EE 516: Mathematical Foundations of Machine Learning

Lecture 2

Mark A. Martin

Portland State University

January 8, 2025

Quick Review

Last class,

- I gave you an overview of machine learning. As we discuss techniques and mathematical tools, I encourage you to think about where they fit into the big picture that I shared.

A Simple Example

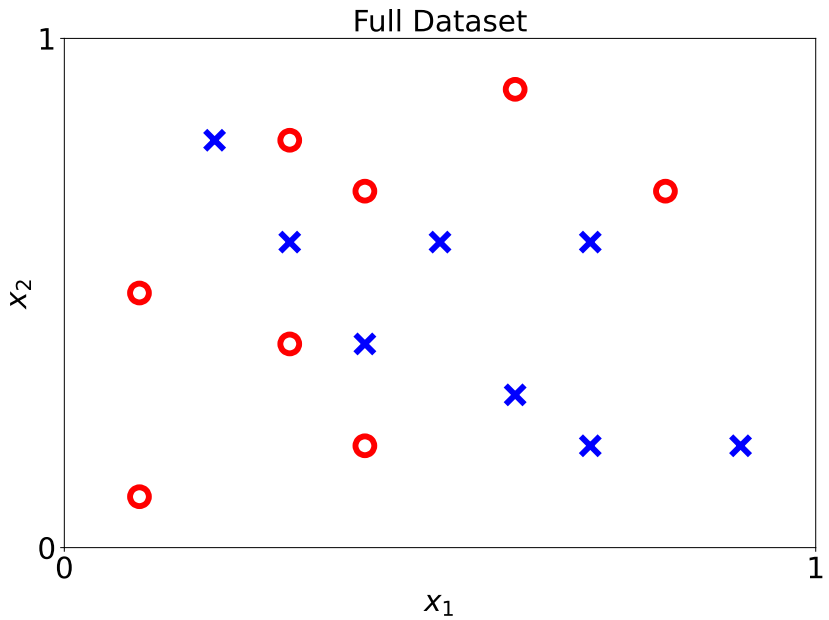
I want to provide an example of machine learning in the context of a simple problem and particular approach to machine learning. This will illustrate some important ideas and terminology and provide concrete examples. This discussion will be based on the paper

"Deep Learning: An Introduction for Applied Mathematicians"

by Catherine F. Higham and Desmond J. Higham, SIAM Review, V. 61, N. 4 (2019). This paper is open-access and the supplementary materials provided at the link above include the MATLAB code from the paper. I posted python versions of the code and slightly modified versions of the MATLAB code in Canvas. I posted the other supplementary materials too.

A Simple Example (Continued)

Using the notation of the paper, this machine learning application begins with a dataset consisting of **two-dimensional points** $\left\{x^{\{i\}}\right\} \in \mathbb{R}^2$ that can be mapped into the unit square. In the previous notation, $x^{\{i\}} = \mathbf{x}_i$. Each point is identified as being in one of **two categories**, one labeled with circles and the other with x's.



Here, I have added a few more points to the dataset in the paper.

A Simple Example (Continued)

These points might have been determined by mapping the latitude and longitude of places on a map to the unit square. One concrete example is that they could represent the locations of successful and unsuccessful oil wells (or water wells) in an area. The **goal** is to create a way to **automatically distinguish** points in the two categories and **predict** whether an arbitrary point in the unit square is in the circle or the x category.

What type of learning fits this problem?

This is a problem in **supervised learning**.

A Simple Example (Continued)

From Lecture 1, **supervised learning has the steps**

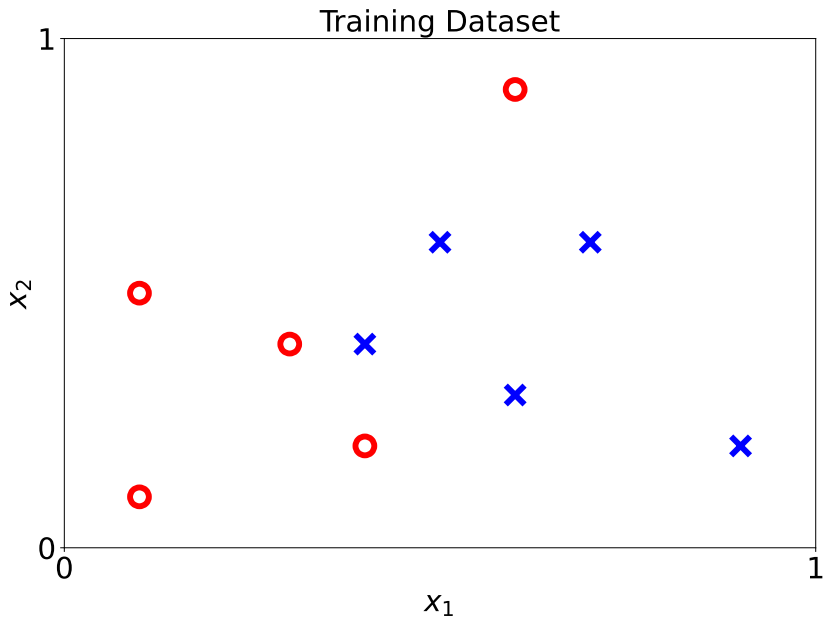
1. Label data
2. Construct a model
3. Specify an objective or cost function
4. Train the model
5. Verify the model

Have any of the steps been completed at this point? At this point, the dataset has been **labeled**. A mathematical representation for the labels will be discussed later.

A Simple Example (Continued)

A dataset is normally split into two parts – **one part** that will be used **for training** and **the remainder** that will be used **for validation**. Points are usually assigned to the training and validation sets randomly and it's typical to use 80% of the data for training.

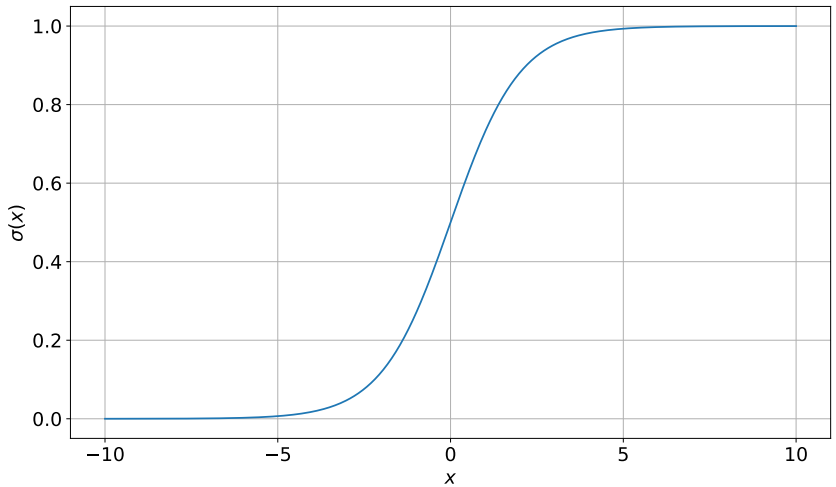
We'll assume that the points provided in the paper were chosen to make up the training set and the points I added will be the validation set. (Note: This isn't 80% of the total augmented data set.)



A Simple Example – The Sigmoid Function

The next step is to **create a model** to represent the data. The model will be a function based on the **sigmoid function**.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad (1)$$



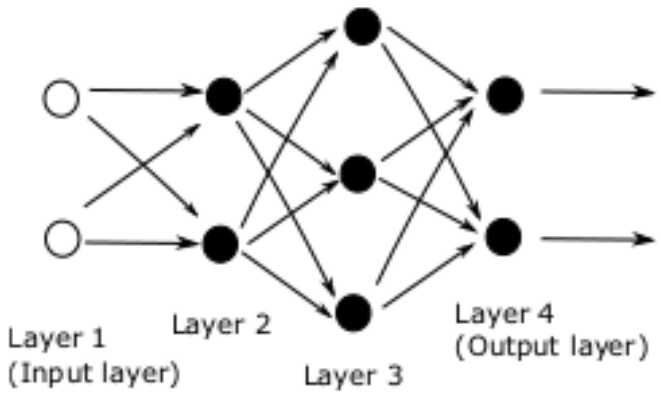
The sigmoid function is a simplified representation of how a neuron behaves. If a neuron is stimulated with a signal x that is below a certain threshold, the neuron doesn't fire. If it's stimulated with a signal above the threshold, it fires. The threshold for Equation (1) is $x = 0$.

A Simple Example – Model Representing the Data

The approach we'll take is called **deep learning** and it requires extending the definition of $\sigma(x)$ to vectors. If

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{pmatrix} \in \mathbb{R}^M, \text{ then define } \sigma(x) = \begin{pmatrix} \sigma(x_1) \\ \sigma(x_2) \\ \vdots \\ \sigma(x_M) \end{pmatrix}. \tag{2}$$

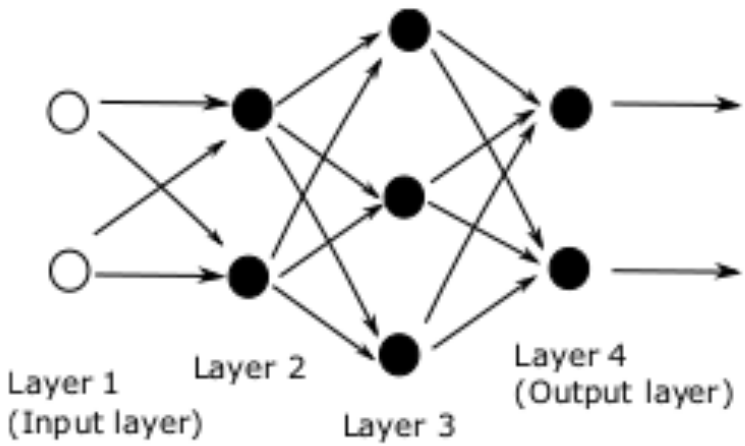
The **model used to represent the data** will be based on the diagram



The adjective **"deep"** comes from the use of multiple layers in the network.

A Simple Example – The Model (Continued)

The circles on a vertical line in the diagram correspond to a vector and each circle corresponds to a component of the vector. Each vector represents a layer of neurons consisting of the same number of neurons as the number of components in the vector.



The vector in the **Input Layer**, i.e. Layer 1, is the point from $\mathbb{R}^2$ being evaluated. At each layer, the output vector a from the previous layer is multiplied by a matrix W of **weights**, a vector of **biases** b is added to

A Simple Example – The Model (Continued)

the product Wa, and the **sigmoid function** is applied to the result. So, the output of the layer is

$$\sigma(Wa + b). \tag{3}$$

For Level l, let $W^{[l]}$ be the matrix of weights and $b^{[l]}$ be the vector of biases. For the network in the diagram,

$$\begin{aligned} W^{[2]} &\in \mathbb{R}^{2\times 2}, & b^{[2]} &\in \mathbb{R}^2 \\ W^{[3]} &\in \mathbb{R}^{3\times 2}, & b^{[3]} &\in \mathbb{R}^3 \\ W^{[4]} &\in \mathbb{R}^{2\times 3}, & b^{[4]} &\in \mathbb{R}^2 \end{aligned} \tag{4}$$

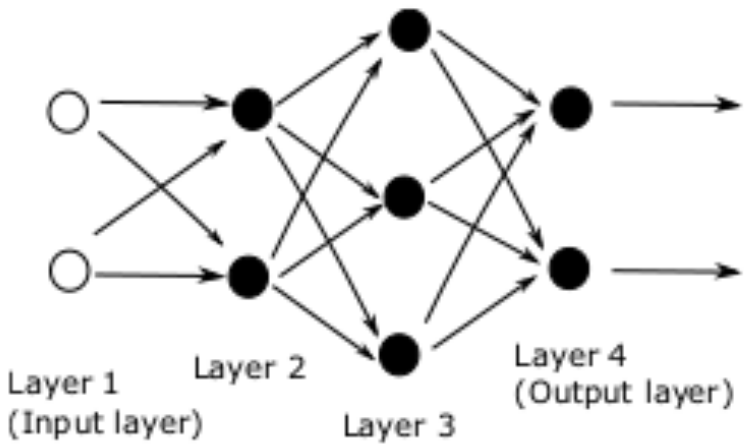
and, if x is the input, then the output of the entire network is

$$\begin{aligned} F(x) &= \sigma\left(W^{[4]}\,\sigma\left(W^{[3]}\,\sigma\left(W^{[2]}x + b^{[2]}\right) + b^{[3]}\right) + b^{[4]}\right) \\ &= \begin{pmatrix} F_1(x) \\ F_2(x) \end{pmatrix} \in \mathbb{R}^2. \end{aligned} \tag{5}$$

A Simple Example – The Model (Continued)

The function $F : \mathbb{R}^2 \to \mathbb{R}^2$ contains 23 parameters – the elements of the matrices of weights and the components of the bias vectors.

Why did we choose a function F that produces a vector with 2 components? Equivalently, why does Layer 4 have 2 neurons?



The reason is that there are two categories for points in the plane.

A Simple Example – Labels and Goal

At this point, **a mathematical representation for the labels** is needed. Define

$$y\left(x^{\{i\}}\right) = \begin{pmatrix} y_1^{\{i\}} \\ y_2^{\{i\}} \end{pmatrix} = \begin{cases} \begin{pmatrix} 1 \\ 0 \end{pmatrix}, & \text{if } x^{\{i\}} \text{ is in the circle category} \\ \begin{pmatrix} 0 \\ 1 \end{pmatrix}, & \text{if } x^{\{i\}} \text{ is in the x category} \end{cases} \tag{6}$$

(In the notation defined earlier, $y\left(x^{\{i\}}\right) = y_i$.) Then **a mathematical formulation of the goal** is to find weights and biases that make

$$F\left(x^{\{i\}}\right) \approx y\left(x^{\{i\}}\right) \tag{7}$$

for all of the data points in the training set. The boundary between points in the two categories will be points where $F_1(x) = F_2(x)$.

A Simple Example – Objective Function

Finally, if $F_1(x) > F_2(x)$, x will be a member of the circle category and, if $F_1(x) < F_2(x)$, x will be a member of the x category. This completes the first 2 steps in the procedure for supervised learning previously described.

Since the mathematical problem is an optimization problem, the next step is to define **an objective function** that measures how close a set of weights and biases comes to meeting the goal. Since **the goal is to minimize the distances between** $F\left(x^{\{i\}}\right)$ and $y\left(x^{\{i\}}\right)$ for all points in the training set, any non-negative cost function that increases as any of these distances becomes larger and decreases as any of them become smaller will work. **Different cost functions will have different numerical properties and computational costs**. A reasonable choice is

$$\text{Cost} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\left\|y\left(x^{\{i\}}\right) - F\left(x^{\{i\}}\right)\right\|_2^2, \tag{8}$$

where N is the number of points in the training set.

A Simple Example – Objective Function

The **cost function** is $\frac{1}{2}$ of the average square of the **Euclidean distance** between

- the **mathematical representation of the label** for a point in the training set
- and the **function** F **evaluated at the point**.

The term in the cost function for the point $x^{\{i\}}$ is

$$\begin{aligned}\left\| y\left(x^{\{i\}}\right) - F\left(x^{\{i\}}\right)\right\|_2^2 &= \left\| \begin{pmatrix} y_1^{\{i\}} \\ y_2^{\{i\}} \end{pmatrix} - \begin{pmatrix} F_1(x^{\{i\}}) \\ F_2(x^{\{i\}}) \end{pmatrix} \right\|_2^2 \\ &= \left\| \begin{pmatrix} y_1^{\{i\}} - F_1(x^{\{i\}}) \\ y_2^{\{i\}} - F_2(x^{\{i\}}) \end{pmatrix} \right\|_2^2 \\ &= \left(\left(y_1^{\{i\}} - F_1(x^{\{i\}}) \right)^2 + \left(y_2^{\{i\}} - F_2(x^{\{i\}}) \right)^2 \right). \end{aligned} \tag{9}$$

The factor of $\frac{1}{2}$ is essentially irrelevant but is helpful theoretically. It simplifies some expressions in the final algorithm.

A Simple Example – Minimizing the Cost

Now that we have a cost function, we need **a way to minimize it** or, in the language of machine learning, we need a way to **train the deep learning neural network**. Since the points in the training data are fixed, the cost is a function of the 23 parameters in F and the cost value is a non-negative real number, i.e.

$$\text{Cost} : \mathbb{R}^{23} \to \mathbb{R}^{+}. \tag{10}$$

Let $p \in \mathbb{R}^{23}$. We want to find $\Delta p \in \mathbb{R}^{23}$ to make $\text{Cost}(p + \Delta p)$ smaller than $\text{Cost}(p)$. The direction that leads to the greatest decrease in the cost is $\Delta p \propto -\nabla \text{Cost}(p)$ (Why?), where

$$(\nabla \text{Cost}(p))_r = \frac{\partial \text{Cost}(p)}{\partial p_r} \text{ for } r = 1, 2, \cdots, 23. \tag{11}$$

To reduce the chances of taking a step that is too large, we define

$$\Delta p = -\eta \nabla \text{Cost}(p), \tag{12}$$

where η is a suitably small constant called the **learning rate**.

A Simple Example – Minimizing the Cost (Continued)

The **algorithm to train the network** is

1. Choose an initial $p \in \mathbb{R}^{23}$. (p consists of the weights and biases.)
2. Iterate
$$p \to p - \eta \nabla \text{Cost}(p) \tag{13}$$
a large number of times or until $\text{Cost}(p)$ falls below a specified threshold.

This is called the **gradient descent** or **steepest descent** method. (See [1], pp 305-7.)

To use this method, we need to be able to evaluate the gradient of the cost function. Define

$$C_{x^{\{i\}}}(p) = \frac{1}{2} \left\| y\left(x^{\{i\}}\right) - F\left(x^{\{i\}}\right) \right\|_2^2, \tag{14}$$

which is the portion of the cost associated with point $x^{\{i\}}$. Then

$$\text{Cost}(p) = \frac{1}{N} \sum_{i=1}^{N} C_{x^{\{i\}}}(p) \tag{15}$$

A Simple Example – Minimizing the Cost (Continued)

and

$$\nabla \text{Cost}(p) = \frac{1}{N} \sum_{i=1}^{N} \nabla C_{x^{\{i\}}}(p). \tag{16}$$

Evaluating this expression requires calculating 23 **partial derivatives for each term** $C_{x^{\{i\}}}(p)$ and $23N$ partial derivatives altogether. This might be feasible for a simple problem but it is **too computationally expensive** for a more realistic problem.

A more practical approach is the **stochastic gradient descent** method:

1. Choose an initial $p \in \mathbb{R}^{23}$.
2. Randomly choose i from $\{1, 2, \cdots, N\}$, i.e. a specific point $x^{\{i\}}$.
3. Perform an iteration that improves the cost for $x^{\{i\}}$

$$p \to p - \eta \nabla C_{x^{\{i\}}}(p) \tag{17}$$

4. Repeat the previous 2 steps a certain number of times or until the cost is lower than a specified threshold.

A Simple Example – Minimizing the Cost (Continued)

Notes:

- The paper lists some **other variations of this method** and associated terminology.
- **These methods are not guaranteed to succeed**, except under certain special circumstances.

What remains is to **apply the stochastic gradient descent method to the cost function with the specific function** F that we're using to represent the data. Suppose that there are L levels and that there are n_l neurons in level l. Let $a^{[l]}$ be the **output of the l^{th} level** and define

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}, \tag{18}$$

which is the **input for the l^{th} level**. This means

$$a^{[l]} = \sigma\left(z^{[l]}\right) \quad \text{and} \quad a^{[L]} = \sigma\left(z^{[L-1]}\right) = F\left(x^{\{i\}}\right). \tag{19}$$

A Simple Example – Minimizing the Cost (Continued)

To apply the stochastic gradient descent method, we only need to determine the gradient of one term in the sum

$$C_{x^{\{i\}}}(p) = \frac{1}{2}\left\| y\left(x^{\{i\}}\right) - F\left(x^{\{i\}}\right)\right\|_2^2. \tag{20}$$

Let

$$\begin{aligned} C(p) = C_{x^{\{i\}}}(p) &= \frac{1}{2}\left\| y\left(x^{\{i\}}\right) - F\left(x^{\{i\}}\right)\right\|_2^2 \\ &= \frac{1}{2}\left\| y - a^{[L]}\right\|_2^2 = \sum_{j=1}^{n_l} \frac{1}{2}\left(y_j - a_j^{[L]}\right)^2, \end{aligned} \tag{21}$$

where $y = y\left(x^{\{i\}}\right)$. (Note that only $a^{[L]}$ depends on p in this expression.) Finally, define

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}}. \tag{22}$$

A Simple Example – Minimizing the Cost (Continued)

In the paper, it is shown that

$$\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]} \quad \text{and} \quad \frac{\partial C}{\partial w_{jk}^{[l]}} = \delta_j^{[l]} a_k^{[l-1]}, \tag{23}$$

where

$$\delta^{[L]} = \sigma'\left(z^{[L]}\right) \circ \left(a^{[L]} - y\right) \tag{24}$$

and

$$\delta^{[l]} = \sigma'\left(z^{[l]}\right) \circ \left(W^{[l+1]}\right)^T \delta^{[l+1]} \text{ for } l = 2, 3, \cdots, L-1. \tag{25}$$

Here, the operator $\circ$ is the Hadamard product, which multiplies two vectors componentwise, i.e. if $x, y \in \mathbb{R}^{n_l}$, then

$$x \circ y = \begin{pmatrix} x_1 y_1 \\ x_2 y_2 \\ \vdots \\ x_{n_l} y_{n_l} \end{pmatrix}. \tag{26}$$

A Simple Example – Minimizing the Cost (Continued)

The vectors $a_1, a_2, \cdots, a_L$ (and $z_2, z_3, \cdots, z_L$) can be calculated successively. Then the previous equations allow $\delta^{[l]}$ and the partial derivatives

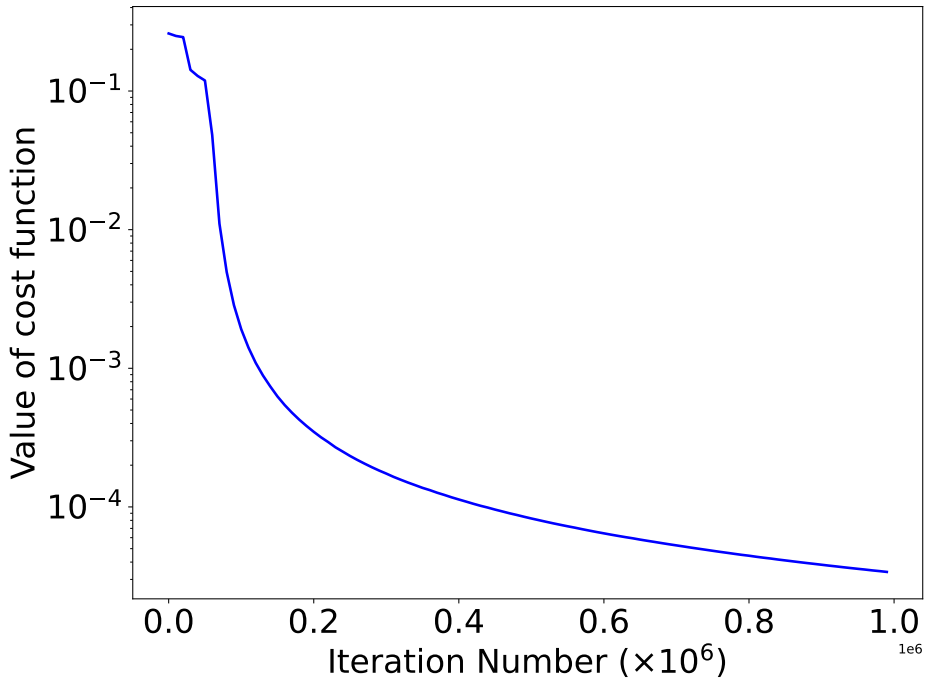
$$\frac{\partial C}{\partial b_j^{[l]}} \quad \text{and} \quad \frac{\partial C}{\partial w_{jk}^{[l]}} \tag{27}$$

to be calculated recursively backward for $l = L, L-1, \cdots, 2$. This is called **backward propagation**.

At this point, we now have everything needed to train the neural network. The script `netbpfull` implements this method. It starts with random weights and biases, uses the learning rate $\eta = 0.05$, and executes a million iterations to find optimal values of the parameters in F.

A Simple Example – Training

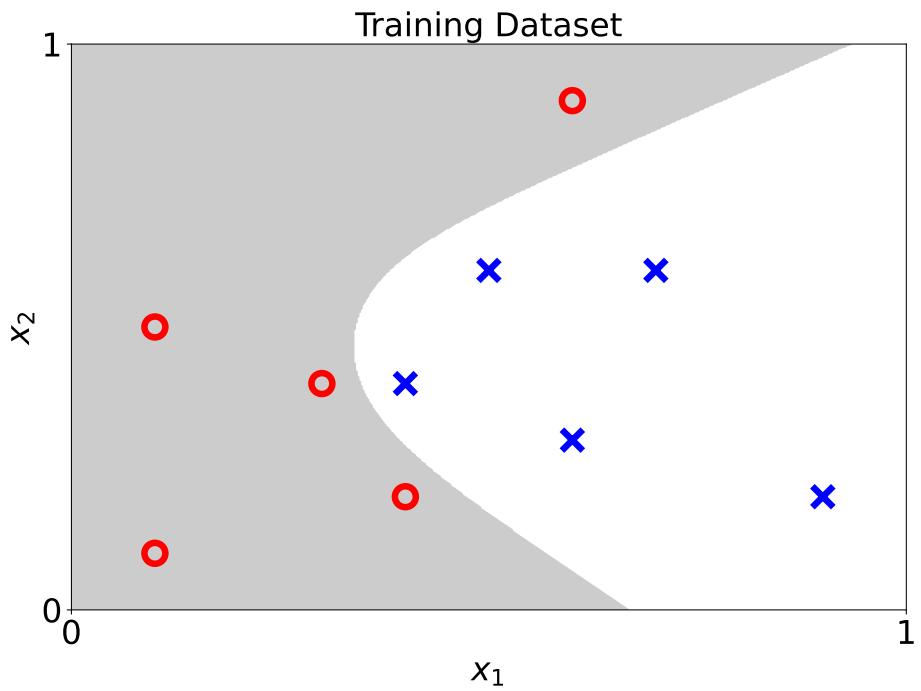
For this particular example, the cost as a function of the iteration number looks like the following.



So, it's clear that the algorithm is performing reasonably well for this example and that most of the improvement occurs before the $200,000^{\text{th}}$ iteration. However, the execution time is fairly long, even for such a small example with only 4 layers and 10 training points.

A Simple Example – Results

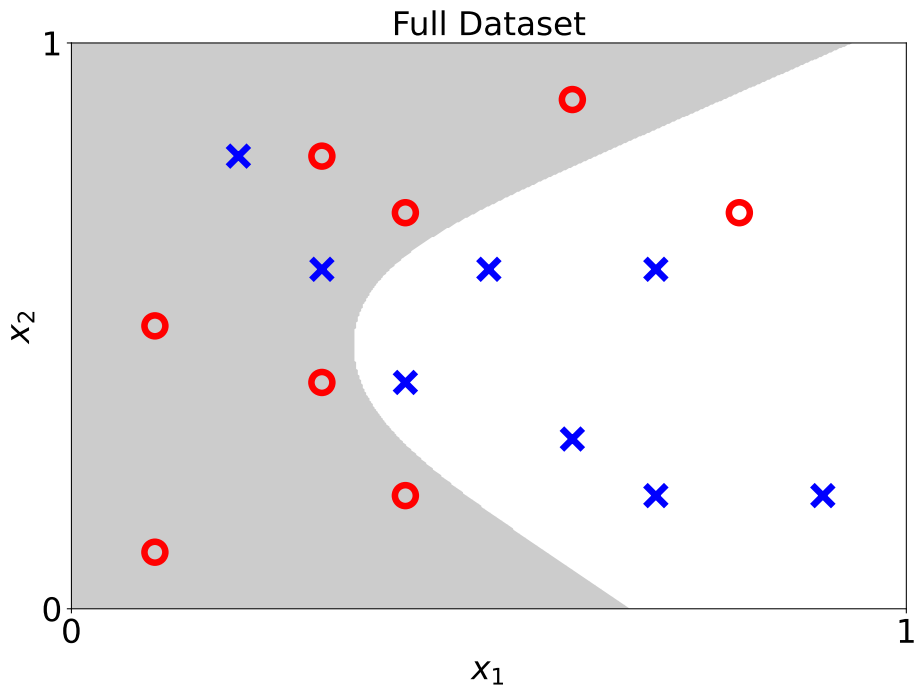
At the end of this process, the function F divides the unit square into the following two regions, where the shaded region corresponds to the circle category and the unshaded region corresponds to the x category.



This plot shows the regions together with the training dataset.

A Simple Example – Results (Continued)

The plot with the full dataset is below.



A Simple Example – Validation

How do we determine which category points are predicted to lie in?

Recall that a point $x \in \mathbb{R}^2$ is predicted to be in the circle category if $F_1(x) > F_2(x)$ and in the x category if $F_1(x) < F_2(x)$. So, to determine the predicted category, we need to evaluate $F(x)$.

			Categories	
Data Point x	$F_1(x)$	$F_2(x)$	**Actual**	**Predicted**
$(0.4, 0.7)$	0.9684	0.0305	circle	circle
$(0.8, 0.7)$	0.0004	0.9996	circle	x
$(0.3, 0.8)$	0.9992	0.0008	circle	circle
$(0.7, 0.2)$	0.0003	0.9997	x	x
$(0.3, 0.6)$	0.9939	0.0059	x	circle
$(0.2, 0.8)$	0.9997	0.0003	x	circle

A Simple Example – Validation (Continued)

The above table can be translated into a confusion matrix.

		Predicted Category	
		circle	x
Actual Category	circle	2	1
	x	2	1

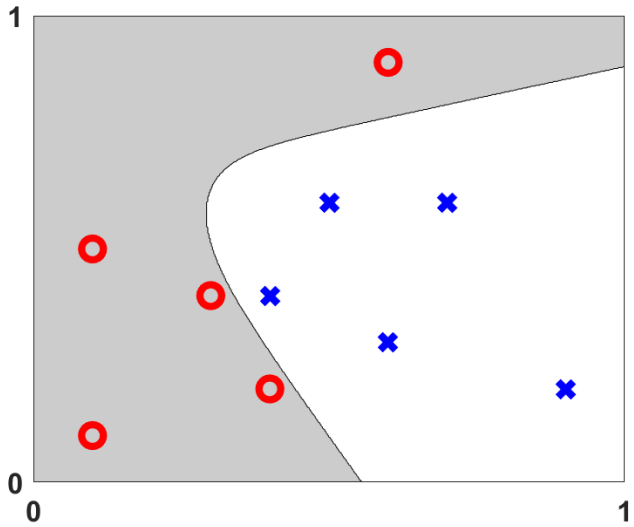
Figure: Confusion matrix for the simple example

The shaded values are the elements of the matrix. In a valid model, the elements on the diagonal would be much larger than the other elements. A confusion matrix shows where there are weaknesses in the model as well as giving a general idea of the validity. For example, in this case, the model performs better for points in the circle category than it does for points in the x category.

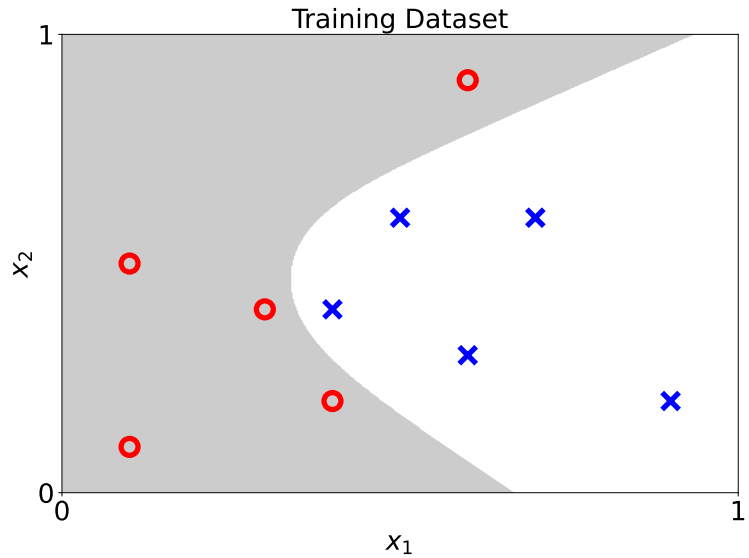
A Simple Example – MATLAB vs Python

The boundary found in the python implementation differs slightly from the one found using MATLAB (and shown in the paper).

MATLAB



Python



Both seem to fit the training set well and the one from python might even fit the training set a little better.

The MATLAB code seems to run quite a bit faster, especially after the small modifications I made to the code.

Vectors, Matrices, and Linear Transformations

Last lecture, I introduced **feature vectors**

$$\mathbf{x}_i = \begin{pmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{Mi} \end{pmatrix} \in \mathbb{F}^M, \tag{28}$$

where $x_{1i}, x_{2i}, \cdots, x_{Mi}$ are the numerical values of the features of $\mathbf{x}_i$, and matrices that contain the feature vectors for a dataset

$$\mathbf{A} = (\mathbf{x}_1 \mathbf{x}_2 \cdots \mathbf{x}_N) = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ x_{M1} & x_{M2} & \cdots & x_{MN} \end{pmatrix} \in \mathbb{F}^{M \times N}. \tag{29}$$

Here, $\mathbb{F}$ is a **field**, which is usually $\mathbb{R}$ or $\mathbb{C}$, and any non-numerical features have been encoded as numerical values.

Examples

- Trees in a forest: Each data point corresponds to a tree. The features are the latitude, longitude, species, age, height, and a measure of the health of the tree.

$$\mathbf{x}_i = \text{ features for tree } i = \begin{pmatrix} -90 \leq x_{1i} \leq 90 \\ -180 \leq x_{2i} \leq 180 \\ x_{3i} = \text{ species number} \\ x_{4i} > 0 \\ x_{5i} > 0 \\ x_{6i} = \text{ percentage} \end{pmatrix} \in \mathbb{R}^6. \quad (30)$$

- Grayscale images of a specific size: Suppose that the dataset consists of grayscale images that are all $K \times L$ pixels in size. Then each image has KL pixels. Let **pixel 1** be the pixel in the **upper left-hand corner** of an image and **number the pixels left-to-right, top-to-bottom**.

Vectors, Matrices, and Linear Transformations (Continued)

If x_{mi} is the **gray level** for image i, then the feature vector for the i^{th} image is

$$\mathbf{x}_i = \begin{pmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{(KL)i} \end{pmatrix} \in \mathbb{Z}^{KL}, \tag{31}$$

where x_{mi} is an integer value with

$$0 \leq x_{mi} \leq 2^B - 1, \tag{32}$$

where B is the **bit-depth** of the gray levels. If there are N images in the dataset, then the data matrix is

$$\mathbf{A} = (\mathbf{x}_1 \mathbf{x}_2 \cdots \mathbf{x}_N) = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ x_{(KL)1} & x_{(KL)2} & \cdots & x_{(KL)N} \end{pmatrix} \in \mathbb{Z}^{(KL)\times N}. \tag{33}$$

Vectors, Matrices, and Linear Transformations (Continued)

Notice that the data matrix has KLN elements. This can be very large for even moderate-sized images. This is why the images and the bit-depth typically used in ML applications are small.

Linear Transformations, Vector Spaces, and Linear Algebra

Encoding features numerically and organizing data in vectors and matrices facilitates applying computational techniques to data. Most algorithms are based on **linear transformations** (also called **linear mappings** or **linear operations**). A linear transformation **maps elements of one vector space to elements of another vector space** and can **always be represented as a matrix**. This means that **linear algebra** is central to machine learning. Some examples showing uses of linear transformations are below.

Solving Systems of Linear Equations

The **main focus of first courses in linear algebra** is finding solutions to systems of linear equations. This means solving equations of the form

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \mathbf{A}\mathbf{x} = \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{pmatrix}. \tag{34}$$

Here, $\mathbf{A} \in \mathbb{F}^{M \times N}$, $\mathbf{x} \in \mathbb{F}^N$, and $\mathbf{b} \in \mathbb{F}^M$. The linear transformation that $\mathbf{A}$ represents maps the **vector space** $\mathbb{F}^N$ to the **vector space** $\mathbb{F}^M$. In both cases, the **operations** of the vector space are **vector addition** and **scalar multiplication by elements of** $\mathbb{F}$. I'll be assuming that you have a firm foundation in this type of problem and won't be reviewing this topic.

Vectors, Matrices, and Linear Transformations (Continued)

Some other simple (and hopefully familiar) examples of linear transformations associated with algorithms that are a little closer to the types we'll discuss are those used in basic signal processing.

Discrete Fourier Transform

The DFT of a DT signal $x[n]$ of length N can be written in matrix form as

$$\begin{pmatrix} X[0] \\ X[1] \\ \vdots \\ X[N-1] \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & W_N^{-1} & \cdots & W_N^{-(N-1)} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & W_N^{-(N-1)} & \cdots & W_N^{-(N-1)^2} \end{pmatrix} \begin{pmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{pmatrix}, \tag{35}$$

i.e. $\mathbf{X} = \mathbf{W}^H \mathbf{x}$, where W_N is the Fourier weight

$$W_N = e^{j2\pi/N}. \tag{36}$$

Discrete-Time Convolution

The time-domain convolution of a finite-length signal $h[n]$ (usually the impulse response of an LTI system) and a finite-length signal $x[n]$ is

$$y[n] = \sum_{k=0}^{K-1} h[k]x[n-k], \tag{37}$$

where $\mathbf{h} \in \mathbb{R}^K$ and $\mathbf{x} \in \mathbb{R}^N$. The sum is the matrix product of a row vector and a column vector for each value of n and the convolution $\mathbf{y}$ can be written as

$$\mathbf{y} = \mathbf{H}\mathbf{x}, \tag{38}$$

where the element of $\mathbf{H}$ in the m^{th} row and n^{th} column is

$$H_{mn} = h[m-n]. \tag{39}$$

Here $\mathbf{y} \in \mathbb{R}^{N+K-1}$.

References I

[1] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical Mathematics*, 2nd ed. Berlin: Springer, 2007, ISBN: 978-3-540-34658-6.

Homework

Read the Higham and Higham paper.

Read Chapter 1 and Sections 2.1 through 2.3 in *Mathematics for Machine Learning*.

Start reading Section 1.1 in Chapter 1 of the draft textbook.