CISC 371 Class 12

Neural Networks – Single Neuron

Article: Rumelhart et al, *Learning internal representations by back-propagating errors* [1]

Main Concepts:

- *A single neuron is a linear term and an activation function*
- *The gradient is found using the Chain Rule*
- *Optimization uses steepest descent*
- *For multiple observations, the descent vector is a vector summation*

Sample Problem, Machine Learning: How can we write a descent method for an artificial neuron using linear algebra?

An artificial neuron is often illustrated as in Figure 12.1. In Class 11, we explored the linear algebra that we will use for computation with an artificial neuron. Here we will complete the mathematical model with an activation function, develop an optimization using steepest descent,

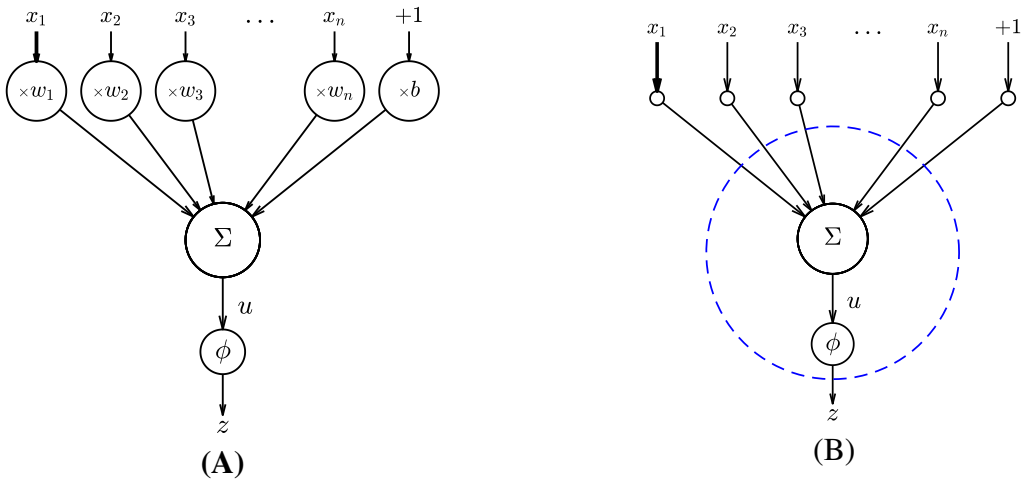


Figure 12.1: Illustrations of an artificial neuron. The input vector $\vec{x}$ is scaled by the weights $w_1, w_2, \ldots, w_n$ and biased by b; the result u is the input to the activation function ϕ that produces the neuron output z. (A) Data flow for an artificial neuron. (B) The external parameter is the input $\vec{x}$ and the output is z.

12.1 Activation Function of an Artificial Neuron

The *activation function* of an artificial neuron is a function that has a scalar argument, expressed as $\phi : \mathbb{R} \to \mathbb{R}$ or written as $\phi(u)$. Many activation functions have been used for artificial neurons. A constraint is that, with neural networks, the labels for observations are either 0 or 1. We will use three activations functions in this course.

12.1.1 Activation Function – Sigmoid

One function, which in early work was used for a neuron in a "hidden" layer of a neural network, is a map of a real number to the open interval $(0, 1)$ that is smooth, invertible, easily differentiated, and asymptotically approaches the bounds of the interval as the argument of the function approaches $\pm\infty$. This "sigmoid" – or S-shaped – activation function, and its derivative, are based on the *logistic* function:

$$\begin{array}{rcccl} z_S & = & \phi_S(u) & = & \dfrac{1}{1+e^{-u}} \\ \psi_S(u) & \stackrel{\text{def}}{=} & \phi_S{}'(u) & = & \phi(u)(1-\phi(u)) \end{array} \tag{12.1}$$

12.1.2 Activation Function – ReLU

Another commonly used activation function is the Rectified Linear Unit function, often written as the ReLU function. This function is zero for a scalar argument that is negative, and is the scalar argument otherwise. The derivative of the ReLU function is the Heaviside function, which implies that the activation function and its derivative can be easily computed. We will write the ReLU function as

$$\begin{array}{rcccl} z_R & = & \phi_R(u) & = & \max(0, u) \\ \psi_R(u) & \stackrel{\text{def}}{=} & \phi_R{}'(u) & = & H(u) \end{array} \tag{12.2}$$

12.1.3 Activation Function – Heaviside

An activation function that is sometimes used for the output neuron of a neural network is the Heaviside function; sometimes this is called a unit step function. This function is zero for a scalar

argument that is negative and unity otherwise. The derivative of the Heaviside function is always unity. We will write the Heaviside function and its derivative as

$$\begin{aligned} z_H \quad &= \quad \phi_H(u) \quad = \quad \begin{cases} 0 & \text{if} \quad u < 0 \\ 1 & \text{if} \quad u \geq 0 \end{cases} \\ \psi_H(u) \quad &\stackrel{\text{def}}{=} \quad \phi_H{}'(u) \quad = \quad 0 \end{aligned} \tag{12.3}$$

12.1.4 Activation Function – Identity

When we compute a neural network, for simplicity we will have the output neuron produce a value that is just the linear term u. This activation function is the *identity* function, which is simply

$$\begin{aligned} z_I \quad &= \quad \phi_R(u) \quad = \quad u \\ \psi_I(u) \quad &\stackrel{\text{def}}{=} \quad \phi_I{}'(u) \quad = \quad 1 \end{aligned} \tag{12.4}$$

12.2 Data Vectors and Data Observations

Our neural networks will have p layers. In this class, we will consider a single neuron, so the network has only a single layer that we will call Layer 1. We will use a leading superscript to denote the layer number for functions, variables, parameters, and data.

For neural networks, the data are typically represented as vectors $\vec{x}_j \in \mathbb{R}^n$. In this course, we prefer to represent data as observations, so we will write

$$\vec{x}_j \stackrel{\text{def}}{=} \underline{x}_j^T \tag{12.5}$$

For a single neuron, the data term will be written as

$$\vec{x} = \underline{x}^T \tag{12.6}$$

We will have to use a bias scalar to find the linear term for a neuron, so we will augment the data parameter with the value 1. This implies that the data 1-form that is the input parameter to a neuron, using the notation of Equation 12.6, is $[\underline{x} \; 1] \in \mathbb{R}^{n+1}$.

12.3 Variables and Parameters

We will use *semicolon* notation to separate the input parameter(s) of a function from the parameters. In general, we will write a function as

$$f(\textit{variables}; \textit{parameters}) \tag{12.7}$$

For example, for a function $f : \mathbb{R} \to \mathbb{R}$ that depends on a parameter $\underline{x}$, we will use Equation 12.7 to write f as

$$f(\vec{w}; \underline{x}) \tag{12.8}$$

12.4 Binary Classification

Our binary supervised classification problem is:

Given:	labelled data $(\vec{x}_j, y_j)$
Find:	augmented weight vector $\vec{w}$ that optimally classifies the data

We can formulate the optimization as a nonlinear least-squares problem. In neural networks, we need to be careful to distinguish when we are using the *classification* from when we are using the *activation* function. The former is used when we formulate the binary output of a neural network; the latter is used when we formulate inner, or "hidden", layers. These are closely related so we will define them together.

For us, the difference between the label y of an augmented data 1-form $[\underline{x}\ 1]$, and the classification or activation function of a data vector, is the *residual error*.

Definition: residual error of $\underline{x}$ for the activation function

> For any $\vec{x} \in \mathbb{R}^n$ augmented as $\underline{x} = \begin{bmatrix}\vec{x}^T & 1\end{bmatrix}$, any label $y \in \{0, 1\}$, any $\vec{w} \in \mathbb{R}^{n+1}$, the function $u : \mathbb{R}^{n+1} \to \mathbb{R}$ that is $u(\vec{w}) = \underline{x}\vec{w}$, and an activation function $\phi : \mathbb{R} \to \mathbb{R}$, the *residual error* r for the activation function is defined as
>
> $$r_\phi(\phi(u(\vec{w}))) \stackrel{\text{def}}{=} y - \phi(u(\vec{w})) \tag{12.9}$$

Definition: residual error of $\vec{x}$ for the classification function

For any 1-form $\underline{x} \in \mathbb{R}^n$ augmented as $[\underline{x}\ 1]$, any label $y \in \{0, 1\}$, any $\vec{w} \in \mathbb{R}^{n+1}$, the function $u : \mathbb{R}^{n+1} \to \mathbb{R}$ that is $u(\vec{w}; \underline{x}) = [\underline{x}\ 1]\vec{w}$, an activation function $\phi : \mathbb{R} \to \mathbb{R}$, and a classification function $q : \mathbb{R} \to \{0, 1\}$, the *residual error* r for the classification function is defined as

$$r_q(q(\phi(u(\vec{w}; \underline{x})))) \stackrel{\text{def}}{=} y - q(\phi(u(\vec{w}; \underline{x}))) \tag{12.10}$$

Our examples will use an objective function that is a quadratic function of the argument; this is often called the *squared-error* loss function. Other loss functions can be considered but we will use a quadratic objective for simplicity. We will use the quadratic function

$$g(r) = \frac{1}{2} r^T r \tag{12.11}$$

We will construct neural networks that have a single "output", which here is the value of the function. The concepts from a single-output network can be generalized to multiple outputs.

12.5 ANN – Single Neuron

Our first artificial neural network is a single neuron. As described above, we will use the Heaviside function for activation. The network will have an input parameter, which we write as $\underline{x}$, that has some number n_1 entries, so $\underline{x} \in \mathbb{R}^{n_1}$.

Recall that we will have to use a bias scalar to find the linear term for the neuron, so we will augment the data parameter with the value 1. This implies that the data 1-form that is the input parameter to this neuron is $[\underline{x}\ 1] \in \mathbb{R}^{n+1}$.

For a single neuron, the weight vector of the network is simply the weight vector of the neuron. Writing the weight vector for the neuron as $\vec{w}$, we therefore have

$$\vec{w} \in \mathbb{R}^{n_1+1} \tag{12.12}$$

For simplicity, and to better match the objective function of Equation 12.11, the artificial neuron that provides the output of the network will have a Heaviside step activation function. We will write this neuron as having the identity function as its activation function, so our examples we will have

$$\phi(u; \underline{x}) \stackrel{\text{def}}{=} \phi_H(u) \tag{12.13}$$

The objective function for the neuron is

$$
\begin{aligned}
f(r; y, \underline{x}) &\stackrel{\text{def}}{=} g(r) && (12.14)\\
g(r; y, \underline{x}) &\stackrel{\text{def}}{=} \frac{1}{2} r^T r \\
r(\phi; y, \underline{x}) &\stackrel{\text{def}}{=} y - \phi(u; \underline{x}) \\
\phi(u; \underline{x}) &\stackrel{\text{def}}{=} \phi_H(\vec{w}; \underline{x}) \\
u(\vec{w}; \underline{x}) &\stackrel{\text{def}}{=} \begin{bmatrix} \vec{x}^T & 1 \end{bmatrix} [\vec{w}] \\
\vec{x}(; \underline{x}) &\stackrel{\text{def}}{=} \underline{x}^T && (12.15)
\end{aligned}
$$

We can differentiate each term in Equation 12.14 as

$$
\begin{aligned}
\frac{\partial f_1}{\partial g} &= 1 \\
\frac{\partial g}{\partial r} &= r \\
\frac{\partial r}{\partial \phi} &= -1 \\
\frac{\partial \phi}{\partial u} &= \psi = 1 && (12.16)\\
\frac{\partial u}{\partial \vec{w}} &= \begin{bmatrix} \vec{x}^T & 1 \end{bmatrix}
\end{aligned}
$$

The gradient of Equation 12.14, using the terms in Equation 12.16 and neglecting the data parameter $\underline{x}$ for conciseness, is

$$
\begin{aligned}
\underline{\nabla} f(\vec{w}) &= \frac{\partial f}{\partial g} \frac{\partial g}{\partial r} \frac{\partial r}{\partial \phi} \frac{\partial \phi}{\partial u} \frac{\partial u}{\partial \vec{w}} && (12.17)\\
&= (1)(r)(-1)(1) \begin{bmatrix} \vec{x}^T & 1 \end{bmatrix}
\end{aligned}
$$

The direction of steepest descent is the negation of the transpose of Equation 12.17, which is

$$
\begin{aligned}
\vec{d} &\stackrel{\text{def}}{=} -\left[\frac{\partial u}{\partial \vec{w}}\right]^T \left[\frac{\partial \phi}{\partial u}\right]^T \left[\frac{\partial r}{\partial \phi}\right]^T \left[\frac{\partial g}{\partial r}\right]^T \left[\frac{\partial f}{\partial g}\right]^T && (12.18)\\
&= -\begin{bmatrix} \vec{x} \\ 1 \end{bmatrix} (1)(-1)(r)(1)
\end{aligned}
$$

We will write the direction vector $\vec{d}(\vec{w})$ using two terms: the *back-propagation* term b from the objective function and the *steepest direction* term $\vec{s}$. The descent direction in Equation 12.18 will be written as

$$\begin{aligned} b &\stackrel{\text{def}}{=} \left[\frac{\partial r}{\partial \phi}\right]^T \left[\frac{\partial g}{\partial r}\right]^T \left[\frac{\partial f}{\partial g}\right]^T && (12.19) \\ &= (-1)(r) \\ \vec{s} &\stackrel{\text{def}}{=} \left[\frac{\partial u}{\partial \vec{w}}\right]^T \left[\frac{\partial \phi}{\partial u}\right]^T && (12.20) \\ &= \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix} (1) \\ \vec{d}(\vec{w}) &= -[\vec{s}]b && (12.21) \end{aligned}$$

In Equation 12.21, the scalar term that is provided by the objective function and the residual function is *back-propagated* to the descent direction $\vec{d}$. The direction $\vec{s}$ in Equation 12.20 is the transpose of $[\underline{x}\ 1]$, where $\vec{x}$ is the transpose of the data parameter $\underline{x}$.

12.6 Optimization by Steepest Descent

In artificial neural networks, the usual training rule has a constant *learning rate*, written as η. This is a method of steepest descent for the objective function of Equation 12.14, which is

$$\begin{aligned} \vec{w}_{k+1} &= \vec{w}_k - \eta \left[\underline{\nabla} f(\vec{w}_k)\right]^T \\ &= \vec{w}_k + \eta\, \vec{d}(\vec{w}_k) && (12.22) \end{aligned}$$

12.7 Multiple Observations

For multiple observations m, each being $\underline{x}_j$, Equation 12.20 and Equation 12.21 to find a descent vector $\vec{d}_j$. The descent vector for the multiple observations is

$$\vec{d} \stackrel{\text{def}}{=} \sum_{j=1}^{m} \vec{d}_j \qquad (12.23)$$

The vector $\vec{d}_j$ of Equation 12.23 is used in the iteration of Equation 12.22 to learn the weight vector for the m observations.

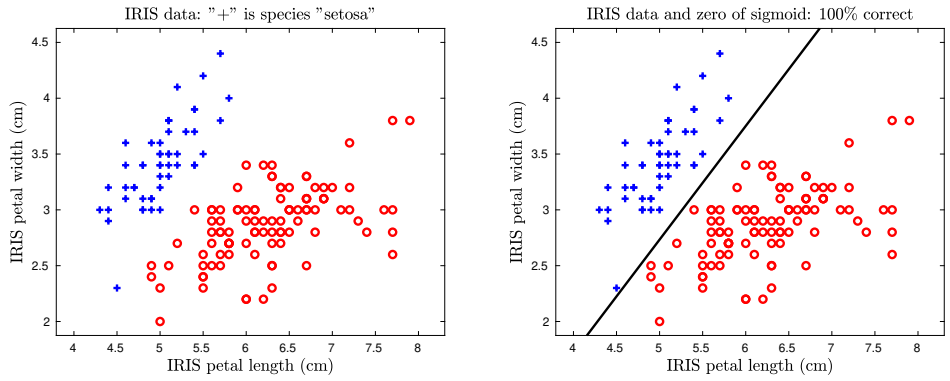
12.8 Example: Fisher's Iris Data

In 1936, Ronald Fisher [2] described a set of data that were botanical measurements of flowers that had been gathered on the Gaspé Peninsula of Canada. He used linear discriminant analysis to study the data and we can use an artificial neuron to replicate his findings.

The data are in an easily accessed `MATLAB` repository. We can extract only the petal measurements and derive binary labels for samples that are the species *Iris setosa* or another species. The petal sizes provide us with vectors $\vec{x}_j$ and the species provide us with labels y_j.

We can use the sigmoid activation function for learning how to classify the data to match the labels. For example, we can use a fixed stepsize of $s = 0.03$ in the algorithm for steepest descent that we derived in Class 8. Using the objective function of Equation 12.14, and the activation function $\phi(\cdot)$ and derivative $\psi(\cdot)$ of Equation 12.1, we can develop a relatively small `MATLAB` function that can be an argument for code that implements the steepest descent with a fixed stepsize.

The data and a separating hyperplane are shown in Figure 12.2. As expected, the method of steepest descent estimated a minimizer that separates the data vectors according to the species.



(A) **(B)**

Figure 12.2: Vector representation of petal sizes for Fisher's iris data. Vectors for species *Iris setosa* are shown as blue crosses and vectors for other species are shown as red circles. (A) The data, labeled as in Fisher's article. (B) Classifications by a single sigmoidal artificial neuron; the 0-level contour of the sigmoidal function is shown as a black line.

12.9 Extra Notes on Output-Layer Differentiation

When we add a layer of neurons to the output neuron, we will need to clarify the structures of the weight vector $\vec{w}$ and the linear term u. We will re-write the weight vector as

$$\begin{aligned} \vec{w} &\in \mathbb{R}^{n_1+1} \\ \vec{w}_{1..n_1} &\stackrel{\text{def}}{=} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n_1} \end{bmatrix} \\ \vec{w} &= \begin{bmatrix} \vec{w}_{1..n_1} \\ w_{n_1+1} \end{bmatrix} \end{aligned} \tag{12.24}$$

The linear term for the output layer, which is u, will be written as

$$\begin{aligned} u(\vec{w}, \vec{x}; \underline{x}) &\stackrel{\text{def}}{=} [\vec{x}^T \ 1][\vec{w}] \\ &= [\vec{w}]^T \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix} \\ &\stackrel{\text{def}}{=} [\vec{w}_{1..n_1}]^T \vec{x} + \vec{w}_{n_1+1} \end{aligned} \tag{12.25}$$

We can differentiate Equation 12.25 with respect to the weight vector $\vec{w}$ and with respect to the data vector $\vec{x}$. For a single neuron, we have set the input to be the data observation $\underline{x}$ so that $\vec{x} = \underline{x}^T$. If $\vec{x}$ is an argument for the linear term, the two derivatives of Equation 12.25 are

$$\frac{\partial u}{\partial \vec{w}} = [\vec{x}^T \ 1] \tag{12.26}$$

$$\frac{\partial u}{\partial \vec{x}} = \vec{w}_{1..n_1}^T \tag{12.27}$$

The gradient of the objective function $f(\cdot)$ in Equation 12.14, with respect to $\vec{x}$, is

$$\begin{aligned} & \frac{\partial f}{\partial g} \frac{\partial g}{\partial r} \frac{\partial r}{\partial \phi} \frac{\partial \phi}{\partial u} \frac{\partial u}{\partial \vec{x}} \\ = \ & (1)(r)(-1)(1)\vec{w}_{1..n_1}^T \\ = \ & \vec{w}_{1..n_1}^T \, b \end{aligned} \tag{12.28}$$

References

[1] Rumelhart DE, Hinton GE, Williams RJ: Learning representations by back-propagating errors. *Nature* 323(6088):533–536, 1986

[2] Fisher RA: The use of multiple measurements in taxonomic problems. *Ann Eugen* 7(2):179–188, 1936