DL in Applied Mathematics

Lecture 1: Introduction of Neural Networks. Multy Layers of Neural Networks and traing Data.

Marat Nurtas
PhD in Mathematical and Computer Modeling
Department of Mathematical and Computer Modeling
International Information Technology University, Almaty, Kazakhstan

- ***Introducing Neural Networks***
- We begin with a general idea of what **neural networks** are and why you might be interested in them. Neural networks, also called **Artificial Neural Networks**, are a type of machine learning often conflated with deep learning.

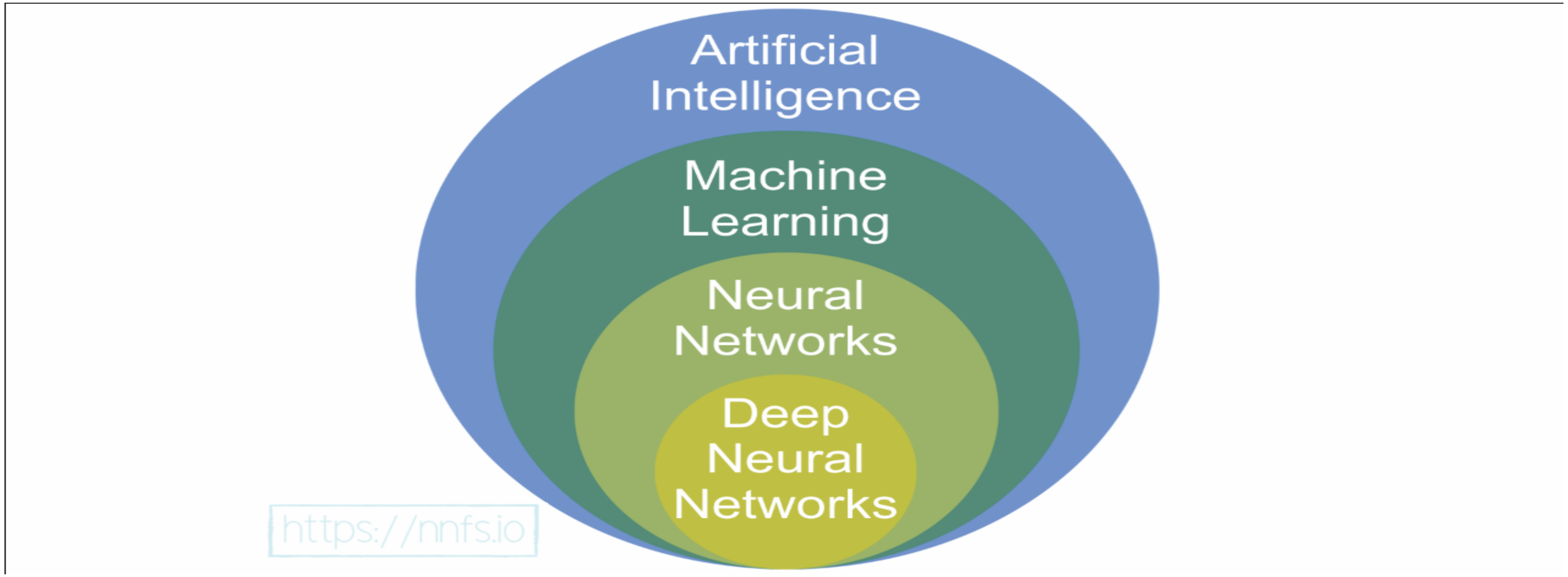


Fig 1.01: Depicting the various fields of artificial intelligence and where they fit in overall.

- **A Brief History**
- Since the advent of computers, scientists have been formulating ways to enable machines to take input and produce desired output for tasks like **classification** and **regression** . Additionally, in general, there's **supervised** and **unsupervised** machine learning.
- The "normal" and "failure" labels are **classifications** or **labels.** You may also see these referred to as **targets** or **ground-truths** while we fit a machine learning algorithm. These targets are the classifications that are the *goal* or *target* , known to be *true and correct* , for the algorithm to learn.
- In addition to classification, there's also **<u>regression</u>**, which is used to predict numerical values, like stock prices. There's also unsupervised machine learning, where the machine finds structure in data without knowing the labels/classes ahead of time.

- Neural networks were conceived in the 1940s, but figuring out how to train them remained a mystery for 20 years. The concept of **backpropagation** came in the 1960s, but neural networks still did not receive much attention until they started winning competitions in 2016.
- Since then, neural networks have been on a meteoric rise due to their sometimes seemingly magical ability to solve problems previously deemed unsolvable, such as image captioning, language translation, audio and video synthesis, and more.
- Currently, neural networks are the primary solution to most competitions and challenging technological problems like self-driving cars, calculating risk, detecting fraud, and early cancer detection, to name a few.

- **What is a Neural Network?**
- "Artificial" neural networks are inspired by the organic brain, translated to the computer. It's not a perfect comparison, but there are neurons, activations, and lots of interconnectivity, even if the underlying processes are quite different.

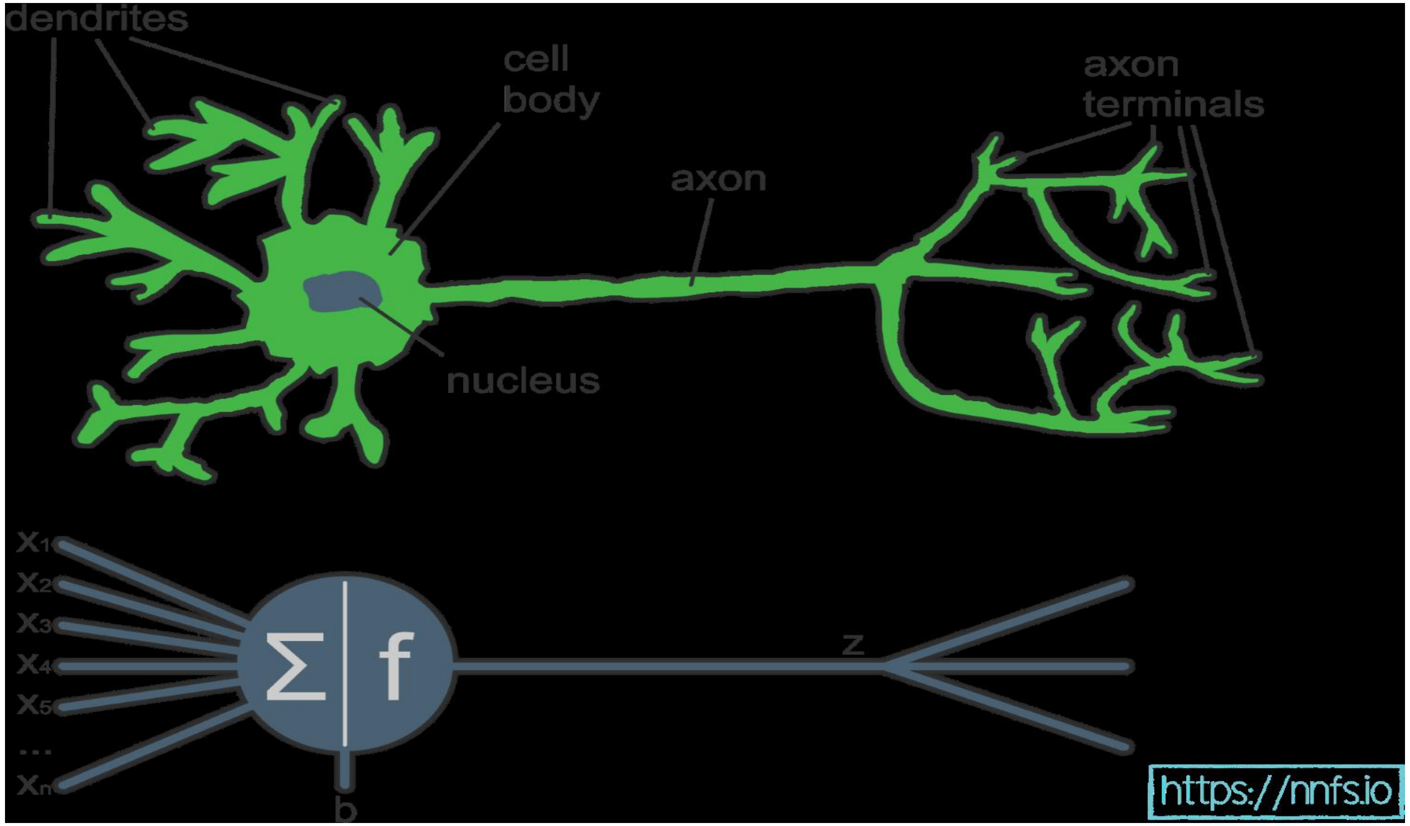


Fig 1.02: Comparing a biological neuron to an artificial neuron.

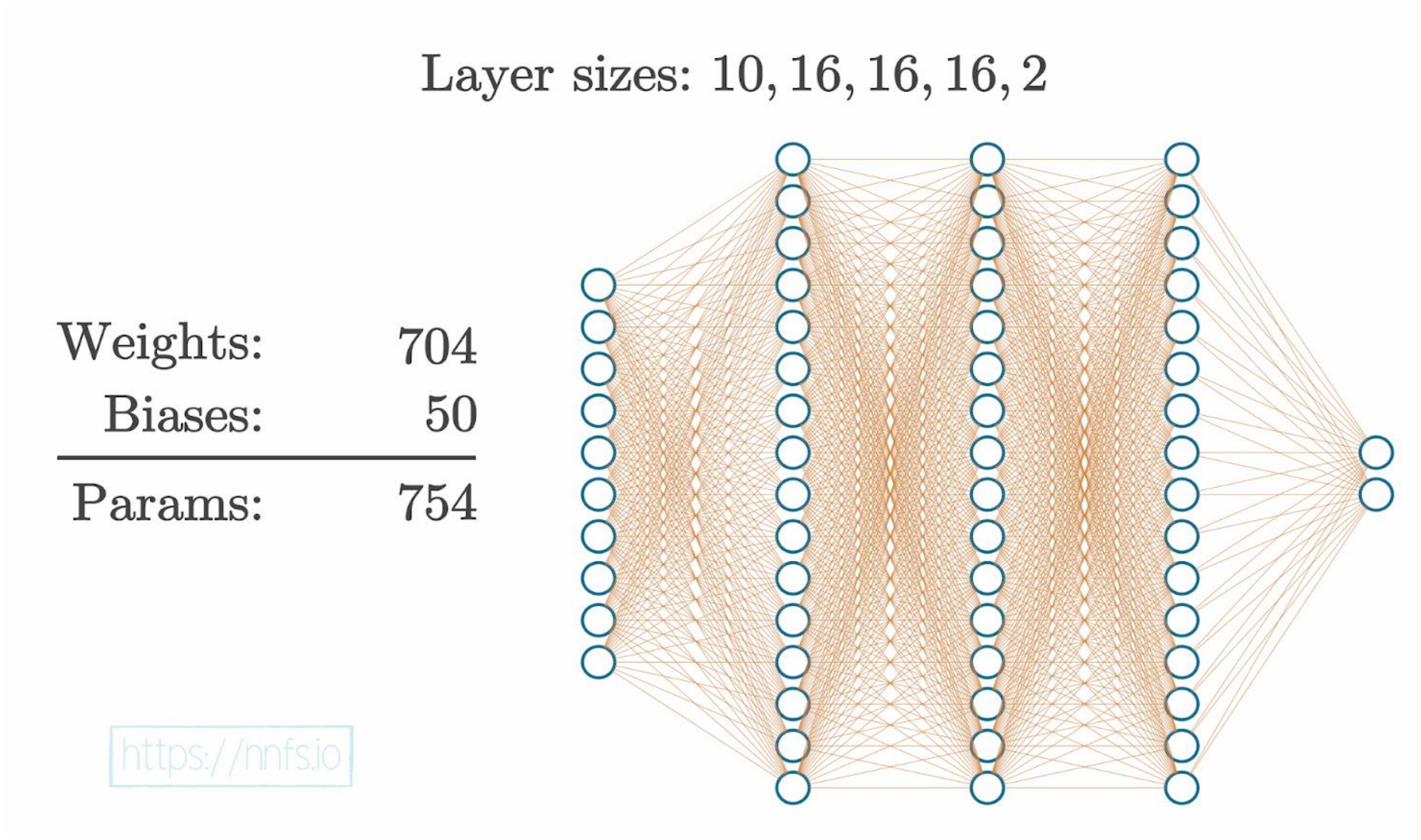


Fig 1.03: Example of a neural network with 3 hidden layers of 16 neurons each.

- The concept of weights and biases can be thought of as “knobs” that we can tune to fit our model to data. In a neural network, we often have thousands or even millions of these parameters tuned by the optimizer during training. Some may ask, “why not just have biases or just weights?”

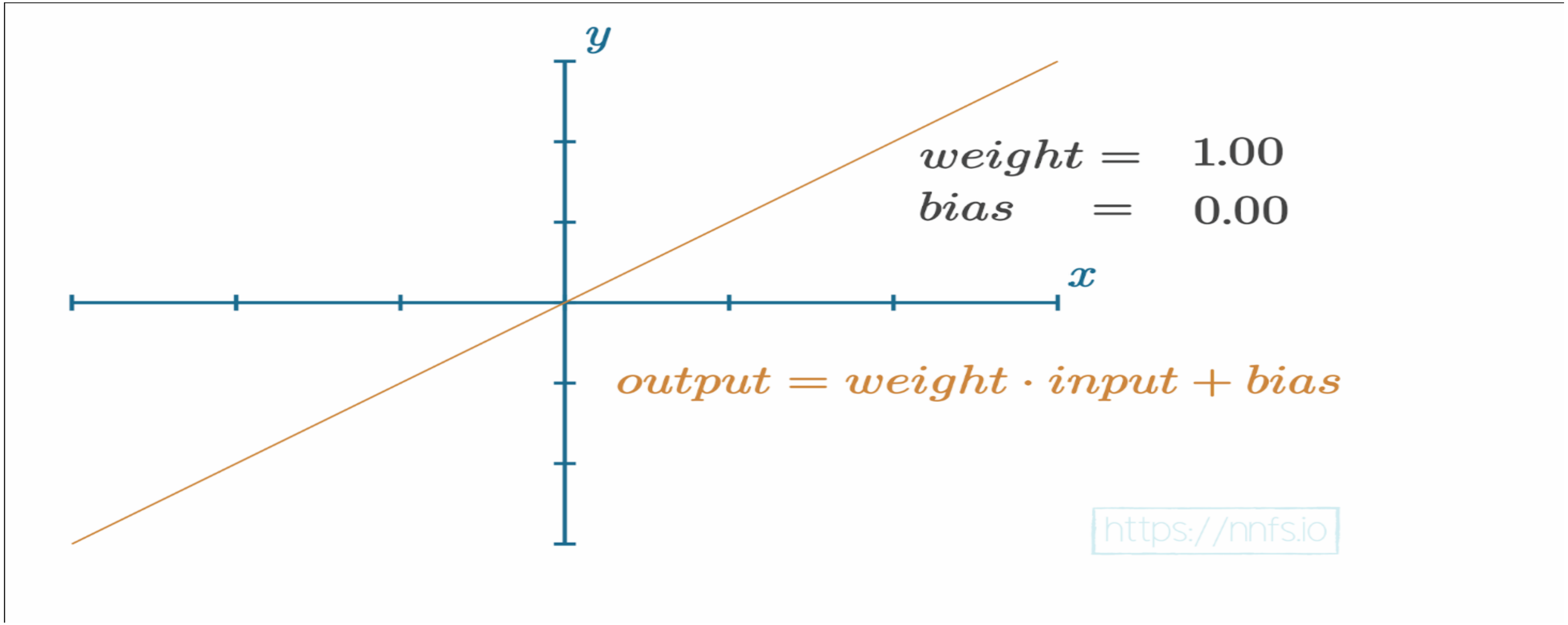


Fig 1.04: Graph of a single-input neuron’s output with a weight of 1, bias of 0 and input *x* .

- Adjusting the weight will impact the slope of the function:

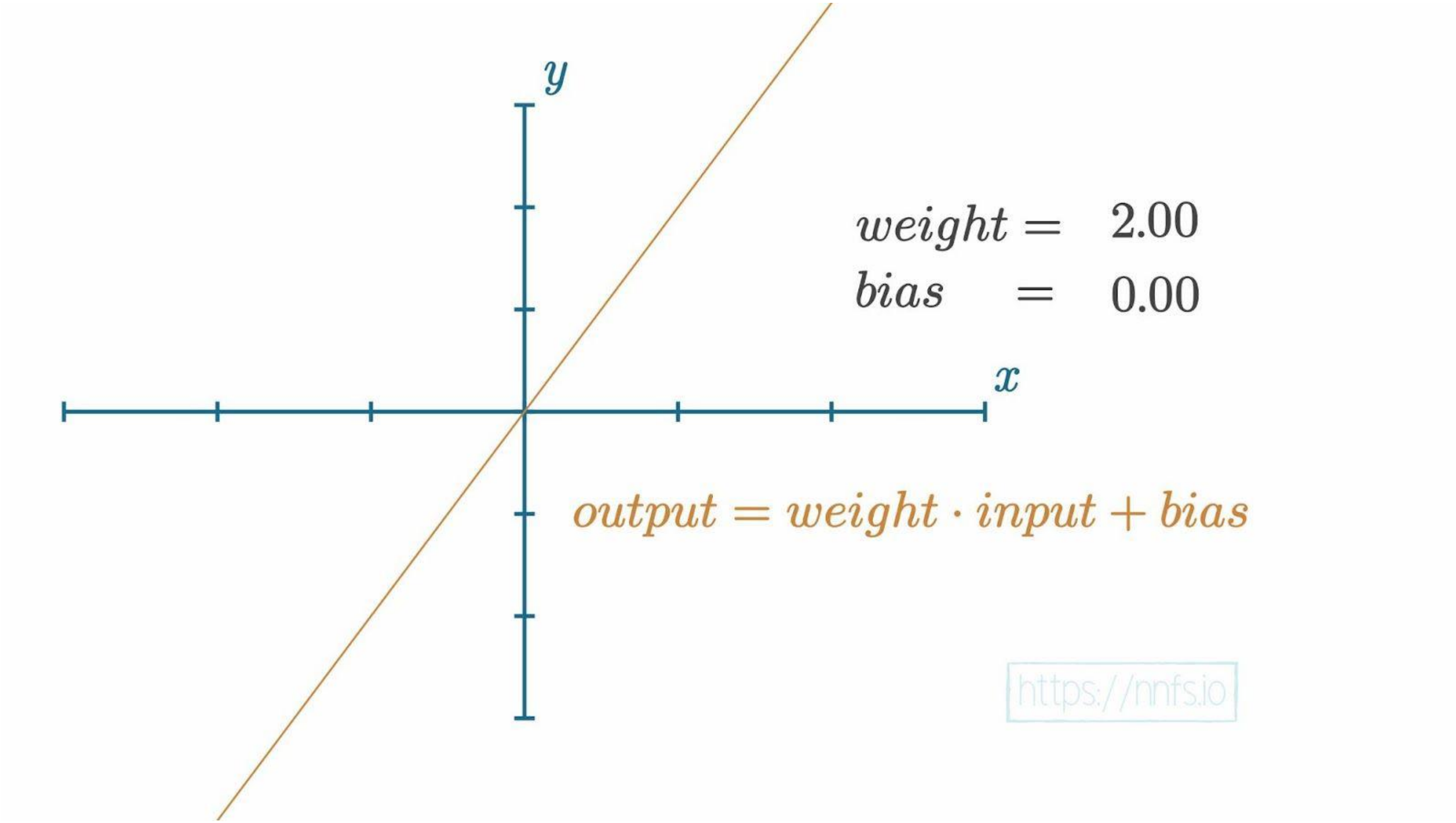


Fig 1.05: Graph of a single-input neuron's output with a weight of 2, bias of 0 and input *x* .

- As we increase the value of the weight, the slope will get steeper. If we decrease the weight, the slope will decrease. If we negate the weight, the slope turns to a negative:

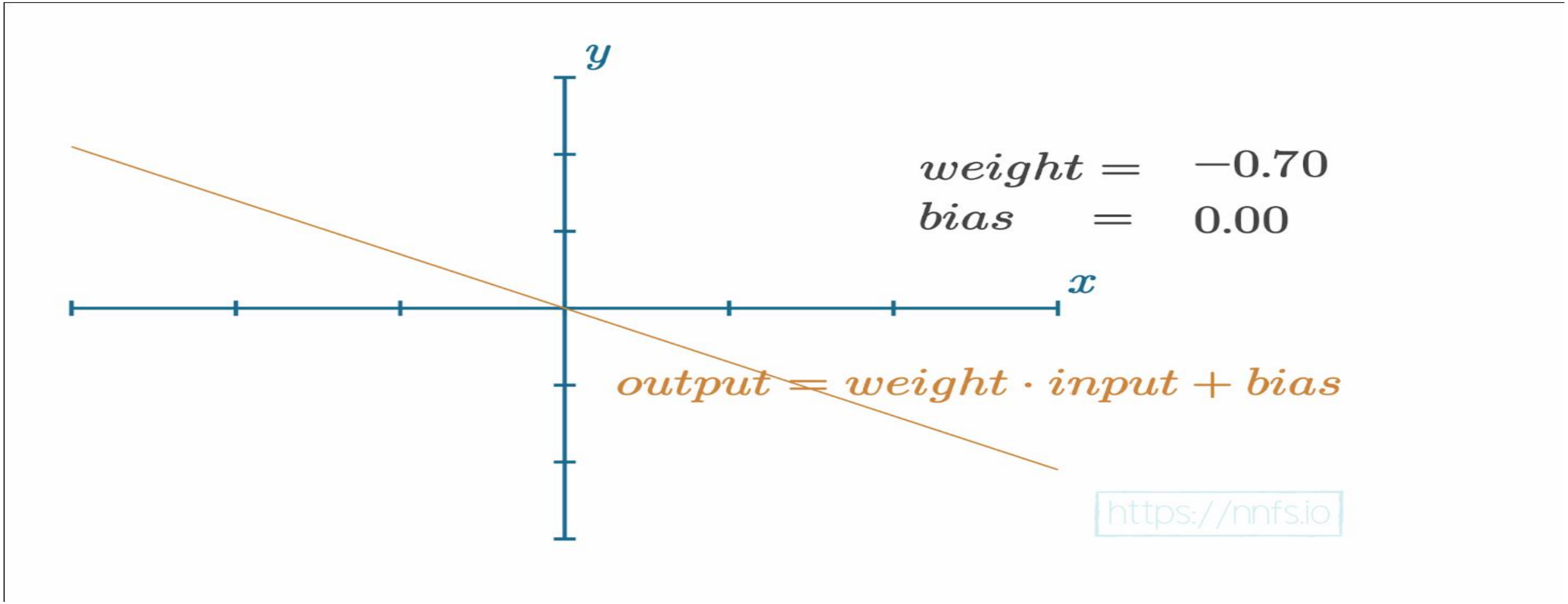


Fig 1.06: Graph of a single-input neuron's output with a weight of -0.70, bias of 0 and input *x* .

- The bias offsets the overall function. For example, with a weight of 1.0 and a bias of 2.0:

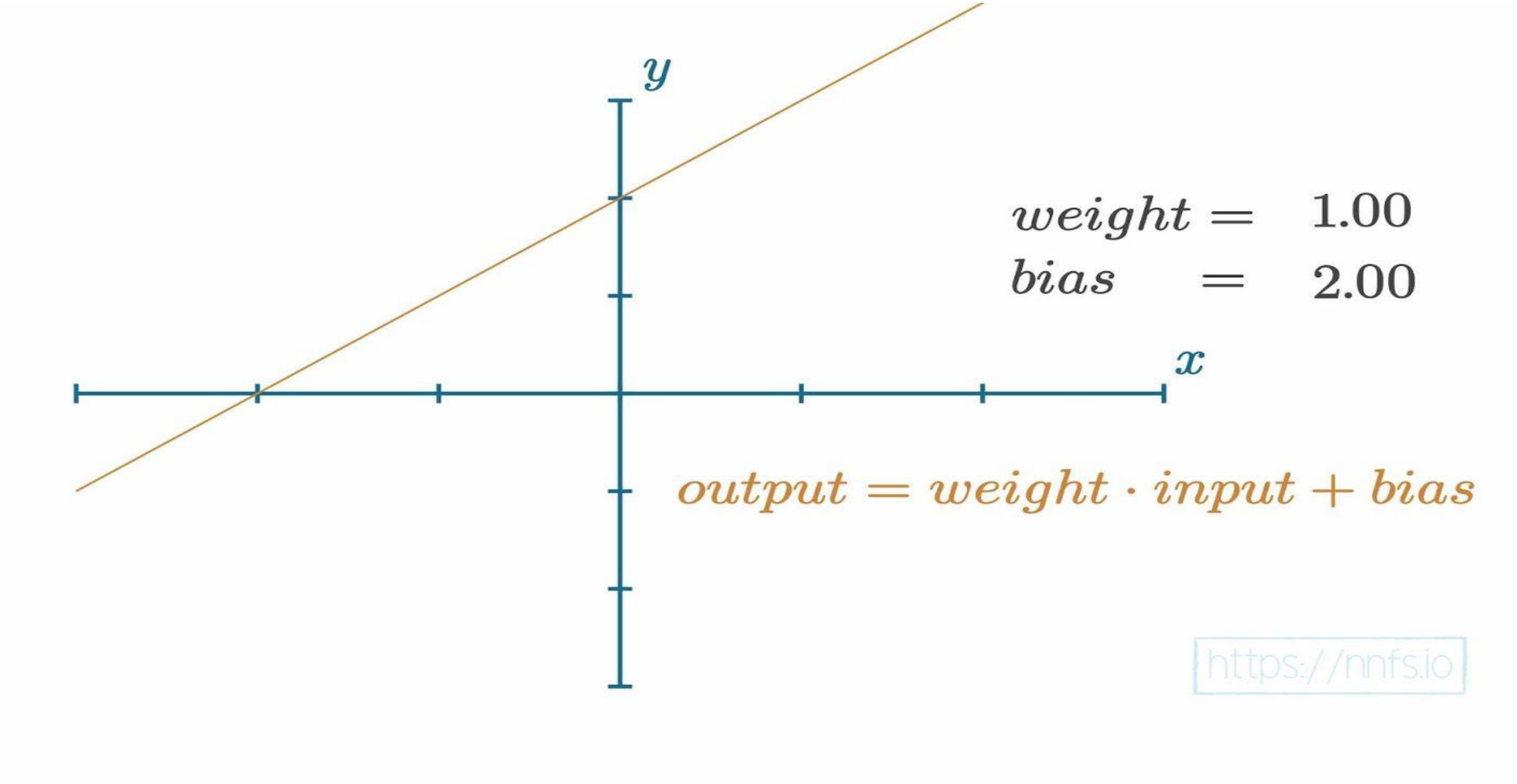


Fig 1.07: Graph of a single-input neuron's output with a weight of 1, bias of 2 and input *x* .

- As we increase the bias, the function output overall shifts upward. If we decrease the bias, then the overall function output will move downward. For example, with a negative bias:

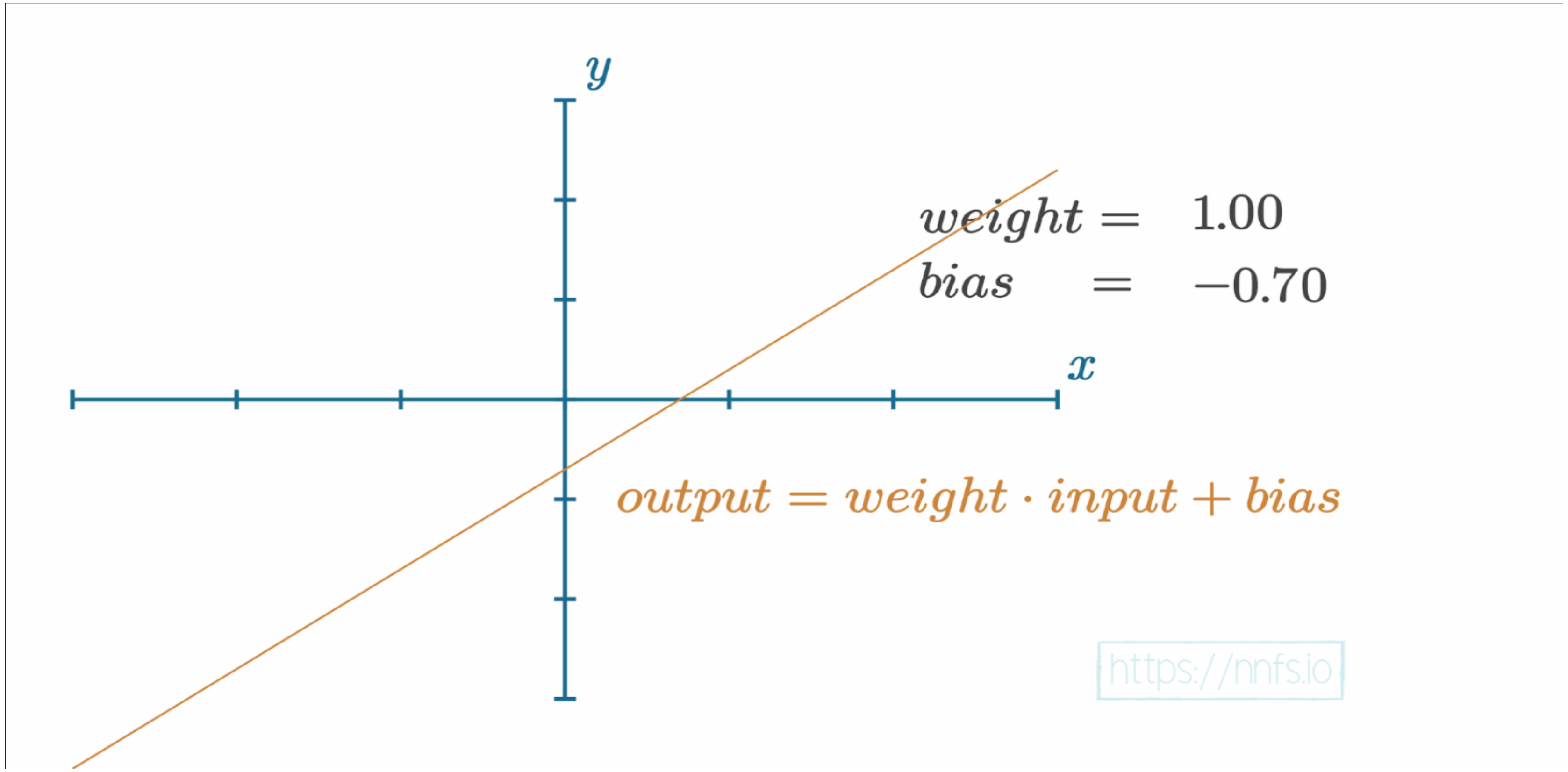


Fig 1.08: Graph of a single-input neuron's output with a weight of 1.0, bias of -0.70 and input *x* .

- In programming, an on-off switch as a function would be called a **step function** because it looks like a step if we graph it.

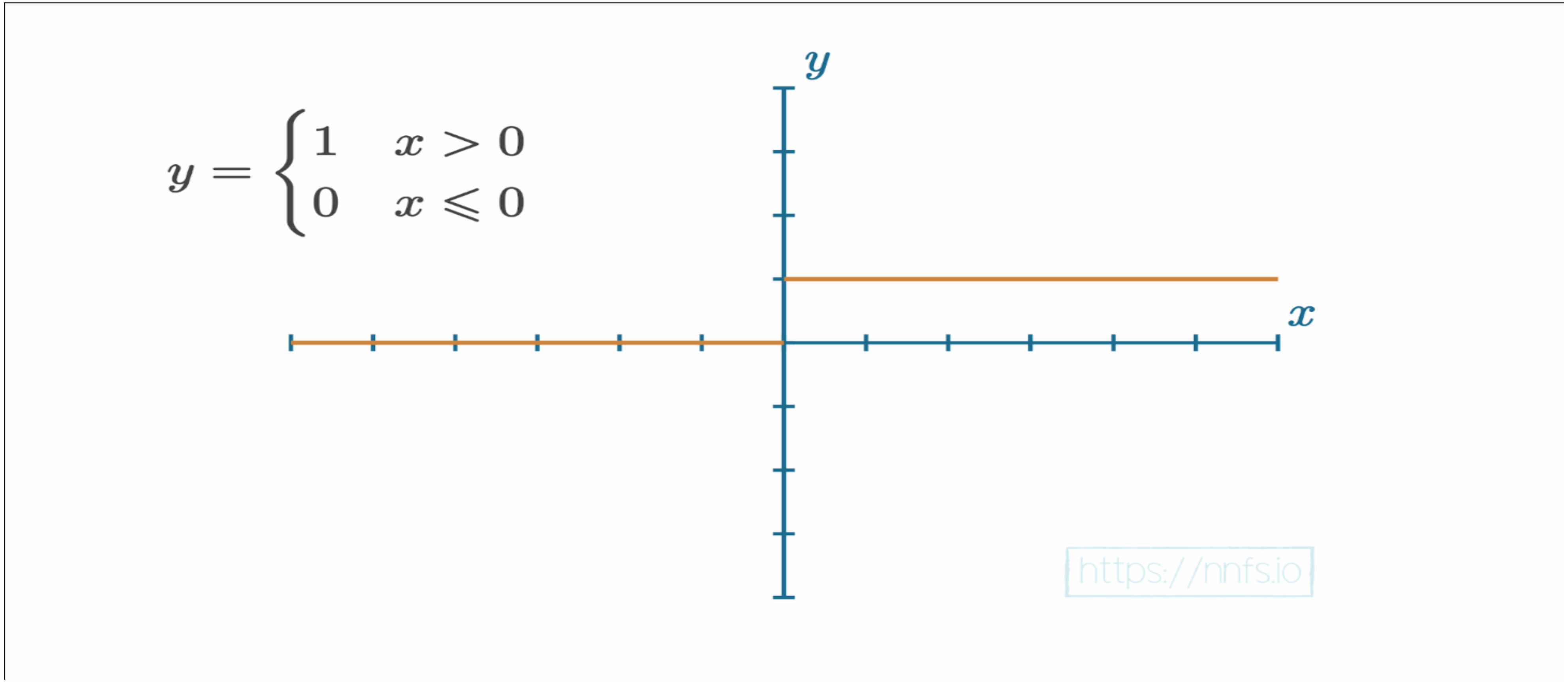


Fig 1.09: Graph of a step function.

The formula for a single neuron might look something like:

```
output = sum (inputs * weights) + bias
```

We then usually apply an activation function to this output, noted by *activation()* :

output = activation(output)

- While you can use a step function for your activation function, we tend to use something slightly more advanced. Neural networks of today tend to use more informative activation functions (rather than a step function), such as the **Rectified Linear** (ReLU) activation function, which we will cover in-depth in Lecture 4. Each neuron's output could be a part of the ending output layer, as well as the input to another layer of neurons.

- Example with 2 hidden layers of 4 neurons each.

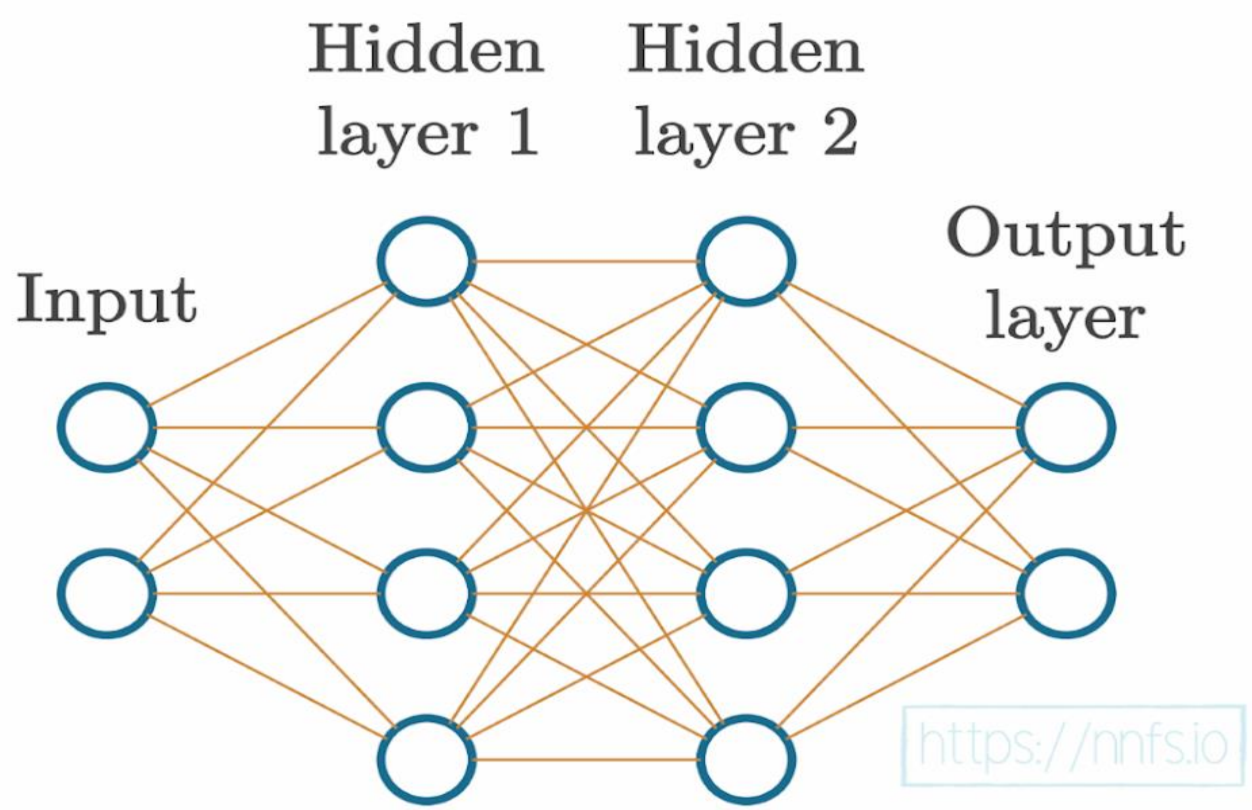
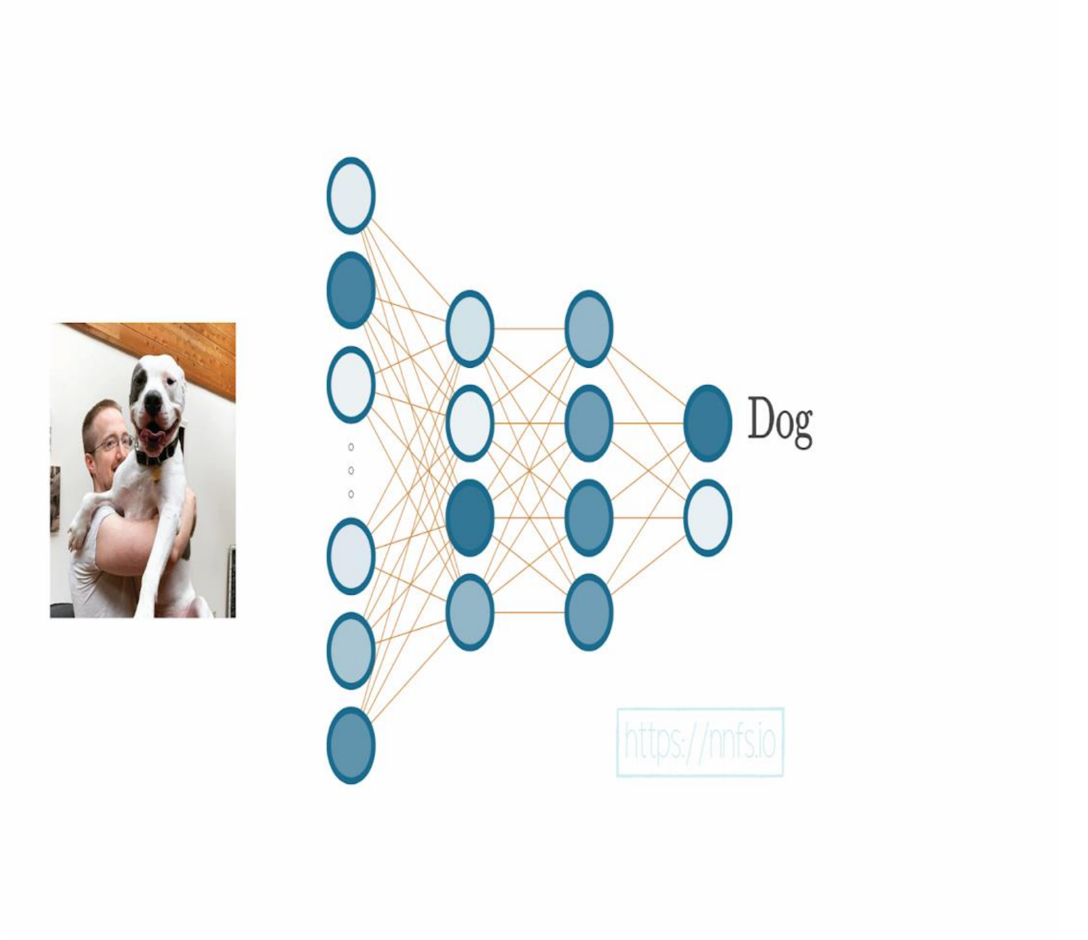


Fig 1.10: Example basic neural network.



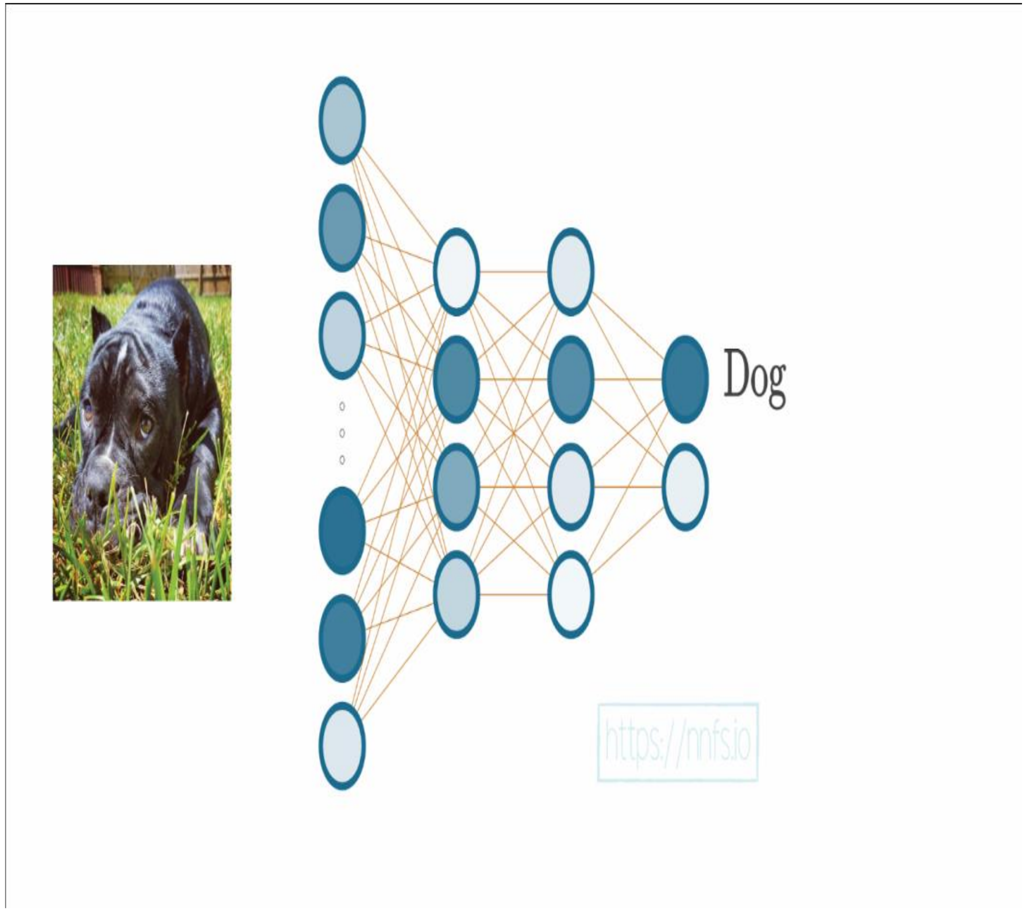


Fig 1.11 and Fig 1.12: Visual depiction of passing image data through a neural network, getting a classification

- When represented as one giant function, an example of a neural network's forward pass would be computed with:

$$L = -\sum_{l=1}^{N} y_l log(\forall_{j=1}^{n_3} \frac{e^{\sum_{i=1}^{n_2}(\forall_{j=1}^{n_2} max(0,\sum_{i=1}^{n_1}(\forall_{j=1}^{n_1} max(0,\sum_{i=1}^{n_0} X_i w_{1,i,j}+b_{1,j}))_i w_{2,i,j}+b_{2,j}))_i w_{3,i,j}+b_{3,j}}}{\sum_{k=1}^{n_3} e^{\sum_{i=1}^{n_2}(\forall_{j=1}^{n_2} max(0,\sum_{i=1}^{n_1}(\forall_{j=1}^{n_1} max(0,\sum_{i=1}^{n_0} X_i w_{1,i,j}+b_{1,k}))_i w_{2,i,j}+b_{2,k}))_i w_{3,i,k}+b_{3,k}}})$$

Fig 1.13: Full formula for the forward pass of an example neural network model.

1 Mathematical Formulation

At the heart of this deep learning revolution are familiar concepts from applied and computational mathematics; notably, in calculus, approximation theory, optimization and linear algebra.

This lesson provides a very brief introduction to the basic ideas that underlie deep learning from an applied mathematics perspective.

We focus on three fundamental questions:

- What is a deep neural network in Applied Mathematics?
- How is a network trained?
- What is the stochastic gradient method?

We illustrate the ideas with a short MATLAB and Python codes that sets up and trains a network.

2 Example of an Artificial Neural Network

- This class takes a data fitting view of artificial neural networks. To be concrete, consider the set of points shown in Figure 1.

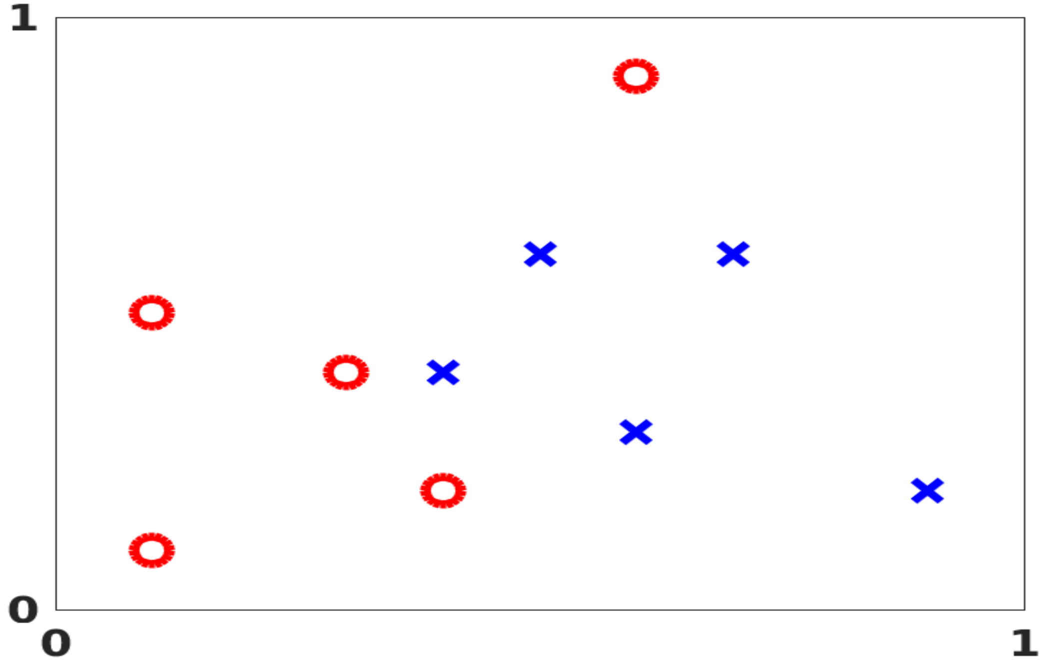


Figure 1: Labeled data points in R^2. Circles denote points in category A. Crosses denote points in category B.

For example, the data may show oil drilling sites on a map, where category A denotes a successful outcome. Can we use this data to categorize a newly proposed drilling site? Our job is to construct a transformation that takes any point in R^2 and returns either a circle or a square.

We will base our network on the sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}}, \qquad (1)$$

which is illustrated in the upper half of Figure 2 over the interval $-10 \leq x \leq 10$.

- The sigmoid also has the convenient property that its derivative takes the simple form

$$\sigma'(x) = \sigma(x)\left(1-\sigma(x)\right), \qquad (2)$$

which is straightforward to verify.

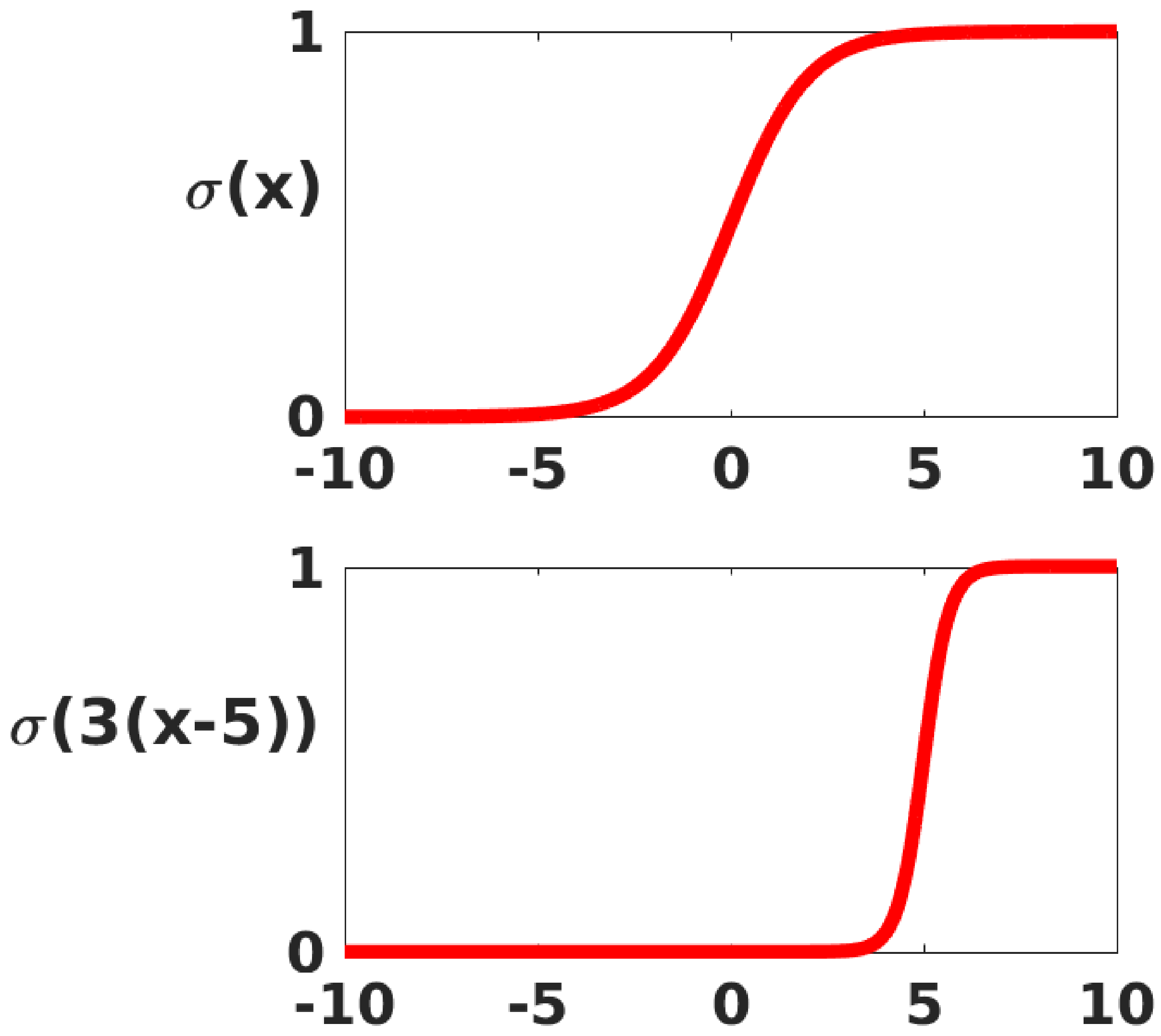


Figure 2: Upper: sigmoid function (1). Lower: sigmoid with shifted and scaled input.

- The lower plot in Figure 2 shows $\sigma(3(x-5))$. The factor 3 has sharpened the changeover and the shift -5 has altered its location. To keep our notation manageable, we need to interpret the sigmoid function in a factorized sense. For $z \in R^m, \sigma : R^m \to R^m$ is defined by applying the sigmoid function in the obvious componentwise manner, so that

$$(\sigma(z))_i = \sigma(z_i).$$

With this notation, we can set up layers of neurons.

- Introducing some mathematics, if the real numbers produced by the neurons in one layer are collected into a vector, **a**, then the vector of outputs from the next layer has the form

$$\sigma(Wa + b). \qquad (3)$$

Here, **W** is matrix and **b** is a vector. We say that W contains the weights and **b** contains the biases.

- To emphasize the role of the ith neuron in (3), we could pick out the ith component as

$$\sigma\left(\sum_j w_{ij}a_j + b_i\right),$$

where the sum runs over all entries in **a**. Throughout this class, we will be switching between the vectorized and componentwise viewpoints to strike a balance between clarity and brevity.

- **Figure 3** represents an artificial neural network with four layers. We will apply this form of network to the problem defined by **Figure 1**.

Since the input data has the form $x \in \mathbb{R}^2$, the weights and biases for layer two may be represented by a matrix $W^{[2]} \in \mathbb{R}^{2\times 2}$ and a vector $b^{[2]} \in \mathbb{R}^2$, respectively. The output from layer two then has the form $\sigma(W^{[2]}x + b^{[2]}) \in \mathbb{R}^2$.

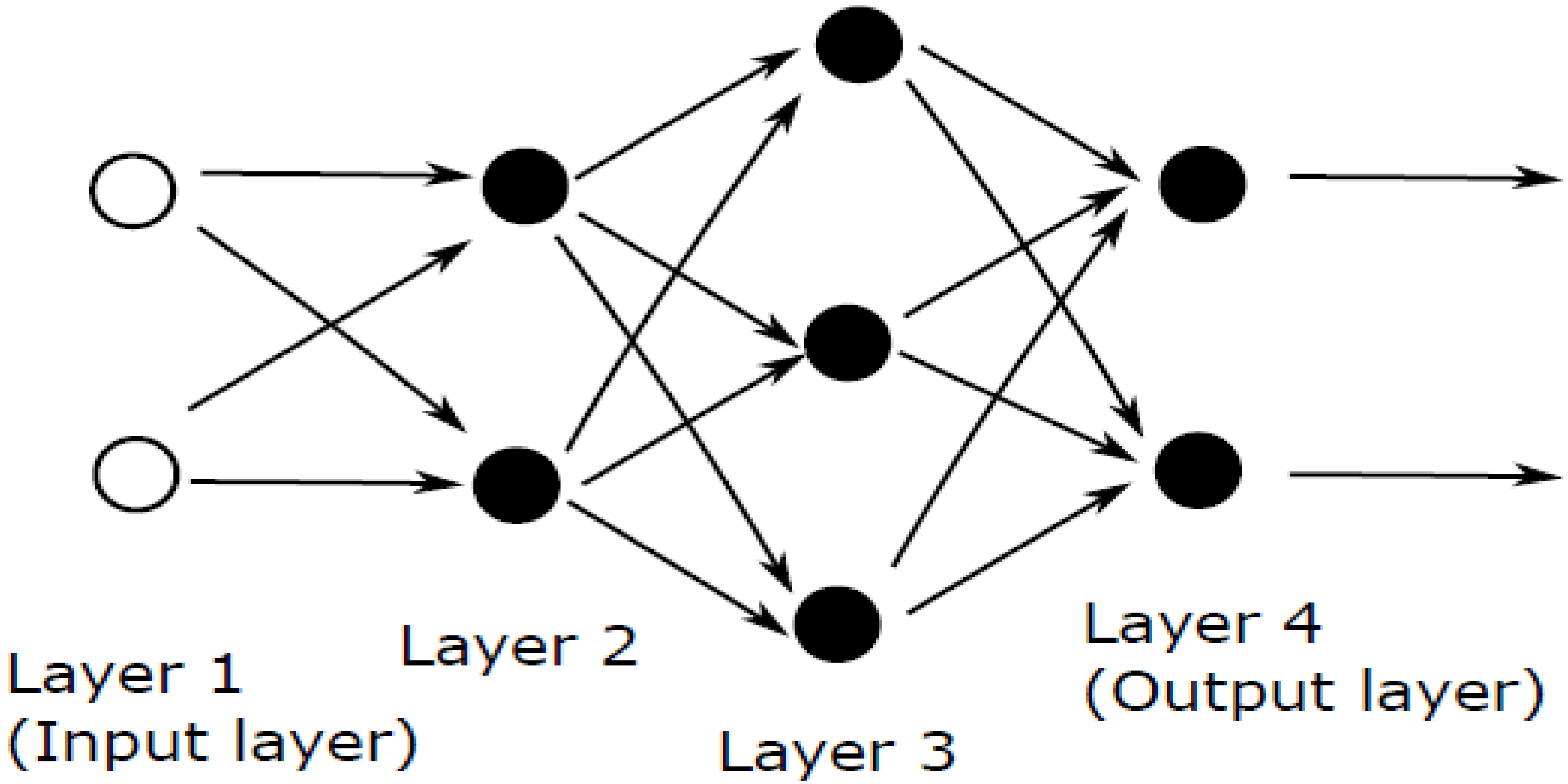


Figure 3: A network with four layers.

Weights and biases for <u>layer three</u> may be represented by a matrix $W^{[3]} \in \mathbb{R}^{3\times 2}$ and a vector $b^{[3]} \in \mathbb{R}^3$, respectively. The output from layer three then has the form

$$\sigma\left(W^{[3]}\sigma(W^{[2]}x + b^{[2]}) + b^{[3]}\right) \in \mathbb{R}^3.$$

The <u>fourth (output) layer</u>: $W^{[4]} \in \mathbb{R}^{2\times 3}$, $b^{[4]} \in \mathbb{R}^{[2]}$, respectively. The output from layer four, and hence from the overall network, has the form

$$F(x) = \sigma\left(W^{[4]}\sigma\left(W^{[3]}\sigma(W^{[2]}x + b^{[2]}) + b^{[3]}\right) + b^{[4]}\right) \in \mathbb{R}^2. \qquad (4)$$

The expression (4) defines a function $F : \mathbb{R}^2 \to \mathbb{R}^2$ in terms of its **23** parameters-the entries in the weight matrices and bias vectors.

- **Recall** that our aim is to produce a classifier based on the data in **Figure 1.**

We will require $F(x)$ to be close to $[1,0]^T$ for data points in category **A** and close to $[0,1]^T$ for data points in category **B.** Then, given a new point $x \in \mathbb{R}^2$, it would be reasonable to classify it according to the largest component of $F(x)$; that is, category **A** If $F_1(x) > F_2(x)$ and category **B** if $F_1(x) < F_2(x)$, with some rule to break ties.

- Denoting the ten data points in **Figure 1** by $\{x^{\{i\}}\}_{i=1}^{10}$, we use $y(x^{\{i\}})$ for the target output; that is,

$$y(x^{\{i\}}) = \begin{cases} \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \text{if } x^{\{i\}} \text{ is in category A,} \\ \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \text{if } x^{\{i\}} \text{ is in category B.} \end{cases} \quad (5)$$

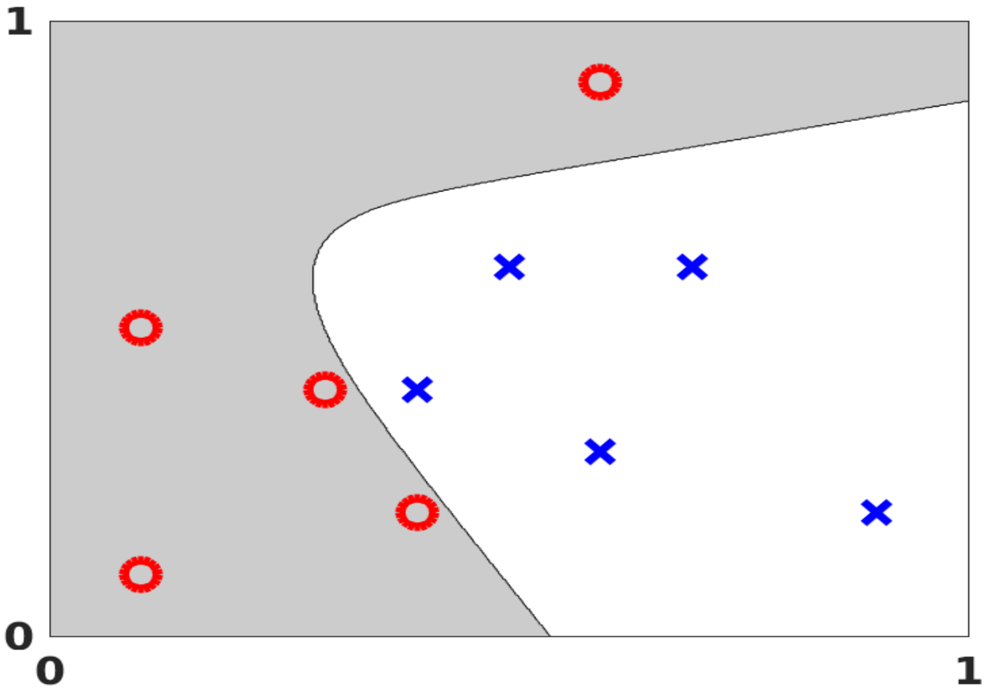


Figure 4: Visualization of output from an artificial neural network applied to the data in Figure 1.

Our cost function then takes the form

$$\mathrm{Cost}\left(W^{[2]}, W^{[3]}, W^{[4]}, b^{[2]}, b^{[3]}, b^{[4]}\right) = \frac{1}{10}\sum_{i=1}^{10} \tfrac{1}{2}\|y(x^{\{i\}}) - F(x^{\{i\}})\|_2^2. \qquad (6)$$

Choosing the weights and biases in a way that minimizes the cost function is refered to as **training the network**.

- For the data in Figure 1, we used the MATLAB optimization toolbox to minimize the cost function (6) over the 23 parameters defining $W^{[2]}$, $W^{[3]}$, $W^{[4]}$, $b^{[2]}$, $b^{[3]}$ and $b^{[4]}$. More precisely, we used the nonlinear least-squares solver **lsqnonlin**. For the trained network, **Figure 4** shows the boundary where $F_1(x) > F_2(x)$. So, with this approach, any point in the shaded region would be assigned to category **A** and any point in the unshaded region to category **B**.

- **Figure 5** shows how the network responds to additional training data. Here we added one further category B point, indicated by the extra cross at (0.3; 0.7), and re-ran the optimization routine.

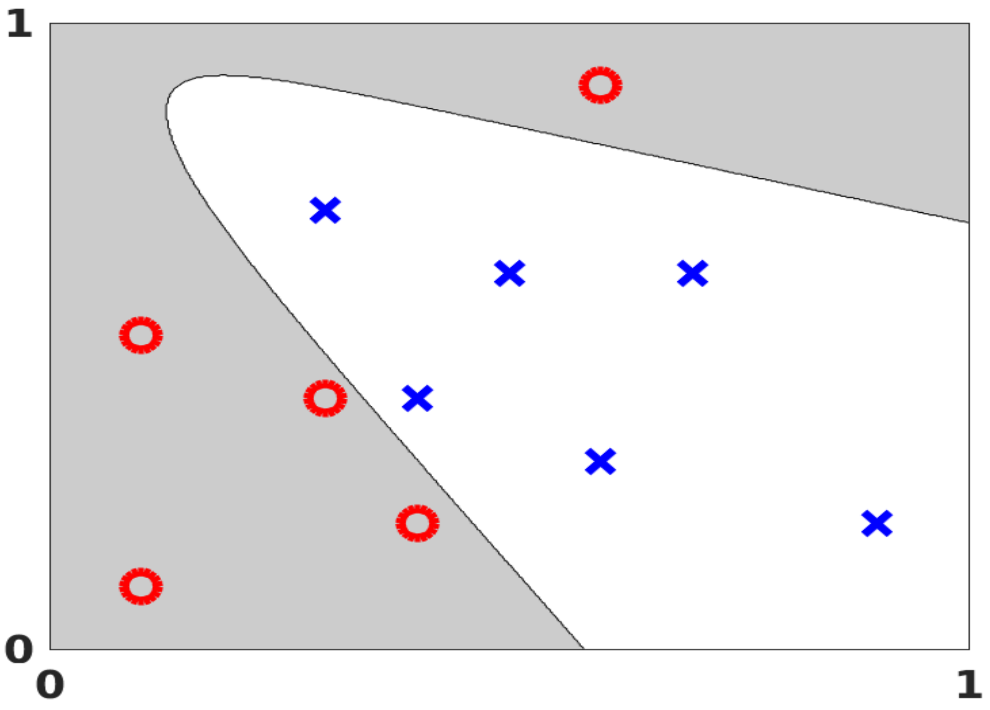


Figure 5: Repeat of the experiment in Figure 4 with an additional data point.

Indeed, some experimentation with the location of the data points in Figure 4 and with the choice of initial guess for the weights and biases makes it clear that **lsqnonlin**, with its default settings, cannot always find an acceptable solution.

3 The General Set-up

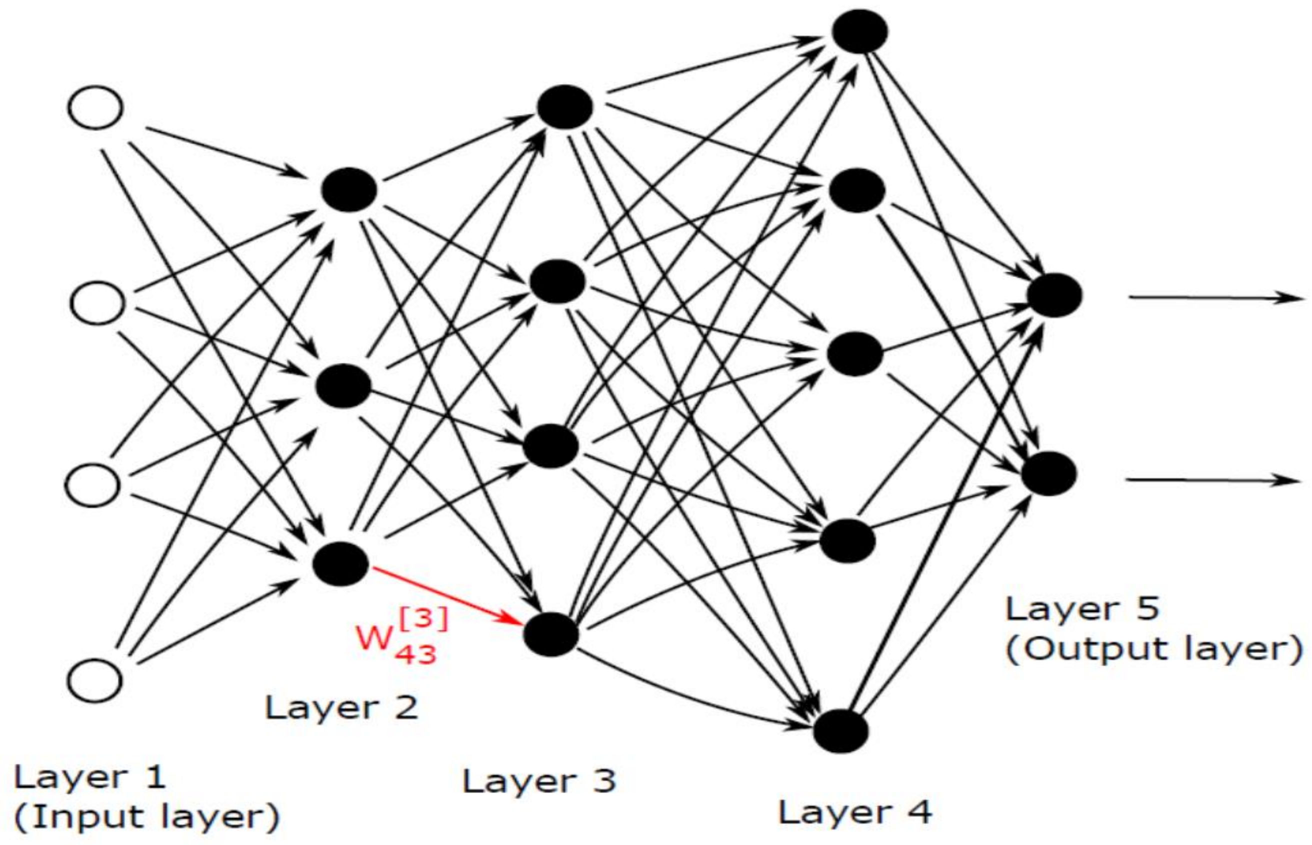


Figure 6: A network with five layers. The edge corresponding to the weight $w^{[3]}_{43}$ is highlighted. The output from neuron number 3 at layer 2 is weighted by the factor $w^{[3]}_{43}$ when it is fed into neuron number 4 at layer 3.

Suppose that layer l, for l = 1, 2, 3, …,L contains n_l neurons. So n_1 is the dimension of input data. Overall, the network maps from $\mathbb{R}^{n_1}$ to $\mathbb{R}^{n_L}$. We use $W^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$ to denote the matrix of weights at layer l. More precisely, $w^{[l]}_{jk}$ is the weight that neuron **j** at layer l applies to the output from neuron k at layer l-1. Similarly, $b^{[l]} \in \mathbb{R}^{n_l}$ is the vector of biases for layer l, so neuron **j** at layer l uses the bias $b^{[l]}_j$.

In Fig 6 we give an example with $L = 5$ layers. Here, $n_1 = 4$, $n_2 = 3$, $n_3 = 4$, $n_4 = 5$ and $n_5 = 2$, so $W^{[2]} \in \mathbb{R}^{3\times 4}$, $W^{[3]} \in \mathbb{R}^{4\times 3}$, $W^{[4]} \in \mathbb{R}^{5\times 4}$, $W^{[5]} \in \mathbb{R}^{2\times 5}$, $b^{[2]} \in \mathbb{R}^3$, $b^{[3]} \in \mathbb{R}^4$, $b^{[4]} \in \mathbb{R}^5$ and $b^{[5]} \in \mathbb{R}^2$.

- Given an input $x \in \mathbb{R}^{n_1}$, we may then neatly summarize the action of the network by letting $a^{[l]}_j$ denote the output, or activation, from neuron **j** at layer l. So, we have

$$a^{[1]} = x \in \mathbb{R}^{n_1}, \quad (7)$$

$$a^{[l]} = \sigma\left(W^{[l]} a^{[l-1]} + b^{[l]}\right) \in \mathbb{R}^{n_l}, \quad \text{for } l = 2, 3, \ldots, L. \quad (8)$$

Now suppose we have N pieces of data, or *training points*, in $\mathbb{R}^{n_1}$, $\{x^{\{i\}}\}_{i=1}^{N}$, for which there are given target outputs $\{y(x^{\{i\}})\}_{i=1}^{N}$ in $\mathbb{R}^{n_L}$. Generalizing (6), the quadratic cost function that we wish to minimize has the form

$$\text{Cost} = \frac{1}{N}\sum_{i=1}^{N} \tfrac{1}{2}\|y(x^{\{i\}}) - a^{[L]}(x^{\{i\}})\|_2^2, \qquad (9)$$

where, to keep notation under control, we have not explicitly indicated that Cost is a function of all the weights and biases.

Thank you for attention!