Neural Network Introduction

Harpreet Singh (Fall 2023)

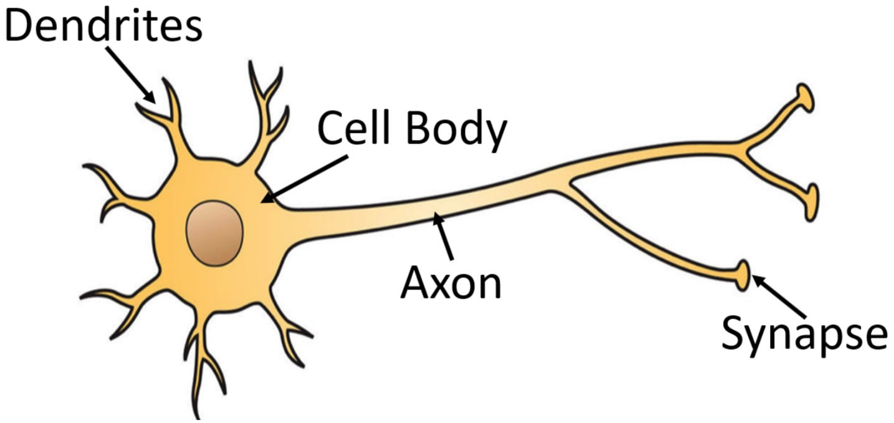
Artificial Neural Network (Feed Forward neural Network

Biological Neural Networks

Biological neurons refer to a collection of interconnected nerve cells organized in layers, which communicate with one another when specific conditions are met.

A biological neuron consists of three primary components:

- Dendrites
- Soma (or cell body)
- Axon



Dendrites receive signals from other neurons.

→ The soma performs the summation of incoming signals.

→ Once sufficient input is received, the neuron fires by transmitting a signal over its axon to other cells

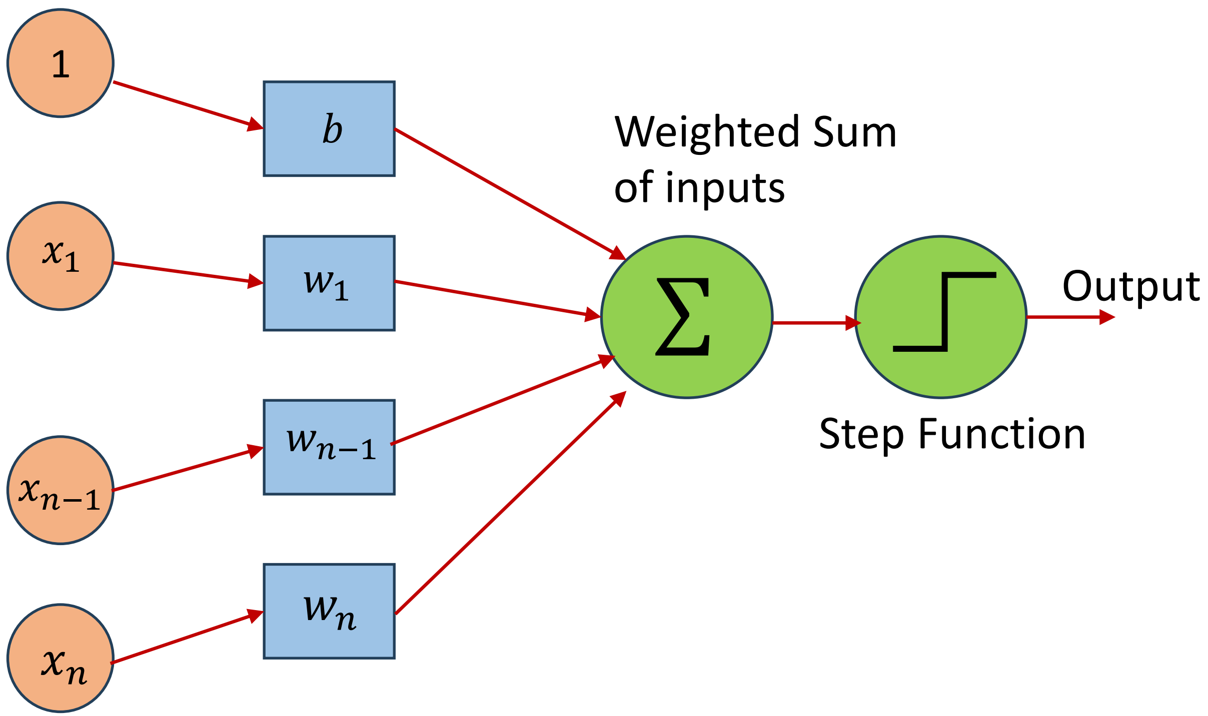
Artificial Neurons

The properties of biological neurons inspire the processing elements in ANN in several ways:

- The processing element receives multiple signals.
- These signals can be adjusted by weights at the receiving connection points.
- The processing element calculates the weighted sum of the inputs.
- When the conditions are appropriate (sufficient input), the neuron transmits a single output.
- The output from a particular neuron can be connected to many other neurons

Perceptron

The Perceptron is one of the simplest ANN architectures, invented in 1957 by Frank Rosenblatt.

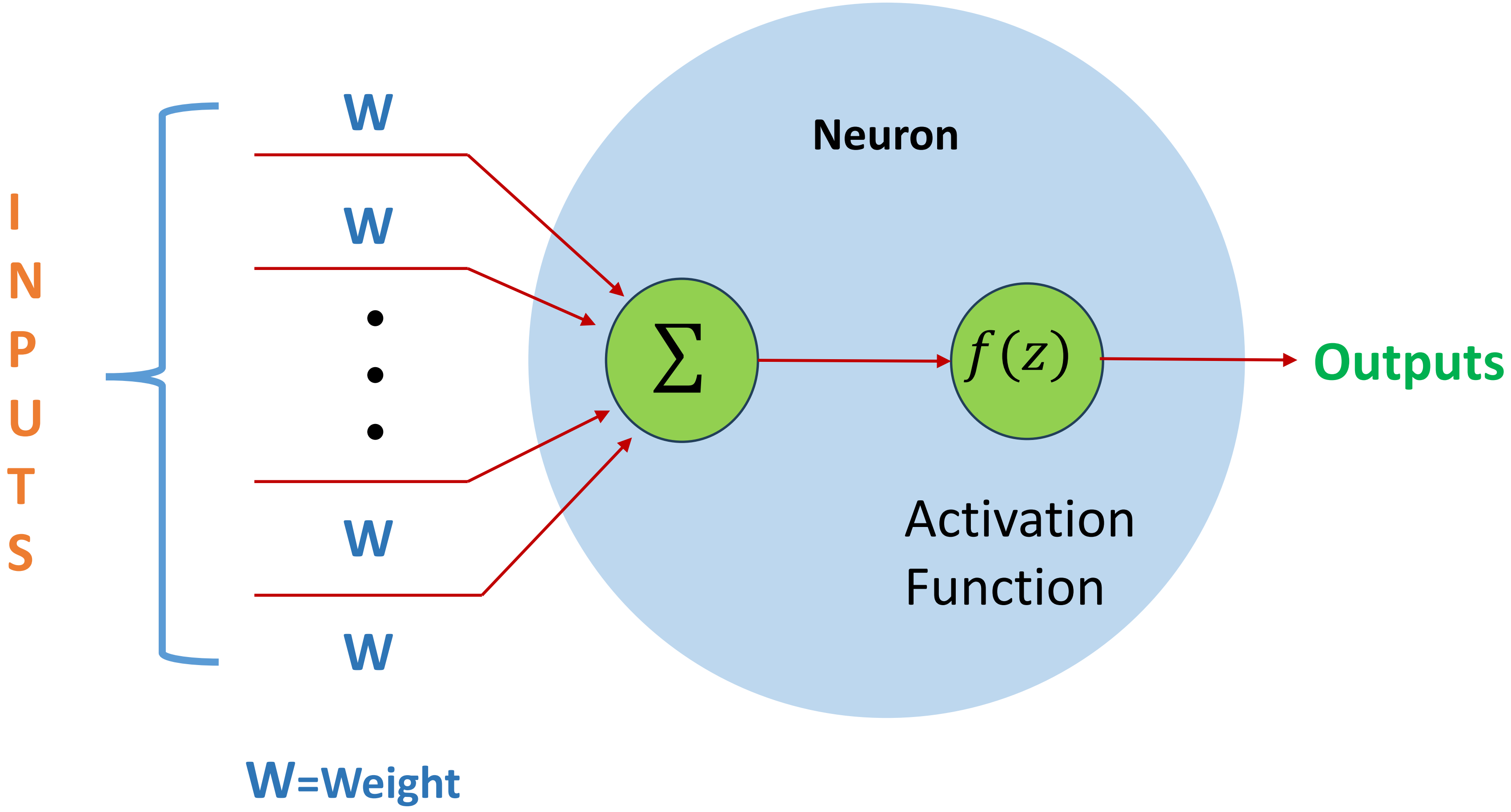


Linear threshold units can be used for simple binary classification.

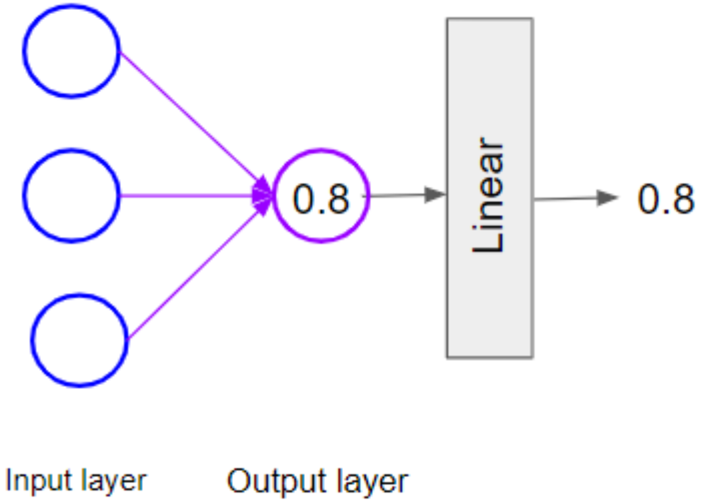
It computes a linear combination of the inputs and if the results exceeds a threshold, it outputs the positive class or else outputs the negative class.

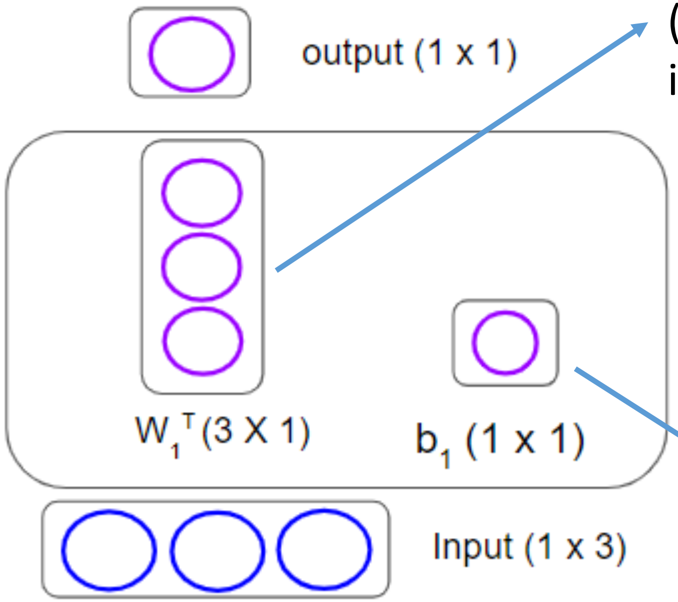
$$f(x) = \begin{cases} 1 & wx + b > 0 \\ 0 & otherwise \end{cases}$$

Artificial Neuron



Linear Regression as a Single-layer Neural Network



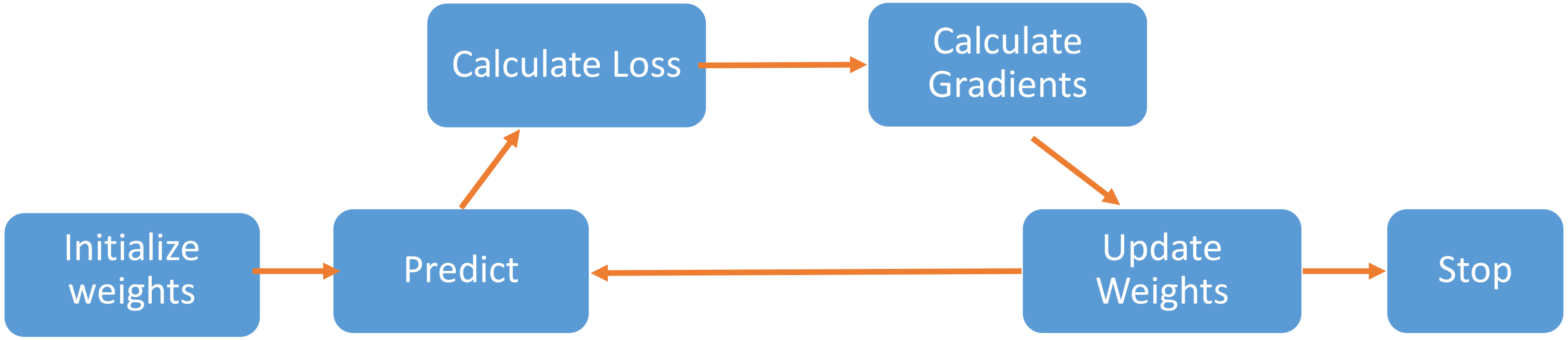


In Pytorch, shape of W: (# of Outputs, # of inputs) = (1, 3)

One for each output

- Number of Neurons in Output Layer: One
- Activation Function for output Layer : Linear (None)
- Prediction $\hat{y} = b + \hat{w}_1 x_1 + \hat{w}_2 x_2 + \cdots + \hat{w}_n x_n$
- Loss Function: Mean Squared Error

Recap- Gradient Descent

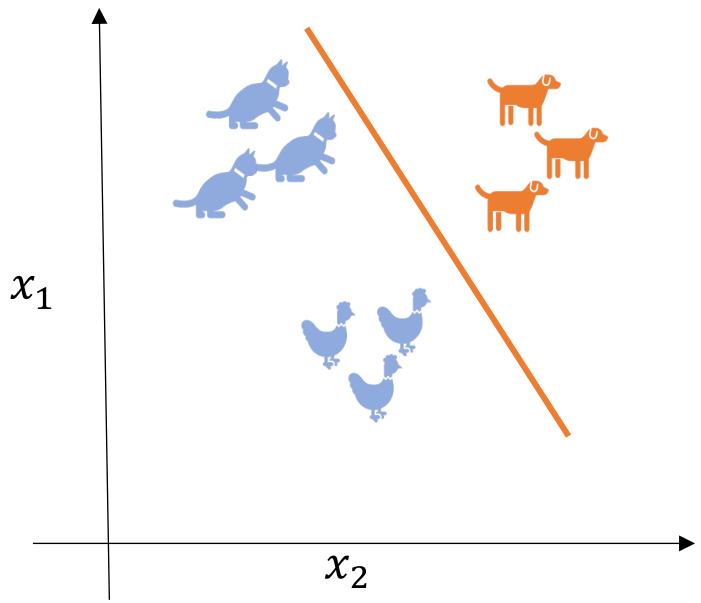


Goal: Optimize model performance by finding suitable weights.

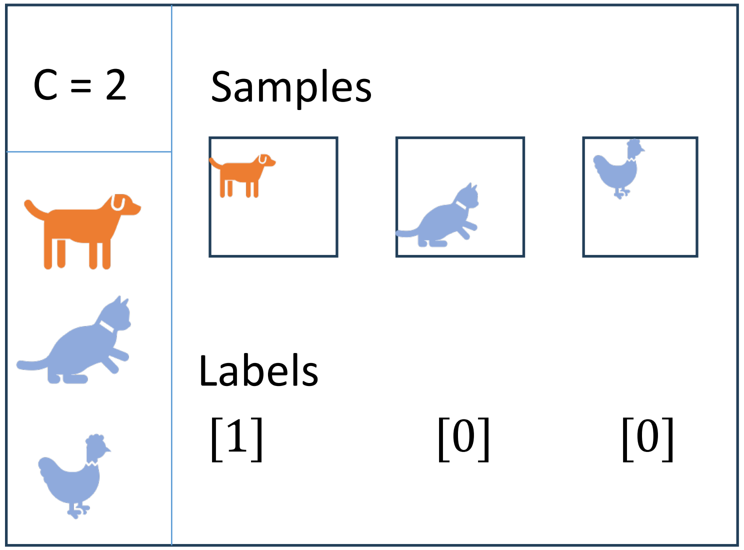
- Define a loss function - a lower value indicates better model performance.
- Calculate gradients - these represent the change in loss with respect to weights. Positive gradients imply a decrease in weight to minimize loss, while negative gradients indicate an increase in weight.
- Utilize gradients to guide weight adjustments, leading to continuous improvement in model performance.

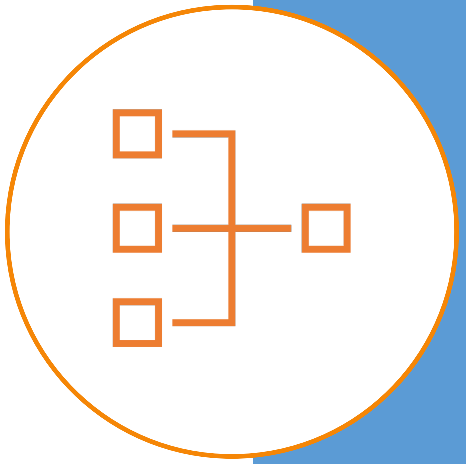
Binary Classification (pick one from two labels)

$$\hat{y} = b + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$



Dog or No Dog





Math (Logistic Regression)

p: probability of class "1"

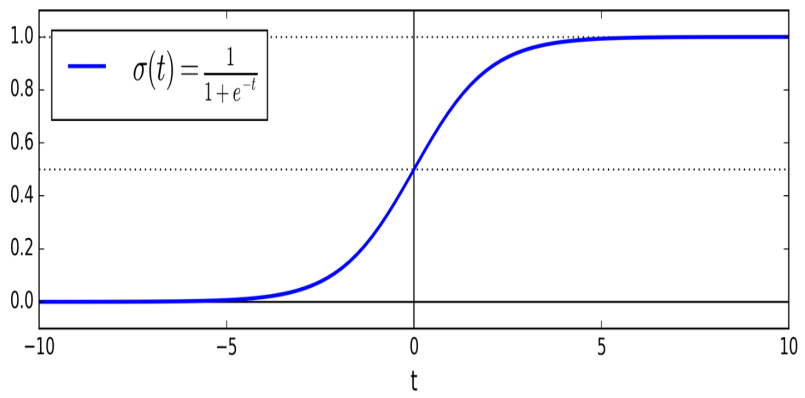
Need to relate p to predictors with a function that guarantees $0 \leq p \leq 1$

The standard linear function does not

Fix: logistic response function (logit/sigmoid)

- $\hat{z} = \hat{b} + \widehat{w}_1 x_1 + \widehat{w}_2 x_2 + \cdots + \widehat{w}_n x_n$
- $\hat{p} = \sigma(\hat{z}) = \frac{e^{\hat{z}}}{1+e^{\hat{z}}} = \frac{1}{1+e^{-\hat{z}}}$

Logit (Sigmoid) Function



Logit (sigmoid) function is a function that returns the result between 0 and 1.

Prediction $\hat{y} = \begin{cases} 0 & if\ \hat{p} < threshold \\ 1 & if\ \hat{p} \geq threshold \end{cases}$, typically, $threshold$ = 0.5

$\hat{p} < 0.5\ , when\ z < 0$

$\hat{p} \geq 0.5\ when\ z \geq 0$

This means Logistic Regression predicts 1 if z is positive and 0 if it is negative

Binary Cross Entropy Loss/Logistic Loss function

Loss function single training instance :

$$c(\theta) = \begin{cases} -\log(\widehat{p}) & if\ y = 1 \\ -\log(1-\hat{p}) & if\ y = 0 \end{cases}$$

Intuition

$c(\theta) = 0\ , if\ \hat{p} = y,$

$c(\theta) \rightarrow \infty$, if $y = 1\ and\ \hat{p} \rightarrow 0$

$c(\theta) \rightarrow \infty$, if $y = 0\ and\ \hat{p} \rightarrow 1$

Alternatively:

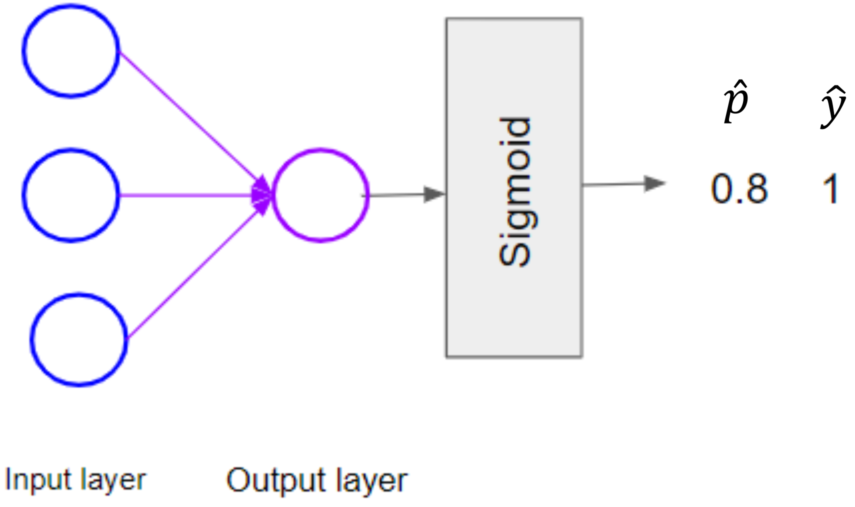
$$c(\theta) = -y\log(\hat{p}) - (1-y)\log(1-\hat{p})$$

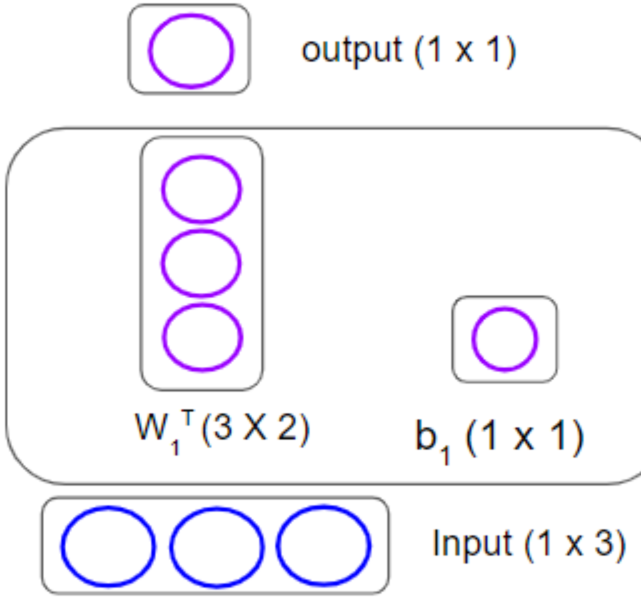
The cost function over the whole triaging set:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-\hat{p}^{(i)}\right)\right]$$

Intuition: Minimizing the loss will maximize the probability of the true label (class)

Logistic Regression as a Single-layer Neural Network





Number of Neurons in Output Layer: One

Activation Function for output Layer : Sigmoid

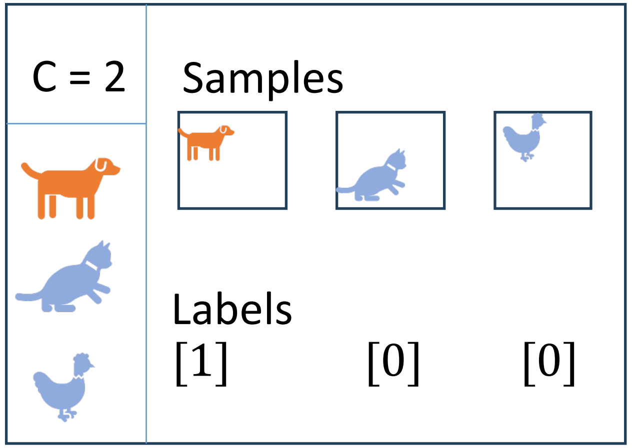
Prediction $\hat{y} = \begin{cases} 0 & if\ \hat{z} < 0 \\ 1 & if\ \hat{z} \geq 0 \end{cases} = \begin{cases} 0 & if\ \hat{p} < 0.5 \\ 1 & if\ \hat{p} \geq 0.5 \end{cases}$

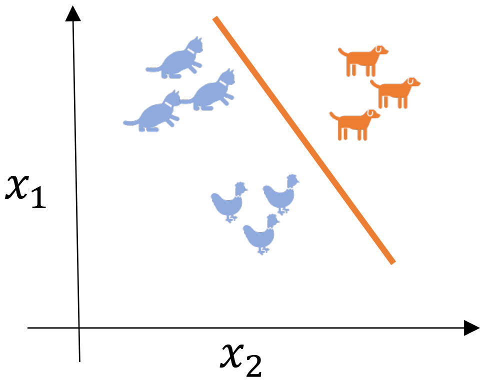
Loss Function : Binary Cross Entropy Loss (Logistic Loss Function)

Multiclass Classification (Softmax Regression)
(pick one from more than two labels)

Binary Classification

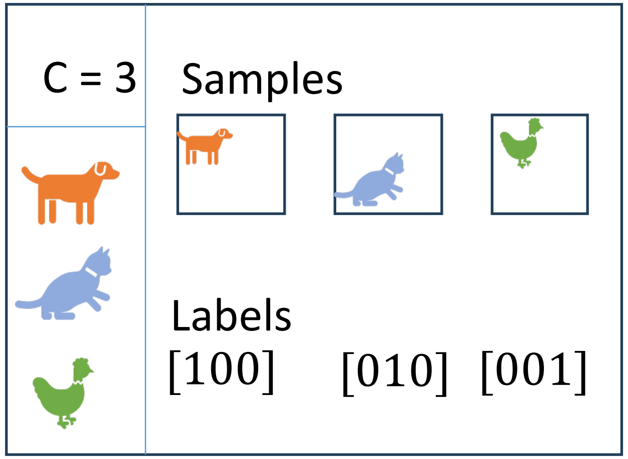
Dog or No Dog

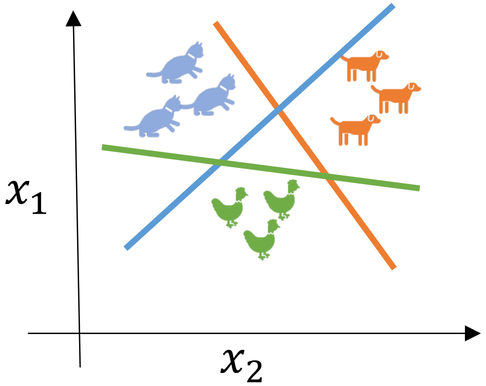




Multi-class Classification

Dog or Cat or Hen





Multiclass Classification (Softmax Regression) (pick one from more than two labels)

Generalization of Logistic regression:

$\hat{z}_k(x) = \hat{b}^k + \hat{w}_1^k x_1 + \cdots + \hat{w}_n^k x_n$, each class has set of weights and bias.

Softmax Function:

$$\hat{p}_k = \frac{e^{z_k(x)}}{\sum_{j=1}^{k} e^{z_j(x)}}, for\ k = 2:\ p_1 = \frac{e^{z_1(x)}}{e^{z_1(x)} + e^{z_2(x)}}, p_2 = \frac{e^{z_2(x)}}{e^{z_1(x)} + e^{z_2(x)}}$$

$$\hat{y} = \underset{k}{argmax}\ \hat{z}_k(x) = \underset{k}{argmax}\ \hat{p}_k(x)$$

Negative Log Likelihood Loss/Cross Entropy Loss

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(\hat{\mathrm{p}}_{\mathrm{k}}^{(\mathrm{i})})$$

For two classes:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y_1^{(i)} log\left(\hat{\mathrm{p}}_1^{(\mathrm{i})}\right) + \mathrm{y}_2^{(\mathrm{i})}\log(\hat{\mathrm{p}}_2^{(\mathrm{i})})$$

$$y_2 = 1 - y_1$$
$$p_2 = 1 - p_1$$

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log\left(\hat{p}^{(i)}\right) + (1 - y^{(i)}) \log\left(1 - \hat{p}^{(i)}\right)\right]$$ Binary Cross Entropy Loss

Negative Log Likelihood Loss/Cross Entropy Loss

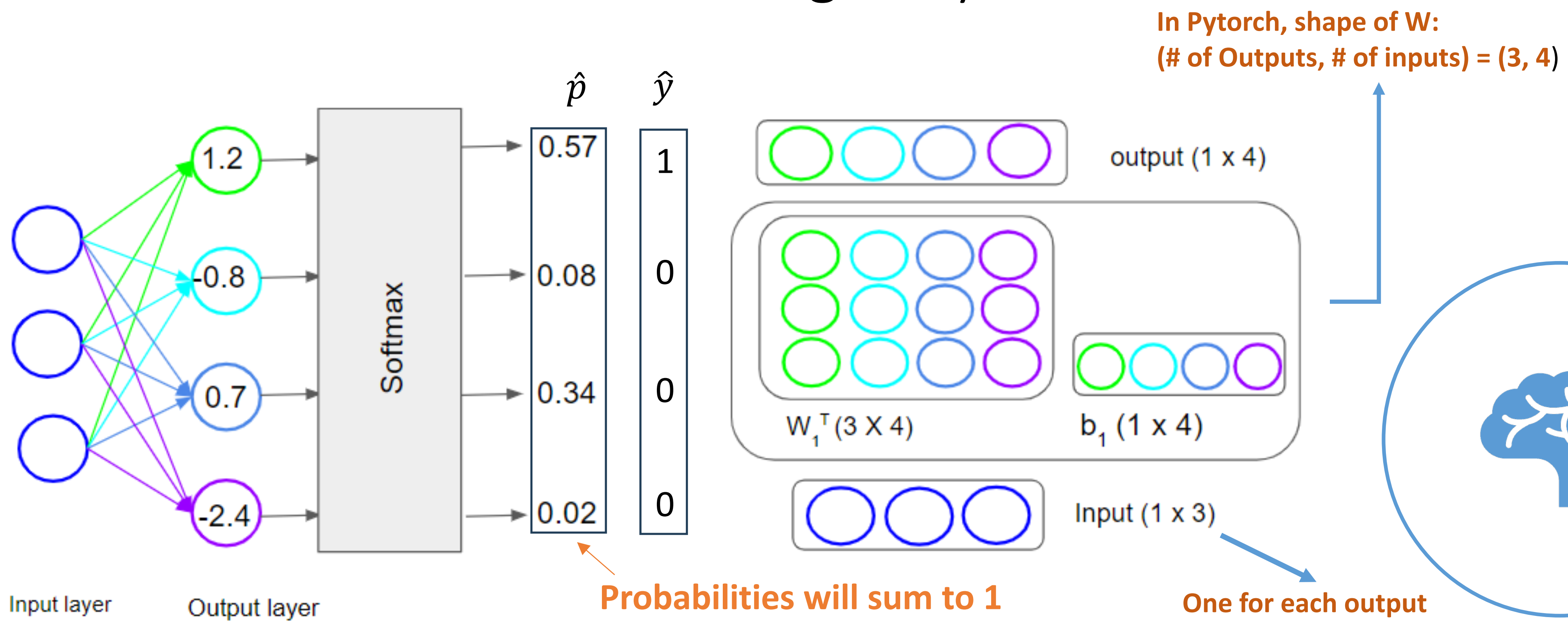
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(\hat{\mathrm{p}}_{\mathrm{k}}^{(\mathrm{i})})$$

Example for two observations, m = 2 and two classes, k = 2

	X	y [A, B]	Cross Entropy Loss	
	$\downarrow$	$\downarrow$	$\downarrow$	
	$Phat = soft\max(b + x_1w_1 + x_2w_2 + \ldots x_nw_n)$	y	sum (-y * log(phat)	
Observation # 1	[p1, p2]	[1, 0]	-log(p1)	mean()
Observation # 2	[p1, p2]	[0, 1]	-log(p2)	

Intuition: Minimizing the loss will maximize the probability of the true label (class)

Multiclass Classification as a Single-layer Neural Network



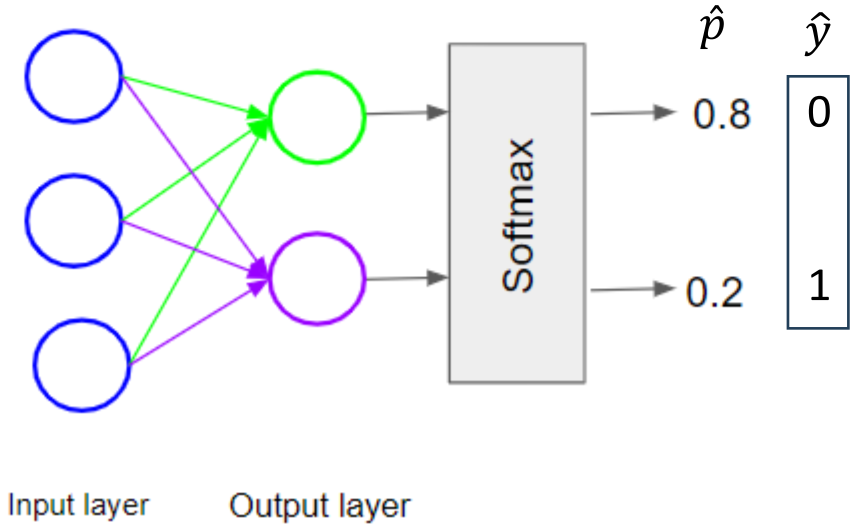
Number of Neurons in Output Layer: Number of Classes

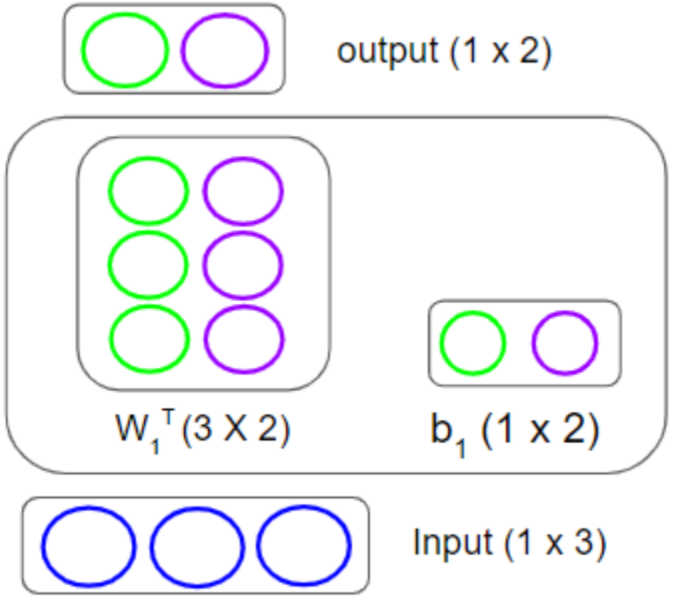
Activation Function for output Layer : Softmax Function

Prediction $\hat{y} = \underset{k}{argmax}\, z_k(x)$ $(Class\ which\ has\ the\ maximum\ probbaility\ value\ or\ z_k\ value)$

Loss Function : Cross Entropy Loss

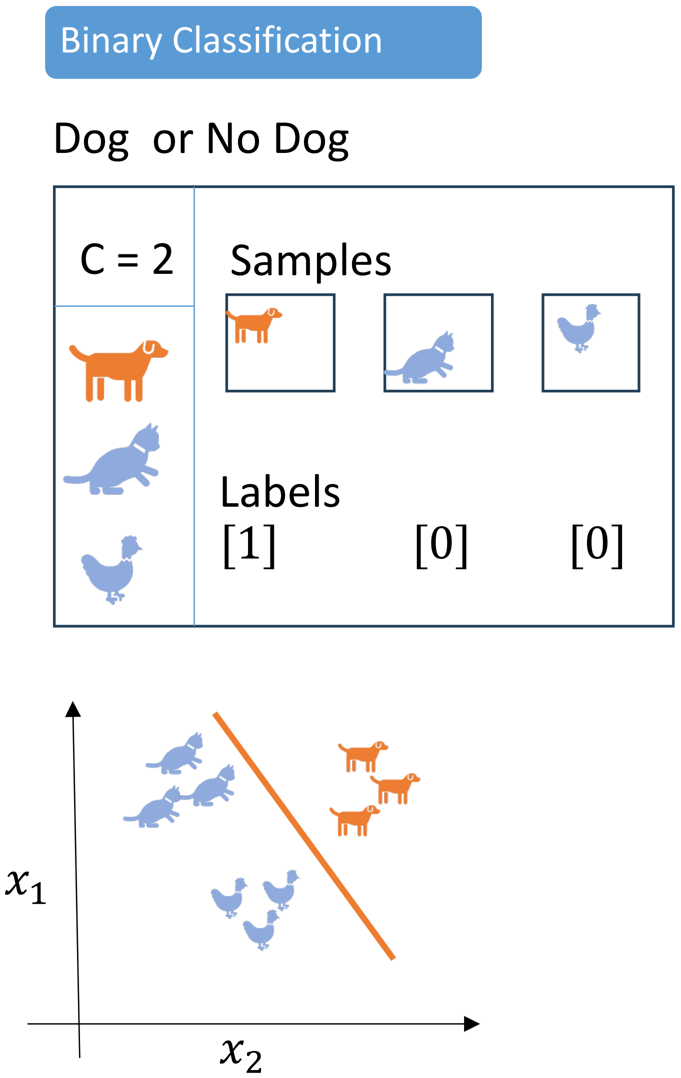
Binary Classification with Softmax and Cross Entropy Loss function

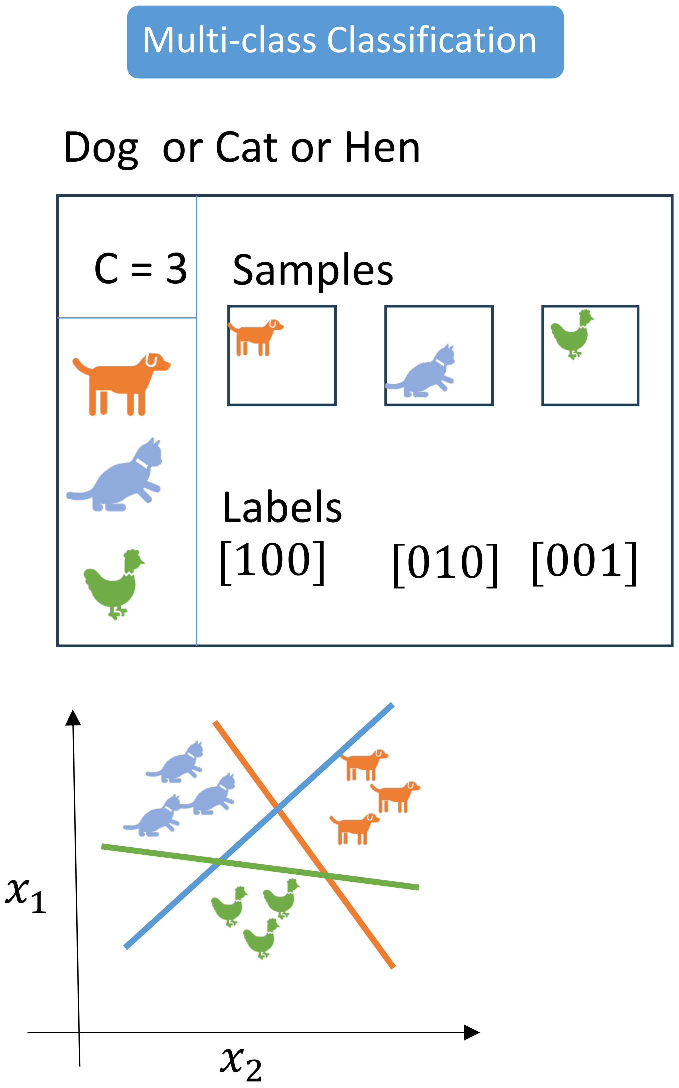


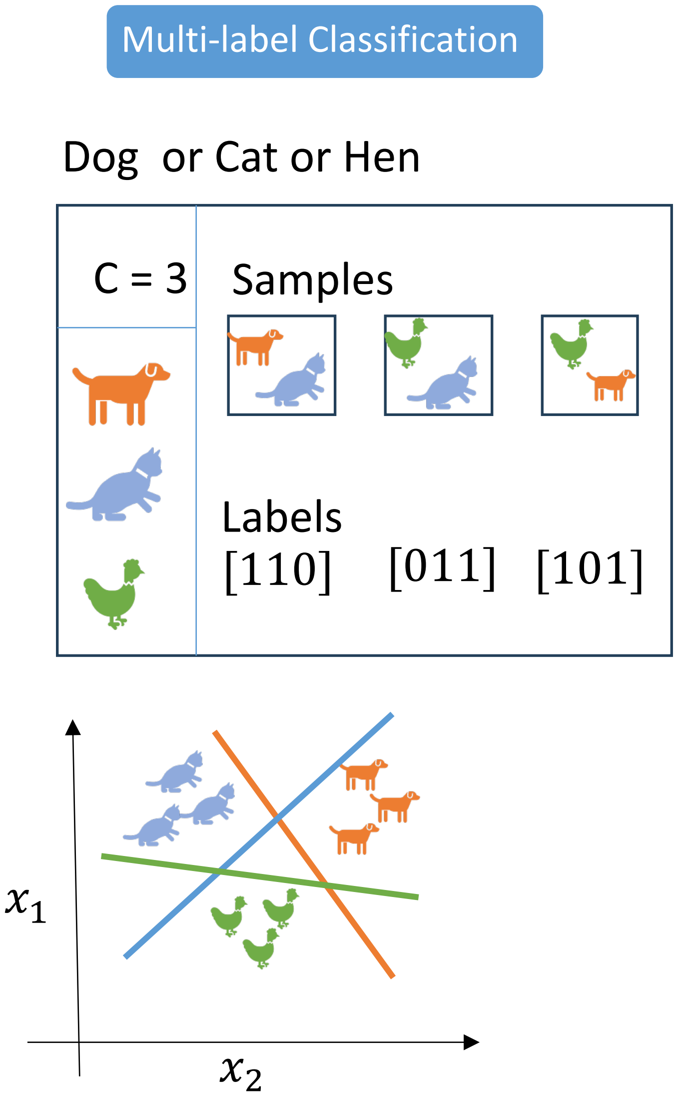


- Number of Neurons in Output Layer: Two
- Activation Function for output Layer : Softmax Function
- Prediction $\hat{y} = \underset{k}{argmax}\, z_k(x)$ $(Class\ which\ has\ the\ maximum\ probbaility\ value\ or\ z_k\ value)$
- Loss Function : Cross Entropy Loss

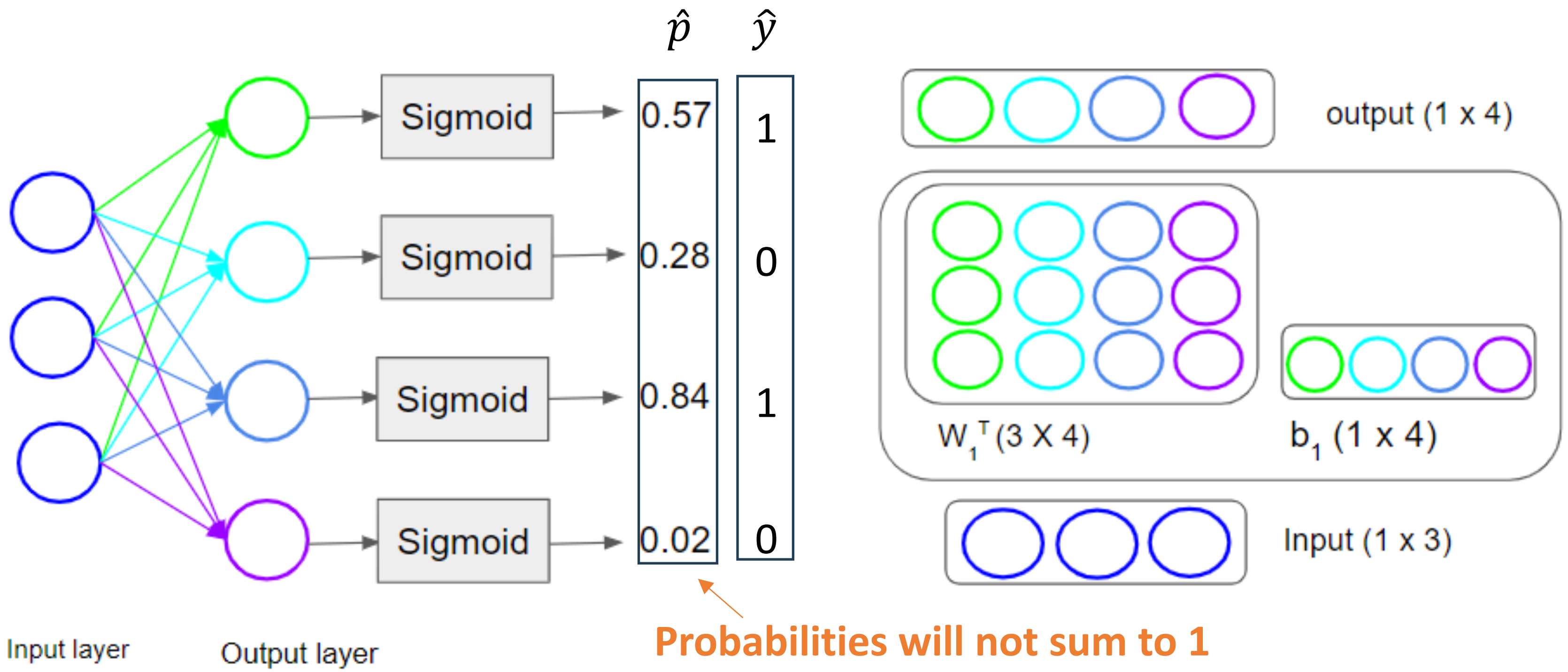
Multilabel Classification (Softmax Regression)







Multilabel Classification as a Single-layer Neural Network



Number of Neurons in Output Layer: Number of Classes

Activation Function for output Layer : Sigmoid

Prediction $\hat{y} = \begin{cases} 0 & if\ \hat{z} < 0 \\ 1 & if\ \hat{z} \geq 0 \end{cases} = \begin{cases} 0 & if\ \hat{p} < 0.5 \\ 1 & if\ \hat{p} \geq 0.5 \end{cases}$

Loss Function : Binary Cross Entropy Loss (Logistic Loss Function)

Linear Classifiers

Predictions in all the previous models can be made based on:

$\hat{z}_k(x) \ = \ \hat{b}^k + \ \hat{w}_1^k x_1 + \cdots + \hat{w}_n^k x_n$

Output equation given x_1 and x_2, is the equation of a line

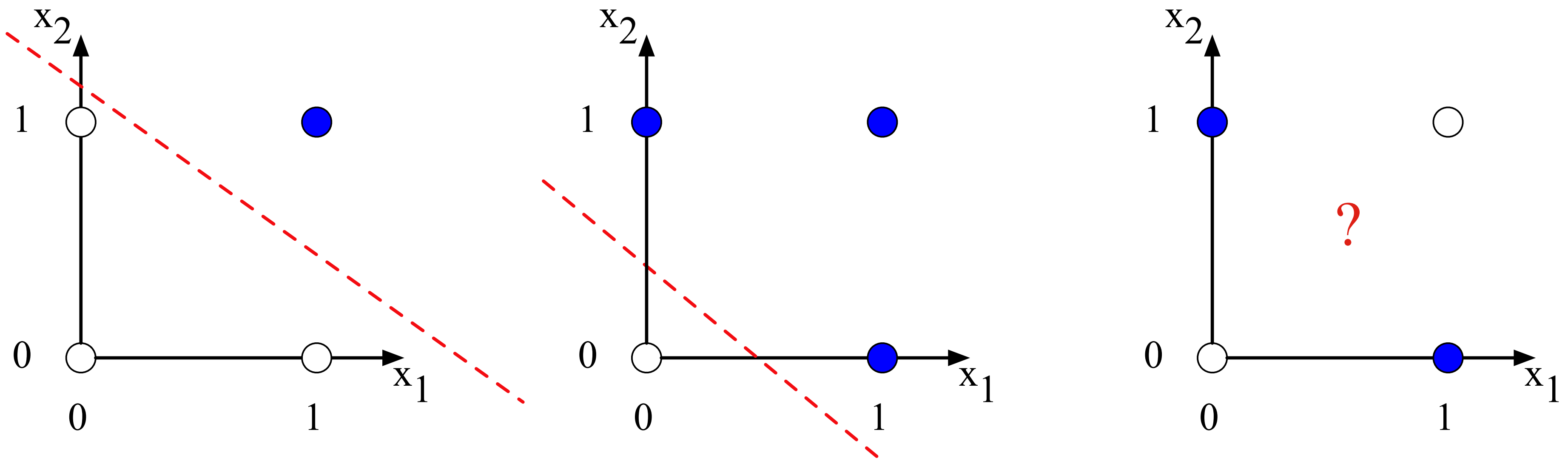
$w_1 x_1 + w_2 x_2 + b = 0$

(in standard linear format: $\quad x_2 = (-w_1/w_2)x_1 + (-b/w_2) \quad$)

This line acts as a decision boundary

- 0 if input is on one side of the line
- 1 if on the other side of the line

Decision Boundaries

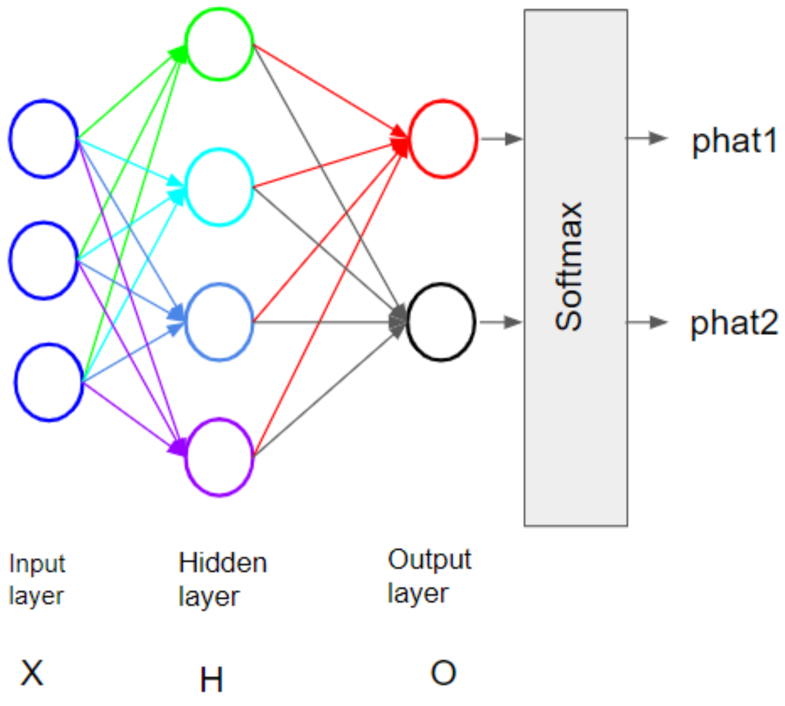


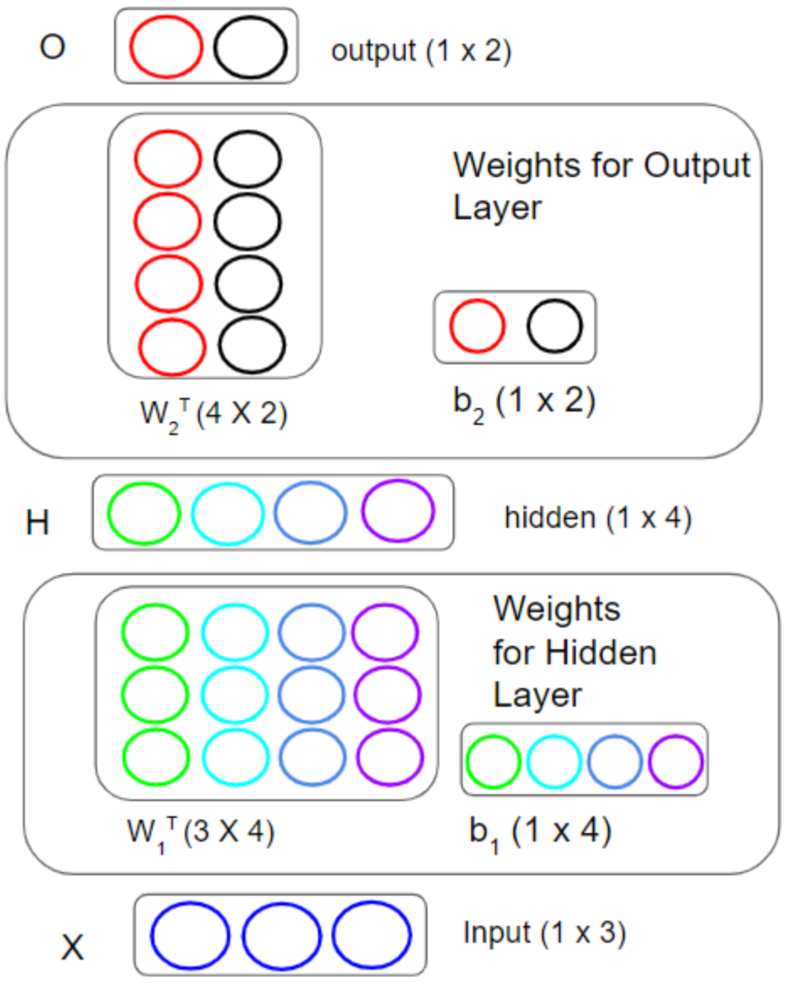
a) x_1 AND x_2 b) x_1 OR x_2 c) x_1 XOR x_2

XOR is not a **linearly separable** function!

Slide Credit: https://web.stanford.edu/~jurafsky/slp3/

Multilayer Perceptron





Multilayer Perceptron

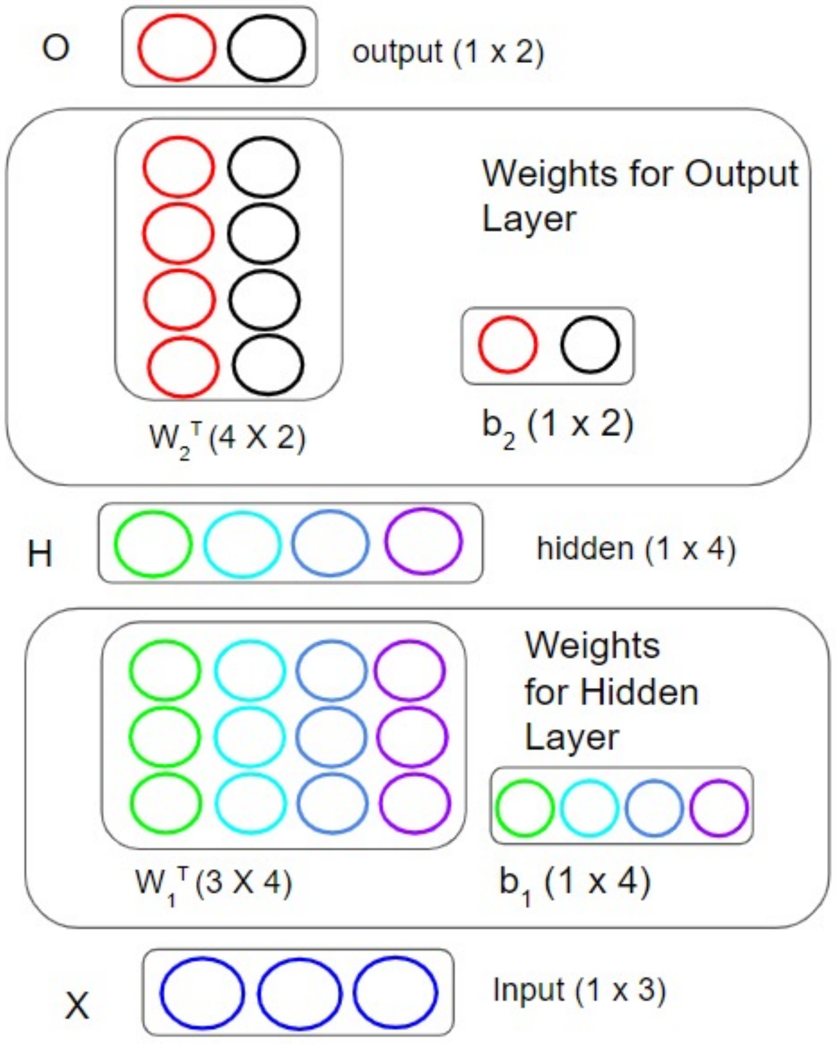
$$H = XW_1^T + b_1$$
$$O = HW_2^T + b_2$$

$$O = (XW_1^T + b_1)W_2^T + b_2$$
$$O = XW_1^T W_2^T + b_1 W_2^T + b_2$$
$$O = XW + b$$

W= $W_1^T W_2^T$
$$b = b_1 W_2^T + b_2$$

Apply Non-Linear Function to Hidden Layers

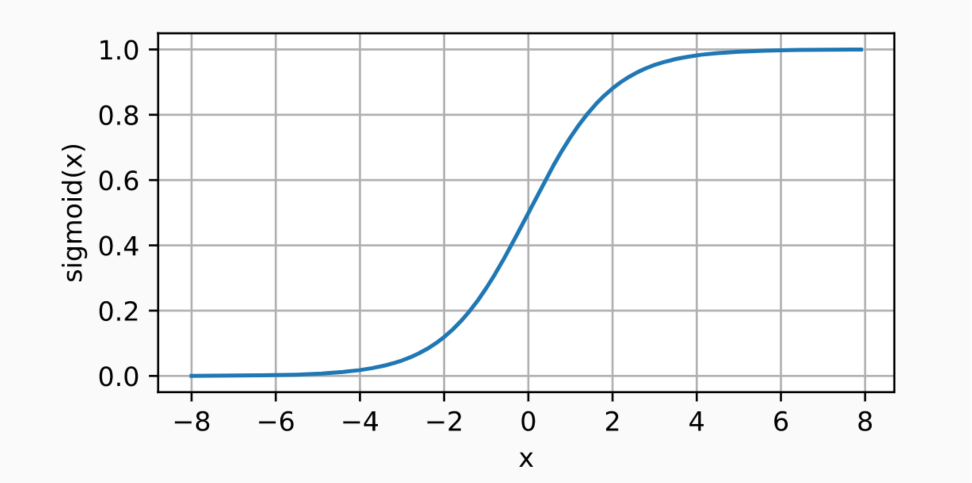
$$H = \sigma(XW_1^T + b_1)$$
$$O = HW_2^T + b_2$$



Sigmoid Activation

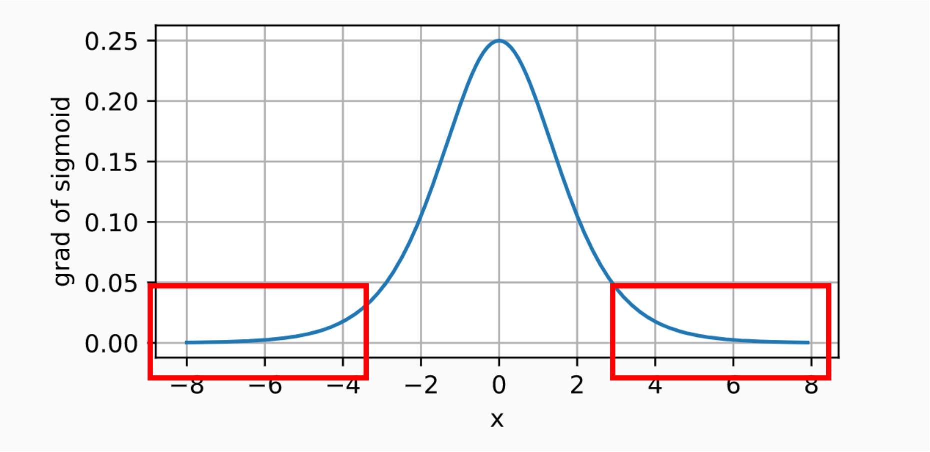
$$sigmoid(x) = \frac{1}{1 + \exp(-x)}$$

$$\frac{d}{dx}\text{sigmoid}(x) = \frac{\exp(-x)}{(1 + \exp(-x))^2}$$



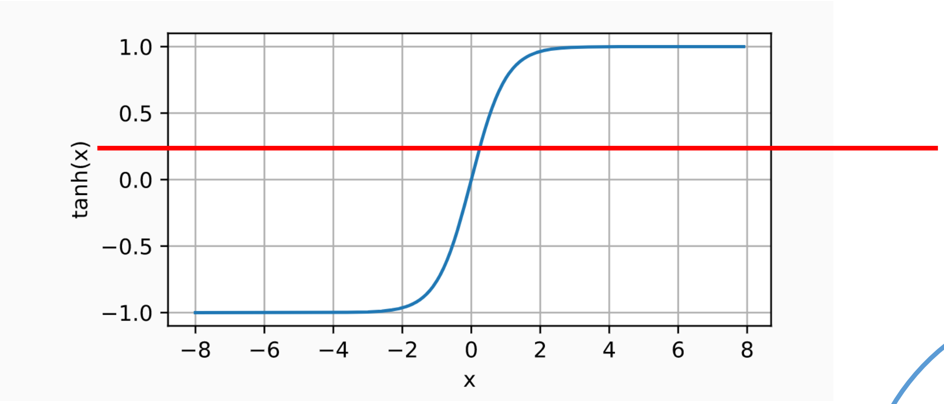
$$\frac{d}{dx}sigmoid(x) = \frac{\exp(-x)}{(1 + \exp(-x)^2}$$

$$= sigmoid(x)(1 - sigmoid(x))$$



Tanh Activation

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$

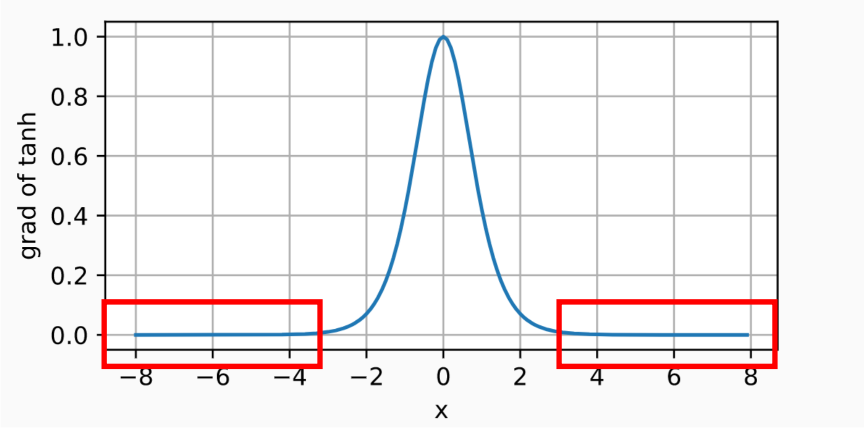


- Output value ranges from –1 to 1 (instead of 0 to 1 in the case of the sigmoid function).

- That range tends to make each layer's output more or less centered around 0 at the beginning of training, which often helps speed up convergence

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x).$$

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$$



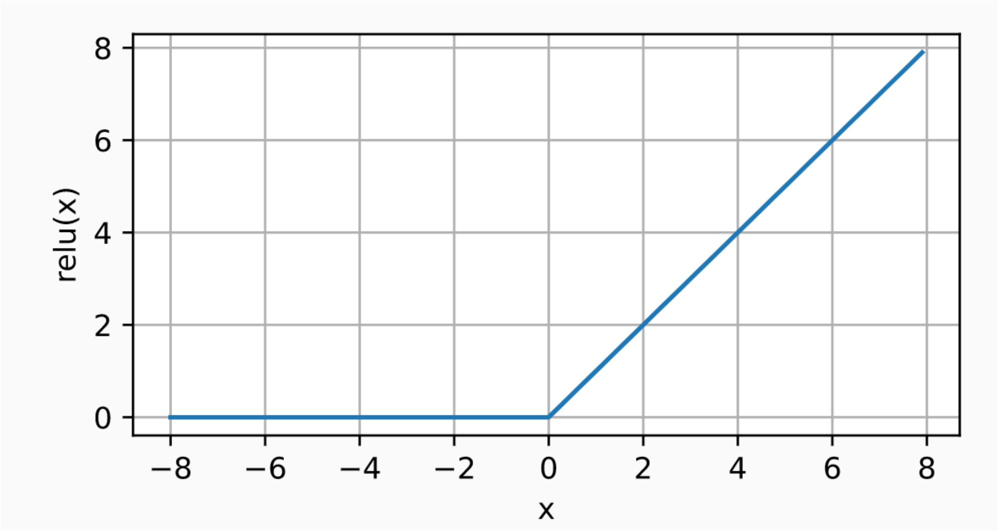
ReLU Activation

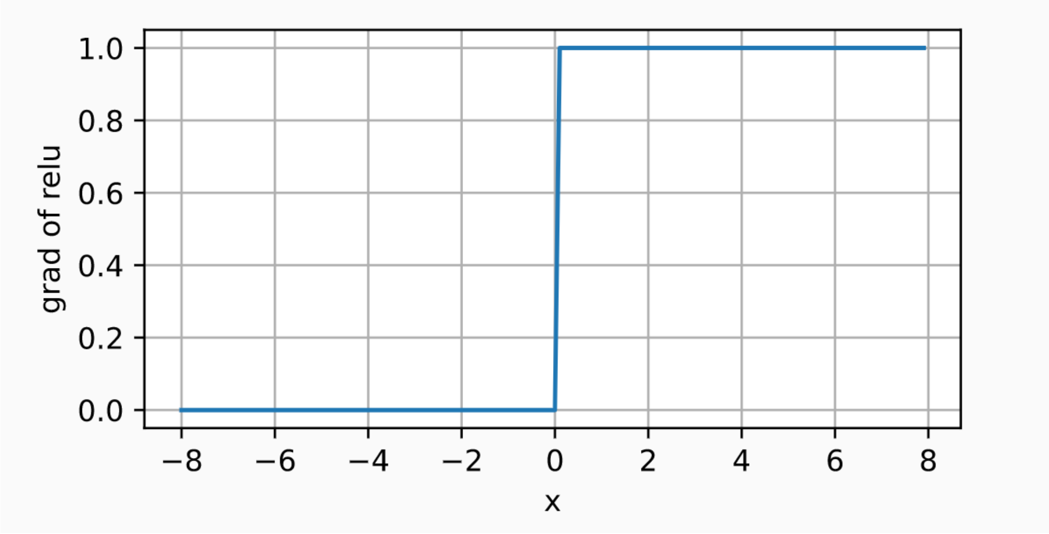
$$ReLU(x) = \max(x, 0)$$

It works very well and has the advantage of being fast to compute, so it has become the default.

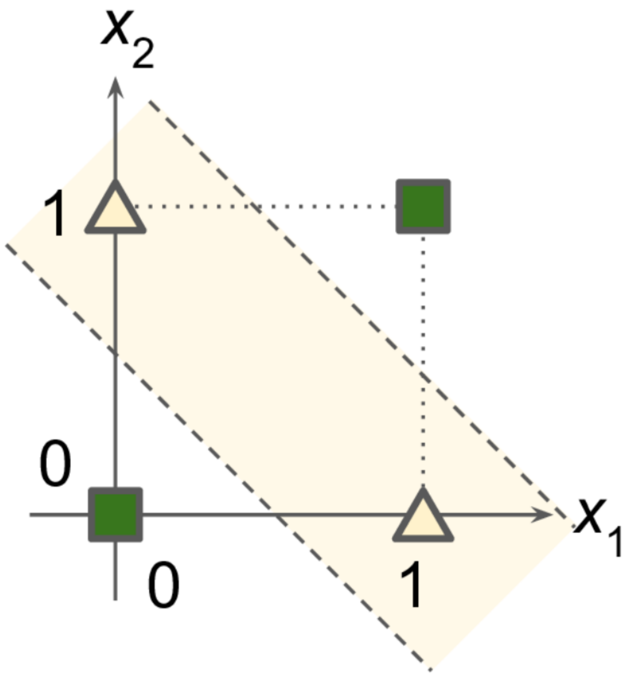
$$\frac{d}{dx} ReLU(x) = \begin{cases} 0, if\ x < 0 \\ 1, if\ x \geq 0 \end{cases}$$

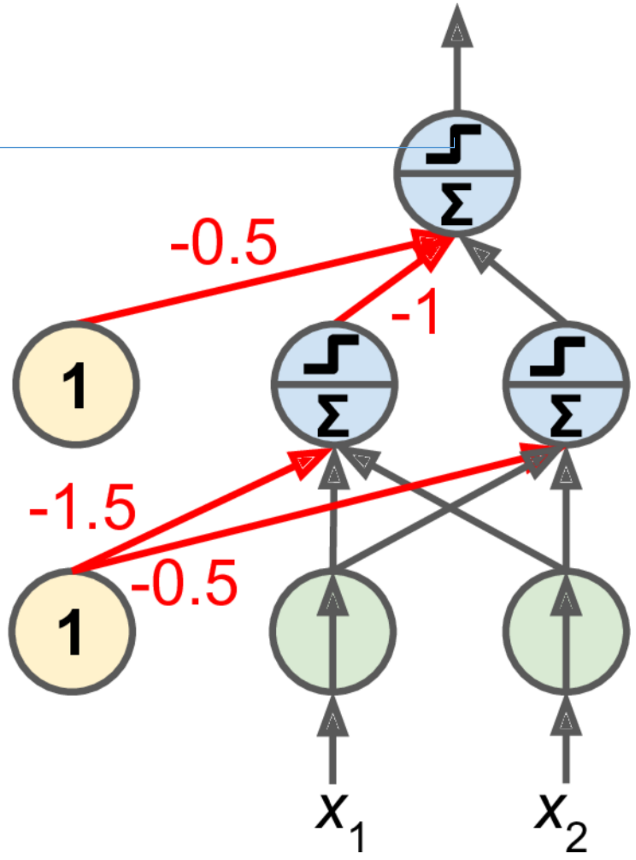
Note: derivative is not defined at $x = 0$





MLP that solves XOR problem



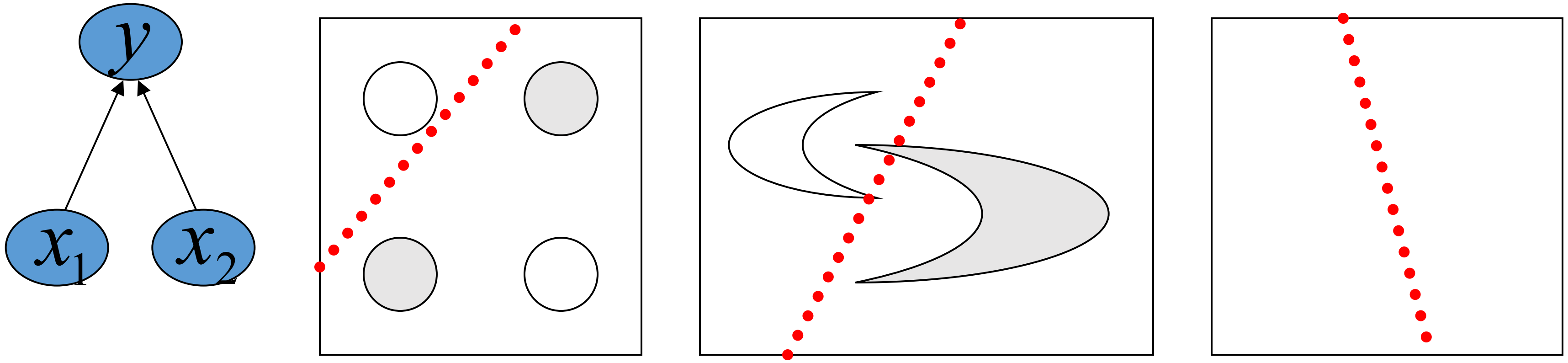


- with inputs (0, 0) or (1, 1), the network outputs 0.
- and with inputs (0, 1) or (1, 0) it outputs 1.
- All connections have a weight equal to 1, except the four connections where the weight is shown.

$$\text{heaviside}\,(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

Decision Boundary

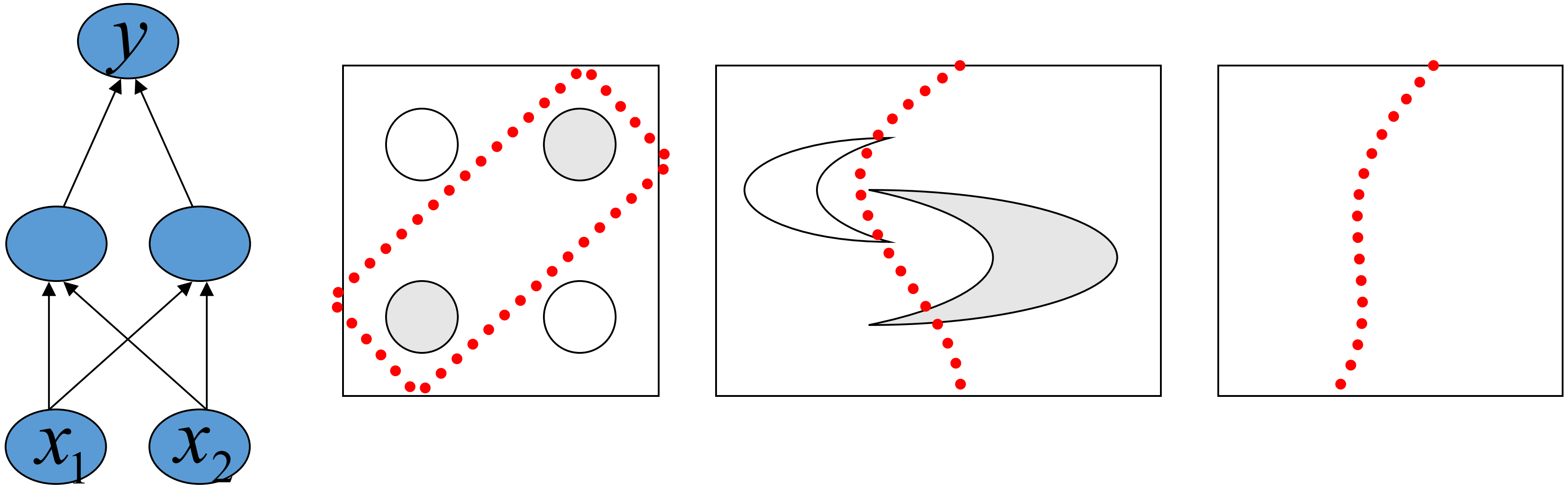
- 0 hidden layers: linear classifier
  - Hyperplanes



Slide Credit: © Eric Xing @ CMU, 2006-2011

Decision Boundary

- 1 hidden layer with nonlinear activation
    - Boundary of convex region (open or closed)

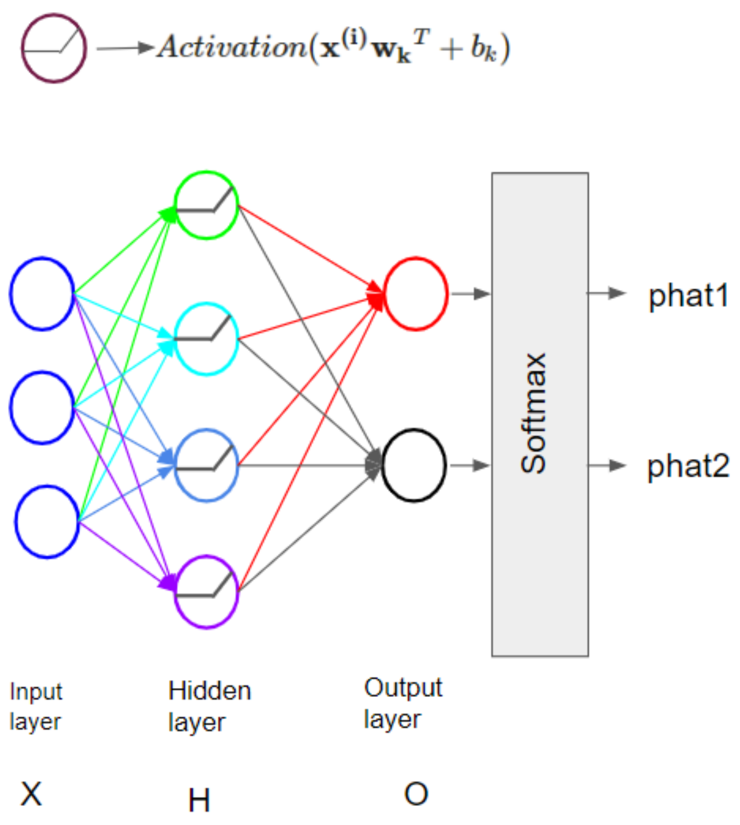


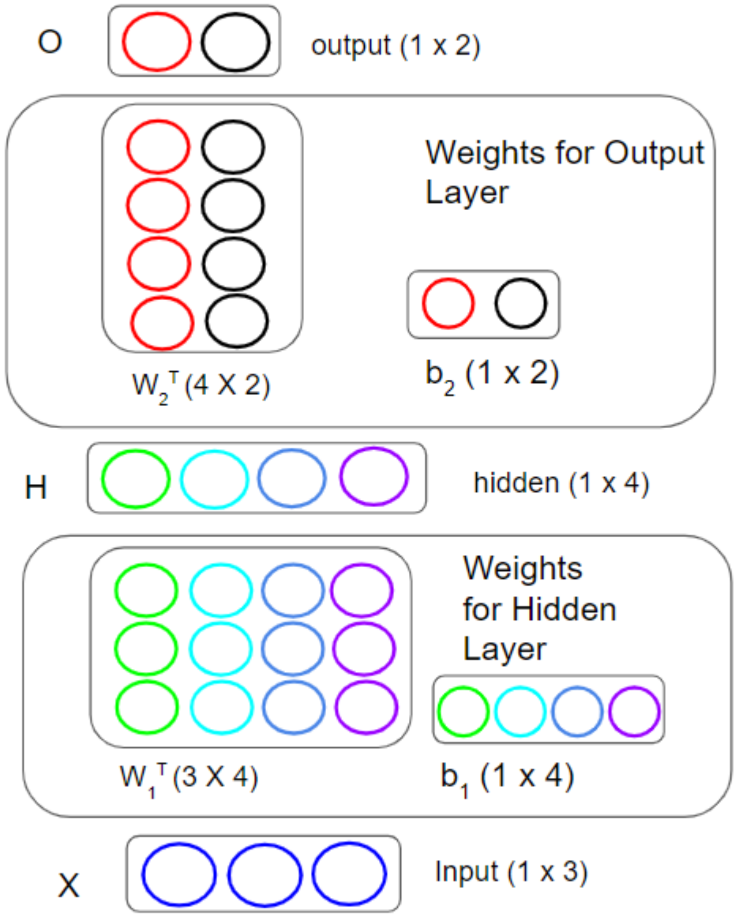
Multiclass Classification

$H = \sigma(XW_1^T + b_1)$

$O = HW_2^T + b_2$

$\widehat{P} = Softmax(O)$





Summary – Layers/Activations/Loss Functions

	Regression	Binary Classification	Multi-class Classification	Multi Label Classification
# input neurons	Depends on the problem (your data) (Number of input variables – (e.g. for 28x28 black & White images = 28*28 =784)			
# hidden layers	Depends on problem (Hyperparameter)			
# neurons per hidden layer	Depends on problem (Hyperparameter)			
# output neurons	One	One (Two)	Number of Classes	Number of Classes
Hidden activation	Hyperparameter (sigmoid, tanh, ReLU, SELU, GELU, MISH etc.)			
Output activation	None	Sigmoid (Softmax)	Softmax	Sigmoid
Loss Function	MSE	Binary Cross Entropy (Cross-Entropy)	Cross-Entropy	Binary Cross Entropy

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:
   – Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   – Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \boxed{\nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)}$$

Slide Credit: © Eric Xing @ CMU, 2006-2011

Computation Graph – Efficient way to calculate gradients

A computation graph is a directed acyclic graph (DAG) used to represent mathematical expressions and operations in a structured manner. Each node in the graph represents an operation, like addition or multiplication, and the edges represent the data flow between these operations, typically carrying tensors or matrices.

Key benefits include:

- **Modularity**: Easy to break down complex expressions into simpler parts.
- **Differentiation**: Facilitates the process of computing gradients efficiently.

Computation graph – Logistic Regression

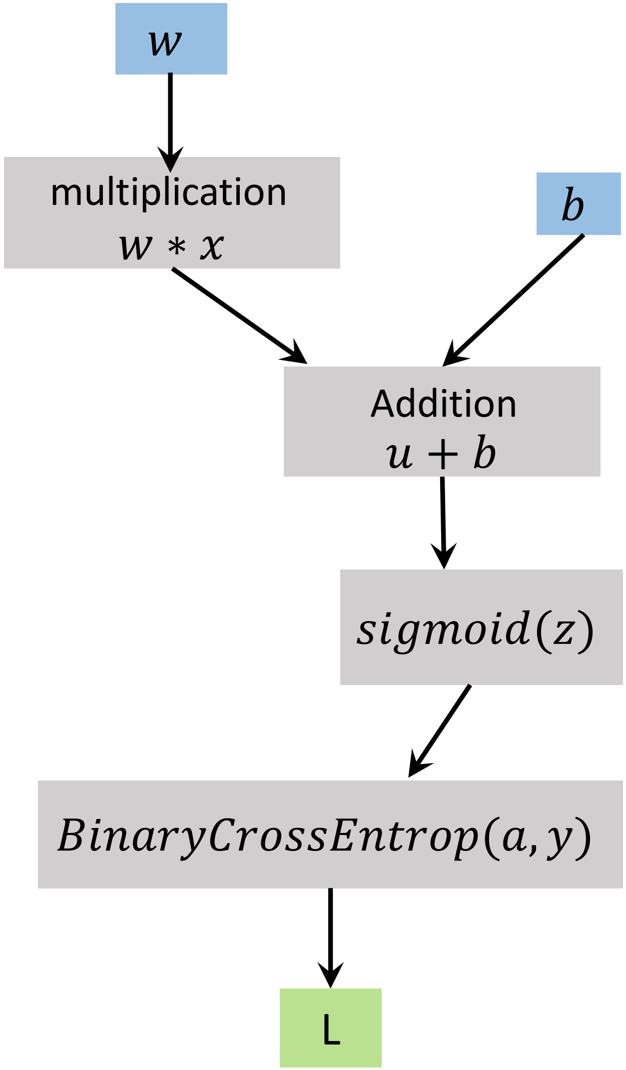
Computations:

$u = w * x$

$z = u + b$

$a = sigmoid(z)$

$L = -(y * \log(a) + (1 - y) * \log(1 - a))$



Logistic Regression – Forward Pass

Initial Model Parameters

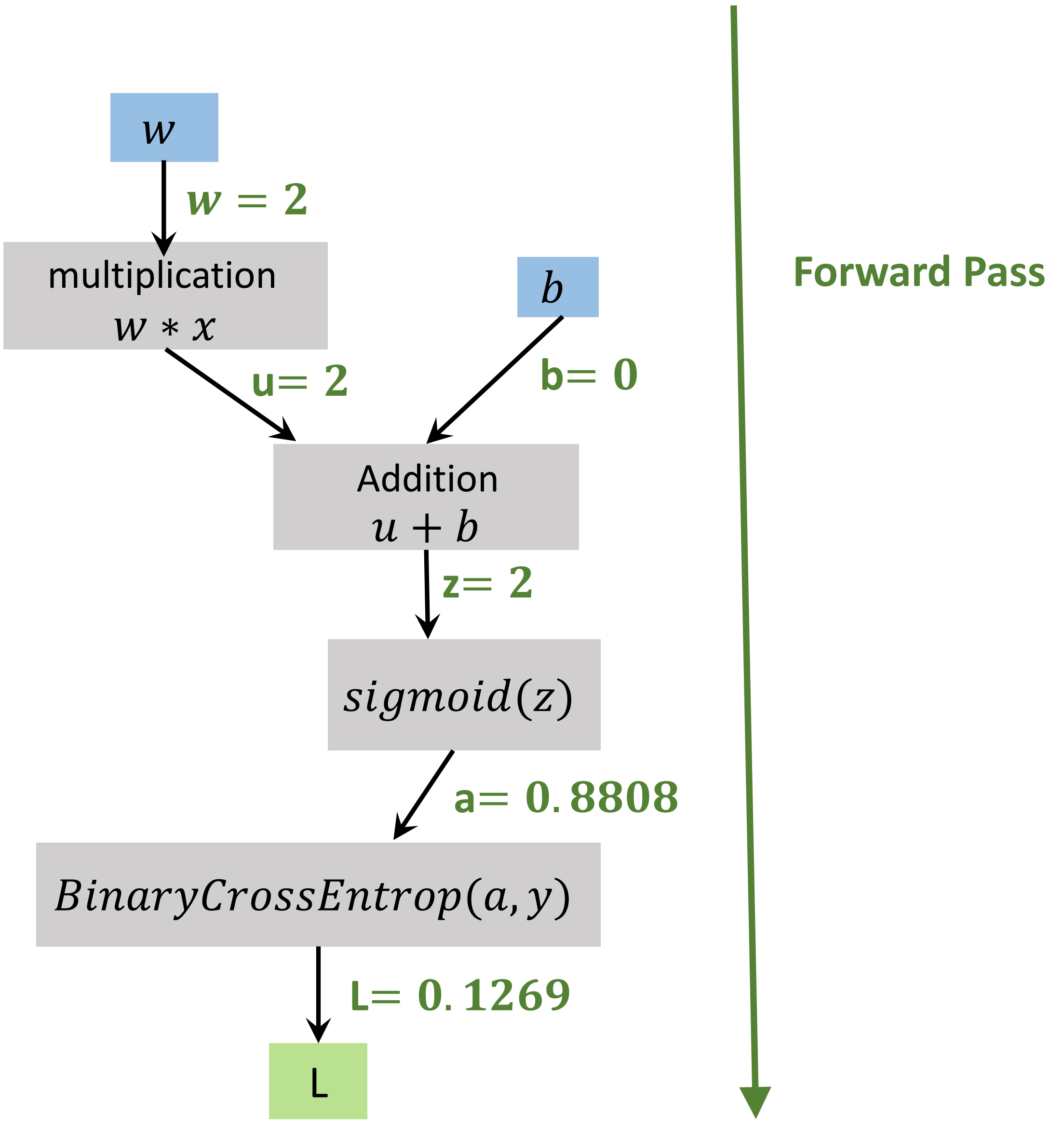
$w = 2$

$b = 0$

Data

$x = 1$

$y = 1$



Logistic Regression – Forward Pass

Computations:

$$u = w \ * x$$

$$z = u + b$$

$$a = sigmoid(z)$$

$$L = -(y * \log(a) + (1 - y) * \log(1 - a))$$

What we want
- Minimize Loss using Gradient descent

What we need: $\frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}$

The derivative $\partial L/\partial w$, tells us how much a small change in w affects L, while holding the other parameters constant

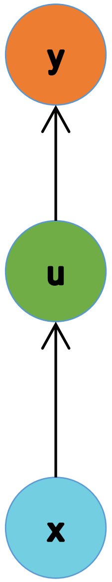
Computation graph – Chain Rule

$\boldsymbol{Given}$:
$y = g(u)\ and\ u = h(x)$

Chain Rule :

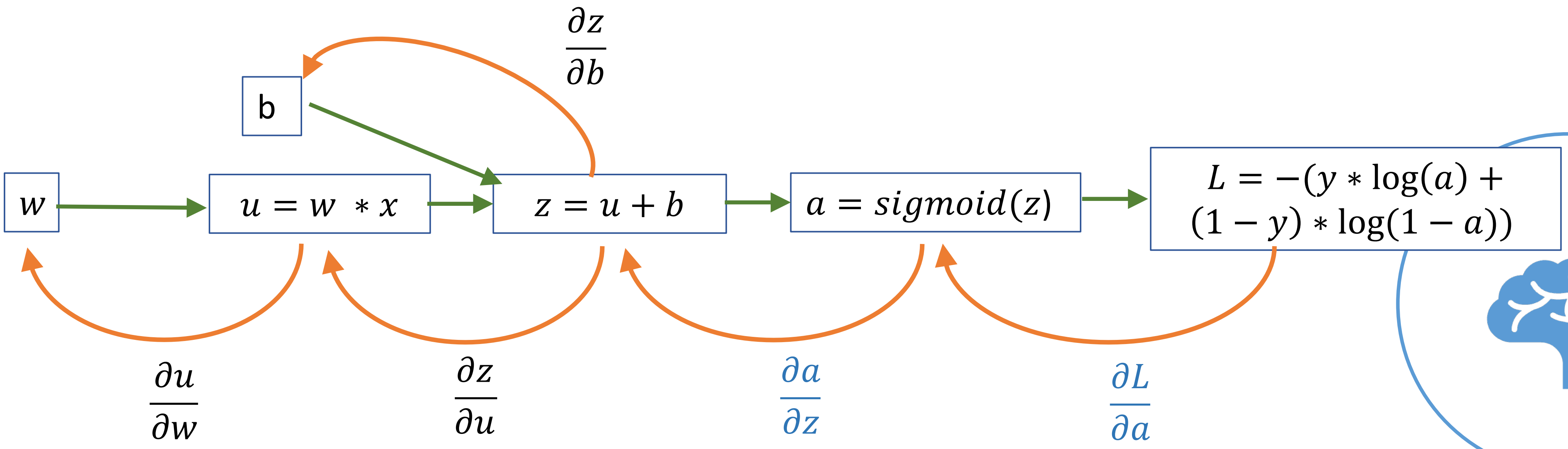
$$\frac{dy}{dx} = \frac{dy}{du} * \frac{du}{dx}$$

Backpropagation is just repeated application of the **chain rule**



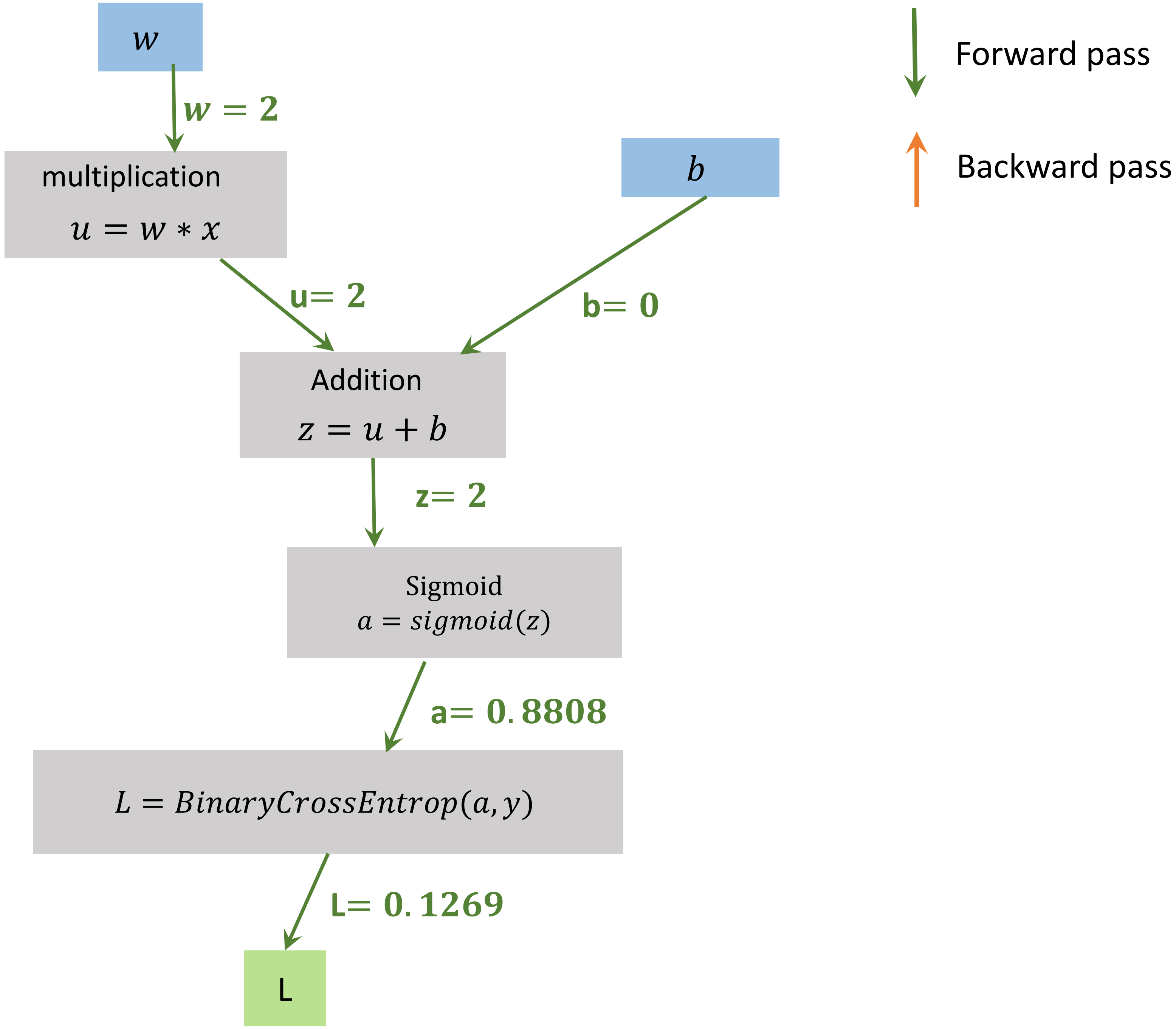
Chain Rule

$$u = f_1(x)\,, \qquad z = f_2(u), \qquad a = f_3(z), \qquad L = f_4(a)$$

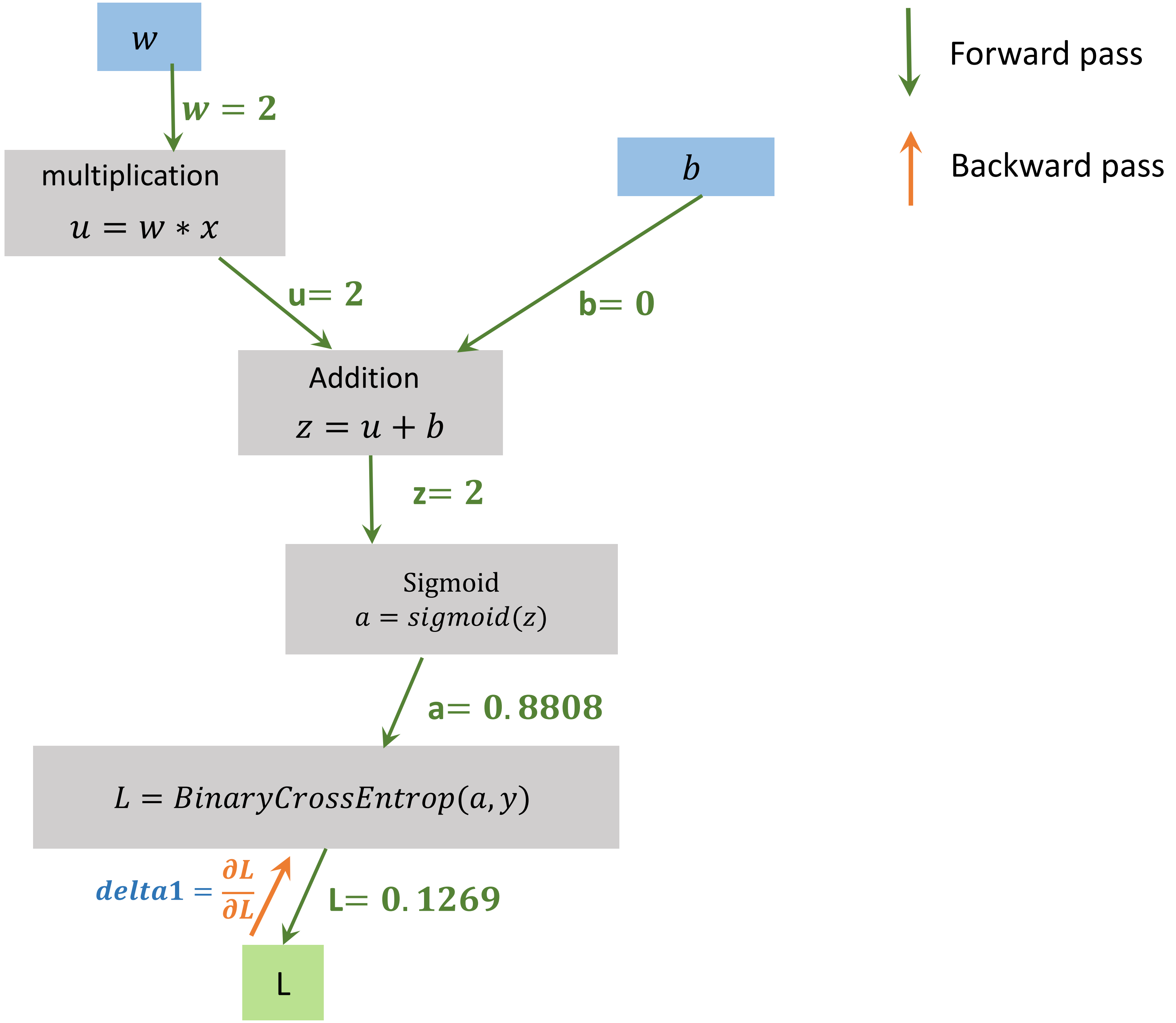


$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} * \frac{\partial a}{\partial z} * \frac{\partial z}{\partial u} * \frac{\partial u}{\partial w}$$

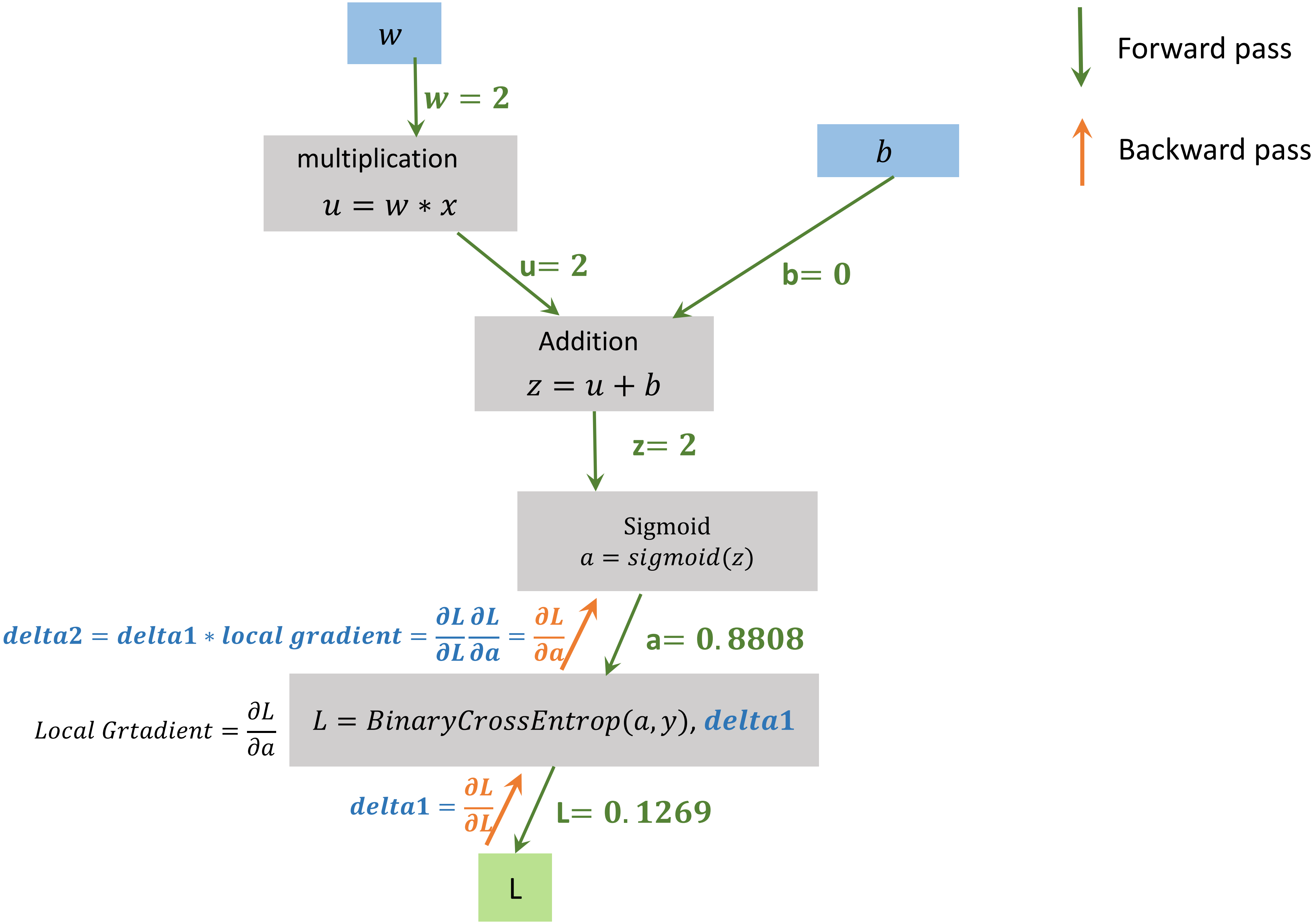
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a} * \frac{\partial a}{\partial z} * \frac{\partial z}{\partial b}$$



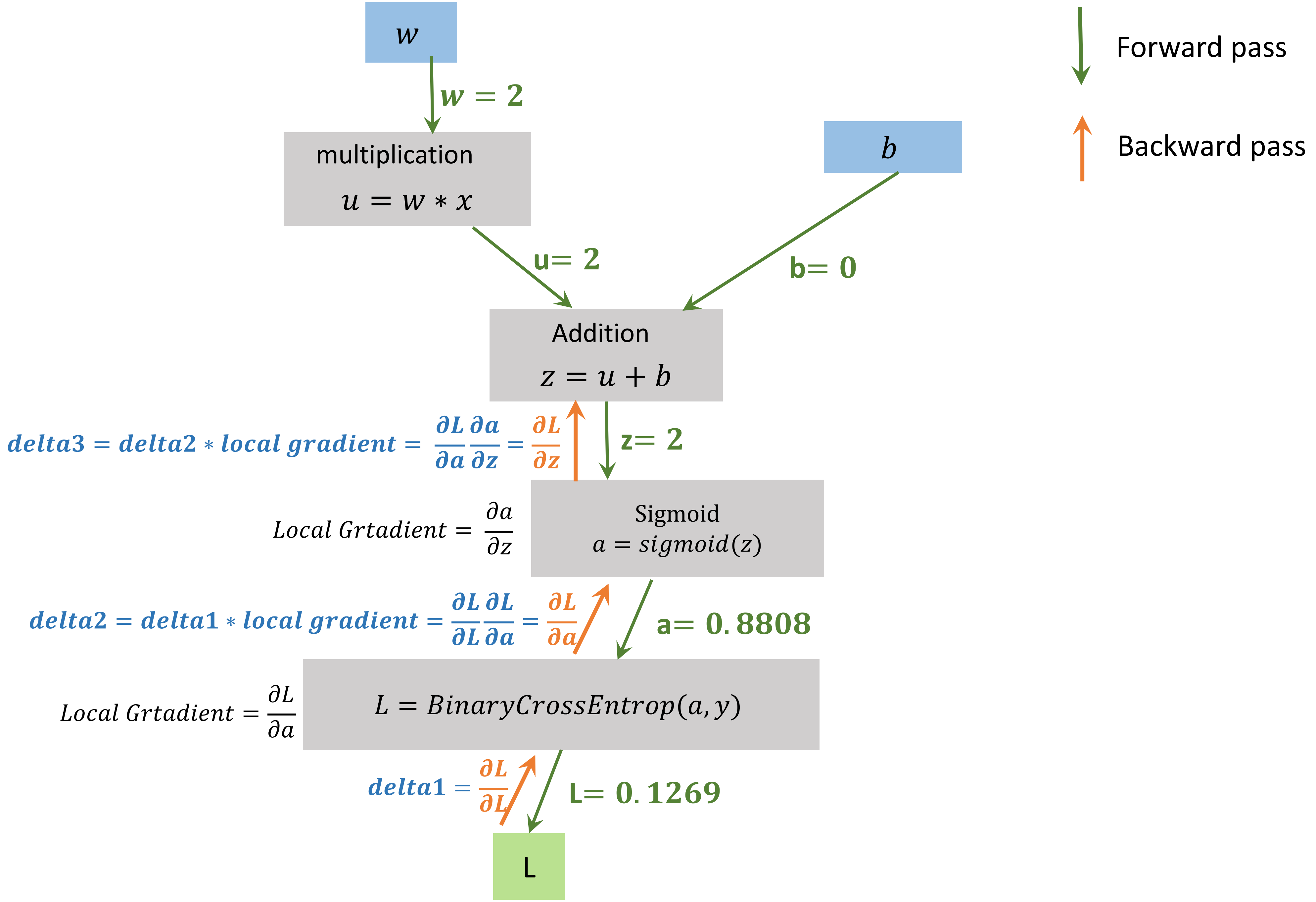
w
$w = 2$
multiplication
$u = w * x$
b
u= 2
b= 0
Addition
$z = u + b$
z= 2
Sigmoid
$a = sigmoid(z)$
a= 0.8808
$L = BinaryCrossEntrop(a, y)$
L= 0.1269
L
Forward pass
Backward pass



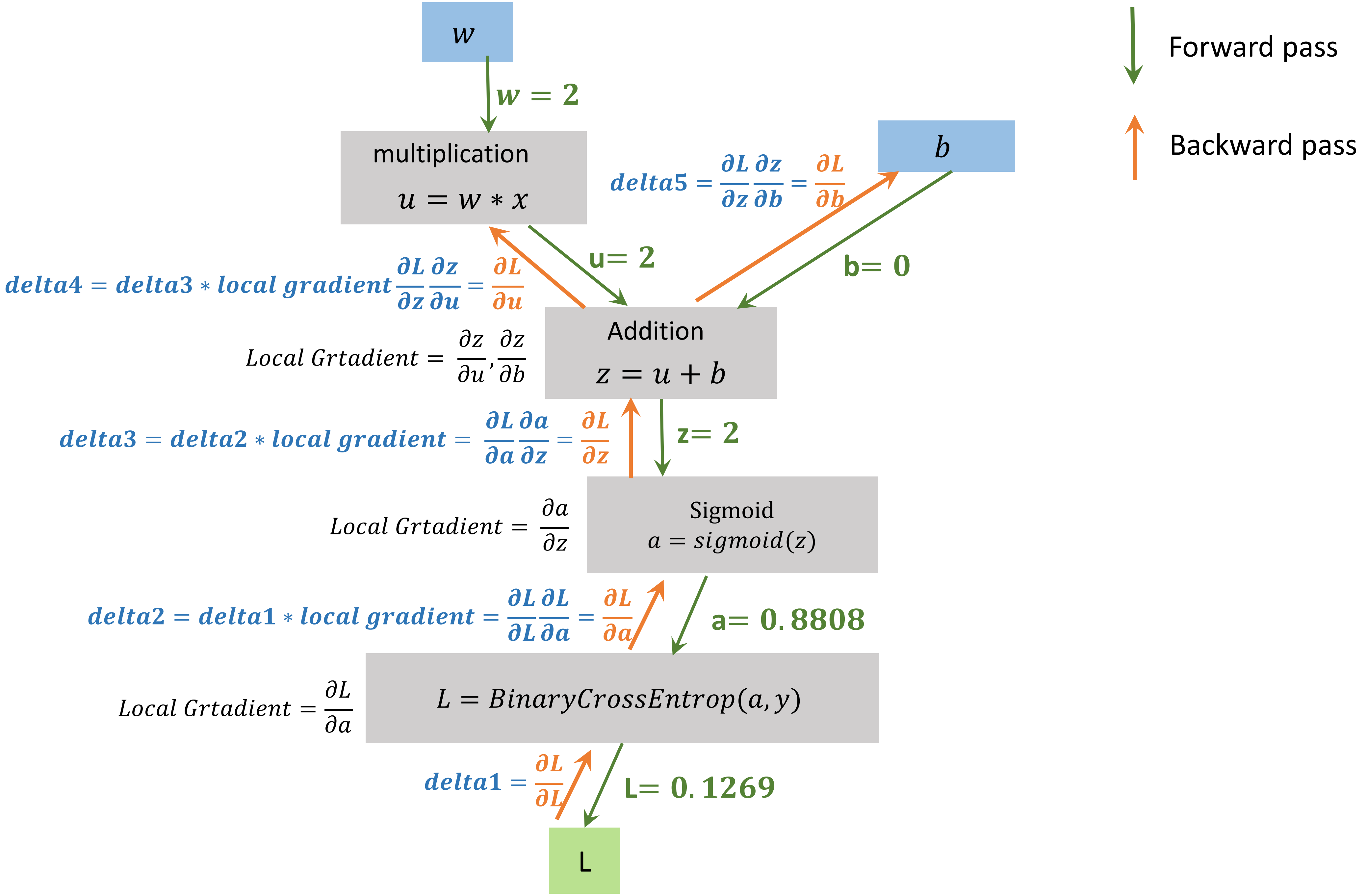
Forward pass
Backward pass
w
$w = 2$
multiplication
$u = w * x$
b
u= 2
b= 0
Addition
$z = u + b$
z= 2
Sigmoid
$a = sigmoid(z)$
a= 0.8808
$L = BinaryCrossEntrop(a, y)$
$delta1 = \frac{\partial L}{\partial L}$
L= 0.1269
L



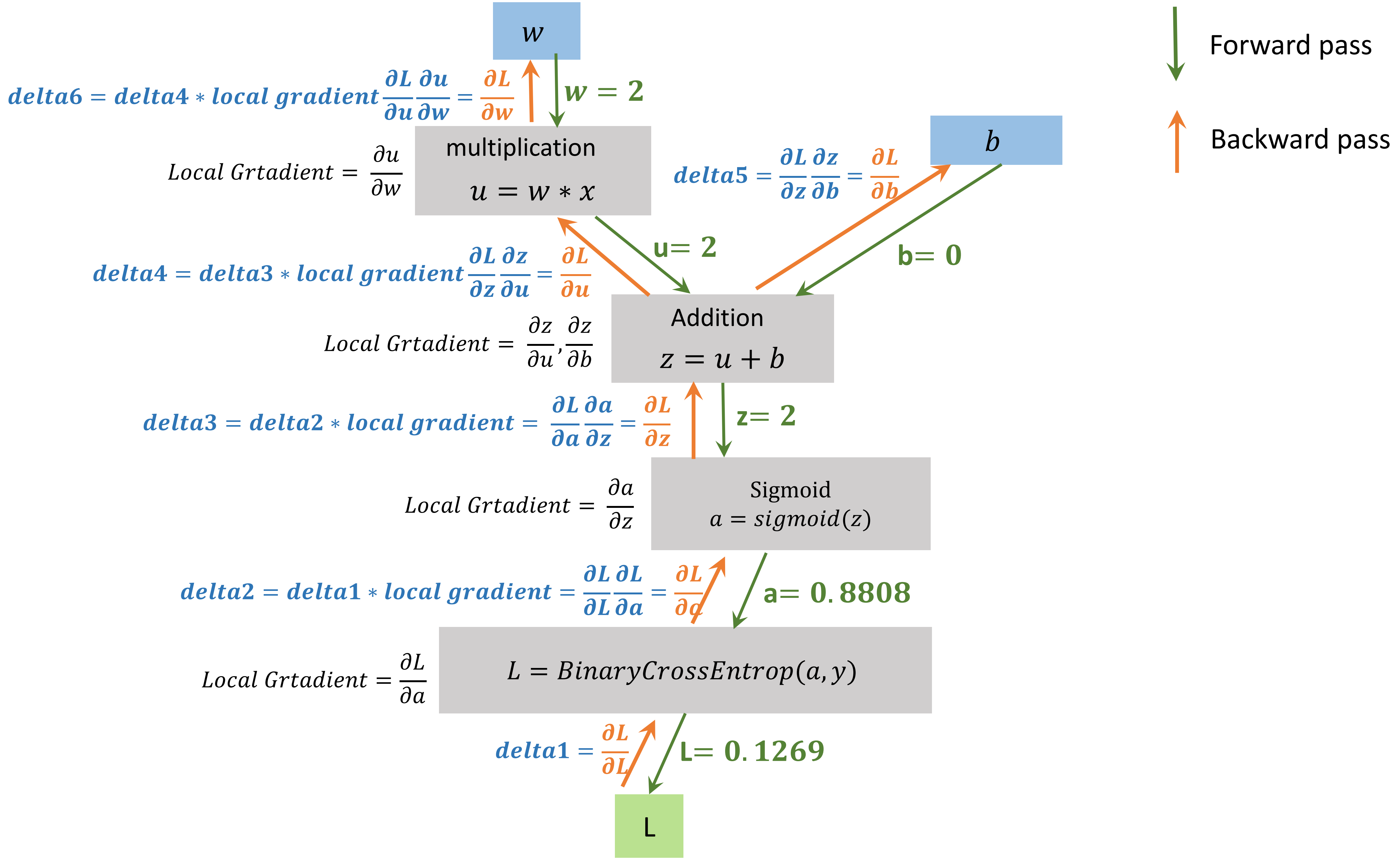
w
$w = 2$
multiplication
$u = w * x$
b
u= 2
b= 0
Addition
$z = u + b$
z= 2
Sigmoid
$a = sigmoid(z)$
$delta2 = delta1 * local\ gradient = \frac{\partial L}{\partial L}\frac{\partial L}{\partial a} = \frac{\partial L}{\partial a}$
a= 0.8808
$Local\ Grtadient = \frac{\partial L}{\partial a}$
$L = BinaryCrossEntrop(a, y), delta1$
$delta1 = \frac{\partial L}{\partial L}$
L= 0.1269
L
Forward pass
Backward pass



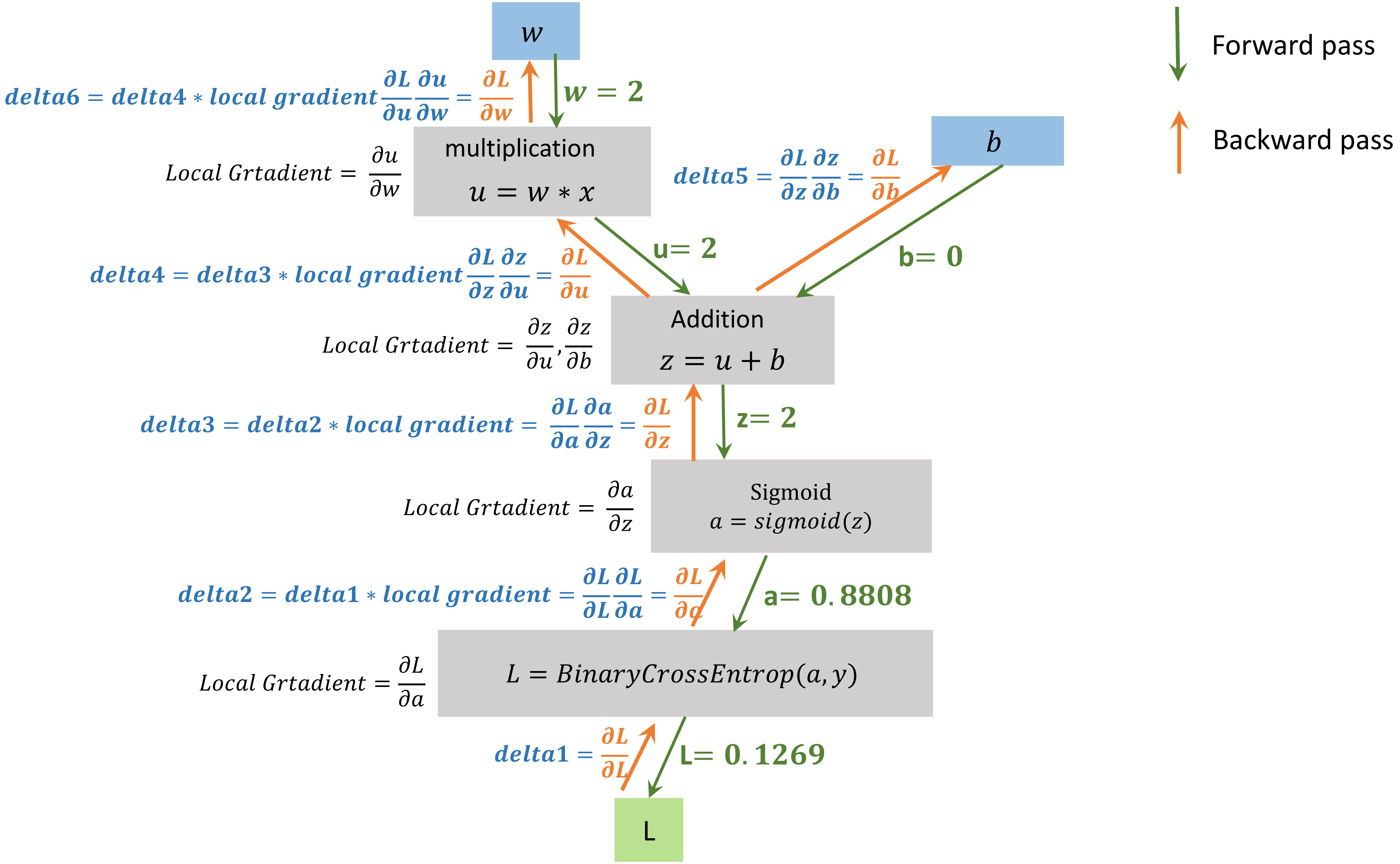
Forward pass
Backward pass
w
$w = 2$
multiplication
$u = w * x$
b
u= 2
b= 0
Addition
$z = u + b$
$delta3 = delta2 * local\ gradient = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial L}{\partial z}$
z= 2
$Local\ Grtadient = \frac{\partial a}{\partial z}$
Sigmoid
$a = sigmoid(z)$
$delta2 = delta1 * local\ gradient = \frac{\partial L}{\partial L}\frac{\partial L}{\partial a} = \frac{\partial L}{\partial a}$
a= 0.8808
$Local\ Grtadient = \frac{\partial L}{\partial a}$
$L = BinaryCrossEntrop(a, y)$
$delta1 = \frac{\partial L}{\partial L}$
L= 0.1269
L



w
$w = 2$
multiplication
$u = w * x$
b
$delta5 = \frac{\partial L}{\partial z}\frac{\partial z}{\partial b} = \frac{\partial L}{\partial b}$
$u = 2$
$b = 0$
$delta4 = delta3 * local\ gradient\ \frac{\partial L}{\partial z}\frac{\partial z}{\partial u} = \frac{\partial L}{\partial u}$
Addition
$z = u + b$
$Local\ Grtadient = \frac{\partial z}{\partial u}, \frac{\partial z}{\partial b}$
$delta3 = delta2 * local\ gradient = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial L}{\partial z}$
$z = 2$
Sigmoid
$a = sigmoid(z)$
$Local\ Grtadient = \frac{\partial a}{\partial z}$
$delta2 = delta1 * local\ gradient = \frac{\partial L}{\partial L}\frac{\partial L}{\partial a} = \frac{\partial L}{\partial a}$
$a = 0.8808$
$L = BinaryCrossEntrop(a, y)$
$Local\ Grtadient = \frac{\partial L}{\partial a}$
$delta1 = \frac{\partial L}{\partial L}$
$L = 0.1269$
L
Forward pass
Backward pass



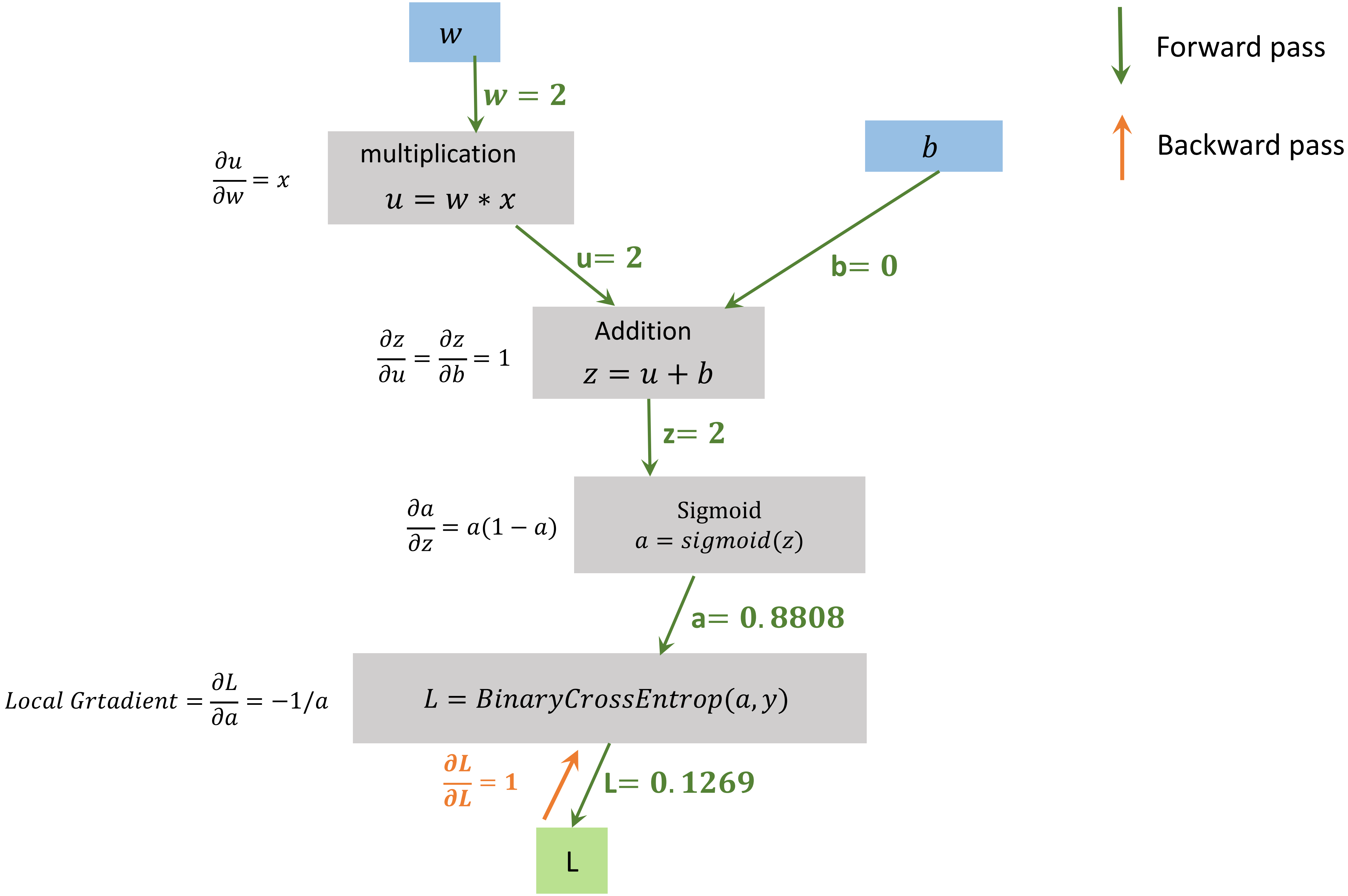
Forward pass
Backward pass
w
$w = 2$
$delta6 = delta4 * local\ gradient \frac{\partial L}{\partial u}\frac{\partial u}{\partial w} = \frac{\partial L}{\partial w}$
multiplication
$u = w * x$
$Local\ Grtadient = \frac{\partial u}{\partial w}$
b
$delta5 = \frac{\partial L}{\partial z}\frac{\partial z}{\partial b} = \frac{\partial L}{\partial b}$
u= 2
b= 0
$delta4 = delta3 * local\ gradient \frac{\partial L}{\partial z}\frac{\partial z}{\partial u} = \frac{\partial L}{\partial u}$
Addition
$z = u + b$
$Local\ Grtadient = \frac{\partial z}{\partial u}, \frac{\partial z}{\partial b}$
z= 2
$delta3 = delta2 * local\ gradient = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial L}{\partial z}$
Sigmoid
$a = sigmoid(z)$
$Local\ Grtadient = \frac{\partial a}{\partial z}$
a= 0.8808
$delta2 = delta1 * local\ gradient = \frac{\partial L}{\partial L}\frac{\partial L}{\partial a} = \frac{\partial L}{\partial a}$
$L = BinaryCrossEntrop(a, y)$
$Local\ Grtadient = \frac{\partial L}{\partial a}$
L= 0.1269
$delta1 = \frac{\partial L}{\partial L}$
L



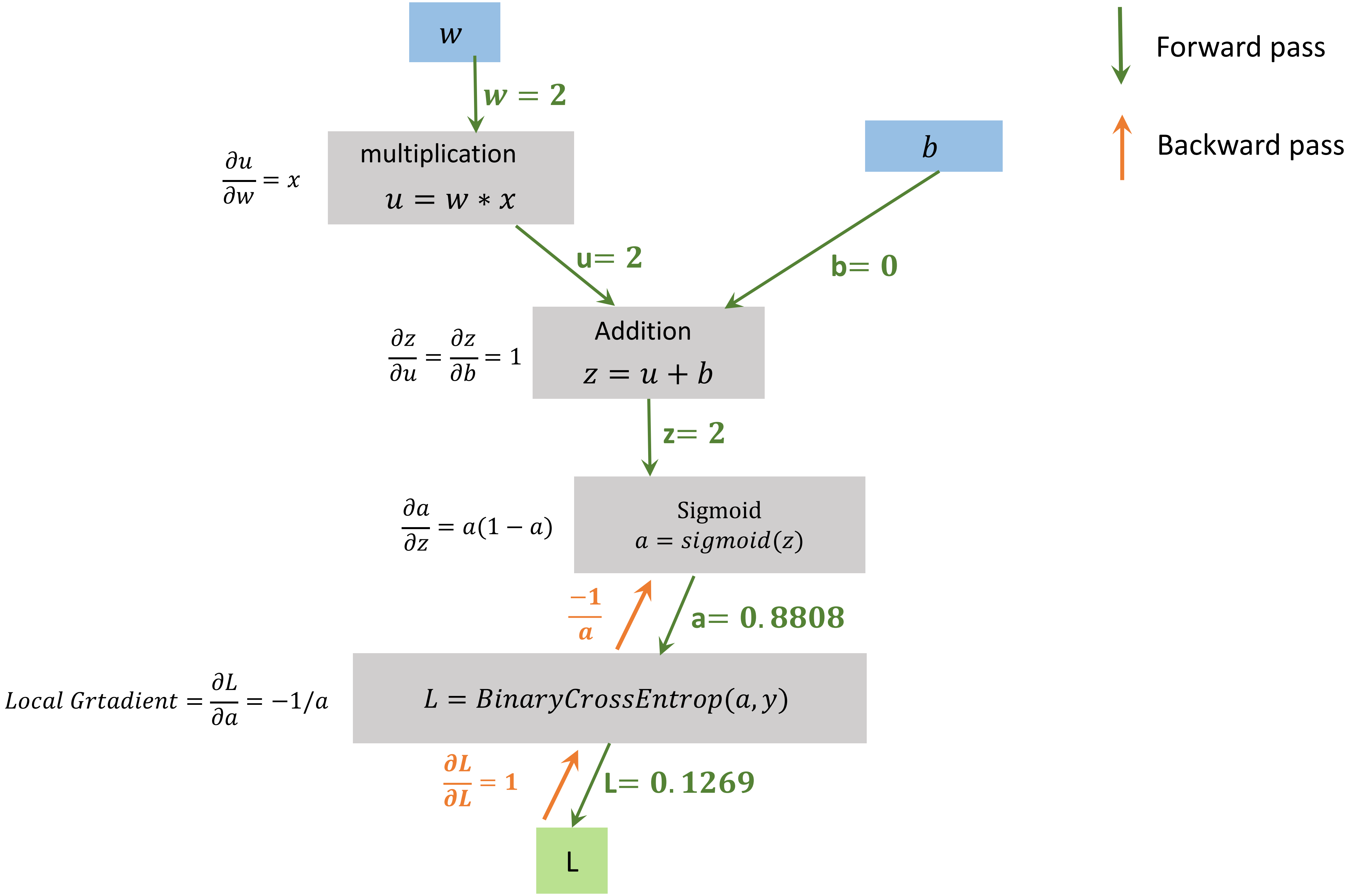
Forward pass
Backward pass
w
$delta6 = delta4 * local\ gradient \frac{\partial L}{\partial u}\frac{\partial u}{\partial w} = \frac{\partial L}{\partial w}$
$w = 2$
$Local\ Grtadient = \frac{\partial u}{\partial w}$
multiplication
$u = w * x$
b
$delta5 = \frac{\partial L}{\partial z}\frac{\partial z}{\partial b} = \frac{\partial L}{\partial b}$
u= 2
b= 0
$delta4 = delta3 * local\ gradient \frac{\partial L}{\partial z}\frac{\partial z}{\partial u} = \frac{\partial L}{\partial u}$
$Local\ Grtadient = \frac{\partial z}{\partial u}, \frac{\partial z}{\partial b}$
Addition
$z = u + b$
$delta3 = delta2 * local\ gradient = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial L}{\partial z}$
z= 2
$Local\ Grtadient = \frac{\partial a}{\partial z}$
Sigmoid
$a = sigmoid(z)$
$delta2 = delta1 * local\ gradient = \frac{\partial L}{\partial L}\frac{\partial L}{\partial a} = \frac{\partial L}{\partial a}$
a= 0.8808
$Local\ Grtadient = \frac{\partial L}{\partial a}$
$L = BinaryCrossEntrop(a, y)$
$delta1 = \frac{\partial L}{\partial L}$
L= 0.1269
L

Local Gradients

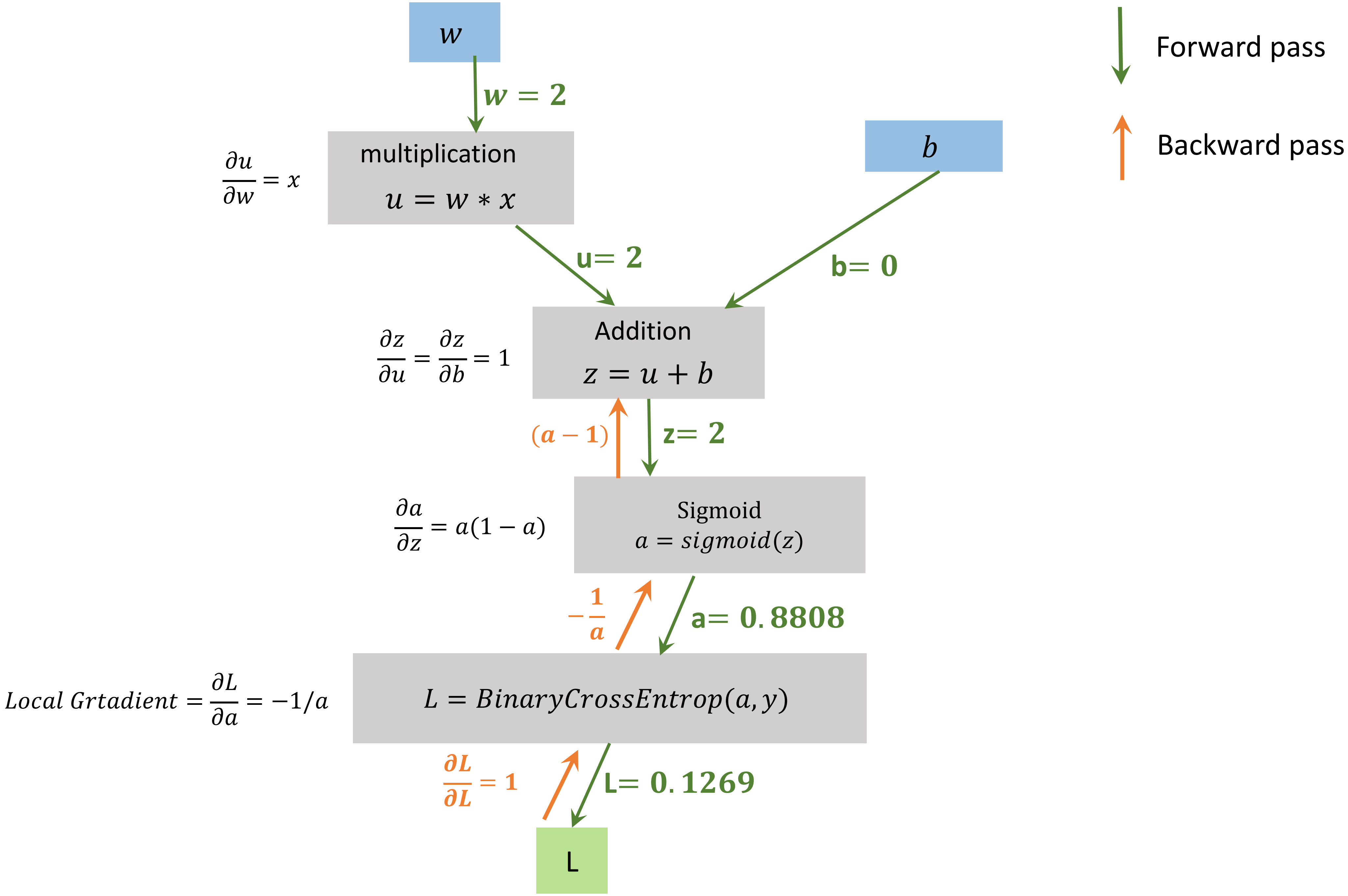
Operation	Local gradient	Need to save from forward pass
Addition $f(x, y) = x + y$	$\frac{\partial f}{\partial x} = 1, \frac{\partial f}{\partial y} = 1$	1
Multiplication $f(w) = w * x$	$\frac{\partial f}{\partial w} = x$	$other = x$ $(self = w, other = x)$
Sigmoid $f(z) = sigmoid(z)$	$\frac{\partial f}{\partial z} = sigmoid(z) * (1 - sigmoid(z))$	$result = sigmoid(z) = a$
Binary Cross entropy $L = -(y * \log(a) + (1 - y) * \log(1 - a))$	$\frac{\partial L}{\partial a} = -\left(\frac{y}{a} + \frac{1 - y}{a - 1}\right)$	a, y



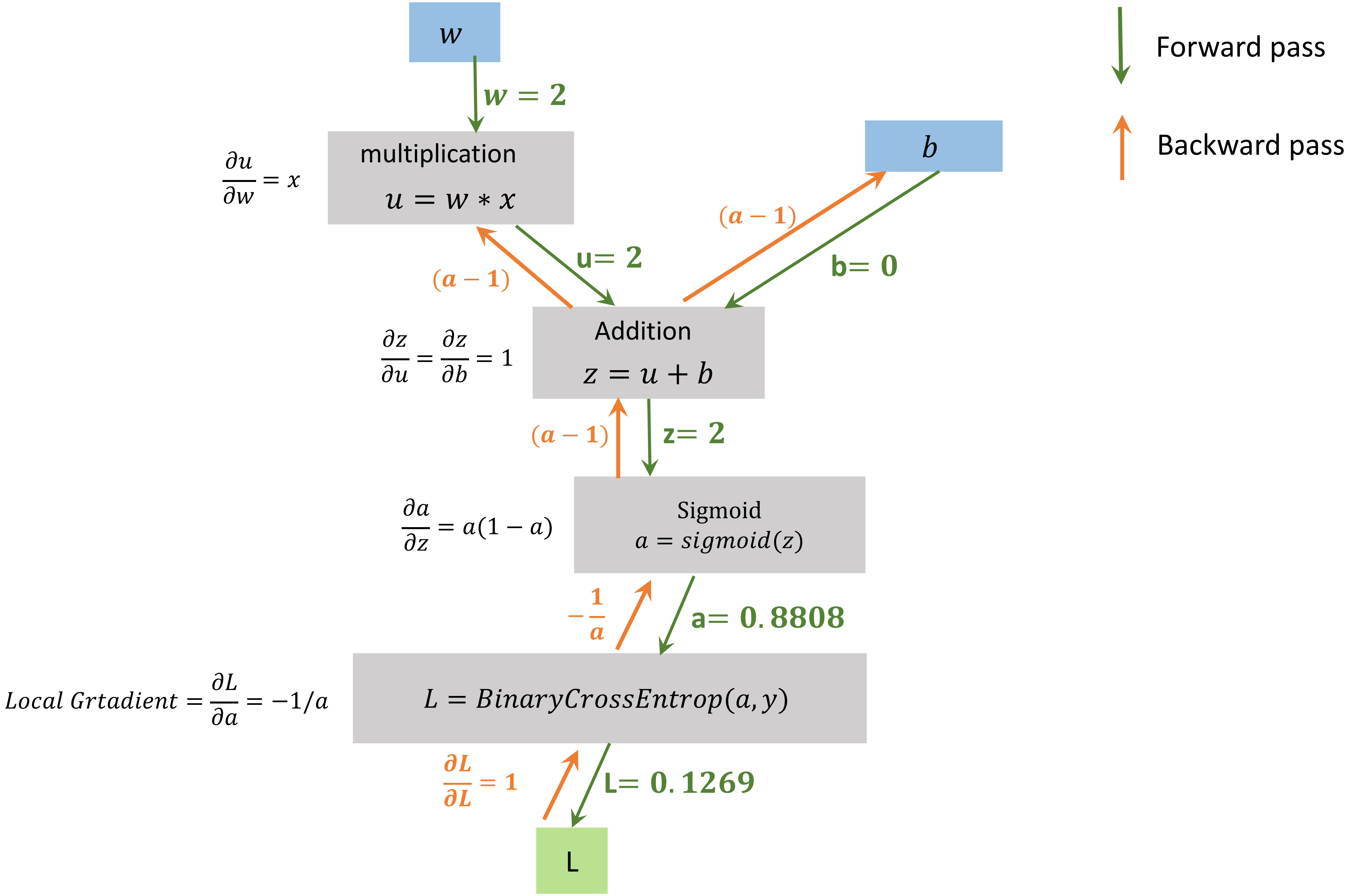
w
$w = 2$
multiplication
$u = w * x$
$\frac{\partial u}{\partial w} = x$
b
u= 2
b= 0
Addition
$z = u + b$
$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial b} = 1$
z= 2
Sigmoid
$a = sigmoid(z)$
$\frac{\partial a}{\partial z} = a(1 - a)$
a= 0.8808
$L = BinaryCrossEntrop(a, y)$
$Local\ Grtadient = \frac{\partial L}{\partial a} = -1/a$
$\frac{\partial L}{\partial L} = 1$
L= 0.1269
L
Forward pass
Backward pass



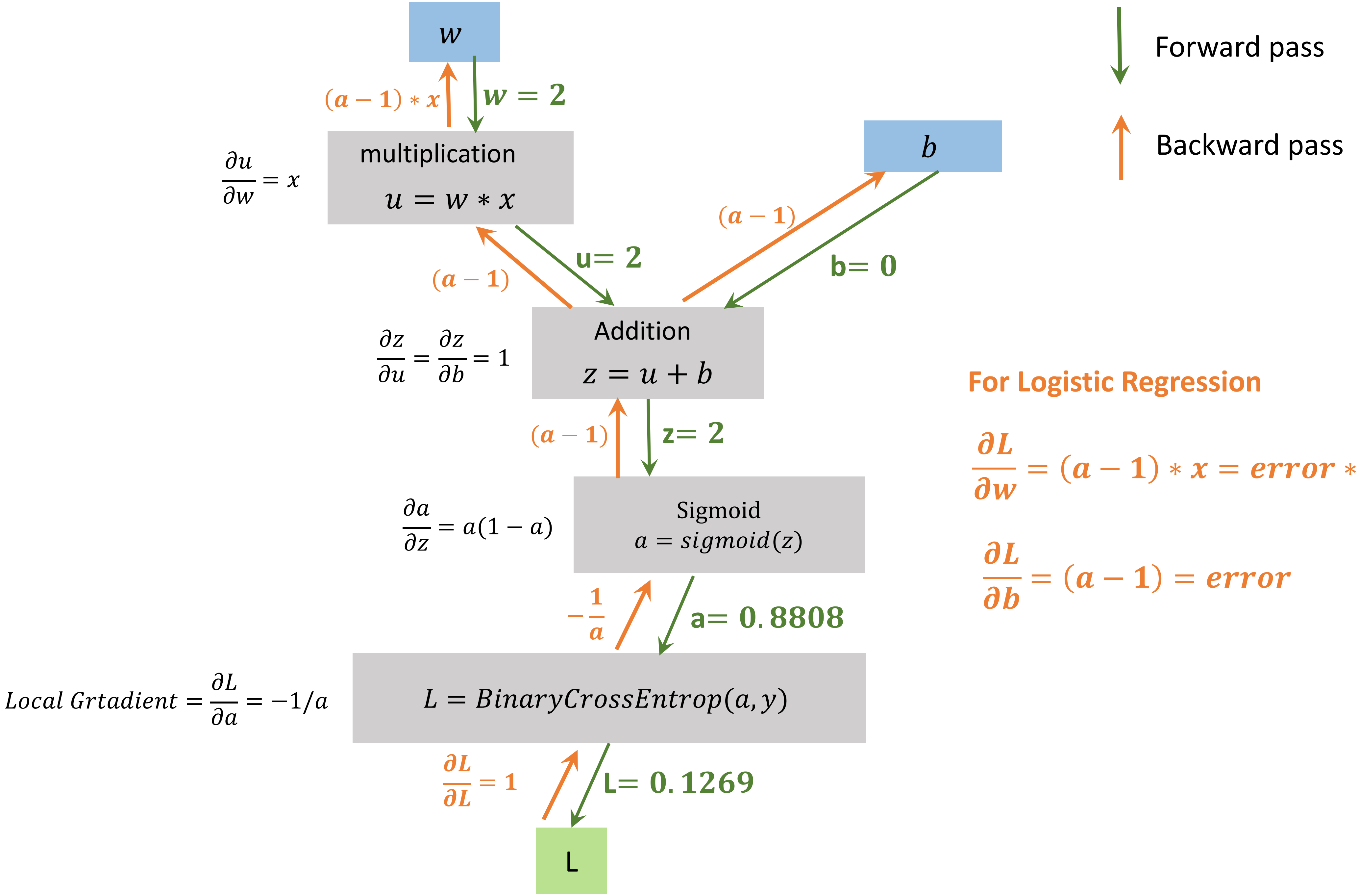
w
$w = 2$
multiplication
$u = w * x$
$\frac{\partial u}{\partial w} = x$
b
u= 2
b= 0
Addition
$z = u + b$
$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial b} = 1$
z= 2
Sigmoid
$a = sigmoid(z)$
$\frac{\partial a}{\partial z} = a(1 - a)$
$\frac{-1}{a}$
a= 0.8808
$L = BinaryCrossEntrop(a, y)$
$Local\ Grtadient = \frac{\partial L}{\partial a} = -1/a$
$\frac{\partial L}{\partial L} = 1$
L= 0.1269
L
Forward pass
Backward pass



w
$w = 2$
multiplication
$u = w * x$
$\frac{\partial u}{\partial w} = x$
b
u= 2
b= 0
Addition
$z = u + b$
$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial b} = 1$
$(a - 1)$
z= 2
Sigmoid
$a = sigmoid(z)$
$\frac{\partial a}{\partial z} = a(1 - a)$
$-\frac{1}{a}$
a= 0.8808
$L = BinaryCrossEntrop(a, y)$
$Local\ Grtadient = \frac{\partial L}{\partial a} = -1/a$
$\frac{\partial L}{\partial L} = 1$
L= 0.1269
L
Forward pass
Backward pass



w
$w = 2$
multiplication
$u = w * x$
$\frac{\partial u}{\partial w} = x$
b
$(a-1)$
$u= 2$
$(a-1)$
$b= 0$
Addition
$z = u + b$
$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial b} = 1$
$(a-1)$
$z= 2$
Sigmoid
$a = sigmoid(z)$
$\frac{\partial a}{\partial z} = a(1-a)$
$-\frac{1}{a}$
$a= 0.8808$
$L = BinaryCrossEntrop(a, y)$
$Local\ Grtadient = \frac{\partial L}{\partial a} = -1/a$
$\frac{\partial L}{\partial L} = 1$
$L= 0.1269$
L
Forward pass
Backward pass



For Logistic Regression

$$\frac{\partial L}{\partial w} = (a-1) * x = error * input$$

$$\frac{\partial L}{\partial b} = (a-1) = error$$

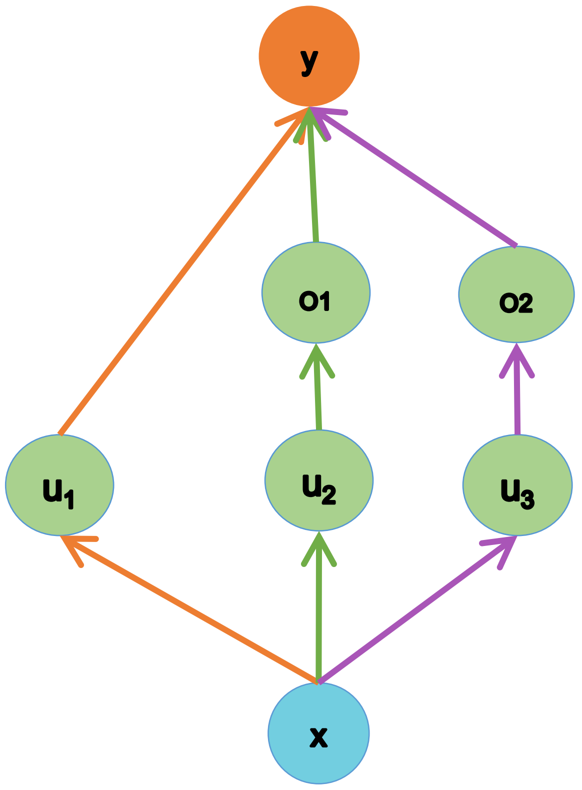
Computation graph – Backward pass

$$Path1 = \frac{\partial y}{\partial u_1} * \frac{\partial u_1}{\partial x}$$

$$Path2 = \frac{\partial y}{\partial o_1} * \frac{\partial o_1}{\partial u_2} * \frac{\partial u_2}{\partial x}$$

$$Path3 = \frac{\partial y}{\partial o_2} * \frac{\partial o_2}{\partial u_3} * \frac{\partial u_3}{\partial x}$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_1} * \frac{\partial u_1}{\partial x} + \frac{\partial y}{\partial o_1} * \frac{\partial o_1}{\partial u_2} * \frac{\partial u_2}{\partial x} + \frac{\partial y}{\partial o_2} * \frac{\partial o_2}{\partial u_3} * \frac{\partial u_3}{\partial x}$$



Backpropagation is just the repeated application of the **chain rule**