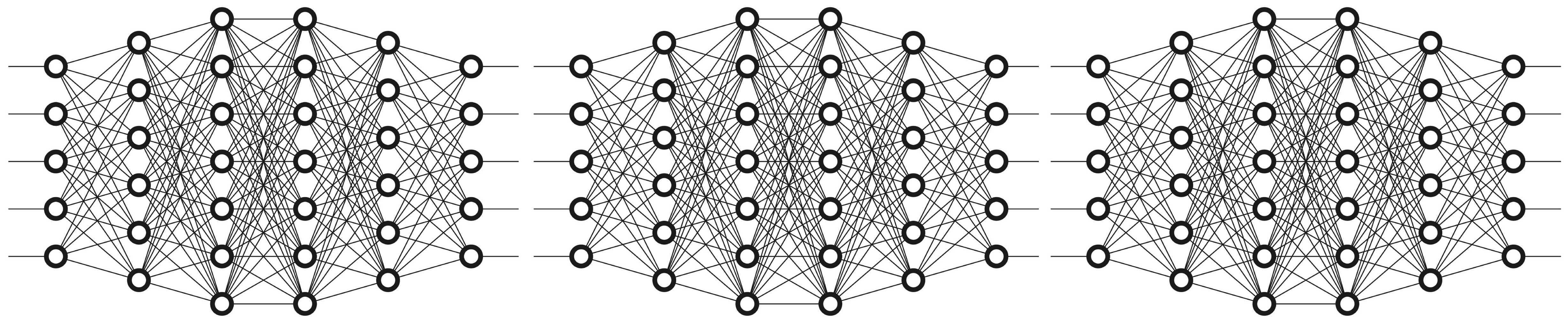
THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision



Lecture 20: Neural Networks Intro

Menu for Today

Topics:

— Introduction to neural networks

Redings:

— **Today's** Lecture: Forsyth & Ponce (2nd ed.) 16.1.3, 16.1.4, 16.1.9

— **Next** Lecture: Forsyth & Ponce (2nd ed.) 17.1–17.2

Reminders:

— **Assignment 2** & **3** graded, grades posted (let us know if there are issues)

— **Assignment 5** is due on **today**

— **Assignment 6** will be out **tonight** or **tomorrow**

Assignment 5

Computing the error for optical flow:

Assuming you are computing optical flow of Image 1 -> Image 2
(note, this is not the same as Image 2 -> Image 1)

1. Warp the Image 1 using estimated optical flow.
2. Subtract warped Image 1 from Image 2 pixel-by-pixel, then compute L2 norm of the difference per pixel. Result is a W x H image of L2 norms.
3. Average the L2 norms over all pixels.

We will **not** grade based on the error itself …

Warning:

Our intro to **Neural Networks** will be light weight …

… if you want to know more, take my **CPSC 532S** next year or **CPEN 455**

Recall: Linear Classifier

Defines a score function:

image features

$$f(\mathbf{x}_i, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x}_i + \mathbf{b}$$

weights (parameters)

bias vector

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Recall: Linear Classifier

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

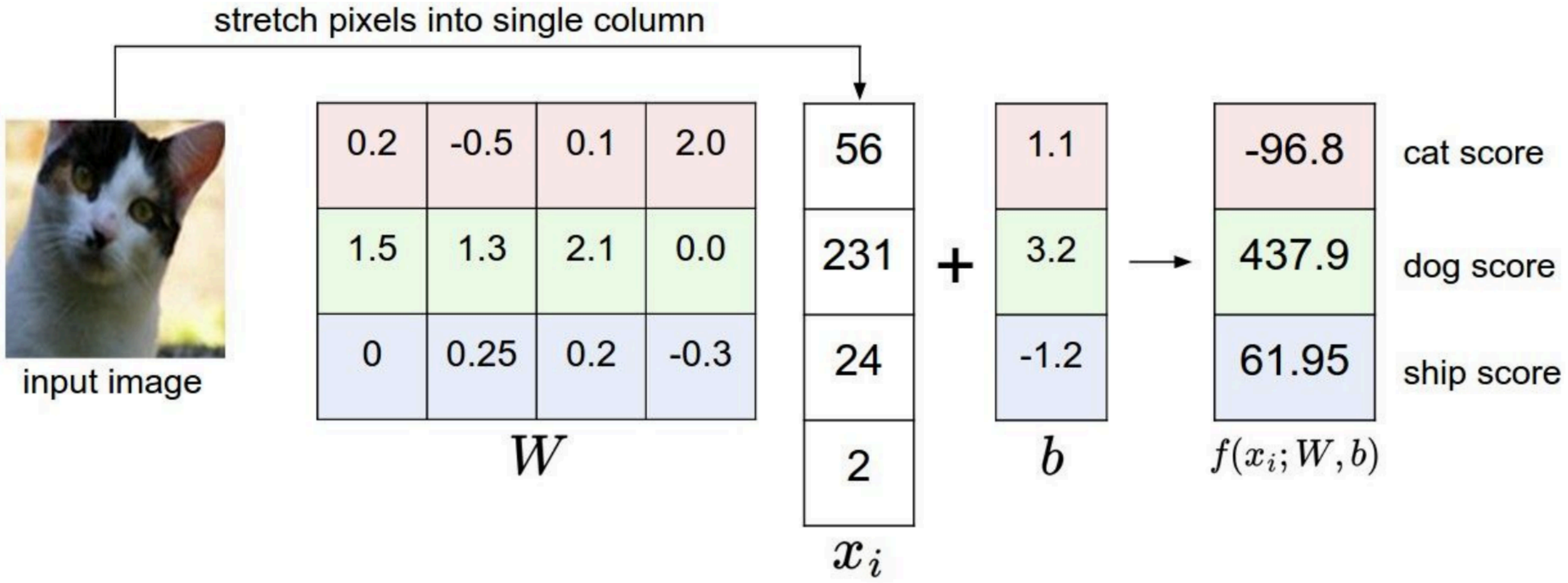
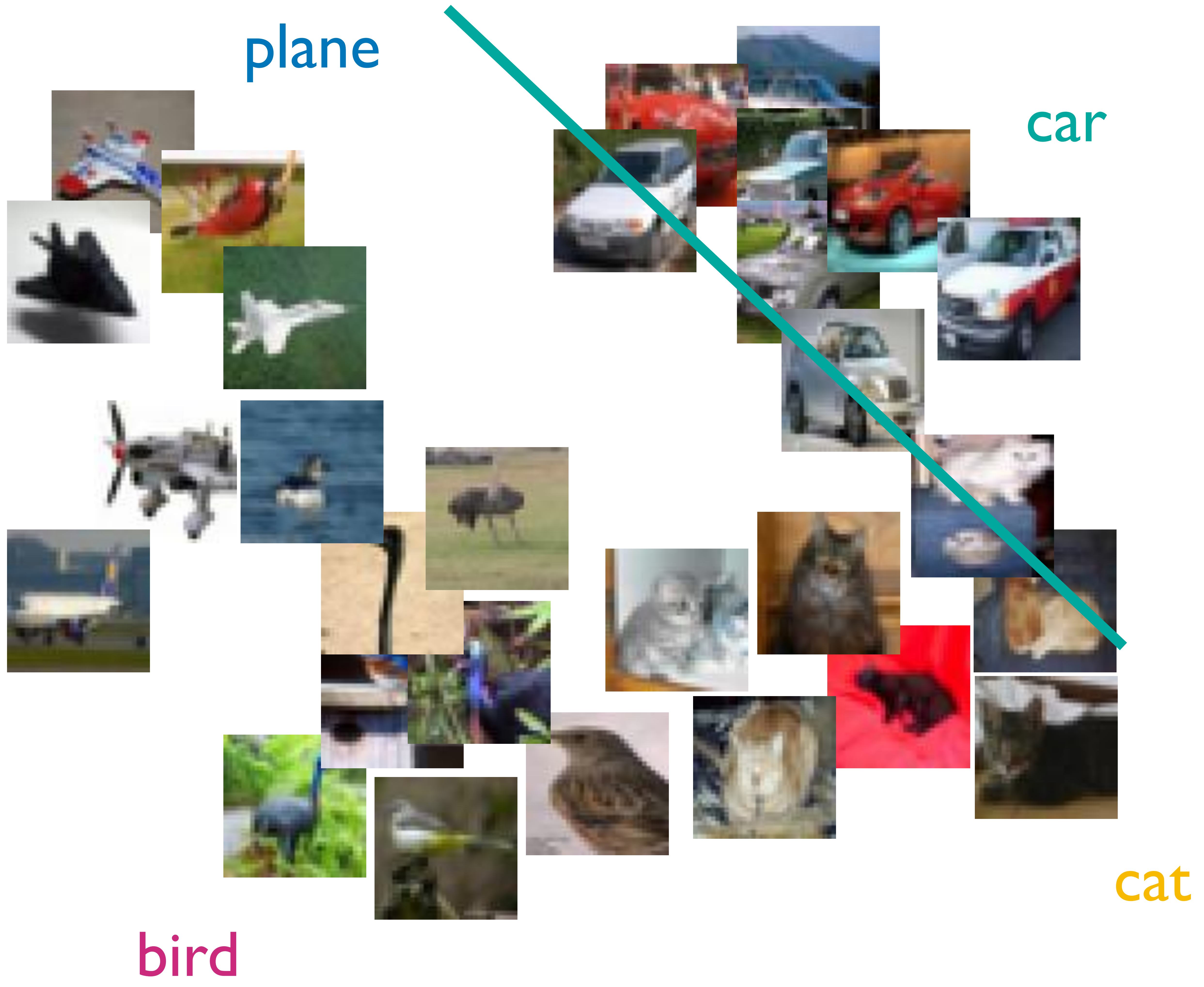


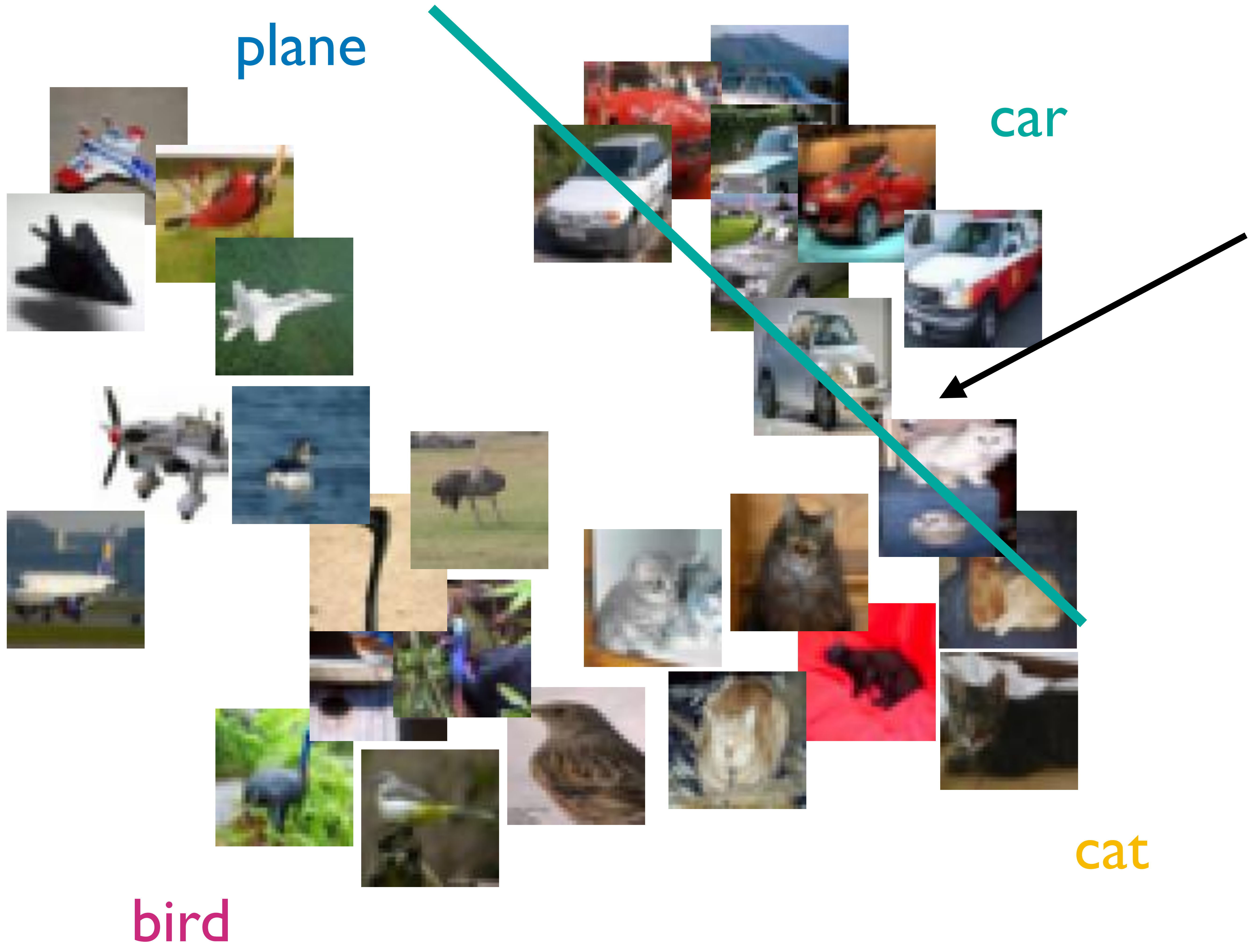
Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

1-vs-All **Linear SVM**

1-vs-All **Linear SVM**



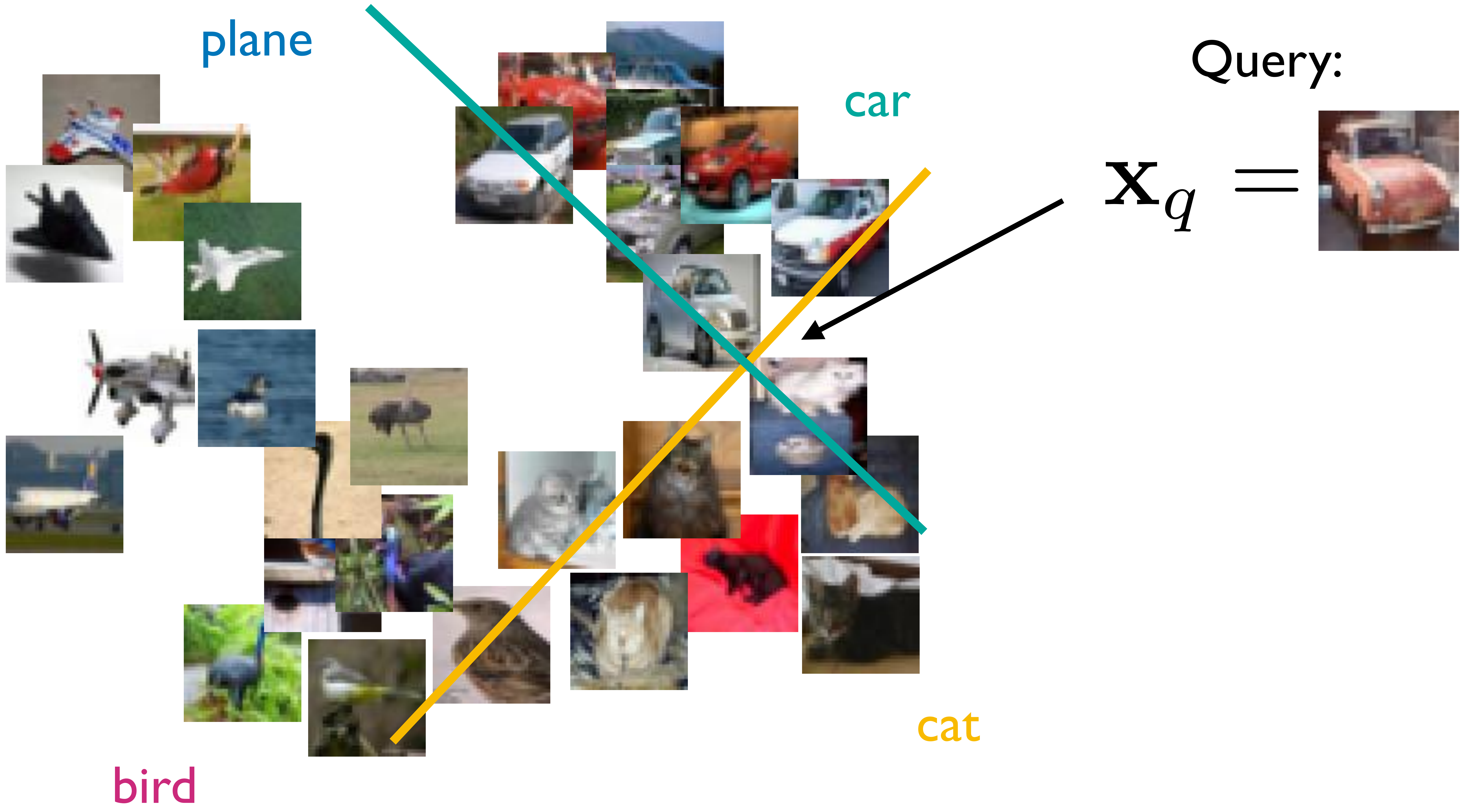
1-vs-All **Linear SVM**



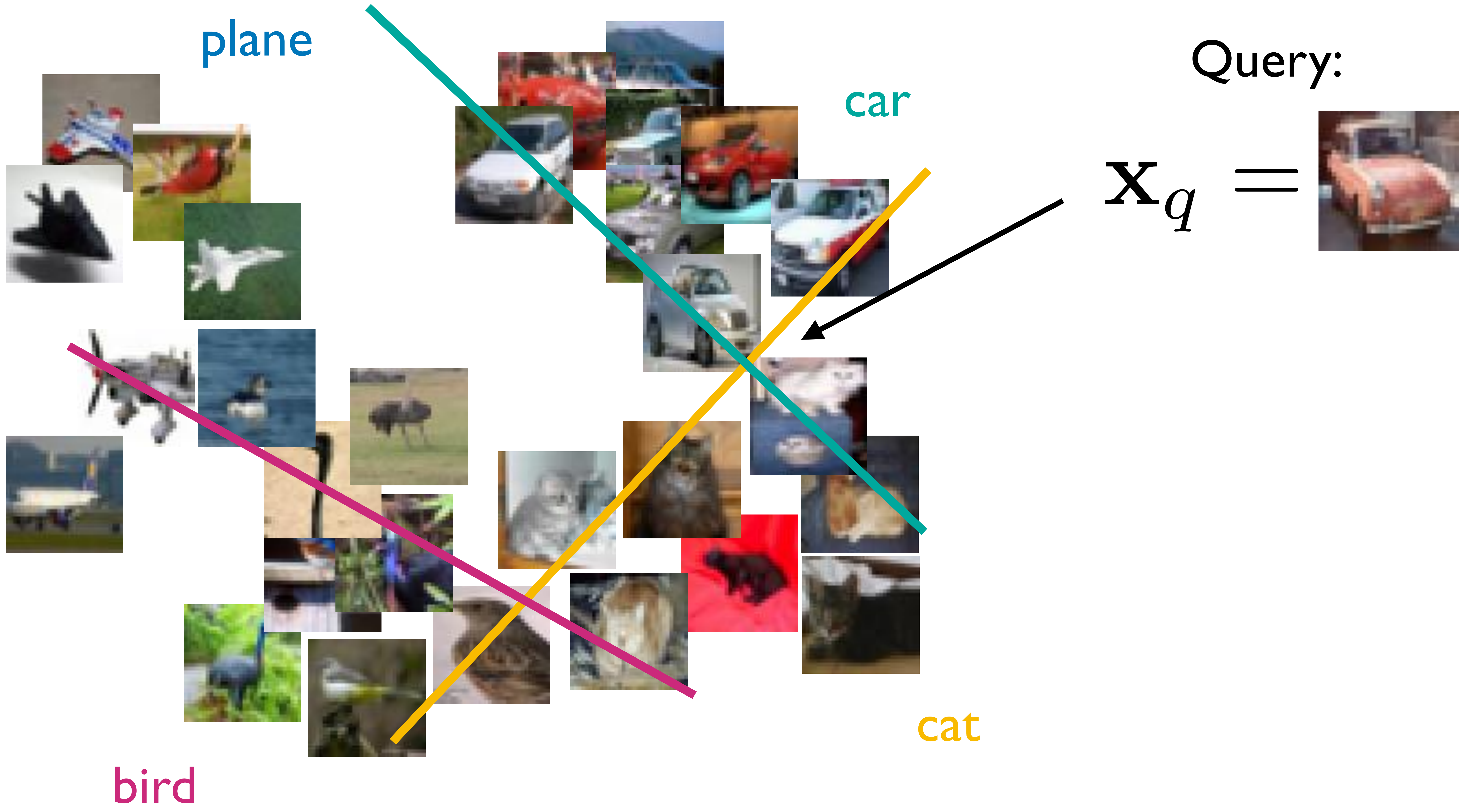
Query:

$$\mathbf{x}_q =$$

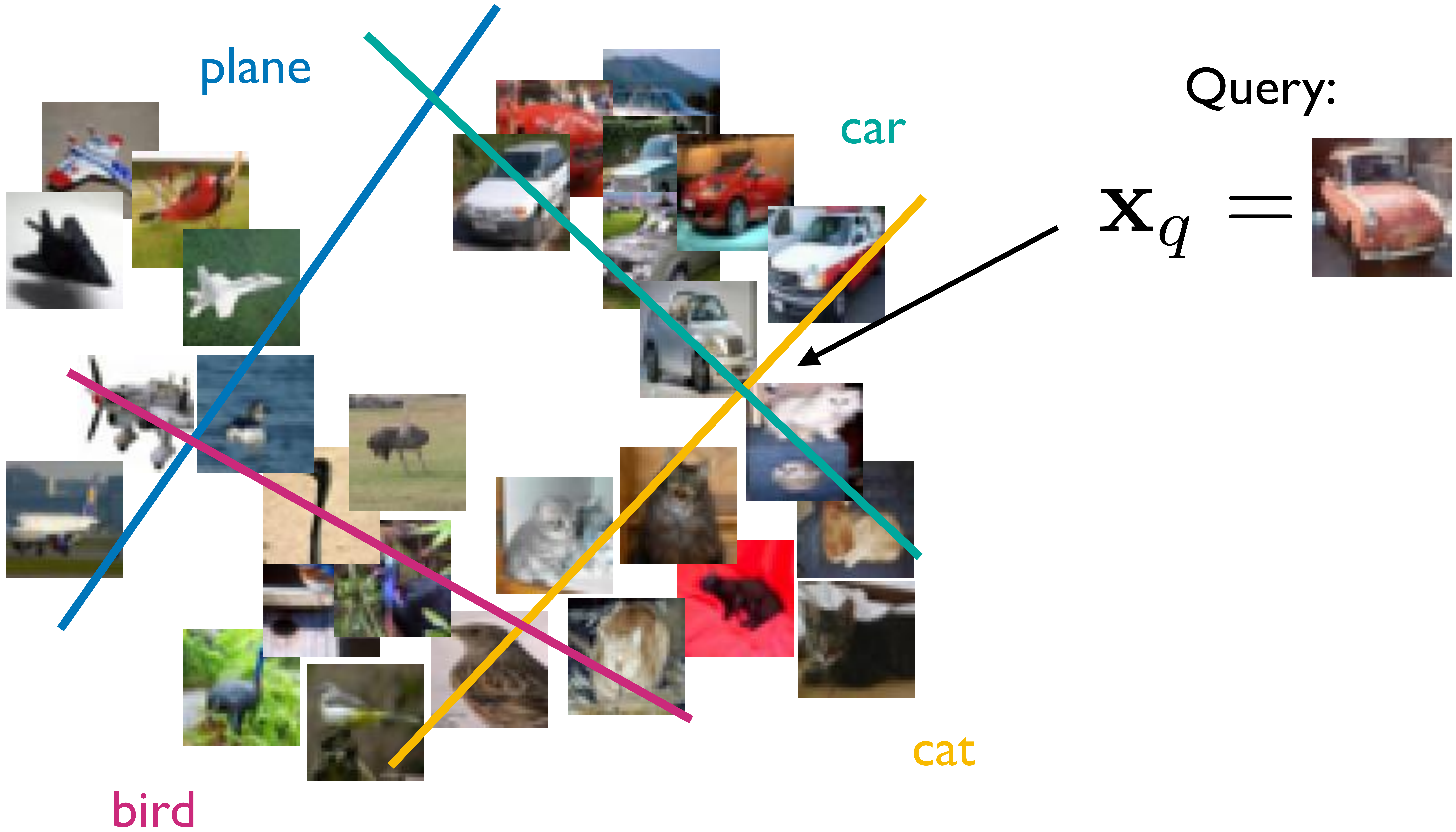
1-vs-All **Linear SVM**



1-vs-All **Linear SVM**



1-vs-All **Linear SVM**



One-hot Regression

An alternative solution is to regress to one-hot targets = 1 vs all classifiers

$$\left[\quad \mathbf{W}^T \quad \right] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \dots \end{bmatrix}$$

class 2 = 'automobile'

$$\left[\quad \mathbf{W}^T \quad \right] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ \dots \end{bmatrix}$$

class 4 = 'cat'

One-hot Regression

An alternative solution is to regress to one-hot targets = 1 vs all classifiers

$$\left[\quad \mathbf{W}^T \quad\right]\begin{bmatrix}x_1\\x_2\\x_3\\x_4\\\cdots\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\\\cdots\end{bmatrix}$$

← **class 2 = 'automobile'**

Why one-hot?

$$\left[\quad \mathbf{W}^T \quad\right]\begin{bmatrix}x_1\\x_2\\x_3\\x_4\\\cdots\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\\\cdots\end{bmatrix}$$

← **class 4 = 'cat'**

One-hot Regression

Transpose

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots \\ x_{21} & x_{22} & x_{23} & \dots \\ x_{31} & x_{32} & x_{33} & \dots \\ & \dots & & \end{bmatrix} \begin{bmatrix} \\ \quad \mathbf{W} \quad \\ \\ \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ & .. & .. & & \end{bmatrix}$$

auto
cat

$$\mathbf{XW} = \mathbf{T}$$

One-hot Regression

Transpose

32 x 32 x 3 = 3072

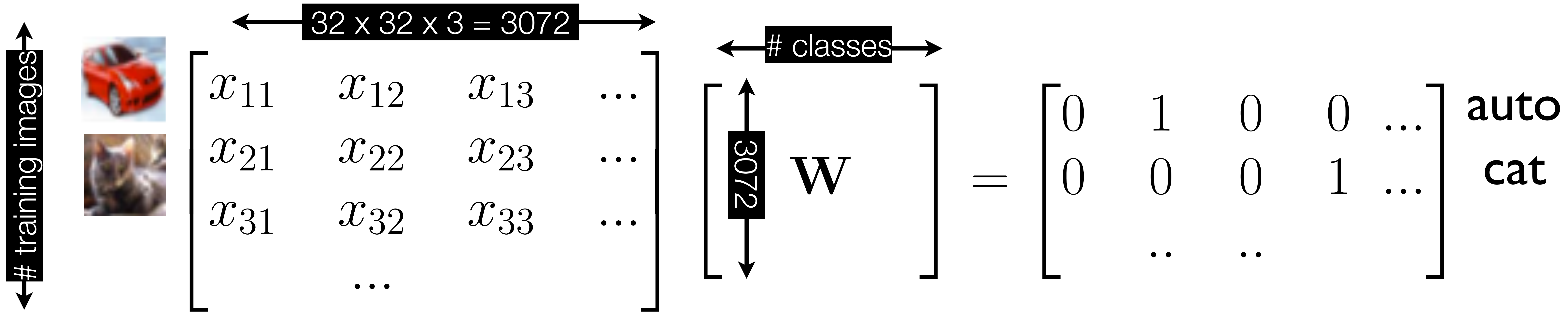
training images

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots \\ x_{21} & x_{22} & x_{23} & \dots \\ x_{31} & x_{32} & x_{33} & \dots \\ & \dots & & \end{bmatrix} \begin{bmatrix} & & \\ & \mathbf{W} & \\ & & \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ & .. & .. & & \end{bmatrix} \begin{matrix} \text{auto} \\ \text{cat} \\ \\ \end{matrix}$$

$$\mathbf{XW} = \mathbf{T}$$

One-hot Regression

Transpose



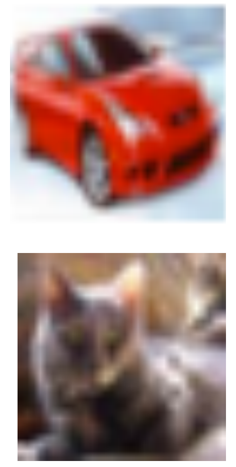
$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots \\ x_{21} & x_{22} & x_{23} & \dots \\ x_{31} & x_{32} & x_{33} & \dots \\ & \dots & & \end{bmatrix} \begin{bmatrix} & \mathbf{W} & \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ & .. & .. & & \end{bmatrix}$$

$$\mathbf{XW} = \mathbf{T}$$

One-hot Regression

Transpose

training images



$$\underbrace{\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots \\ x_{21} & x_{22} & x_{23} & \dots \\ x_{31} & x_{32} & x_{33} & \dots \\ & \dots & & \end{bmatrix}}_{32 \text{ x } 32 \text{ x } 3 = 3072} \underbrace{\begin{bmatrix} & & \\ 3072 & \mathbf{W} & \\ & & \end{bmatrix}}_{\#\text{ classes}} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ & .. & .. & & \end{bmatrix} \begin{matrix} \text{auto} \\ \text{cat} \\ \end{matrix}$$

$$\mathbf{XW} = \mathbf{T}$$

Solve regression problem by Least Squares

$$\mathcal{L} = |\mathbf{XW} - \mathbf{T}|^2$$

One-hot Regression

Transpose

training images

32 x 32 x 3 = 3072

classes

3072

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots \\ x_{21} & x_{22} & x_{23} & \dots \\ x_{31} & x_{32} & x_{33} & \dots \\ & \dots & & \end{bmatrix} \begin{bmatrix} & \mathbf{W} & \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ & .. & .. & & \end{bmatrix} \begin{matrix} \textbf{auto} \\ \textbf{cat} \\ \\ \end{matrix}$$

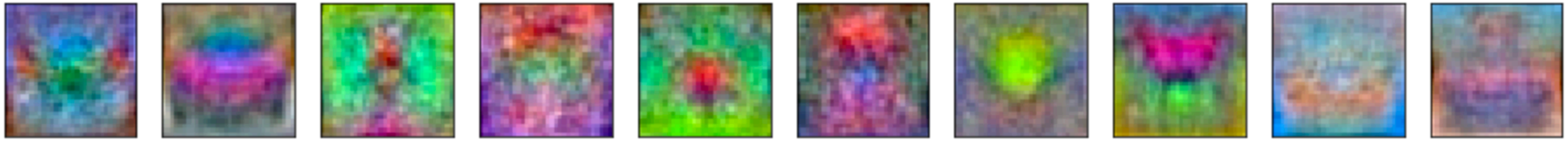
$$\mathbf{XW} = \mathbf{T}$$

Solve regression problem by Least Squares

$$\mathcal{L} = |\mathbf{XW} - \mathbf{T}|^2$$

Why this maybe sub-optimal?

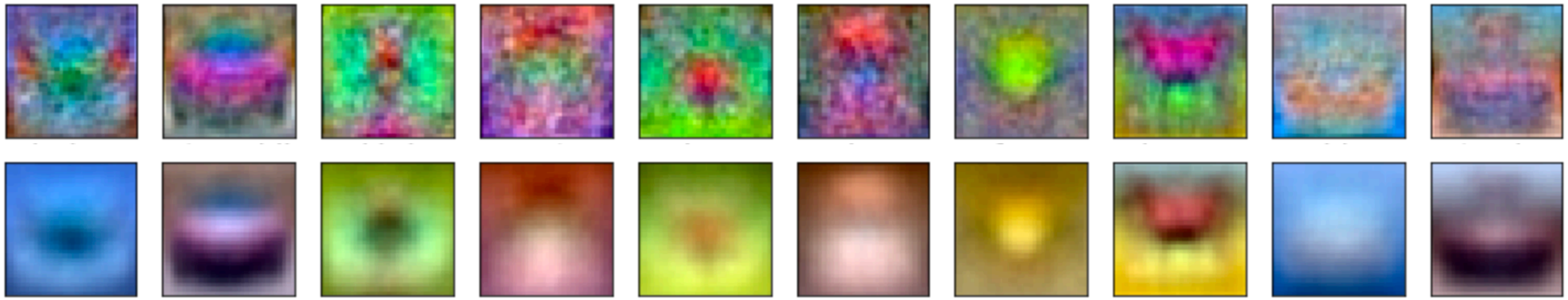
One-hot Regression



Solve regression problem by Least Squares

$$\mathcal{L} = |\mathbf{XW} - \mathbf{T}|^2$$

One-hot Regression



Solve regression problem by Least Squares

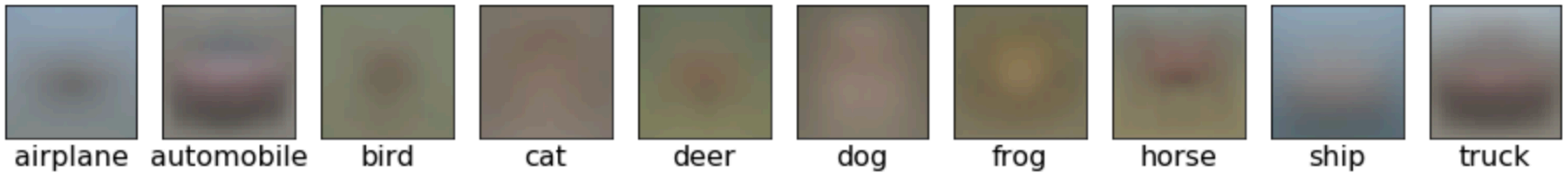
$$\mathcal{L} = |\mathbf{XW} - \mathbf{T}|^2 + \lambda|\mathbf{W}|^2$$

Recall: **Nearest Mean** Classifier

Find the nearest mean and assign class:

$$c_q = \arg\min_i |\mathbf{x}_q - \mathbf{m}_i|^2$$

CIFAR10 class means:

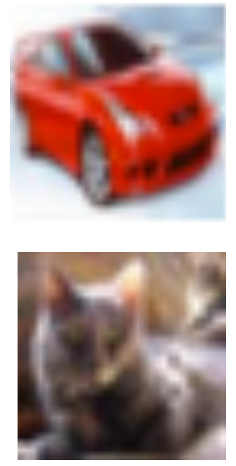


One-hot Regression

10 Neurons with simple-linear (identity) activation function

Transpose

training images



32 x 32 x 3 = 3072

classes

3072

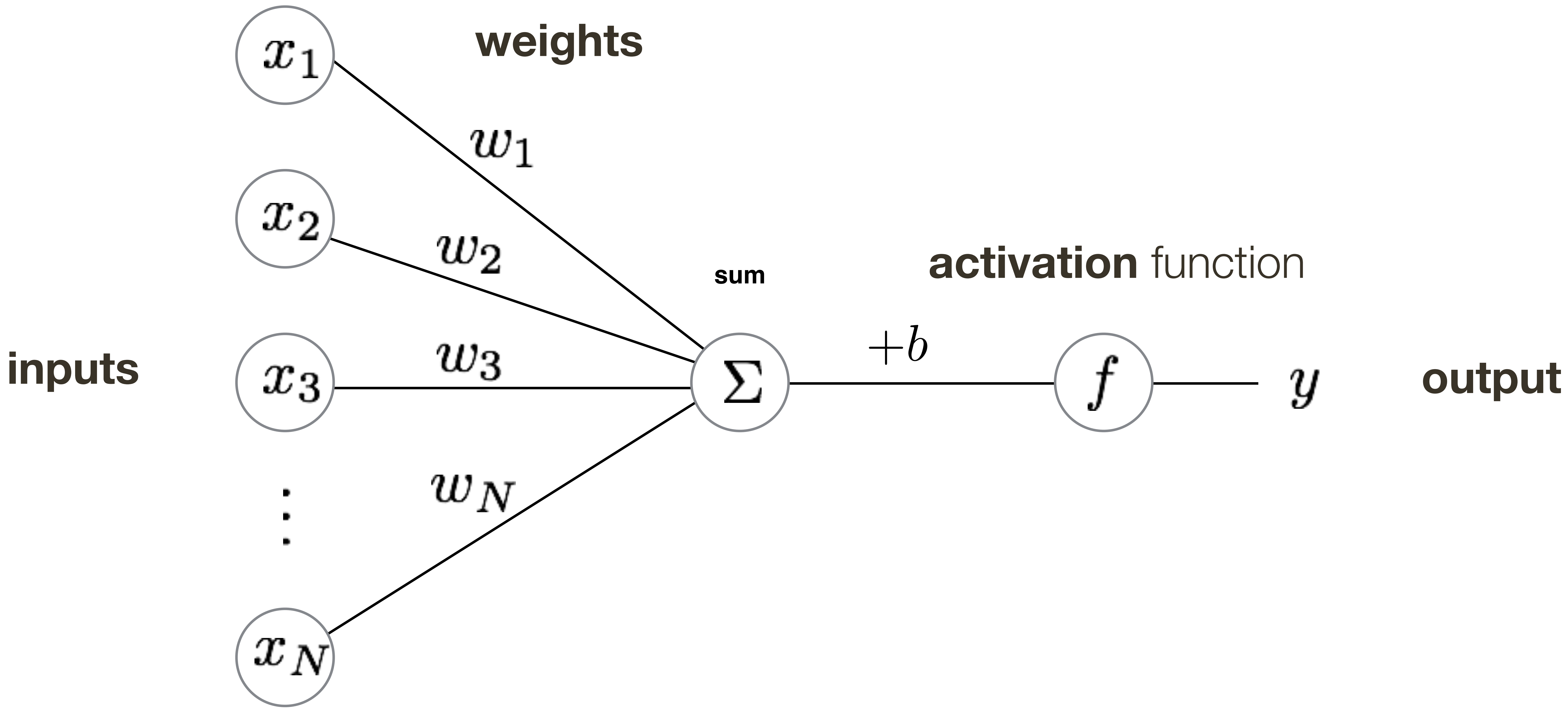
$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots \\ x_{21} & x_{22} & x_{23} & \dots \\ x_{31} & x_{32} & x_{33} & \dots \\ & \dots & & \end{bmatrix} \begin{bmatrix} \mathbf{W} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ & .. & .. & & \end{bmatrix} \begin{matrix} \text{auto} \\ \text{cat} \\ \\ \end{matrix}$$

$$\mathbf{XW} = \mathbf{T}$$

Solve regression problem by Least Squares

$$\mathcal{L} = |\mathbf{XW} - \mathbf{T}|^2 + \lambda|\mathbf{W}|^2$$

A **Neuron**

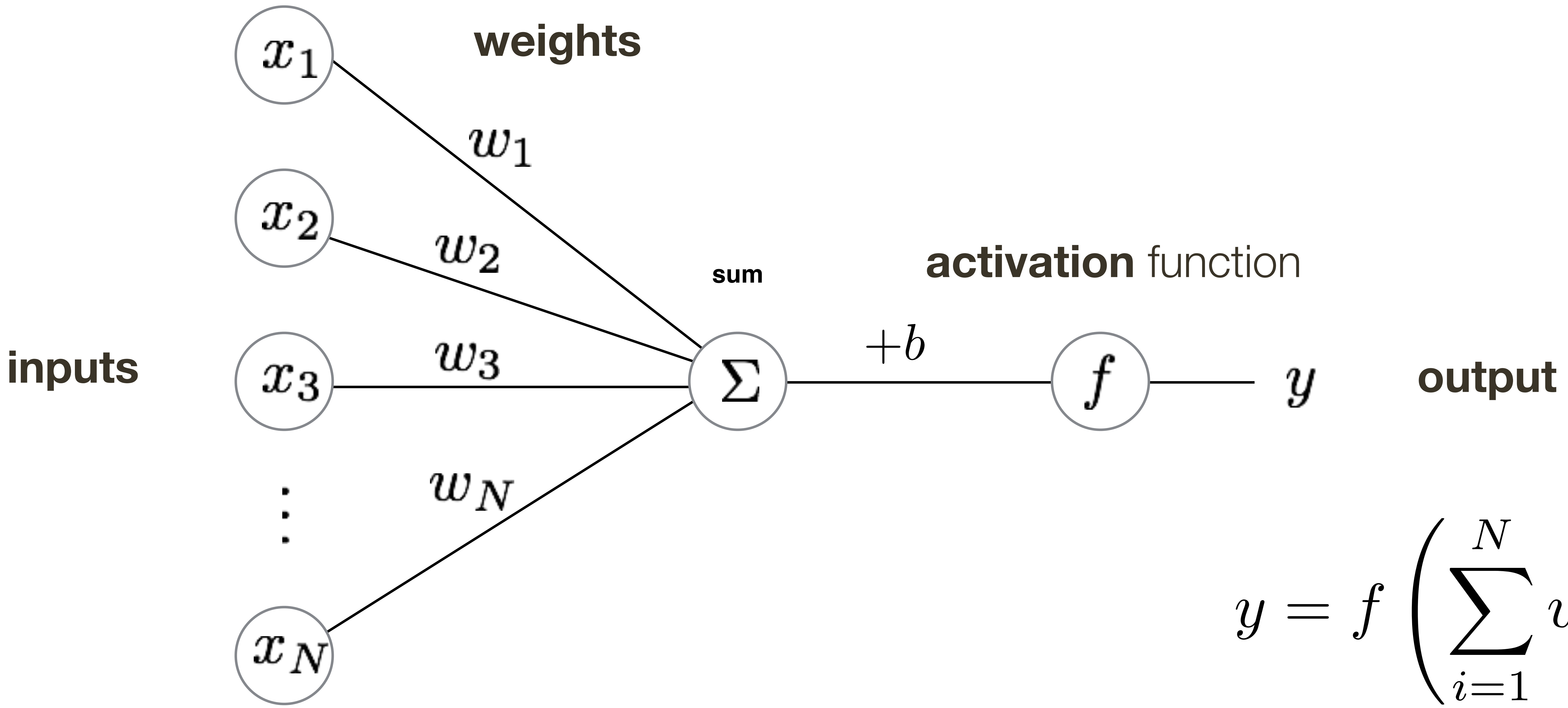


— The basic unit of computation in a neural network is a neuron.

— A neuron accepts some number of input signals, computes their weighted sum, and applies an **activation function** (or **non-linearity**) to the sum.

— Common activation functions include sigmoid and rectified linear unit (ReLU)

A **Neuron**



$$y = f\left(\sum_{i=1}^{N} w_i x_i + b\right)$$

— The basic unit of computation in a neural network is a neuron.

— A neuron accepts some number of input signals, computes their weighted sum, and applies an **activation function** (or **non-linearity**) to the sum.

— Common activation functions include sigmoid and rectified linear unit (ReLU)

Recall: Linear Classifier

Defines a score function:

$$f(\mathbf{x}_i, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x}_i + \mathbf{b}$$

image features

weights (parameters)

bias vector

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Recall: Linear Classifier

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

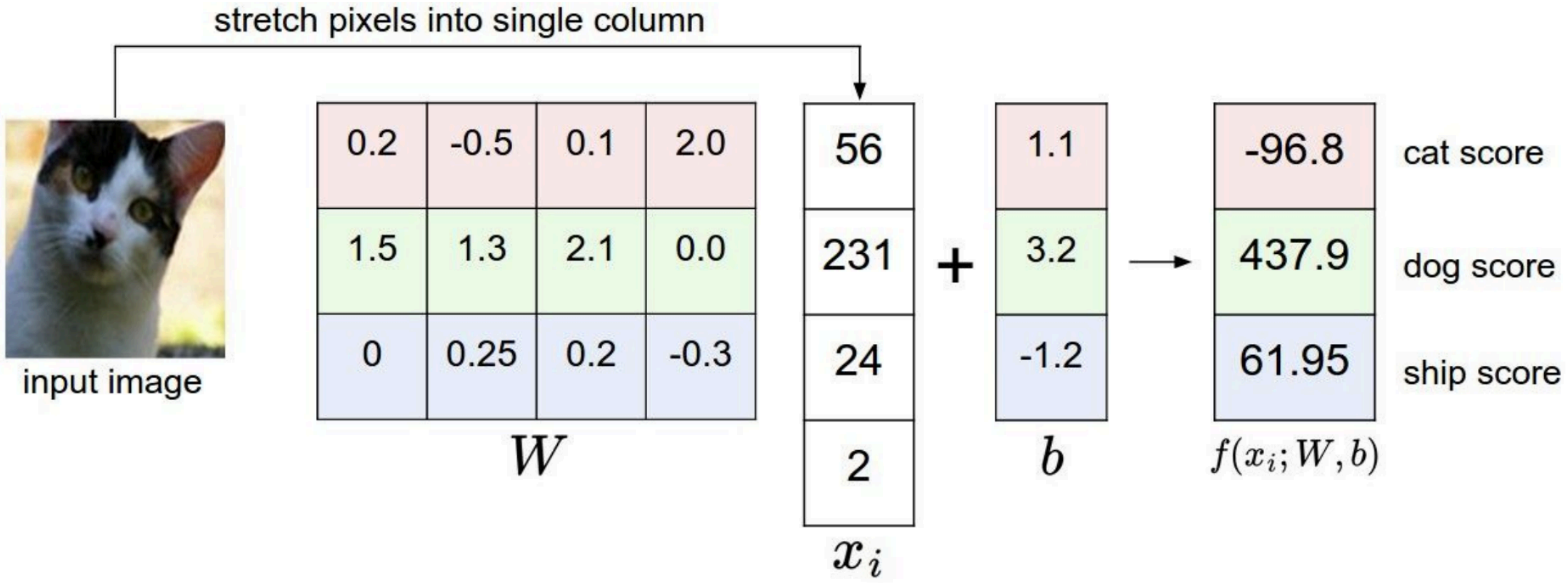
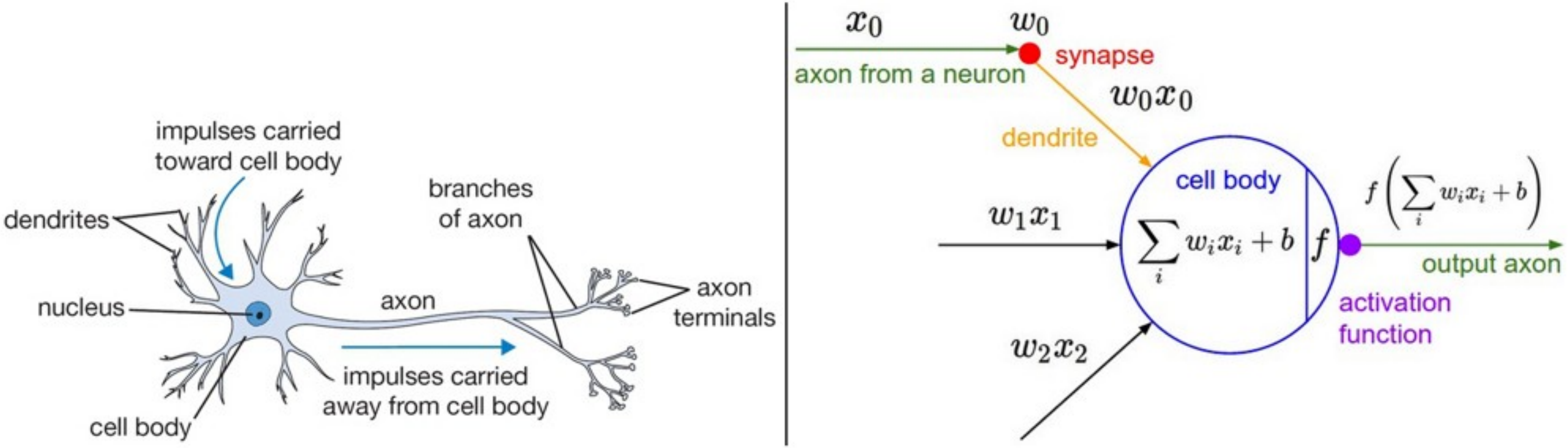


Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Aside: Inspiration from Biology

Figure credit: Fei-Fei and Karpathy



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Neural nets/perceptrons are loosely inspired by biology.

But they certainly are not a model of how the brain works, or even how neurons work.

Activation Function: **Sigmoid**

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

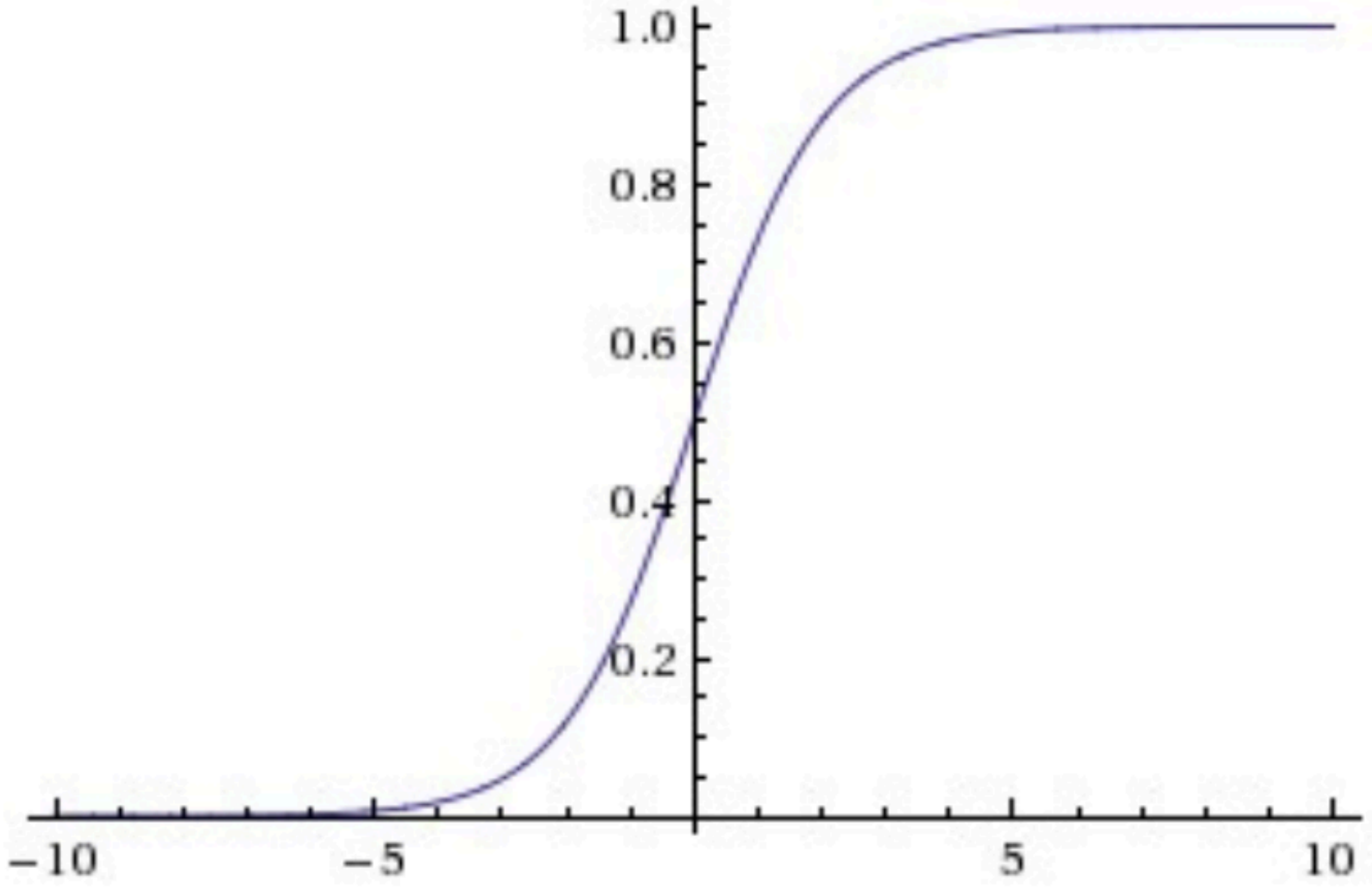


Figure credit: Fei-Fei and Karpathy

Common in many early neural networks

Biological analogy to saturated firing rate of neurons

Maps the input to the range [0,1]

Activation Function: **ReLU** (Rectified Linear Unit)

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

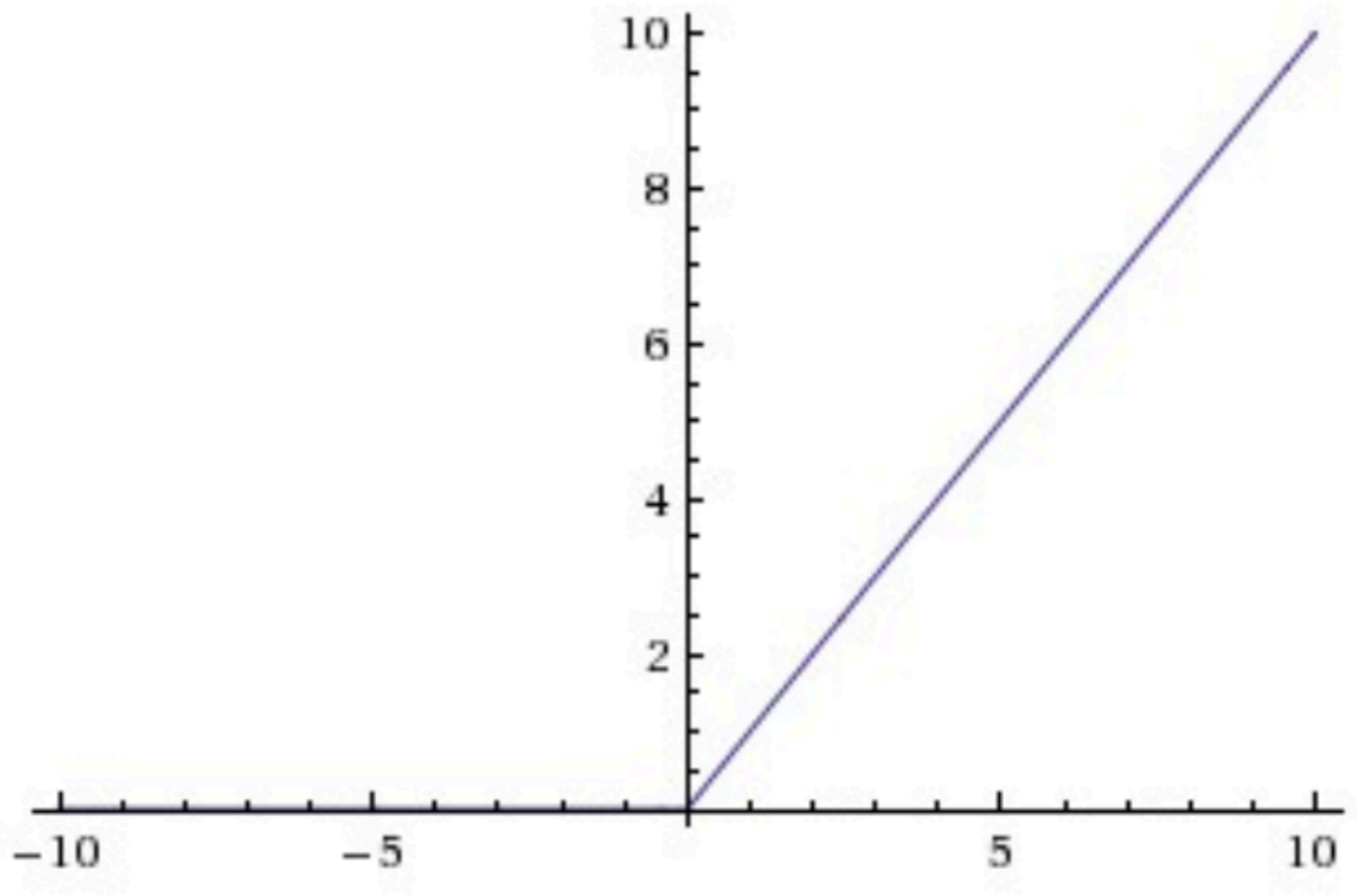
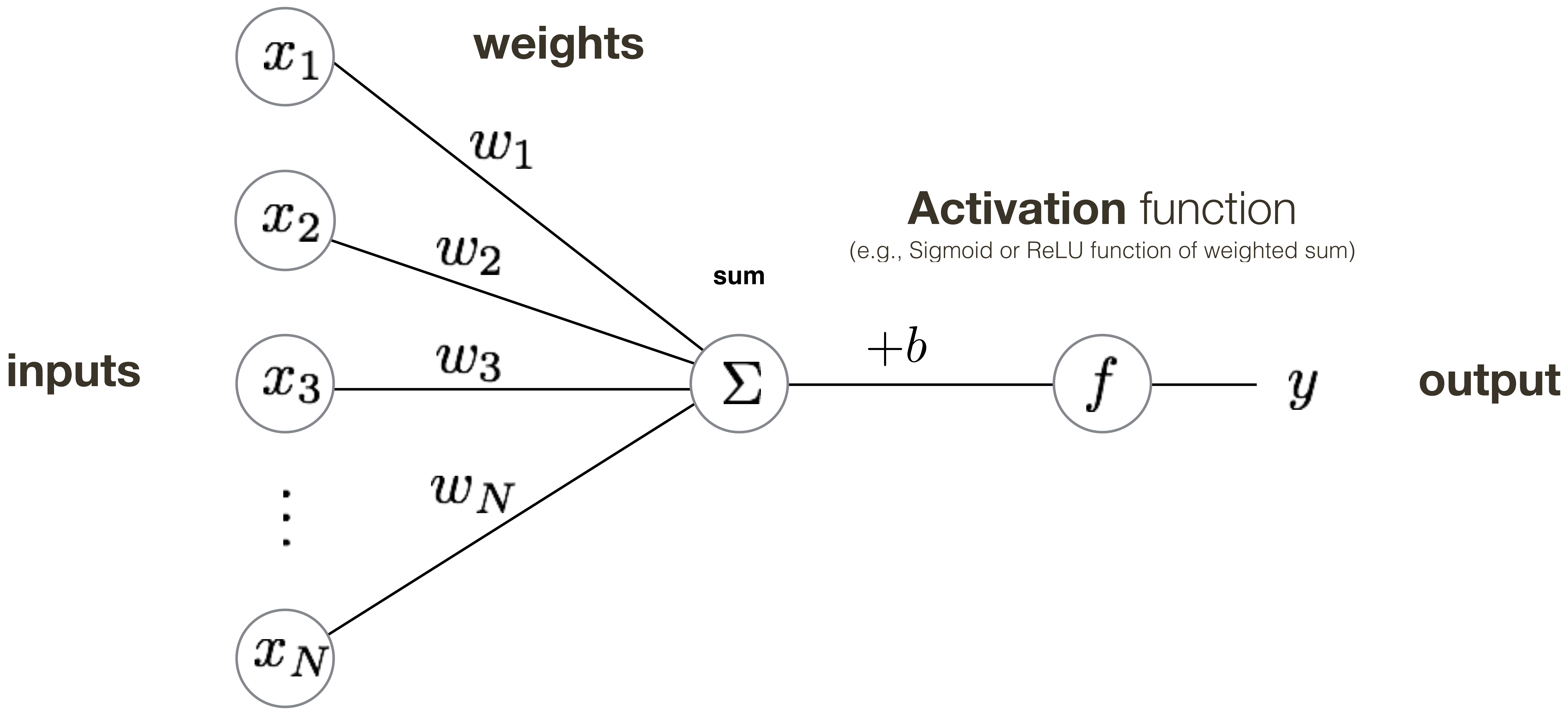


Figure credit: Fei-Fei and Karpathy

Found to accelerate convergence during learning

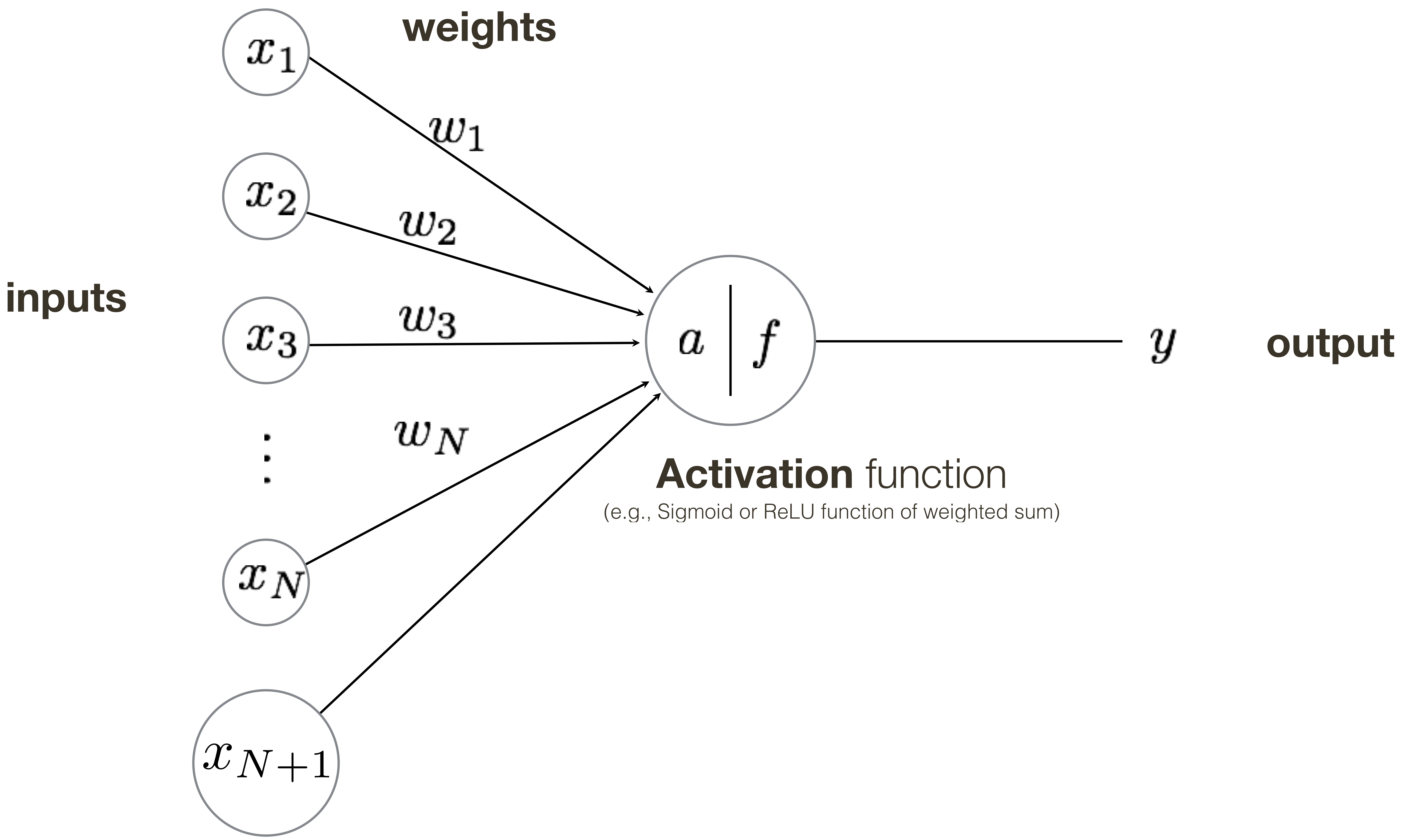
Used in the most recent neural networks

A **Neuron**



weights
w_1
w_2
w_3
w_N
inputs
x_1
x_2
x_3
x_N
sum
Σ
$+b$
Activation function
(e.g., Sigmoid or ReLU function of weighted sum)
f
y
output

A **Neuron** … another way to draw it …

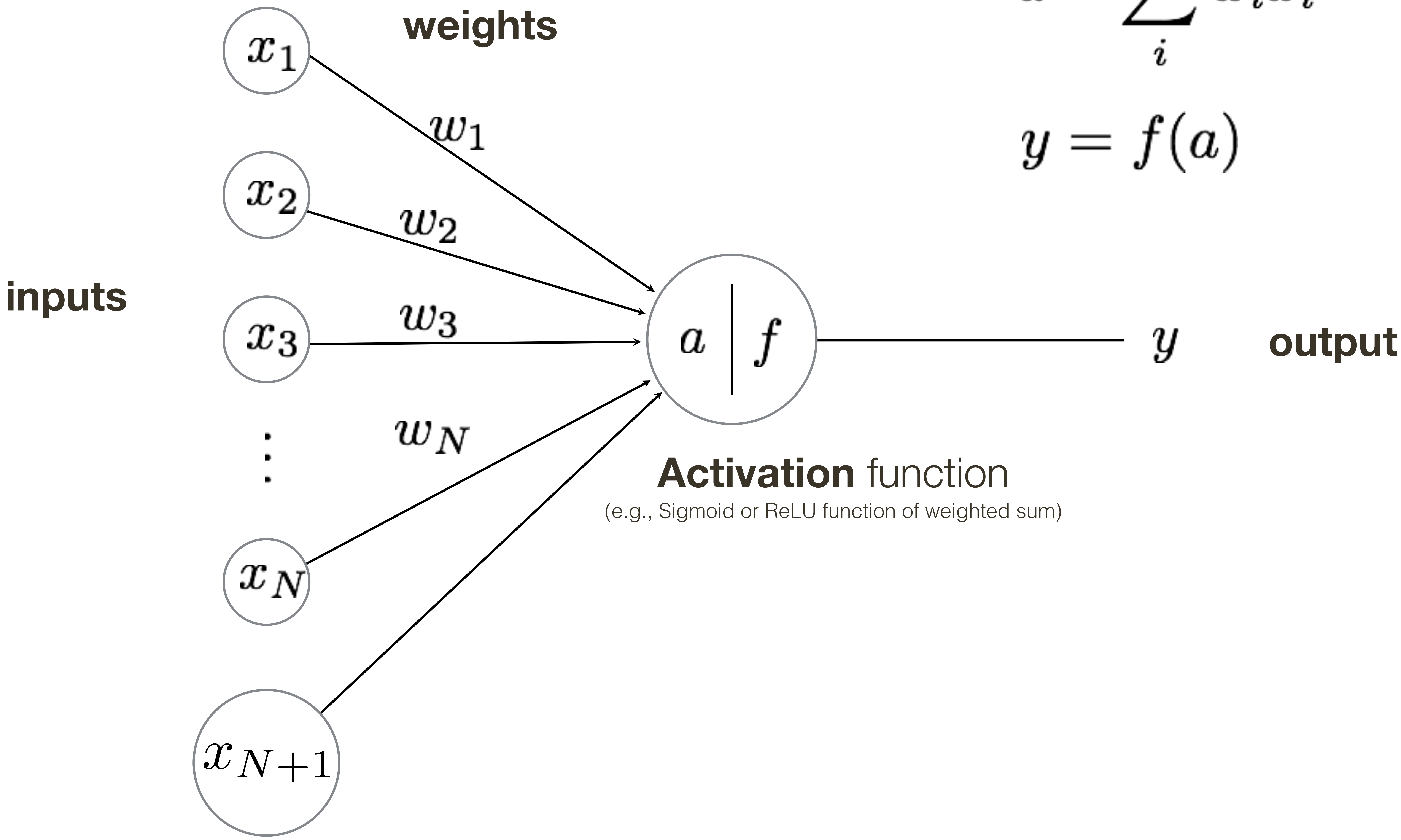


A **Neuron** … another way to draw it …

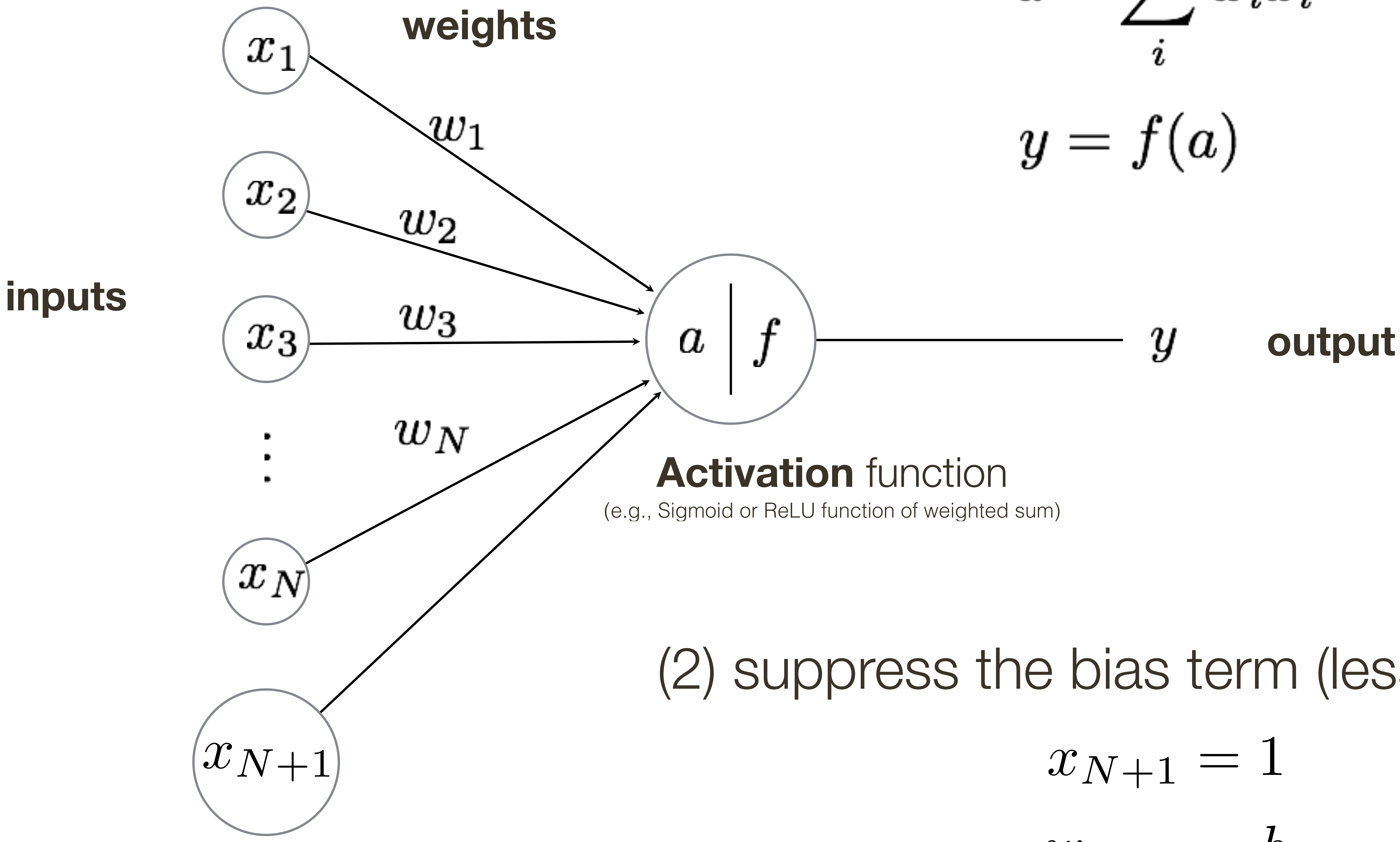
(1) Combine the sum and activation function

$$a = \sum_i w_i x_i$$

$$y = f(a)$$



A **Neuron** … another way to draw it …



(1) Combine the sum and activation function

$$a = \sum_i w_i x_i$$

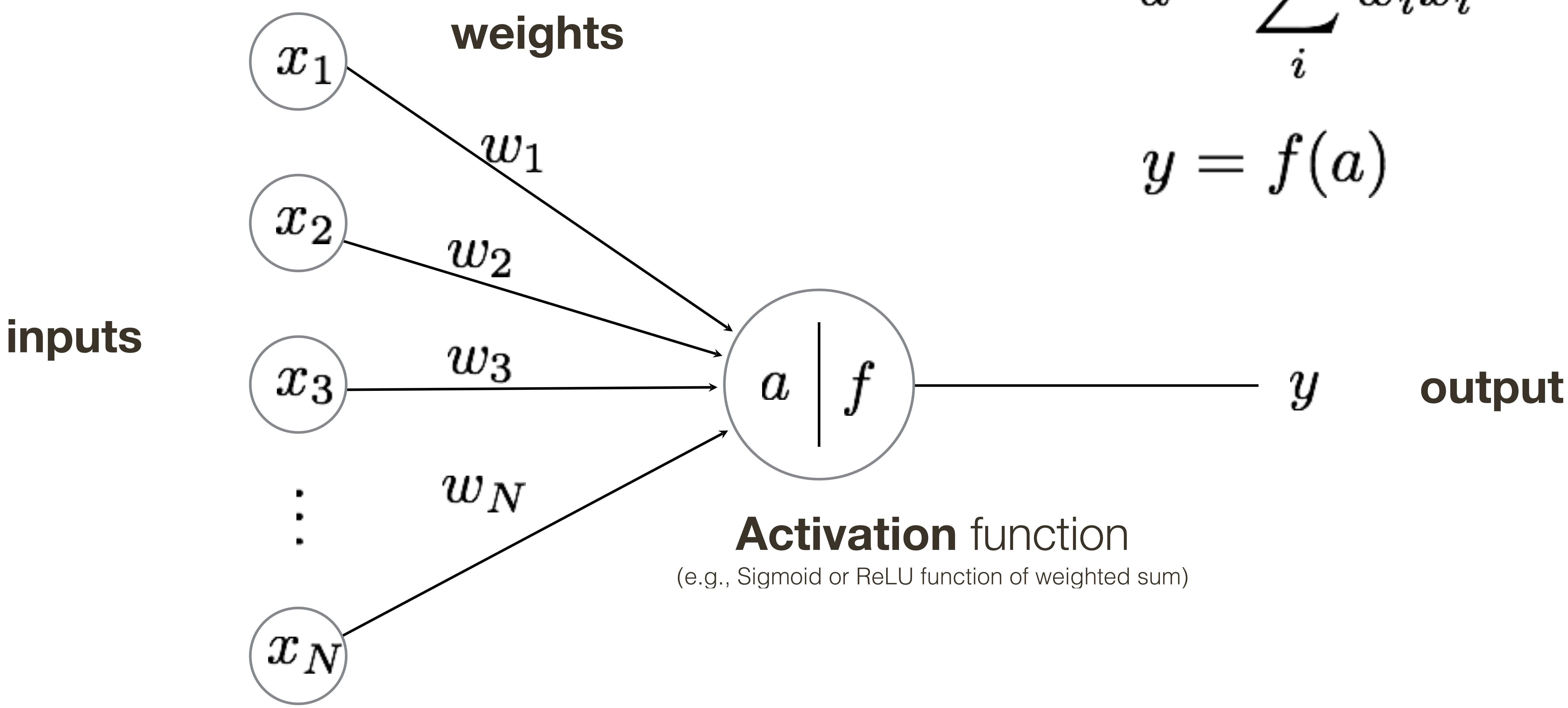
$$y = f(a)$$

(2) suppress the bias term (less clutter)

$$x_{N+1} = 1$$

$$w_{N+1} = b$$

A **Neuron** … another way to draw it …



Activation function

(e.g., Sigmoid or ReLU function of weighted sum)

(1) Combine the sum and activation function

$$a = \sum_i w_i x_i$$

$$y = f(a)$$

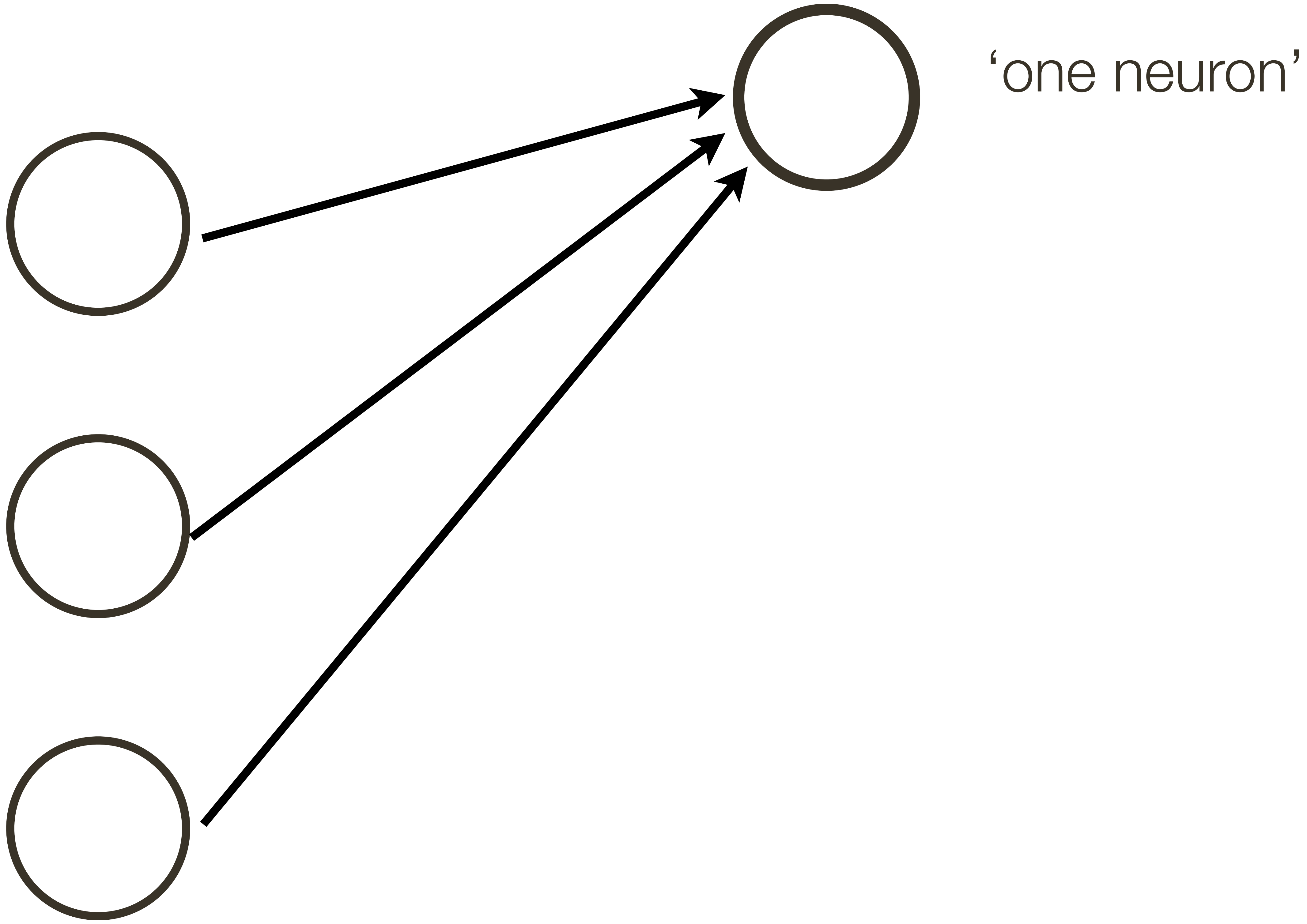
(2) suppress the bias term (less clutter)

$$x_{N+1} = 1$$

$$w_{N+1} = b$$

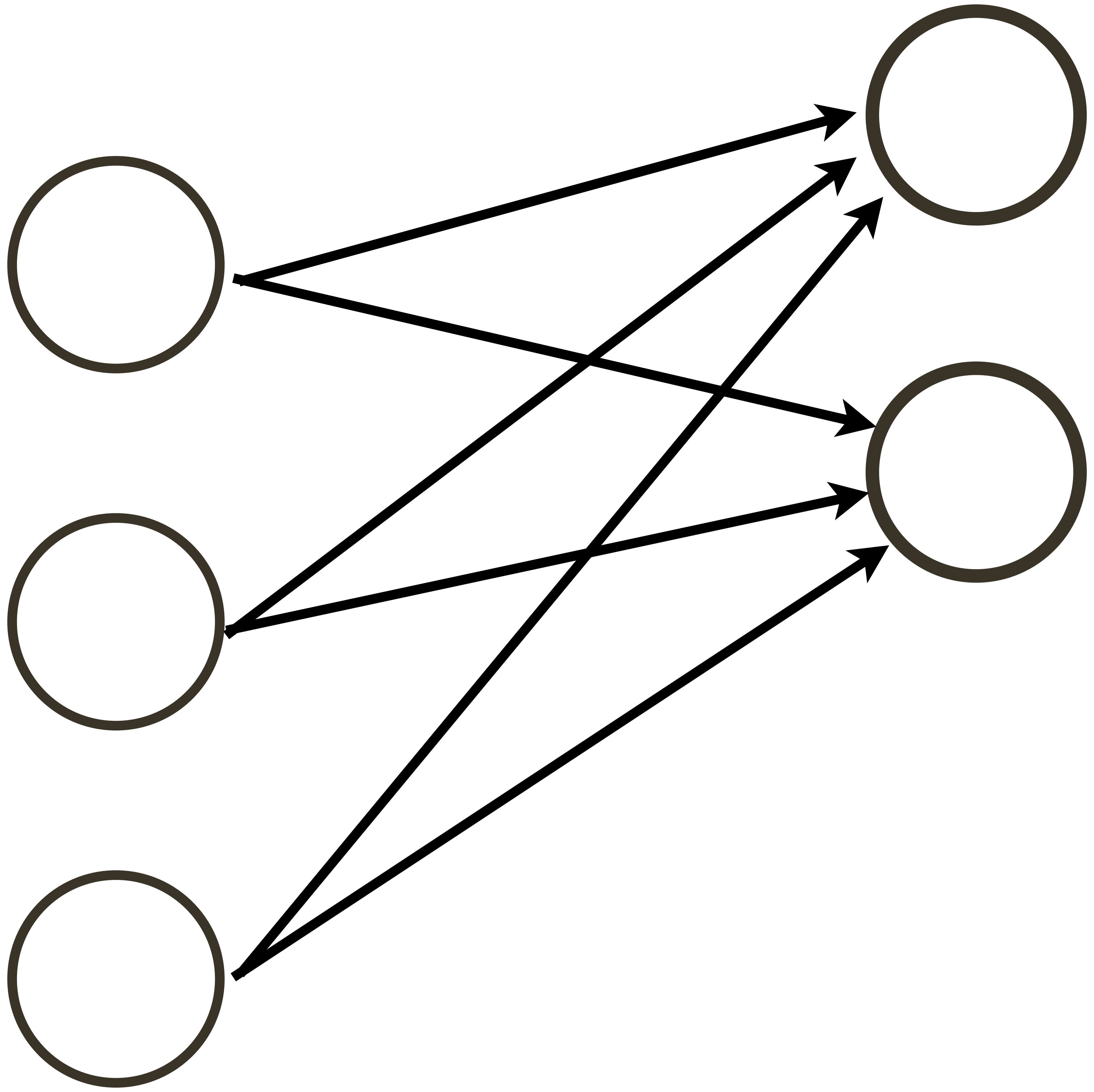
Neural Network

Connect a bunch of neurons together — a collection of connected neurons



Neural Network

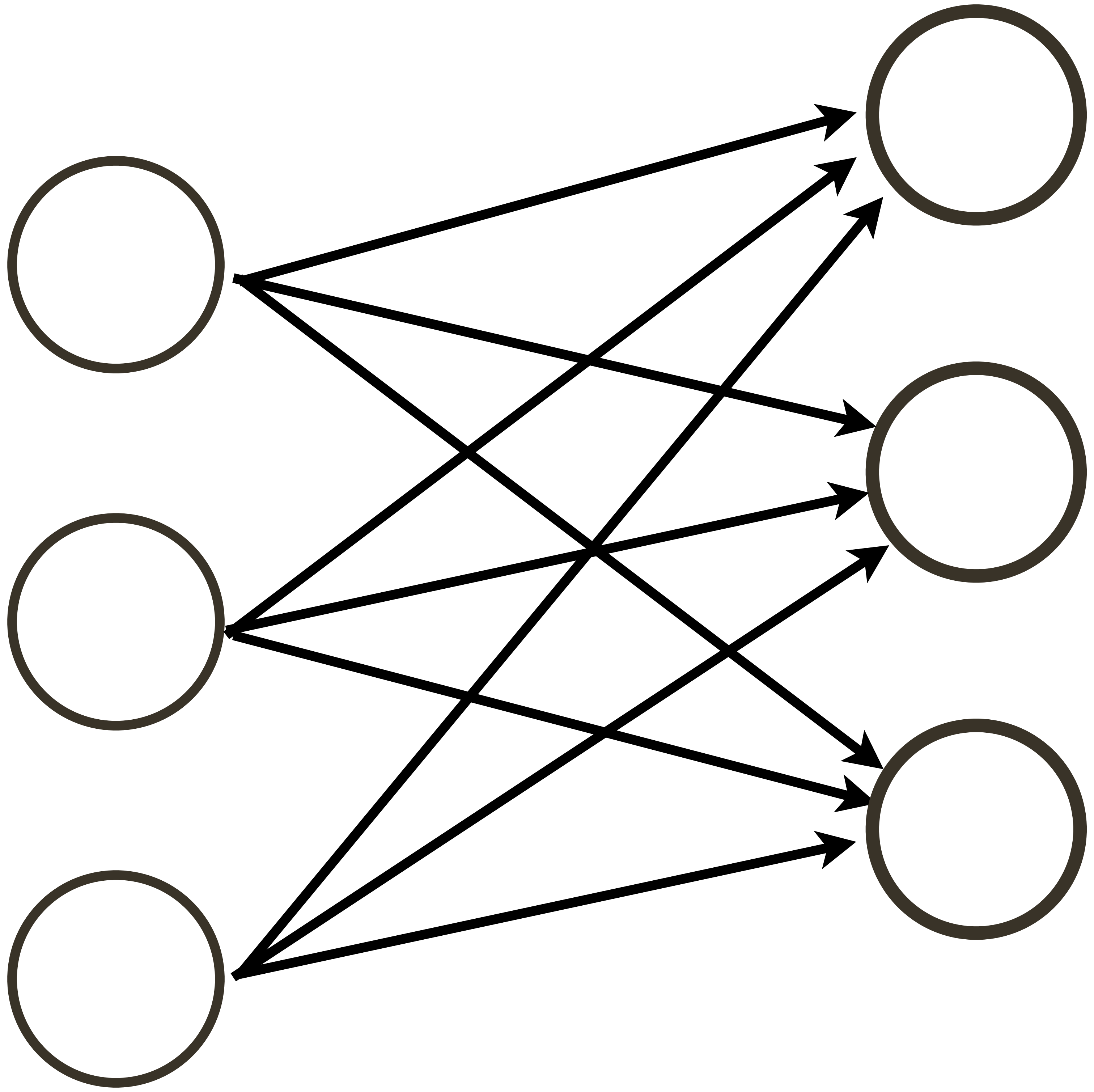
Connect a bunch of neurons together — a collection of connected neurons



‘two neurons’

Neural Network

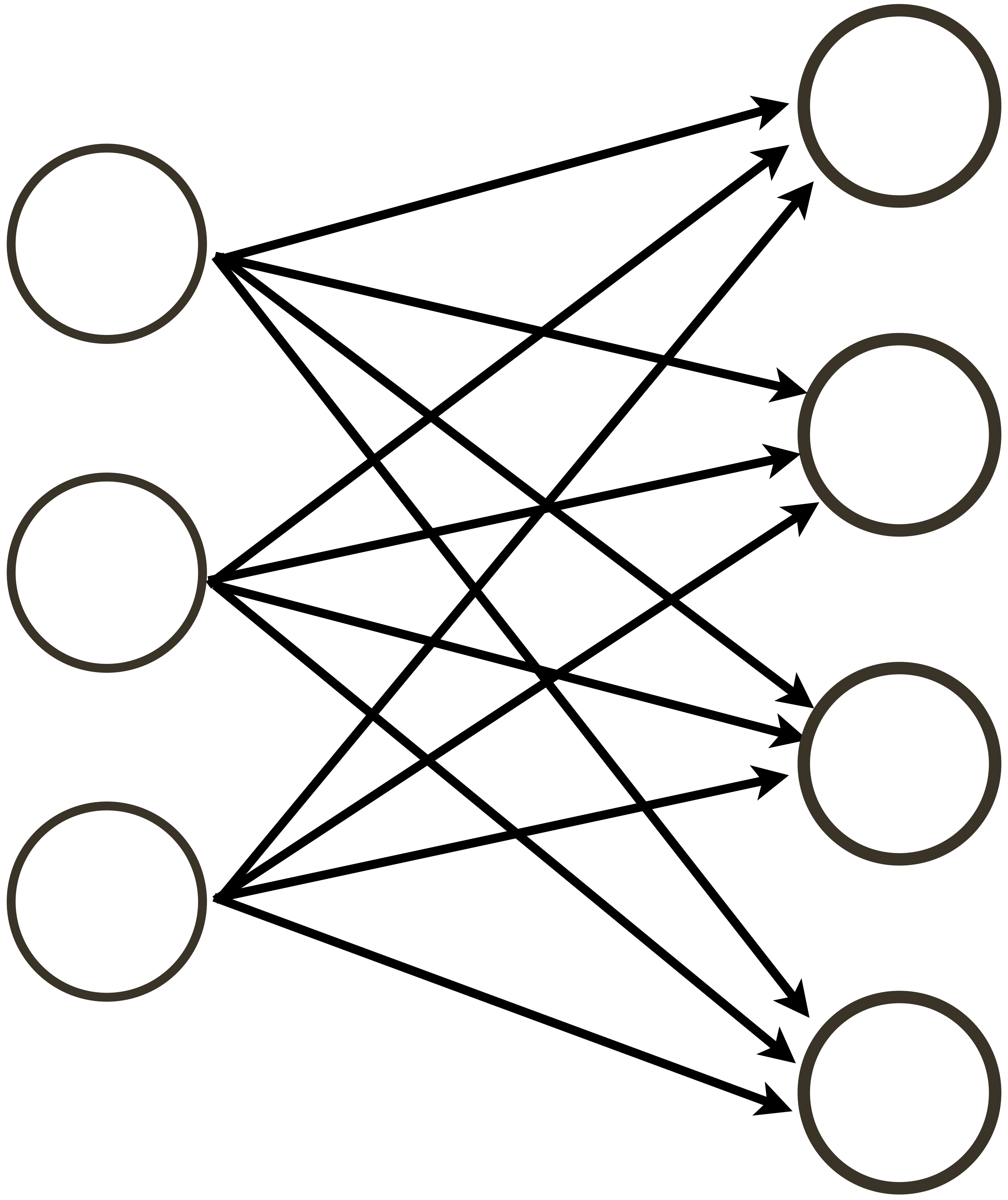
Connect a bunch of neurons together — a collection of connected neurons



'three neurons'

Neural Network

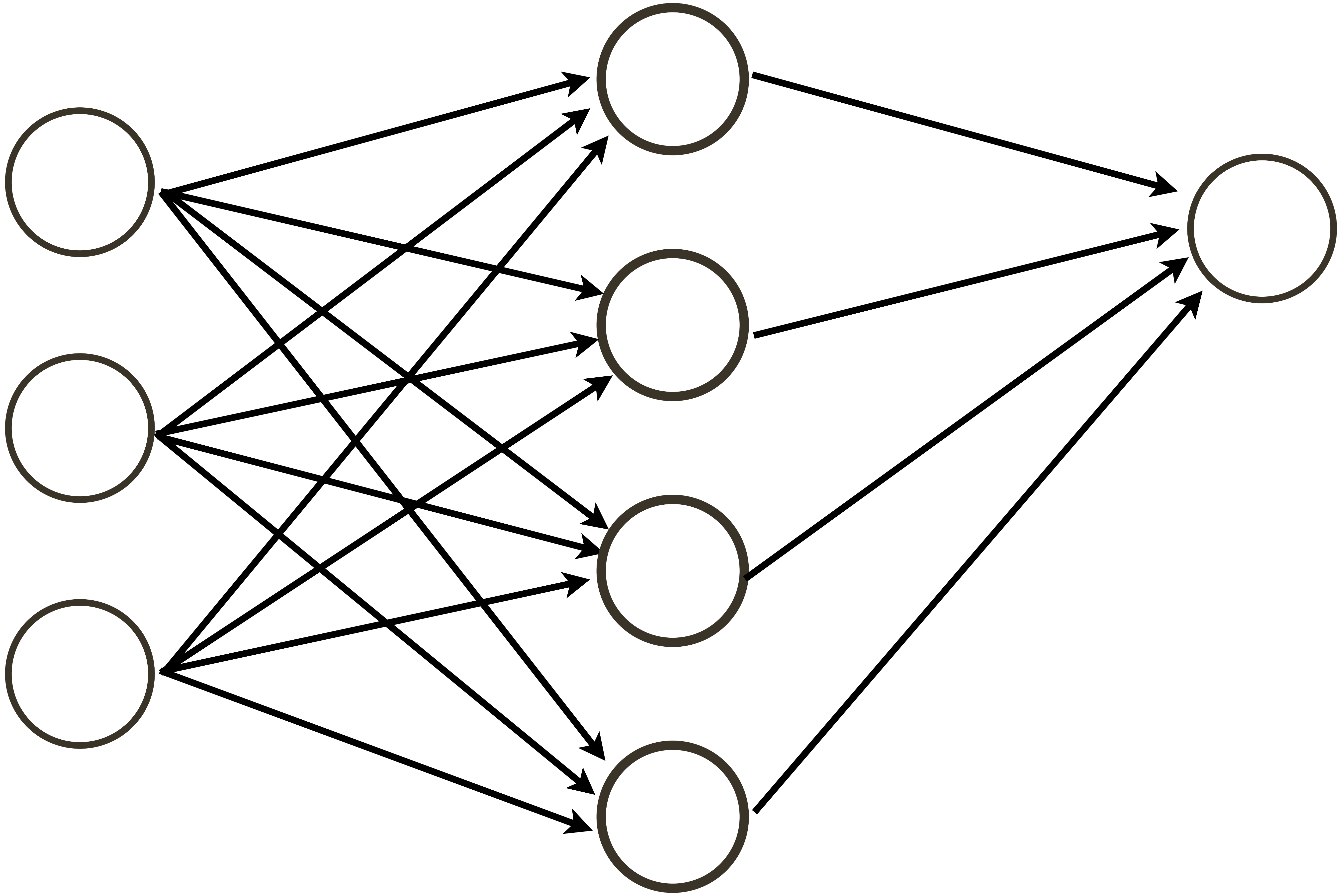
Connect a bunch of neurons together — a collection of connected neurons



‘four neurons’

Neural Network

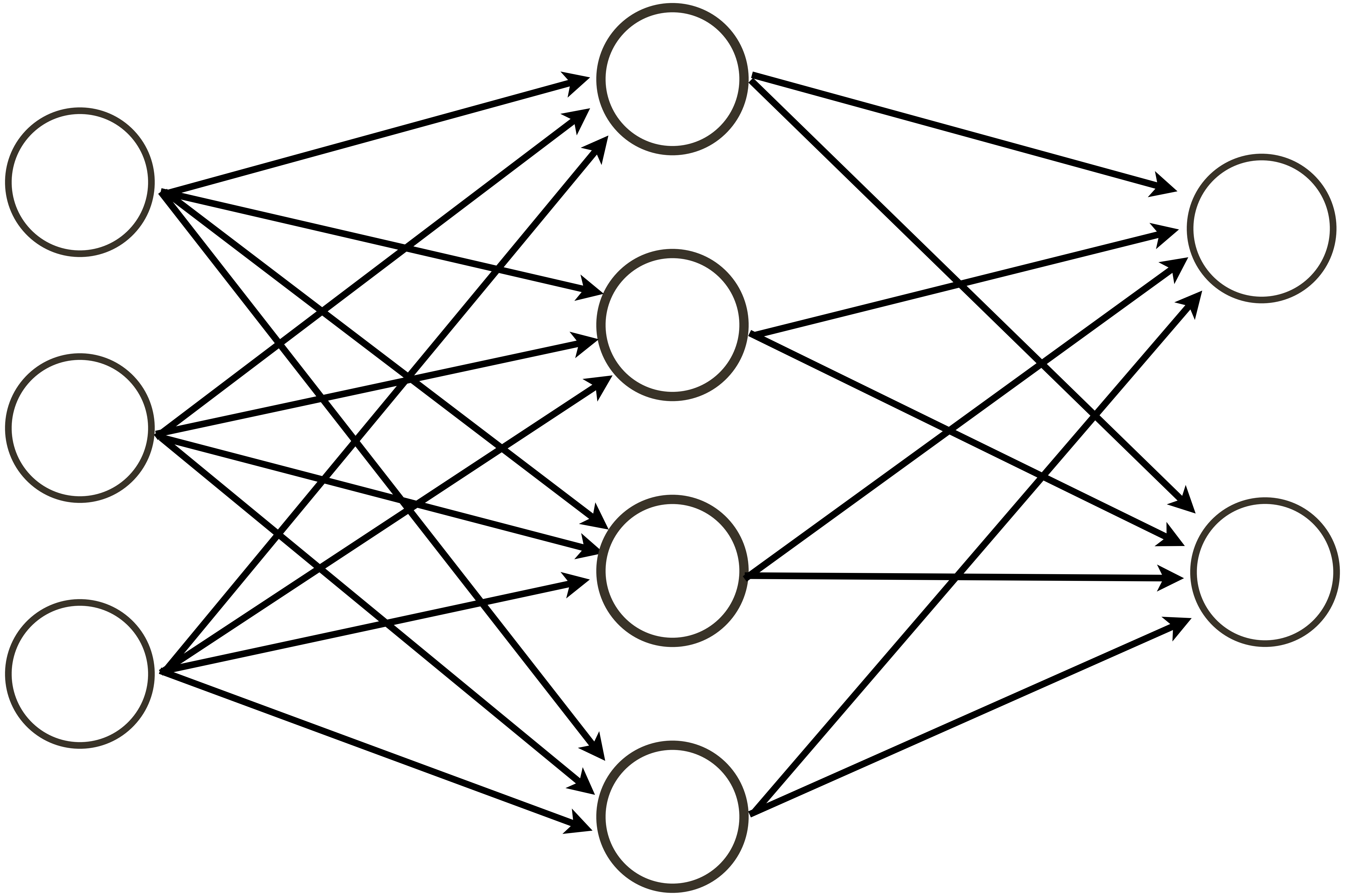
Connect a bunch of neurons together — a collection of connected neurons



‘five neurons’

Neural Network

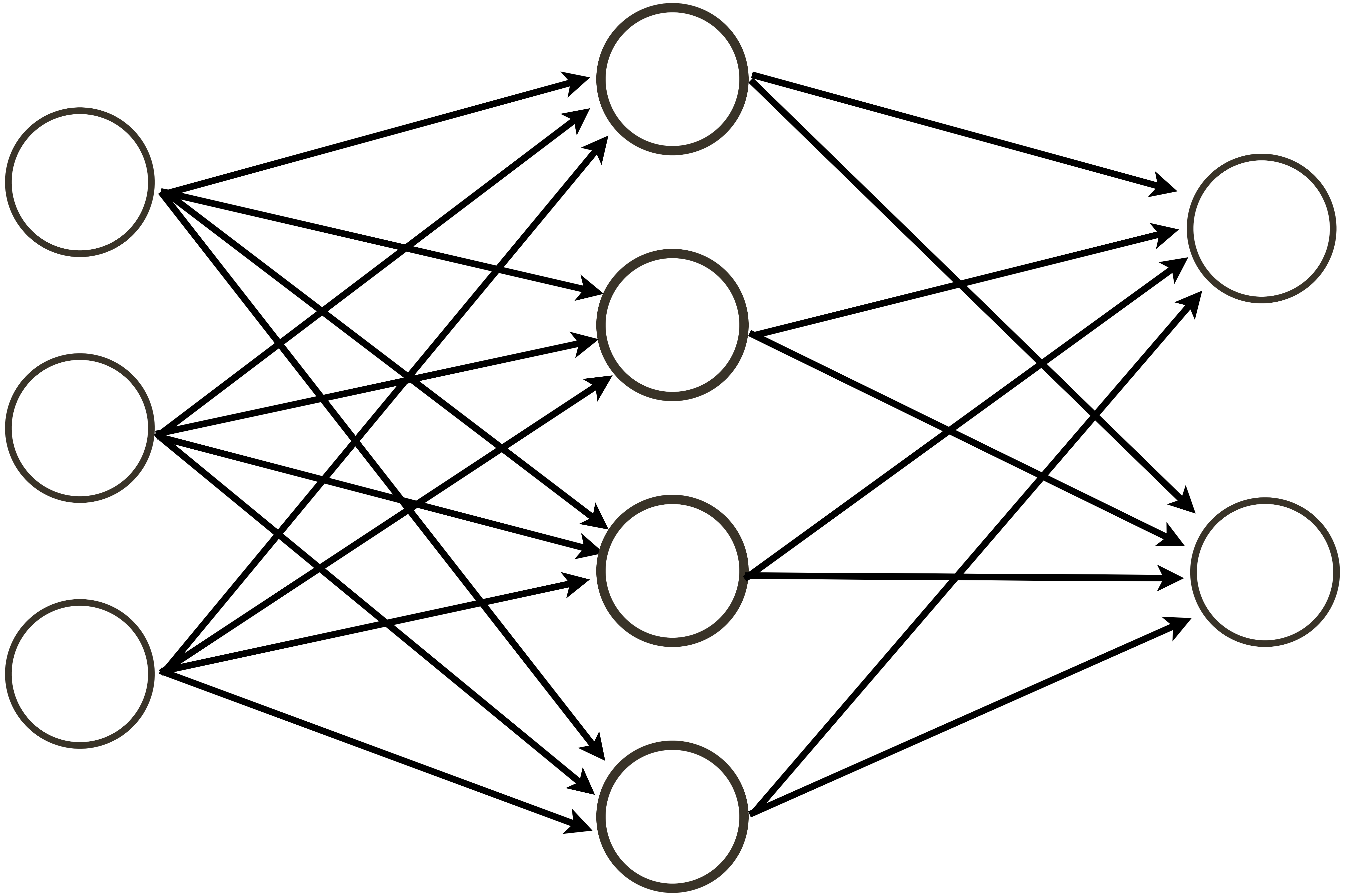
Connect a bunch of neurons together — a collection of connected neurons



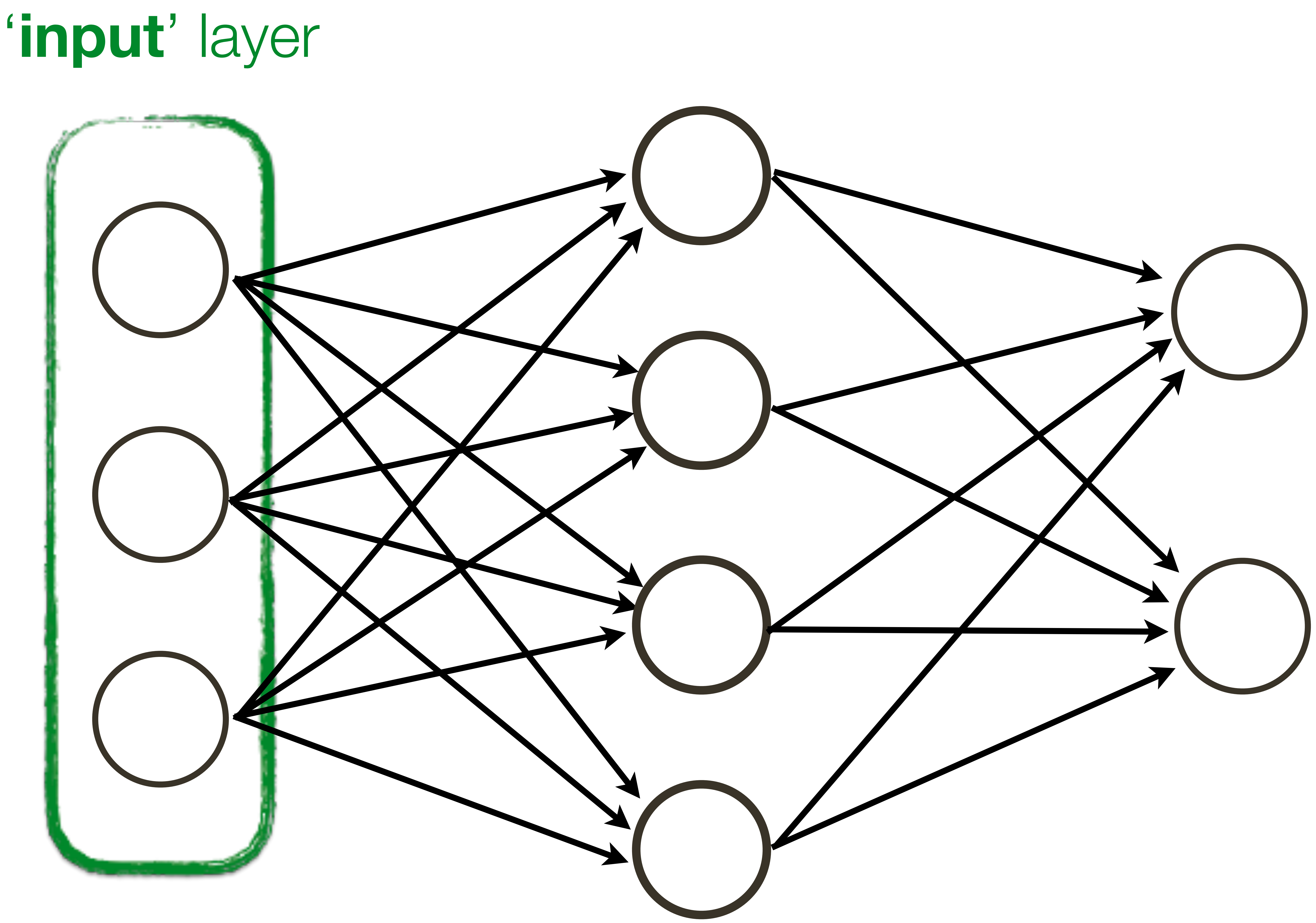
‘six neurons’

Neural Network

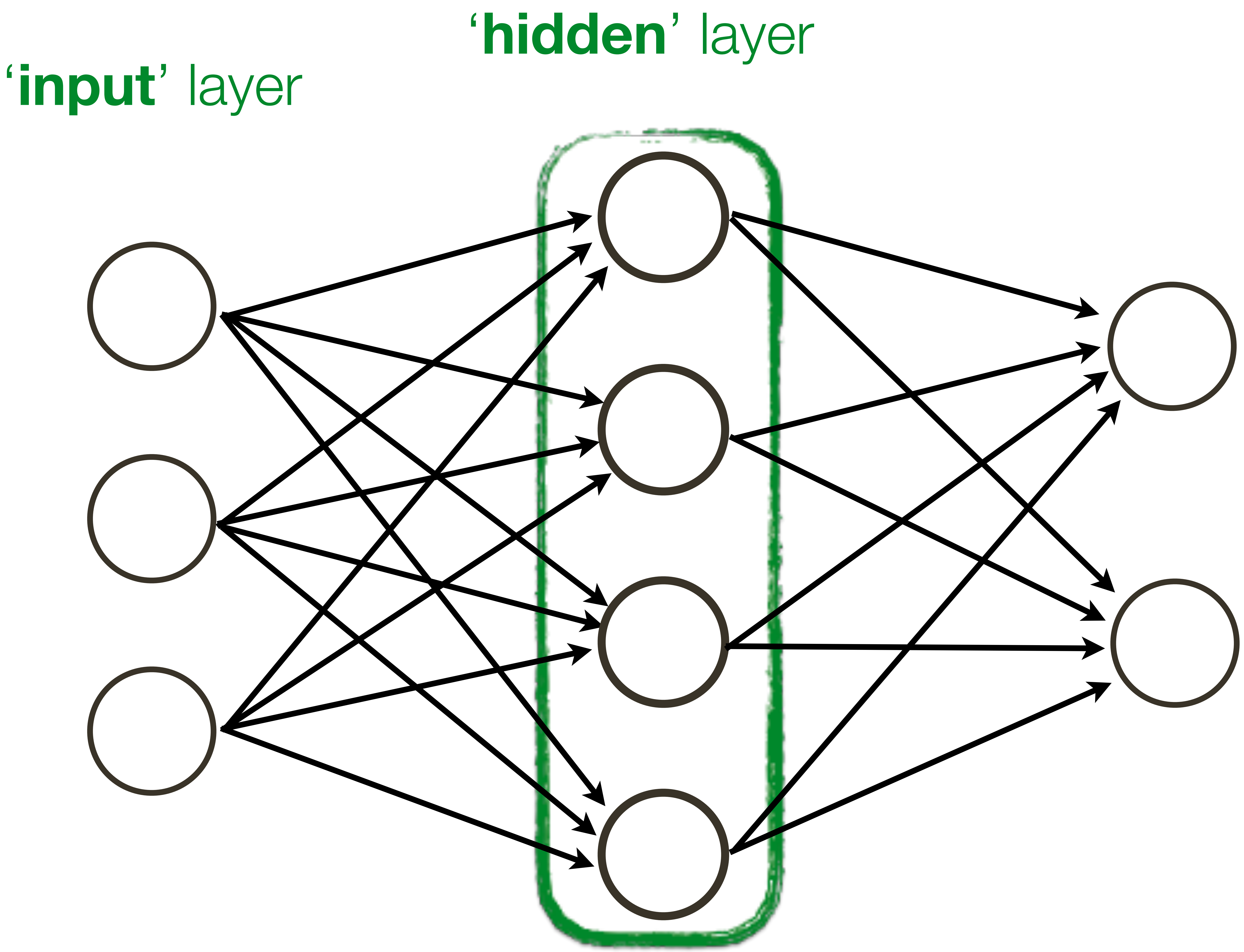
This network is also called a **Multi-layer Perceptron** (MLP)



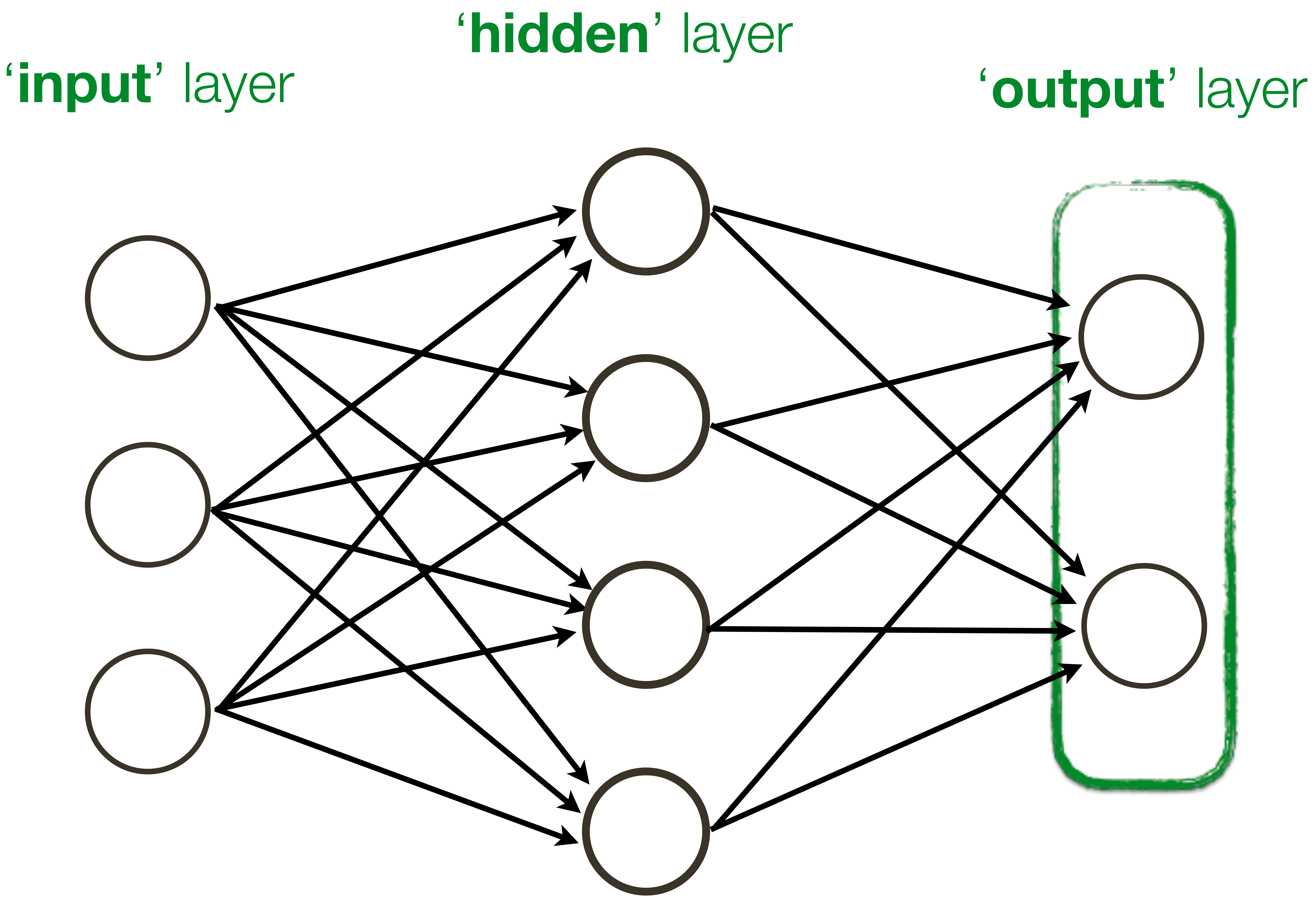
Neural Network: **Terminology**



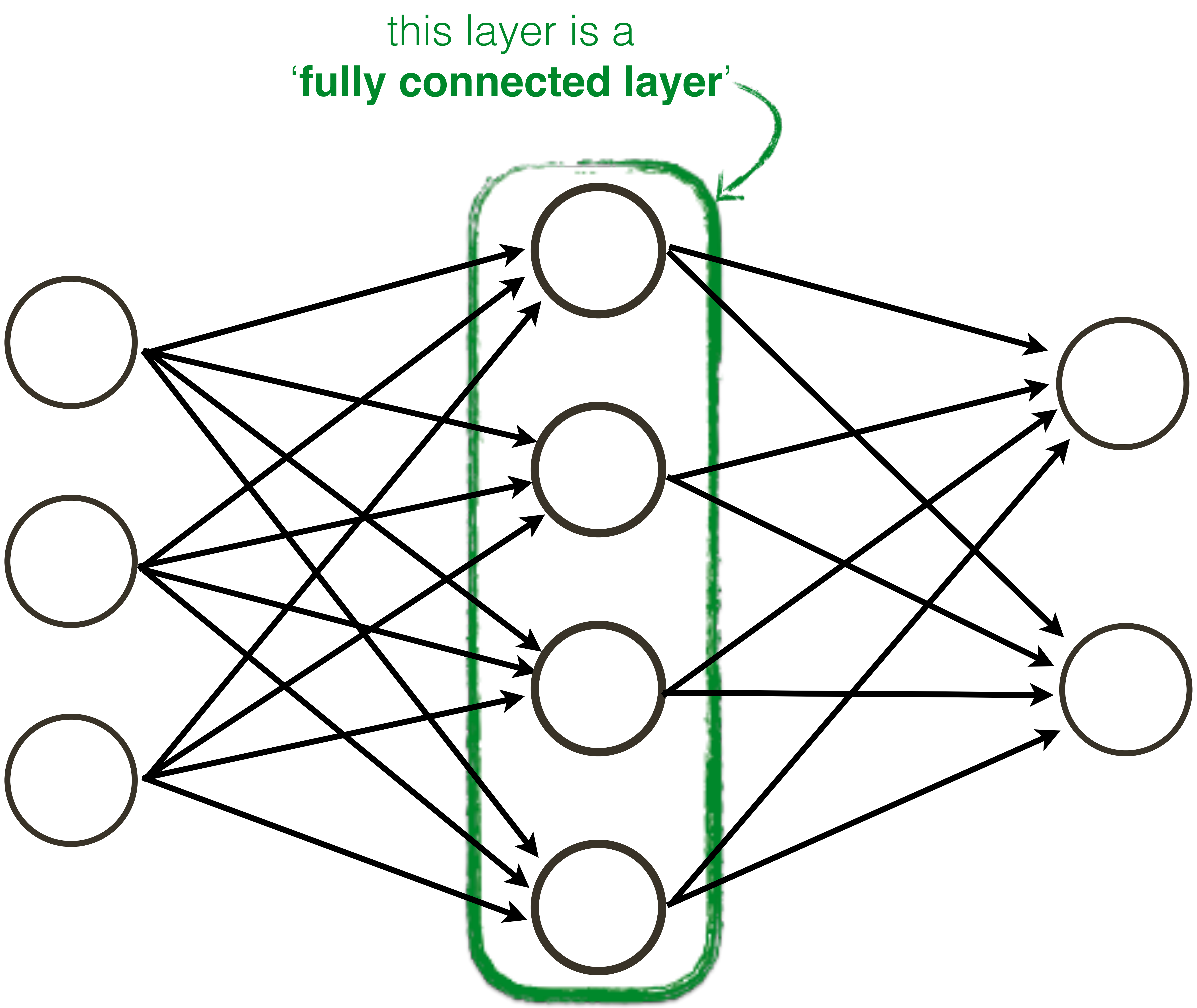
Neural Network: **Terminology**



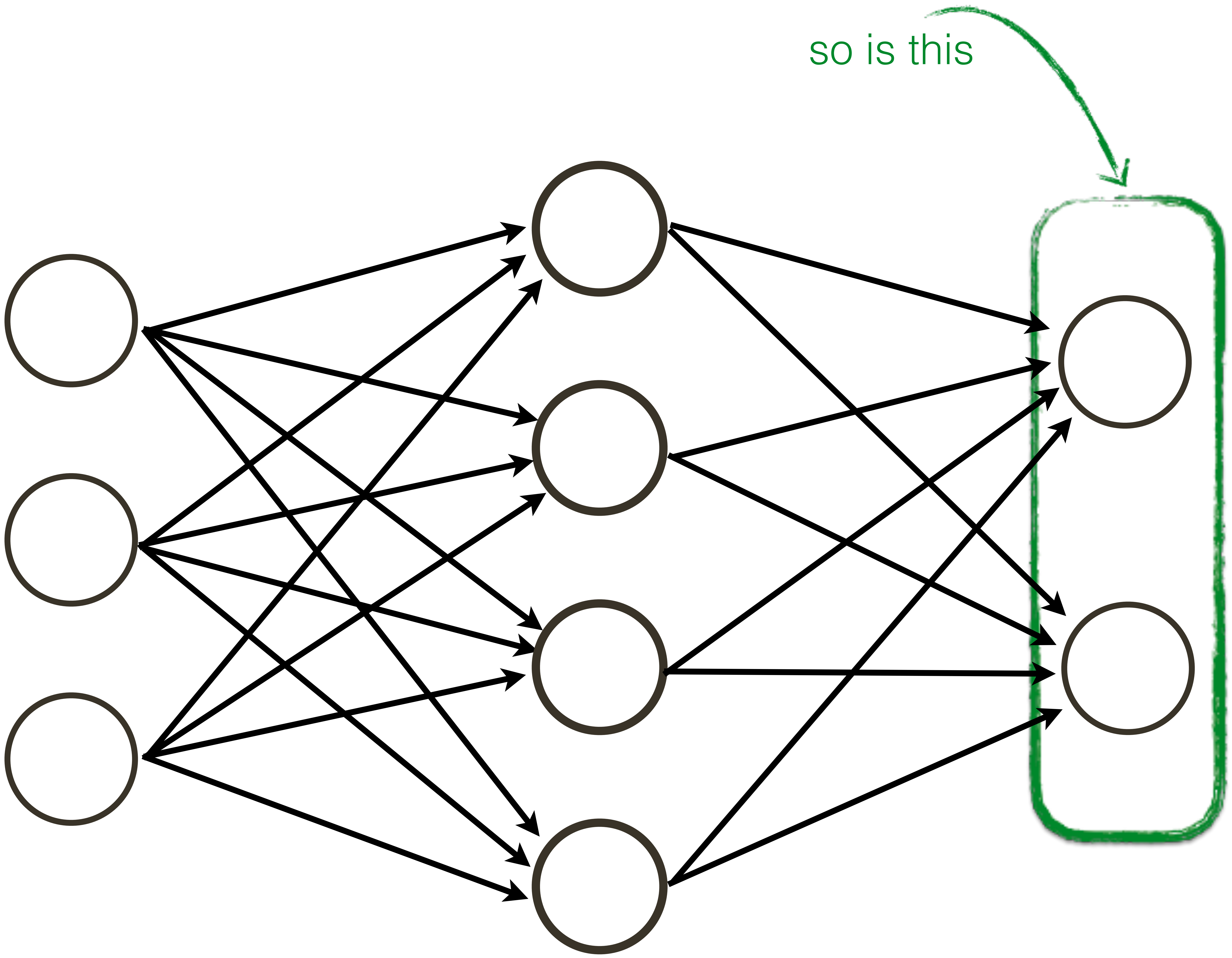
Neural Network: **Terminology**



Neural Network: **Terminology**



Neural Network: **Terminology**



Neural Network

A neural network comprises neurons connected in an acyclic graph

The outputs of neurons can become inputs to other neurons

Neural networks typically contain multiple layers of neurons

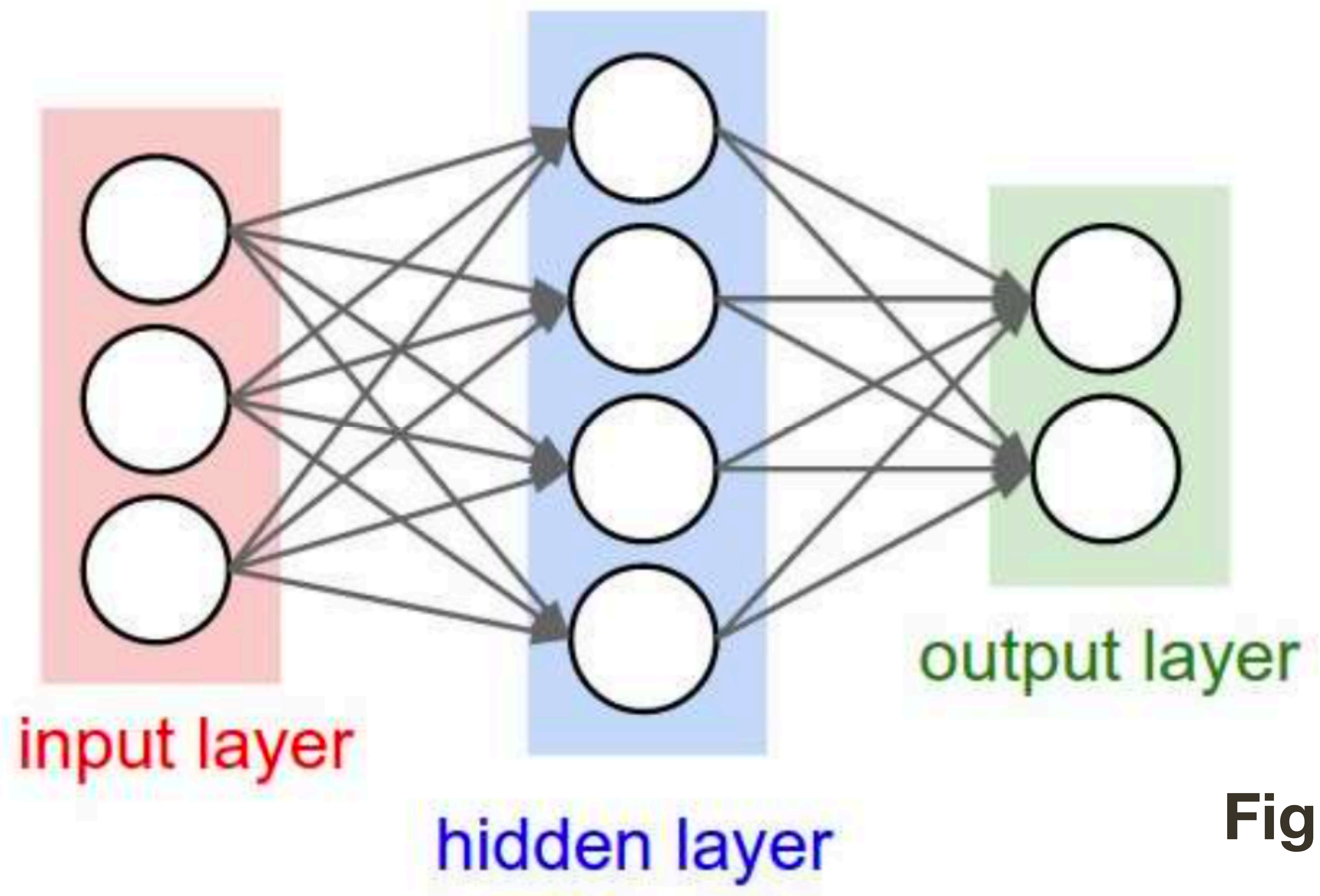


Figure credit: Fei-Fei and Karpathy

Example of a neural network with three inputs, a single hidden layer of four neurons, and an output layer of two neurons

Neural Network

A neural network comprises neurons connected in an acyclic graph

The outputs of neurons can become inputs to other neurons

Neural networks typically contain multiple **layers** of neurons

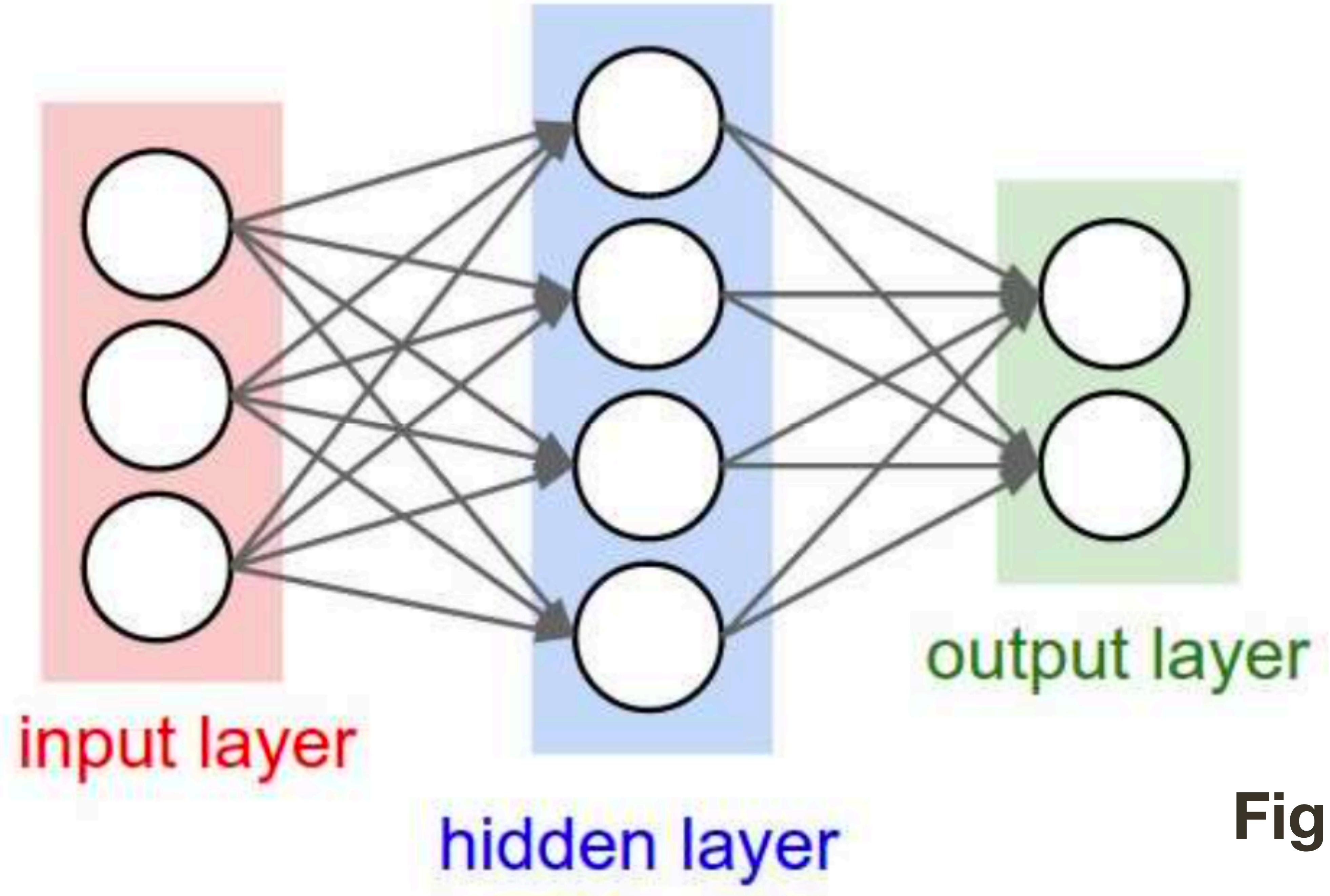


Figure credit: Fei-Fei and Karpathy

Example of a neural network with three inputs, a single hidden layer of four neurons, and an output layer of two neurons

Neural Network **Intuition**

Question: What is a Neural Network?

Answer: Complex mapping from an input (vector) to an output (vector)

* slide from Marc'Aurelio Renzato

Neural Network **Intuition**

Question: What is a Neural Network?

Answer: Complex mapping from an input (vector) to an output (vector)

Question: What class of functions should be considered for this mapping?

Answer: Compositions of simpler functions (a.k.a. layers)? We will talk more about what specific functions next …

* slide from Marc’Aurelio Renzato

Neural Network **Intuition**

Question: What is a Neural Network?

Answer: Complex mapping from an input (vector) to an output (vector)

Question: What class of functions should be considered for this mapping?

Answer: Compositions of simpler functions (a.k.a. layers)? We will talk more about what specific functions next …

Question: What does a hidden unit do?

Answer: It can be thought of as classifier or a feature.

* slide from Marc'Aurelio Renzato

Neural Network **Intuition**

Question: What is a Neural Network?

Answer: Complex mapping from an input (vector) to an output (vector)

Question: What class of functions should be considered for this mapping?

Answer: Compositions of simpler functions (a.k.a. layers)? We will talk more about what specific functions next …

Question: What does a hidden unit do?

Answer: It can be thought of as classifier or a feature.

Question: Why have many layers?

Answer: 1) More layers = more complex functional mapping

2) More efficient due to distributed representation

* slide from Marc'Aurelio Renzato

Neural Network

A neural network comprises neurons connected in an acyclic graph

The outputs of neurons can become inputs to other neurons

Neural networks typically contain multiple layers of neurons

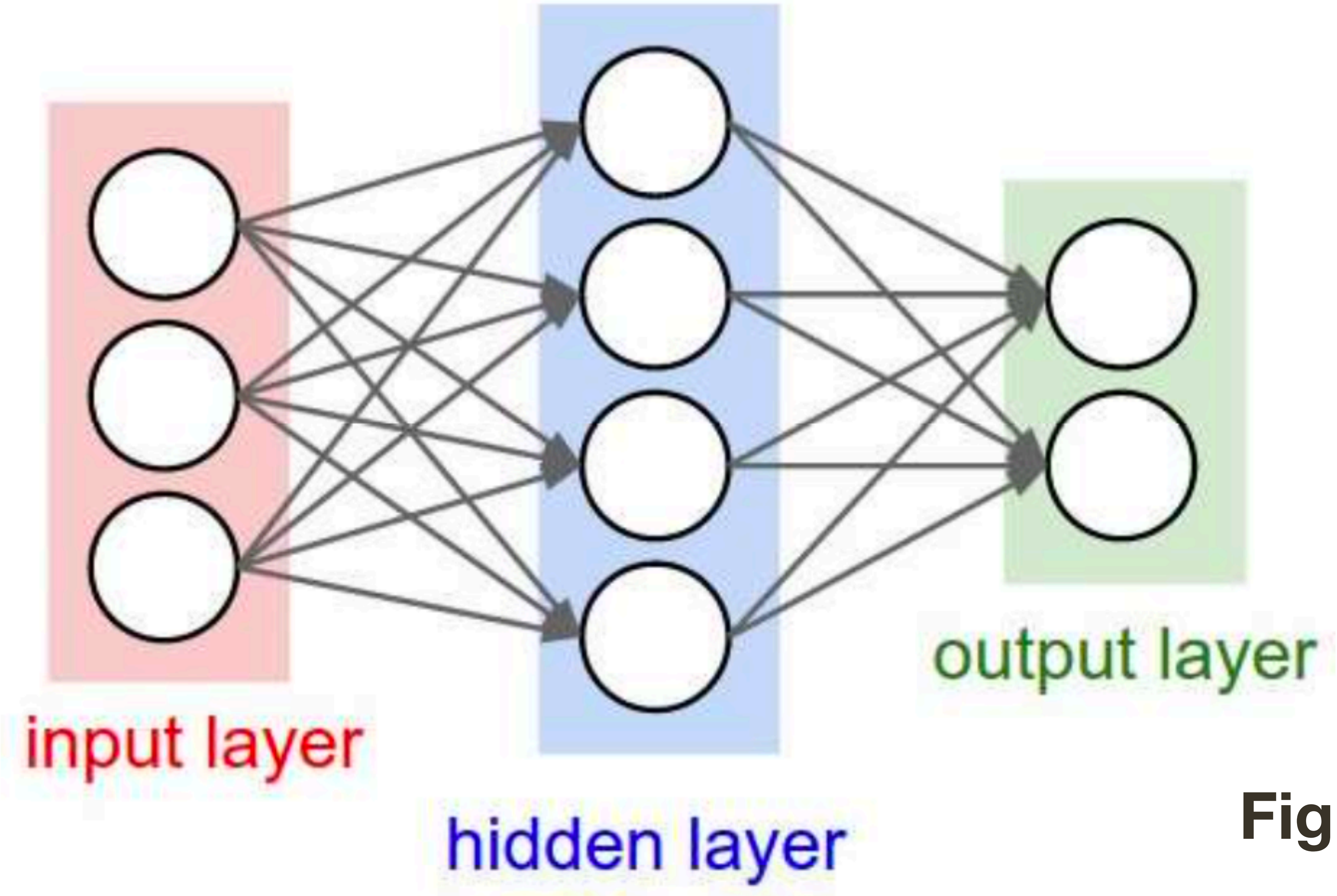


Figure credit: Fei-Fei and Karpathy

Example of a neural network with three inputs, a single hidden layer of four neurons, and an output layer of two neurons

Neural Network

Note: each neuron will have its own vector of weights and a bias, its easier to think of all neurons in a layer as a single entity with a matrix of weights (size = number of inputs x number of neurons) and a vector of biases (size = number of neurons)

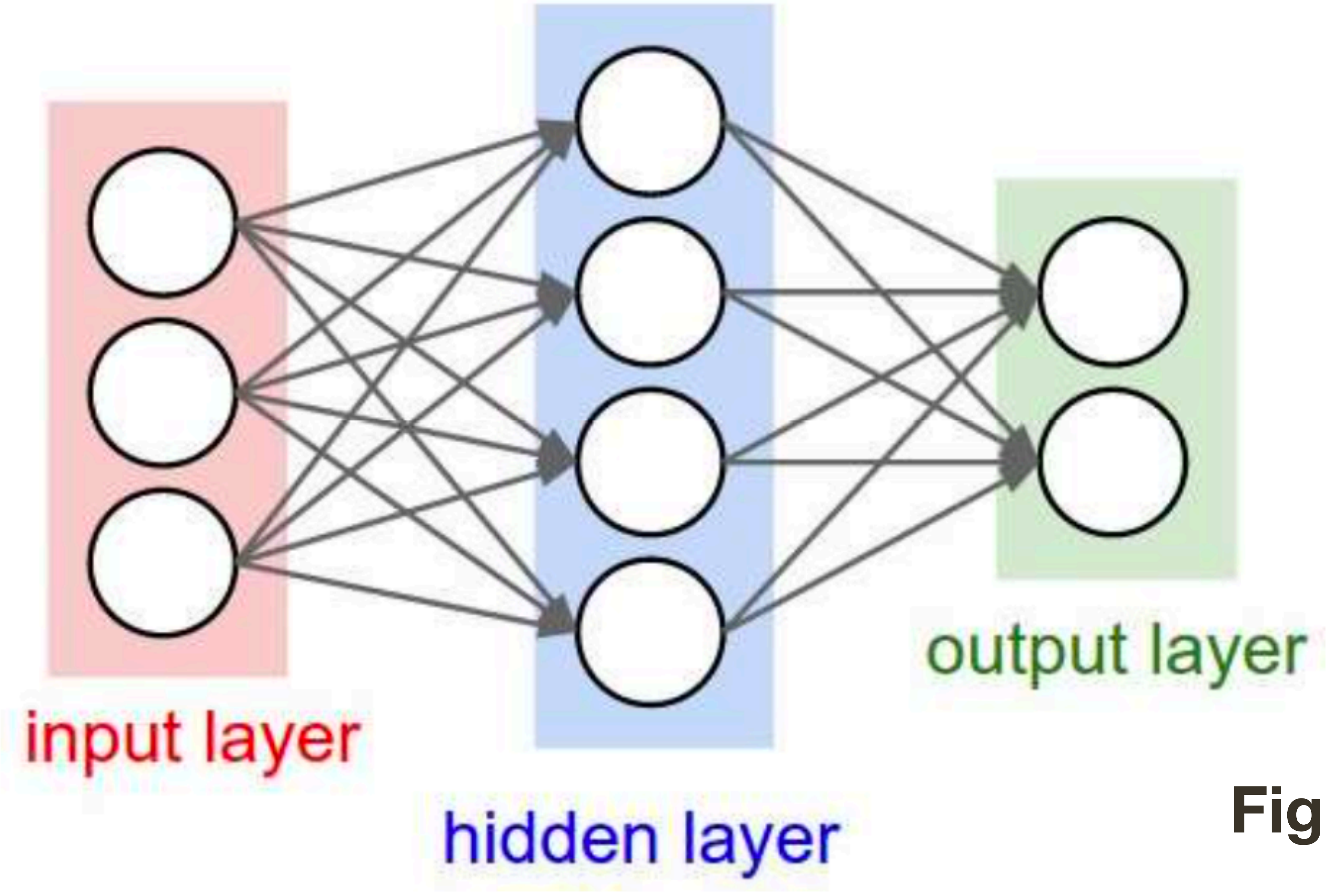


Figure credit: Fei-Fei and Karpathy

Neural Network

Note: each neuron will have its own vector of weights and a bias, its easier to think of all neurons in a layer as a single entity with a matrix of weights (size = number of inputs x number of neurons) and a vector of biases (size = number of neurons)

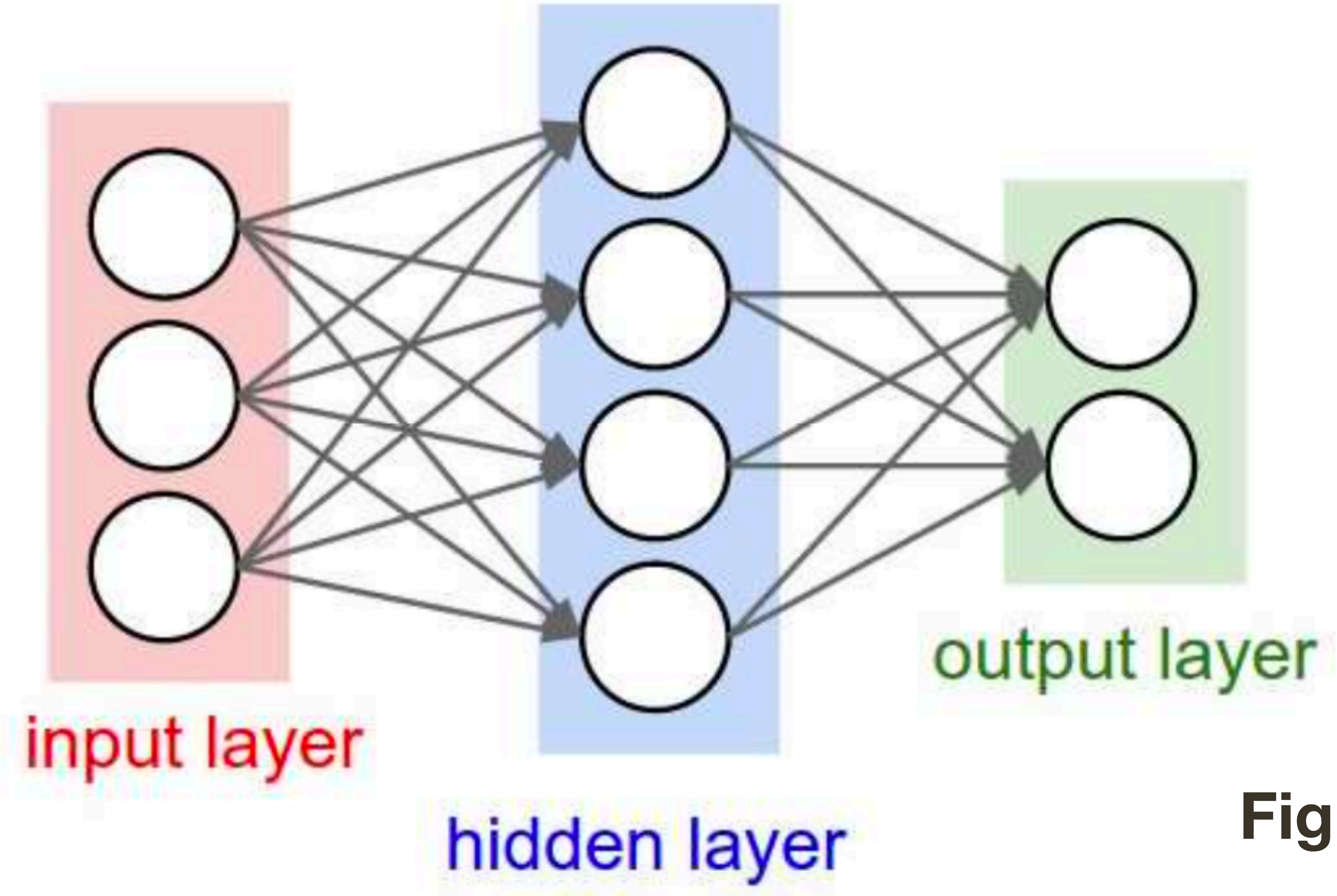
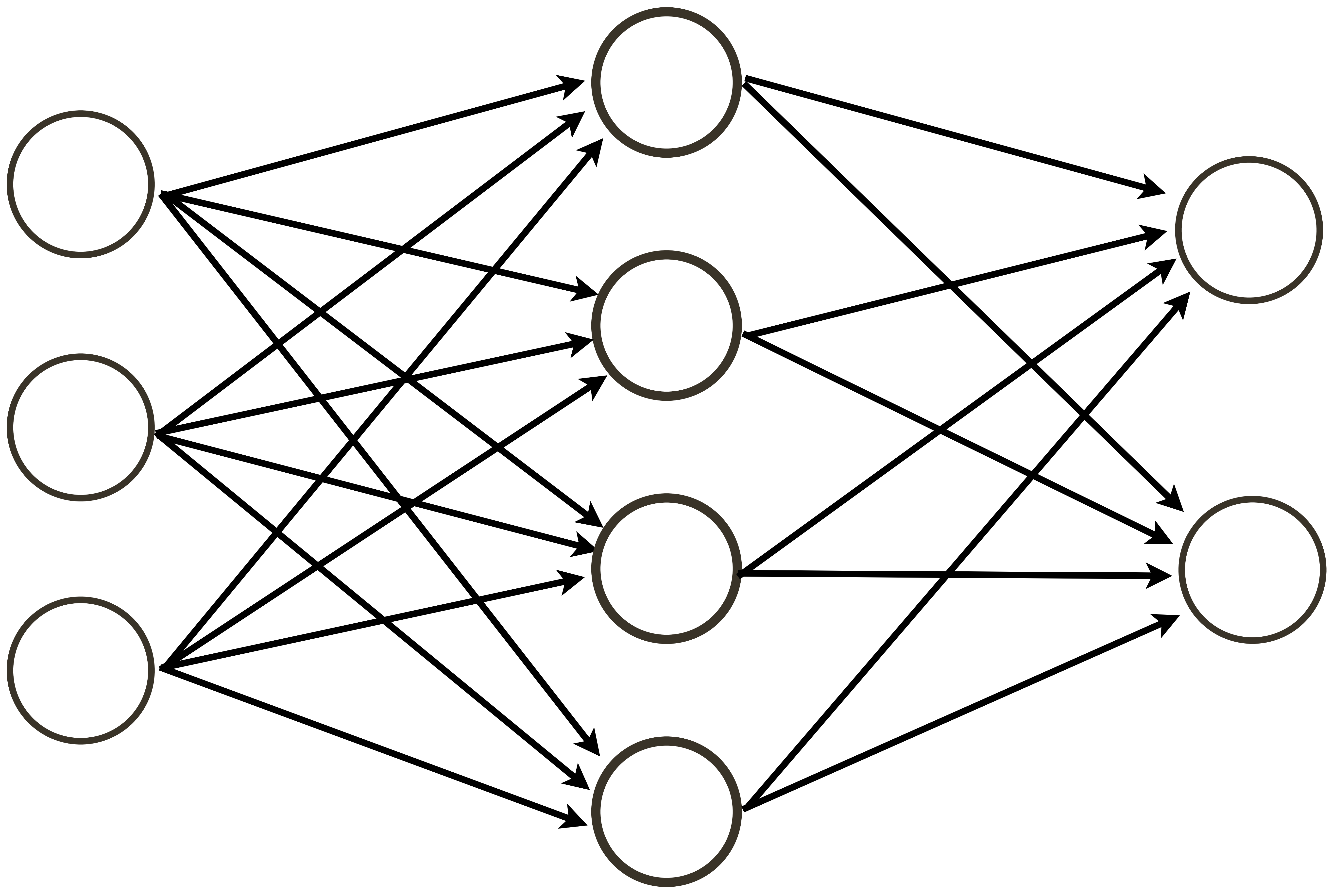


Figure credit: Fei-Fei and Karpathy

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times 4)}\sigma\left(\mathbf{W}_1^{(4\times 3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

Activation Function

Why can’t we have **linear** activation functions? Why have non-linear activations?



Activation Function

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times4)}\sigma\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

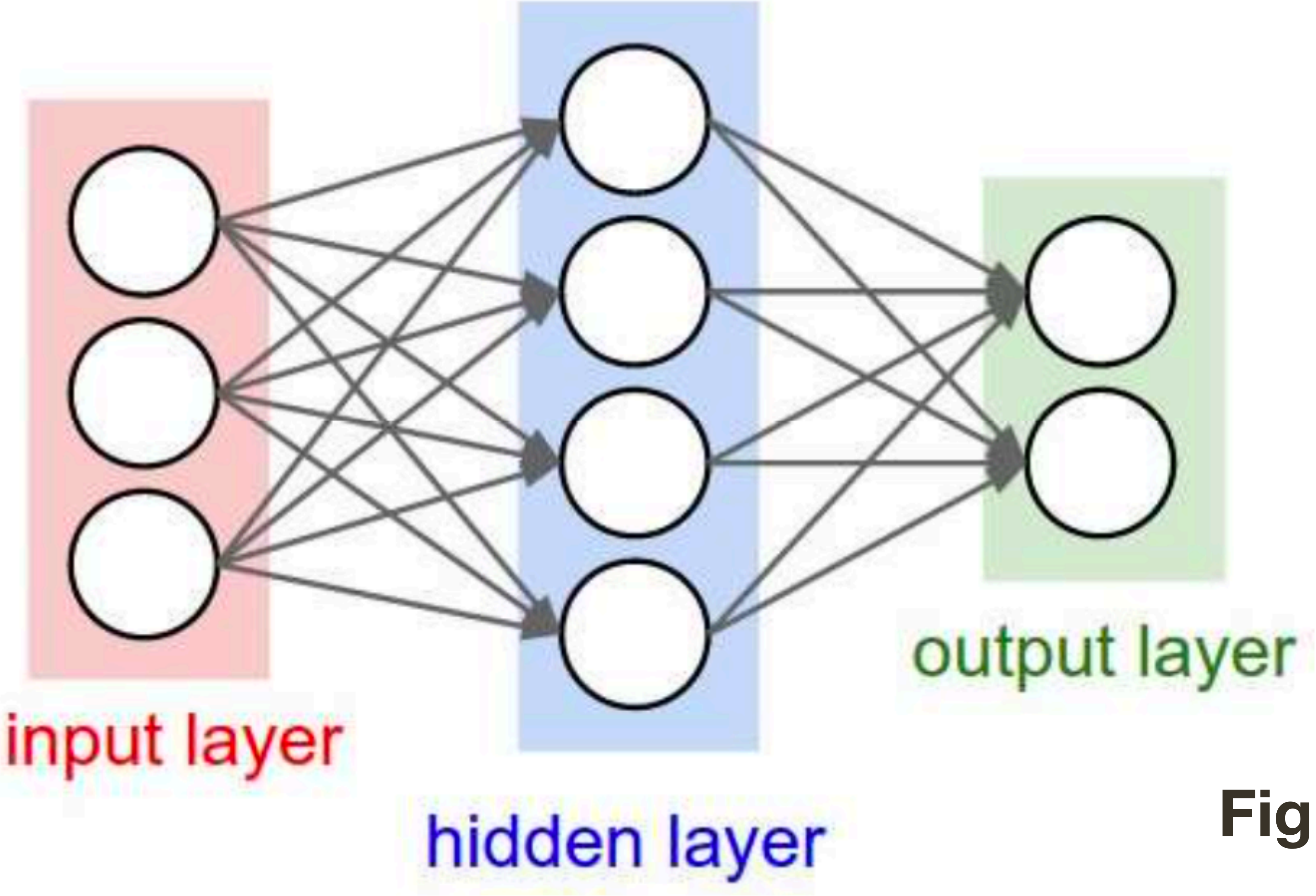


Figure credit: Fei-Fei and Karpathy

Activation Function

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times4)}\sigma\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$
$$= \mathbf{W}_2^{(2\times4)}\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}$$

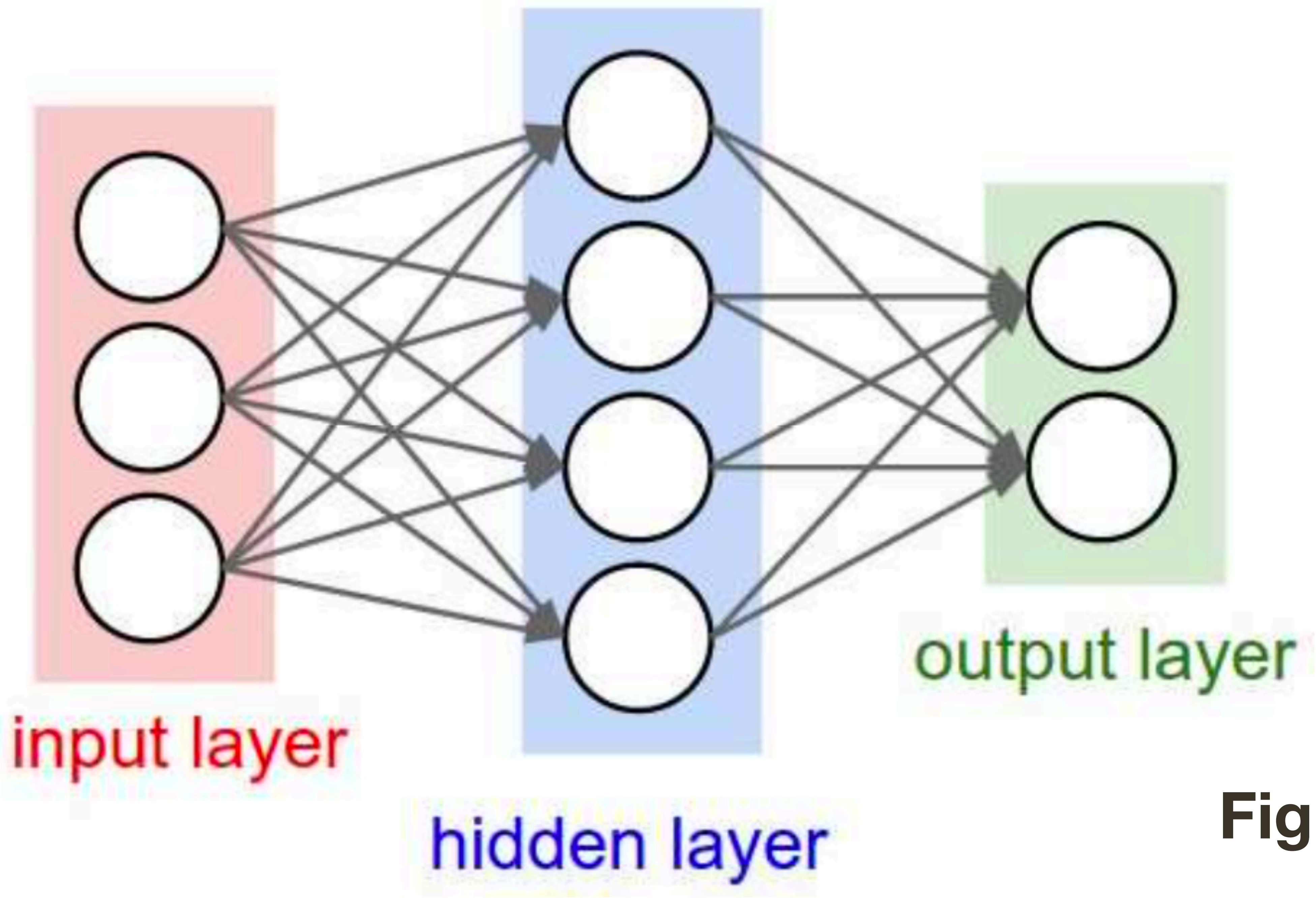


Figure credit: Fei-Fei and Karpathy

Activation Function

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times 4)}\sigma\left(\mathbf{W}_1^{(4\times 3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

$$= \mathbf{W}_2^{(2\times 4)}\left(\mathbf{W}_1^{(4\times 3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}$$

$$= \mathbf{W}_2^{(2\times 4)}\mathbf{W}_1^{(4\times 3)}\mathbf{x} + \mathbf{W}_2^{(2\times 4)}\mathbf{b}_1^{(4)} + \mathbf{b}_2^{(2)}$$

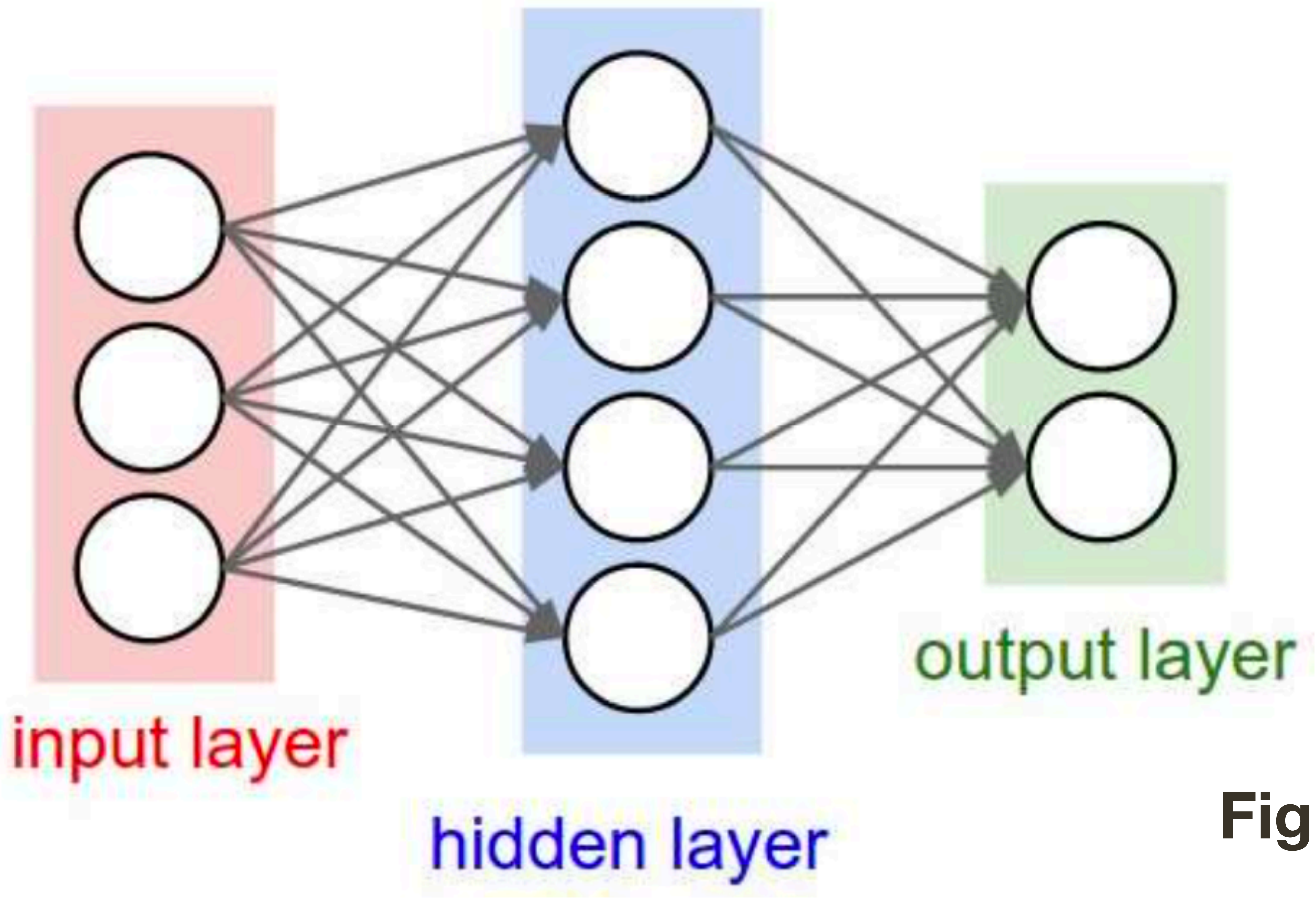


Figure credit: Fei-Fei and Karpathy

Activation Function

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times4)}\sigma\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

$$= \mathbf{W}_2^{(2\times4)}\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}$$

$$= \underbrace{\mathbf{W}_2^{(2\times4)}\mathbf{W}_1^{(4\times3)}}_{\mathbf{W}_*^{(2\times3)}}\mathbf{x} + \underbrace{\mathbf{W}_2^{(2\times4)}\mathbf{b}_1^{(4)} + \mathbf{b}_2^{(2)}}_{\mathbf{b}^{(2)}}$$

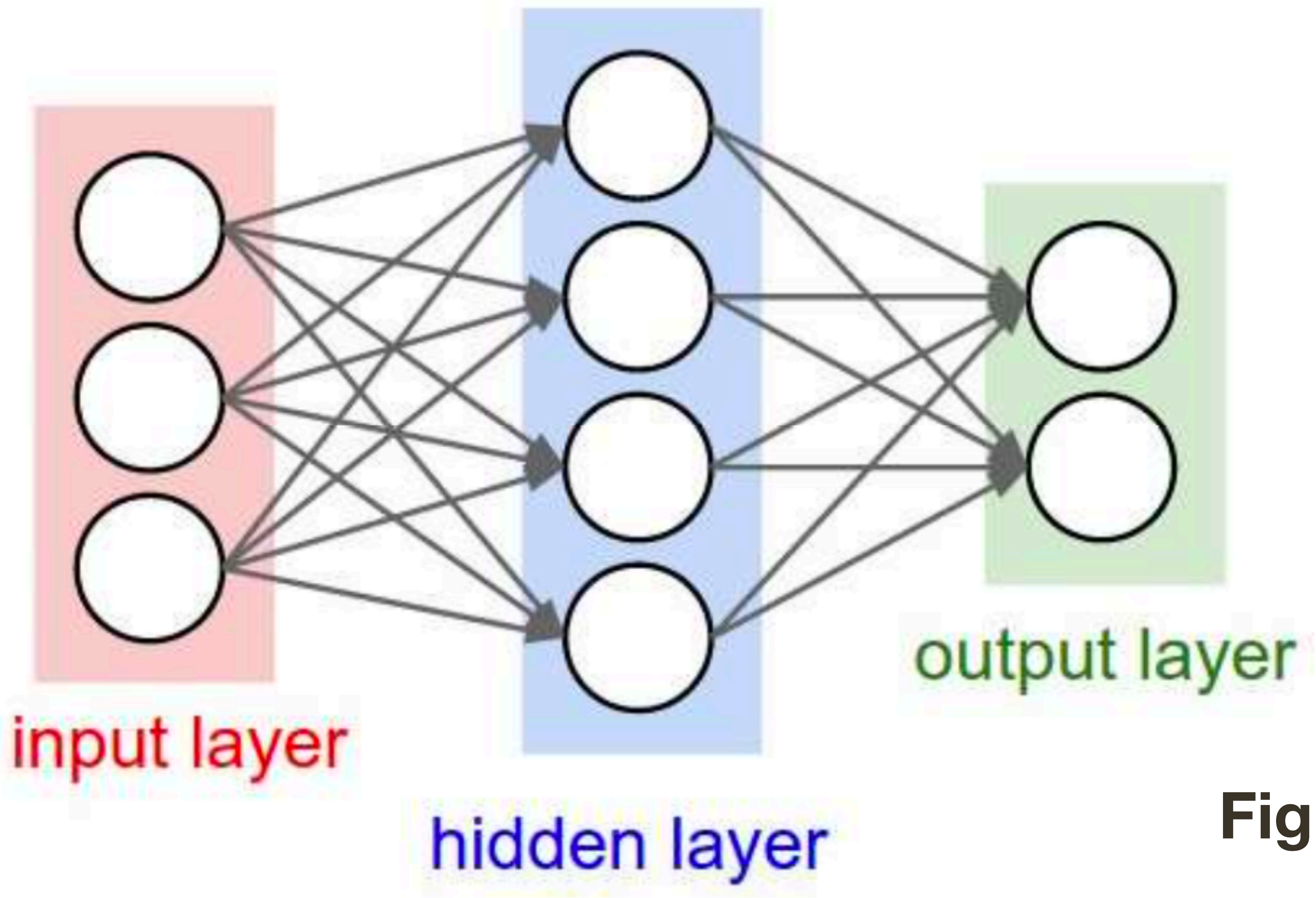
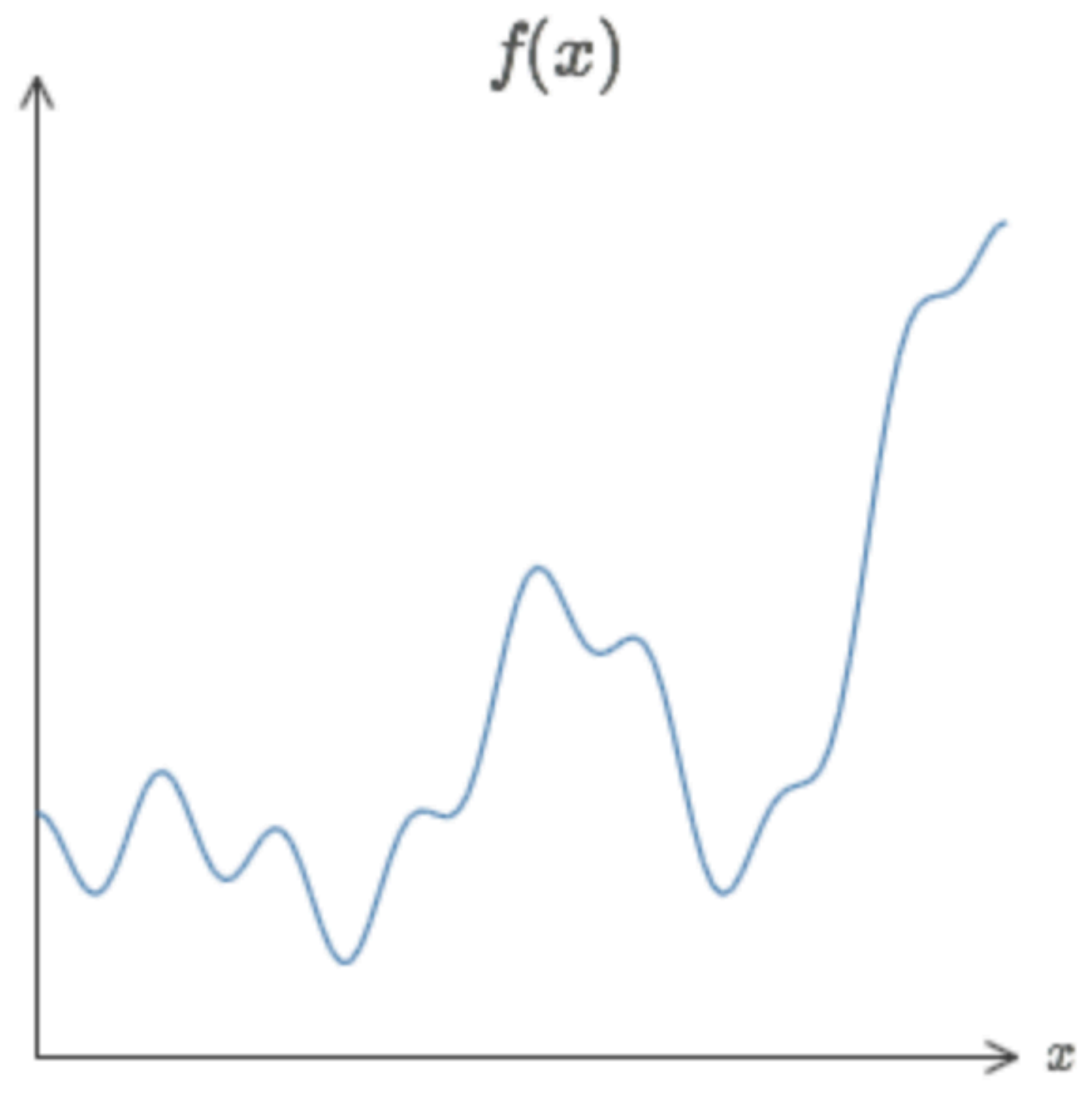


Figure credit: Fei-Fei and Karpathy

Light Theory: Neural Network as Universal Approximator

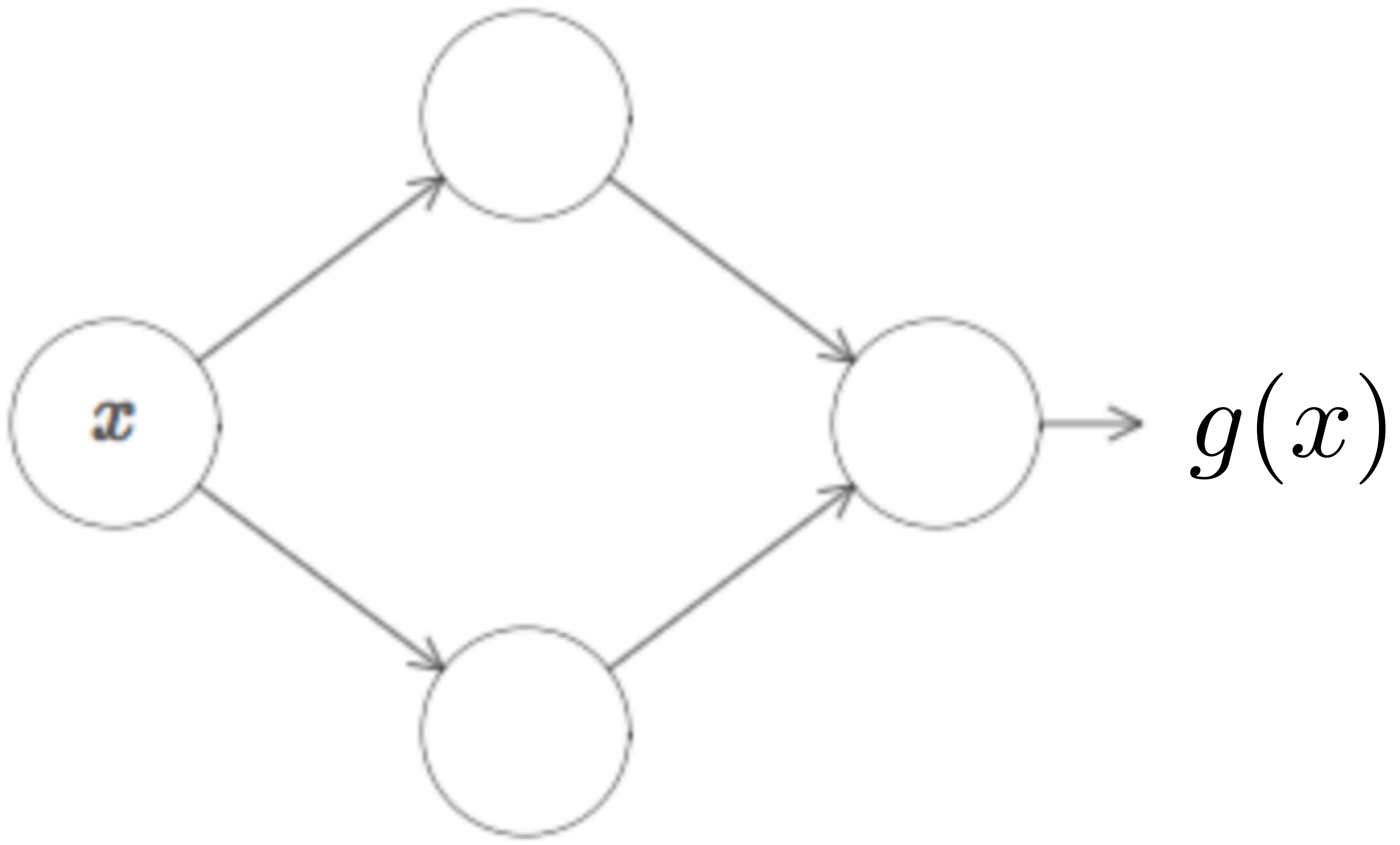
Neural network can arbitrarily approximate *any* **continuous** function for every value of possible inputs

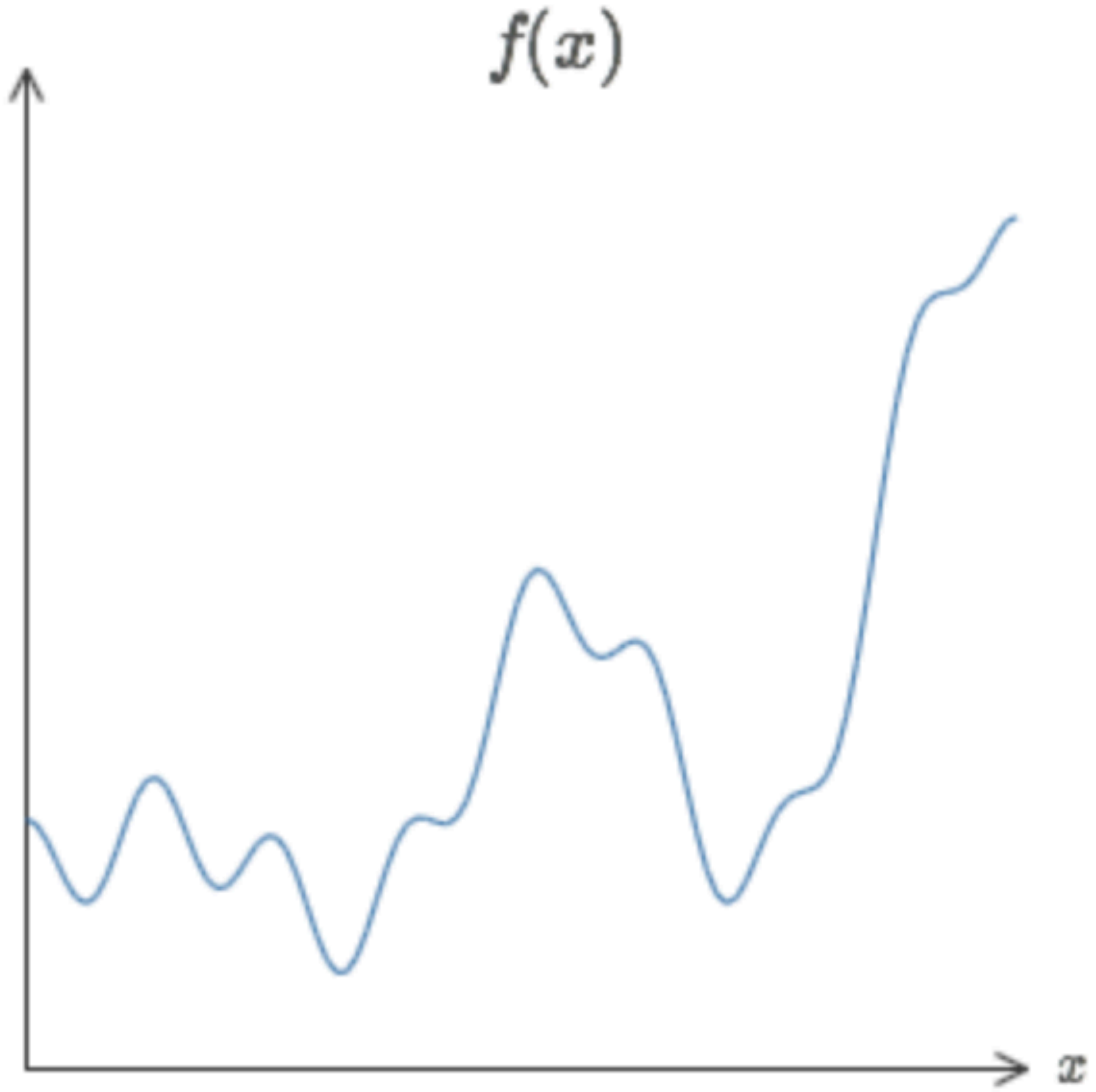


*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

Light Theory: Neural Network as Universal Approximator

Neural network can arbitrarily approximate *any* **continuous** function for every value of possible inputs



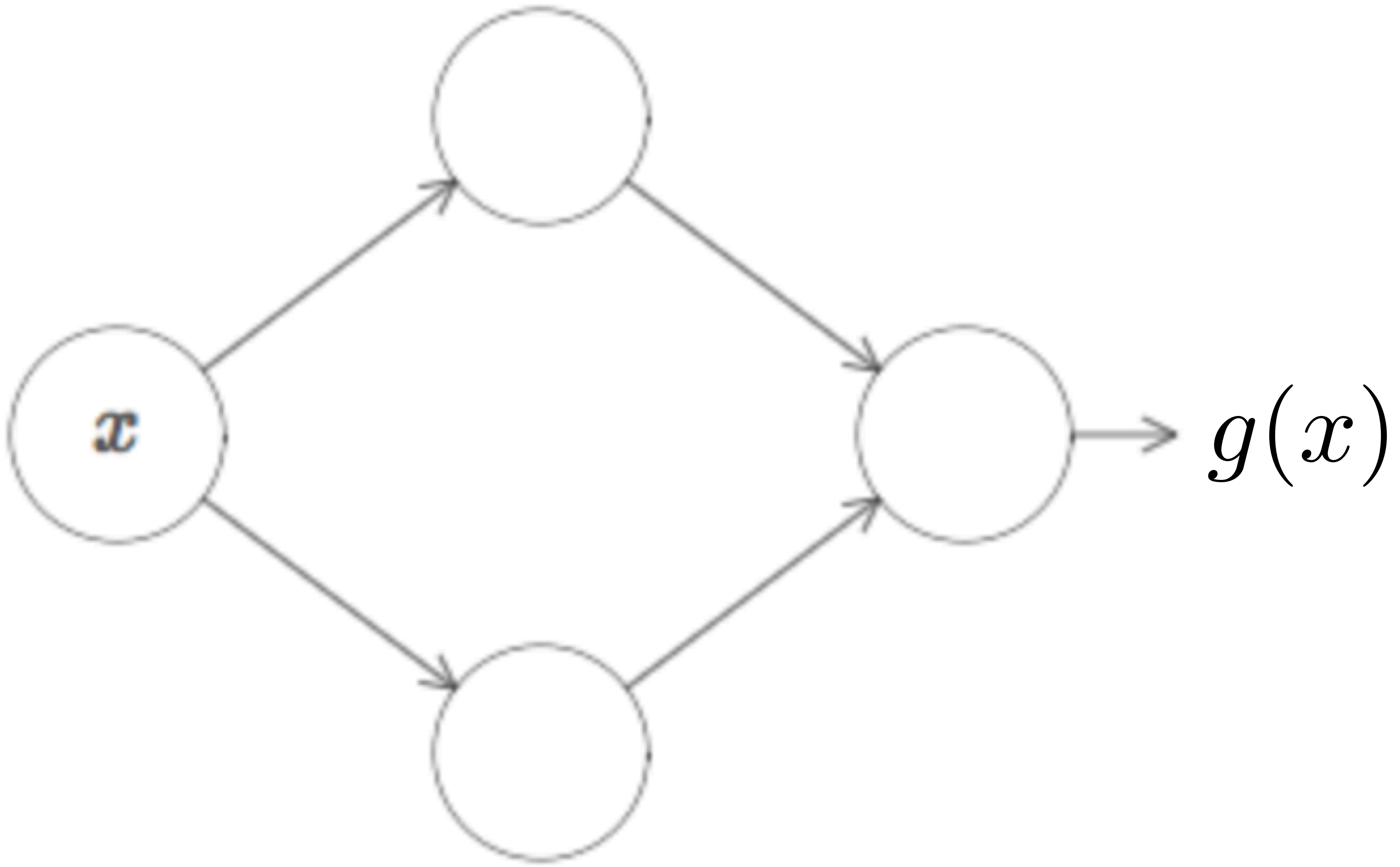


The guarantee is that by using enough hidden neurons we can always find a neural network whose output $g(x)$ satisfies $|g(x) - f(x)| < \epsilon$ for an arbitrarily small ϵ

*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

Light Theory: Neural Network as Universal Approximator

Lets start with a simple network: one hidden layer with two hidden neurons and a single output layer with one neuron (with sigmoid activations)

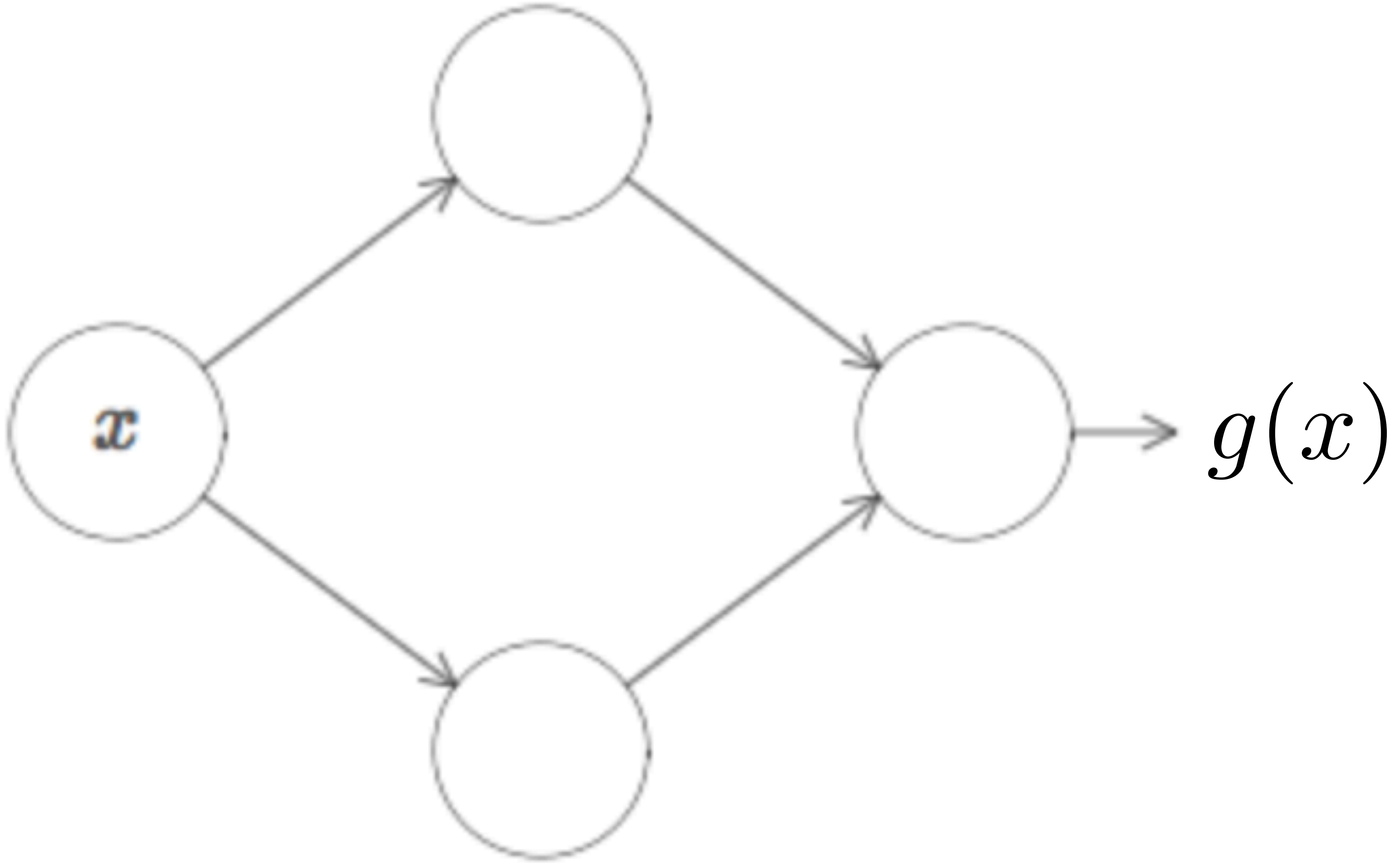


*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

Light Theory: Neural Network as Universal Approximator

Lets start with a simple network: one hidden layer with two hidden neurons and a single output layer with one neuron (with sigmoid activations)

Let's look at output of this (hidden) neuron as a function of parameters (weight, bias)

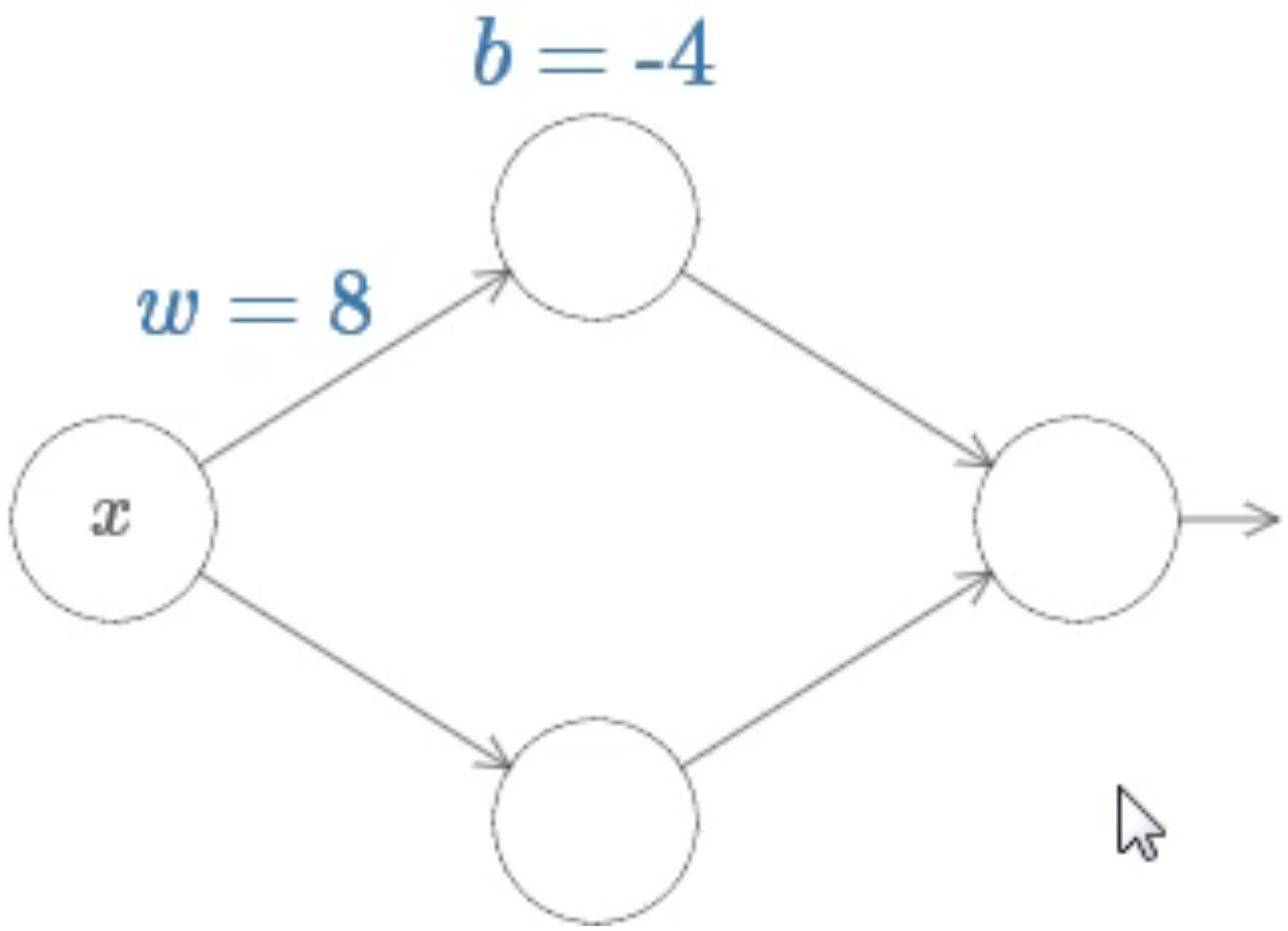


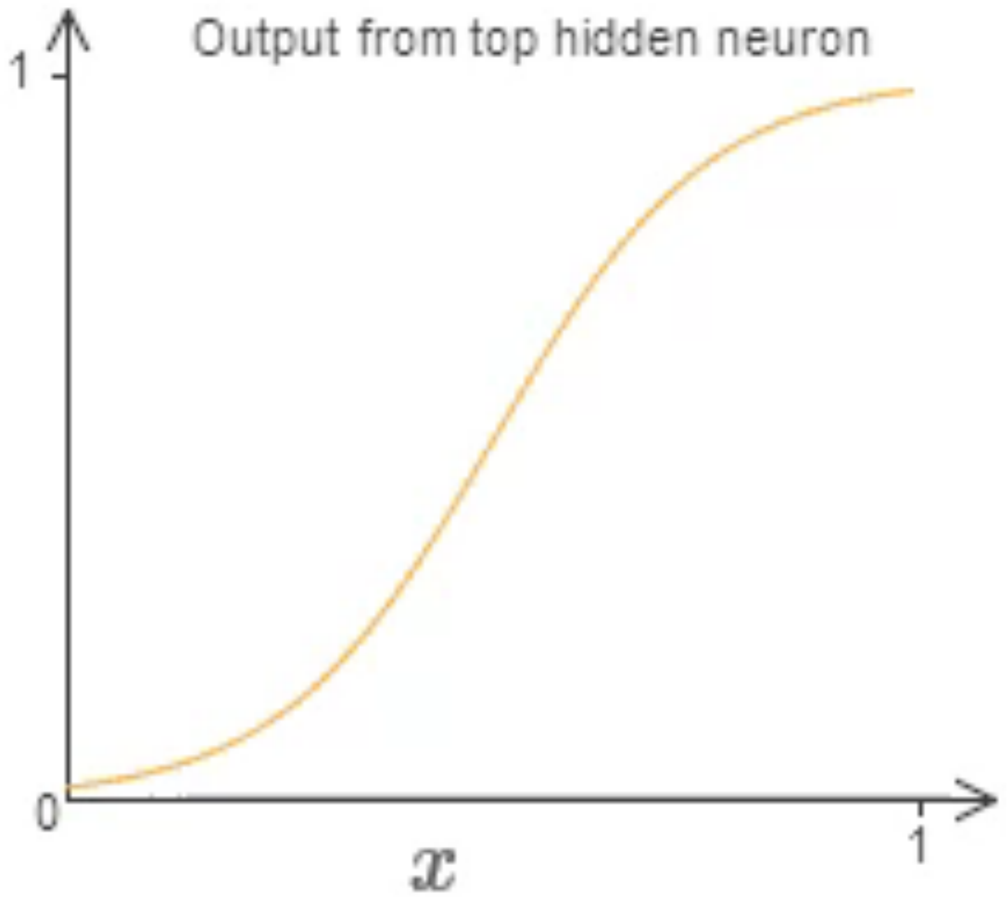
*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

Light Theory: Neural Network as Universal Approximator

Lets start with a simple network: one hidden layer with two hidden neurons and a single output layer with one neuron (with sigmoid activations)

Let's look at output of this (hidden) neuron as a function of parameters (weight, bias)



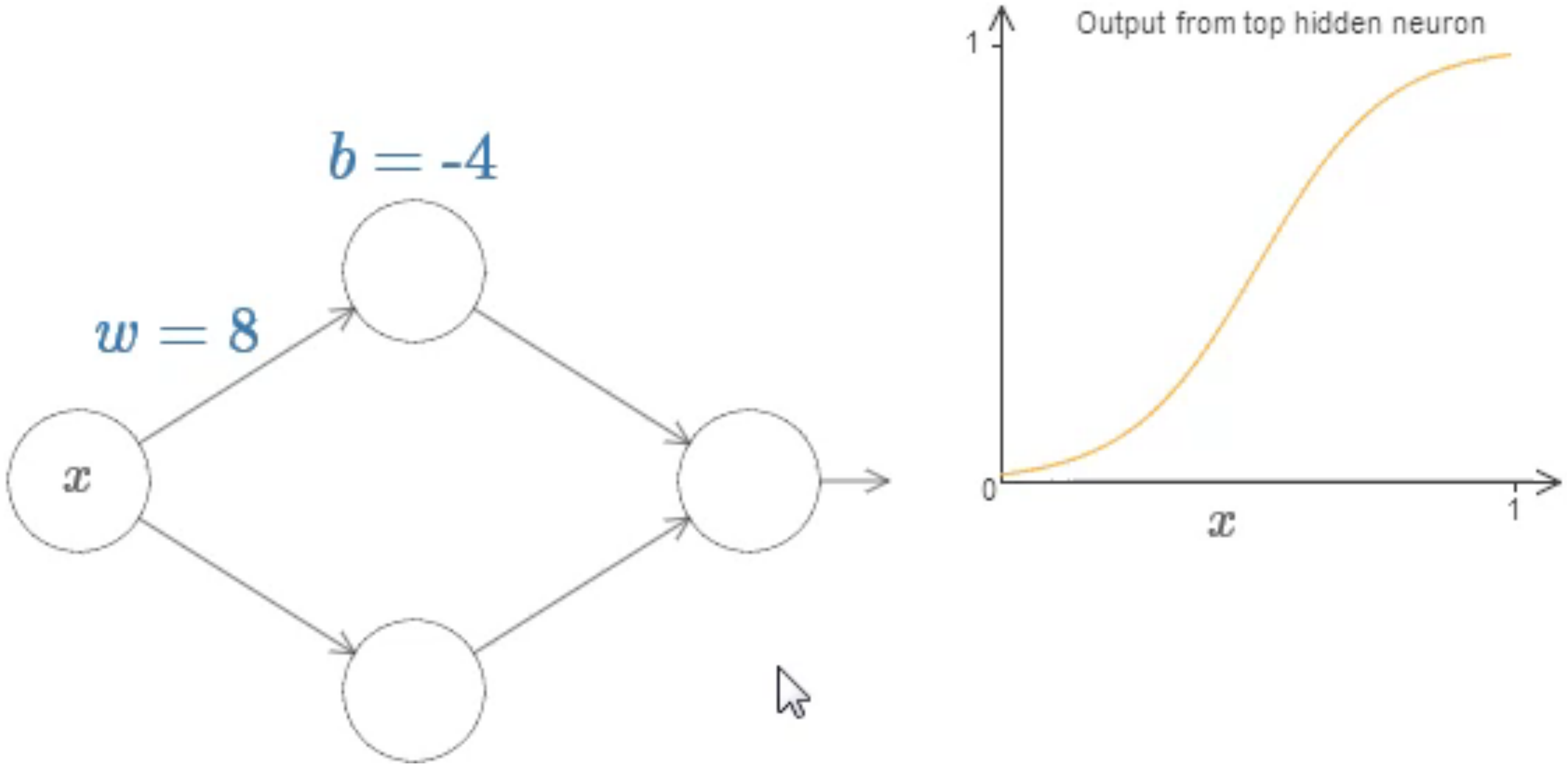


*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

Light Theory: Neural Network as Universal Approximator

Lets start with a simple network: one hidden layer with two hidden neurons and a single output layer with one neuron (with sigmoid activations)

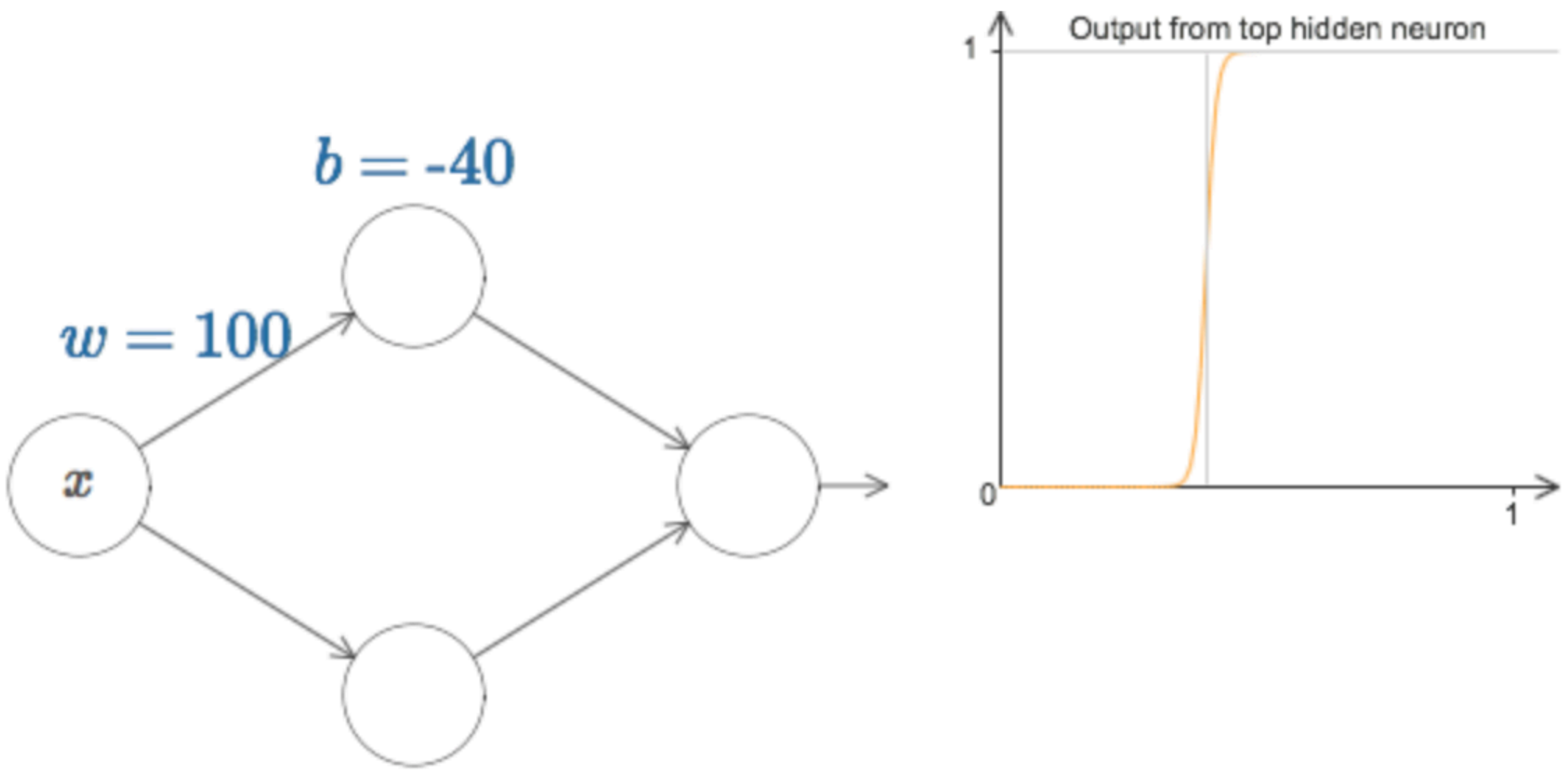
Let's look at output of this (hidden) neuron as a function of parameters (weight, bias)



*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

Light Theory: Neural Network as Universal Approximator

By dialing up the weight (e.g. $w = 999$) we can actually create a “step” function

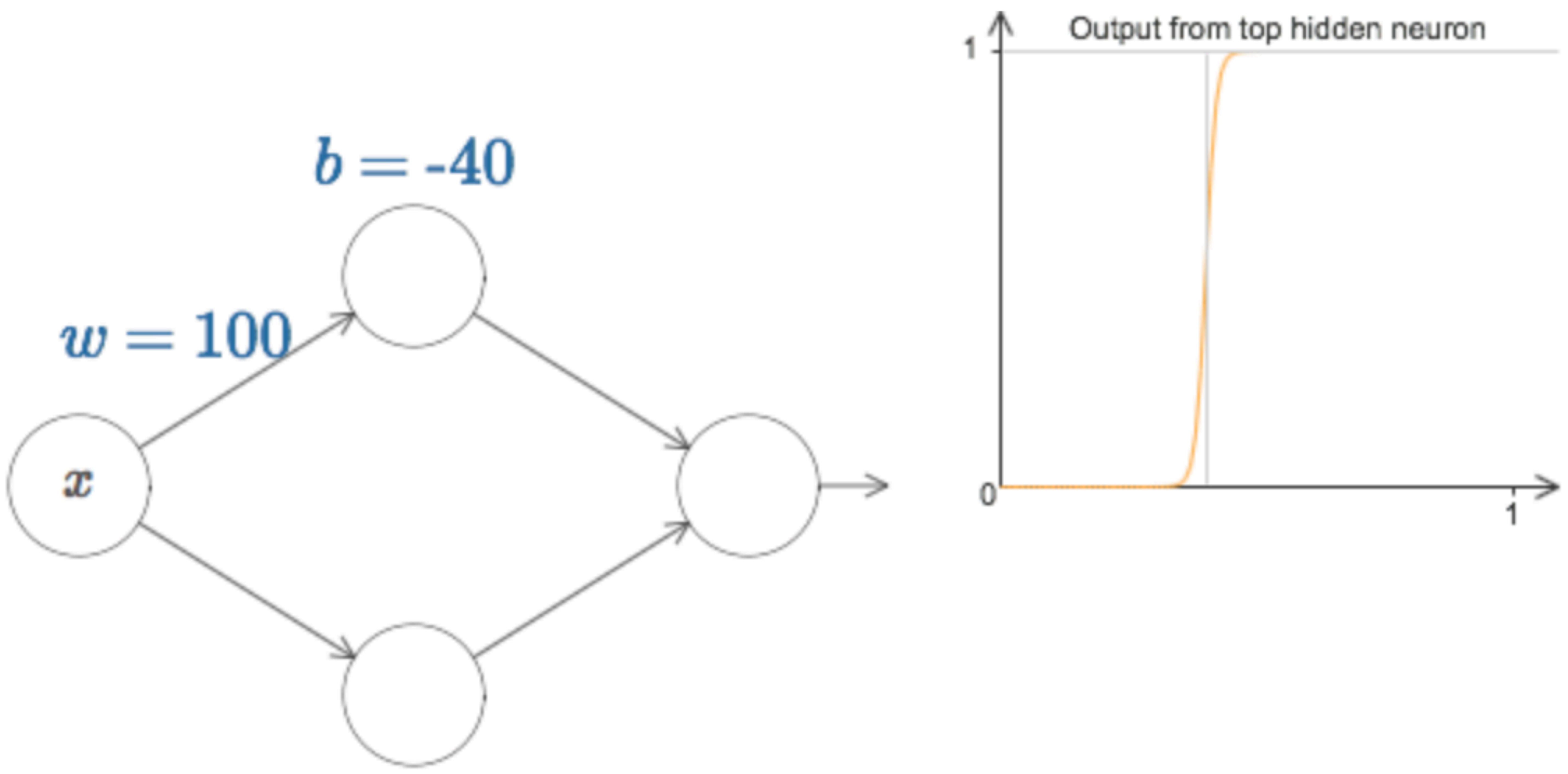


*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

Light Theory: Neural Network as Universal Approximator

By dialing up the weight (e.g. $w = 999$) we can actually create a “step” function

It is easier to work with sums of step functions, so we can assume that every neuron outputs a step function.



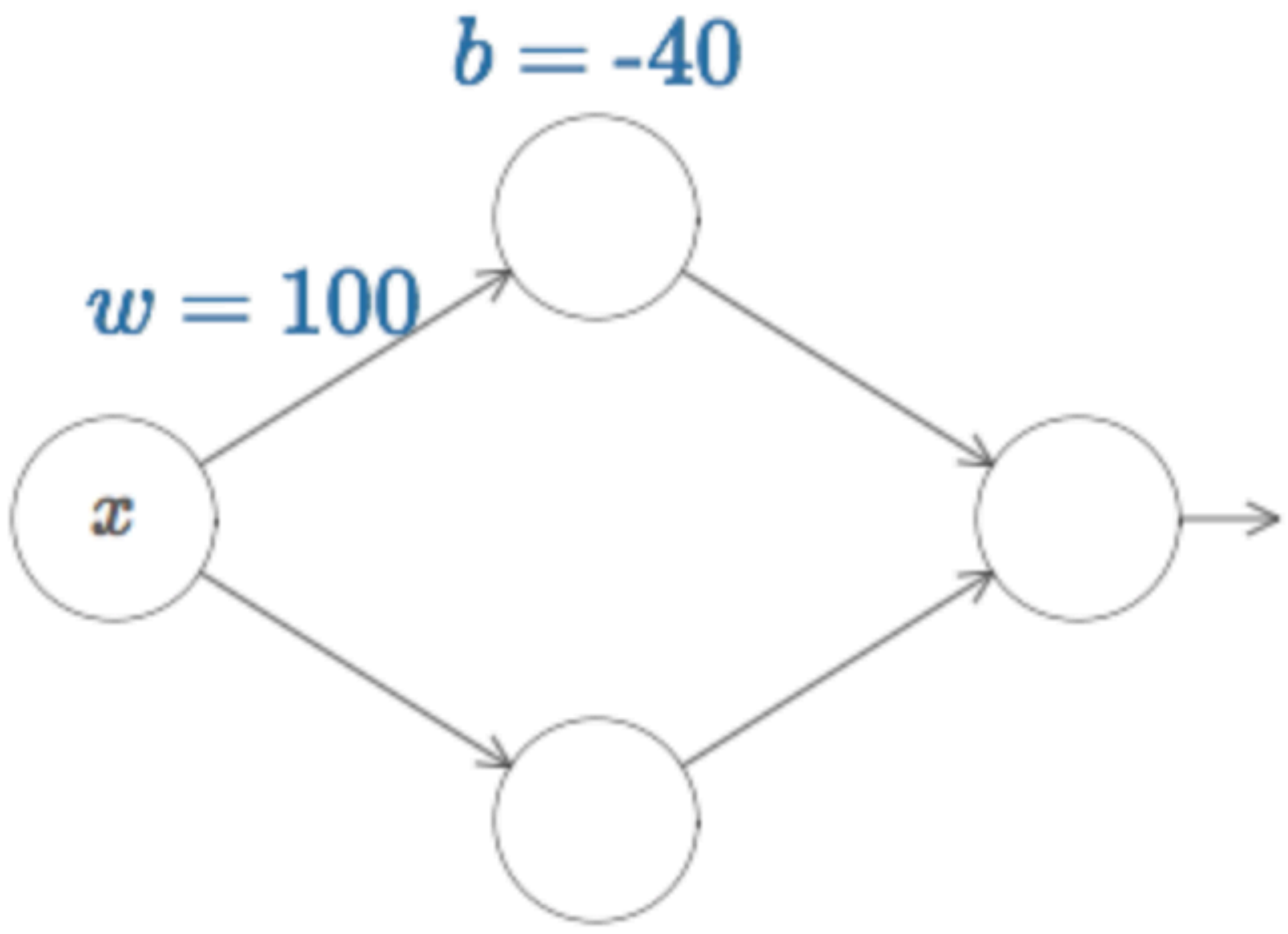
*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

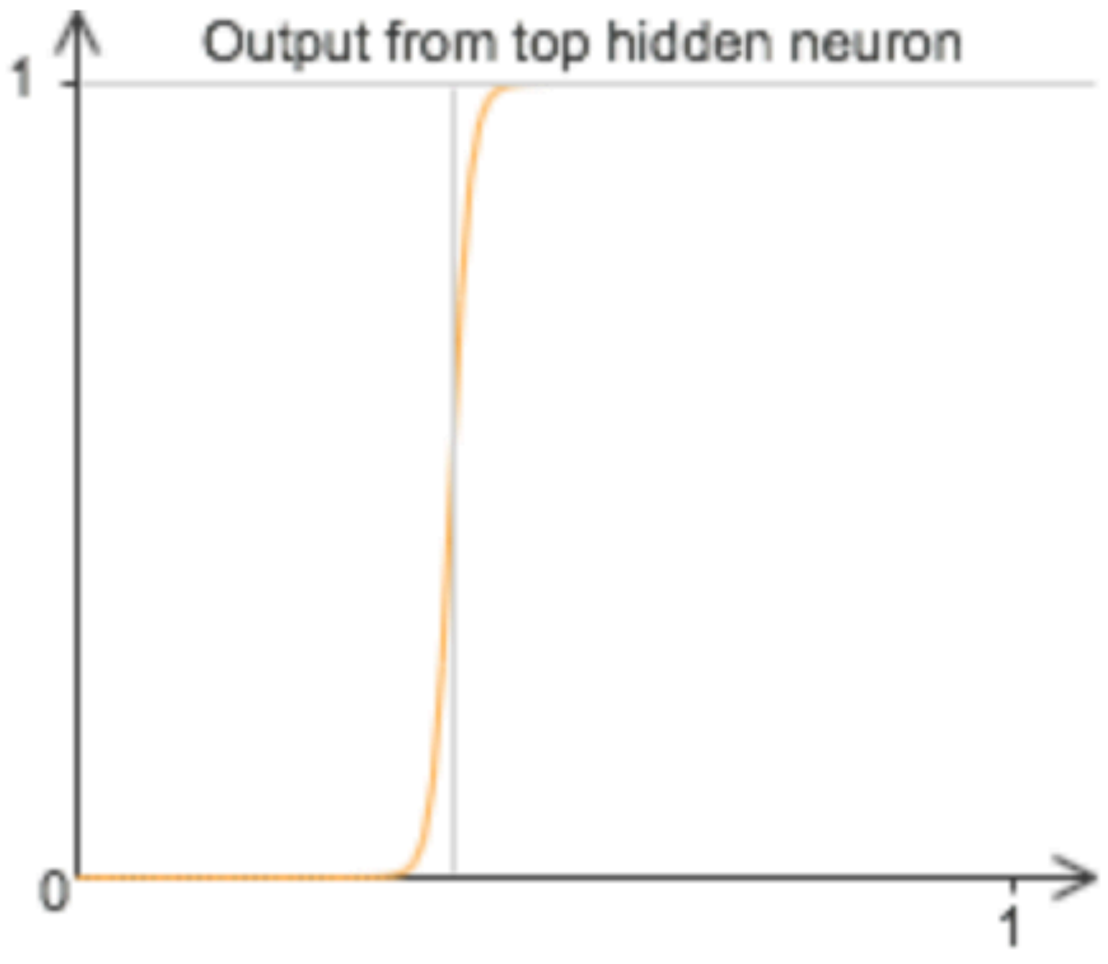
Light Theory: Neural Network as Universal Approximator

By dialing up the weight (e.g. $w = 999$) we can actually create a "step" function

It is easier to work with sums of step functions, so we can assume that every neuron outputs a step function.

Location of the step?





*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

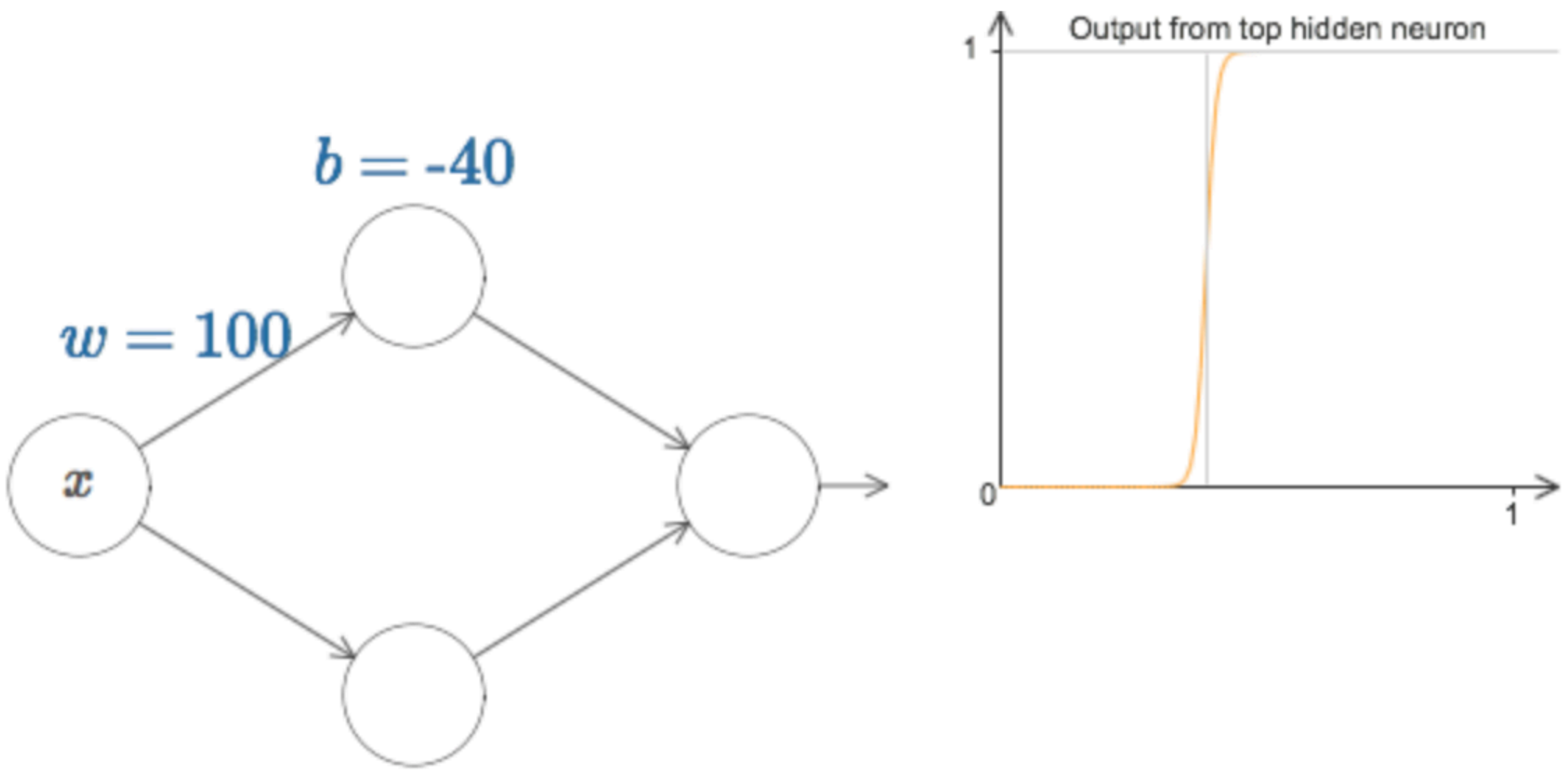
Light Theory: Neural Network as Universal Approximator

By dialing up the weight (e.g. $w = 999$) we can actually create a “step” function

It is easier to work with sums of step functions, so we can assume that every neuron outputs a step function.

Location of the step?

$$s = -\frac{b}{w}$$



*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

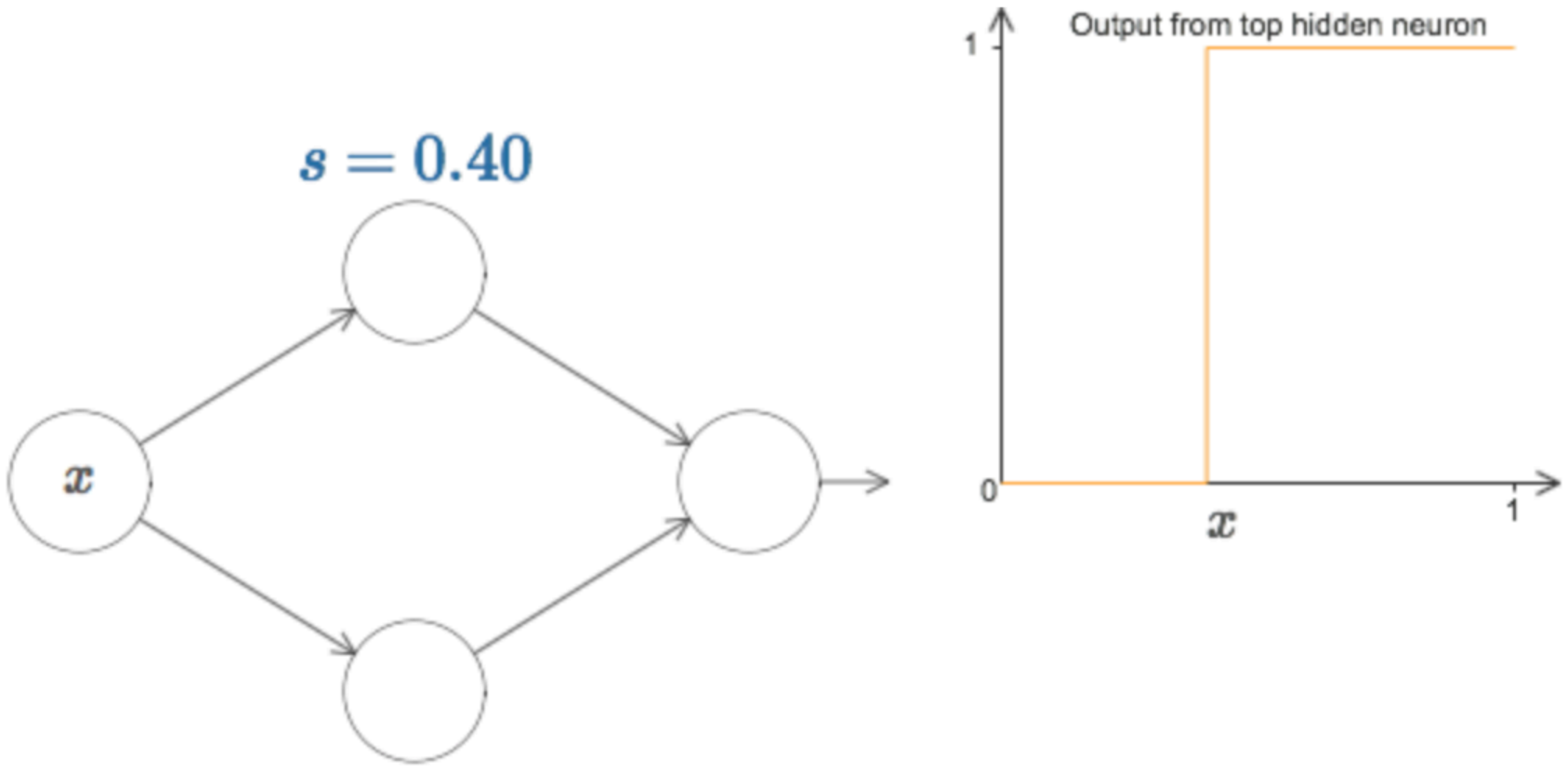
Light Theory: Neural Network as Universal Approximator

By dialing up the weight (e.g. $w = 999$) we can actually create a "step" function

It is easier to work with sums of step functions, so we can assume that every neuron outputs a step function

Location of the step?

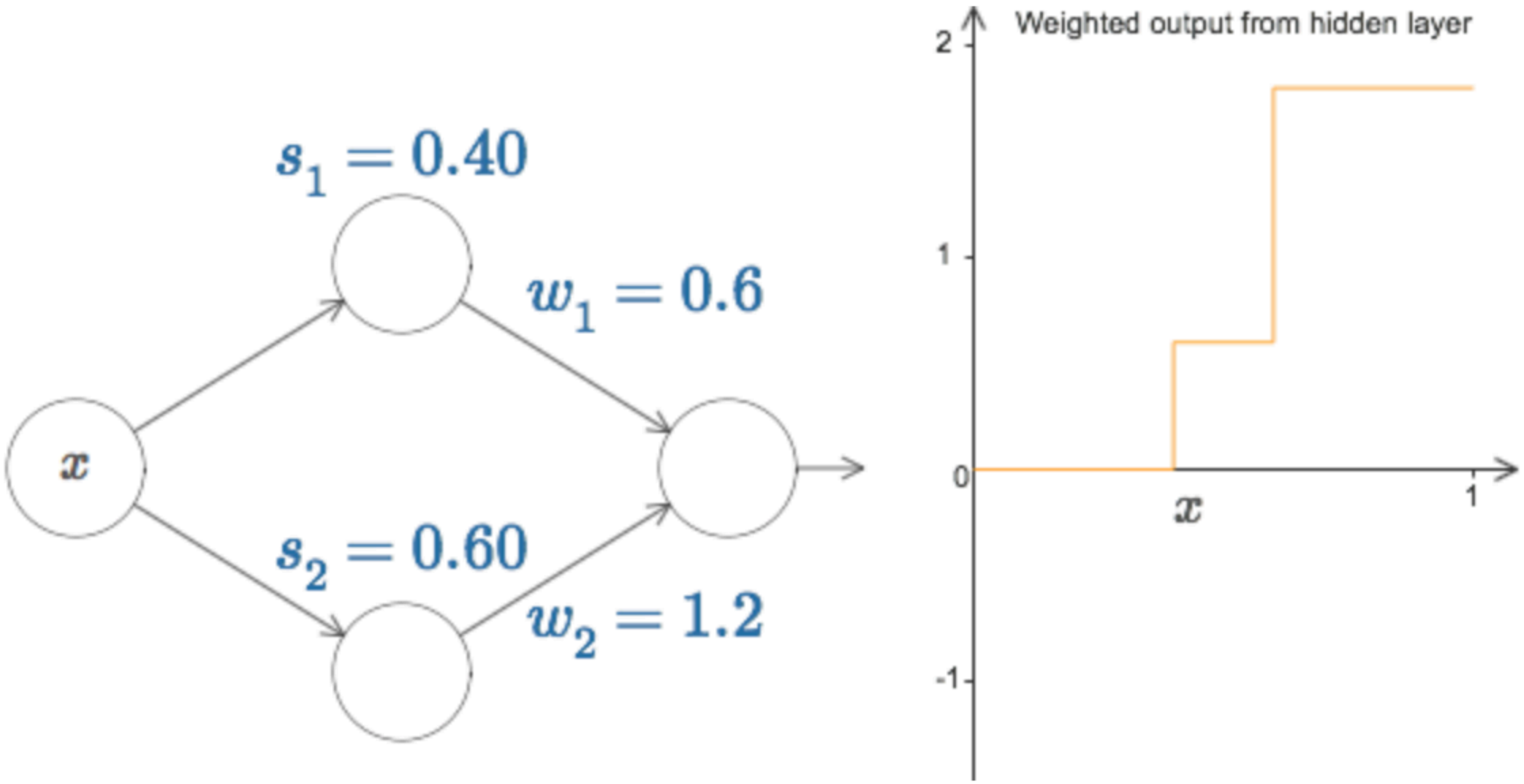
$$s = -\frac{b}{w}$$



*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

Light Theory: Neural Network as Universal Approximator

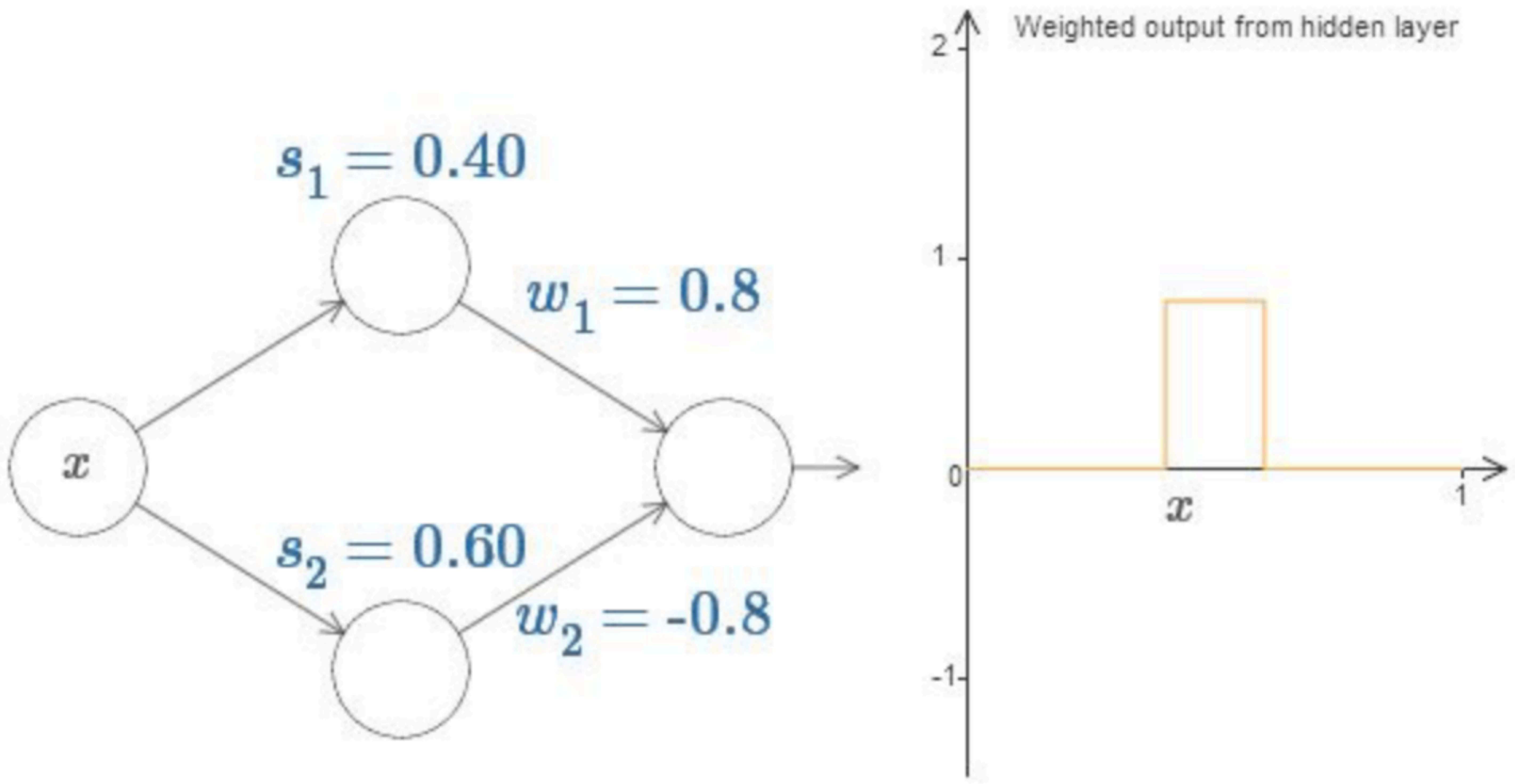
The output neuron is a **weighted combination of step functions** (assuming bias for that layer is 0)



*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

Light Theory: Neural Network as Universal Approximator

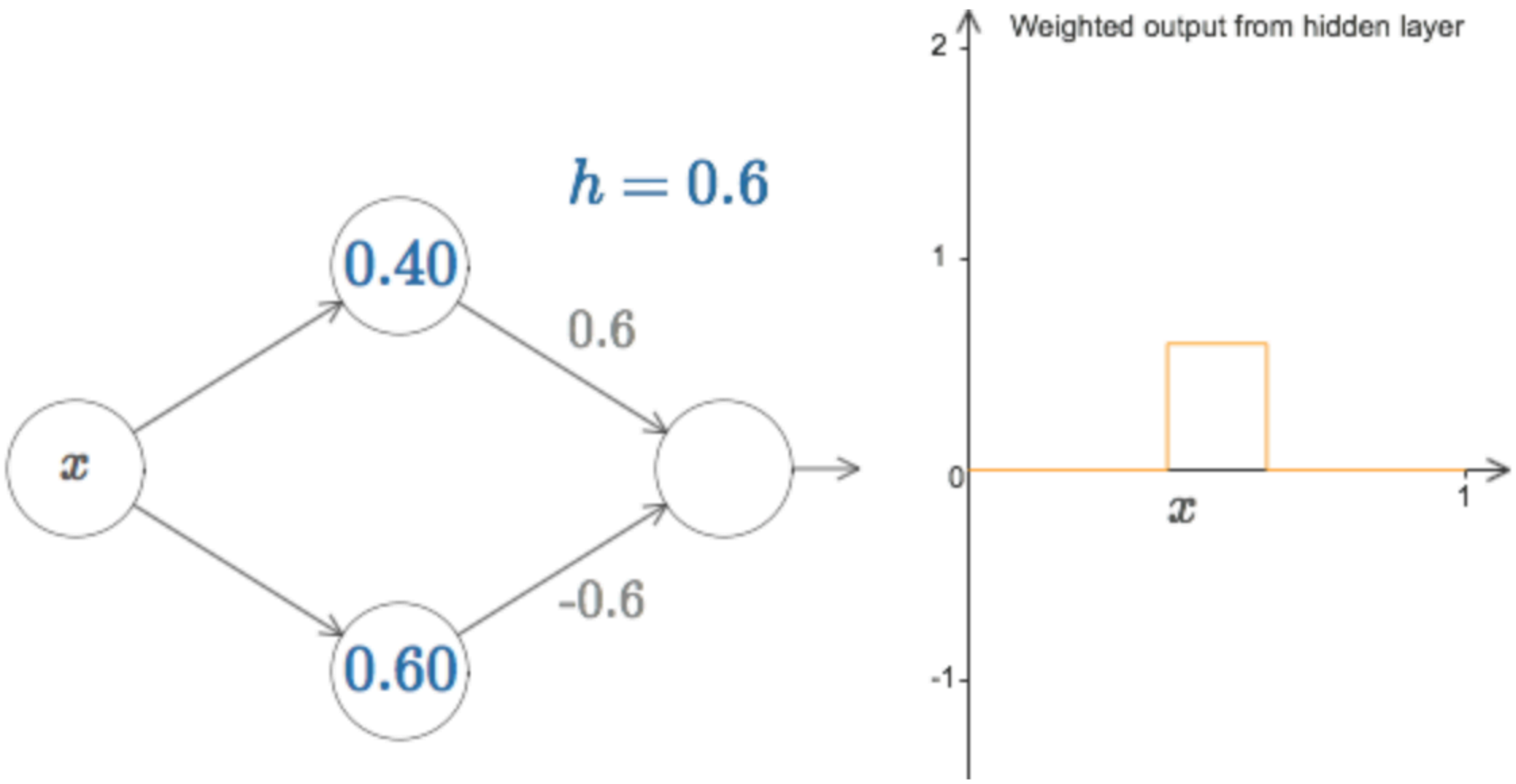
The output neuron is a **weighted combination of step functions** (assuming bias for that layer is 0)



*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

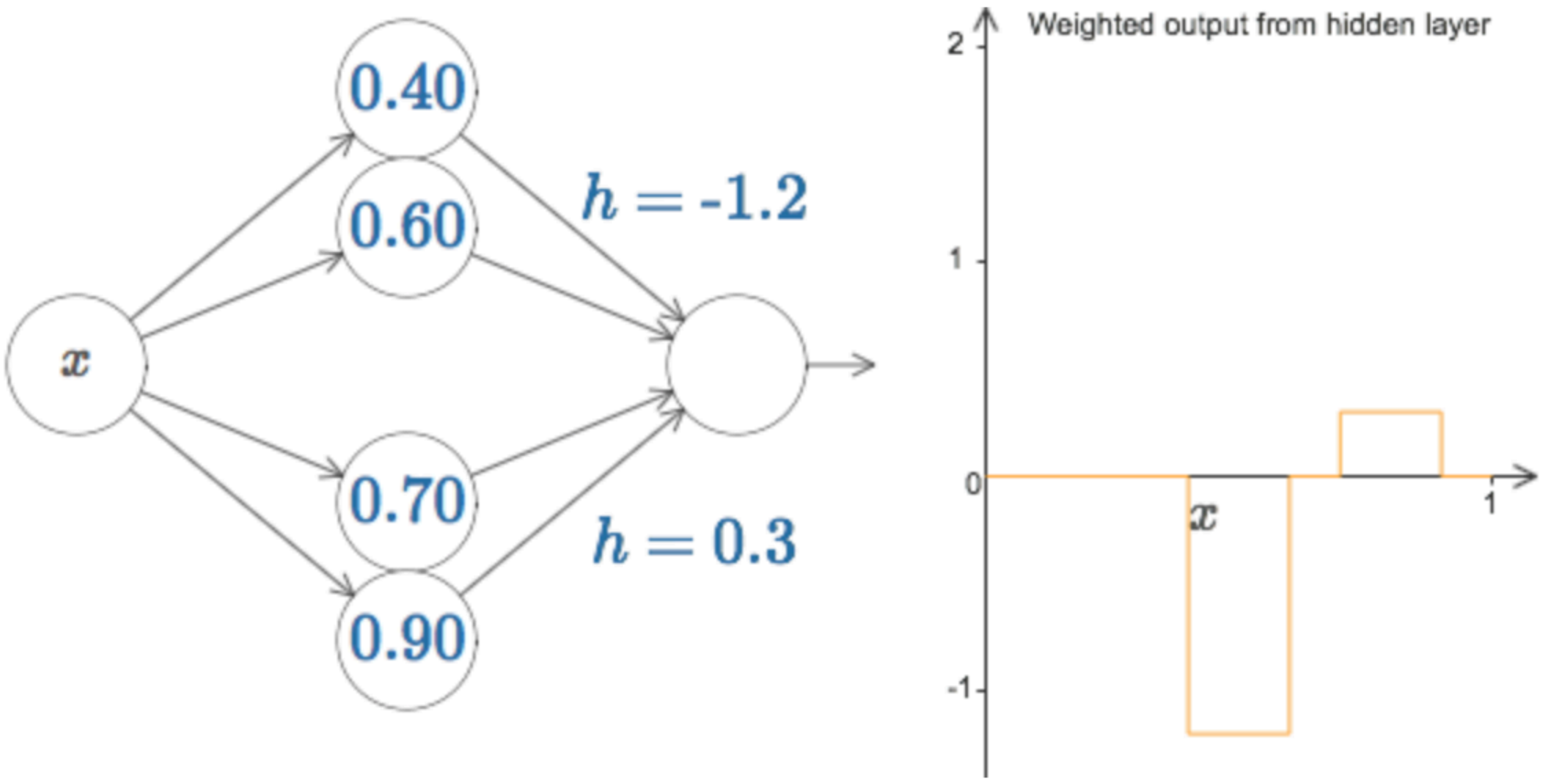
Light Theory: Neural Network as Universal Approximator

The output neuron is a **weighted combination of step functions** (assuming bias for that layer is 0)



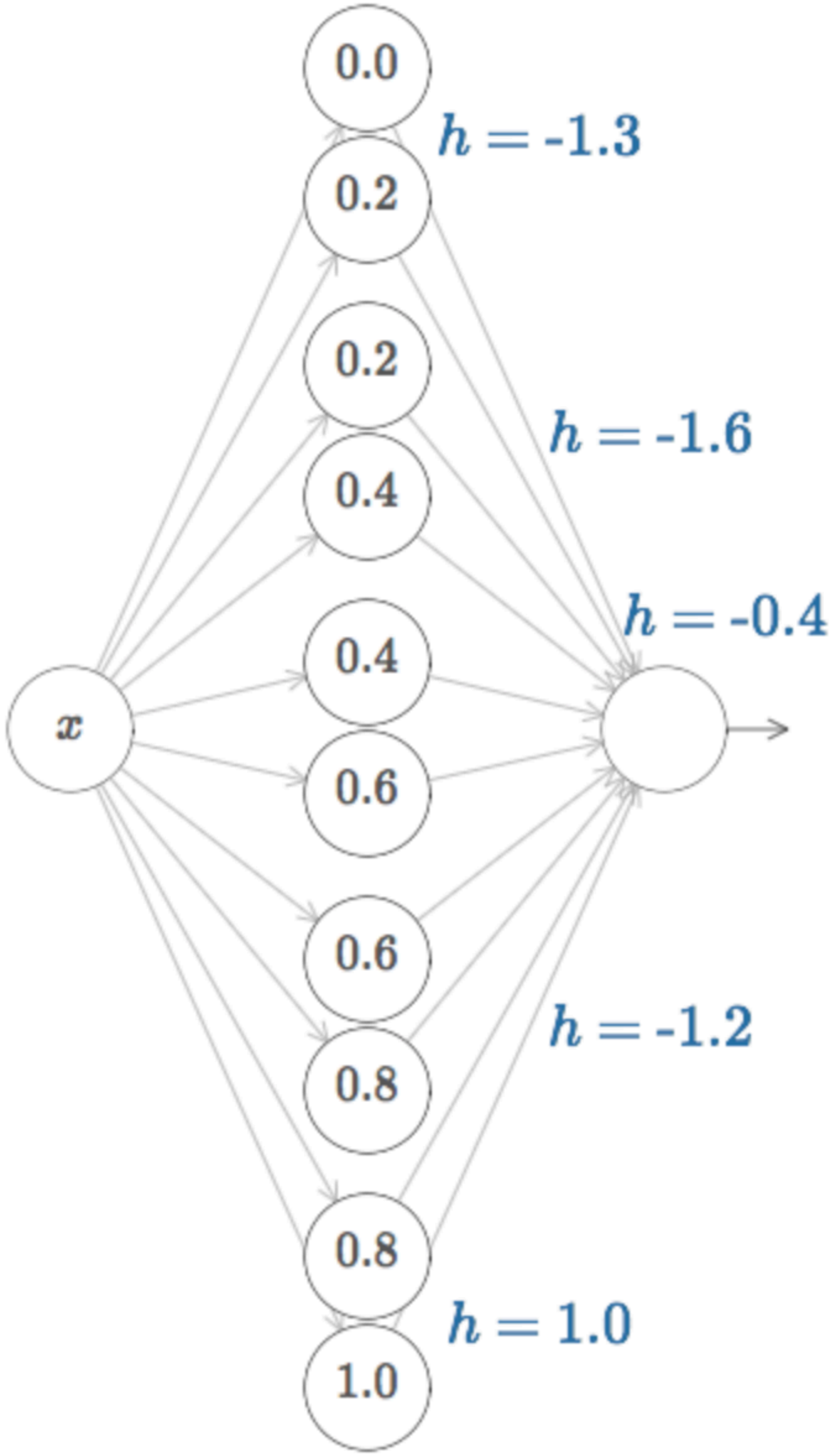
*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

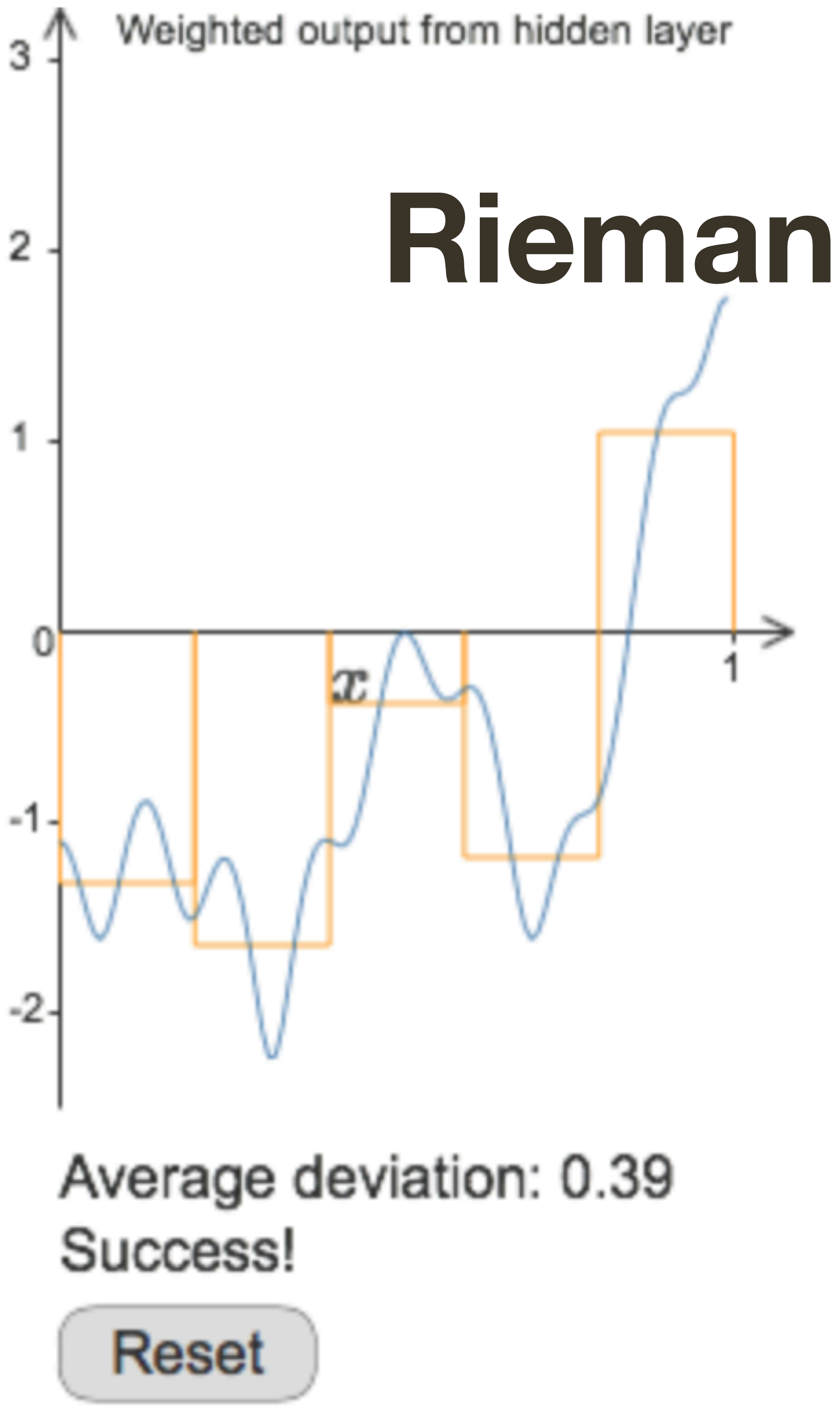
Light Theory: Neural Network as Universal Approximator



*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

Light Theory: Neural Network as Universal Approximator



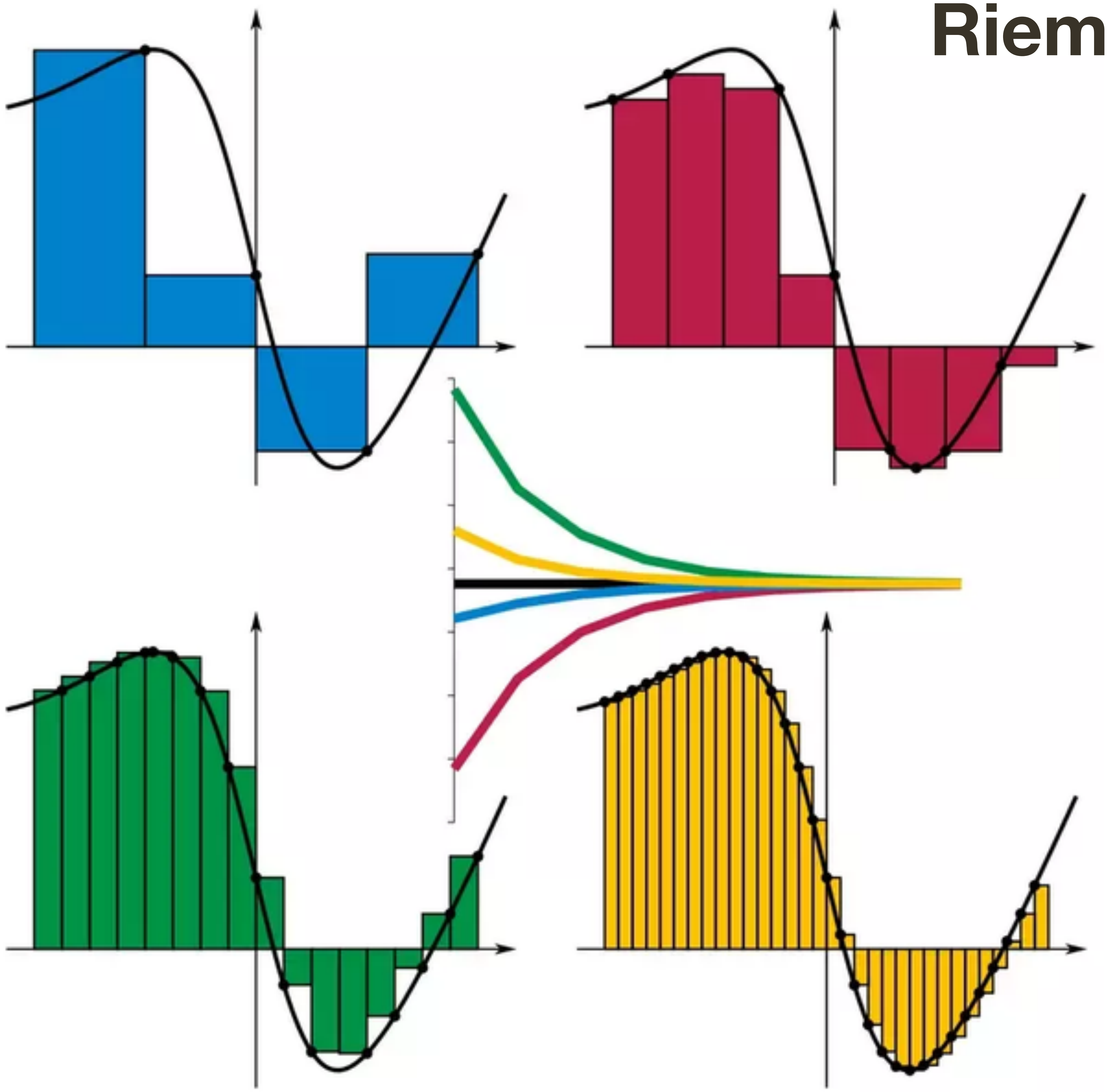


Riemann sum approximation

*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

Light Theory: Neural Network as Universal Approximator

Riemann sum approximation



Light Theory: Neural Network as Universal Approximator

Conditions needed for proof to hold: Activation function needs to be well defined

$$\lim_{x \to \infty} a(x) = A$$

$$\lim_{x \to -\infty} a(x) = B$$

$$A \neq B$$

*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

Light Theory: Neural Network as Universal Approximator

Conditions needed for proof to hold: Activation function needs to be well defined

$$\lim_{x \to \infty} a(x) = A$$

$$\lim_{x \to -\infty} a(x) = B$$

$$A \neq B$$

Note: This gives us another way to provably say that linear activation function cannot produce a neural network which is an universal approximator.

*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

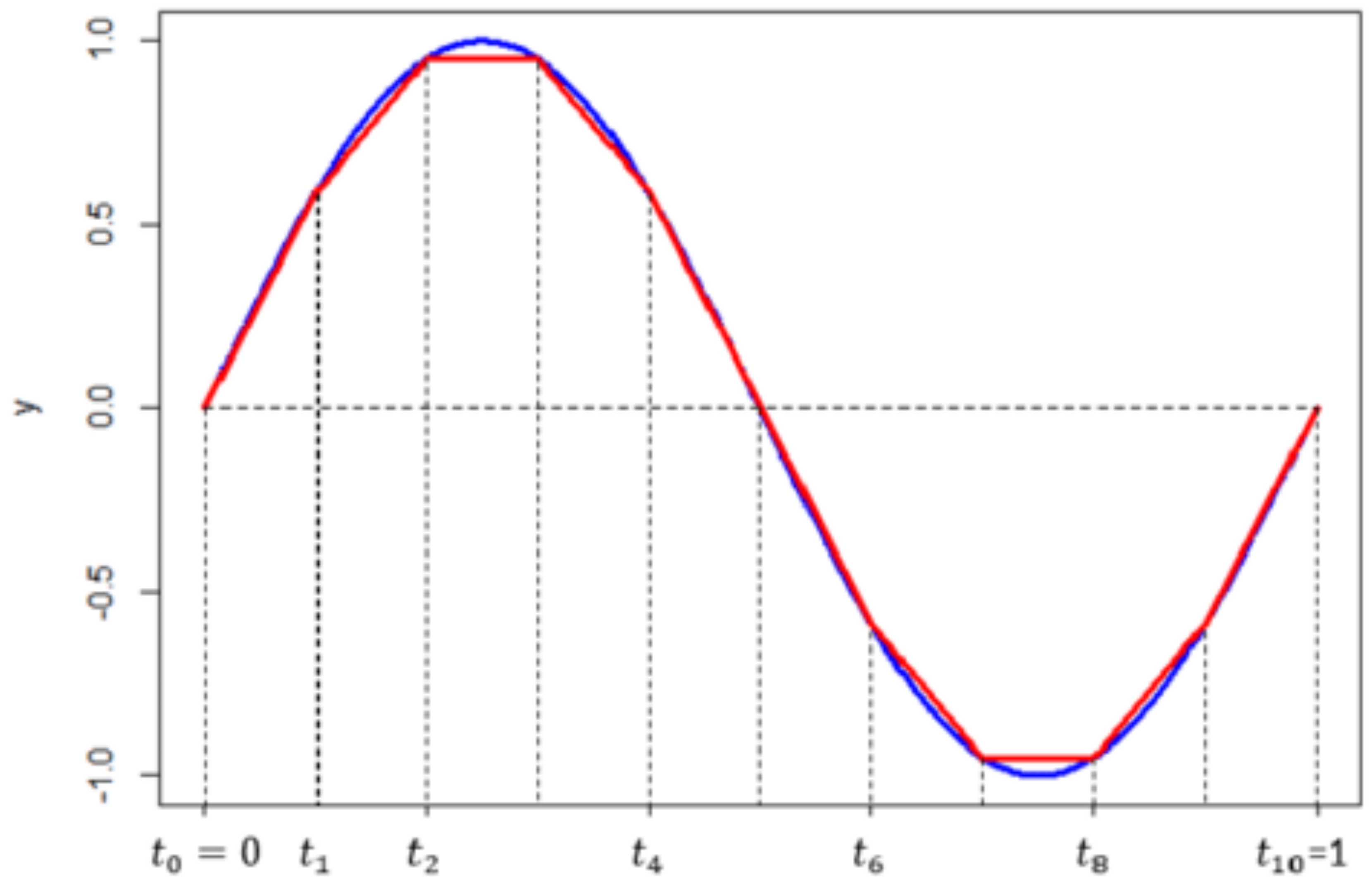
Activation Function

Non-linear activation is required to provably make the Neural Net a **universal function approximator**

Intuition: with ReLU activation, we effectively get a linear spline approximation to any function.

Optimization of neural net parameters = finding slops and transitions of linear pieces

The quality of approximation depends on the number of linear segments



Number of linear segments for large input dimension: $\Omega(2^{\frac{2}{3}Ln})$

Light Theory: Neural Network as Universal Approximator

Universal Approximation Theorem: Single hidden layer can approximate any continuous function with compact support to arbitrary accuracy, when the width goes to infinity.

[Hornik *et al*., 1989]

Universal Approximation Theorem (revised): A network of infinite depth with a hidden layer of size $d+1$ neurons, where d is the dimension of the input space, can approximate any continuous function.

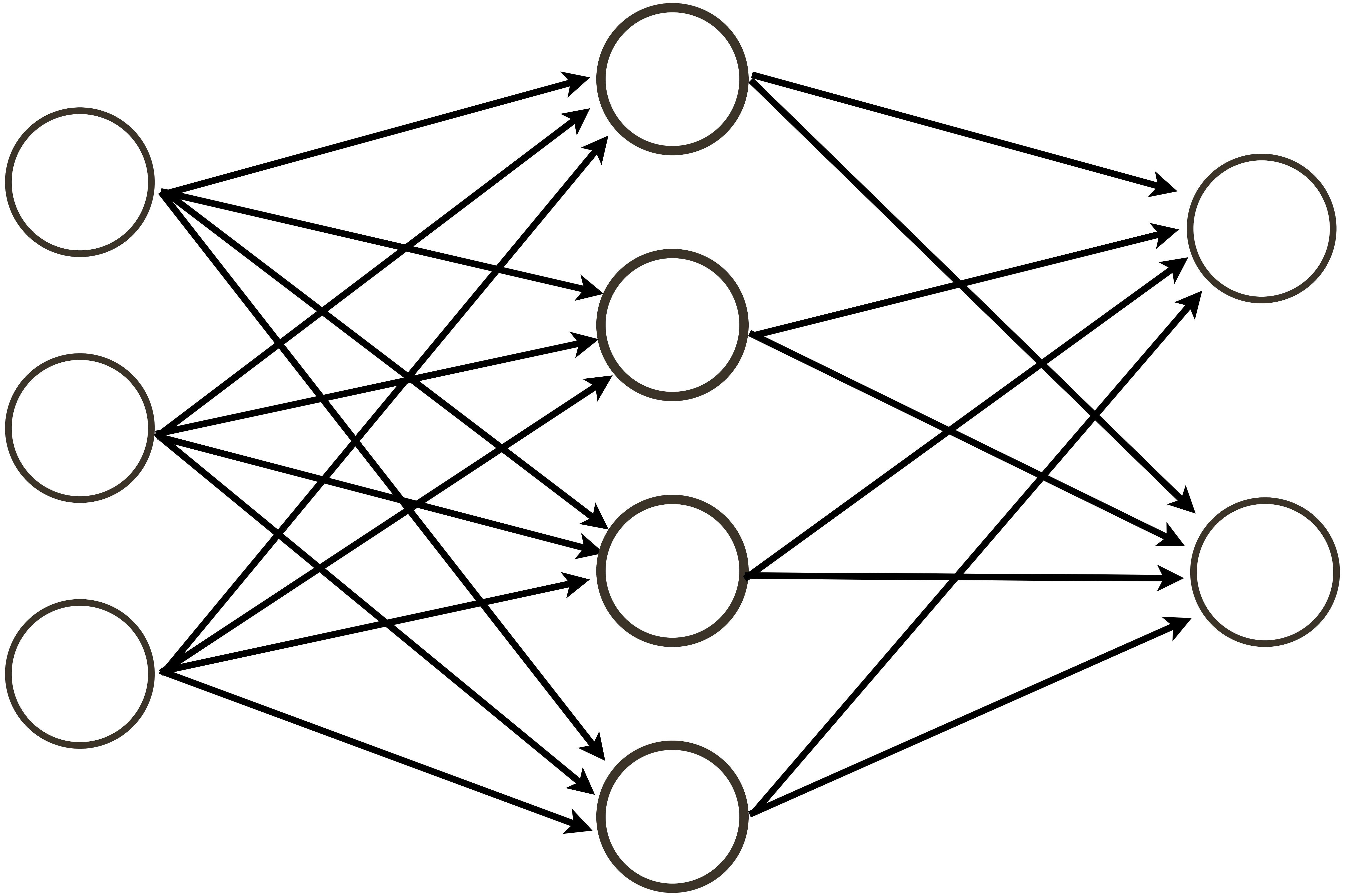
[Lu *et al*., NIPS 2017]

Universal Approximation Theorem (further revised): ResNet with a single hidden unit and infinite depth can approximate any continuous function.

[Lin and Jegelka, NIPS 2018]

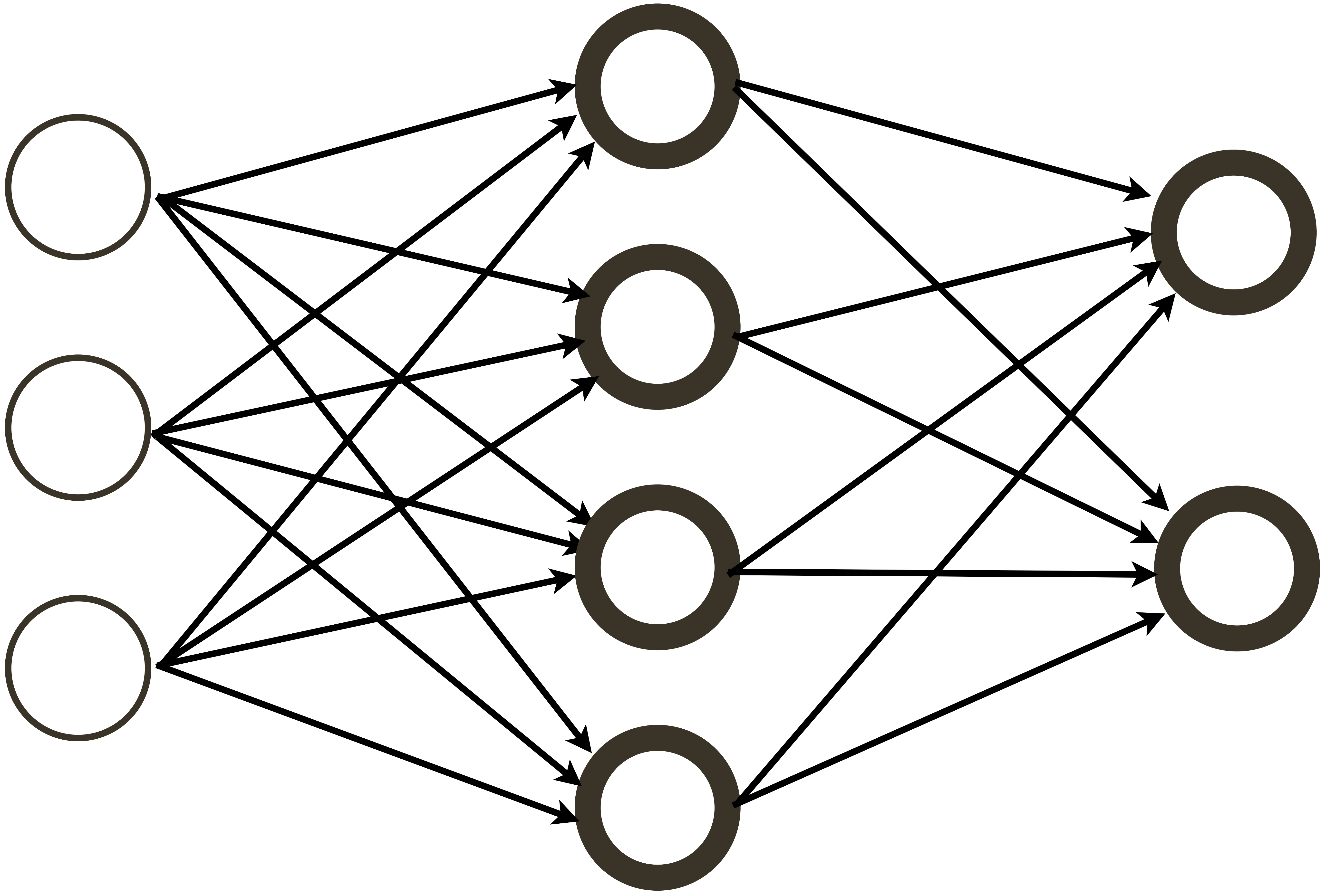
Neural Network

How many neurons?



Neural Network

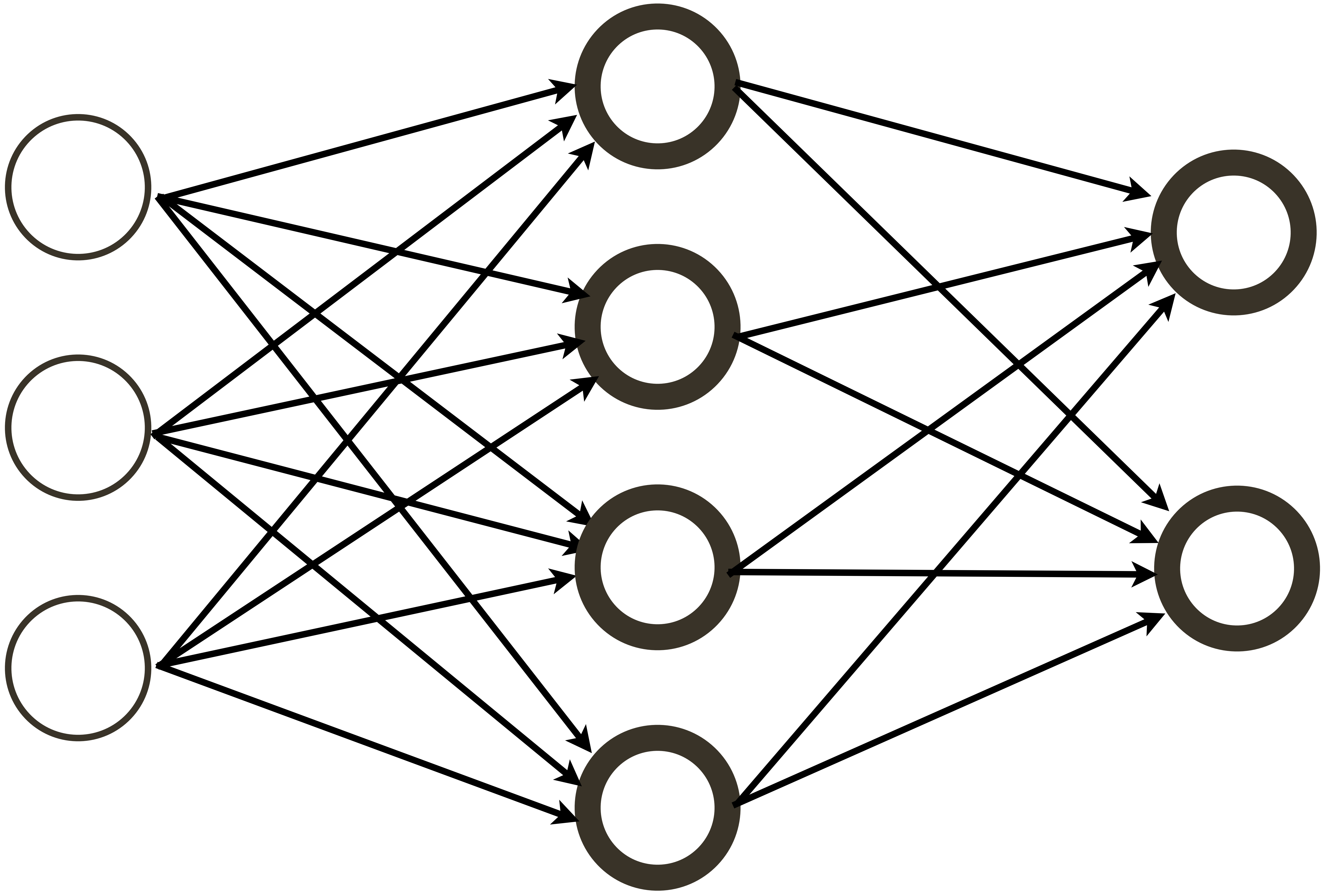
How many neurons? 4+2 = 6



Neural Network

How many neurons? 4+2 = 6

How many weights?

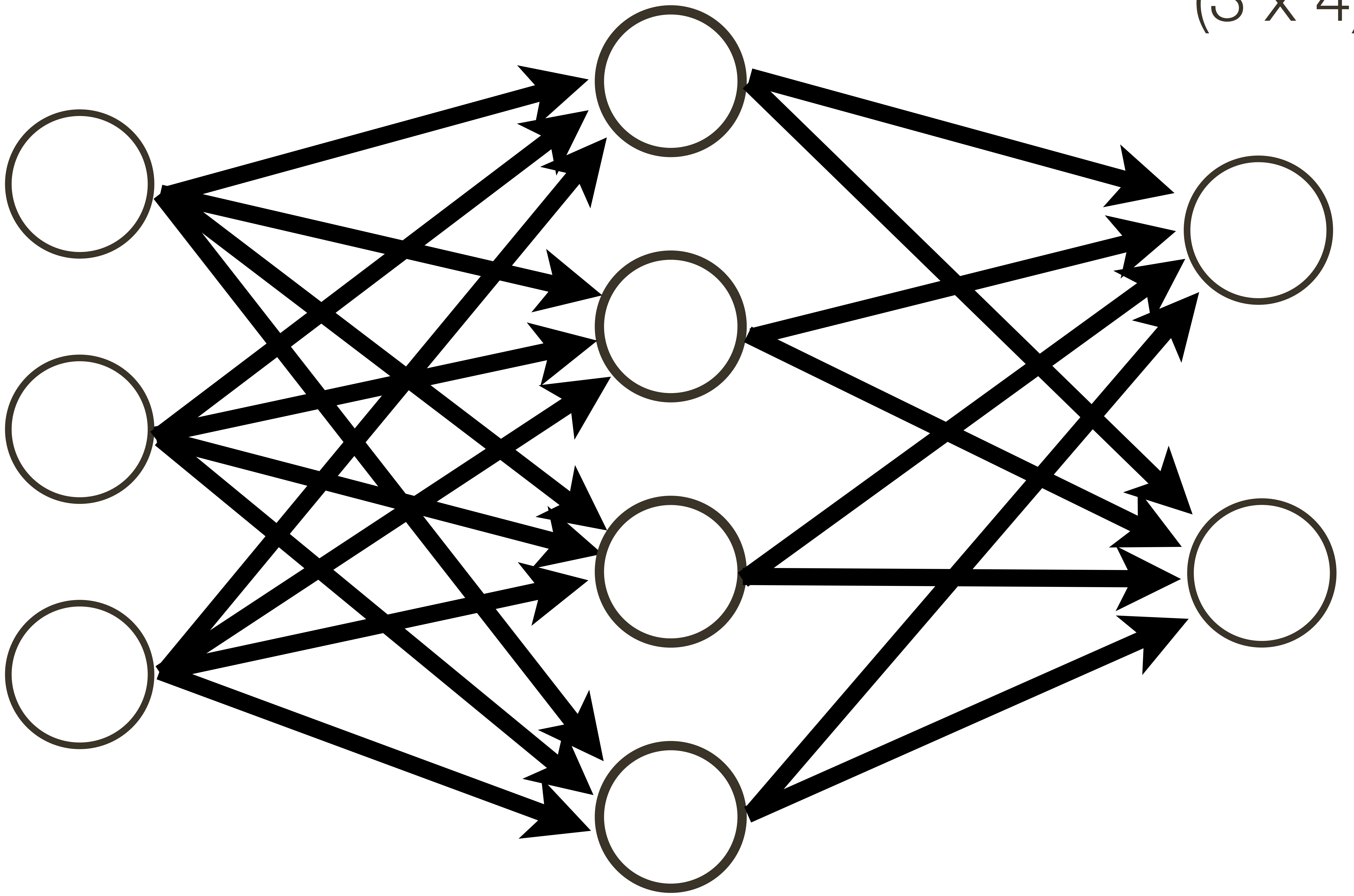


Neural Network

How many neurons? 4+2 = 6

How many weights?

(3 x 4) + (4 x 2) = 20

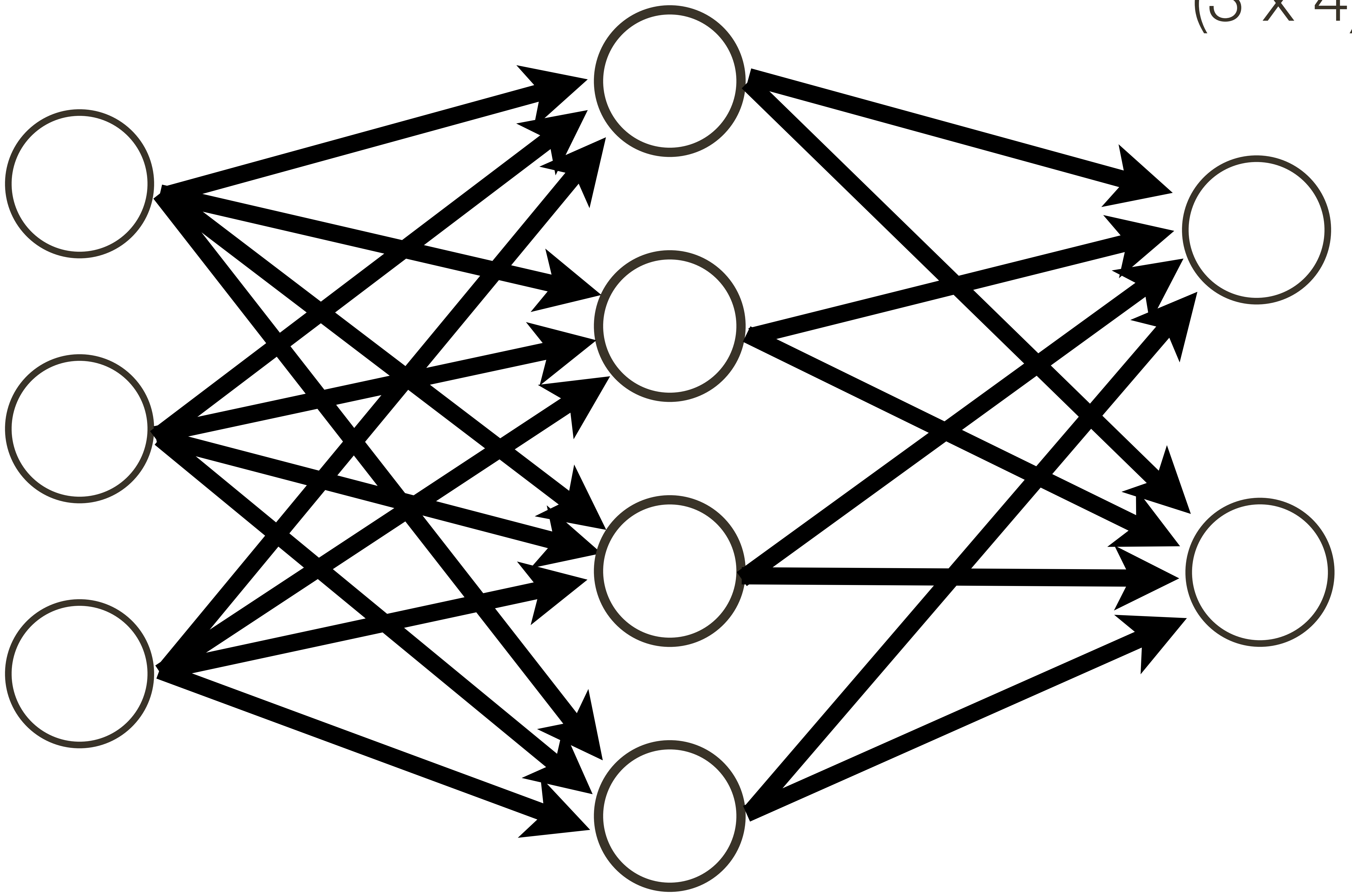


Neural Network

How many neurons? 4+2 = 6

How many weights?

(3 x 4) + (4 x 2) = 20



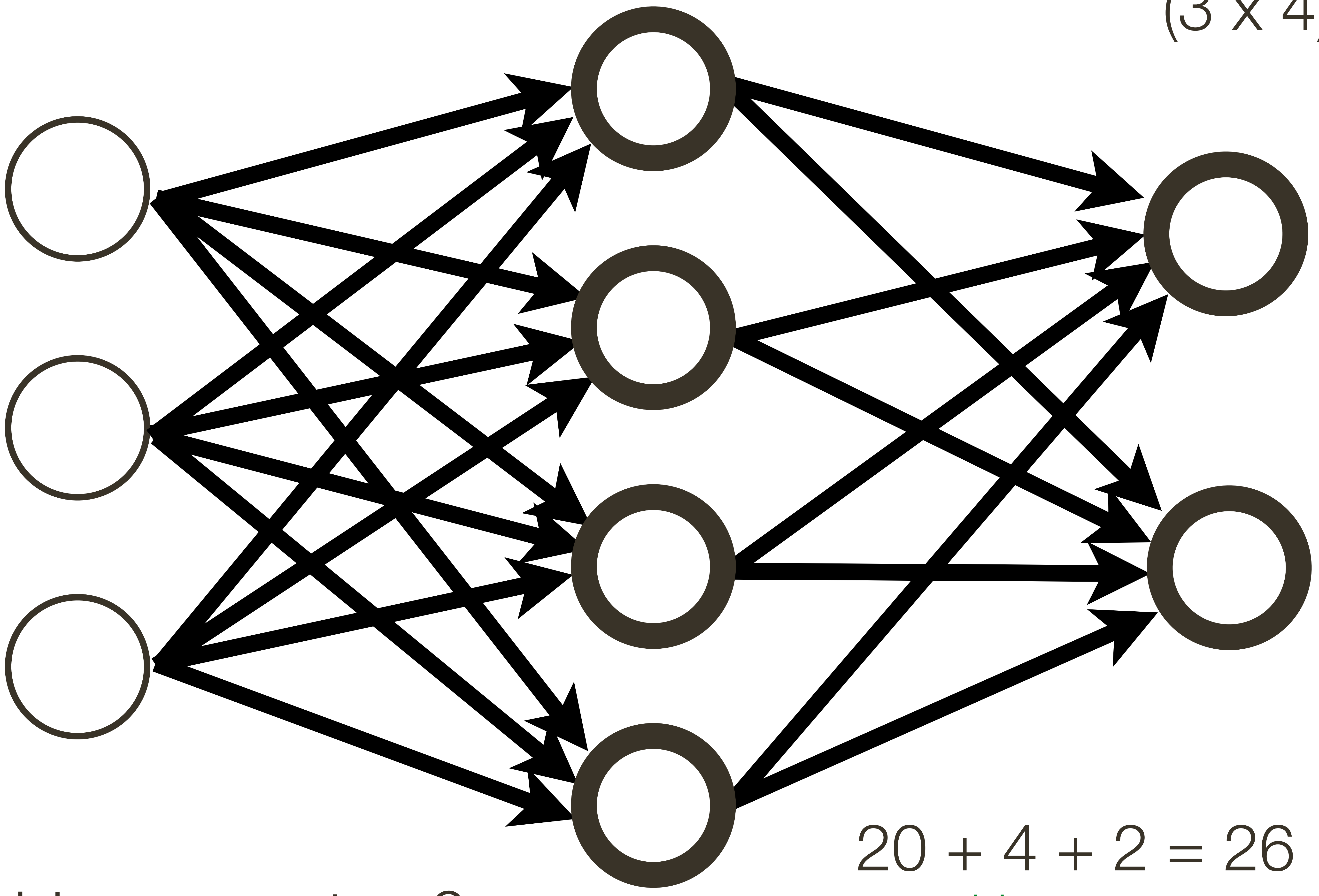
How many learnable parameters?

Neural Network

How many neurons? 4+2 = 6

How many weights?

(3 x 4) + (4 x 2) = 20



How many learnable parameters?

20 + 4 + 2 = 26

bias terms

Neural Networks

Modern **convolutional neural networks** contain 10-20 layers and on the order of 100 million parameters

Training a neural network requires estimating a large number of parameters

Training a Neural Network

Input Image

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix}$$

Vectorized Input

$$f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$$

Neural Network
(hidden layers)

Prediction
(score)

Training a Neural Network

Input Image

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix}$$

Vectorized Input

$$f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$$

Neural Network
(hidden layers)

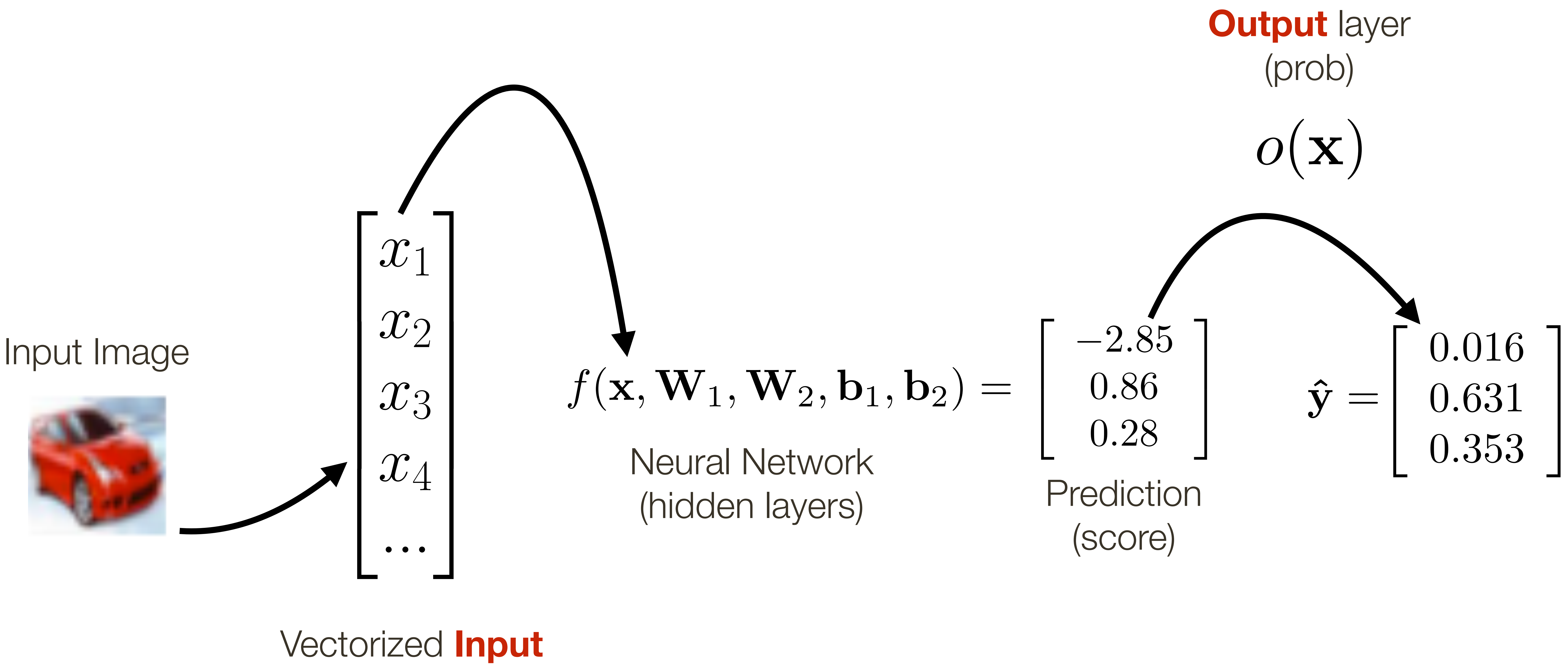
Prediction
(score)

Output layer
(prob)

$$o(\mathbf{x})$$

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.016 \\ 0.631 \\ 0.353 \end{bmatrix}$$

Training a Neural Network



Input and **output** layers (size and form) are dictated by the problem, intermediate hidden layers have few constraints and can be *anything*

Training a Neural Network

Input Image

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix}$$

Vectorized Input

$$f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$$

Neural Network (hidden layers)

Prediction (score)

Output layer (prob)

$$o(\mathbf{x})$$

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.016 \\ 0.631 \\ 0.353 \end{bmatrix}$$

Inference: $o(f(\mathbf{x}, \cdots))$

Training a Neural Network

Input Image

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix}$$

Vectorized Input

$$f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$$

Neural Network (hidden layers)

Prediction (score)

Output layer (prob)

$$o(\mathbf{x})$$

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$$

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.016 \\ 0.631 \\ 0.353 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

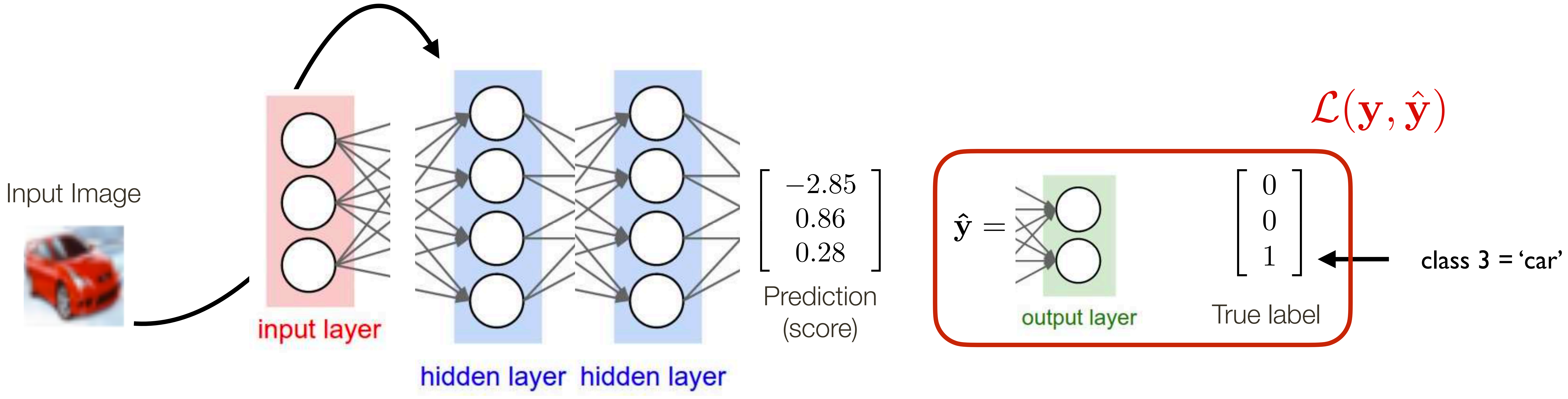
True label

class 3 = ‘car’

Inference: $o(f(\mathbf{x}, \cdots))$

Learning: $\mathcal{L}(\mathbf{y}, o(f(\mathbf{x}, \cdots)))$

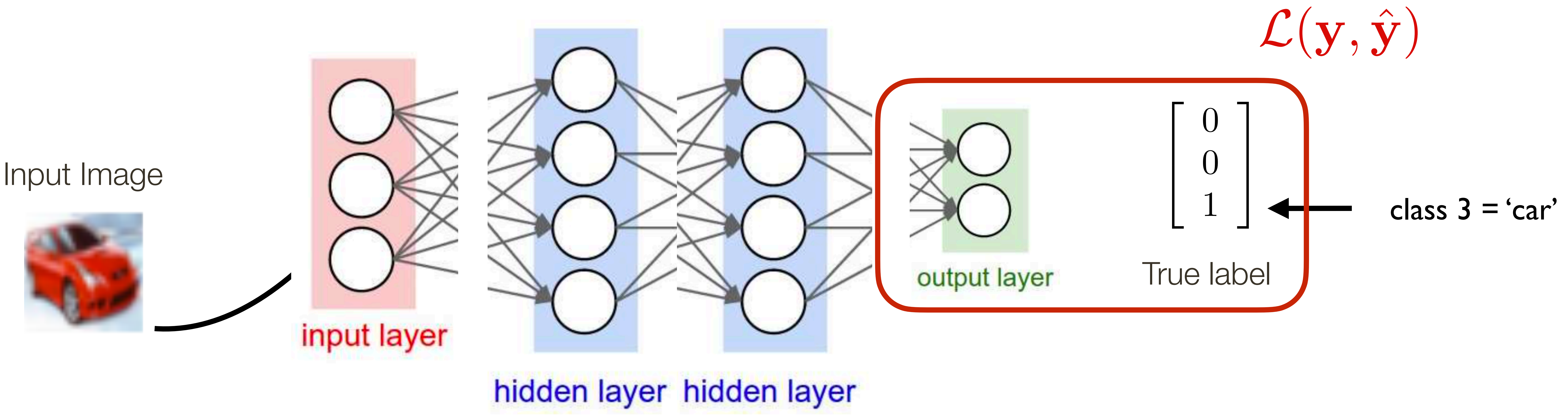
Training a Neural Network



Inference: $o(f(\mathbf{x}, \cdots))$

Learning: $\mathcal{L}(\mathbf{y}, o(f(\mathbf{x}, \cdots)))$

Training a Neural Network



Inference: $o(f(\mathbf{x}, \cdots))$

Learning: $\mathcal{L}(\mathbf{y}, o(f(\mathbf{x}, \cdots)))$

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $\mathcal{L} = -\sum_i y_i \log(\hat{y}_i) \quad \hat{y}_i = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $\mathcal{L} = -\sum_i y_i \log(\hat{y}_i) \quad \hat{y}_i = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $\mathcal{L} = -\sum_i y_i \log(\hat{y}_i) \quad \hat{y}_i = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

$$f$$

$$c_1 = -2.85$$

$$c_2 = 0.86$$

$$c_3 = 0.28$$

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $\mathcal{L} = -\sum_i y_i \log(\hat{y}_i) \quad \hat{y}_i = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

$$f$$

$$c_1 = -2.85 \qquad \xrightarrow{\exp} \qquad 0.058$$

$$c_2 = 0.86 \qquad \qquad 2.36$$

$$c_3 = 0.28 \qquad \qquad 1.32$$

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $\mathcal{L} = -\sum_i y_i \log(\hat{y}_i)$ $\quad \hat{y}_i = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

f				
$c_1 = -2.85$	exp →	0.058	Normalize to sum to 1 →	0.016
$c_2 = 0.86$		2.36		0.631
$c_3 = 0.28$		1.32		0.353

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $\mathcal{L} = -\sum_i y_i \log(\hat{y}_i) \quad \hat{y}_i = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

f				probability of a class
$c_1 = -2.85$	exp →	0.058	Normalize to sum to 1 →	0.016
$c_2 = 0.86$		2.36		0.631
$c_3 = 0.28$		1.32		0.353

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $\mathcal{L} = -\sum_i y_i \log(\hat{y}_i)$ $\hat{y}_i = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$

softmax function
multi-class classifier

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

f				probability of a class
$c_1 = -2.85$	exp →	0.058	Normalize to sum to 1 →	0.016
$c_2 = 0.86$		2.36		0.631
$c_3 = 0.28$		1.32		0.353

Backpropagation

When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $\mathcal{L} = -\sum_i y_i \log(\hat{y}_i) \quad \hat{y}_i = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

Consider neural net which takes input vector $\mathbf{x}_i$ and predicts scores for 3 classes, with true class being class 3:

f				probability of a class	
$c_1 = -2.85$	exp →	0.058	Normalize to sum to 1 →	0.016	
$c_2 = 0.86$		2.36		0.631	$\mathcal{L} = -\log(0.353) = 1.04$
$c_3 = 0.28$		1.32		0.353	

Backpropagation

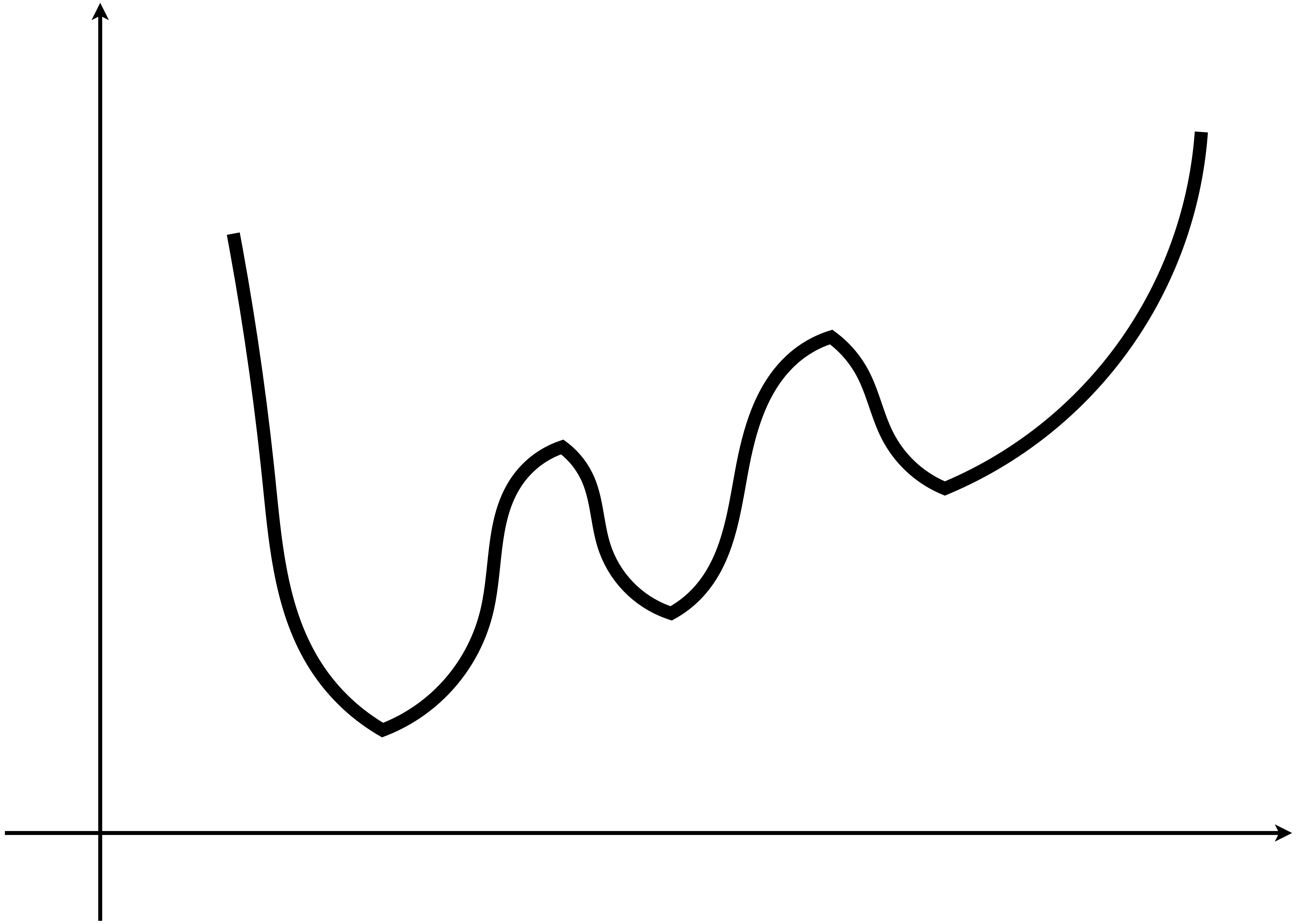
When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $\mathcal{L} = -\sum_i y_i \log(\hat{y}_i) \quad \hat{y}_i = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$

which defines loss for i-th training example with true class index y_i; and f_j is the j-th element of the vector of class scores coming from neural net.

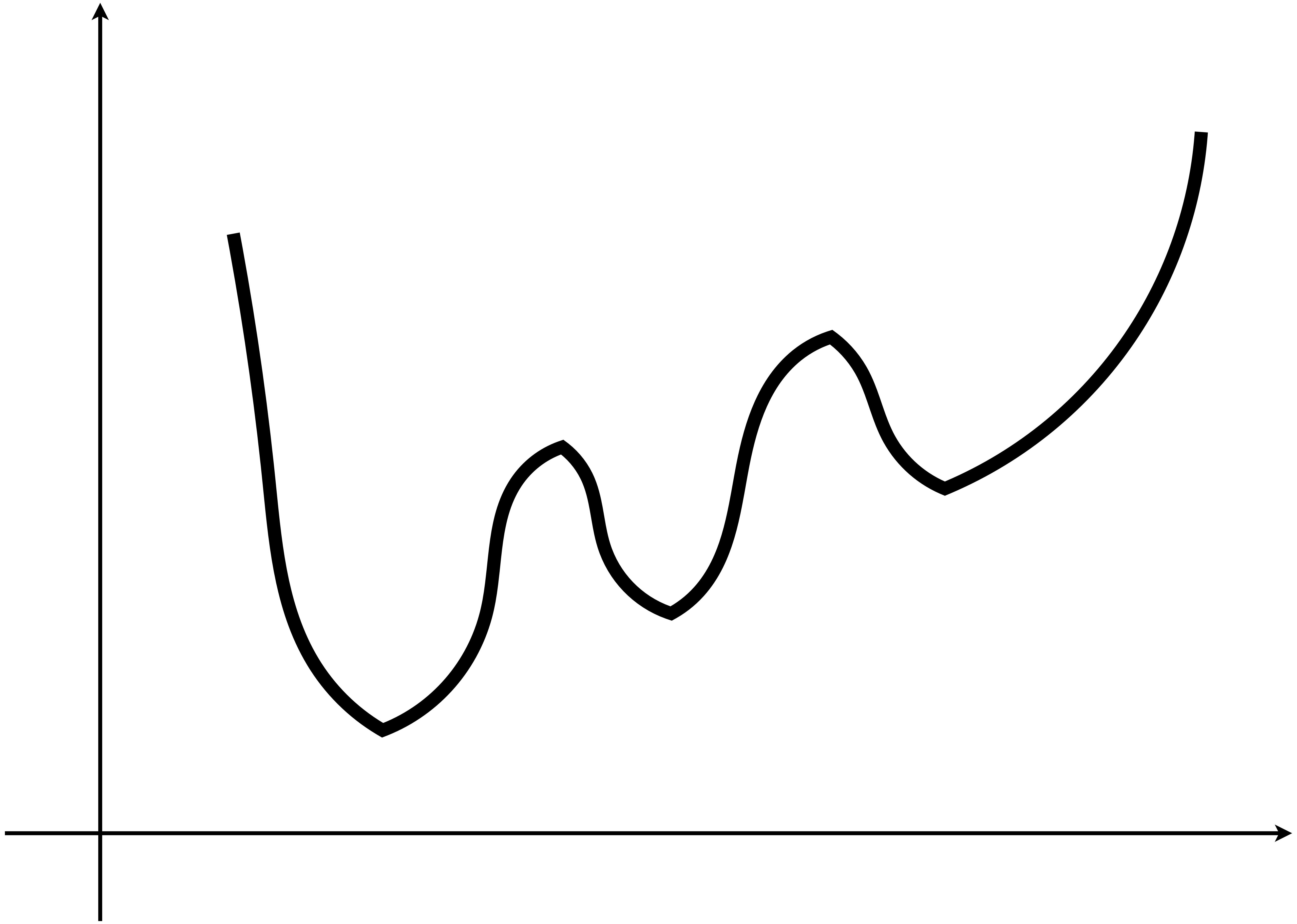
We want to compute the **gradient** of the loss with respect to the network parameters so that we can incrementally adjust the network parameters

Gradient Descent



*slide adopted from V. Ordonex

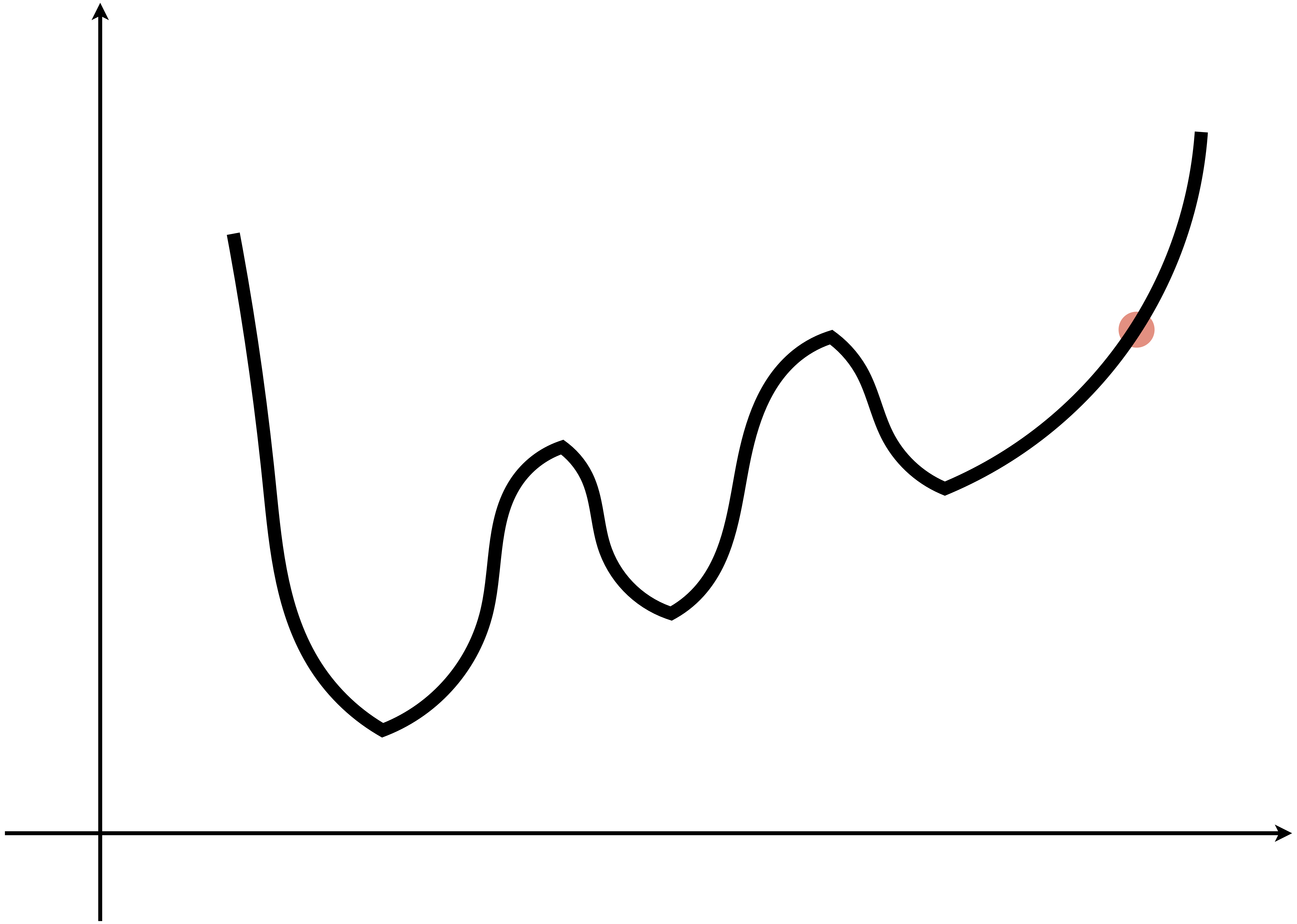
Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

*slide adopted from V. Ordonex

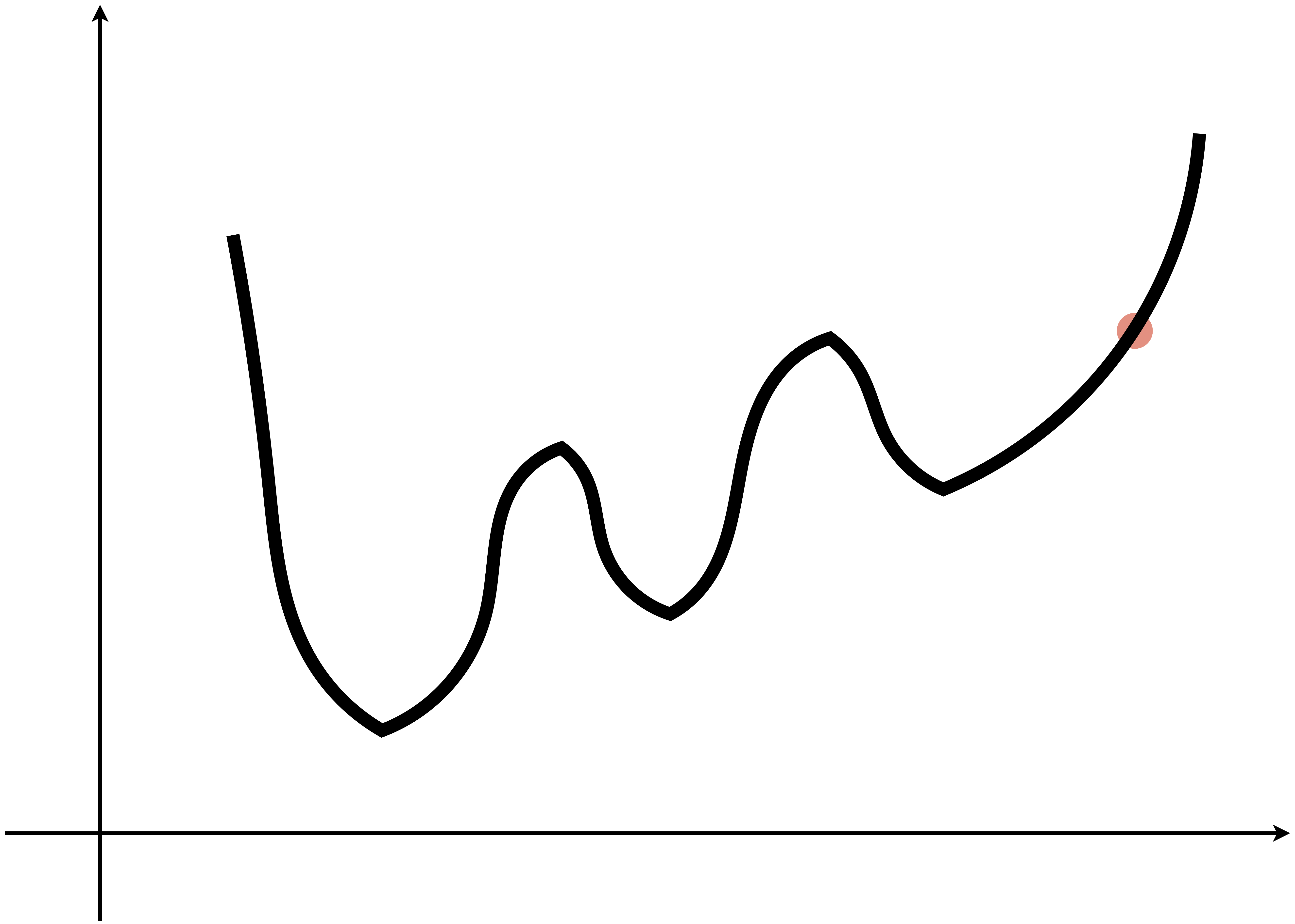
Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

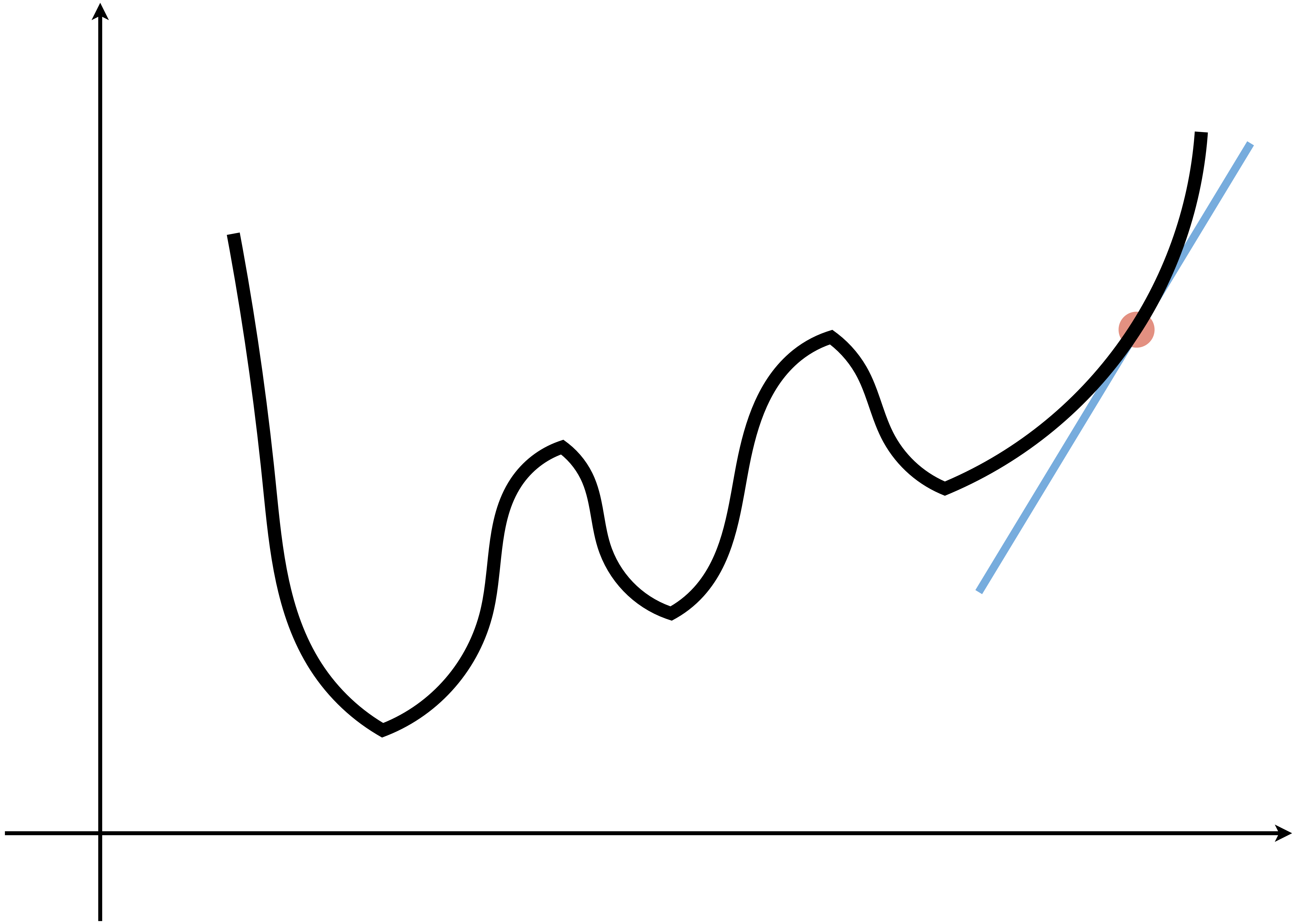
For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla\ \mathcal{L}(\mathbf{W}, \mathbf{b})|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

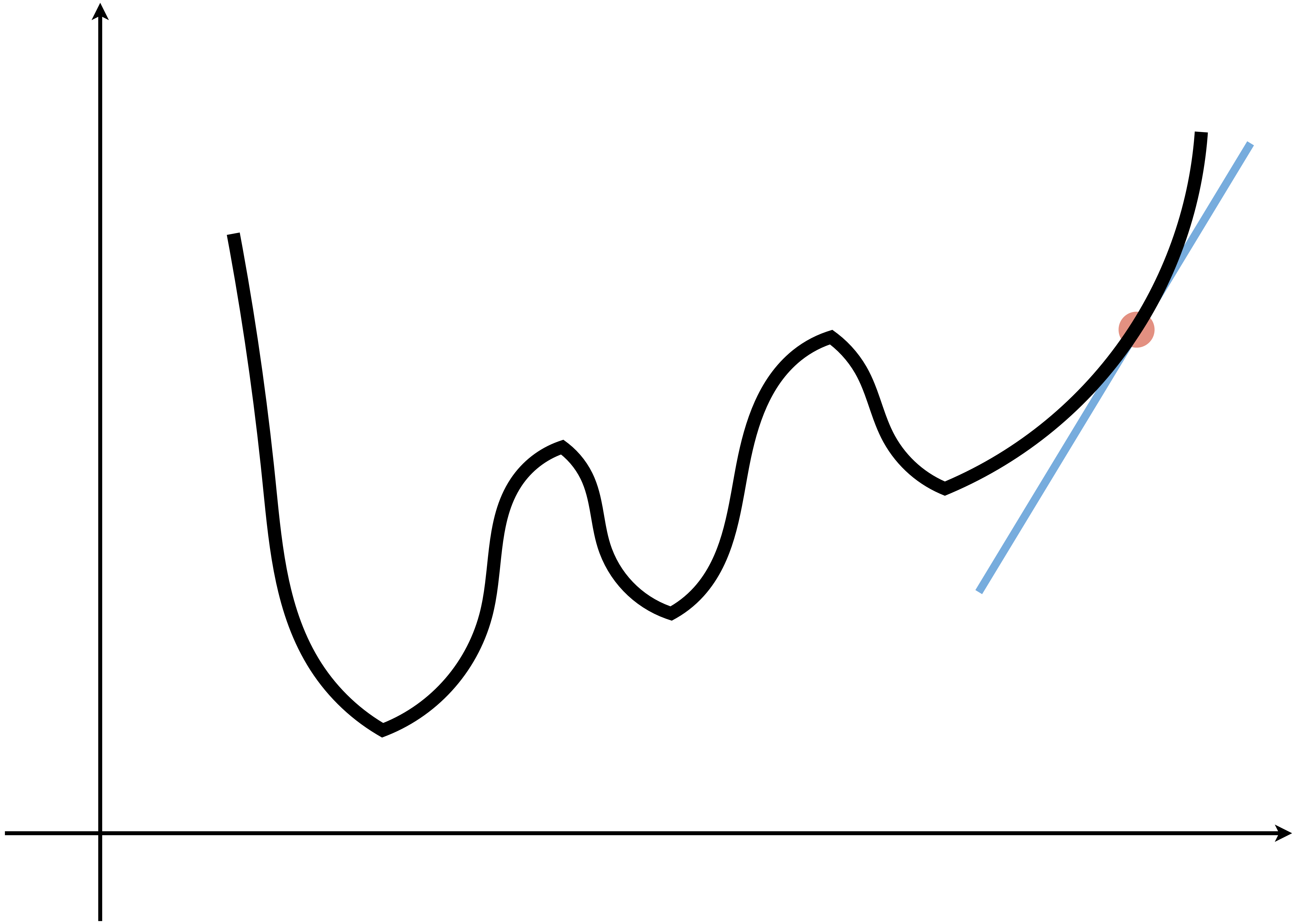
For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

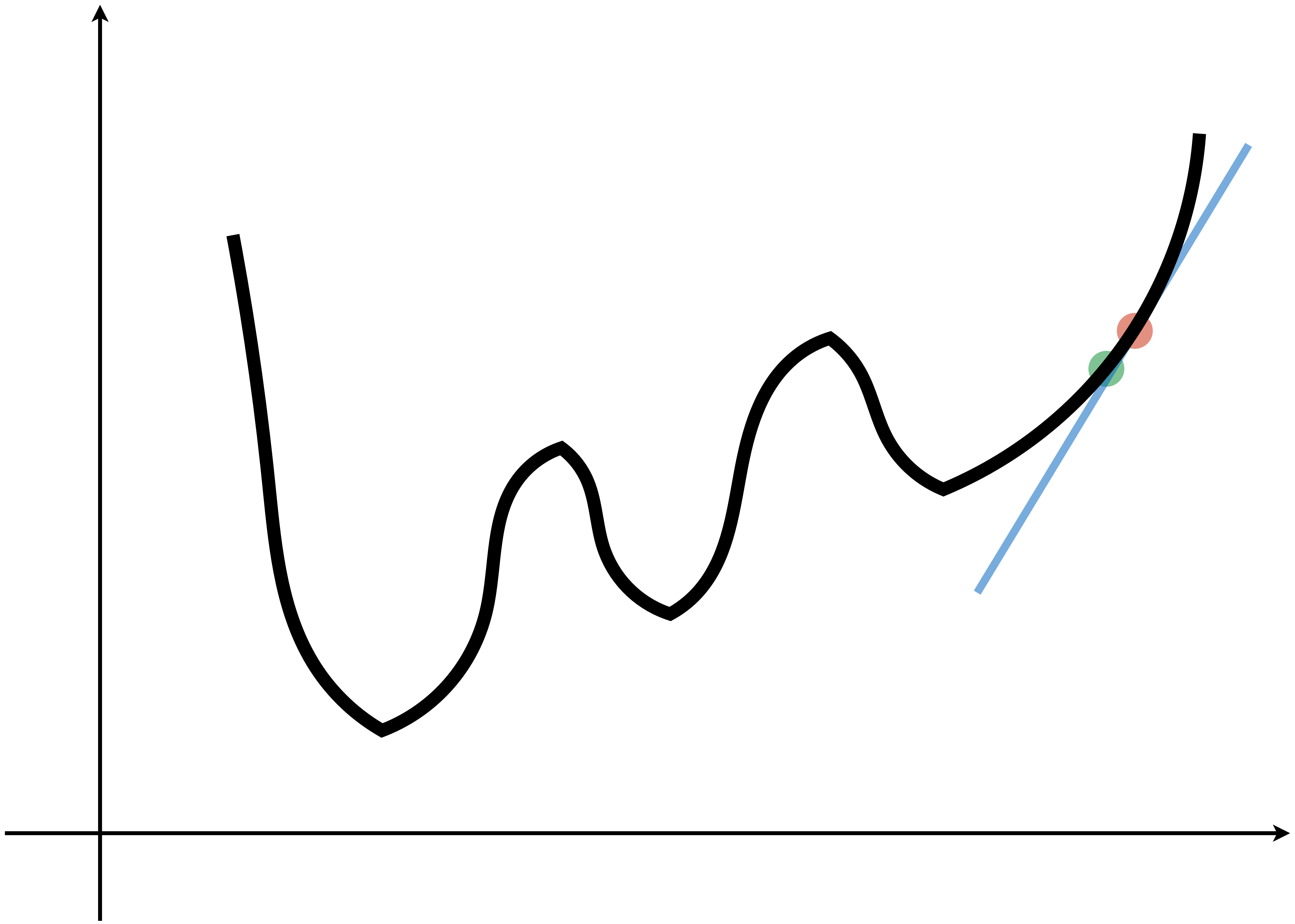
3. Re-estimate the parameters

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \lambda \left.\frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}}\right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \lambda \left.\frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}}\right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})\big|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

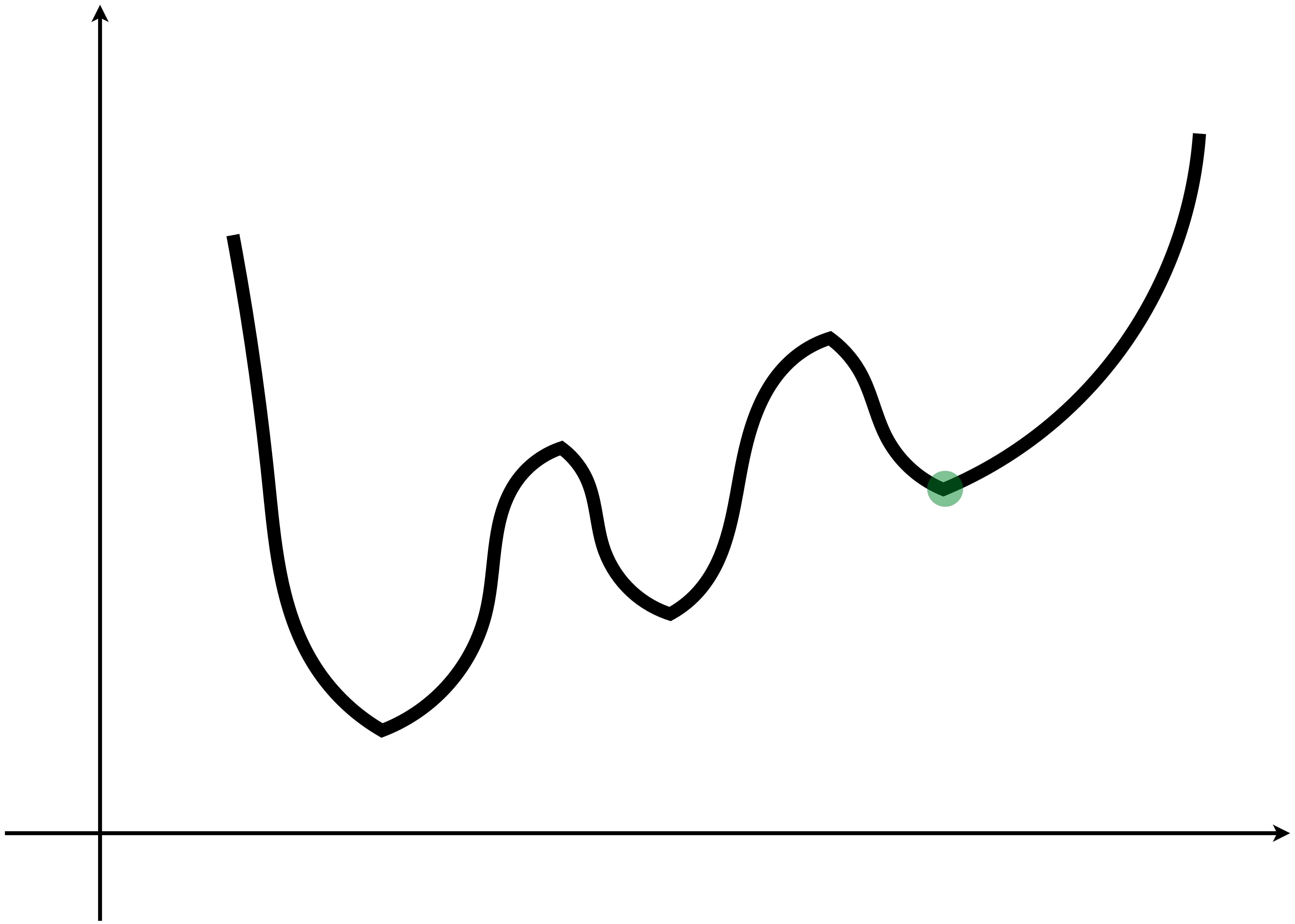
3. Re-estimate the parameters

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \lambda \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}}\bigg|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \lambda \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}}\bigg|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

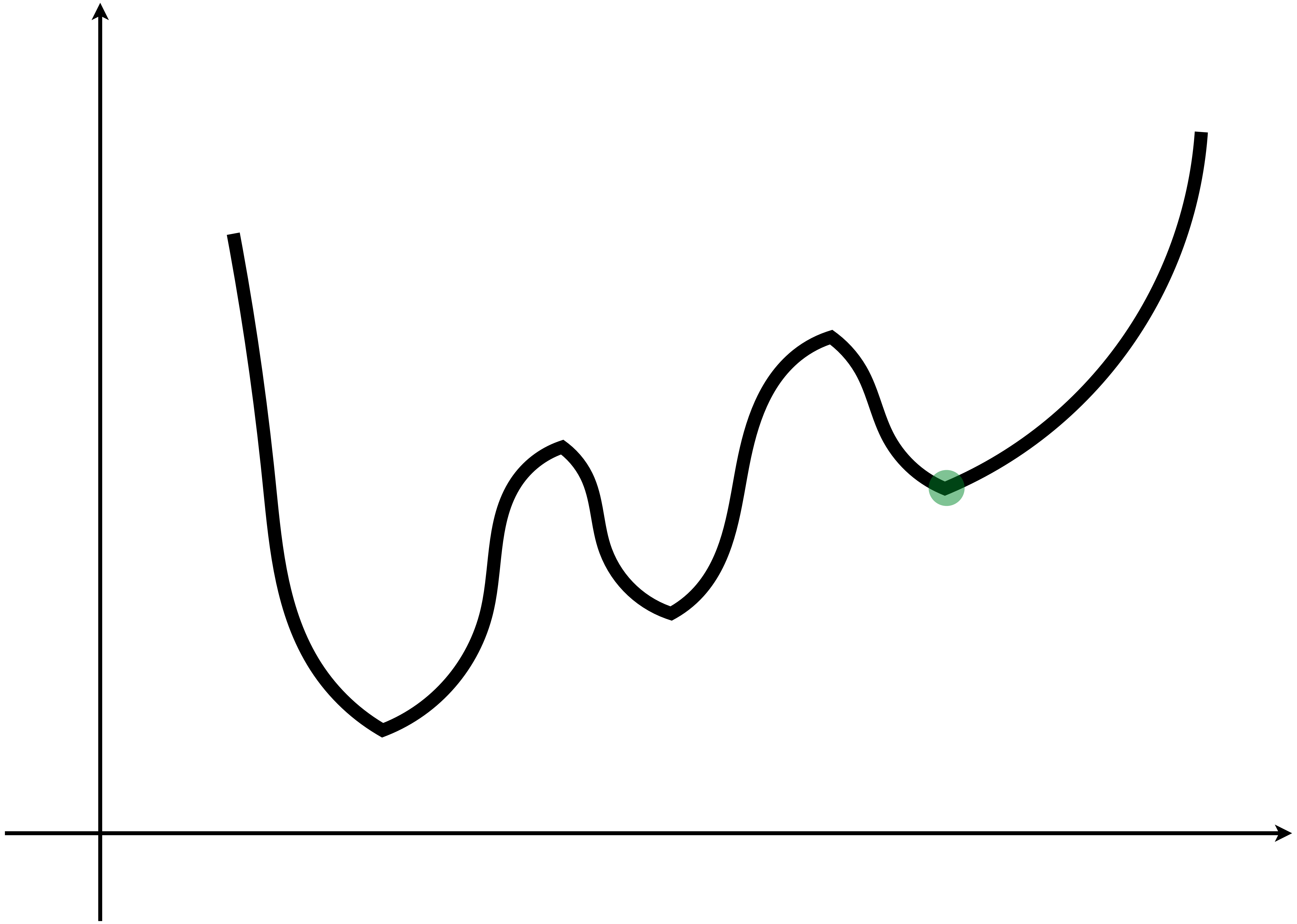
3. Re-estimate the parameters

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \lambda \left. \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}} \right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \lambda \left. \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}} \right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})\Big|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

3. Re-estimate the parameters

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \underline{\lambda} \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}}\Big|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \underline{\lambda} \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}}\Big|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

λ - is the learning rate

*slide adopted from V. Ordonex

Gradient Descent

Loss: $$\mathcal{L} = \sum_{i=1}^{|\mathcal{D}_{train}|} ||\mathbf{y}_i - \hat{\mathbf{y}}_i|| = \sum_{i=1}^{|\mathcal{D}_{train}|} ||\mathbf{y}_i - f(\mathbf{x}_i, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)||$$

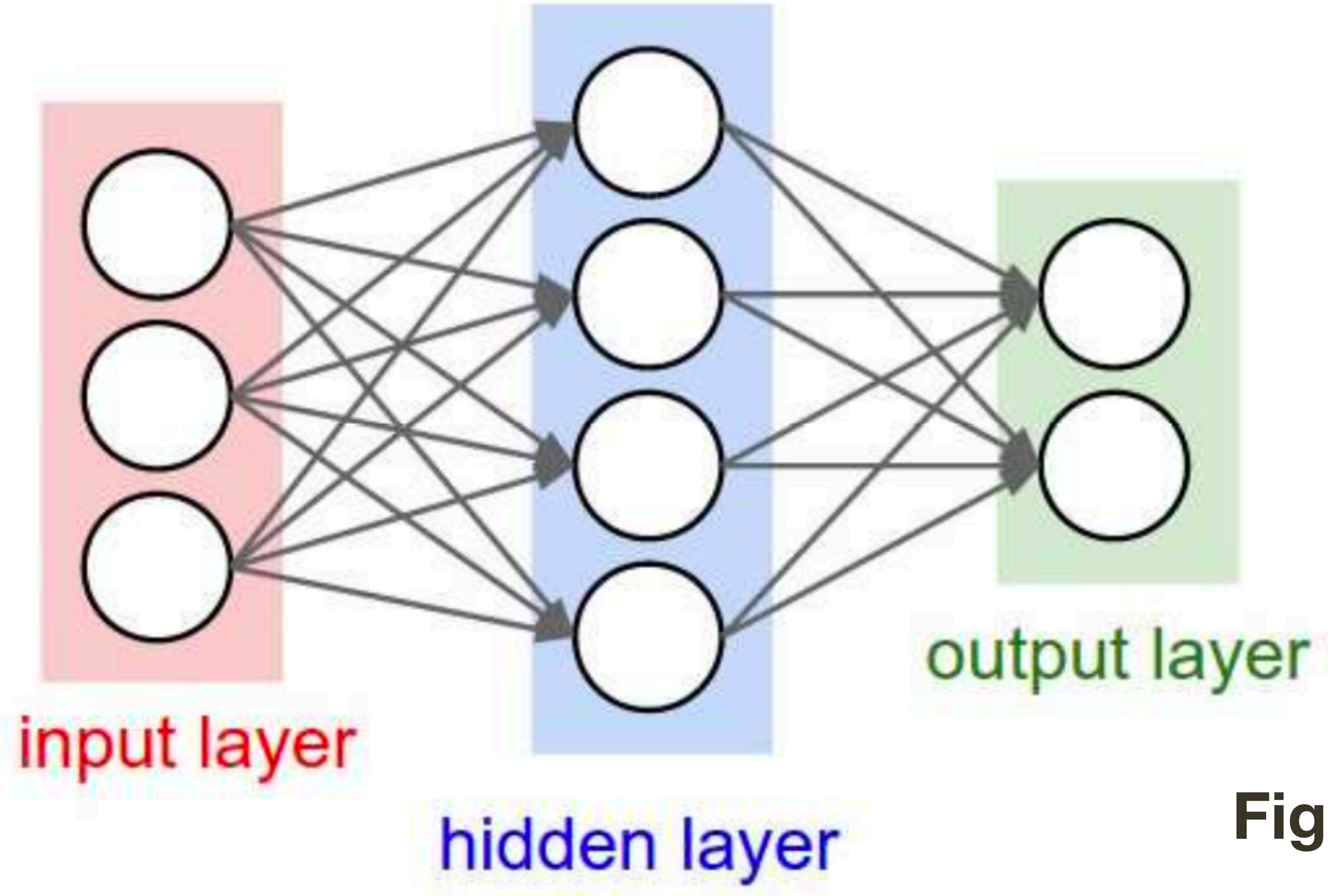


Figure credit: Fei-Fei and Karpathy

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times 4)} \sigma\left(\mathbf{W}_1^{(4\times 3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

Gradient Descent

Loss: $\mathcal{L} = \sum_{i=1}^{|\mathcal{D}_{train}|} ||\mathbf{y}_i - \hat{\mathbf{y}}_i|| = \sum_{i=1}^{|\mathcal{D}_{train}|} ||\mathbf{y}_i - f(\mathbf{x}_i, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)||$

Gradient Descent

$$\mathbf{W}_{1,i,j} = \mathbf{W}_{1,i,j} - \lambda \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}_{1,i,j}}$$

$$\mathbf{b}_{1,i} = \mathbf{b}_{1,i} - \lambda \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}_{1,i}}$$

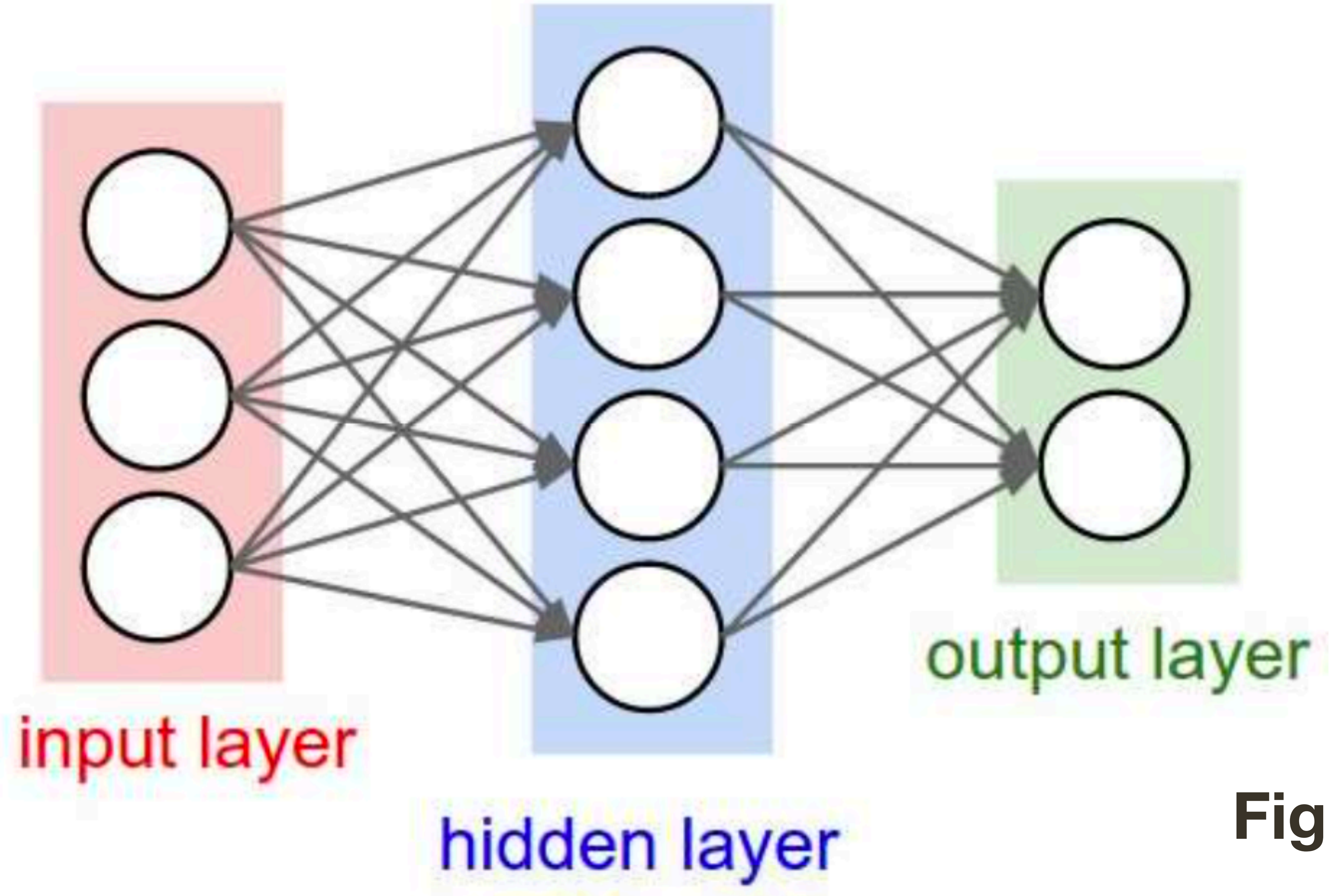


Figure credit: Fei-Fei and Karpathy

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times4)}\sigma\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

Stochastic Gradient Descent

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{1,i,j}} = \frac{\partial}{\partial \mathbf{W}_{1,i,j}} \sum_{i=1}^{|\mathcal{D}_{train}|} [\mathbf{y}_i - f(\mathbf{x}_i, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)]^2$$

Stochastic Gradient Descent

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{1,i,j}} = \frac{\partial}{\partial \mathbf{W}_{1,i,j}} \sum_{i=1}^{|\mathcal{D}_{train}|} [\mathbf{y}_i - f(\mathbf{x}_i, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)]^2$$

Problem: For large datasets computing sum is expensive

Stochastic Gradient Descent

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{1,i,j}} = \frac{\partial}{\partial \mathbf{W}_{1,i,j}} \sum_{i=1}^{|\mathcal{D}_{train}|} [\mathbf{y}_i - f(\mathbf{x}_i, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)]^2$$

Problem: For large datasets computing sum is expensive

Solution: Compute approximate gradient with mini-batches of much smaller size (as little as 1-example sometimes)

Stochastic Gradient Descent

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{1,i,j}} = \frac{\partial}{\partial \mathbf{W}_{1,i,j}} \sum_{i=1}^{|\mathcal{D}_{train}|} [\mathbf{y}_i - f(\mathbf{x}_i, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)]^2$$

Problem: For large datasets computing sum is expensive

Solution: Compute approximate gradient with mini-batches of much smaller size (as little as 1-example sometimes)

Problem: How do we compute the actual gradient?

Numerical Differentiation

$\mathbf{1}_i$ - Vector of all zeros, except for one 1 in i-th location

We can approximate the gradient numerically, using:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x})}{h}$$

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Numerical Differentiation

$\mathbf{1}_i$ - Vector of all zeros, except for one 1 in i-th location

We can approximate the gradient numerically, using:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x})}{h}$$

Even better, we can use central differencing:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x} - h\mathbf{1}_i)}{2h}$$

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Numerical Differentiation

$\mathbf{1}_i$ - Vector of all zeros, except for one 1 in i-th location

We can approximate the gradient numerically, using:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x})}{h}$$

Even better, we can use central differencing:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x} - h\mathbf{1}_i)}{2h}$$

However, both of theses suffer from rounding errors and are not good enough for learning.

$$h = 0.000001$$

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Numerical Differentiation

$\mathbf{1}_i$ - Vector of all zeros, except for one 1 in i-th location

$\mathbf{1}_{ij}$ - Matrix of all zeros, except for one 1 in (i,j)-th location

We can approximate the gradient numerically, using:

$$\frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial w_{ij}} \approx \lim_{h \to 0} \frac{\mathcal{L}(\mathbf{W} + h\mathbf{1}_{ij}, \mathbf{b}) - \mathcal{L}(\mathbf{W}, \mathbf{b})}{h} \qquad \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial b_j} \approx \lim_{h \to 0} \frac{\mathcal{L}(\mathbf{W}, \mathbf{b} + h\mathbf{1}_j) - \mathcal{L}(\mathbf{W}, \mathbf{b})}{h}$$

Even better, we can use central differencing:

$$\frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial w_{ij}} \approx \lim_{h \to 0} \frac{\mathcal{L}(\mathbf{W} + h\mathbf{1}_{ij}, \mathbf{b}) - \mathcal{L}(\mathbf{W} + h\mathbf{1}_{ij}, \mathbf{b})}{2h} \qquad \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial b_j} \approx \lim_{h \to 0} \frac{\mathcal{L}(\mathbf{W}, \mathbf{b} + h\mathbf{1}_j) - \mathcal{L}(\mathbf{W}, \mathbf{b} + h\mathbf{1}_j)}{2h}$$

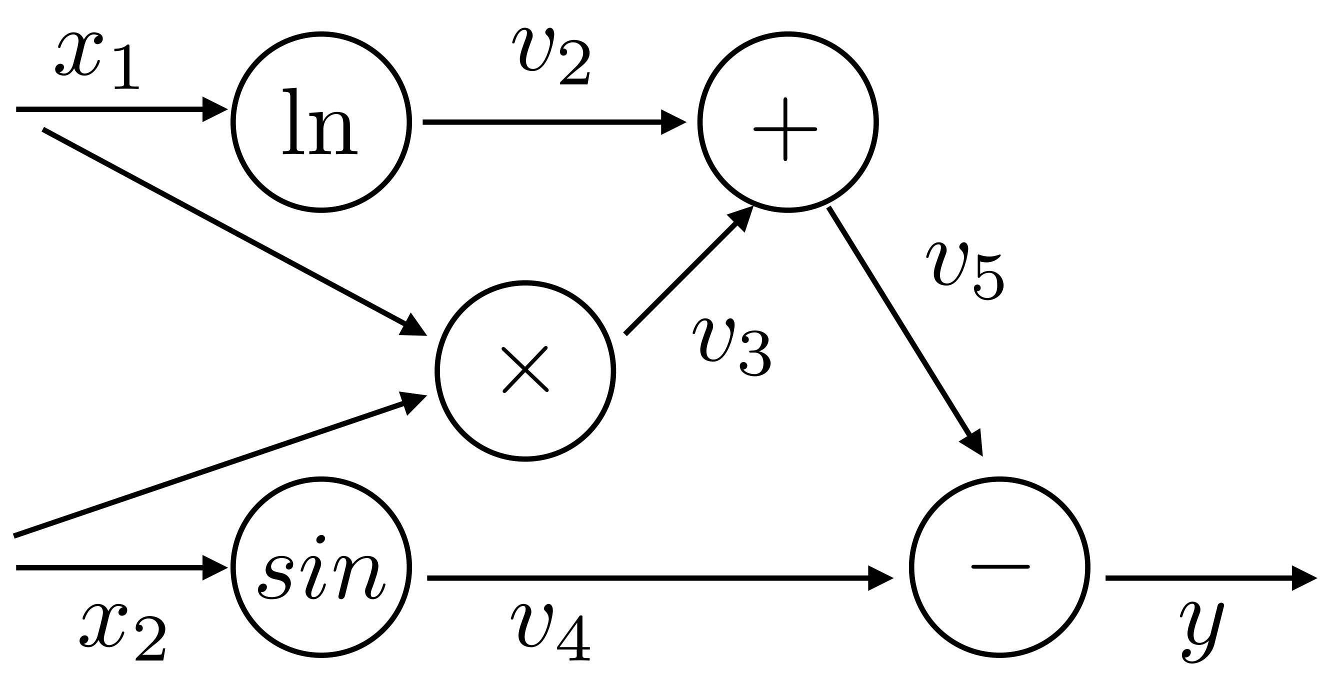
However, both of theses suffer from rounding errors and are not good enough for learning.

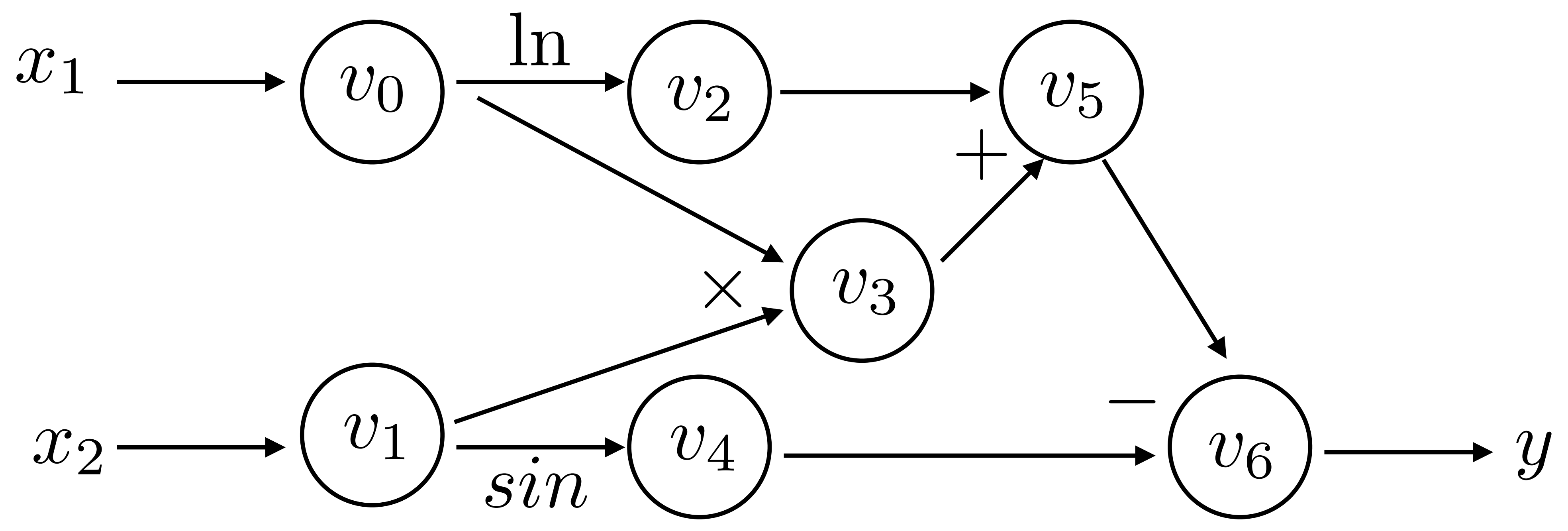
$$h = 0.000001$$

Symbolic Differentiation

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Input function is represented as **computational graph** (a symbolic tree)





Implements differentiation rules for composite functions:

Sum Rule

$$\frac{\mathrm{d}\,(f(x) + g(x))}{\mathrm{d}x} = \frac{\mathrm{d}f(x)}{\mathrm{d}x} + \frac{\mathrm{d}g(x)}{\mathrm{d}x}$$

Product Rule

$$\frac{\mathrm{d}\,(f(x) \cdot g(x))}{\mathrm{d}x} = \frac{\mathrm{d}f(x)}{\mathrm{d}x} g(x) + f(x) \frac{\mathrm{d}g(x)}{\mathrm{d}x}$$

Chain Rule

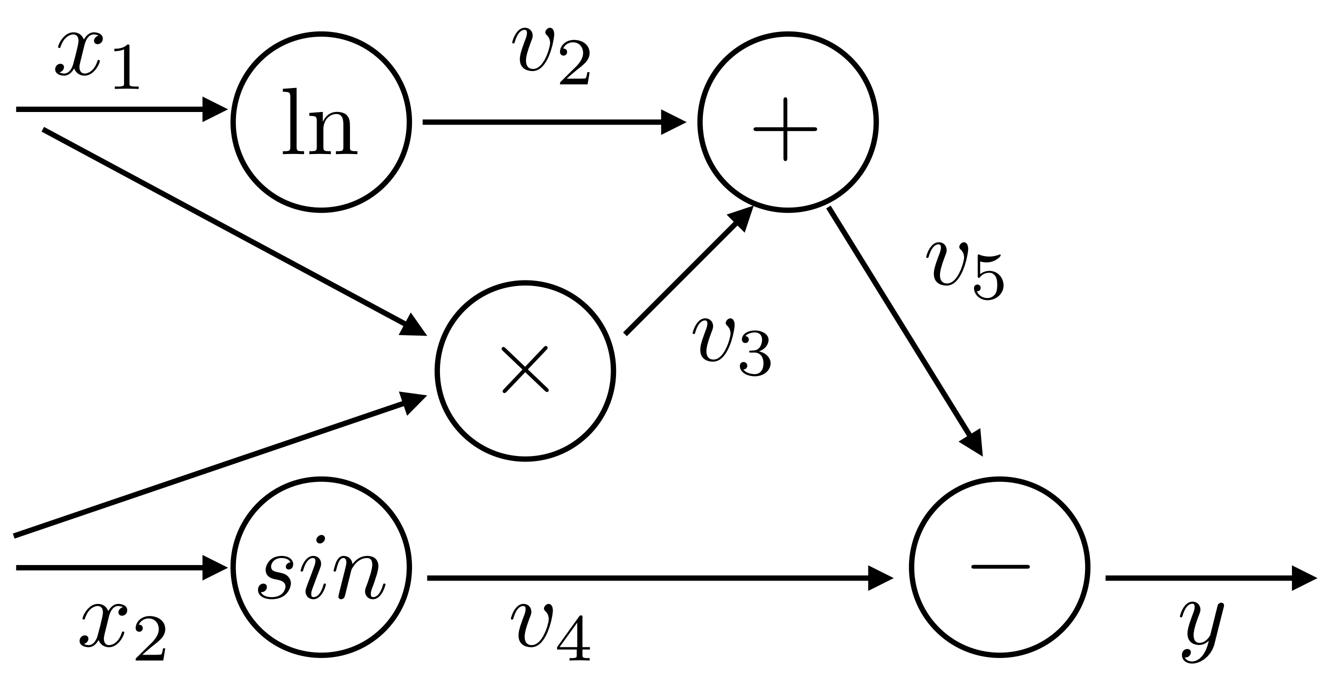
$$\frac{\mathrm{d}(f(g(x)))}{\mathrm{d}x} = \frac{\mathrm{d}f(g(x))}{\mathrm{d}g(x)} \cdot \frac{\mathrm{d}g(x)}{\mathrm{d}x}$$

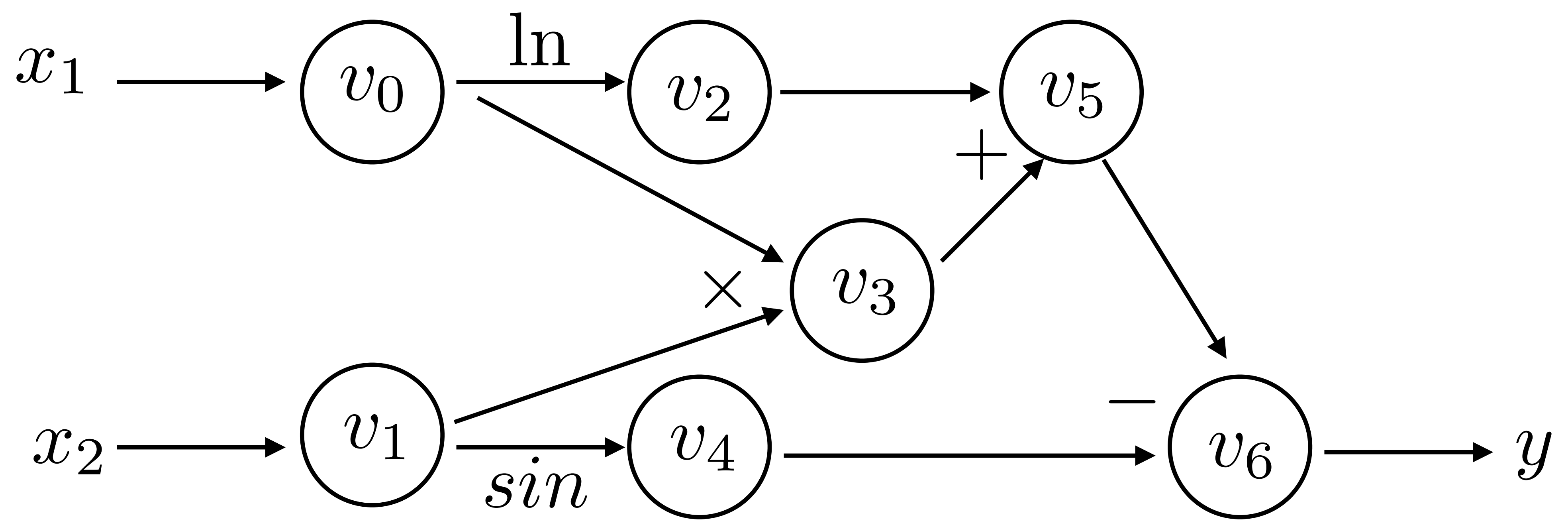
*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Symbolic Differentiation

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Input function is represented as **computational graph** (a symbolic tree)





Implements differentiation rules for composite functions:

Sum Rule

$$\frac{\mathrm{d}\,(f(x) + g(x))}{\mathrm{d}x} = \frac{\mathrm{d}f(x)}{\mathrm{d}x} + \frac{\mathrm{d}g(x)}{\mathrm{d}x}$$

Product Rule

$$\frac{\mathrm{d}\,(f(x) \cdot g(x))}{\mathrm{d}x} = \frac{\mathrm{d}f(x)}{\mathrm{d}x} g(x) + f(x)\frac{\mathrm{d}g(x)}{\mathrm{d}x}$$

Chain Rule

$$\frac{\mathrm{d}(f(g(x)))}{\mathrm{d}x} = \frac{\mathrm{d}f(g(x))}{\mathrm{d}g(x)} \cdot \frac{\mathrm{d}g(x)}{\mathrm{d}x}$$

Problem: For complex functions, expressions can be exponentially large; also difficult to deal with piece-wise functions (creates many symbolic cases)

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Intuition: Interleave symbolic differentiation and simplification

Key Idea: apply symbolic differentiation at the elementary operation level, evaluate and keep intermediate results

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Automatic Differentiation (AutoDiff)

$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$

Intuition: Interleave symbolic differentiation and simplification

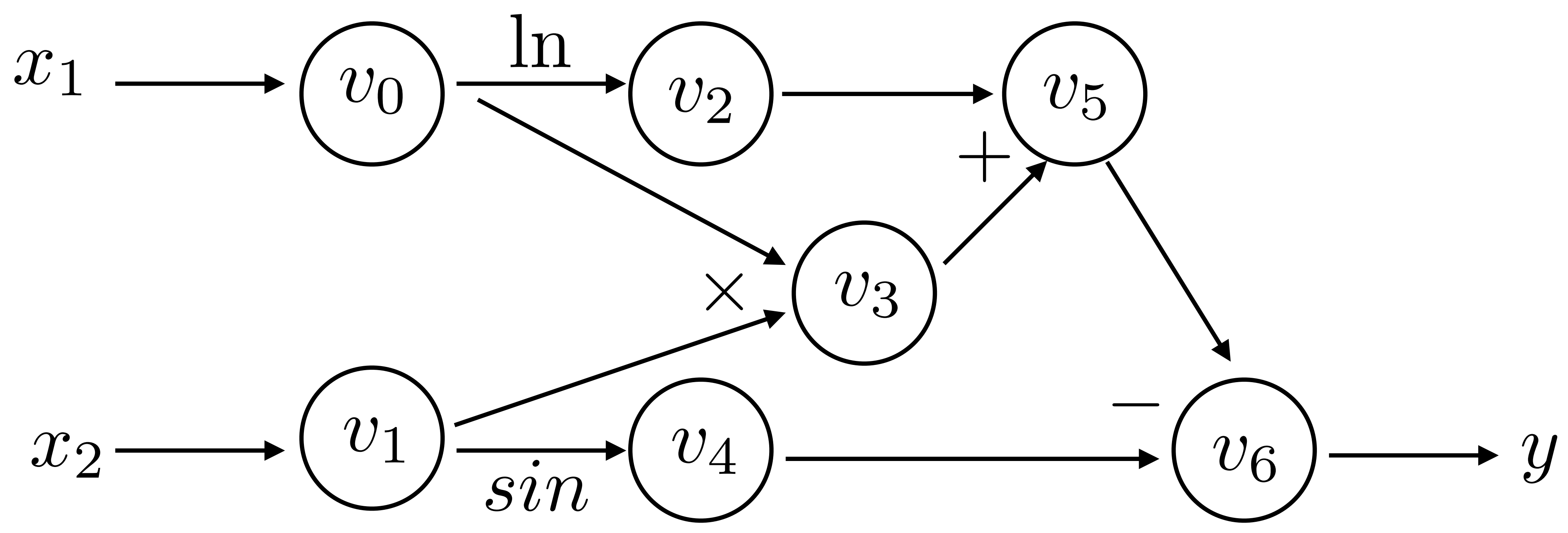
Key Idea: apply symbolic differentiation at the elementary operation level, evaluate and keep intermediate results

Success of **deep learning** owes A LOT to success of AutoDiff algorithms
(also to advances in parallel architectures, and large datasets, …)

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



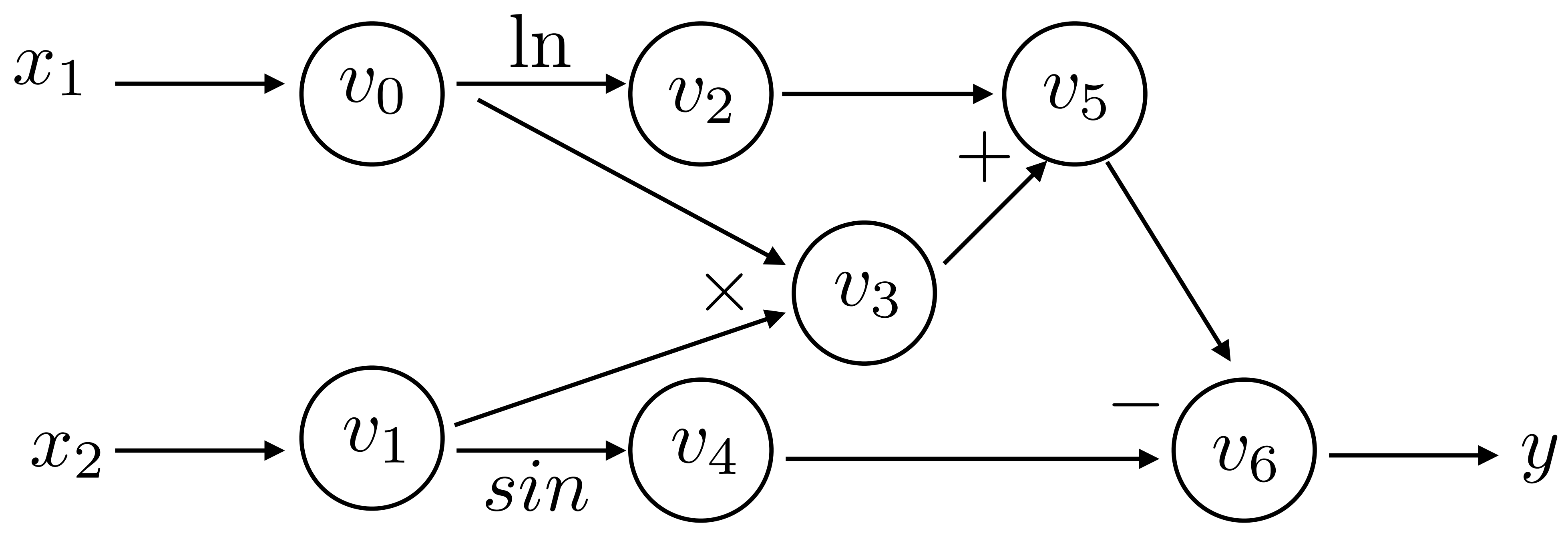
Each **node** is an input, intermediate, or output variable

Computational graph (a DAG) with variable ordering from topological sort.

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Each **node** is an input, intermediate, or output variable

Computational graph (a DAG) with variable ordering from topological sort.

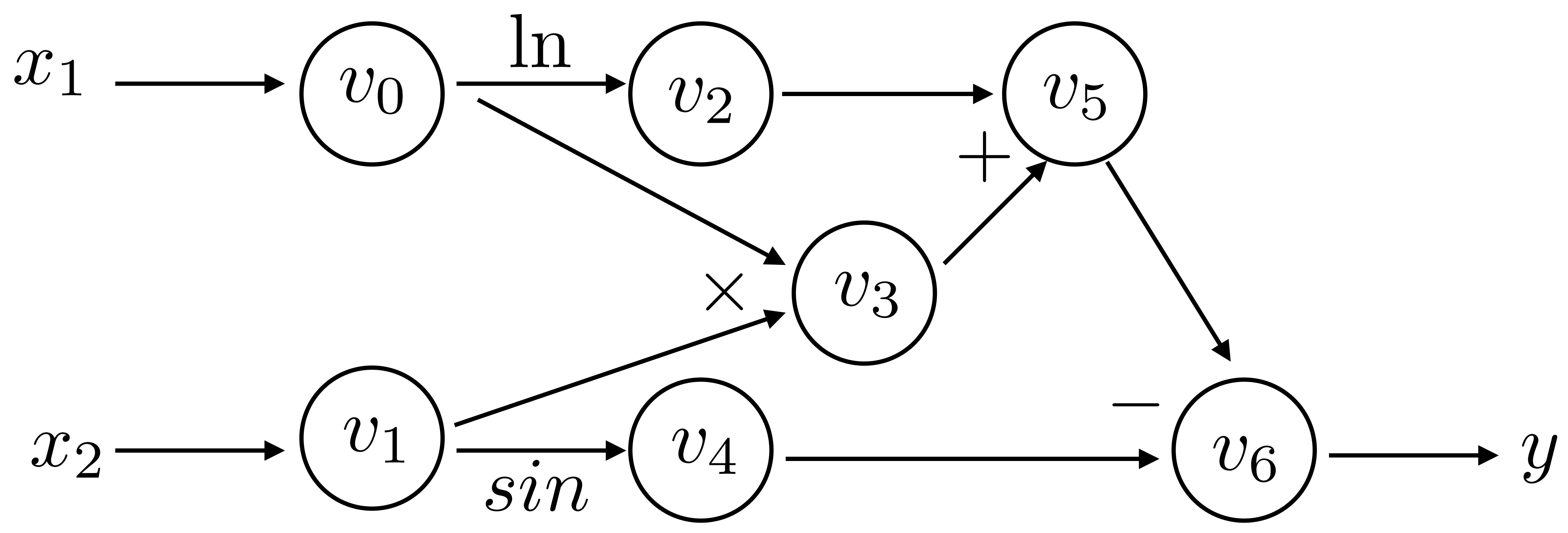
Computational graph is governed by these equations

$$v_0 = x_1$$
$$v_1 = x_2$$
$$v_2 = \ln(v_0)$$
$$v_3 = v_0 \cdot v_1$$
$$v_4 = sin(v_1)$$
$$v_5 = v_2 + v_3$$
$$v_6 = v_5 - v_4$$
$$y = v_6$$

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Lets see how we can **evaluate a function** using computational graph (DNN inferences)

Computational graph is governed by these equations

$$v_0 = x_1$$
$$v_1 = x_2$$
$$v_2 = \ln(v_0)$$
$$v_3 = v_0 \cdot v_1$$
$$v_4 = sin(v_1)$$
$$v_5 = v_2 + v_3$$
$$v_6 = v_5 - v_4$$
$$y = v_6$$

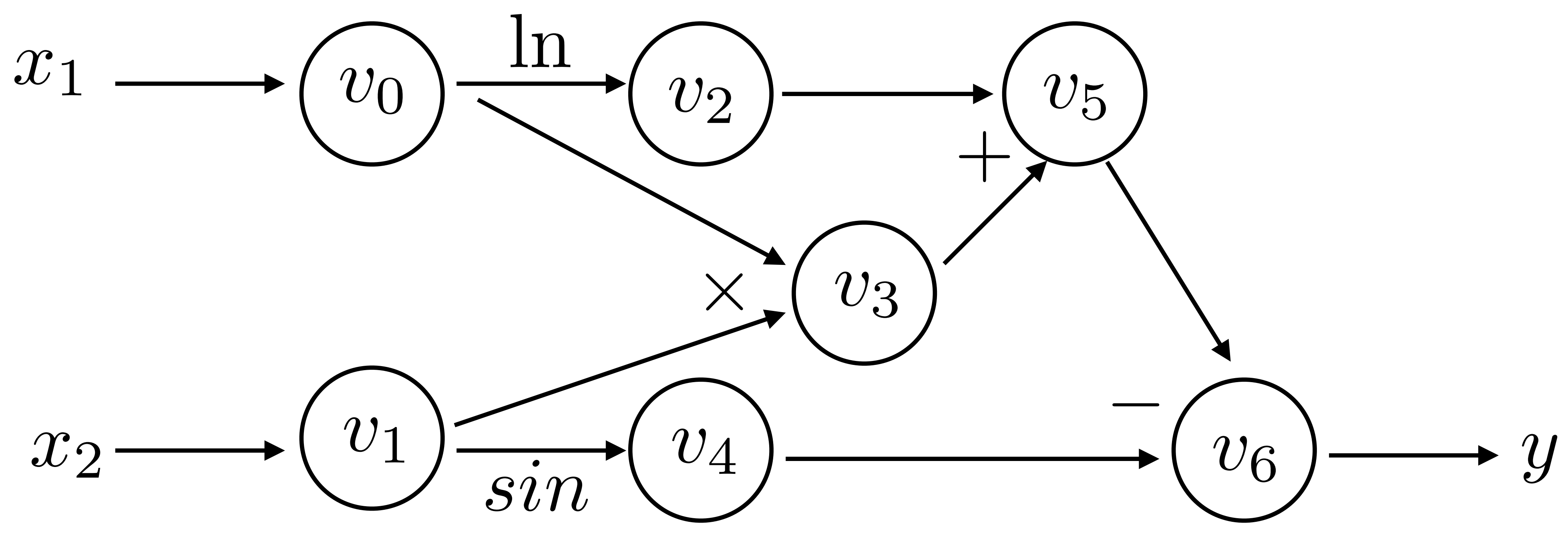
Each **node** is an input, intermediate, or output variable

Computational graph (a DAG) with variable ordering from topological sort.

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Lets see how we can **evaluate a function** using computational graph (DNN inferences)

Forward Evaluation Trace:

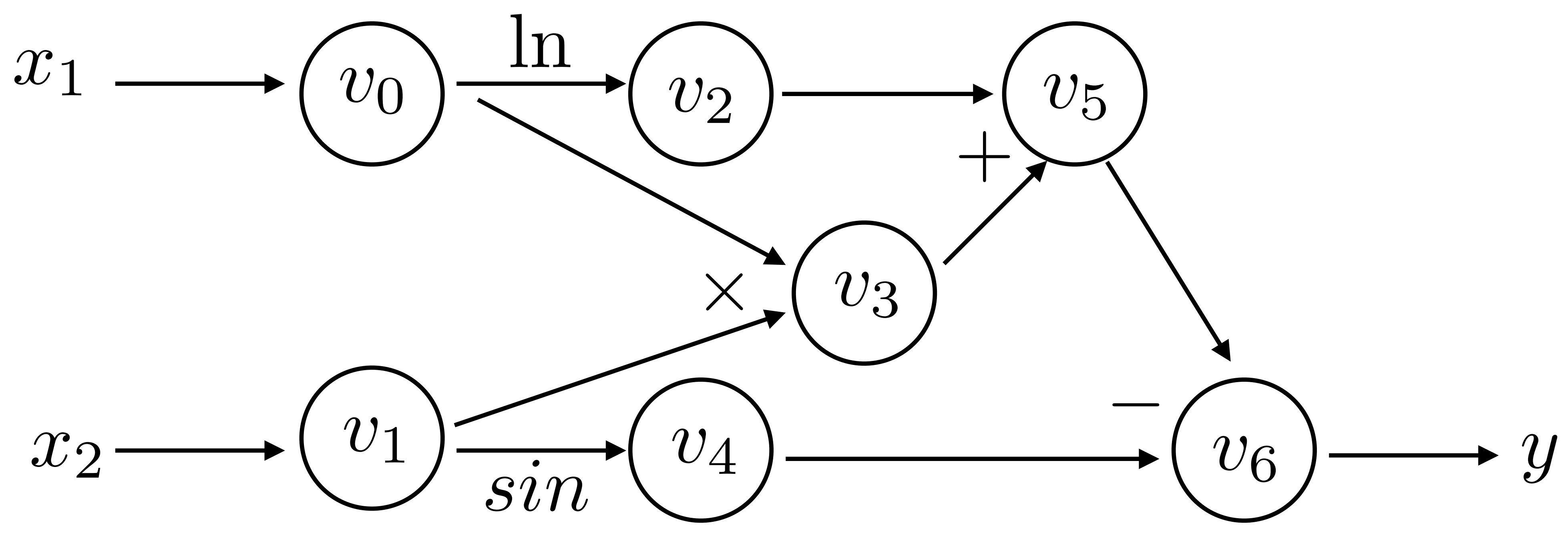
	$f(2,5)$
$v_0 = x_1$	
$v_1 = x_2$	
$v_2 = \ln(v_0)$	
$v_3 = v_0 \cdot v_1$	
$v_4 = sin(v_1)$	
$v_5 = v_2 + v_3$	
$v_6 = v_5 - v_4$	
$y = v_6$	

Each **node** is an input, intermediate, or output variable

Computational graph (a DAG) with variable ordering from topological sort.

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Each **node** is an input, intermediate, or output variable

Computational graph (a DAG) with variable ordering from topological sort.

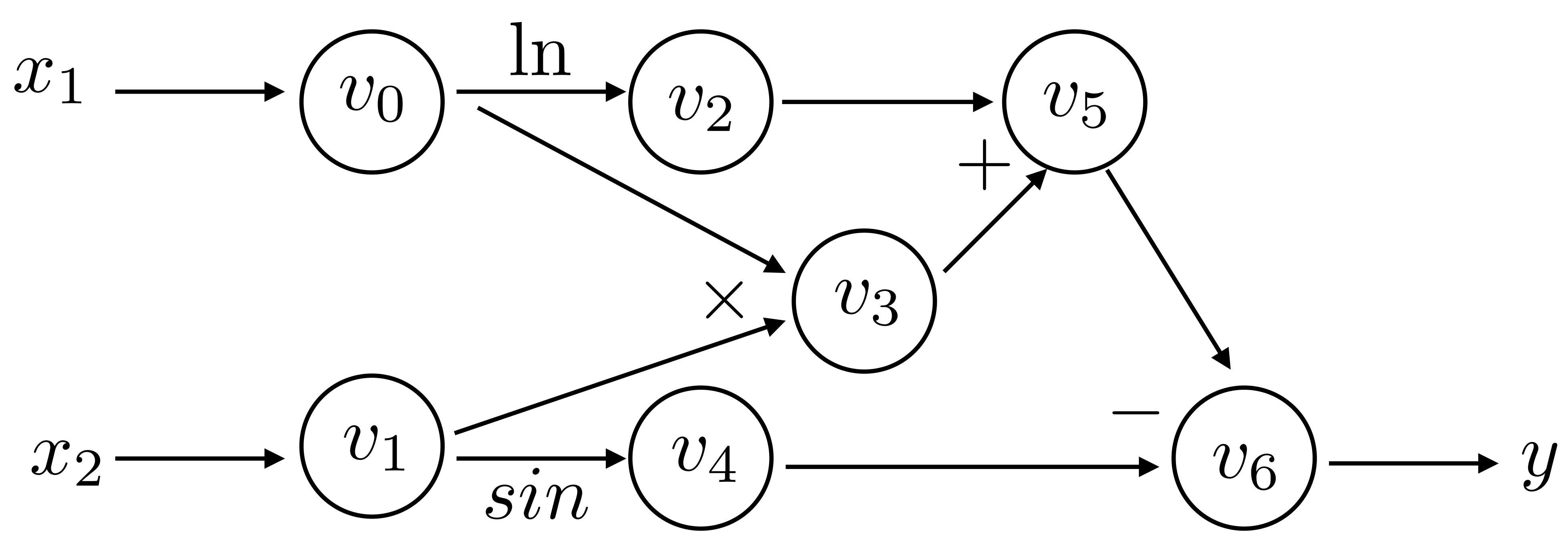
Lets see how we can **evaluate a function** using computational graph (DNN inferences)

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	
$v_2 = \ln(v_0)$	
$v_3 = v_0 \cdot v_1$	
$v_4 = sin(v_1)$	
$v_5 = v_2 + v_3$	
$v_6 = v_5 - v_4$	
$y = v_6$	

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Lets see how we can **evaluate a function** using computational graph (DNN inferences)

Each **node** is an input, intermediate, or output variable

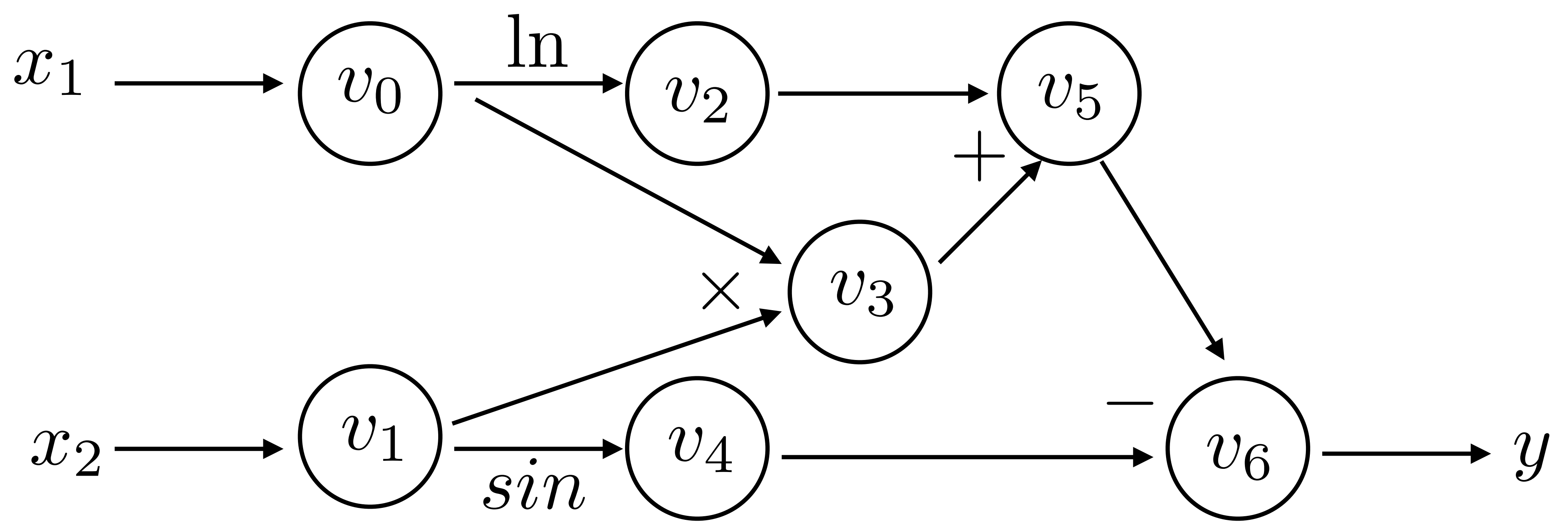
Computational graph (a DAG) with variable ordering from topological sort.

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	
$v_3 = v_0 \cdot v_1$	
$v_4 = sin(v_1)$	
$v_5 = v_2 + v_3$	
$v_6 = v_5 - v_4$	
$y = v_6$	

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Each **node** is an input, intermediate, or output variable

Computational graph (a DAG) with variable ordering from topological sort.

Lets see how we can **evaluate a function** using computational graph (DNN inferences)

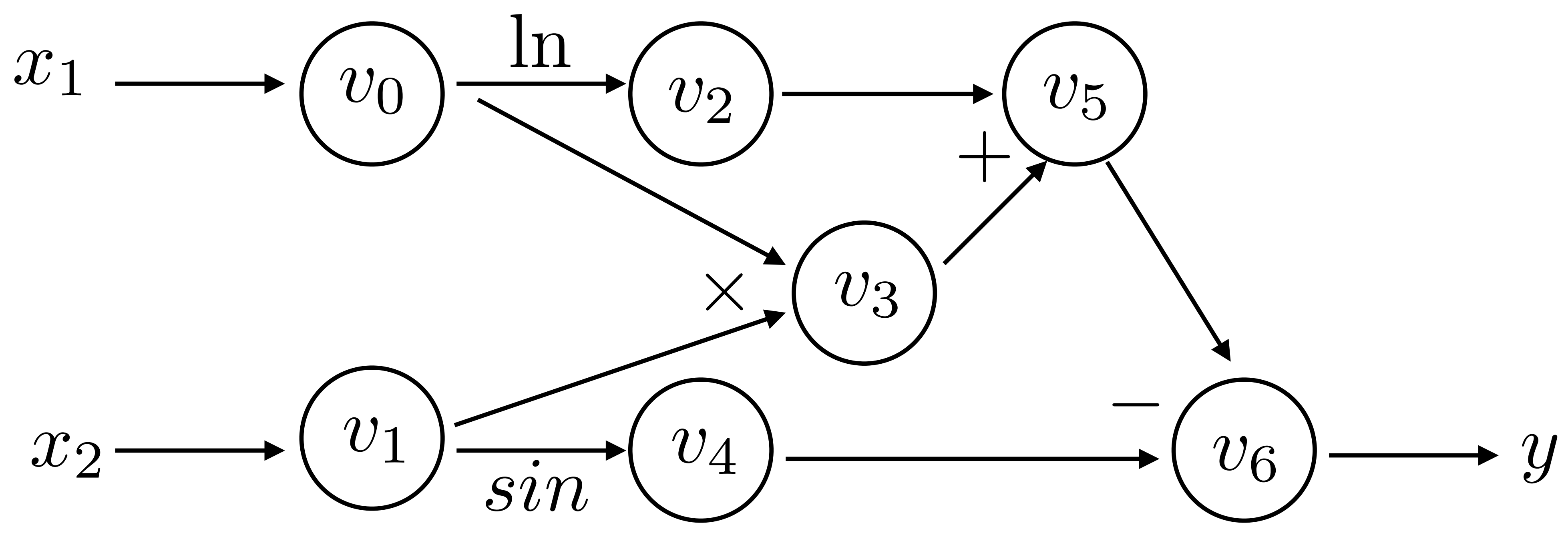
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	
$v_4 = sin(v_1)$	
$v_5 = v_2 + v_3$	
$v_6 = v_5 - v_4$	
$y = v_6$	

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Lets see how we can **evaluate a function** using computational graph (DNN inferences)



Each **node** is an input, intermediate, or output variable

Computational graph (a DAG) with variable ordering from topological sort.

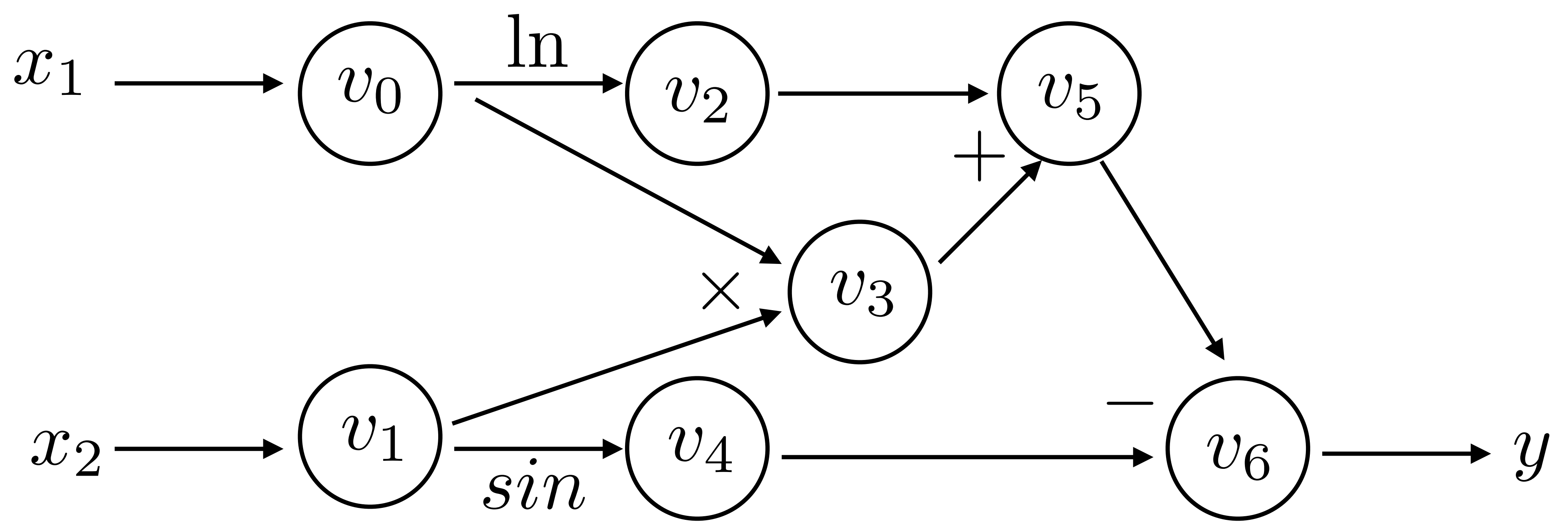
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Lets see how we can **evaluate a function** using computational graph (DNN inferences)



Forward Evaluation Trace:

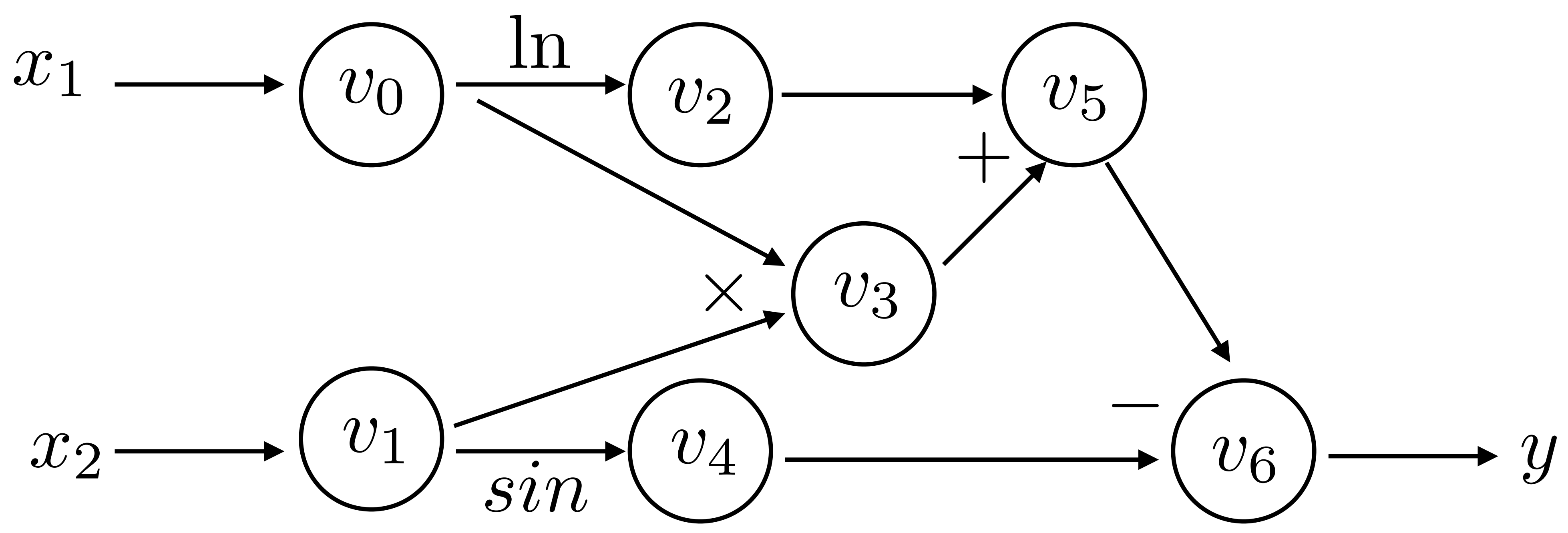
	$f(2, 5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Each **node** is an input, intermediate, or output variable

Computational graph (a DAG) with variable ordering from topological sort.

Automatic Differentiation (AutoDiff)

$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$

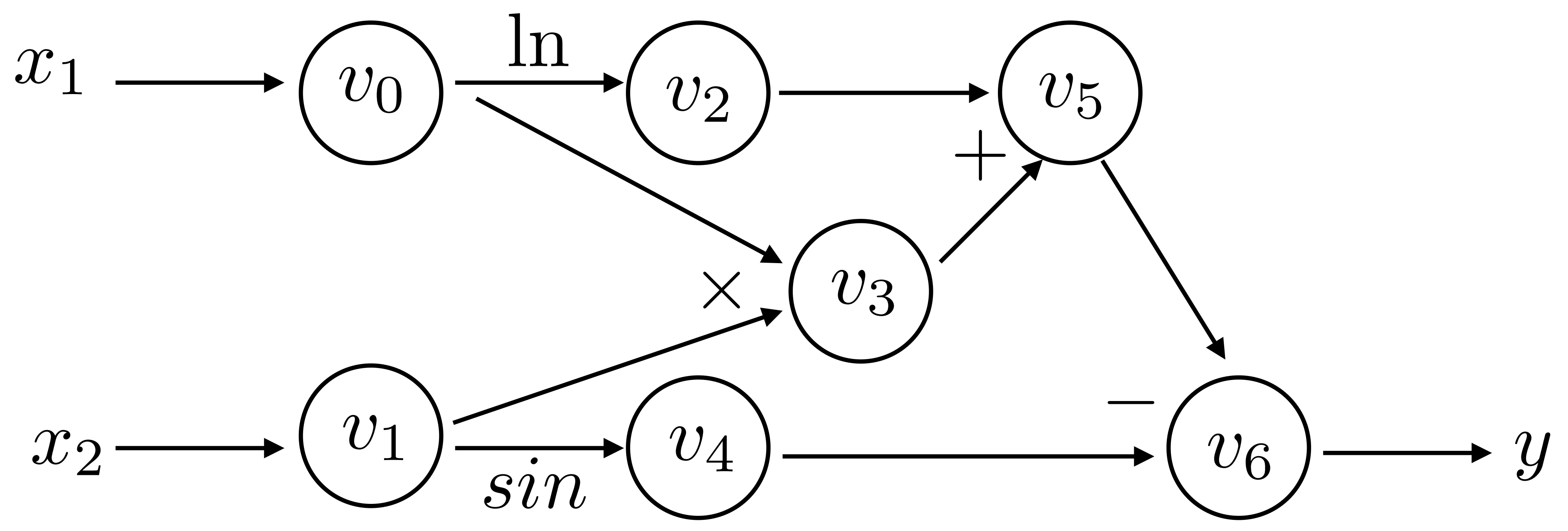


Forward Evaluation Trace:

	$f(2, 5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Lets see how we can **evaluate a derivative** using computational graph (DNN learning)

$$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right|_{(x_1=2, x_2=5)}$$

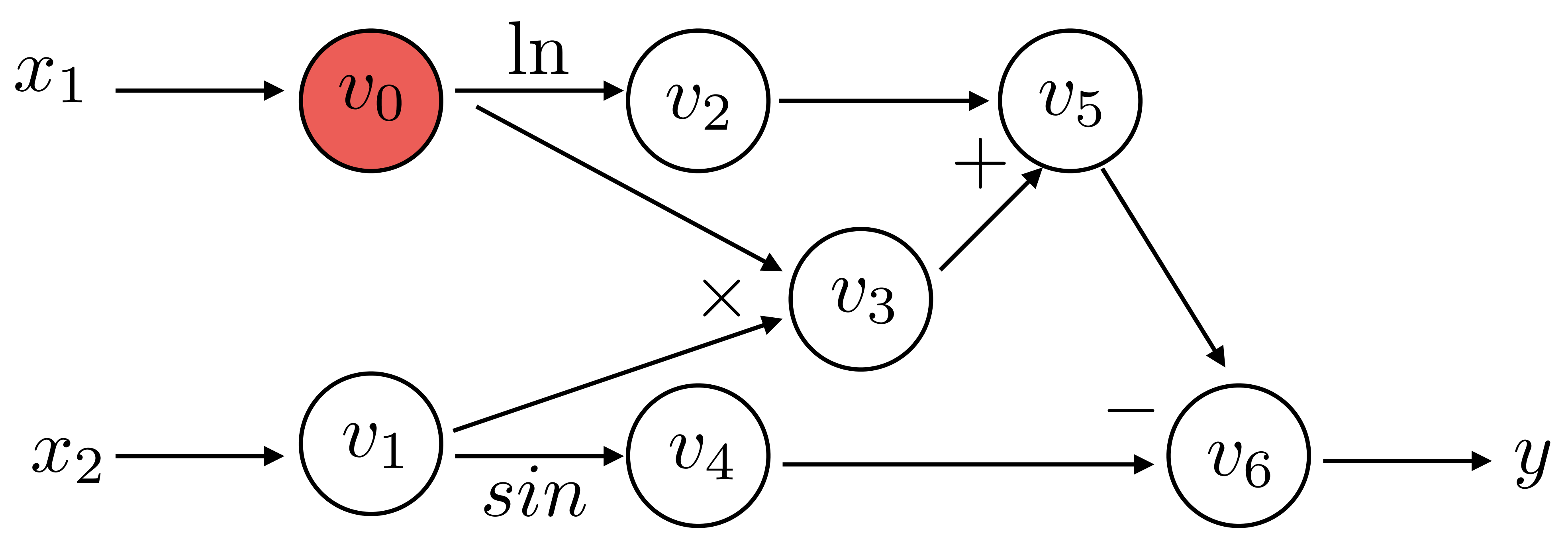
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

We will do this with **forward mode** first, by introducing a derivative of each variable node with respect to the input variable.

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

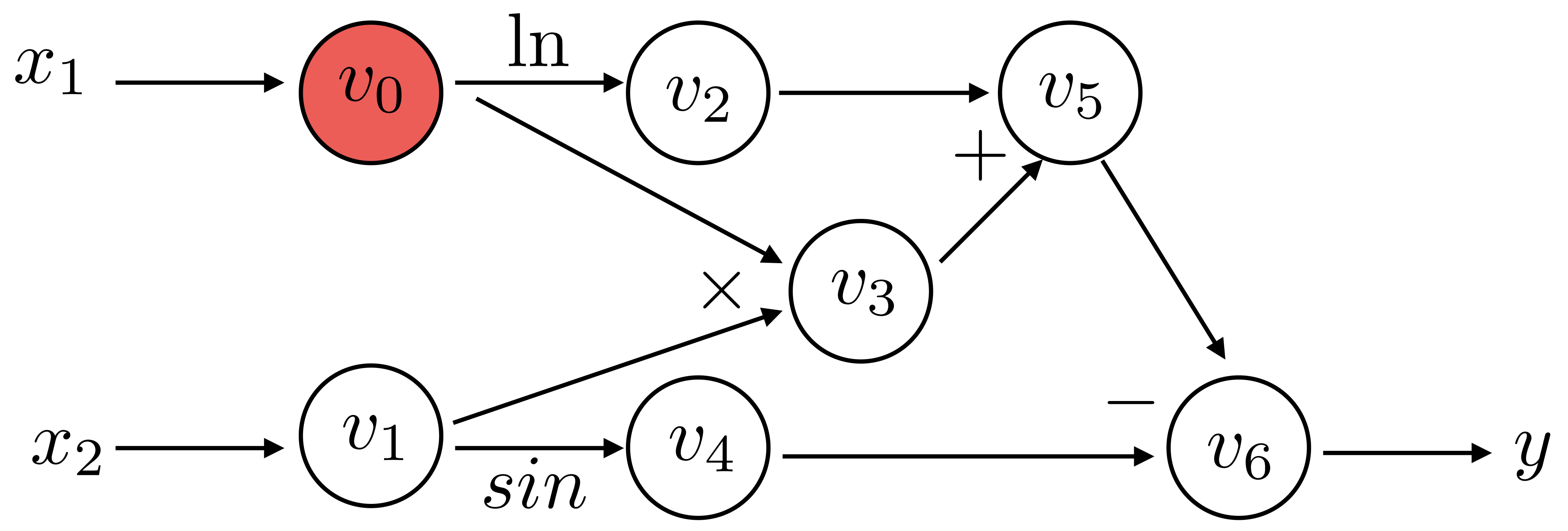
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\vert_{(x_1=2, x_2=5)}$

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

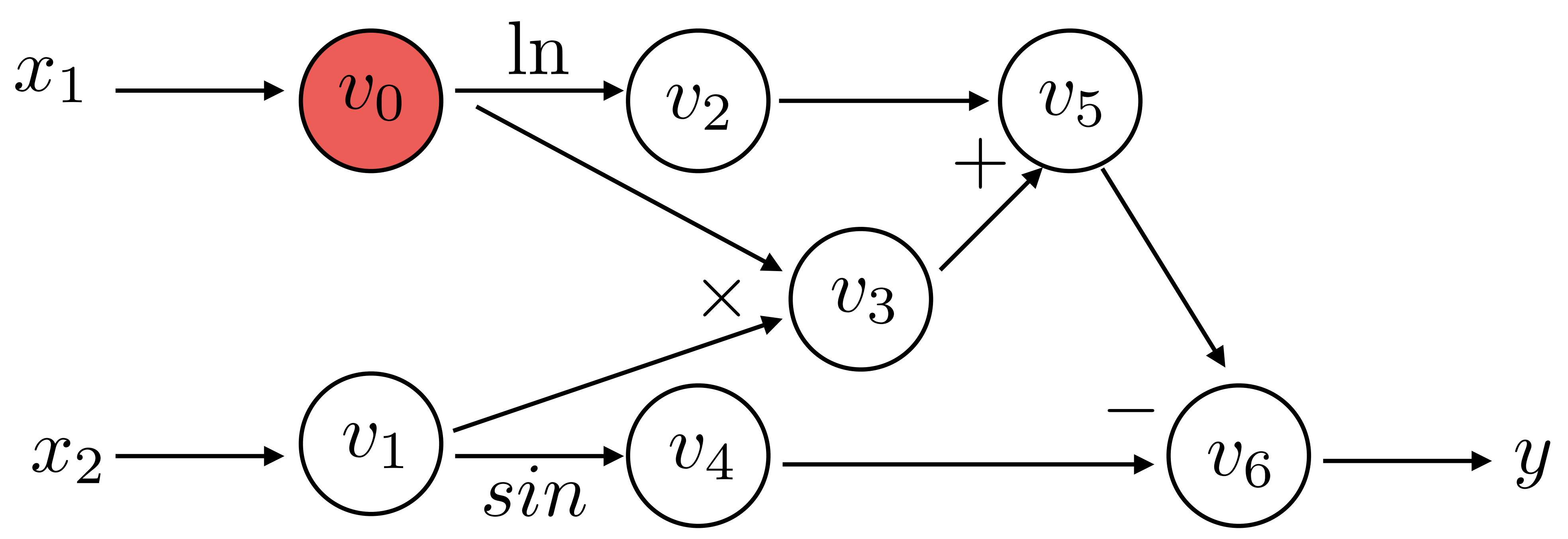
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\rvert_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

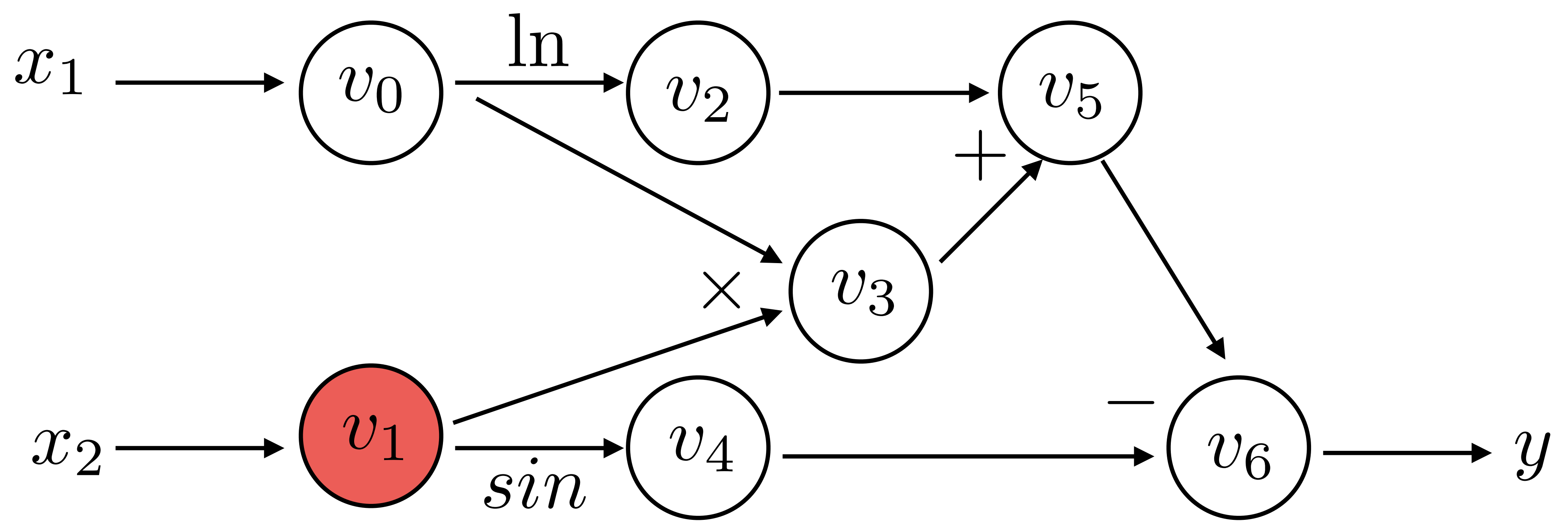
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\rvert_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

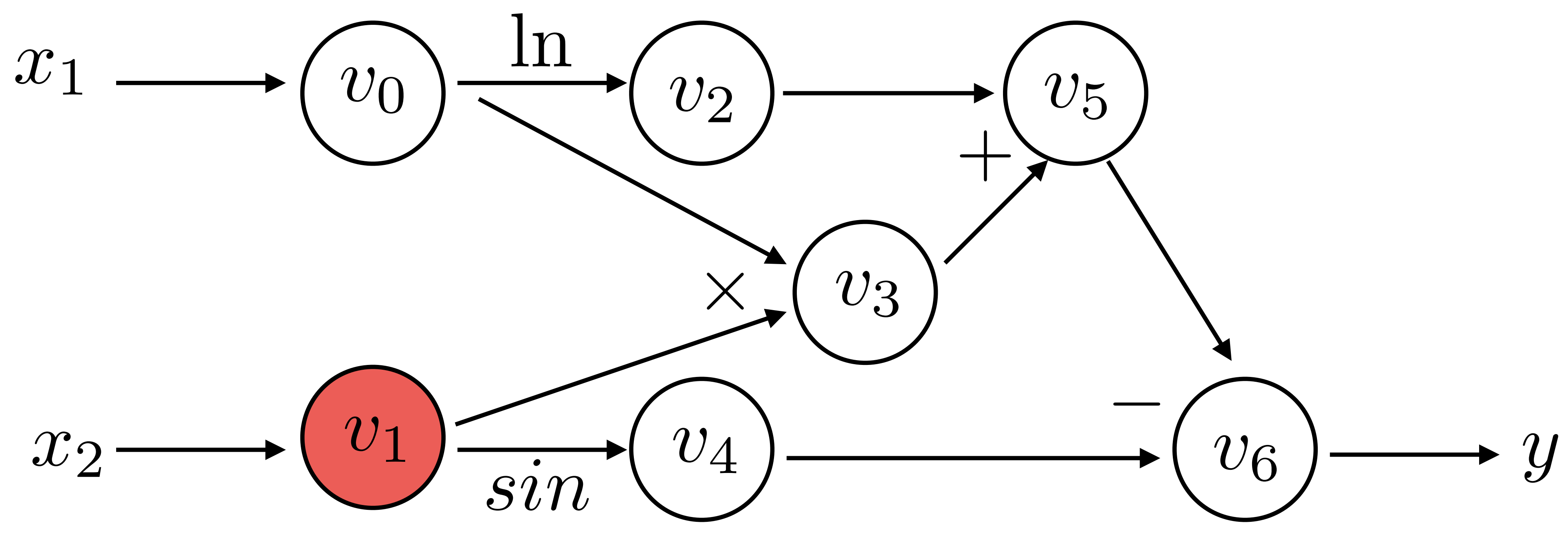
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\vert_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

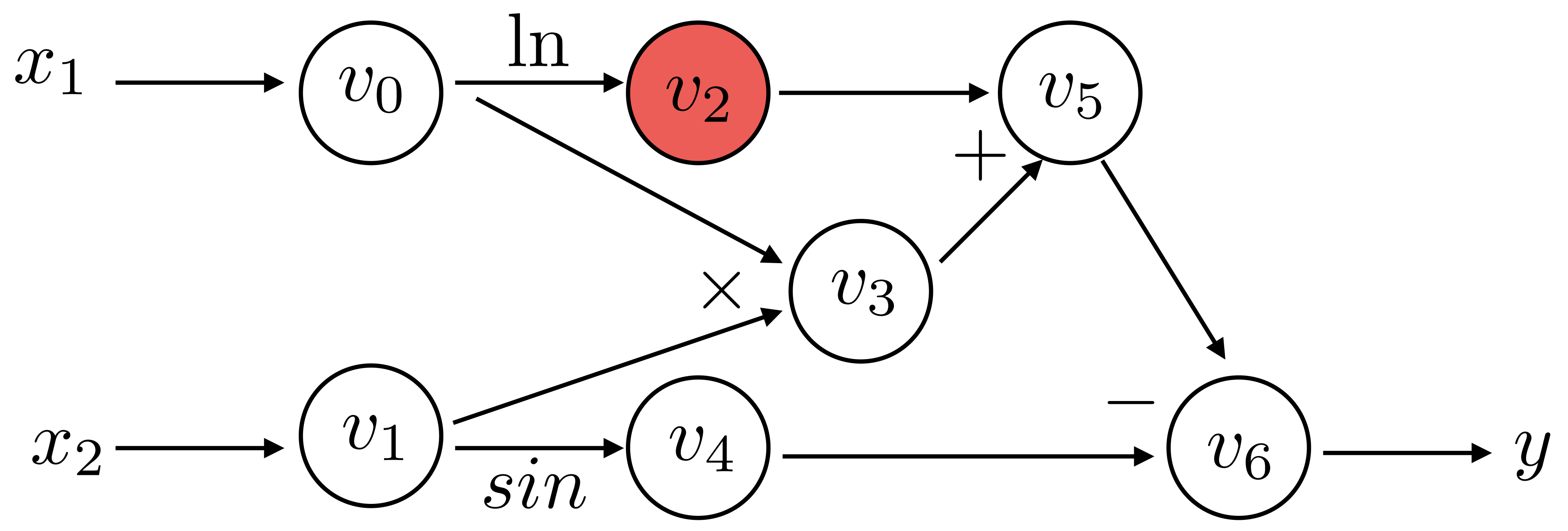
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\frac{\partial f(x_1, x_2)}{\partial x_1}\Big\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

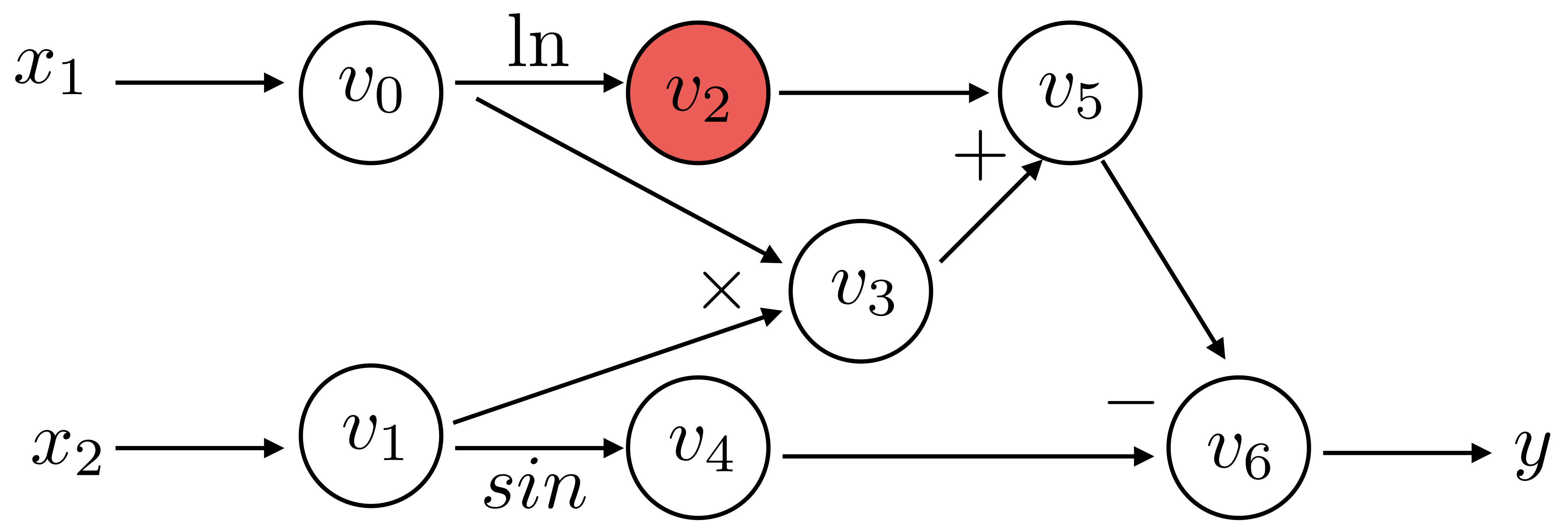
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1}$	

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

	$f(2, 5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

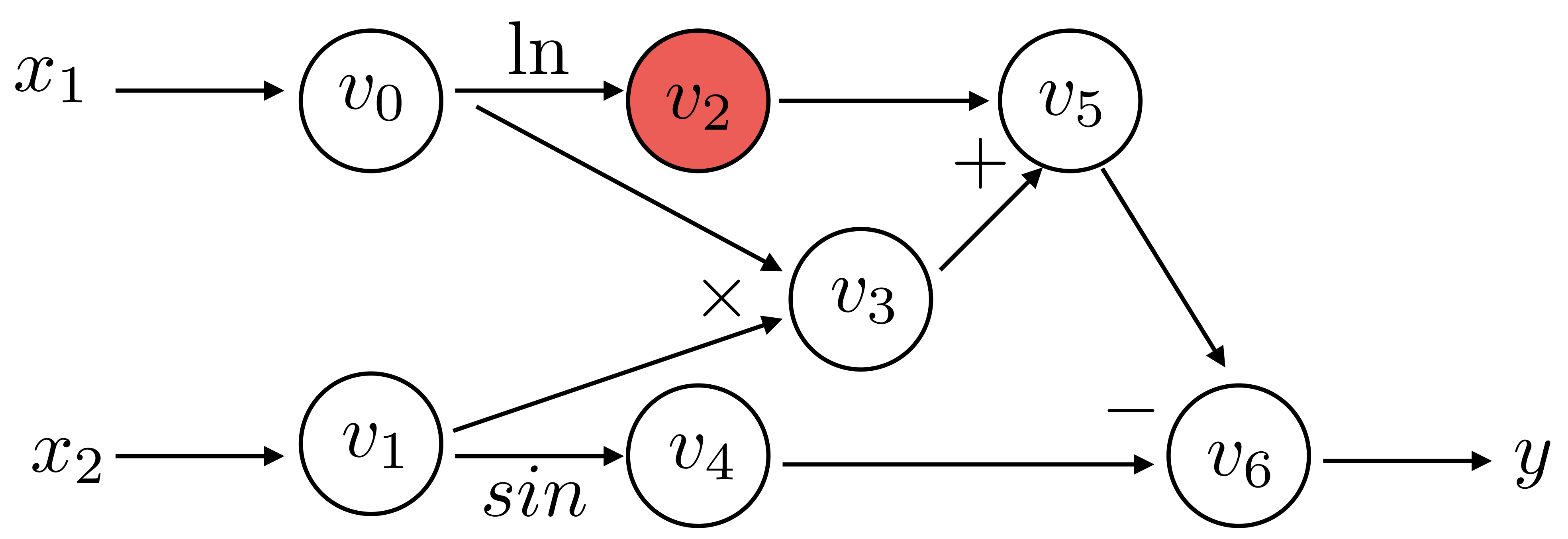
Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\vert_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1}$	

Chain Rule

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

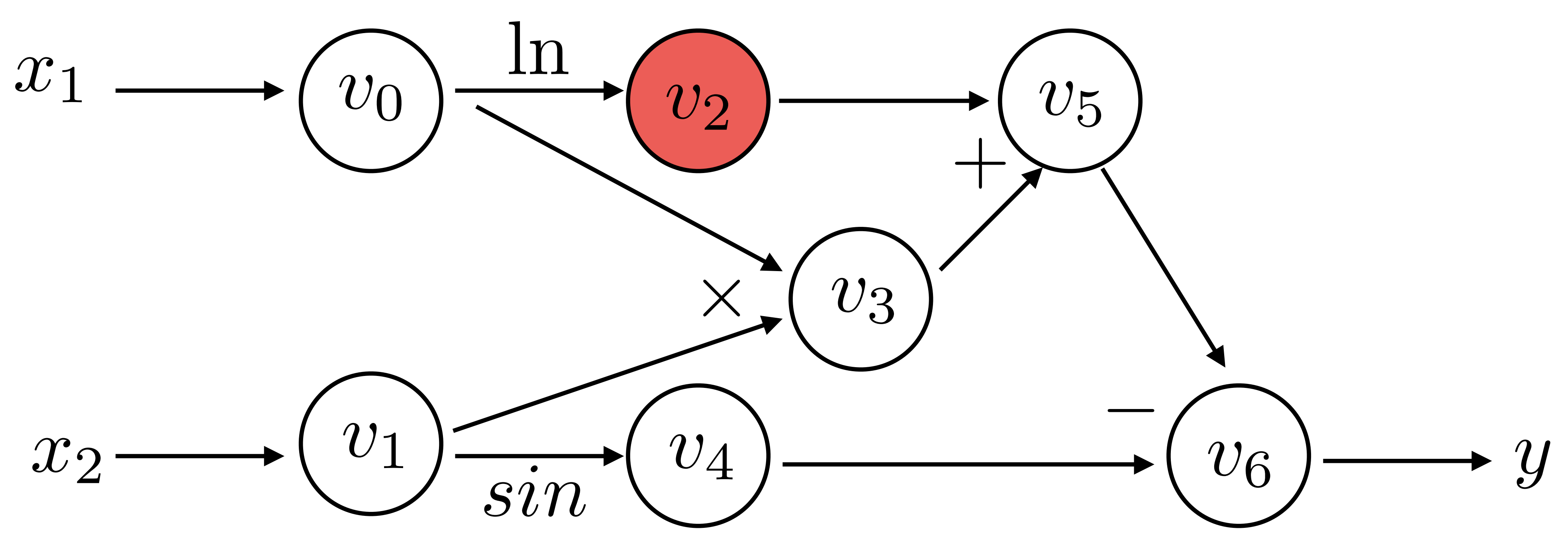
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\frac{\partial f(x_1, x_2)}{\partial x_1}\Big\vert_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0} \frac{\partial v_0}{\partial x_1}$	
Chain Rule	

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

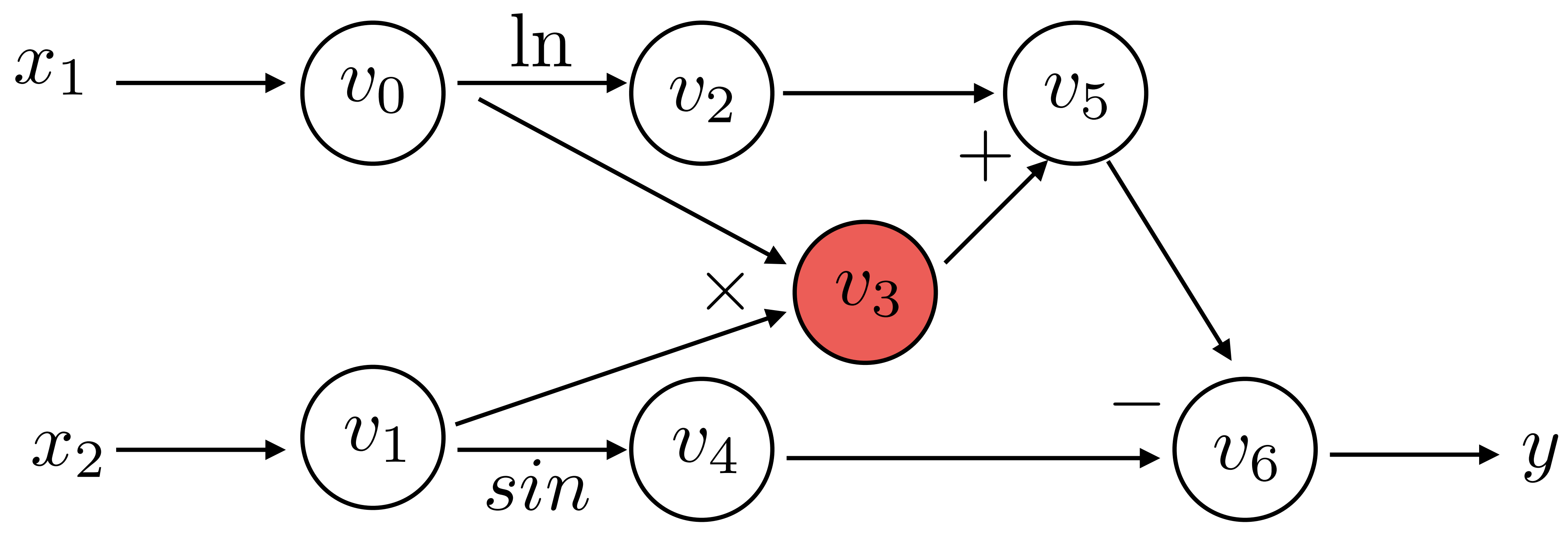
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1,x_2)}{\partial x_1}\right\vert_{(x_1=2,x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
Chain Rule	

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

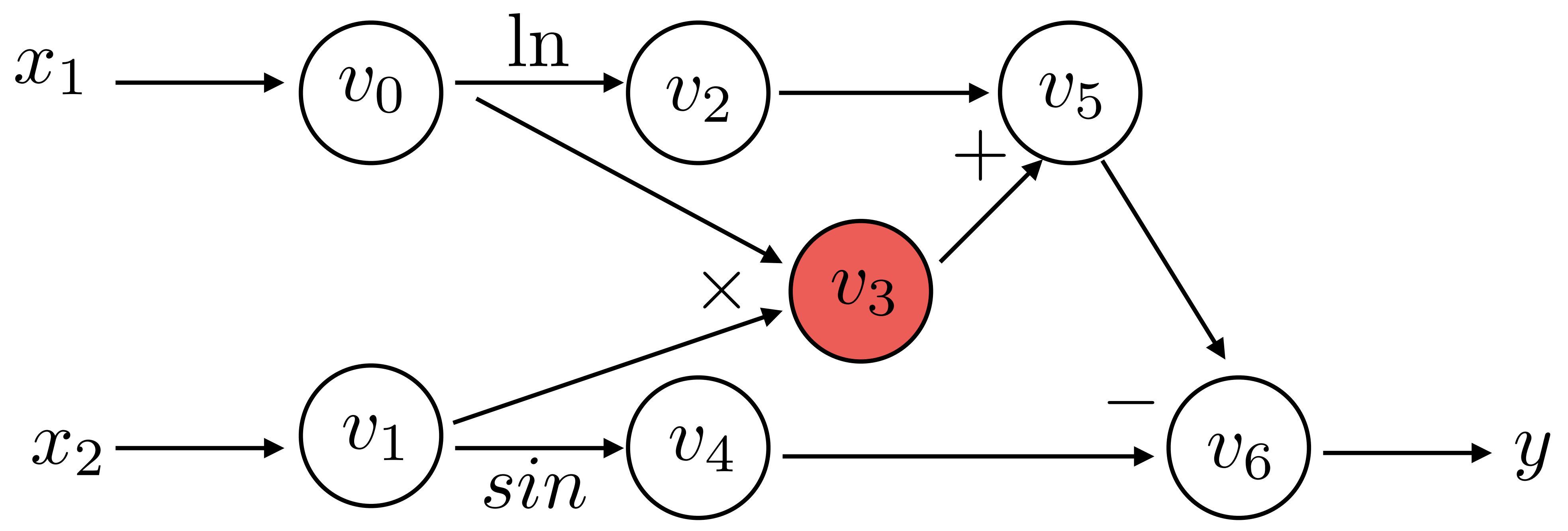
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\frac{\partial f(x_1,x_2)}{\partial x_1}\Big\|_{(x_1=2,x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1}$	

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

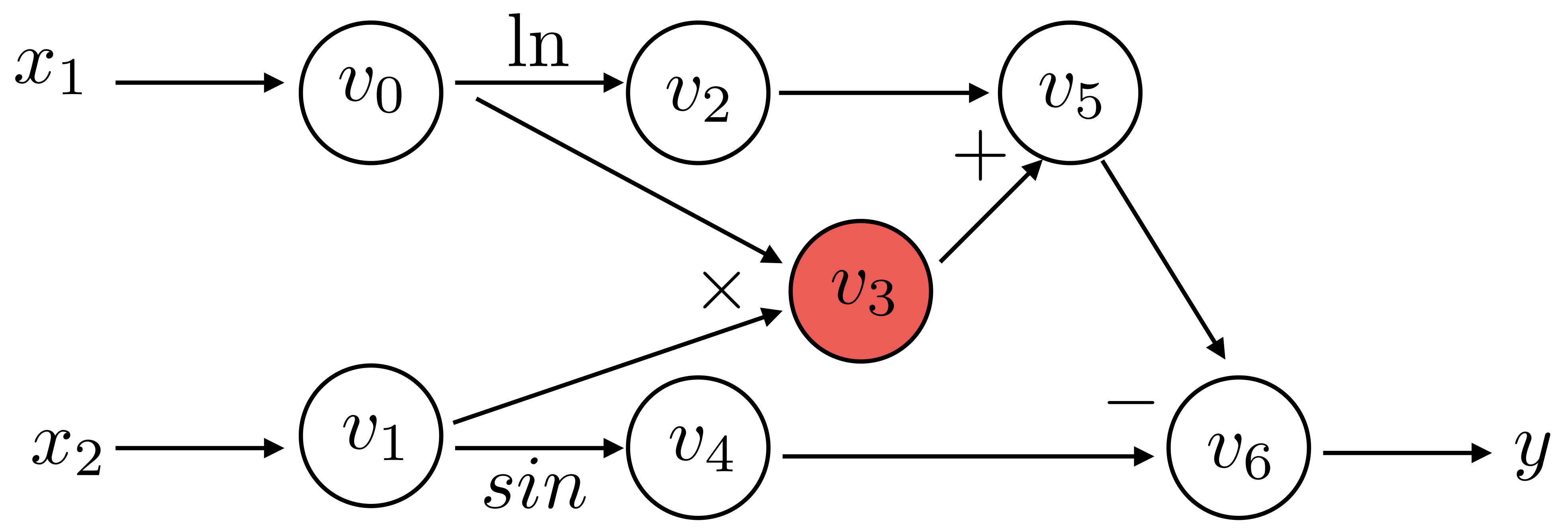
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1,x_2)}{\partial x_1}\right\vert_{(x_1=2,x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1}$	
Product Rule	

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

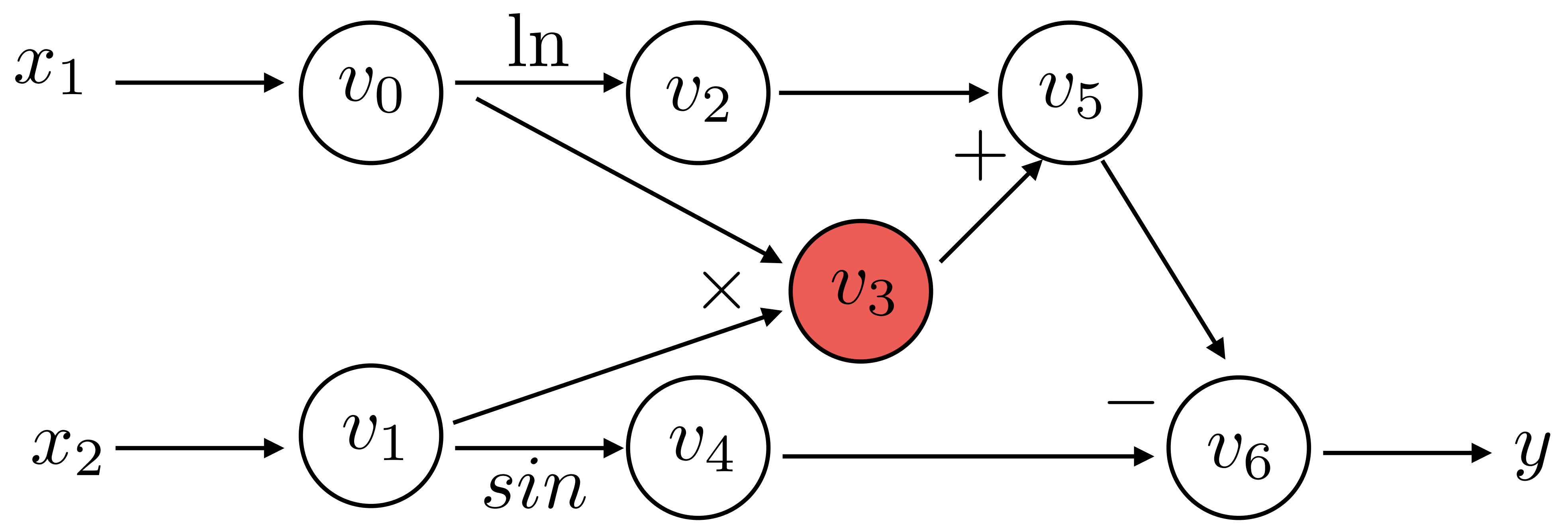
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1,x_2)}{\partial x_1}\right\|_{(x_1=2,x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_0}{\partial x_1} \cdot v_1 + v_0 \cdot \frac{\partial v_1}{\partial x_1}$ **Product** Rule	

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

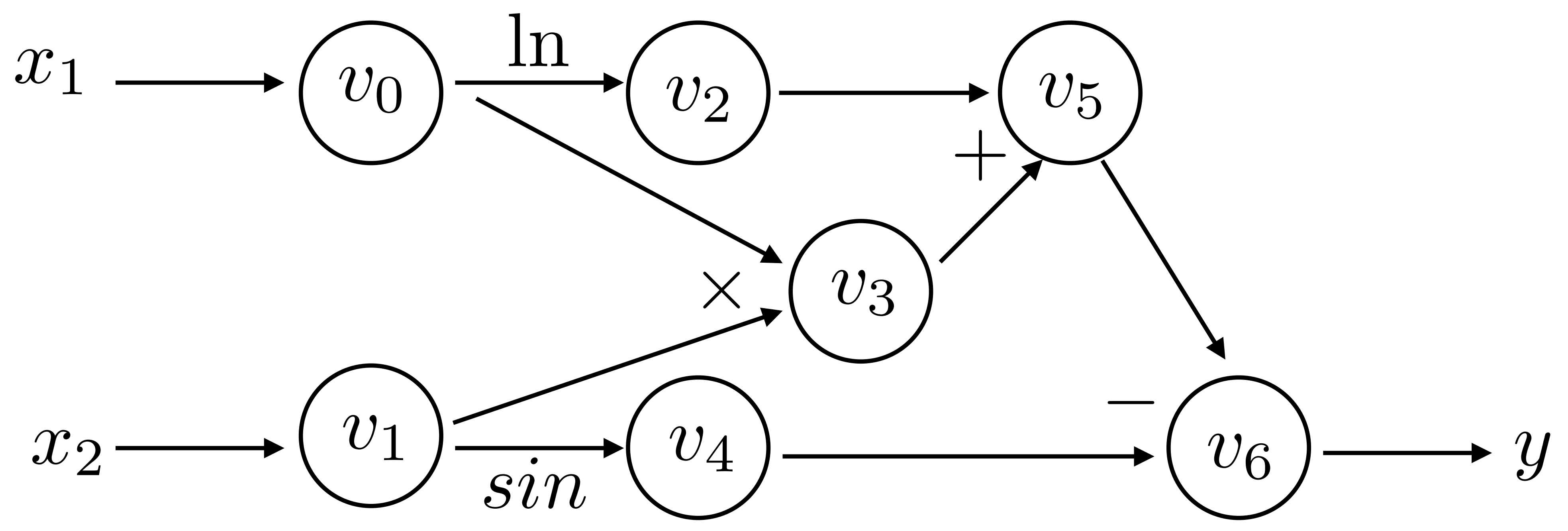
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1,x_2)}{\partial x_1}\right\rvert_{(x_1=2,x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_0}{\partial x_1} \cdot v_1 + v_0 \cdot \frac{\partial v_1}{\partial x_1}$ **Product** Rule	1*5 + 2*0 = 5

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

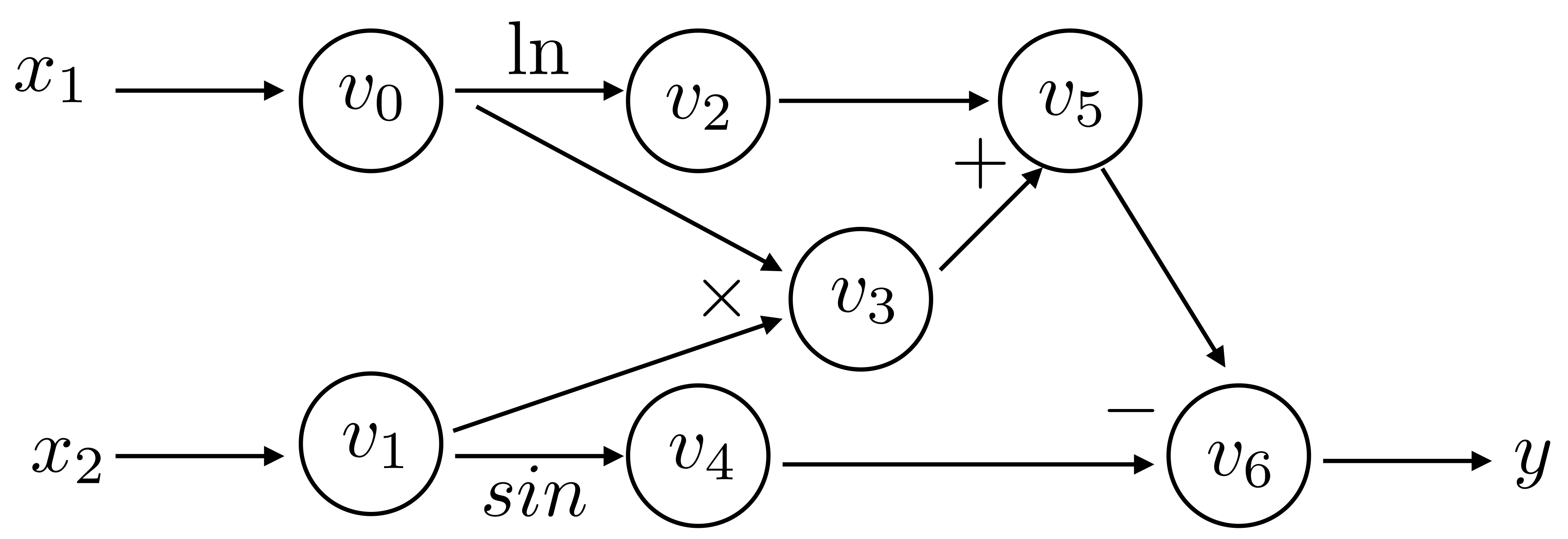
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1,x_2)}{\partial x_1}\right\vert_{(x_1=2,x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_0}{\partial x_1} \cdot v_1 + v_0 \cdot \frac{\partial v_1}{\partial x_1}$	1*5 + 2*0 = 5
$\frac{\partial v_4}{\partial x_1} = \frac{\partial v_1}{\partial x_1} cos(v_1)$	0 * cos(5) = 0
$\frac{\partial v_5}{\partial x_1} = \frac{\partial v_2}{\partial x_1} + \frac{\partial v_3}{\partial x_1}$	0.5 + 5 = 5.5
$\frac{\partial v_6}{\partial x_1} = \frac{\partial v_5}{\partial x_1} - \frac{\partial v_4}{\partial x_1}$	5.5 - 0 = 5.5
$\frac{\partial y}{\partial x_1} = \frac{\partial v_6}{\partial x_1}$	5.5

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



We now have:

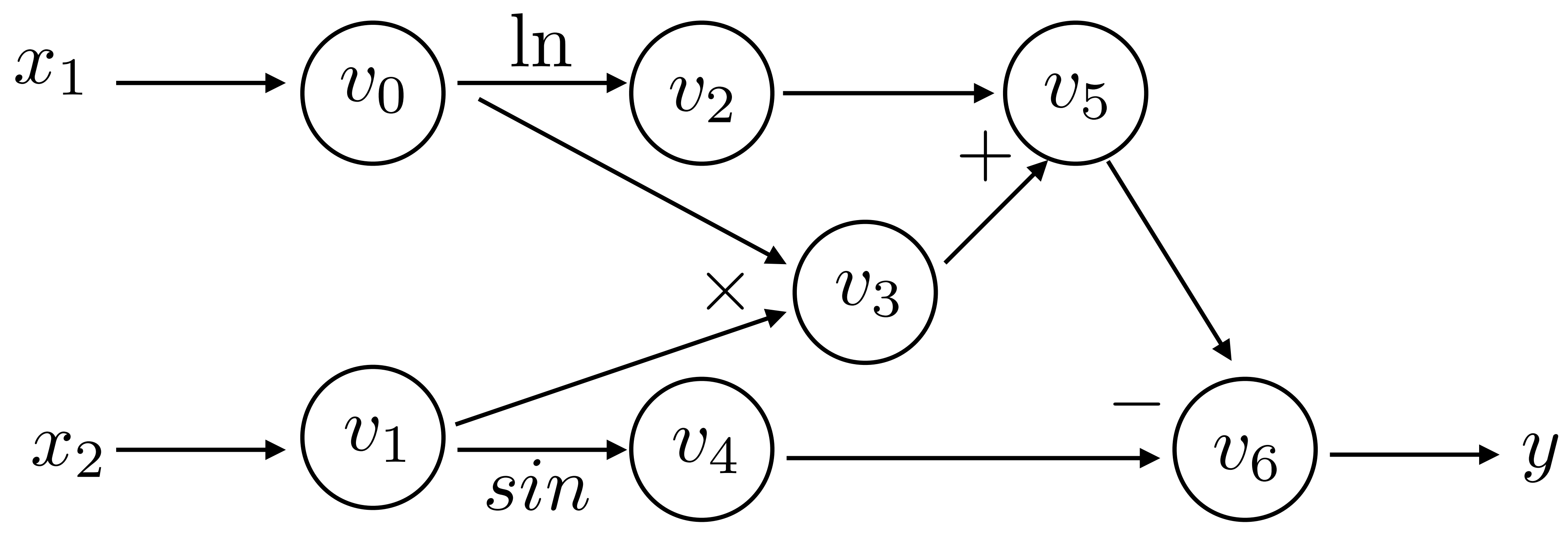
$$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right|_{(x_1=2, x_2=5)} = 5.5$$

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0} \frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_0}{\partial x_1} \cdot v_1 + v_0 \cdot \frac{\partial v_1}{\partial x_1}$	1*5 + 2*0 = 5
$\frac{\partial v_4}{\partial x_1} = \frac{\partial v_1}{\partial x_1} cos(v_1)$	0 * cos(5) = 0
$\frac{\partial v_5}{\partial x_1} = \frac{\partial v_2}{\partial x_1} + \frac{\partial v_3}{\partial x_1}$	0.5 + 5 = 5.5
$\frac{\partial v_6}{\partial x_1} = \frac{\partial v_5}{\partial x_1} - \frac{\partial v_4}{\partial x_1}$	5.5 - 0 = 5.5
$\frac{\partial y}{\partial x_1} = \frac{\partial v_6}{\partial x_1}$	5.5

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



We now have:

$$\left.\frac{\partial f(x_1,x_2)}{\partial x_1}\right|_{(x_1=2,x_2=5)} = 5.5$$

Still need:

$$\left.\frac{\partial f(x_1,x_2)}{\partial x_2}\right|_{(x_1=2,x_2=5)}$$

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1,x_2)}{\partial x_1}\right\|_{(x_1=2,x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_0}{\partial x_1} \cdot v_1 + v_0 \cdot \frac{\partial v_1}{\partial x_1}$	1*5 + 2*0 = 5
$\frac{\partial v_4}{\partial x_1} = \frac{\partial v_1}{\partial x_1} cos(v_1)$	0 * cos(5) = 0
$\frac{\partial v_5}{\partial x_1} = \frac{\partial v_2}{\partial x_1} + \frac{\partial v_3}{\partial x_1}$	0.5 + 5 = 5.5
$\frac{\partial v_6}{\partial x_1} = \frac{\partial v_5}{\partial x_1} - \frac{\partial v_4}{\partial x_1}$	5.5 - 0 = 5.5
$\frac{\partial y}{\partial x_1} = \frac{\partial v_6}{\partial x_1}$	5.5

AutoDiff - **Forward Mode**

Forward mode needs m forward passes to get a full Jacobian (all gradients of output with respect to each input), where m is the number of inputs

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}^n$$

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

AutoDiff - **Forward Mode**

Forward mode needs m forward passes to get a full Jacobian (all gradients of output with respect to each input), where m is the number of inputs

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^m \rightarrow \mathbb{R}^n$$

Problem: DNN typically has large number of inputs:

image as an input, plus all the weights and biases of layers = millions of inputs!

and very few outputs (many DNNs have $n = 1$)

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

AutoDiff - **Forward Mode**

Forward mode needs m forward passes to get a full Jacobian (all gradients of output with respect to each input), where m is the number of inputs

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}^n$$

Problem: DNN typically has large number of inputs:

image as an input, plus all the weights and biases of layers = millions of inputs!

and very few outputs (many DNNs have $n = 1$) **Why?**

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

AutoDiff - **Forward Mode**

Forward mode needs m forward passes to get a full Jacobian (all gradients of output with respect to each input), where m is the number of inputs

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^m \rightarrow \mathbb{R}^n$$

Problem: DNN typically has large number of inputs:

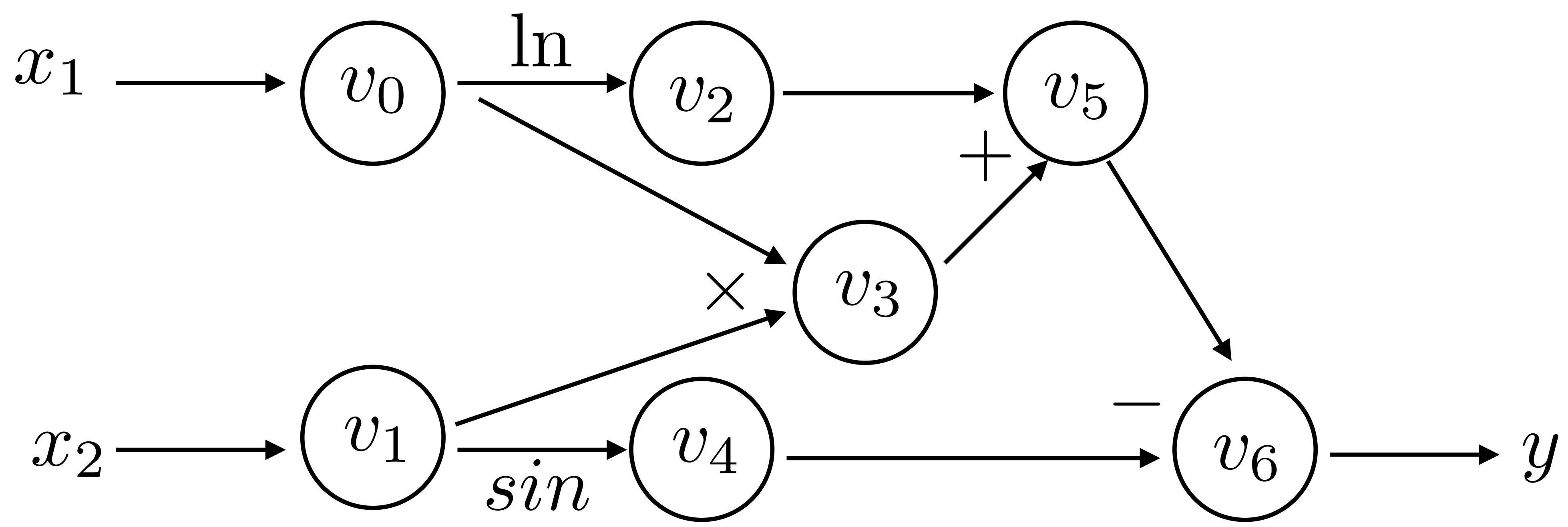
image as an input, plus all the weights and biases of layers = millions of inputs!

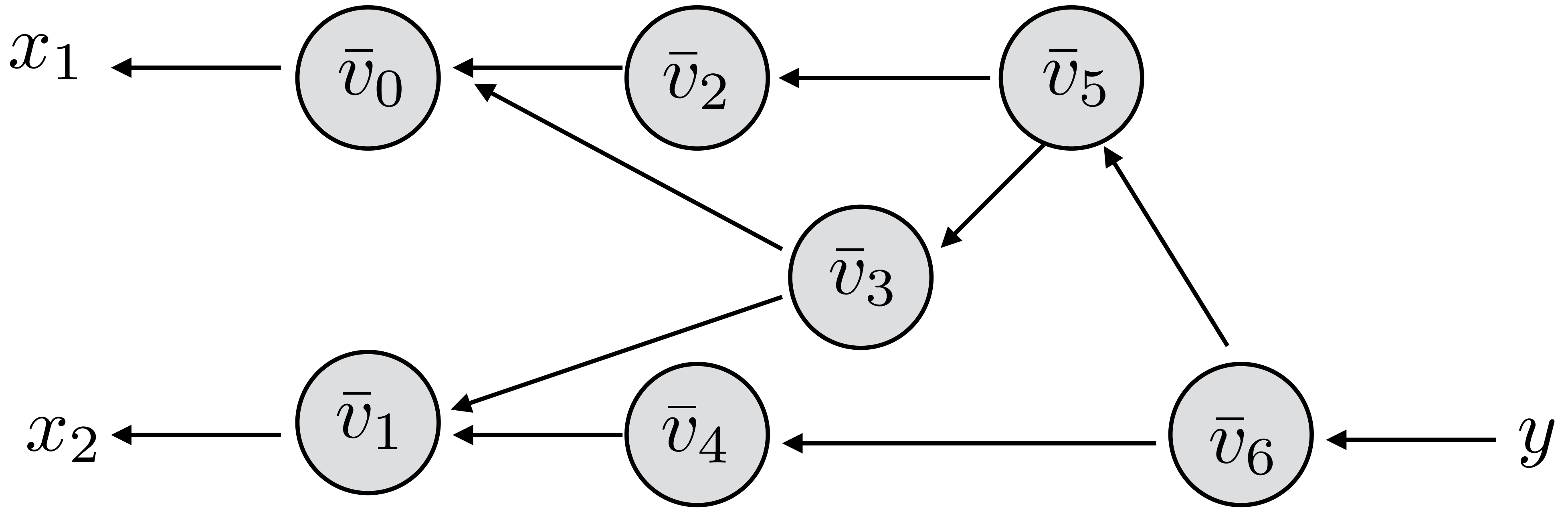
and very few outputs (many DNNs have $n = 1$)

Automatic differentiation in **reverse mode** computes all gradients in n backwards passes (so for most DNNs in a single back pass — **back propagation**)

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

AutoDiff - **Reverse Mode**





Forward Evaluation Trace:

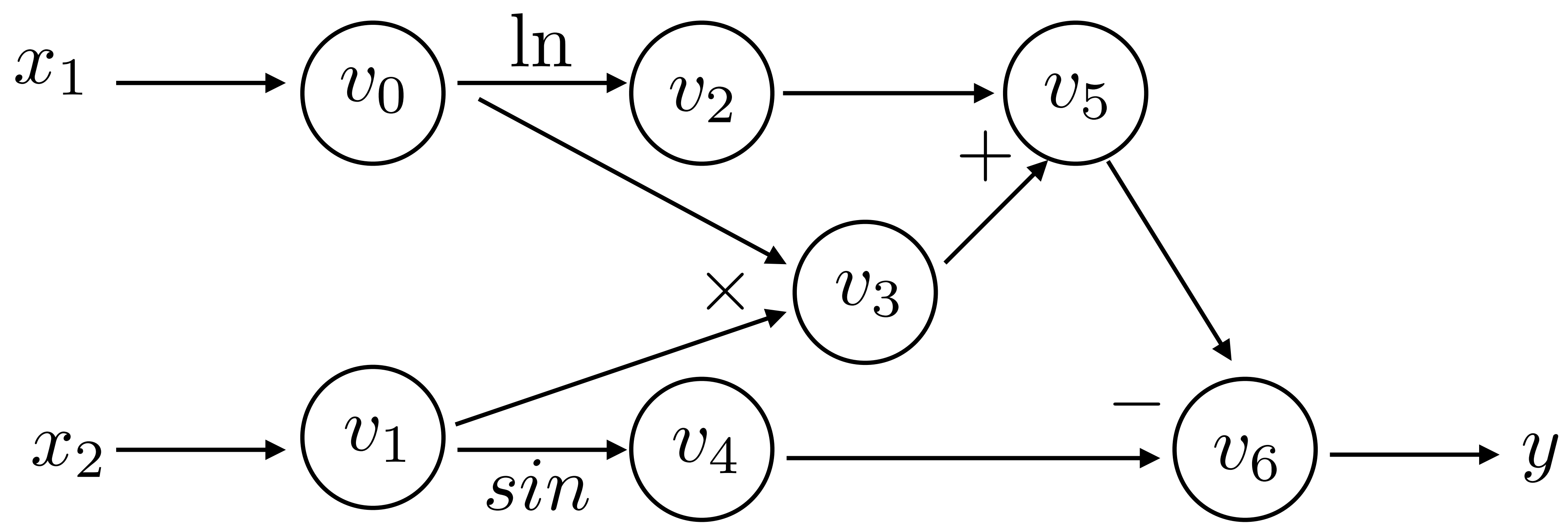
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

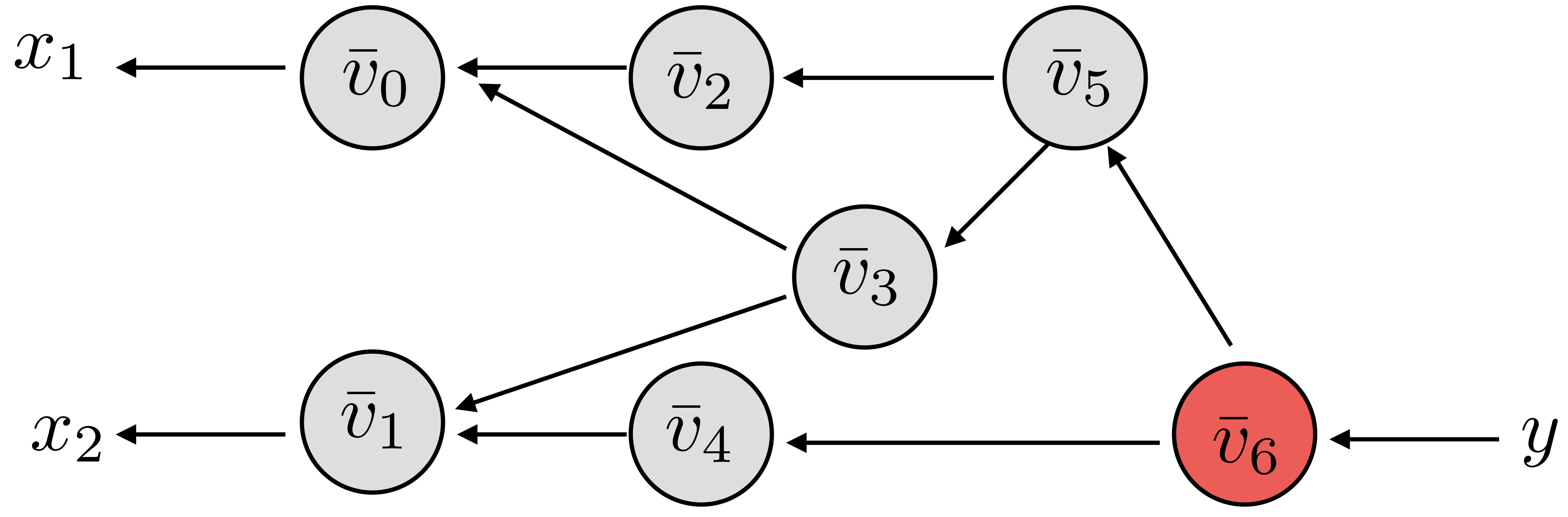
Traverse the original graph in the *reverse* topological order and for each node in the original graph introduce an **adjoint node**, which computes derivative of the output with respect to the local node (using Chain rule):

$$\bar{v}_i = \frac{\partial y_j}{\partial v_i} = \sum_{k \in \mathrm{pa}(i)} \frac{\partial v_k}{\partial v_i} \frac{\partial y_j}{\partial v_k} = \sum_{k \in \mathrm{pa}(i)} \frac{\partial v_k}{\partial v_i} \bar{v}_k$$

"**local**" derivative

AutoDiff - **Reverse Mode**





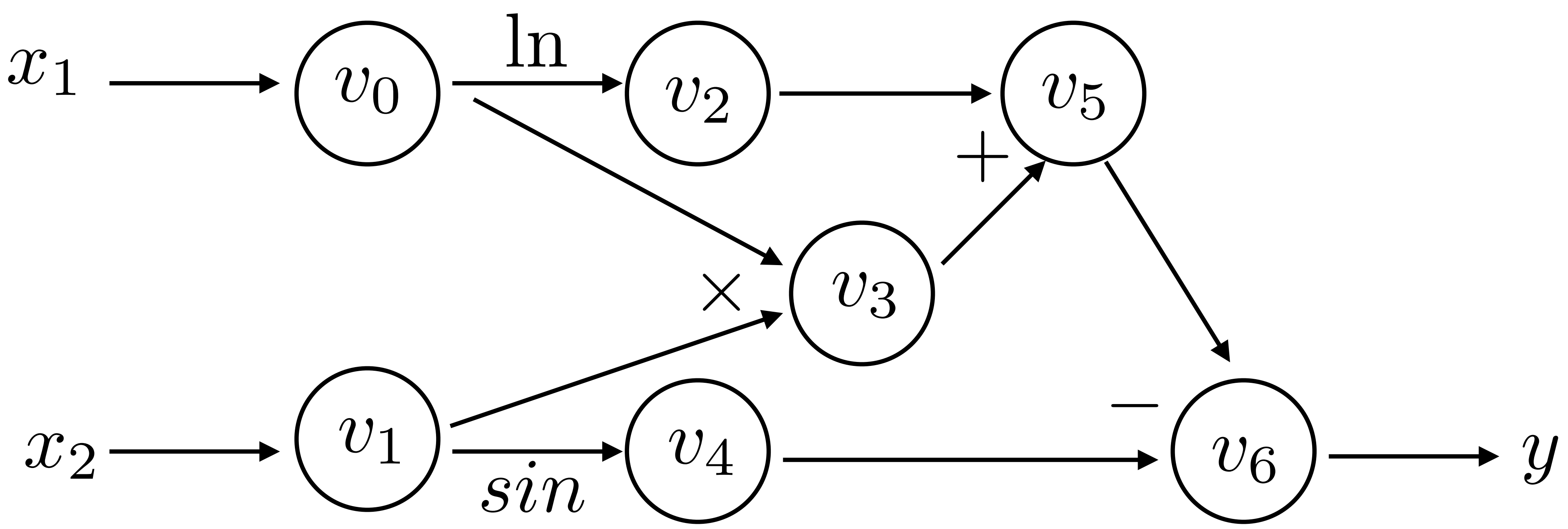
Forward Evaluation Trace:

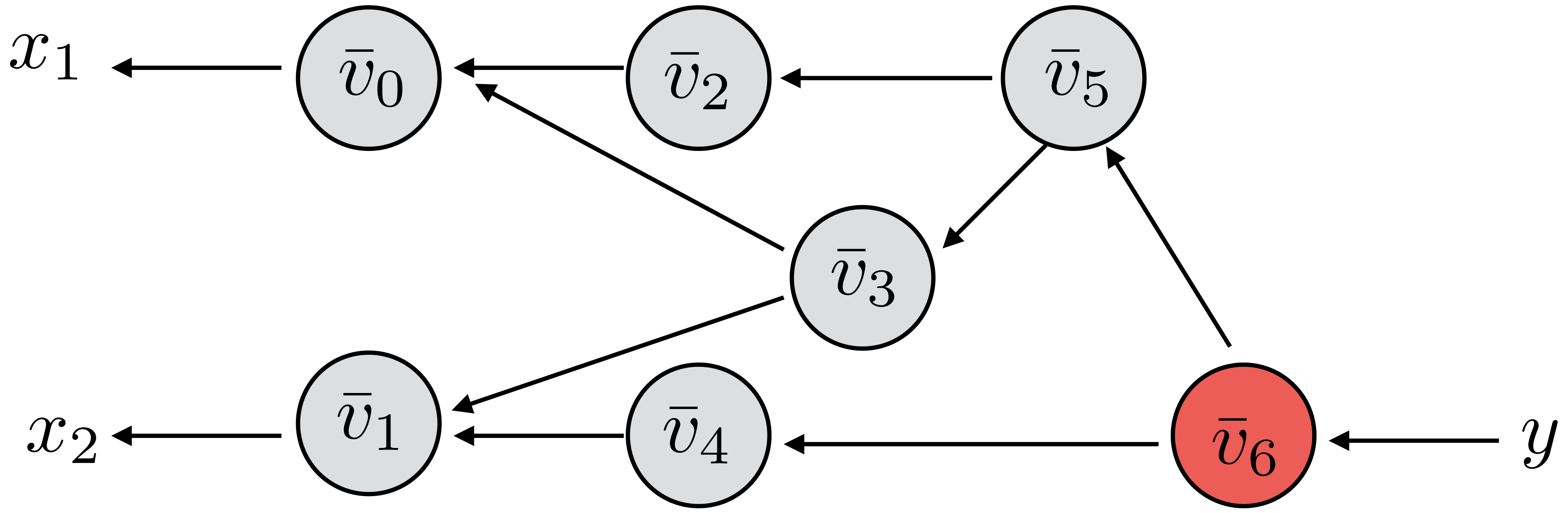
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$$\bar{v}_6 = \frac{\partial y}{\partial v_6}$$

AutoDiff - **Reverse Mode**





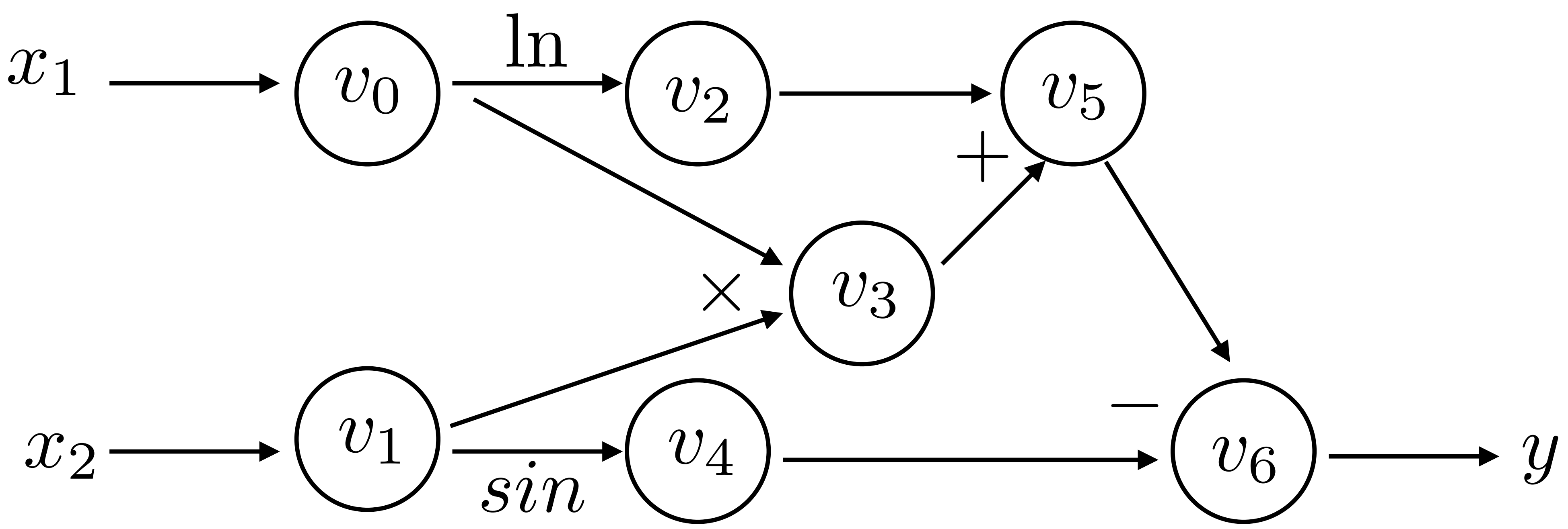
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

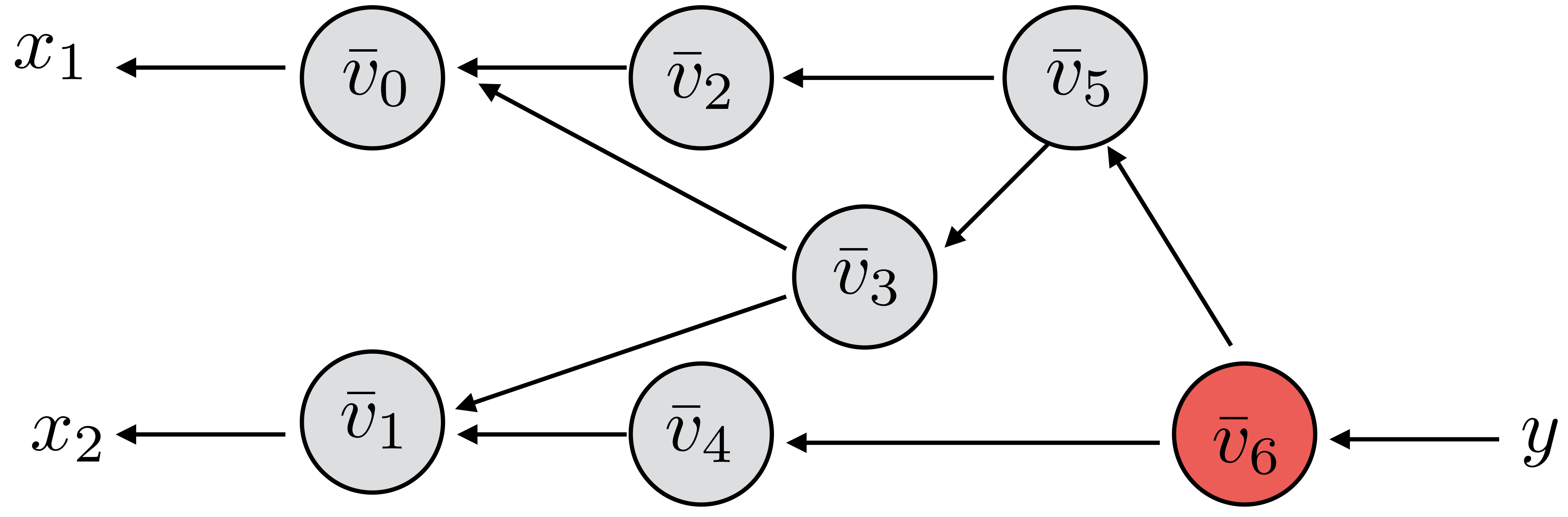
$$\bar{v}_6 = \frac{\partial y}{\partial v_6}$$

AutoDiff - **Reverse Mode**



Forward Evaluation Trace:

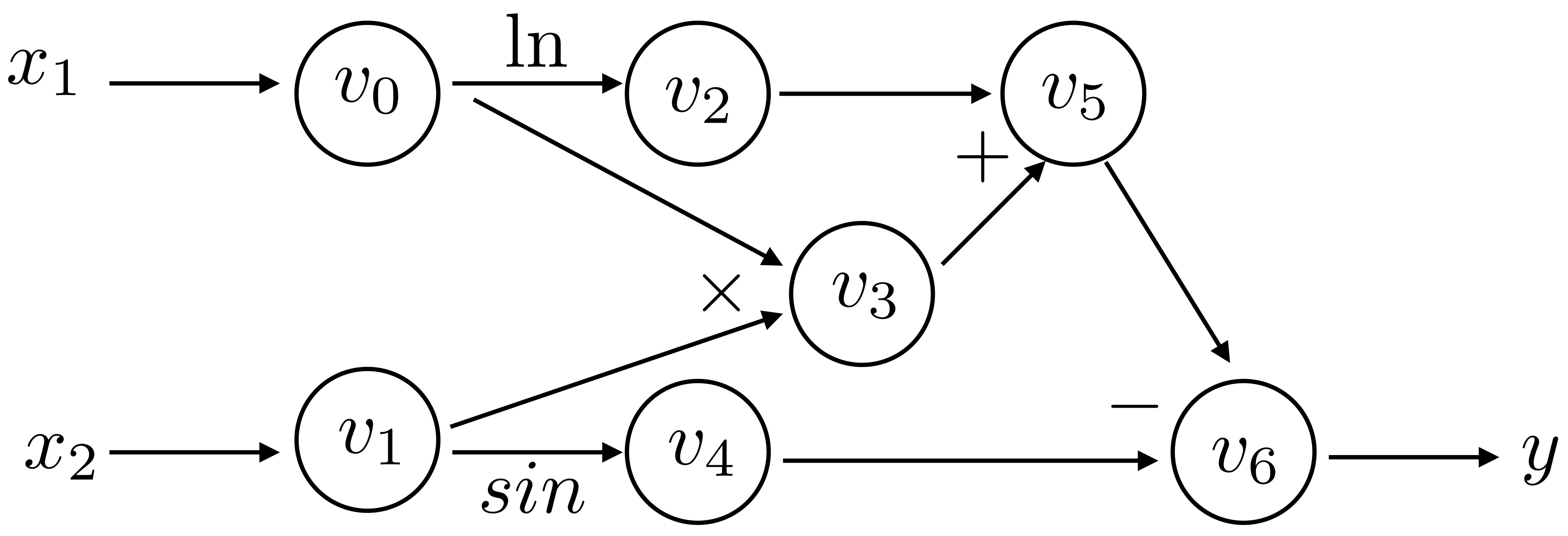
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

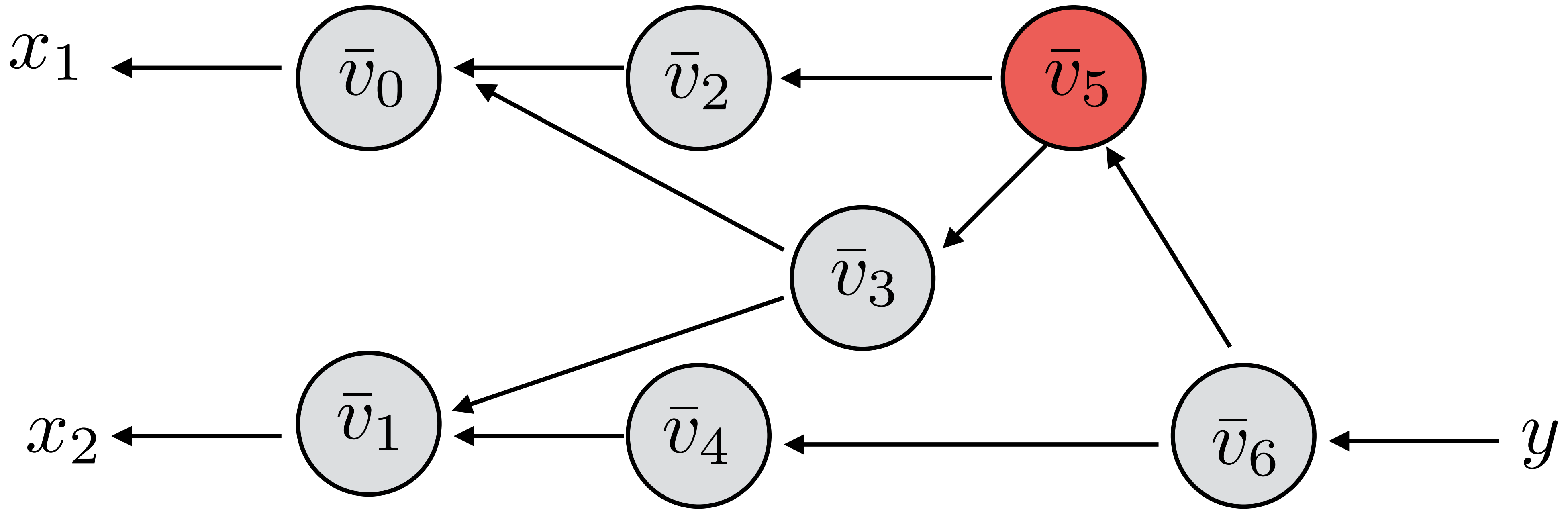


Backwards Derivative Trace:

$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





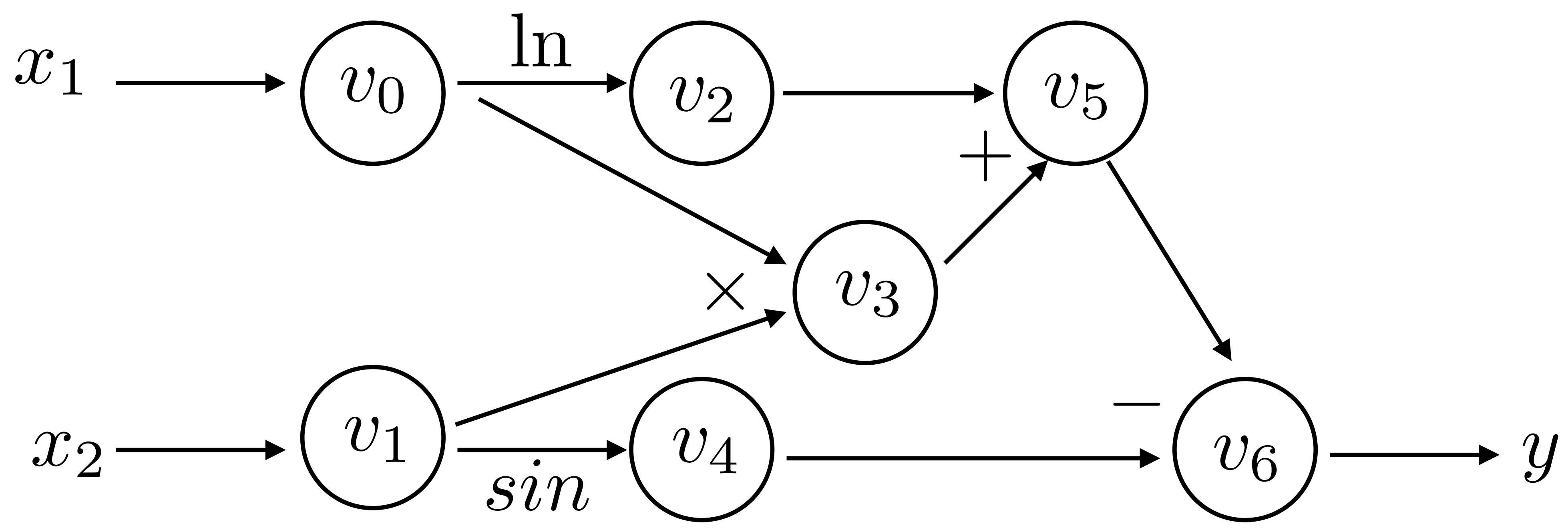
Forward Evaluation Trace:

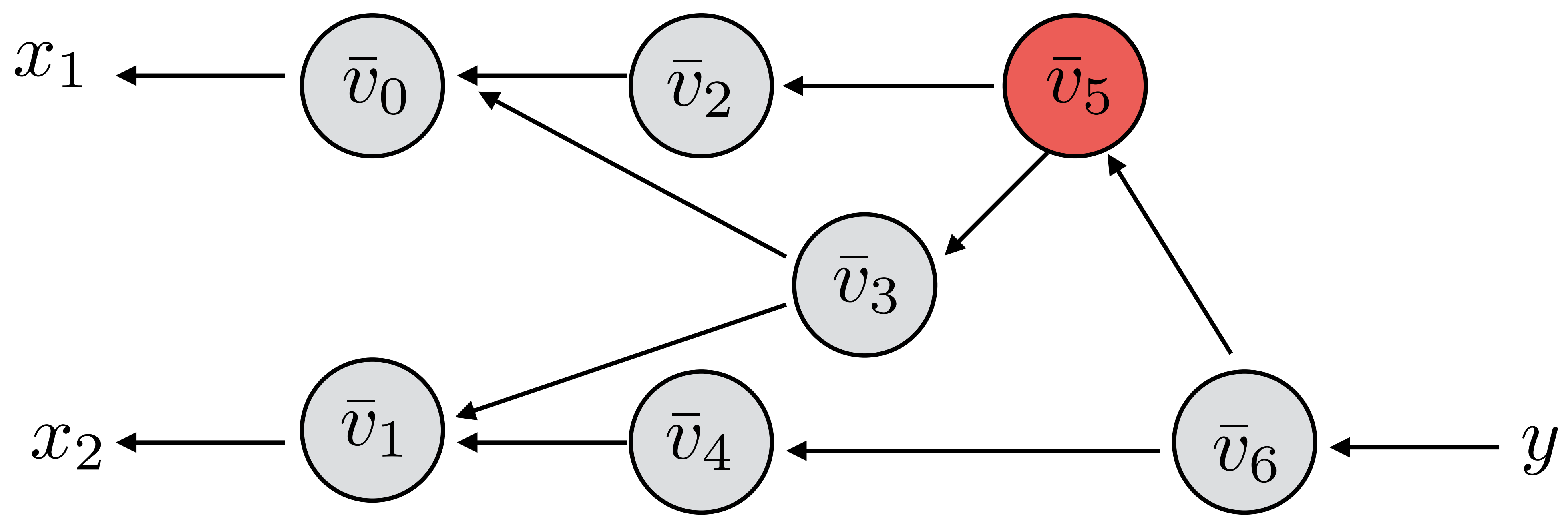
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5}$	
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





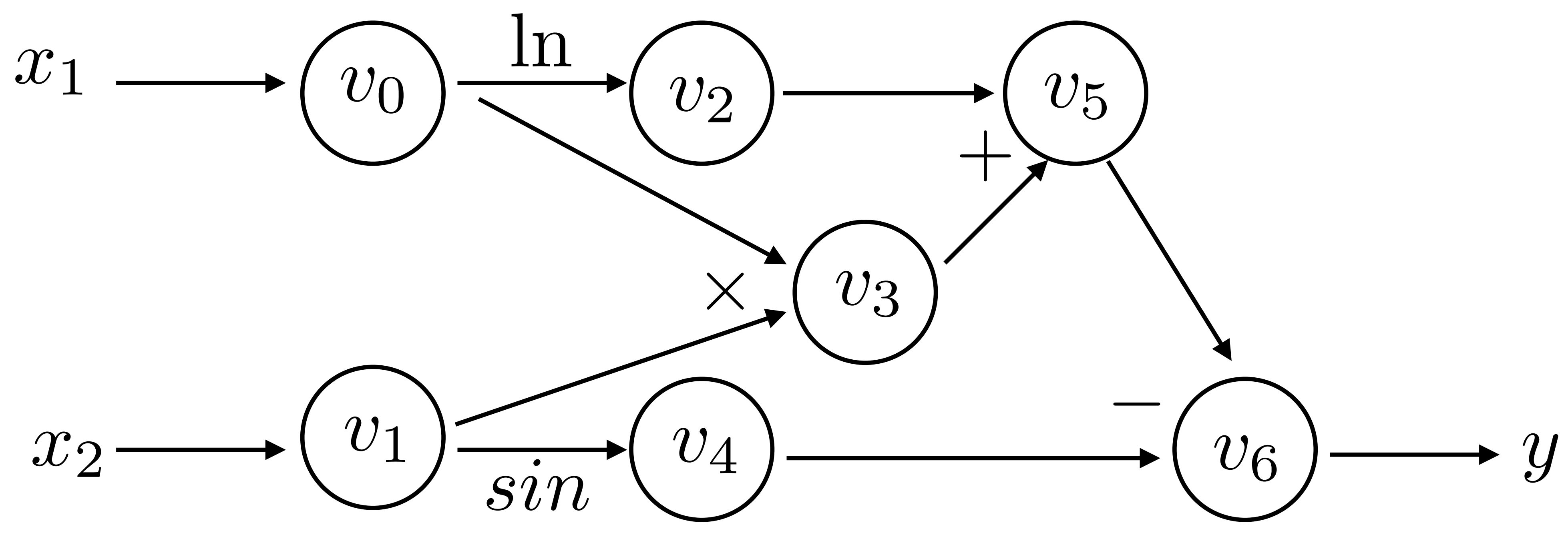
Forward Evaluation Trace:

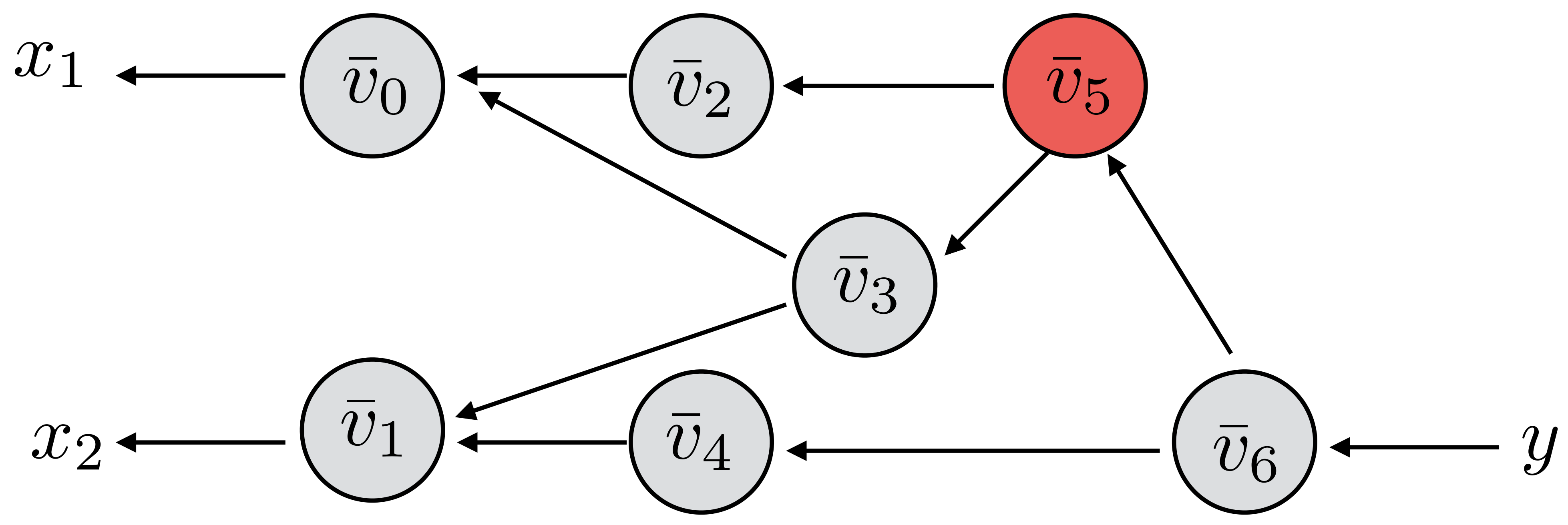
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5}$	
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





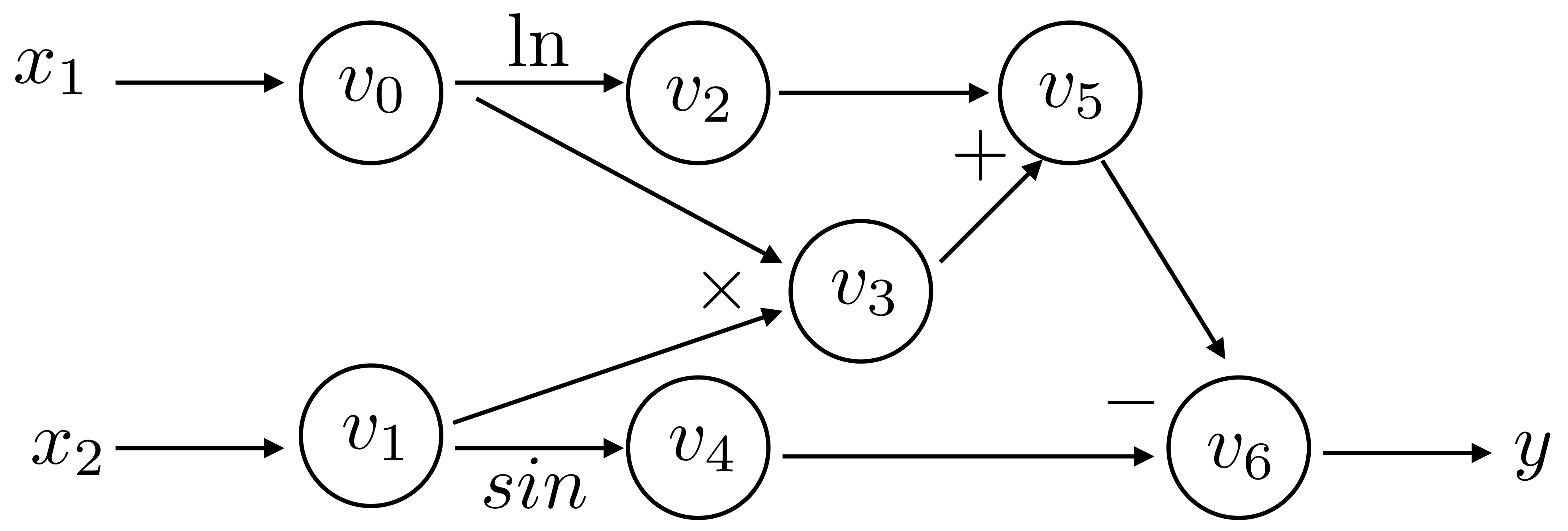
Forward Evaluation Trace:

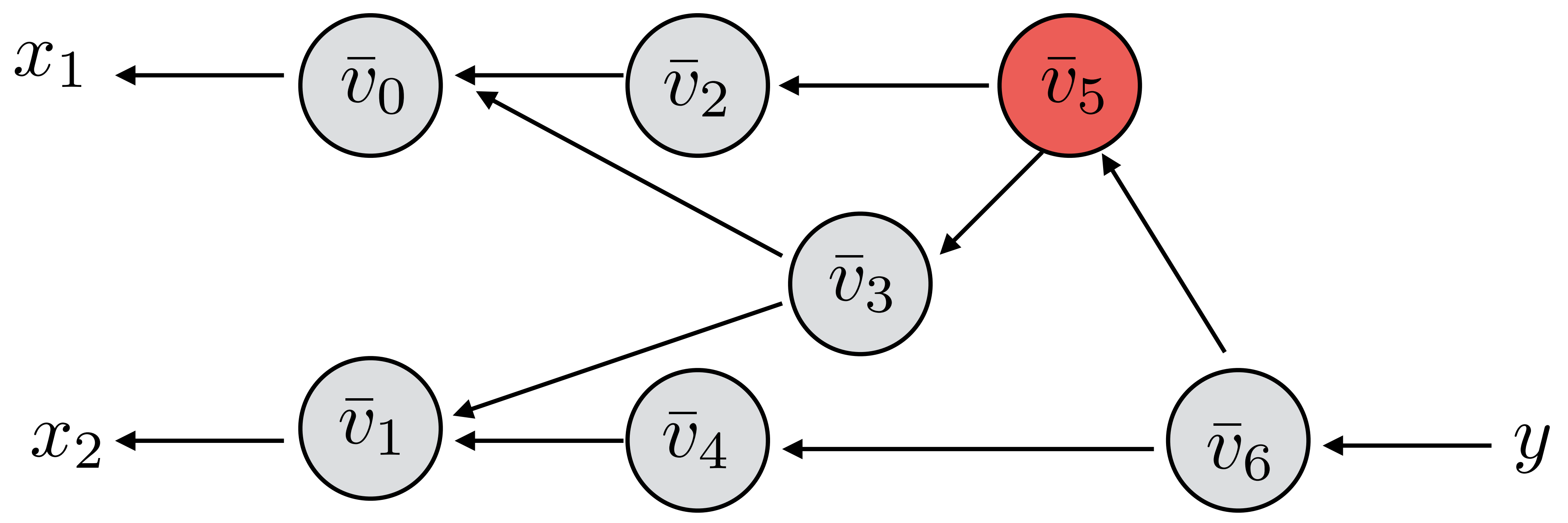
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





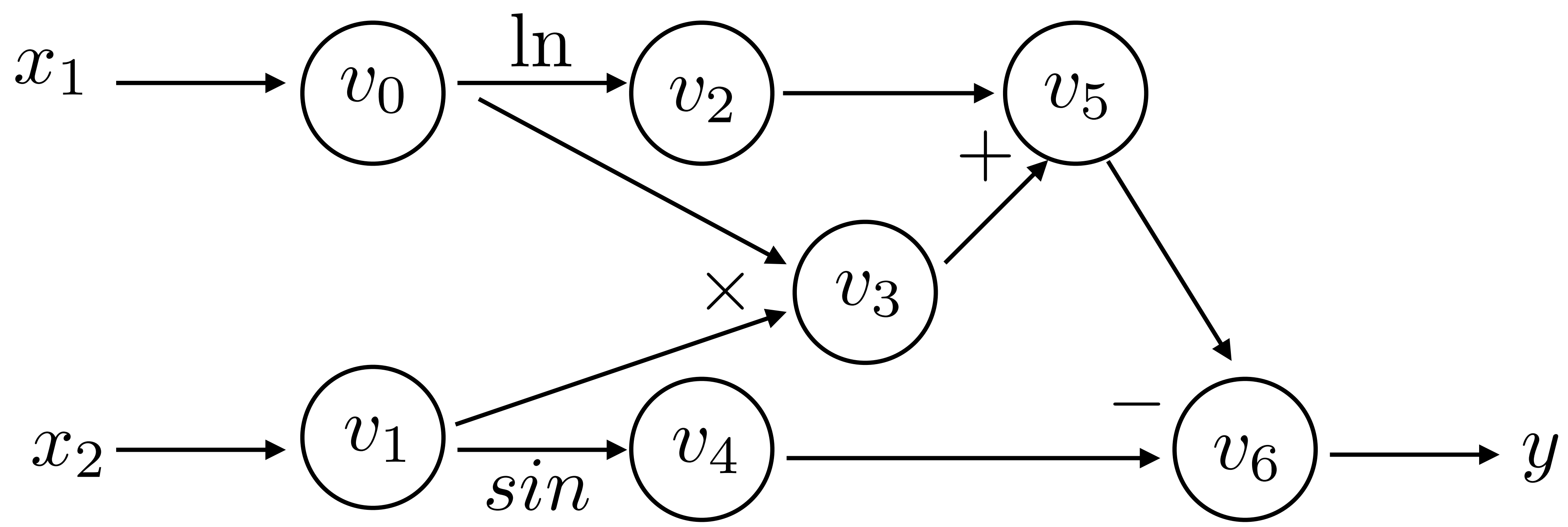
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

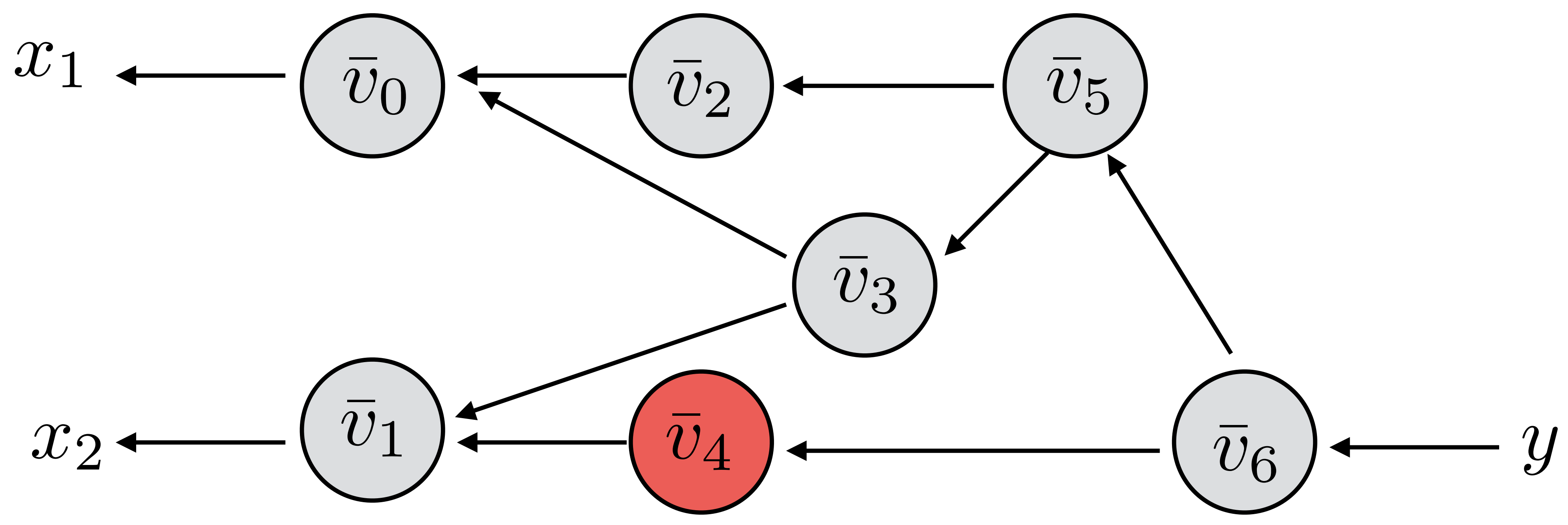
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**



Forward Evaluation Trace:

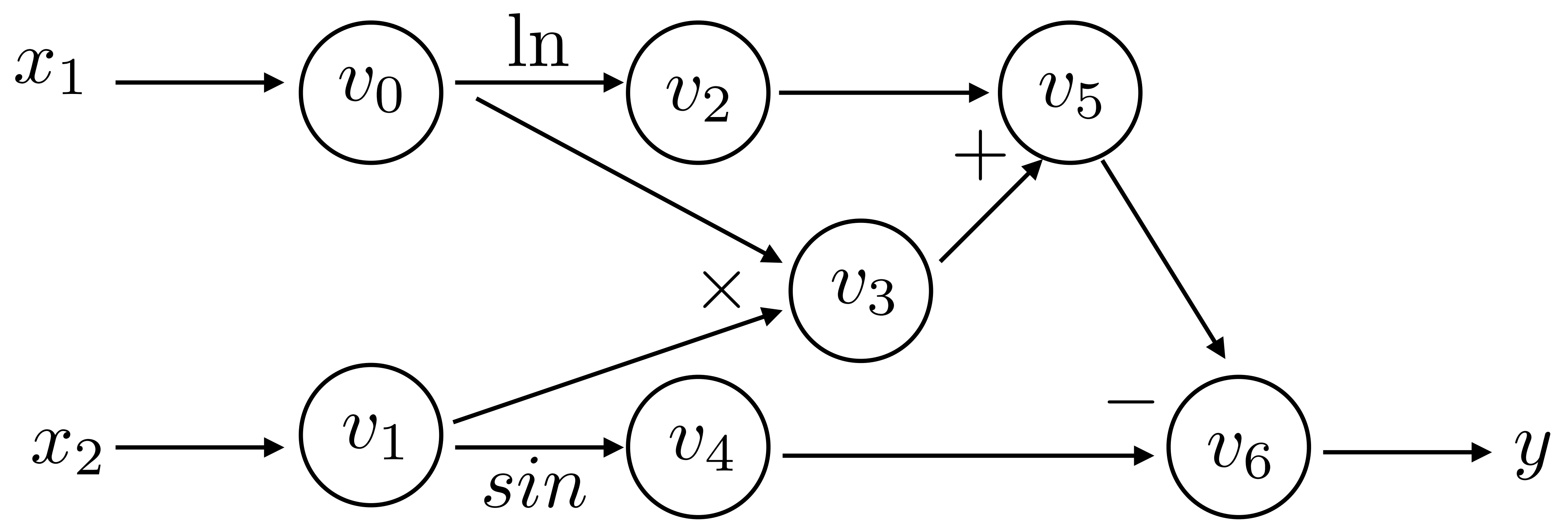
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652



Backwards Derivative Trace:

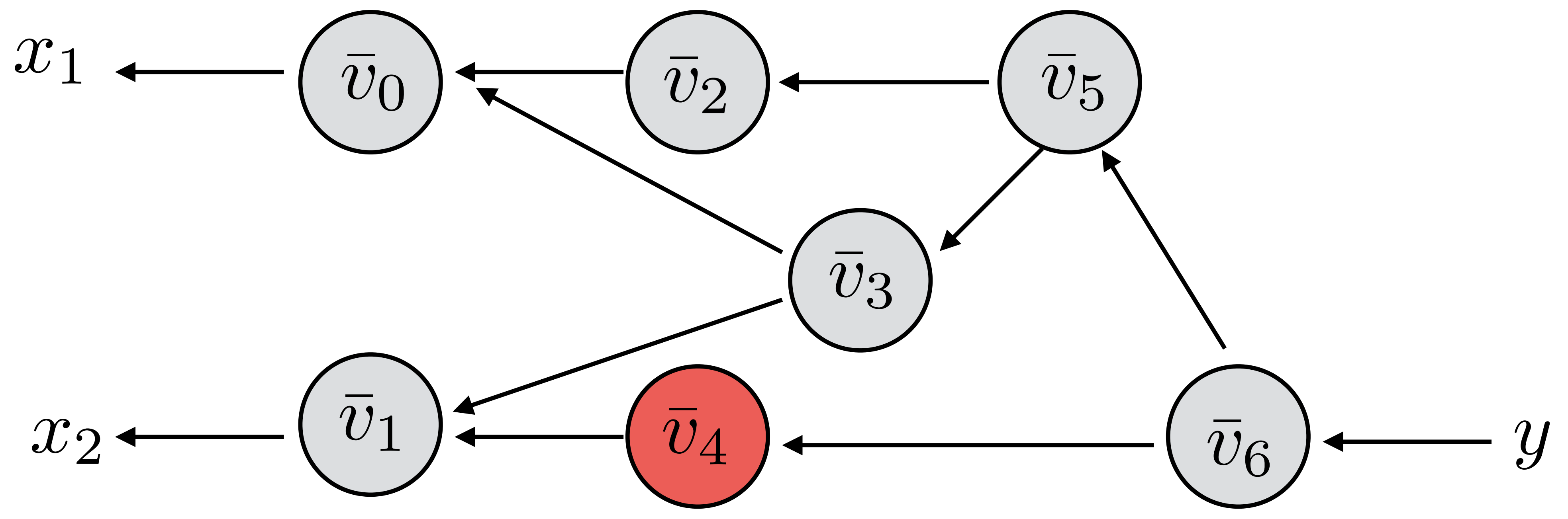
$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4}$	
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**



Forward Evaluation Trace:

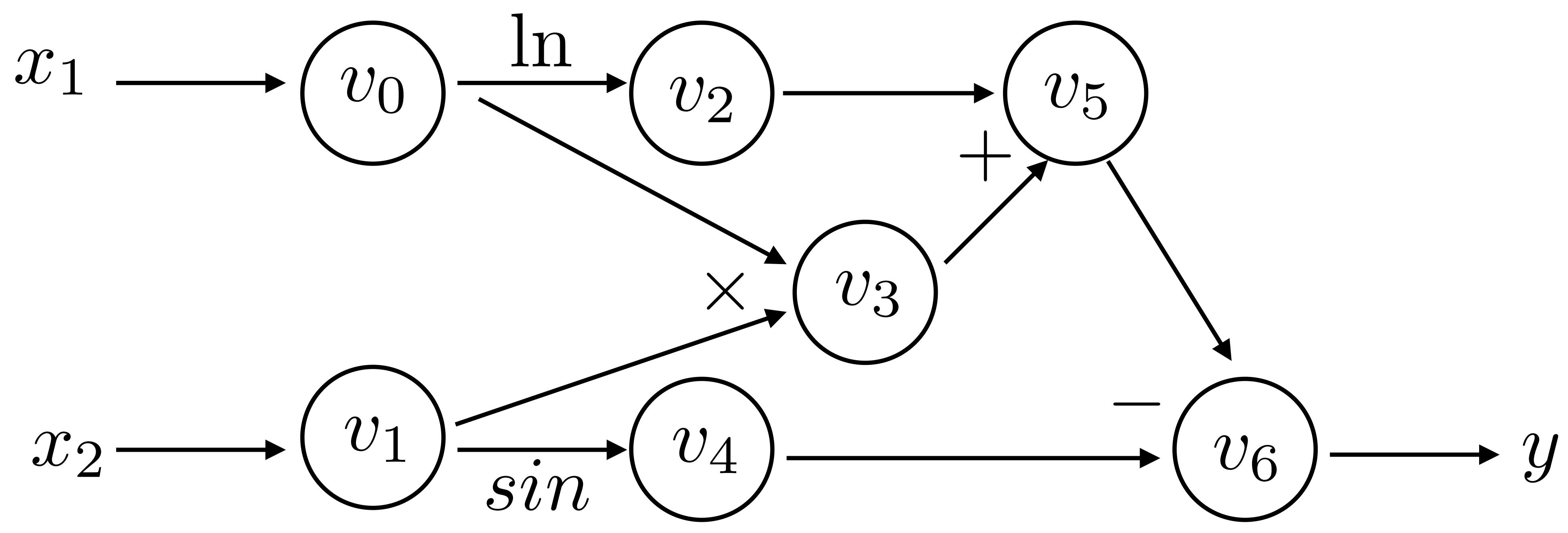
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

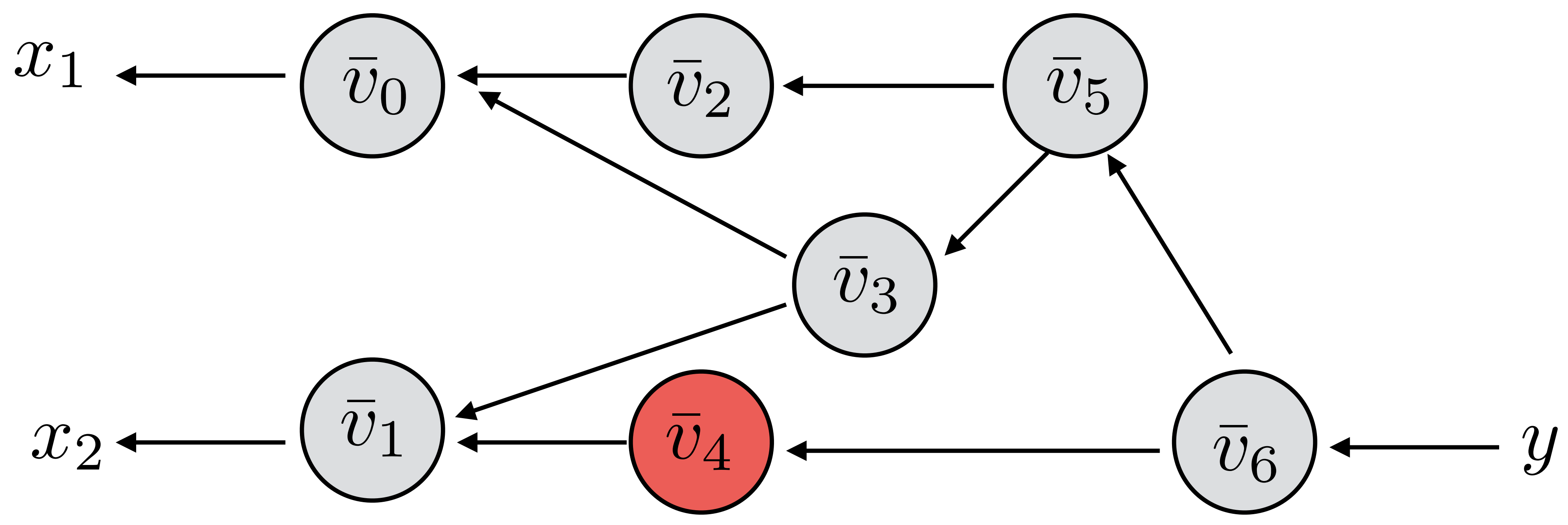


Backwards Derivative Trace:

$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4}$	
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





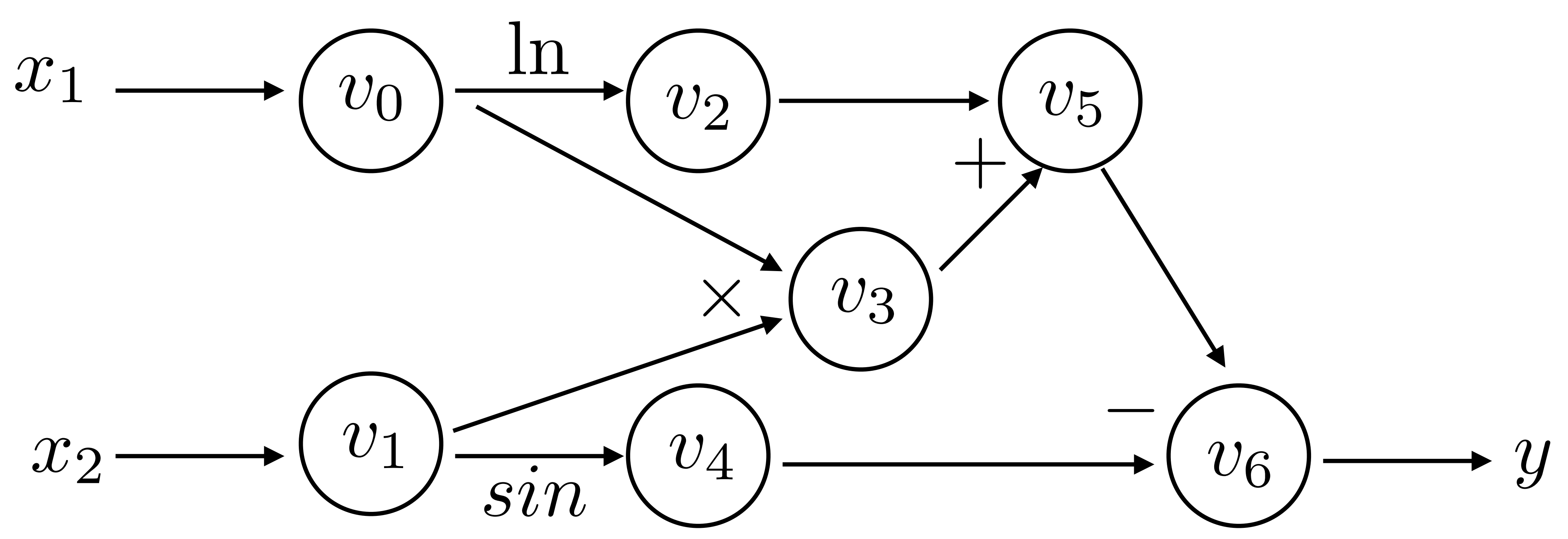
Forward Evaluation Trace:

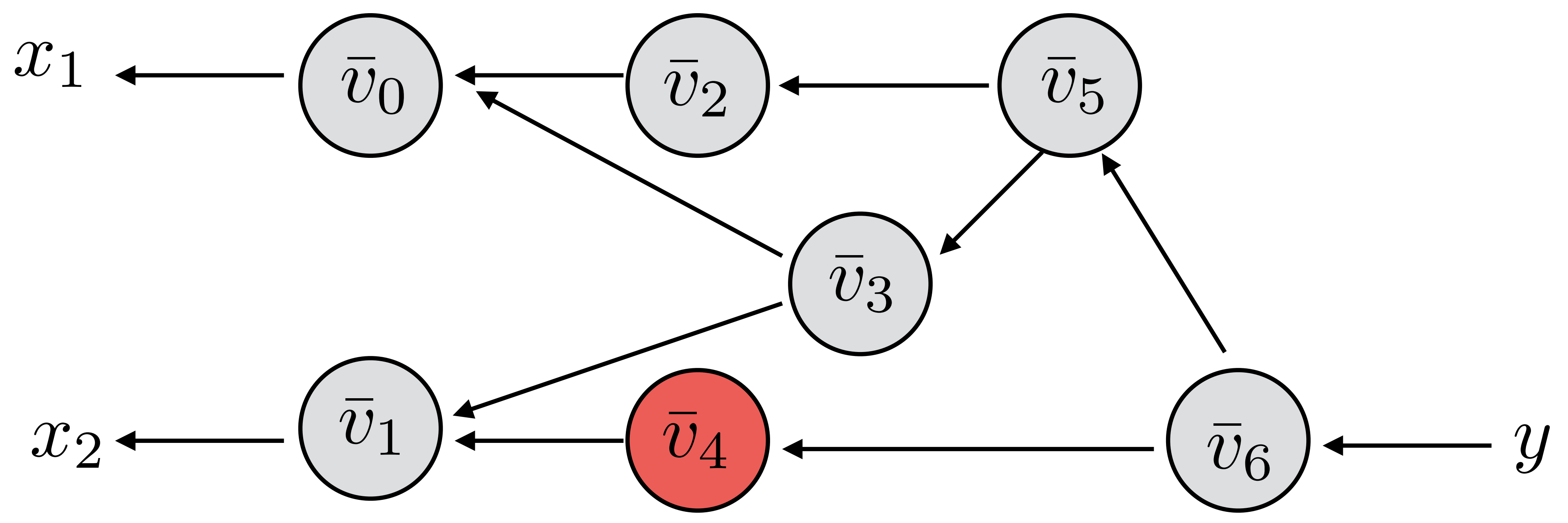
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





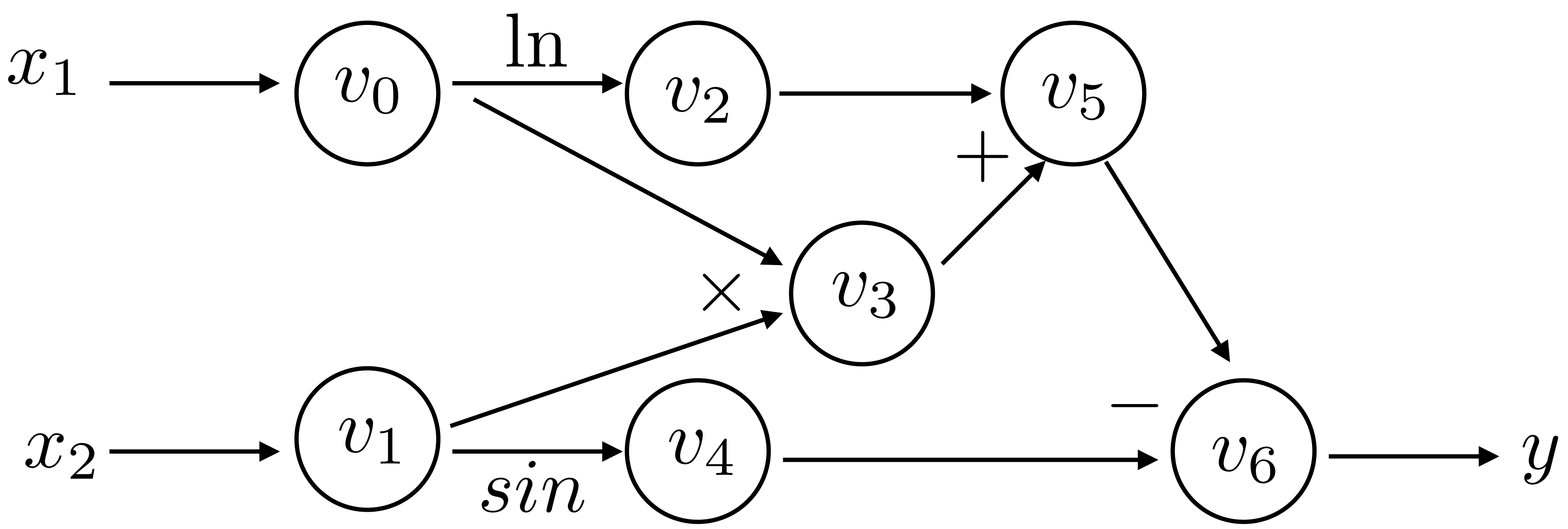
Forward Evaluation Trace:

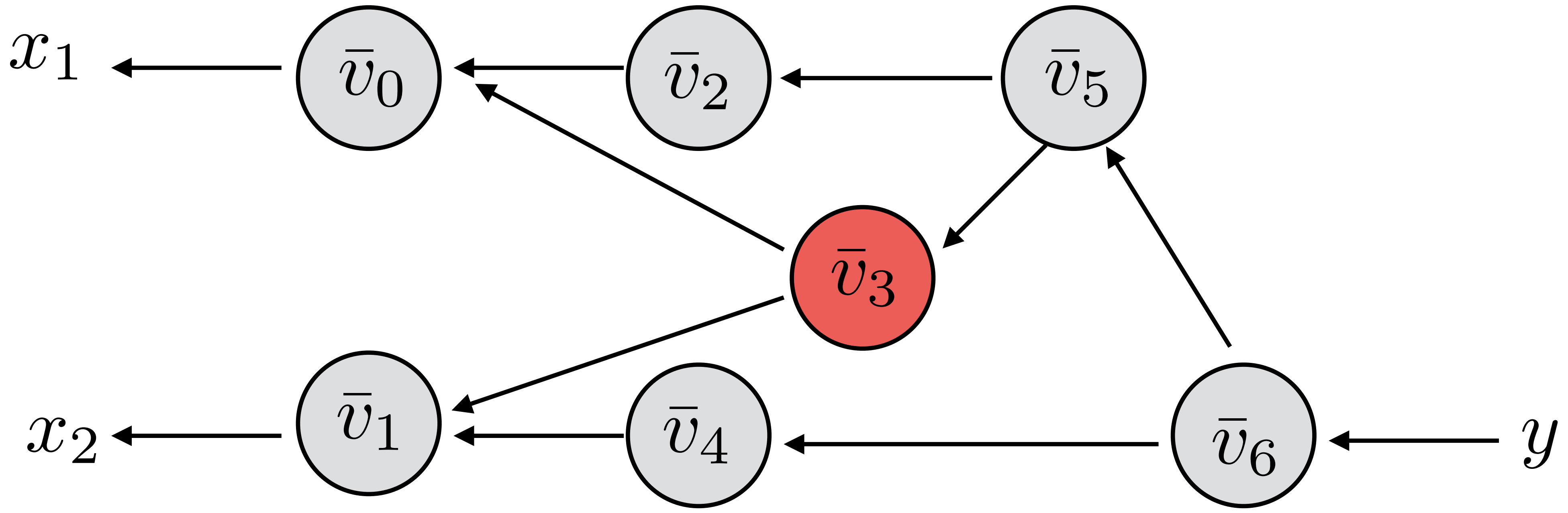
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





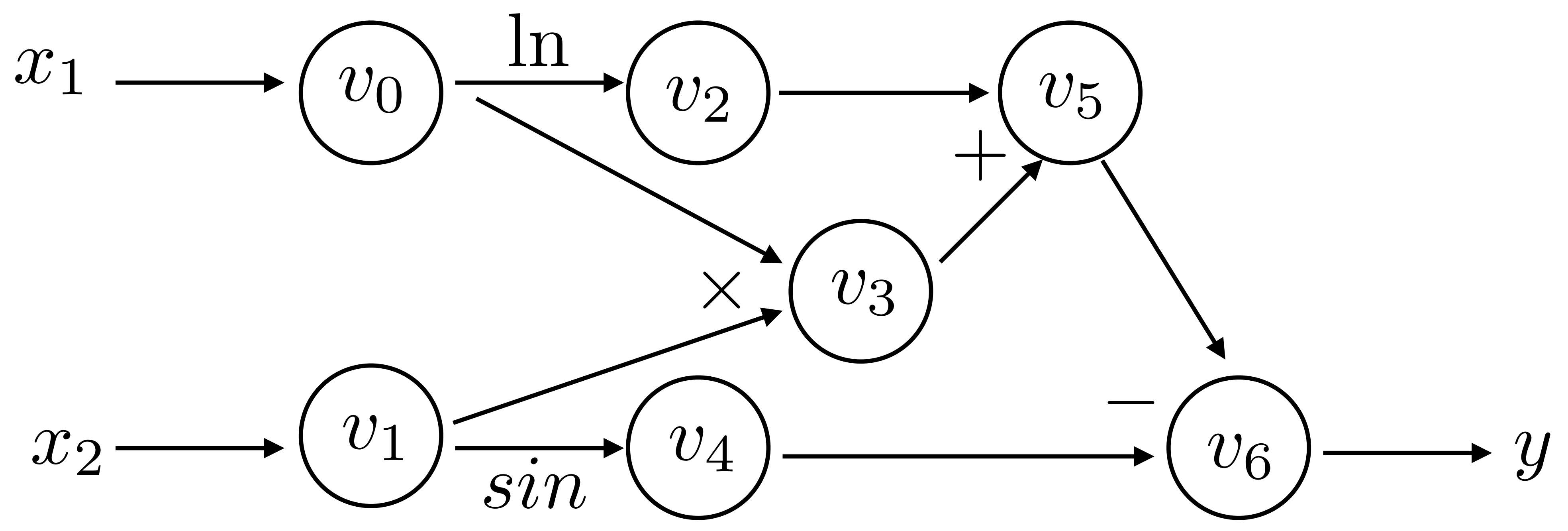
Forward Evaluation Trace:

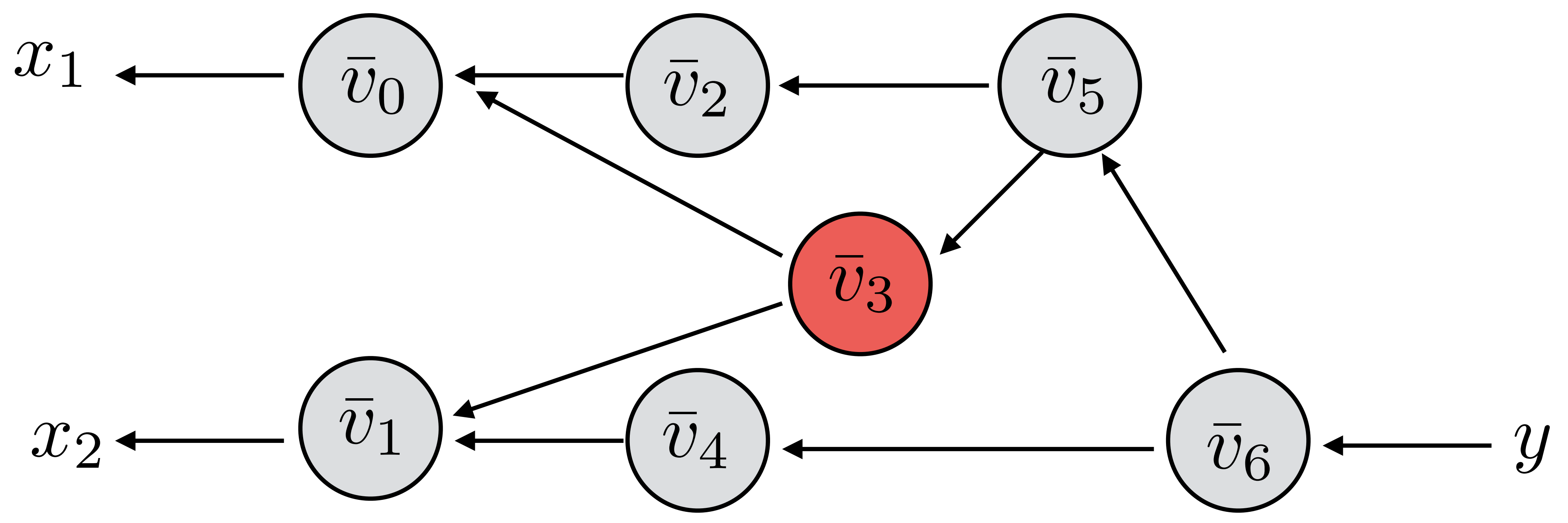
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





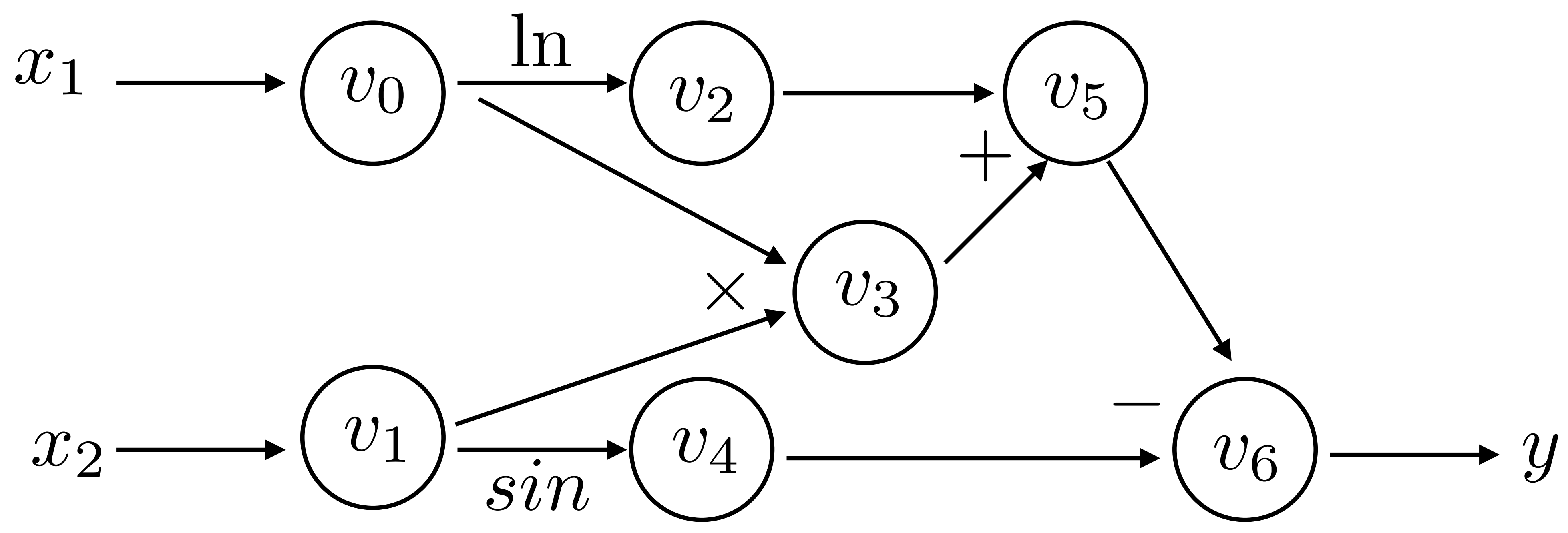
Forward Evaluation Trace:

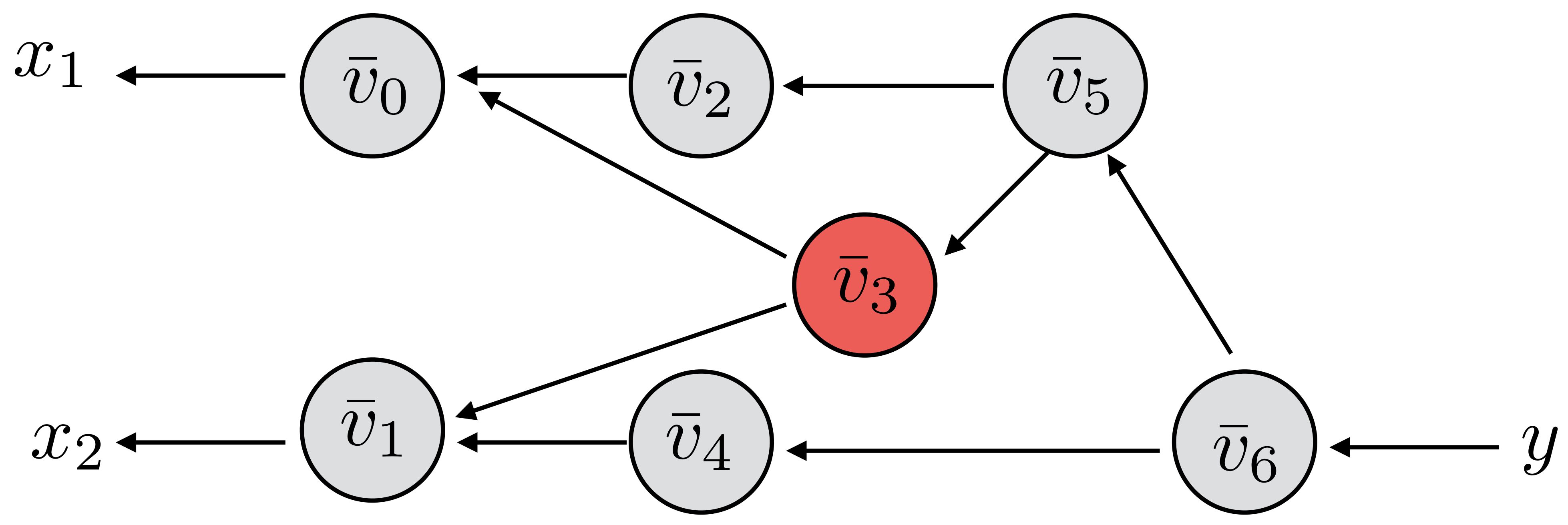
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





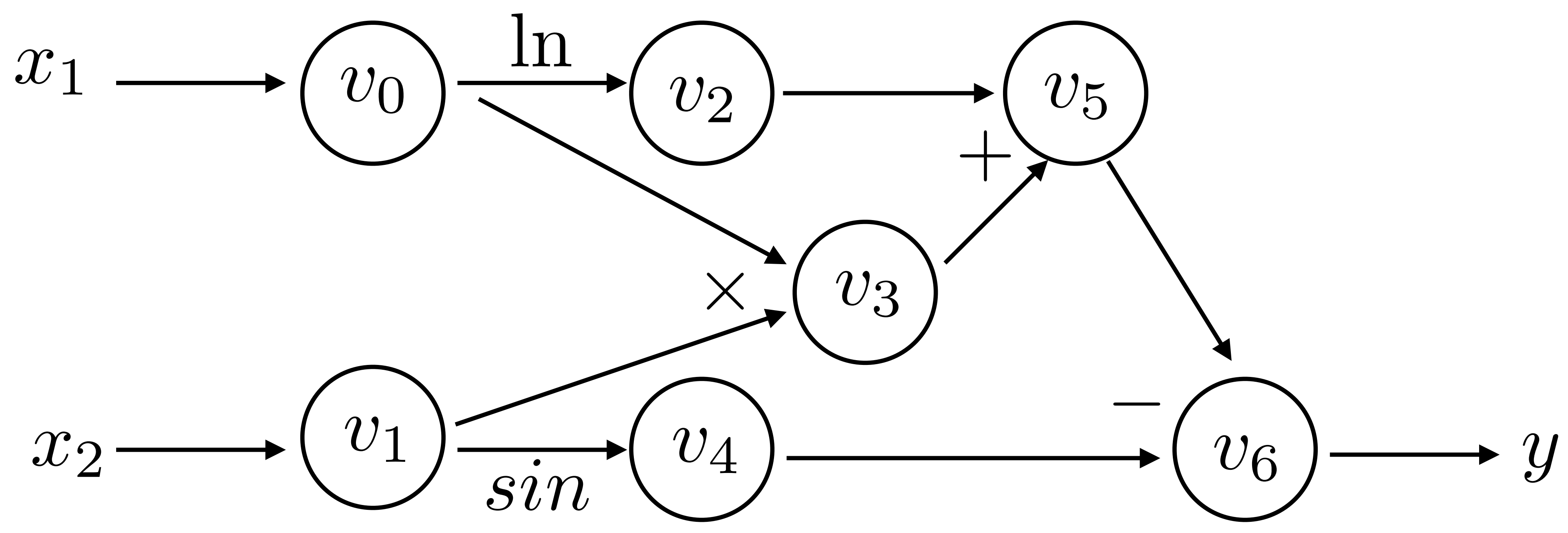
Forward Evaluation Trace:

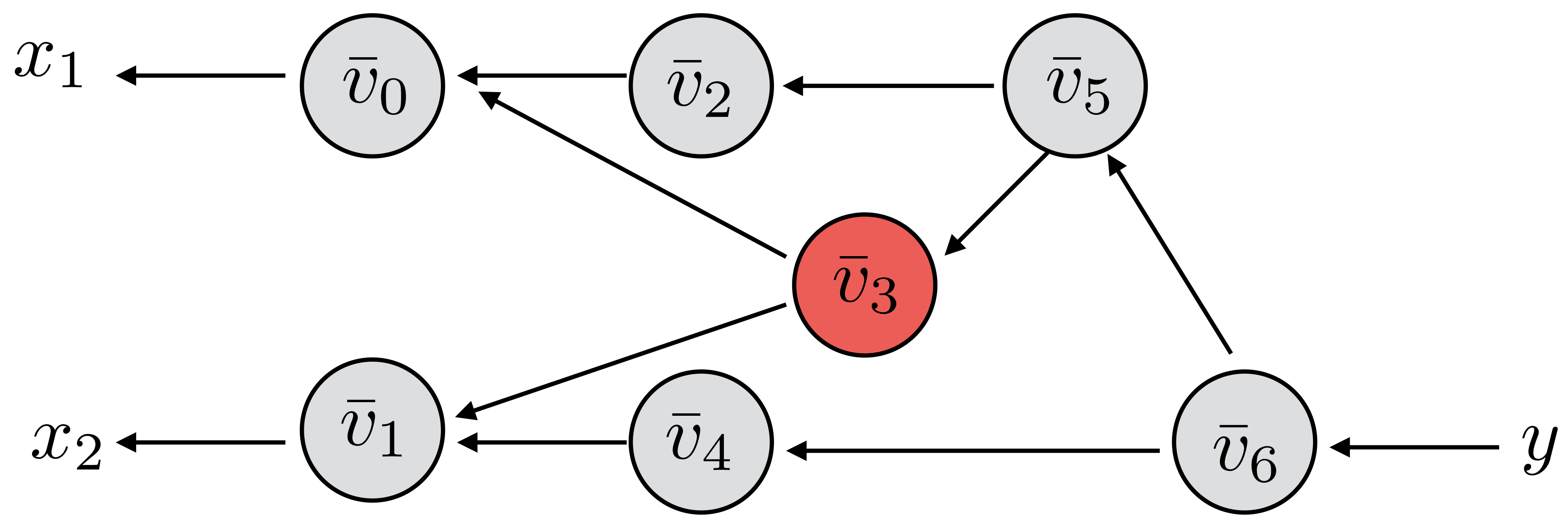
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_3 = \bar{v}_5 \dfrac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	
$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





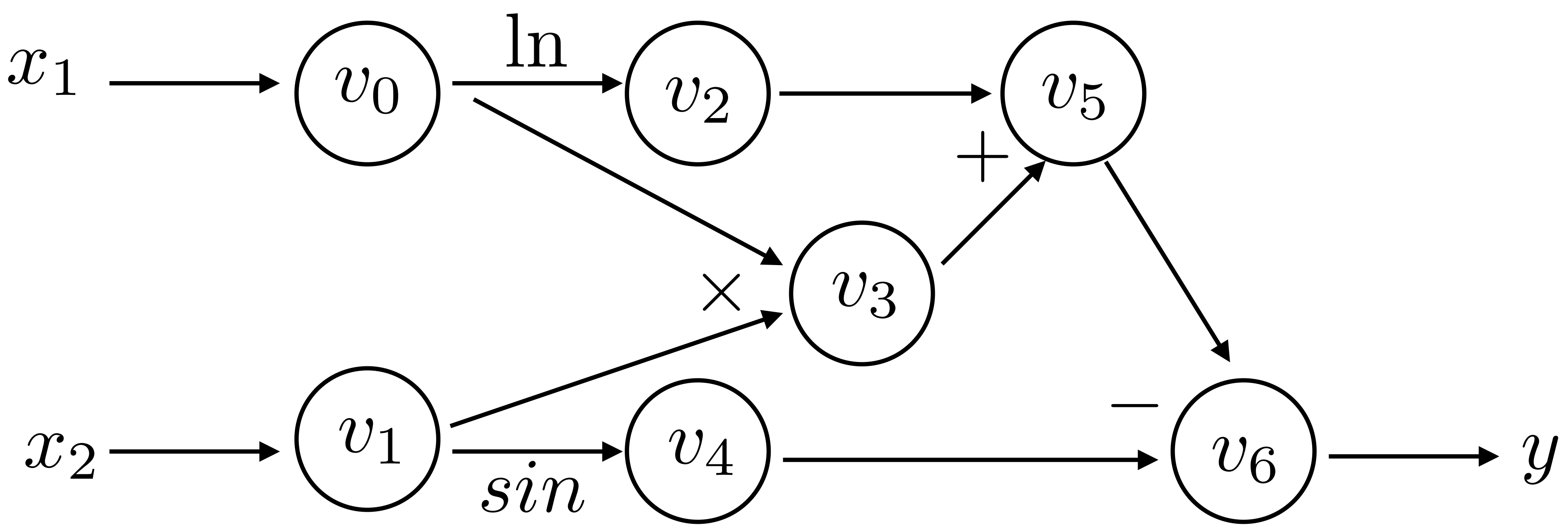
Forward Evaluation Trace:

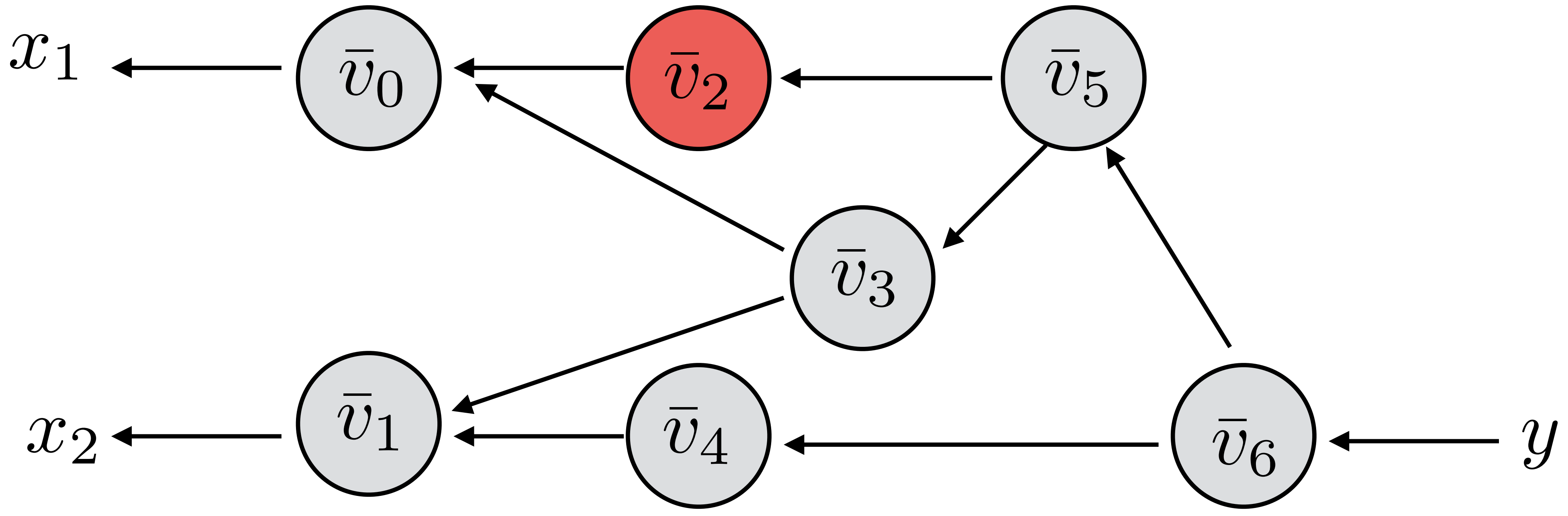
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_3 = \bar{v}_5 \dfrac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





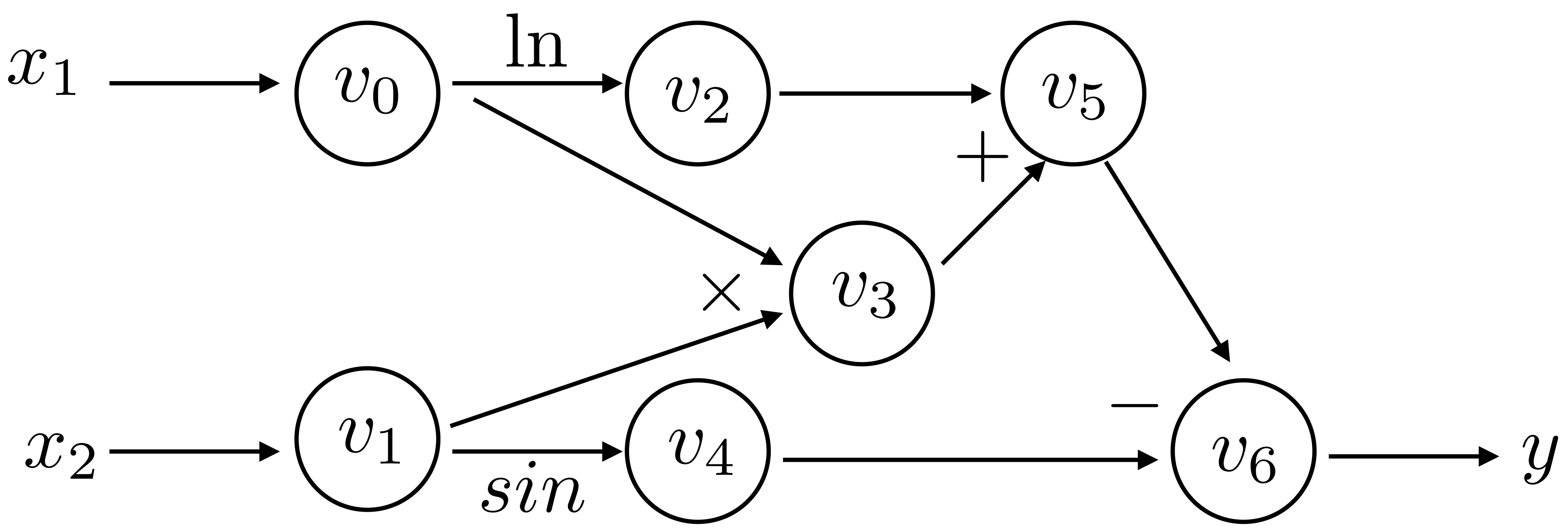
Forward Evaluation Trace:

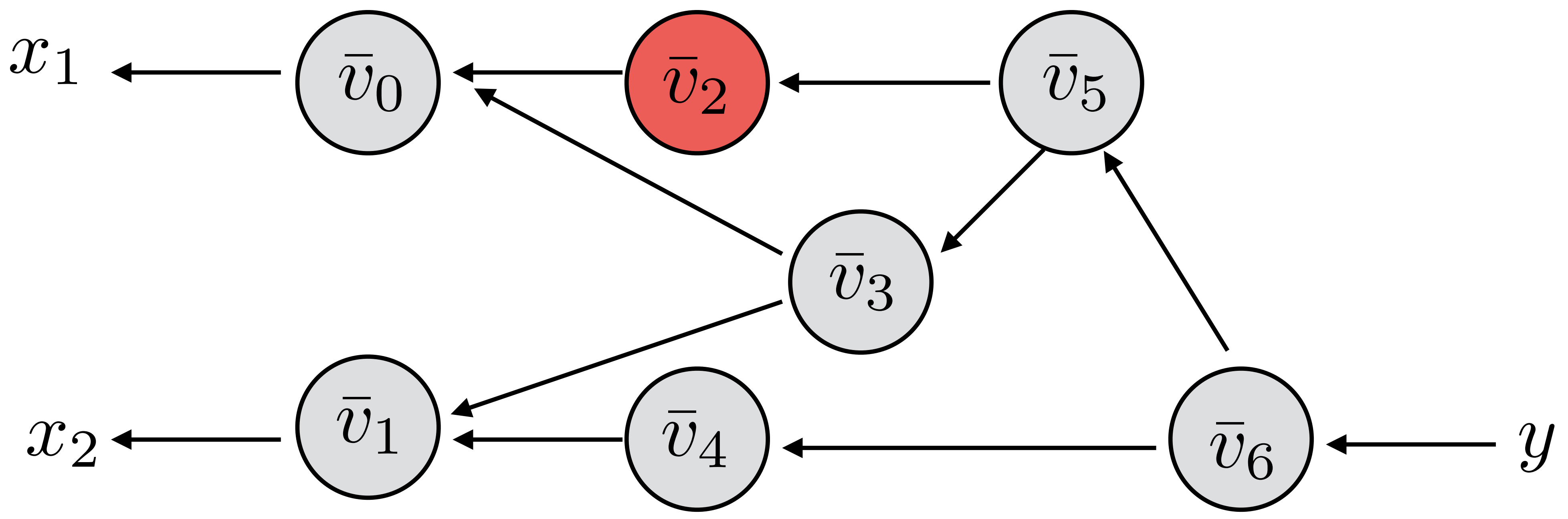
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_2 = \bar{v}_5 \dfrac{\partial v_5}{\partial v_2}$	
$\bar{v}_3 = \bar{v}_5 \dfrac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





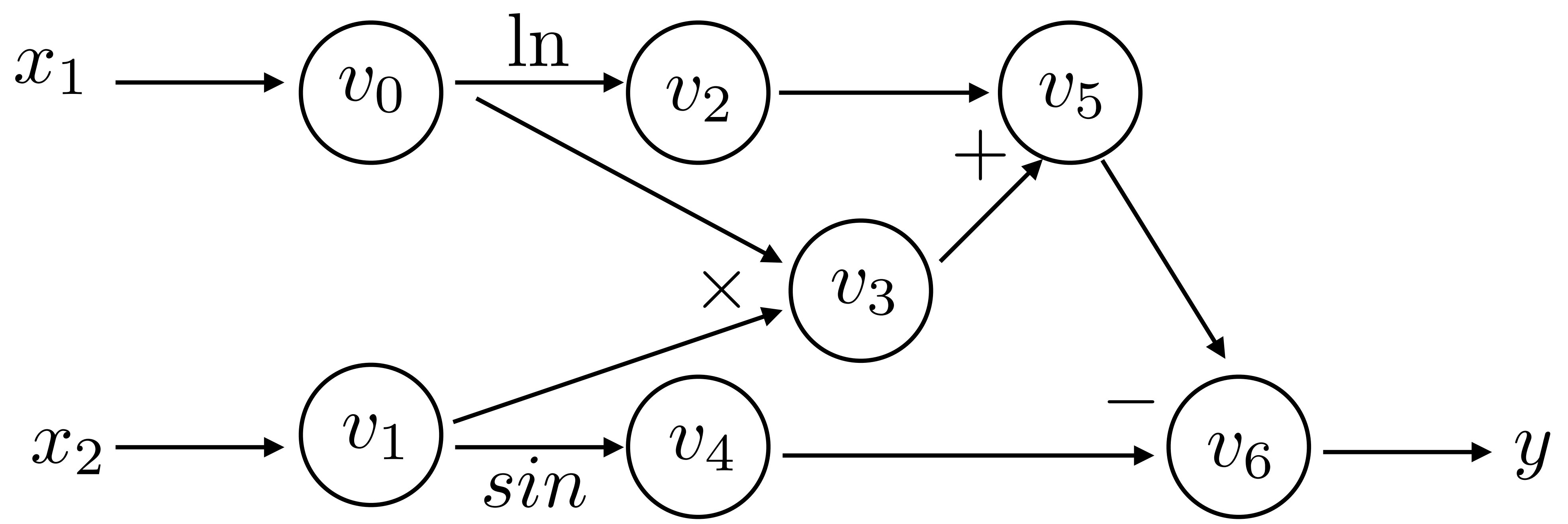
Backwards Derivative Trace:

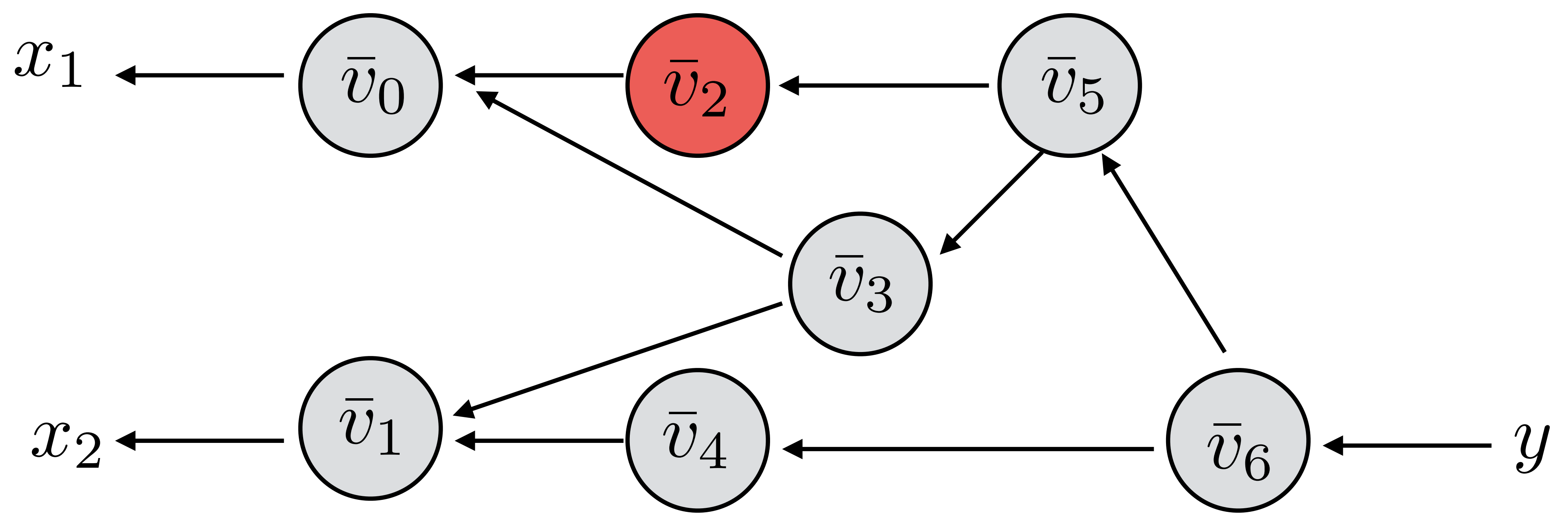
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2}$	
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

AutoDiff - **Reverse Mode**





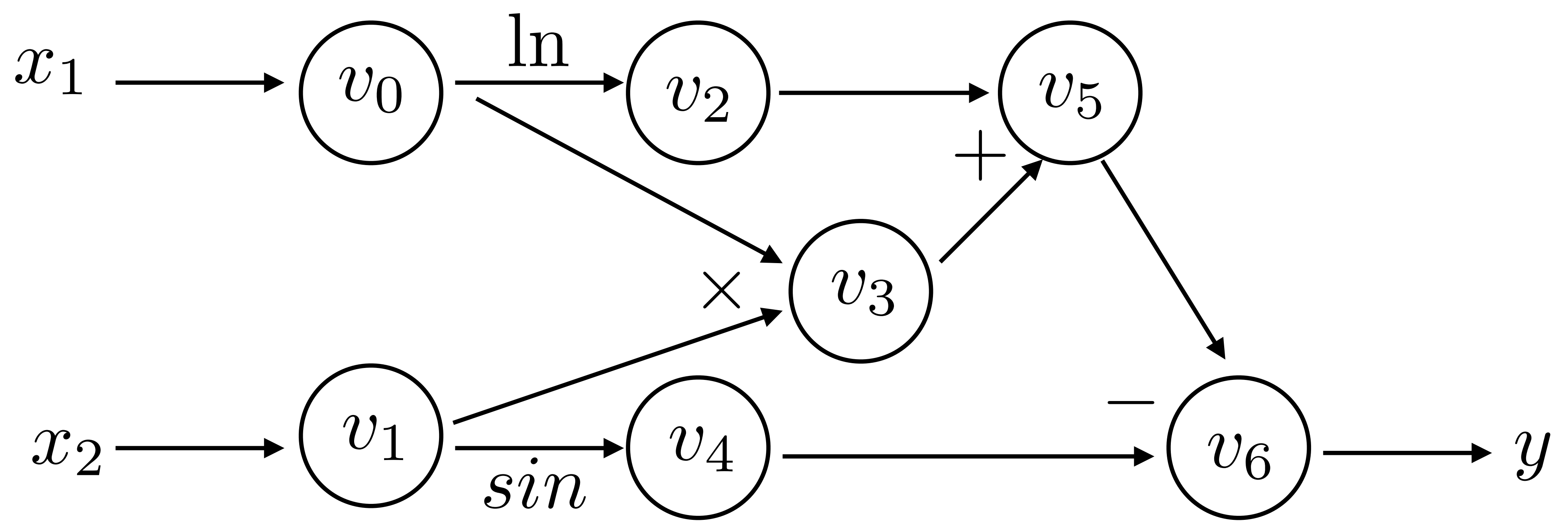
Backwards Derivative Trace:

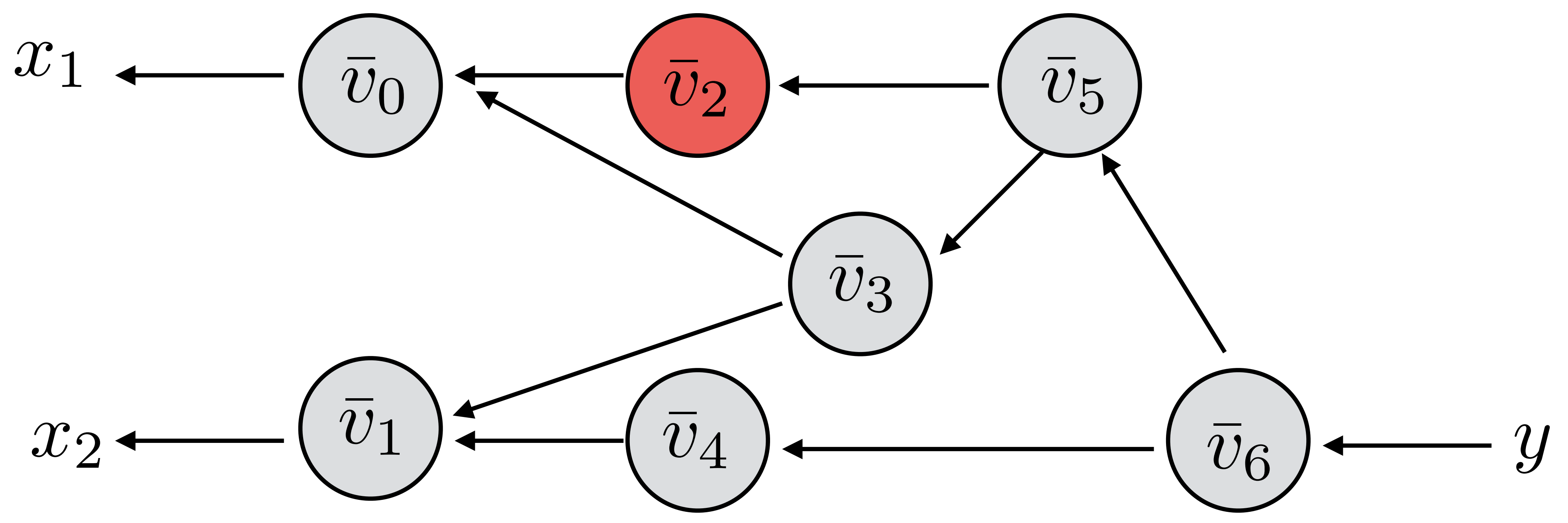
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

AutoDiff - **Reverse Mode**





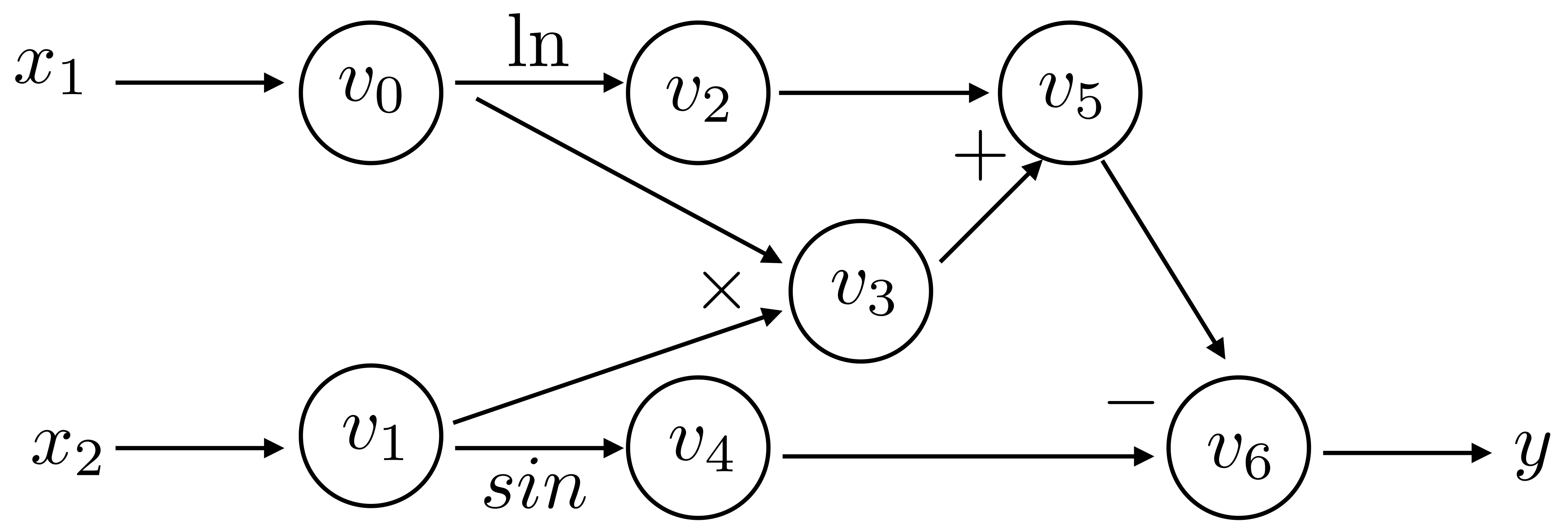
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

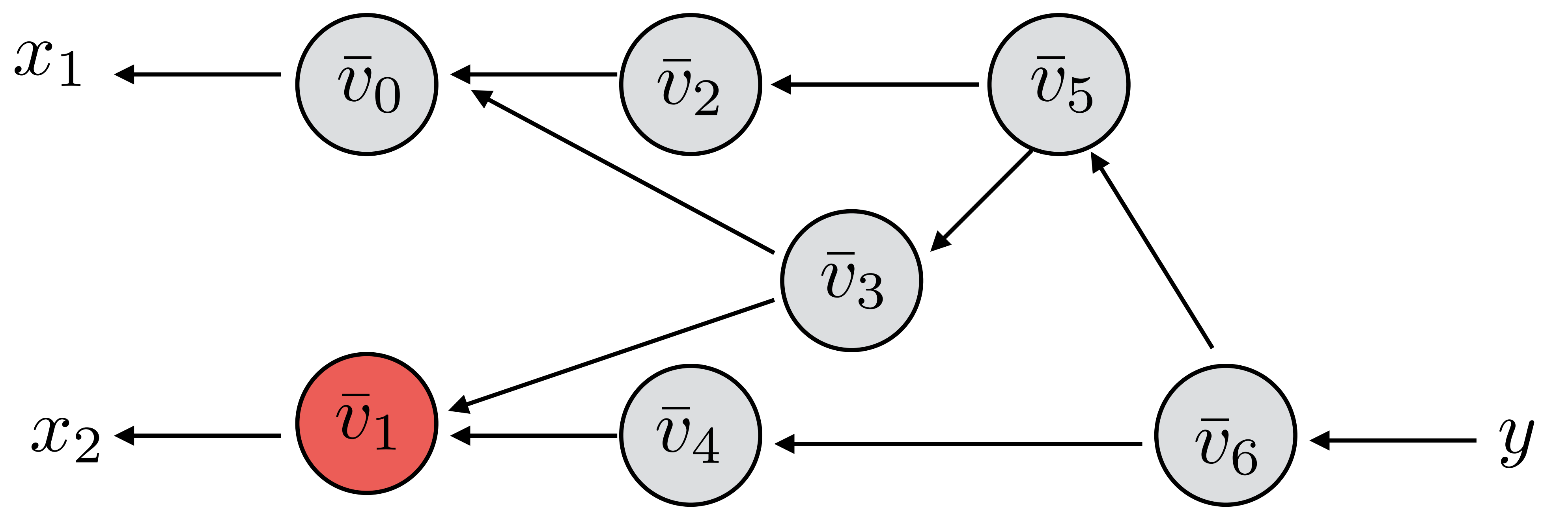
$\bar{v}_2 = \bar{v}_5 \dfrac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \dfrac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**



Forward Evaluation Trace:

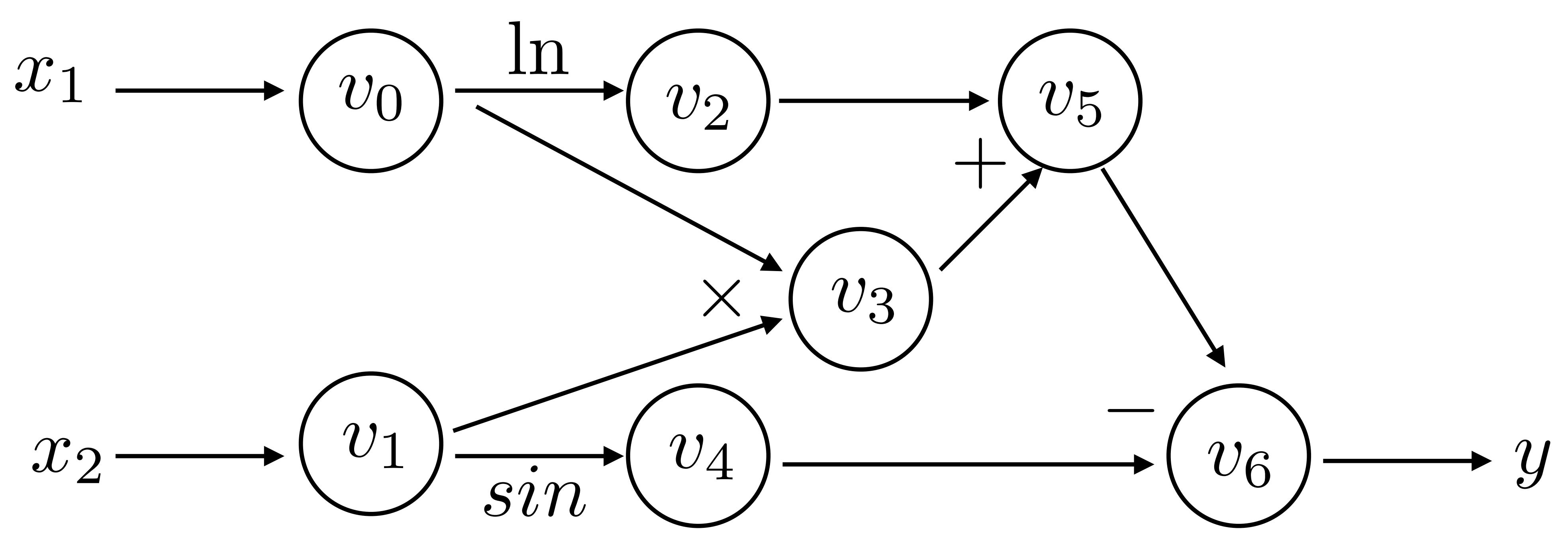
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652



Backwards Derivative Trace:

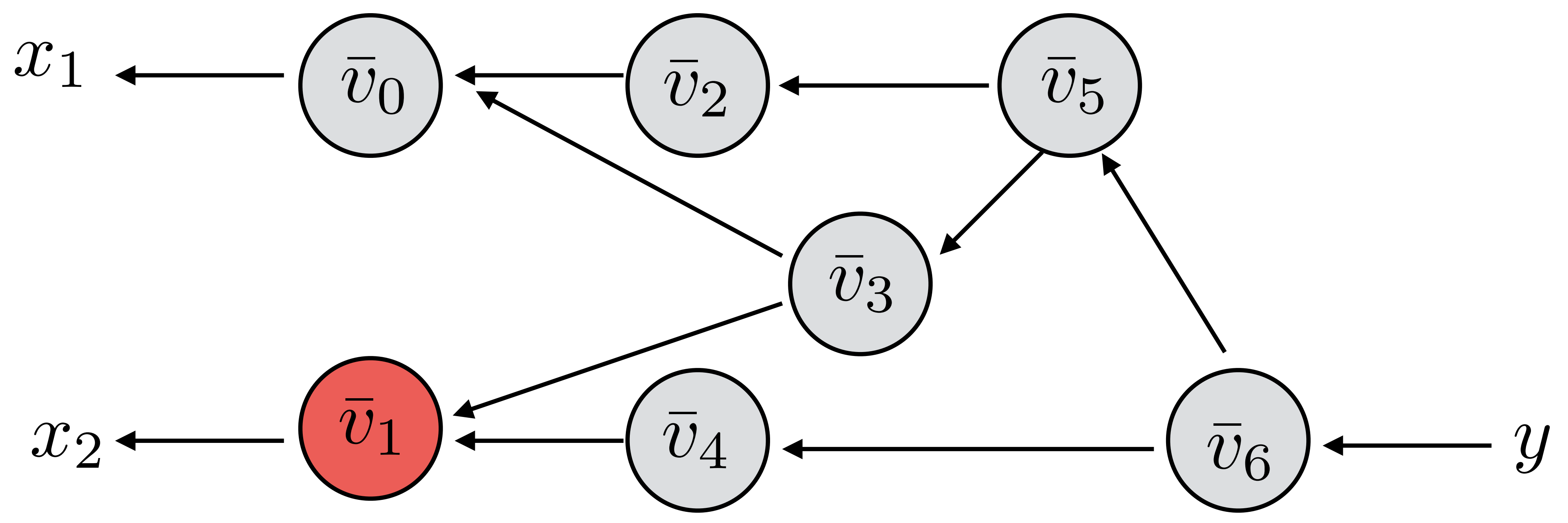
$\bar{v}_1$	
$\bar{v}_2 = \bar{v}_5 \dfrac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \dfrac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**



Forward Evaluation Trace:

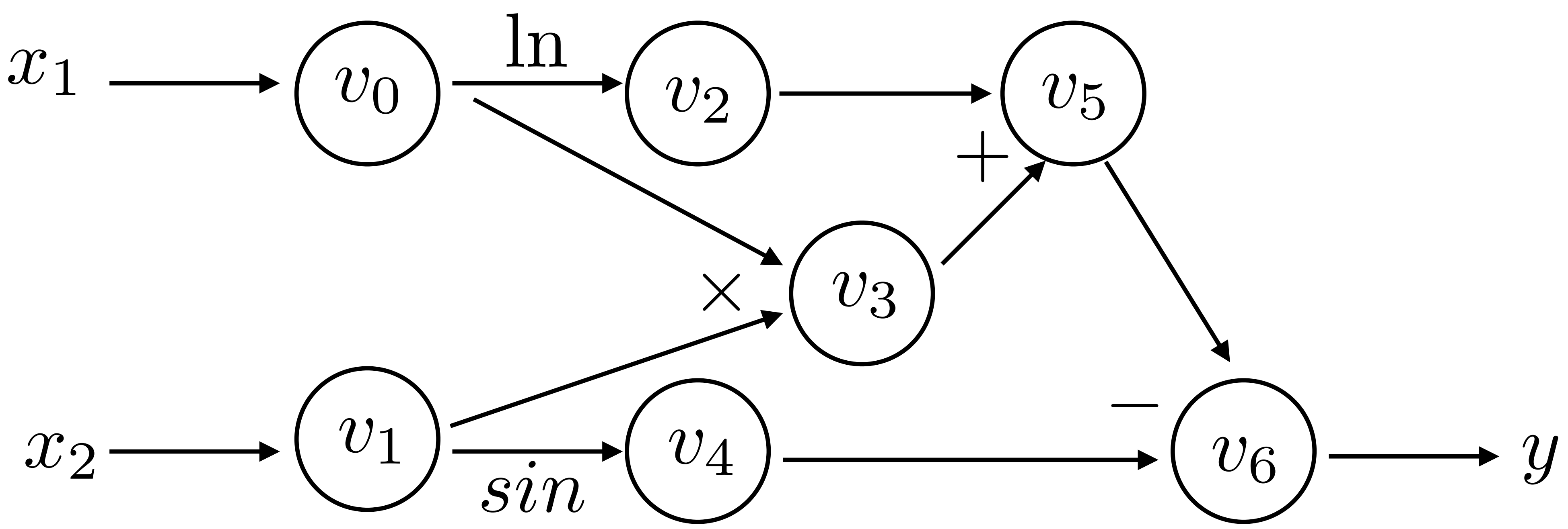
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

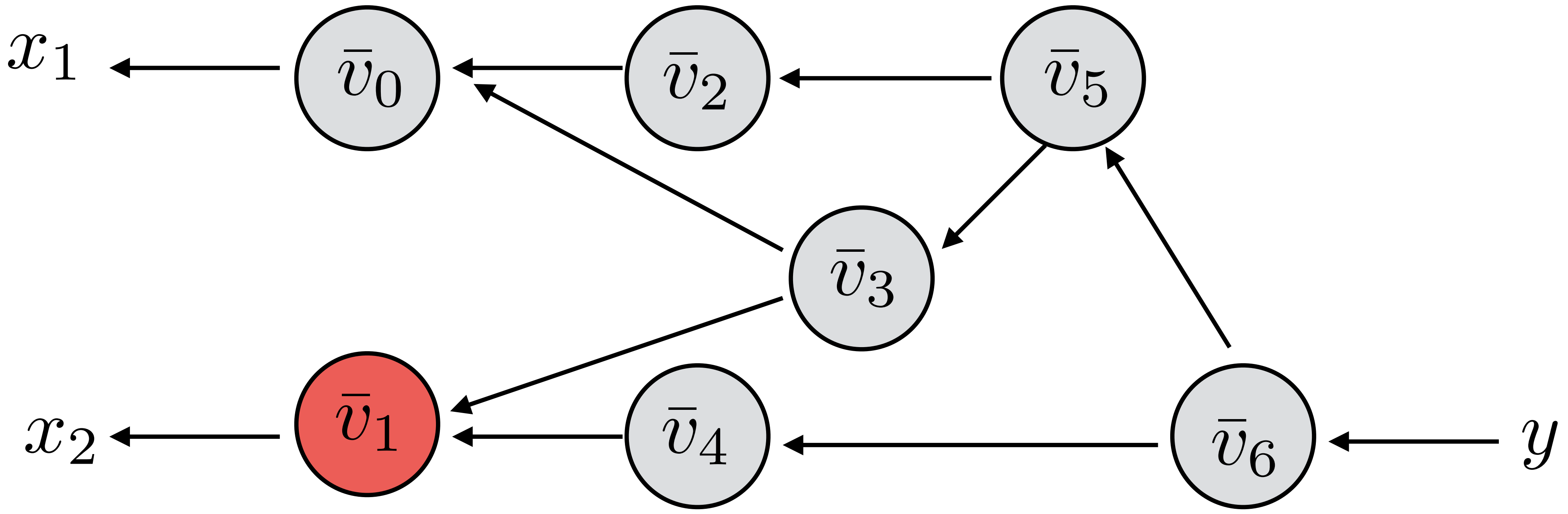


Backwards Derivative Trace:

$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





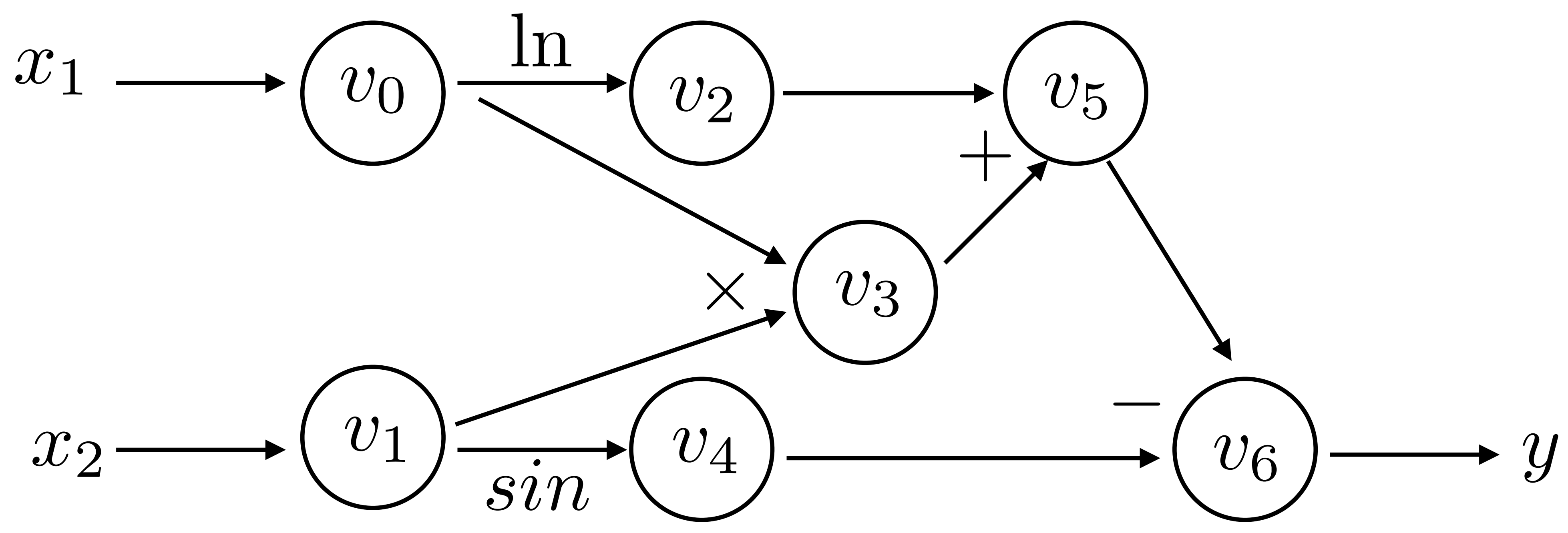
Forward Evaluation Trace:

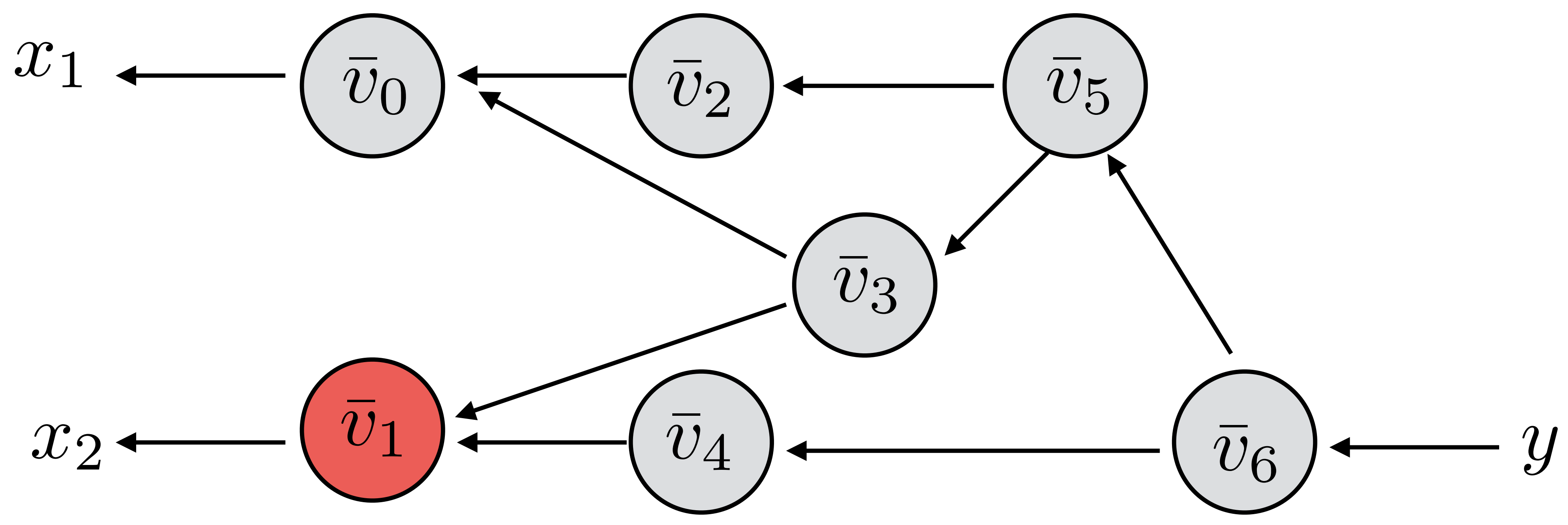
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





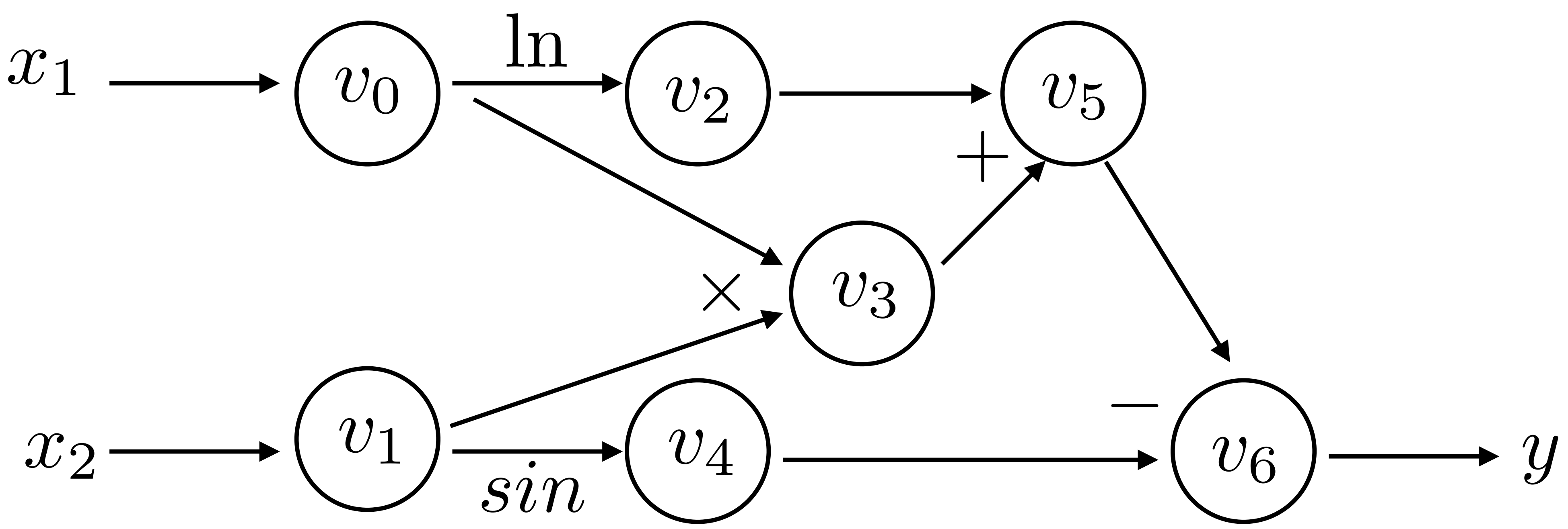
Forward Evaluation Trace:

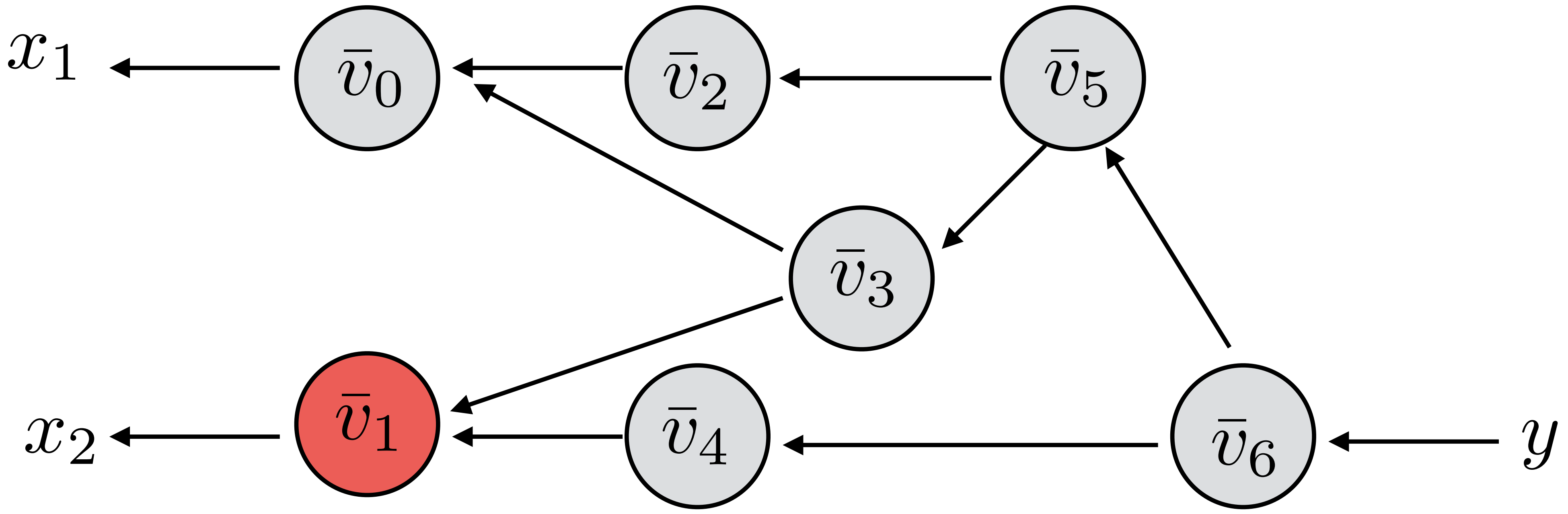
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1} = \bar{v}_3 v_0 + \bar{v}_4 cos(v_1)$	
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





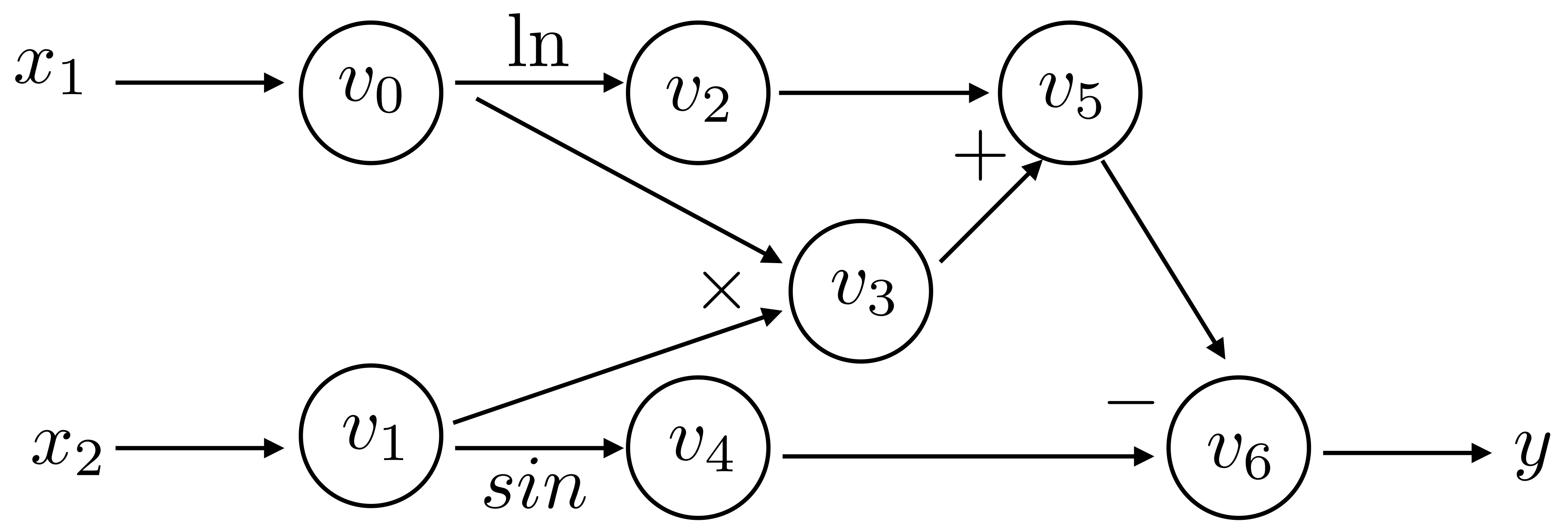
Forward Evaluation Trace:

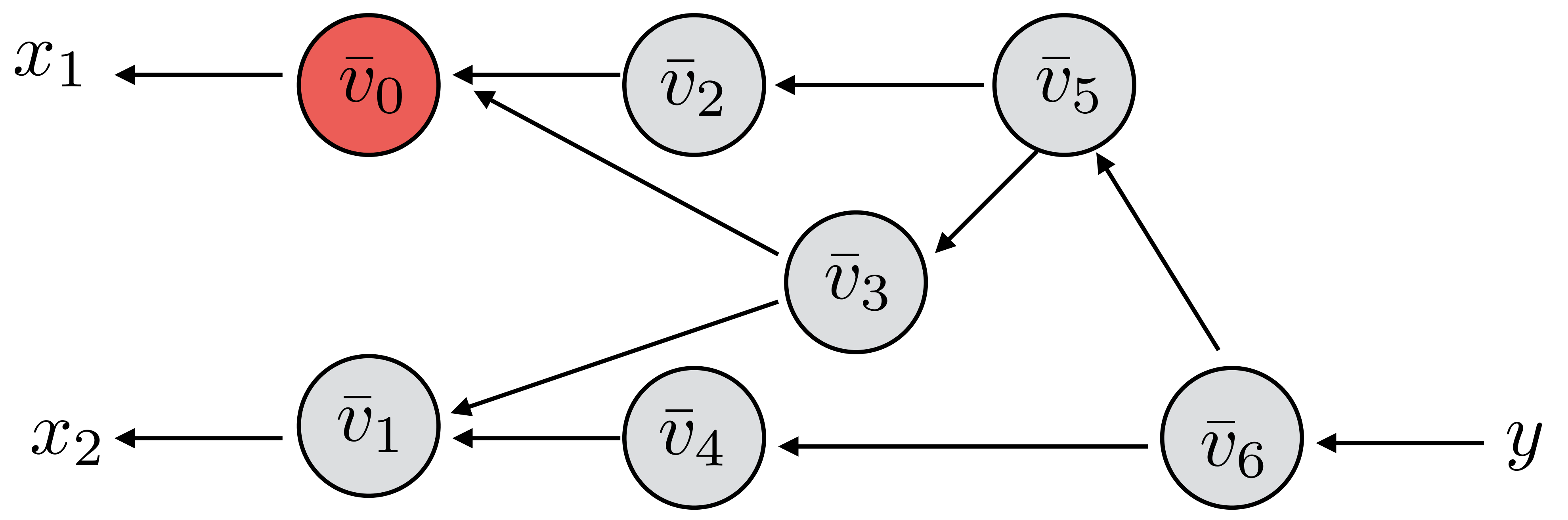
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1} = \bar{v}_3 v_0 + \bar{v}_4 cos(v_1)$	1.716
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





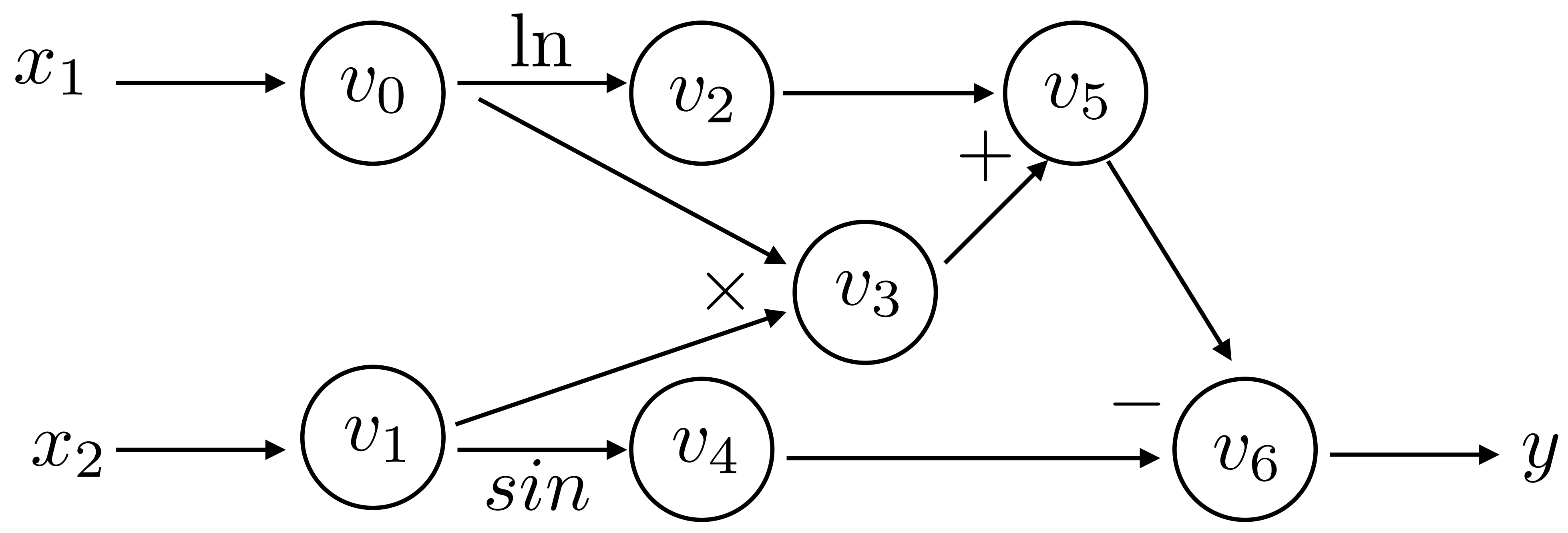
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

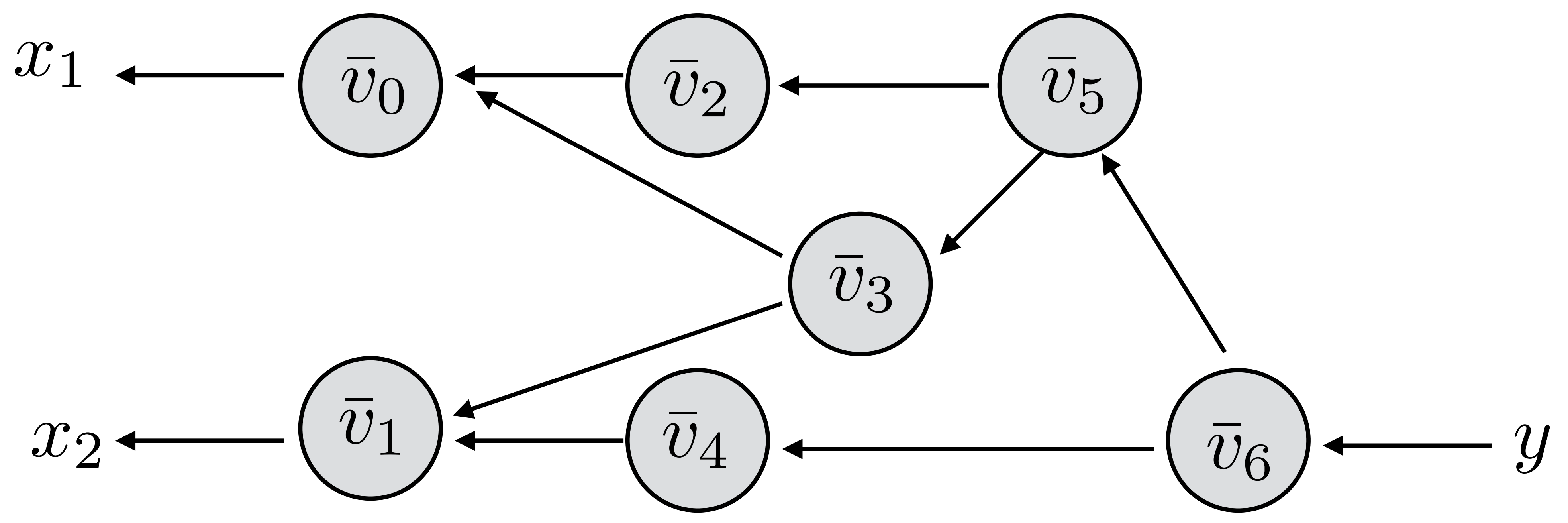
$\bar{v}_0 = \bar{v}_3 \frac{\partial v_3}{\partial v_0} + \bar{v}_2 \frac{\partial v_2}{\partial v_0} = \bar{v}_3 v_1 + \bar{v}_2 \frac{1}{v_0}$	5.5
$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1} = \bar{v}_3 v_0 + \bar{v}_4 cos(v_1)$	1.716
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**



Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652



Backwards Derivative Trace:

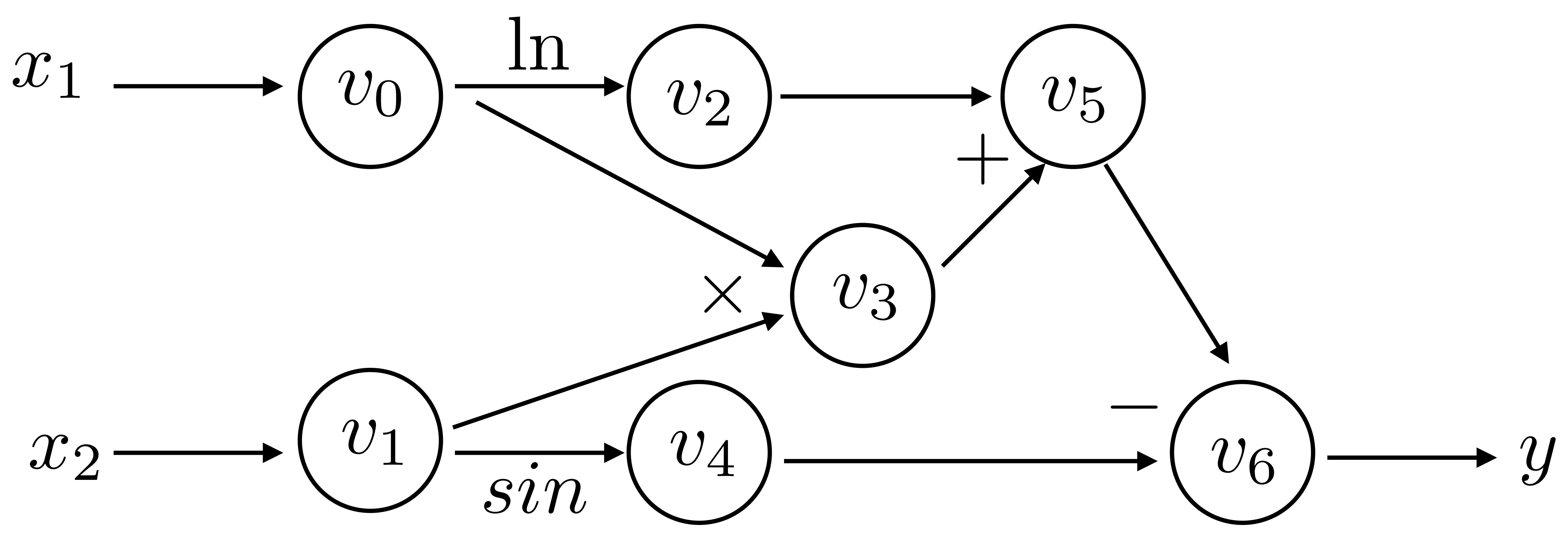
$\bar{v}_0 = \bar{v}_3 \frac{\partial v_3}{\partial v_0} + \bar{v}_2 \frac{\partial v_2}{\partial v_0} = \bar{v}_3 v_1 + \bar{v}_2 \frac{1}{v_0}$	**5.5**
$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1} = \bar{v}_3 v_0 + \bar{v}_4 cos(v_1)$	**1.716**
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

Automatic Differentiation (AutoDiff)

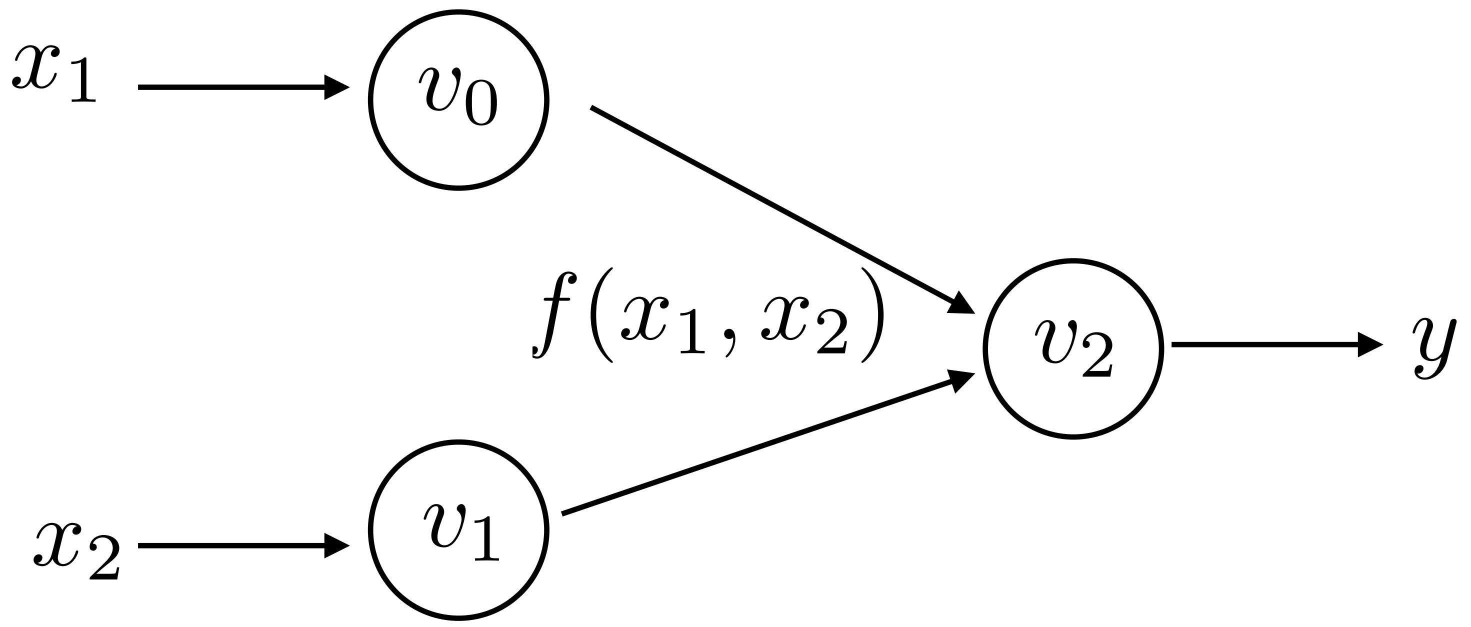
$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

AutoDiff can be done at various **granularities**

Elementary function granularity:

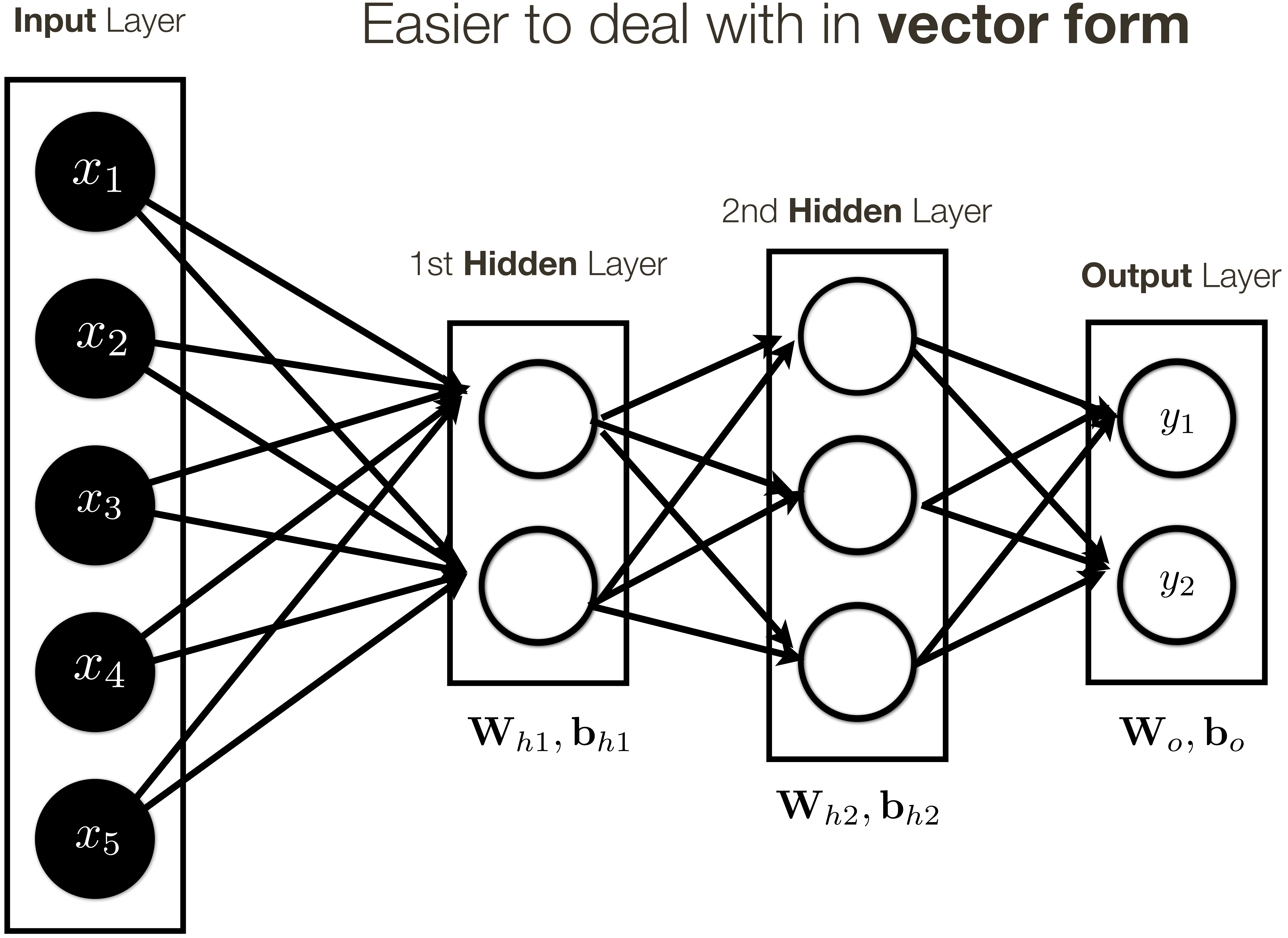


Complex function granularity:



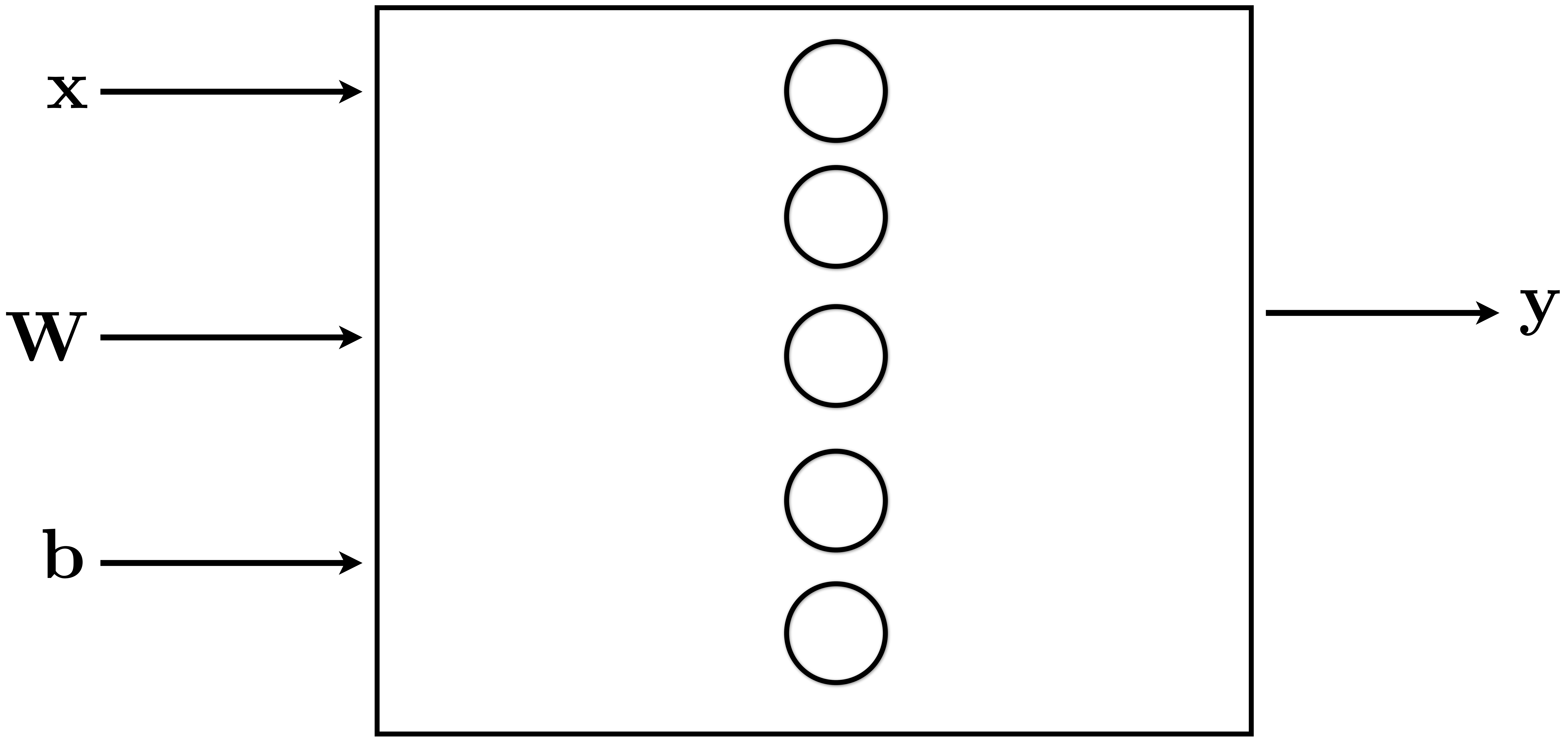
Backpropagation Practical Issues

Easier to deal with in **vector form**



Backpropagation Practical Issues

$$\mathbf{y} = f(\mathbf{W}, \mathbf{b}, \mathbf{x}) = \mathbf{sigmoid}(\mathbf{W} \cdot \mathbf{x} + \mathbf{b})$$

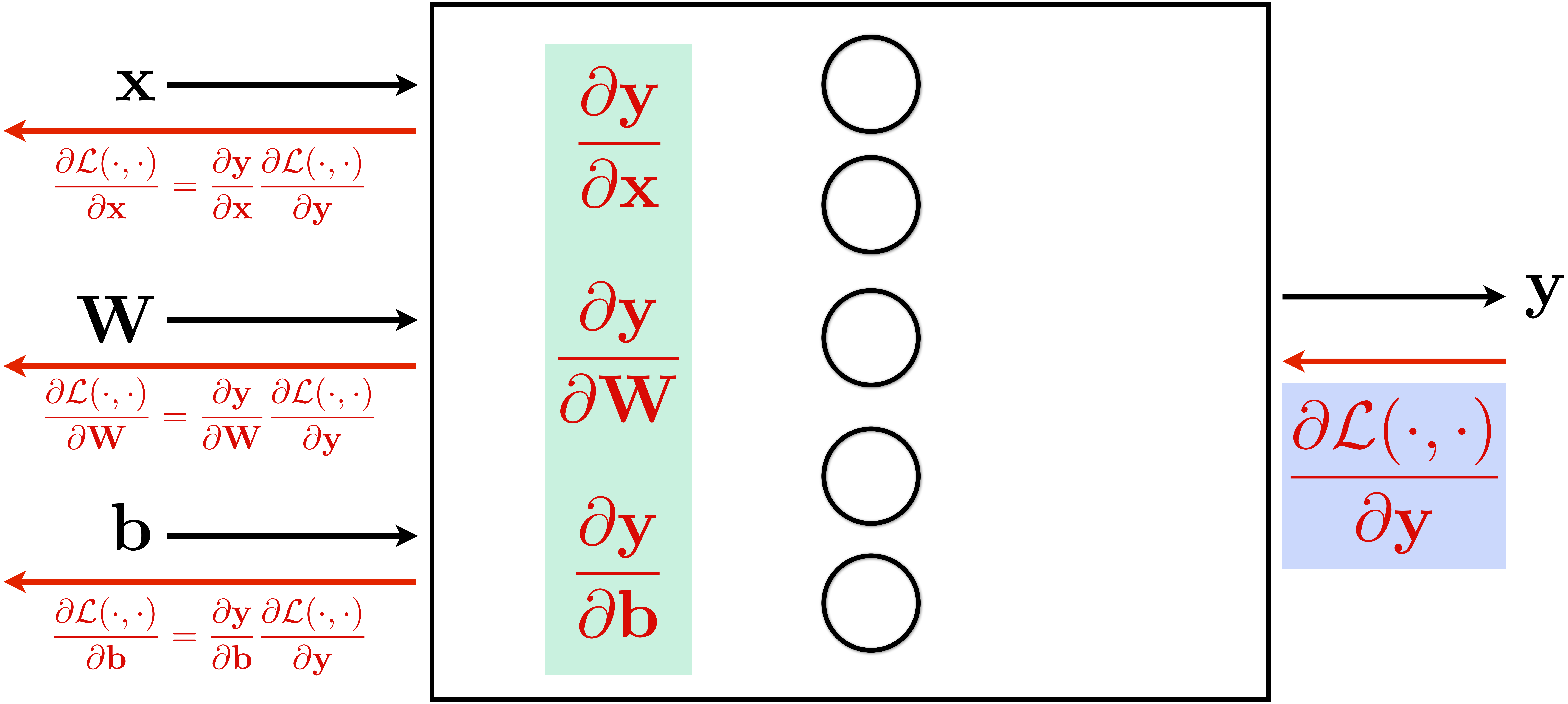


Backpropagation Practical Issues

"**local**" Jacobians
(matrix of partial derivatives, e.g. size |x| x |y|)

"**backprop**" Gradient

$$\mathbf{y} = f(\mathbf{W}, \mathbf{b}, \mathbf{x}) = \mathbf{sigmoid}(\mathbf{W} \cdot \mathbf{x} + \mathbf{b})$$

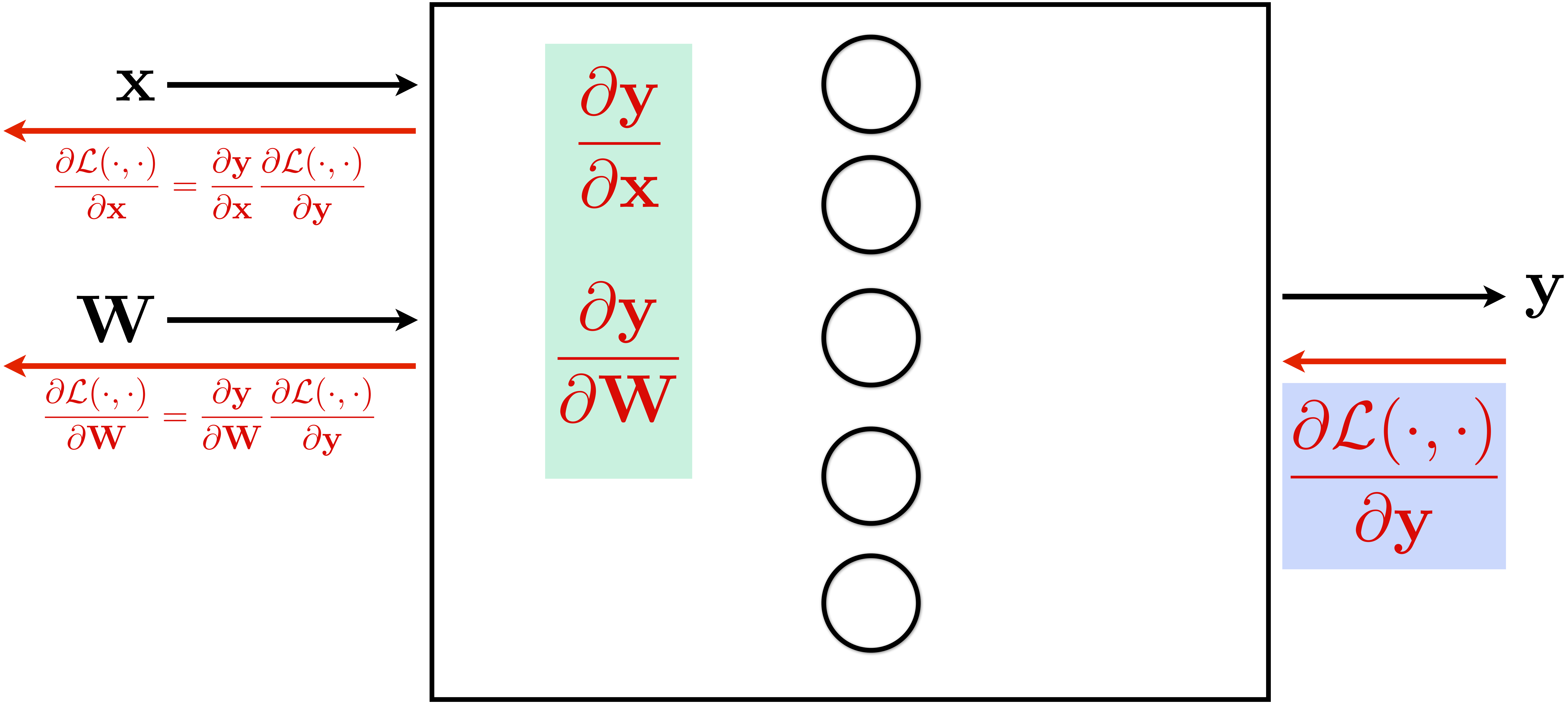


Backpropagation Practical Issues

"**local**" Jacobians
(matrix of partial derivatives, e.g. size |x| x |y|)

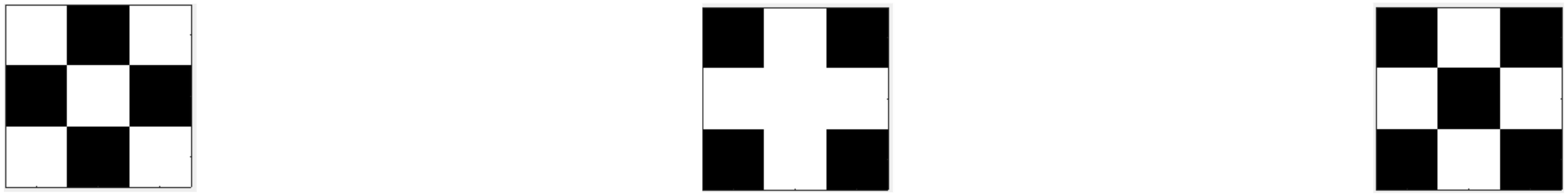
"**backprop**" Gradient

$$\mathbf{y} = f(\mathbf{W}, \mathbf{b}, \mathbf{x}) = \mathbf{W} \cdot \mathbf{x}$$



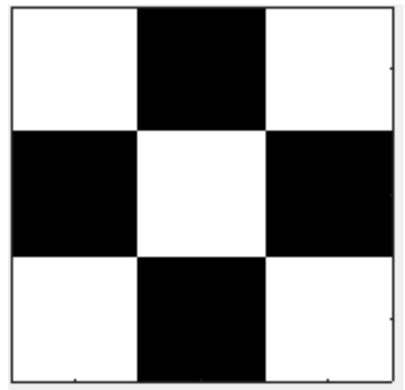
Example: Let's Build (world smallest) Neural Network

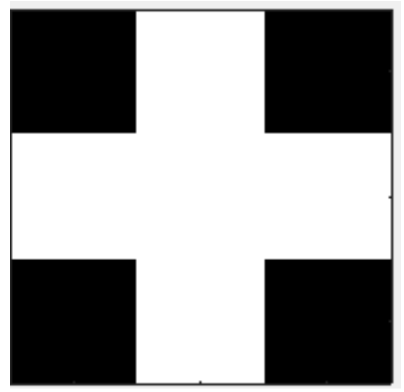
Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

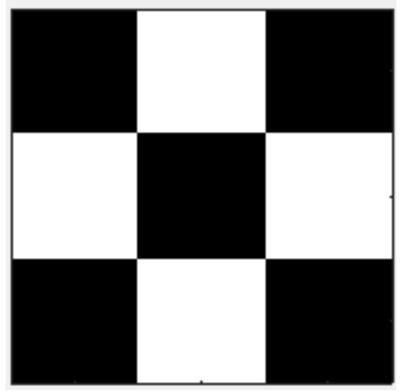


Example: Let’s Build (world smallest) Neural Network

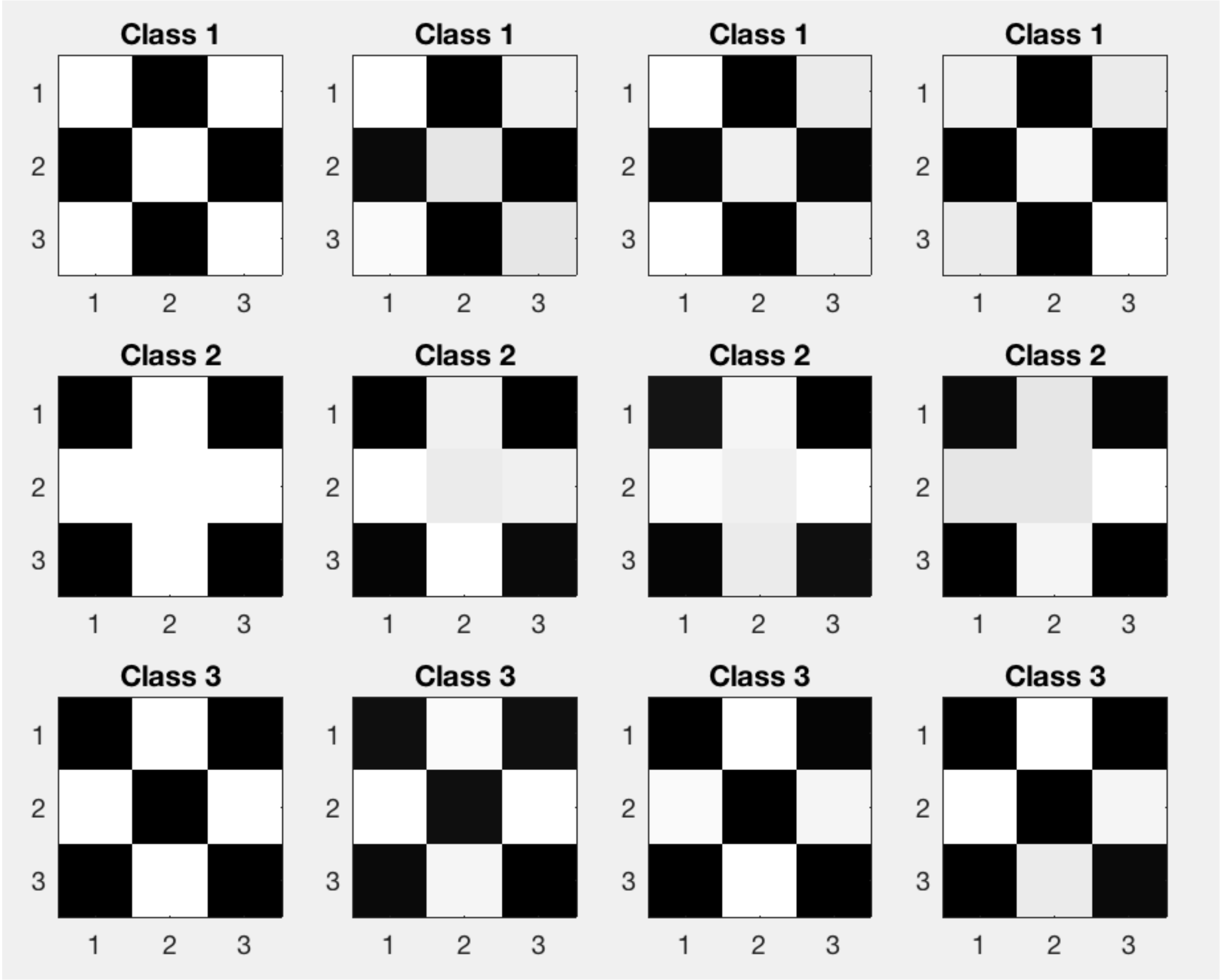
Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images





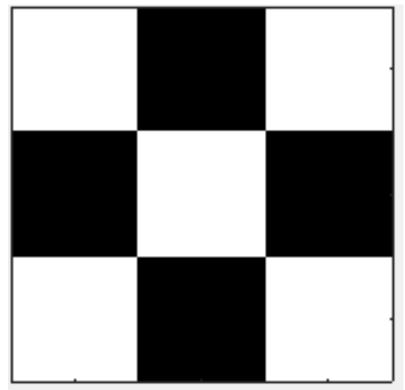


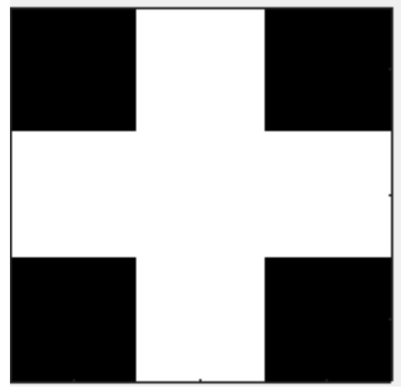
We will need some labeled data

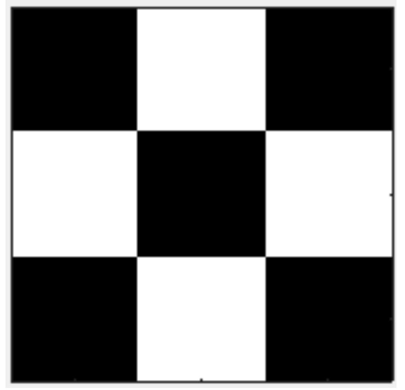


Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

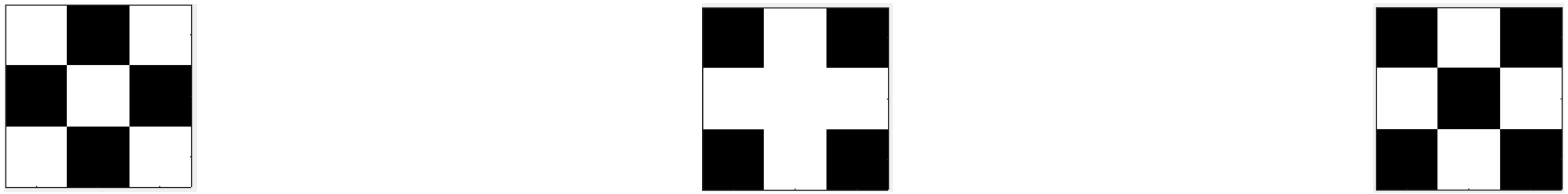


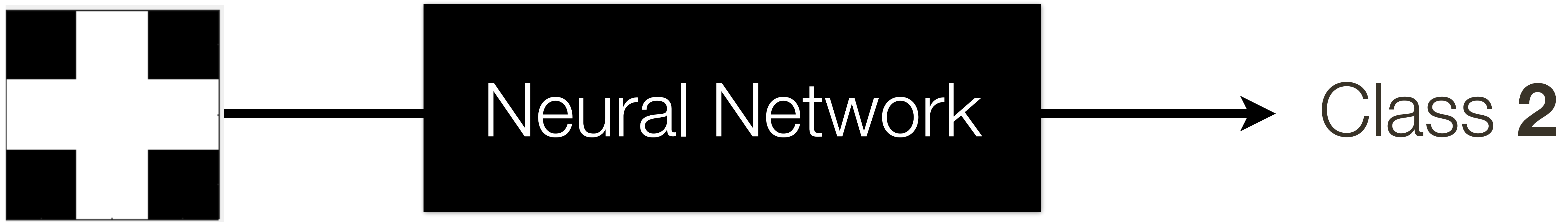




Example: Let's Build (world smallest) Neural Network

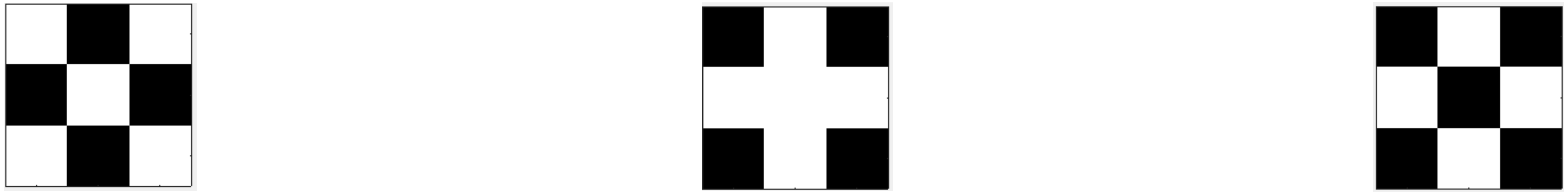
Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

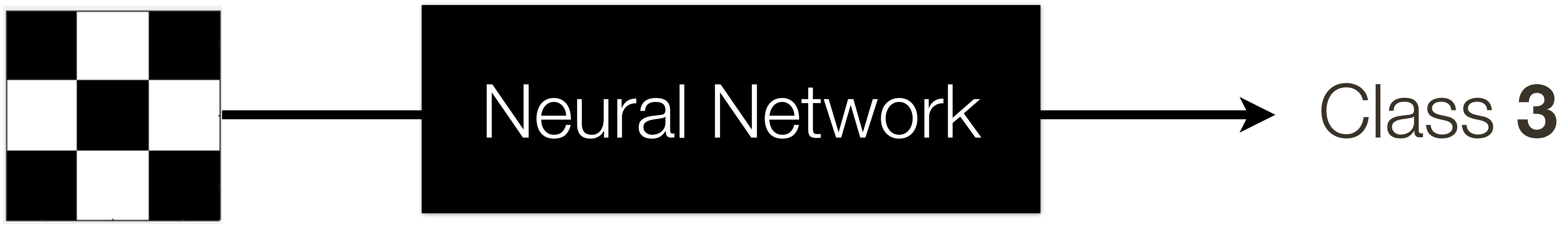




Example: Let's Build (world smallest) Neural Network

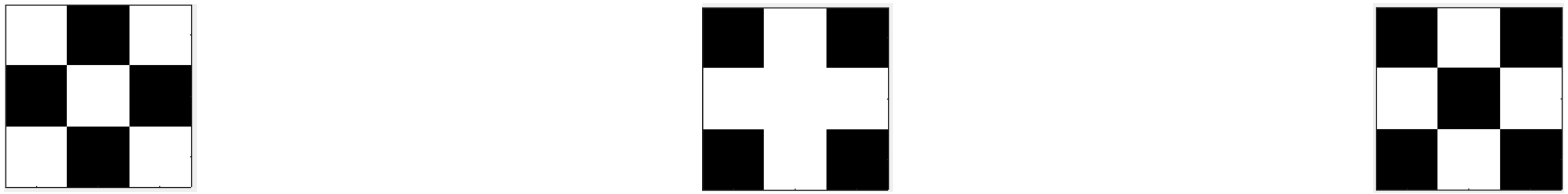
Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images





Example: Let’s Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

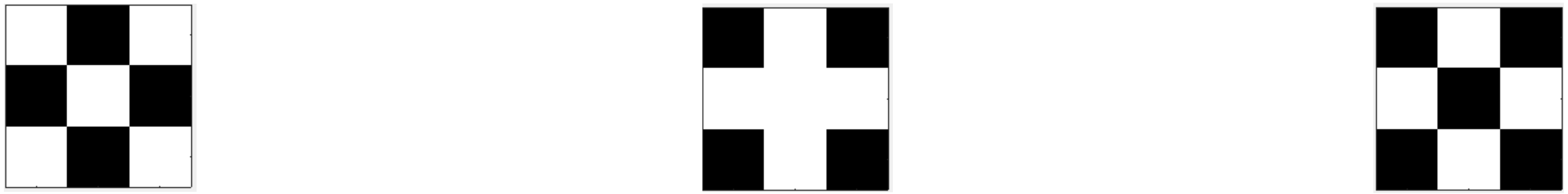


What do we need to do?

First, lets re-formulate the problem

Example: Let’s Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

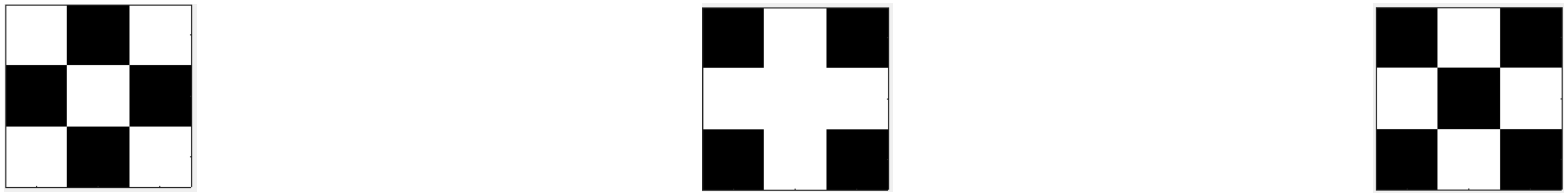


What do we need to do?

First, lets re-formulate the problem

Example: Let’s Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

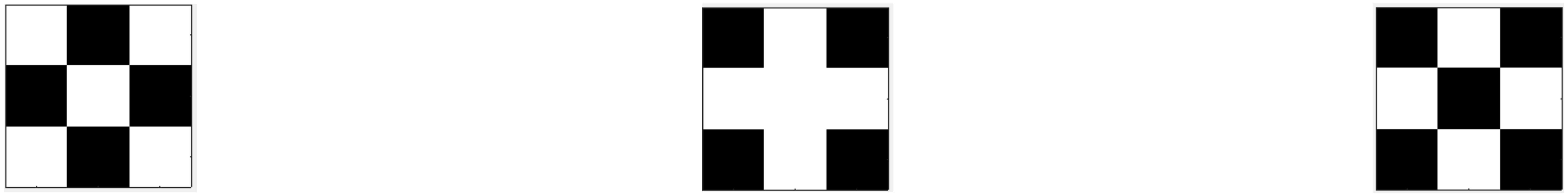


Now, lets build a **network**!

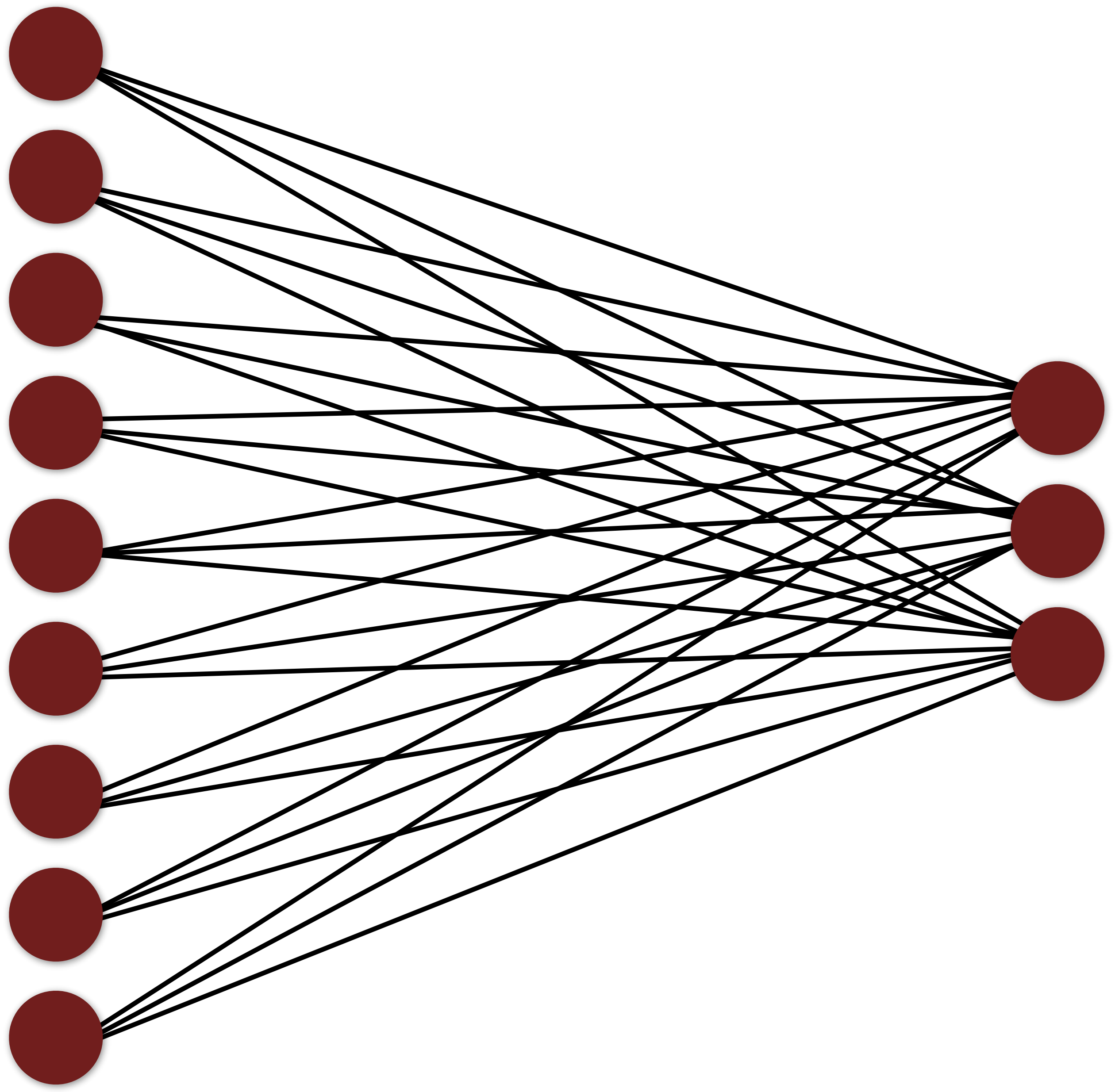
How many inputs should the network have? How neuron outputs?

Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images



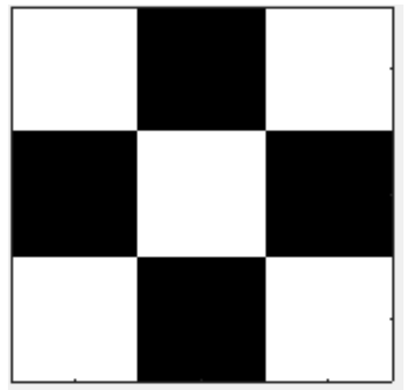
Input Layer **Output** Layer

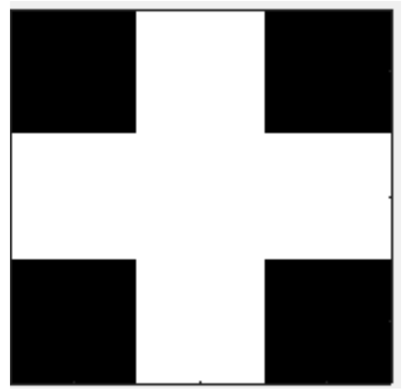


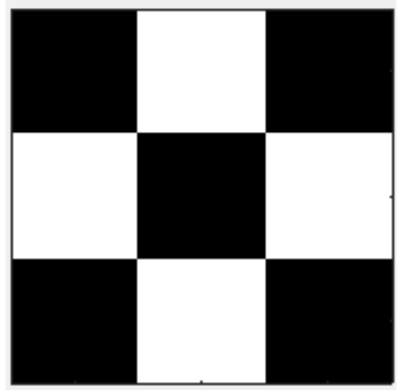
What else is missing for us to train it?

Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images



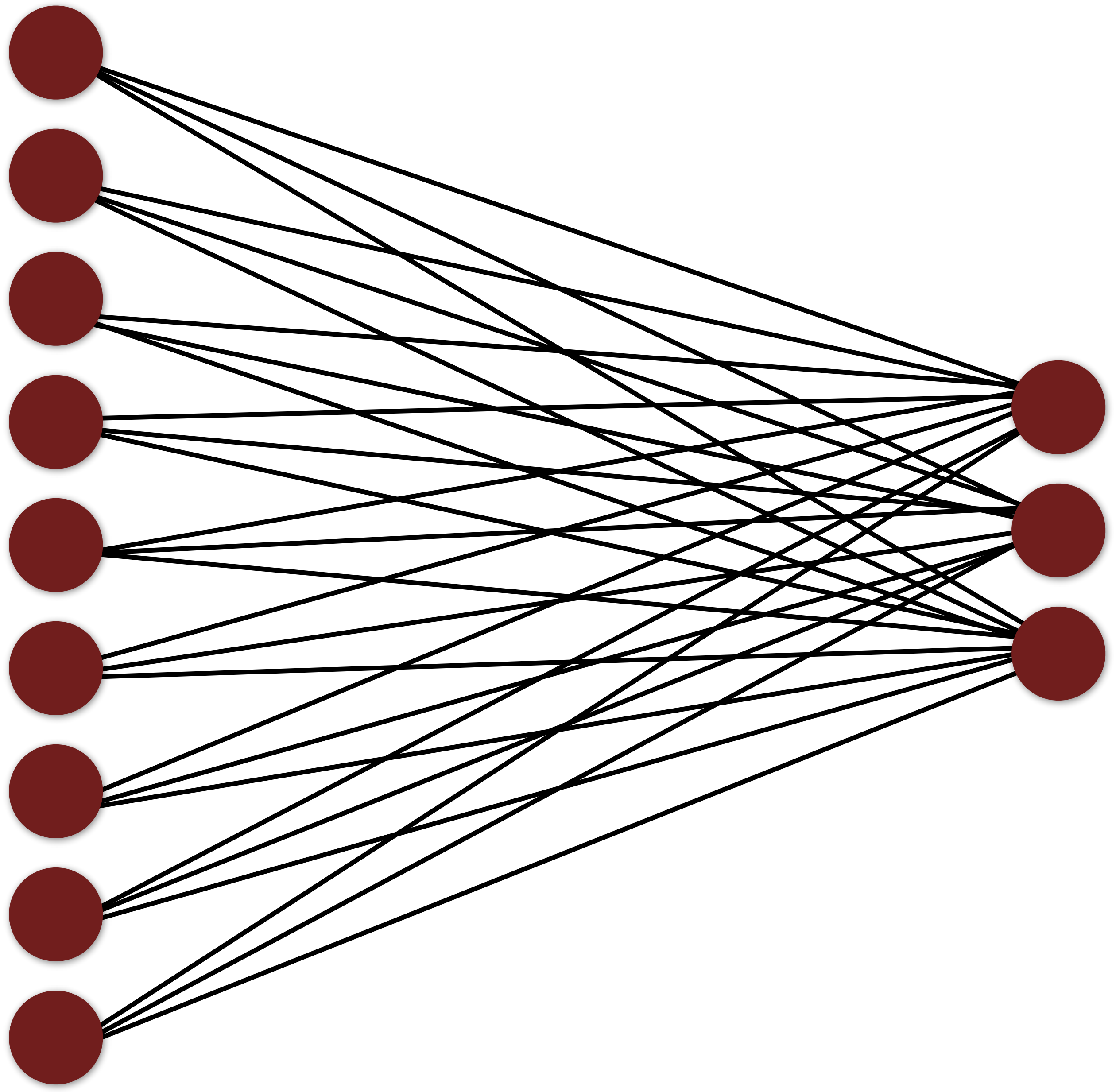




Input Layer

Output Layer

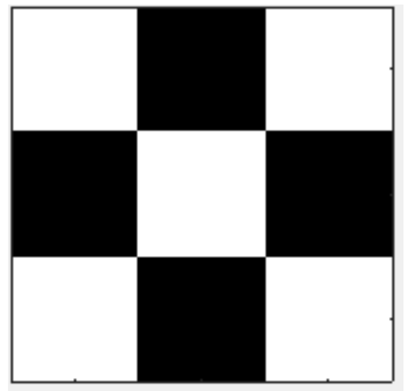
Loss

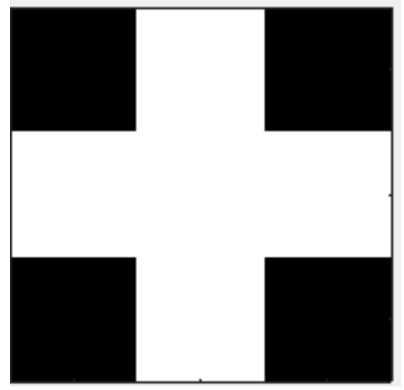


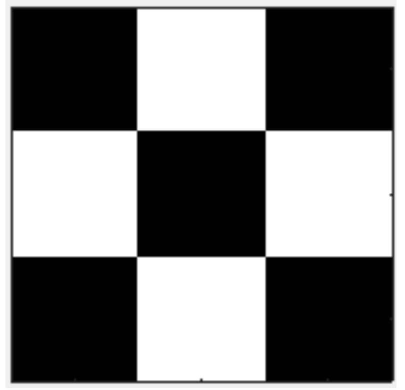
$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images



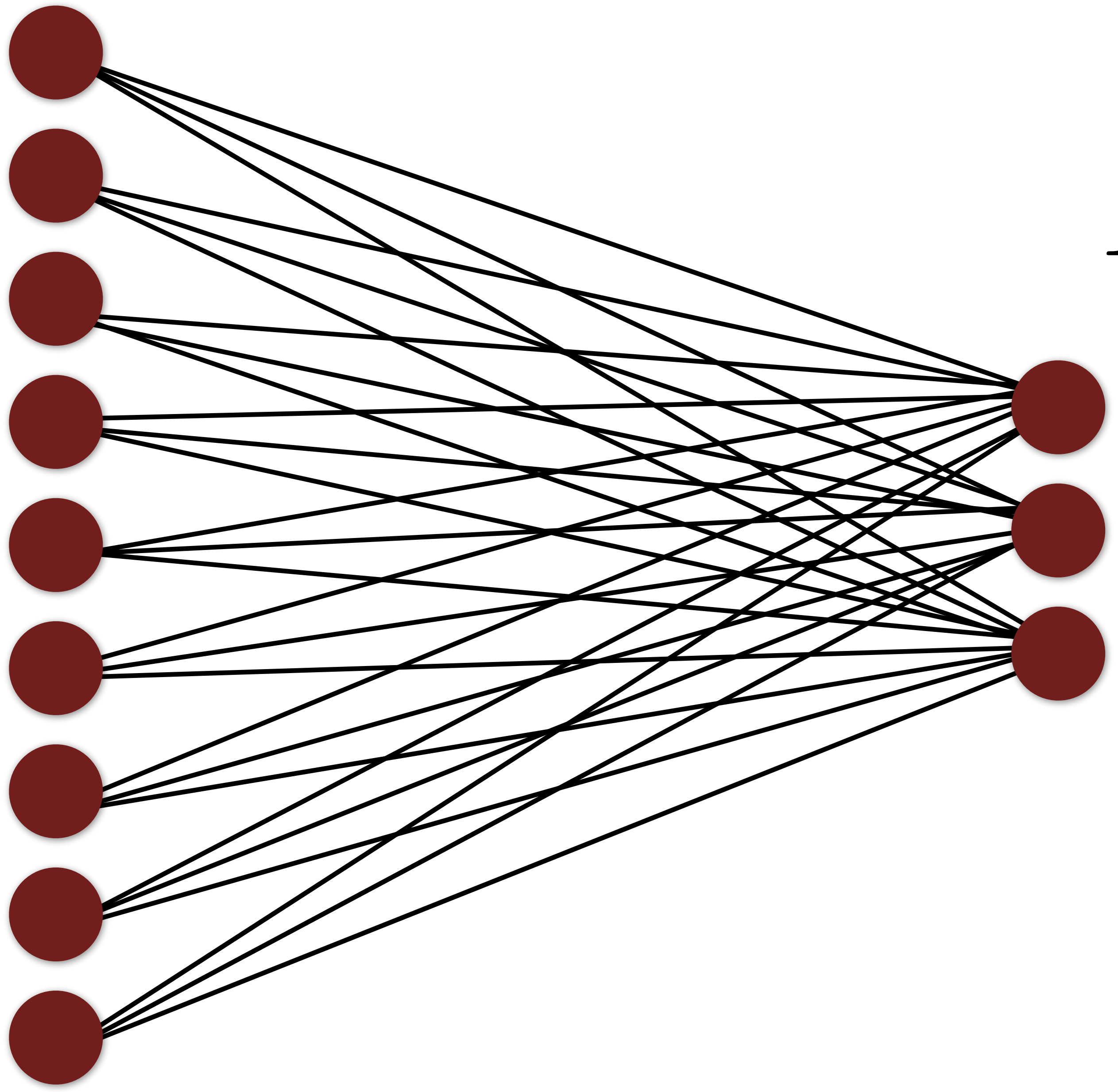




Input Layer

Output Layer

Loss



$$L_1 = -log\left(\frac{e^{\sum_{i=1}^{9}\sigma(w_{1,i}x_i+b_1)}}{\sum_{j=1}^{3} e^{\sum_{i=1}^{9}\sigma(w_{1,i}x_i+b_1)}}\right)$$