Electrical and Computer Engineering Department
Tarbiat Modares University

# Numerical Computation

Foad Ghaderi, PhD

## Numerical Computation

Machine learning algorithms usually require a high amount of numerical computation.

Solving mathematical problems:

- Analytically providing a symbolic expression for the correct solution
- Updating estimates of the solution via an iterative process

## Numerical Computation

Algorithms are often specified in terms of real numbers; real numbers cannot be implemented in a finite computer

- ❑ Does the algorithm still work when implemented with a finite number of bits?

- ❑ Do small changes in the input to a function cause large changes to an output?
    - o Rounding errors, noise, measurement errors can cause large changes
    - o Iterative search for best input is difficult

**Example**: Underflow and overflow in the softmax function.

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$$

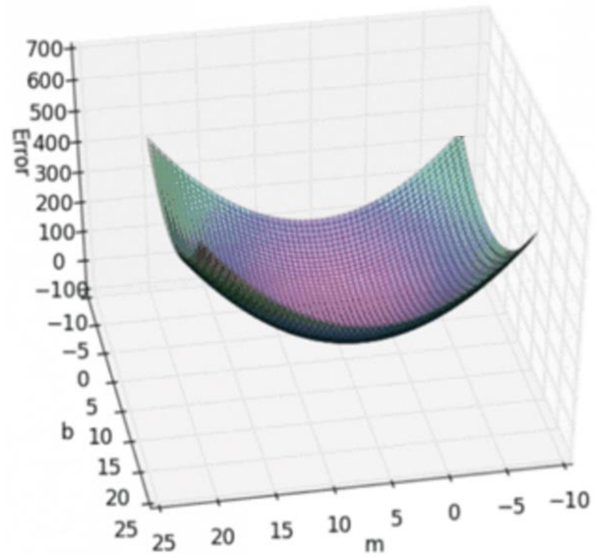
## Iterative Optimization

Optimization refers to the task of either minimizing or maximizing some function $f(\mathbf{x})$ by altering $\boldsymbol{x}$.

The function we want to minimize or maximize is called the *objective function* or *criterion*.

When we are minimizing it, we may also call it the *cost function*, *loss function*, or *error function*.

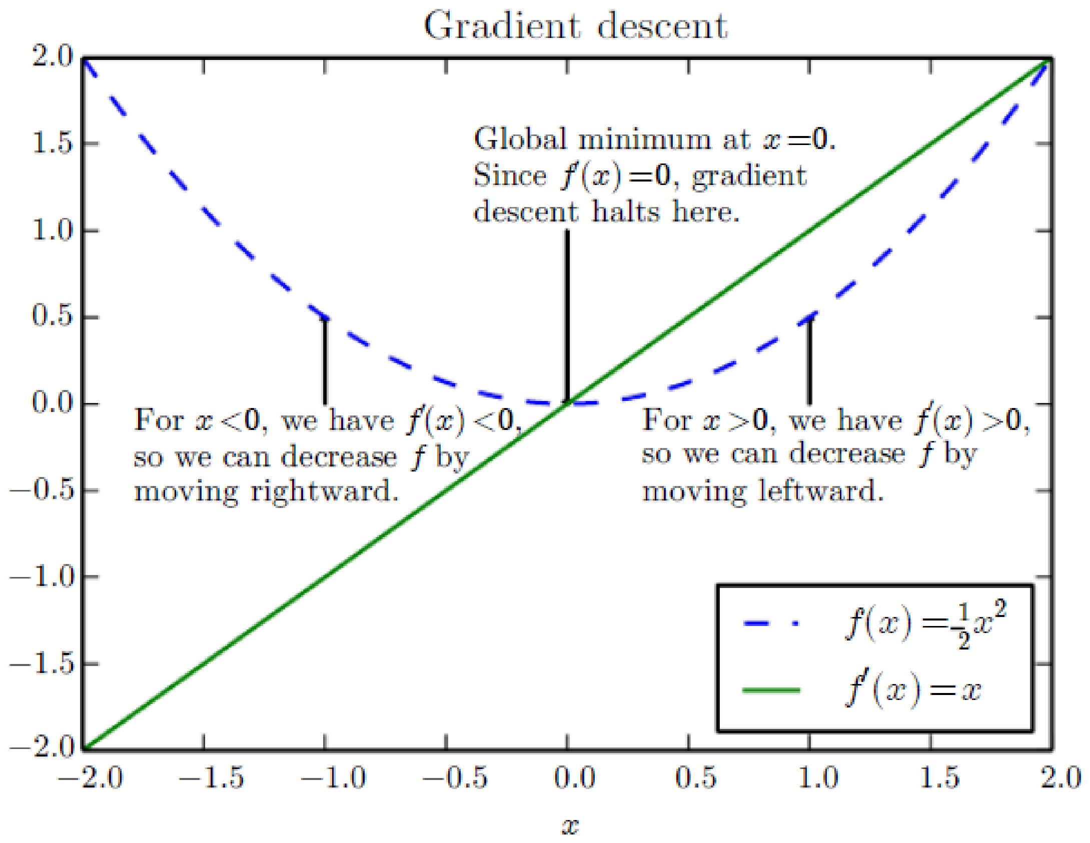
We often denote the value that minimizes or maximizes a function with a superscript $*$. For example, we might say $x^* = \arg\min f(\boldsymbol{x})$.



## Gradient-Based Optimization

**Gradient descent**

The derivative tells us how to change ***x*** in order to make a small improvement in *y*. For example, we know that $f(x - \varepsilon \operatorname{sign}(f'(x)))$ is less than $f(x)$ for small enough ε.



## Gradient-Based Optimization

**Gradient descent**

***For functions with multiple inputs:***

The *partial derivative* $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$ measures how *f* changes as only the variable $x_i$ increases at point $\boldsymbol{x}$.

The *gradient* of *f* is the vector containing all of the partial derivatives, denoted $\nabla_x f(\boldsymbol{x})$.

*The directional derivative* in direction $\boldsymbol{u}$ (a unit vector) is the slope of the function *f* in direction $\boldsymbol{u}$. Using the chain rule, we can see that $\frac{\partial}{\partial \alpha} f(\boldsymbol{x} + \alpha \boldsymbol{u}) = \boldsymbol{u}^T \nabla_x f(\boldsymbol{x})$.

# Gradient-Based Optimization

## Gradient descent

***For functions with multiple inputs:***

To minimize $f$ , we would like to find the direction in which $f$ decreases the fastest. Using the directional derivative:

$$\min_{\boldsymbol{u}, \boldsymbol{u}^\top \boldsymbol{u}=1} \boldsymbol{u}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

$$= \min_{\boldsymbol{u}, \boldsymbol{u}^\top \boldsymbol{u}=1} ||\boldsymbol{u}||_2 ||\nabla_{\boldsymbol{x}} f(\boldsymbol{x})||_2 \cos\theta$$

where $\theta$ is the angle between $\boldsymbol{u}$ and the gradient. Substituting in $\|\boldsymbol{u}\|_2 = 1$ and ignoring factors that do not depend on $\boldsymbol{u}$, this simplifies to

$$\min_{\boldsymbol{u}} \cos\theta$$

## Gradient-Based Optimization

**Gradient descent**

***For functions with multiple inputs:***

$$\min_{\boldsymbol{u}} \cos\theta$$

This is minimized when ***u*** points in the opposite direction as the gradient.

Therefore, we can decrease *f* by moving in the direction of the negative gradient. This is known as the *method of steepest descent* or *gradient descent*, proposing the new point as

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

where ε is the *learning rate*, a positive scalar determining the size of the step.

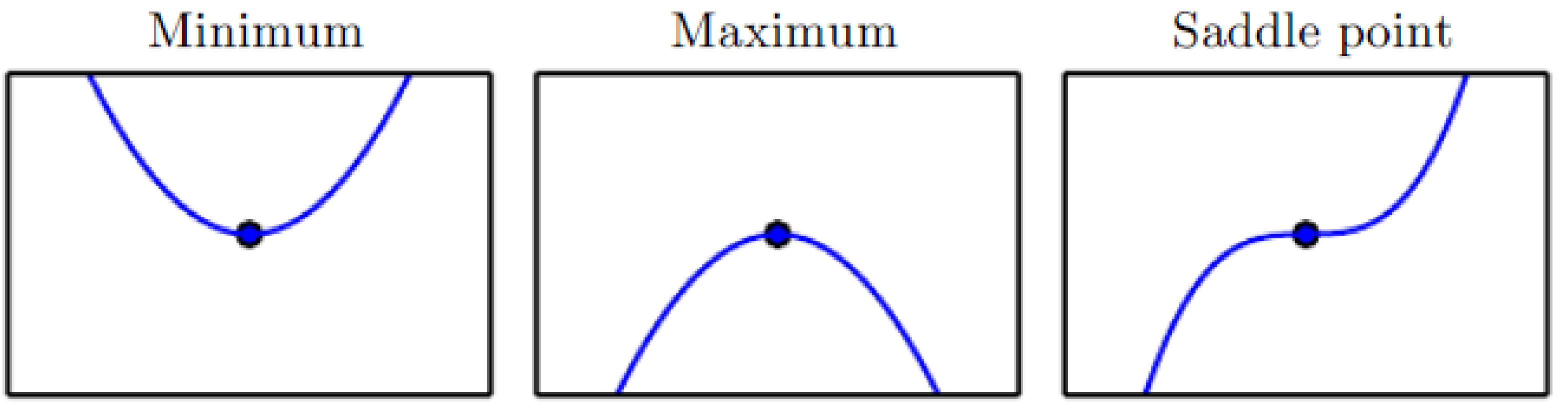
## Gradient-Based Optimization

When $f'(x) = 0$, the derivative provides no information about which direction to move.

**Critical points** or **Stationary points**

Points where $f'(x) = 0$ (the derivative provides no information about which direction to move.)

Examples of each of the three types of critical points in 1-D.



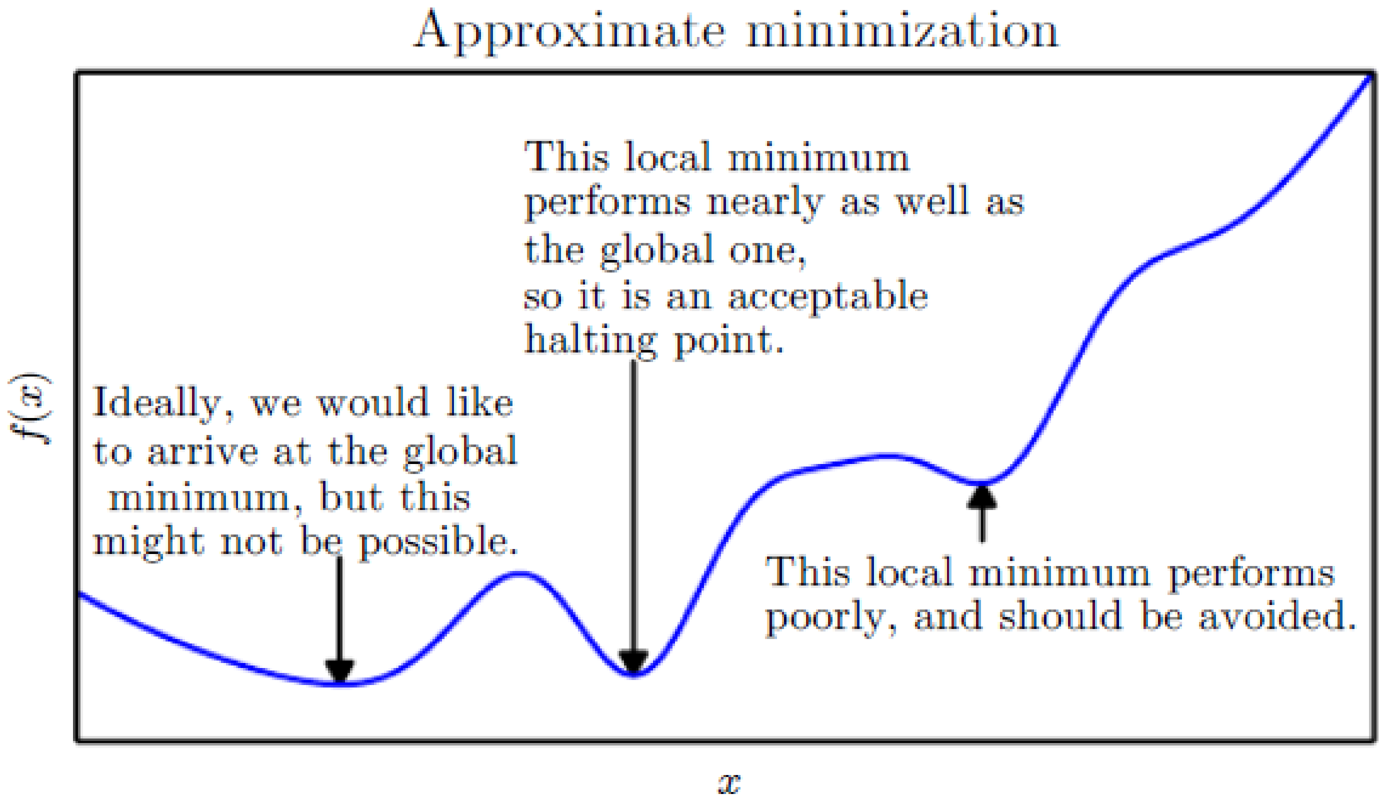
**Local minimum**: lower than the neighboring points,

**Local maximum**: higher than the neighboring points,

**Saddle point**: has neighbors that are both higher and lower

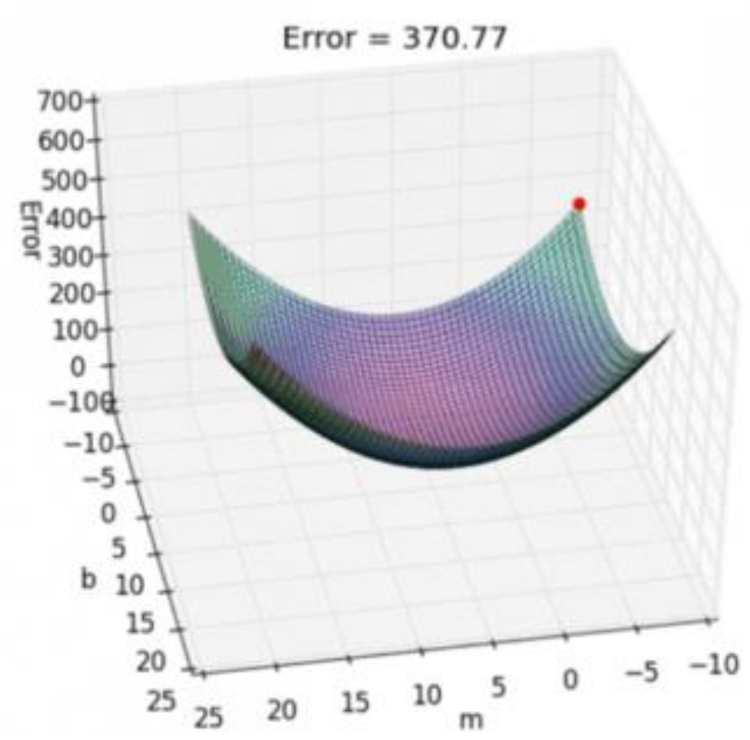
## Gradient-Based Optimization

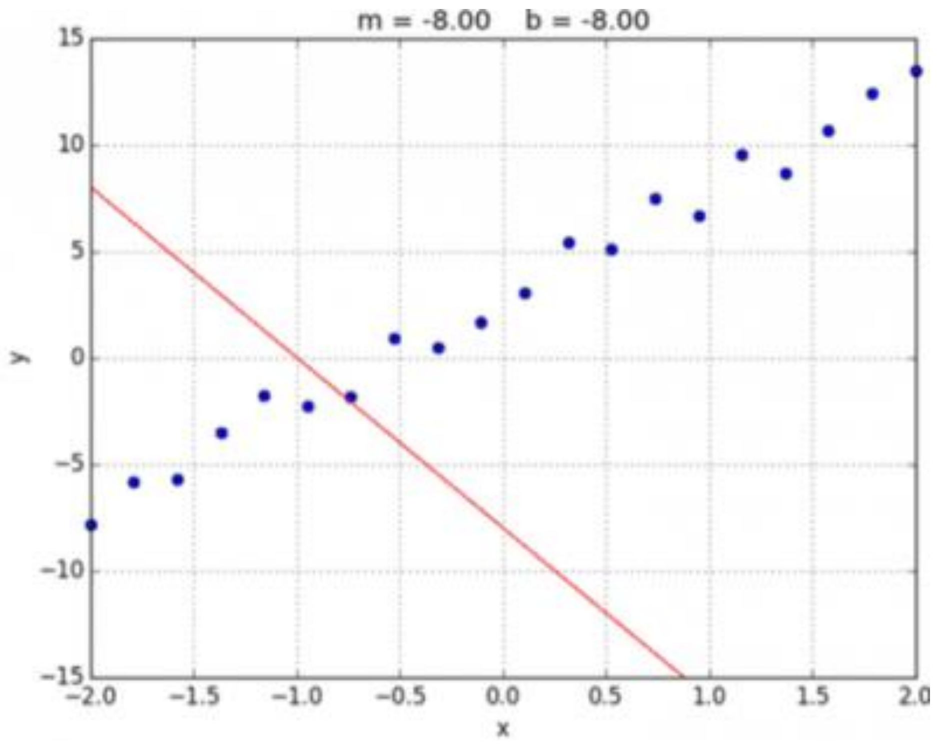
Optimization algorithms may fail to find a global minimum when there are multiple local minima or plateaus present.



In the context of deep learning, we generally accept local minima or plateaus solutions even though they are not truly minimal, so long as they correspond to significantly low values of the cost function.

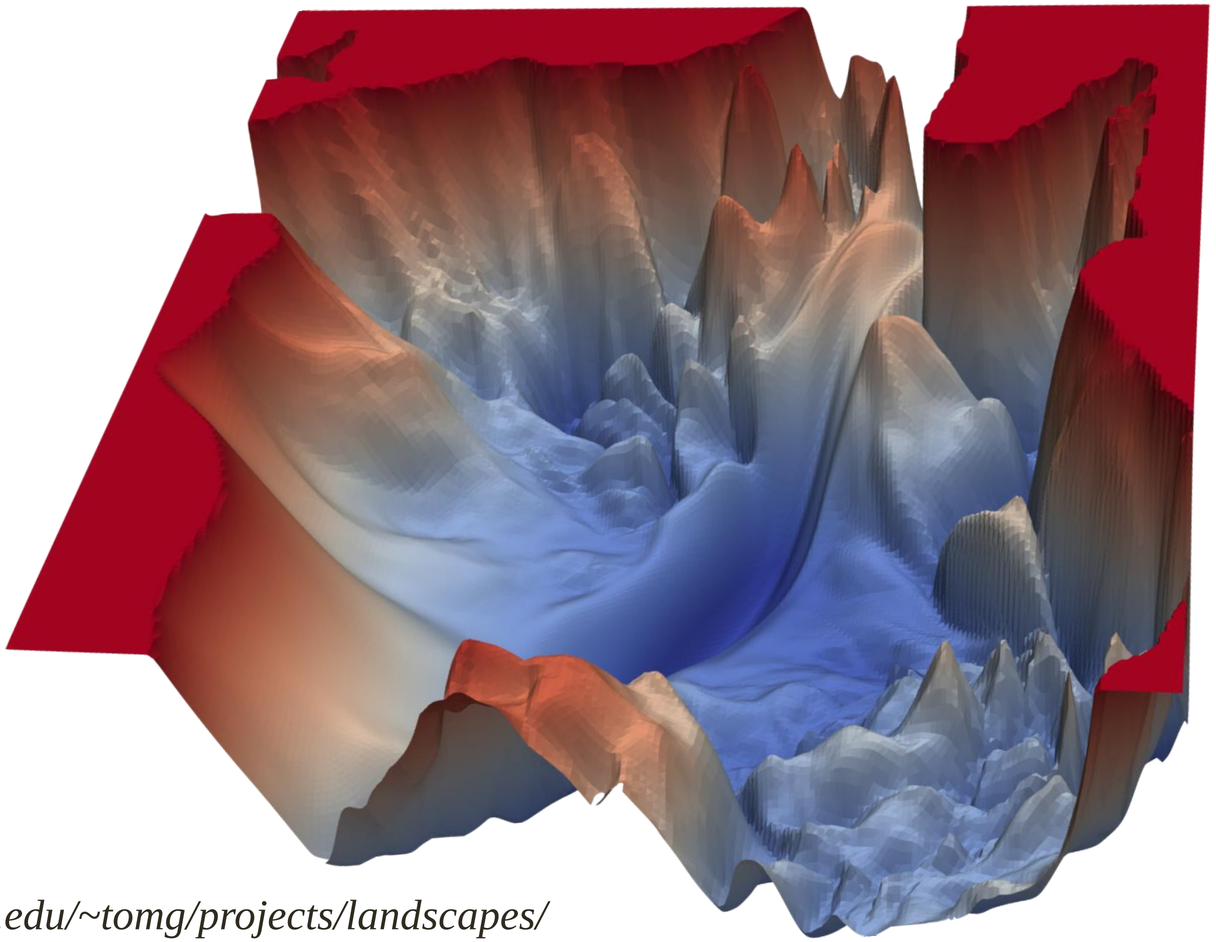
## Gradient-Based Optimization





## Gradient-Based Optimization

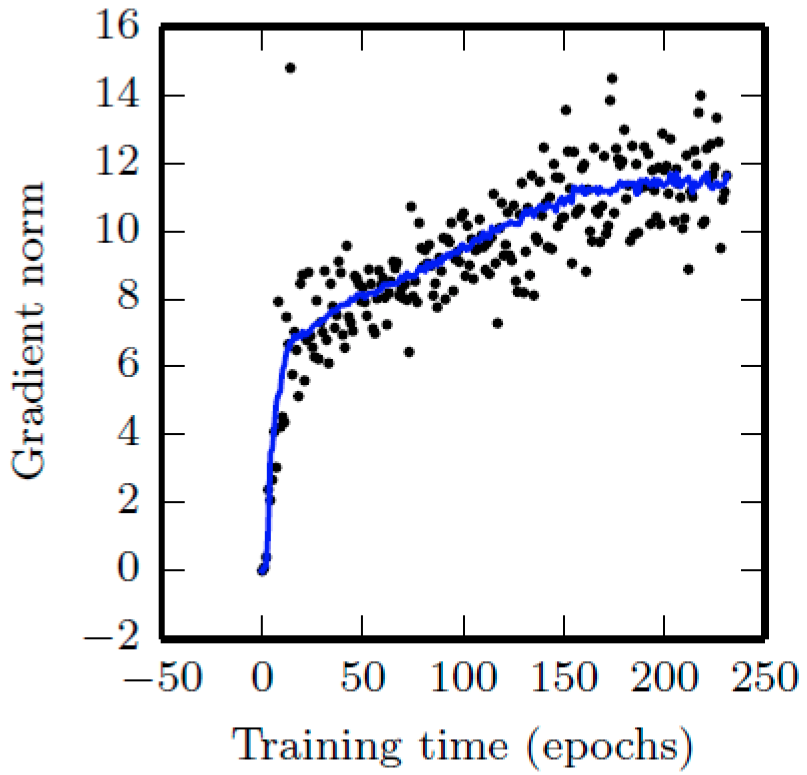
**A Complicated Loss Landscape**

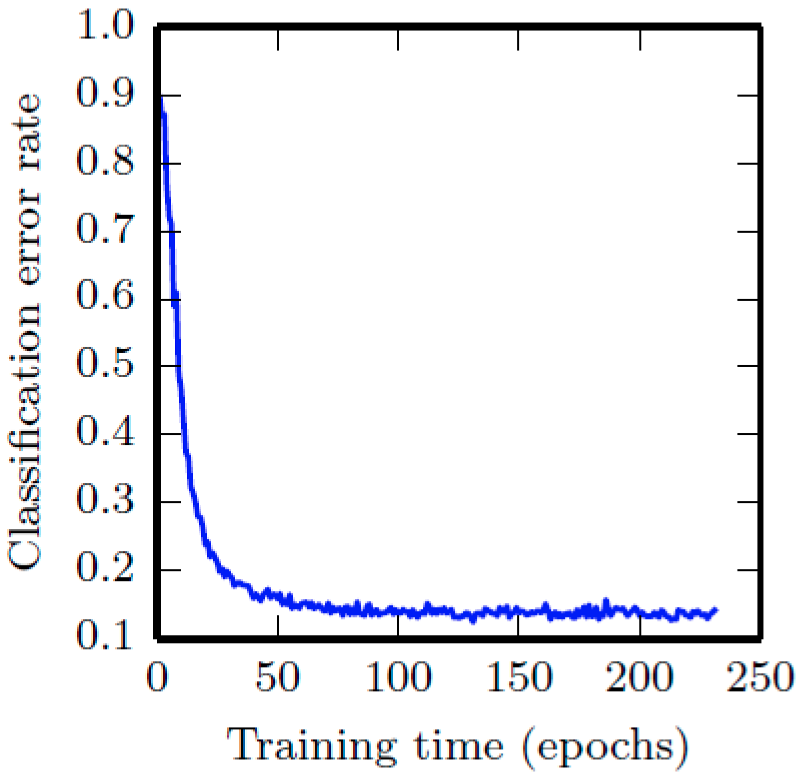


***Source:***
*https://www.cs.umd.edu/~tomg/projects/landscapes/*

## Gradient-Based Optimization

We usually don’t even reach a local minimum





## Constrained Optimization

Sometimes we wish not only to maximize or minimize a function $f(\boldsymbol{x})$ over all possible values of $\boldsymbol{x}$. Instead we may wish to find the maximal or minimal value of $f(\boldsymbol{x})$ for values of $\boldsymbol{x}$ in some set $\mathbb{S}$. This is known as *constrained optimization*.

Points $\boldsymbol{x}$ that lie within the set $\mathbb{S}$ are called ***feasible points*** in constrained optimization terminology.

We often wish to find a solution that is small in some sense. A common approach in such situations is to impose a norm constraint, such as $||\boldsymbol{x}|| \leq 1$.

One simple approach to constrained optimization is simply to modify gradient descent taking the constraint into account.

# Constrained Optimization

A more sophisticated approach is to design a different, unconstrained optimization problem whose solution can be converted into a solution to the original, constrained optimization problem.

**Example**

Minimize $f(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathbb{R}^{\mathbf{2}}$ with $\boldsymbol{x}$ constrained to have exactly unit $L^2$ norm.

we can instead minimize $g(\theta) = f([\cos\theta, \sin\theta]^T)$ with respect to $\theta$, then return $[\cos\theta, \sin\theta]$ as the solution to the original problem.

## Constrained Optimization

*Karush–Kuhn–Tucker* (KKT) Multipliers
A new function called the *generalized Lagrangian* or *generalized Lagrange function* is introduced.

we first need to describe $\mathbb{S}$ in terms of equations and inequalities.

$$\mathbb{S} = \{\boldsymbol{x} \mid \forall i, g^{(i)}(\boldsymbol{x}) = 0 \text{ and } \forall j, h^{(j)}(\boldsymbol{x}) \leq 0\}$$

The equations involving $g^{(i)}$ are called the *equality constraints* and the inequalities involving $h^{(j)}$ are called *inequality constraints*.
***KKT multipliers***: the new variables $\lambda_i$ and $\alpha_j$ for each constraint
**Generalized Lagrangian**

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x})$$

## Constrained Optimization

So long as at least one feasible point exists and *f* (***x***) is not permitted to have value ∞, then

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha},\boldsymbol{\alpha}\geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha})$$

has the same optimal objective function value and set of optimal points ***x*** as

$$\min_{\boldsymbol{x}\in\mathbb{S}} f(\boldsymbol{x})$$

Any time the constraints are satisfied,

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha},\boldsymbol{\alpha}\geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x})$$

while any time a constraint is violated

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha},\boldsymbol{\alpha}\geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = \infty$$

## Constrained Optimization

**Example: Linear Least Squares**

$$f(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2$$

Gradient-based optimization:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \boldsymbol{A}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top \boldsymbol{b}$$

**Algorithm 4.1** An algorithm to minimize $f(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2$ with respect to $\boldsymbol{x}$ using gradient descent.

```
Set the step size ($\epsilon$) and tolerance ($\delta$) to small, positive numbers.
while $||\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top \boldsymbol{b}||_2 > \delta$ do
  $\boldsymbol{x} \leftarrow \boldsymbol{x} - \epsilon (\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top \boldsymbol{b})$
end while
```

## Constrained Optimization

**Example: Linear Least Squares**

$$f(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2$$

Now suppose we have the constraint $\boldsymbol{x}^T\boldsymbol{x} \leq 1$.

$$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda\left(\boldsymbol{x}^\top\boldsymbol{x} - 1\right)$$

$$\min_{\boldsymbol{x}} \max_{\lambda, \lambda \geq 0} L(\boldsymbol{x}, \lambda)$$

The smallest-norm unconstraint solution (may be found using the Moore-Penrose pseudoinverse): $\boldsymbol{x} = \boldsymbol{A}^+ \boldsymbol{b}$,

- If feasible, then it is the solution,
- Otherwise, by differentiating the Lagrangian with respect to *x*

$$\boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top\boldsymbol{b} + 2\lambda\boldsymbol{x} = 0 \quad \rightarrow \quad \boldsymbol{x} = (\boldsymbol{A}^\top\boldsymbol{A} + 2\lambda\boldsymbol{I})^{-1}\boldsymbol{A}^\top\boldsymbol{b}$$

# Constrained Optimization

**Example: Linear Least Squares**

$$f(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2 \qquad L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda\left(\boldsymbol{x}^\top \boldsymbol{x} - 1\right)$$

$$\boldsymbol{x} = (\boldsymbol{A}^\top \boldsymbol{A} + 2\lambda \boldsymbol{I})^{-1}\boldsymbol{A}^\top \boldsymbol{b}$$

The magnitude of λ must be chosen such that the result obeys the constraint. We can find this value by performing gradient ascent on λ.

$$\frac{\partial}{\partial \lambda} L(\boldsymbol{x}, \lambda) = \boldsymbol{x}^\top \boldsymbol{x} - 1$$

If norm of *x* exceeds 1, the derivative is positive, so we increase λ in order to increase the Lagrangian.

✓ The coefficient on the $x^T x$ penalty has increased,

→ solving the linear equation for *x* will now yield a solution with smaller norm.