Gradient Descent

Mothers of Machine Learning Course

March 30, 2022

Tutorial 3

Outline

1. Learning and Optimization
2. Convexity
3. Optimization in Practice
4. Code Examples

Learning and Optimization

How to Quantify Learning

- We have many possible ways of learning about something, maybe by collecting large amounts of data with definite labels for categorization (KNN, Decision Trees), or maybe by proposing an explicit model for how outputs behave as functions of inputs (Regression, Feature Selection), or ....
- How can we get a sense of the difficulty of our learning task?
- How can we know how much we have learned throughout some training process?
- How can we estimate how long these methods will take to learn to a sufficient level, or if they even will learn?

Formulating Problems in terms of Optimization

- It turns out, that almost every problem can be formulated in terms of something called *optimization*.
- Optimization refers to seeking an extreme value of some quantity of interest: Maybe we are trying to *maximize* the profits of a business, or maybe we are trying to *minimize* how long it takes to drive somewhere.
- Optimization problems can also be supplemented by *constraints*: Maybe our business has certain expenses, or quotas that must be met, or our journey in our car can only take certain roads, or must pass by a gas station along the route.

Objective Functions

- Mathematically, we can say that we have an *objective* function $f(x|\mathscr{D})$ that we are trying to optimize (we will focus on minimization):
    - f is the quantity of interest to be optimized (maximize our business profits, minimize how long we are driving for)
    - $\mathscr{D}$ are the inputs (how many customers we have, what is our average speed)
    - x are the parameters of the function that we can tune to optimize (what our business model will be, what route we should take)

VECTOR INSTITUTE

Optimization in Learning

- In learning applications, we generally define an objective *loss* function, for example

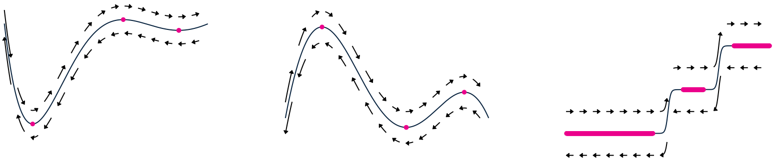
$$\begin{aligned} f(x|\mathscr{D}) &= \left\|y(x|\mathscr{D}) - y(\mathscr{D})\right\|^2 && \text{continuous} \\ f(x|\mathscr{D}) &= y(x|\mathscr{D}) == y(\mathscr{D}) && \text{discrete} \end{aligned} \quad (1)$$

- This quantifies the *difference* between a model's *predictions* $y(x|\mathscr{D})$, and known *labels* $y(\mathscr{D})$, given input data $\mathscr{D}$ and model parameters x.
- We want to define a procedure to *minimize* this difference, to *learn* the best model.
- The specific loss function is very *problem-dependent.*

Convexity

Convexity

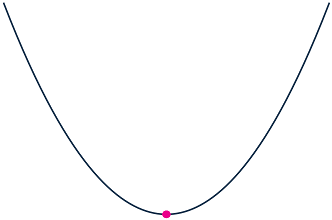
- We first must gain intuition about the peaks and valleys of a function $f(x)$, known as its *curvature*



- How we may reach minimums, depends on this curvature, or how *convex* the function is.
- From calculus, *extremal* values occur when $\frac{\partial f}{\partial x} = 0$, and curvature relates to if $\frac{\partial^2 f}{\partial x^2} > 0$ or $\frac{\partial^2 f}{\partial x^2} < 0$.

Quadratic Functions

- The simplest function that is *strictly convex* is the quadratic function $f(x) = x^2$



- This can be extended to d dimensions with the matrix $A \in R^{d \times d} \quad \rightarrow \quad f(x) = x^T A x.$
- The curvature now depends on the elements A_{ij} .

VECTOR INSTITUTE

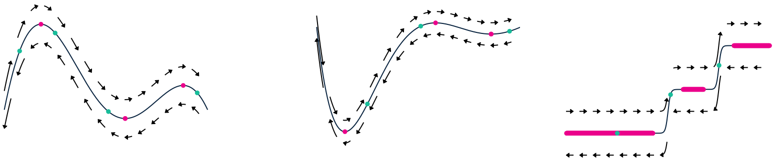
Optimization in Practice

Practical Considerations

- How do we even *compute* derivatives of our model and our loss function? (Analytically, numerically, automatically, ... ; discrete variables, continuous variables, ...)
- How do we know where to *start* in the landscape of possible model parameters?
- How do we know along which *direction* to head if we want to search for the *global* minimum?
- How do we choose a smart procedure that will be as *efficient* as possible? (Least function calls, least number of steps, closest to the true minimum, ...)

Gradient Descent

- To minimize a function, we want to move from our starting point x_0 in a direction *towards* the *global* minimum, avoiding *local* minima.



- We know we want to go to the point x^* where $\frac{\partial f}{\partial x}|_{x=x^*} = 0$, and we know that the gradient points in the direction of *greatest increase* in the function.

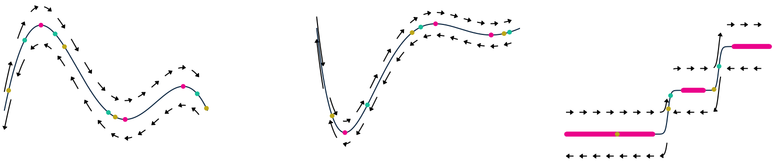
VECTOR INSTITUTE

Iterative Parameter Updates

- Shifting the points in the *opposite* direction of the gradient should move us towards the minimum value:

$$x \to x \; - \; \alpha \, \nabla f(x) \tag{2}$$

- α controls by how much we move in that direction.
- We can keep updating our points until we *converge* to an optimal value at x^*.



Variants of Gradient Descent

- For complicated problems, especially with many parameters, we often risk getting stuck in a *local-minimum*, or regions where the gradients *vanish*.
- To force our way towards the *global-minimum*, we need to update the parameters in a smarter way.
- For example, recalling that balls rolling down a valley can reach the other side if they have enough initial speed, means we can add in terms to the update that account for the *history* of the updates.

$$x \to x' \to x'' = x' \ - \ \alpha \, \nabla f(x') \ - \ \beta \, \nabla f(x) \tag{3}$$

- β controls by how much we move based on the previous gradient, *two steps previous*.

Code Examples

Linear Algebra II

Mothers of Machine Learning Course

April 13, 2022

Tutorial 5

Outline

Recap of Linear Algebra

Types of Objects and Operations

- Scalar: $\alpha \in \mathbb{R}$
- Vector: $v = \{v_i\} \in \mathbb{R}^n$
- Matrix: $A = \{A_{ij}\} \in \mathbb{R}^{n \times m}$
- Tensor: $T = \{T_{ijk\cdots}\} \in \mathbb{R}^{n \times m \times p \times \cdots}$
- Vector-Vector:
$$u \cdot v = \sum_i^n u_i \ v_i \ \in \mathbb{R} \ \text{ for } u, v \in \mathbb{R}^n$$
- Matrix-Vector:
$$A \cdot v = \sum_j^m A_{ij} \ v_j \ \in \mathbb{R}^n \ \text{ for } A \in \mathbb{R}^{n \times m}, \ v \in \mathbb{R}^m$$
- Matrix-Matrix:
$$A \cdot B = \sum_k^m A_{ik} \ B_{kj} \ \in \mathbb{R}^{n \times p} \ \text{ for } A \in \mathbb{R}^{n \times m}, B \in \mathbb{R}^{m \times p}$$
- Symmetric and Orthogonal Matrices: $A^T = A, \quad A^T = A^{-1}$
- Trace and Determinant: $\operatorname{tr} A = \sum_i^n A_{ii}, \ \det A \in \mathbb{R}$

VECTOR INSTITUTE

Coordinate Transformations

- An important application of linear transformations are coordinate transformations.
- For example, let $u = (w, z) \to v = (x, y)$ such that

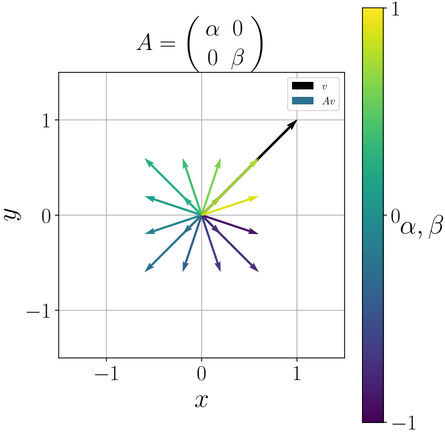
$$v = Ju \tag{1}$$

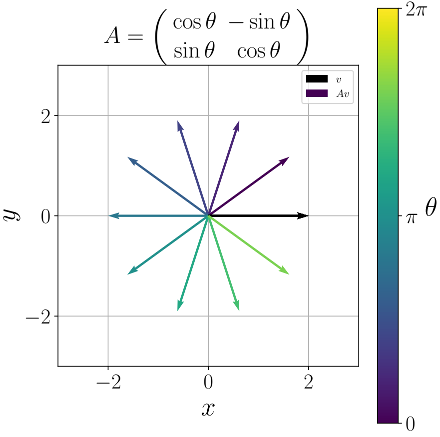
$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} w \\ z \end{pmatrix} = \begin{pmatrix} aw + bz \\ cw + dz \end{pmatrix} \tag{2}$$

- J, or its determinant $\det J$ are sometimes called the *Jacobian* of the transformation.
- For example, we could stretch our coordinates by an amount α and β, or rotate them by an angle θ

$$J = \begin{pmatrix} \alpha & 0 \\ 0 & \beta \end{pmatrix}, \qquad J = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{3}$$

Coordinate Transformations





Coordinate Transformations

- Recall changes of variables for integrals of a single variable $x \to x(t)$:

$$\int f(x)\ dx = \int f(x(t)) \underline{\frac{dx}{dt}} dt \tag{4}$$

and similarly in higher dimensions $(w, z) \to J\ (w, z)$

$$\int f(w, z)\ dw\ dz = \int f(w(x, y), z(x, y))\ \underline{\det J}\ dx\ dy. \tag{5}$$

- The determinant of the transformation $\det J$ ensures the *volumes* over which we are integrating, are equivalent, no matter which coordinates we use.

Eigenvalues and Eigenvectors

Conceptualizing Linear Transformations

- Recall feature selection, that expresses values along *different directions* of functions.
  i.e) $y = \alpha_0 + \alpha_1 x + \alpha_2 x^2$ can be thought of as a sum of $1, x, x^2$ directions.

- Think of matrices A as *transformations* that take input x, and express a function as a linear combination of the *columns* of $A = [a_0, a_1, \cdots a_{m-1}]$, where each column $a_i \in \mathbb{R}^n$.

- $y = Ax = \sum_i^m x_i a_i$ is therefore a value along the directions of the $\{a_i\}$, with steps along each direction of size x_i.

What are Eigenvalues?

- An important question for square $A \in \mathbb{R}^{n \times n}$:
  Along which directions does A transform an input to be along its same direction?

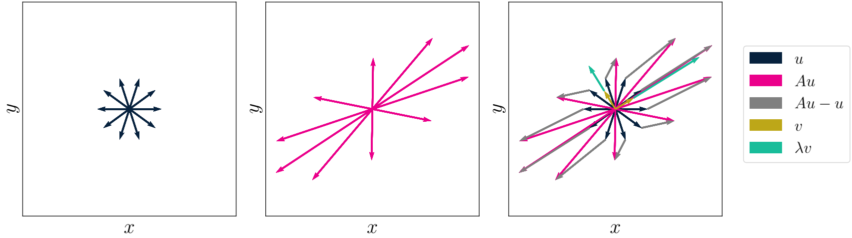
$$Av = \lambda v \tag{6}$$

- These directions v, the *eigenvectors*, and scalings λ, the *eigenvalues*, reveal fundamental properties about what kind of transformation A is.
- A can be expressed in many ways, many of which involve its eigenvalues and eigenvectors.

$$\operatorname{tr} A = \sum_i^n \lambda_i \qquad \det A = \prod_i^n \lambda_i \tag{7}$$

VECTOR INSTITUTE

How do Linear Transformations act on Vectors?



Invertability and Eigenvalues

- For a matrix A to be *invertible*:

  For each x, there must be a *unique* $y = Ax$.

- Imagine that there is some non-zero vector z such that $Az = 0$. Then there is no longer a unique input-output pair $y = Ax$, since for $w = x + z$:

$$y = Ax = Ax + 0 = Ax + Az = A(x + z) = Aw. \quad (8)$$

- What is the equation $Az = 0$, an eigenvalue equation with eigenvalue 0!
- Therefore we have another way to tell whether a matrix is invertible:

  A must have *no zero-eigenvalues*.

Matrix Decompositions

Factoring a Matrix

- Similar to how numbers can be factored, matrices can be also be *decomposed* into *products* of matrices

$$a = b\ c \ \longleftrightarrow\ A = B\ C\ . \tag{9}$$

- Many matrices are *diagonalizable*, meaning they can be expressed with their eigenvectors $V = [v_0, v_1, \cdots v_{n-1}]$ and eigenvalues $\Lambda = \mathrm{diag}(\lambda_0, \lambda_1, \cdots \lambda_{n-1})$:

$$AV = V\Lambda \ \longleftrightarrow\ A = V\Lambda V^{-1} \tag{10}$$

- All matrices can be expressed as a *polar decomposition*:

$$A = RL, \tag{11}$$

  where R is an orthogonal *rotation*, and L is a symmetric *scaling*.

Singular Value Decomposition

- Possibly the most important, and most general decomposition that *always* exists is the *singular value decomposition*

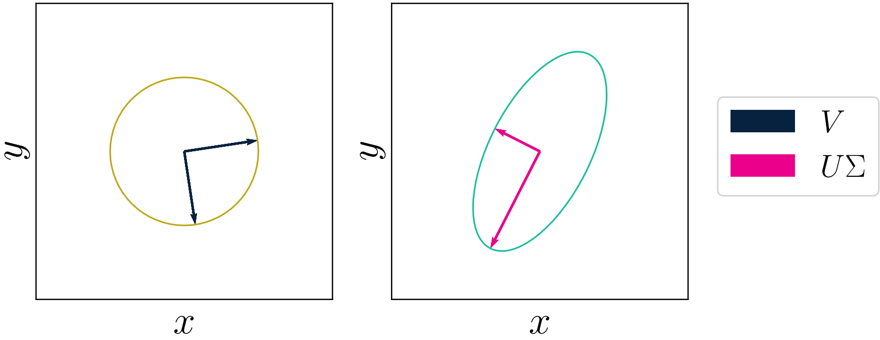
$$AV = U\Sigma \;\longleftrightarrow\; A = U\Sigma V^T \tag{12}$$

where U, V are *orthogonal*, and Σ is *real* and *diagonal* of *singular values*.

- Looking at the equation as $AV = U\Sigma$, we are really finding along which directions of V, the matrix A transforms to be along the directions of U, with scalings by Σ.

VECTOR INSTITUTE

Transformations of Circles



VECTOR INSTITUTE

Code Examples

PCA and Visualization

Mothers of Machine Learning Course

April 20, 2022

Tutorial 6

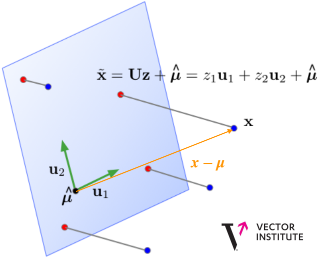
Outline

1. Recap of PCA
2. Code Examples

Recap of PCA

Projections and Reconstructions

- Given n, d-dimensional points $X = [x_0, x_1, \cdots x_n]$, where $x_i \in \mathbb{R}^d$, how may we project with U this data onto a suitable subspace of dimension $k < d$?
- $\mu \in \mathbb{R}^d$ is the mean of each *row* of X (over the n points).
- $U = [u_0, \cdots, u_{k-1}]$ are k orthonormal vectors $u_j \in \mathbb{R}^d$ for the subspace.
- Projective representations are $Z = U^T(X - \mu)$.
- Reconstructions are $\tilde{X} = \mu + UZ$.



VECTOR INSTITUTE

Maximizing Variance of Reconstructions

- We can derive that maximizing the reconstructed covariance

$$\tilde{\Sigma} = (\tilde{X} - \mu)(\tilde{X} - \mu)^T, \tag{1}$$

is equivalent to finding the k *eigenvectors* U of the empirical covariance

$$\Sigma = (X - \mu)(X - \mu)^T = \hat{U}\hat{\Lambda}\hat{U}^T, \tag{2}$$

where U is the first k columns of $\hat{U}$.

- Since Σ is *symmetric* and has this nice relationship with $X - \mu$, we can find U from its *SVD*

$$X - \mu = \hat{U}\hat{S}\hat{V}^T, \tag{3}$$

and $Z = U^T(X - \mu) = SV^T$!

VECTOR INSTITUTE

Code Examples

Your turn!

- Please get a copy of *Tutorial 6 PCA.ipynb*.
- The first sections *Modules*, *Utils*, *Setup Datasets* can be run first (and the details not worried about).
- There are 3 datasets to play around with PCA, that call the *setup(properties)* function to load the data and labels, and the *train(data,labels,k)* function to perform and plot the PCA with k components.
  - *Spiral* (spiral dataset from Lauren's tutorial)
  - *Animal* (256 x 256 images of 90 animal classes)
  - *MNIST* (28 x 28 images of handwritten digits 0 ... 9)