{\partial x}$ is practically 0. How does that affect backprop?
- Multiplication via chain-rule will basically make $\frac{\partial L}{\partial x}$ approach 0



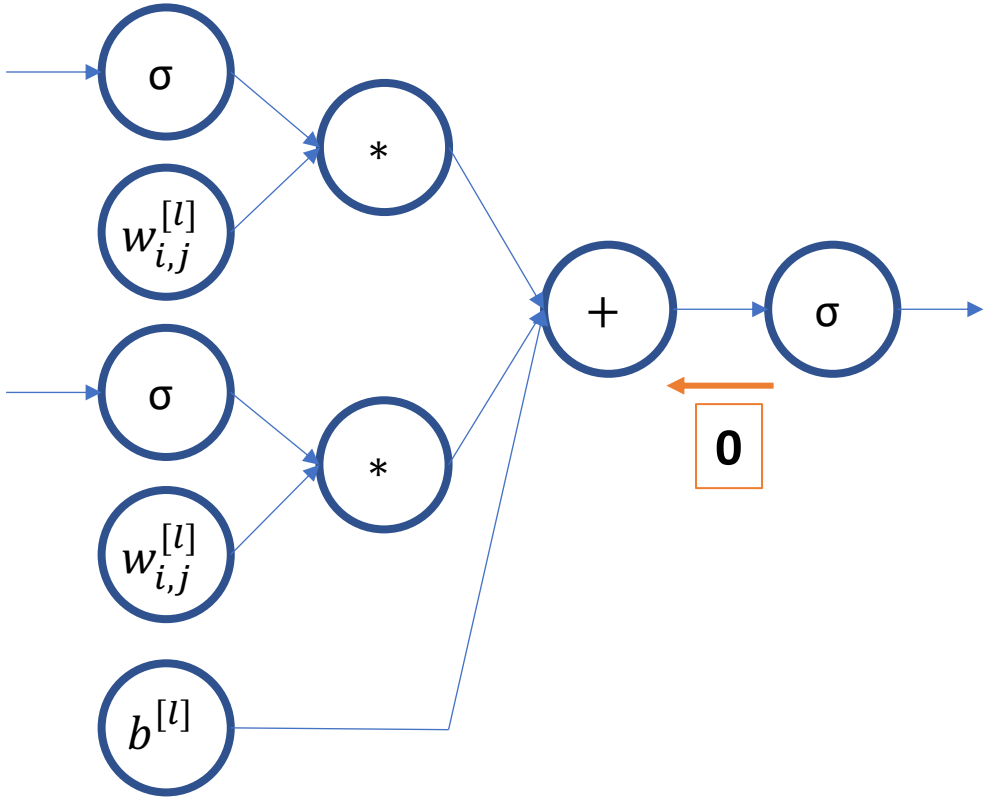
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial x}$$

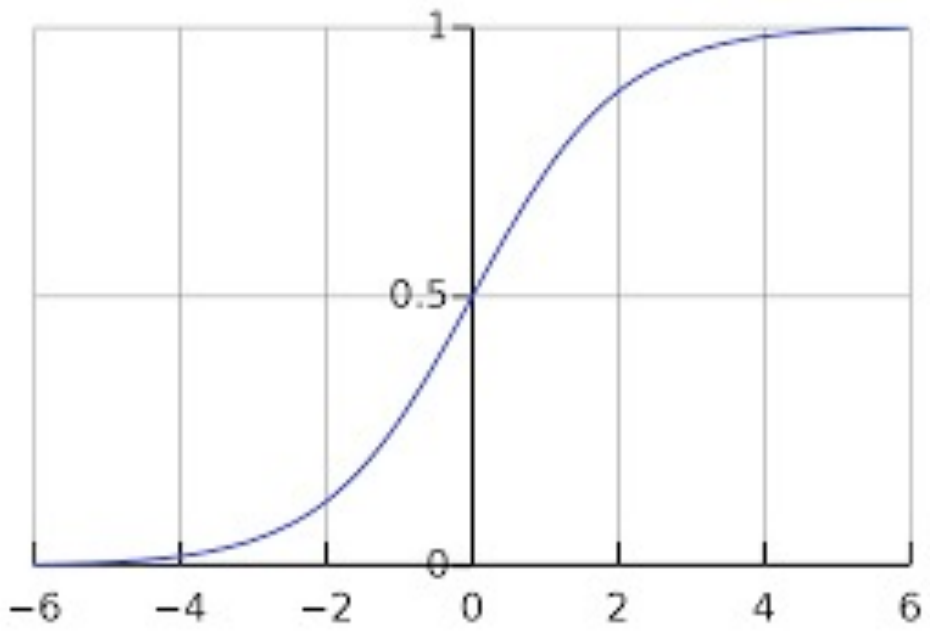
$$\frac{\partial L}{\partial a}$$



Saturated Neuron

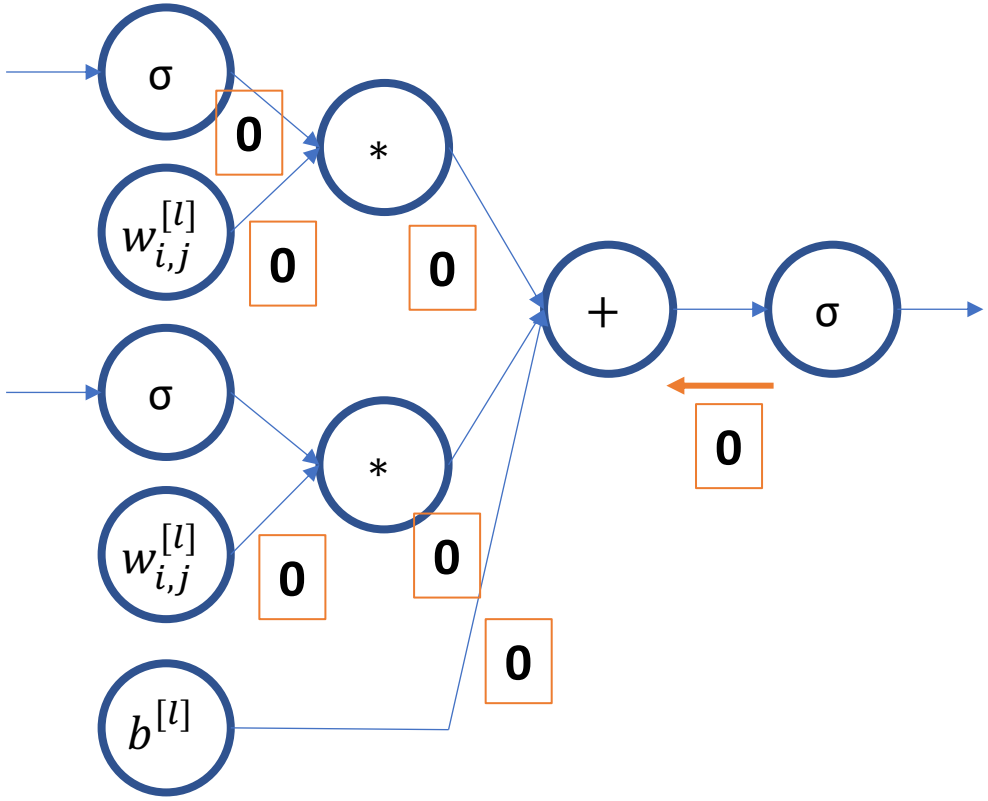
- When in saturated region, we call this a Saturated Neuron
- Active (Unsaturated) region is small

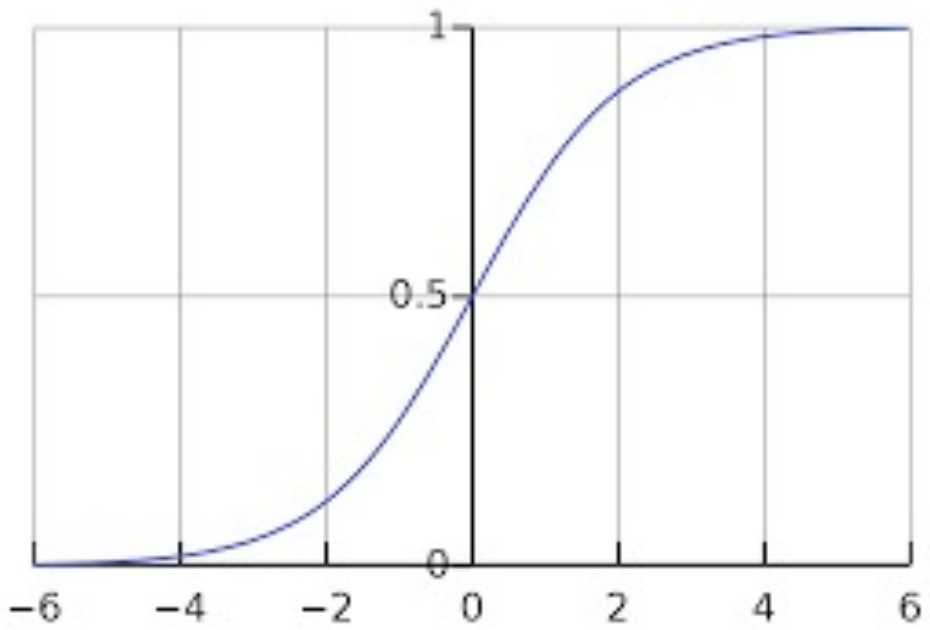




Saturated Neuron

- When in saturated region, we call this a Saturated Neuron
- Active (Unsaturated) region is small





Vanishing Gradients

- What happens during Gradient Descent update when the gradient is very small or 0?

$$w_{i,j}^{[l]} = w_{i,j}^{[l]} - \propto \frac{\partial L}{\partial w_{i,j}^{[l]}}$$

Vanishing Gradients

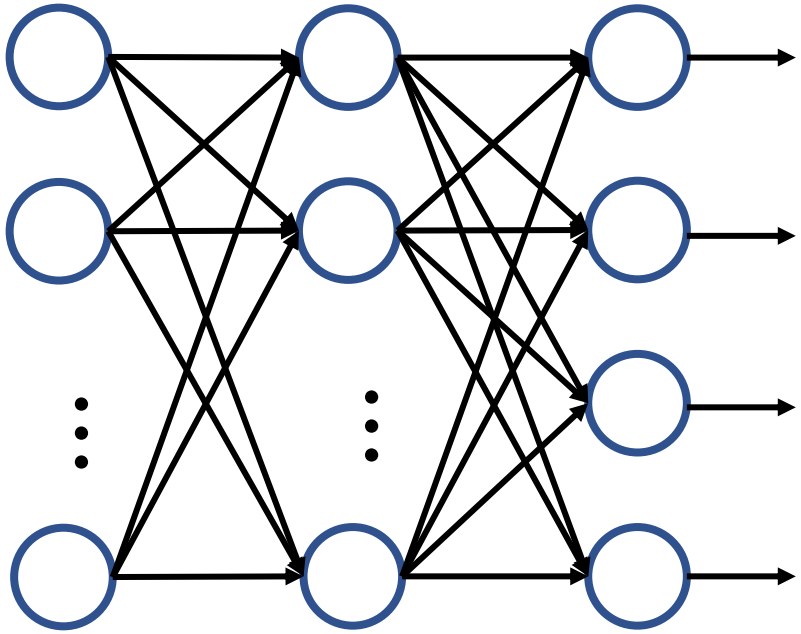
- What happens during Gradient Descent update when the gradient is very small or 0?

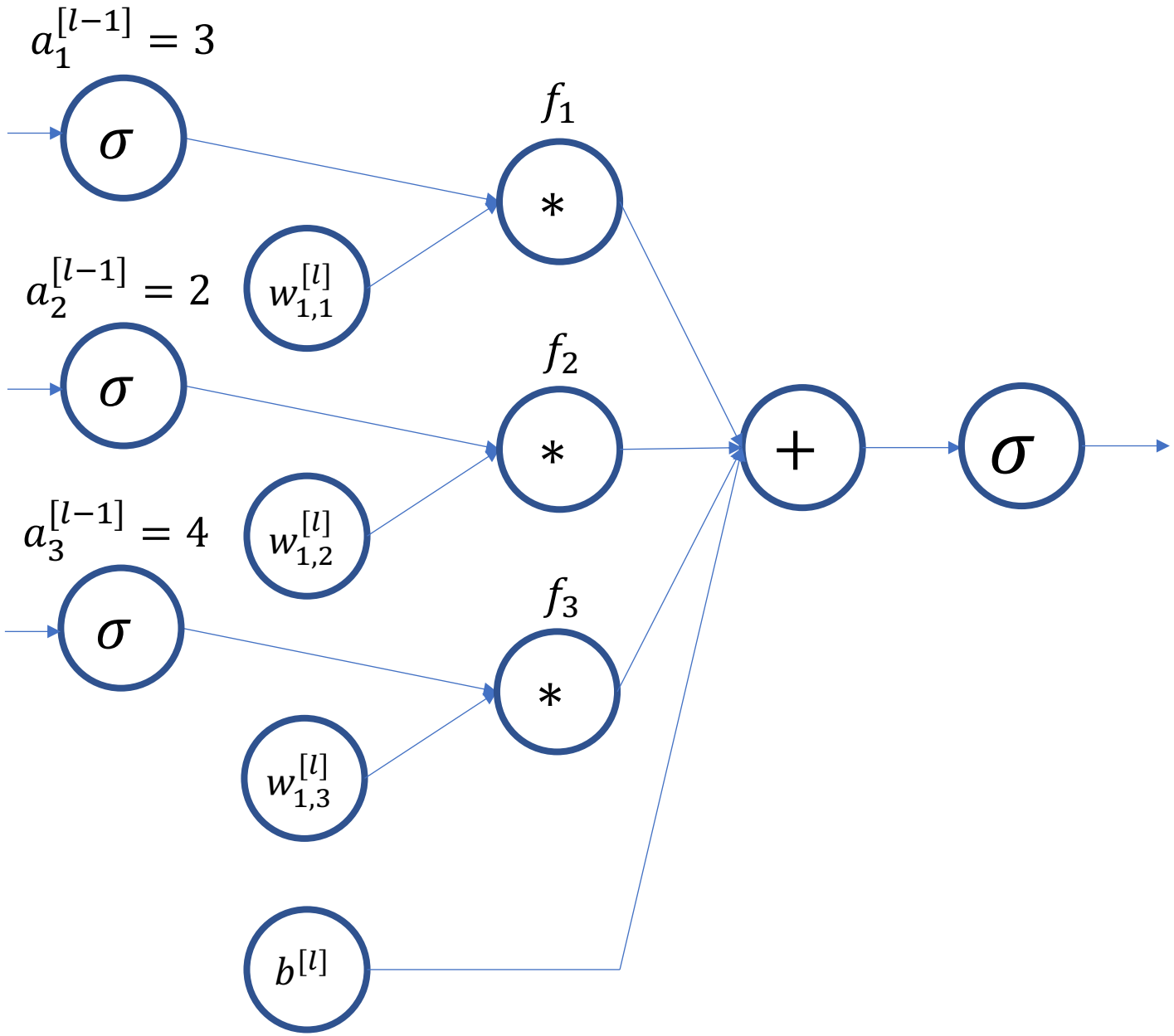
$$w_{i,j}^{[l]} = w_{i,j}^{[l]} - \propto \frac{\partial L}{\partial w_{i,j}^{[l]}}$$

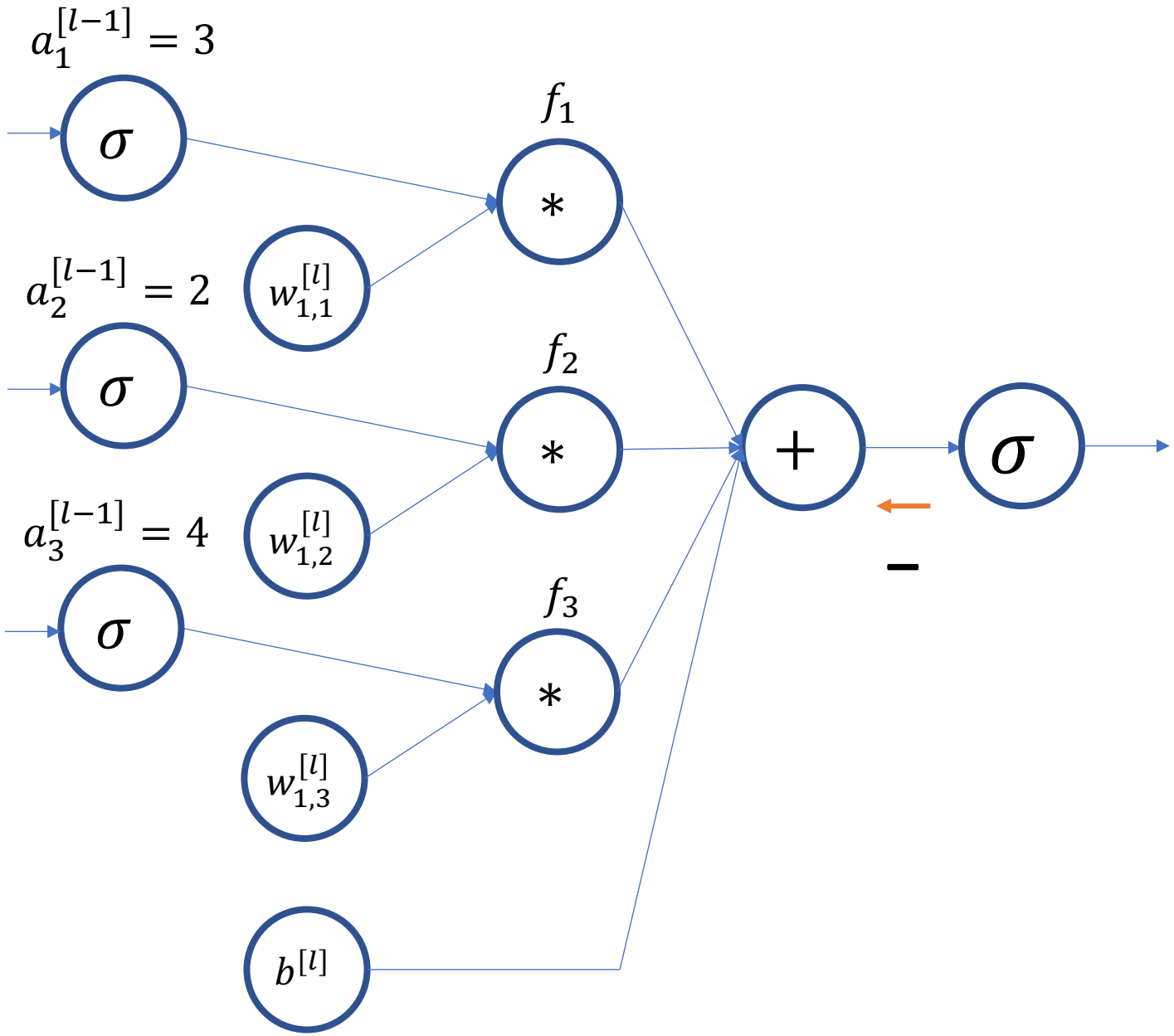
- When small, learning will be slooooooow
  - Parameters will change extremely slowly.
  - Once a Sigmoid Neuron is in saturation, very hard for training to update the neuron's weights to improve the model

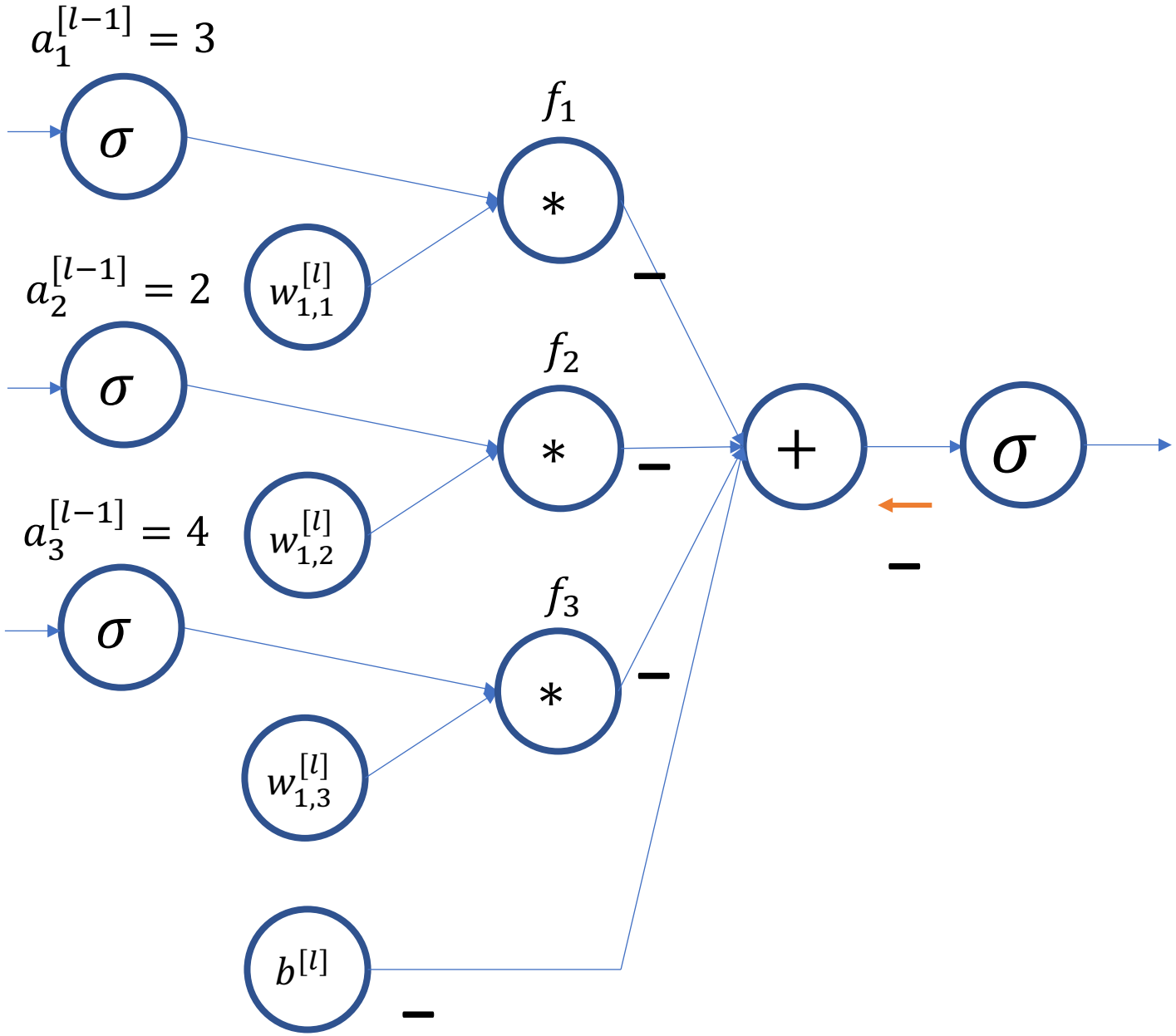
Sigmoid – Always positive

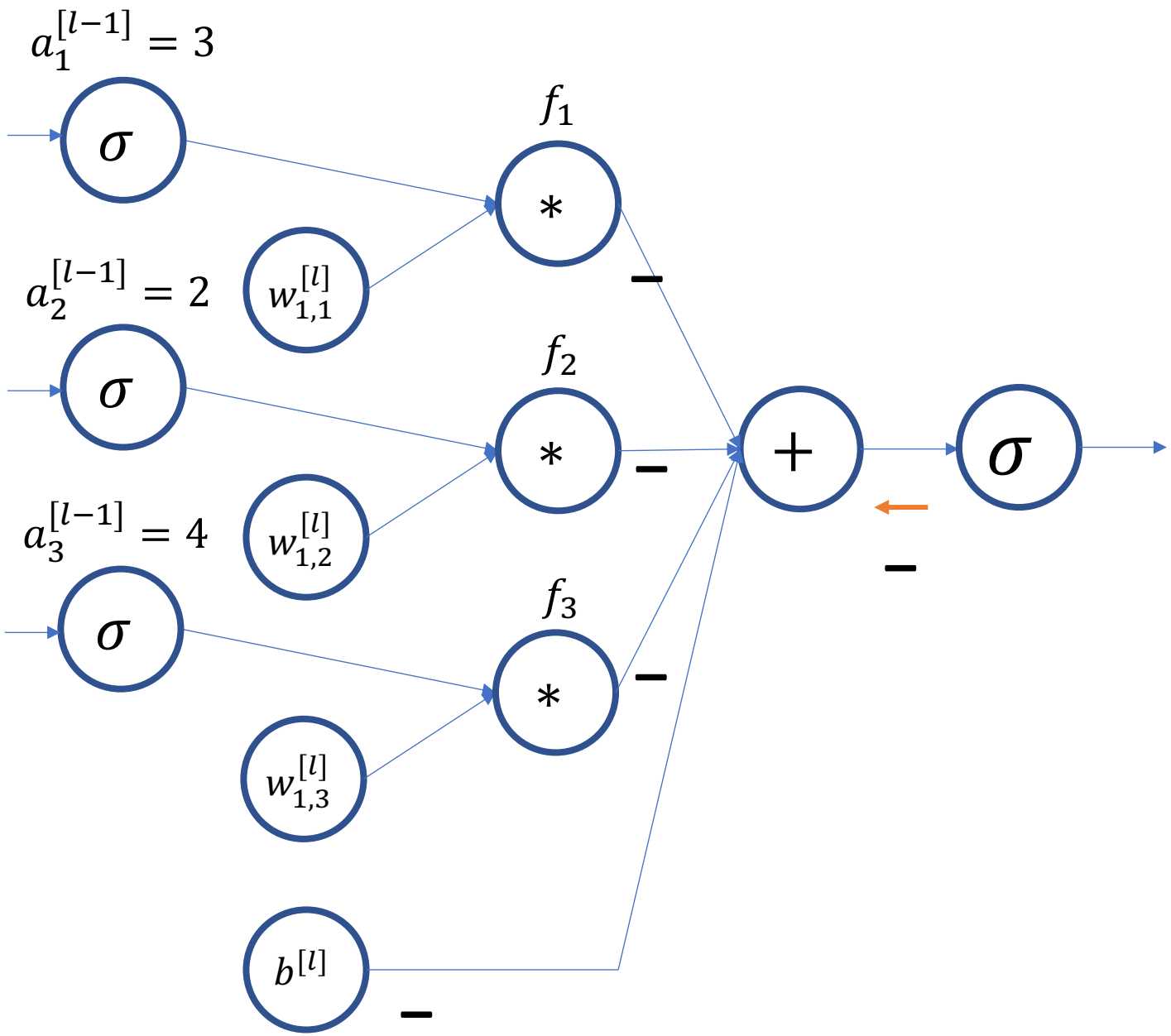
- Another Problem with Sigmoid is that it is always positive
- Assuming all units in the layer use the same activation function, then all units in a layer will output a positive number



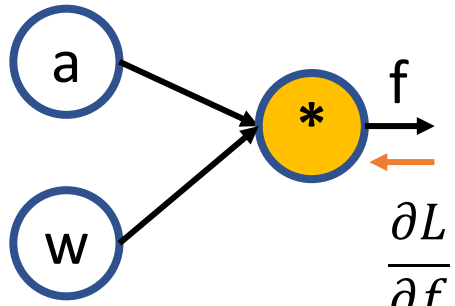

$a_1^{[l-1]} = 3$
$a_2^{[l-1]} = 2$
$a_3^{[l-1]} = 4$
σ
σ
σ
$w_{1,1}^{[l]}$
$w_{1,2}^{[l]}$
$w_{1,3}^{[l]}$
f_1
f_2
f_3
$*$
$*$
$*$
$+$
σ
$b^{[l]}$


$a_1^{[l-1]} = 3$
$a_2^{[l-1]} = 2$
$a_3^{[l-1]} = 4$
σ
σ
σ
$w_{1,1}^{[l]}$
$w_{1,2}^{[l]}$
$w_{1,3}^{[l]}$
$b^{[l]}$
f_1
f_2
f_3
*
*
*
+
σ
−


$a_1^{[l-1]} = 3$
$a_2^{[l-1]} = 2$
$a_3^{[l-1]} = 4$
σ
σ
σ
$w_{1,1}^{[l]}$
$w_{1,2}^{[l]}$
$w_{1,3}^{[l]}$
$b^{[l]}$
f_1
f_2
f_3
*
*
*
+
σ
−
−
−
−
−

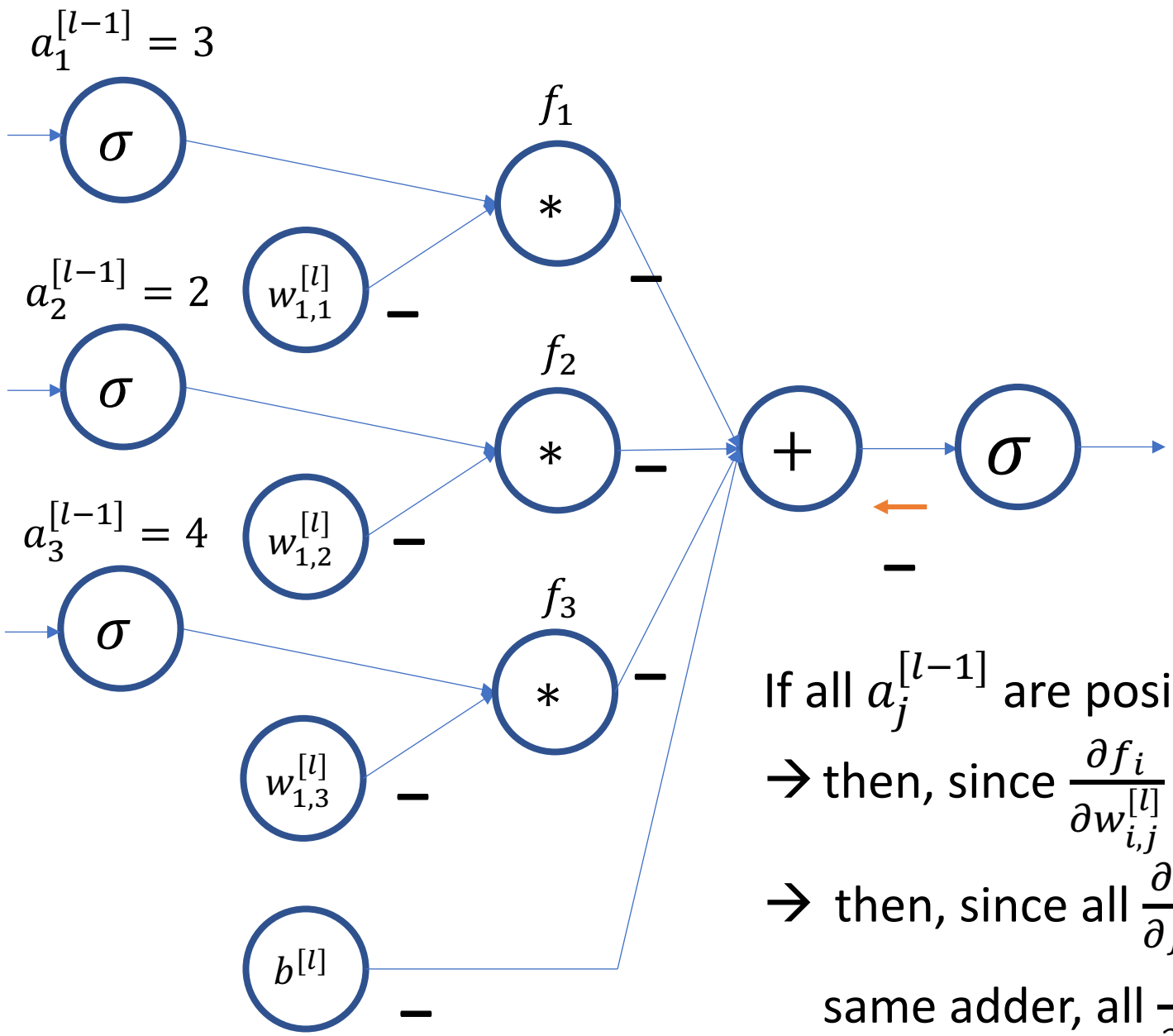


Recall:

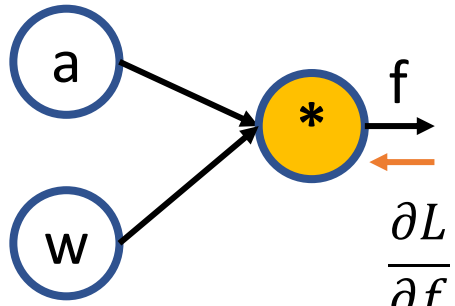


$$\frac{\partial f}{\partial w} = a, \qquad \frac{\partial f}{\partial a} = w$$

$$\frac{\partial L}{\partial w} = \frac{\partial f}{\partial w}\frac{\partial L}{\partial f} = a\frac{\partial L}{\partial f}$$



Recall:



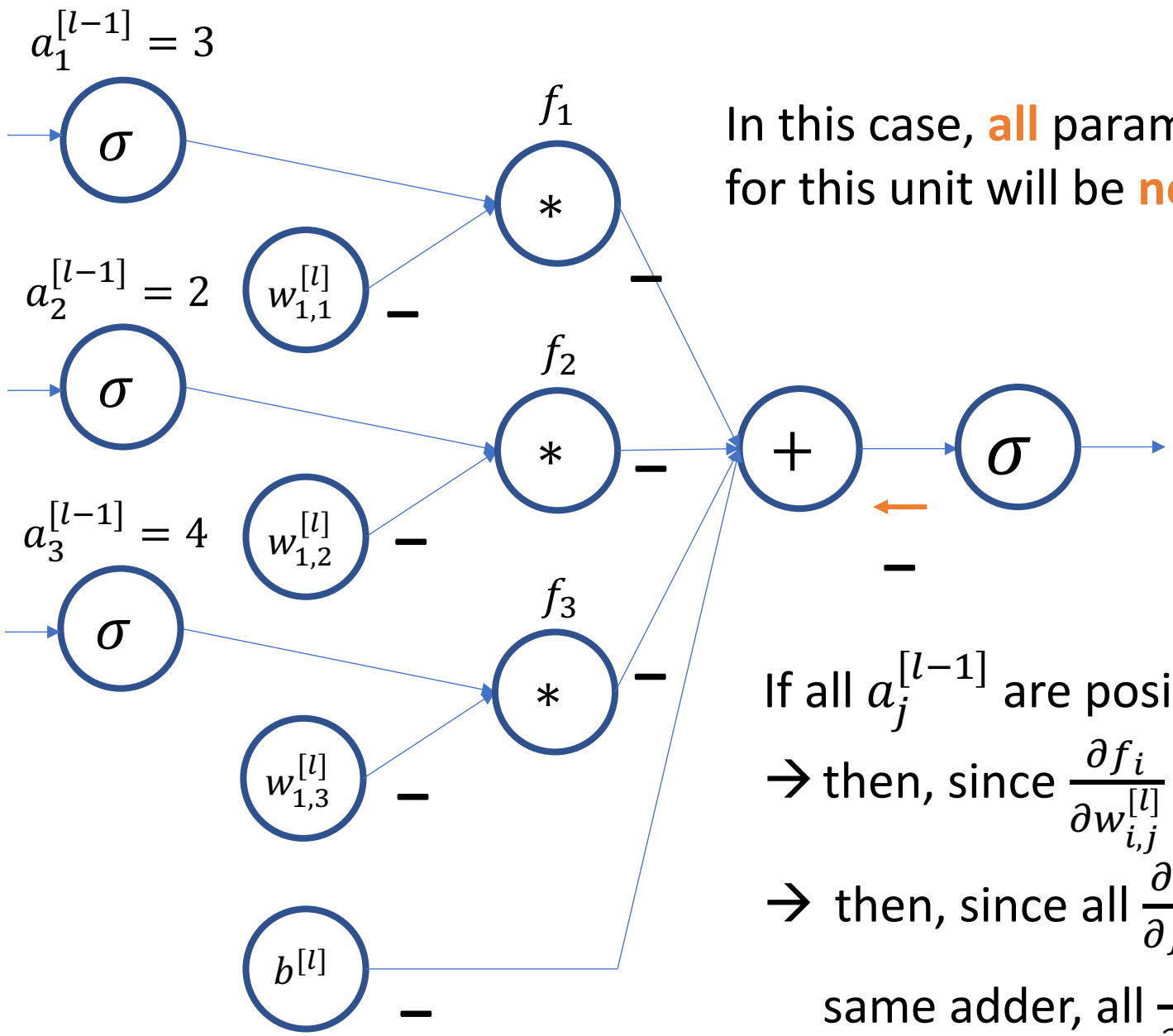
$$\frac{\partial f}{\partial w} = a, \qquad \frac{\partial f}{\partial a} = w$$

$$\frac{\partial L}{\partial w} = \frac{\partial f}{\partial w}\frac{\partial L}{\partial f} = a\frac{\partial L}{\partial f}$$

If all $a_j^{[l-1]}$ are positive

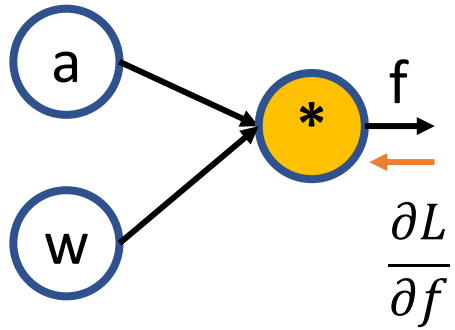
→ then, since $\frac{\partial f_i}{\partial w_{i,j}^{[l]}} = a_j^{[l-1]}$, all $\frac{\partial f_i}{\partial w_{i,j}^{[l]}}$ will be positive

→ then, since all $\frac{\partial L}{\partial f_i}$ are equal since them come from the same adder, all $\frac{\partial L}{\partial w_{i,j}^{[l]}}$ will have the same sign



In this case, **all** parameter updates for this unit will be **negative**!

Recall:



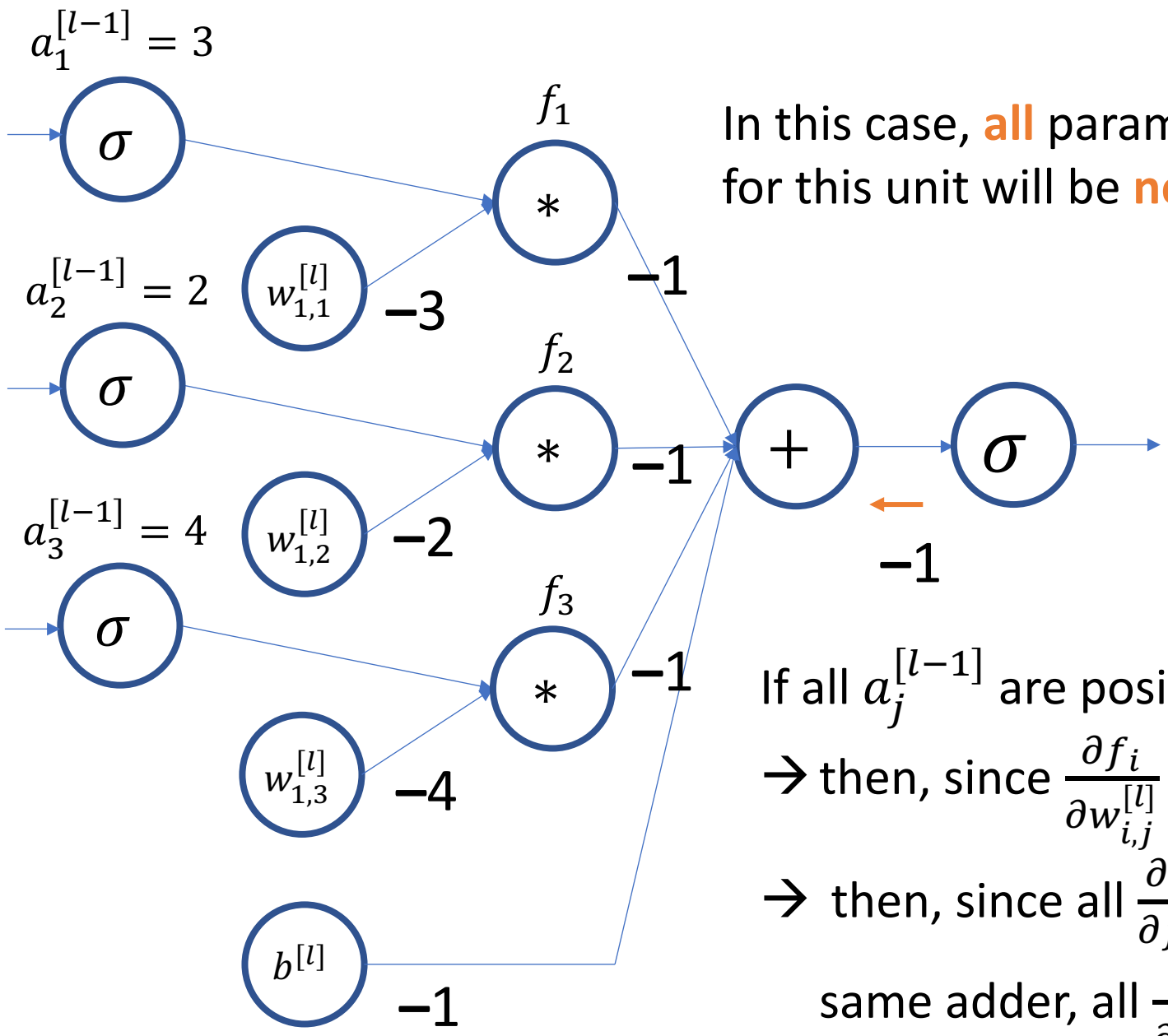
$$\frac{\partial f}{\partial w} = a, \qquad \frac{\partial f}{\partial a} = w$$

$$\frac{\partial L}{\partial w} = \frac{\partial f}{\partial w}\frac{\partial L}{\partial f} = a\frac{\partial L}{\partial f}$$

If all $a_j^{[l-1]}$ are positive

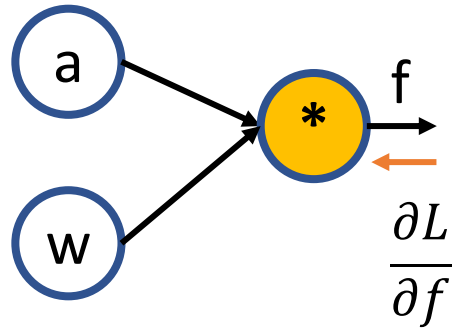
→ then, since $\frac{\partial f_i}{\partial w_{i,j}^{[l]}} = a_j^{[l-1]}$, all $\frac{\partial f_i}{\partial w_{i,j}^{[l]}}$ will be positive

→ then, since all $\frac{\partial L}{\partial f_i}$ are equal since them come from the same adder, all $\frac{\partial L}{\partial w_{i,j}^{[l]}}$ will have the same sign



In this case, **all** parameter updates for this unit will be **negative**!

Recall:



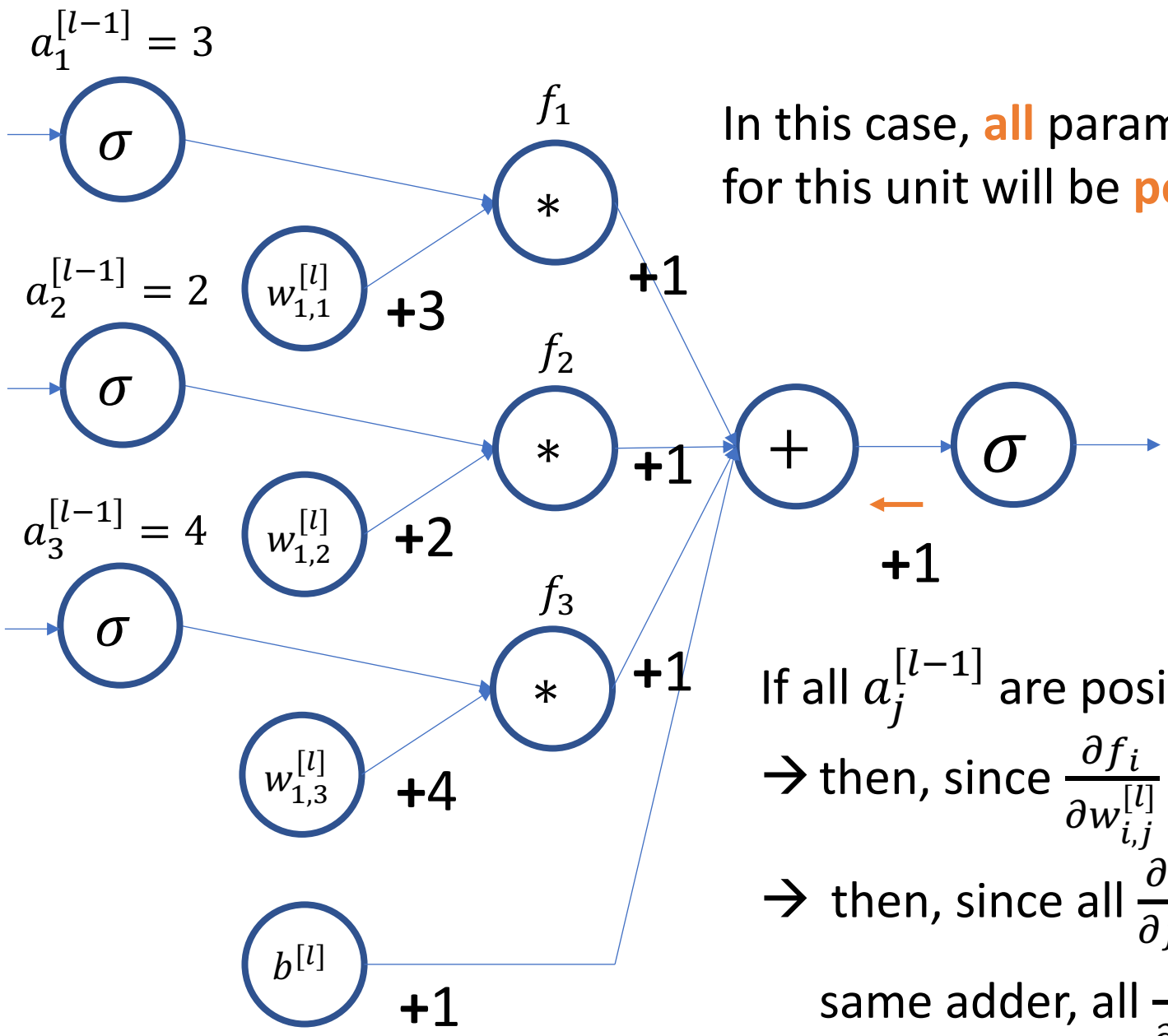
$$\frac{\partial f}{\partial w} = a, \qquad \frac{\partial f}{\partial a} = w$$

$$\frac{\partial L}{\partial w} = \frac{\partial f}{\partial w}\frac{\partial L}{\partial f} = a\frac{\partial L}{\partial f}$$

If all $a_j^{[l-1]}$ are positive

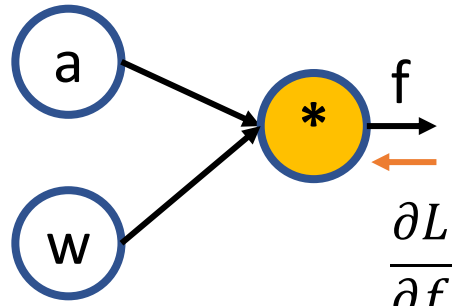
→ then, since $\frac{\partial f_i}{\partial w_{i,j}^{[l]}} = a_j^{[l-1]}$, all $\frac{\partial f_i}{\partial w_{i,j}^{[l]}}$ will be positive

→ then, since all $\frac{\partial L}{\partial f_i}$ are equal since them come from the same adder, all $\frac{\partial L}{\partial w_{i,j}^{[l]}}$ will have the same sign



In this case, **all** parameter updates for this unit will be **positive**!

Recall:



$$\frac{\partial f}{\partial w} = a, \qquad \frac{\partial f}{\partial a} = w$$

$$\frac{\partial L}{\partial w} = \frac{\partial f}{\partial w}\frac{\partial L}{\partial f} = a\frac{\partial L}{\partial f}$$

If all $a_j^{[l-1]}$ are positive

→ then, since $\frac{\partial f_i}{\partial w_{i,j}^{[l]}} = a_j^{[l-1]}$, all $\frac{\partial f_i}{\partial w_{i,j}^{[l]}}$ will be positive

→ then, since all $\frac{\partial L}{\partial f_i}$ are equal since them come from the same adder, all $\frac{\partial L}{\partial w_{i,j}^{[l]}}$ will have the same sign

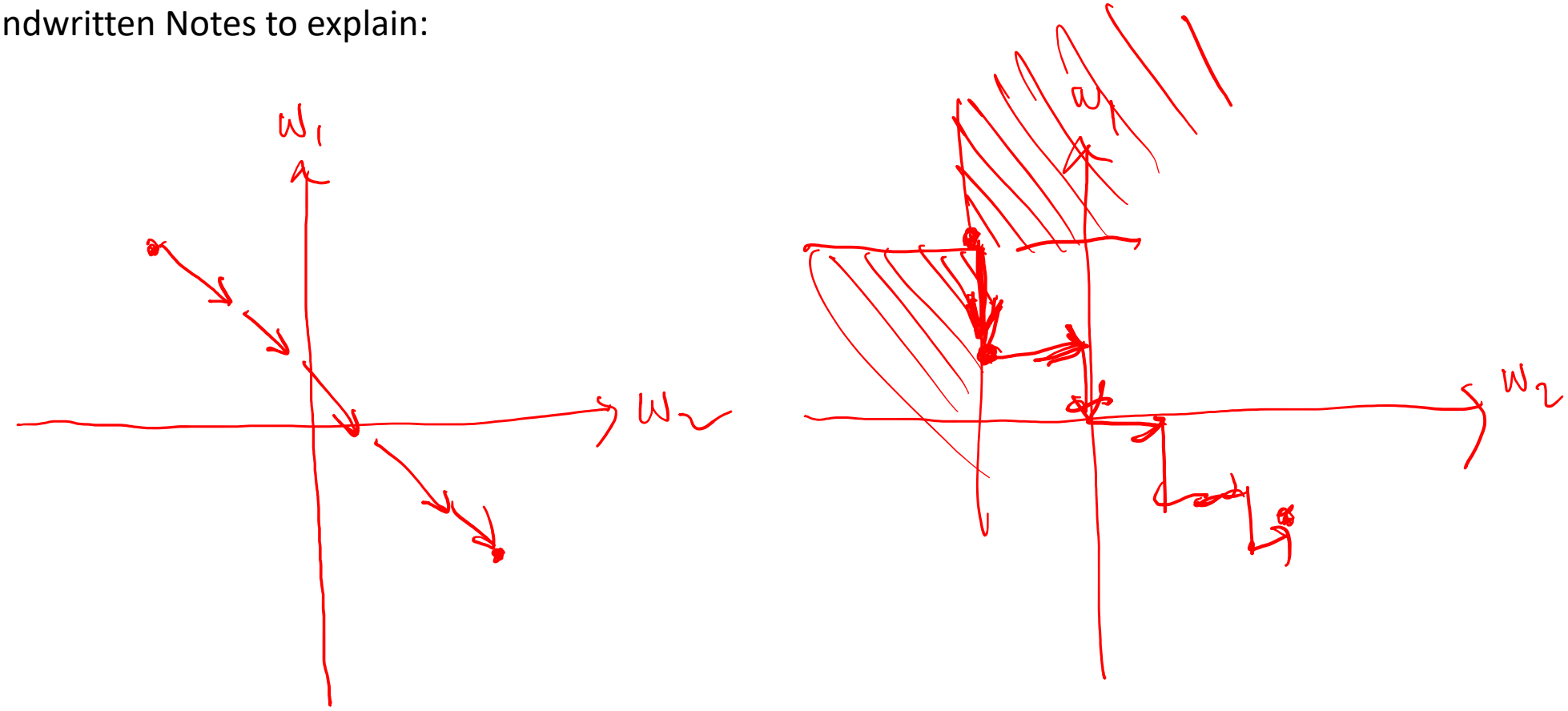
Why does this matter?

Generally,

- If all inputs to a unit are the same sign, then all weights for that unit have the same sign for $\frac{\partial L}{\partial w}$
  (positive in the previous example due to Sigmoid activations),
- This means, gradient descent will update all those weights in the same direction. i.e.
  - All increase or
  - All decrease.
  - Cannot have some decrease some increase.
- Why is this bad?

Handwritten Notes to explain:

Handwritten Notes to explain:

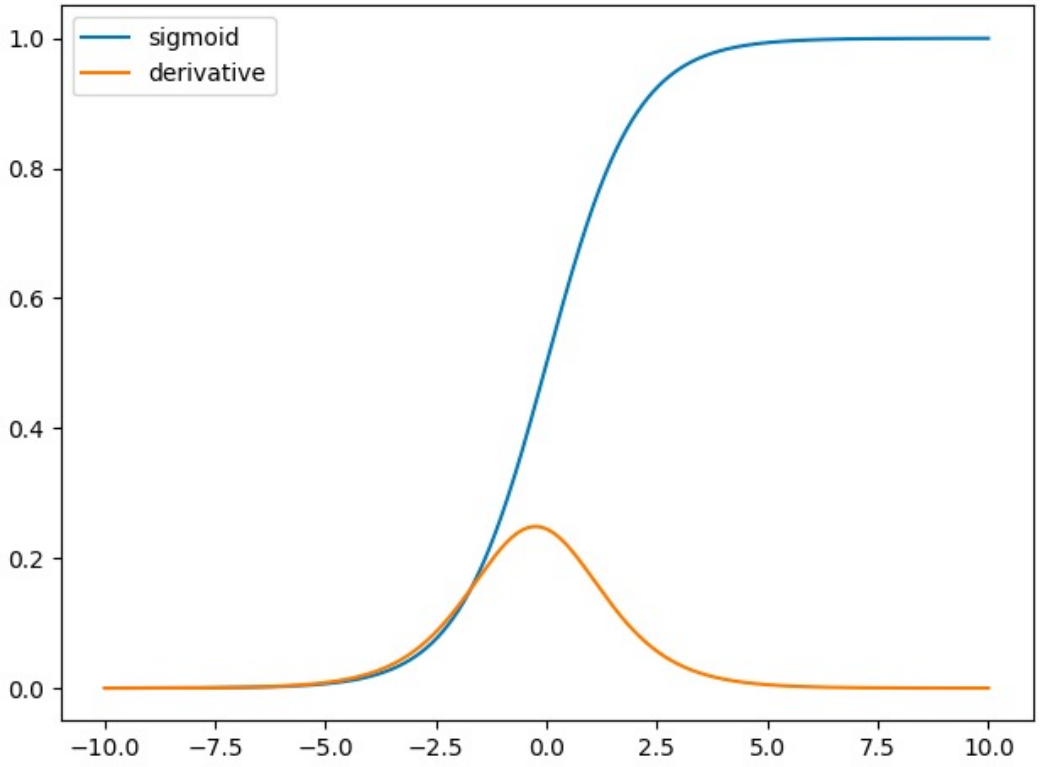


Non Zero-Centered Inputs

- More generally, this is the problem of Non Zero-Centered Inputs
- The examples of all inputs being positive is an extreme case
- The other extreme is if all inputs were negative

Max value of Sigmoid Gradient

- The maximum value for the gradient of sigmoid is 0.25 (at x=0)
- Each time gradients flow through a sigmoid function, it is reduced to ¼ or more
- This also contributes to the vanishing gradients problem as the gradient signal attenuates through each layer



Sigmoid Activation – Summary

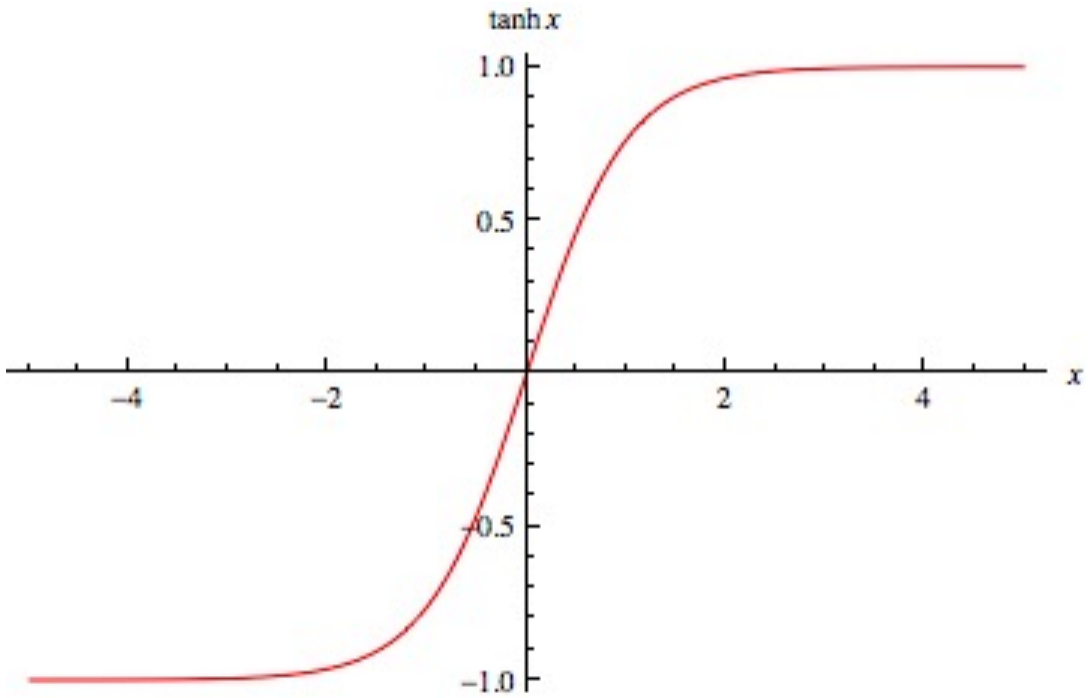
- Two Major Drawbacks

1. Vanishing Gradients Problem ← Big problem for deep networks
2. Non zero-centered data ← More of an inconvenience

- Do not use Sigmoid for hidden layers
- You can still use it on the output because typically, accompanying binary cross-entropy loss removes the saturation effect.
- Technically, Sigmoid Function is a **class** of functions with the S shape.
- The one we've been talking about is called the Logistic Function

Tanh Activation Function

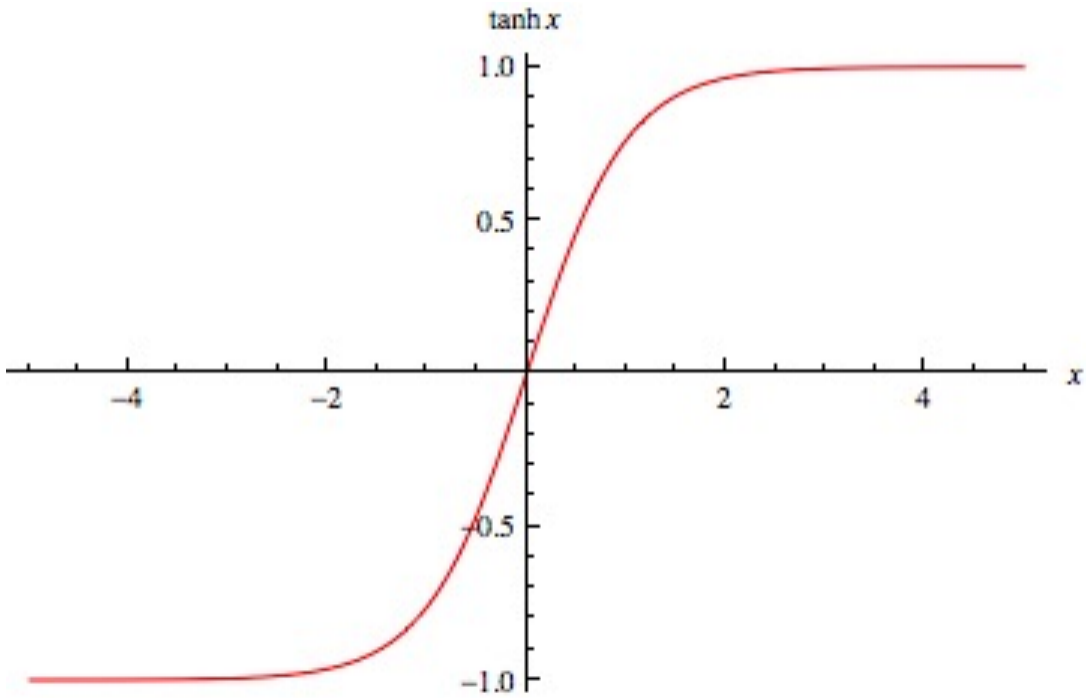
- Another type of Sigmoid Function is the Hyperbolic Tangent function
- Similar to Logistic Function except its range of output is between -1 and 1.
- Solves the problem of non zero-centered outputs
- Still has saturated regions and vanishing gradients problem



Efficient Backprop, Lecun et al, http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

Tanh Activation Function

- tanh is empirically shown to generally lead to faster learning compared to sigmoid
- However still has vanishing gradient problem. So be wary of using for deep networks



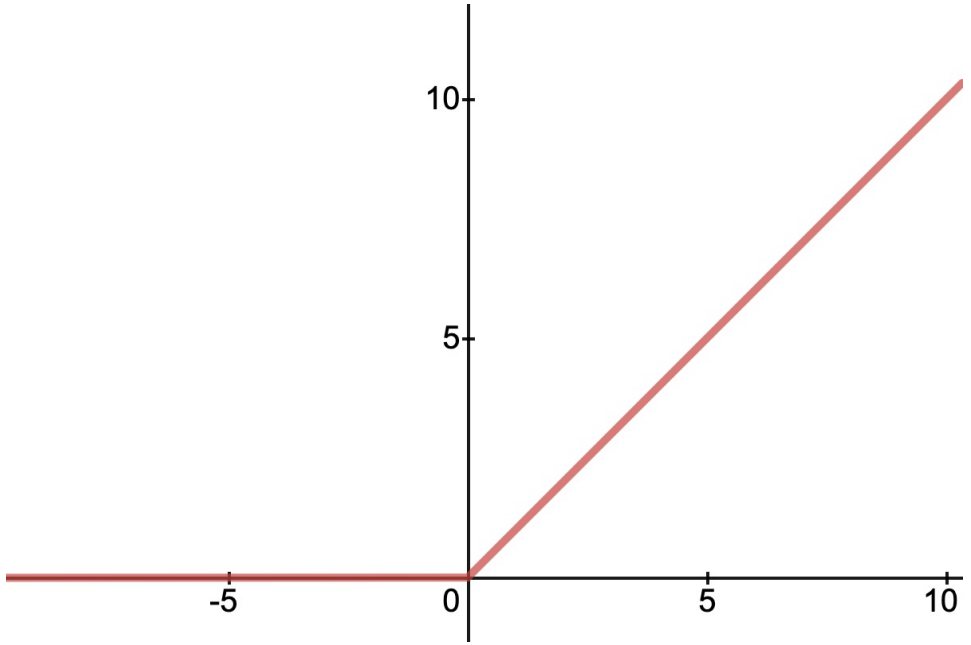
$$\tanh(x) = 2\sigma(2x) - 1$$

Efficient Backprop, Lecun et al, http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

Rectified Linear Activation Unit (ReLU)

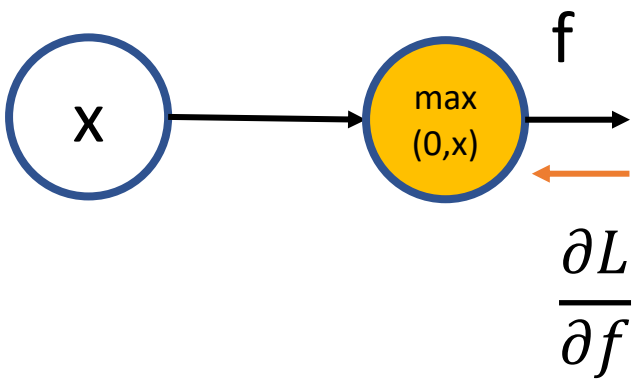
Knowing what you now know about backpropagation, what's good and what's bad about this activation function?

$$f(x) = max(x, 0)$$

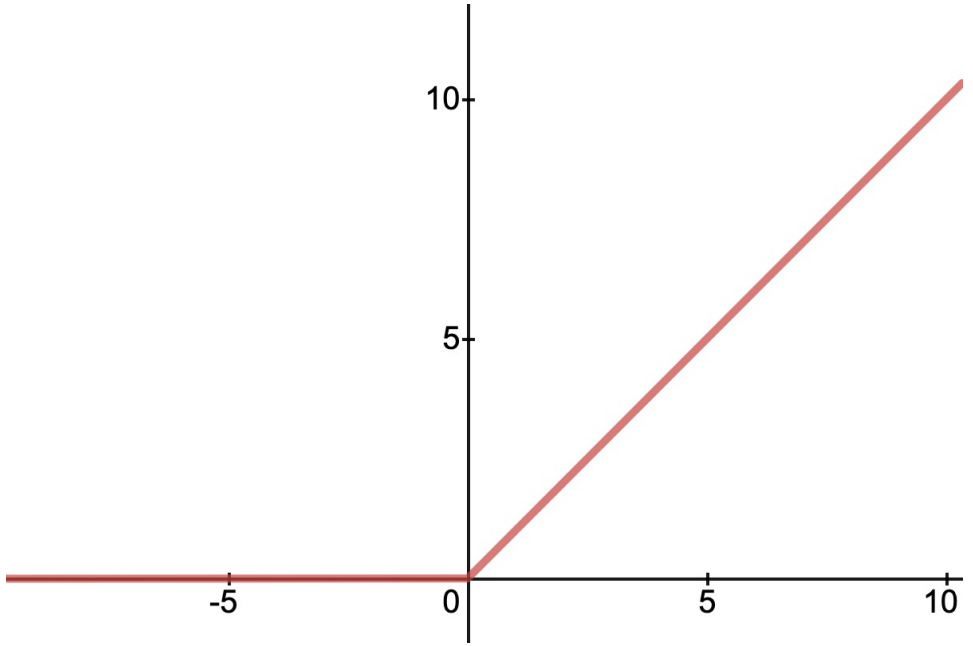


"Deep Sparse Rectifier Neural Networks", Glorot, Bordes, Bengio, 2011, http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

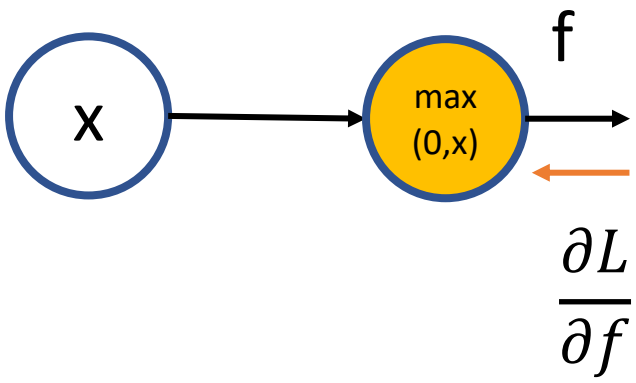
Backprop through ReLU



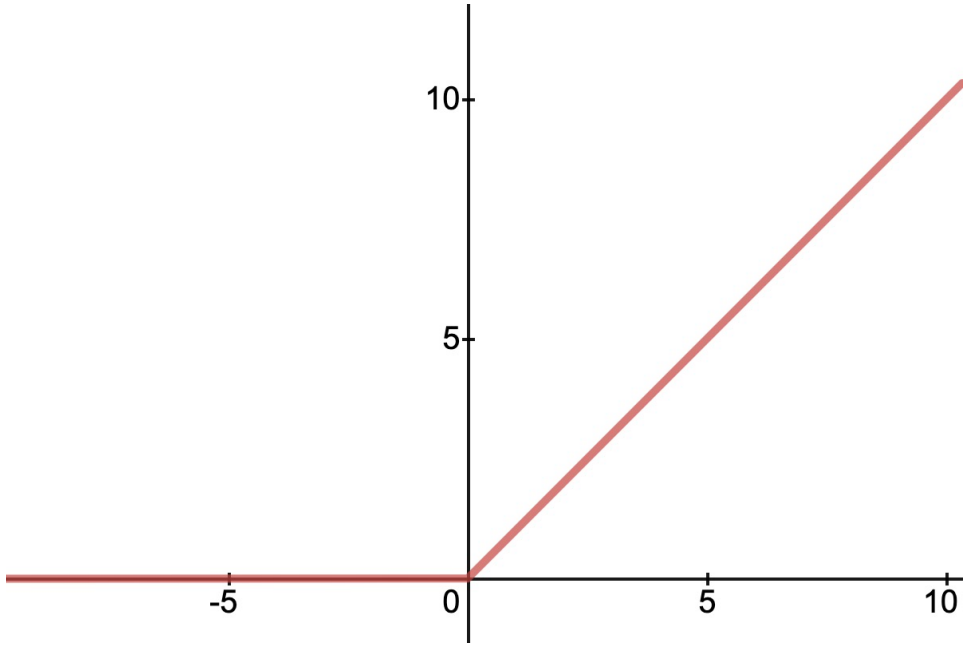
$$f(x) = max(x, 0)$$



Backprop through ReLU

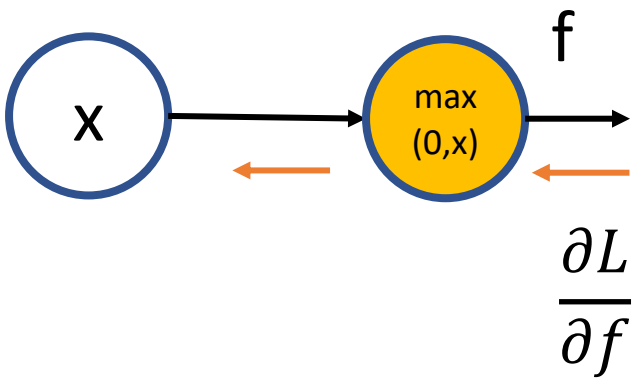


$$f(x) = max(x, 0)$$

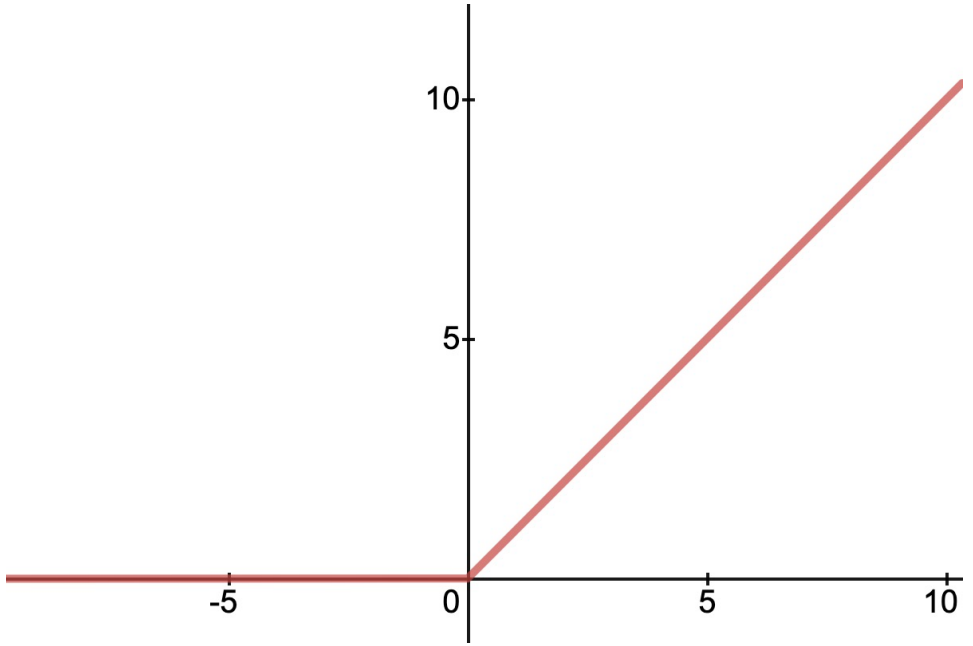


Local Gradient: $$\frac{\partial f}{\partial x} = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Backprop through ReLU



$$f(x) = max(x, 0)$$



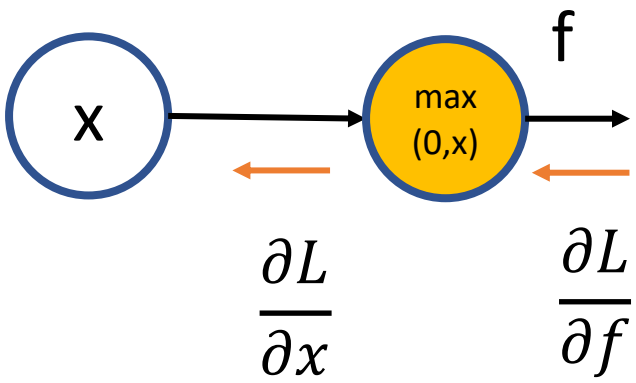
Local Gradient:

$$\frac{\partial f}{\partial x} = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

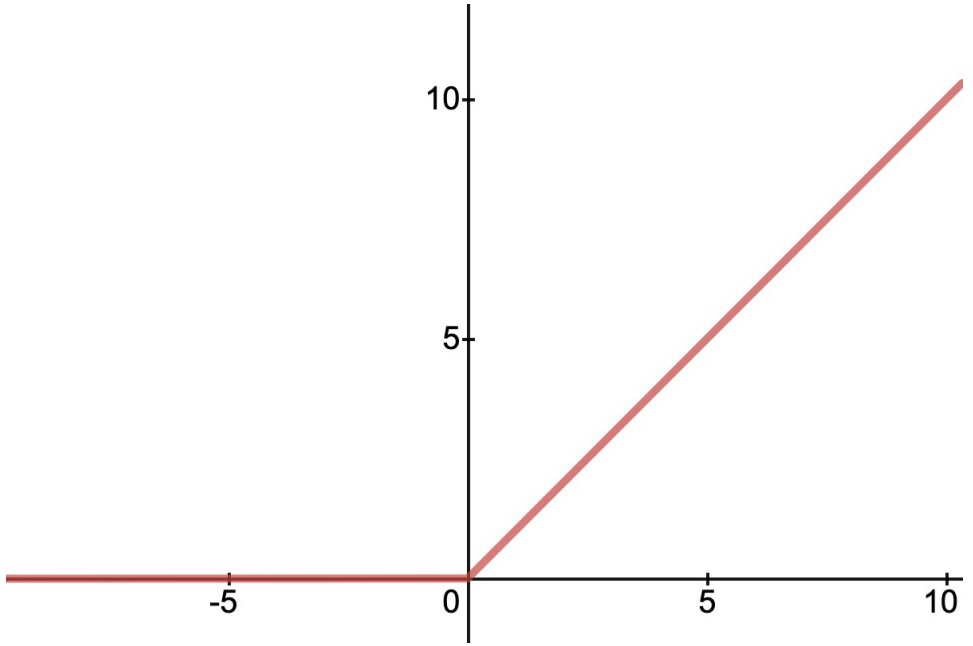
Downstream Gradient:

$$\frac{\partial L}{\partial x} = \begin{cases} \frac{\partial L}{\partial f}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Backprop through ReLU



$$f(x) = max(x, 0)$$

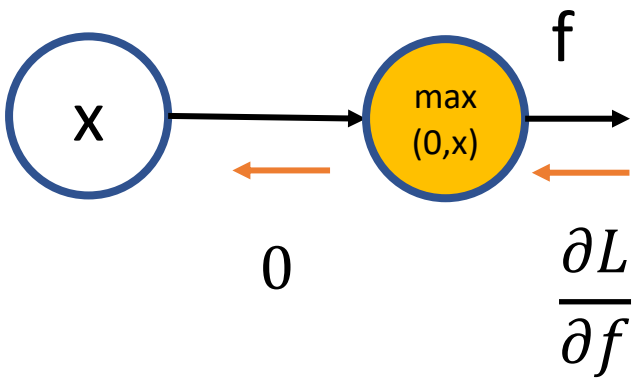


Local Gradient: $$\frac{\partial f}{\partial x} = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

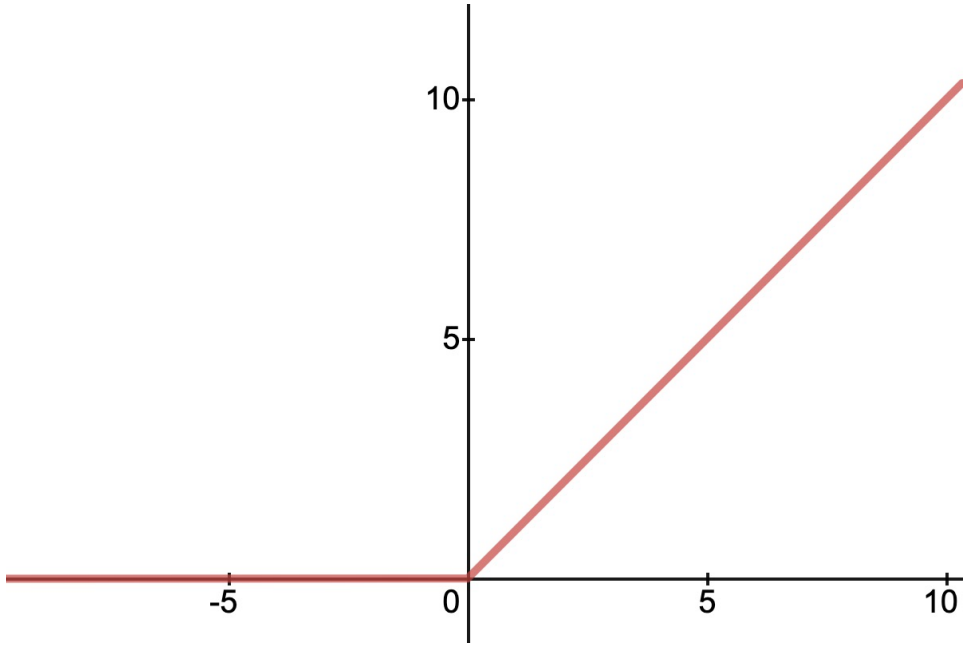
Downstream Gradient: $$\frac{\partial L}{\partial x} = \begin{cases} \frac{\partial L}{\partial f}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Pass through

Backprop through ReLU



$$f(x) = max(x, 0)$$



Local Gradient: $$\frac{\partial f}{\partial x} = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Downstream Gradient: $$\frac{\partial L}{\partial x} = \begin{cases} \frac{\partial L}{\partial f}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

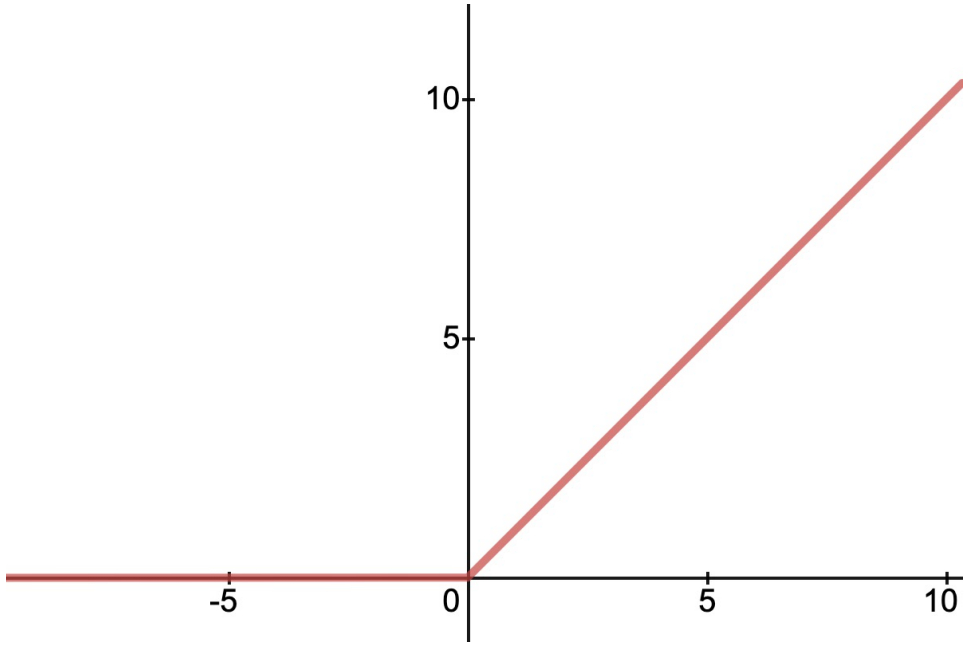
No Gradient Flow!

Rectified Linear Activation Unit (ReLU)

$$f(x) = max(x, 0)$$

Pros:

- No vanishing gradient problem.
- Passthrough for gradient flow
- Easy to compute
  - Speeds up training
  - Speeds up prediction
- Sparse Activations



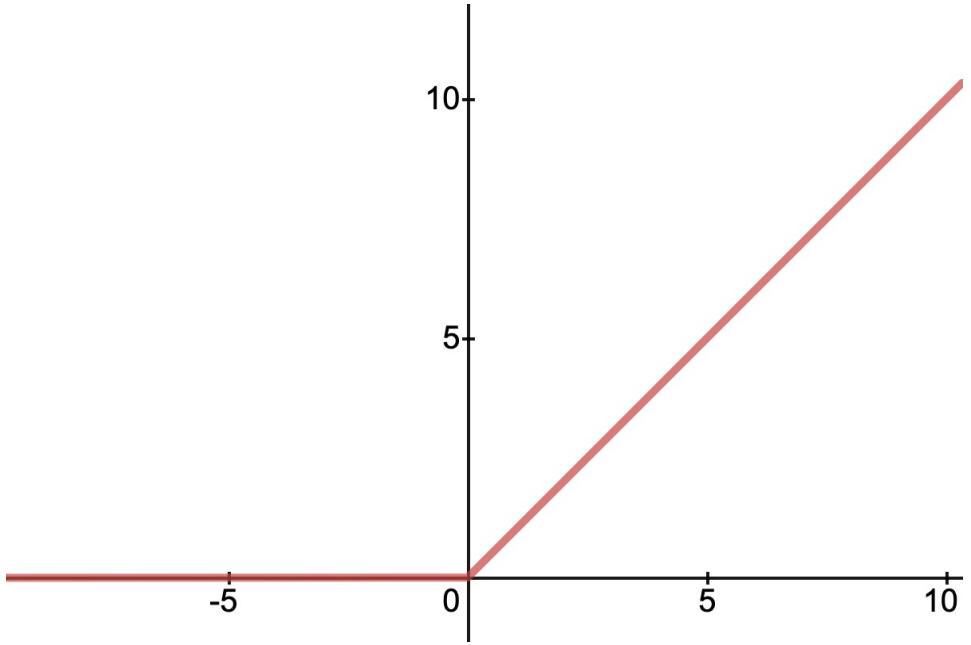
"Deep Sparse Rectifier Neural Networks", Glorot, Bordes, Bengio, 2011, http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

Rectified Linear Activation Unit (ReLU)

$$f(x) = max(x, 0)$$

Pros:

- No vanishing gradient problem.
- Passthrough for gradient flow
- Easy to compute
  - Speeds up training
  - Speeds up prediction
- Sparse Activations

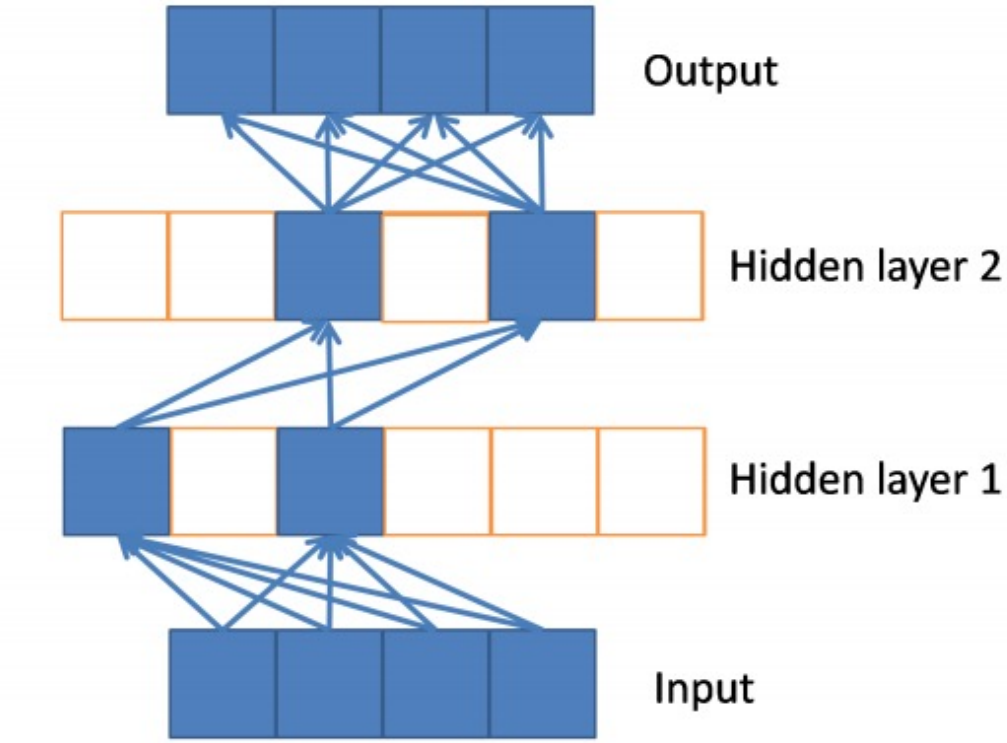


Currently the most popular activation function, and the default choice!

"Deep Sparse Rectifier Neural Networks", Glorot, Bordes, Bengio, 2011, http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

ReLU – Sparse Activations

- ReLU can output a true 0
  - Sigmoid can only output near 0
  - Tanh can only output zero at one specific point
- True 0s lead to sparse activations of neurons
- Hypothesized that this has advantages (see paper below)



"Deep Sparse Rectifier Neural Networks", Glorot, Bordes, Bengio, 2011, http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

History of ReLU

- Major breakthrough that allowed efficient training of deep neural networks
- AlexNet CNN architecture used ReLU and found **6x** faster convergence vs tanh (in terms of training iterations)
- When in doubt, use ReLU for Fully-Connected Neural Networks and Convolutional Neural Networks.
- Need to be careful for RNNs due to exploding gradient problem (more on this later)

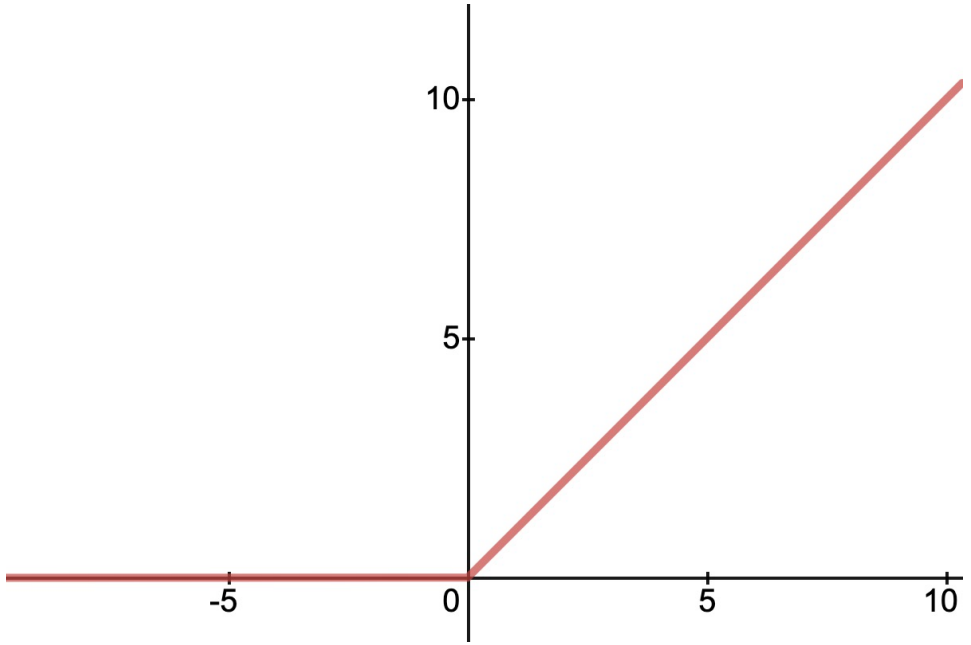
"ImageNet Classification with Deep Convolutional Neural Networks", Krizhevsky, Sutskever, Hinton, 2012
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Rectified Linear Activation Unit (ReLU)

$$f(x) = max(x, 0)$$

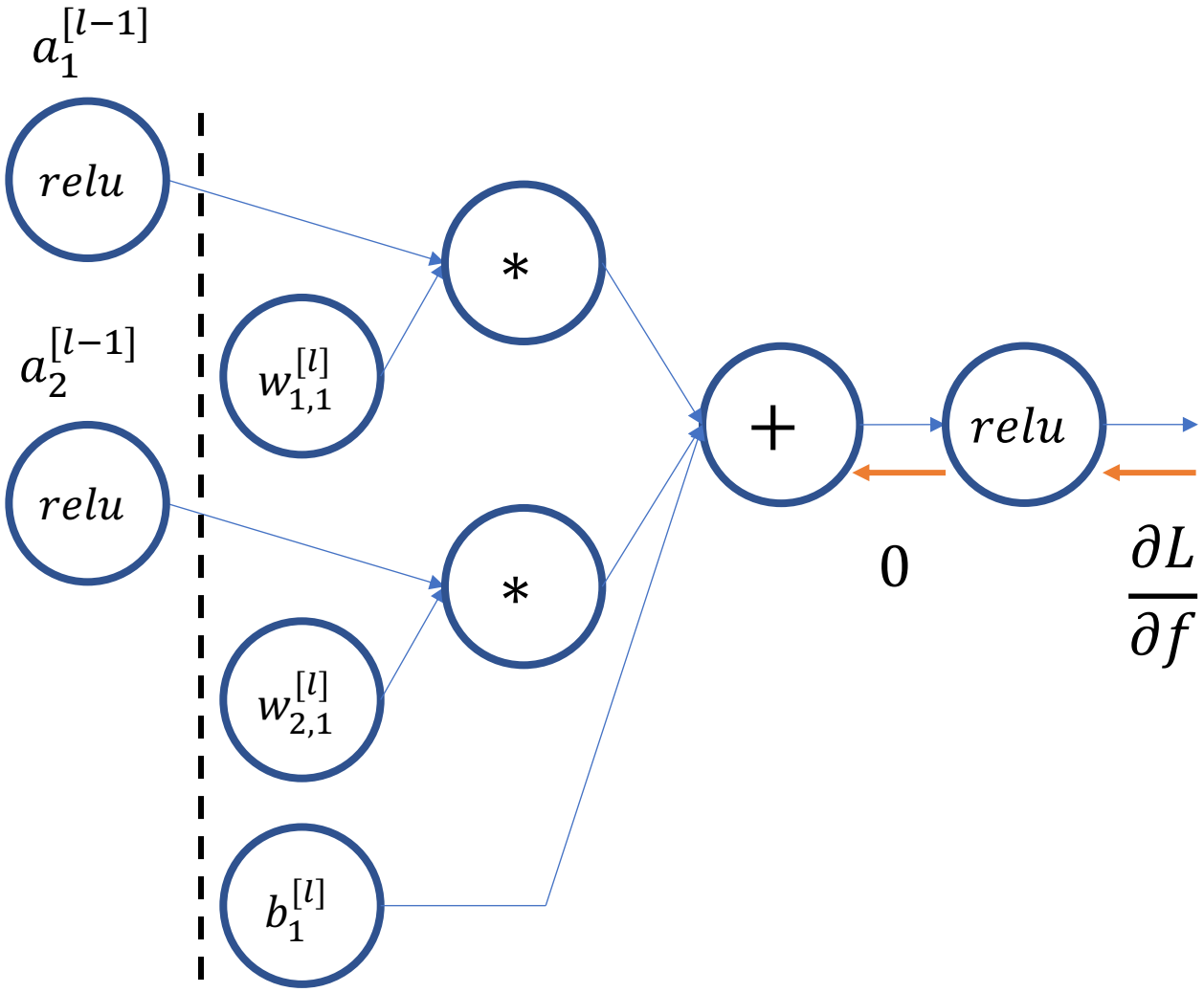
Cons:

- Dead ReLU problem
- Non-zero-centered output



"Deep Sparse Rectifier Neural Networks", Glorot, Bordes, Bengio, 2011, http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

Dead ReLU



- If no gradient flows through a ReLU neuron, its associated parameters won't receive "info" on how to change
- Ok if it happens on one (or some) training samples
- If this is the case for ALL training samples, then those parameters will never update

Causes of Dead ReLU Neuron

$$a_i^{[l]} = relu\left(z_i^{[l]}\right), \qquad z_i^{[l]} = W_i^{[l]} a^{[l-1]} + b_i^{[l]}$$

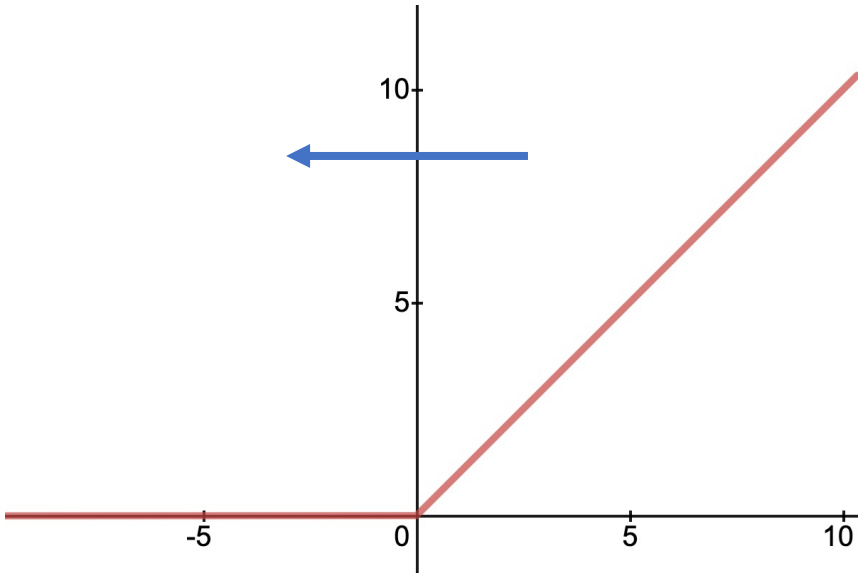
Dead ReLU if $z_i^{[l]} < 0$ for all training samples.

This happens when:

1. $W_i^{[l]}$, $b_i^{[l]}$ initialized such that $z_i^{[l]} < 0$ (i.e. dead)
   → Dead from the start. Never had a chance to learn
2. Learning rate is too big. During a gradient descent iteration, $W_i^{[l]}$, $b_i^{[l]}$ are updated such that $z_i^{[l]} < 0$
   → Neuron learned for a bit but then died

Avoiding Dead ReLU

- Initialize bias terms with small positive value
- Need to be mindful about how we initialize weight parameters. Use “He Initialization”
- We will cover various weight initialization strategies later in the course.



$$a_i^{[l]} = relu\left(W^{[l]} a^{[l-1]} + b_i^{[l]} \right)$$

“Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”, He et al, 2015, https://arxiv.org/abs/1502.01852

Dead ReLU vs Neuron Saturation (e.g. Sigmoid)

- Similar effect but not exactly the same
- For ReLU where input is < 0, gradient is exactly 0.
- For Sigmoid (and Tanh), gradient of saturated regions **approach** 0 but are never exactly 0. Can still flow gradient but practically none.

ReLU – Non Zero-Centered Output

- We talked about how producing only positive values can be a problem that slows down convergence
- However, we had said this issue is more of an inconvenience. Empirically, ReLU's other advantages seem to greatly outweigh this disadvantage

ReLU Summary

Pros:

- Better gradient propagation
- Sparse Activations
- Fast to compute
  - Speeds up training
  - Speeds up prediction

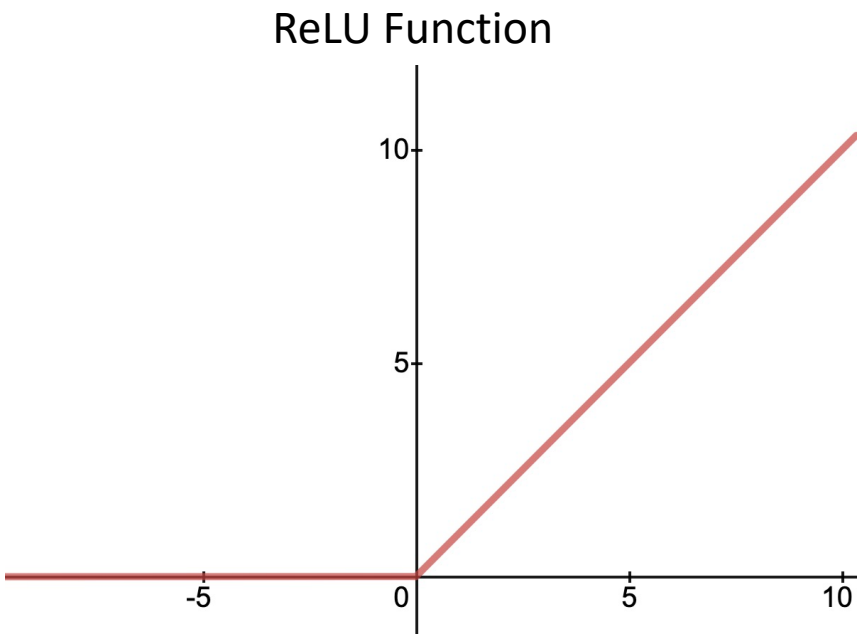
Cons:

- Dead ReLU problem
- Output is non zero-centered (all positive)

Variations on ReLU

Variants try to fix dead ReLU problem by changing the x<0 region

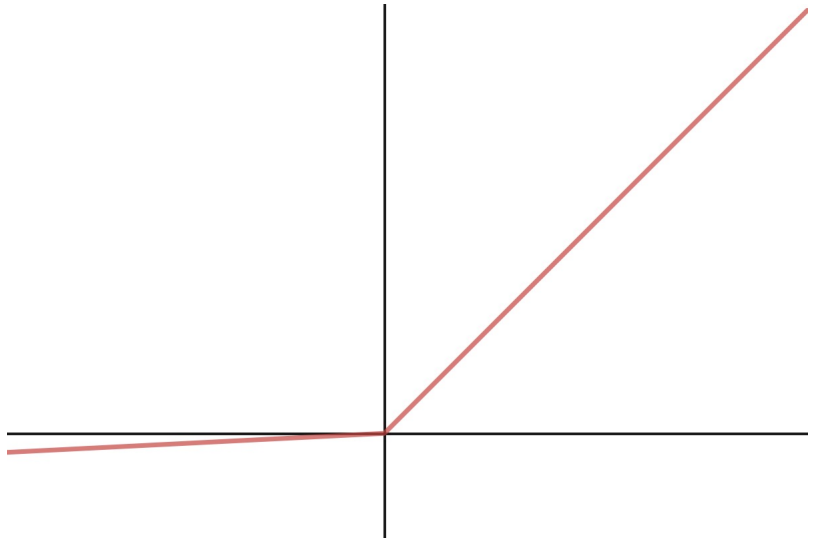
- Leaky ReLU
- Parametric ReLU
- ELU
- SELU
- GELU
- …



Leaky ReLU activation

- Instead of output 0 in negative region, output a small linear function
  → still has a gradient
- Gives a chance to get out of “Dead ReLU” region

$$f(x) = max(x, 0.01x)$$

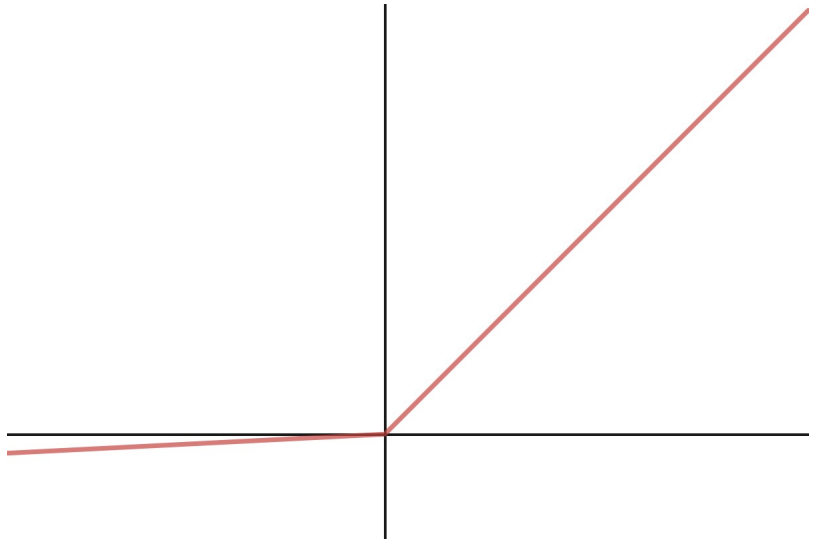


“Rectifier Nonlinearities Improve Neural Network Acoustic Models”, Mass, Hannun, Ng, 2013, https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf

Parametric ReLU

- Generalization of Leaky ReLU except slope of line at x < 0 is a **learned parameter**
- a is a learned parameter
  i.e. it is updated during gradient descent.

$$f(x) = max(x, ax)$$



"Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", He, Zhang, Ren, Sun, Microsoft Research, 2015, https://arxiv.org/pdf/1502.01852.pdf

More Variants: ELU, SELU, GELU, …

$$elu(x) = \begin{cases} x, & x < 0 \\ a(e^x - 1), & x \geq 0 \end{cases}$$

$$selu(x) = \begin{cases} \lambda x, & x < 0 \\ \lambda a(e^x - 1), & x \geq 0 \end{cases}$$

$$\alpha = 1.67326324235437728481704299167 17$$
$$\lambda = 1.05070098735548049341933498529 46$$

Self-Normalizing Neural Networks, Klambauer, et al., 2017 https://arxiv.org/abs/1706.02515

A Comparison on Image Classification

Model	ResNet	WRN	DenseNet
LReLU	94.2	95.6	94.7
PReLU	94.1	95.1	94.5
Softplus	94.6	94.9	94.7
ELU	94.1	94.1	94.4
SELU	93.0	93.2	93.9
GELU	94.3	95.5	94.8
ReLU	93.8	95.3	94.8
Swish-1	94.7	95.5	94.8
Swish	94.5	95.5	94.8

Table 4: CIFAR-10 accuracy.

Model	ResNet	WRN	DenseNet
LReLU	74.2	78.0	83.3
PReLU	74.5	77.3	81.5
Softplus	76.0	78.4	83.7
ELU	75.0	76.0	80.6
SELU	73.2	74.3	80.8
GELU	74.7	78.0	83.8
ReLU	74.2	77.8	83.7
Swish-1	75.1	78.5	83.8
Swish	75.1	78.0	83.9

Table 5: CIFAR-100 accuracy.

"Searching for Activation Functions", Ramachandran et al., 2018, https://arxiv.org/abs/1710.05941

Summary of Activation Functions

- ReLU is a good default choice.
  - Monitor number of Dead ReLUs and tune your learning rate
- You can try others ELU, Leaky ReLU, SELU, etc.,
- ReLU is strictly better than tanh
- ReLU and tanh are strictly better than Sigmoid. Don’t even bother trying to use it on hidden layers

Learning Objectives

- Gain more experience with how backprop works on computation graphs through examples
  Be able to analyze why certain activation functions are better than other in the context of training and backpropagation