Writing an expression in terms of the trace operator opens up opportunities to manipulate the expression using many useful identities. For example, the trace operator is invariant to the transpose operator:

$$\text{Tr}(\boldsymbol{A}) = \text{Tr}(\boldsymbol{A}^\top). \tag{2.50}$$

The trace of a square matrix composed of many factors is also invariant to moving the last factor into the first position, if the shapes of the corresponding matrices allow the resulting product to be defined:

$$\text{Tr}(\boldsymbol{ABC}) = \text{Tr}(\boldsymbol{CAB}) = \text{Tr}(\boldsymbol{BCA}) \tag{2.51}$$

or more generally,

$$\text{Tr}(\prod_{i=1}^{n} \boldsymbol{F}^{(i)}) = \text{Tr}(\boldsymbol{F}^{(n)} \prod_{i=1}^{n-1} \boldsymbol{F}^{(i)}). \tag{2.52}$$

This invariance to cyclic permutation holds even if the resulting product has a different shape. For example, for $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times m}$, we have

$$\text{Tr}(\boldsymbol{AB}) = \text{Tr}(\boldsymbol{BA}) \tag{2.53}$$

even though $\boldsymbol{AB} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{BA} \in \mathbb{R}^{n \times n}$.

Another useful fact to keep in mind is that a scalar is its own trace: $a = \text{Tr}(a)$.

2.11 The Determinant

The determinant of a square matrix, denoted $\det(\boldsymbol{A})$, is a function that maps matrices to real scalars. The determinant is equal to the product of all the eigenvalues of the matrix. The absolute value of the determinant can be thought of as a measure of how much multiplication by the matrix expands or contracts space. If the determinant is 0, then space is contracted completely along at least one dimension, causing it to lose all its volume. If the determinant is 1, then the transformation preserves volume.

2.12 Example: Principal Components Analysis

One simple machine learning algorithm, **principal components analysis** (PCA), can be derived using only knowledge of basic linear algebra.

Suppose we have a collection of m points $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ in $\mathbb{R}^n$ and we want to apply lossy compression to these points. Lossy compression means storing the points in a way that requires less memory but may lose some precision. We want to lose as little precision as possible.

One way we can encode these points is to represent a lower-dimensional version of them. For each point $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$ we will find a corresponding code vector $\boldsymbol{c}^{(i)} \in \mathbb{R}^l$. If l is smaller than n, storing the code points will take less memory than storing the original data. We will want to find some encoding function that produces the code for an input, $f(\boldsymbol{x}) = \boldsymbol{c}$, and a decoding function that produces the reconstructed input given its code, $\boldsymbol{x} \approx g(f(\boldsymbol{x}))$.

PCA is defined by our choice of the decoding function. Specifically, to make the decoder very simple, we choose to use matrix multiplication to map the code back into $\mathbb{R}^n$. Let $g(\boldsymbol{c}) = \boldsymbol{D}\boldsymbol{c}$, where $\boldsymbol{D} \in \mathbb{R}^{n \times l}$ is the matrix defining the decoding.

Computing the optimal code for this decoder could be a difficult problem. To keep the encoding problem easy, PCA constrains the columns of $\boldsymbol{D}$ to be orthogonal to each other. (Note that $\boldsymbol{D}$ is still not technically "an orthogonal matrix" unless $l = n$.)

With the problem as described so far, many solutions are possible, because we can increase the scale of $\boldsymbol{D}_{:,i}$ if we decrease c_i proportionally for all points. To give the problem a unique solution, we constrain all the columns of $\boldsymbol{D}$ to have unit norm.

In order to turn this basic idea into an algorithm we can implement, the first thing we need to do is figure out how to generate the optimal code point $\boldsymbol{c}^*$ for each input point $\boldsymbol{x}$. One way to do this is to minimize the distance between the input point $\boldsymbol{x}$ and its reconstruction, $g(\boldsymbol{c}^*)$. We can measure this distance using a norm. In the principal components algorithm, we use the L^2 norm:

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} ||\boldsymbol{x} - g(\boldsymbol{c})||_2. \tag{2.54}$$

We can switch to the squared L^2 norm instead of using the L^2 norm itself because both are minimized by the same value of $\boldsymbol{c}$. Both are minimized by the same value of $\boldsymbol{c}$ because the L^2 norm is non-negative and the squaring operation is monotonically increasing for non-negative arguments.

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} ||\boldsymbol{x} - g(\boldsymbol{c})||_2^2. \tag{2.55}$$

The function being minimized simplifies to

$$(\boldsymbol{x} - g(\boldsymbol{c}))^\top (\boldsymbol{x} - g(\boldsymbol{c})) \tag{2.56}$$

(by the definition of the L^2 norm, equation 2.30)

$$= \boldsymbol{x}^\top \boldsymbol{x} - \boldsymbol{x}^\top g(\boldsymbol{c}) - g(\boldsymbol{c})^\top \boldsymbol{x} + g(\boldsymbol{c})^\top g(\boldsymbol{c}) \tag{2.57}$$

(by the distributive property)

$$= \boldsymbol{x}^\top \boldsymbol{x} - 2\boldsymbol{x}^\top g(\boldsymbol{c}) + g(\boldsymbol{c})^\top g(\boldsymbol{c}) \tag{2.58}$$

(because the scalar $g(\boldsymbol{c})^\top \boldsymbol{x}$ is equal to the transpose of itself).

We can now change the function being minimized again, to omit the first term, since this term does not depend on $\boldsymbol{c}$:

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} -2\boldsymbol{x}^\top g(\boldsymbol{c}) + g(\boldsymbol{c})^\top g(\boldsymbol{c}). \tag{2.59}$$

To make further progress, we must substitute in the definition of $g(\boldsymbol{c})$:

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} -2\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{c} + \boldsymbol{c}^\top \boldsymbol{D}^\top \boldsymbol{D}\boldsymbol{c} \tag{2.60}$$

$$= \arg\min_{\boldsymbol{c}} -2\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{c} + \boldsymbol{c}^\top \boldsymbol{I}_l \boldsymbol{c} \tag{2.61}$$

(by the orthogonality and unit norm constraints on $\boldsymbol{D}$)

$$= \arg\min_{\boldsymbol{c}} -2\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{c} + \boldsymbol{c}^\top \boldsymbol{c}. \tag{2.62}$$

We can solve this optimization problem using vector calculus (see section 4.3 if you do not know how to do this):

$$\nabla_{\boldsymbol{c}}(-2\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{c} + \boldsymbol{c}^\top \boldsymbol{c}) = \boldsymbol{0} \tag{2.63}$$

$$-2\boldsymbol{D}^\top \boldsymbol{x} + 2\boldsymbol{c} = \boldsymbol{0} \tag{2.64}$$

$$\boldsymbol{c} = \boldsymbol{D}^\top \boldsymbol{x}. \tag{2.65}$$

This makes the algorithm efficient: we can optimally encode $\boldsymbol{x}$ using just a matrix-vector operation. To encode a vector, we apply the encoder function

$$f(\boldsymbol{x}) = \boldsymbol{D}^\top \boldsymbol{x}. \tag{2.66}$$

Using a further matrix multiplication, we can also define the PCA reconstruction operation:

$$r(\boldsymbol{x}) = g\left(f\left(\boldsymbol{x}\right)\right) = \boldsymbol{D}\boldsymbol{D}^\top \boldsymbol{x}. \tag{2.67}$$

Next, we need to choose the encoding matrix $\boldsymbol{D}$. To do so, we revisit the idea of minimizing the L^2 distance between inputs and reconstructions. Since we will use the same matrix $\boldsymbol{D}$ to decode all the points, we can no longer consider the points in isolation. Instead, we must minimize the Frobenius norm of the matrix of errors computed over all dimensions and all points:

$$\boldsymbol{D}^* = \arg\min_{\boldsymbol{D}} \sqrt{\sum_{i,j} \left(x_j^{(i)} - r(\boldsymbol{x}^{(i)})_j \right)^2} \text{ subject to } \boldsymbol{D}^\top \boldsymbol{D} = \boldsymbol{I}_l. \tag{2.68}$$

To derive the algorithm for finding $\boldsymbol{D}^*$, we start by considering the case where $l = 1$. In this case, $\boldsymbol{D}$ is just a single vector, $\boldsymbol{d}$. Substituting equation 2.67 into equation 2.68 and simplifying $\boldsymbol{D}$ into $\boldsymbol{d}$, the problem reduces to

$$\boldsymbol{d}^* = \arg\min_{\boldsymbol{d}} \sum_i ||\boldsymbol{x}^{(i)} - \boldsymbol{d}\boldsymbol{d}^\top \boldsymbol{x}^{(i)}||_2^2 \text{ subject to } ||\boldsymbol{d}||_2 = 1. \tag{2.69}$$

The above formulation is the most direct way of performing the substitution but is not the most stylistically pleasing way to write the equation. It places the scalar value $\boldsymbol{d}^\top \boldsymbol{x}^{(i)}$ on the right of the vector $\boldsymbol{d}$. Scalar coefficients are conventionally written on the left of vector they operate on. We therefore usually write such a formula as

$$\boldsymbol{d}^* = \arg\min_{\boldsymbol{d}} \sum_i ||\boldsymbol{x}^{(i)} - \boldsymbol{d}^\top \boldsymbol{x}^{(i)} \boldsymbol{d}||_2^2 \text{ subject to } ||\boldsymbol{d}||_2 = 1, \tag{2.70}$$

or, exploiting the fact that a scalar is its own transpose, as

$$\boldsymbol{d}^* = \arg\min_{\boldsymbol{d}} \sum_i ||\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(i)\top} \boldsymbol{d}\boldsymbol{d}||_2^2 \text{ subject to } ||\boldsymbol{d}||_2 = 1. \tag{2.71}$$

The reader should aim to become familiar with such cosmetic rearrangements.

At this point, it can be helpful to rewrite the problem in terms of a single design matrix of examples, rather than as a sum over separate example vectors. This will enable us to use more compact notation. Let $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ be the matrix defined by stacking all the vectors describing the points, such that $\boldsymbol{X}_{i,:} = \boldsymbol{x}^{(i)\top}$. We can now rewrite the problem as

$$\boldsymbol{d}^* = \arg\min_{\boldsymbol{d}} ||\boldsymbol{X} - \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top||_F^2 \text{ subject to } \boldsymbol{d}^\top \boldsymbol{d} = 1. \tag{2.72}$$

Disregarding the constraint for the moment, we can simplify the Frobenius norm portion as follows:

$$\arg\min_{\boldsymbol{d}} ||\boldsymbol{X} - \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top||_F^2 \tag{2.73}$$

$$= \arg\min_{\boldsymbol{d}} \operatorname{Tr}\left(\left(\boldsymbol{X} - \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top\right)^\top \left(\boldsymbol{X} - \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top\right)\right) \tag{2.74}$$

(by equation 2.49)

$$= \arg\min_{\boldsymbol{d}} \operatorname{Tr}(\boldsymbol{X}^\top\boldsymbol{X} - \boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top - \boldsymbol{d}\boldsymbol{d}^\top\boldsymbol{X}^\top\boldsymbol{X} + \boldsymbol{d}\boldsymbol{d}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \tag{2.75}$$

$$= \arg\min_{\boldsymbol{d}} \operatorname{Tr}(\boldsymbol{X}^\top\boldsymbol{X}) - \operatorname{Tr}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) - \operatorname{Tr}(\boldsymbol{d}\boldsymbol{d}^\top\boldsymbol{X}^\top\boldsymbol{X}) + \operatorname{Tr}(\boldsymbol{d}\boldsymbol{d}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \tag{2.76}$$

$$= \arg\min_{\boldsymbol{d}} -\operatorname{Tr}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) - \operatorname{Tr}(\boldsymbol{d}\boldsymbol{d}^\top\boldsymbol{X}^\top\boldsymbol{X}) + \operatorname{Tr}(\boldsymbol{d}\boldsymbol{d}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \tag{2.77}$$

(because terms not involving $\boldsymbol{d}$ do not affect the arg min)

$$= \arg\min_{\boldsymbol{d}} -2\operatorname{Tr}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) + \operatorname{Tr}(\boldsymbol{d}\boldsymbol{d}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \tag{2.78}$$

(because we can cycle the order of the matrices inside a trace, equation 2.52)

$$= \arg\min_{\boldsymbol{d}} -2\operatorname{Tr}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) + \operatorname{Tr}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top\boldsymbol{d}\boldsymbol{d}^\top) \tag{2.79}$$

(using the same property again).

At this point, we reintroduce the constraint:

$$\arg\min_{\boldsymbol{d}} -2\operatorname{Tr}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) + \operatorname{Tr}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top\boldsymbol{d}\boldsymbol{d}^\top) \text{ subject to } \boldsymbol{d}^\top\boldsymbol{d} = 1 \tag{2.80}$$

$$= \arg\min_{\boldsymbol{d}} -2\operatorname{Tr}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) + \operatorname{Tr}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \text{ subject to } \boldsymbol{d}^\top\boldsymbol{d} = 1 \tag{2.81}$$

(due to the constraint)

$$= \arg\min_{\boldsymbol{d}} -\operatorname{Tr}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \text{ subject to } \boldsymbol{d}^\top\boldsymbol{d} = 1 \tag{2.82}$$

$$= \arg\max_{\boldsymbol{d}} \operatorname{Tr}(\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \text{ subject to } \boldsymbol{d}^\top\boldsymbol{d} = 1 \tag{2.83}$$

$$= \arg\max_{\boldsymbol{d}} \operatorname{Tr}(\boldsymbol{d}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{d}) \text{ subject to } \boldsymbol{d}^\top\boldsymbol{d} = 1. \tag{2.84}$$

This optimization problem may be solved using eigendecomposition. Specifically, the optimal $\boldsymbol{d}$ is given by the eigenvector of $\boldsymbol{X}^\top\boldsymbol{X}$ corresponding to the largest eigenvalue.

This derivation is specific to the case of $l = 1$ and recovers only the first principal component. More generally, when we wish to recover a basis of principal

components, the matrix $\boldsymbol{D}$ is given by the l eigenvectors corresponding to the largest eigenvalues. This may be shown using proof by induction. We recommend writing this proof as an exercise.

Linear algebra is one of the fundamental mathematical disciplines necessary to understanding deep learning. Another key area of mathematics that is ubiquitous in machine learning is probability theory, presented next.