QBasics Lecture notes

Ben McDonough

July 2024

1 Probabilistic Computing and Linear Algebra

1.1 Mathematical model of classical computing

In classical computing, we start with a state space consisting of all possible sequences of bits on which we can perform a computation. A *bit*, or *binary digit*, is the smallest unit of information. A bit b is a member of the space $\{0,1\}$, representing a set with two possible values. This is the smallest possible 'alphabet,' of which b is a single letter. A bit-string of length n is a sequence of bits, which we represent as a sequence $\underline{b} \in \{0,1\}^n$. The number of possible length-n bitstrings, or the cardinality of $\{0,1\}^n$, is 2^n. Numbers can be represented in binary by matching a digit b_i at the i^{th} place in $\underline{b}$ with 2^i and then summing over all of the digits, e.g. $(b_{n-1}, \dots, b_1, b_0) \mapsto 2^{n-1}b_{n-1} + \cdots + 2^1 b_1 + 2^0 b_0$. For example,

$$00 \mapsto 0$$
$$01 \mapsto 1$$
$$10 \mapsto 2$$
$$11 \mapsto 3$$

A *computation* is a map $f : \{0,1\}^n \mapsto \{0,1\}^m$, denoting that fact that f takes in a binary input and "spits out" another binary output. The cardinality of the number of possible computations is 2^{m2^n}, which grows very quickly with n and m. We can represent a computation, or binary function, with an inputs-to-outputs table called a *truth table.* To add

$\underline{b}$	$f(\underline{b})$
$00\dots00$	$f(00\dots00)$
$00\dots10$	$f(00\dots01)$
$00\dots10$	$f(00\dots10)$
$\vdots$	$\vdots$
$11\dots11$	$f(11\dots11)$

structure to the impassible size of this space, and for implementation on hardware, large computations are typically built from smaller computations, the simplest of which are basic *logic gates.* The simplest logic gate is a NOT gate. This gate flips a given bit. Symbolically, this is often denoted with a line overtop: $\text{NOT}(b) = \overline{b}$ or $\neg b$. Other important gates to know are the AND, OR and XOR gates. The AND gate compares two bits and outputs 1 if *both* inputs are 1, and 0 otherwise. In notation, we write $\text{AND}(a,b) = a \cdot b$ or $a \wedge b$. The OR gate compares two bits and outputs 0 only if both are 0, outputting 1 otherwise. Notationally, we write $\text{OR}(a,b) = a + b$ or $a \vee b$. In fact, NOT and AND together can be used to compute any desired binary function, a property called *universality.* Lastly, the XOR gate, standing for exclusive-or, takes two inputs and outputs 1 only if exactly one input is 1. This is the same as addition mod-2, and is typically denoted $\text{XOR}(a,b) = a \oplus b$. The truth tables for these gates are shown below.

b	$\text{NOT}(b)$
0	1
1	0

ab	$\text{AND}(a,b)$
00	0
01	0
10	0
11	1

ab	$\text{OR}(a,b)$
00	0
01	1
10	1
11	1

ab	$\text{XOR}(a,b)$
00	0
01	1
10	1
11	0

For more fun with these logic gates, check out the NAND game, where you are guided through building a computer from scratch: https://nandgame.com/. One important mathematical tool for simplifying logical expressions is called DeMorgan's laws. These relations are succinctly expressed as the following:

$$(1)\ \ \overline{a \cdot b} = \overline{a} + \overline{b}$$
$$(2)\ \ \overline{a + b} = \overline{a} \cdot \overline{b}$$

This can help simplifying more complicated functions, such as the following:

$$\overline{a \cdot b + c \cdot d} = \overline{a \cdot b} \cdot \overline{a \cdot b} = (\overline{a} + \overline{b}) \cdot (\overline{c} + \overline{d})$$

1.2 Reversible computation

Lastly, for reasons which will make sense later, we need the notion of *reversible computing.* A reversible computation f is one for which f^{-1} is well-defined, i.e. f maps every input to exactly one output (injective) and every output has some input mapped to it (surjective.) Such a function is invertible. It turns out that simply by adding more bits to the output space, we can turn any non-reversible computation into a reversible one, using the following scheme: Let $f : \{0,1\}^n \to \{0,1\}^n$ by an arbitrary computation. Then let $g : \{0,1\}^{2n} \to \{0,1\}^{2n}$ be defined by

$$g(\underline{b}, \underline{x}) = (f(\underline{b}) \oplus \underline{x}, \underline{b})$$

where the XOR between the sequences $f(\underline{b})$ and $\underline{x}$ is element-wise. The function $g^{-1} : (\underline{b}, \underline{x} \mapsto (\underline{x}, \underline{b} \oplus f(\underline{x}))$ is the well-defined inverse. If we set $\underline{x} = (0, 0, \ldots, 0)$, then $g(\underline{b}, 0) = (f(\underline{b}), \underline{b})$, so we recover the original function by looking at the first n bits of the output.

1.3 Computing with probabilities

Imagine a coin is placed in a closed box. The coin is weighted so that it lands on heads with probability p. The state of the coin is then defined by a distribution over the possible outcomes, $\{\text{heads}, \text{tails}\}$, which we will call $\{0,1\}$ going forward. The distribution is a function $P : \{0,1\}^n \mapsto [0,1]$, where $P(0) = p$ and $P(1)$ is determined by the condition that the probabilities add to 1, so $P(1) = 1 - p$. We can write this function in vector form as

$$P = \begin{pmatrix} P(0) \\ P(1) \end{pmatrix} = \begin{pmatrix} p \\ 1-p \end{pmatrix} = p\begin{pmatrix} 1 \\ 0 \end{pmatrix} + (1-p)\begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Notice that the basis vectors $\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right)$ represent the two definite outcomes. Now imagine this scenario: after the coin comes out of the box, we flip the coin and then look at it. What is the distribution over the outcomes? Intuition tells us that the coin should now be in the state 1 with probability p and 0 with probability $1 - p$, and this is indeed correct. Remarkably, we see that the NOT gate is actually a map on the *probability distributions*, which we will call $X : \mathcal{P}(\{0,1\}) \to \mathcal{P}(\{0,1\})$, where $\mathcal{P}(\{0,1\})$ denotes the set of all possible probability distributions on $\{0,1\}$. So what kind of map is X? We notice that

$$X\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right) = \left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right) \qquad\qquad X\left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right) = \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$$

because NOT flips $0 \to 1$ and $1 \to 0$. Furthermore, because the outcome of applying NOT to the coin in the box is the probability distribution $p\left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right) + (1-p)\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$, which represents the coin now in the state 1 with probability p, we in fact have

$$X(P) = X\left(p\begin{pmatrix} 1 \\ 0 \end{pmatrix} + (1-p)\begin{pmatrix} 0 \\ 1 \end{pmatrix}\right) = p\begin{pmatrix} 0 \\ 1 \end{pmatrix} + (1-p)\begin{pmatrix} 1 \\ 0 \end{pmatrix} = pX\begin{pmatrix} 1 \\ 0 \end{pmatrix} + (1-p)X\begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

This property of distributing over sums and scalar multiplication is called linearity, and this makes X a linear function. Therefore, we can represent X as a matrix, i.e.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

We have succeeded in a complete representation of the NOT operation that captures its behavior when the exact state of the coin is unknown.

1.4 Probabilistic gates

So far, we have started with a random bit and then performed an operation with a deterministic outcome. However, consider the situation where we start with a bit in the state 0, and flip it with probability p. Call this operation X_p. Clearly the outcome distribution should be

$$X_p\begin{pmatrix} 1 \\ 0 \end{pmatrix} = (1-p)\begin{pmatrix} 1 \\ 0 \end{pmatrix} + p\begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Similarly, if we start in the state 1, the outcome distribution is

$$X_p\begin{pmatrix} 0 \\ 1 \end{pmatrix} = p\begin{pmatrix} 1 \\ 0 \end{pmatrix} + (1-p)\begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

This allows us to write X_p as a matrix:

$$X_p = \begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix}$$

This allows us to easily answer more complicated questions about probabilistic operations on probabilistic bits. For example, consider a randomized bit with bias p_1, and apply the operation X_{p_2}. The outcome state would be

$$X_{p_2}\begin{pmatrix} p_1 \\ 1-p_1 \end{pmatrix} = \begin{pmatrix} 1-p_2 & p_2 \\ p_2 & 1-p_2 \end{pmatrix}\begin{pmatrix} p_1 \\ 1-p_1 \end{pmatrix} = \begin{pmatrix} p_1(1-p_2)+(1-p_1)p_2 \\ p_2p_1+(1-p_1)(1-p_2) \end{pmatrix}$$

We have additionally furnished an interpretation of operations themselves as probability distributions, e.g. applying a bit-flip with probability p. Fascinatingly, both our bits *and* our logic gates form their own probability spaces!

1.5 Two-bit states

Now suppose we have two bits a and b which are weighted by p_a and p_b. Both of these coins are placed in the box together and then shaken. How would we describe the outcome distribution? Clearly they are both independent of each other, so

$$P_a = \begin{pmatrix} p_a \\ 1-p_a \end{pmatrix} \qquad\qquad P_b = \begin{pmatrix} p_b \\ 1-p_b \end{pmatrix}$$

Together, they define a join distribution over the possibilities $\{00, 01, 10, 11\}$, which we can also write down as a vector, using the rule that independent probabilities are multiplicative:

$$P_{ab} = \begin{pmatrix} P_a(0)P_b(0) \\ P_a(0)P_b(1) \\ P_a(1)P_b(0) \\ P_a(1)P_b(1) \end{pmatrix} = \begin{pmatrix} P_a(0)\begin{pmatrix} P_b(0) \\ P_b(1) \end{pmatrix} \\ P_a(1)\begin{pmatrix} P_b(0) \\ P_b(1) \end{pmatrix} \end{pmatrix} \equiv \begin{pmatrix} P_a(0) \\ P_a(1) \end{pmatrix} \otimes \begin{pmatrix} P_b(0) \\ P_b(1) \end{pmatrix}$$

This is more than just a single example–we have introduced the *kronecker product* $\otimes$, and the pattern above can be generalized to as many bits as necessary. Notice that the kronecker product provides an explicit formula for consistently making smaller vectors into bigger ones, i.e.

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

One can see that the vectors on the right are counting 0-1-2-3 based on where the 1 is placed, and the vectors on the right are counting in binary 00-01-10-11.

1.6 Creating correlations

So far, we have only considered operations that affect a single bit. We can now ask what happens when we apply an operation such as XOR to two probabilistic bits. First, we will make this gate reversible: we will define a new gate, $\text{CNOT}(a, b) = (a, a \oplus b)$. This gate computes $a \oplus b$ in the second bit and leaves the first one unchanged. The "C" in CNOT stands for control, indicating that bit a controls whether b is flipped. Suppose that the first bit, a, is randomized to be in the state 0 or 1 with equal probability, so $P_a = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$, and the second bit b is in the state 0. After applying CNOT, what probability distribution will describe the two bits? This time, the probabilities are not uncorrelated, because b is flipped *only if* a is 1. Therefore the output distribution will be $(0, 0)$ with 50% probability and $(1, 1)$ with 50% probability, so

$$P_{ab} = \frac{1}{2}\begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Let us now consider the matrix representing CNOT, which we will call CX. Let the state of bit b be the vector $\vec{p}$. Then

$$\text{CX}\begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \vec{p} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \vec{p} \qquad\qquad \text{CX}\begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \vec{p} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes X(\vec{p})$$

This shows us that CX takes the block form

$$\text{CX} = \begin{pmatrix} I & 0 \\ 0 & X \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

where I is the identity operation, which does nothing. We profit greatly from this block structure, because it easily generalizes to as many controls as we want. For example, a gate on three bits a, b, c which flips c if and only if both a and b are 1 can be written as

$$\text{CCX} = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & X \end{pmatrix}$$

This is known as a Toffolli gate, and it plays a fundamental role in quantum computing.

2 Quantum leap of faith

The way we transition to quantum mechanics is by replacing the probabilities of 0 and 1 with complex numbers. The situation can be reduced to the following two basic axioms:

(1) *Qubit states* are represented by vectors with complex amplitudes, i.e. $\psi = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$, where α and β are complex.

(2 If the qubit is "measured," the state "collapses" into $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ with probability $|\alpha|^2$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ with probability $|\beta|^2$.

The measurement axiom, called the "Born rule," is necessary to interpret the complex vector ψ as a probability distribution over outcomes. It certainly bears a similarity to the notion of a probabilitstic bit, which is probabilistic until one observes it and records the outcome. This similarity is misleading, because the coin in the box after being shaken *did* have a value 0 or 1, but through our lack of knowledge we were forced to describe it through a probability distribution. The qubit state is *fundamentally* different in that before observation, it was neither in the state 0 or 1, but literally in the state ψ. This is a well-tested fact that starkly separates quantum probability from classical probability.

The most important mathematical consequence of this rule is that $|a|^2 + |b|^2 = 1$. A qubit state ψ, which is commonly called a *wavefunction* in connection to physics, is called *normalized* when it satisfies this requirement.

2.1 Digression on complex numbers

If you have not used complex numbers before, they may seem daunting; However, they are really a very natural extension of the real numbers. To understand what makes the complex numbers special, note that there are very simple polynomials with real coefficients that have no real roots; for instance, $x^2 + 1 = 0$ has no real solutions. The solutions, if they existed, would be $\pm\sqrt{-1}$. To construct the complex numbers, we simply incorporate $\sqrt{-1}$ to the reals, and we give it the special symbol $i \equiv \sqrt{-1}$. In other words, every complex number $z \in \mathbb{C}$ can be constructed as $z = a + bi$, where a, b are both real numbers. Then, every polynomial of degree n with coefficients in $\mathbb{C}$ has exactly n (potentially non-distinct) complex roots. This property makes $\mathbb{C}$ *algebraically closed*.

Complex numbers can be added and multiplied similarly to the way real numbers are added and multiplied. If $z_1 = a_1 + b_1 i$ and $z_2 = a_2 + b_2 i$, then their sum and product can be found by FOILing the terms and then grouping together the terms multiplying i:

$$z_1 + z_2 = (a_1 + a_2) + (b_1 + b_2)i$$

$$z_1 z_2 = (a_1 + b_1 i)(a_2 + b_2 i) = a_1 a_2 + b_1 a_2 i + a_1 b_2 i + b_1 b_2 i^2 = (a_1 a_1 - b_1 b_2) + (a_1 b_2 + b_1 a_2)i$$

The two important operations we can perform on a complex number are conjugation and magnitude. If $z = a + bi$, we obtain the conjugate of z by flipping the sign of the imaginary part: $z^* = a - bi$. The magnitude of z is defined as $|z| = \sqrt{a^2 + b^2}$. At this point, it is instructive to view z as a vector in $\mathbb{R}^2$, with a as the $x-$coordinate and b as the y-coordinate. Then the magnitude of z is just the Pythagorean length of this vector. A much more compact way to write the magnitude is as

$$z^* z = (a + bi)(a - bi) = a^2 + b^2 = |z|^2$$

This is a Cartesian picture of z as a vector. We can also represent z in polar coordinates as $z = |z|\cos(\theta) + i|z|\sin(\theta)$, where θ is the angle formed between z and the x-axis. We can simplify this notation further through the Euler formula:

$$\cos(\theta) + i\sin(\theta) = e^{i\theta}$$

This formula is proven in the attached activities. In this way we may write $z = |z|e^{i\theta}$, another compact way to represent complex numbers. The number θ is known as the *argument* of z. We can use this to show that division by complex numbers is well-defined:

$$\frac{1}{z} = \frac{1}{|z|e^{i\theta}} = \frac{1}{|z|}e^{-i\theta}$$

so the reciprocal of a complex number is obtained by taking the reciprocal of the magnitude and flipping the sign of the argument. Using the property that sin is an odd function, i.e. $-\sin(\theta) = \sin(-\theta)$, we have

$$(e^{i\theta})^* = \cos(\theta) - i\sin(\theta) = \cos(\theta) + i\sin(-\theta) = e^{-i\theta}$$

2.2 Dirac notation

There is nothing special about using $\begin{pmatrix}1\\0\end{pmatrix}$ and $\begin{pmatrix}0\\1\end{pmatrix}$ to represent the states 0 and 1. In Dirac notation, we choose the special labels

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix} \qquad\qquad |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$$

These are called "kets," and we typically write a qubit state in this notation as

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

Dirac notation is also useful to compactly represent dual vectors. A dual vector is formed by changing a row vector into a column vector and taking the complex conjugate of each entry, e.g.

$$\langle\psi| = \begin{pmatrix}\alpha^* & \beta^*\end{pmatrix}$$

This is commonly referred to as a "bra," and so together $\langle\psi|$ and $|\psi\rangle$ are called "bras" and "kets." The reason for this odd choice of naming convention will shortly become clear.

2.3 Inner products and brakets

An inner product is a way of comparing two vectors. For instance, if $\vec{u}, \vec{v}$ are real, then the inner product is just the dot product, which is given by $\vec{u} \cdot \vec{v} = \|\vec{u}\|\|\vec{v}\|\cos(\theta)$.