CISC 271   Class 26

Classification – Single Artificial Neuron

Text Correspondence: Hastie *et al.*, 2009 [6], pp 130–132

Main Concepts:

- *Biological inspiration: cortical nerve cell*
- *Hyperplane with augmented vectors*
- *Neural networks use binary classes 0 and 1*

Sample Problem, Machine Inference: How can we model a neuron?

Historically, the idea of an *artificial neuron* dates to 1943, when McCulloch and Pitts [9] began to produce computational models of basic nerve cells. An illustration of a human neuron is shown in Figure 26.1.

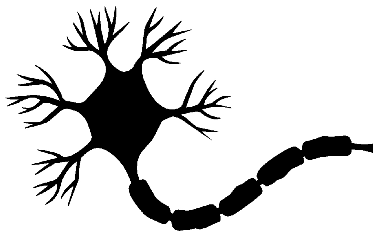


Figure 26.1: Illustration of a human neuron. The small projections from the cell body represent dendrites that provide electrical inputs to the neuron. The long projection represents the axon, which transmits the output to other cells. The axon is wrapped by electrically insulating cells that help to preserve the electrical signal from the neuron.

From early electrophysiology, McCulloch and Pitts observed that some neurons had a behavior that was roughly approximated as a linear sum of the inputs. If the j^{th} electrical input to a neuron was a value a_j, the input appeared to be weighted by a cell-specific value m_j. Taking n such inputs, the linear sum was

$$\sum_{j=1}^{n} a_j m_j = \vec{a} \cdot \vec{m} = \vec{a}^T \vec{m} \tag{26.1}$$

If the weighted sum in Equation 26.1 was greater than a cell-specific threshold value β, the cell would "fire" and produce an output; otherwise, the cell would not have significant electrical activity. The actual electrophysiology is considerably more complicated, involving time-dependent sequences of electrical “spikes”, but this simple mathematical model is still used in many current neural networks. A common terminology in machine learning is that the value β is referred to as the *bias* value. This condition for “firing” an artificial neuron can be written as

$$\sum_{j=1}^{n} a_j m_j \geq \beta \quad \text{or} \quad \sum_{j=1}^{n} a_j m_j - \beta \geq 0 \tag{26.2}$$

26.1 Data Matrix and Label Vector

We can see that Equation 26.2 has a familiar form: it looks much like a decision for whether the vector $\vec{a}$ is on the positive side of a hyperplane. We will use this computation often, so we will *augment* the vector $\vec{a}$ to create a data vector $\vec{x}$ that is defined as

$$\vec{x} \stackrel{\text{def}}{=} \begin{bmatrix} \vec{a} \\ 1 \end{bmatrix} \tag{26.3}$$

We will represent the weight vector for an artificial neuron as the vector $\vec{m}$ augments with the bias scalar $b \equiv -\beta$, which is

$$\vec{w} \stackrel{\text{def}}{=} \begin{bmatrix} \vec{m} \\ b \end{bmatrix} \tag{26.4}$$

Using Definition 26.3 and Definition 26.4, the computation in Equation 26.2 can be written as

$$u = \sum_{j=1}^{n} a_j m_j + b = \begin{bmatrix} \vec{a} & 1 \end{bmatrix} \begin{bmatrix} \vec{m} \\ b \end{bmatrix} = \vec{x}^T \vec{w} \tag{26.5}$$

Usually, we want to *learn* the weights $\vec{w}$ for a set of input data. If each input data vector is $\vec{a}_i$, then we can augment the data and gather the augmented observations into a matrix as

$$X = \begin{bmatrix} \vec{x}_1^T \\ \vec{x}_2^T \\ \vdots \\ \vec{x}_m^T \end{bmatrix} = \begin{bmatrix} A & \vec{1} \end{bmatrix} \tag{26.6}$$

26.1.1 Labels of Data for Artificial Neurons

To learn weight vectors, artificial neurons need each input data vector to have a label. For mathematical convenience, the convention in neural networks is to have each data vector labelled as either 0 or 1. Formally, for the i^{th} augmented data vector $\vec{x}_i$, the label is

$$y_i \in \{0, 1\} \quad \text{and} \quad \vec{y} \overset{\text{def}}{=} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \tag{26.7}$$

In machine learning, we must take care to see how data are labelled. In some literature, binary labels might be ± 1; for neural networks, binary labels are 0 or 1.

26.2 Activation Function

The way that an artificial neuron "fires" is also referred to as its *activation function*, written here as $\phi(\cdot)$. The activation function maps the linear sum u in Equation 26.5 to a real number, which we will call the *score*, as

$$z = \phi(u) \quad \text{where} \quad u \overset{\text{def}}{=} \vec{x}^T \vec{w} \tag{26.8}$$

Depending on how the artificial neuron is specified, the score z in Equation 26.8 might be any real number so $z \in \mathbb{R}$, or the score might be restricted to an interval such as $z \in [0\ 1]$. We need to read carefully when we are studying other source material on this topic.

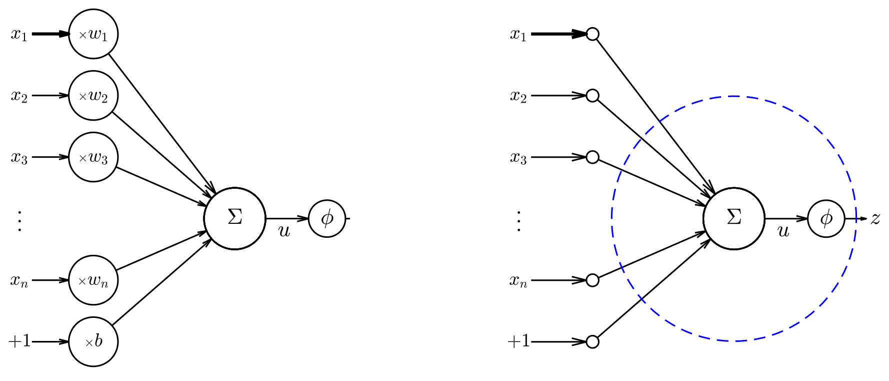
A commonly used diagram illustrates the inputs to an artificial neuron as "arriving" from the left. The values x_i are weighted by the values w_i; the bias b is associated with the augmented constant $x_{n+1} \overset{\text{def}}{=} +1$. The weighted sum of these are the scalar value u, which passes through the activation function $\phi(u)$ to produce an output z on the right of the diagram. Such an artificial neuron is often illustrated as in Figure 26.2.

The score z is a real number that might not be the "final" output of an artificial neuron. In some applications, the output of the artificial neurons must be *quantized* into two distinct classes. Although most of machine learning uses the classes $\{-1, +1\}$, in neural networks the biological inspiration is used so that the quantization is

$$q(z) \in \{0, 1\} \tag{26.9}$$

26.3 Hyperplane For An Artificial Neuron

Consider Equation 26.5, which computes the pseudo-distance of a vector $\vec{x}_i$ to a hyperplane $\mathbb{H}$ that is specified by a general normal vector $\vec{m}$ and a bias scalar b. We might find the notational overhead associated with the bias scalar b to be tedious.



(A) **(B)**

Figure 26.2: Illustrations of an artificial neuron. The input vector $\vec{x}$ is scaled by the weights $w_1, w_2, \ldots, w_n$ and biased by b; the result u is the input to the activation function ϕ that produces the neuron output z. (A) Data flow for an artificial neuron. (B) External variables are input $\vec{x}$ and output z.

In a very simple model of an artificial neuron, we can omit the activation function $\phi(\cdot)$ of Equation 26.8; equivalently, we can use the identity function that maps the input to the output as $\phi(u) = u$. This implies that we can take the pseudo-distance u_i of $\vec{x}_i$ to the hyperplane $\mathbb{H}$, computed with Equation 26.5, and quantize u_i as 0 for a negative pesudo-distance and 1 as a non-negative pseudo-distance.

A common quantization is the *Heaviside* function, named after the mathematician and physicist Oliver Heaviside. This is also call the *unit step* function and it plays a prominent part in neural networks. The Heaviside function is defined as

$$H(u) \overset{\text{def}}{=} \begin{cases} 0 & \text{if} \quad u < 0 \\ 1 & \text{if} \quad u \geq 0 \end{cases} \tag{26.10}$$

Our simple model of an artificial neuron will be that it uses Equation 26.10 to classify an augmented observation vector $\vec{x}_i$ by using the augmented weight vector $\vec{w}$, with the computation

$$q(\vec{w}; \vec{x}_i) = H(\vec{x}_i^T \vec{w}) \tag{26.11}$$

In Equation 26.11, the argument of the quantization function $q(\cdot)$ is the weight vector $\vec{w}$ and the additional parameter is the augmented data vector $\vec{x}_i$.