1 Background - linear algebra

As a single subfield of mathematics, problems in linear algebra are disproportionately represented: they will be about 60% of what you encounter. As such it is important to be familiar with techniques, methods, and concepts used in linear algebra including (but not limited to) vectors/vector spaces, matrix multiplication, inner products/norms, and transposes. Properties of matricies and vectors that are important to deep learning are given here, and matrix multiplication is outlined below. Due to the limited time in the course we will not cover linear algebra in great detail, but I encourage everyone to take a course in abstract linear algebra to gain proficiency with the subject.

1.1 Matrix multiplication

Consider the $n \times d$ matrix $[A]_{n\times d}$ and column vector $\vec{x}$ whose shape is $1 \times d$. Then the matrix multiplication $A\vec{x} = \vec{y}$ is given by:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & . & . & . & a_{1d} \\ a_{21} & a_{22} & a_{23} & . & . & . & a_{2d} \\ a_{31} & a_{32} & a_{33} & . & . & . & a_{3d} \\ . & & & . & & & . \\ . & & & & . & & . \\ . & & & & & . & . \\ a_{n1} & a_{n2} & a_{n3} & . & . & . & a_{nd} \end{bmatrix}_{n\times d} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \\ x_d \end{bmatrix}_{d\times 1} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + ... + a_{1d}x_d \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + ... + a_{2d}x_d \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + ... + a_{3d}x_d \\ . \\ . \\ . \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + ... + a_{nd}x_d \end{bmatrix}_{n\times 1} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \\ y_d \end{bmatrix}_{n\times 1}$$

The above is the product of a matrix and a vector. In general when given two matricies $[A]_{a\times b}$ and $[B]_{c\times d}$, the matrix product only exists when $b = c$, and the final shape is $[A]_{a\times b}[B]_{c\times d} = [C]_{a\times d}$. This rule of dimensionality is important! It will help us when designing our neural net.

1.2 Background - The rotation operator

Suppose we have a vector, $\vec{x} \in \mathbb{R}^2$ which we wish to rotate around the origin by some angle θ (Fig. 1). This can be expressed in matrix form as:

$$R\vec{x} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}_{2\times 2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{2\times 1} = \begin{bmatrix} \cos(\theta)x_1 - \sin(\theta)x_2 \\ \sin(\theta)x_1 + \cos(\theta)x_2 \end{bmatrix}_{2\times 1} = \begin{bmatrix} x'_1 \\ x'_2 \end{bmatrix}_{2\times 1} = \vec{x'}$$

Here, the matrix R is called the rotation operator.

1.3 Background - The Jacobian matrix

Recall the gradient descent algorithm, where subsequent guesses for the minimum were updated according to:

$$\hat{b}_{i+1} = \hat{b}_i - \alpha \frac{df}{d\hat{b}}|_{\hat{b}_i}$$

where $\hat{b}$ is the parameter we are updating, α is the learning rate, and $\frac{df}{d\hat{b}}$ is the gradient of the objective (i.e. error) function before the parameter update. This formulation is what you have implemented in Lab 1, but is quite restrictive in terms of the classes of functions that it can be used to optimize. Specifically, this formulation only holds for 1-dimensional functions, i.e. single

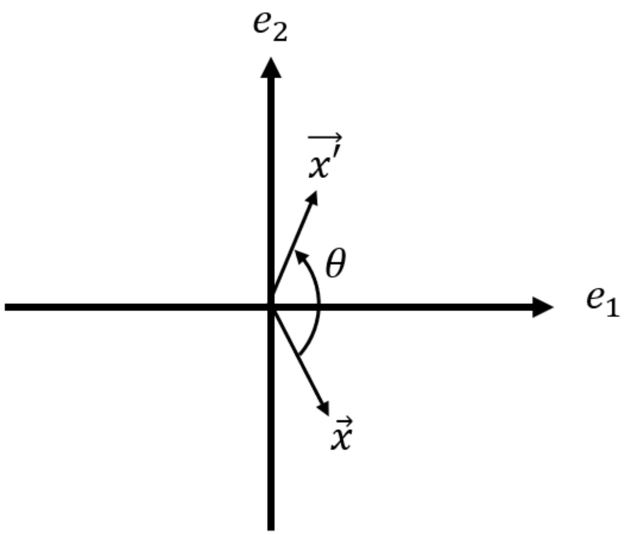

Figure 1: Rotation operator visualization

parameter updates. In practice, the data we work with is not 1-dimensional, but n-dimensional. When this is the case, the 1-dimensional gradient descent algorithm can be written as

$$W_{i+1} = W_i - \alpha \nabla f(W_i)$$

where W_i is the parameter matrix:

$$W_i = \begin{bmatrix} \hat{b}_{11} & \hat{b}_{12} & \hat{b}_{13} & . & . & . & \hat{b}_{1d} \\ \hat{b}_{21} & \hat{b}_{22} & \hat{b}_{23} & . & . & . & \hat{b}_{2d} \\ \hat{b}_{31} & \hat{b}_{32} & \hat{b}_{33} & . & . & . & \hat{b}_{3d} \\ . & & & . & & & . \\ . & & & & . & & . \\ . & & & & & . & . \\ \hat{b}_{n1} & \hat{b}_{n2} & \hat{b}_{n3} & . & . & . & \hat{b}_{nd} \end{bmatrix}_{n \times d}$$

$\nabla f(W_i)$ is the Jacobian matrix, which is the matrix collection of derivatives with respect to each parameter:

$$\nabla f(W_i) = \begin{bmatrix} \frac{df}{d\hat{b}_{11}} & . & . & . & \frac{df}{d\hat{b}_{1d}} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ \frac{df}{d\hat{b}_{n1}} & . & . & . & \frac{df}{d\hat{b}_{nd}} \end{bmatrix}_{n \times d}$$

2 Lab instructions

In this lab, we will be using your gradient descent optimizer from Lab 1 to build a neural net surrogate for the rotation operator.

2.1 Download lab files

Download function files, the Lab 2 boilerplate code, and the lab 2 dataset from Github into your working directory.

2.2 Define your objective function

In the function file, define an objective function which takes in $R\vec{x}$, the neural net predicted value of your data point $\hat{R\vec{x}}$, and outputs a measure of the error. This can be done a number of ways, such as the angular distance, euclidean distance, or any other distance function. The exact objective function will be up to you.

2.3 Define a function which computes the Jacobian matrix

In the function file, define a function which computes the Jacobian of a weight matrix. This may be done by:

1. Use your initial guess for your weight matrix, and your training data, to compute the value of the objective function
2. Perturb a single weight in W by a small amount δ
3. Compute the new value of the objective function with the perturbed weight matrix
4. Measure the gradient between values gained from step 2 and 3
5. Record this gradient in the same index as the perturbed weight
6. Repeat steps 1-5 for all weights

2.4 Build your NN surrogate

The dataset I have included is a collection of 2-dimensional vectors where the target data is original data which has been rotated by some angle θ. Your job is to use the generalized formulation for gradient descent to find a best-estimate (i.e. surrogate model) for R, using the above functions to help.

2.5 Questions

1. What is the dimensionality of the *output* data? What are the dimensions of R and the Jacobian?
2. What was your test/training split?
3. What is the angle θ?
4. Does your choice of objective function impact runtime and accuracy? If so, how?