CS50's Introduction to Artificial Intelligence with Python
CS50 的人工智能入门与 Python

OpenCourseWare 开放课程

Donate (https://cs50.harvard.edu/donate)

Brian Yu (https://brianyu.me)
brian@cs.harvard.edu

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan) (https://www.instagram.com/davidjmalan/)
(https://www.linkedin.com/in/malan/) (https://www.reddit.com/user/davidjmalan)
(https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

Lecture 5 第五讲

Neural Networks 神经网络

AI neural networks are inspired by neuroscience. In the brain, neurons are cells that are connected to each other, forming networks. Each neuron is capable of both receiving and sending electrical signals. Once the electrical input that a neuron receives crosses some threshold, the neuron activates, thus sending its electrical signal forward.

人工智能神经网络受到神经科学的启发。在大脑中，神经元是相互连接的细胞，形成网络。每个神经元都能够接收和发送电信号。一旦神经元接收到的电信号超过某个阈值，神经元就会激活，从而发送其电信号向前。

An **Artificial Neural Network** is a mathematical model for learning inspired by biological neural networks. Artificial neural networks model mathematical functions that map inputs to outputs based on the structure and parameters of the network. In artificial neural networks, the structure of the network is shaped through training on data.

人工神经网络是一种受生物神经网络启发的学习数学模型。人工神经网络模拟将输入映射到输出的数学函数，基于网络的结构和参数。在人工神经网络中，通过数据训练塑造网络结构。

When implemented in AI, the parallel of each neuron is a **unit** that's connected to other units. For example, like in the last lecture, the AI might map two inputs, x_1 and x_2, to whether it is going to rain today or not. Last lecture, we suggested the following form for this hypothesis function: $h(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2$, where w_1 and w_2 are weights that modify the inputs, and w_0 is a constant, also called **bias**, modifying the value of the whole expression.

在人工智能中实现时，每个神经元的并行对应物是一个与其他单元相连的单位。例如，就像在上一节课中一样，人工智能可能会将两个输入 x_1和 x_2映射到今天是否会下雨。上一节课我们建议了以下形式的假设函数：$h(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2$，其中 w_1和 w_2是修改输入的权重，而 w_0是一个常数，也称为偏差，用于修改整个表达式的值。

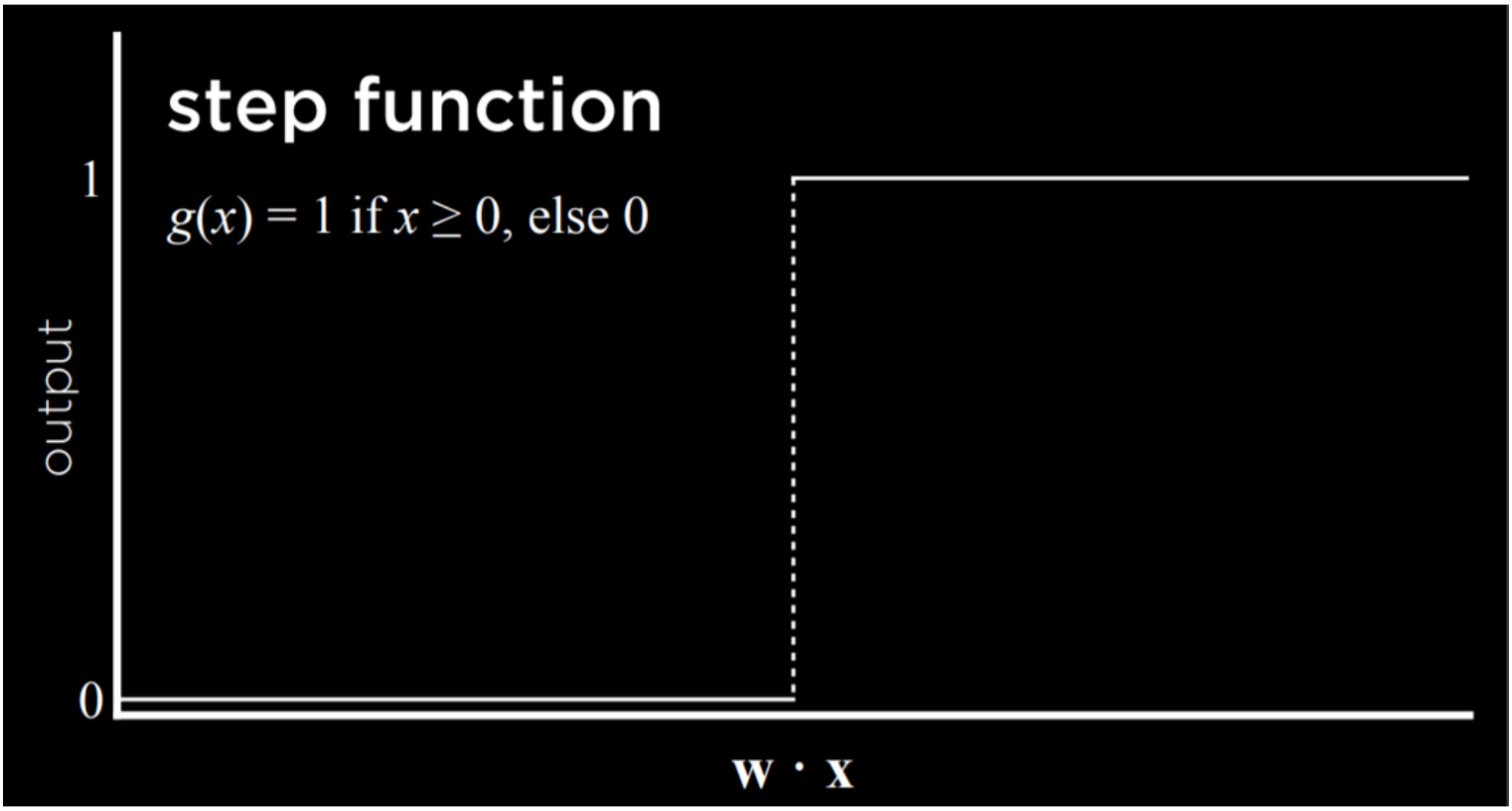
Activation Functions 激活函数

To use the hypothesis function to decide whether it rains or not, we need to create some sort of threshold based on the value it produces.

使用假设函数来决定是否下雨，我们需要根据其产生的值创建某种阈值。

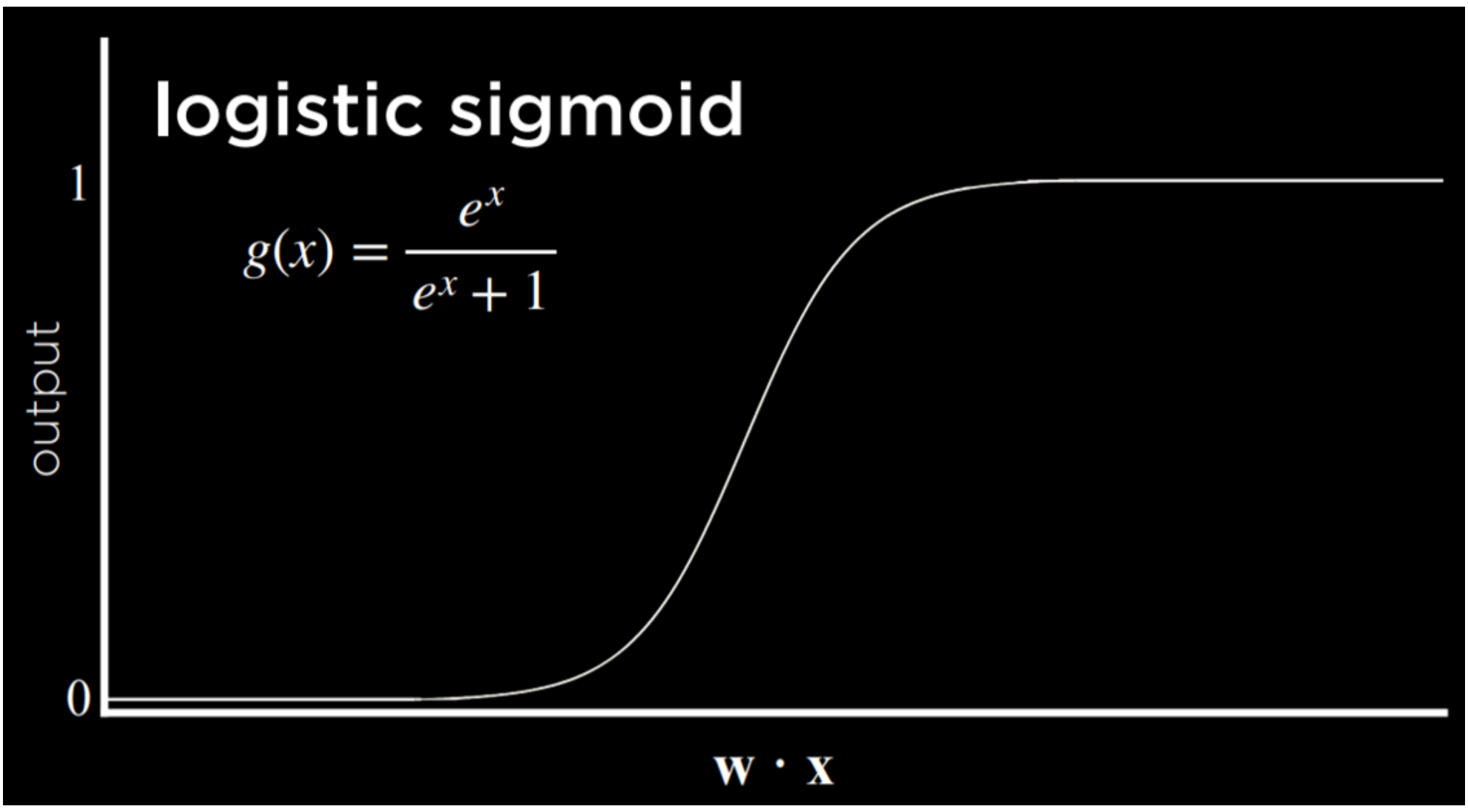
One way to do this is with a step function, which gives 0 before a certain threshold is reached and 1 after the threshold is reached.

实现这一目标的一种方法是使用阶跃函数，该函数在达到某个阈值之前为 0，在达到阈值之后为 1。



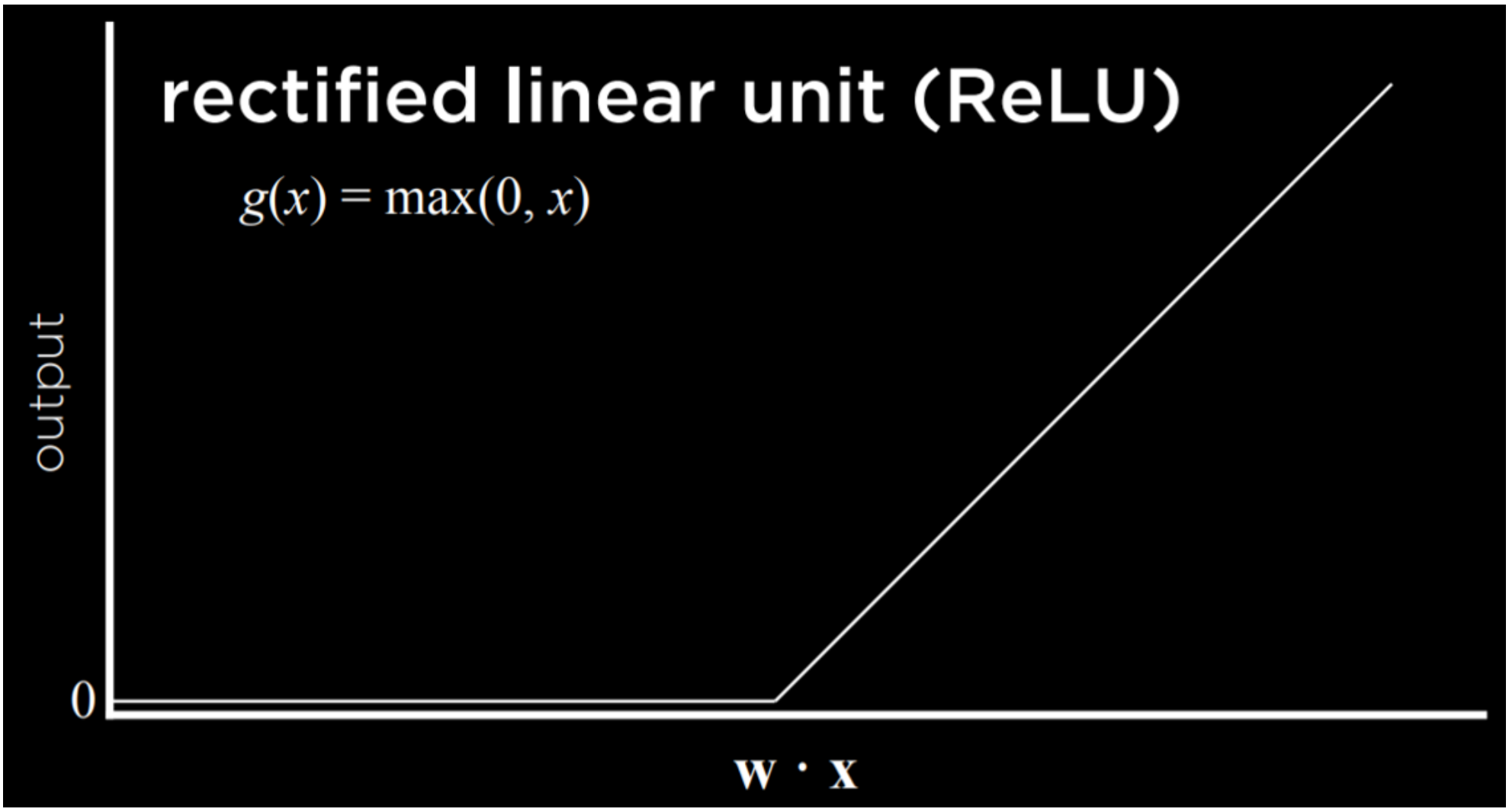
Another way to go about this is with a logistic function, which gives as output any real number from 0 to 1, thus expressing graded confidence in its judgment.

另一种方法是使用逻辑函数，它能输出从 0 到 1 的任何实数，从而表达其判断的分级信心。



Another possible function is Rectified Linear Unit (ReLU), which allows the output to be any positive value. If the value is negative, ReLU sets it to 0.

另一种可能的功能是修正线性单元（ReLU），它允许输出为任何正数值。如果值为负数，ReLU 将其设置为 0。



Whichever function we choose to use, we learned last lecture that the inputs are modified by weights in addition to the bias, and the sum of those is passed to an activation function. This stays true for simple neural networks.

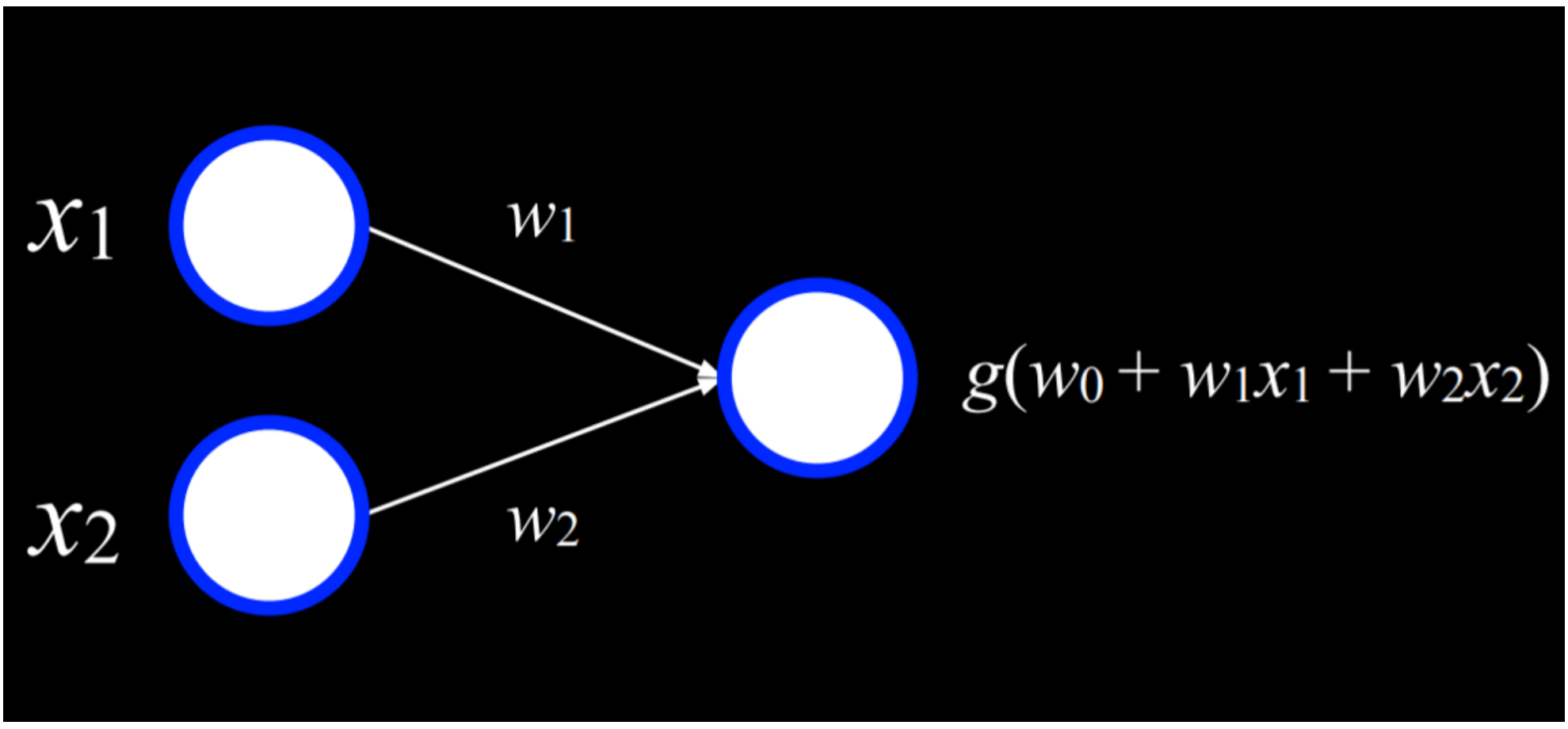
无论我们选择使用哪种功能，上一节课我们学到输入会受到权重和偏差的修改，然后将这些修改后的值传递给激活函数。对于简单的神经网络，这一点仍然成立。

Neural Network Structure

神经网络结构

A neural network can be thought of as a representation of the idea above, where a function sums up inputs to produce an output.

神经网络可以被认为是上述想法的一种表示，其中函数将输入相加以产生输出。



The two white units on the left are the input and the unit on the right is an output. The inputs are connected to the output by a weighted edge. To make a decision, the output unit multiplies the inputs by their weights in addition to the bias (w_0), and the uses function g to determine the output.

左侧的两个白色单元是输入，右侧的单元是输出。输入通过加权边与输出相连。为了做出决定，输出单元将输入乘以它们的权重，再加上偏置（w_0），然后使用函数 g 来确定输出。

右侧单元的输入是 $w_0 + w_1x_1 + w_2x_2$，输出为 g 的函数值。

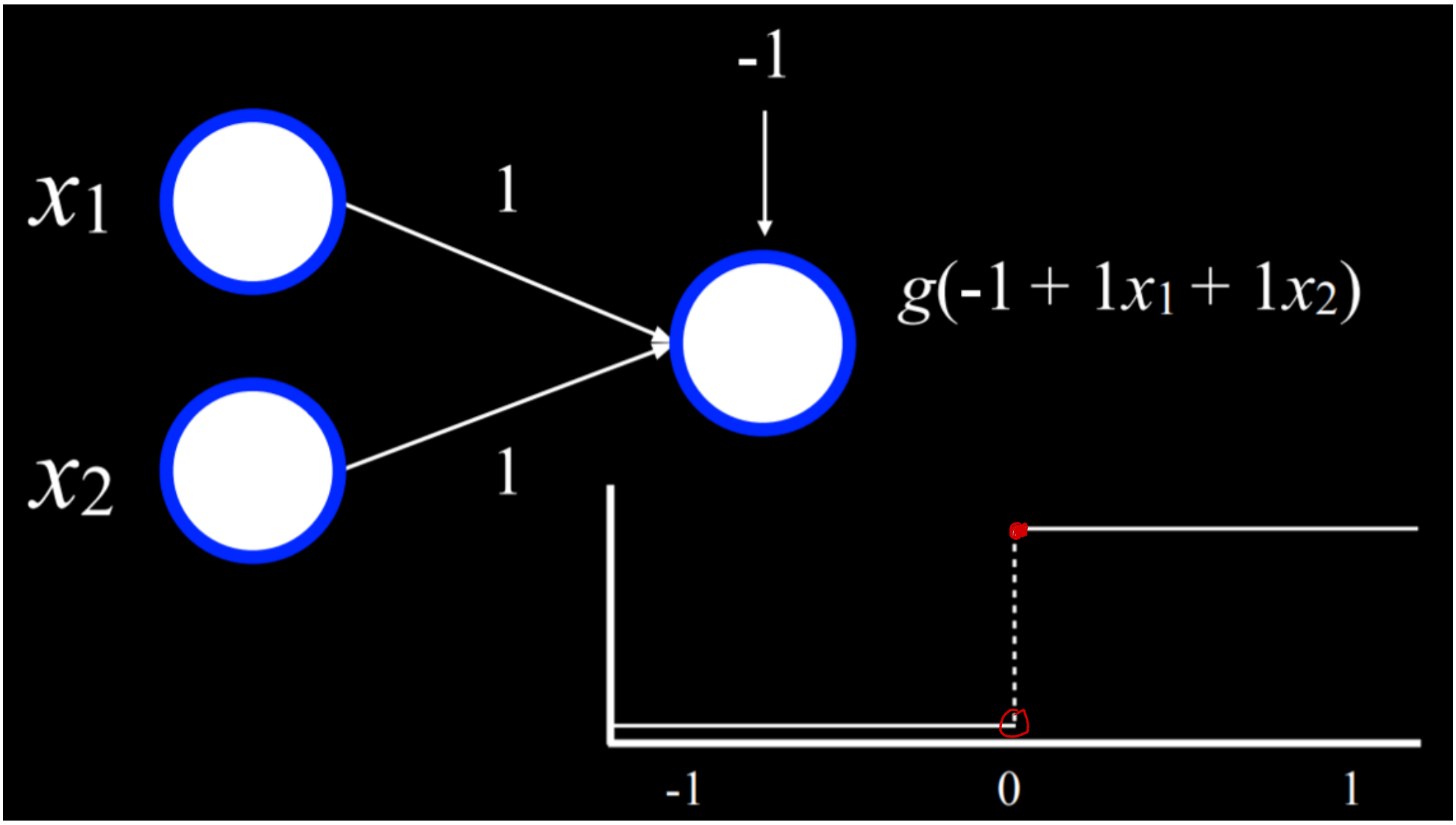
For example, an Or logical connective can be represented as a function *f* with the following truth table:

例如，一个或逻辑连接符可以表示为一个函数 f，其具有以下真值表：| 输入 1 | 输入 2 | 函数 f | |-------|-------|-------| | 真 | 真 | 真 | | 真 | 假 | 真 | | 假 | 真 | 真 | | 假 | 假 | 假 |

x	*y*	*f(x, y)*
0	0	0
0	1	1
1	0	1
1	1	1

We can visualize this function as a neural network. x_1 is one input unit, and x_2 is another input unit. They are connected to the output unit by an edge with a weight of 1. The output unit then uses function $g(-1 + 1x_1 + 2x_2)$ with a threshold of 0 to output either 0 or 1 (false or true).

我们可以将此功能可视化为一个神经网络。x_1 是一个输入单元，x_2 是另一个输入单元。它们通过权重为 1 的边与输出单元相连。然后，输出单元使用函数 $g(-1 + 1x_1 + 2x_2)$ 并带有阈值 0，输出要么是 0（假）要么是 1（真）。



For example, in the case where $x_1 = x_2 = 0$, the sum is (-1). This is below the threshold, so the function *g* will output 0. However, if either or both of x_1 or x_2 are equal to 1, then the sum of all inputs will be either 0 or 1. Both are at or above the threshold, so the function will output 1.

例如，在 $x_1 = x_2 = 0$ 的情况下，总和是-1。这低于阈值，因此函数 g 将输出 0。但是，如果 x_1或 x_2中的任何一个或两个等于 1，那么所有输入的总和将是 0 或 1。两者都在或高于阈值，因此函数将输出 1。

A similar process can be repeated with the And function (where the bias will be (-2)). Moreover, inputs and outputs don't have to be distinct. A similar process can be used to take humidity and air pressure as input, and produce the probability of rain as output. Or, in a different example, inputs can be money spent on advertising and the month when it was spent to get the output of expected revenue from sales. This can be extended to any number of inputs by multiplying each input $x_1 \dots x_n$ by weight $w_1 \dots w_n$, summing up the resulting values and adding a bias w_0.

类似的过程可以用 And 函数重复（偏差为 -2）。此外，输入和输出不必是唯一的。可以使用类似的过程将湿度和气压作为输入，产生降雨概率作为输出。或者，在不同的例子中，输入可以是用于广告的花费和花费的月份，以获得销售预期收入的输出。通过将每个输入 $x_1 \dots x_n$ 乘以权重 $w_1 \dots w_n$，将结果值相加，并添加偏差 w_0，可以将输入的数量扩展到任意数量。

Gradient Descent 梯度下降

链接：_梯度下降解释

Gradient descent is an algorithm for minimizing loss when training neural networks. As was mentioned earlier, a neural network is capable of inferring knowledge about the structure of the network itself from the data. Whereas, so far, we defined the different weights, neural networks allow us to compute these weights based on the training data. To do this, we use the gradient descent algorithm, which works the following

way:

梯度下降是一种在训练神经网络时最小化损失的算法。正如之前提到的，神经网络能够从数据中推断出网络本身的结构。到目前为止，我们定义了不同的权重，而神经网络允许我们根据训练数据计算这些权重。为了做到这一点，我们使用梯度下降算法，其工作方式如下：

- Start with a random choice of weights. This is our naive starting place, where we don't know how much we should weight each input.

  从随机选择的权重开始。这是我们无知的起点，我们不知道应该如何给每个输入赋予权重。

- Repeat: 重复：
  - Calculate the gradient based on all data points that will lead to decreasing loss. Ultimately, the gradient is a vector (a sequence of numbers).

    根据所有将导致损失减少的数据点计算梯度。最终，梯度是一个向量（一系列数字）。
  - Update weights according to the gradient.

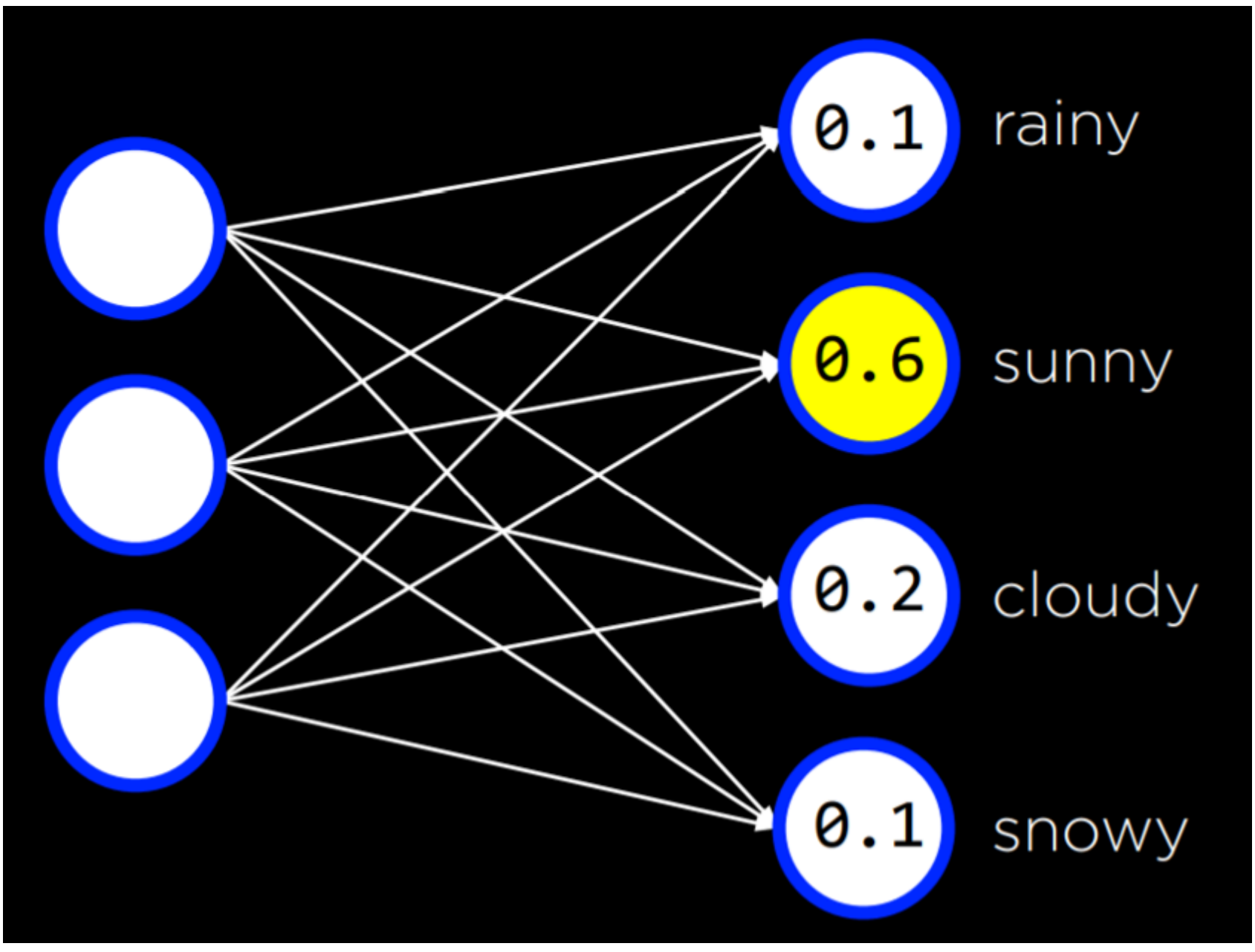
    根据梯度更新权重。

The problem with this kind of algorithm is that it requires to calculate the gradient based on *all data points*, which is computationally costly. There are a multiple ways to minimize this cost. For example, in **Stochastic Gradient Descent**, the gradient is calculated based on one point chosen at random. This kind of gradient can be quite inaccurate, leading to the **Mini-Batch Gradient Descent** algorithm, which computes the gradient based on on a few points selected at random, thus finding a compromise between computation cost and accuracy. As often is the case, none of these solutions is perfect, and different solutions might be employed in different situations.

这种算法的问题在于，它需要基于所有数据点来计算梯度，这在计算上成本很高。有多种方法可以减少这种成本。例如，在随机梯度下降中，基于随机选择的一个点来计算梯度。这种梯度可能相当不准确，导致了基于随机选择的几个点来计算梯度的 mini-batch 梯度下降算法，从而在计算成本和准确性之间找到一个平衡。如常，这些解决方案中没有一个是完美的，不同的解决方案可能在不同的情况下被采用。

Using gradient descent, it is possible to find answers to many problems. For example, we might want to know more than "will it rain today?" We can use some inputs to generate probabilities for different kinds of weather, and then just choose the weather that is most probable.

使用梯度下降法，可以解决许多问题。例如，我们可能想知道的不仅仅是"今天会下雨吗？"我们可以使用一些输入来生成不同天气类型的概率，然后选择最有可能的天气。



This can be done with any number of inputs and outputs, where each input is connected to each output, and where the outputs represent decisions that we can make. Note that in this kind of neural networks the outputs are not connected. This means that each output and its associated weights from all the inputs can be be seen as an individual neural network and thus can be trained separately from the rest of the

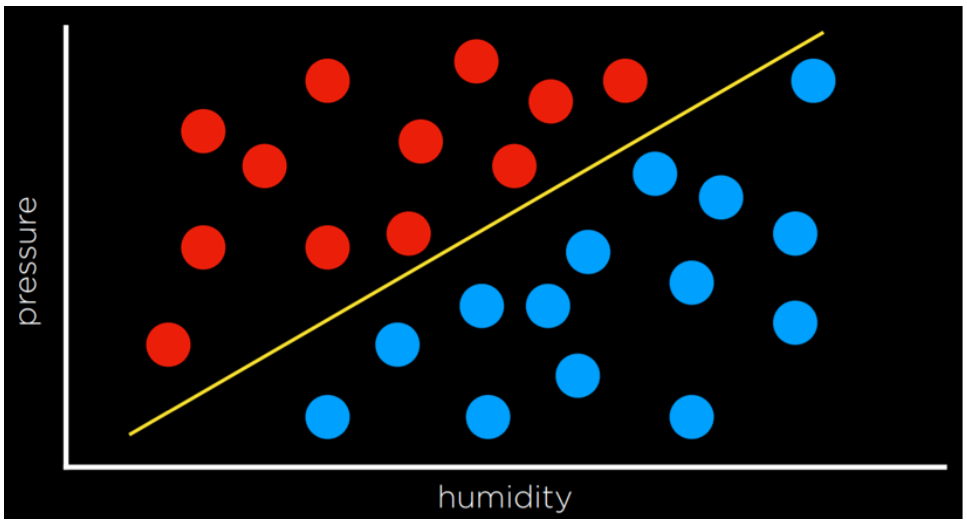
outputs.

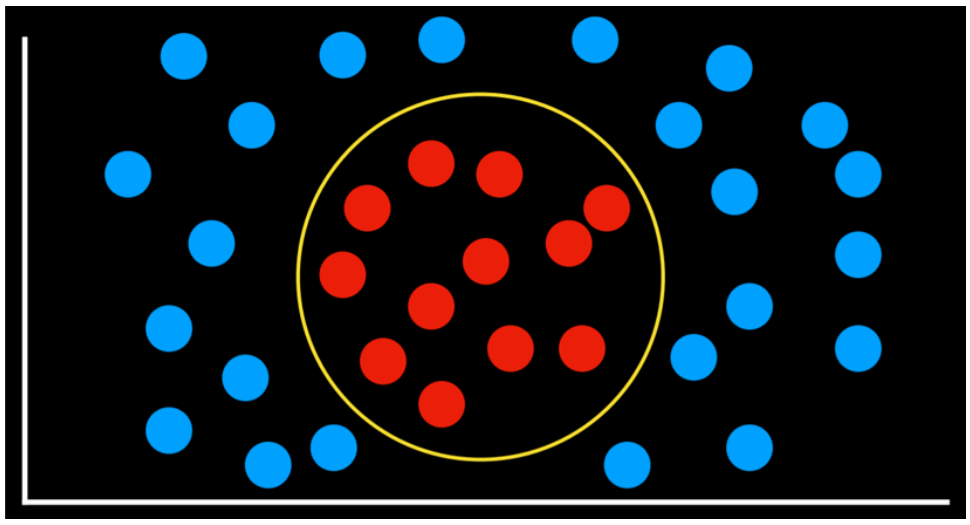
这种操作可以使用任意数量的输入和输出完成，其中每个输入连接到每个输出，而输出代表我们可以做出的决策。请注意，在这种类型的神经网络中，输出不连接。这意味着每个输出及其与所有输入关联的权重可以被视为独立的神经网络，因此可以单独训练，与其余输出分开。

即输出之间没有联系

So far, our neural networks relied on **perceptron** output units. These are units that are only capable of learning a linear decision boundary, using a straight line to separate data. That is, based on a linear equation, the perceptron could classify an input to be one type or another (e.g. left picture). However, often, data are not linearly separable (e.g. right picture). In this case, we turn to multilayer neural networks to model data non-linearly.

到目前为止，我们的神经网络依赖于感知器输出单元。这些单元只能学习线性决策边界，使用直线来分离数据。也就是说，基于线性方程，感知器可以将输入分类为一类或另一类（例如，左侧图片）。然而，通常情况下，数据是不可线性分离的（例如，右侧图片）。在这种情况下，我们转向多层神经网络来非线性地建模数据。





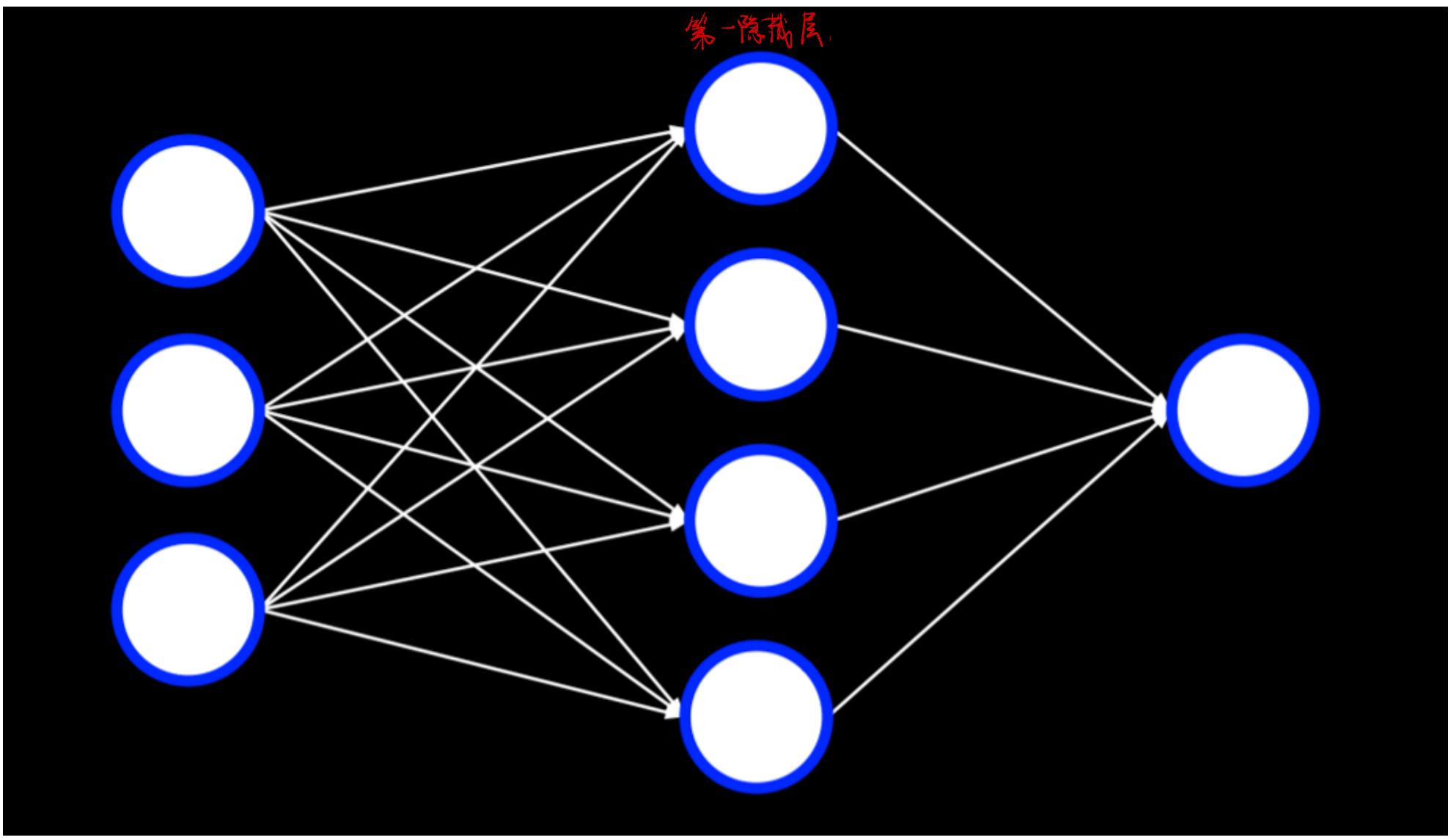
Multilayer Neural Networks

多层神经网络

A multilayer neural network is an artificial neural network with an input layer, an output layer, and at least one **hidden** layer. While we provide inputs and outputs to train the model, we, the humans, don't provide any values to the units inside the hidden layers. Each unit in the first hidden layer receives a weighted value from each of the units in the input layer, performs some action on it and outputs a value. Each of these values is weighted and further propagated to the next layer, repeating the process until the output layer is reached. Through hidden layers, it is possible to model non-linear data.

多层神经网络是一种人工神经网络，包含输入层、输出层和至少一个隐藏层。虽然我们提供输入和输出来训练模型，但人类不会向隐藏层内的单元提供任何值。第一隐藏层中的每个单元从输入层的每个单元接收加权值，对其进行一些操作并输出一个值。这些值中的每一个都被加权，并进一步传递到下一层，重复此过程直到达到输出层。通过隐藏层，可以建模非线性数据。

作用：线性→非线性



Backpropagation 反向传播

Backpropagation is the main algorithm used for training neural networks with hidden layers. It does so by starting with the errors in the output units, calculating the gradient descent for the weights of the previous layer, and repeating the process until the input layer is reached. In pseudocode, we can describe the algorithm as follows:

反向传播是用于训练具有隐藏层的神经网络的主要算法。它通过从输出单元的误差开始，计算前一层的权重的梯度下降，然后重复此过程直到达到输入层。在伪代码中，我们可以将算法描述如下：

- Calculate error for output layer

  ①计算输出层的误差

- For each layer, starting with output layer and moving inwards towards earliest hidden layer:

  ②对于每一层，从输出层开始，向最早隐藏层内部移动：

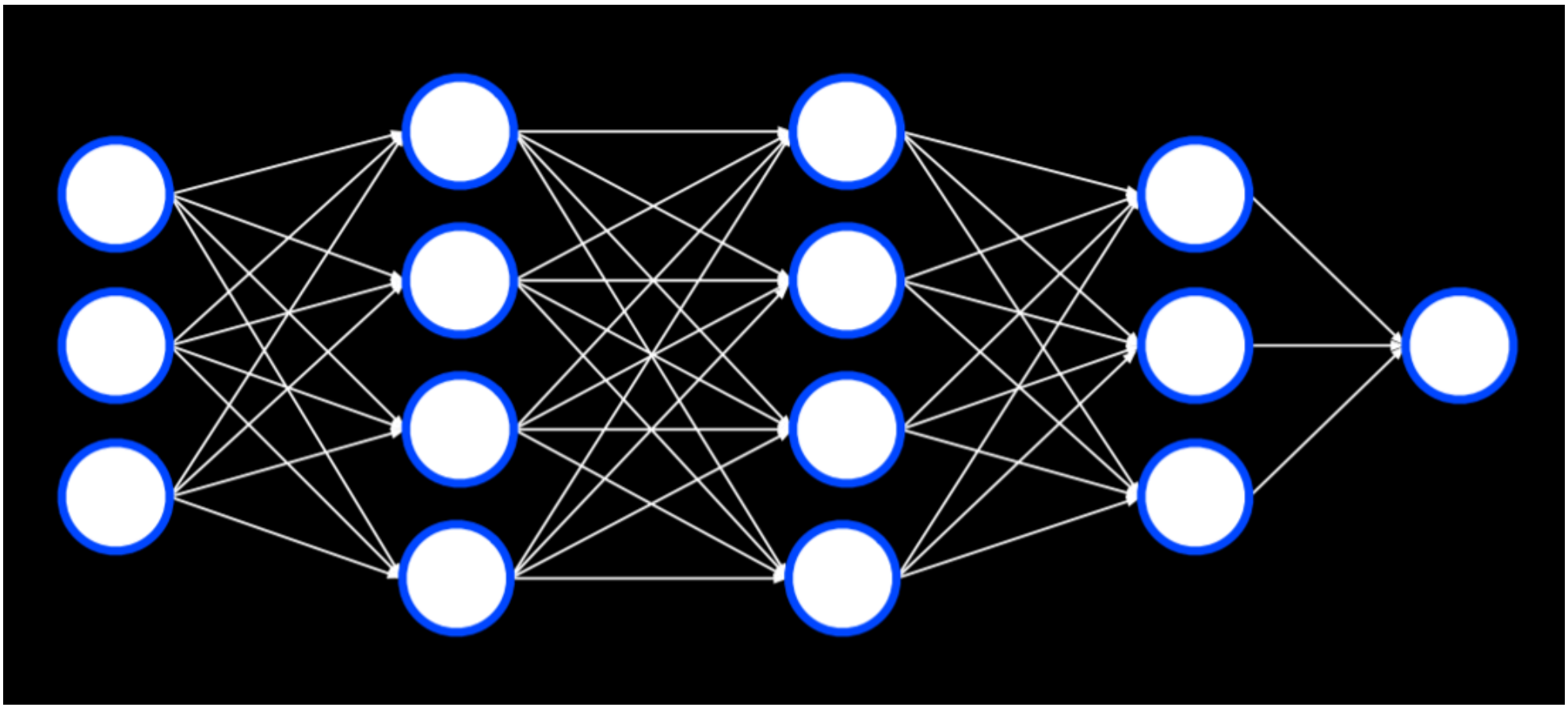
  - Propagate error back one layer. In other words, the current layer that's being considered sends the errors to the preceding layer.

    传播误差到上一层。换句话说，当前正在考虑的层将误差发送到前一层。

  - Update weights. 更新权重

This can be extended to any number of hidden layers, creating **deep neural networks**, which are neural networks that have more than one hidden layer.

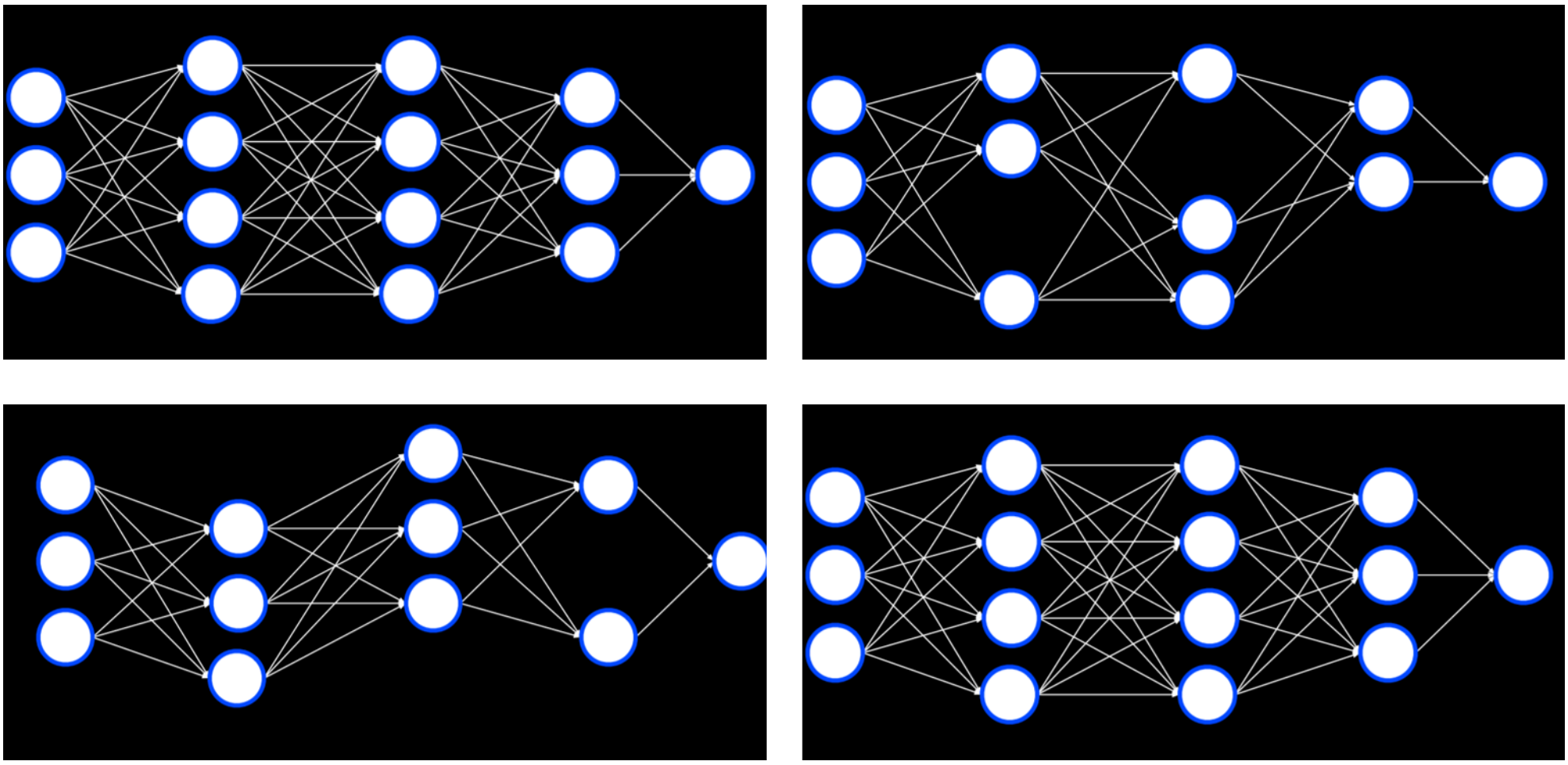
这可以扩展到任何数量的隐藏层，创建深度神经网络，这些神经网络具有多于一层的隐藏层。



Overfitting 过拟合

Overfitting is the danger of modeling the training data too closely, thus failing to generalize to new data. One way to combat overfitting is by **dropout**. In this technique, we temporarily remove units that we select at random during the learning phase. This way, we try to prevent over-reliance on any one unit in the network. Throughout training, the neural network will assume different forms, each time dropping some other units and then using them again:

过拟合是模型过于紧密地拟合训练数据的危险，从而无法泛化到新数据。对抗过拟合的一种方法是 dropout。在这一技术中，我们在学习阶段随机选择并暂时移除一些单元。这样，我们试图防止网络过于依赖任何一个单元。在整个训练过程中，神经网络会采取不同的形式，每次都会移除一些其他单元，然后再次使用它们：



Note that after the training is finished, the whole neural network will be used again.

请注意，训练完成后，整个神经网络将再次被使用。

TensorFlow

Like often is the case in python, multiple libraries already have an implementation for neural networks using the backpropagation algorithm, and TensorFlow is one such library. You are welcome to experiment with TensorFlow neural networks in this web application (http://playground.tensorflow.org/), which lets you define different properties of the network and run it, visualizing the output. We will now turn to an example of how we can use TensorFlow to perform the task we discussed last lecture: distinguishing counterfeit notes from genuine notes.

如 Python 中经常发生的情况一样，多个库已经实现了使用反向传播算法的神经网络，TensorFlow 就是这样的一个库。您可以在这个网络应用中尝试使用 TensorFlow 的神经网络，它允许您定义网络的不同属性并运行它，可视化输出。我们现在将转向一个示例，展示我们如何使用 TensorFlow 来完成上一节课讨论的任务：区分假钞与真钞。

```
import csv
import tensorflow as tf
from sklearn.model_selection import train_test_split
```

We import TensorFlow and call it tf (to make the code shorter).

我们导入 TensorFlow 并将其命名为 tf（以使代码更短）。

```
# Read data in from file
with open("banknotes.csv") as f:
    reader = csv.reader(f)
    next(reader)

    data = []
    for row in reader:
        data.append({
            "evidence": [float(cell) for cell in row[:4]],
            "label": 1 if row[4] == "0" else 0
        })

# Separate data into training and testing groups
evidence = [row["evidence"] for row in data]
labels = [row["label"] for row in data]
X_training, X_testing, y_training, y_testing = train_test_split(
    evidence, labels, test_size=0.4
)
```

- with open("banknotes.csv"): 以读取模式打开CSV文件。
- csv.reader(f): 使用csv.reader读取文件内容，reader是一个迭代器。
- next(reader): 跳过文件的第一行（通常是表头），因为表头不包含数据。
- data.append(...): 对每一行，将前四列作为特征（evidence），第五列作为标签（label），并添加到data列表。
- [float(cell) for cell in row[:4]]: 把每一行的前四个值转换为浮点数，作为机器学习模型的输入特征。
- "label": 1 if row[4] == "0" else 0: 将标签转换为0或1，用于分类任务。
- evidence: 从data中提取所有行的evidence值，构成模型的特征矩阵。
- labels: 从data中提取所有行的label值，构成标签列表。
- train_test_split: 这是一个来自sklearn.model_selection模块的函数，用于随机将数据分为训练集和测试集。
- test_size=0.4: 表示将40%的数据作为测试集，60%的数据作为训练集。
- X_training 和 y_training: 分别是训练集的特征和标签。
- X_testing 和 y_testing: 分别是测试集的特征和标签。

We provide the CSV data to the model. Our work is often required in making the data fit the format that the library requires. The difficult part of actually coding the model is already implemented for us.

我们向模型提供 CSV 数据。我们的工作经常需要使数据适应库所需的形式。实际编码模型的困难部分已经为我们实现。

```
# Create a neural network
```

```
model = tf.keras.models.Sequential()
```

Keras is an api that different machine learning algorithms access. A sequential model is one where layers follow each other (like the ones we have seen so far).

Keras 是一个 API，不同的机器学习算法访问该 API。序列模型是指层依次相随（如同我们迄今为止所看到的那样）。

```
# Add a hidden layer with 8 units, with ReLU activation
model.add(tf.keras.layers.Dense(8, input_shape=(4,), activation="relu"))
```

A dense layer is one where each node in the current layer is connected to all the nodes from the previous layer. In generating our hidden layers we create 8 dense layers, each having 4 input neurons, using the ReLU activation function mentioned above.

密集层是指当前层中的每个节点都与前一层的所有节点相连。在生成隐藏层时，我们创建了 8 个密集层，每个层包含 4 个输入神经元，使用了上述提到的 ReLU 激活函数。

```
# Add output layer with 1 unit, with sigmoid activation
model.add(tf.keras.layers.Dense(1, activation="sigmoid"))
```

In our output layer, we want to create one dense layer that uses a sigmoid activation function, an activation function where the output is a value between 0 and 1.

在我们的输出层中，我们想要创建一个密集层，该层使用 Sigmoid 激活函数，这是一种输出值在 0 和 1 之间的激活函数。

```
# Train neural network
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)
model.fit(X_training, y_training, epochs=20)

# Evaluate how well model performs
model.evaluate(X_testing, y_testing, verbose=2)
```

Finally, we compile the model, specifying which algorithm should optimize it, what type of loss function we use, and how we want to measure its success (in our case, we are interested in the accuracy of the output). Finally, we fit the model on the training data with 20 repetitions (epochs), and then evaluate it on the testing data.

最终，我们编译模型，指定用于优化它的算法，使用的损失函数类型，以及我们如何衡量其成功（在我们的情况下，我们对输出的准确性感兴趣）。最后，我们使用 20 次重复（周期）在训练数据上拟合模型，然后在测试数据上评估它。

Computer Vision 计算机视觉

Computer vision encompasses the different computational methods for analyzing and understanding digital images, and it is often achieved using neural networks. For example, computer vision is used when social media employs face recognition to automatically tag people in pictures. Other examples are handwriting recognition and self-driving cars.

计算机视觉涵盖了分析和理解数字图像的不同计算方法，通常使用神经网络来实现。例如，当社交媒体使用面部识别自动标记照片中的人时，就会用到计算机视觉。其他例子包括手写识别和自动驾驶汽车。

Images consist of pixels, and pixels are represented by three values that range from 0 to 255, one for red, one for green and one for blue. These values are often referred to with the acronym RGB. We can use this to create a neural network where each color value in each pixel is an input, where we have some hidden layers, and the output is some number of units that tell us what it is that was shown in the image. However, there are a few drawbacks to this approach. First, by breaking down the image into pixels and the values of their colors, we can't use the structure of the image as an aid. That is, as humans, if we see a part of a face we know to expect to see the rest of the face, and this quickens computation. We want to be able to use a similar advantage in our neural networks. Second, the sheer number of inputs is very big, which means that we will have to calculate a lot of weights.

图像由像素组成，每个像素由三个从 0 到 255 的值表示，分别对应红色、绿色和蓝色。这些值通常用 RGB 这个缩写来指代。我们可以使用这种方法创建一个神经网络，其中每个像素的每个颜色值都是输入，我们有一些隐藏层，输出是一些告诉我们图像中显示的是什么的单位。然而，这种方法有一些缺点。首先，通过将图像分解为像素及其颜色值，我们无法利用图像的结构作为辅助。也就是说，作为人类，如果我们看到面部的一部分，我们知道会期待看到面部的其余部分，这会加快计算。我们希望在我们的神经网络中能够利用类似的优势。其次，输入的数量非常大，这意味着我们将不得不计算大量的权重。

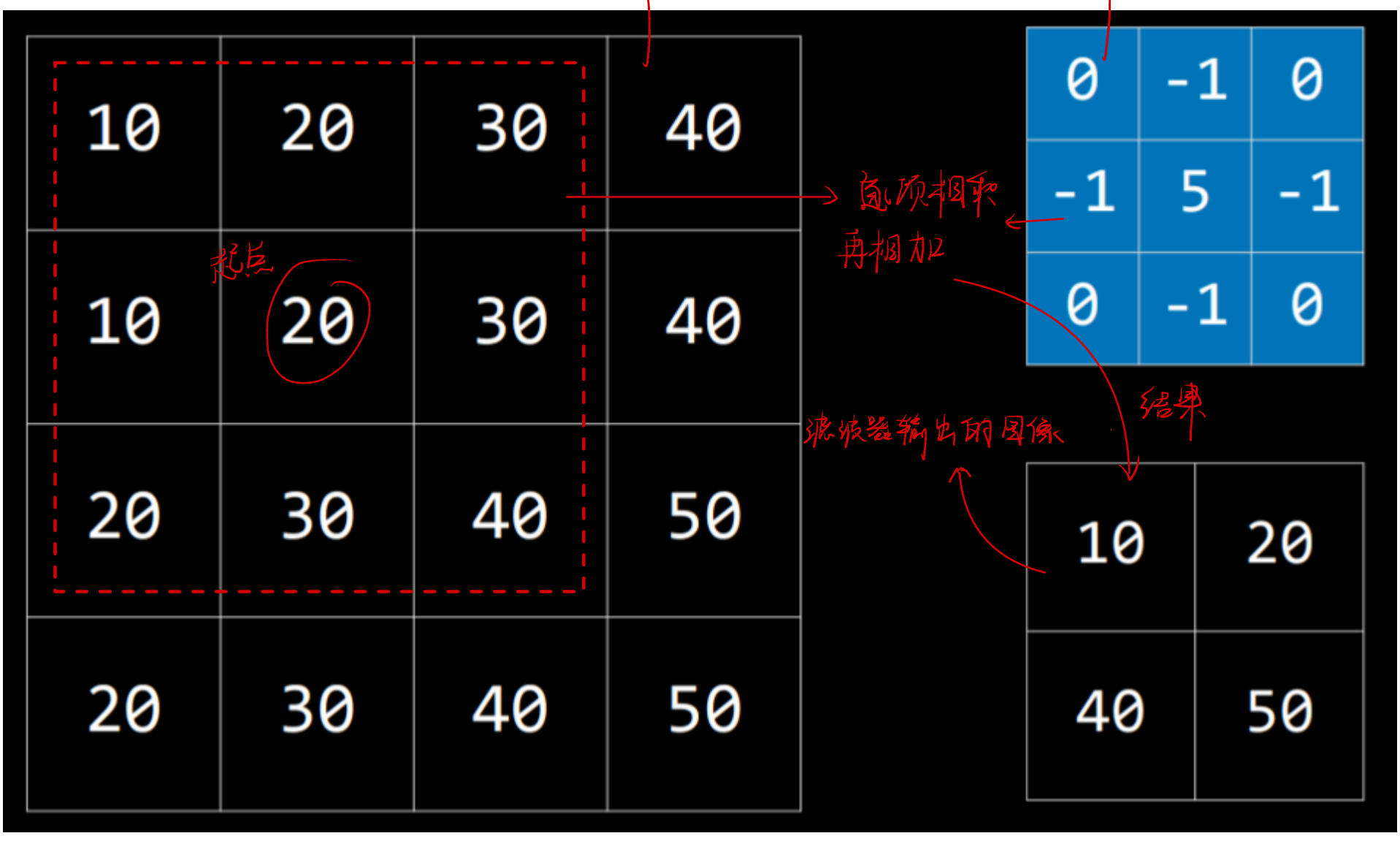
Image Convolution 图像卷积

Image convolution is applying a filter that adds each pixel value of an image to its neighbors, weighted according to a kernel matrix. Doing so alters the image and can help the neural network process it.

图像卷积是应用一个滤波器，该滤波器将图像中的每个像素值与其邻居加权相加，加权根据内核矩阵。这样做会改变图像，并有助于神经网络处理它。

Let's consider the following example:

让我们考虑以下示例： Translated Text:

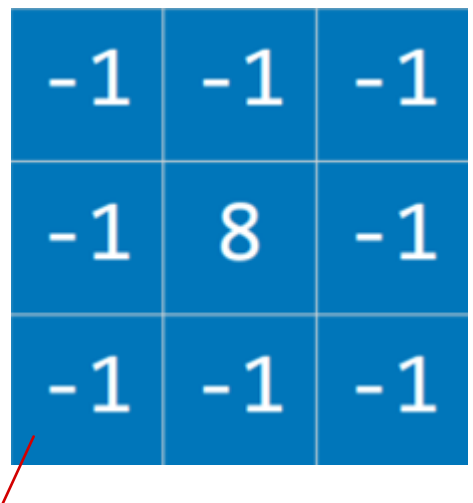


The kernel is the blue matrix, and the image is the big matrix on the left. The resulting filtered image is the small matrix on the bottom right. To filter the image with the kernel, we start with the pixel with value 20 in the top-left of the image (coordinates 1,1). Then, we will multiply all the values around it by the corresponding value in the kernel and sum them up (10*0 + 20*(-1) + 30*0 + 10*(-1) + 20*5 + 30*(-1) + 20*0 + 30*(-1) + 40*0), producing the value 10. Then we will do the same for the pixel on the right (30), the pixel below the first one (30), and the pixel to the right of this one (40). This produces a filtered image with the values we see on the bottom right.

内核是蓝色的矩阵，图像在左侧的大矩阵。得到的过滤后图像在右下角的小矩阵。使用内核过滤图像时，我们从图像左上角值为 20 的像素（坐标 1,1）开始。然后，我们将它周围的所有值乘以内核中相应的值并求和（10*0 + 20*(-1) + 30*0 + 10*(-1) + 20*5 + 30*(-1) + 20*0 + 30*(-1) + 40*0），产生值 10。然后我们对右侧的像素（值为 30）重复同样的操作，第一个像素下方的像素（值为 30），以及这个像素右侧的像素（值为 40）。这产生了一个过滤后的图像，我们看到的值在右下角。

Different kernels can achieve different tasks. For edge detection, the following kernel is often used:

不同的内核可以完成不同的任务。对于边缘检测，通常使用以下内核： Translation:



The idea here is that when the pixel is similar to all its neighbors, they should cancel each other, giving a value of 0. Therefore, the more similar the pixels, the darker the part of the image, and the more different they are the lighter it is. Applying this kernel to an image (left) results in an image with pronounced edges (right):

这里的概念是，当像素与其所有邻居相似时，它们应该相互抵消，给出一个值为 0。因此，像素越相似，图像的这部分就越暗，差异越大，就越亮。将这个内核应用到一张图片（左）上，结果会得到一张有明显边缘的图片（右）：

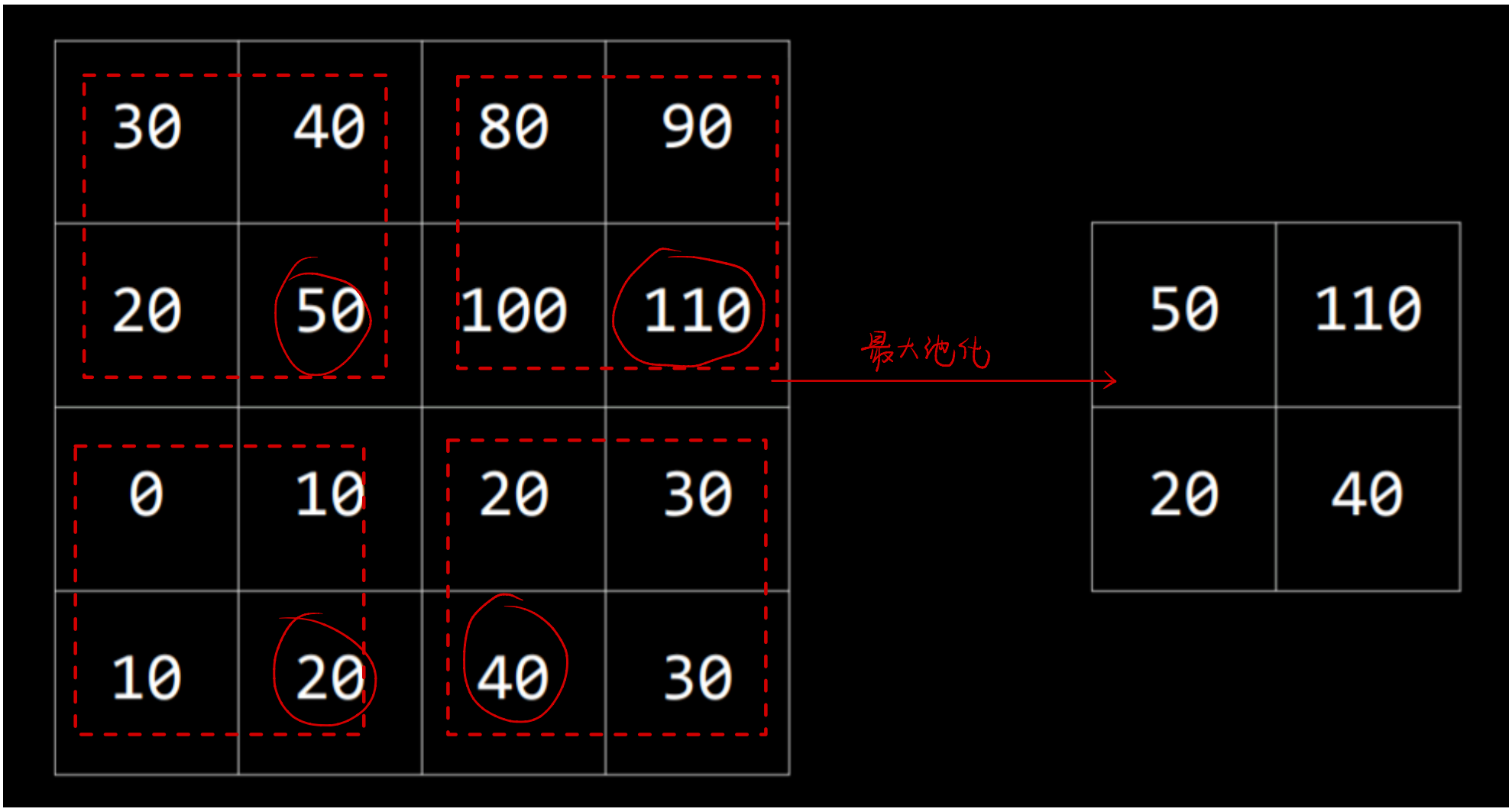
Let's consider an implementation of image convolution. We are using the PIL library (stands for Python Imaging Library) that can do most of the hard work for us.

让我们考虑一下图像卷积的实现。我们使用 PIL 库（代表 Python 图像库），它可以为我们完成大部分繁重的工作。

```
import math
import sys

from PIL import Image, ImageFilter

# Ensure correct usage
if len(sys.argv) != 2:
    sys.exit("Usage: python filter.py filename")

# Open image
image = Image.open(sys.argv[1]).convert("RGB")

# Filter image according to edge detection kernel
filtered = image.filter(ImageFilter.Kernel(
    size=(3, 3),
    kernel=[-1, -1, -1, -1, 8, -1, -1, -1, -1],
    scale=1
))

# Show resulting image
filtered.show()
```

Still, processing the image in a neural network is computationally expensive due to the number of pixels that serve as input to the neural network. Another way to go about this is **Pooling**, where the size of the input is reduced by sampling from regions in the input. Pixels that are next to each other belong to the same area in the image, which means that they are likely to be similar. Therefore, we can take one pixel to represent a whole area. One way of doing this is with **Max-Pooling**, where the selected pixel is the one with the highest value of all others in the same region. For example, if we divide the left square (below) into four 2X2 squares, by max-pooling from this input, we get the small square on the right.

尽管如此，将图像处理通过神经网络在计算上是昂贵的，因为作为神经网络输入的像素的数量。另一种方法是池化，通过从输入中采样来减少输入的大小。相邻的像素属于图像中的同一区域，这意味着它们很可能相似。因此，我们可以用一个像素来代表整个区域。实现这一目标的一种方式是最大池化，其中所选的像素是同一区域中所有其他像素中值最高的一个。例如，如果我们从左方的正方形（下方）将其分为四个 2X2 的正方形，通过从这个输入进行最大池化，我们得到右侧的小正方形。

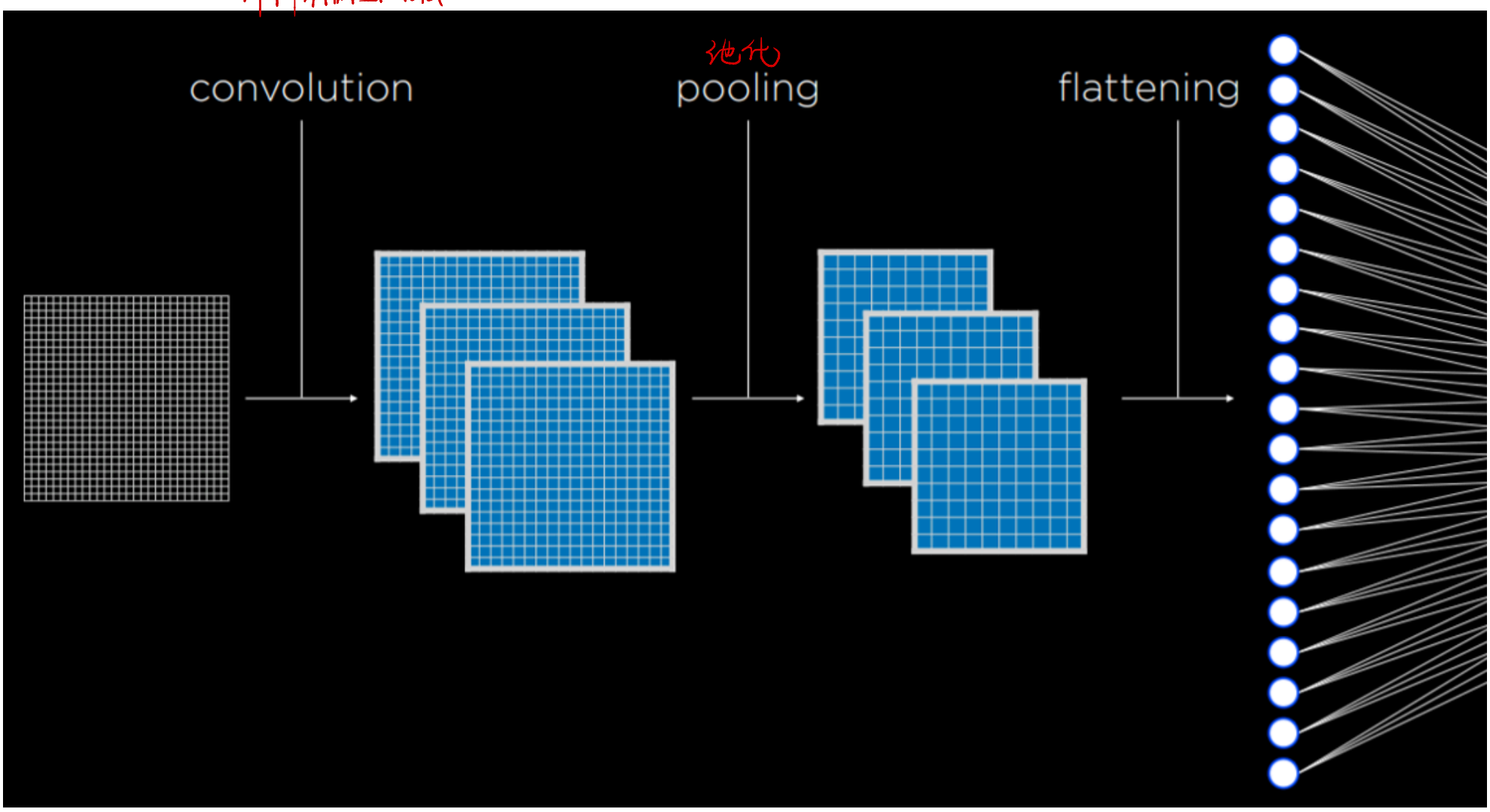


Convolutional Neural Networks

卷积神经网络

A convolutional neural network is a neural network that uses convolution, usually for analyzing images. It starts by applying filters that can help distill some features of the image using different kernels. These filters can be improved in the same way as other weights in the neural network, by adjusting their kernels based on the error of the output. Then, the resulting images are pooled, after which the pixels are fed to a traditional neural network as inputs (a process called **flattening**).

卷积神经网络是一种使用卷积的神经网络，通常用于分析图像。它从应用可以帮助提取图像中某些特征的过滤器开始，使用不同的内核。这些过滤器可以通过调整其内核来改进，就像神经网络中的其他权重一样，根据输出的误差进行调整。然后，对结果图像进行聚合，之后将像素作为输入提供给传统的神经网络（称为扁平化的过程）。



The convolution and pooling steps can be repeated multiple times to extract additional features and reduce the size of the input to the neural network. One of the benefits of these processes is that, by convoluting and pooling, the neural network becomes less sensitive to variation. That is, if the same picture is taken from slightly different angles, the input for convolutional neural network will be similar, whereas, without

convolution and pooling, the input from each image would be vastly different.

卷积和池化步骤可以重复多次，以提取额外特征并减少输入到神经网络的大小。这些过程的一个好处是，通过卷积和池化，神经网络变得对变化不那么敏感。也就是说，如果从略微不同的角度拍摄同一张图片，卷积神经网络的输入会相似，而如果没有卷积和池化，每张图片的输入会大不相同。

In code, a convolutional neural network doesn't differ by much from a traditional neural network. TensorFlow offers datasets to test our models on. We will be using MNIST, which contains pictures of black and white handwritten digits. We will train our convolutional neural network to recognize digits.

在代码中，卷积神经网络与传统神经网络的差异不大。TensorFlow 提供了数据集供我们测试模型。我们将使用 MNIST，其中包含黑白手写数字的图片。我们将训练我们的卷积神经网络来识别数字。

```
import sys
import tensorflow as tf

# Use MNIST handwriting dataset
mnist = tf.keras.datasets.mnist

# Prepare data for training
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)
x_train = x_train.reshape(
    x_train.shape[0], x_train.shape[1], x_train.shape[2], 1
)
x_test = x_test.reshape(
    x_test.shape[0], x_test.shape[1], x_test.shape[2], 1
)

# Create a convolutional neural network
model = tf.keras.models.Sequential([

    # Convolutional layer. Learn 32 filters using a 3x3 kernel
    tf.keras.layers.Conv2D(
        32, (3, 3), activation="relu", input_shape=(28, 28, 1)
    ),

    # Max-pooling layer, using 2x2 pool size
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),

    # Flatten units
    tf.keras.layers.Flatten(),

    # Add a hidden layer with dropout
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.5),

    # Add an output layer with output units for all 10 digits
    tf.keras.layers.Dense(10, activation="softmax")
])

# Train neural network
model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
model.fit(x_train, y_train, epochs=10)

# Evaluate neural network performance
model.evaluate(x_test,  y_test, verbose=2)
```

Since the model takes time to train, we can save the already trained model to use it later.

由于模型需要时间进行训练，我们可以保存已经训练好的模型以便日后使用。

```
# Save model to file
if len(sys.argv) == 2:
    filename = sys.argv[1]
    model.save(filename)
    print(f"Model saved to {filename}.")
```

Now, if we run a program that receives hand-drawn digits as input, it will be able to classify and output the digit using the model. For an implementation of such a program, refer to recognition.py in the source code for this lecture.

现在，如果我们运行一个接收手绘数字作为输入的程序，它将能够使用模型对数字进行分类并输出。对于此类程序的实现，请参阅本讲义的源代码中的 recognition.py。

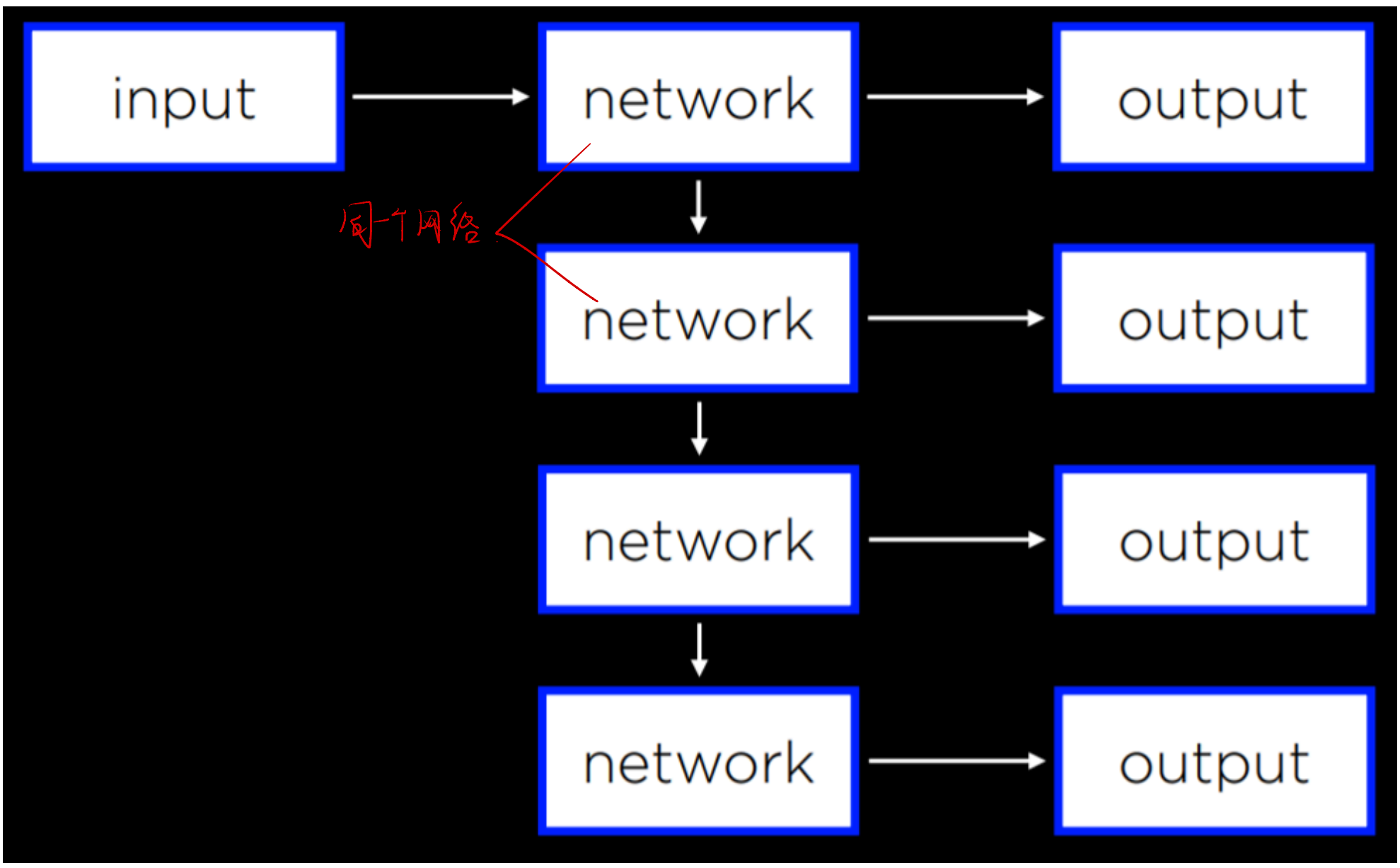
Recurrent Neural Networks
循环神经网络

Feed-Forward Neural Networks are the type of neural networks that we have discussed so far, where input data is provided to the network, which eventually produces some output. A diagram of how feed-forward neural networks work can be seen below.

前文中我们讨论过的神经网络类型是前馈神经网络，其中输入数据被提供给网络，最终产生一些输出。前馈神经网络的工作原理的示意图如下所示。

As opposed to that, **Recurrent Neural Networks** consist of a non-linear structure, where the network uses its own output as input. For example, Microsoft's captionbot (https://www.captionbot.ai) is capable of describing the content of an image with words in a sentence. This is different from classification in that the output can be of varying length based on the properties of the image. While feed-forward neural networks are incapable of varying the number of outputs, recurrent neural networks are capable to do that due to their structure. In the captioning task, a network would process the input to produce an output, and then continue processing from that point on, producing another output, and repeating as much as necessary.

相反，循环神经网络由非线性结构组成，网络使用自己的输出作为输入。例如，微软的字幕机器人能够用句子中的单词描述图片的内容。这与分类不同，输出的长度可以根据图片的特性而变化。而前馈神经网络无法改变输出的数量，由于其结构，循环神经网络能够做到这一点。在字幕任务中，网络会处理输入以产生输出，然后从这一点继续处理，产生另一个输出，并重复这一过程，直到必要为止。



Recurrent neural networks are helpful in cases where the network deals with sequences and not a single individual object. Above, the neural network needed to produce a sequence of words. However, the same principle can be applied to analyzing video files, which consist of a sequence of images, or in translation tasks, where a sequence of inputs (words in the source language) is processed to produce a sequence of outputs (words in the target language).

循环神经网络在处理序列而非单个个体时非常有用。上面，神经网络需要生成一系列单词。然而，同样的原理可以应用于分析视频文件，这些文件由一系列图像组成，或者在翻译任务中，处理一系列输入（源语言的单词）以生成一系列输出（目标语言的单词）。