Introduction to Machine Learning (CS419M)

Lecture 11:

- Feedforward neural networks
- Backpropagation

Mar 6, 2020

What is deep learning?

"Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction."

"Representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level."

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436

History of (Deep) Neural Networks

- McCulloch-Pitts Neuron Model (1943)
- Perceptrons (1957)
- Backpropagation (1960)
- Backpropagation for neural networks (1986)
- Convolutional neural networks (1989)

⋮

- Deep learning for speech recognition (2009)
- AlexNet (2012)
- Generative Adversarial Networks (GANs) (2014)
- AlphaGo (2016)

Why the resurgence?

- McCulloch-Pitts Neuron Model (1943)
- Perceptrons (1957)
- Backpropagation (1960)
- Backpropagation for neural networks (1986)
- Convolutional neural networks (1989)

⋮

- Deep learning for speech recognition (2009)
- AlexNet (2012)
- Generative Adversarial Networks (GANs) (2014)
- AlphaGo (2016)

Vast amounts of data

+

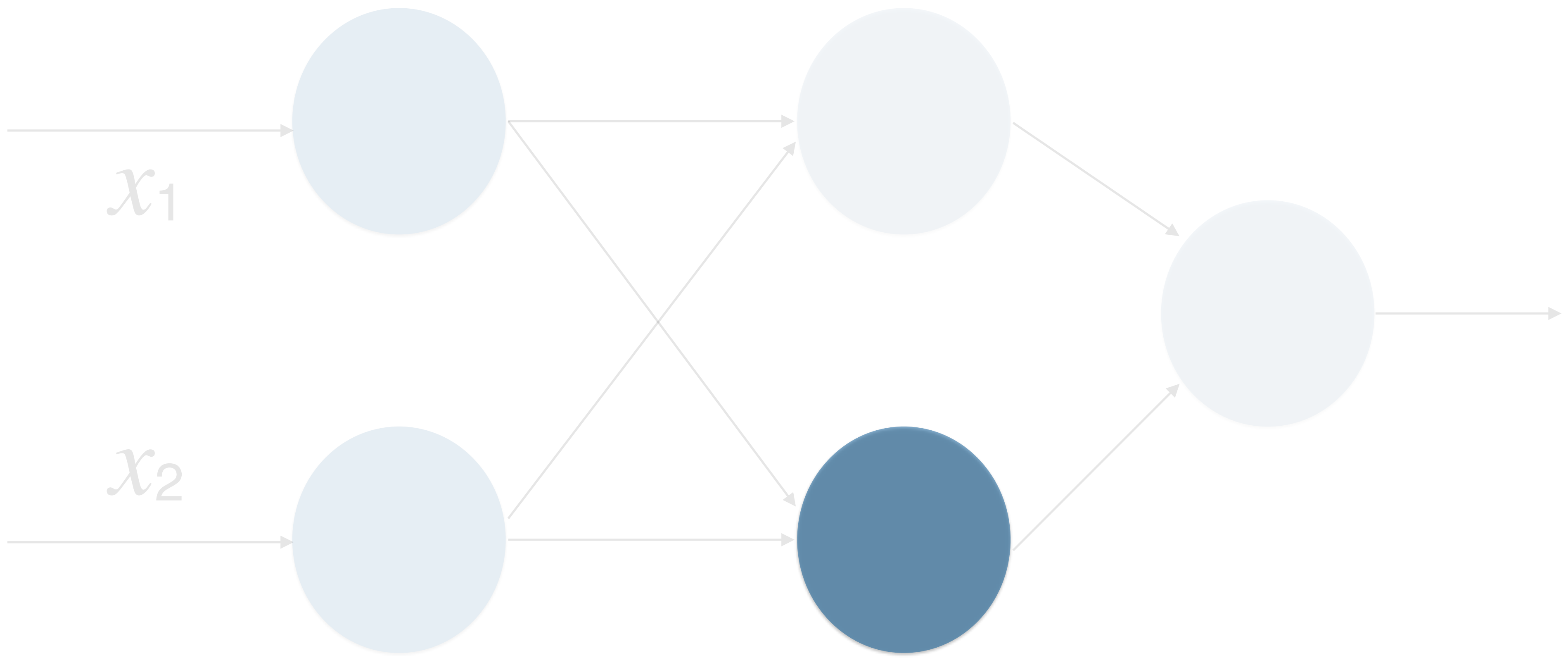
Specialized hardware,
Graphics Processing Units (GPUs)

+

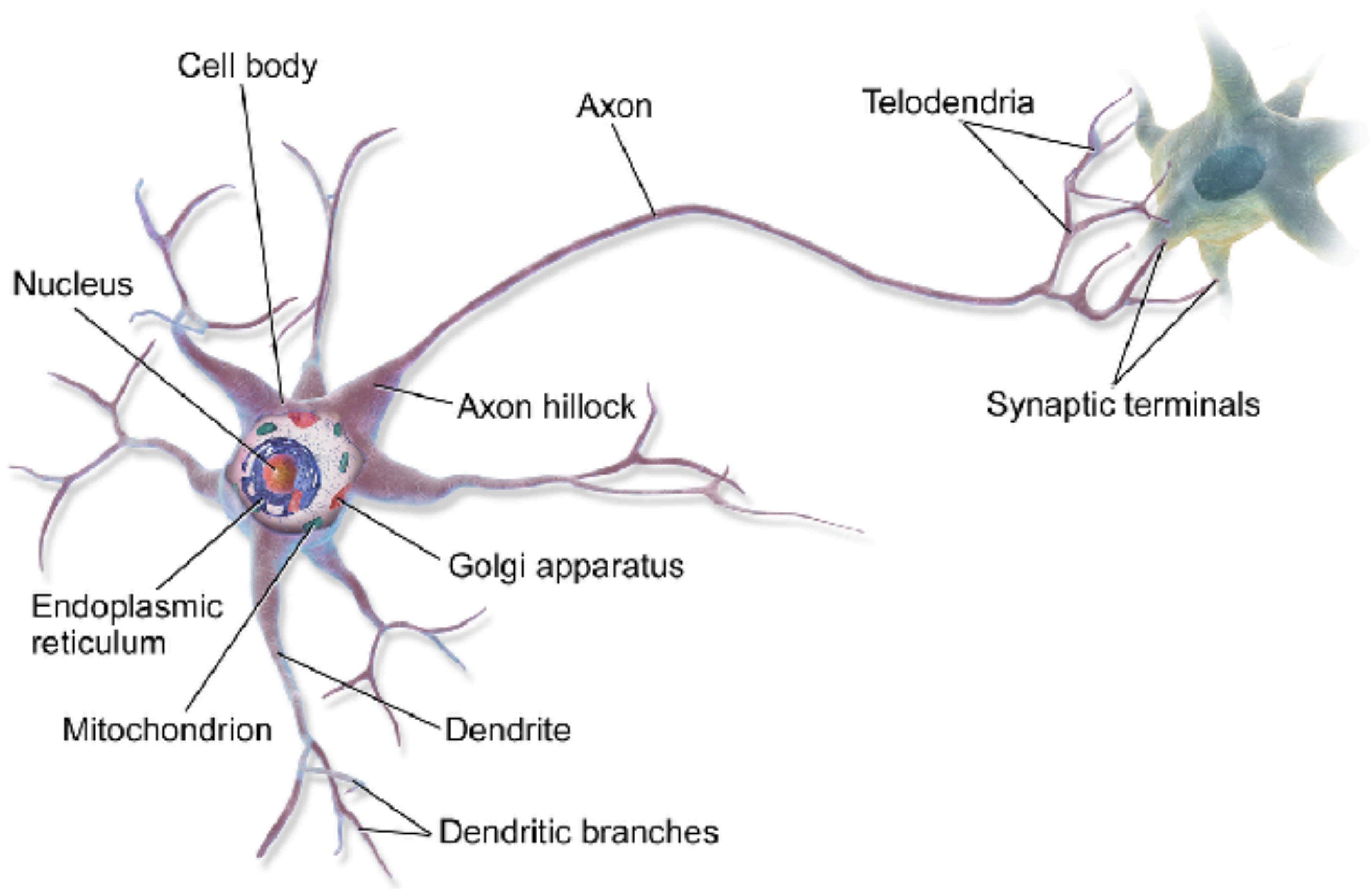
Improved optimization techniques
and new model variants/libraries/toolkits

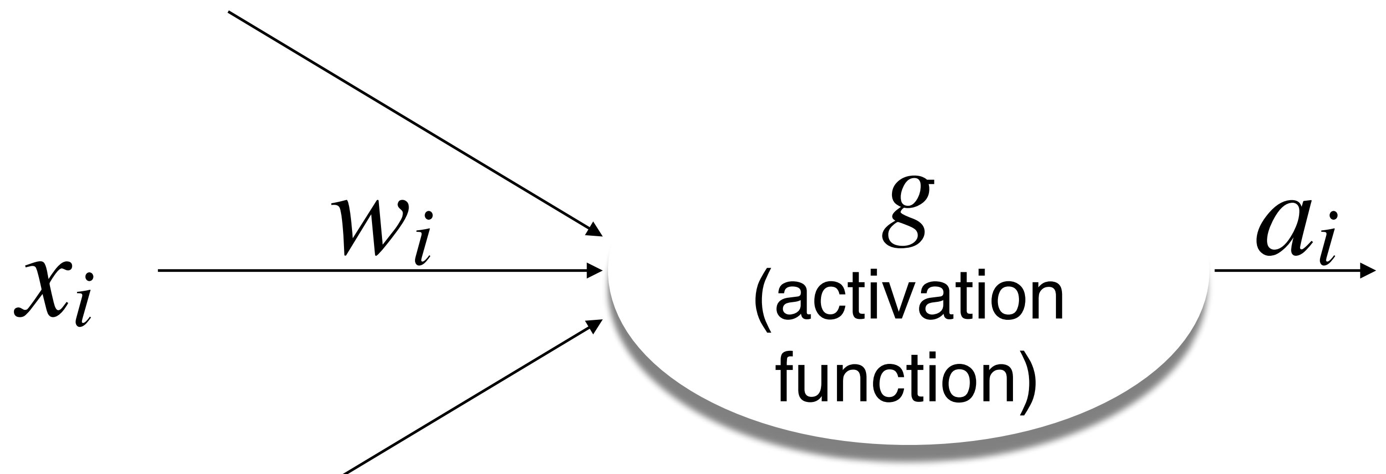
Feed-forward Neural Network

Single Neuron



Single neuron



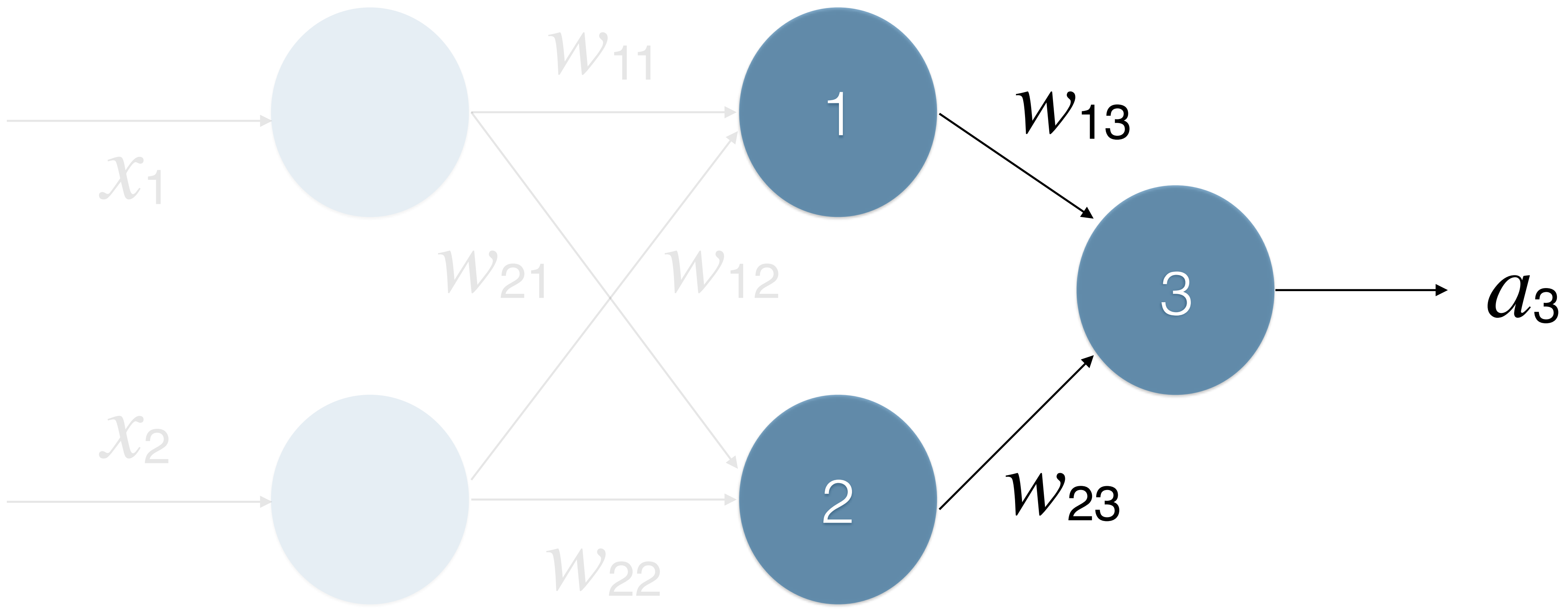


$$a_i = g(\sum_i w_i x_i)$$

Image from: https://upload.wikimedia.org/wikipedia/commons/1/10/Blausen_0657_MultipolarNeuron.png

Feed-forward Neural Network

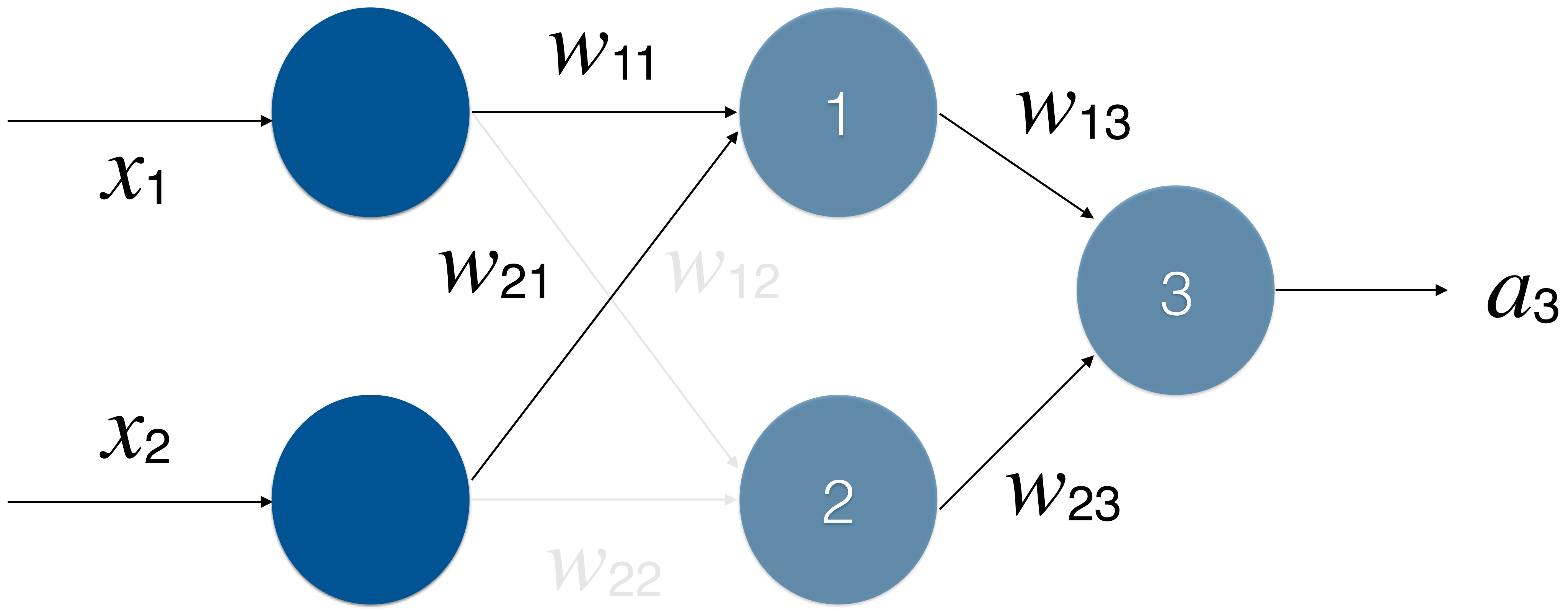
Parameterized Model



$$a_3 = g(w_{13} \cdot a_1 + w_{23} \cdot a_2 + b_3)$$

Feed-forward Neural Network

Parameterized Model



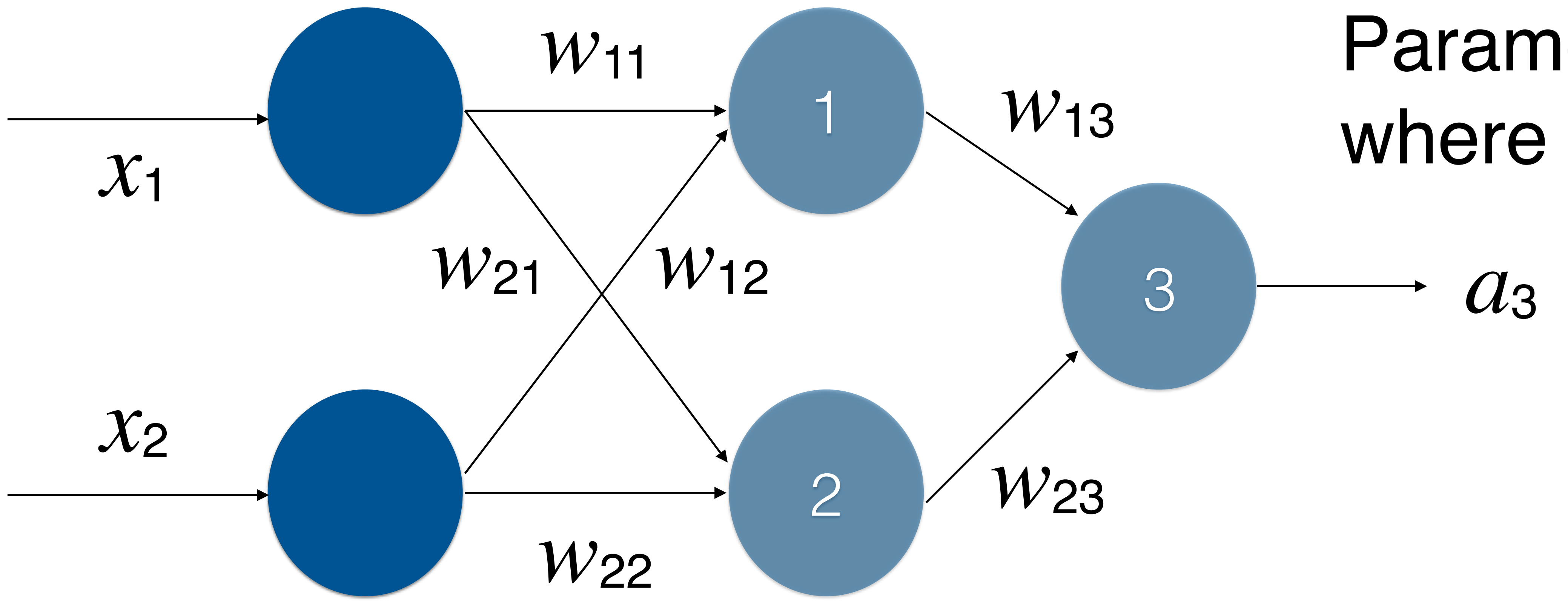
$$a_3 = g(w_{13} \cdot a_1 + w_{23} \cdot a_2 + b_3)$$

$$= g(w_{13} \cdot \big(g(w_{11} \cdot x_1 + w_{21} \cdot x_2 + b_1)\big)$$

$$+ \qquad \cdots$$

Feed-forward Neural Network

Parameterized Model



Parameters of the network: $\mathrm{NN}(\mathbf{x};\theta)$ where θ refers to all w_{ij} and b_j

$$
\begin{aligned}
a_3 &= g(w_{13}\cdot a_1 + w_{23}\cdot a_2 + b_3) \\
&= g(w_{13}\cdot \big(g(w_{11}\cdot x_1 + w_{21}\cdot x_2 + b_1)\big) \\
&\qquad + \ w_{23}\cdot \big(g(w_{12}\cdot x_1 + w_{22}\cdot x_2 + b_2)\big) + b_3)
\end{aligned}
$$

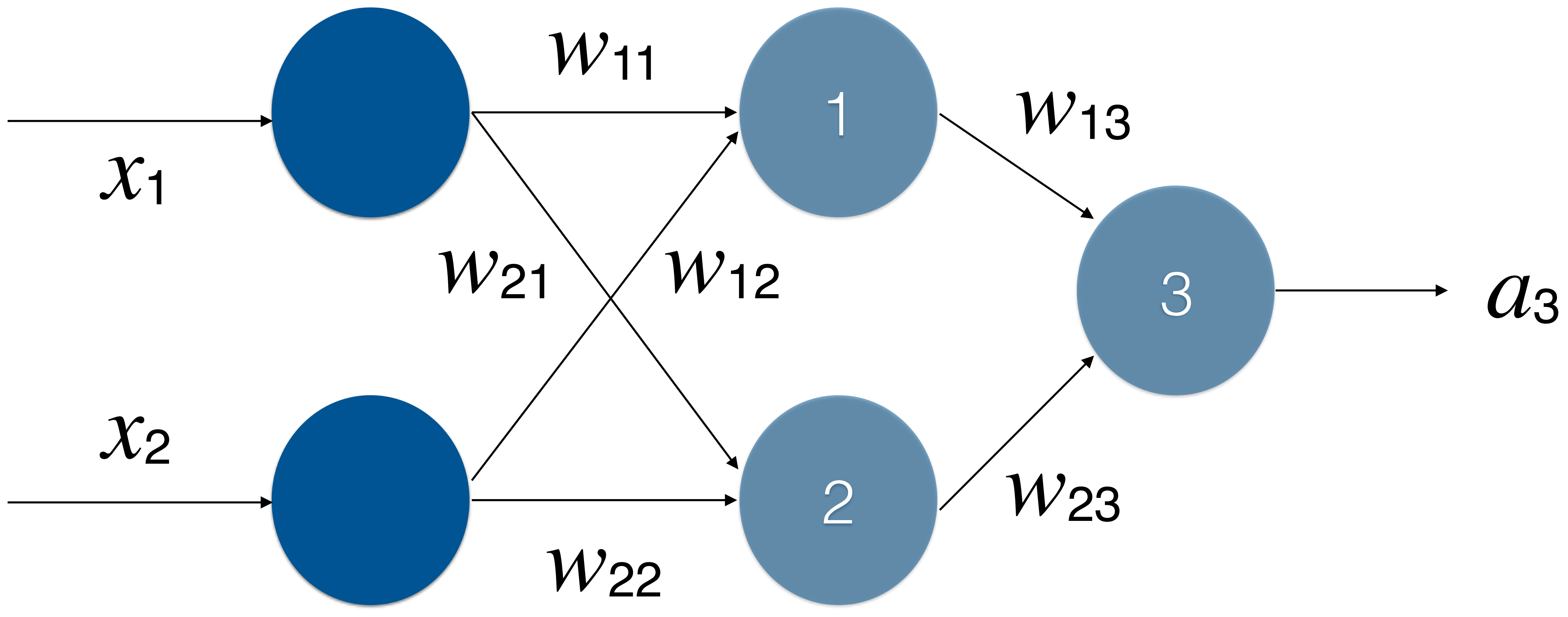
Compact matrix notation: Input $\mathbf{x} = [x_1, x_2]$ is written as a 2-dimensional vector and the layer above it is a 2-dimensional vector $\mathbf{h}$, a fully-connected layer is associated with:

$$\mathbf{h} = \mathbf{x}\mathbf{W} + \mathbf{b}$$

where w_{ij} in $\mathbf{W}$ is the weight of the connection between i^{th} neuron in the input row and j^{th} neuron in the first hidden layer and $\mathbf{b}$ is the bias vector.

Feed-forward Neural Network

Parameterized Model



$$\begin{aligned} a_3 &= g(w_{13} \cdot a_1 + w_{23} \cdot a_2 + b_3) \\ &= g(w_{13} \cdot \big(g(w_{11} \cdot x_1 + w_{21} \cdot x_2 + b_1)\big) \\ &\qquad + \; w_{23} \cdot \big(g(w_{12} \cdot x_1 + w_{22} \cdot x_2 + b_2)\big) + b_3) \end{aligned}$$

The simplest neural network is the perceptron:

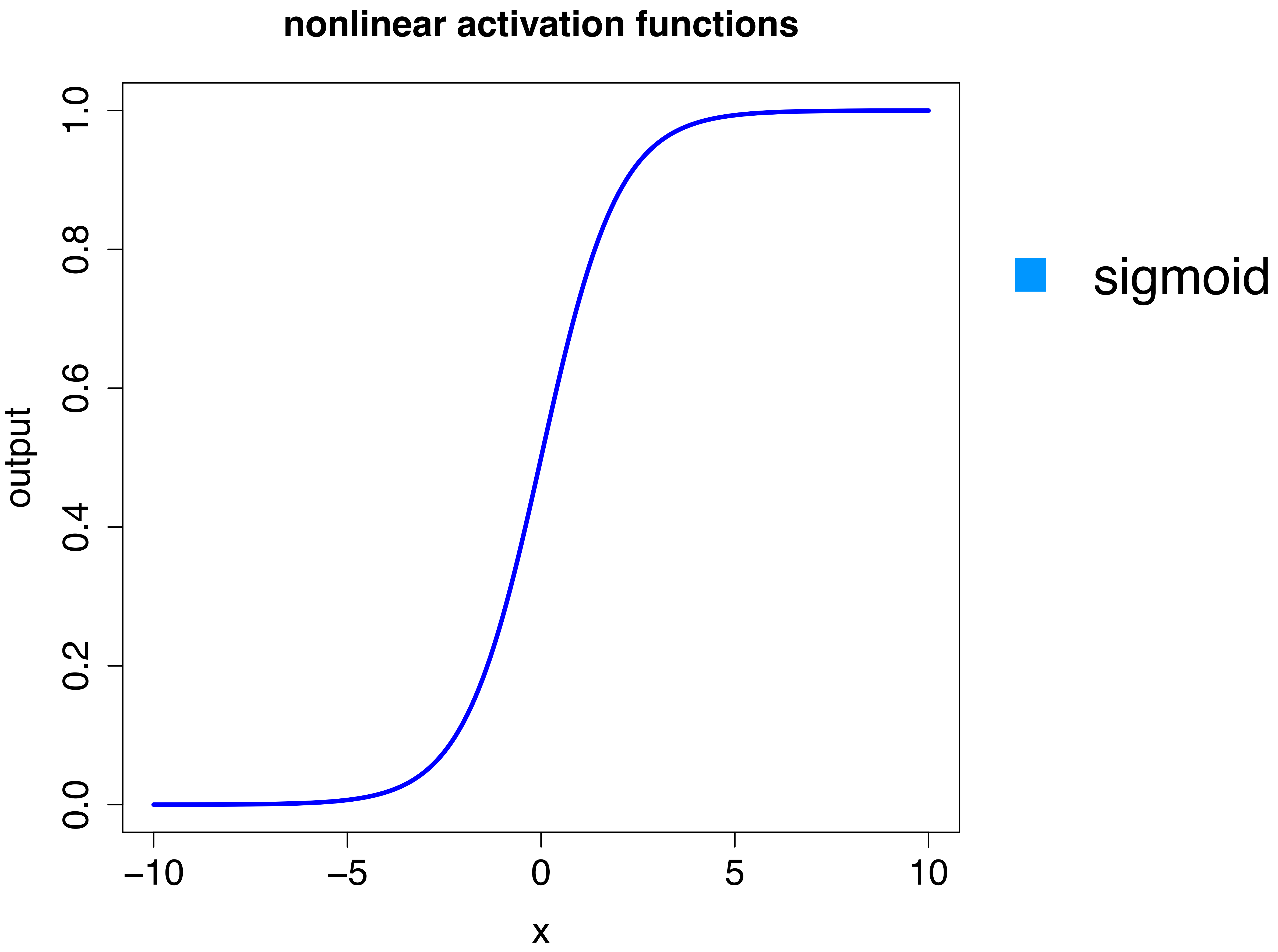
$$\text{Perceptron}(\mathbf{x}) = \mathbf{x}\mathbf{W} + \mathbf{b}$$

A 1-layer feedforward neural network (multi-layer perceptron) has the form:

$$\text{MLP}(\mathbf{x}) = g(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

Common Activation Functions (g)

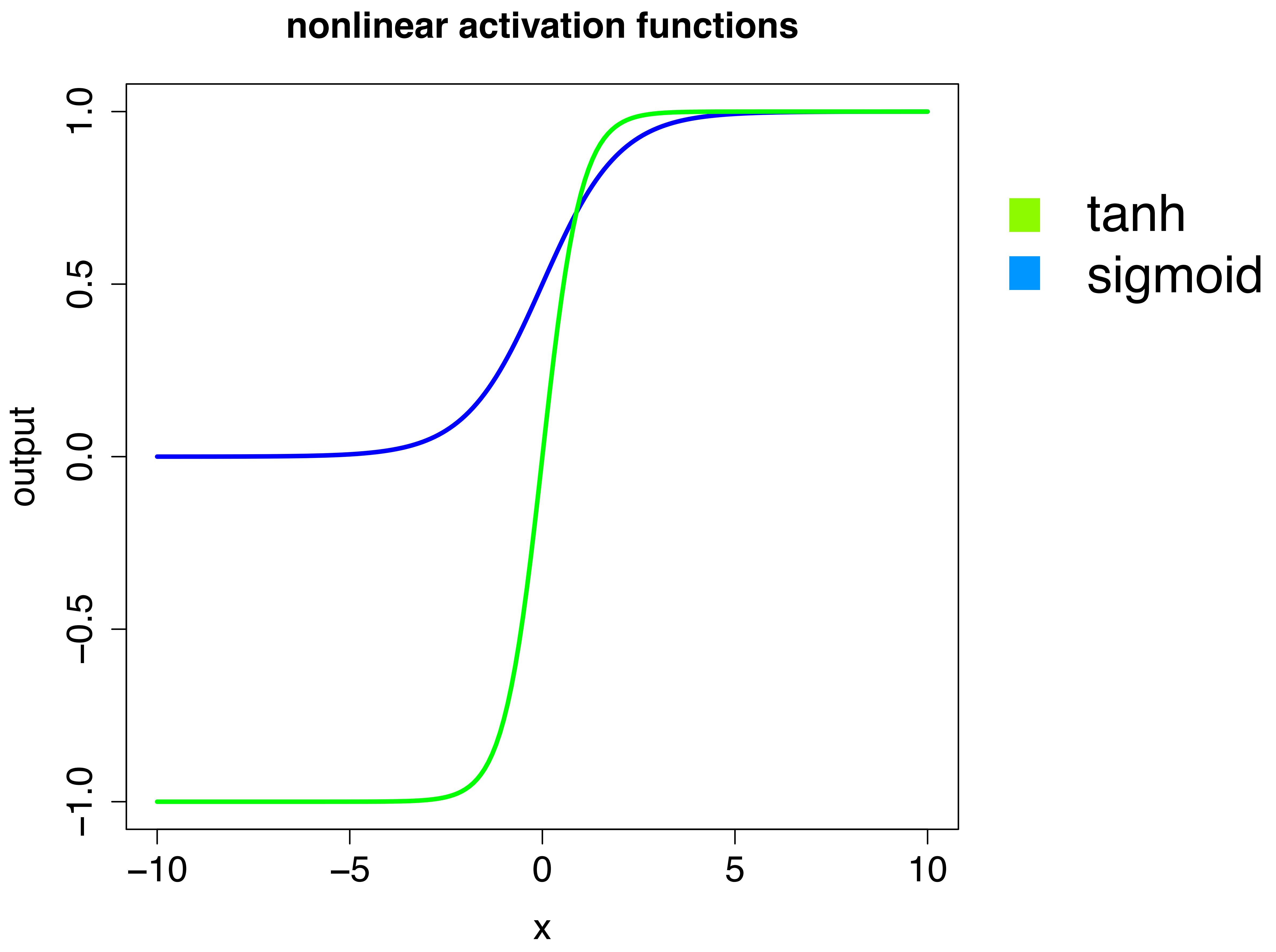
Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$



Common Activation Functions (g)

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

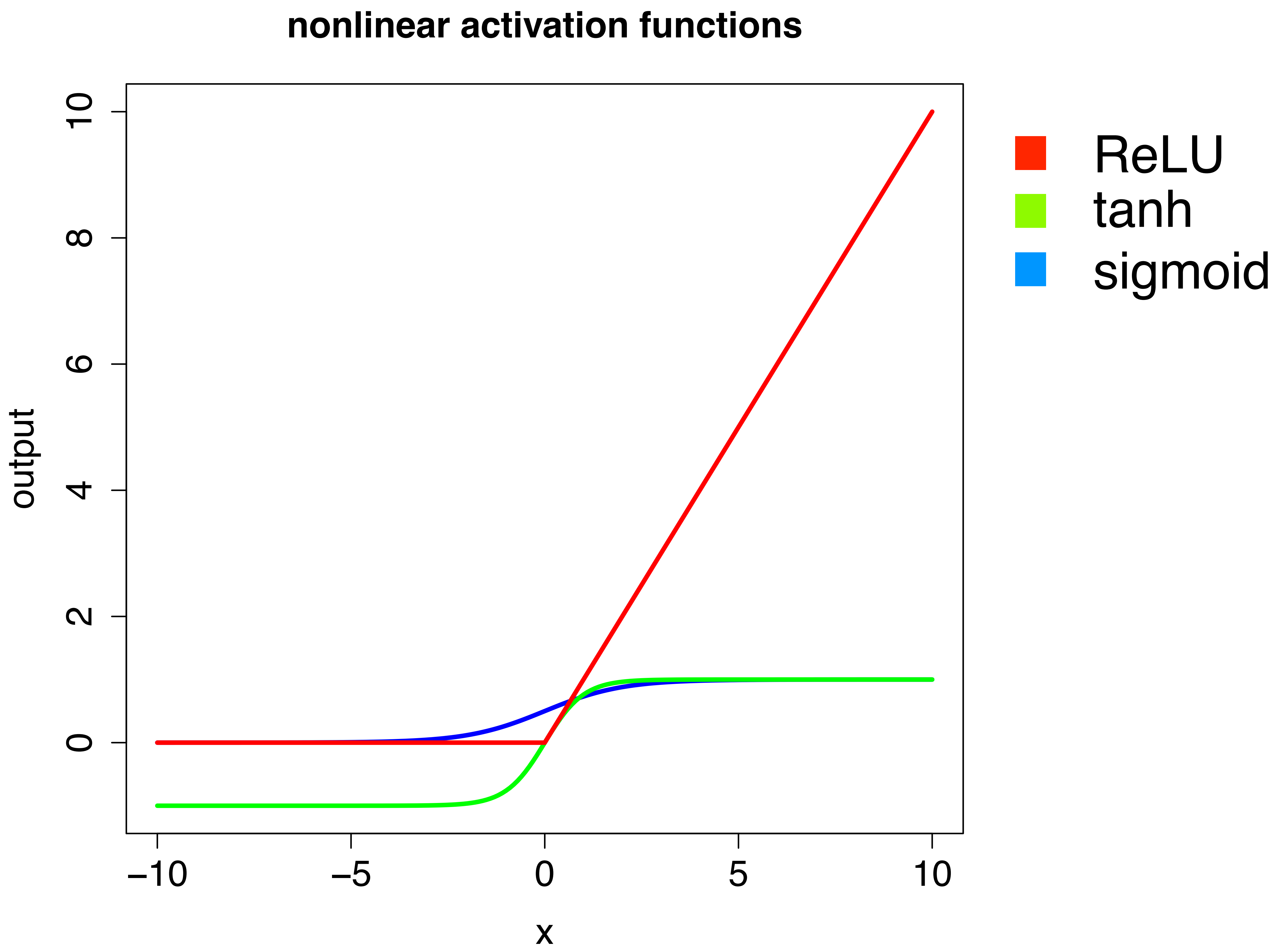


Common Activation Functions (g)

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

Rectified Linear Unit (ReLU): $\mathrm{RELU}(x) = \max(0, x)$



Training Neural Networks

Optimization Problem

- To train a neural network, define a loss function L(y,ỹ): a function of the true output y and the predicted output ỹ
- L(y,ỹ) assigns a non-negative numerical score to the neural network's output, ỹ
- The parameters of the network are set to minimise L over the training examples (i.e. a sum of losses over different training samples)
- L is typically minimised using a gradient-based method

Stochastic Gradient Descent (SGD)

SGD Algorithm

Inputs: $\mathrm{NN}(x; \theta)$, Training examples, $x_1 \dots x_n$; outputs, $y_1 \dots y_n$ and Loss function L

Randomly initialize θ
do until **stopping criterion**
    Pick a training example $\{x_i, y_i\}$
    Compute the loss $L(\mathrm{NN}(x_i; \theta), y_i)$
    Compute gradient of L, $\nabla_\theta L$ with respect to θ
    $\theta \leftarrow \theta - \eta \nabla_\theta L$
done

Weight Update Rule

Learning Rate

Return: θ

Mini-batch Gradient Descent (GD)

Mini-batch GD Algorithm

Inputs: $\mathrm{NN}(x; \theta)$, Training examples, $x_1 \ldots x_n$; outputs, $y_1 \ldots y_n$ and Loss function L

Randomly initialize θ
do until **stopping criterion**
    Randomly sample a batch of training examples $\{x_i, y_i\}_{i=1}^{b}$
        (where the batch size, b, is a hyperparameter)
    Compute gradient of L over the batch, $\nabla_\theta L$ with respect to θ
    $\theta \leftarrow \theta - \eta \nabla_\theta L$
done

Return: θ

Loss Function

Overall loss function, $J(\theta)$, measures the total loss over the entire training set:

$$J(\theta) = \sum_{i=1}^{N} L(\mathrm{NN}(\mathbf{x}_i; \theta), y_i)$$

Cross-entropy loss is one of the most popular classification-based loss functions. Assuming $\mathrm{NN}(\mathbf{x}_i; \theta)$ returns a probability, binary cross-entropy can be defined as:

$$J(\theta) = -\sum_{i=1}^{N} y_i \log\left(\mathrm{NN}(\mathbf{x}_i; \theta)\right) + (1 - y_i)\log\left(1 - \mathrm{NN}(\mathbf{x}_i; \theta)\right)$$

Training a Neural Network

Define the Loss function to be minimised as a node L

Goal: Learn weights for the neural network which minimise L

Gradient Descent: Find $\partial L/\partial w$ for every weight w, and update it as $w \leftarrow w - \eta\ \partial L/\partial w$

How do we efficiently compute $\partial L/\partial w$ for all w?

Will compute $\partial L/\partial u$ for every node u in the network!

$\partial L/\partial w = \partial L/\partial u \cdot \partial u/\partial w$ where u is the node which uses w

Training a Neural Network

New goal: compute $\partial L/\partial u$ for every node u in the network

Simple algorithm: Backpropagation

Key fact: Chain rule of differentiation

If L can be written as a function of variables $v_1,\ldots, v_n$, which in turn depend (partially) on another variable u, then

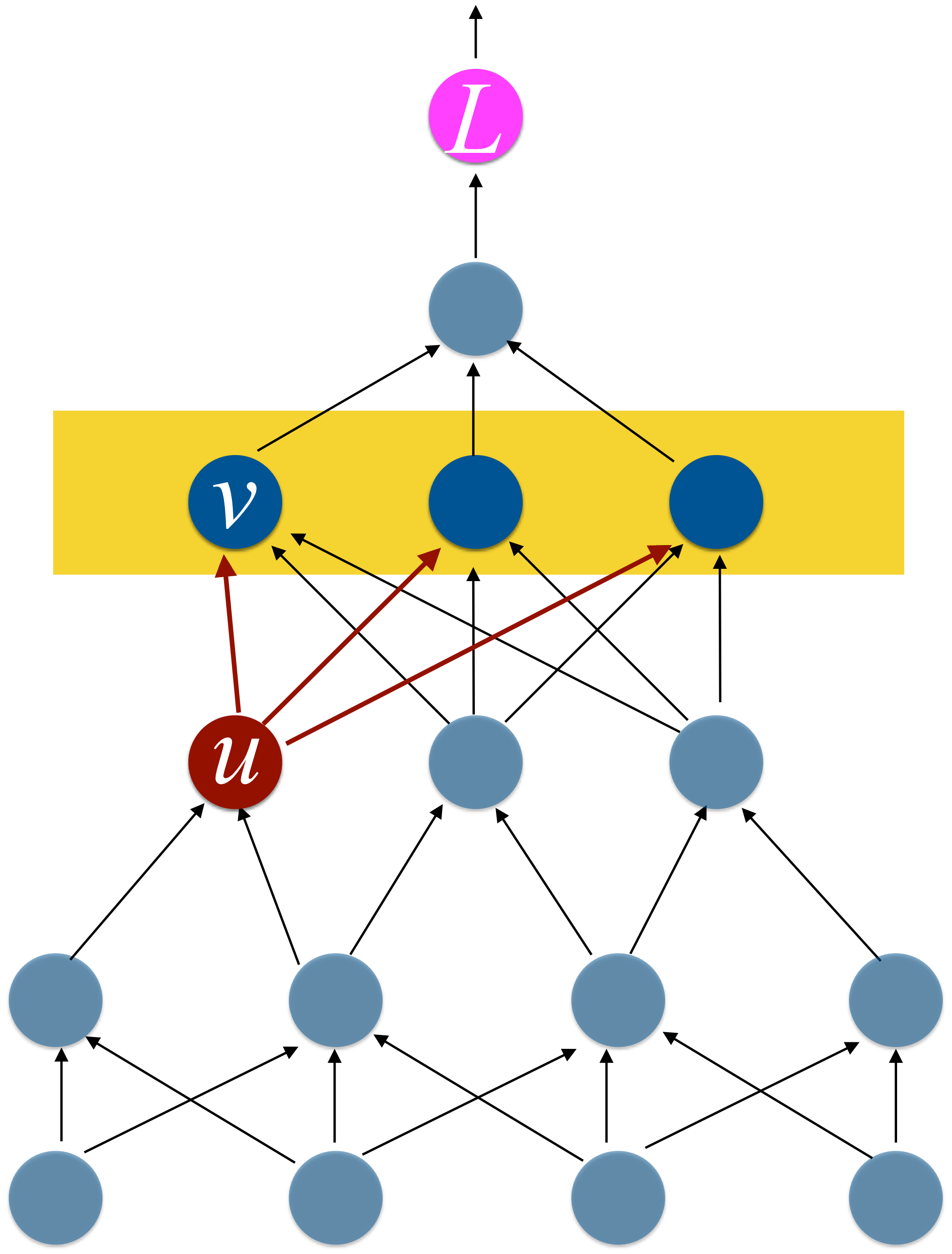
$$\partial L/\partial u = \Sigma_i \, \partial L/\partial v_i \cdot \partial v_i/\partial u$$

Backpropagation

If L can be written as a function of variables $v_1, \ldots, v_n$, which in turn depend (partially) on another variable u, then

$$\partial L/\partial u = \Sigma_i \, \partial L/\partial v_i \cdot \partial v_i/\partial u$$

Consider $v_1, \ldots, v_n$ as the layer above u, $\Gamma(u)$



Then, the chain rule gives

$$\partial L/\partial u = \Sigma_{v \in \Gamma(u)} \, \partial L/\partial v \cdot \partial v/\partial u$$

Backpropagation

$$\partial L/\partial u = \Sigma_{v \in \Gamma(u)}\ \partial L/\partial v \cdot \partial v/\partial u$$

<u>Backpropagation</u>

Base case: $\partial L/\partial L = 1$

For each u (top to bottom):

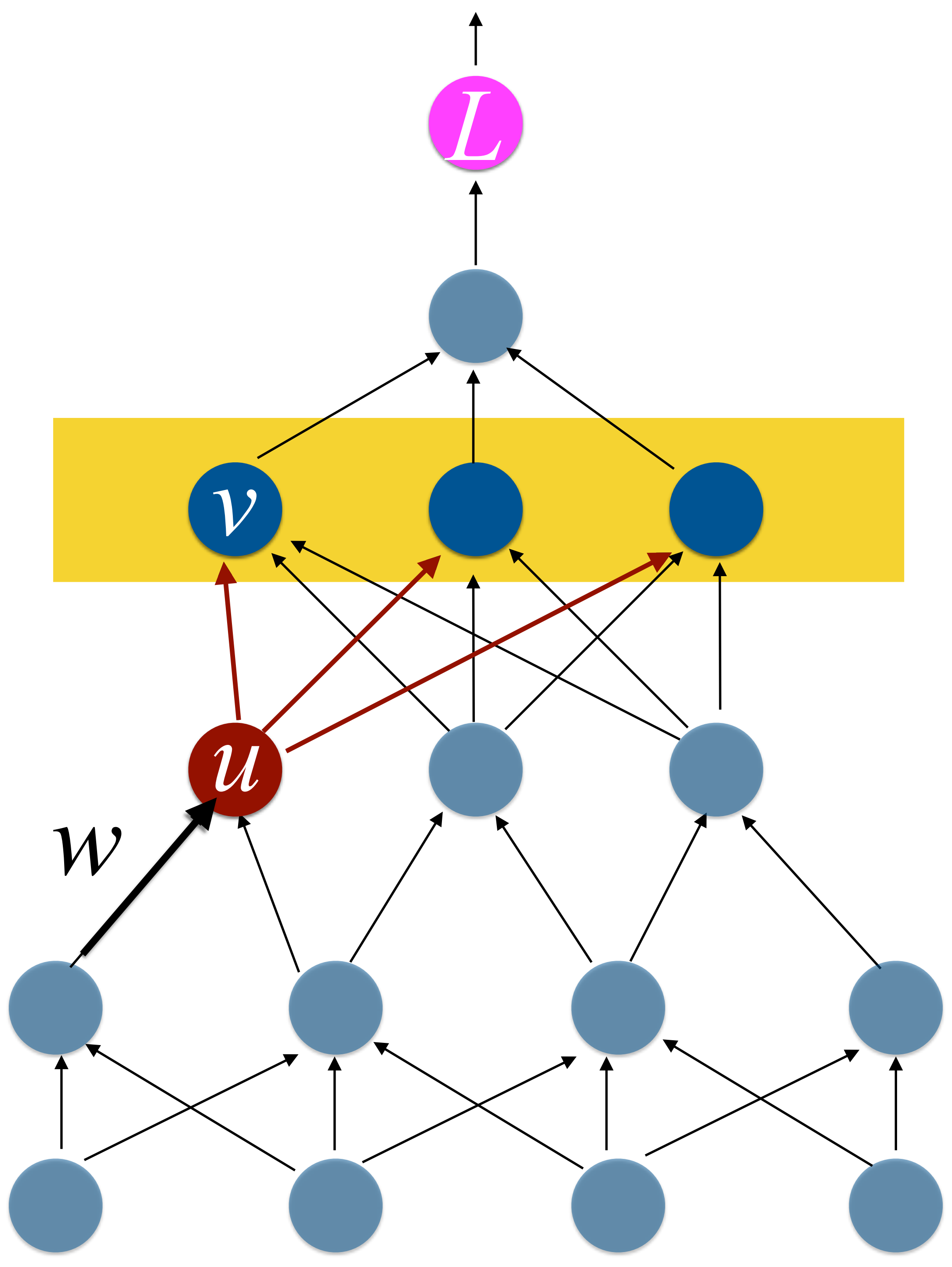
For each $v \in \Gamma(u)$:

Inductively, have computed $\partial L/\partial v$

Directly compute $\partial v/\partial u$

Compute $\partial L/\partial u$

Compute $\partial L/\partial w$
where $\partial L/\partial w = \partial L/\partial u \cdot \partial u/\partial w$



Where values computed in the forward pass may be needed

<u>Forward Pass</u>

First, in a forward pass, compute values of all nodes given an input
(The values of each node will be needed during backprop)

In-class quiz 3

Our goal is to obtain a neuron N which takes two inputs $x_1, x_2 \in \{0, 1\}$ and outputs a Boolean operator applied to the two inputs, interpreting 0 as false and 1 as true. That is, we want $\mathrm{B}(N(x_1, x_2)) = F(\mathrm{B}(x_1), \mathrm{B}(x_2))$, where $\mathrm{B}(0) = \mathsf{false}$ and $\mathrm{B}(1) = \mathsf{true}$, and F is some Boolean operator.

In the following problems, the neuron is defined as $N(x_1, x_2) = \tau(w_0 + w_1 x_1 + w_2 x_2)$ where $w_0, w_1, w_2 \in \mathbb{R}$ are real-valued weights, and $\tau : \mathbb{R} \to \{0, 1\}$ is defined so that $\tau(x) = 1$ iff $x \geq 0$.

The Boolean operator NAND is defined as follows: $\mathrm{NAND}(x, y) = \mathsf{false}$ iff $x = y = \mathsf{true}$. (For Boolean logic circuits, the NAND gate is a universal gate.)

Given $w_0 = 1$ and $w_1 = -0.3$, give the set of all possible values of w_2 such that $\mathrm{B}(N(x_1, x_2)) = \mathrm{NAND}(\mathrm{B}(x_1), \mathrm{B}(x_2))$.

You are given a function, $f(\mathbf{x}, \mathbf{w}) = \sigma(\sigma(x_1 w_1) w_2 + x_2)$ where $\sigma(x) = \frac{1}{1+exp(-x)}$, which takes a two-dimensional input $\mathbf{x} = (x_1, x_2)$ and has two parameters $\mathbf{w} = (w_1, w_2)$. The parameters are both initialized to 0. Assume we are given a training instance $x_1 = 11, x_2 = 5, y = 15$. What is the value of $\frac{\partial f}{\partial w_2}$?