1 Linear Algebra

(a) Let $\boldsymbol{A} \in \mathbb{R}^{M \times N}$ and $\boldsymbol{B} \in \mathbb{R}^{P \times N}$. What are the dimensions of $\boldsymbol{C} = (\boldsymbol{A}\boldsymbol{B}^T)^T$?

Vector 2-Norm

Throughout this course, we will often see the sum of squares of a vector. For example, in lecture, we already saw this as part of mean squared error.

We can write the sum of squares of a vector, $\boldsymbol{x} = [x_1, x_2, \cdots, x_M]^T$, as a summation, as a dot product, as a vector product, or as the 2-norm of a vector, squared. All of the following are equal:

$$\sum_{i=1}^{M} x_i^2 = \boldsymbol{x} \cdot \boldsymbol{x} = \boldsymbol{x}^T \boldsymbol{x} = \|\boldsymbol{x}\|_2^2 \tag{1}$$

Note: the 2-norm of a vector (without the square) is defined as $\|\boldsymbol{x}\|_2 = \sqrt{\sum_i x_i^2}$.

(b) For $\boldsymbol{a} = \begin{bmatrix} 2 \\ 5 \\ 3 \end{bmatrix}$, calculate $\|\boldsymbol{a} - 2\|_2^2$.

(c) For $\boldsymbol{z} \in \mathbb{R}^2$ and $\boldsymbol{w} \in \mathbb{R}^2$, expand $\|\boldsymbol{z} - \boldsymbol{w}\|_2^2$, writing it in terms of z_1, z_2, w_2, and w_2.

2 Multidimensional Calculus

Given a function $f(\boldsymbol{x})$ that inputs an M-dimensional vector and outputs a scalar value, the gradient of f, ∇f, is a vector with each entry containing the partial derivative of f with respect to the M different components of $\boldsymbol{x}$:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_M} \end{bmatrix} \tag{2}$$

(a) Let $f(\boldsymbol{x}) = 3x_1^2 \sin x_2$, write the ∇f.

(b) Let $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2$ and $\boldsymbol{x} \in \mathbb{R}^3$.
Write the ∇f in terms of x_1, x_2, and x_3.

Now, write the ∇f in terms of $\boldsymbol{x}$.

(c) Let $f(\boldsymbol{u}, \boldsymbol{v}) = 3\boldsymbol{u}^T\boldsymbol{v}$ and $\boldsymbol{u} \in \mathbb{R}^3$, $\boldsymbol{v} \in \mathbb{R}^3$.
Write $\nabla_{\boldsymbol{u}} f$.

Write $\nabla_{\boldsymbol{v}} f$.

3 Formulating Linear Regression with Linear Algebra

Scalar input

In lecture, we discussed using a linear model and mean squared error with a regression task with a one-dimensional input, $x \in \mathbb{R}$, and a one-dimensional output, $y \in \mathbb{R}$. With a training set of N points, we wrote the objective function, $J(w)$, with summation notation:

$$J(w) = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - wx^{(i)} \right)^2 \tag{3}$$

Note: we are dropping the offset b for simplicity.

(a) Linear algebra is pretty good at simplifying notation by replacing summations with matrix/vector notation. Rewrite the objective function above by packing all training input into an N-dimensional vector, $\boldsymbol{x}$, and all training output into an N-dimensional vector, $\boldsymbol{y}$.

Vector input

Suppose we have more than one input to our regression model. For example, using both the mileage and the age of a car to predict its price. Now each input training point is a vector, $\boldsymbol{x}^{(i)} \in \mathbb{R}^M$, where $M = 2$ for our car example.

Our new linear model will have a weight vector, $\boldsymbol{w} \in \mathbb{R}^M$, containing one weight parameter, w_j for each of the input features, x_j:

$$J(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)} \right)^2 \tag{4}$$

Once again we'll place all of the training output into an N-dimensional vector, $\boldsymbol{y}$. However, we'll need a matrix to fit all of the input data. Specifically, we will pack the input data into an $N \times M$ matrix, $\boldsymbol{X}$, where the i-th row contains the i-th training example, $[x_1^{(i)}, x_2^{(i)}, \cdots, x_M^{(i)}]$.

(b) Rewrite the objective function in terms of $\boldsymbol{y}$, $\boldsymbol{X}$, $\boldsymbol{w}$, and N.

Faster code!

In addition to simpler mathematical representation, formulating problems in linear algebra can lead us to much, *much*, faster code. Rather than the for loop that would correspond to the summation notation, we can use Python libraries such as NumPy or PyTorch to store and compute with vectors, matrices, and even high-dimensional arrays. Libraries like PyTorch are designed to carry out our linear algebra operations with parallel computing on super fast GPUs.

(c) Try implementing either the scalar input or the vector input objective functions above, two different ways, using 1) for loops, and 2) using numpy. See which one is faster!

Pick your own size for N (and M), and use `numpy.random.uniform` to generate some random input and output data. .