CPE 490 590: Machine Learning for Engineering Applications

05 Optimization and Gradient Descent

Rahul Bhadani

Electrical & Computer Engineering, The University of Alabama in Huntsville

Outline

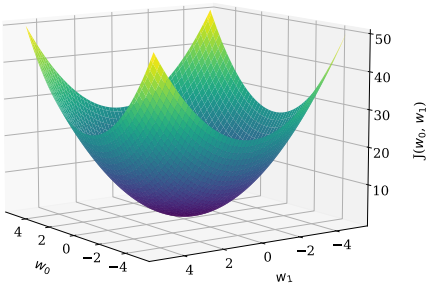
Logistics

Optimization

What is Optimization?

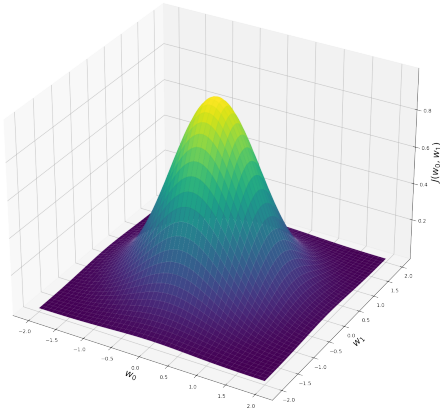
1. Maximizing or minimizing some function relative to some set, often representing a range of choices available in a certain situation.
2. The function allows comparison of the different choices for determining which might be "best".

A Function with a Minimum



This function has a single global minimum.

A Function with a Maximum



This function has a single global maximum.

How do we find global maximum/minimum

Finding Maxima/Minima of One Variable

A Function of One Variable: $f(x)$

To find the local maxima and minima of a function f on an interval $[a, b]$:

- Solve for $f'(x) = 0$ to obtain critical point(s) c.
- Drop from the list any critical points that aren't in the interval $[a, b]$.
- $x = c$, is a point of local maxima if $f'(c) = 0$, and $f''(c) < 0$. The point at $x = c$ is the local maxima and $f(c)$ is called the **local maximum** value of $f(x)$.
- $x = c$ is a point of local minima if $f'(c) = 0$, and $f''(c) > 0$. The point at $x = c$ is the local minima and $f(c)$ is called the **local minimum** value of $f(x)$.

Example 1

Find the local maxima and local minima of the function

$$f(x) = 2x^3 + 3x^2 - 12x + 5$$

Example 1

Solution:

$$\frac{\partial f(x)}{\partial x} = 2 \cdot 3x^2 + 3 \cdot 2x - 12 = 0$$

$$6x^2 + 6x - 12 = 0$$

$$x^2 + x - 2 = 0$$

$$x^2 + 2x - x - 2 = 0$$

$$x(x+2) - 1(x+2) = 0$$

$$(x+2)(x-1) = 0$$

$$\Rightarrow x = -2, x = 1$$

$$\frac{\partial^2 f(x)}{\partial x^2} = 12x + 6$$

$$\left.\frac{\partial^2 f(x)}{\partial x^2}\right|_{x=-2} = -18 < 0$$

$$\left.\frac{\partial^2 f(x)}{\partial x^2}\right|_{x=1} = 18 > 0$$

Hence, $x = -2$ for local maximum, and $x = 1$ is for local minimum. $f(-2)$ is local maximum and $f(1)$ is local minimum.

Finding Maxima/Minima of Two Variables

Consider a function $f(x, y)$ of two variables x and y.
If f has a local extremum (or critical point) at (a, b), and if the first order partial derivatives of f exist there, then

$$\frac{\partial f}{\partial x}(a, b) = 0 \quad \text{and} \quad \frac{\partial f}{\partial y}(a, b) = 0$$

or

$$\nabla f(a, b) = \begin{bmatrix} \frac{\partial f}{\partial x}(a, b) & \frac{\partial f}{\partial y}(a, b) \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix} = \mathbf{0}$$

At a critical point, a function could have a local maximum, or a local minimum, or neither.

Hessian Function

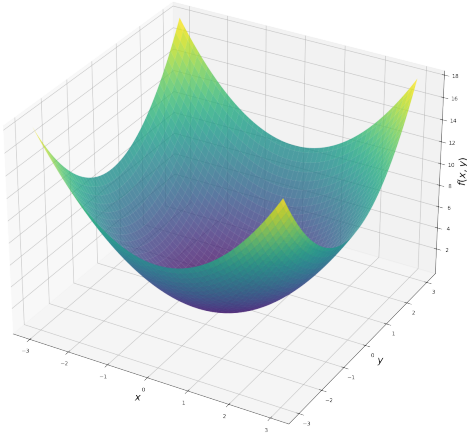
If a $f(x, y, z)$ has critical points (a, b, c), then in order to check if it is a maximum, minimum or a saddle point, we need to compute Hessian matrix first.

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

1. If all eigenvalues of Hessian function are positive, then H is positive definite, and critical point is local minimum.
2. If all eigenvalues of Hessian function are negative, then H is positive definite, and critical point is local maximum.
3. If eigenvalues of Hessian function are both positive and negative, then critical point is a saddle point.

Example 2

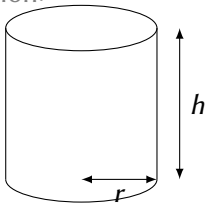
Let $z = f(x, y) = x^2 + y^2$. The only critical point is $(0, 0)$. At $(0, 0)$, f has a minimum.



Example 3

A cylindrical hot water tank is to have a capacity of 4 m^3. Find the radius and height that would have the least surface area.

Solution:



$$\pi r^2 h = 4$$

$$h = \frac{4}{\pi r^2}$$

Surface Area = $2\pi r^2 + 2\pi rh$

Example 3 Continues ...

Solution:

$$S = 2\pi r^2 + 8r^{-1}$$

$$\frac{\partial S}{\partial r} = 4\pi r - 8r^{-2}$$

$$4\pi r - 8r^{-2} = 0$$

$$r^3 = \frac{8}{4\pi}$$

$$\Rightarrow r = 0.860$$

$$\frac{\partial^2 S}{\partial r^2} = 4\pi + 16r^{-3}$$

$$= 4\pi + \frac{16}{(0.860)^3} > 0$$

Hence $r = 0.860$ gives minimum surface area,

and $S = 2\pi(0.860)^2 + \frac{8}{0.860} = 13.949$

Example 4

Compute local minima/maxima of $f(x, y, z) = x^2 + 2y^2 + 3z^2 - 4x - 6y - 8z + 10$

Solution:

$$\nabla f(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right) = (2x - 4, 4y - 6, 6z - 8)$$

Setting them to 0, gives $x = 2$, $y = \frac{3}{2}$, $z = \frac{4}{3}$.
Computing the Hessian matrix:

$$H = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 6 \end{bmatrix}$$

The Hessian matrix's eigen values are $2, 4, 6$ and hence the Hessian matrix is positive definite. Hence, the critical point given by $x = 2$, $y = \frac{3}{2}$, $z = \frac{4}{3}$ is arg minimum with minimum value at $f(2, \frac{3}{2}, \frac{4}{3})$.

Constraint Optimization: Lagrange Multiplier

Lagrange Multiplier

The method of Lagrange Multipliers is an excellent technique for finding the global maximum and global minimum values of a function $f(x, y)$ when the values of x and y that need to be considered are subject to some form of constraint, usually expressed as an equation $g(x, y) = 0$.

Problem Formulation

Minimize/Maximize $f(x, y)$

subject to

$g(x, y) = 0.$

In such a case, the optimality occurs when $\nabla f + \lambda \nabla g = 0$.

Example 5

Maximize/Minimize

Cost function: $f(x, y) = 81x^2 + y^2$, subject to the constraint $4x^2 + y^2 = 0$

Solution:

Lagrange Multiplier is $L(x, y, \lambda) = f(x, y) - \lambda g(x, y)$ where $g(x, y) = 4x^2 + y^2 - 0 = 0$.

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 162x & 2y \end{bmatrix}, \quad \nabla g = \begin{bmatrix} \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{bmatrix} = \begin{bmatrix} 8x & 2y \end{bmatrix}$$

Example 5 ...

$$\frac{\partial f}{\partial x} - \lambda \frac{\partial g}{\partial x} = 0 \Rightarrow 162x - \lambda 8x = 0$$
$$\frac{\partial f}{\partial y} - \lambda \frac{\partial g}{\partial y} = 0 \Rightarrow 2y - \lambda 2y = 0$$

From the first equation, $8x(21 - \lambda) = 0 \Rightarrow \lambda = 21$ or $x = 0$. From the second equation, $2y(1 - \lambda) = 0 \Rightarrow y = 0$ or $\lambda = 1$.
If $x = 0 \Rightarrow 4 \cdot 0^2 + y^2 = 9 \Rightarrow y = \pm 3$. Hence the first critical point is $(0, \pm 3)$.
If $y = 0 \Rightarrow 4x^2 + 0^2 = 9 \Rightarrow x = \pm 3/2$. Hence the second critical point is $(\pm 3/2, 0)$.
$f(0, \pm 3) = 9$ and $f(\pm 3/2, 0 = 729/4 = 182.25$.

Example 6

Maximize/Minimize

Cost function: $f(x, y) = 8x^2 + 2y$, subject to the constraint $x^2 + y^2 = 1$

Example 7

Maximize/Minimize

Cost function: $f(x, y, z) = y^2 - 10z$, subject to the constraint $x^2 + y^2 + z^2 = 36$

↴ Gradient Descent for Optimization

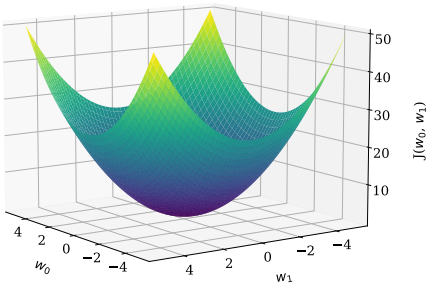
Algorithmic Approach to Minimizing the Cost Function: Gradient Descent

The cost function for linear regression with one variable:

$$J(w_0, w_1) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2 \tag{1}$$

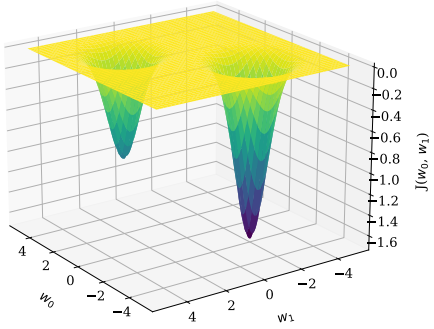
which is quadratic in w_0, w_1.

Gradient Descent



This function has a single global minimum.

Gradient Descent



This function has two local minima (out of which one is the global minimum).

Gradient Descent Algorithm

We find the minima iteratively:

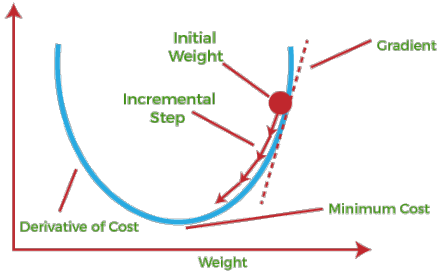
$$w_0 \leftarrow w_0 - \eta \frac{\partial}{\partial w_0} J(w_0, w_1)$$
$$w_1 \leftarrow w_1 - \eta \frac{\partial}{\partial w_1} J(w_0, w_1) \tag{2}$$

Gradient Descent Algorithm: Vectorize Form

The partial derivative is the **gradient** (slope towards the minimum). We iteratively calculate the above equation until w_0 and w_1 converge. η is called the learning rate. Succinctly, we can write it as

$$\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) - \eta \nabla J(\mathbf{w}(t)) \quad \text{where} \quad \nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial}{\partial w_0} J(w_0, w_1) \\ \frac{\partial}{\partial w_1} J(w_0, w_1) \end{bmatrix} \tag{3}$$

Gradient Descent Visualization



Gradient Descent Algorithm: Example 8

Simple Cost Function

$$J(w_1) = (w_1 - 5)^2 + 10$$

Gradient Descent Algorithm: Example 8

Simple Cost Function

$$J(w_1) = (w_1 - 5)^2 + 10$$
$$\frac{\partial J(w_1)}{\partial w_1} = 2w_1 - 10 \tag{4}$$

Gradient Descent Algorithm: Example 8

Simple Cost Function

$$
\begin{aligned}
J(w_1) &= (w_1 - 5)^2 + 10 \\
\frac{\partial J(w_1)}{\partial w_1} &= 2w_1 - 10
\end{aligned}
\tag{4}
$$

We could solve it analytically, but currently, we are interested in solving it numerically using a gradient descent algorithm: $w_1 = w_1 - \eta \frac{\partial J(w_1)}{\partial w_1}$.

Gradient Descent Algorithm: Example 8

Simple Cost Function

$$J(w_1) = (w_1 - 5)^2 + 10$$
$$\frac{\partial J(w_1)}{\partial w_1} = 2w_1 - 10 \tag{4}$$

We could solve it analytically, but currently, we are interested in solving it numerically using a gradient descent algorithm: $w_1 = w_1 - \eta \frac{\partial J(w_1)}{\partial w_1}$.

We start with a few values of w_1 and η.

Batch Gradient Descent

- All training data are taken into account to take a single step.
- Take the mean of the gradients of all the training examples to update our parameters.
- Requires the entire dataset to be available in the memory.

For example, we calculate the gradient of the following:

$$J(w_0, w_1) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2 \tag{5}$$

Mini-batch Gradient Descent

- Instead of the entire dataset: we take small random subsets of the training data (mini-batches) at each iteration.

We sample a random subset $C_t \subset \{1, 2, \cdots, n\}$, $|C| = b << m$, the batch size, and our update rule is

$$\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) - \eta \frac{1}{b} \sum_{i \in C_t} \nabla J_i(\mathbf{w}(t)) \tag{6}$$

and full gradient is approximated as

$$\mathbb{E}\left[\frac{1}{b} \sum_{i \in C_t} \nabla J_i(\mathbf{w})\right] = \nabla J(\mathbf{w}) \tag{7}$$

Stochastic Gradient Descent

- Just use a single training example (or data sample) until a sufficient value is reached for $\mathbf{w}$.
  1. Take a training data sample,
  2. Feed to the equation model,
  3. Calculate the gradient,
  4. Update the weights,
  5. Repeated 1-4 for all data samples.

While the batch gradient descent always converges, it is slow and costly. Stochastic gradient descent is good for the large dataset, makes progress faster, but oscillates in terms converging.

TensorFlow for Gradient Descent

```
import tensorflow as tf
import matplotlib.pyplot as plt
# Define the cost function J(w_1)
def J(w_1):
    return (w_1 - 5)**2 + 10

# Initialize w_1
w_1 = tf.Variable(0.0)
# Set up the optimizer
optimizer =
  tf.optimizers.SGD(learning_rate=0.1)
# Store the values of w_1 for plotting
w_1_values = []
```

```
# Perform gradient descent
for i in range(100):
    with tf.GradientTape() as tape:
        loss = J(w_1)
    gradients =
      tape.gradient(loss, [w_1])
    optimizer.apply_gradients(
        zip(gradients, [w_1]))
    w_1_values.append(w_1.numpy())

plt.plot(w_1_values)
plt.xlabel('Iteration')
plt.ylabel('$w_1$')
```

Automatic Differentiation

Automatic Differentiation

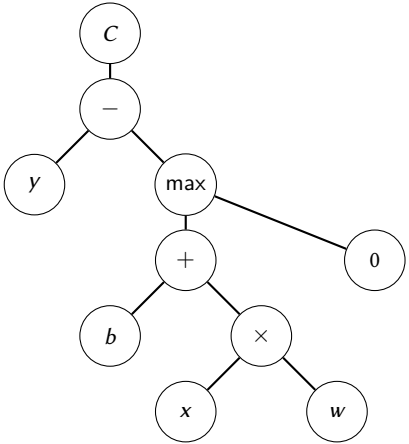
Automatically calculate the derivative of a function by repeatedly applying the chain rule. It can calculate the partial differentiation wrt many inputs. Read more:

1. https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec10.pdf
2. https://en.wikipedia.org/wiki/Automatic_differentiation
3. https://www.tensorflow.org/guide/autodiff

Automatic Differentiation Example

We will construct expression graph of $C(y, wx + b) = y - max(0, wx + b)$ to understand how automatic differentiation works. Automatic differentiation only works with numerical values, hence, let $y = 5,$, $w = 2$, $x = 1$, $b = 1$.

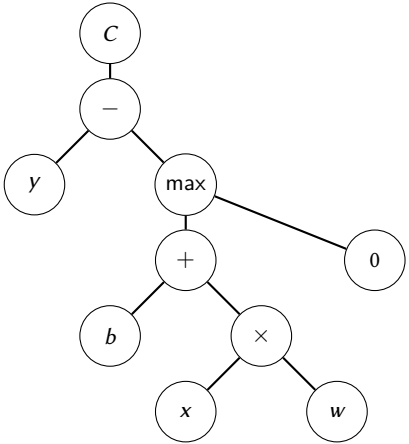
Our target is to calculate $\frac{\partial C}{\partial w_1}$.



Automatic Differentiation Example

Let:

$$w_1 = w = 2 \quad w_2 = x = 1$$
$$w_3 = b = 1 \quad w_4 = y = 5$$
$$w_5 = w_1 w_2 = 2 \quad w_6 = w_3 + w_5 = 3$$
$$w_7 = max(0, w_6) = 3$$
$$w_8 = w_4 - w_7 = 2$$
$$C = w_8 = 2$$



Automatic Differentiation Example

$$\frac{\partial w_5}{\partial w_1} = \frac{\partial (w_1 \cdot w_2)}{\partial w_1} = w_2$$
$$\frac{\partial w_5}{\partial w_2} = \frac{\partial (w_1 \cdot w_2)}{\partial w_2} = w_1$$
$$\frac{\partial w_6}{\partial w_5} = \frac{\partial (w_5 + w_3)}{\partial w_3} = 1$$
$$\frac{\partial w_7}{\partial w_6} = \frac{\partial max(0, w_6)}{\partial w_6} = \begin{cases} 0, & w_6 < 0 \\ 1, & w_6 > 0 \end{cases}$$

$$\frac{\partial w_8}{\partial w_7} = \frac{\partial (w_4 - w_7)}{\partial w_7} = -1$$
$$\frac{\partial w_8}{\partial w_4} = \frac{\partial (w_4 - w_7)}{\partial w_4} = 1$$
$$\frac{\partial C}{\partial w_8} = 1$$

Apply chain rule:

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial w_8} \times \frac{\partial w_8}{\partial w_7} \times \frac{\partial w_7}{\partial w_6} \times \frac{\partial w_6}{\partial w_5} \times \frac{\partial w_5}{\partial w_1} = ?$$

Gradient Tape

Gradient Tape

TensorFlow provides Gradient Tape as a python context `tf.GradientTape`. It means any operations within the context of `tf.GradientTape` will compute the gradients and record them on a *tape*.

Example 9

```
x = tf.Variable(3.0)
with tf.GradientTape() as tape:
  y = x**3
dy_dx = tape.gradient(y, x)
dy_dx.numpy()
```

What's the output?

Some Other Optimizers

Gradient Descent is one of the many optimizers that can be used to optimize a function and find its minima. Some other commonly popular optimizers are

1. RMSProp
2. Adagrad (Adaptive Gradient)
3. SGD with Momentum
4. SGD with Nesterov Accelerated Gradient
5. AdaDelta
6. ADAM

RMSProp

RMSProp (Root Mean Square Propagation) is an adaptive learning rate method that divides the learning rate by an exponentially decaying average of squared gradients.

- Update rule:

$$
\begin{aligned}
g_t &= \nabla_\theta J(\theta_t) \\
v_t &= \beta v_{t-1} + (1-\beta) g_t^2 \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} g_t
\end{aligned}
$$

- η: Learning rate
- β: Decay rate (typically 0.9)
- ϵ: Small constant for numerical stability

Adagrad (Adaptive Gradient)

Adagrad adapts the learning rate to the parameters, performing smaller updates for frequently occurring features and larger updates for infrequent ones.

- Update rule:

$$
\begin{aligned}
g_t &= \nabla_\theta J(\theta_t) \\
G_t &= G_{t-1} + g_t^2 \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{G_t} + \epsilon} g_t
\end{aligned}
$$

- η: Learning rate
- G_t: Accumulated squared gradients
- ϵ: Small constant for numerical stability

SGD with Momentum

SGD with Momentum accelerates gradient descent by adding a fraction of the previous update to the current update.

- Update rule:

$$
\begin{aligned}
g_t &= \nabla_\theta J(\theta_t) \\
v_t &= \gamma v_{t-1} + \eta g_t \\
\theta_{t+1} &= \theta_t - v_t
\end{aligned}
$$

- η: Learning rate
- γ: Momentum term (typically 0.9)
- v_t: Velocity vector

SGD with Nesterov Accelerated Gradient

Nesterov Accelerated Gradient (NAG) improves momentum by calculating the gradient at the estimated future position.

- Update rule:

$$
\begin{aligned}
g_t &= \nabla_\theta J(\theta_t - \gamma v_{t-1}) \\
v_t &= \gamma v_{t-1} + \eta g_t \\
\theta_{t+1} &= \theta_t - v_t
\end{aligned}
$$

- η: Learning rate
- γ: Momentum term (typically 0.9)
- v_t: Velocity vector

AdaDelta

AdaDelta is an extension of Adagrad that reduces its aggressive, monotonically decreasing learning rate.

- Update rule:

$$
\begin{aligned}
g_t &= \nabla_\theta J(\theta_t) \\
E[g^2]_t &= \rho E[g^2]_{t-1} + (1-\rho) g_t^2 \\
\Delta\theta_t &= -\frac{\sqrt{E[\Delta\theta^2]_{t-1} + \epsilon}}{\sqrt{E[g^2]_t + \epsilon}} g_t \\
E[\Delta\theta^2]_t &= \rho E[\Delta\theta^2]_{t-1} + (1-\rho)\Delta\theta_t^2 \\
\theta_{t+1} &= \theta_t + \Delta\theta_t
\end{aligned}
$$

- ρ: Decay rate (typically 0.9)
- ϵ: Small constant for numerical stability

ADAM (Adaptive Moment Estimation)

ADAM combines the benefits of RMSProp and Momentum by adapting learning rates and using momentum.

- Update rule:

$$
\begin{aligned}
g_t &= \nabla_\theta J(\theta_t) \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t
\end{aligned}
$$

- η: Learning rate
- β_1, β_2: Exponential decay rates (typically 0.9 and 0.999)
- ϵ: Small constant for numerical stability

Penalty Method in Constraint Optimization

Pentalty Function

In order to use PyTorch and Gradient Descent Method, we need to reformulate constraint optimization problem using penalty function.

minimize $f(\mathbf{x})$
subject to $h(\mathbf{x})$

changes to

minimize $f(\mathbf{x}) + \mu h^2(\mathbf{x})$
where μ is a penalty term.

An optimal solution theoretically would have $h^2(\mathbf{x})$ close to zero. The square term ensures that the penalty is always non-negative.

Limitations of Penalty Method

- If the penalty term is too small, then the constraint may not be satisfied
- If the penalty term is too large, then the solution may be ill-condition, potentially leading to NaN (Not a Number).

Why does Penalty Method Work

The penalty term $\mu h^2(\mathbf{x})$ acts as a "soft constraint", pushing the solution toward the feasible region where $h(\mathbf{x}) = 0$.

Reference:

1. https://ocw.mit.edu/courses/15-084j-nonlinear-programming-spring-2004/13b4c46d590cf3df94475e0ae1a815c3_lec10_penalty_mt.pdf
2. https://solmaz.eng.uci.edu/Teaching/MAE206/Lecture14.pdf
3. https://www.rose-hulman.edu/~bryan/lottamath/penalty.pdf

Projected Gradient Method

Recall Gradient Descent

$$\mathbf{x}(t+1) \leftarrow \mathbf{x}(t) - \eta \nabla J(\mathbf{x}(t)) \tag{8}$$

for minimizing a function $J(\mathbf{x})$. If the constraint is $g(\mathbf{x}) = 0$, assume set a set $\mathbf{x} \in C$ that satisfies the constraint, then mathematically, we can also write the minimization problem as

$$\min_{x \in C} J(\mathbf{x}) \tag{9}$$

Projected Gradient with Gradient Descent

In projected gradient method, the idea is to take a step in the direction of the gradient descent and then project the updated solution back onto the feasible region to ensure the constraints are satisfied.

Hence, in **Projected Gradient Method** that incorporates the constraint, our update rule becomes

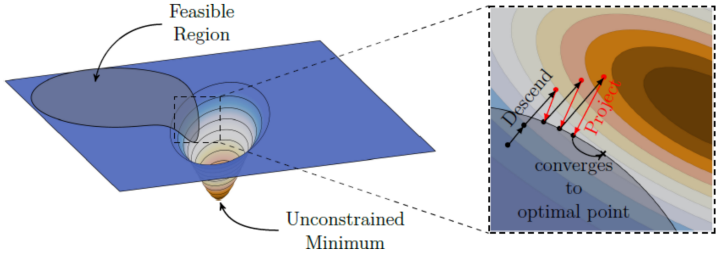
$$\mathbf{y}(t+1) \leftarrow \mathbf{x}(t) - \eta \nabla J(\mathbf{x}(t))$$
$$\mathbf{x}(t+1) = P_C(\mathbf{y}(t+1))$$

Finding Projection Point C

The next step is to find the projection point C closest to $\mathbf{y}(t+1)$ that is, we need to solve the optimization problem

$$\mathbf{x}(t+1) = P_C(\mathbf{y}(t+1)) \leftarrow \arg\min_{x \in C} \frac{1}{2}||\mathbf{x} - \mathbf{y}(t+1)||_2^2$$

Projected Gradient



Source: "Adaptive Model for Magnetic Particle Mapping Using Magnetoelectric Sensors, Friedrich & Faupel"

Further Reading

- https://neos-guide.org/guide/algorithms/gradient-projection/
- https://www.stat.cmu.edu/~siva/teaching/725/lec6.pdf
- https://www.cs.ubc.ca/~schmidtm/Courses/5XX-S20/S5.pdf

Code Availability and Python Notebook

Code used for this chapter is available at
https://github.com/rahulbhadani/CPE490_590_Sp2025/blob/master/Code/Chapter05_Optimization_and_Gradient_Descent.ipynb

The End