CISC 271 Class 28

Artificial Neuron – Learning Weights

Text Correspondence: Hastie *et al.*, 2009 [6], pp 119–122 and pp 130–132

Main Concepts:

- *Perceptron – simple binary neuron cell*
- *Steepest Descent – numerical method for learning weights*
- *Linear activation – limited ability to separate*
- *Semilinear – early use of logistic function*

Sample Problem, Machine Inference: How can we learn the "best" binary separation of data?

Our notation for artificial neurons will differ slightly from our notation so far in this course. We will suppose that data are provided in a data matrix $A \in \mathbb{R}^{m \times n}$ and that binary labels are provided as $y_i \in \{0\ 1\}$. We will transpose and augment each observation $\underline{a_i}$ with the value 1; for a hyperplane $\mathbb{H}$ that is specified by a general non-zero normal vector $\vec{m}$ with its bias scalar b, we will augment the vector with the bias scalar. Our augmented data vector, and our augmented weight vector that represents $\mathbb{H}$, will be

$$\underline{x} \stackrel{\text{def}}{=} \begin{bmatrix} \underline{a}^T & 1 \end{bmatrix} \qquad \vec{w} \stackrel{\text{def}}{=} \begin{bmatrix} \vec{m} \\ b \end{bmatrix} \tag{28.1}$$

Previously, we used Equation 25.6 to find the probability that an observation $\underline{x}$ was on the "positive" side of a hyperplane $\mathbb{H}$. For an artificial neuron, we usually do not use a unit normal $\vec{n}$; instead, we are using a general non-zero normal vector $\vec{m}$. The *linear response* of the artificial neuron to the observation $\underline{x}$ is

$$u(\vec{w}; \underline{x}) \stackrel{\text{def}}{=} \underline{x}\vec{w} \tag{28.2}$$

Sometimes, the quantity u in Equation 28.2 is called the *hyperplane pseudo-distance*.

28.1 History – The Perceptron Rule

A basic algorithm for finding the weight vector $\vec{w}$ is the Perceptron Rule. Historically, a Perceptron was one version of an artificial neuron that was developed by Frank Rosenblatt [11] who worked independently from McCulloch and Pitts [9]. His algorithm is the basis of a great deal of current work in machine learning.

In a landmark 1969 book – re-issued in 1988 and 2017 – Marvin Minsky and Seymour Papert [10] proved that entire classes of seemingly simple problems cannot be solved by Perceptron-like models of an artificial neuron. The models included the original Perceptron of Frank Rosenblatt [11], plus the Widrow-Hoff original model [16] and variants [17]. The criticism applied also to a single "layer" of simple artificial neurons.

28.2 Logistic Activation of an Artificial Neuron

Work that is now considered foundations for neural networks, was first published as a technical report by Rumelhart *et al.* in 1985 [12] and communicated by Hinton in a 1986 article that is widely cited [13]. This work re-discovered work by Paul Werbos in 1974, re-published in 1994 [15], that described how to automatically differentiate complicated compositions of functions.

The concept is simple. Instead of using the linear weighted sum of an observation, which we compute as u_i from Equation 28.2, they considered a nonlinear activation function $\phi(\cdot)$. This activation function must have a few key attributes by being:

- Easy to compute
- Continuous
- Differentiable
- Optionally, had bounded derivatives

In the original technical report, on Page 9, Rumelhart *et al.* we emphasize that they wrote:

"... we have used the *logistic* activation function ..."

As we have explored earlier, the logistic function satisfies these key attributes.

In artificial neural networks, we use the linear response of Equation 28.2 and the logistic function to find the response of the artificial neuron to an observation $\underline{x}_i$ as

$$\phi_i = \phi(\vec{w}; \underline{x}_i) = \frac{1}{1 + e^{-\underline{x}_i \vec{w}}} \tag{28.3}$$

The scaling effects that distinguish Equation 28.3 from Equation 25.6 are empirically negligible in extensive iterative computations. When plotted, the curve of Equation 28.3 is "S"-shaped and the computation is sometimes called a *sigmoid* activation function.

28.3 The Method of Steepest Descent

The basic learning problem for an artificial neuron is, given an augmented data matrix X and a label vector $\vec{y}$, how to learn the "best" weight vector $\vec{w}$. To date, there is no known closed-form solution for inverting Equation 28.3 to find $\vec{w}$ from a set of augmented data X. There is, however, a simple iterative computation that is widely used: the method of steepest descent, described in Class 26.

Instead of having a derivative, a multi-variable function has a *gradient* that is a 1-form with entries that are partial derivatives. The mathematical basis of the extended form is provided in the extra notes for this class and is summarized here.

The key concept is how to compute the gradient of the inner product of two vectors. In our case, the scalar value u_i is computed from Equation 28.2. We can think of u_i as a function that has a vector argument $\vec{w}$ and depends on a data observation $\underline{x}_i$, which can be written as

$$u_i \stackrel{\text{def}}{=} u(\vec{w}; \underline{x}_i) = \underline{x}_i \vec{w} \tag{28.4}$$

The gradient of u_i in Equation 28.4, from multivariate calculus, can be written as

$$\underline{\nabla} u_i = \underline{x}_i \tag{28.5}$$

We want to minimize the residual error between the label y_i and the score $\phi(u_i)$. In linear regression, we used the square of the residual error; doing likewise here, we start by observing that minimizing r_i^2 is the same as minimizing $\frac{1}{2}e_i^2$ and the constant factor $1/2$ will simplify subsequent differentiation. Thus, we would minimize

$$\begin{aligned} f_i(e_i; \underline{x}_i, y_i) &= \frac{1}{2} r_i^2 \\ r_i(z_i) &= y_i - z_i \\ z_i(u_i) &= \phi(u_i) = \frac{1}{1 + e^{-u_i}} \\ u_i(\vec{w}; \underline{x}_i) &= \underline{x}_i \vec{w} \end{aligned} \tag{28.6}$$

Consider an activation function $\phi(u)$ that is smooth and differentiable, such as the logistic function. The derivative is $\phi'(u)$. Applying the Chain Rule to Equation 28.6, the gradient is

$$\begin{aligned}
\underline{\nabla} f_i &= \left(\frac{\partial f_i}{\partial e_i}\right)\left(\frac{\partial e_i}{\partial z_i}\right)\left(\frac{\partial z_i}{\partial u_i}\right)[\underline{\nabla} u_i] \\
&= (e_i)\,(-1)\,(\phi'(u_i))\,[\underline{\nabla} u_i] \\
&= -\left(y_i - \phi(u_i)\right)\phi'(u_i)\underline{x}_i \qquad (28.7) \\
&= -b_i\,\underline{x}_i \\
\text{where} \quad b_i &\stackrel{\text{def}}{=} \left(y_i - \phi(u_i)\right)\phi'(u_i)
\end{aligned}$$

The scalar b_i in Equation 28.7 is the *back-propagation* factor; it scales the gradient direction $\underline{x}_i$ to account for the residual error and derivative term ϕ' that are propagated "back" when we use the Chain Rule to find the gradient.

The vector of steepest descent is $-[\underline{\nabla} f_i]^T$. Transposing Equation 28.7, and using the logistic function as the activation function $\phi(u_i)$, and using the derivative of the activation function as $\phi'(u_i) = \phi(u_i)(1 - \phi(u_i)$, the vector of steepest descent is a scaled version of the augmented observation $\underline{x}_i$, which is

$$\begin{aligned}
\vec{g}_i &= (y_i - \phi(u_i))\phi(u_i)(1 - \phi(u_i))\underline{x}_i^T \\
&= b_i\,\underline{x}_i^T \qquad (28.8)
\end{aligned}$$

The steepest vector for all of the data in the data matrix X is the sum of the steepest vectors for each observation, so

$$\vec{g} = \sum_{i=1}^{m} \vec{g}_i = \sum_{i=1}^{m} b_i\,\underline{x}_i^T \qquad (28.9)$$

When we implement Equation 28.9, we usually compute $\phi(u_i)$ once and use it repeatedly in the formula.

To implement the algorithm of steepest descent for a set of data in the matrix X, with corresponding labels in the vector y, we modify Algorithm 26.1 to handle vectors instead of scalars. We still need a hyper-parameter η, plus convergence criteria $k_{\max}$ and $g_{\min}$. An initial estimate $\vec{w}_0$ can be provided by the user or can be randomly generated.

Algorithm 28.1 is often mis-described as "gradient descent", which is incorrect because:

- the gradient is a 1-form, not a vector; and
- the direction of steepest descent is the negation of the transpose of the gradient

Algorithm 28.1 Artificial neuron optimization, fixed stepsize, $k_{\max}$ steps

```
w ← w0
k ← 0
while k ≤ k_max do
    g ← Σ_{i=1}^{m} (y_i − φ(u_i)) φ(u_i) (1 − φ(u_i)) x_i
    w ← w + η g^T
    k ← k+1
end while
```

Details of descent methods are topics in numerical optimization for machine learning.

Extra Notes

28.4 Extra Notes: Gradient of Squared-Error Objective

Consider an artificial neuron in which an augmented data observation is $\underline{x} \in \mathbb{R}^2$ and a label is $y \in \{0, 1\}$. The augmented data vector and the augmented weight vector are, in this simple example,

$$\underline{x} = \begin{bmatrix} x_1 & 1 \end{bmatrix} \qquad \vec{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \qquad \text{where} \qquad w_2 = -\beta$$

The linear term u can be expanded as

$$u = \underline{x}\vec{w} = \begin{bmatrix} x_1 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = x_1 w_1 + w_2$$

The partial derivative of u with respect to w_1, where we hold w_2 to be constant, is

$$\frac{\partial u}{\partial w_1} = x_1 \tag{28.10}$$

The partial derivative of u with respect to w_2, where we hold w_1 to be constant, is

$$\frac{\partial u}{\partial w_2} = 1 \tag{28.11}$$

Using a squared-error objective function, Equation 28.6 can be expanded with the term $u =$

$x_1 w_1 + w_2$ to produce

$$\begin{aligned} f(r; \underline{x}, y) &= \frac{1}{2} r^2 \\ r(z) &= y - z \\ z(u) &= \phi(u) = \frac{1}{1 + e^{-u}} \\ u(\vec{w}; \underline{x}) &= \underline{x}\vec{w} \\ &= x_1 w_1 + w_2 \end{aligned} \tag{28.12}$$

We can differentiate Equation 28.12 with respect to w_1, using Equation 28.10, as

$$\begin{aligned} \frac{\partial f}{\partial w_1} &= \left(\frac{\partial f}{\partial r}\right)\left(\frac{\partial r}{\partial z}\right)\left(\frac{\partial z}{\partial u}\right)\left(\frac{\partial u}{\partial w_1}\right) \\ &= (r)\,(-1)\,(\phi'(u))\,(x_1) \\ &= -(y_i - \phi(u_i))\,\phi'(u_i) x_1 \end{aligned} \tag{28.13}$$

Likewise, using Equation 28.10, we can differentiate Equation 28.12 with respect to w_2 as

$$\begin{aligned} \frac{\partial f}{\partial w_2} &= \left(\frac{\partial f}{\partial e}\right)\left(\frac{\partial e}{\partial z}\right)\left(\frac{\partial z}{\partial u}\right)\left(\frac{\partial u}{\partial w_2}\right) \\ &= (r)\,(-1)\,(\phi'(u))\,(1) \\ &= -(y_i - \phi(u_i))\,\phi'(u_i) 1 \end{aligned} \tag{28.14}$$

The gradient of $f(\vec{w}; \underline{x}, y)$ is the 1-form with entries that are the partial derivatives with respect to the entries of $\vec{w}$. Using the common scalar factors in Equation 28.13 and Equation 28.14, the gradient 1-form is

$$\begin{aligned} \underline{\nabla} f &= \begin{bmatrix} \frac{\partial f}{w_1} & \frac{\partial f}{w_2} \end{bmatrix} \\ &= \begin{bmatrix} (-(y_i - \phi(u_i))\,\phi'(u_i) x_1) & (-(y_i - \phi(u_i))\,\phi'(u_i) 1) \end{bmatrix} \\ &= -(y_i - \phi(u_i))\,\phi'(u_i) \begin{bmatrix} x_1 & 1 \end{bmatrix} \\ &= -(y_i - \phi(u_i))\,\phi'(u_i) \underline{x} \end{aligned} \tag{28.15}$$

Equation 28.15 can easily be extended to data observations and weight vectors of higher dimensions.

End of Extra Notes