Deep Learning for Computer Vision

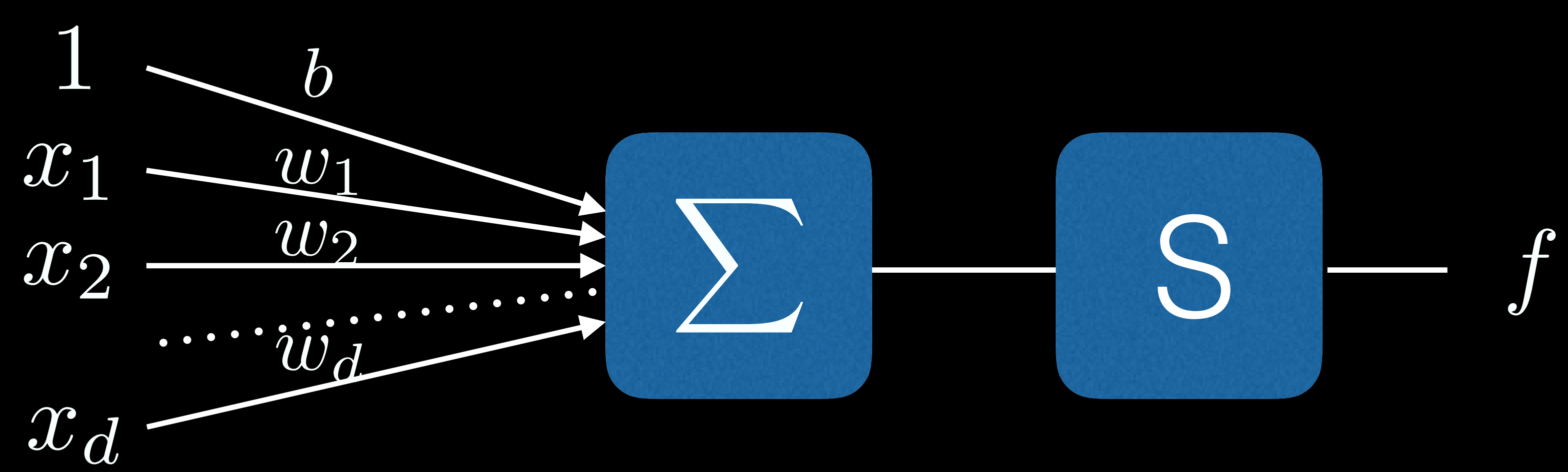
Lecture 6: The Perceptron, the XOR Challenge, Going Deep, Love for Feed Forward Networks, Jacobians, and Tensors

Peter Belhumeur

Computer Science
Columbia University

The Perceptron

[Rosenblatt 57]



$$P(y = 1|\mathbf{x}, \mathbf{w}, b)$$

$$f(\mathbf{x}; \mathbf{w}) = S(\mathbf{w}^T\mathbf{x} + b) = \frac{1}{1 + e^{-w_1x_1 - \ldots - w_dx_d + b}}$$

$$S(z) = \frac{1}{1 + e^{-z}}$$

Feedforward Networks

- Let $y = f^*(\mathbf{x})$ be some function we are trying to approximate
- This function could be assignment of an input to a category as in a classifier
- Let a feedforward network approximate this mapping $y = f(\mathbf{x}; \theta)$ by learning parameters θ

Feedforward Networks

- Feedforward networks have **NO** feedback
- These networks can be represented as directed acyclic graphs describing the composition of functions
- These networks are composed of functions represented as **"layers"** $f(\mathbf{x}) = l^3(l^2(l^1(\mathbf{x})))$
- The length of the chain of compositions gives the **"depth"** of the network

Feedforward Networks

- The functions defining the layers have been influenced by neuroscience

- Our training dictates the values to be produced output layer and the weights are chosen accordingly

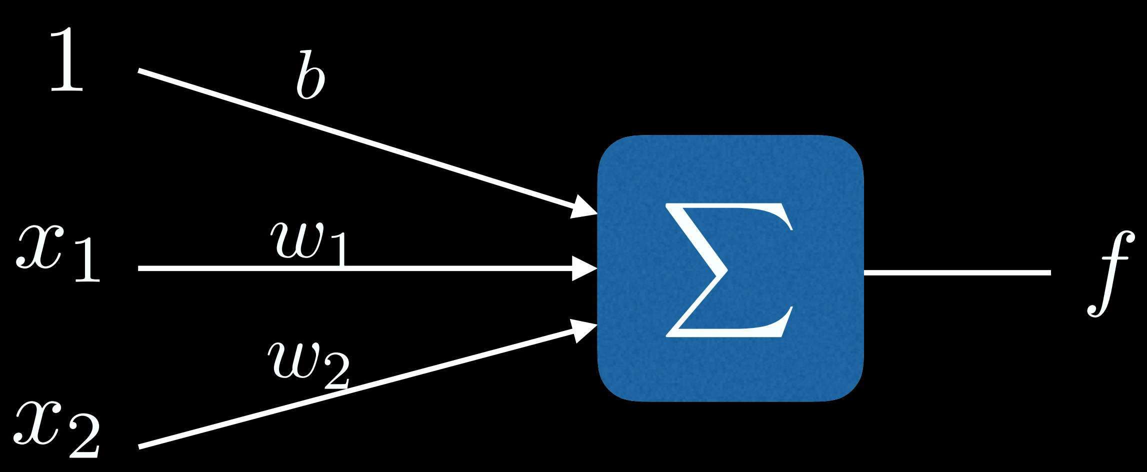
- The weights for intermediate or **“hidden”** layers are learned and not specified directly

- You can think of the network as mapping the raw input space $\mathbf{x}$ to some transformed feature space $\phi(\mathbf{x})$ where the samples are ideally linearly separable

Feedforward Networks

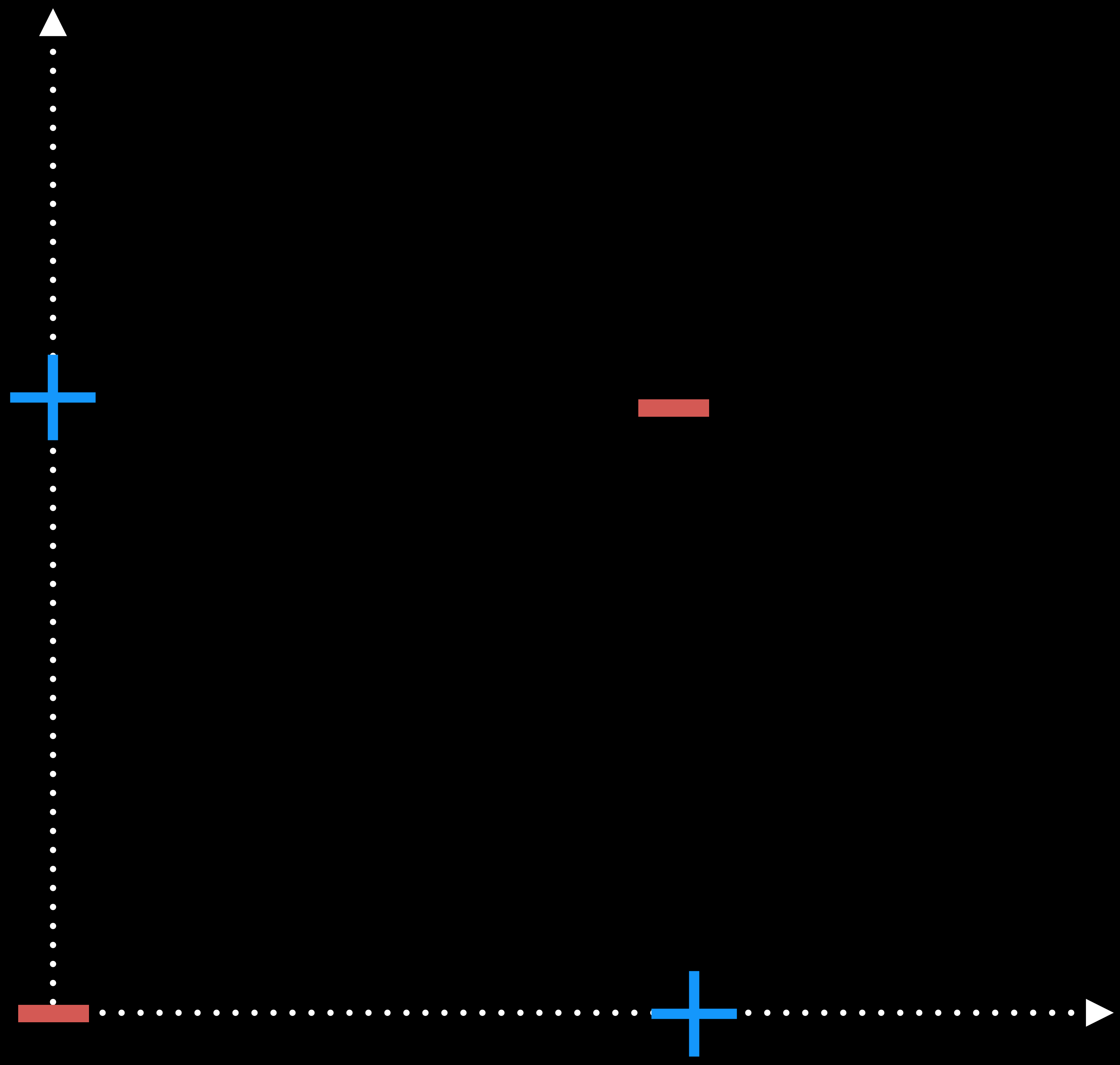
- The goal of the feedforward network is then to learn this mapping $y = f(\mathbf{x}; \theta, \mathbf{w}) = \phi(\mathbf{x}; \theta)^T \mathbf{w}$
- If this mapping is simply the identity then our network is linear and this certainly not going to work for most computer vision problems.
- Consider approximating something as simple as the XOR function with a linear network…

2D Linear Network

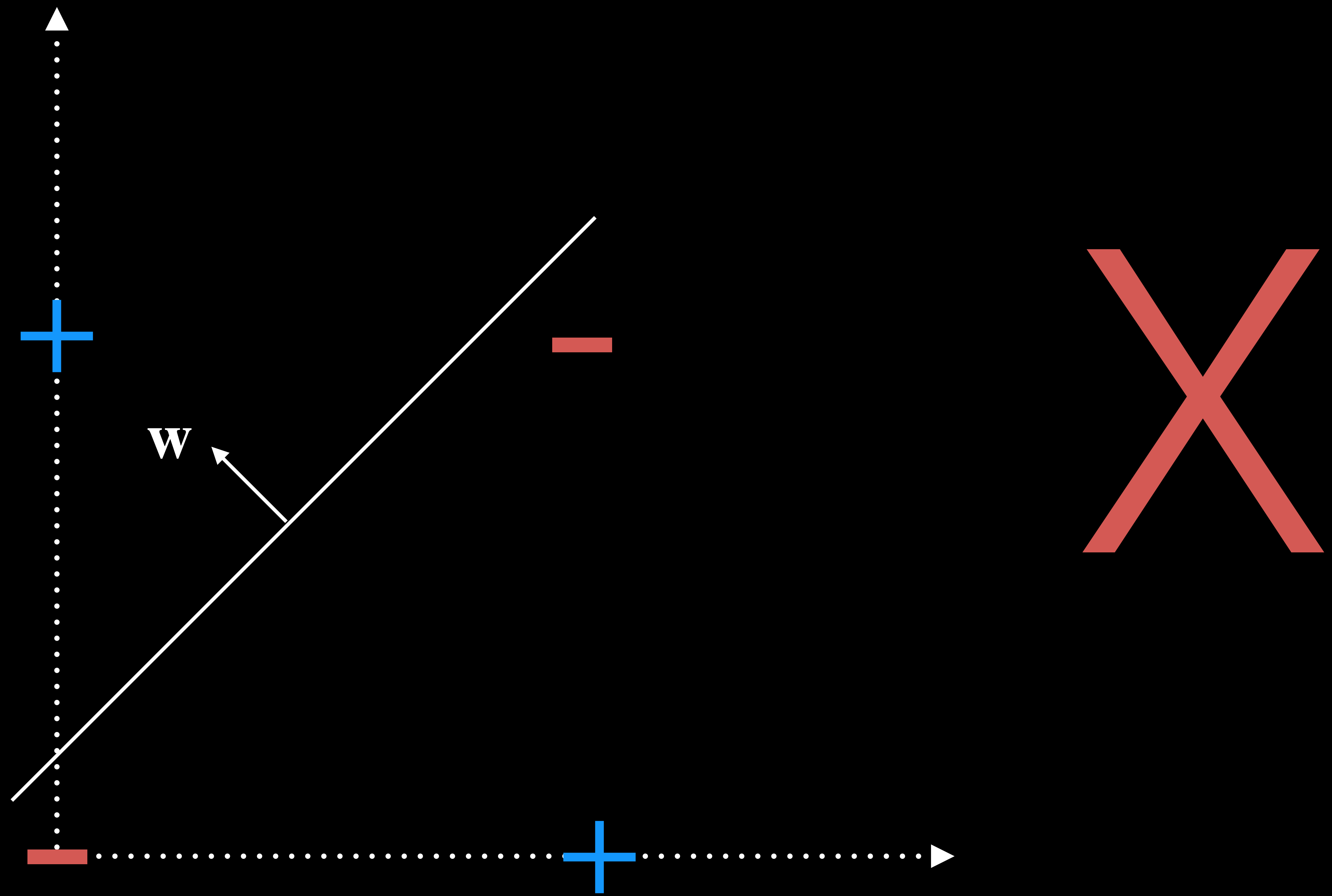


$$f(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^T \mathbf{x} + b$$

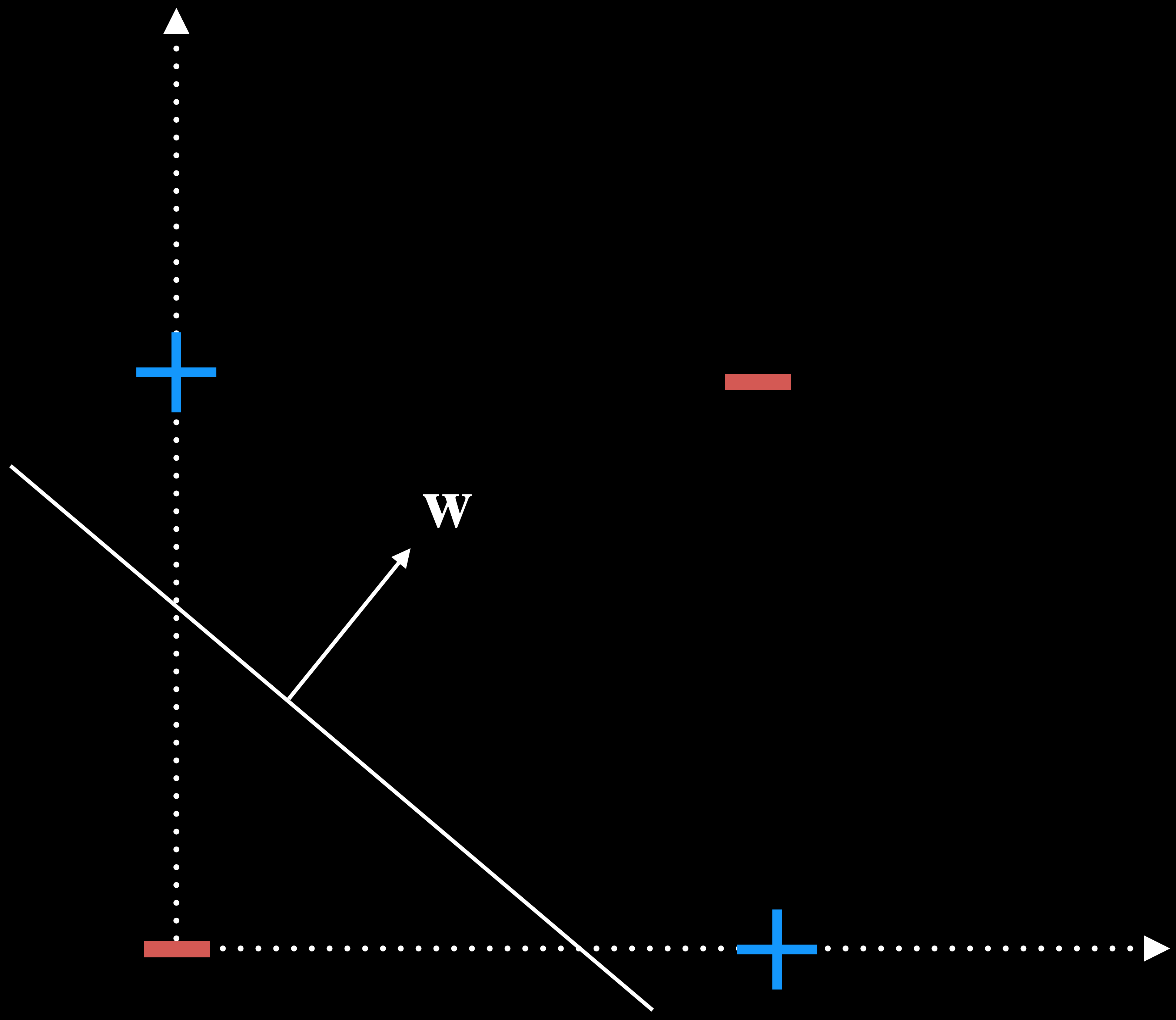
XOR



XOR



XOR



Let's show this by trying fit a linear network to this data and choose the loss function to be mean squared error (MSE).

XOR with Linear Network

$$L(\mathbf{w}, b) = \frac{1}{4} \sum_{\mathbf{x} \in \mathbb{X}} (f^*(\mathbf{x}) - f(\mathbf{x}; \mathbf{w}, b))^2$$

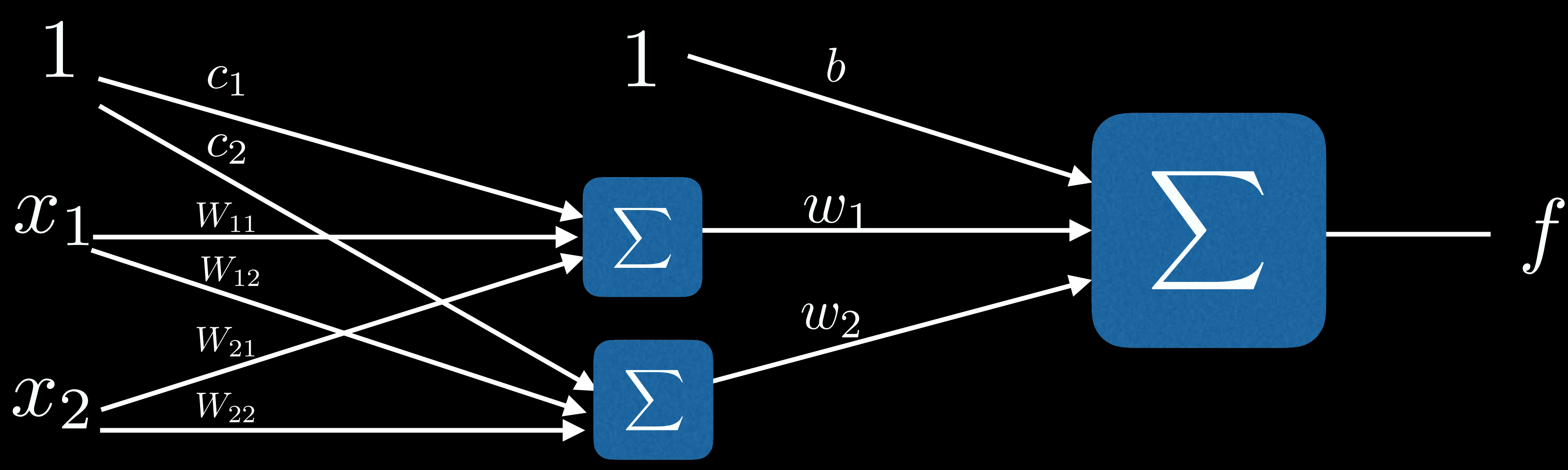
$$f(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^T \mathbf{x} + b$$

$$\hat{\mathbf{w}}, \hat{b} = \underset{\mathbf{w}, b}{\operatorname{argmax}}\ L(\mathbf{w}, b) = \mathbf{0}, \frac{1}{2}$$

There is no way to approximate the XOR function with this single layer linear network!

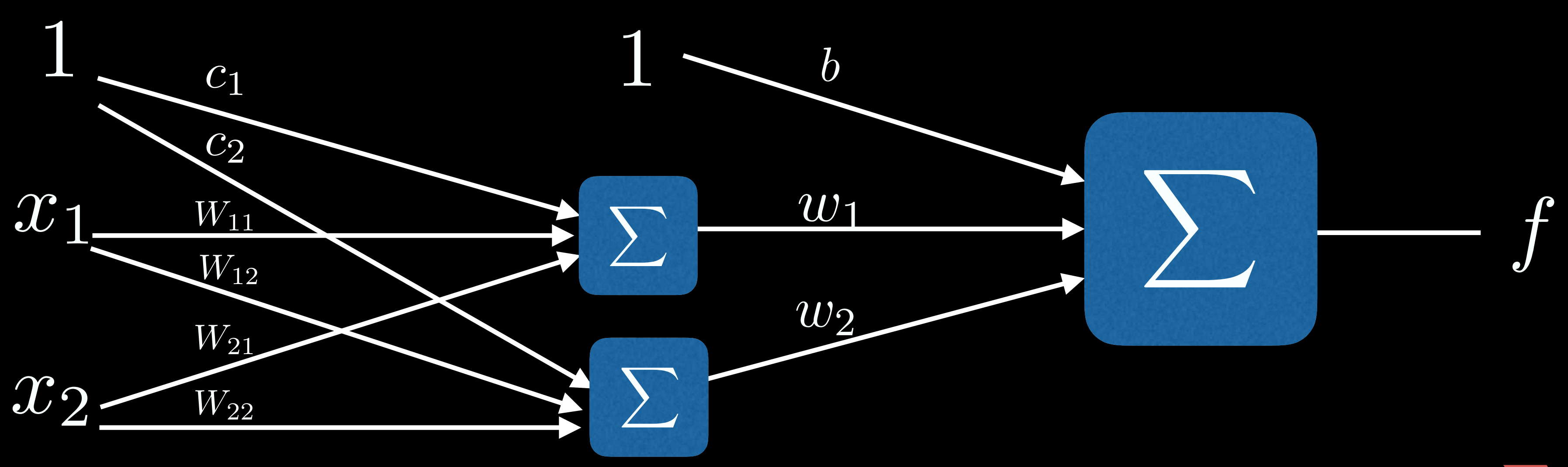
What if we introduced a hidden layer and chose this layer to be linear?

Linear Network with Hidden Layer



$$
\begin{aligned}
f(\mathbf{x}; W, \mathbf{c}, \mathbf{w}, b) &= \mathbf{w}^T (W\mathbf{x} + \mathbf{c}) + b \\
&= \tilde{\mathbf{w}}^T \mathbf{x} + \tilde{b}
\end{aligned}
$$

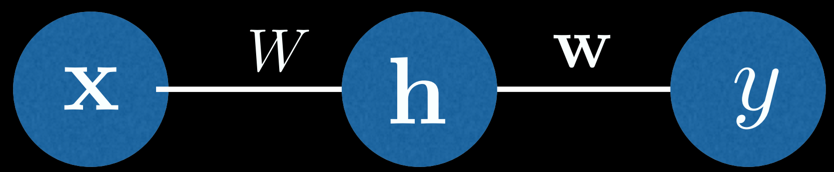
Linear Network with Hidden Layer



$$f(\mathbf{x}; W, \mathbf{c}, \mathbf{w}, b) = \mathbf{w}^T(W^T\mathbf{x} + \mathbf{c}) + b$$
$$= \tilde{\mathbf{w}}^T\mathbf{x} + \tilde{b}$$

Network notation is cumbersome. Let’s simplify.

"Deep" Network with Hidden Layer

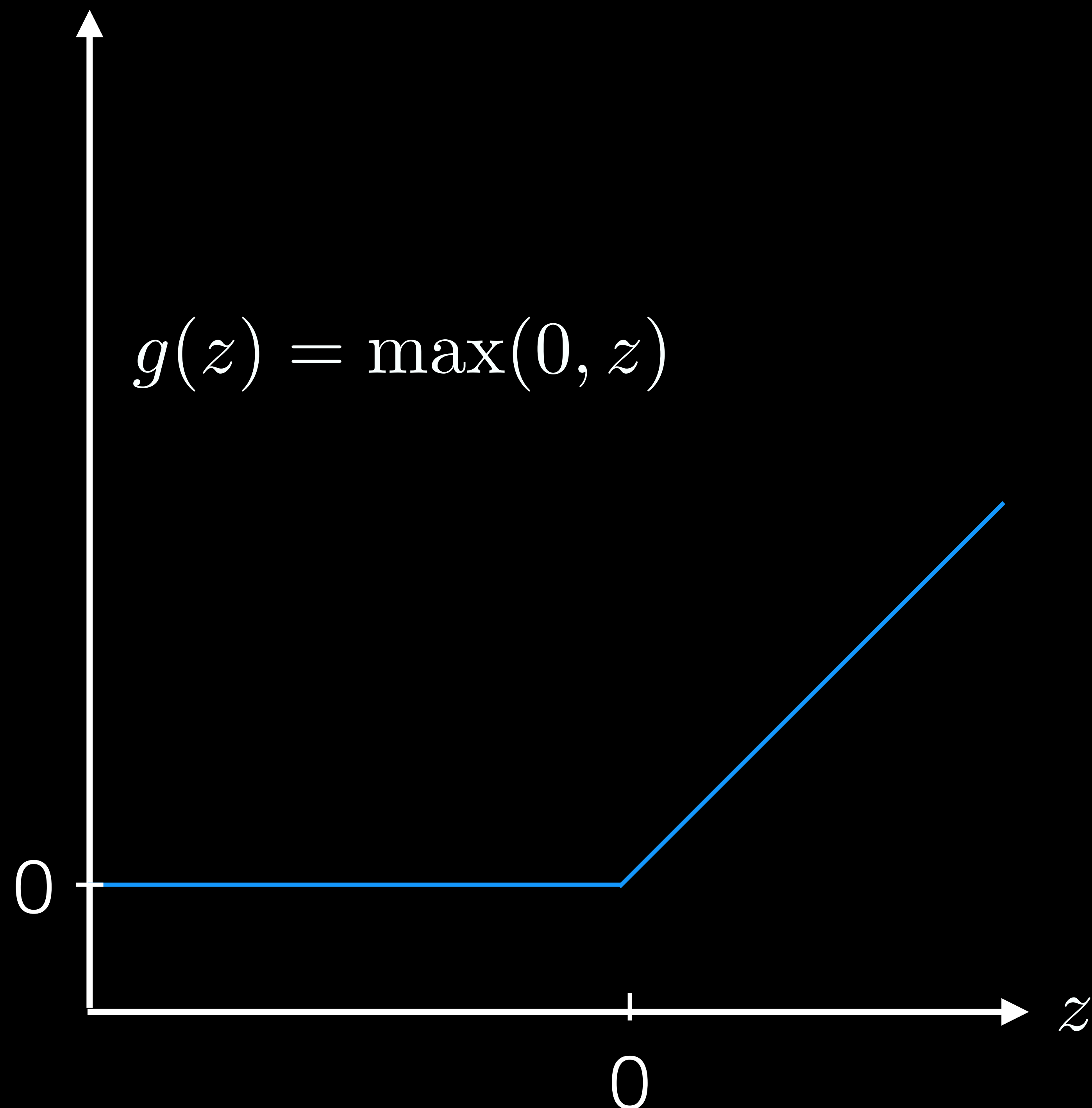


$$\mathbf{h} = g(W^T\mathbf{x} + \mathbf{c})$$

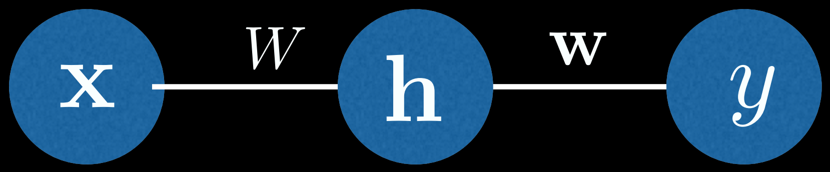
$$\begin{aligned} y &= \mathbf{w}^T\mathbf{h} + b \\ &= \mathbf{w}^T g(W^T\mathbf{x} + \mathbf{c}) + b \end{aligned}$$

Let's choose $\mathbf{g}$ to be a non-linear function as linear did not work for us before!

Rectified Linear Unit ReLU



Hidden Layer with ReLU



$$\mathbf{h} = \max(\mathbf{0}, W^T\mathbf{x} + \mathbf{c})$$

$$\begin{aligned} y &= \mathbf{w}^T\mathbf{h} + b \\ &= \mathbf{w}^T \max(\mathbf{0}, W^T\mathbf{x} + \mathbf{c}) + b \end{aligned}$$

XOR and Simple Non-Linear Network

Let $W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ $\mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$ $\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$ $b = 0$

Our XOR samples are $X = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$

XOR and Simple Non-Linear Network

Then $W^T X + c = \begin{bmatrix} 0 & 1 & 1 & 2 \\ -1 & 0 & 0 & 1 \end{bmatrix}$

$$\mathbf{h} = \max(\mathbf{0}, W^T X + c) = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$y = \mathbf{w}^T \max(\mathbf{0}, W^T \mathbf{x} + \mathbf{c}) + b = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}$$

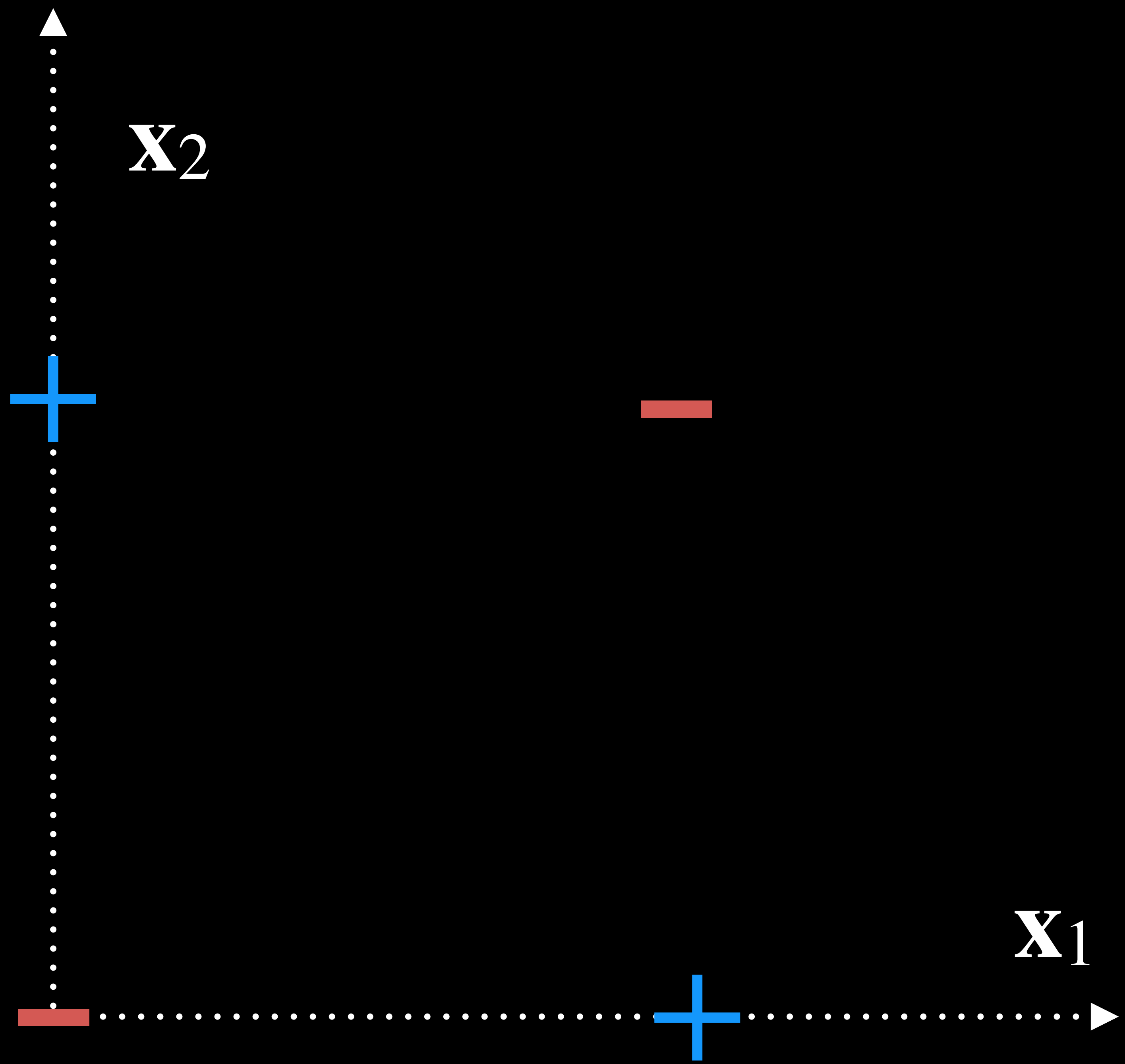
XOR and Simple Non-Linear Network

Then $$W^T X + c = \begin{bmatrix} 0 & 1 & 1 & 2 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

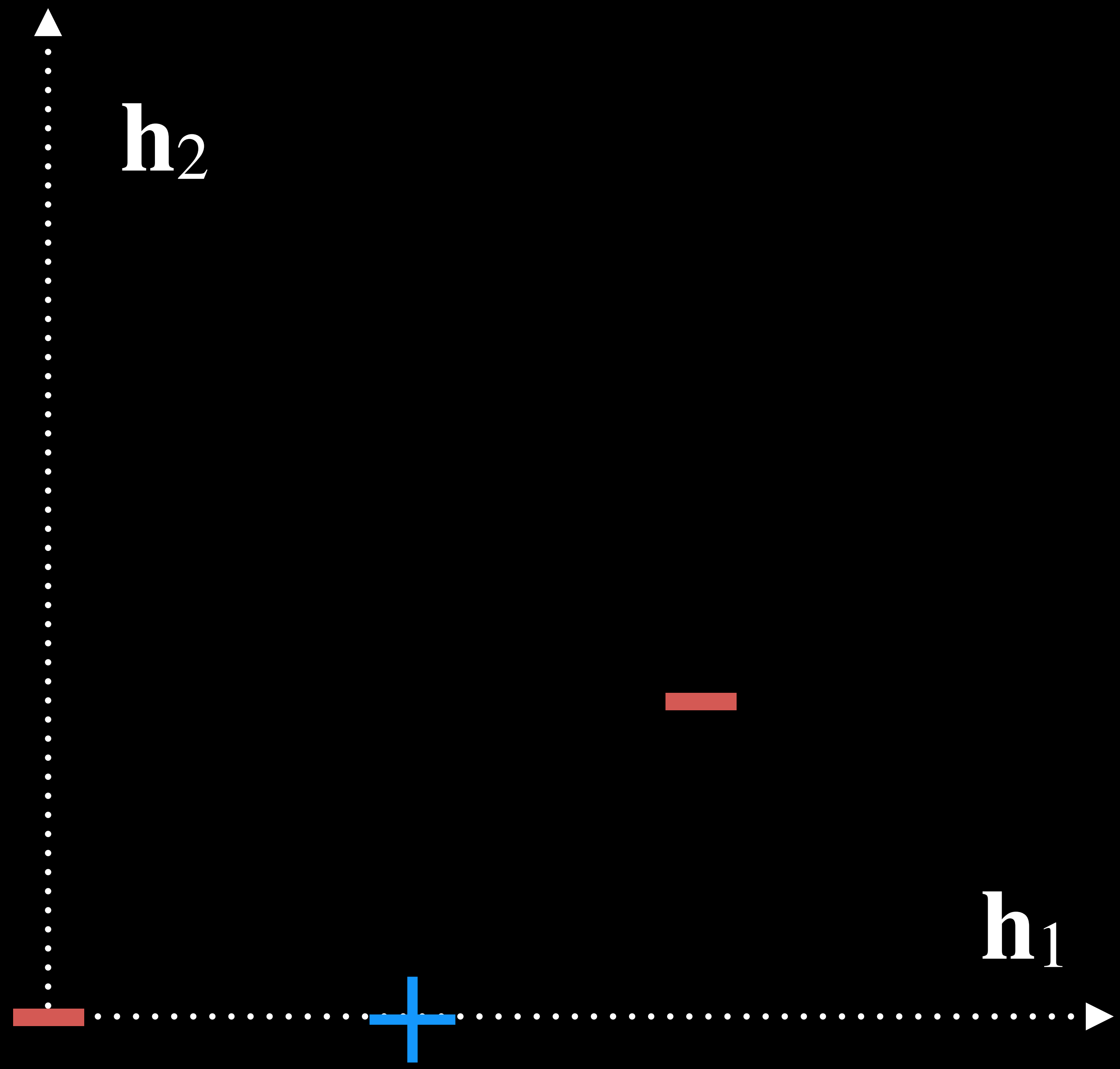
$$\mathbf{h} = \max(\mathbf{0}, W^T X + c) = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$y = \mathbf{w}^T \max(\mathbf{0}, W^T \mathbf{x} + \mathbf{c}) + b = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}$$

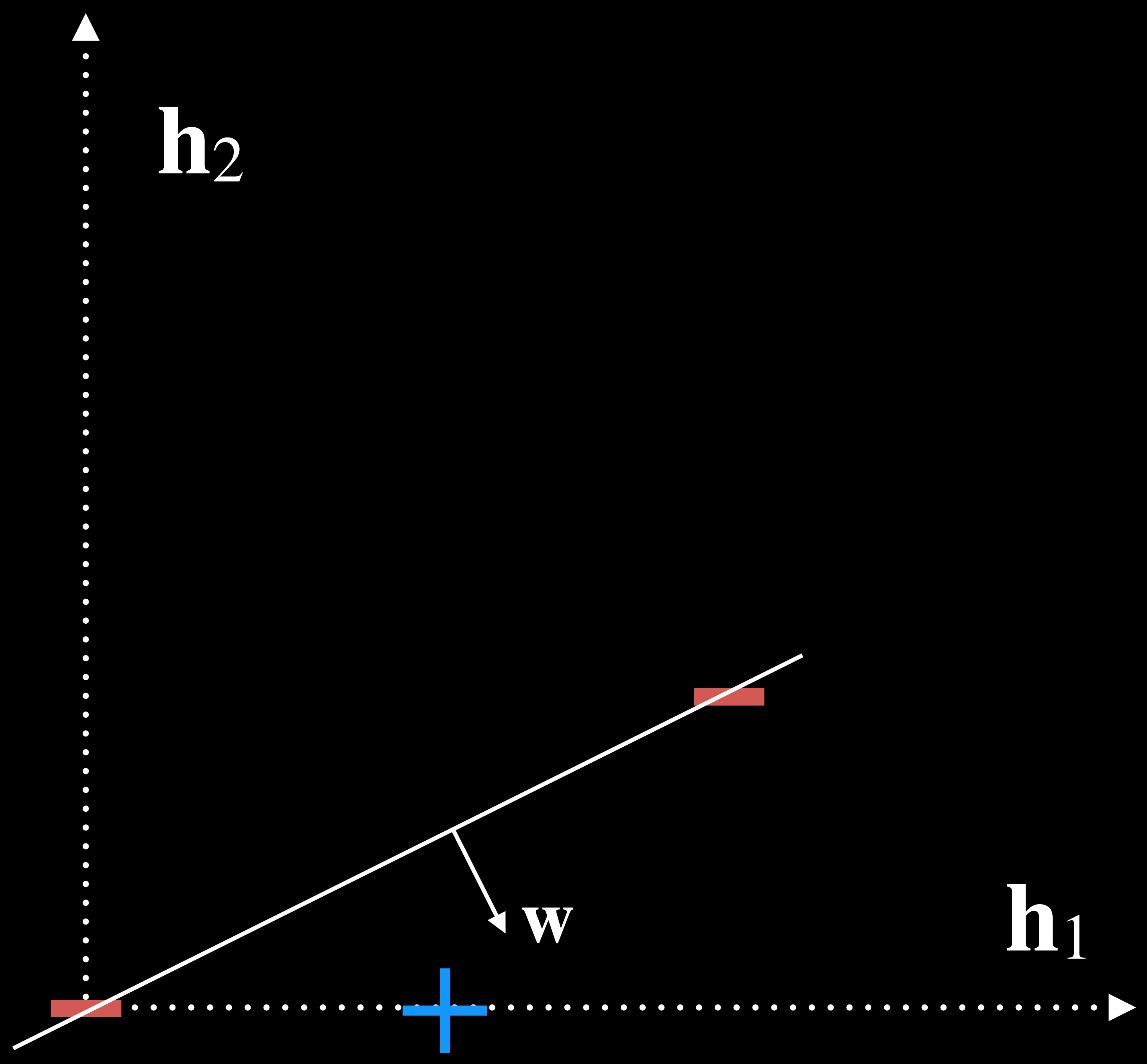
Input Feature Space



Transformed Feature Space: $\mathbf{h} = \phi(\mathbf{x})$



Transformed Feature Space: $\mathbf{h} = \phi(\mathbf{x})$



Non-linear mappings are f#$king awesome!

For this problem, we had a binary classifier with desired outputs of 0 and 1.

Ok. Let’s create some noisy XOR data and see if we can train this network using backprop.

First we will need a cost (loss) function to optimize.

We do this by assigning a cost/loss to the network output relative to the desired output.

Common Loss Function for Binary Classifier

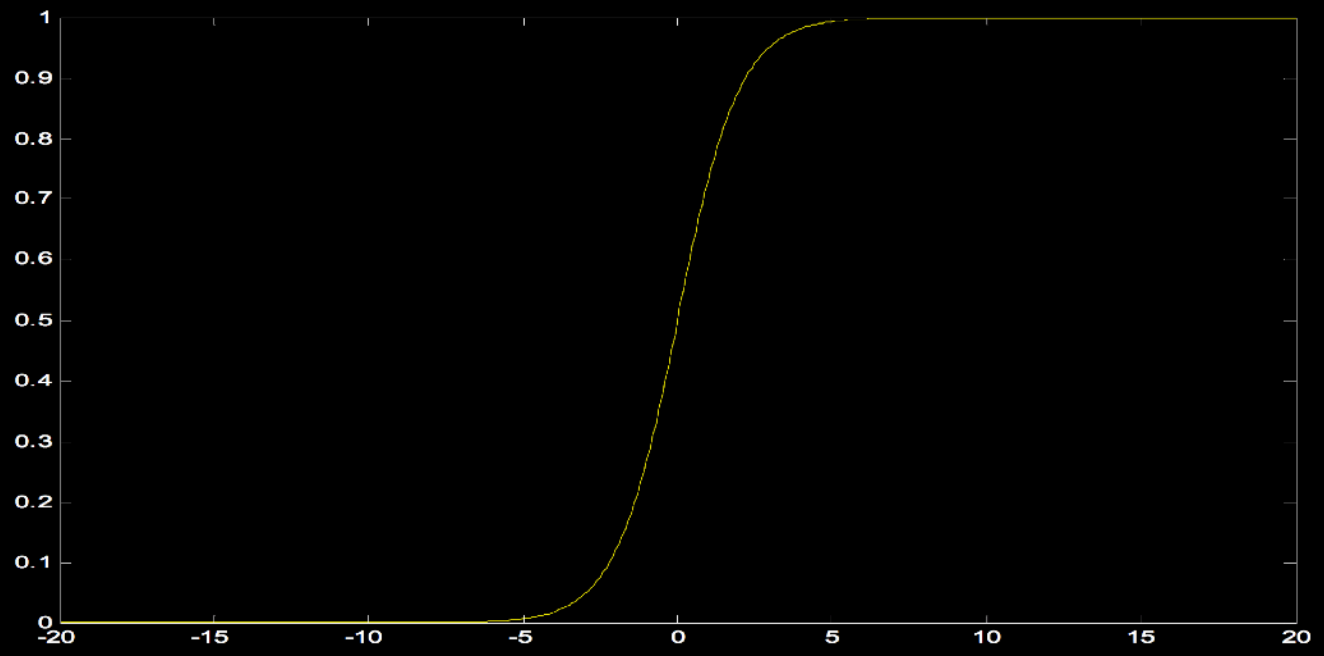
$$P(y) = \sigma((2y-1)z)$$

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

$$\begin{aligned} L(\theta) &= -\log P(y|\mathbf{x}) \\ &= -\log \sigma((2y-1)z) \\ &= \zeta((1-2y)z) \end{aligned}$$

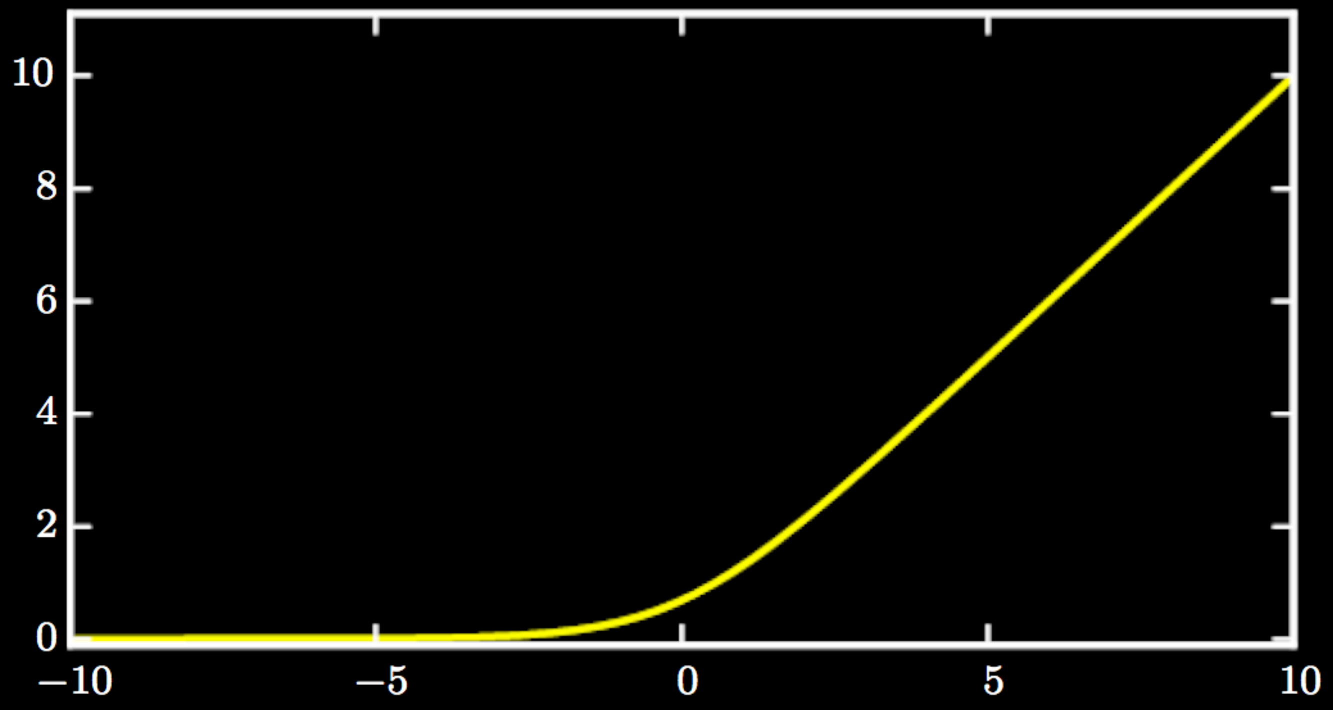
$$\zeta(r) = \log(1+e^{r})$$ Softplus

Sigmoid and Friend…Softplus



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

Sigmoid



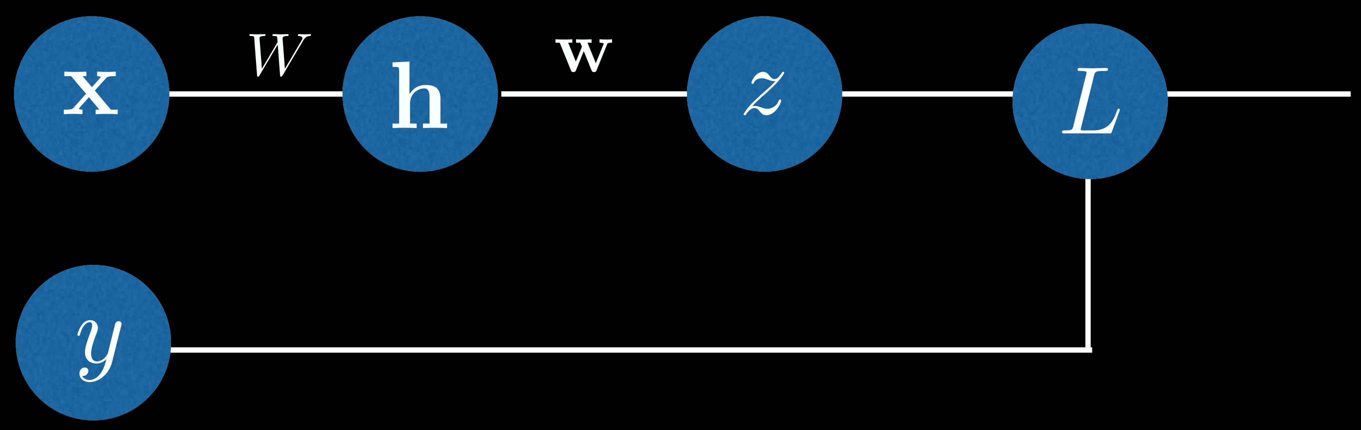
$$\zeta(r) = \log(1+e^{r})$$
$$\simeq \max(0, r)$$

Softplus

Common Loss Function for Binary Classifier

$$
\begin{aligned}
L(\theta) &= -\log P(y|\mathbf{x}) \\
&= -\log \sigma((2y-1)z) \\
&= \zeta((1-2y)z)
\end{aligned}
$$

Softplus



Common Loss Function for Binary Classifier

$$
\begin{aligned}
L(\theta) &= -\log P(y|\mathbf{x}) \\
&= -\log \sigma((2y-1)z) \\
&= \zeta((1-2y)z)
\end{aligned}
$$

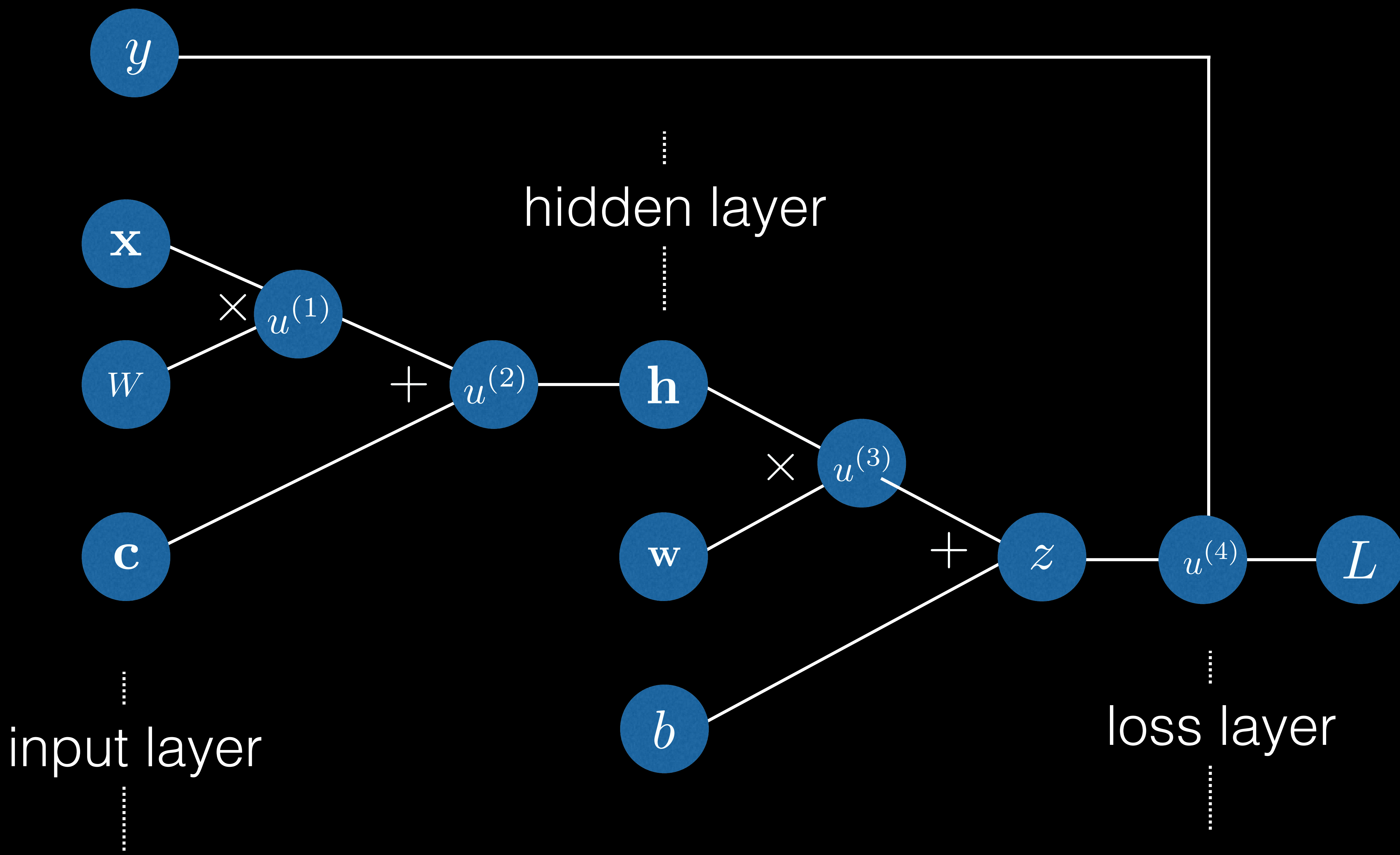
$$\zeta(r) = \log(1 + e^r)$$ Softplus

- Softplus has strong gradients when training sample is misclassified. It does not saturate.

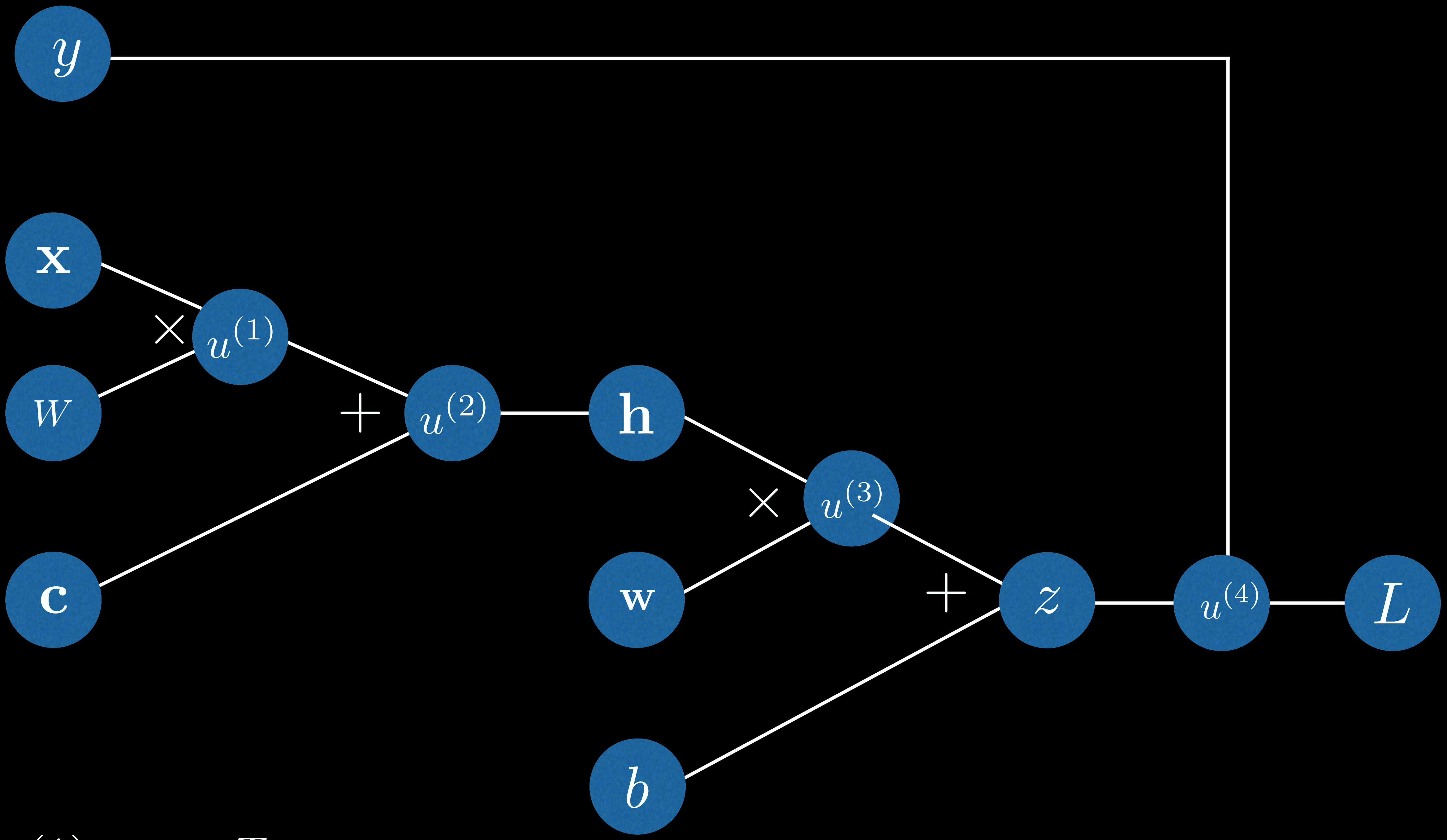
- Softplus is differentiable everywhere.

Let’s make a more descriptive version of the computational graph for this network and then use backdrop to compute the gradient with respect to the network parameters.

Forward Propagation

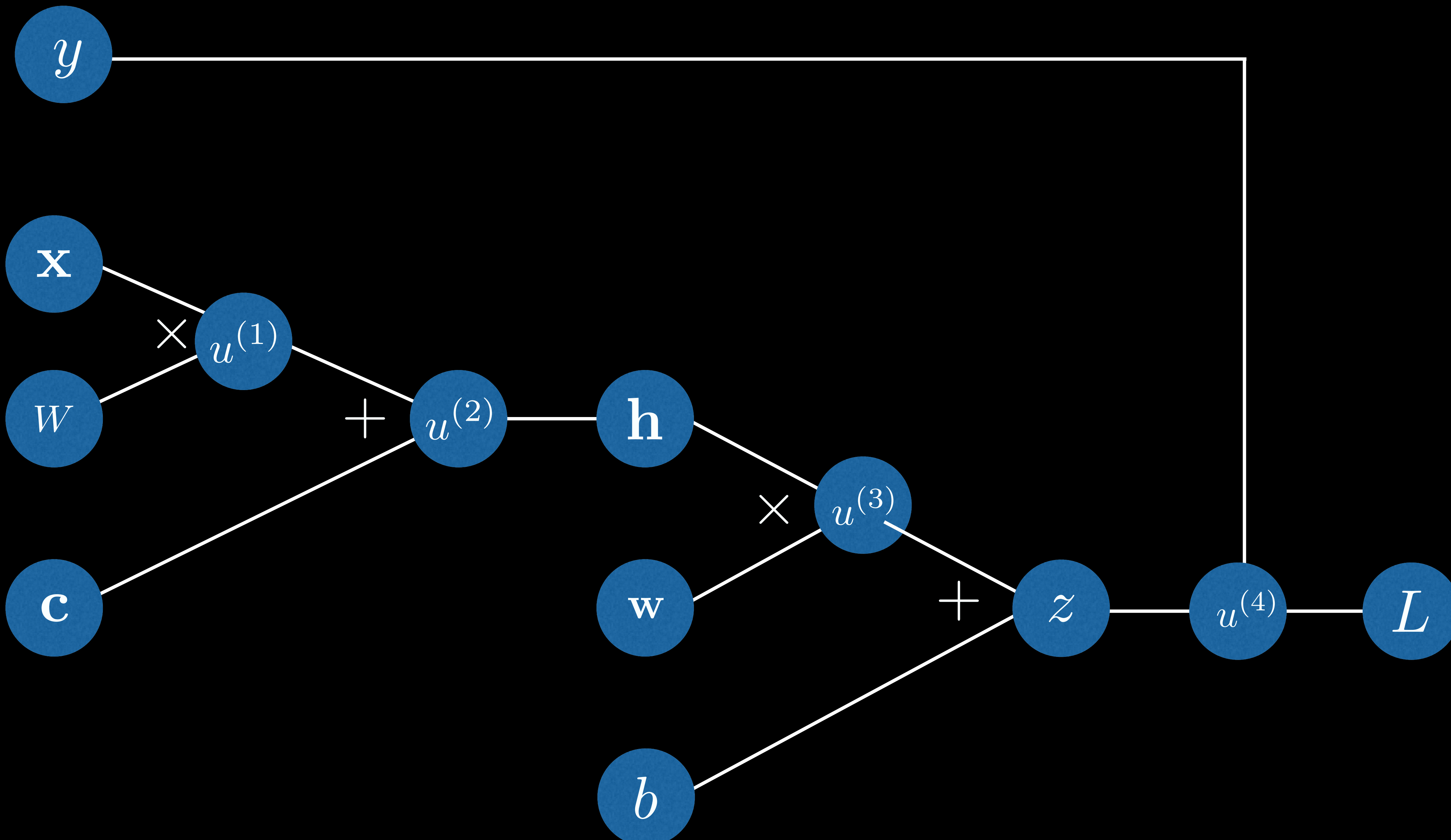


Forward Propagation



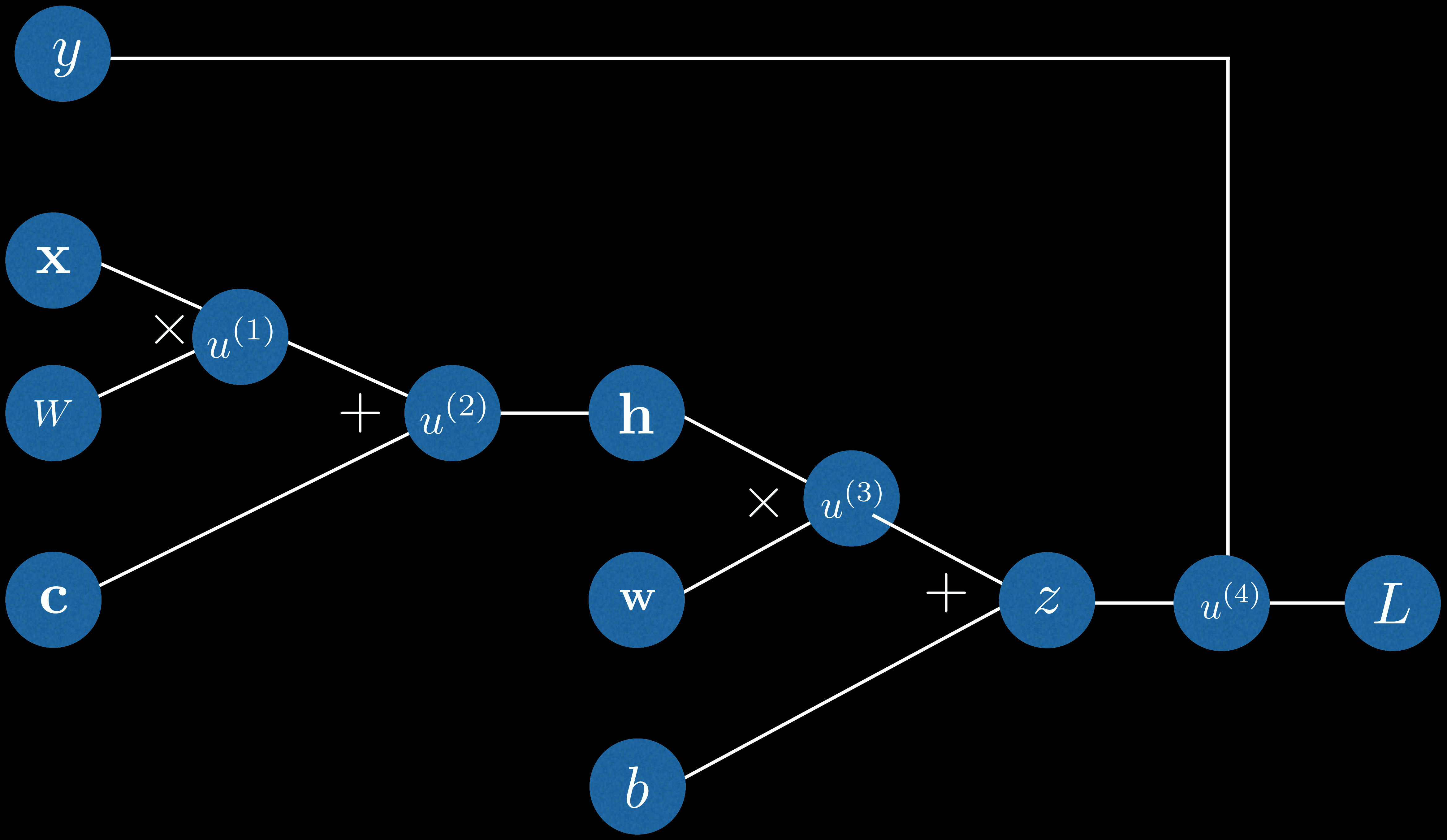
$$u^{(1)} = W^T \mathbf{x}$$

Forward Propagation



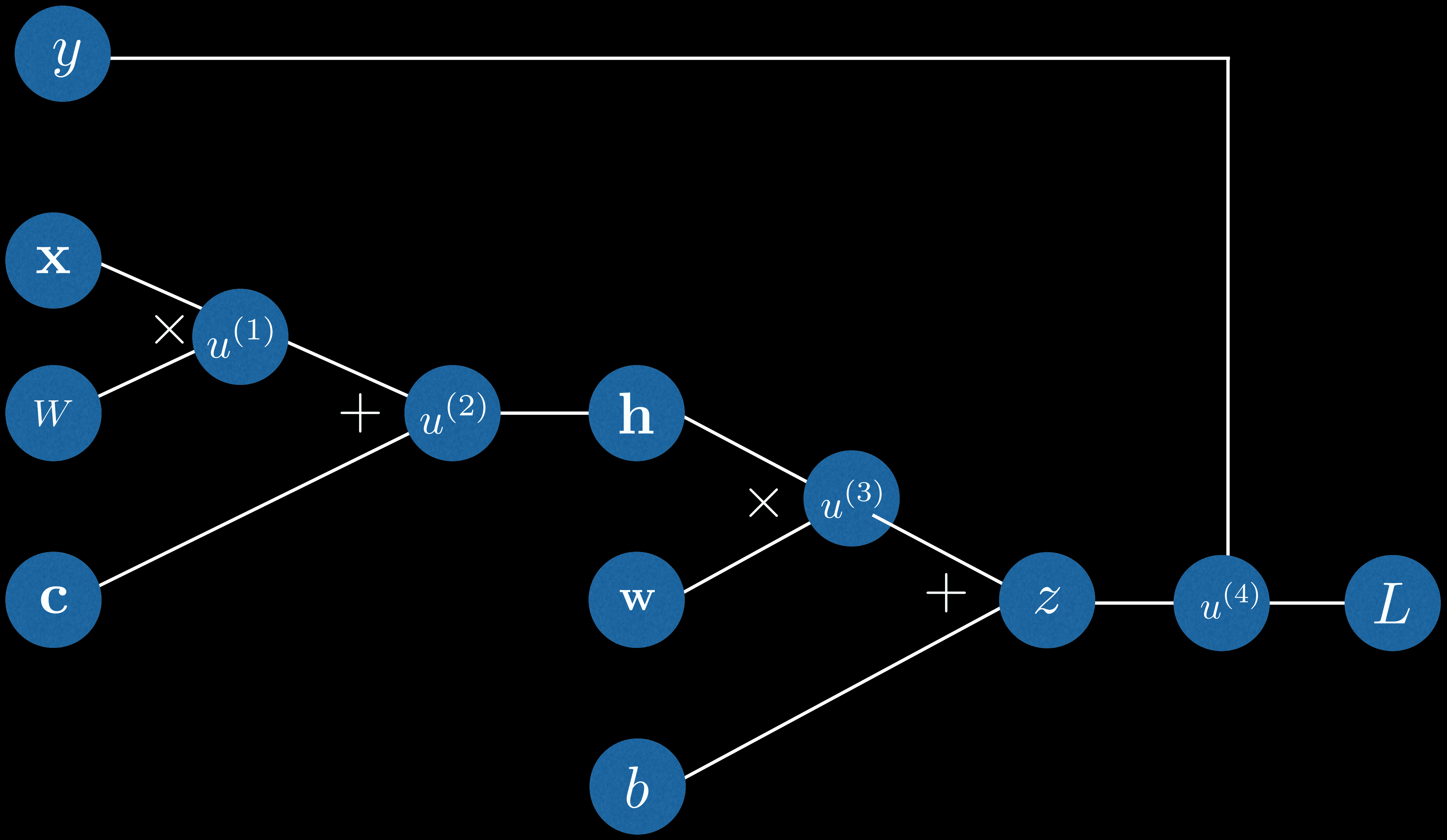
$$u^{(2)} = u^{(1)} + \mathbf{c}$$

Forward Propagation



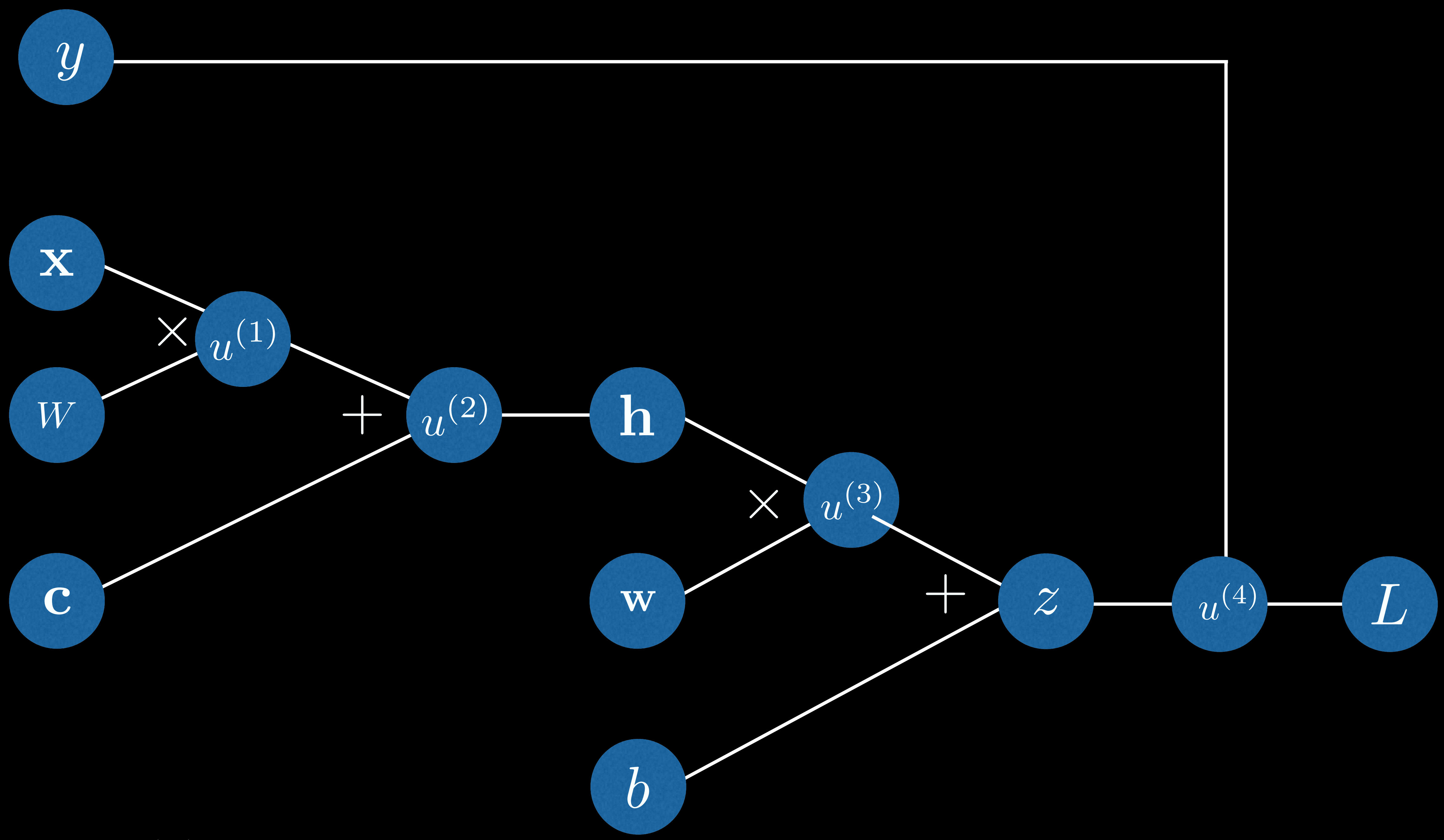
$$\mathbf{h} = \max(\mathbf{0}, \mathbf{u}^{(\mathbf{2})})$$

Forward Propagation



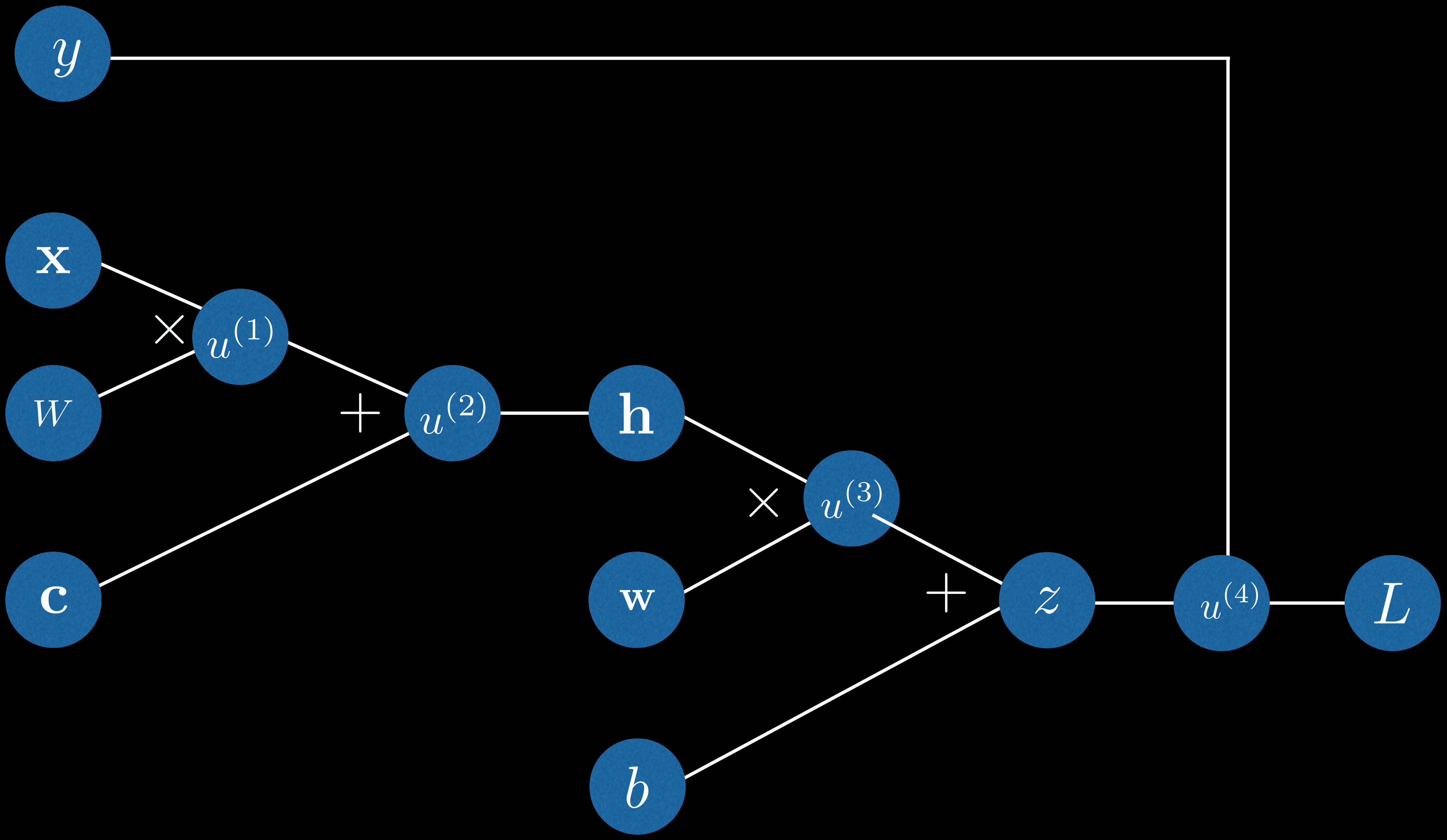
$$u^{(3)} = \mathbf{w}^T \mathbf{h}$$

Forward Propagation



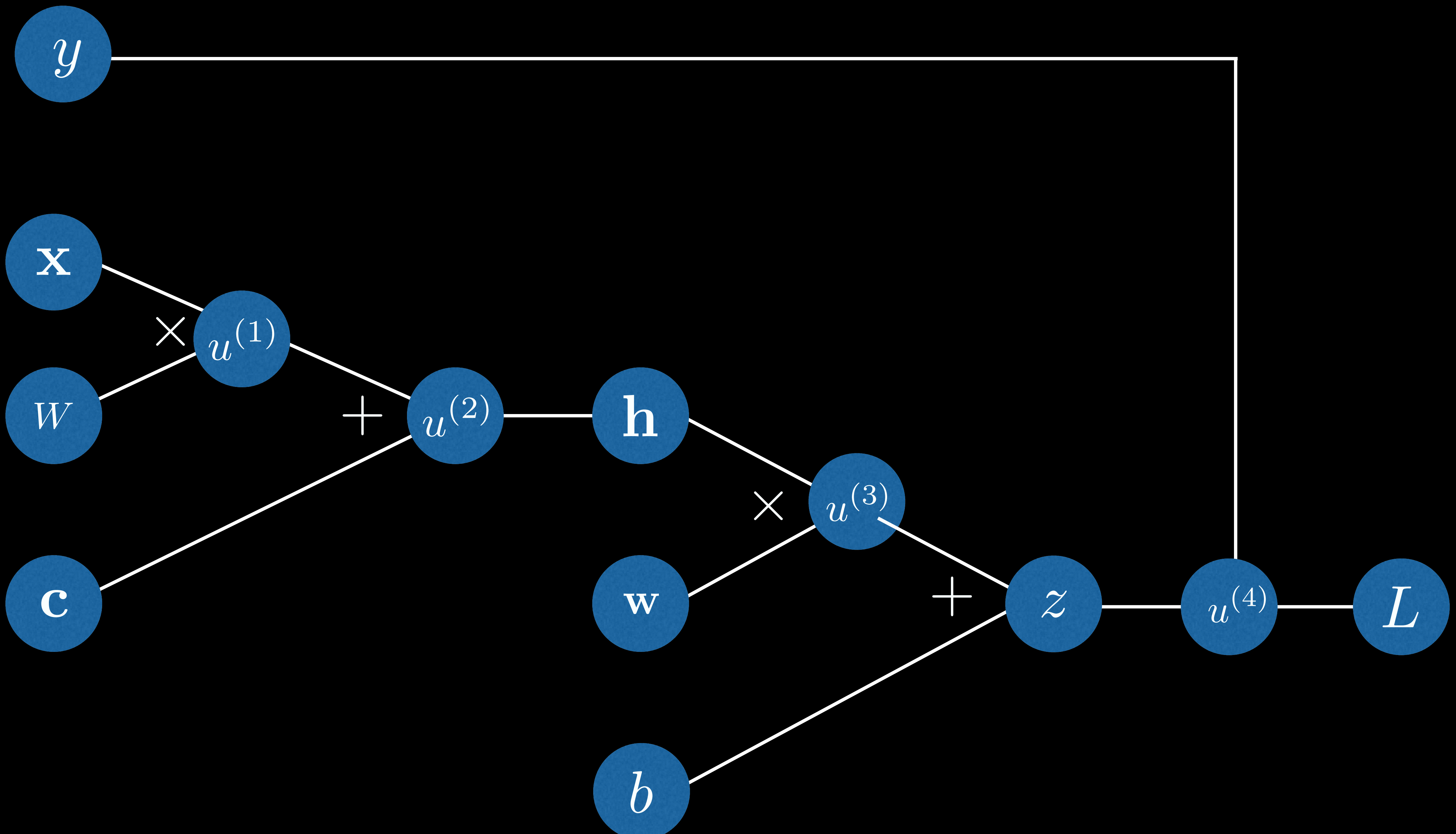
$$z = u^{(3)} + b$$

Forward Propagation



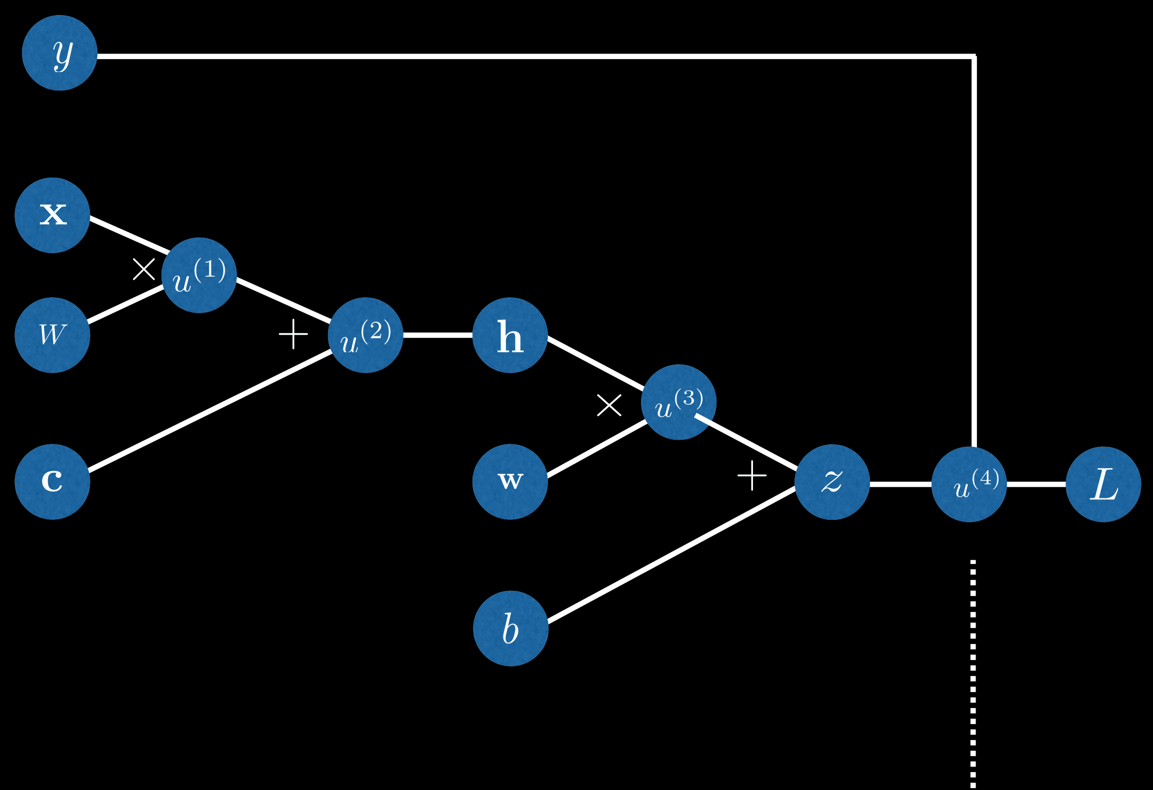
$$u^{(4)} = (1 - 2y)z$$

Forward Propagation



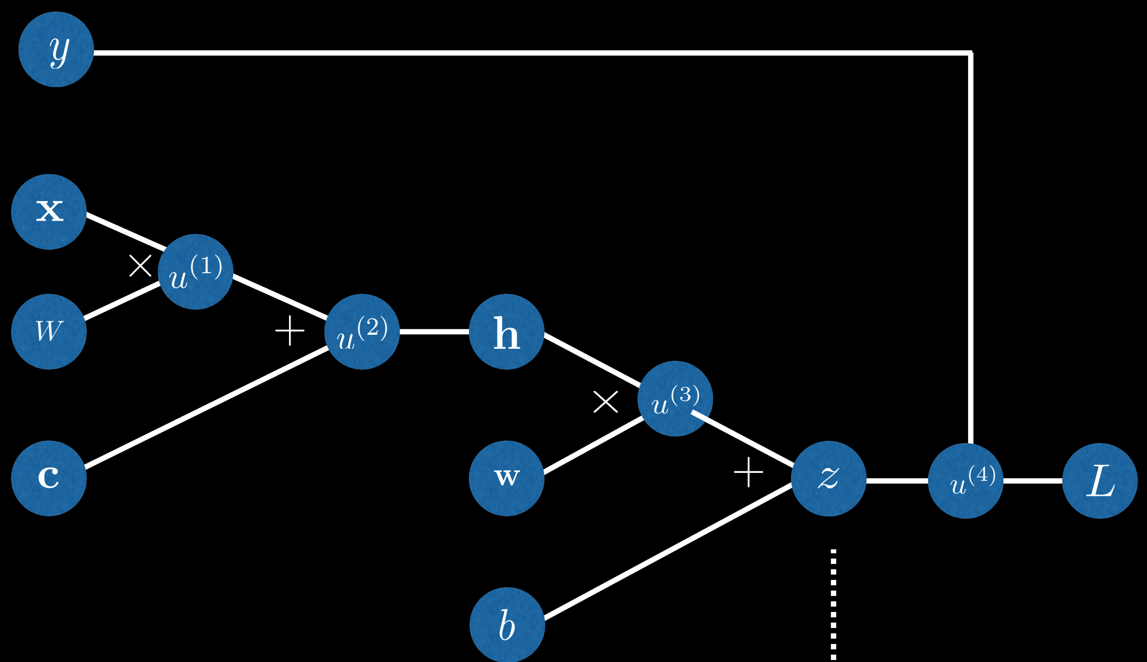
$$L = \zeta(u^{(4)})$$

Back-Propagation



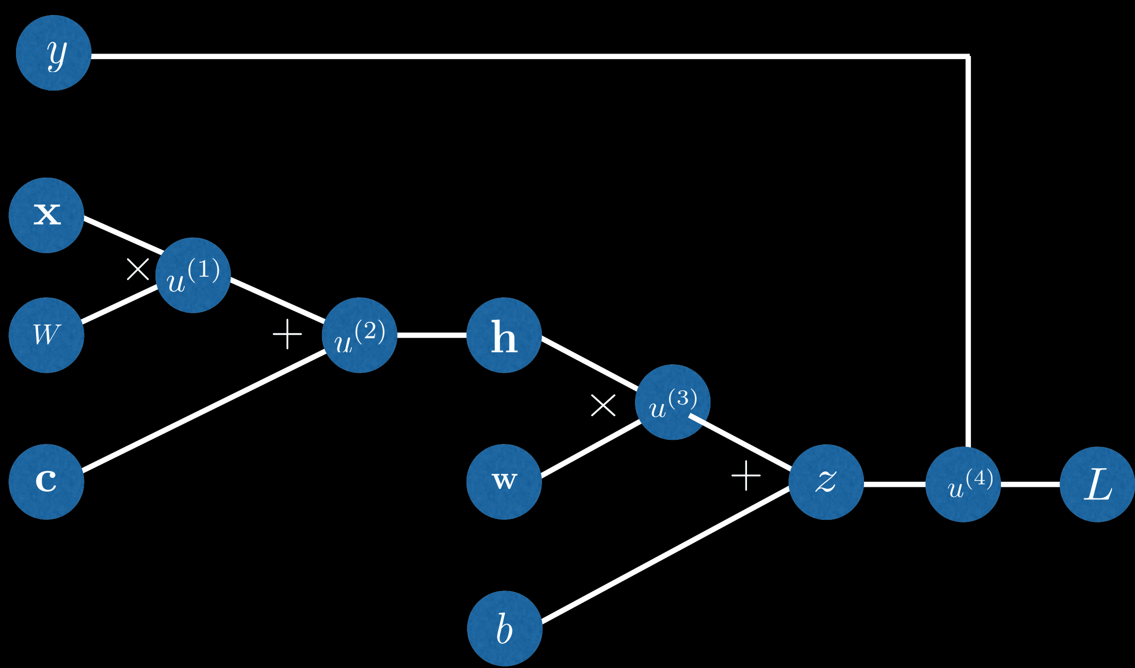
$$\frac{\partial L}{\partial u^{(4)}} = \sigma(u^{(4)})$$

Back-Propagation



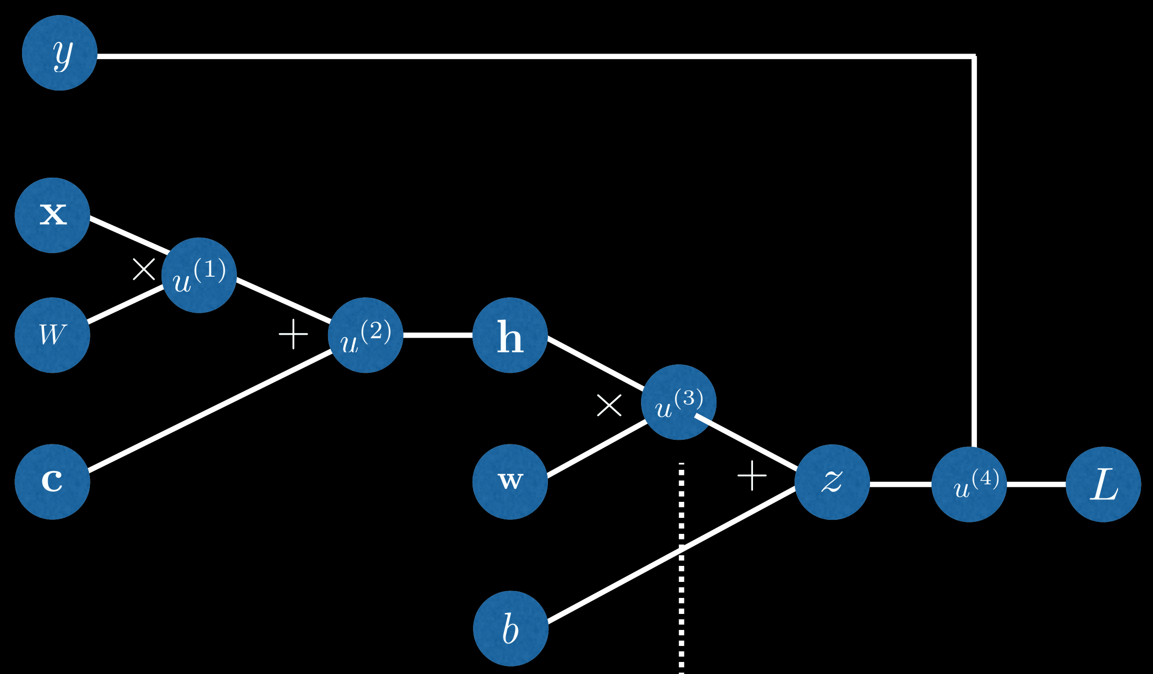
$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial u^{(4)}} \frac{\partial u^{(4)}}{\partial z} = \sigma(u^{(4)})(1 - 2y)$$

Back-Propagation



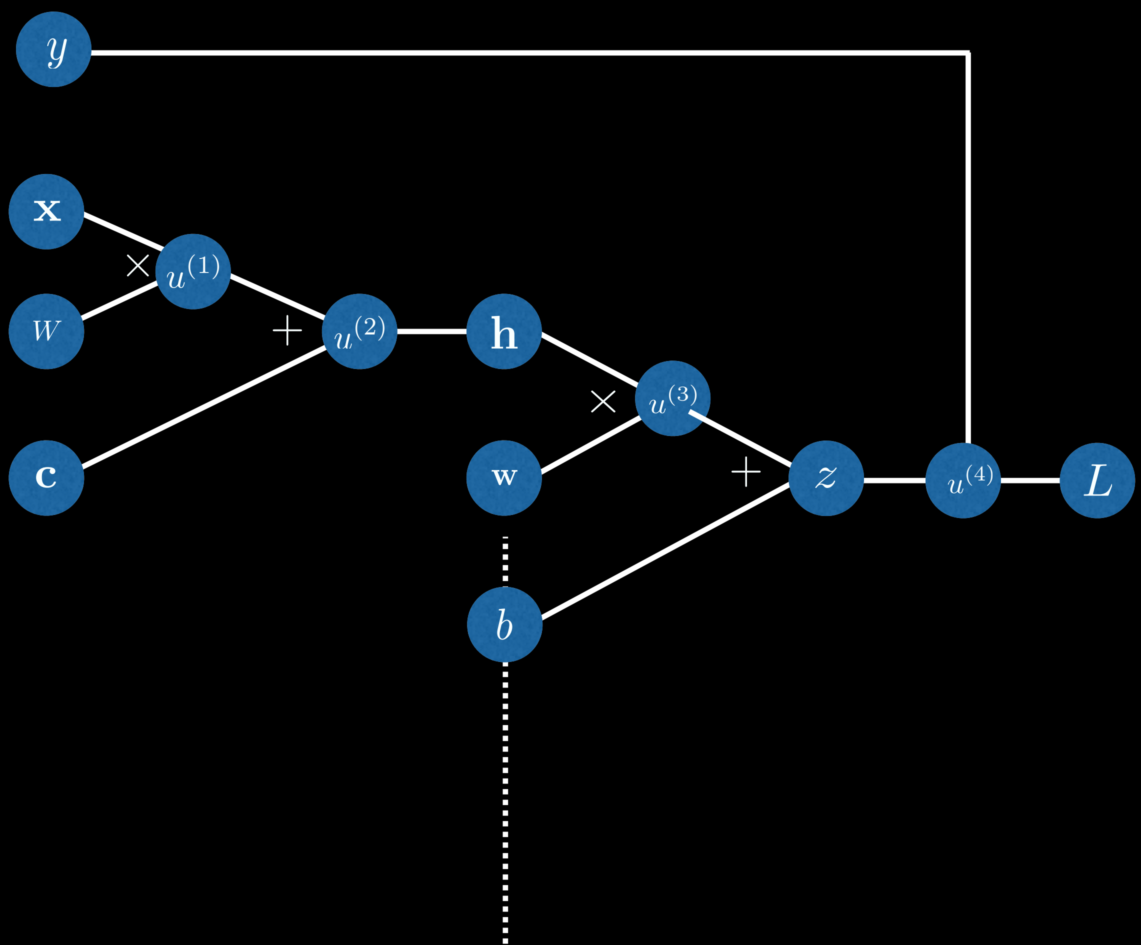
$$\boxed{\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial b} = \sigma(u^{(4)})(1-2y)}$$

Back-Propagation



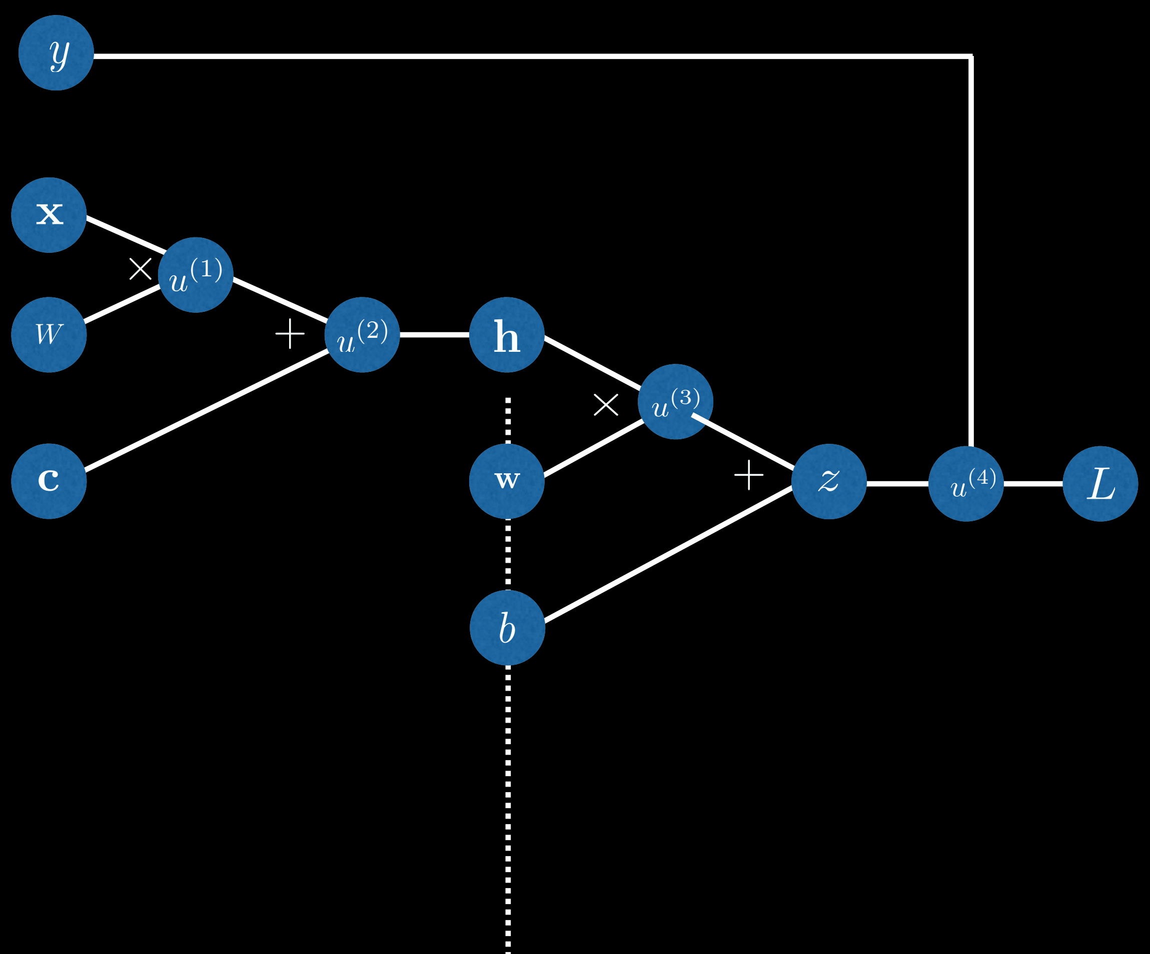
$$\frac{\partial L}{\partial u^{(3)}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial u^{(3)}} = \sigma(u^{(4)})(1-2y)(1)$$

Back-Propagation



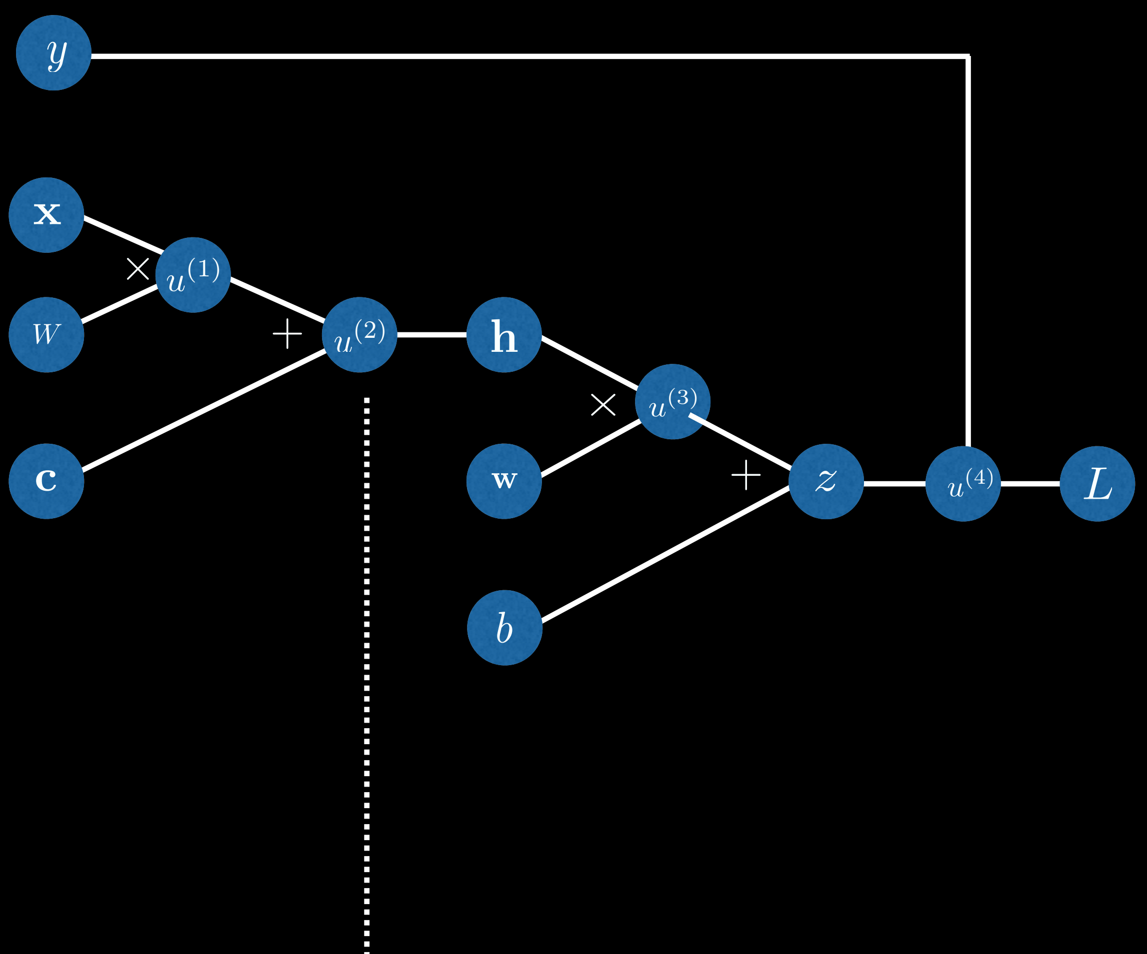
$$\boxed{\nabla_{\mathbf{w}} L = \frac{\partial L}{\partial u^{(3)}} \nabla_{\mathbf{w}} u^{(3)} = \sigma(u^{(4)})(1 - 2y)\mathbf{h}}$$

Back-Propagation



$$\nabla_{\mathbf{h}} L = \frac{\partial L}{\partial u^{(3)}} \nabla_{\mathbf{h}} u^{(3)} = \sigma(u^{(4)})(1 - 2y)\mathbf{w}$$

Back-Propagation



Here we have a mapping from $\mathbb{R}^2 \rightarrow \mathbb{R}^2$

$$\mathbb{R}^2 \leftarrow \mathbb{R}^2$$

$$\mathbf{h} = \max(\mathbf{0}, \mathbf{u^{(2)}})$$

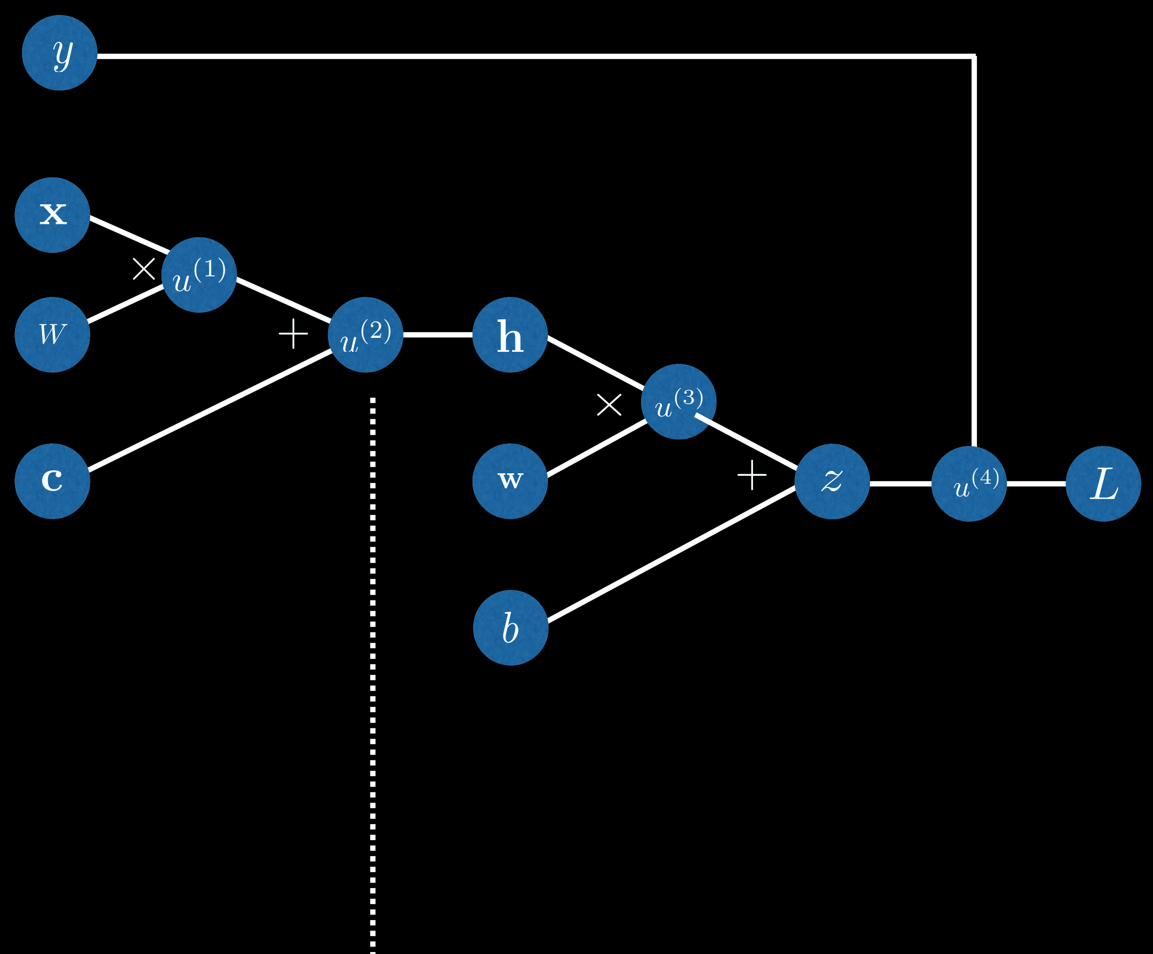
Jacobians and more…

$$\mathbb{R}^n \leftarrow \mathbb{R}^m \qquad \mathbb{R} \leftarrow \mathbb{R}^n$$

$$\mathbf{y} = g(\mathbf{x}) \qquad z = f(\mathbf{y}) \qquad \Longrightarrow \quad \nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^T \nabla_{\mathbf{y}} z$$

where the Jacobian $$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_m} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial y_n}{\partial x_1} & \frac{\partial y_n}{\partial x_2} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix}_{n \times m}$$

Back-Propagation



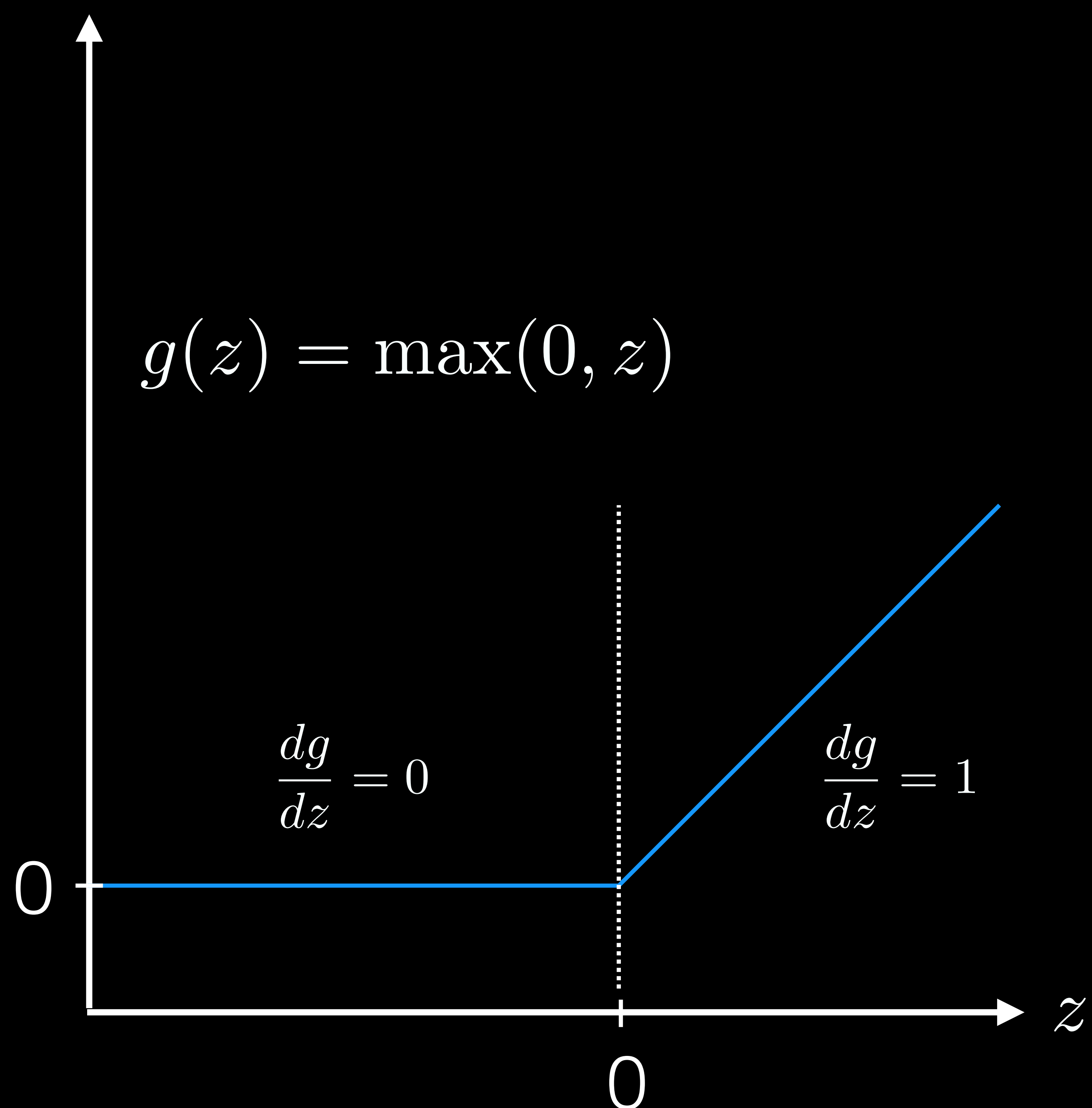
Here we have a mapping from $\mathbb{R}^2 \to \mathbb{R}^2$

$$\mathbb{R}^2 \leftarrow \mathbb{R}^2$$

$$\mathbf{h} = \max(\mathbf{0}, \mathbf{u^{(2)}})$$

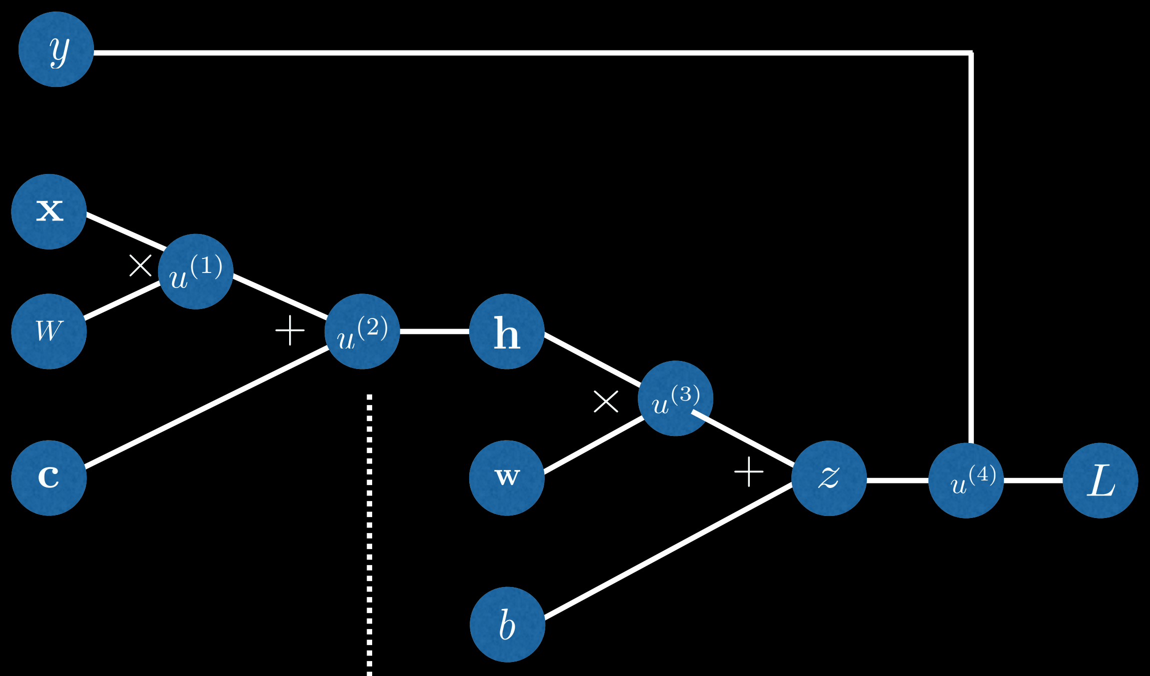
$$\frac{\partial \mathbf{h}}{\partial \mathbf{u^{(2)}}} = \begin{bmatrix} \frac{\partial h_1}{\partial u_1^{(2)}} & \frac{\partial h_1}{\partial u_2^{(2)}} \\ \frac{\partial h_2}{\partial u_1^{(2)}} & \frac{\partial h_2}{\partial u_2^{(2)}} \end{bmatrix} = \begin{bmatrix} \frac{\partial h_1}{\partial u_1^{(2)}} & 0 \\ 0 & \frac{\partial h_2}{\partial u_2^{(2)}} \end{bmatrix}$$

Rectified Linear Unit ReLU



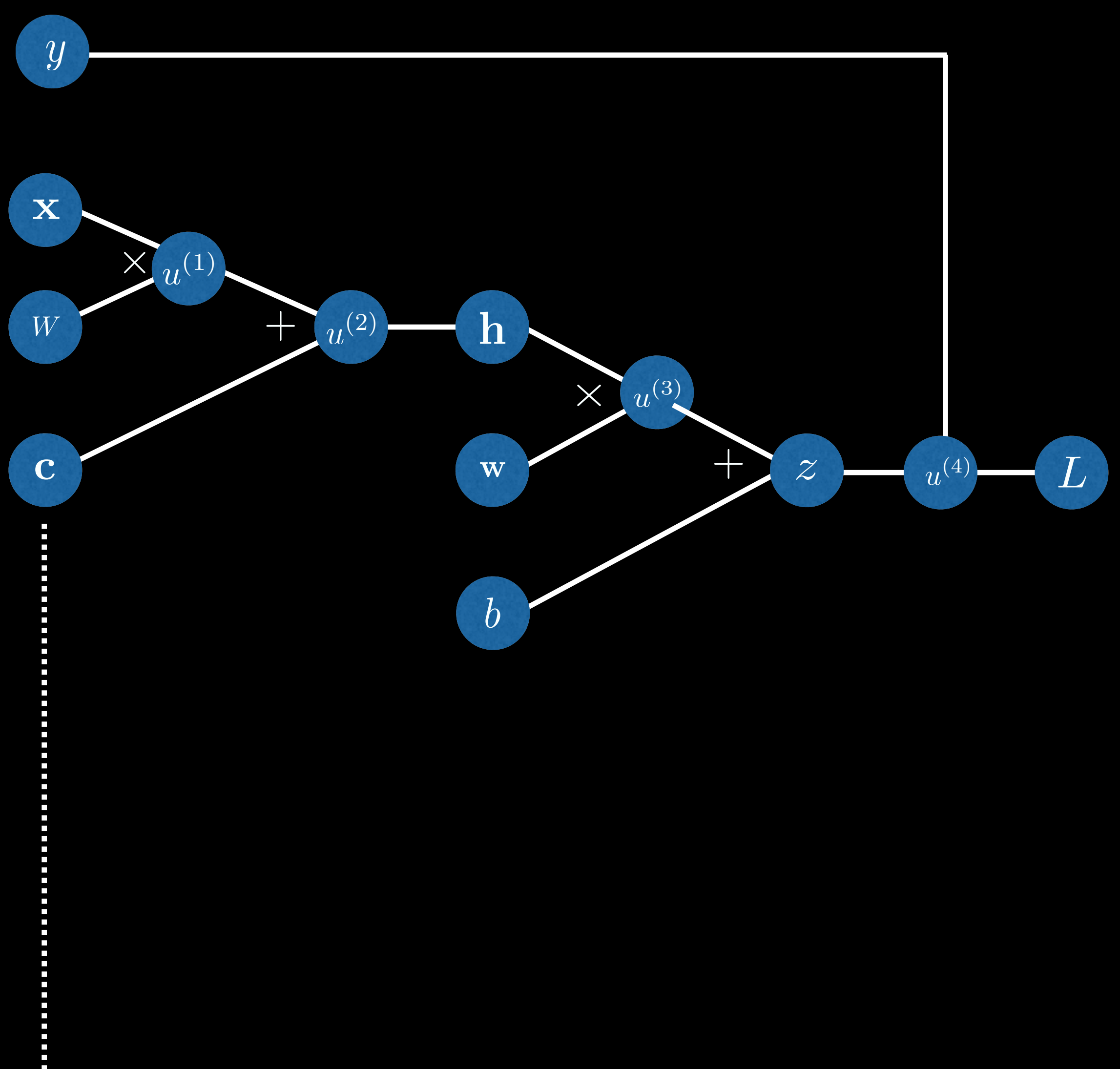
Let $f = \text{``}\frac{dg}{dz}\text{''}$

Back-Propagation



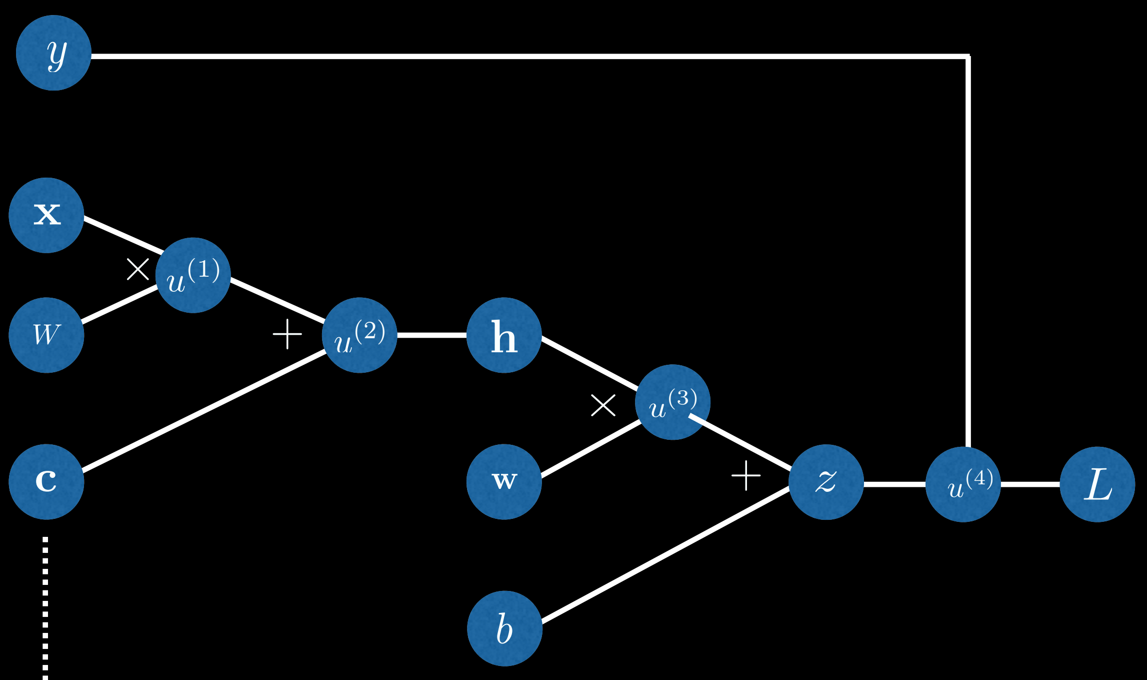
$$\nabla_{\mathbf{u}^{(2)}} L = \left(\frac{\partial \mathbf{h}}{\partial \mathbf{u}^{(2)}} \right)^T \nabla_{\mathbf{h}} L = \sigma(u^{(4)})(1-2y)\mathbf{f}(\mathbf{u}^{(2)}) \odot \mathbf{w}$$

Back-Propagation



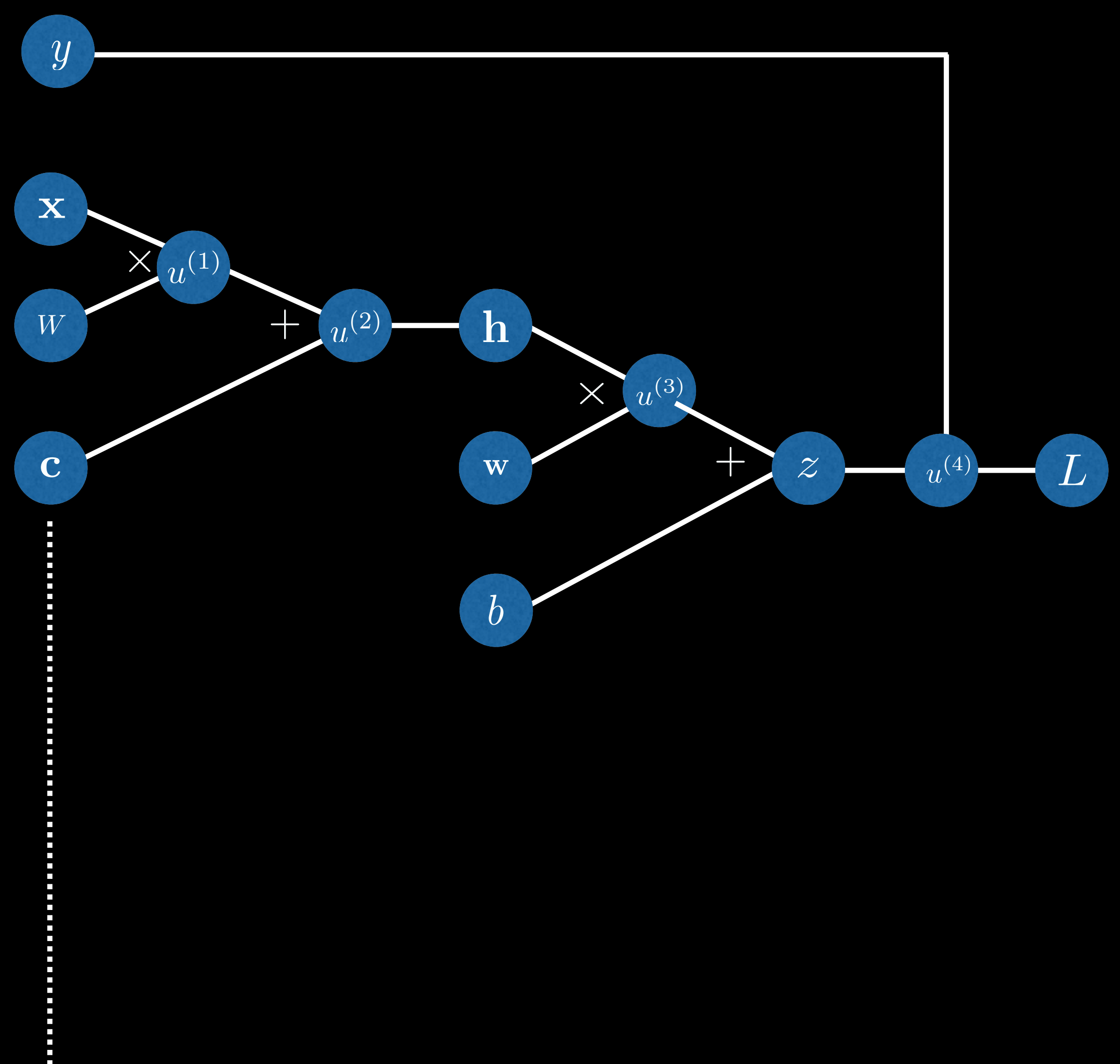
$$\nabla_{\mathbf{c}} L = \left(\frac{\partial \mathbf{u}^{(\mathbf{2})}}{\partial \mathbf{c}} \right)^T \nabla_{\mathbf{u}^{(\mathbf{2})}} L$$

Back-Propagation



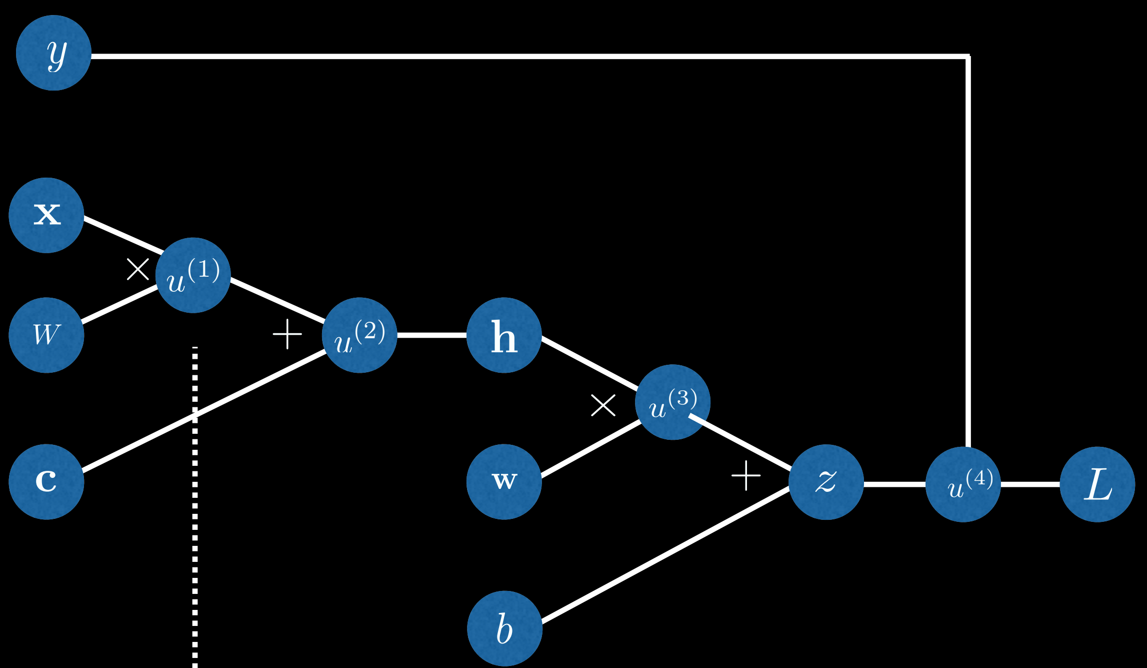
$$\nabla_{\mathbf{c}} L = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T \nabla_{\mathbf{u^{(2)}}} L$$

Back-Propagation



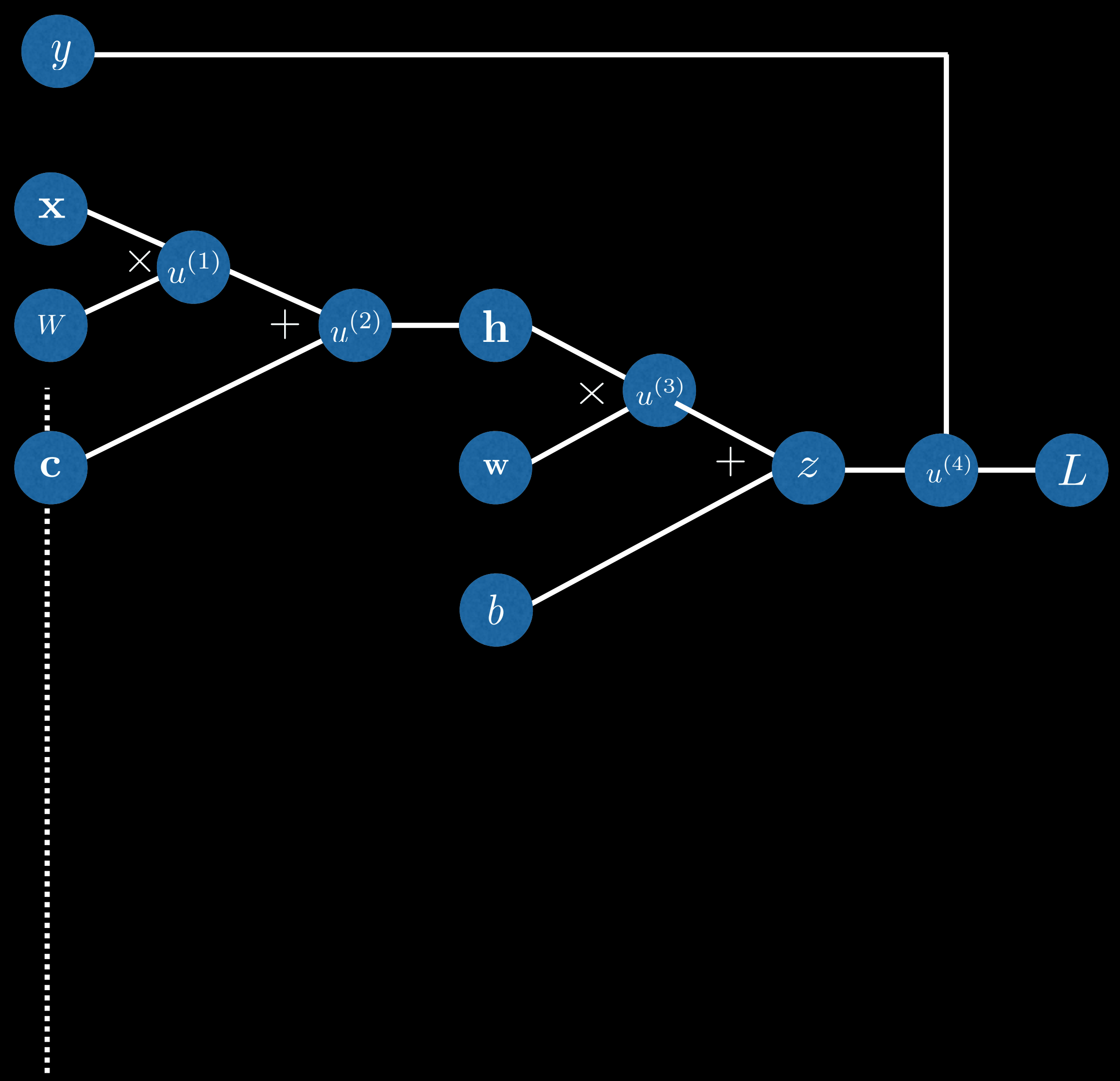
$$\nabla_{\mathbf{c}} L = \sigma(u^{(4)})(1 - 2y)\mathbf{f}(\mathbf{u}^{(\mathbf{2})}) \odot \mathbf{w}$$

Back-Propagation



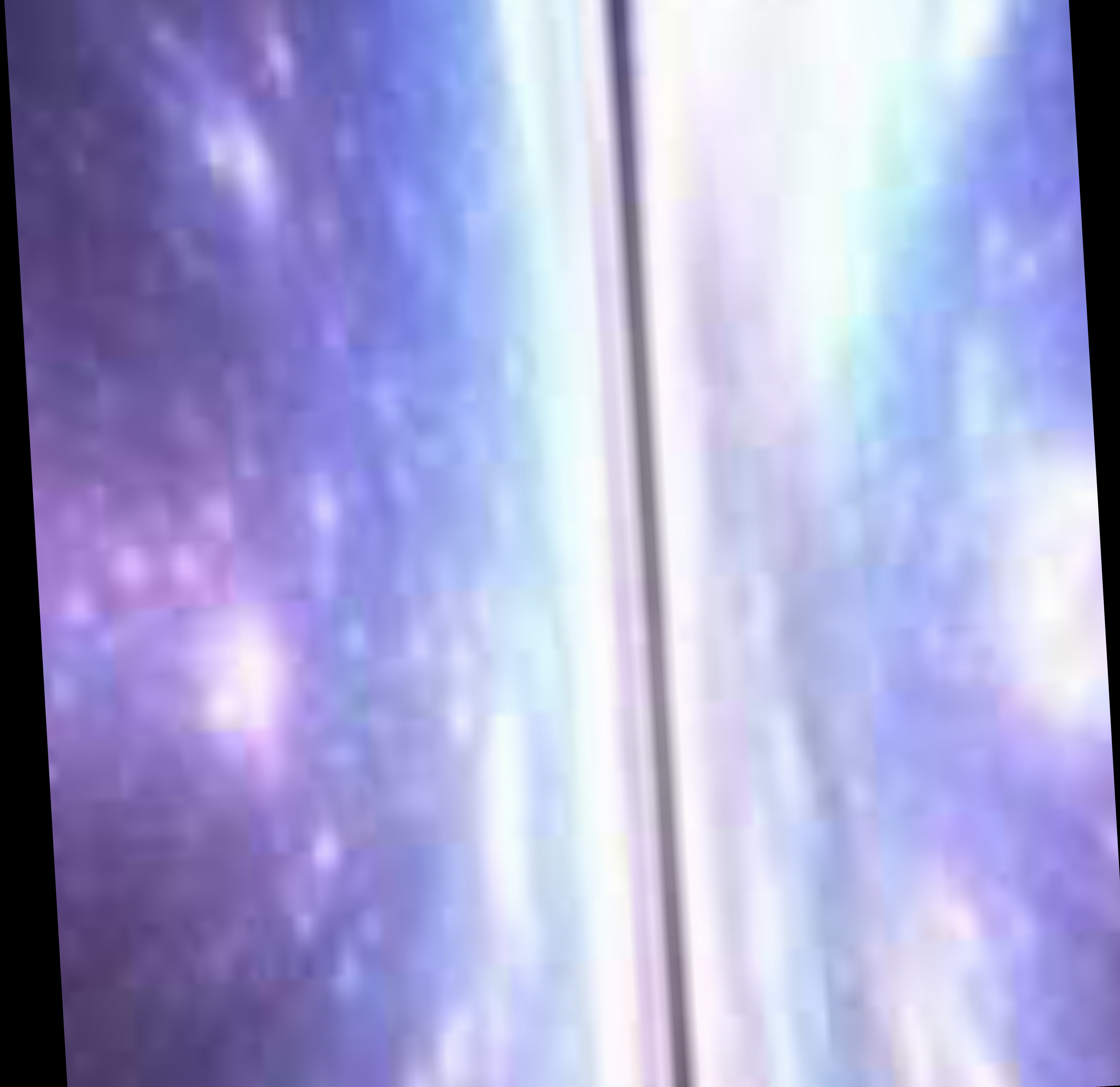
$$\nabla_{\mathbf{u}^{(1)}} L = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \nabla_{\mathbf{u}^{(2)}} L = \sigma(u^{(4)})(1-2y)\mathbf{f}(\mathbf{u}^{(2)}) \odot \mathbf{w}$$

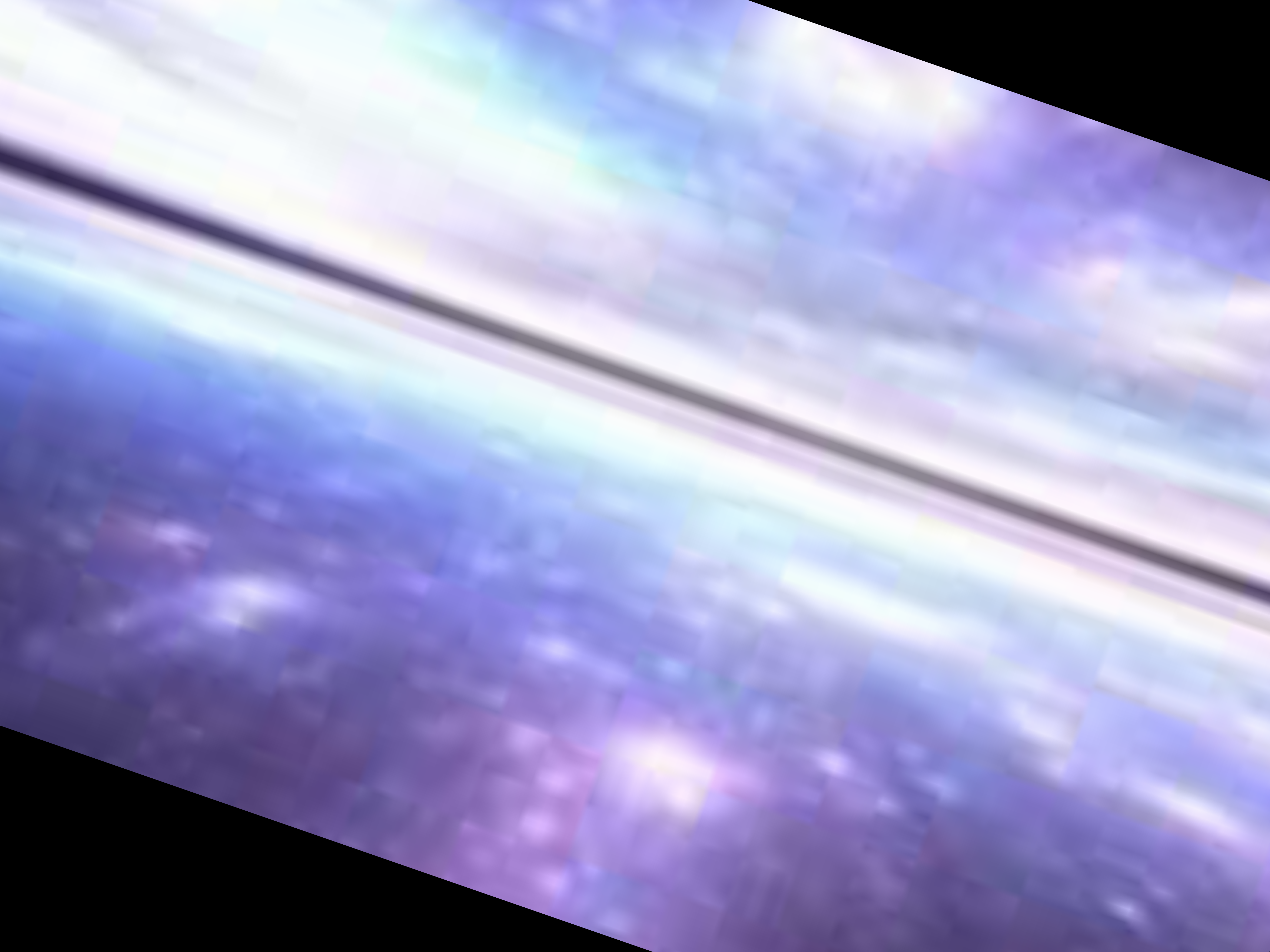
Back-Propagation



Here is mapping from $\mathbb{R}^{2\times 2} \rightarrow \mathbb{R}^2$

$$u^{(1)} = W^T \mathbf{x}$$





Tensors! …WTF?

- In physics tensors have specific roles and rules.
- In ML we usually think of tensors as a generalization of vectors and matrices.
- So we use tensors to organize data into some grid form. A **vector** has 1 index, a **matrix** has 2 indices, and **tensors** can be used when more indices are needed.

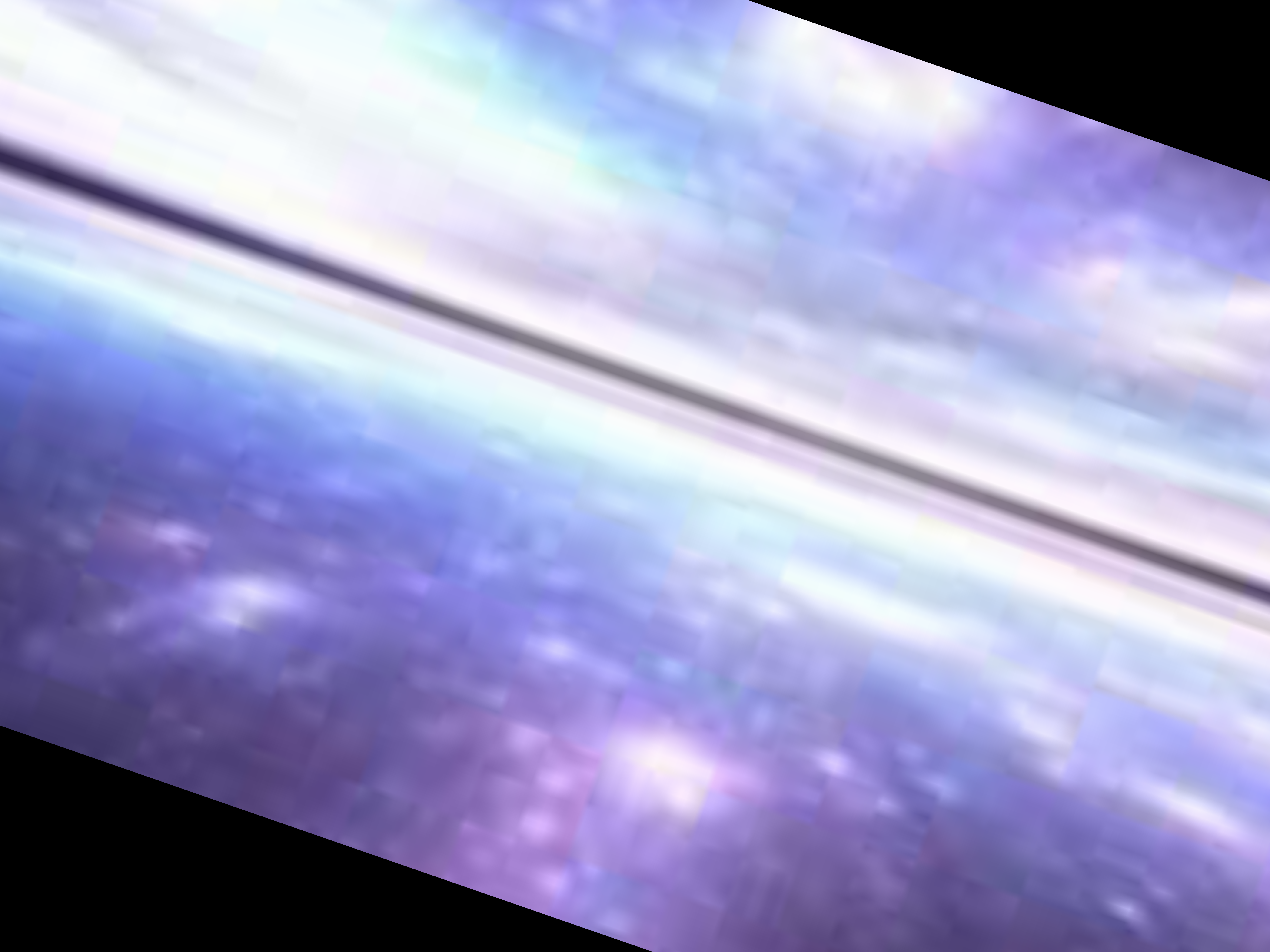
Tensors

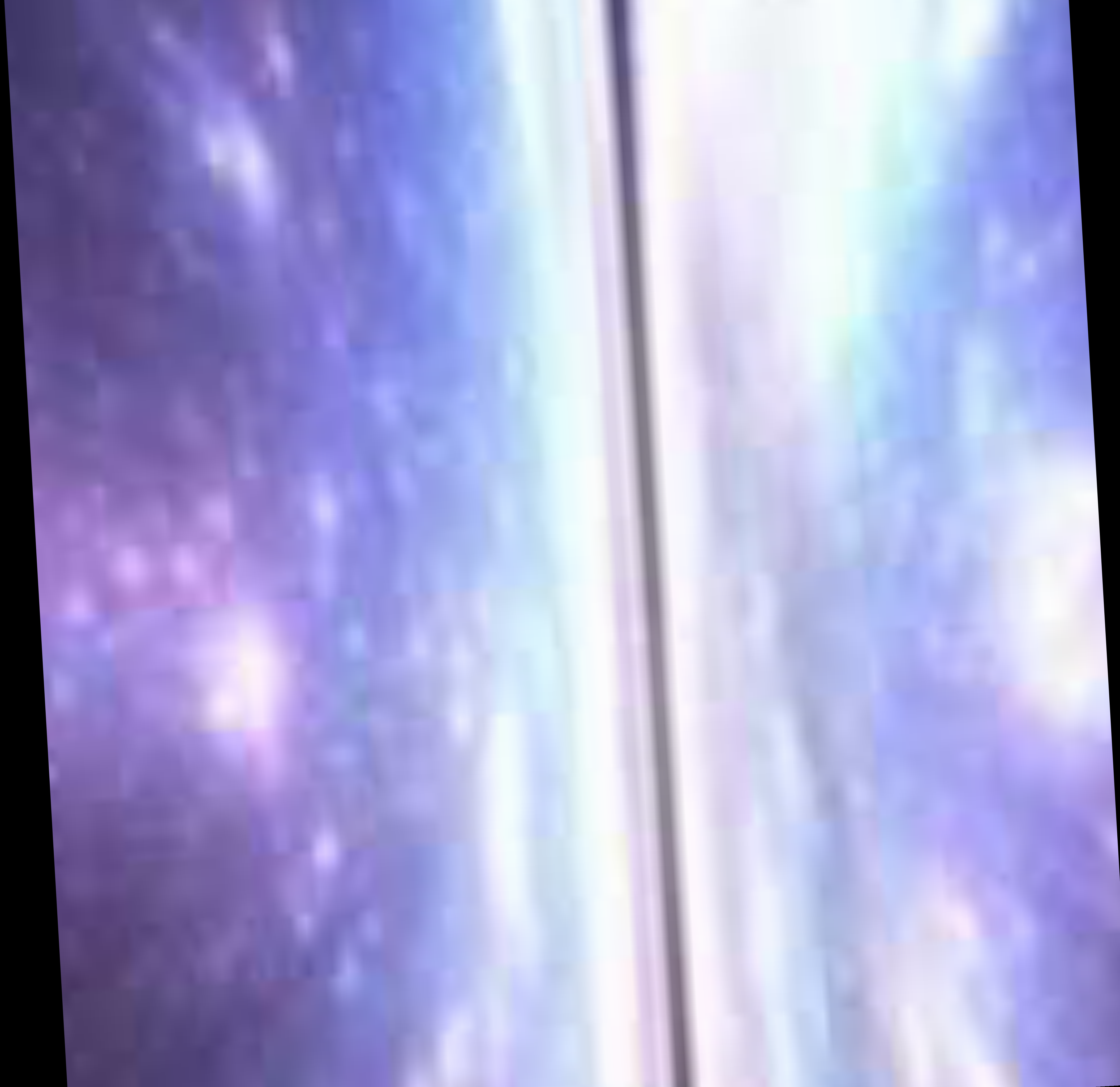
- Let **X** be a tensor.
- Even though **X** may have any number of indices we can think of this tuple of indices as simply $\mathbf{i}$
- So an element of our tensor can be written $\mathbf{X_i}$
- And the gradient of z wrt to tensor **X** is $\nabla_{\mathbf{X}} z$

Tensors

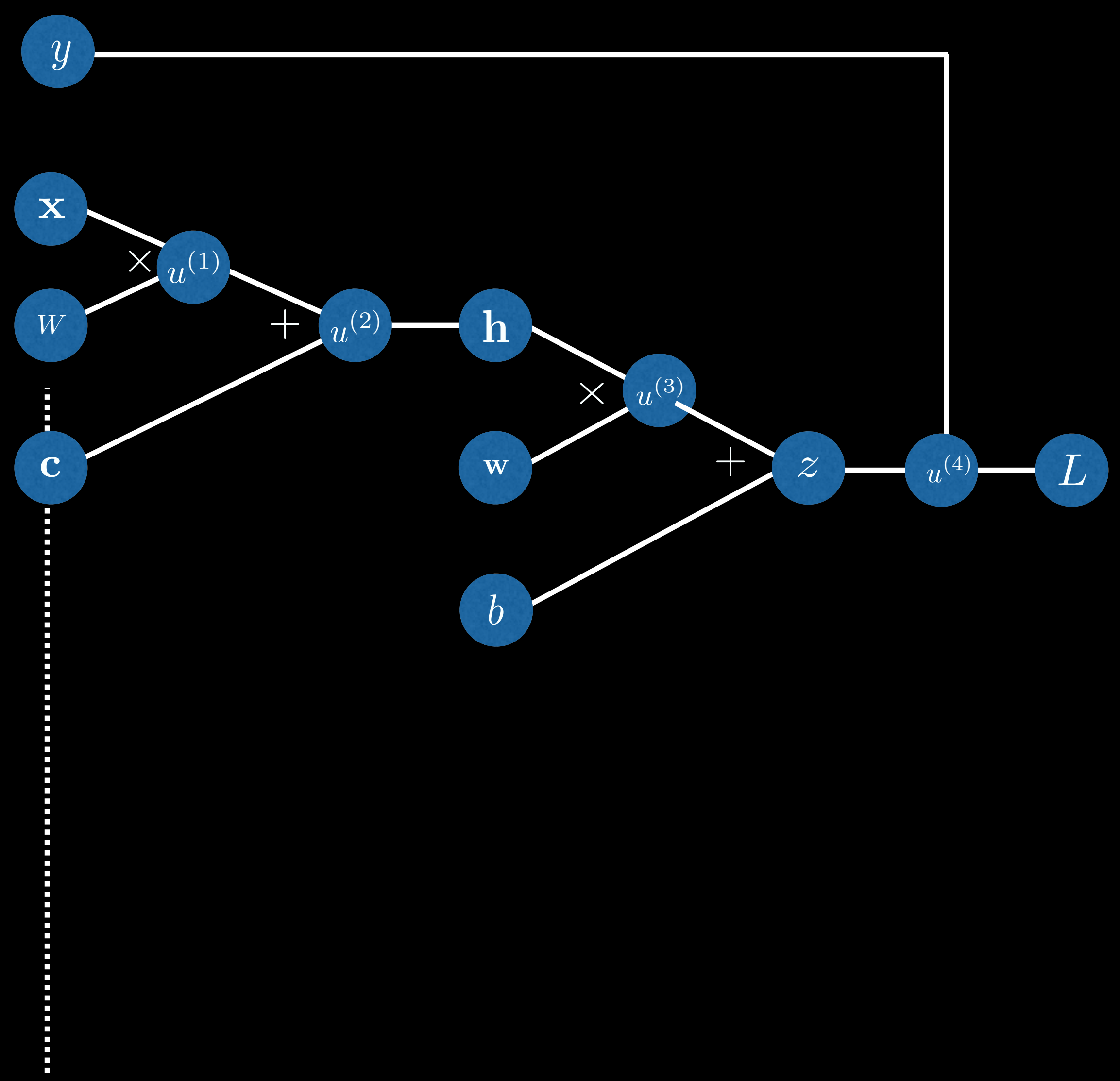
- So if **Y** = g(**X**) and z = f(**Y**) then

$$\nabla_{\mathbf{X}} z = \sum_j \left(\nabla_{\mathbf{X}} \mathbf{Y}_j\right) \frac{\partial z}{\partial \mathbf{Y}_j}$$





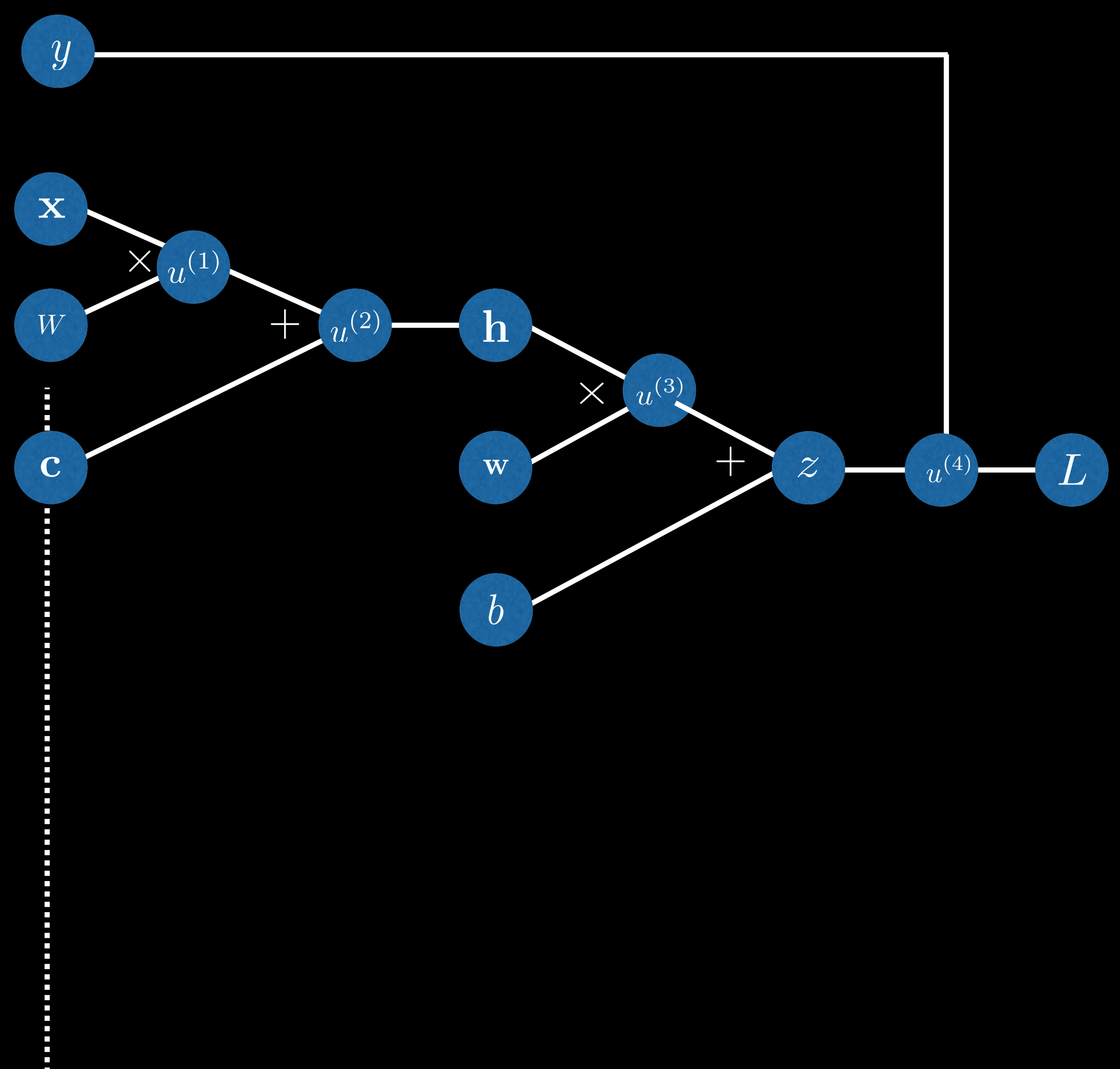
Back-Propagation



Here is mapping from $\mathbb{R}^{2\times 2} \rightarrow \mathbb{R}^2$

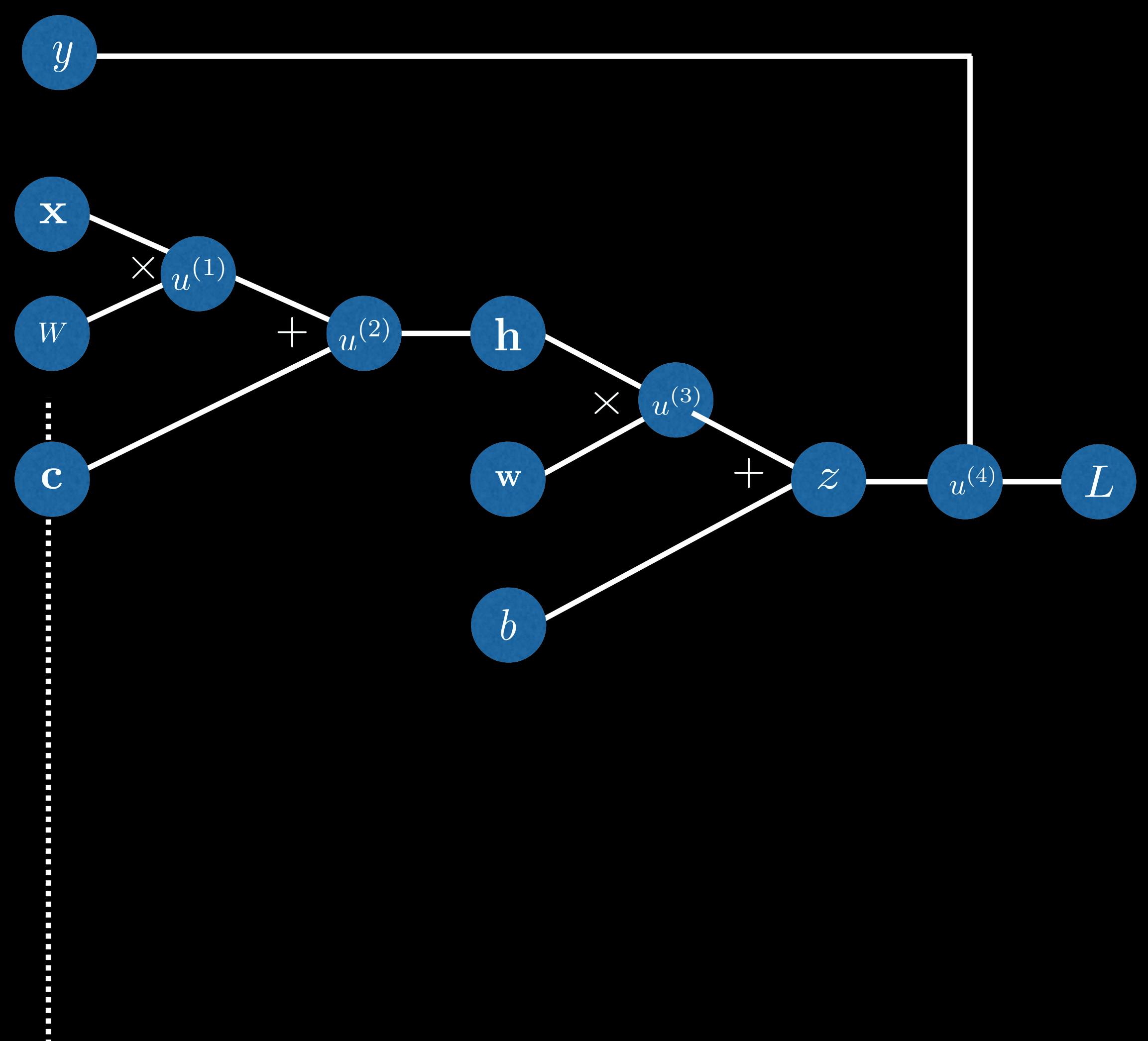
$$u^{(1)} = W^T \mathbf{x}$$

Back-Propagation



$$\nabla_W L = \sum_j \left(\nabla_W u_j^{(1)} \right) (\nabla_{\mathbf{u}^{(1)}} L)_j$$

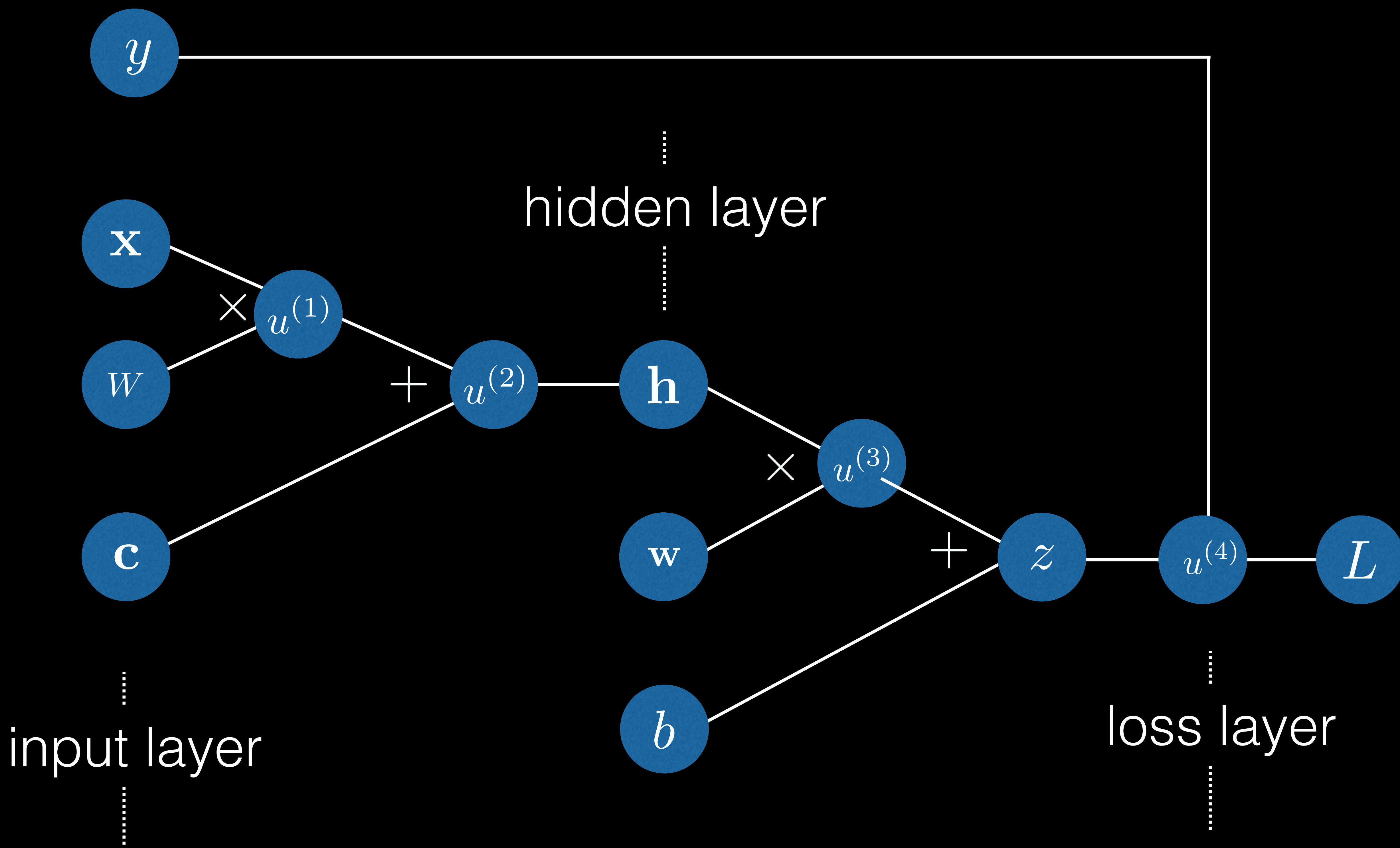
Back-Propagation



$$\nabla_W L = \sigma(u^{(4)})(1-2y)\mathbf{x}\left(\mathbf{f}(\mathbf{u^{(2)}}) \odot \mathbf{w}\right)^T$$

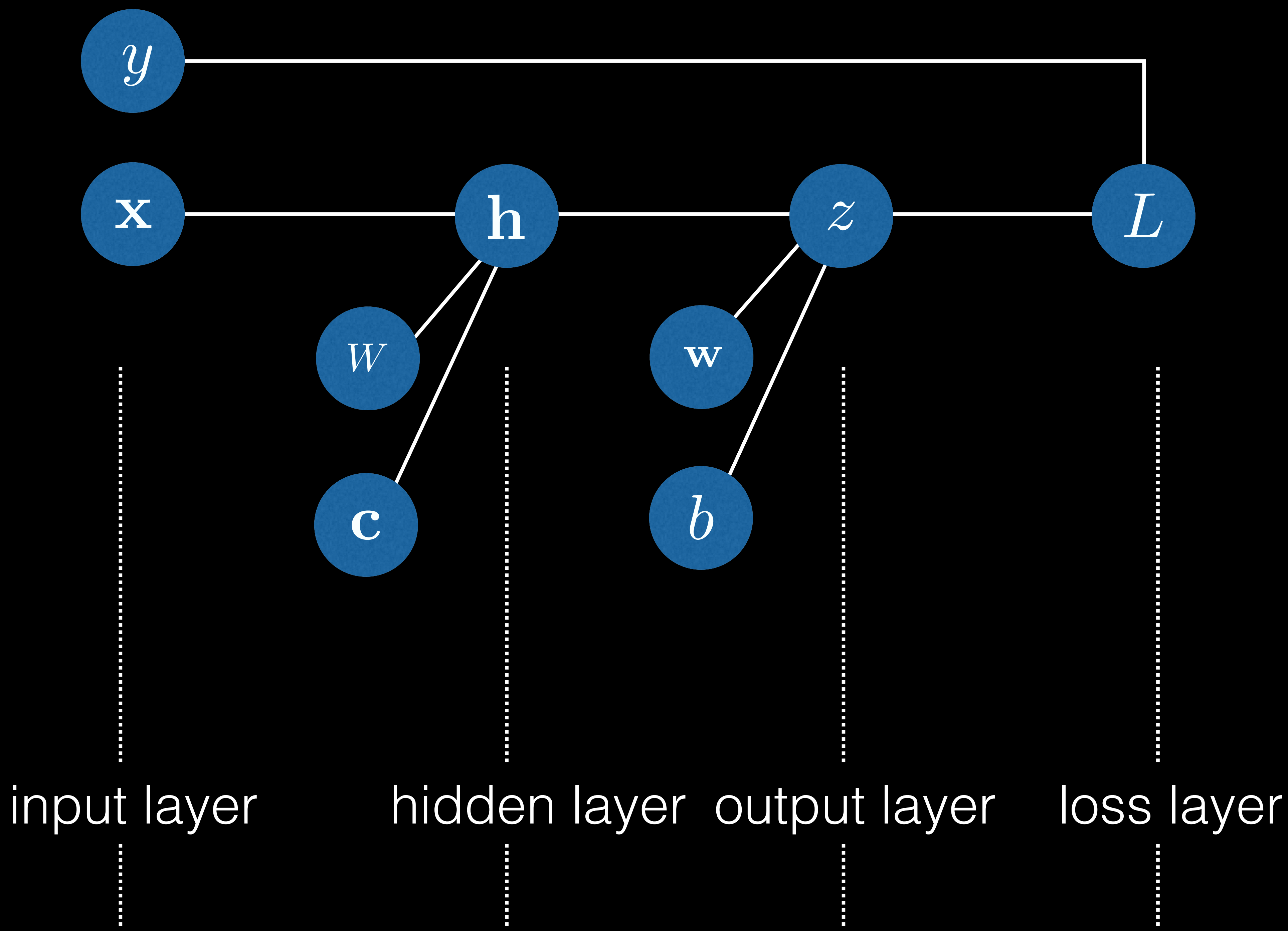
So let’s collapse this diagram into functional layers.

Simple MLP



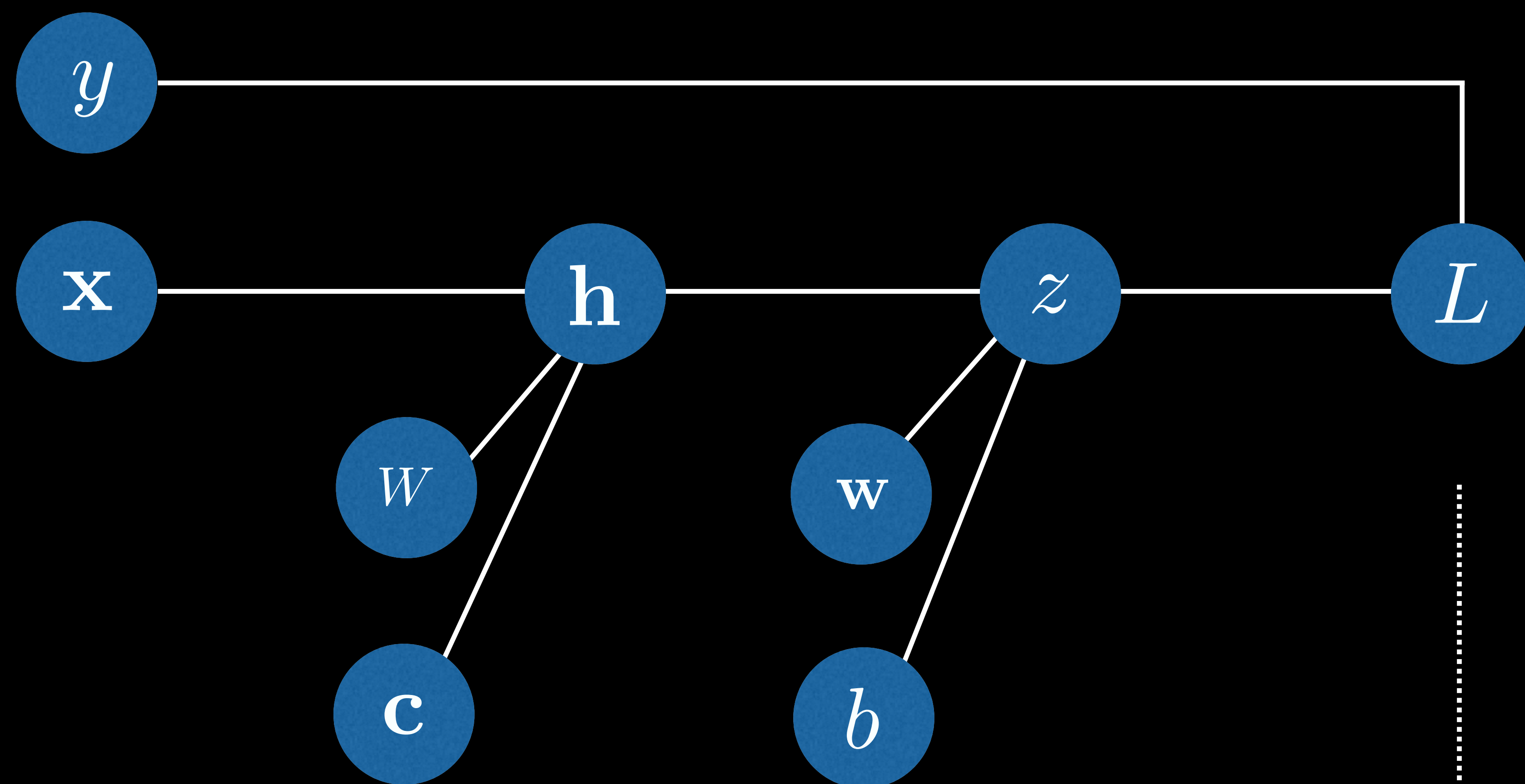
y
hidden layer
$\mathbf{x}$
$\times$
$u^{(1)}$
W
$+$
$u^{(2)}$
$\mathbf{h}$
$\times$
$u^{(3)}$
$\mathbf{c}$
$\mathbf{w}$
$+$
z
$u^{(4)}$
L
b
input layer
loss layer

Simple MLP



y
$\mathbf{x}$
$\mathbf{h}$
z
L
W
$\mathbf{w}$
$\mathbf{c}$
b
input layer
hidden layer
output layer
loss layer

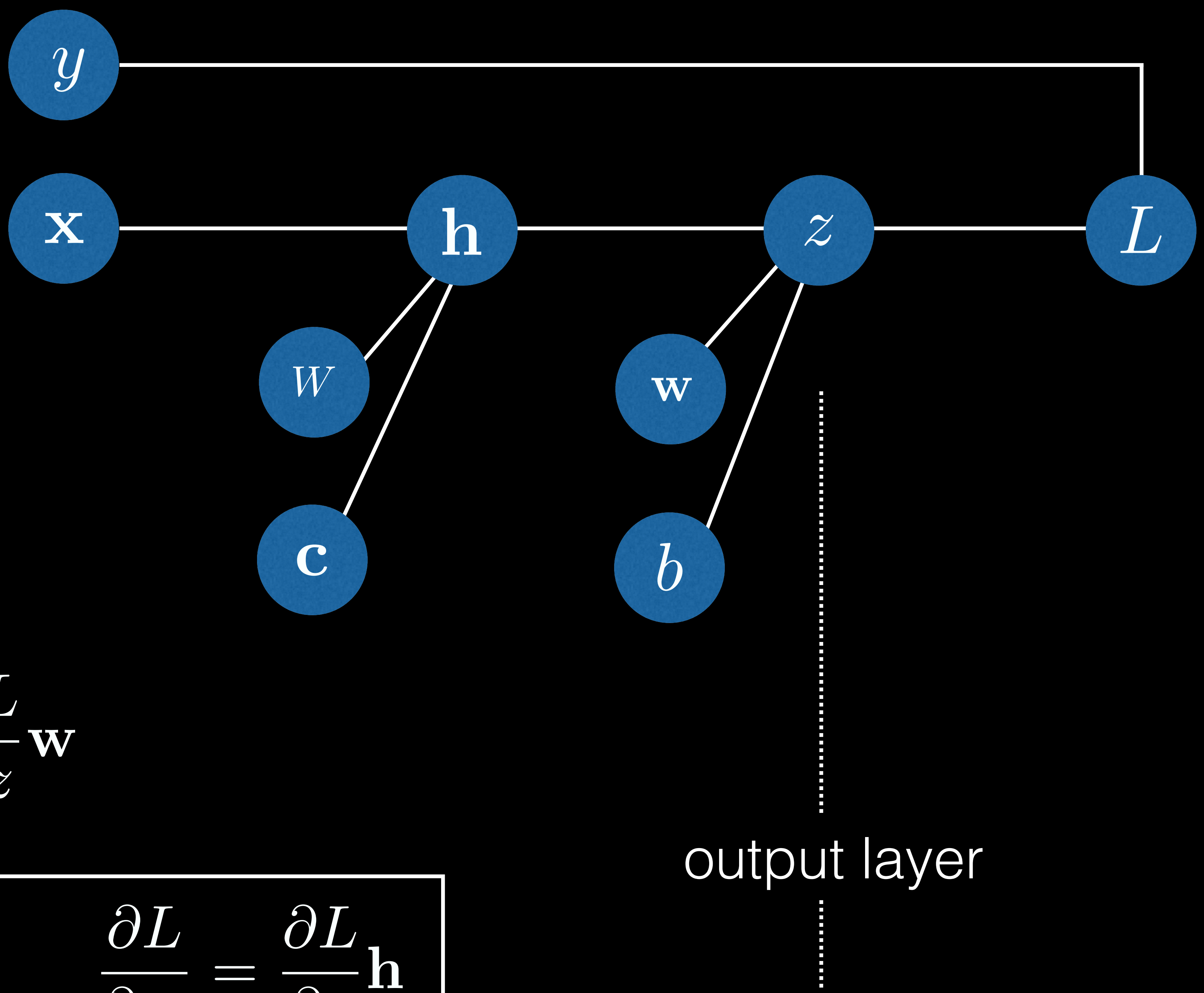
Simple MLP



$$\frac{\partial L}{\partial z} = \sigma((1-2y)z)(1-2y)$$

loss layer

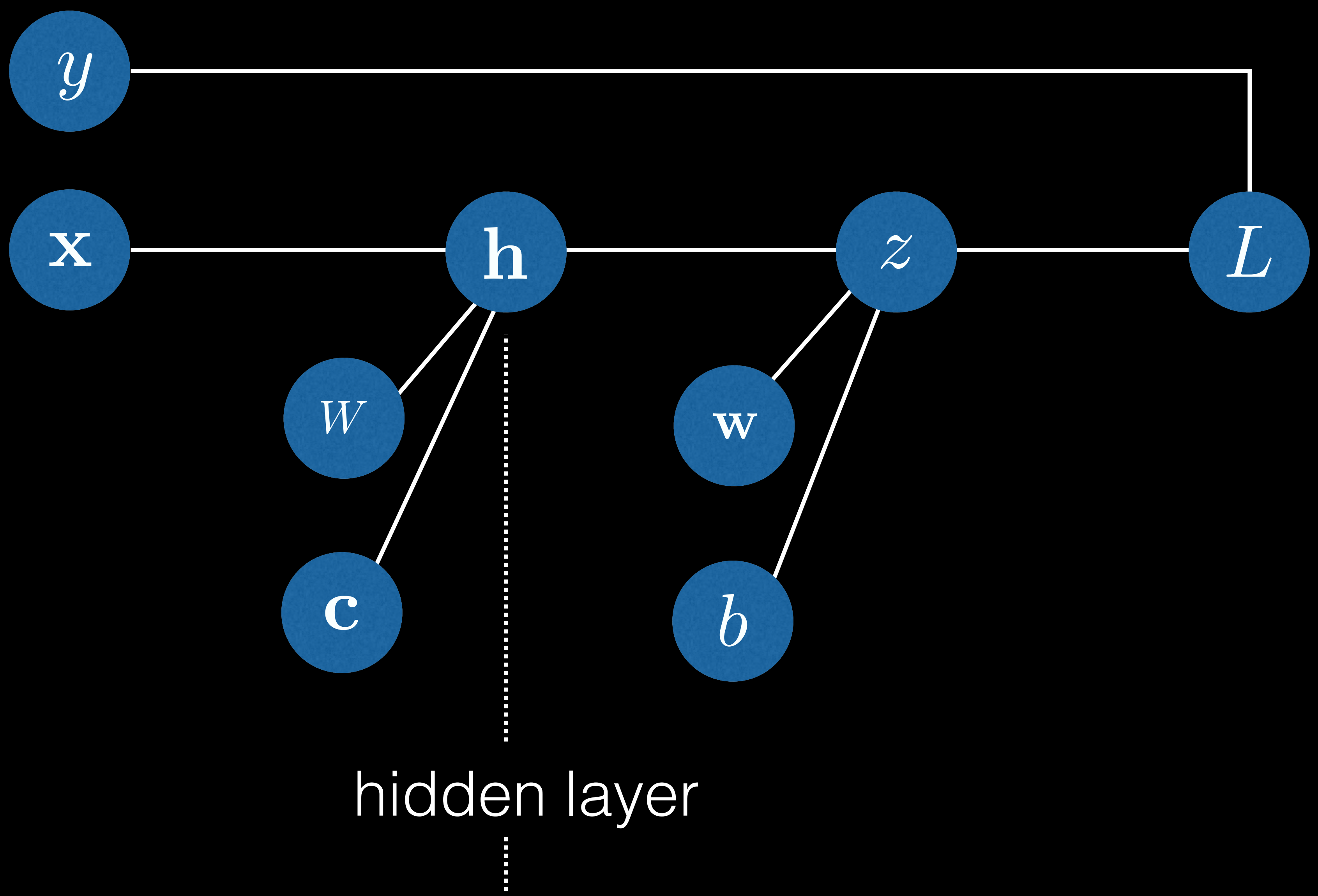
Simple MLP



$$\nabla_{\mathbf{h}} L = \frac{\partial L}{\partial z} \mathbf{w}$$

$$\boxed{\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z} \qquad \frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial z} \mathbf{h}}$$

Simple MLP



$$\nabla_{\mathbf{c}} L = \mathbf{f} \odot \nabla_{\mathbf{h}} L \qquad \nabla_W L = \mathbf{x} \left(\mathbf{f} \odot \nabla_{\mathbf{h}} L \right)^T$$