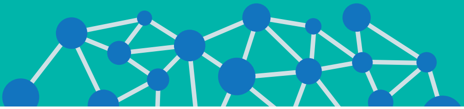


II. How neural networks are built

As we said earlier, neurons are very simple processing units. Having discussed linear and logistic regression in Chapter 4, the essential technical details of neural networks can be seen as slight variations of the same idea.

Note

Weights and inputs

The basic artificial neuron model involves a set of adaptive parameters, called weights like in linear and logistic regression. Just like in regression, these weights are used as multipliers on the inputs of the neuron, which are added up. The sum of the weights times the inputs is called the linear combination of the inputs. You can probably recall the shopping bill analogy: you multiply the amount of each item by its price per unit and add up to get the total.

If we have a neuron with six inputs (analogous to the amounts of the six shopping items: potatoes, carrots, and so on), input1, input2, input3, input4, input5, and input6, we also need six weights. The weights are analogous to the prices of the items. We'll call them weight1, weight2, weight3, weight4, weight5, and weight6. In addition, we'll usually want to include an intercept term like we did in linear regression. This can be thought of as a fixed additional charge due to processing a credit card payment, for example.

We can then calculate the linear combination like this: linear combination = intercept + weight1 × input1 + ... + weight6 × input6 (where the ... is a shorthand notation meaning that the sum include all the terms from 1 to 6).

With some example numbers we could then get:

```
10.0 + 5.4 × 8 + (-10.2) × 5 + (-0.1) × 22 + 101.4 ×
(-5) + 0.0 × 2 + 12.0 × (-3) = -543.0
```

Unanswered

Exercise 21: Weights and inputs

In this exercise, consider the following expression that has both weights and inputs:

10.0 + 5.4 × 8 + (-10.2) × 5 + (-0.1) × 22 + 101.4 × (-5) + 0.0 × 2 + 12.0 × (-3) = -543.0

What is the intercept term in the expression?

a) 543.0
b) 10.0
c) -3
d) 5.4?

A B C D

What are the inputs?

a) 8, 5, 22, -5, 2, -3
b) 5.4, 8, -10.2, 5, -0.1, 22, 101.4, -5, 0.0, 2, 12.0, -3
c) 5.4, -10.2, -0.1, 101.4, 0.0, 12.0
d) 43.2, -51.0, -2.2, -507.0, 0.0, -36.0

A B C D

Which of the inputs needs to be changed the least to increase the output by a certain amount?

a) first
b) second
c) third
d) fourth

A B C D

What happens when the fifth input is incremented by one?

a) nothing
b) the output increases by one
c) the output increases by two
d) something else

A B C D

Submit

The weights are almost always learned from data using the same ideas as in linear or logistic regression, as discussed previously. But before we discuss this in more detail, we'll introduce another important stage that a neuron completes before it sends out an output signal.

Activations and outputs

Once the linear combination has been computed, the neuron does one more operation. It takes the linear combination and puts it through a so-called activation function. Typical examples of the activation function include:

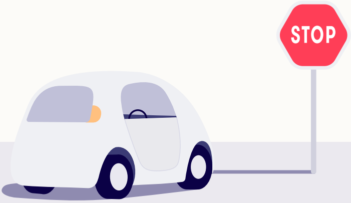
- identity function: do nothing and just output the linear combination
- step function: if the value of the linear combination is greater than zero, send a pulse (ON), otherwise do nothing (OFF)
- sigmoid function: a "soft" version of the step function

Note that with the first activation function, the identity function, the neuron is exactly the same as linear regression. This is why the identity function is rarely used in neural networks: it leads to nothing new and interesting.

Note

How neurons activate

Real, biological neurons communicate by sending out sharp, electrical pulses called “spikes”, so that at any given time, their outgoing signal is either on or off (1 or 0). The step function imitates this behavior. However, artificial neural networks tend to use activation functions that output a continuous numerical activation level at all times, such as the sigmoid function. Thus, to use a somewhat awkward figure of speech, real neurons communicate by something similar to the Morse code, whereas artificial neurons communicate by adjusting the pitch of their voice as if yodeling.



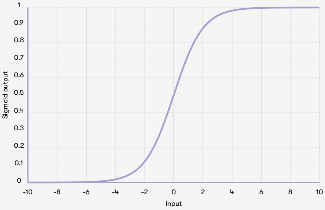
The output of the neuron, determined by the linear combination and the activation function, can be used to extract a prediction or a decision. For example, if the network is designed to identify a stop sign in front of a self-driving car, the input can be the pixels of an image captured by a camera attached in front of the car, and the output can be used to activate a stopping procedure that stops the car before the sign.

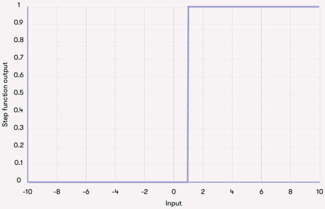
Learning or adaptation in the network occurs when the weights are adjusted so as to make the network produce the correct outputs, just like in linear or logistic regression. Many neural networks are very large, and the largest contain hundreds of billions of weights. Optimizing them all can be a daunting task that requires massive amounts of computing power.

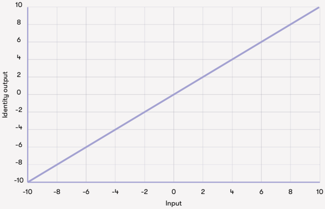
Unanswered

Exercise 22: Activations and outputs

Below are graphs for three different activation functions with different properties. First we have the sigmoid function, then the step function, and finally the identity function. IMPORTANT: Note the different y-axis (vertical) scale in the identity function chart.







Which of the activations described above gives: the highest output for an input of 5?	Sigmoid	Step	Identity
the lowest output for an input of -5?	Sigmoid	Step	Identity
the highest output for an input of -2.5?	Sigmoid	Step	Identity

Submit

Perceptron: the mother of all ANNs

The perceptron is simply a fancy name for the simple neuron model with the step activation function we discussed above. It was among the very first formal models of neural computation and because of its fundamental role in the history of neural networks, it wouldn't be unfair to call it the "mother of all artificial neural networks".

It can be used as a simple classifier in binary classification tasks. A method for learning the weights of the perceptron from data, called the Perceptron algorithm, was introduced by the psychologist Frank Rosenblatt in 1957. We will not study the Perceptron algorithm in detail. Suffice to say that it is just about as simple as the nearest neighbor classifier. The basic principle is to feed the network training data one example at a time. Each misclassification leads to an update in the weight.

Note

AI hyperbole

After its discovery, the Perceptron algorithm received a lot of attention, not least because of optimistic statements made by its inventor, Frank Rosenblatt. A classic example of AI hyperbole is a New York Times article published on July 8th, 1958:
"The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, reproduce itself and be conscious of its existence."

Please note that neural network enthusiasts are not at all the only ones inclined towards optimism. The rise and fall of the logic-based expert systems approach to AI had all the same hallmark features of an AI-hype and people claimed that the final breakthrough is just a short while away. The outcome both in the early 1960s and late 1980s was a collapse in the research funding called an AI Winter.

The history of the debate that eventually lead to almost complete abandoning of the neural network approach in the 1960s for more than two decades is extremely fascinating. The article *A Sociological Study of the Official History of the Perceptrons Controversy* by Mikel Olazaran (published in *Social Studies of Science*, 1996) reviews the events from a sociology of science point of view. Reading it today is quite thought provoking. Reading stories about celebrated AI heroes who had developed neural networks algorithms that would soon reach the level of human intelligence and become self-conscious can be compared to some statements made during the current hype. If you take a look at the above article, even if you wouldn't read all of it, it will provide an interesting background to today's news. Consider for example an article in the MIT Technology Review published in September 2017, where Jordan Jacobs, co-founder of a multimillion dollar Vector institute for AI compares Geoffrey Hinton (a figure-head of the current deep learning boom) to Einstein because of his contributions to development of neural network algorithms in the 1980s and later. Also recall the Human Brain project mentioned in the previous section.

According to Hinton, "the fact that it doesn't work is just a temporary annoyance" (although according to the article, Hinton is laughing about the above statement, so it's hard to tell how serious he is about it). The Human Brain project claims to be "close to a profound leap in our understanding of consciousness". Doesn't that sound familiar?

No-one really knows the future with certainty, but knowing the track record of earlier announcements of imminent breakthroughs, some critical thinking is advised. We'll return to the future of AI in the final chapter, but for now, let's see how artificial neural networks are built.

Putting neurons together: networks

A single neuron would be way too simple to make decisions and prediction reliably in most real-life applications. To unleash the full potential of neural networks, we can use the output of one neuron as the input of other neurons, whose outputs can be the input to yet other neurons, and so on. The output of the whole network is obtained as the output of a certain subset of the neurons, which are called the output layer. We'll return to this in a bit, after we discussed the way neural networks adapt to produce different behaviors by learning their parameters from data.

Key terminology

Layers

Often the network architecture is composed of layers. The input layer consists of neurons that get their inputs directly from the data. So for example, in an image recognition task, the input layer would use the pixel values of the input image as the inputs of the input layer. The network typically also has hidden layers that use the other neurons´ outputs as their input, and whose output is used as the input to other layers of neurons. Finally, the output layer produces the output of the whole network. All the neurons on a given layer get inputs from neurons on the previous layer and feed their output to the next.

A classical example of a multilayer network is the so-called multilayer perceptron. As we discussed above, Rosenblatt's Perceptron algorithm can be used to learn the weights of a perceptron. For multilayer perceptron, the corresponding learning problem is way harder and it took a long time before a working solution was discovered. But eventually, one was invented: the backpropagation algorithm lead to a revival of neural networks in the late 1980s. It is still at the heart of many of the most advanced deep learning solutions.

Note

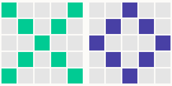
Meanwhile in Helsinki...

The path(s) leading to the backpropagation algorithm are rather long and winding. An interesting part of the history is related to the computer science department of the University of Helsinki. About three years after the founding of the department in 1967, a Master's thesis was written by a student called Seppo Linnainmaa. The topic of the thesis was "Cumulative rounding error of algorithms as a Taylor approximation of individual rounding errors" (the thesis was written in Finnish, so this is a translation of the actual title "Algoritmin kumulatiivinen pyöristysvirhe yksittäisten pyöristysvirheiden Taylor-kehitelmänä").

The automatic differentiation method developed in the thesis was later applied by other researchers to quantify the sensitivity of the output of a multilayer neural network with respect to the individual weights, which is the key idea in backpropagation.

A simple neural network classifier

To give a relatively simple example of using a neural network classifier, we'll consider a task that is very similar to the MNIST digit recognition task, namely classifying images in two classes. We will first create a classifier to classify whether an image shows a cross (x) or a circle (o). Our images are represented here as pixels that are either colored or white, and the pixels are arranged in 5 × 5 grid. In this format our images of a cross and a circle (more like a diamond, to be honest) look like this:



In order to build a neural network classifier, we need to formalize the problem in a way where we can solve it using the methods we have learned. Our first step is to represent the information in the pixels by numerical values that can be used as the input to a classifier. Let's use 1 if the square is colored, and 0 if it is white. Note that although the symbols in the above graphic are of different color (green and blue), our classifier will ignore the color information and use only the colored/white information. The 25 pixels in the image make the inputs of our classifier.

To make sure that we know which pixel is which in the numerical representation, we can decide to list the pixels in the same order as you'd read text, so row by row from the top, and reading each row from left to right. The first row of the cross, for example, is represented as 1,0,0,0,1; the second row as 0,1,0,1,0, and so on. The full input for the cross input is then: 1,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,1.

We'll use the basic neuron model where the first step is to compute a linear combination of the inputs. Thus need a weight for each of the input pixels, which means 25 weights in total.

Finally, we use the step activation function. If the linear combination is negative, the neuron activation is zero, which we decide to use to signify a cross. If the linear combination is positive, the neuron activation is one, which we decide to signify a circle.

Let's try what happens when all the weights take the same numerical value, 1. With this setup, our linear combination for the cross image will be 9 (9 colored pixels, so 9 × 1, and 16 white pixels, 16 × 0), and for the circle image it will be 8 (8 colored pixels, 8 × 1, and 17 white pixels, 17 × 0). In other words, the linear combination is positive for both images and they are thus classified as circles. Not a very good result given that there are only two images to classify.

To improve the result, we need to adjust the weights in such a way that the linear combination will be negative for a cross and positive for a circle. If we think about what differentiates images of crosses and circles, we can see that circles have no colored pixels in the center of the image, whereas crosses do. Likewise, the pixels at the corners of the image are colored in the cross, but white in the circle.

We can now adjust the weights. There are an infinite number of weights that do the job. For example, assign weight -1 to the center pixel (the 13th pixel), and weight 1 to the pixels in the middle of each of the four sides of the image, letting all the other weights be 0. Now, for the cross input, the center pixel produce the value −1, while for all the other pixels either the pixel value or the weight is 0, so that −1 is also the total value. This leads to activation 0, and the cross is correctly classified.

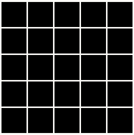
How about the circle then? Each of the pixels in the middle of the sides produces the value 1, which makes 4 × 1 = 4 in total. For all the other pixels either the pixel value or the weight is zero, so 4 is the total. Since 4 is a positive value, the activation is 1, and the circle is correctly recognized as well.

Happy or not?

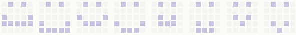
We will now follow similar reasoning to build a classifier for smiley faces. You can assign weights to the input pixels in the image by clicking on them. Clicking once sets the weight to 1, and clicking again sets it to -1. The activation 1 indicates that the image is classified as a happy face, which can be correct or not, while activation –1 indicates that the image is classified as a sad face.

Don't be discouraged by the fact that you will not be able to classify all the smiley faces correctly: it is in fact impossible with our simple classifier! This is one important learning objective: sometimes perfect classification just isn't possible because the classifier is too simple. In this case the simple neuron that uses a linear combination of the inputs is too simple for the task. Observe how you can build classifiers that work well in different cases: some classify most of the happy faces correctly while being worse for sad faces, or the other way around.

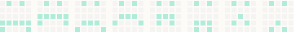
Can you achieve 6/8 correct for both happy and sad faces?



0/8 happy faces classified correctly



0/8 sad faces classified correctly



Next section

III. Advanced neural network techniques →