Mathematics Fundamentals

Reid Pryzant

September 3rd, 2015

This section introduces the basic mathematical concepts that are needed to understand deep learning. It represents a bare-bones framework and makes no pretensions about completeness.

1 Algebra

A *function* is a relation between a set of inputs and outputs. Each input has a unique output. You've probably seen functions before. $f(x) = x + 2$ is an example of a function. There are a few ways of thinking about functions, however, that you may not have been exposed to.

A function can be thought of as a *mapping* of inputs to outputs. In this way, a function f is written as

$$f : X \to Y$$

f maps elements $x \in X$ (the $\in$ means "in") to $y \in Y$. The set of possible inputs (in this case, X) is the *domain* of f and the possible outputs (Y) is the *range* of f.

We are used to functions of the form $f : \mathbb{R} \to \mathbb{R}$ (functions that map real numbers to real numbers). However, functions can be abstracted to represent almost any kind of mapping. We will be especially interested in functions of the form $f : \mathbb{R} \to A$ where A is a set of classifications or labels.

2 Linear Algebra

Linear algebra is the mathematics of matrices and vectors. This makes it appropriate for dealing with large systems of equations, something we will encounter frequently during our exploration of deep learning.

A *matrix* is a 2-D array of numbers. Matrices are referred to with bold uppercase letters. The *height* of a matrix is the number of rows in it and the *width* of a matrix is the number of columns. The elements of a matrix have two indices. The first is the row index and the second is the column index. The size of a matrix $\boldsymbol{A}$ is denoted $|\boldsymbol{A}|$. Below is a matrix with height 3, length 2, and size 6.

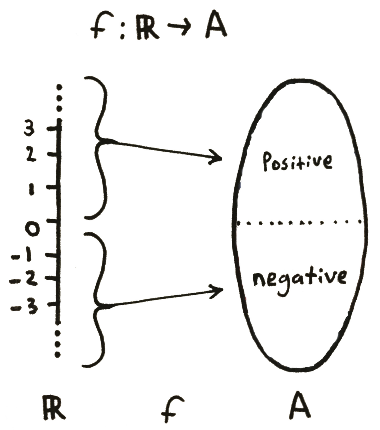


Figure 1: In this case, f is a function that assigns one of two labels to every real number. The labels are representative of sign.

We may also refer to this matrix as a "3 by 2" matrix.

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

There are two operations on matrices that you should know:

1. The *transpose* of a matrix is a new matrix whose rows are the columns of the original. The transpose of a matrix $\boldsymbol{A}$ is denoted $\boldsymbol{A}^\intercal$.

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \boldsymbol{A}^\intercal = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \end{bmatrix}$$

2. *Matrix multiplication* is done in a row-by-column fashion. The rows of the multiplicand are combined with the columns of the multiplier:

$$\boldsymbol{AB} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} & a_{21}b_{13} + a_{22}b_{23} \\ a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22} & a_{31}b_{13} + a_{32}b_{23} \end{bmatrix}$$

Higher dimensional arrays of numbers are called *tensors*. These have the same properties and notation as matrices, so we will refer to them simply as matrices for the rest of this book.

A *vector* is a 1-D array of numbers that is signified by a bold lower-case letter. Each element in a vector is known as an *attribute*, and can be identified by its index. A vector can also be thought of as a single column of a matrix. This means that vectors share many properties with matrices, like multiplication:

$$\boldsymbol{x}^\intercal \boldsymbol{v} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \cdots \\ v_n \end{bmatrix} = \sum_{i=1}^{n} x_i v_i$$

An invaluable property of vectors is their spatial interpretation. A vector can be placed into a coordinate system. In this interpretation, each vector can be visualized as an arrow pointing from the origin to whatever point is specified by the vector's elements. The number of dimensions a vector exists in is equal to its size (the number of attributes).

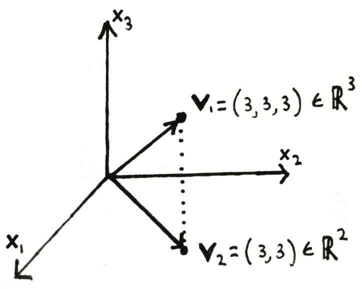

x_3
$v_1 = (3,3,3) \in \mathbb{R}^3$
x_2
x_1
$v_2 = (3,3) \in \mathbb{R}^2$

Figure 2: Vector-space visualization of two vectors $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$.

Vectors can be added to each other (if they are the same dimension) and multiplied by *scalars*, or numbers. Adding two vectors is akin to attaching two arrows to each other in our coordinate system, while multiplying a vector by a number x is equivalent to scaling the arrow by x.

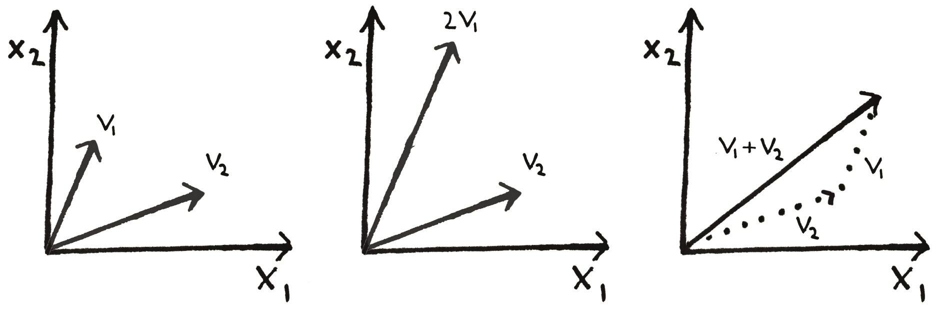


Figure 3: Vectors can be scaled *(middle)* or summed *(right)*.

3 Multivariable Calculus

The *derivative* of a function $f(x)$ measures the slope of the line tangent to f. It is written $\frac{df}{dx}$ or $f'(x)$ because you are taking the derivative of f *with respect to* x.

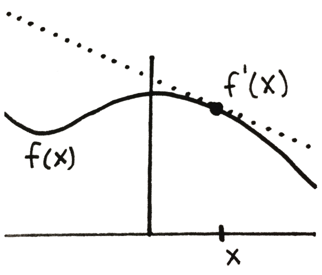


Figure 4: A pointwise derivative of a function f at a location x.

Many functions take several inputs. Some functions can even take a vector or matrix as input. If this is the case and we want to see how the function depends on one of its many inputs, we can take a *partial derivative*. If f takes many inputs such that $f(x_1, x_2, ..., x_n) = \cdots$, then we say that $\frac{\partial f}{\partial x_i}$ is the derivative of f *with respect to* x_i.

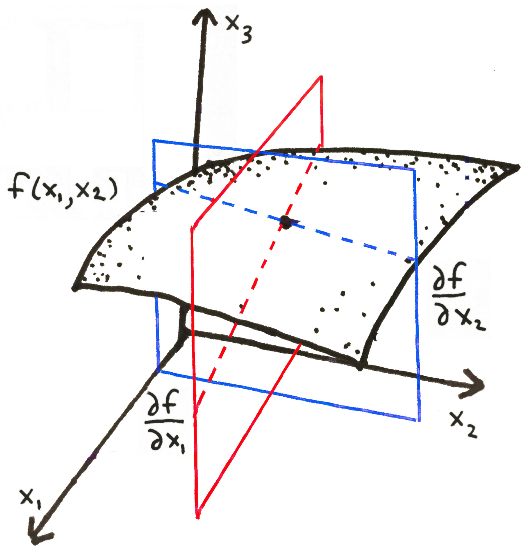

x_3
$f(x_1, x_2)$
$\frac{\partial f}{\partial x_2}$
$\frac{\partial f}{\partial x_1}$
x_2
x_1

Figure 5: Partial derivatives of a function f with respect to each of its inputs x_1 and x_2.

The derivative of a vector produces a special matrix called the *Jacobian*. If f is a function such that $f : \mathbb{R}^n \to \mathbb{R}^m$ (f takes vectors of length n as input and produces vectors of length m as output), then the Jacobian of f (denoted $\boldsymbol{J}$) is a matrix with m rows and n columns, where $j_{ij} = \frac{\partial f_i}{\partial x_j}$.

4 Machine Learning Fundamentals

Deep learning is a branch of machine learning. In order to build an understanding of deep learning, then, we need background on the basic principles of machine learning.

A machine learning algorithm is one that is capable of improving automatically from experience. In most cases, "experience" in this context means data, but how does an algorithm actually go about learning how to do things? In answering this question, it is advantageous to formalize our discussion. A popular definition of learning in the context of algorithms and code is as follows:

Definition: "A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [**?**].

In other words, when a program is trying to accomplish some task, it measures its performance as it runs. A program *learns* when its performance on this task improves over time. For example, if a computer program is learning how to play ping pong with a robotic arm, the task is *playing ping pong*, its performance is measured by its *ability to win*, and it accrues experience by *playing games of ping pong*.

By this point, we have assembled the needed framework to understand our first machine learning algorithm: *linear regression*. It is a simple example that underlines many of the fundamental ideas we will encounter. It is also a good chance to wet our toes with the kind of mathematical thinking we will encounter during our exploration of deep learning. Even if you know what linear regression is, it is valuable and refreshing to reconsider from a machine learning perspective.

In the case of linear regression the task is predicting the value of a scalar $y \in \mathbb{R}$ (a number) based on a vector input $\boldsymbol{x} \in \mathbb{R}^{n+1}$ (several numbers denoted $x_0,\ x_1,\ x_2, ...,\ x_n$ where $x_0 = 1$). This task can be thought of as follows: we have a bunch of observations $(\boldsymbol{x}, y)$. We assume the y's are generated by a function $f(\boldsymbol{x}) = y$. We want to learn what f is, so we develop an approximation of f called $\hat{f}$. Note that our $\hat{f}$ will produce approximations of the y's: $\hat{f}(\boldsymbol{x}) = \hat{y}$. In a machine learning context, f is often assumed to be a probability distribution that is generating each y. $\hat{f}$ maps $\boldsymbol{x}$ to $\hat{y}$ by taking a linear combination of $\boldsymbol{x}$ and its *parameters* $\boldsymbol{w}$. This means that

$$\begin{aligned}
\hat{f}(\boldsymbol{x}) &= \hat{y} \\
&= \boldsymbol{w}^\intercal \boldsymbol{x} \\
&= \sum_{i=0}^{n} w_i x_i \\
&= w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n
\end{aligned}$$

where $\boldsymbol{w} \in \mathbb{R}^{n+1}$ is a vector of parameters. In a machine learning context, the parameters are known as *weights*. Weights control the behavior of our predictions. Each weight w_i controls how much power the i^{th} input has in determining the output $\hat{y}$.

We now have a good grasp on our task: to produce a mapping of $\boldsymbol{x}$'s to $\hat{y}$'s such that $\hat{y} = \boldsymbol{w}^\intercal \boldsymbol{x}$. An example of this might be predicting the probability of a honey bee swarm given some recent weather data and hive demographics.

Now we need to be able to track our performance. In linear regression, it makes sense to equate performance with *error*. Good performance is a small error rate, and poor performance is a large error rate. Coming up with a performance measurement, then, is analogous to measuring the error made by our model.

A first attempt at defining the error E might simply be the difference between the prediction of our model and the value we are trying to predict:

$$\begin{aligned} E(\boldsymbol{x}) &= f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}) \\ &= y - \hat{y} \end{aligned}$$

The problem with this is that if $\hat{y} > y$, then E will be negative. This doesn't make sense (what does a negative error mean?), so we might redefine a new error, E', that takes the absolute value of this difference

$$\begin{aligned} E'(\boldsymbol{x}) &= |f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})| \\ &= |y - \hat{y}| \end{aligned}$$

This works, but our intuition tells us that the more wrong a prediction is, the worse its error should be. This leads us to the *squared error* measure, one we will use later in our discussion of neural networks:

$$\begin{aligned} E''(\boldsymbol{x}) &= (f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}))^2 \\ &= (y - \hat{y})^2 \end{aligned}$$

Now that we have a task and a performance measurement, we need to think about what *experience* means in the context of regression. Experience is the same thing as data. More experience is more data. The data come in two types. The first, $\boldsymbol{X}^{m+1 \times n}$, is a special matrix called the *design matrix*. It consists of a list of n *observations* $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, ..., $\boldsymbol{x}_n$. Each observation is a single vector or data point. Observations consist of m *attributes* $(1, x_1, x_2, ..., x_m) \in \boldsymbol{x}$. In a graphical interpretation, each $x_i \in \boldsymbol{x}_k$ is a coordinate for the i^{th} dimension for data point k.

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x_1} & ... & \boldsymbol{x_n} \end{bmatrix} = \begin{bmatrix} 1 & 1 & ... & 1 \\ x_{11} & x_{12} & ... & x_{1n} \\ ... & ... & ... & ... \\ x_{m1} & x_{m2} & ... & x_{mn} \end{bmatrix}$$

The second data matrix is $\boldsymbol{y}$. This is a vector of regression targets – things we are trying to predict.

We can expand our definition of error so that it takes this data into account. The result, *mean squared error* (*MSE*), is an average of all the E'' values each $\boldsymbol{x}_i$:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \left(f(\boldsymbol{x}_i) - \hat{f}(\boldsymbol{x}_i) \right)^2$$

Notice that the format our linear regression problem has taken is now the same as the kinds of problems machine learning algorithms try to solve. We have a bunch of stuff to be done (predict $\hat{y}$'s from all of the $\boldsymbol{x}_i$'s) and a bunch of stuff that's already done (y_i's for each $\boldsymbol{x}_i$). All that remains is to show the model this data and *teach* it how to improve its predictions.

When we say *learn from data*, we mean leveraging our experience to improve performance. With regards to linear regression this means using our data ($\boldsymbol{X}$ and $\boldsymbol{y}$) to find the weights $\boldsymbol{w}$ that perform the best (produce the smallest MSE). This process is called *optimization*.

There are many ways to optimize for $\boldsymbol{w}$, each with their own interpretation. My personal favorite comes from linear algebra. To start, lets put the task at hand into matrix notation. We are trying to solve for

$$\hat{\boldsymbol{y}} = \boldsymbol{X}^\intercal \boldsymbol{w}$$

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \cdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & ... & x_{1m} \\ ... & ... & ... & ... \\ 1 & x_{n1} & ... & x_{nm} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ ... \\ w_m \end{bmatrix}$$

so that our choice for $\boldsymbol{w}$ produces the optimal $\hat{\boldsymbol{y}}$. This can be rewritten as solving

$$\boldsymbol{y} = \boldsymbol{X}^\intercal \boldsymbol{w} + \boldsymbol{\epsilon}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & ... & x_{1m} \\ ... & ... & ... & ... \\ 1 & x_{n1} & ... & x_{nm} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ ... \\ w_m \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ ... \\ \epsilon_n \end{bmatrix}$$

where the individual ϵ_i's are the *error terms* for each observation i:

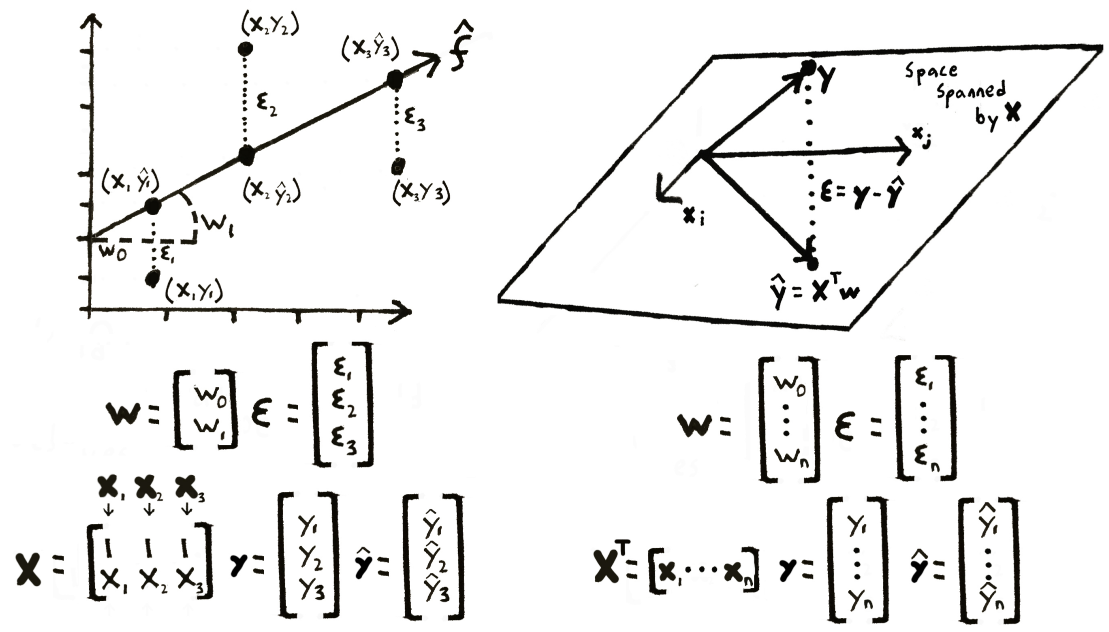
$$\epsilon_i = (y_i - \hat{y}_i)^2$$

Optimizing $\hat{f}$, then, becomes the task of choosing a $\boldsymbol{w}$ such that the mean of all ϵ_i's is minimized.

Now for the fun stuff. Remember that each vector has a visual interpretation as a point in a coordinate system. The entire $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ vectors are merely points in n-dimensional space (because $|\boldsymbol{y}| = |\hat{\boldsymbol{y}}| = n$). Each observation is a point in $m + 1$ dimensional space (because each $\boldsymbol{x}_i \in \boldsymbol{X}$ has length $m + 1$). Running data points through $\hat{f}$ is equivalent to taking a *linear combination* of each $\boldsymbol{x} \in \boldsymbol{X}$ because we can only perform linear operations on each observation: we can multiply attributes by weights and sum the results. This means that the only $\hat{\boldsymbol{y}}$'s that are possible are those reachable by taking different linear combinations of our $\boldsymbol{x}$'s. All of the possibilities for $\hat{\boldsymbol{y}}$, then, create a region within our $m + 1$ dimensional space. This region is known as the space *spanned* by $\boldsymbol{X}$.

The process of optimizing $\boldsymbol{w}$ now has a nice visual interpretation to it. Because $\boldsymbol{y}$ exists in n dimensions and the space spanned by $\boldsymbol{X}$ exists in only $m+1$

dimensions, we need to *project* $\boldsymbol{y}$ into this lower-dimensional world (note that we are assuming $n > (m+1)$). We do this by selecting the $\boldsymbol{w}$ that makes $\hat{\boldsymbol{y}} = \boldsymbol{X}^\mathsf{T}\boldsymbol{w}$ as close as possible to $\boldsymbol{y}$!



(x_2, y_2)
$(x_3, \hat{y}_3)$
$\hat{f}$
ε_2
ε_3
$(x_1, \hat{y}_1)$
$(x_2, \hat{y}_2)$
(x_3, y_3)
w_1
w_0
ε_1
(x_1, y_1)
y
Space spanned by X
x_j
x_i
$\varepsilon = y - \hat{y}$
$\hat{y} = X^T w$
$w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$ $\varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \end{bmatrix}$
x_1 x_2 x_3
$X = \begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \end{bmatrix}$ $y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$ $\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix}$
$w = \begin{bmatrix} w_0 \\ \vdots \\ w_n \end{bmatrix}$ $\varepsilon = \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix}$
$X^T = [x_1 \cdots x_n]$ $y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$ $\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{bmatrix}$

Figure 6: Two interpretations of linear regression. To the *left* is the coordinate-space interpretation of regression with 1-dimensional inputs. To the *right* is the vector-space interpretation of regression with high dimensional inputs.

By selecting a $\boldsymbol{w}$ in this way, our algorithm is learning how to improve (produce better predictions that minimize error) by using experience, or data.

This concludes our discussion of linear regression. The core idea that is at the heart of this algorithm, improving with experience, is one that is central to deep learning and indeed all of machine learning.