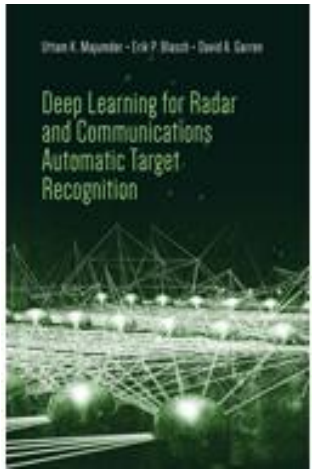
Machine Learning Techniques for Radar Automatic Target Recognition (ATR)

Chapter 2: Mathematical Foundations for Machine Learning

Uttam K. Majumder

Outline



Multi-Variate Calculus

If a function $f: \mathbb{R}^n \to \mathbb{R}$ is differentiable, then the function ∇f or gradient of f is defined by:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{bmatrix} = Df(x)^T$$

The gradient of a function of two variables is defined by:

$$\nabla f(x, y) = \left[\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right].$$

Consider a vector function of the form:

$$f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix}.$$

Then, the gradient is given by:

$$\frac{\partial f}{\partial x_k}(x_0) = \begin{bmatrix} \frac{\partial f_{1(x_0)}}{\partial x_j} \\ \vdots \\ \frac{\partial f_{m(x_0)}}{\partial x_k} \end{bmatrix}.$$

Multi-Variate Calculus

The derivative matrix or ***Jacobian*** matrix of $f(x)$ can be defined as:

$$\left[\frac{\partial f}{\partial x_1}(x_0) \quad \dots \quad \frac{\partial f}{\partial x_n}(x_0)\right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x_0) & \dots & \frac{\partial f_1}{\partial x_n}(x_0) \\ \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial x_1}(x_0) & \dots & \frac{\partial f_m}{\partial x_n}(x_0) \end{bmatrix}.$$

Given $f: \mathbb{R}^n \to \mathbb{R}$, if ∇f is differentiable, then f is twice differentiable. The derivative of ∇f is defined by:

$$D^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

Multi-Variate Calculus

The matrix $D^2 f(x)$ is called the ***Hessian*** matrix of f at x. This is also denoted as $\boldsymbol{F}(\boldsymbol{x})$.

A matrix, M is symmetric if $M = M^T$. Consider the quadratic form $f: R^n \to R$ that is the function:

$$f(x) = x^T A x \ .$$

Let $M \in \mathbb{R}^{mxn}$ and $y \in \mathbb{R}^m$, then we can define derivative D with respect to x as:

$$D(y^T M x) = y^T M \ ,$$

$$D(x^T M x) = x^T M + M x = x^T M + x^T M^T = x^T (M + M^T) \ .$$

Since M is symmetric matrix, it follows that:

$$D(x^T M x) = 2x^T M \ ,$$

$$D(x^T x) = 2x^T \ .$$

Gradient Descent Algorithm (GDA)

The gradient descent algorithm minimizes an objective function iteratively. Consider an objective function $f(x)$. Suppose we are given a point x^k at iteration k. To move to the next point (minimum or optimum) x^{k+1}, we begin at x^k and then add an amount $-\eta_k \nabla f(x^k)$, where η_k is a positive scaler known as ***step size*** or ***learning rate***, and $-\nabla f(x^k)$ is the direction of maximum rate of decrease. Hence, an equation for GDA can be written:

$$x^{(k+1)} = x^k - \eta_k \nabla f(x^k) \, .$$

The learning rate η can be selected as a small or large value. A small value of η takes longer compute time to reach the minimum point. On the other hand, a larger value of η results in faster (compute time) convergent to the minimum point, as it requires only few gradient evaluations. There are variations of gradient based algorithms. Among these, the steepest descent is the most commonly used. We can derive a closed form GDA solution for a quadratic function. |

Consider a quadratic function of the form:

$$f(x) = \frac{1}{2} x^T M x - b^T x.$$

Here, $M \in \mathbb{R}^{nxn}$ is a symmetric positive definitive matrix, $b \in \mathbb{R}^n$, $x \in \mathbb{R}^n$. First, compute the gradient:

$$\nabla f(x) = Mx - b.$$

Gradient Descent Algorithm

Now, the Hessian of $f(x)$ is $F(x) = M = M^T > 0$. For notational simplicity, consider:

$$g^k = \nabla f(x^k) = Mx^k - b.$$

Then we can write the steepest descent algorithm for the quadratic function as:

$$x^{k+1} = x^k - \eta_k g^k \ .$$

Now for the steepest descent, the *step size* or *learning rate* can be computed as:

$$\eta_k = \arg\min_{\eta_k \geq 0} f(x^k - \eta_k \ g^k) = \arg\min_{\eta_k \geq 0} \{\frac{1}{2}(x^k - \eta_k g^k)^T M(x^k - \eta_k g^k) - (x^k - \eta_k g^k)^T b\}$$

By taking derivative (with respect to η_k) of the above minimizer and setting to zero, we find:

$$(x^k - \eta_k g^k)^T M(-g^k) + b^T g^k = 0 \ ,$$

or, equivalently,

$$\eta_k (g^k)^T M g^k = ((x^k)^T M - b^T) g^k \ .$$

However, we also can use:

$$((x^k)^T M - b^T) = (g^k)^T \ .$$

Hence, the learning rate or step size η_k for the quadratic function can be written as:

$$\eta_k = \frac{(g^k)^T g^k}{(g^k)^T M g^k}.$$

Finally, we derive the closed form equation for the steepest descent algorithm for the quadratic function as:

$$x^{\{k+1\}} = x^{\{k\}} - \eta_k g^{\{k\}} = x^{\{k\}} - \frac{(g^{\{k\}})^T g^{\{k\}}}{(g^{\{k\}})^T M g^{\{k\}}} g^{\{k\}} \ ,$$

in terms of the following:

$$g^k = \nabla f(x^k) = Mx^k - b$$

Backpropagation

- An important component in artificial neural networks (ANNs) is the training of the weights based upon labeled training data via **backpropagation** processing. This theory is based upon optimization theory involving vectors

- The strategy begins with sets of given labeled training data. Assume that the vector **x** describe the form of the input data sets. There is much freedom in the form of the selected input data

- The output vector **y** describes the information desired by human users

- The backpropagation technique is an iterative approach based on gradient descent methods. Fundamentally, an initial guess for the solution is selected, which is often based on random weights. Then, gradient descent techniques are used to compute the updated weights for each iteration

Backpropagation

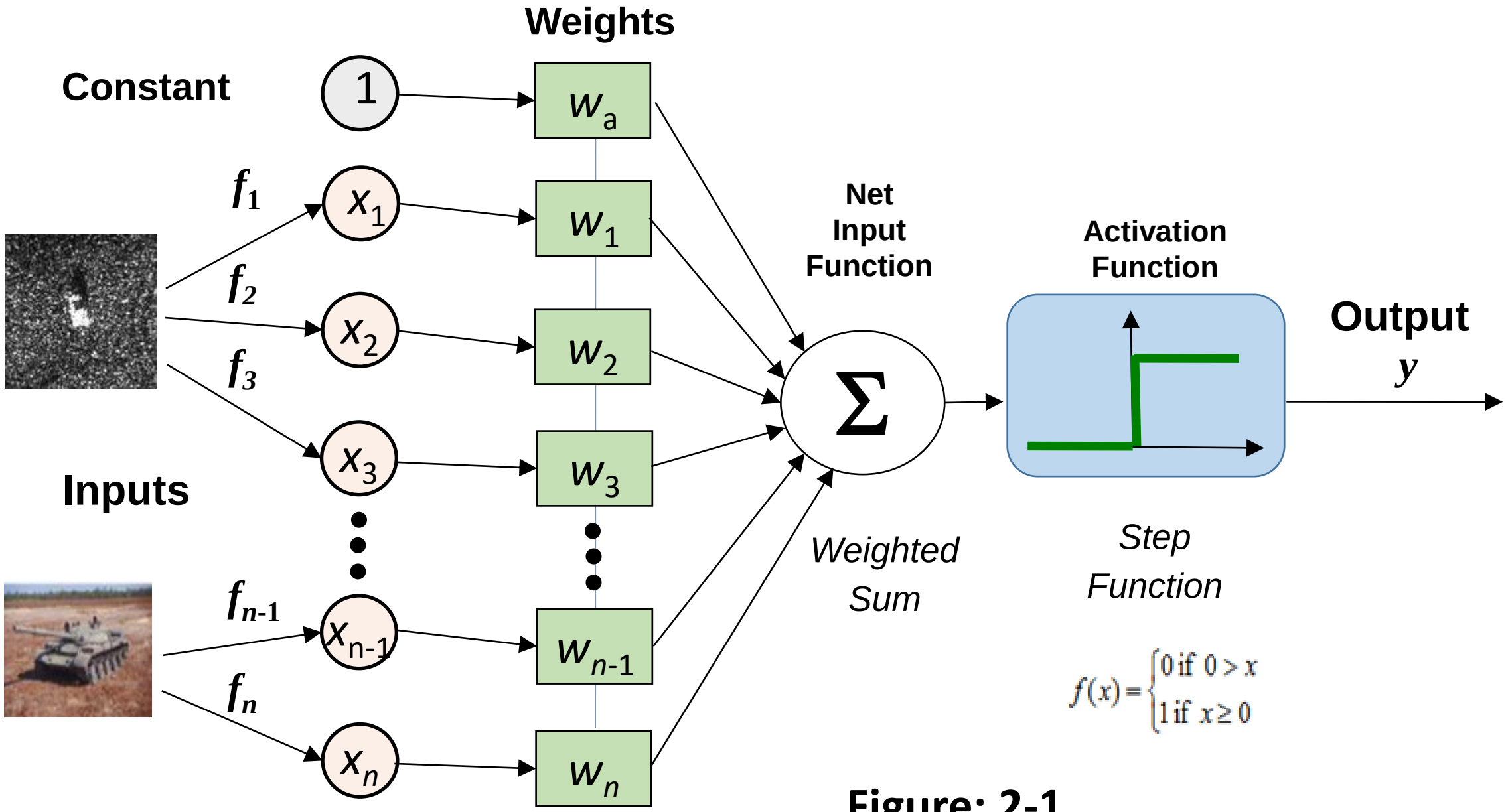


Figure: 2-1

Backpropagation

Consider a general ANN shown in Figure 2-1. Here, there are some number P of initial images, each of which comprises a particular input data vector $\mathbf{x}$. Each of the these data vectors are processed by some number L of layer of the ANN. Let the index m be a particular node in layer ℓ. Assume the output $o_{\ell-1,m}$ from a particular node with index m in the previous layer $\ell-1$ feeds forward into the subject node with index n in the current layer be given by $w_{\ell,m,n}$. In addition, a non-zero constant bias $b_{\ell,n}$, which does not depend upon the outputs of the previous layer, can be applied to the subject node with index n in layer ℓ.

The linear activation $a_{\ell,n}$ is the sum of the product contributions from all $M_{\ell-1}$ connected nodes at the previous layer $\ell-1$ plus the bias $b_{\ell,n}$ and can be expressed via:

$$a_{\ell,n} = b_{\ell,n} + \sum_{m=1}^{M_{\ell-1}} w_{\ell,m,n} o_{\ell-1,m} \,.$$

Next, this linear activation $a_{\ell,n}$ is input into a general non-linear function $f_\ell(\cdot)$ at layer ℓ in order to yield the output from node n in layer ℓ, i.e.,

$$o_{\ell,n} = f_\ell(a_{\ell,n}) \,.$$

It is convenient to incorporate the bias $b_{\ell,n}$ into the general weight formulation by defining a output variable corresponding to a non-existent node $n = 0$ to be:

$$o_{\ell-1,0} = 1.$$

Next, the bias weight term is defined by:

$$w_{\ell,0,n} = b_{\ell,n}.$$

Therefore, the summation in (2.54) can be extended to apply with the lower index equal to unity rather than zero:

$$a_{\ell,n} = \sum_{m=0}^{M_{\ell-1}} w_{\ell,m,n} o_{\ell-1,m} \,.$$

The backpropagation weight update is computed often by using the mean squared error of the difference between the measured output data vector $\mathbf{y}$ and the vector predicted at the output of the forward model:

$$E(\mathbf{x},\mathbf{w}) = \frac{1}{2P}\sum_{p=0}^{P} \|\hat{\mathbf{y}}_p - \mathbf{y}_p\|^2 = \frac{1}{2P}\sum_{p=0}^{P} \{\hat{\mathbf{y}}_p - \mathbf{y}_p\} \cdot \{\hat{\mathbf{y}}_p - \mathbf{y}_p\} \,.$$

Backpropagation

Here, the summation is over the P individual pairs of input-output data vectors, i.e., $\{\mathbf{x}_1, \mathbf{y}_1\}, \{\mathbf{x}_2, \mathbf{y}_2\}, \ldots, \{\mathbf{x}_P, \mathbf{y}_P\}$. Specifically, the vector $\hat{\mathbf{y}}_p$ denotes the final estimate of $\mathbf{y}_p$ as is obtained from executing the ANN in the forward direction from input to output. Also, the vector $\mathbf{w}$ denotes the set of all of the weight coefficients $\mathbf{w} = \{w_{1,1,1}, w_{1,2,1}, \ldots\}$ that are to be optimized is this iterative backpropagation processing.

The overall iterative updates to each of the weights $\mathbf{w} = \{w_{1,1,1}, w_{1,2,1}, \ldots\}$ proceeds by applying the partial derivative of $E(\mathbf{x}, \mathbf{w})$ with respect to each weight individually. Here, we show a particular iteration using the superscript $\{i\}$, so that the weight $w_{\ell,m,n}^{\{i+1\}}$ at iteration $i+1$ is expressed in terms of that at the previous iteration $w_{\ell,m,n}^{\{i\}}$ and the partial derivative of the optimization function of $E(\mathbf{x}, \mathbf{w})$ with respect to the weight $w_{\ell,m,n}^{\{i\}}$ via:

$$w_{\ell,m,n}^{\{i+1\}} = w_{\ell,m,n}^{\{i\}} - \eta_0 \frac{\partial E(\mathbf{x}, \mathbf{w})}{\partial w_{\ell,m,n}} \quad .$$

In this equation, η_0 is the learning rate or step size.

Statistical and Probability Theory

Statistics and probability theory are necessary for machine learning and artificial intelligence. Probability theory models uncertain events and estimates parameters from measurements. As more data are available (i.e., additional observations), the accuracy of classification improves significantly. Statistics and probability theory explain these phenomena. In this section, we present some of the important concepts in statistics and probability theory applied to machine learning.

Probability Density Functions

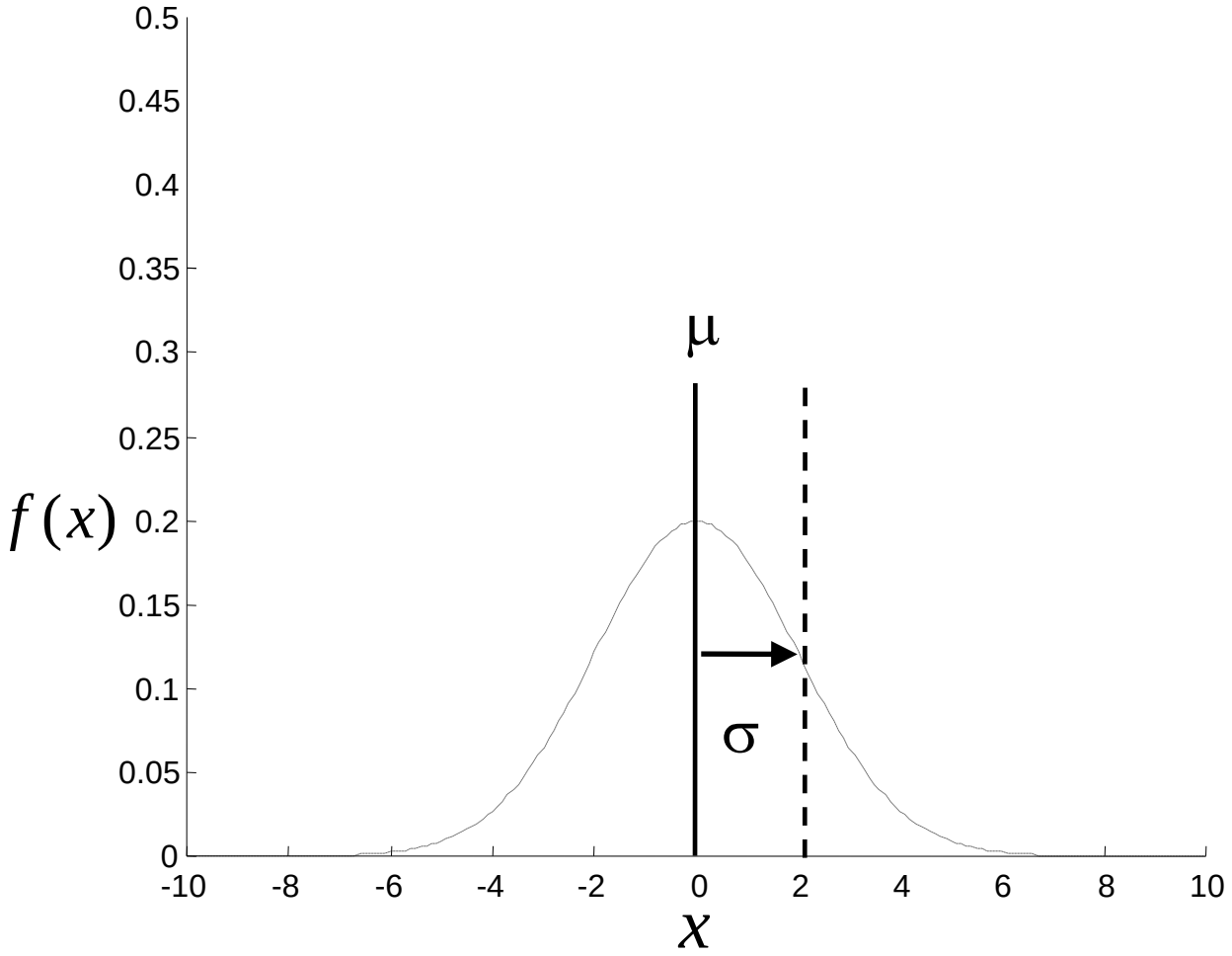


Figure 2.2 Target pdfs with Gaussian normal distribution of $\mu = 0$ and $\sigma = 2$; N (0, 2.0)

A Priori Target Likelihood

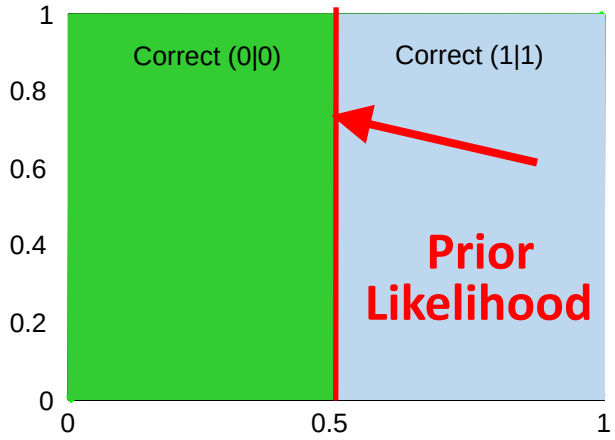


Figure: A Priori Target likelihood p = [0.5, 0.5]

A priori information has been collected over time. For the example, the processing begins with the ATR assessing the measurement. Assuming the collection of equivalent data, the distinction between the vehicle classification (1) and the incorrect classification (0) is $p = [0.5\ 0.5]$

Summary

- We briefly discussed important mathematical foundation necessary for artificial intelligence and machine learning (AI/ML)