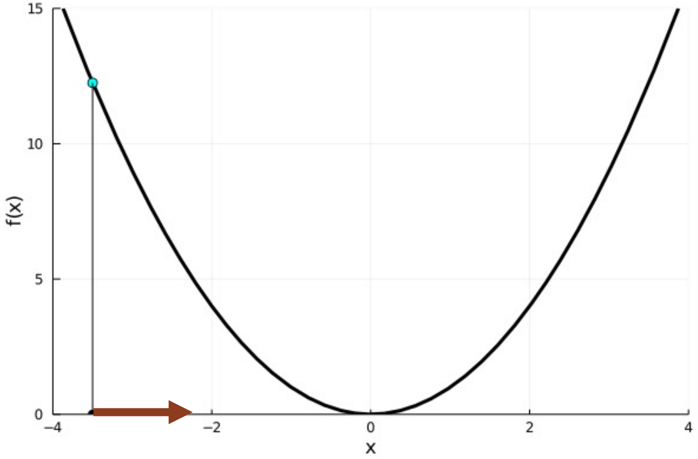
Local Descent

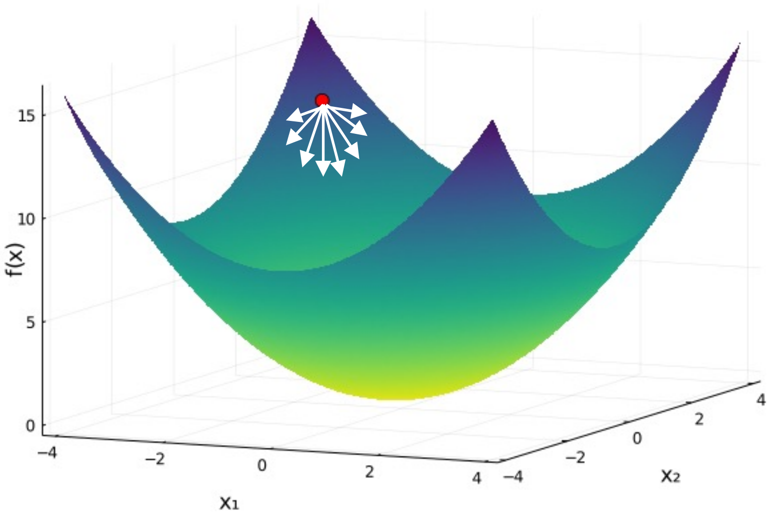
SYDNEY KATZ

AA222 Lecture
04.06.2021

smkatz@stanford.edu

Descent Direction Iteration





DEMO

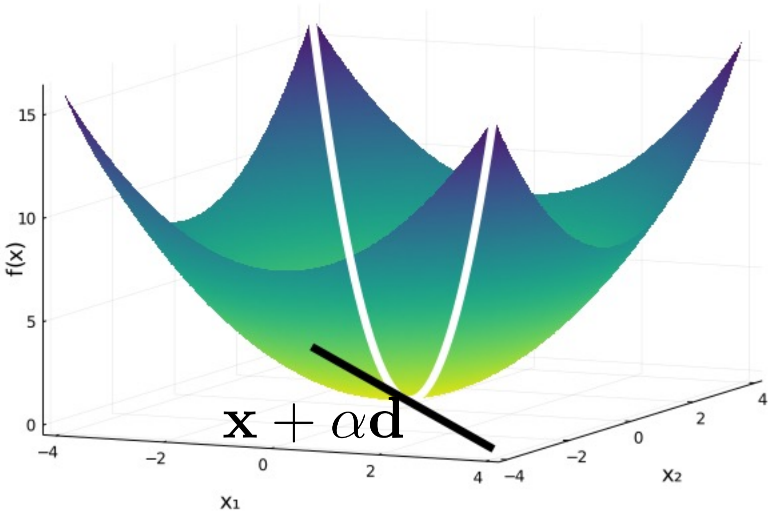
1 **Which direction** should we move in next?

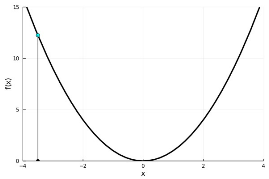
2 **How far** should we go?

DEMO

Stanford University

Line Search





$$\underset{\alpha}{\text{minimize}}\, f(\mathbf{x} + \alpha\mathbf{d})$$

```
function line_search(f, x, d)
    objective = α -> f(x + α*d)
    a, b = bracket_minimum(objective)
    α = minimize(objective, a, b)
    return x + α*d
end
```

This can be expensive!

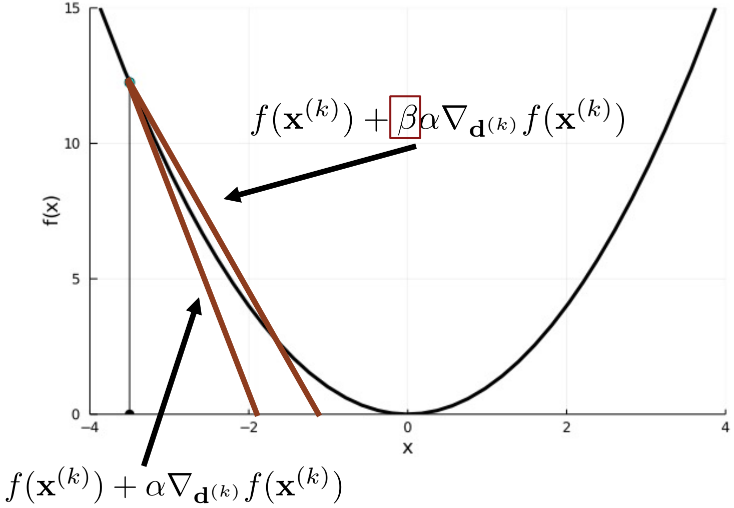
In practice, often use a fixed step size α, called the learning rate.

It is also common to decay the learning rate over time.

Approximate Line Search

We can enforce some conditions on our step size in order to encourage faster convergence.

Sufficient Decrease: $f(\mathbf{x}^{(k+1)}) \leq f(\mathbf{x}^{(k)}) + \beta\alpha\nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k)})$



$$0 \leq \beta \leq 1$$

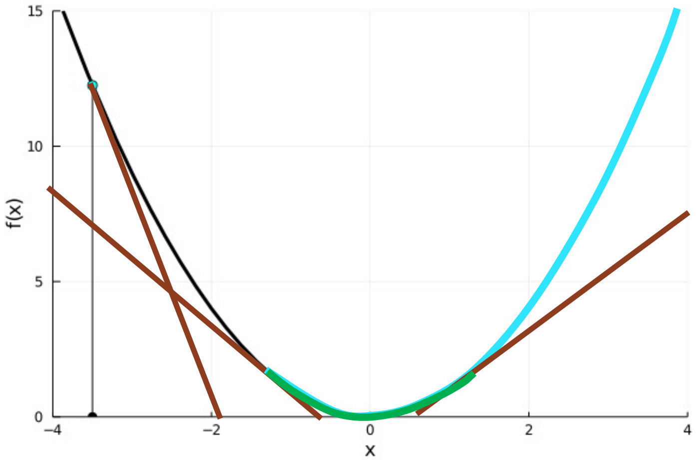
This is called the first Wolfe condition.

DEMO

Approximate Line Search

Small steps will always satisfy the sufficient decrease condition, but this does not guarantee convergence.

Curvature Condition: $\nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k+1)}) \geq \sigma \nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k)})$



Strong Curvature Condition:

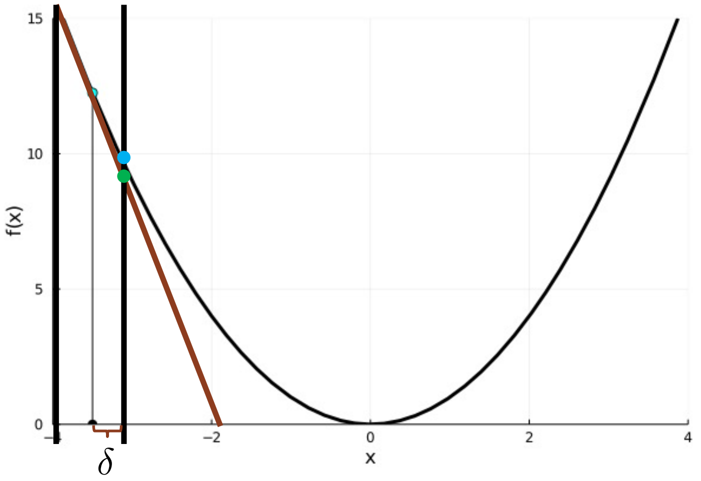
$$|\nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k+1)})| \geq -\sigma \nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k)})$$

This is called the second Wolfe condition.

DEMO

Trust Region Methods

Our local model (gradient information) can only be trusted in a region around our current point.



DEMO

1. Select a radius δ from the current point.

2. Optimize a local model of the function within that region.

$$\underset{\mathbf{x}'}{\text{minimize}}\ \hat{f}(\mathbf{x}') \longleftarrow$$

First or second order Taylor approximation

$$\text{subject to } \|\mathbf{x} - \mathbf{x}'\| \le \delta$$

3. Select the next radius δ based on local model's performance.

$$\eta = \frac{\text{actual improvement}}{\text{expected improvement}} = \frac{f(\mathbf{x}) - f(\mathbf{x}')}{f(\mathbf{x}) - \hat{f}(\mathbf{x}')}$$

When do we stop?

Maximum Iterations $k > k_{\max}$

Absolute Improvement $f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k+1)}) < \epsilon_a$

Relative Improvement $f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k+1)}) < \epsilon_r |f(\mathbf{x}^{(k)})|$

Gradient Magnitude $\|\nabla f(\mathbf{x}^{(k+1)})\| < \epsilon_g$

First-Order Methods

Gradient Descent

In gradient descent, we choose to move in the direction of steepest descent.

$$\mathbf{g}^{(k)} = \nabla f(\mathbf{x}^{(k)})$$

$$\mathbf{d}^{(k)} = -\frac{\mathbf{g}^{(k)}}{\|\mathbf{g}^{(k)}\|}$$

If we optimize the step size, descent directions for consecutive steps will be orthogonal to one another.

DEMO

Conjugate Gradient Descent

Borrows ideas from optimizing quadratic functions.

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + c$$

Directions are mutually conjugate with respect to **A**:

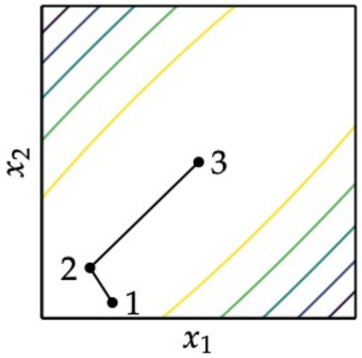
$$\mathbf{d}^{(i)\top} \mathbf{A} \mathbf{d}^{(j)} = 0 \text{ for all } i \neq j$$

Start with direction of steepest descent:

$$\mathbf{d}^{(1)} = -\frac{\mathbf{g}^{(1)}}{\|\mathbf{g}^{(1)}\|}$$

Next direction is a combination of next gradient and current descent direction:

$$\mathbf{d}^{(k+1)} = -\mathbf{g}^{(k+1)} + \beta^{(k)} \mathbf{d}^{(k)}$$



Conjugate Gradient Descent

$$\mathbf{d}^{(k+1)} = -\mathbf{g}^{(k+1)} + \beta^{(k)}\mathbf{d}^{(k)}$$

Knowing that we want directions to be mutually conjugate, we can determine $\beta^{(k)}$ for a known **A**.

$$\beta^{(k)} = \frac{\mathbf{g}^{(k+1)\top}\mathbf{A}\mathbf{d}^{(k)}}{\mathbf{d}^{(k)\top}\mathbf{A}\mathbf{d}^{(k)}}$$

What about for nonquadratic functions where we don't know **A**?

Fletcher-Reeves

$$\beta^{(k)} = \frac{\mathbf{g}^{(k)\top}\mathbf{g}^{(k)}}{\mathbf{g}^{(k-1)\top}\mathbf{g}^{(k-1)}}$$

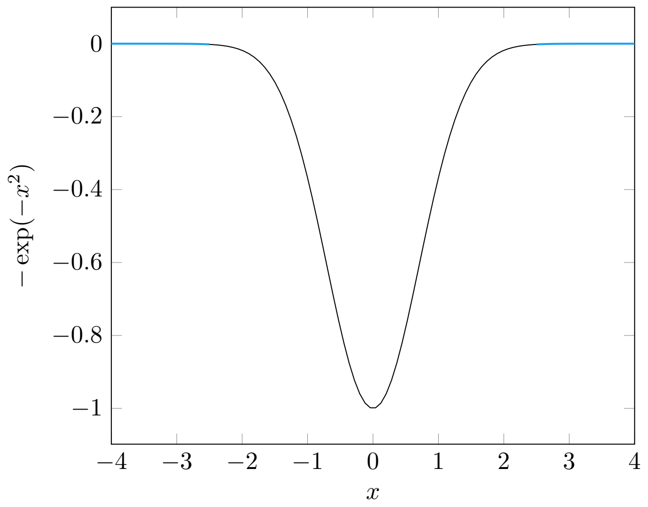
Polak-Ribière

$$\beta^{(k)} = \frac{\mathbf{g}^{(k)\top}(\mathbf{g}^{(k)} - \mathbf{g}^{(k-1)})}{\mathbf{g}^{(k-1)\top}\mathbf{g}^{(k-1)}}$$

DEMO

Momentum

Gradient descent moves slowly on flat surfaces.



DEMO

To fix this, we can incorporate the idea of momentum.

$$\mathbf{v}^{(k+1)} = \beta \mathbf{v}^{(k)} - \alpha \mathbf{g}^{(k)}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{v}^{(k+1)}$$

Ex: $\alpha = 1.0,\ \beta = 1.0, \mathbf{g}^{(k)} = 0.1$

$\mathbf{v}^{(k+1)}$

Nesterov Momentum

Momentum does not slow itself down enough at the bottom of a valley.

Momentum

$$\mathbf{v}^{(k+1)} = \beta \mathbf{v}^{(k)} - \alpha \boxed{\mathbf{g}^{(k)}}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{v}^{(k+1)}$$

Nesterov Momentum

$$\mathbf{v}^{(k+1)} = \beta \mathbf{v}^{(k)} - \alpha \boxed{\nabla f(\mathbf{x}^{(k)} + \beta \mathbf{v}^{(k)})}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{v}^{(k+1)}$$

DEMO

Adagrad

While the methods we have seen so far use the same learning rate in every dimension, adaptive subgradient method adapts the learning rate for each component of **x**.

$$x_i^{(k+1)} = x_i^{(k)} - \left(\frac{\alpha}{\epsilon + \sqrt{s_i^{(k)}}} \right) g_i^{(k)}$$

Sum of the gradients so far in direction i.

$$s_i^{(k)} = \sum_{j=1}^{k} (g_i^{(j)})^2$$

Dulls out parameters with consistently high gradients.

Issue: learning rate monotonically decreases

RMSProp

RMSProp extends Adagrad to fix the monotonically decreasing gradient problem.

$$x_i^{(k+1)} = x_i^{(k)} - \left(\frac{\alpha}{\epsilon + \sqrt{s_i^{(k)}}}\right) g_i^{(k)}$$

$$\mathbf{s}^{(k+1)} = \gamma \mathbf{s}^{(k)} + (1-\gamma)(\mathbf{g}^{(k)} \odot \mathbf{g}^{(k)})$$

Decaying average of squared gradients.

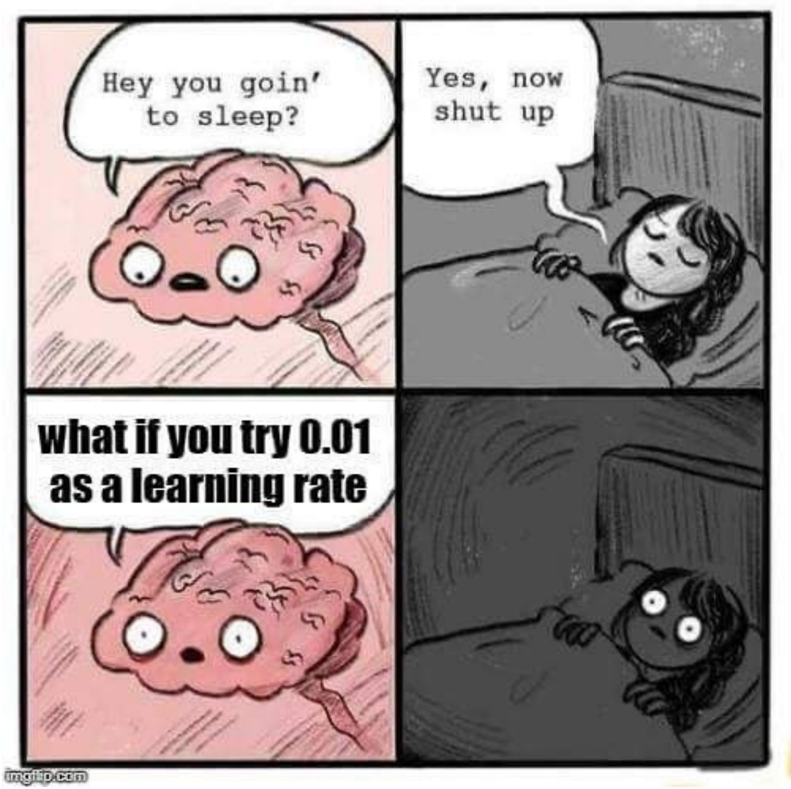
Adam

Combines elements from the previous algorithms.

Adaptive Moment Estimation

1. Biased decaying momentum
2. Biased decaying squared gradient
3. Corrected decaying momentum
4. Corrected decaying squared gradient

Hypergradient Descent



Accelerated descent methods tend to be extremely sensitive to the choice of learning rate.

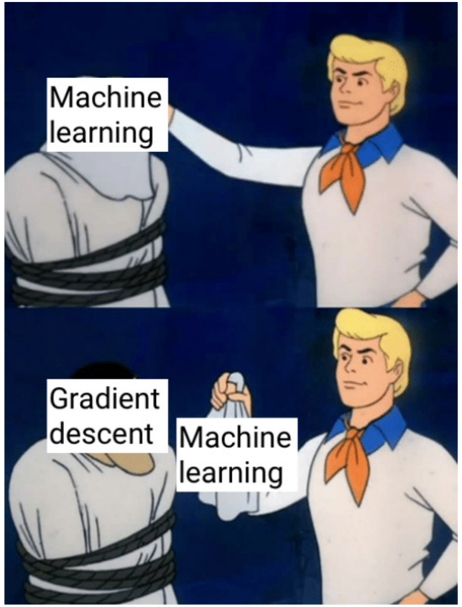
Hypergradient descent applies gradient descent to the learning rate.

Requires computing the derivative of the objective function with respect to the learning rate.

DEMO

Applications in the Real World

Machine Learning



Machine learning behind the scenes

Physics

Minimizing potential energy.

$$U = \sum_i m_i g y_i + \sum_i \frac{1}{2} k \left[(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2 \right]$$