Contents

1 Optimizers

In ml and other fields requiring to minimize a "cost" function, we enact the methods of iterative optimizers. This section presents the classic gradient descent with momentum addition, the RMSProp optimizer, and the Gauss-Newton technique.

For notations, let

- $f(w)$ be the cost function against weights w
- $J_f(w)$ be the jacobian of the cost function
- $\nabla f(w)$ be the gradient of the cost function. If $f(w)$ outputs real values, then $\nabla f = J_f$
- $[f(w)]_i = r_i(w)$ for the case of residues in the Gauss-Newton technique

we denote vectors implicitly and assume vectorization of scalars.

Gradient Descent We consider the iteration on z_i and m_i,

$$\begin{aligned} m_{i+1} &= \beta m_i + \mu \nabla f(z_i) \\ z_{i+1} &= z_i - m_{i+1} \end{aligned}$$

where

- z_i denotes the successive weights
- m_i the successive momentum
- β the momentum retaining percentage, often $\beta = 0.9$
- μ the learning rate, often $\mu = 0.15$

The intuition is that m represents the sum of the current direction of steepest ascent ($\nabla f(w)$) and the historical ascents. By continuously heading downhill with momentum, we hope to reach the global minimum of the cost function f.

RMSProp We consider the iteration on z_i and β_i,

$$\begin{aligned} m_{i+1} &= \beta m_i + (1-\beta)[\nabla f(z_i)]^2 \\ z_{i+1} &= z_i - \frac{\mu}{\sqrt{m_{i+1}}} \nabla f(z_i) \end{aligned}$$

where

- z_i denotes the successive weights
- m_i the squared average gradients

- β the decay rate of the squared average gradients, often $\beta = 0.9$
- μ the learning rate

Note that $[\nabla f(z)]^2$ represents the element-wise power of 2. $\sqrt{m_{i+1}}$ and the following multiplication are both element-wise square roots and multiplications.

The intuition is that we will simply use the sign of the gradient $\nabla f(z)$ and discard its magnitude by dividing against itself. The squared average gradients is to ensure the past gradients are considered and the divider is positive to maintain the gradient sign.

The RMSprop optimizer is more commonly used in mini-batched trainings, where the global cost gradient is estimated by the mean of a few samples'. A full-batch technique, where the true cost gradient is computed by averaging the individual sample gradients, is often too slow for large datasets. The common Stochastic Gradient descent is the mini-batched technique applied with a batch size of 1.

Gauss-Newton method Consider the iteration on z_i,

$$\begin{aligned} \Delta &= (J_f^T J_f)^{-1} J_f^T f(z_i) \\ z_{i+1} &= z_i - \alpha\Delta \end{aligned}$$

where

- $z_i = \begin{bmatrix} w_1(i) \\ w_2(i) \\ \vdots \end{bmatrix}$ are the successive weights
- $f(z_i) = \begin{bmatrix} r_1(z_i) \\ r_2(z_i) \\ \vdots \end{bmatrix}$ is the multivariate residue function
- $J_f = \begin{bmatrix} \frac{\partial r_1}{\partial w_1} & \frac{\partial r_1}{\partial w_2} & \cdots \\ \frac{\partial r_2}{\partial w_1} & \frac{\partial r_2}{\partial w_2} & \cdots \\ \vdots & & \ddots \end{bmatrix}$ is the residue jacobian
- α is the learning rate, this is often set to $\alpha = 1$ unless the initial conditions are far away from the solution and the second derivative estimate fails.

The algorithm aims to find a weight z that minimizes $\sum_i r_i(w)^2$, representing minimizing the sum of squared residues.

The intuition is that the delta term, Δ, represents the direction towards the domain that minimizes a second-order approximation of the residues. There is a Hessian matrix some-

where here and I still don't understand the specifics. The Δ expression seems like a least squares approximation to fit the residues, if that helps.

If the dimension of the vector residue function equals the weights, the Δ calculation is simplified to

$$\Delta = (J_f)^{-1} f(z_i)$$

This algorithm is extremely effective in converging of nice residue functions. However, it may become uncontrollable with a far away initial condition z_0.

2 Trilateration

For trilateration using 3 points in 3-dimensions, we use the following residues and jacobians

$$
\begin{aligned}
r_1(z) &= ||z - p_1||^2 - r_1^2 \\
r_2(z) &= ||z - p_2||^2 - r_2^2 \\
r_3(z) &= ||z - p_3||^2 - r_3^2 \\
J(z) &= 2 \begin{bmatrix} (z - p_1)^T \\ (z - p_2)^T \\ (z - p_3)^T \end{bmatrix}
\end{aligned}
$$

where

- p_i is the ith point as a 3-vector
- r_i is the distance from p_i that z must be
- $(z - p_i)^T = [(z_x - p_{ix}), (z_y - p_{iy}), (z_z - p_{iz})]$ is a row vector

By substituting the above into the Gauss-Newton iteration, we get a fast converging algorithm for trilateration.

The intuition is that the SSE of the residues gets minimized, implying a z where

$$||z - p_i||^2 - r_i^2 = 0$$

or $||z - p_i|| = r_i$. The squaring on the norm term is to cancel out the square root in the norm expansion, leading to a better Jacobian function that doesn't contain singularities — division by the norm.

3 Explorations

3.1 Convolutions

The PDF of the sum of two independent random variables is the convolution between their PDFs. The central limit theorem, that the PDF that is the mean of the infinite sum of any distributed random variable is a normal distribution, bases on the fact that only a normal distribution with its gaussian PDF is invariant under a convolution.

3.2 p-adic Numbers

We define the following notations against describing p-adic numbers:

- A p-adic number of prime $p \in \mathbb{Z}$ is denoted by and equivalent to

  $$\ldots 00123_{(p)} = 3p^0 + 2p^1 + 1p^2 + 0p^3 + \ldots$$

  where the digits and subsequent operations on the digits to be performed in $\mathbb{Z}^p$, the modulo finite field of size p.

- The absolution value, or norm, of a p-adic number n is represented by $|n|$, where $|n| = \frac{1}{p^k}$ with k equal to the index (0-based) of the left-most non-zero digit in n. This metric satisfies the standard axioms of a metric/distance function.

Intuitions The p-adic numbers is a number system that is unique in its inverted metric: that "larger" numbers approach an absolute value of 0 and the "smaller" numbers approach 1.

Algebraically, we can treat p-adic numbers as their infinite series expansion. A connection is that the p-adic number is equivalent to the classic generating function $G(z; a_i)$ on the infinite series a_i

$$G(z; a_i) = \sum_{i=0}^{\infty} a_i z^i = \ldots a_2 a_1 a_{0(z)}$$

albeit with a prime integer $p = z$.

This algebraic property means that for a number with repeating digits, we can exploit the geometric series summation expression without convergence worries. In general, given digit $d \in \mathbb{Z}^+$ in p-adics, we have

$$\ldots dddd_{(p)} = dp^0 + dp^1 + \cdots = \frac{d}{1-p}$$

This equivalence allows some "rational" numbers in p-adic representations to be converted into the reals.

The reason for the base to be prime is to ensure the algebraic properties. Say the base is composite instead, for any number a and b, $ab = 0$ does not imply that either a or b is zero — the addition identity 0 is not unique. This is because several options in b may be possible in the first digit to set the resulting product's first digit 0, as the field of digits does not form a field itself. A prime cardinal field will instead, require all digits of b to be zero. This property lets all p-adic sets be fields that follow the usual algebraic rules on the reals.

Operations The standard operation algorithms on the reals work the same on the p-adics. Here are their definitions:

- For addition, perform digit-wise addition from the least significant digit, allow for carries.
- For multiplication, perform standard digit-wise multiplication from the least significant digit, allow for carries.
- For subtraction, first consider the negation. To negate a number, apply ps complement on its digit (the digit with value d is mapped to $p - d - 1$) and add the value 1. For example

  $$-2 = -\ldots 002_{(3)} = \ldots 221_{(3)}$$

  Then we can simply negate the second argument in a subtraction operation, and compute its addition like normal
- For division, say $\frac{a}{b}$, we aim to find the digit expansions of x such that $xb = a$. This is a system of linear equations that can be solved iteratively
- For roots and other irrational operations, we require numerical estimates from algorithms such as the Newton's method

That's all. No general applications yet.

3.3 Perlin Noise and Perlin Curl Noise

Perlin noise presents a smooth noise function in an arbitrary dimension. The algorithm is as so.

First suppose a random vector field of unit gradients, each positioned (virtually) on the vertices of a mesh grid. To compute the value of a sample s within this space:

1. consider the nearest 2^d vertices v_i each with gradient g_i
2. for every vertex v_i, consider the displacement vector from the sample position to the vertex $d_i = v_i - s$. Compute the dot product between the displacement vector and the gradient vector, with $p_i = d_i \cdot g_i$
3. interpolate all the gradient dot products p_i using the percentage relative positioning of the sample s within the grid cell. It is common to use the smoothstep function

$$st(x) = 3x^2 - 2x^3$$

The interpolated gradient dot products is the computed value in a Perlin noise field.

An example for the 2D Perlin noise with grid cell size 1, we have the sample function $P(x, y)$ be computed by

$$\begin{aligned} d_{ij} &= (v_{ij} - (x, y)) \cdot g_{ij} \\ \text{top} &= d_{01} + st(x - x_i)(d_{11} - d_{01}) \\ \text{bottom} &= d_{00} + st(x - x_i)(d_{10} - d_{00}) \\ P(x, y) &= \text{bottom} + st(y - y_i)(\text{top} - \text{bottom}) \end{aligned}$$

where

- v_{ij} is the relative vertex against the floored anchor (x_i, y_i) of (x, y), with v_{00} representing the anchor vertex

The order of the 2D smoothstep functions between the 4 vertices here should not matter too much.

Perlin curl noise is an extended Perlin noise that has no divergence. It is useful as a starting velocity field for fluid simulations.

4 Curves and Splines

This section concerns curves and continuity with specifics into bezier curves and splines.

A curve can be described as a parametric function, say $c(t)$, where t ranges from 0 to 1. To define continuity, we consider the derivatives of said parametric function against its parameter t.

The continuity classes C^n where $n \in \mathbb{N}$ is defined as so: for a function $c(t)$ to be C^n continuous

- the nth derivative of c is continuous — ie, not having jumps or singularities.
- the function is also C^k continuous for all $k < n$

A function is C^0 continuous if it has no jumps or singularities; it is C^∞ continuous if any of its derivatives are continuous.

The geometric continuity class is a less strict definition of continuity, it is one where only the directions of its derivatives are limited.

G^0 continuity is the same as C^0 continuity. G^1 continuity implies the unit derivative of the function is continuous. G^2 requires the curvature function to be continuous. Higher G^n continuity requires the derivatives of the curvature function to be continuous.

The curves described here are either C^0, C^1, or C^2 continuous. Most are also G^1 continuous.

4.1 Bezier curves

In general, an order n bezier curve is an n dimensional polynomial with vector coefficients. They form a generalized basis of parametric smooth functions.

The degree 1 bezier curve is a simple linear interpolation, or lerp. Between point P_0 and P_1, it is

$$\text{lerp}(P_0, P_1, t) = P_0 + t(P_1 - P_0)$$

The second degree bezier curve can be treated as lerps between the lerp of point 0 and 1, and point 1 and 2. To compute the second degree bezier $c(t)$ between P_0, P_1, and P_2,

$$\begin{aligned} a &= \text{lerp}(P_0, P_1, t) \\ b &= \text{lerp}(P_1, P_2, t) \\ c(t) &= \text{lerp}(a, b, t) \end{aligned}$$

it can also be written in explicit form with its characteristic polynomial.

The third degree bezier curve can similarly be treated as consecutive lerps, or alternative, computed with its characteristic matrix

$$c(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

This makes its derivative easily calculated (by differentiating the t dependent row vector).

For a cubic bezier curve, the points P_1 and P_2 are called control points; the points P_0 and P_3 are called the endpoints.

4.2 Bezier splines

Bezier splines are connected bezier curves. For a cubic bezier spline, the first point of the next bezier curve is connected to the last point of the previous bezier curve.

The position of neighboring control points around a shared endpoint determines the spline's continuity class. Mirrored endpoints implies C^1 continuity and G^1 continuity; aligned endpoints means C^0 continuity but G^1 continuity; otherwise the point is C^0 and G^0 continuous.

Mirrored endpoints are often preferred for their velocity smoothness.

4.3 Rendering

To render a bezier spline/curve, we can discretize the parameter t and render the curve as a sequence of lines.

For each line from t_0 to t_1, first compute the position of the curve at the parameters: $c(t_0)$ and $c(t_1)$. Then compute the normal at t_0 and t_1 — that is, the rotated and normalized curve derivatives at the parameters, denoted by $n(t_0)$ and $n(t_1)$. The four rectangle corners of the line are then

$$c(t_0) \pm hn(t_0)$$
$$c(t_1) \pm hn(t_1)$$

where h is the half thickness of the line supplied by the user.

The whole curve can be converted into these discrete rectangles, and thus, vertices for the graphic pipeline to render. We can also automatically vary the segments on the bezier curve length.

4.4 Other curves

The other popular splines, which are just variations of the bezier spline, include

- The Hermite curve used in physics and animation
- The Catmull-Rom for path generation
- The b-spline for C^2 smoothness at the cost of losing interpolation

They are just a bit harder than the cubic bezier spline to compute; it may be interesting to discuss them in isolation sometime in the future.

5 Linear Programming

A linear program under normal form has the format:

$$\begin{aligned} \max_x (c^T x) \\ Ax &= b \\ x &\geq 0 \end{aligned}$$

for the vectors c, $x \in \mathbb{R}^n$ and b, with the matrix A.

To transform a generic linear program to its normal form (ie, one with inequalities and/or equalities), we must perform some transformations.

Define the program tableau to be

$$\begin{bmatrix} 1 & -c^T & 0 \\ 0 & A & b \end{bmatrix}$$

The simplex method also works for $x \in \mathbb{Z}^n$ under some modifications.

6 Miller-rabin test

The Miller-rabin test is a probabilistic prime testing algorithm of complexity $O(k \log n)$, where k is a constant depending on the level of confidence.

For an odd number x, let the positive integers s and d be such that

$$x - 1 = d \times 2^s$$

Define a witness to be a natural number a, with x being a probable prime of base a if it satisfies the expressions:

$$\begin{aligned} a &\equiv 0 \mod n \\ a^d &\equiv 1 \mod n \\ \forall r \in [0, s), a^{d \times 2^r} &\equiv n - 1 \mod n \end{aligned}$$

If x is not a probable prime of some base a', then x must be composite. There is a possibility that x is a probable prime of base a' but is indeed composite, then for x, a' is a false witness.

To deterministically detect all primes of size $< 2^{64}$, we use the following witnesses

$$[2, 325, 9375, 28178, 450775, 9780504, 1795265022]$$

6.1 Implementation

7 BWT, MFT

8 PGA, Projective Geometric Algebra

8.1 Linear space of lines

9 Table allocations

Given N tables of size K, and $P = N \times K$ people seated at said tables. If x_{ij} denotes the perceived value of seating person j in the same table as i (which may not be commutative), and y_{ij} denotes the cost of not seating j next to i, what is the optimal seating plan that maximizes the overall value.

Denote the state space of all possible seating plan as the vector S_{ki} representing whether person i is seated at table k, and $\dim S = N^2K$. The total value of this seating plan V is

$$V(S) = \sum_{k}^{K} \sum_{i}^{P} S_{ki} \left[\sum_{j \neq i}^{P} \left(\underbrace{S_{kj} x_{ij}}_{\text{Value}} - \underbrace{(1 - S_{kj}) y_{ij}}_{\text{Cost}} \right) \right]$$

If $x_{ij} = y_{ij}$, that is, the cost of not seating person j next to i is the value of seating J next to i, the valuation function simplifies to

$$V(S) = \sum_{k}^{K} \sum_{i}^{P} S_{ki} \left[\sum_{j \neq i}^{P} x_{ij} (2S_{kj} - 1) \right]$$

We can minimize $V(S)$ and find the optimal (or a list of optimal) seating state using a blackbox optimizer. Note that the cost function is non-linear.

10 Random Walks

A random walk is depicted as a time series where the successive values are dependent by the previous values. Namely, the random walk $\{x_i\}$ with initial value x_0 is represented by

$$x_{i+1} = f(x_i)$$

We are often interested in

- The expected value of the random walk
- The long term convergence of the random walk
- The variance of the walk
- The distribution of time in reaching a value

10.1 Simple Random Walk

Consider the simple additive random walk

$$f(x) = x + K$$

where $K \sim B(p, a, b)$ is a Bernoulli random variable with a p probability of taking a, and a $1 - p$ probability of taking b.

Because K is simply a rescaled Bernoulli random variable,

$$K = (b - a)X + a$$

where $X \sim B(p)$. The mean and variance of K is

$$\begin{aligned} E[K] &= (b - a)E[X] + a \\ &= (b - a)p + a \\ &= bp + (1 - p)a \\ V[K] &= (b - a)^2 V[X] \\ &= (b - a)^2 p(1 - p) \end{aligned}$$

Notice that the random walk at time t is

$$x_t = x_0 + tK$$

hence

$$\begin{aligned} E[x_t] &= x_0 + t(bp + (1 - p)a) \\ V[x_t] &= t(b - a)^2 p(1 - p) \end{aligned}$$