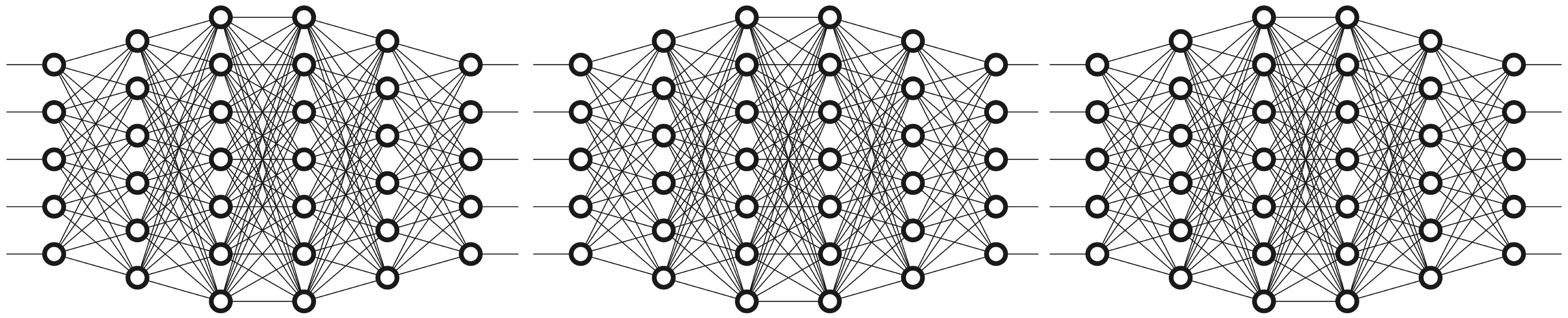
CPSC 425: Computer Vision



Lecture 33: Neural Networks

Menu for Today (**November 26, 2018**)

Topics:

— Neuron

— Neural Networks

— Layers and activation functions

— Backpropagation

Redings:

— **Today's** Lecture: N/A

— **Next** Lecture: N/A

Reminders:

— **Assignment 5**: Scene Recognition with Bag of Words due **last day of classes**

— Rules for **competition** are posted

Recall: Pareidolia

Pereidolia

Photo credit: reddit user [Liammm]

Today's "**fun**" Example: Deep Dream — Algorithmic Pareidolia

Lecture 32: Re-cap

Detection scores in the **deformable part model** are based on both appearance and location

The **deformable part model** is trained iteratively by alternating the steps

1. Assume components and part locations given; compute appearance and offset models
2. Assume appearance and offset models given; compute components and part locations

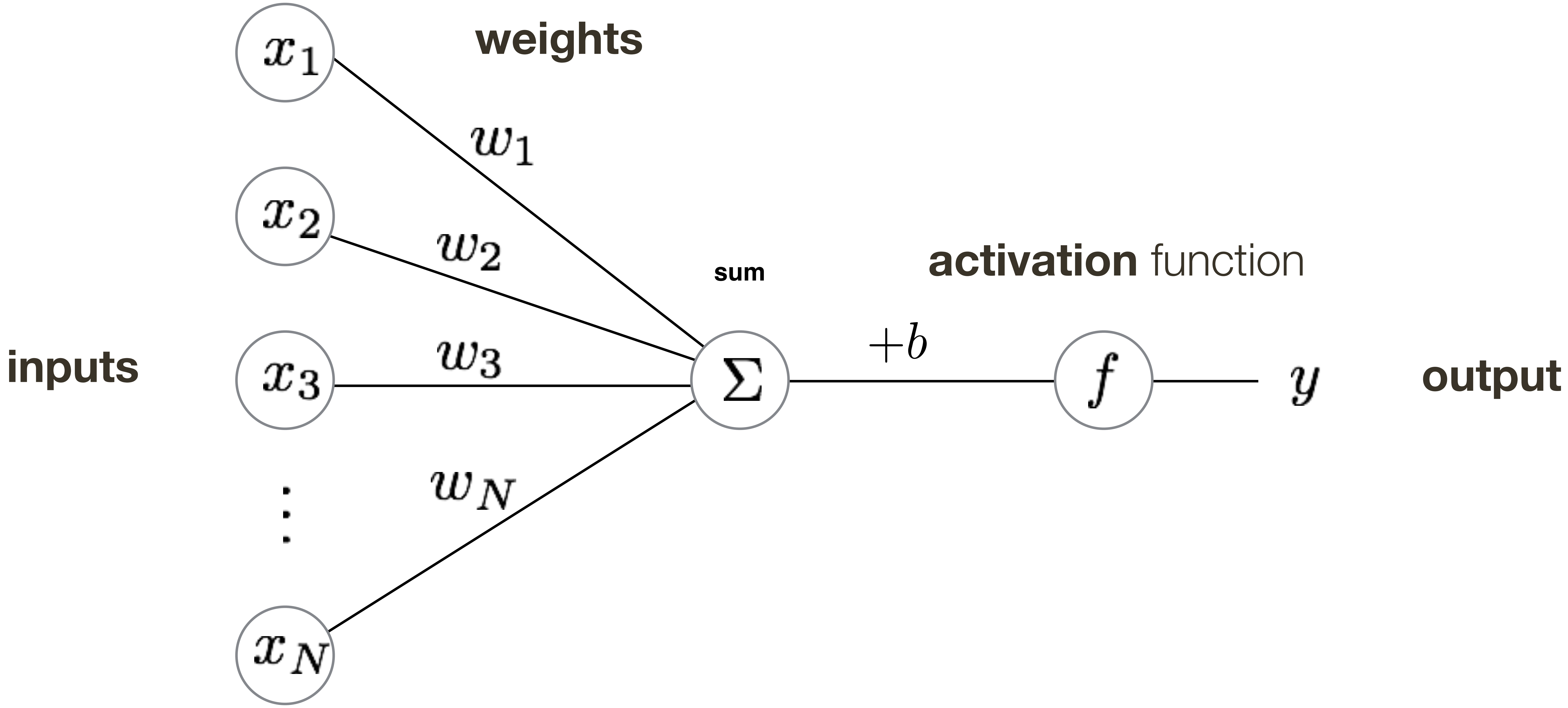
An **object proposal** algorithm generates a short list of regions with generic object-like properties that can be evaluated by an object detector in place of an exhaustive sliding window search

Warning:

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

… if you want to know more, take my **CPSC 532S**

A **Neuron**

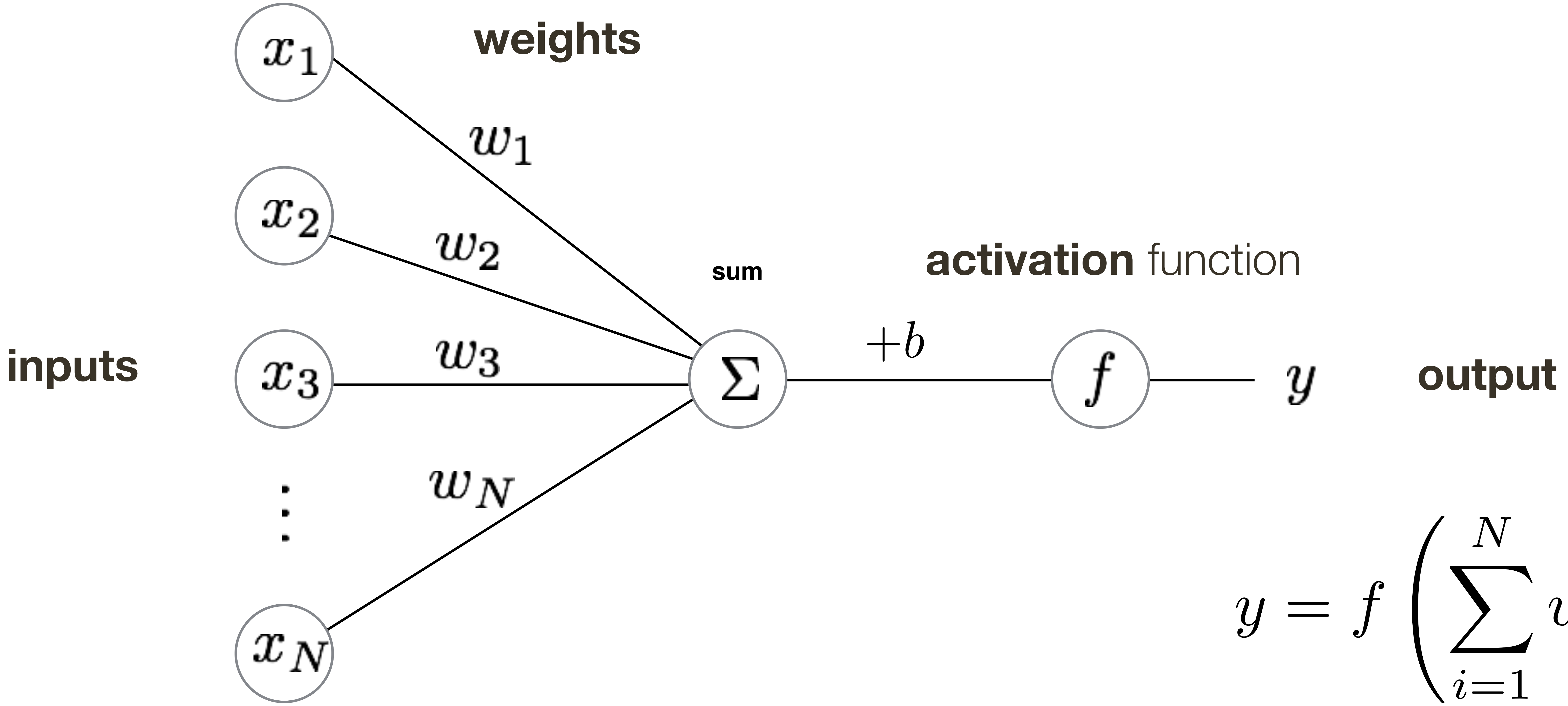


— The basic unit of computation in a neural network is a neuron.

— 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)

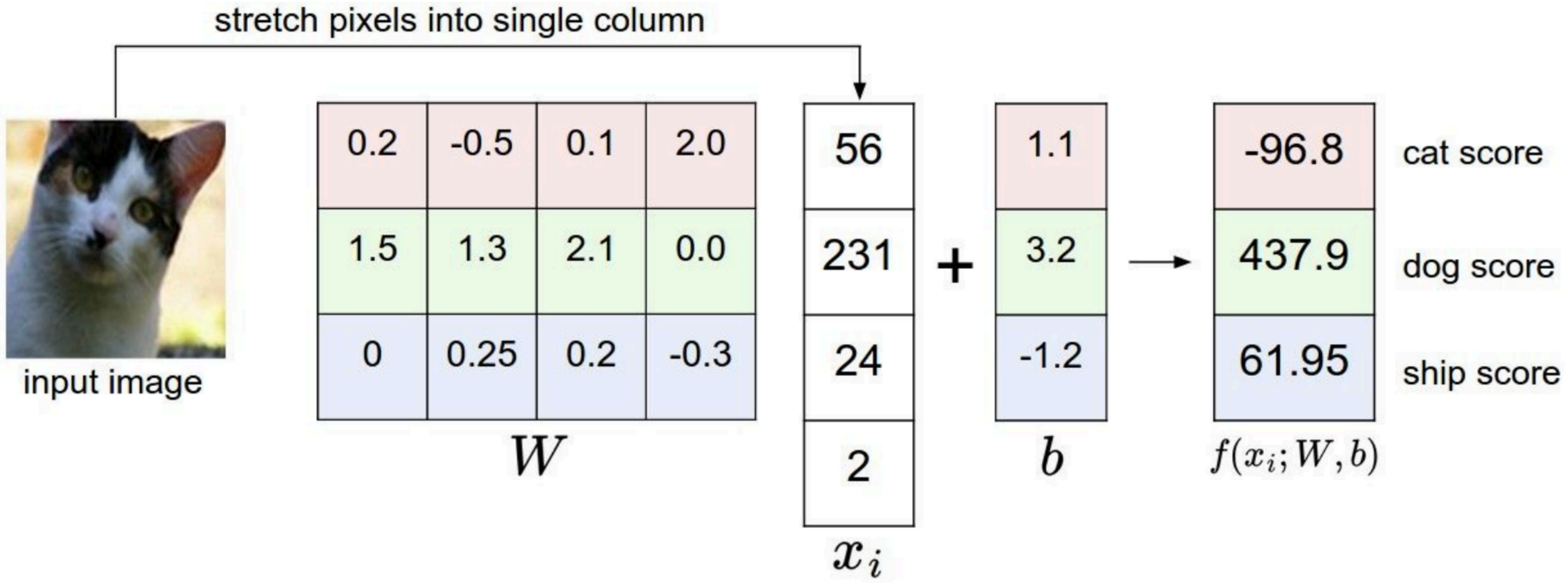
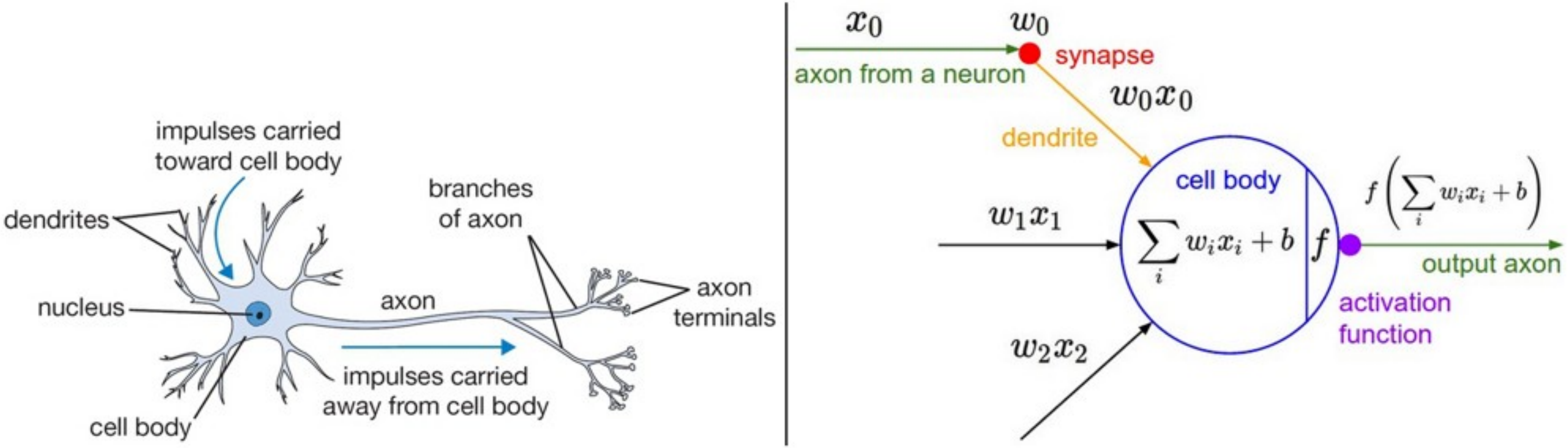


Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

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**

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

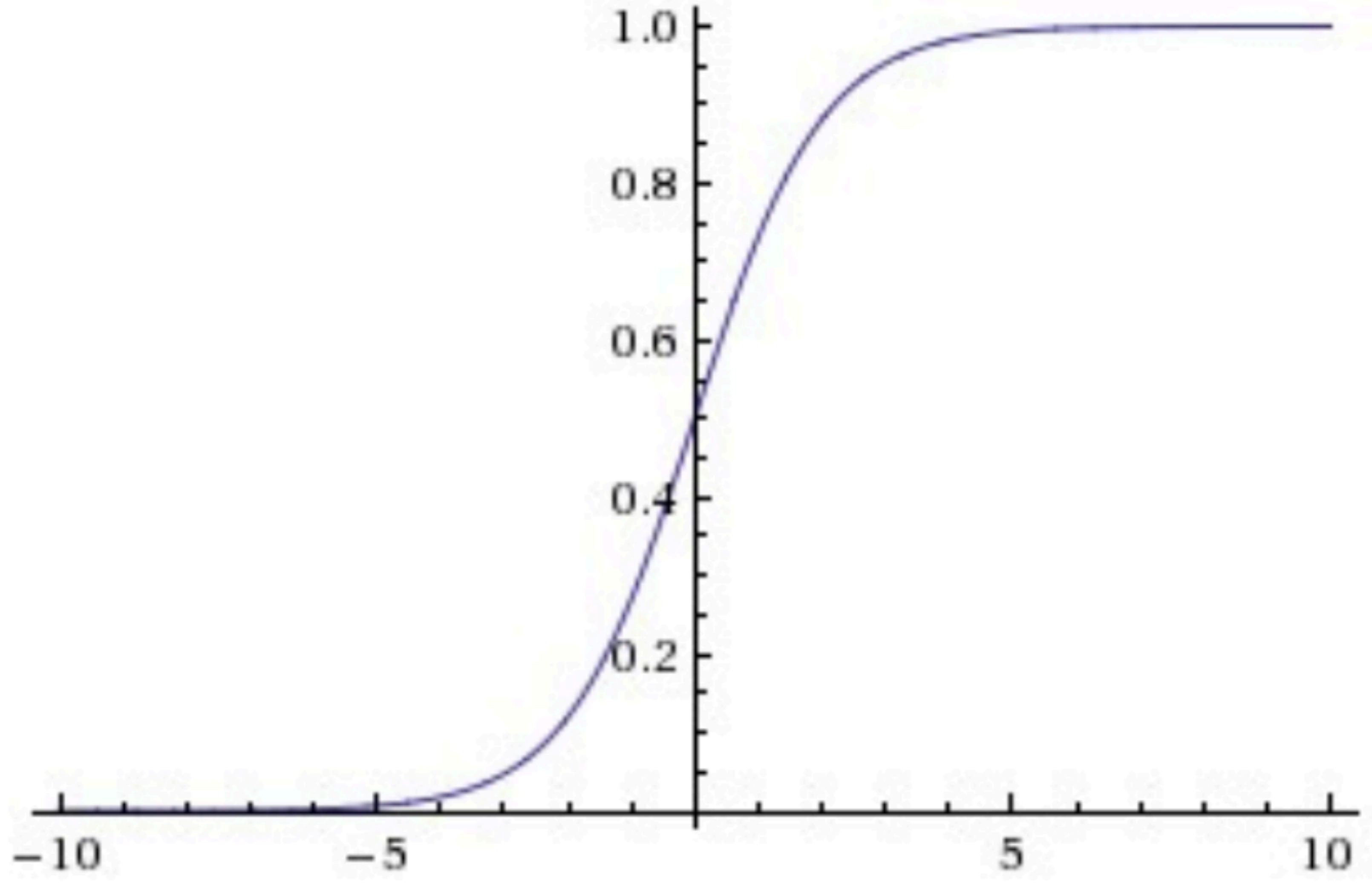


Figure credit: Fei-Fei and Karpathy

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)

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

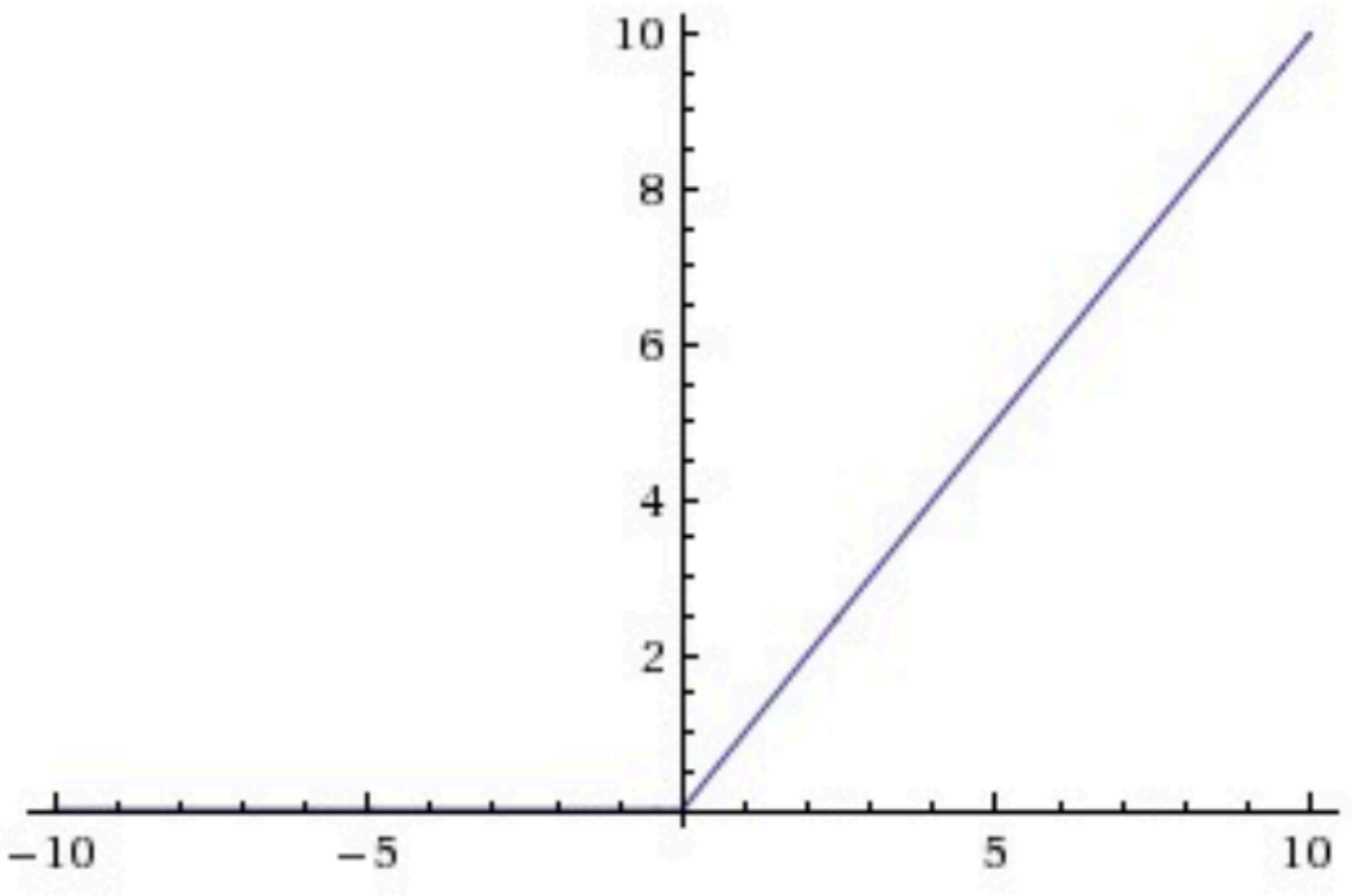
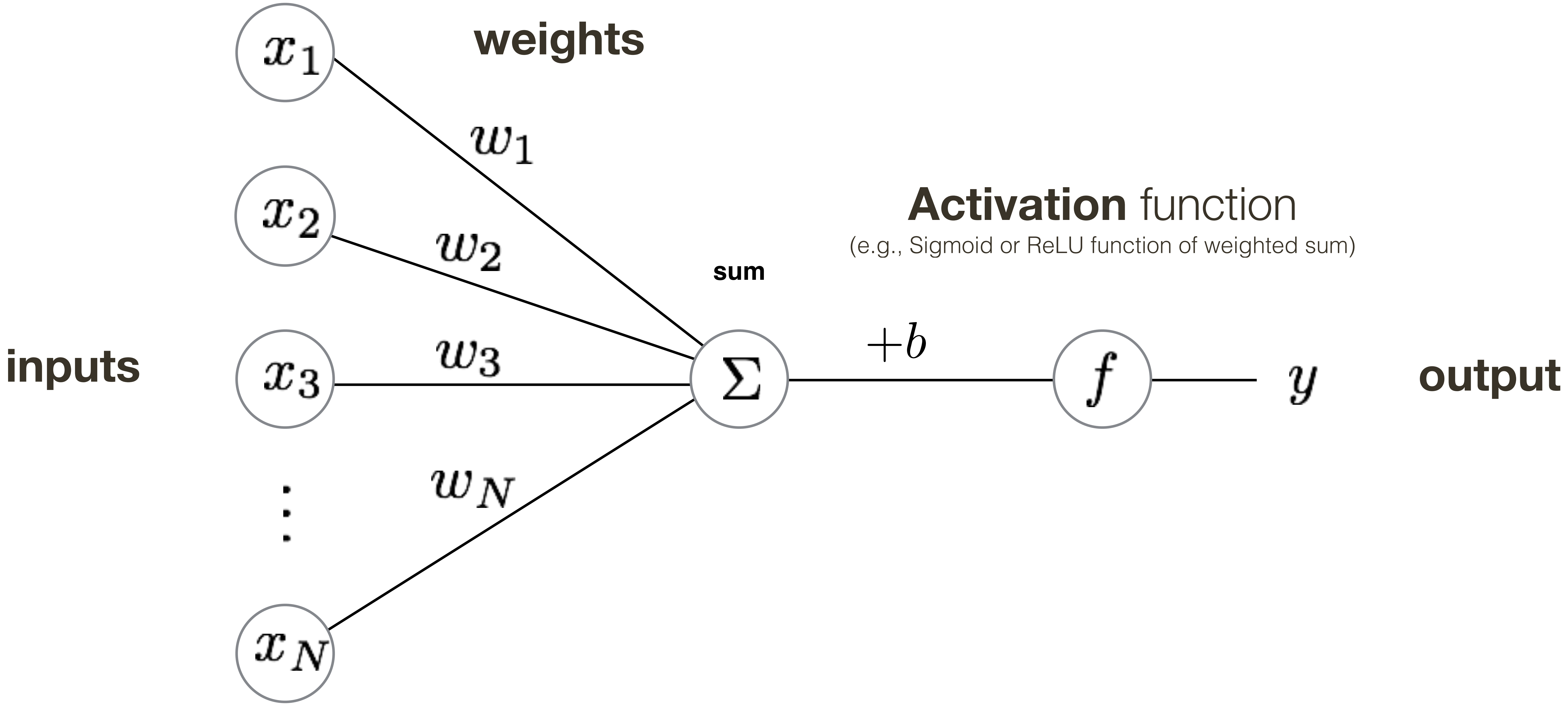


Figure credit: Fei-Fei and Karpathy

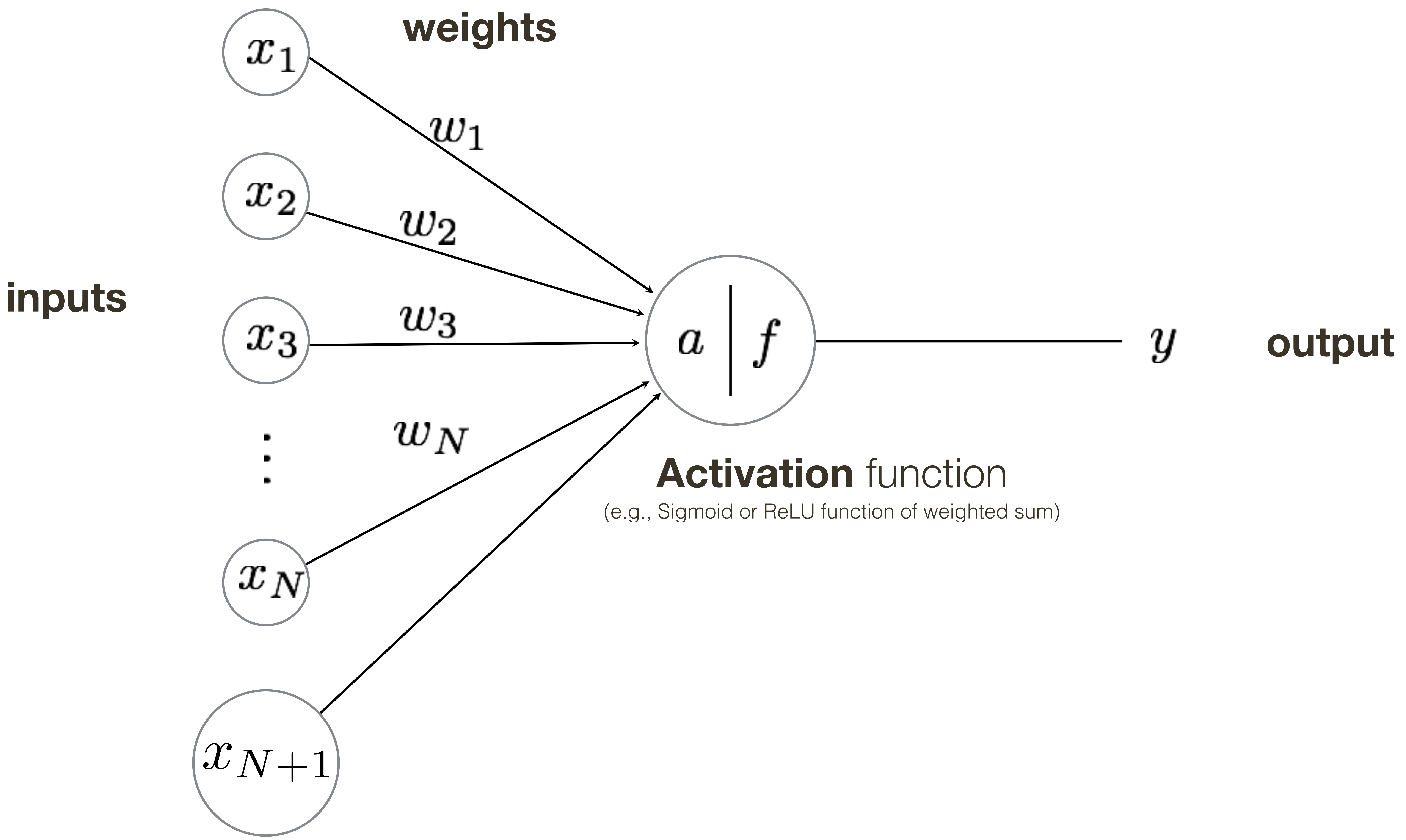
Found to accelerate convergence during learning

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
$\vdots$
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 …

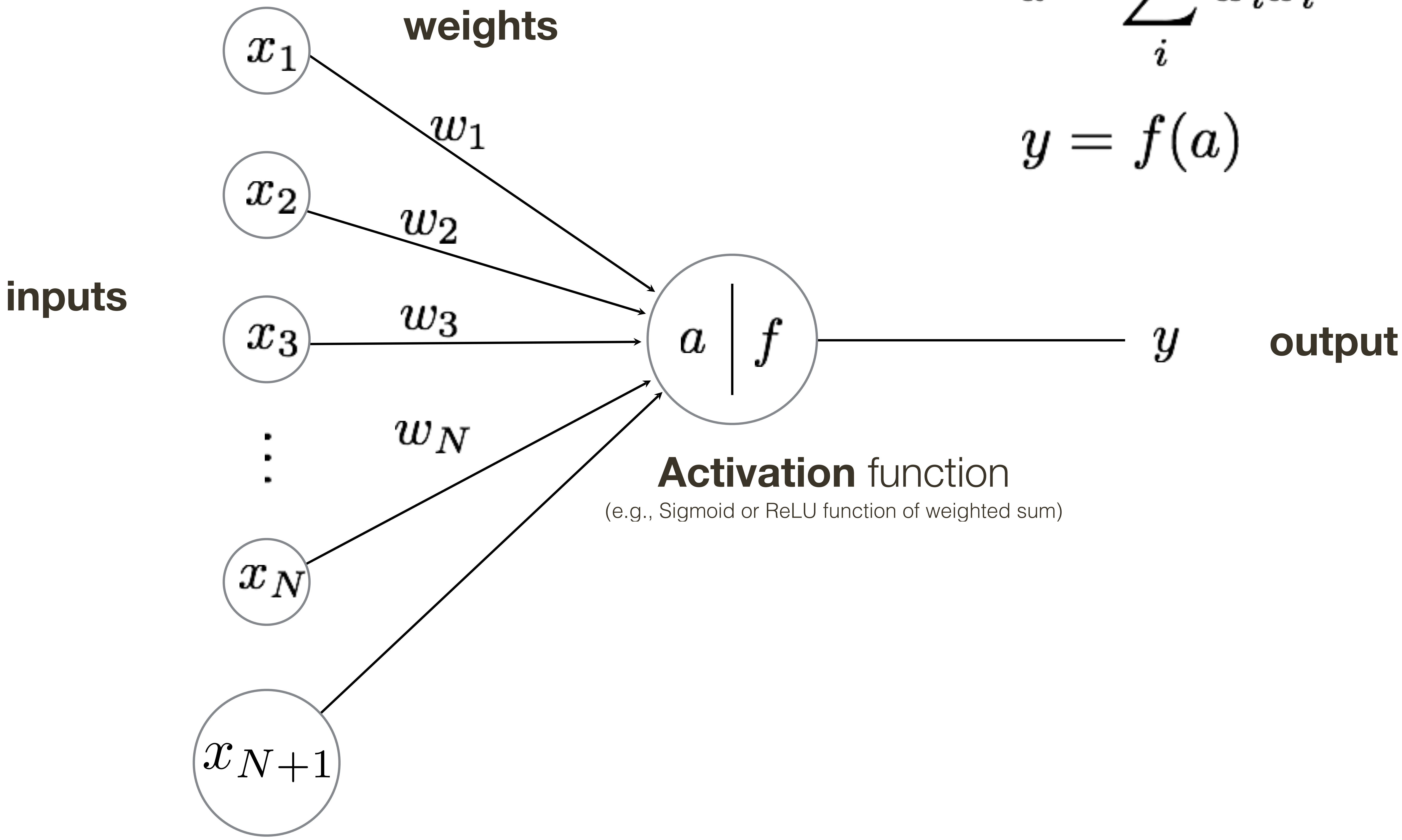


A **Neuron** … another way to draw it …

(1) Combine the sum and activation function

$$a = \sum_i w_i x_i$$

$$y = f(a)$$



Activation function

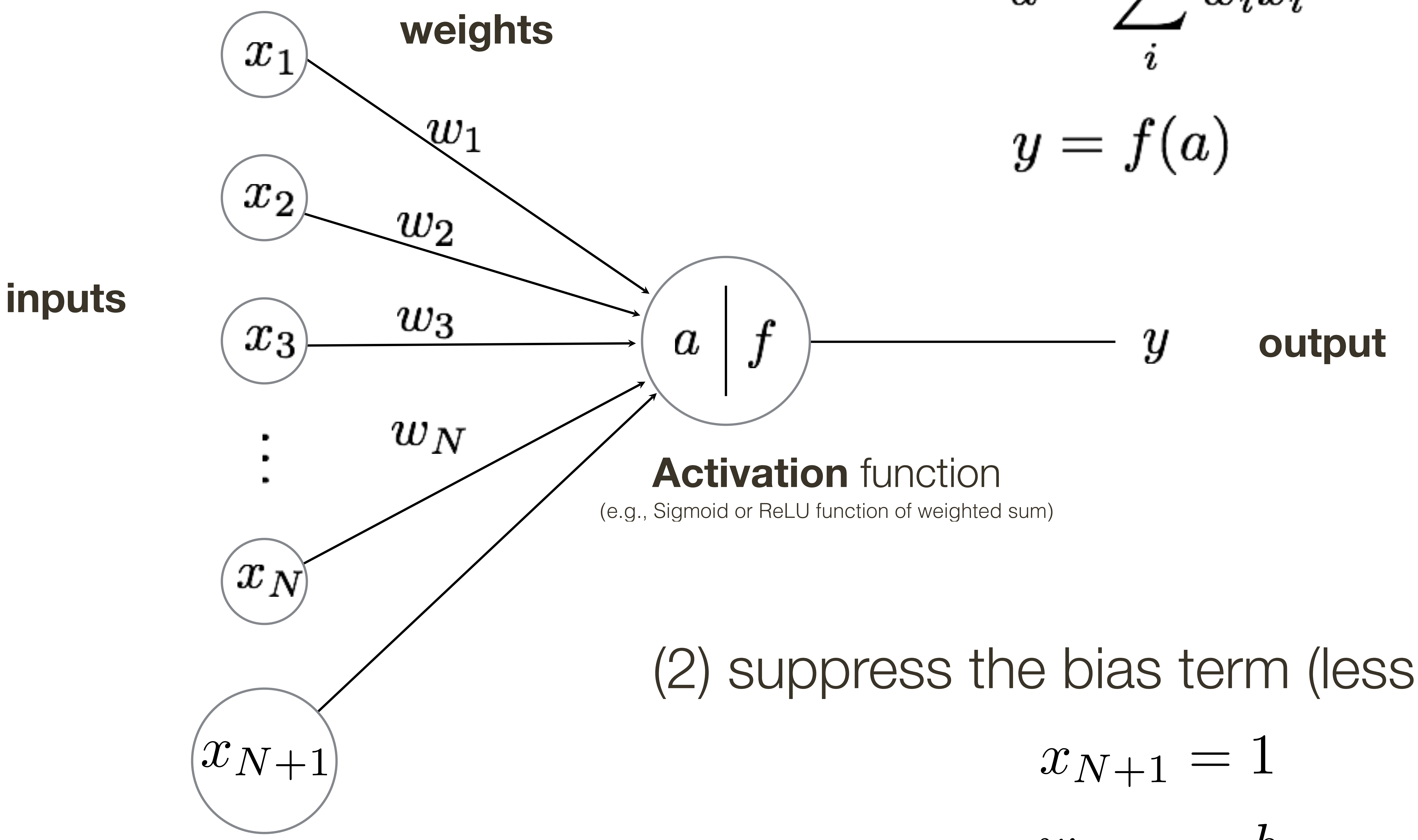
(e.g., Sigmoid or ReLU function of weighted sum)

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)

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

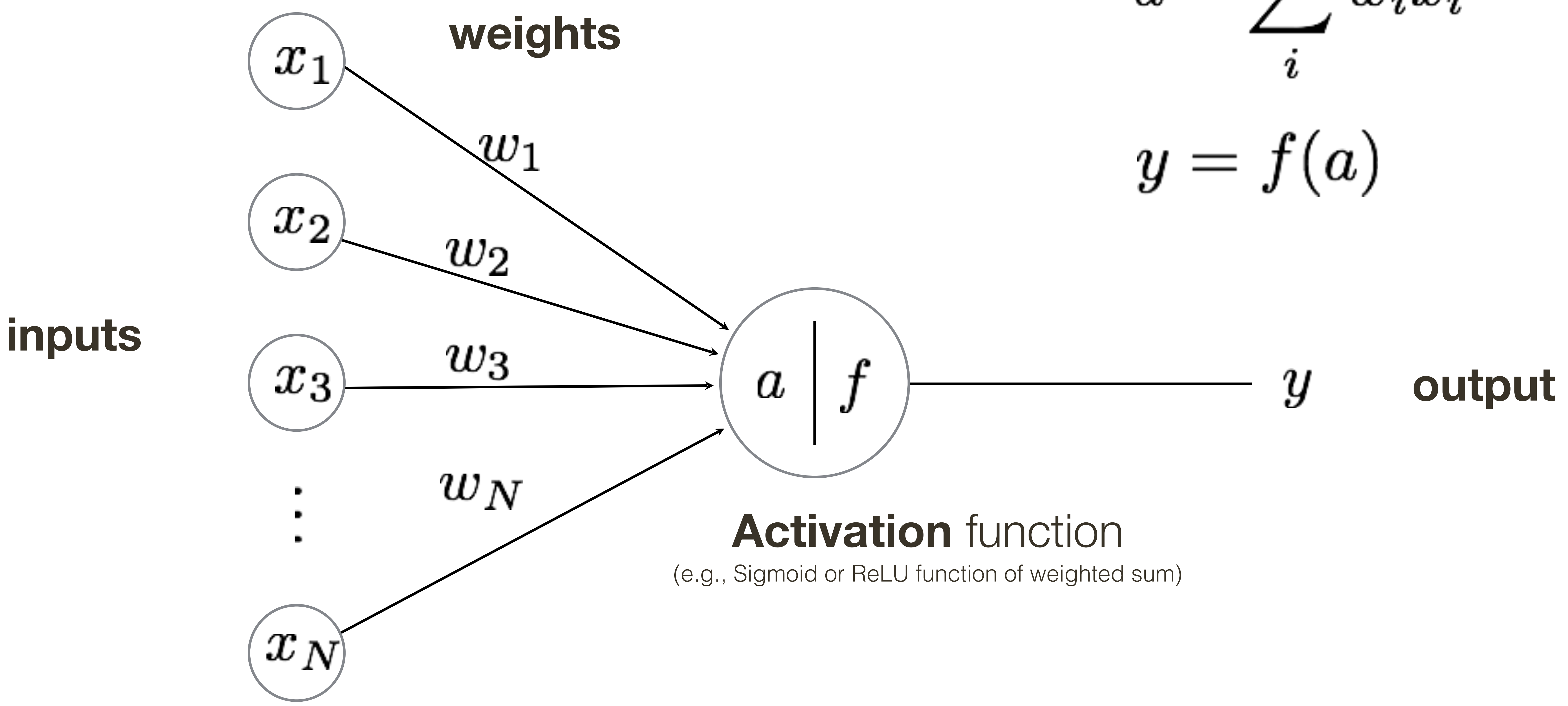
$$w_{N+1} = b$$

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

(1) Combine the sum and activation function

$$a = \sum_i w_i x_i$$

$$y = f(a)$$



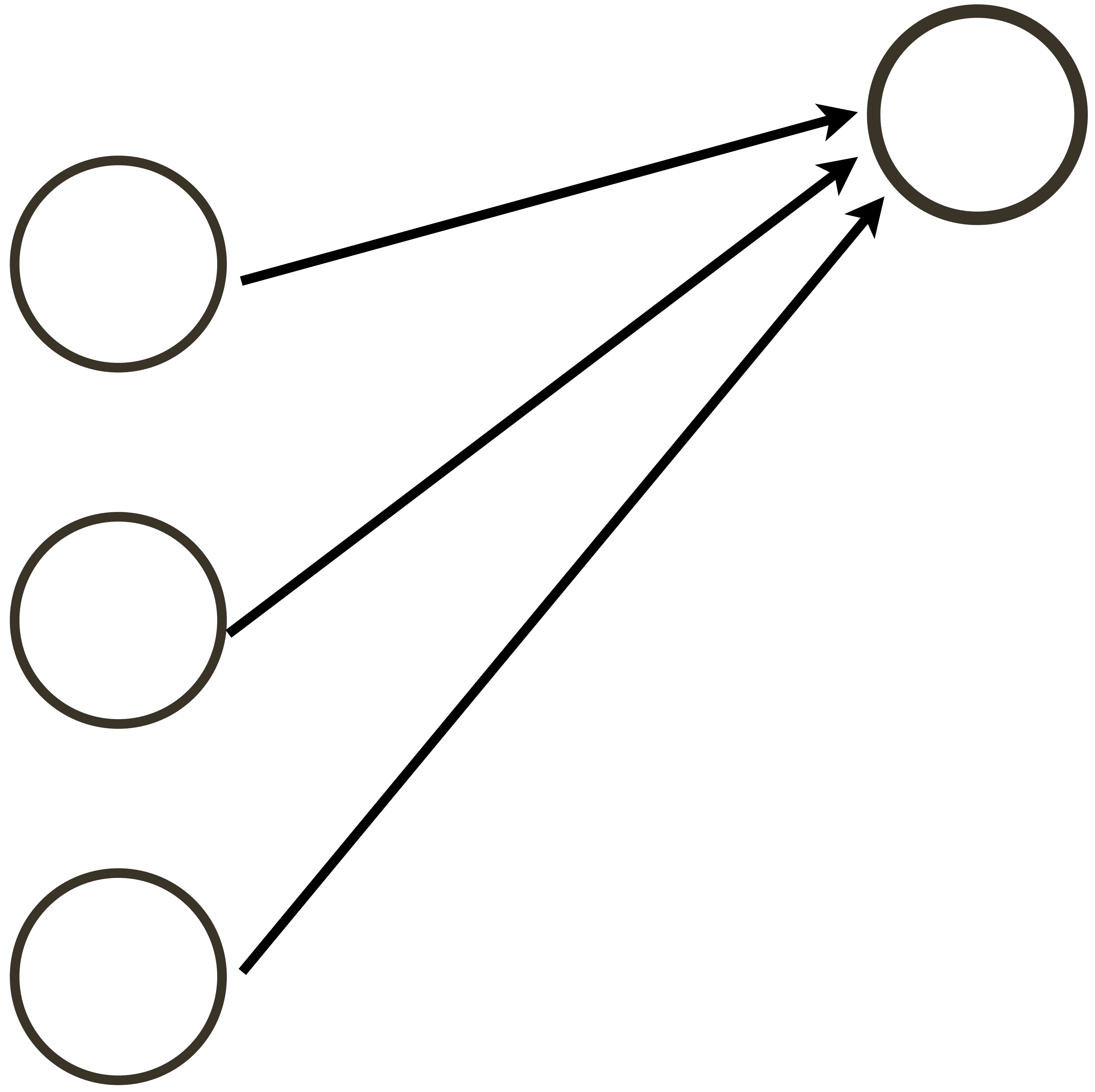
(2) suppress the bias term (less clutter)

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

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

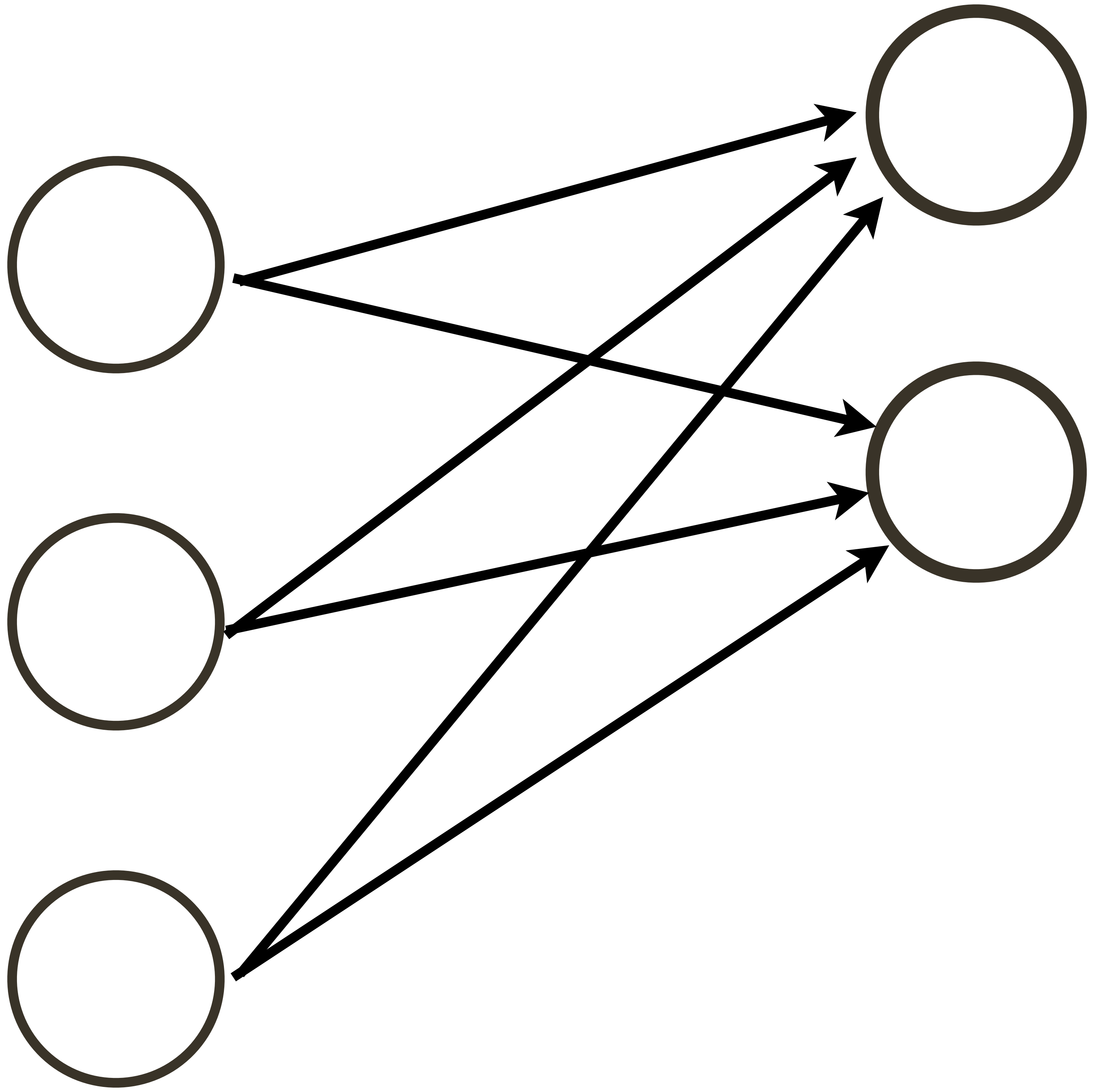
Neural Network

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



Neural Network

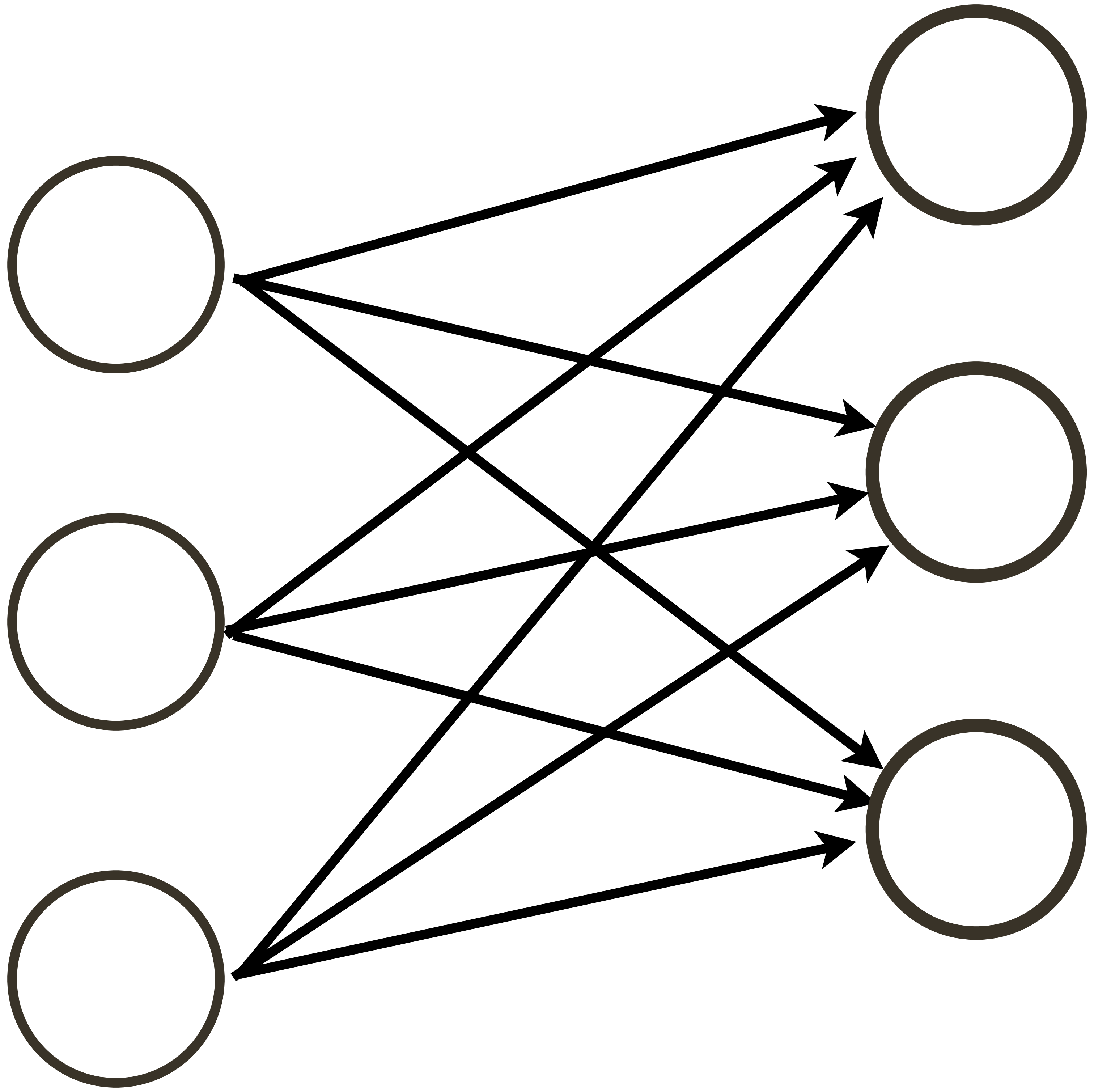
Connect a bunch of neurons together — a collection of connected neurons



‘two neurons’

Neural Network

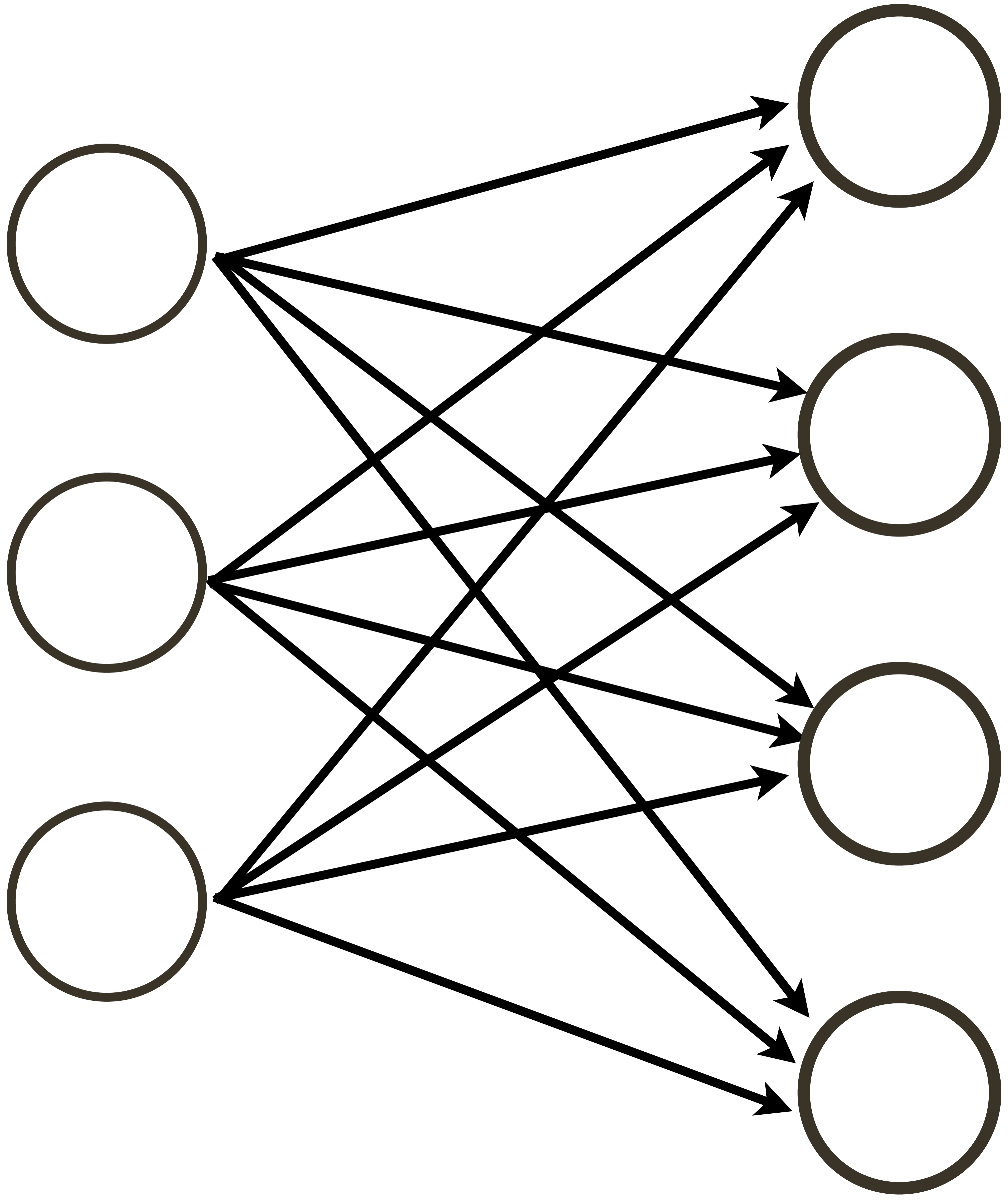
Connect a bunch of neurons together — a collection of connected neurons



‘three neurons’

Neural Network

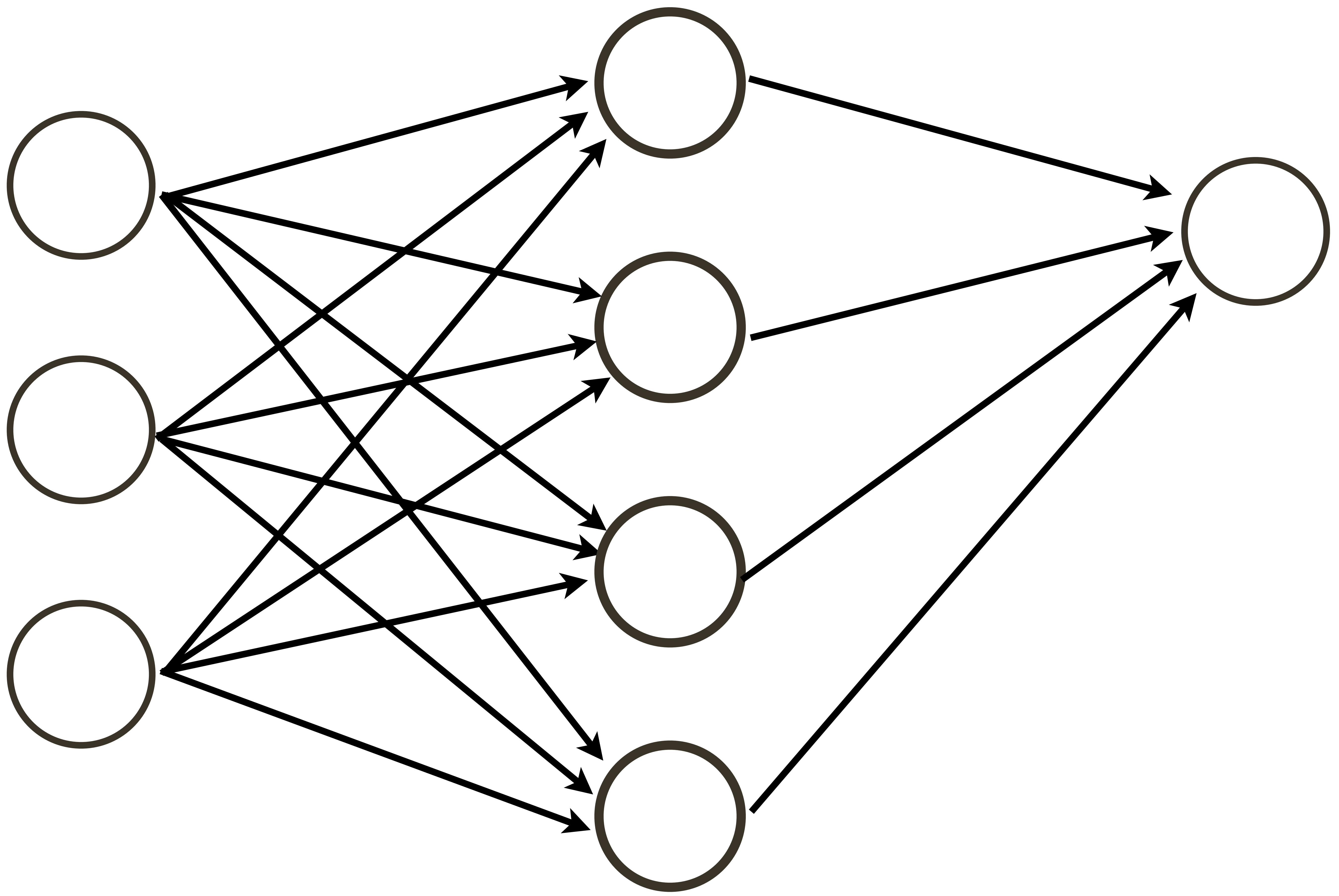
Connect a bunch of neurons together — a collection of connected neurons



‘four neurons’

Neural Network

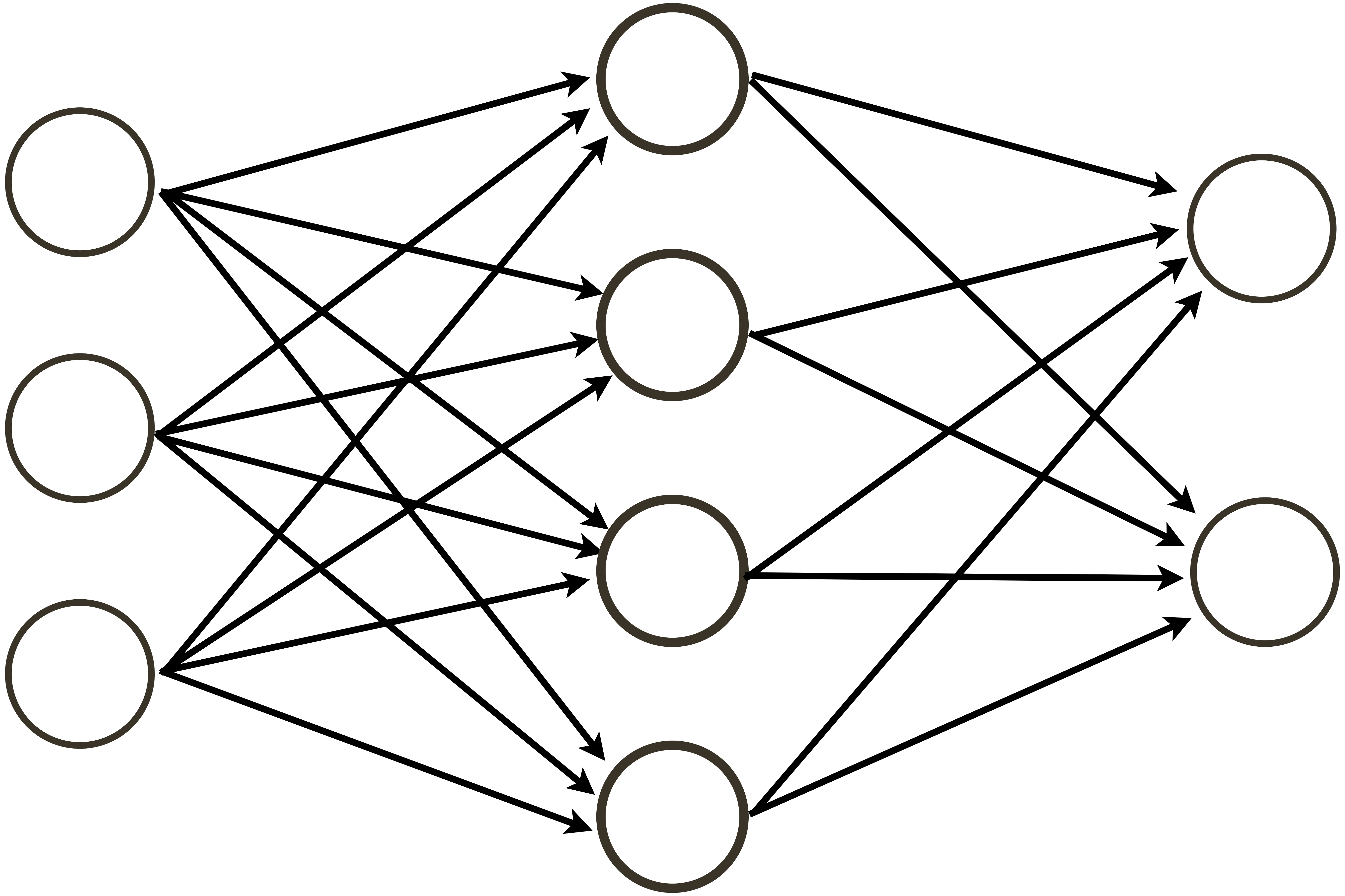
Connect a bunch of neurons together — a collection of connected neurons



‘five neurons’

Neural Network

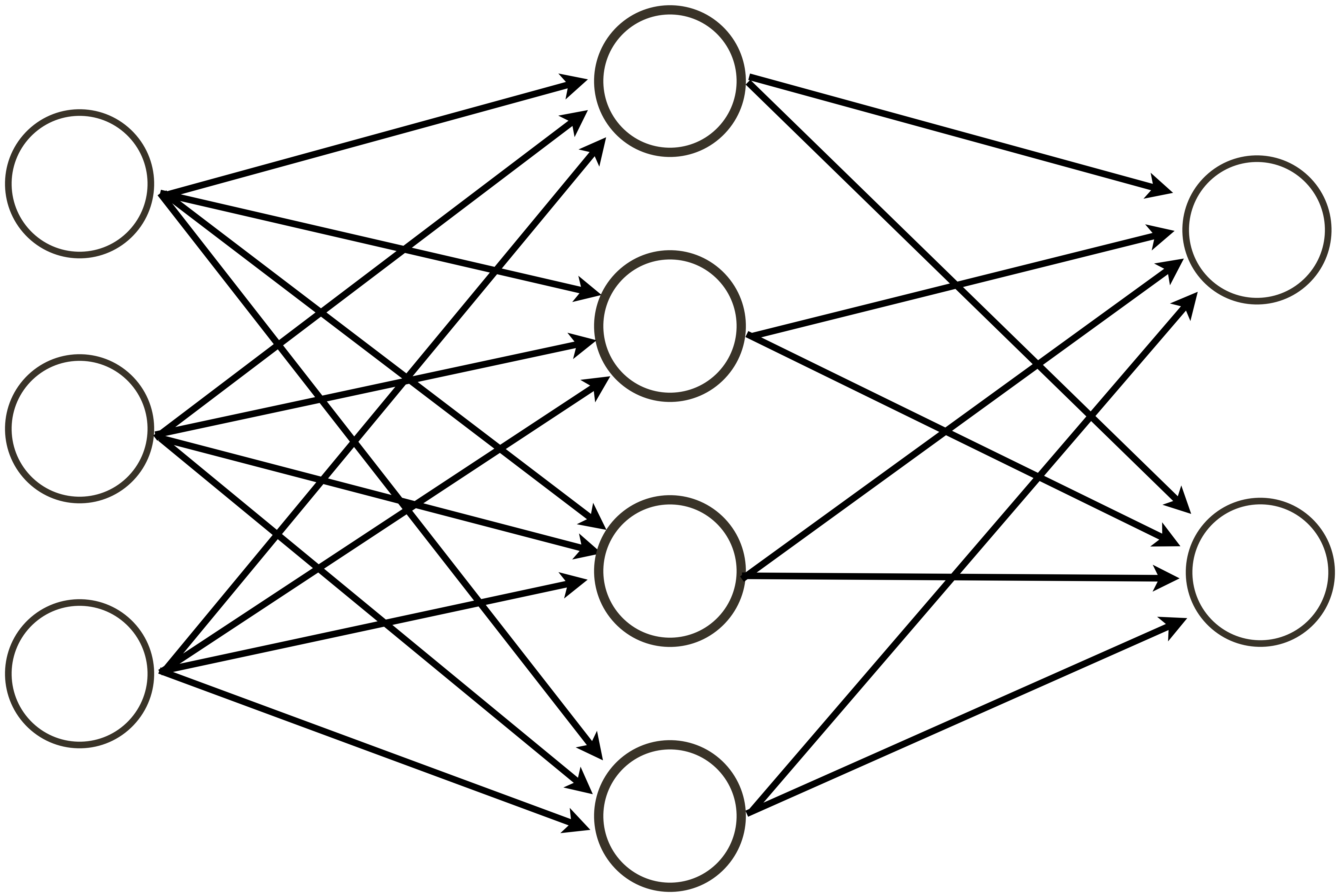
Connect a bunch of neurons together — a collection of connected neurons



'six neurons'

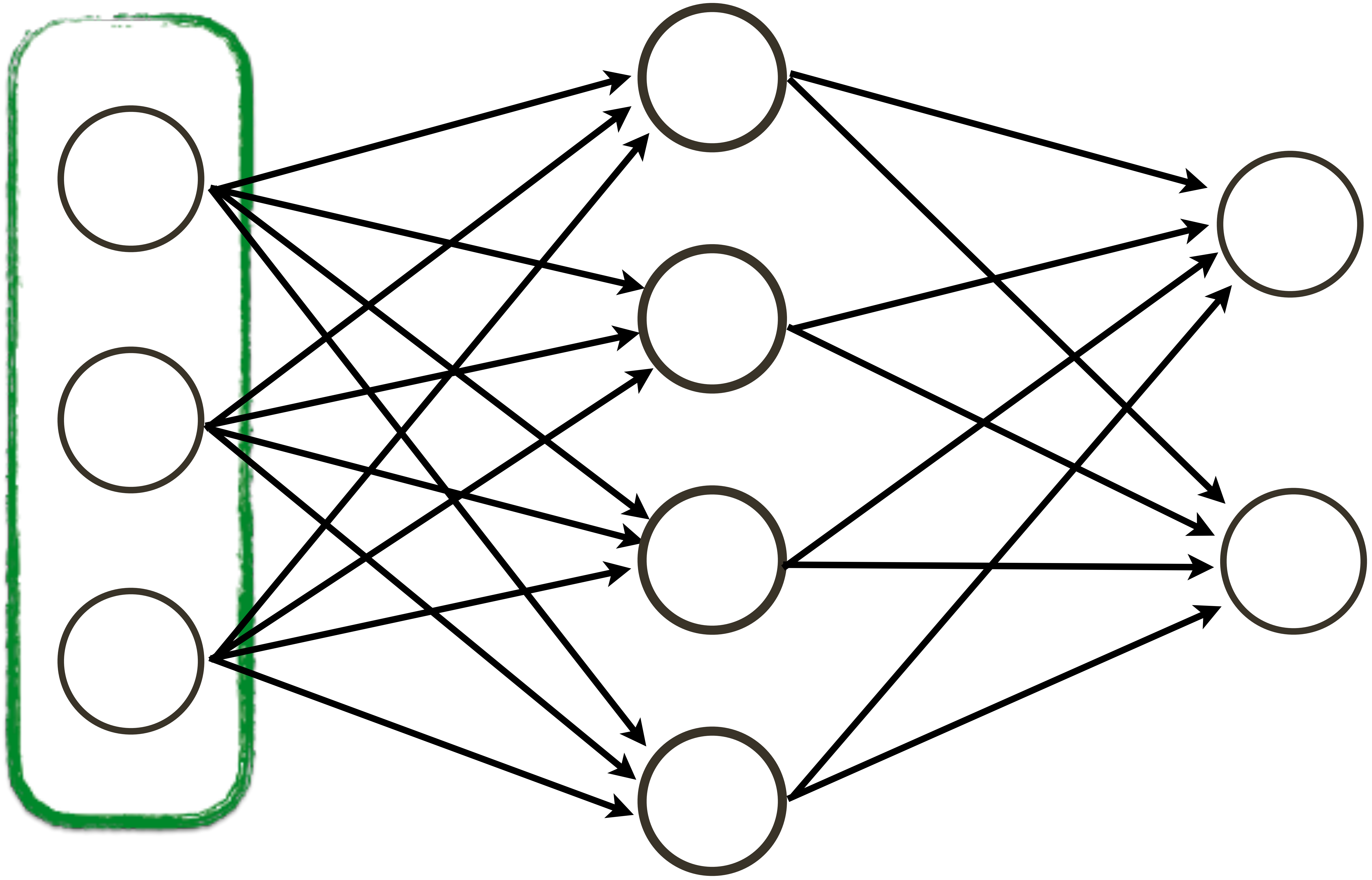
Neural Network

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

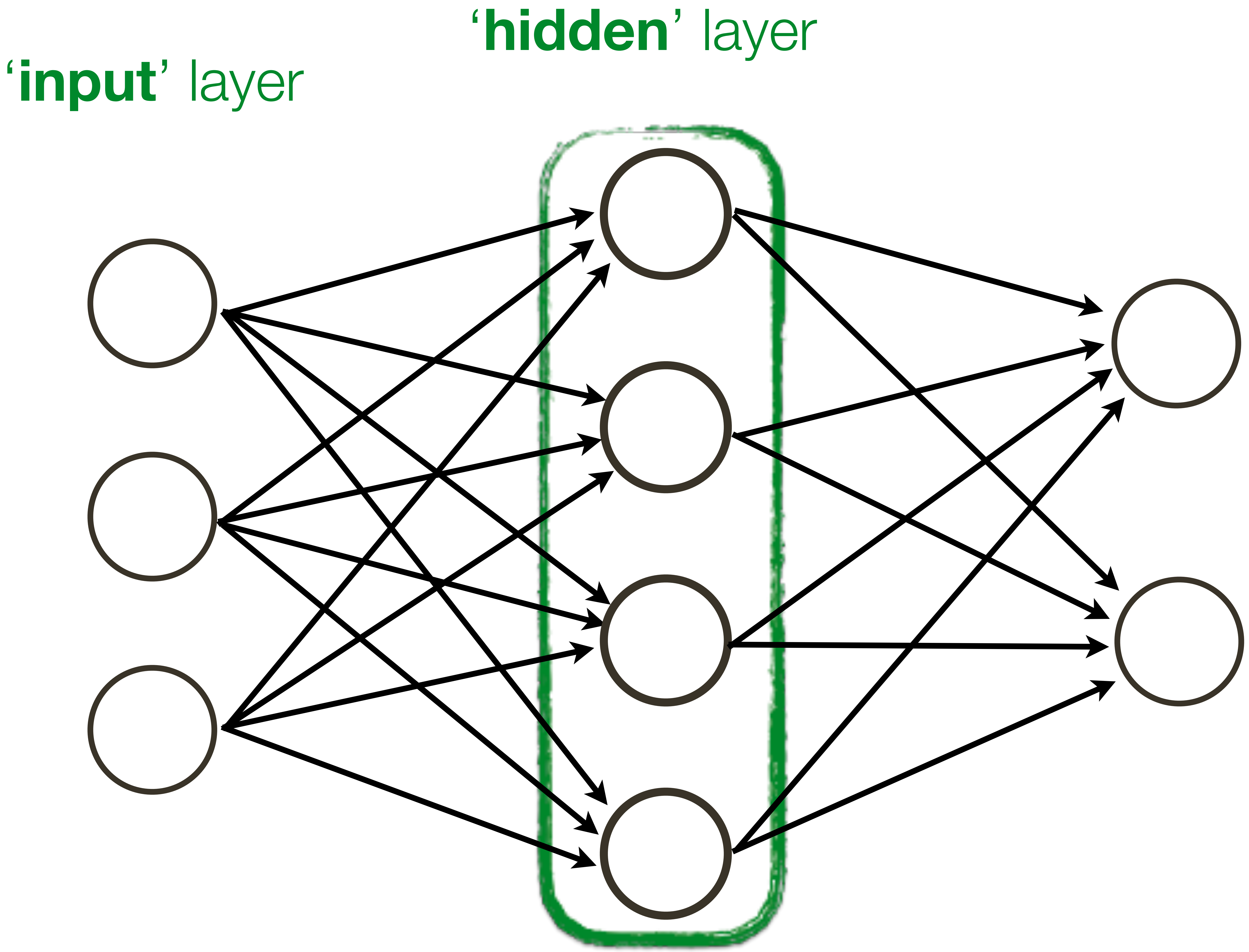


Neural Network: **Terminology**

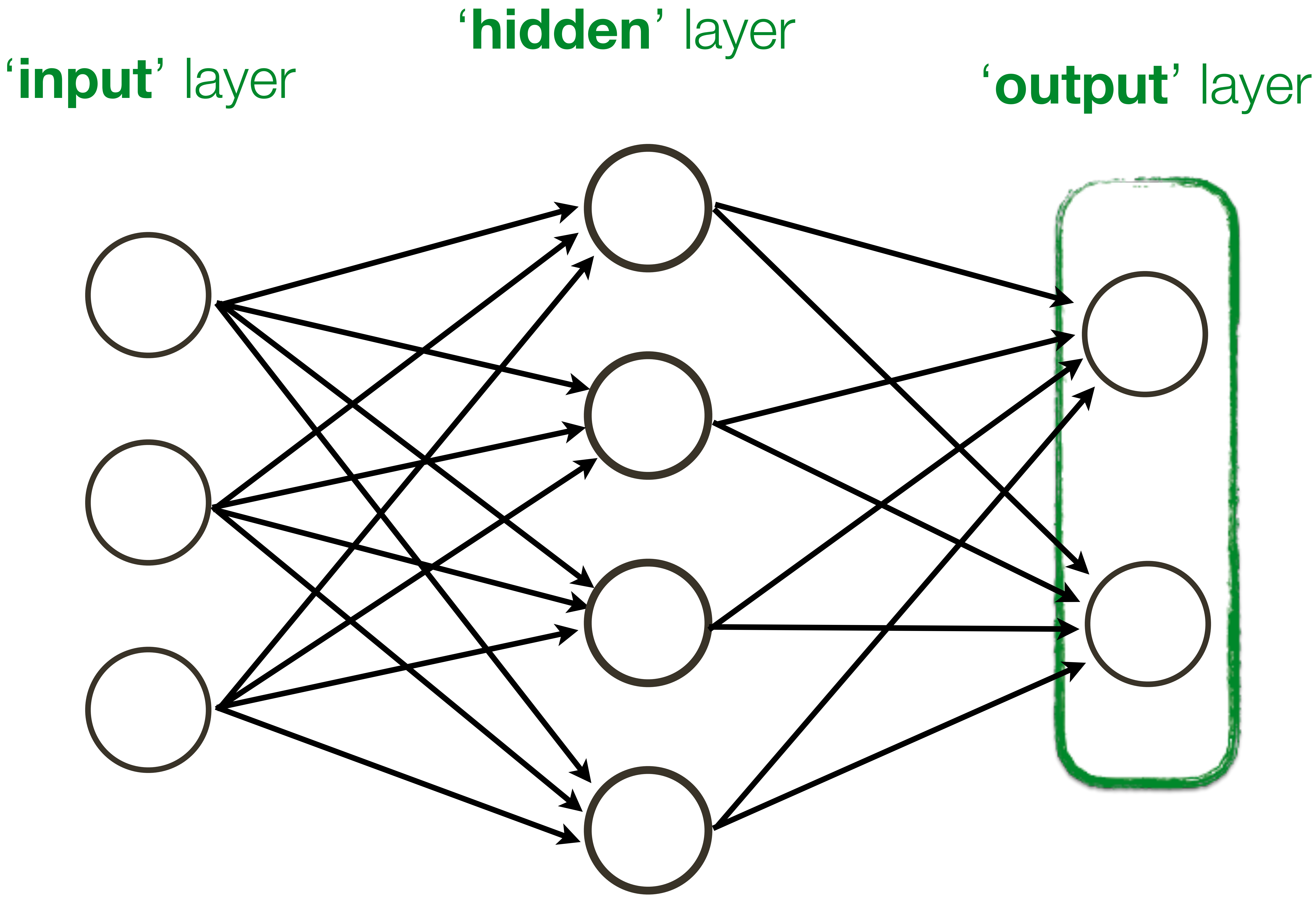
‘**input**’ layer



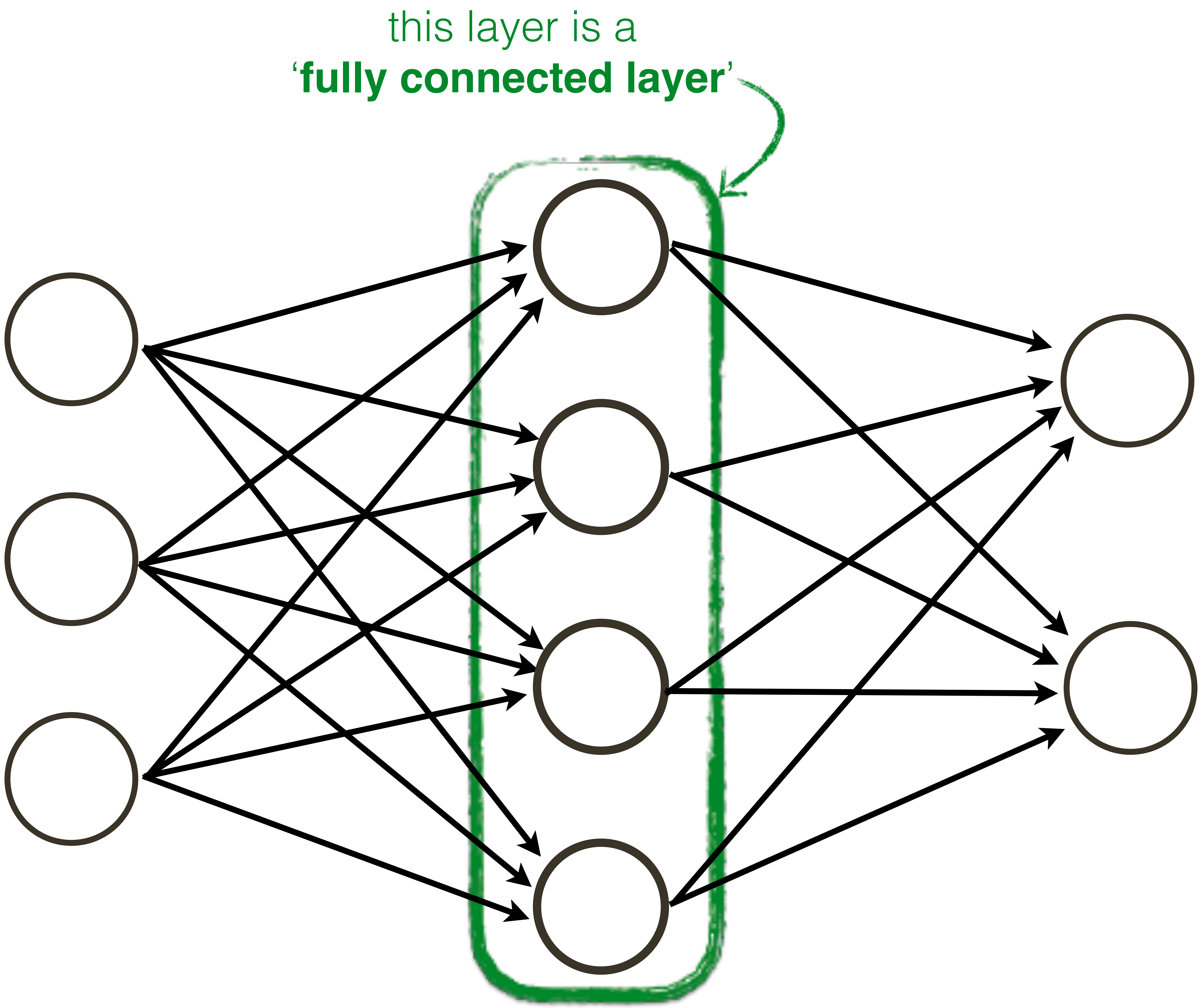
Neural Network: **Terminology**



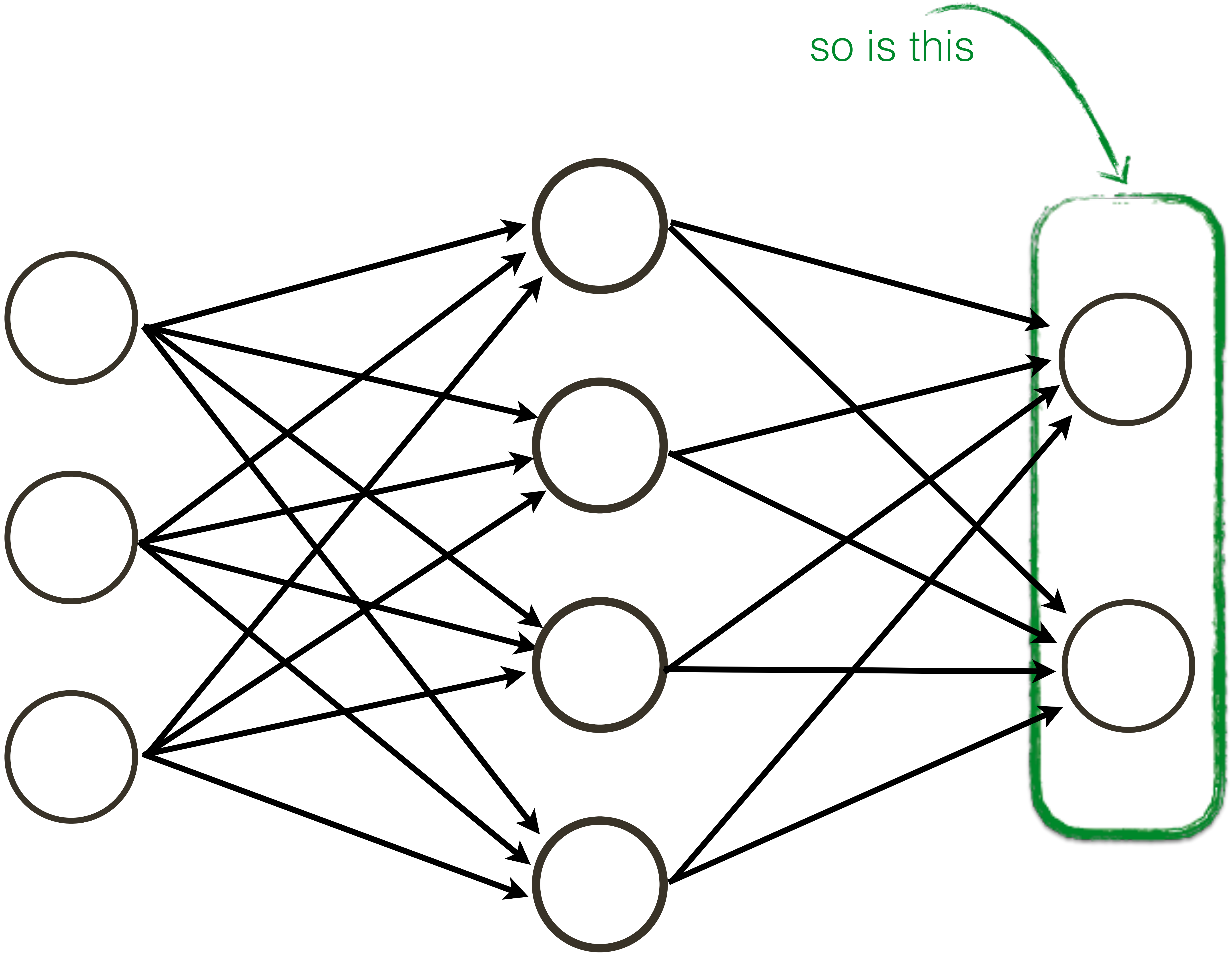
Neural Network: **Terminology**



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

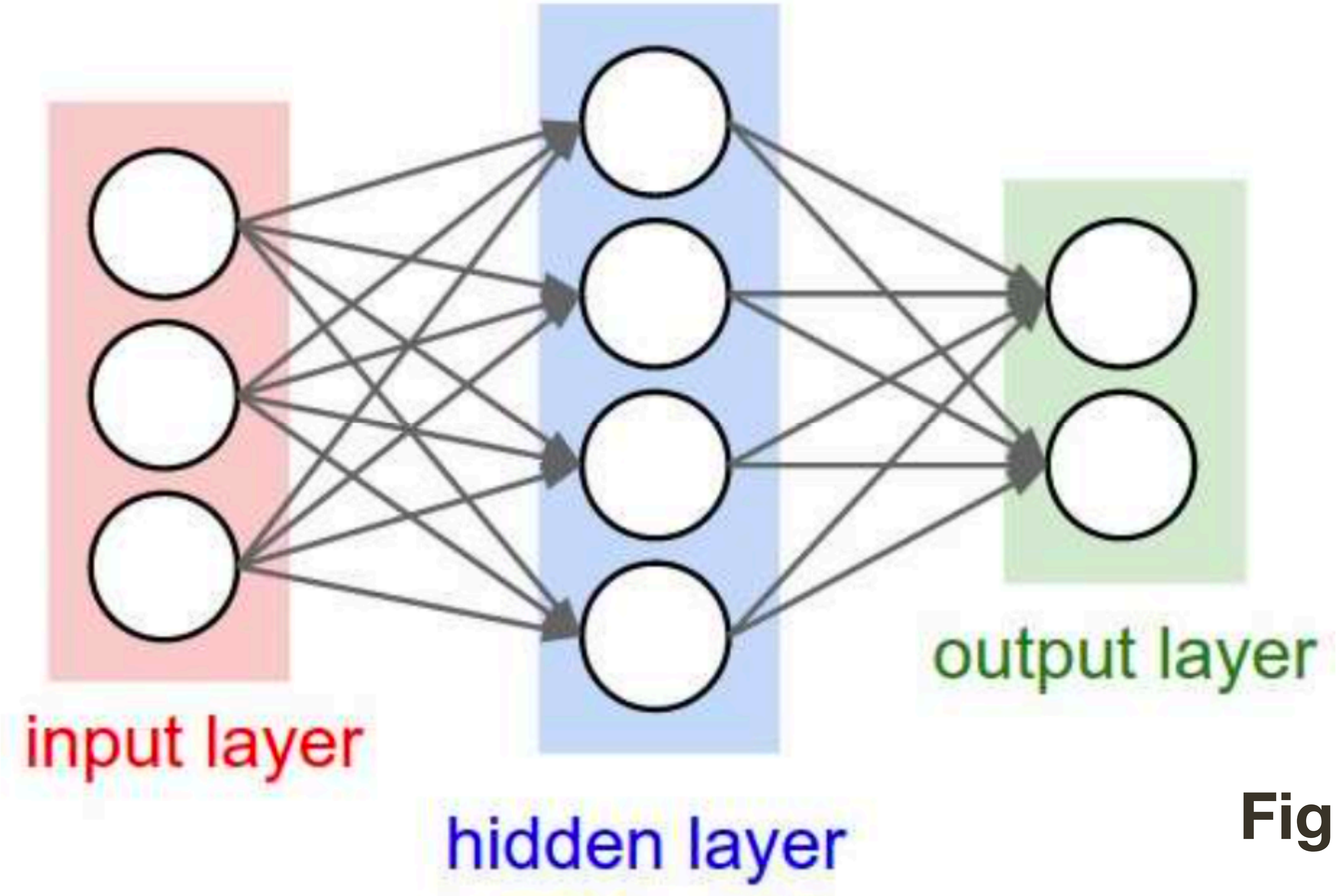


Figure credit: Fei-Fei and Karpathy

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

Neural Network **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?

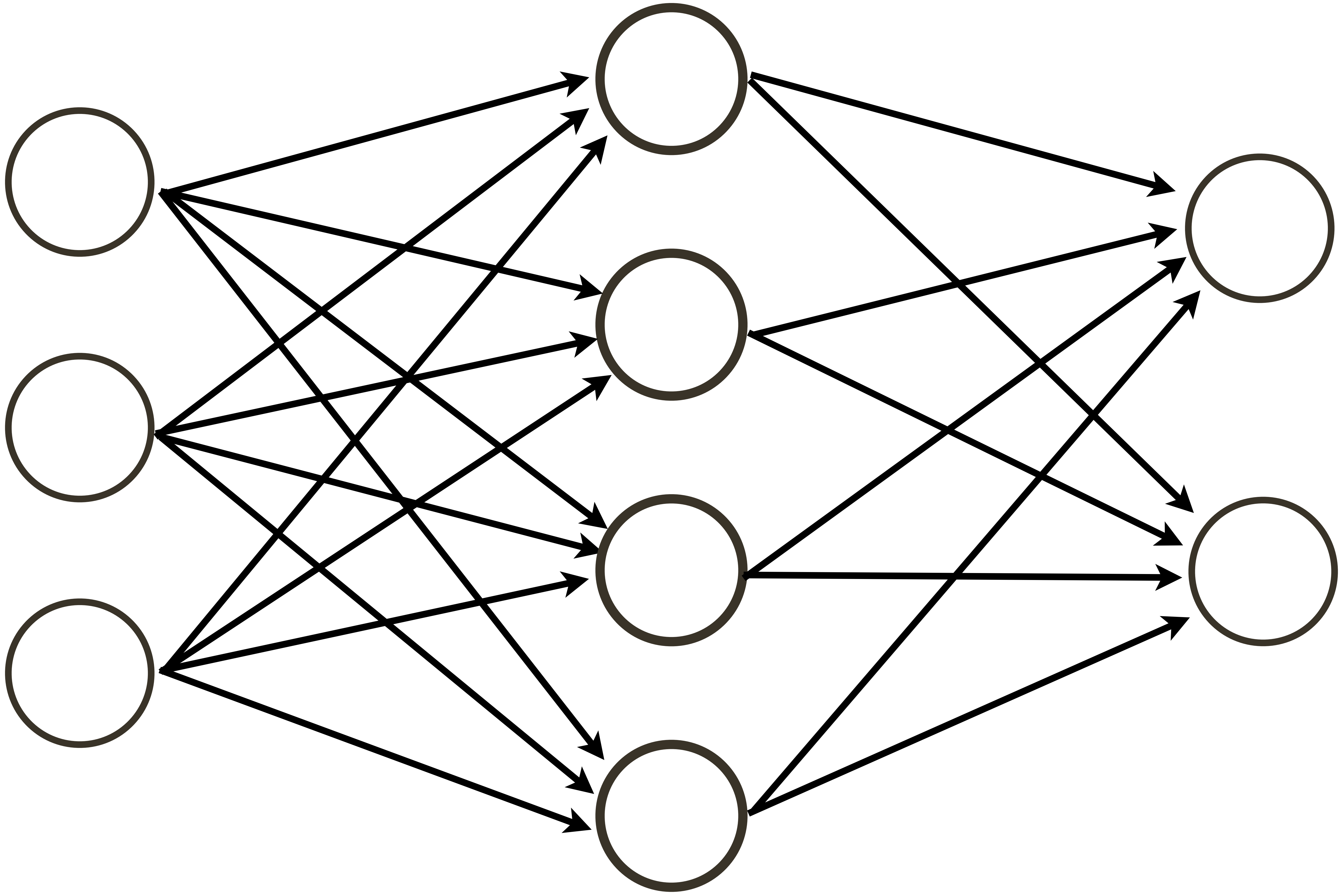
Answer: 1) More layers = more complex functional mapping

2) More efficient due to distributed representation

* slide from Marc'Aurelio Renzato

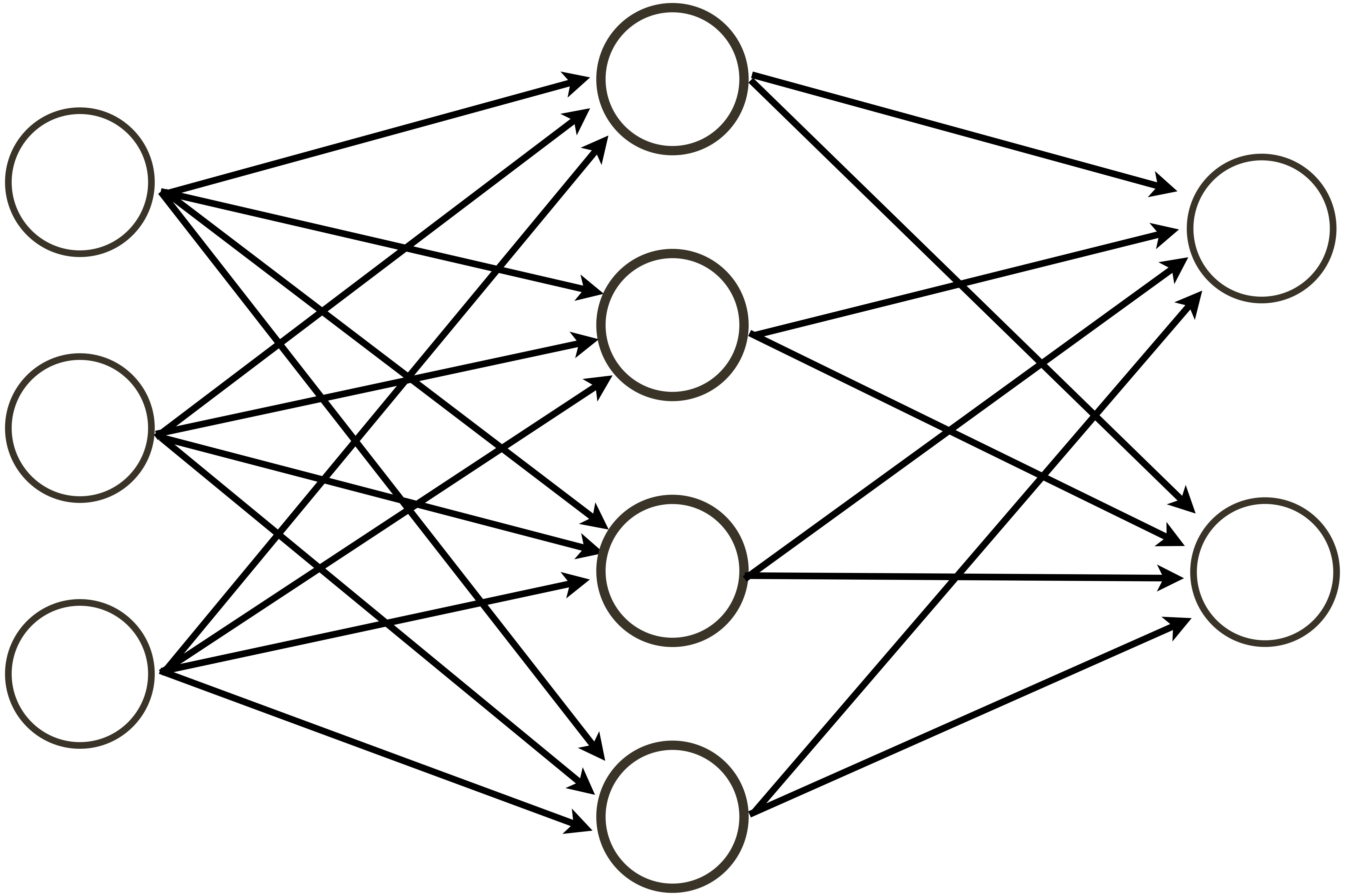
Activation Function

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



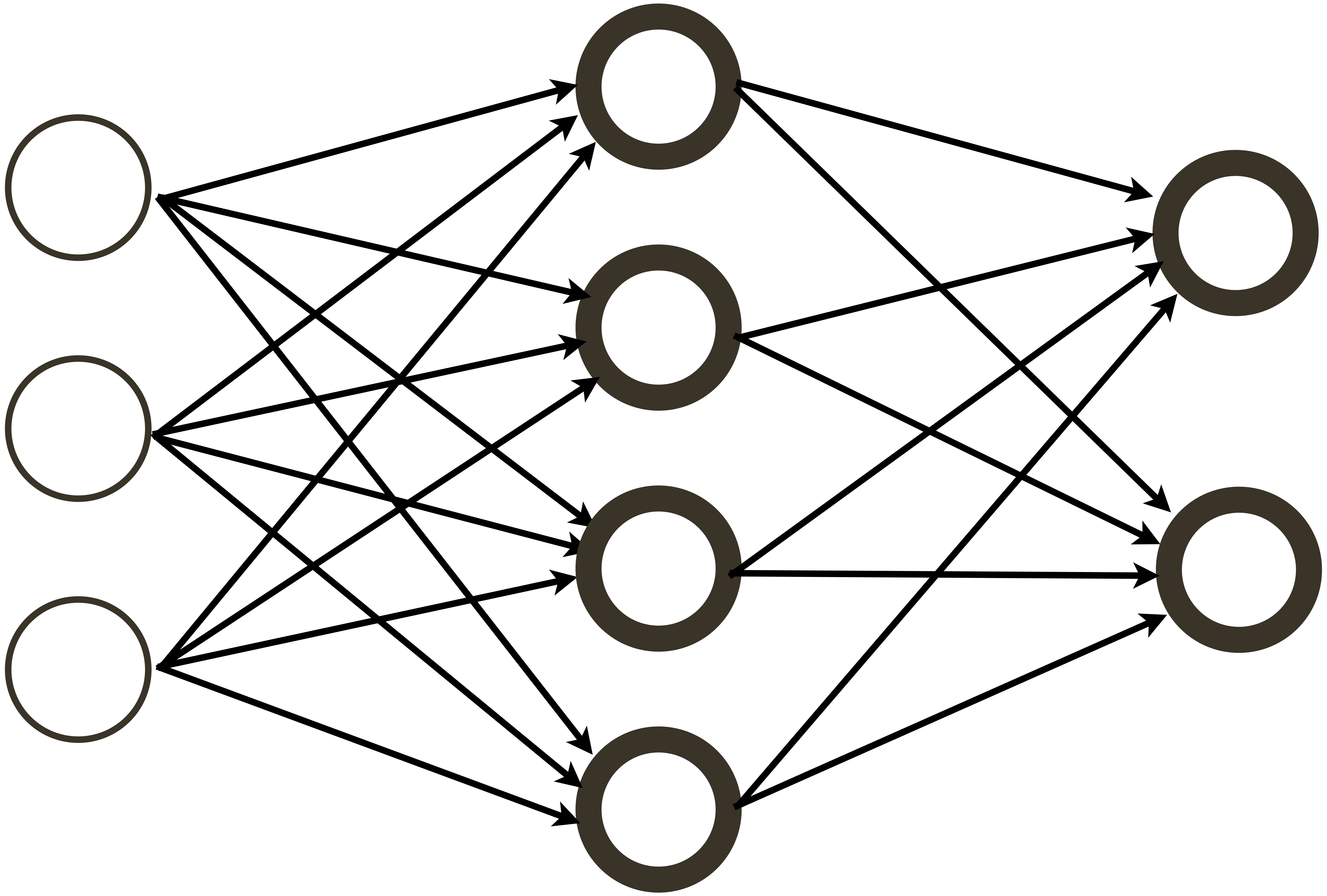
Neural Network

How many neurons?



Neural Network

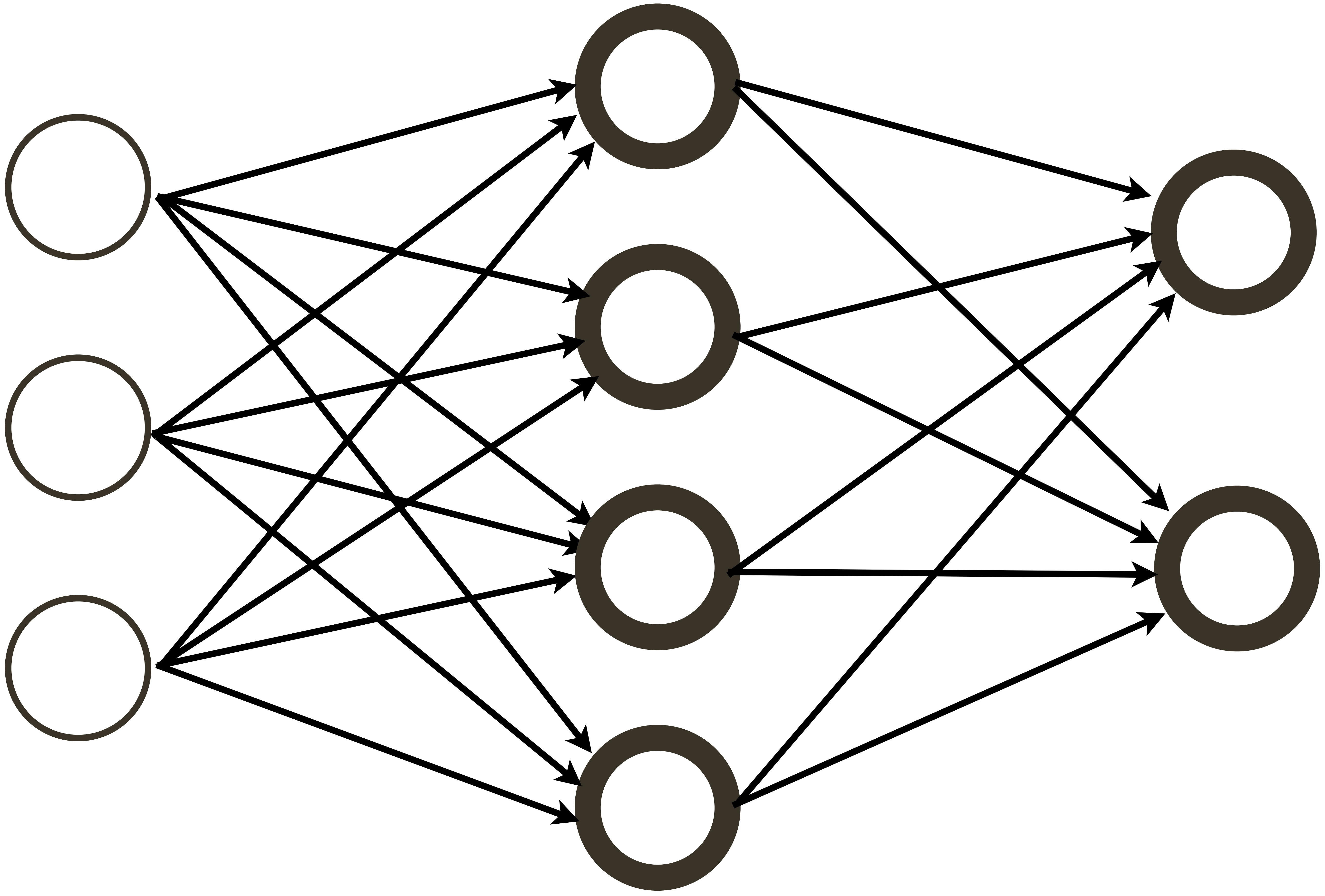
How many neurons? 4+2 = 6



Neural Network

How many neurons? 4+2 = 6

How many weights?

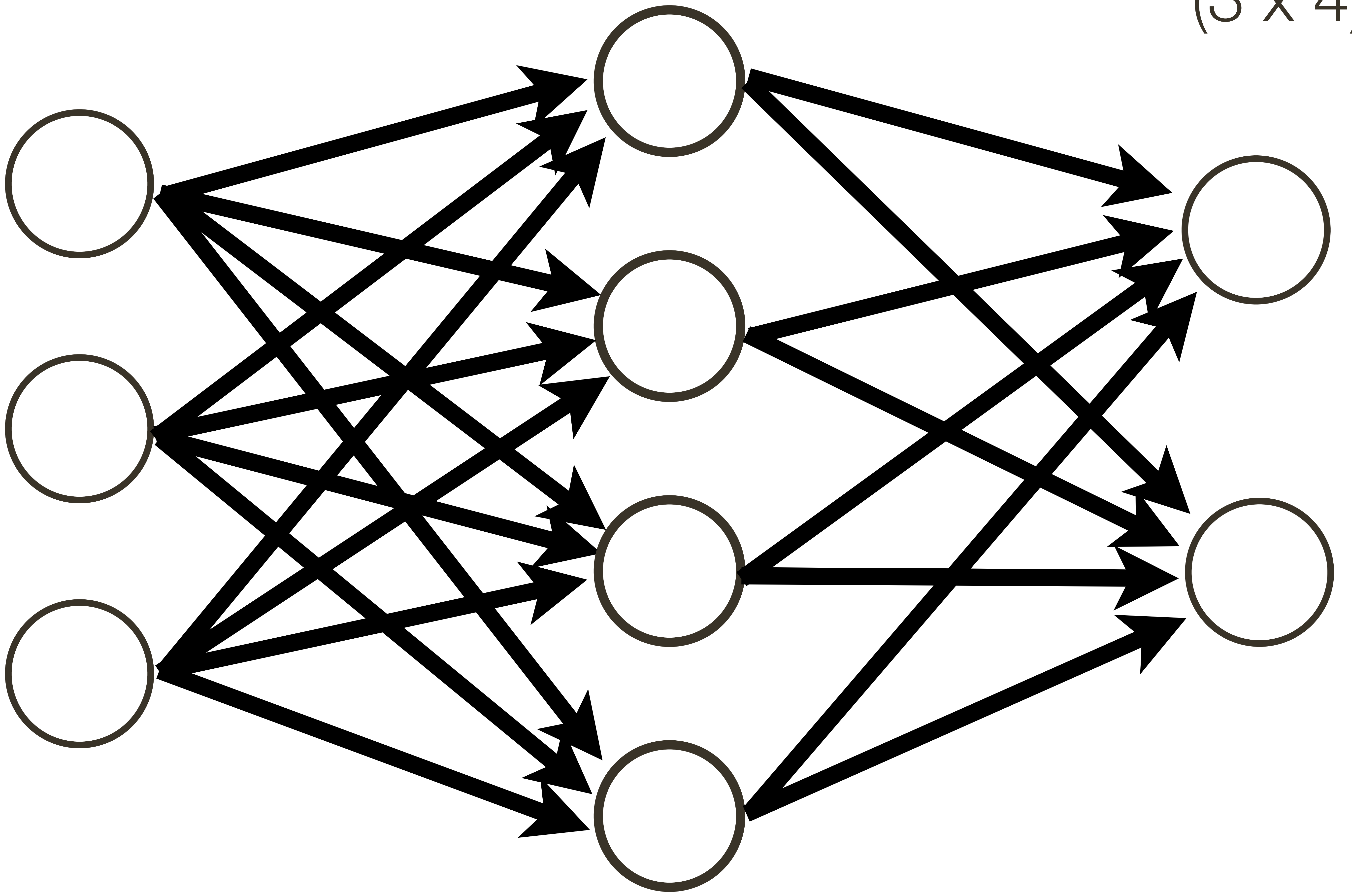


Neural Network

How many neurons? 4+2 = 6

How many weights?

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

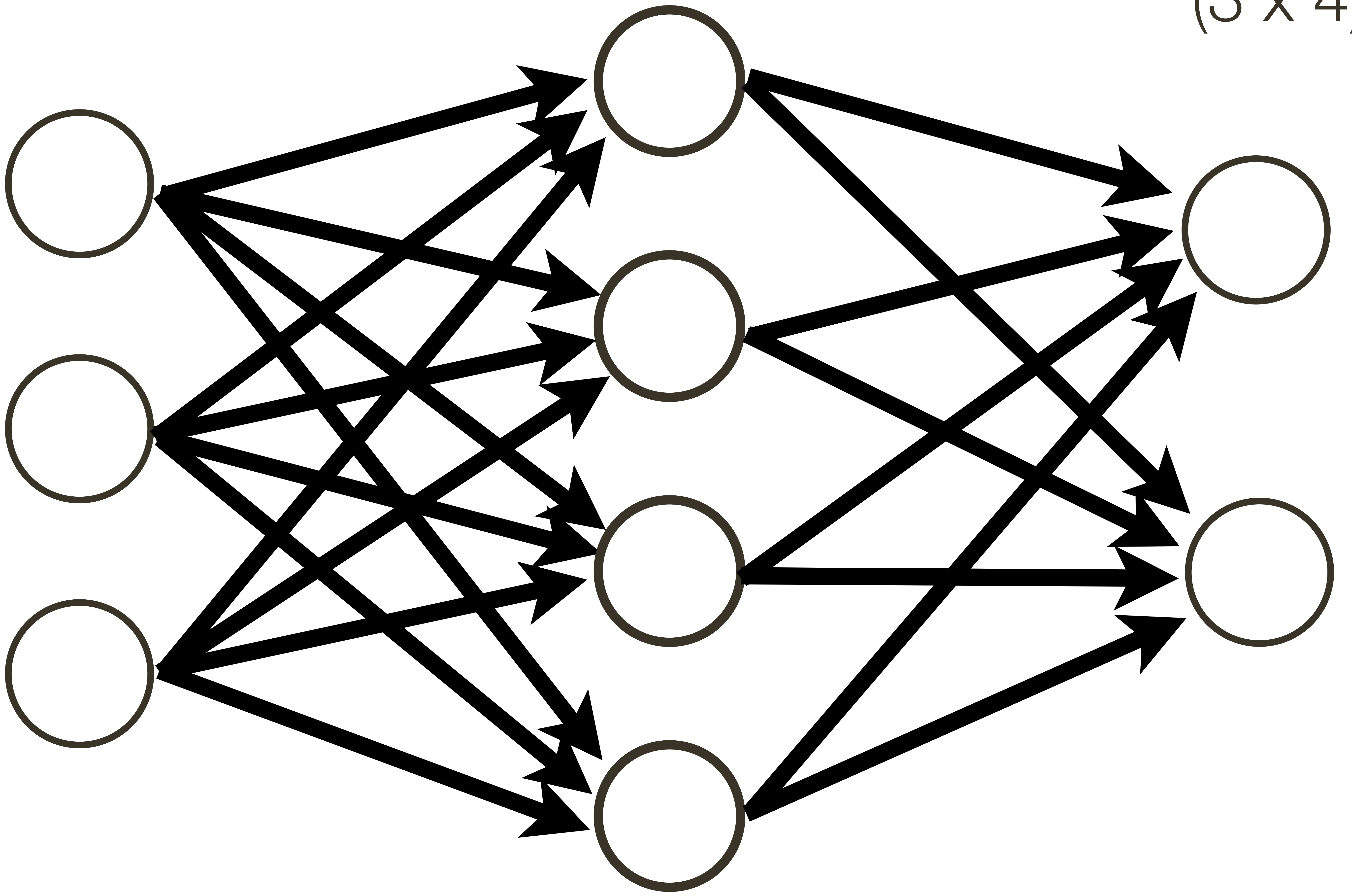


Neural Network

How many neurons? 4+2 = 6

How many weights?

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



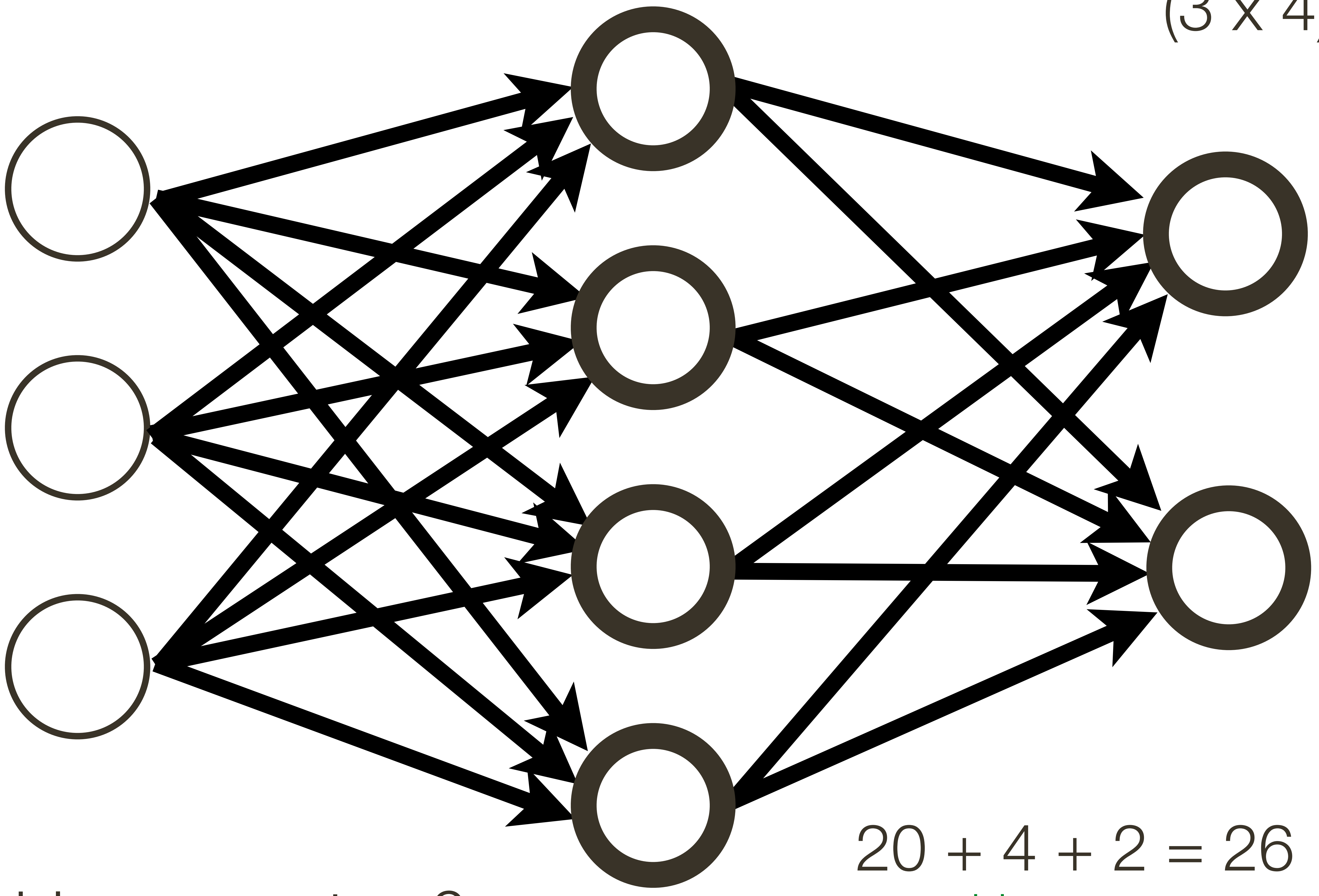
How many learnable parameters?

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

Backpropagation

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

— e.g. cross-entropy loss: $$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

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

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

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

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:

$$\begin{array}{lcl} \quad f & & \\ c_1 = -2.85 & & 0.058 \\ c_2 = 0.86 & \xrightarrow{\text{exp}} & 2.36 \\ c_3 = 0.28 & & 1.32 \end{array}$$

Backpropagation

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

— e.g. cross-entropy loss: $$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

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:

$$\begin{array}{lcccc} f & & & & \\ c_1 = -2.85 & & 0.058 & \text{Normalize to sum to 1} & 0.016 \\ c_2 = 0.86 & \xrightarrow{\exp} & 2.36 & \longrightarrow & 0.631 \\ c_3 = 0.28 & & 1.32 & & 0.353 \end{array}$$

Backpropagation

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

— e.g. cross-entropy loss: $$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

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:

probability of a class

$$\begin{array}{c} f \\ c_1 = -2.85 \\ c_2 = 0.86 \\ c_3 = 0.28 \end{array} \xrightarrow{\exp} \begin{array}{c} 0.058 \\ 2.36 \\ 1.32 \end{array} \xrightarrow{\text{Normalize to sum to 1}} \begin{array}{c} 0.016 \\ 0.631 \\ 0.353 \end{array}$$

Backpropagation

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

— e.g. cross-entropy loss: $L_i = -\log\left(\boxed{\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}}\right)$

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:

$$\begin{array}{c} f \\ c_1 = -2.85 \\ c_2 = 0.86 \\ c_3 = 0.28 \end{array} \xrightarrow{\exp} \begin{array}{c} 0.058 \\ 2.36 \\ 1.32 \end{array} \xrightarrow{\text{Normalize to sum to 1}} \begin{array}{c} \text{probability of a class} \\ 0.016 \\ 0.631 \\ 0.353 \end{array}$$

Backpropagation

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

— e.g. cross-entropy loss: $$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

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:

probability of a class

$$\begin{array}{c} f \\ c_1 = -2.85 \\ c_2 = 0.86 \\ c_3 = 0.28 \end{array} \xrightarrow{\text{exp}} \begin{array}{c} 0.058 \\ 2.36 \\ 1.32 \end{array} \xrightarrow{\text{Normalize to sum to 1}} \begin{array}{c} 0.016 \\ 0.631 \\ 0.353 \end{array} \quad L_i = -\log(0.353) = 1.04$$

Backpropagation

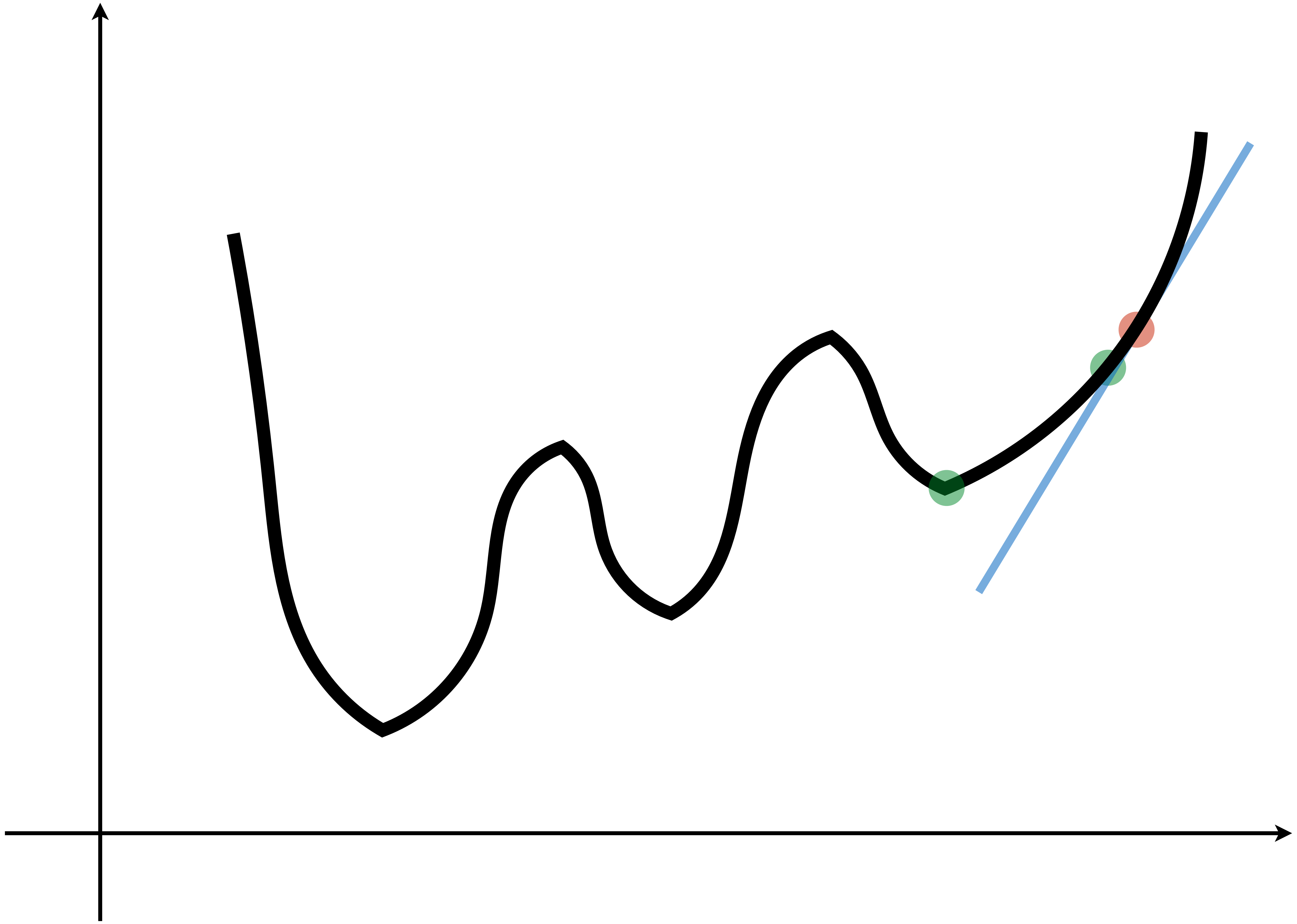
When training a neural network, the final output will be some loss (error) function

— e.g. cross-entropy loss: $$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

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



λ - is the learning rate

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

Backpropagation

The parameters of a neural network are learned using **backpropagation**, which computes gradients via recursive application of the **chain rule** from calculus

Backpropagation

The parameters of a neural network are learned using **backpropagation**, which computes gradients via recursive application of the **chain rule** from calculus

Suppose $f(x, y) = xy$. What is the partial derivative of f with respect to x? What is the partial derivative of f with respect to y?

Backpropagation

The parameters of a neural network are learned using **backpropagation**, which computes gradients via recursive application of the **chain rule** from calculus

Suppose $f(x, y) = xy$. What is the partial derivative of f with respect to x? What is the partial derivative of f with respect to y?

$$\frac{\partial f}{\partial x} = y \qquad\qquad \frac{\partial f}{\partial y} = x$$

Backpropagation

Suppose $f(x, y) = x + y$. What is the partial derivative of f with respect to x? What is the partial derivative of f with respect to y?

Backpropagation

Suppose $f(x, y) = x + y$. What is the partial derivative of f with respect to x? What is the partial derivative of f with respect to y?

$$\frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$

Backpropagation

A trickier example: $f(x, y) = \max(x, y)$

Backpropagation

A trickier example: $f(x, y) = \max(x, y)$

$$\frac{\partial f}{\partial x} = \mathbf{1}(x \geq y) \qquad \frac{\partial f}{\partial y} = \mathbf{1}(y \geq x)$$

That is, the (sub)gradient is 1 on the input that is larger, and 0 on the other input

— For example, say x = 4, y = 2. Increasing y by a tiny amount does not change the value of f (f will still be 4), hence the gradient on y is zero.

Backpropagation

We can compose more complicated functions and compute their gradients by applying the **chain rule** from calculus

Backpropagation

We can compose more complicated functions and compute their gradients by applying the **chain rule** from calculus

Suppose $f(x, y, z) = (x + y)z$. What are the partial derivatives of f with respect to x? y? z?

Backpropagation

We can compose more complicated functions and compute their gradients by applying the **chain rule** from calculus

Suppose $f(x, y, z) = (x + y)z$. What are the partial derivatives of f with respect to x? y? z?

For illustration we break this expression into $q = x + y$ and $f = qz$. This is a sum and a product, and we have just seen how to compute partial derivatives for these.

Backpropagation

We can compose more complicated functions and compute their gradients by applying the **chain rule** from calculus

Suppose $f(x, y, z) = (x + y)z$. What are the partial derivatives of f with respect to x? y? z?

For illustration we break this expression into $q = x + y$ and $f = qz$. This is a sum and a product, and we have just seen how to compute partial derivatives for these.

By the chain rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1 = z$$

Backpropagation

We can compose more complicated functions and compute their gradients by applying the **chain rule** from calculus

Suppose $f(x, y, z) = (x + y)z$. What are the partial derivatives of f with respect to x? y? z?

For illustration we break this expression into $q = x + y$ and $f = qz$. This is a sum and a product, and we have just seen how to compute partial derivatives for these.

By the chain rule

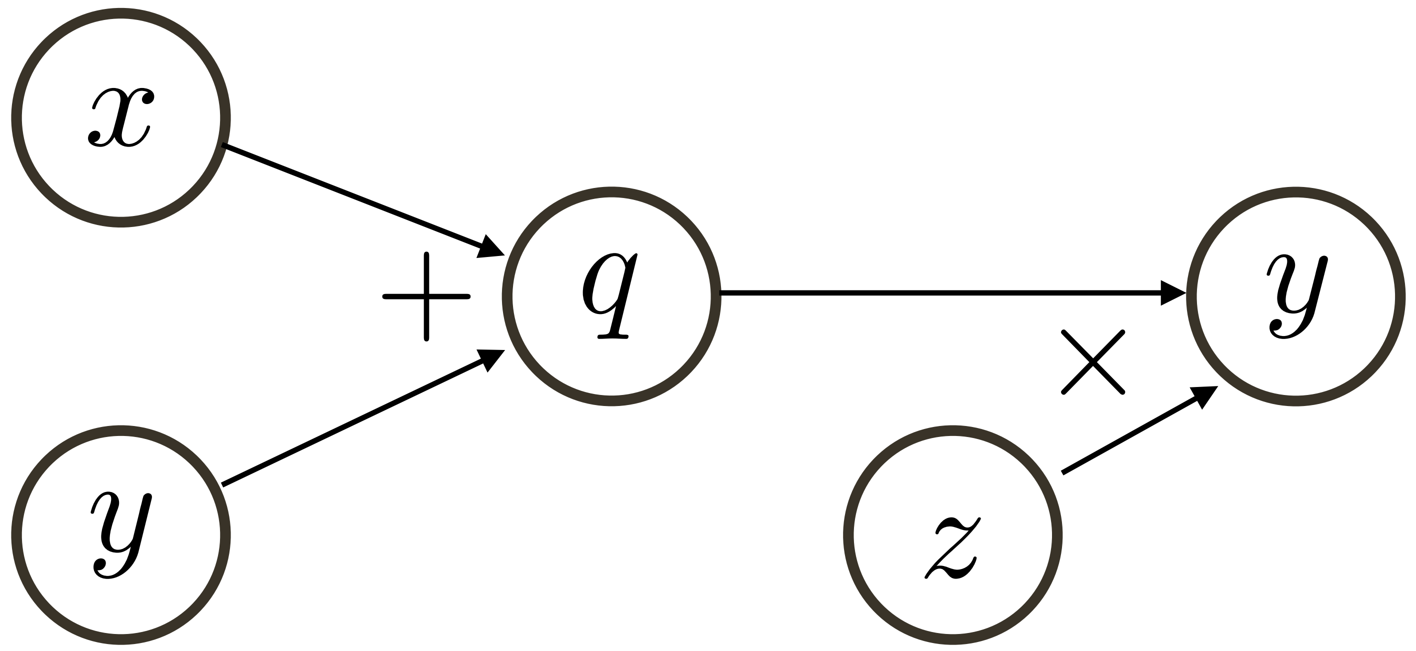
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = z \cdot 1 = z \qquad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = z \cdot 1 = z \qquad \frac{\partial f}{\partial z} = q$$

Backpropagation

$$f(x, y, z) = (x + y)z$$

Backpropagation

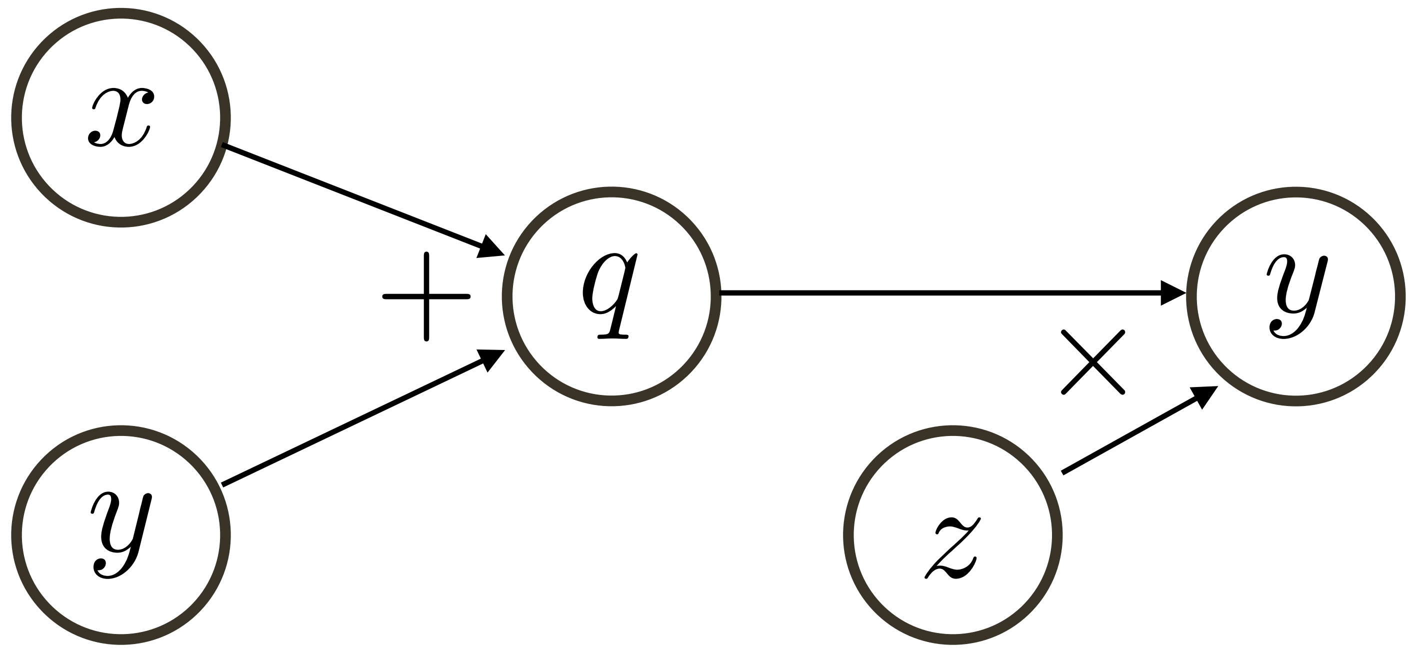
$$f(x, y, z) = (x + y)z$$



Computational graph (a DAG) with variable ordering from topological sort, where each **node** is an input, intermediate, or output variable

Backpropagation

$$f(x, y, z) = (x + y)z$$



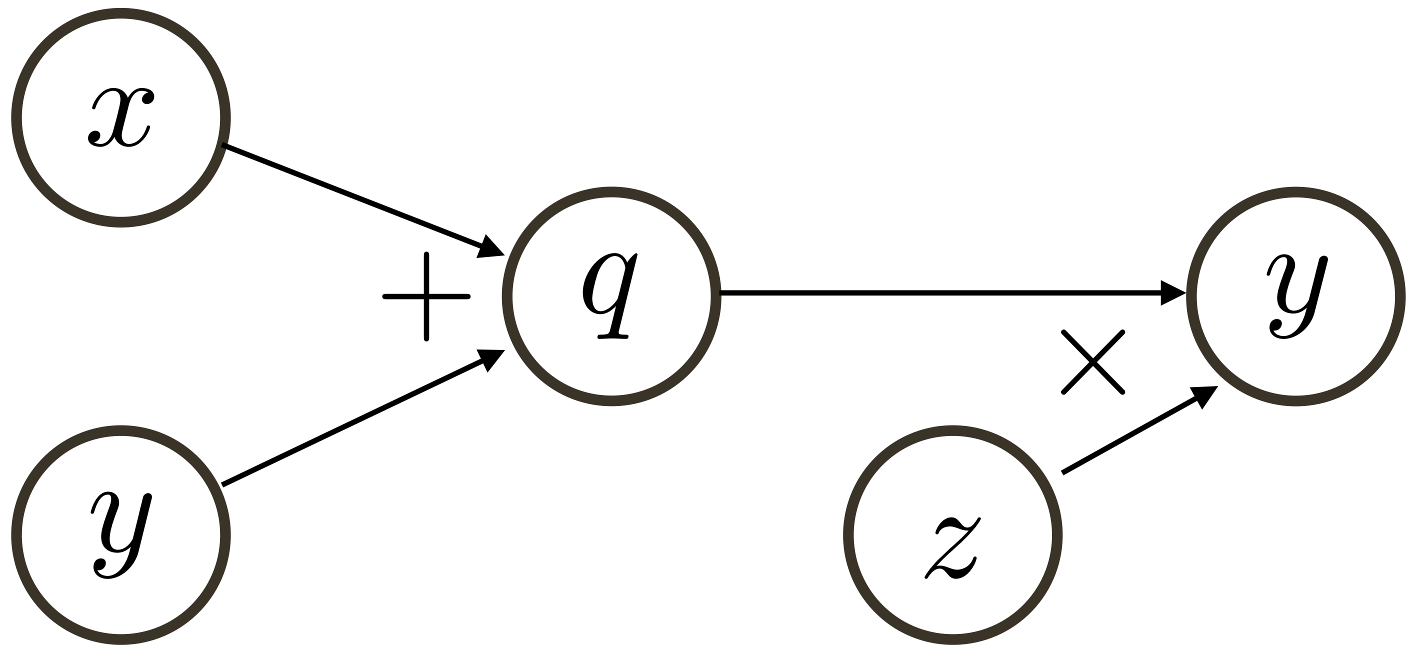
Computational graph (a DAG) with variable ordering from topological sort, where each **node** is an input, intermediate, or output variable

Suppose the network input is: $(x, y, z) = (-2, 5, -4)$

Then: $q = x + y = 3$ $\qquad f = qz = -12$ $\qquad$ (**forward** pass)

Backpropagation

$$f(x, y, z) = (x + y)z$$



Suppose the network input is: $(x, y, z) = (-2, 5, -4)$

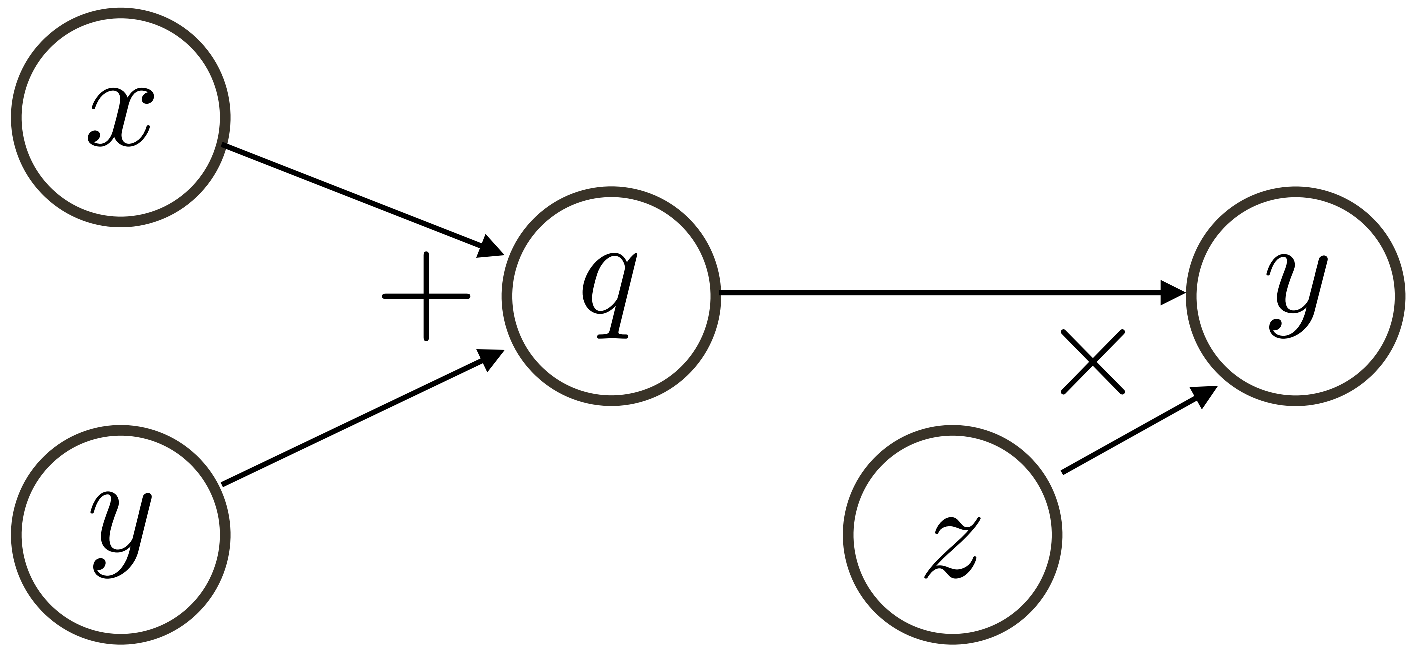
Then: $q = x + y = 3$ $\quad f = qz = -12$ (**forward** pass)

$$\frac{\partial f}{\partial q} = z = -4$$

(**backward** pass)

Backpropagation

$$f(x, y, z) = (x + y)z$$



$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = \frac{\partial f}{\partial q} \cdot 1$$

Suppose the network input is: $(x, y, z) = (-2, 5, -4)$

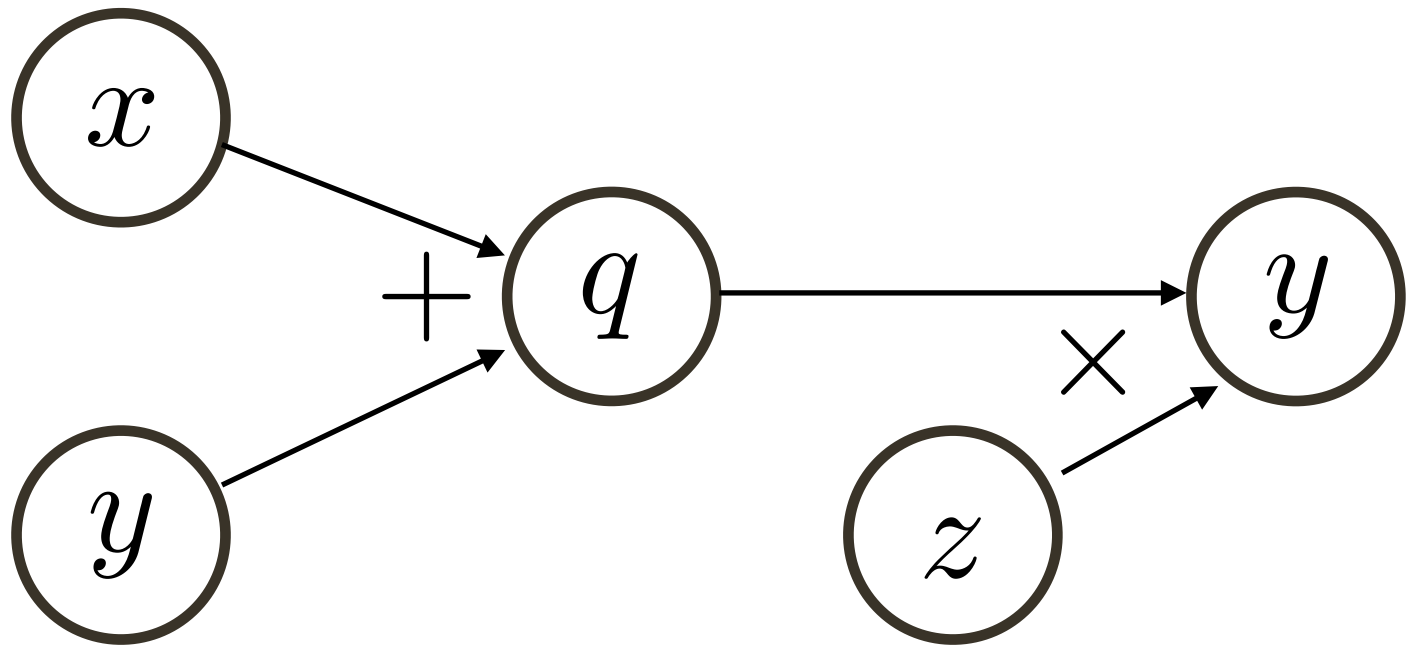
Then: $q = x + y = 3$ $\quad f = qz = -12$ $\quad$ (**forward** pass)

$$\frac{\partial f}{\partial q} = z = -4$$

(**backward** pass)

Backpropagation

$$f(x, y, z) = (x + y)z$$



$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = \frac{\partial f}{\partial q} \cdot 1$$

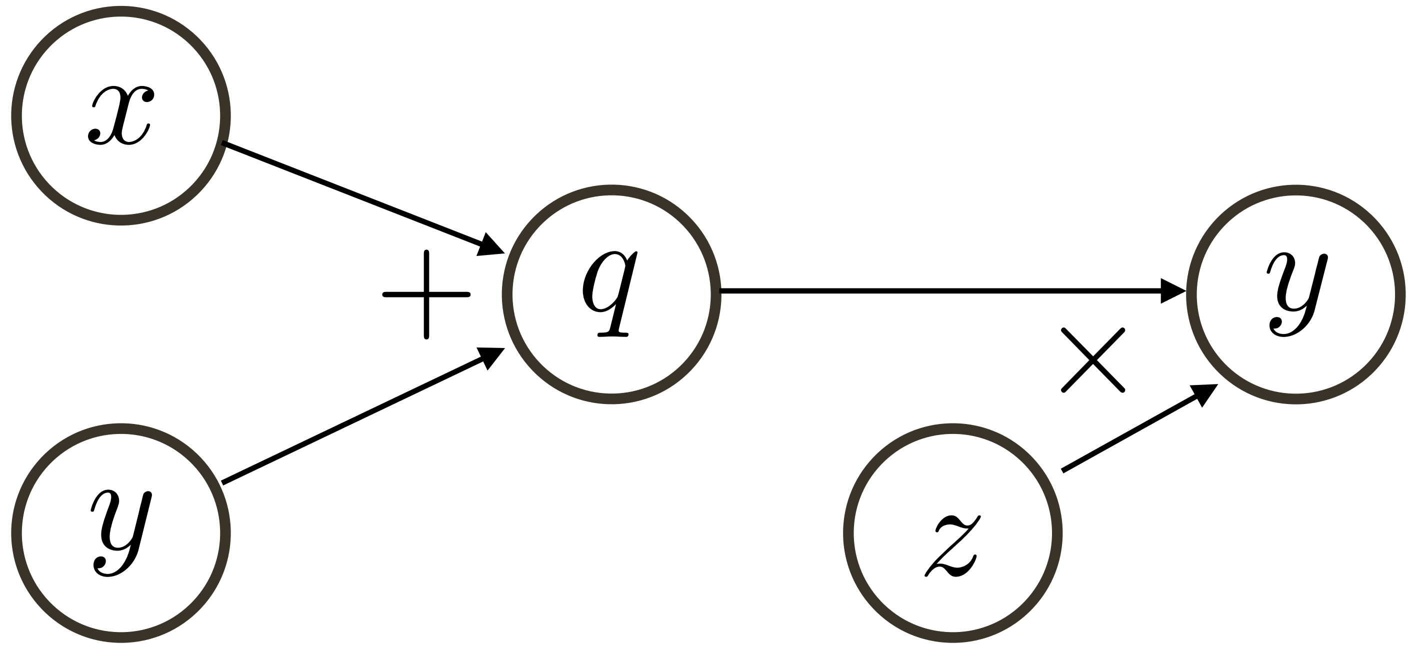
Suppose the network input is: $(x, y, z) = (-2, 5, -4)$

Then: $q = x + y = 3$ $\quad f = qz = -12$ (**forward** pass)

$$\frac{\partial f}{\partial q} = z = -4 \qquad \frac{\partial f}{\partial x} = -4$$

(**backward** pass)

Backpropagation



$$f(x, y, z) = (x + y)z$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = \frac{\partial f}{\partial q} \cdot 1 \qquad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = \frac{\partial f}{\partial q} \cdot 1 \qquad \frac{\partial f}{\partial z} = q$$

Suppose the network input is: $(x, y, z) = (-2, 5, -4)$

Then: $q = x + y = 3$ $\quad f = qz = -12$ $\quad$ (**forward** pass)

$$\frac{\partial f}{\partial q} = z = -4 \qquad \frac{\partial f}{\partial x} = -4 \qquad \frac{\partial f}{\partial y} = -4 \qquad \frac{\partial f}{\partial z} = 3$$

(**backward** pass)