

Math Foundation for Deep Learning

2018-08-29 DeepLearning Math

Content

- Linear Algebra
- Calculus
- Probability
- Optimization Techniques
- Comments

Linear Algebra

1. Vector

An array of numbers, either continuous or discrete, is called a **vector**, and the space consisting of vectors is called a **vector space**. Vector space dimensions can be finite or infinite, but most machine-learning or datascience problems deal with fixed-length vectors;

2. Scalar

A one-dimensional vector is a scalar that has only magnitude and no direction.

3. Matrix

A matrix is a **two-dimensional** array of numbers arranged in rows and columns. The size of the matrix is determined by its row length and column length.

◦ Addition and Subtraction of Matrices

The addition and subtraction of matrices implies their element-wise addition and subtraction.

◦ Product of Two Matrices

For two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times q}$ to be multipliable, n should be equal to p . The resulting matrix is $C \in \mathbb{R}^{m \times q}$. The elements of C can be expressed as:

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

◦ Dot Product of Two Vectors

Any vector of dimension n can be represented as a matrix $v \in \mathbb{R}^{n \times 1}$. Let us denote two n dimensional vectors $v_1 \in \mathbb{R}^{n \times 1}$ and $v_2 \in \mathbb{R}^{n \times 1}$. The dot product of two vectors is the sum of the product of corresponding components—i.e., components along the same dimension—and can be expressed as:

$$v_1 \cdot v_2 = v_1^T v_2 = v_2^T v_1 = v_{11}v_{21} + v_{12}v_{22} + \dots + v_{1n}v_{2n} = \sum_{k=1}^n v_{1k}v_{2k}$$

◦ Matrix Working on a Vector

When a matrix is multiplied by a vector, the result is another vector. Let's say $A \in \mathbb{R}^{m \times n}$ is multiplied by the vector $x \in \mathbb{R}^{n \times 1}$. The result would produce a vector $b \in \mathbb{R}^{m \times 1}$:

$$A = \begin{bmatrix} c_{11}, c_{12}, \dots, c_{1n} \\ c_{21}, c_{22}, \dots, c_{2n} \\ \dots \\ c_{m1}, c_{m2}, \dots, c_{mn} \end{bmatrix} x = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

Rewrite A as column vectors:

$$b = Ax = \begin{bmatrix} c_1 & c_2 & c_3 & \dots & c_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = x_1 c_1 + x_2 c_2 + \dots + x_n c_n$$

As we can see, the product is nothing but the linear combination of the column vectors of matrix A , with the components of vector x being the linear coefficients.

4. Tensor

A tensor is a **multidimensional** array of numbers. In fact, vectors and matrices can be treated as 1-D and 2-D tensors. In deep learning, tensors are mostly used for storing and processing data.

5. Linear Independence of Vectors

A vector is said to be linearly dependent on other vectors if it can be expressed as the linear combination of other vectors.

6. Rank of a Matrix

The rank of a matrix is the number of **linearly independent** column vectors or row vectors. The number of independent columns vectors would always be equal to the number of independent row vectors for a matrix. A square matrix $A \in \mathbb{R}^{n \times n}$ is said to be full rank if the rank of A is n . A square matrix of rank n implies that all the n column vectors and even the n row vectors for that matter are linearly independent, and hence it would be possible to span the whole n -dimensional space by taking the linear combination of the n column vectors of the matrix A . A square matrix $A \in \mathbb{R}^{n \times n}$ is not full rank, then it is a singular matrix; i.e., all its column vectors or row vectors are not linearly independent. A singular matrix has an undefined matrix inverse and zero determinant.

7. Determinant of a Matrix

A determinant of a square matrix A is a number and is denoted by $\det(A)$. It can be interpreted as the absolute value of the matrix determining the volume enclosed by the row vectors acting as edges.

8. Inverse of a Matrix

An inverse of a square matrix $A \in \mathbb{R}^{n \times n}$ is denoted by A^{-1} and produces the identity matrix $I \in \mathbb{R}^{n \times n}$ when multiplied by A , i.e. $AA^{-1} = A^{-1}A = I$.

If a square matrix $A \in \mathbb{R}^{n \times n}$ is singular — i.e., if A doesn't have n independent column or row vectors — then the inverse of A doesn't exist. This is because for a singular matrix $\det(A) = 0$ and hence the inverse becomes undefined. A few rules for inverses of a matrix:

- $(AB)^{-1} = B^{-1}A^{-1}$
- $I^{-1} = I$

9. Pseudo Inverse of a Matrix

When matrix A is singular or is a rectangular matrix, then A^{-1} doesn't exist. In cases where we need to get the inverse of A , we can use its pseudo inverse, which is defined as $(A^T A)^{-1} A^T$.

10. Unit Vector

Unit vector in the direction of the specific vector is the vector divided by its magnitude or norm. For a Euclidian space, the unit vector in the direction of the vector x is

$$\frac{x}{\|x\|_2} = \frac{x}{(x^T x)^{1/2}}$$

Projection of a vector v_1 in the direction of v_2 is the dot product of v_1 with the unit vector in the direction of v_2 . $v_{12} = v_1^T u_2$, where u_2 is the unit vector in the direction of v_2 .

11. Eigen Vectors

Eigen values and Eigen vectors come up in several areas of machine learning. For example, the principal components in principal-component analysis are the Eigen vectors of the covariance matrix, while the Eigen values are the covariances along the principal components. Similarly, in Google's page-rank algorithm the vector of the page-rank score is nothing but an Eigen vector of the page transition probability matrix corresponding to the Eigen value of 1.

A matrix works on a vector as an operator. The operation of the matrix on the vector is to transform the vector into another vector whose dimensions might or might not be same as the original vector based on

the matrix dimension.

If in such a scenario the newly generated vector has the same direction or exactly the opposite direction as that of the original vector, then any vector in such a direction is called an Eigen vector. If in such a scenario the newly generated vector has the same direction or exactly the opposite direction as that of the original vector, then any vector in such a direction is called an Eigen vector. The magnitude by which the vector gets stretched is called the Eigen value.

$$Ax = \lambda x$$

where A is the matrix operator operating on the vector x by multiplication, which is also the Eigen vector, and λ is the Eigen value.

12. Norm of a Vector

The norm of a vector is a measure of its **magnitude**. There are several kinds of such norms. The most familiar is the **Euclidean norm**, defined next. It is also known as the l^2 norm.

$$||x||_2 = (|x_1|^2 + |x_2|^2 + |x_3|^2 + \dots + |x_n|^2)^{1/2} = (x \cdot x)^{1/2} = (x^T x)^{1/2}$$

l^1 norm is the sum of the absolute values of the vector components.

$$||x||_1 = (|x_1| + |x_2| + |x_3| + \dots + |x_n|)$$

the l^p norm of a vector can be defined as follows:

$$(|x_1|^p + |x_2|^p + |x_3|^p + \dots + |x_n|^p)^{1/p}$$

when $p \rightarrow \infty$ then the norm is called **Supremum norm** and is defined as follows:

$$\lim_{p \rightarrow \infty} ||x||_p = \lim_{p \rightarrow \infty} (|x_1|^p + |x_2|^p + |x_3|^p + \dots + |x_n|^p)^{1/p} = \max(x_1, x_2, \dots, x_n)$$

Generally, for machine learning we use both l^2 and l^1 norms for several purposes. For instance, the least square cost function that we use in linear regression is the l^2 norm of the error vector; i.e., the difference between the actual target-value vector and the predicted target-value vector. Similarly, very often we would have to use regularization for our model, with the result that the model doesn't fit the training data very well and fails to generalize to new data. To achieve regularization, we generally add the square of either the l^2 norm or the l^1 norm of the parameter vector for the model as a penalty in the cost function for the model. When the l^2 norm of the parameter vector is used for regularization, it is generally known as Ridge Regularization, whereas when the l^1 norm is used instead it is known as Lasso Regularization.

Calculus

In its very simplest form, calculus is a branch of mathematics that deals with differentials and integrals of functions. Having a good understanding of calculus is important for machine learning.

1. Differentiation

Differentiation of a function generally means the rate of change of a quantity represented by a function with respect to another quantity on which the function is dependent on.

The derivative of a single variable function is defined as $\frac{df(t)}{dt}$. When we deal with a function that is dependent on multiple variables, the derivative of the function with respect to each of the variables keeping the others fixed is called a partial derivative, and the vector of partial derivatives is called the gradient of the function. In a partial derivative, except the variable with respect to which the derivative is being taken, are held constant.

2. Gradient of a Function

For a function with two variables $z = f(x, y)$, the **vector of partial derivatives** $\left[\frac{\partial z}{\partial x} \quad \frac{\partial z}{\partial y} \right]^T$ is called the gradient of the function and is denoted by ∇z . The same can be generalized for a function with n variables. The gradient vector for the multivariate function $f(x)$ (where $x = [x_1, x_2, x_3, \dots, x_n]^T$) with respect to x can be expressed as:

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \dots, \frac{\partial f}{\partial x_n} \right]^T.$$

The gradient and the partial derivatives are important in machine-learning algorithms when we try to maximize or minimize cost functions with respect to the model parameters, since at the maxima and

minima the gradient vector of a function is zero. At the maxima and minima of a function, the gradient vector of the function should be a zero vector.

3. Successive Partial Derivatives

We can have successive partial derivatives of a function with respect to multiple variables. For example, for a function $z = f(x, y)$. The partial derivative of z with respect to x first and then with respect to y .

$$\frac{\partial}{\partial y} \left(\frac{\partial z}{\partial x} \right) = \frac{\partial^2 z}{\partial y \partial x}$$

Similarly, The partial derivative of z with respect to y first and then with respect to x .

$$\frac{\partial}{\partial x} \left(\frac{\partial z}{\partial y} \right) = \frac{\partial^2 z}{\partial x \partial y}$$

If the second derivatives are continuous, the order of partial derivatives doesn't matter and we have:

$$\frac{\partial^2 z}{\partial y \partial x} = \frac{\partial^2 z}{\partial x \partial y}$$

4. Hessian Matrix of a Function

The Hessian of a multivariate function is a **matrix of second-order partial derivatives**. For a function $f(x, y, z)$, the Hessian is defined as follows:

$$\begin{bmatrix} \frac{\partial^2 f}{\partial^2 x} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial^2 y} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial^2 z} \end{bmatrix}$$

The Hessian is useful in the optimization problems that we come across so frequently in the machine-learning domain. For instance, in minimizing a cost function to arrive at a set of model parameters, the Hessian is used to get better estimates for the next set of parameter values, especially if the cost function is non-linear in nature. Non-linear optimization techniques, such as Newton's method, Broyden-Fletcher-Goldfarb-Shanno (BFGS), and its variants, use the Hessian for minimizing cost functions.

5. Positive Semi-Definite and Positive Definite

A **square matrix** $A \in \mathbb{R}^{n \times n}$ is **positive semi-definite** if for any non-zero vector $x \in \mathbb{R}^{n \times 1}$ the expression $x^T A x \geq 0$. The matrix A is **positive definite** if the expression $x^T A x > 0$. All the **Eigen values** for a positive semi-definite matrix should be non-negative, whereas for a positive definite matrix the Eigen values should be positive.

6. Taylor Series

Any function can be expressed as an infinite sum by considering the value of the function and its derivatives at a specific point. Such an expansion of the function is called **Taylor Series expansion**. The Taylor Series expansion of a univariate function around a point x can be expressed as follows:

$$f(x + h) = f(x) + hf'(x) + \frac{1}{2!}h^2f''(x) + \frac{1}{3!}h^3f'''(x) + \dots + \frac{1}{n!}h^n f^n(x) + \dots$$

where $f^n x$ is the n th derivative of the function $f(x)$ and $n!$ denotes the factorial of the number n . The term h has the same dimension as that of x , and both h, x are scalars.

Taylor Series expansion for multivariate functions around a point x can be expressed as

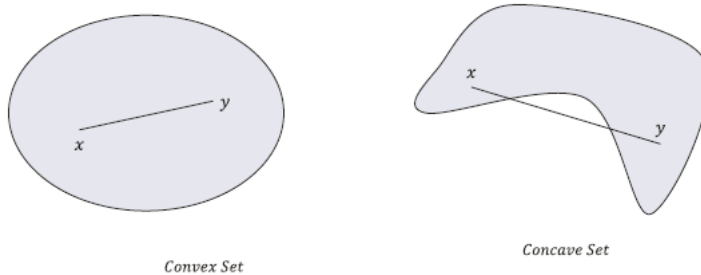
$$f(x + \Delta x) = f(x) + \Delta x^T \nabla f(x) + \frac{1}{2} \Delta x^T \nabla^2 f(x) \Delta x + \dots$$

where $\nabla f(x)$ is the gradient vector and $\nabla^2 f(x)$ is the Hessian matrix for the function $f(x)$.

Generally, for practical purposes, we don't go beyond second-order Taylor Series expansion in machine-learning applications since in numerical methods they are hard to compute. Even for second-order expansion computing the Hessian is cost intensive and hence several second-order optimization methods rely on computing the approximate Hessians from gradients instead of evaluating them directly. Please note that the third-order derivatives object $\nabla^3 f(x)$ would be a three-dimensional tensor.

7. Convex Set

A set of points is called convex if, given any two points x and y belonging to the set, all points joining the straight line from x to y also belong to the set. In the figure below, a convex set and a non-convex set are illustrated.



8. Convex Function

A function $f(x)$ is said to be a convex function if the straight line joining any two points in the function lies **above or on** the graph of the function. A convex function has the local minima as its global minima. Bear in mind that there can be more than one global minima, but the value of the function would be same at each of the global minima for a convex function.

9. Non-convex Function

A non-convex function can have many local minima, all of which are not global minima. In any machine-learning model building process where we try to learn the model parameters by minimizing a cost function, we prefer the cost function to be convex, since with a proper optimization technique we would attain the global minima for sure. For a non-convex cost function, there is a high chance that the optimization technique will get stuck at a local minima or a saddle point, and hence it might not attain its global minima.

Probability

1. Unions, Intersection, and Conditional Probability

$P(A \cup B)$ is the probability of the event A or event B or both. $P(A \cap B)$ is the probability event A and event B . $P(A/B)$ is the probability event A given that B has already occurred.

$$P(A \cap B) = P(A/B)P(B) = P(B/A)P(A)$$

$$P(A - B) = P(A) - P(AB)$$

2. Chain Rule of Probability for Intersection of Event

If $A_1, A_2, A_3, \dots, A_n$ is the set of n events, then the joint probability of these events can be expressed as follows:

$$P(A_1 A_2 \dots A_n) = P(A_1)P(A_2/A_1)P(A_3/A_2 A_1) \dots P(A_n/A_1 A_2 \dots A_{n-1})$$

3. Bayes Rule

From 1 we know that $P(AB) = P(A/B)P(B) = P(B/A)P(A)$, so $P(A/B) = P(A)P(B/A)/P(B)$. This is called the **Bayes Rule**, and it would come handy in many areas of machine learning, such as in computing posterior distribution from likelihood, using Markov chain models, maximizing a posterior algorithm, and so forth.

4. Probability Mass Function

The probability mass function (pmf) of a random variable is a function that gives the probability of each **discrete** value that the random variable can take up. The sum of the probabilities must add up to 1.

5. Probability Density Function

The probability density function (pdf) gives the probability density of a **continuous** random variable at each value in its domain. Since it's a continuous variable, the integral of the probability density function

over its domain must be equal to 1.

Let X be a random variable with domain D . $P(x)$ denotes it's a probability density function, so that

$$\int_D P(x)dx = 1$$

6. Expectation of a Random Variable

Expectation of a random variable is nothing but the mean of the random variable. For discrete variables the expectation is $E[X] = x_1p_1 + x_2p_2 + \dots + x_np_n = \sum_{i=1}^n x_ip_i$.

If X is a continuous random variable with a probability density function of $P(x)$, the expectation of X is given by $E[X] = \int_D xp(x)dx$

7. Variance of a Random Variable

Variance of a random variable measures the variability in the random variable. It is the mean (expectation) of the squared deviations of the random variable from its mean (or expectation). Variance of random variable X is defined as:

$$Var[X] = E[(X - \mu)^2] \text{ where } \mu = E[X], \text{ which is the mean of } X.$$

If X is a discrete random variable that takes n discrete values with a pmf given by $P(X = x_i) = p_i$, the variance of X can be expressed as:

$$Var[X] = E[(X - \mu)^2] = \sum_{i=1}^n (x_i - \mu)^2 p_i$$

If X is a continuous random variable having a probability density function of $p(x)$, then $Var[X]$ can be expressed as:

$$Var[X] = \int_D (x - \mu)^2 p(x)dx.$$

8. Covariance and Correlation Coefficient

The covariance between two random variables X and Y is a measure of their joint variability. The covariance is positive if higher values of X correspond to higher values of Y and lower values of X correspond to lower values of Y . Otherwise, it is negative. The covariance between X and Y is defined as follows:

$$cov(X, Y) = E[X - \mu_x][Y - \mu_y] = E[XY] - \mu_x\mu_y$$

$$\text{where } \mu_x = E[X], \mu_y = E[Y]$$

If two variables are independent, their covariance is zero since $E[XY] = E[X]E[Y] = \mu_x\mu_y$

The covariance in general does not provide much information about the degree of association between two variables, because the two variables maybe on very different scales. Getting a measure of the linear dependence between two variables' correlation coefficients, which is a normalized version of covariance, is much more useful.

The correlation coefficient between two variables X and Y is expressed as:

$$\rho = \frac{cov(X, Y)}{\sigma_x\sigma_y}$$

where σ_x and σ_y are the standard deviation of X and Y .

9. Uniform Distribution

The probability density function for a uniform distribution is constant. For a continuous random variable that takes up values between a and b ($b > a$), the probability density function is expressed as:

$$P(X = x) = f(x) = \begin{cases} \frac{1}{b-a} & \text{for } x \in [a, b] \\ 0 & \text{elsewhere} \end{cases}$$

10. Normal Distribution

This is probably the most important scenario for probability distribution in the real-world. The probability density function of a normal distribution can be expressed as:

$$P(X = x) = f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad -\infty < x < +\infty$$

where μ is the mean and σ^2 is the variance of the random variable X .

11. Bernoulli Distribution

An experiment in which the two outcomes are mutually exclusive and exhaustive (the sum of probability of the two outcomes is 1) is called a Bernoulli trial. The probability mass function of Bernoulli Distribution can be written as:

$$P(X = x) = f(x) = p^x(1-p)^{(1-x)} \quad x \in 0, 1$$

The expectation and variance of the probability mass function are as follows:

$$E[X] = p; \quad \text{Var}[X] = p(1 - p)$$

The Bernoulli distribution can be extended to multiclass events that are mutually exclusive and exhaustive. Any two-class classification problem can be modeled as a Bernoulli trial. For instance, the logistic regression likelihood function is based on a Bernoulli distribution for each training data point, with the probability p being given by the sigmoid function.

12. Binomial Distribution

In a sequence of Bernoulli trials, we are often interested in the probability of the total number of successes and failures instead of the actual sequence in which they occur. The probability mass function of binomial distribution is defined as:

$$P(X = x) = f(x) = \binom{n}{x} p^x (1 - p)^{n-x} \quad x \in \{0, 1, 2, \dots, n\}$$

where p is the probability of success. The expectation and variance of the probability mass function are as follows:

$$E[X] = np; \quad \text{Var}[X] = np(1 - p)$$

13. Poisson Distribution

Whenever the rate of some quantity is of concern, such as the number of alpha particles emitted by a radioactive substance in the previous four-hour duration, Poisson distribution is generally the best way to represent such phenomenon. The probability mass function for Poisson distribution is as follows:

$$P(X = x) = f(x) = \frac{e^{-\lambda} \lambda^x}{x!}; \quad E[X] = \lambda; \quad \text{Var}[X] = \lambda$$

14. Likelihood Function

Likelihood is the probability of the observed data given the parameters that generate the underlying data. Let's suppose we observe n observations x_1, x_2, \dots, x_n and assume that the observations are independent and identically normally distributed with mean μ and variance σ^2 . The likelihood function in this case would be as follows:

$$P(\text{data}/\text{model parameters}) = \prod_{i=1}^n P(x_i/\mu, \sigma^2)$$

Since x_i is normally distributed so the above equation can be expanded as:

$$P(\text{data}/\text{model parameters}) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi}} e^{-\frac{(x_i - \mu)^2}{2\sigma^2}}$$

Optimization Techniques

1. Gradient Descent

Gradient descent is an iterative method that starts with a random model parameter and uses the gradient of the cost function with respect to the model parameter to determine the direction in which the model parameter should be updated.

Suppose we have a cost function $C(\theta)$, where θ represents the model parameters. We know the gradient of the cost function with respect to θ gives us the direction of maximum increase of $C(\theta)$ in a linear sense at the value of θ at which the gradient is evaluated. So, to get the direction of maximum decrease of $C(\theta)$ in a linear sense, one should use the negative of the gradient.

The update rule of the model parameter θ at iteration $(t + 1)$ is given by:

$$\theta^{t+1} = \theta^t - \eta \nabla C(\theta^t)$$

where η represents the learning rate and θ^{t+1} and θ^t represent the parameter vector at iteration $t + 1$ and t respectively.

Because of rounding errors and other limitations of computers, converging to true minima might not be achievable. Thus, we would have to come up with some other logic to stop the iterative process when we believe we have reached minima—or at least close enough to the minima—to stop the iterative process of model training. One of the ways generally used is to check the magnitude of the gradient vector and if it is less than some predefined threshold, stop the iterative process. The other crude way that can be used is to stop the iterative process of the parameter update after a fixed number of iterations, like 1000.

2. Gradient Descent for a Multivariate Cost Function

Let's consider a cost function with multiple variables $C(\theta)$ where $\theta \in \mathbb{R}^{n \times 1}$. We can use a Taylor Series expansion of the cost function. The Taylor Series expansion around a point θ can be expressed as:

$$C(\theta + \Delta\theta) = C(\theta) + \Delta\theta^T \nabla C(\theta) + \frac{1}{2} \Delta\theta^T H(\theta) \Delta\theta + \dots$$

where $\Delta\theta$ is the change in θ vector

$\nabla C(\theta)$ is the gradient vector of $C(\theta)$

$H(\theta)$ is the Hessian matrix of $C(\theta)$

If we assume linearity of the function in the neighborhood of θ , i.e. second and higher order derivatives are zero, then from Taylor Series expansion we get the following:

$$C(\theta + \Delta\theta) = C(\theta) + \Delta\theta^T \nabla C(\theta)$$

To minimize the cost, we need $\Delta\theta$ to be proportional to the negative of the gradient vector $\nabla C(\theta)$. The rule for updating θ can be expressed as:

$$\theta^{t+1} = \theta^t - \eta \nabla C(\theta^t)$$

This is the famous equation for gradient descent.

3. Steepest Descent

Steepest descent is a form of gradient descent where the learning rate is not constant but rather is computed at every iteration to ensure that the parameter update through gradient descent takes the cost function to minima with respect to the learning rate. In other words, the learning rate at every iteration in steepest descent is optimized to ensure that the movement in the direction of the gradient is utilized to the maximum extent.

From the above analysis we know that:

$$\theta^{t+1} = \theta^t - \eta \nabla C(\theta^t)$$

So, the cost function at iteration $t + 1$ can be expressed as:

$$C(\theta^{t+1}) = C(\theta^t - \eta \nabla C(\theta^t))$$

To minimize the cost function at iteration $t + 1$ with respect to the learning rate:

$$\frac{\partial C(\theta^{t+1})}{\partial \eta} = 0$$

which is equivalent to

$$\nabla C(\theta^{t+1}) \frac{\partial [C(\theta^t) - \eta \nabla C(\theta^t)]}{\partial \eta} = 0$$

which can be simplified as:

$$-\nabla C(\theta^{t+1})^T \nabla C(\theta^t) = 0$$

So, for steepest descent, the dot product of the gradients at $t + 1$ and t is 0, which implies that the gradient vector at every iteration should be perpendicular to the gradient vector at its previous iteration.

4. Stochastic Gradient Descent

Both steepest descent and gradient descent are full-batch models; i.e., the gradients are computed based on the whole training dataset. So, if the dataset is huge, the gradient computation becomes expensive and the memory requirement increases. Also, if the dataset has huge redundancy, then computing the gradient on the full dataset is not useful since similar gradients can be computed by using much smaller batches called mini batches. The most popular method to overcome the preceding problems is to use an optimization technique called stochastic gradient descent.

Since the gradients at each iteration are not based on the entire training dataset but rather on single training samples, they are generally very noisy and may change direction rapidly. This may lead to oscillations near the minima of the cost function, and hence the learning rate should be less while converging to the minima so that the update to the parameter vector is as small as possible. The gradients are cheaper and faster to compute, and so the gradient descent tends to converge faster.

One thing that is important in stochastic gradient descent is that the training samples should be as random as possible. This will ensure that a stochastic gradient descent over a period of a few training samples provides a similar update to the model parameter as that resulting from an actual gradient descent, since the random samples are more likely to represent the total training dataset. If the samples at each iteration of stochastic gradient descent are biased, they don't represent the actual dataset, and hence the update to the model parameter might be in a direction that would result in it taking a long time for the stochastic gradient descent to converge.

5. Newton's Method

In gradient descent we assumed the linearity of the cost function between successive iterations, which is

a very simplified assumption and would not yield good directions for gradient descent if the cost function is highly non-linear or has curvature.

Let's take our usual cost function $C(\theta)$, We can approximate the cost function $C(\theta)$ in the neighborhood of θ by its second-order Taylor series expansion:

$$C(\theta + \Delta\theta) = C(\theta) + \Delta\theta^T \nabla C(\theta) + \frac{1}{2} \Delta\theta^T H(\theta) \Delta\theta$$

Take the gradient with respect to $\theta^{(t+1)}$ and set it to zero, we have:

$$\nabla C(\theta^{(t+1)}) = \nabla C(\theta^{(t)}) + H(\theta^{(t)}) \Delta\theta = 0$$

So, the parameter update for Newton's method is as follows:

$$\theta^{(t+1)} = \theta^{(t)} - H(\theta^{(t)})^{-1} \nabla C(\theta^{(t)})$$

We don't have a learning rate for Newton's method, but one may choose to use a learning rate, much like with gradient descent. Since the directions for non-linear cost functions are better with Newton's method, the iterations to converge to the minima would be fewer as compared to gradient descent. One thing to note is that if the cost function that we are trying to optimize is a quadratic cost function, such as the one in linear regression, then Newton's method would technically converge to the minima in one step.

However, computing the Hessian matrix and its inverse is computationally expensive or intractable at times, especially when the number of input features is large. Also, at times there might be functions for which the Hessian is not even properly defined. So, for large machine-learning and deep-learning applications, gradient descent—especially Stochastic gradient descent—techniques with mini batches are used since they are relatively less computationally intensive and scale up well when the data size is large.

6. Constrained Optimization Problem

In a constrained optimization problem, along with the cost function that we need to optimize, we have a set of constraints that we need to adhere to. The constraints might be equations or inequalities.

Whenever we want to minimize a function that is subject to an equality constraint, we use the Lagrange formulation. For example, we need to minimize $f(\theta)$ subject to $g(\theta) = 0$ where $\theta \in \mathbb{R}^{n \times 1}$. For such a constrained optimization problem, we need to minimize a function $L(\theta, \lambda) = f(\theta) + \lambda g(\theta)$ Taking the gradient of L , which is called the Lagrangian, with respect to the combined vector θ, λ , and setting it to 0 would give us the required θ that minimizes $f(\theta)$ and adheres to the constraint. λ is called the Lagrange multiplier. When there are several constraints, we need to add all such constraints, using a separate Lagrange multiplier for each constraint. Let's say we want to minimize $f(\theta)$ subject to m constraints

$g_i(\theta) = 0$, the Lagrangian is defined as:

$$L(\theta, \lambda) = f(\theta) + \sum_{i=1}^m \lambda_i g_i(\theta)$$

To minimize the function, the gradient of $L(\theta, \lambda)$ with respect to both θ and λ , vectors should be a zero vector; i.e.,

$$\nabla_{\theta}(\theta, \lambda) = 0; \quad \nabla_{\lambda}(\theta, \lambda) = 0$$

The preceding method can't be directly used for constraints with inequality. In such cases, a more generalized approach called the **Karush Kahn Tucker** method can be used.

Let $C(\theta)$ be the cost function that we wish to minimize, where $\theta \in \mathbb{R}^{n \times 1}$. Also, let there be k number of constraint on θ such that

$$f_1(\theta) = a_1; \quad f_2(\theta) \leq a_2; \quad f_3(\theta) \geq a_3; \quad \dots \quad f_k(\theta) = a_k$$

This becomes a constrained optimization problem since there are constraints that θ should adhere to.

Every inequality can be transformed into a standard form where a certain function is less than or less than equal to zero. For example:

$$f_3(\theta) \geq a_3 \Rightarrow a_3 - f_3(\theta) \leq 0$$

Let each such constraint strictly less than, or less than equal to, zero be represented by $g_i(\theta)$. Also, let there be some strict equality equations $e_j(\theta)$. Such minimization problems are solved through the Karush Kuhn Tucker version of Lagrangian formulation.

Instead of minimizing $C(\theta)$, we need to minimize a cost function $L(\theta, \alpha, \beta)$ as follows:

$$L(\theta, \alpha, \beta) = C(\theta) + \sum_{i=1}^{k_1} \alpha_i g_i(\theta) + \sum_{j=1}^{k_2} \beta_j e_j(\theta)$$

where scalars α_i and β_j are called the Lagrangian multipliers, and there would be k of them

corresponding to k constraints. So, we have converted a constrained minimization problem into an

unconstrained minimization problem.

To solve the problem, the Karush Kuhn Tucker conditions should be met at the minima point as follows:

- The gradient of $L(\theta, \alpha, \beta)$ with respect to θ should be the zero vector; i.e.,

$$\nabla_{\theta}(\theta, \alpha, \beta) = 0$$

$$\Rightarrow \nabla_{\theta}C(\theta) + \sum_{i=1}^{k_1} \alpha_i \nabla_{\theta}g_i(\theta) + \sum_{j=1}^{k_2} \beta_j \nabla_{\theta}e_j(\theta) = 0$$

- The gradient of $L(\theta, \alpha, \beta)$ with respect to θ should be the zero vector; i.e.,

$$\nabla_{\beta}(\theta, \alpha, \beta) = 0$$

$$\Rightarrow \nabla_{\beta}C(\theta) + \sum_{i=1}^{k_1} \alpha_i \nabla_{\beta}g_i(\theta) + \sum_{j=1}^{k_2} \beta_j \nabla_{\beta}e_j(\theta) = 0$$

- The inequality conditions should become equality conditions at the minima point. Also, the inequality Lagrange multipliers should be non-negative:

$$\alpha_i g_i(\theta) = 0 \quad \text{and} \quad \alpha_i \geq 0$$

Solving for the preceding conditions would provide the minima to the constrained optimization problem.

7. Regularization

The process of building a machine-learning model involves deriving parameters that fit the training data. If the model is simple, then the model lacks sensitivity to the variation in data and suffers from high bias. However, if the model is too complex, it tries to model for as much variation as possible and in the process models for random noise in the training data. High variance for a model is not a good thing, especially if the noise in the data is considerable. In such cases, the model in the pursuit of performing too well on the training data performs poorly on the test dataset since the model loses its capability to generalize well with the new data. This problem of models' suffering from high variance is called overfitting.

To put things into perspective, let's look at the linear regression cost function that we looked at earlier:

$$C(\theta) = ||X\theta - Y||_2^2$$

As discussed earlier, models with high variance have model parameters with a large magnitude. We can put an extra component into the cost function $C(\theta)$ that penalizes the overall cost function in case the magnitude of the model parameter vector is high.

So, we can have a new cost function:

$$C(\theta) = ||X\theta - Y||_2^2 + \lambda ||\theta||_2^2$$

Taking the gradient ∇L with respect to θ and setting it to 0 gives us $\theta = (X^T X + \lambda I)^{-1} X^T Y$.

Now, as we can see because of the regularization term in the cost function, the model parameter's magnitude can't be too large since it would penalize the overall cost function. λ determines the weight of the regularization term. A higher value for λ would result in smaller values of $||\theta||_2^2$, thus making the model simpler and prone to high bias or underfitting. In general, even smaller values of λ go a long way in reducing the model complexity and the model variance. λ is generally optimized using cross validation.

When the square of the l^2 norm is used as the regularization term, the optimization method is called l^2 regularization. At times, the l^1 norm of model parameter vectors is used as the regularization term, and the optimization method is termed l^1 regularization. l^2 regularization applied to regression problems is called **ridge regression**, whereas l^1 regularization applied to such regression problems is termed **lasso regression**.

Disclaimer: This post includes my personal reflections and notes on reading Pro Deep Learning with TensorFlow by Santanu Pattanayak. Some texts and images are from the book for better educational purposes.

[Previous post](#) MapReduce Algorithm Design

Comments

0 Comments <http://leiluoray.com/>

 Arefe ▾




 Recommend  Share

Sort by Best ▾



Start the discussion...

Be the first to comment.

 Subscribe  Add Disqus to your siteAdd DisqusAdd  Disqus' Privacy PolicyPrivacy PolicyPrivacy PolicyPrivacy Policy

Contact me at:     

Total visited by 2610 times, The page viewed by 1428 times, This post viewed by 331 times