10-315 Recitation

Review of Gradient Descent & Kernels

Misha

21 February 2019

Gradient Descent: Why do we need it?

Gradient Descent: Why do we need it?

- Many learning algorithms can be reduced to an optimization problem:
  - Logistic regression: $\min_{w,c} \sum_i \log\left(1 + \exp\left(-y_i\left(x_i^T w + c\right)\right)\right)$
  - Linear regression: $\min_{w,c} \sum_i \left\| x_i^T w + c - y_i \right\|_2^2$
  - Training deep neural networks
  - Many other methods (kernel methods, some clustering methods, …)

Gradient Descent: Why do we need it?

- Gradient descent is a simple, efficient local search heuristic for solving optimization problems

  - Most interesting problems can’t be solved analytically, or the analytic solution may be hard to compute.

  - If we start from a random point of a function and move in a descent direction we hope to decrease the objective value and reach a better solution.

Gradient Descent: How do we do it?

- Suppose we want to minimize a function $f(x)$ over $x \in \mathbb{R}^d$

Gradient Descent: How do we do it?

- Suppose we want to minimize a function $f(x)$ over $x \in \mathbb{R}^d$
  - Start at $x^{\{(0)\}} \in \mathbb{R}^d$
  - For $k = 0, \ldots, K$:
    - Compute gradient: $g = \nabla \mathrm{f}(\mathrm{x}^{\{(k)\}})$
    - Update position: $x^{\{(k+1)\}} = x^{\{(k)\}} - \eta g$
  - Here η is called the *step-size* or sometimes *learning rate*

Gradient Descent: When does it work?

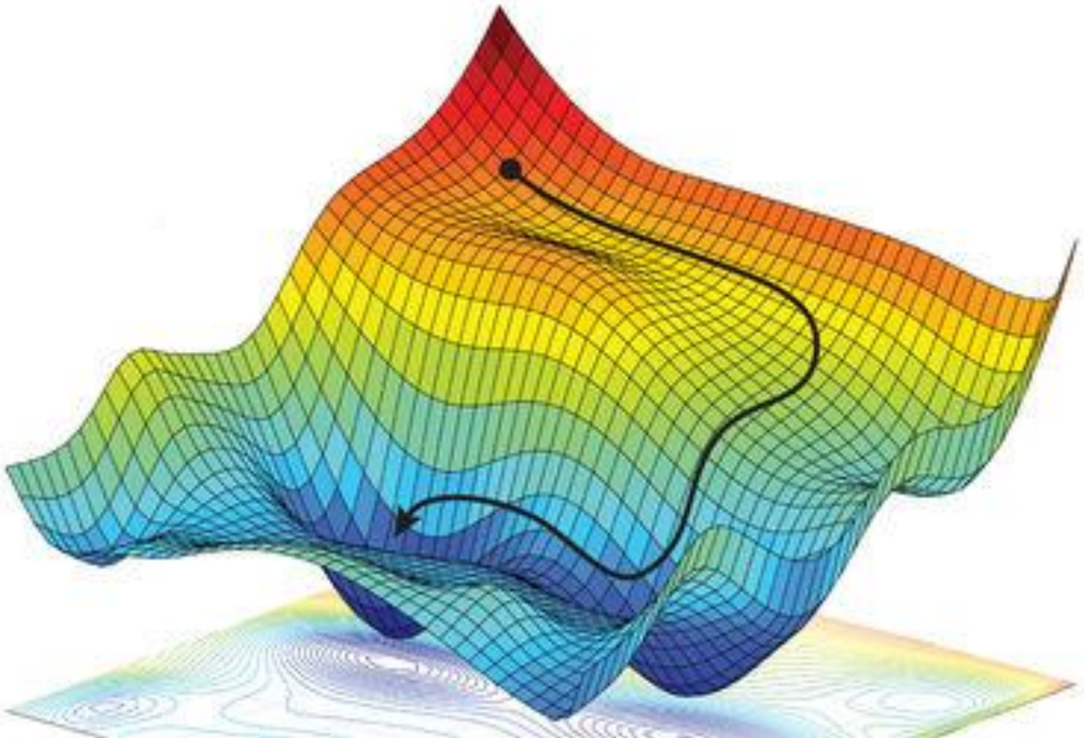


Image from Alexander Amini, Daniela Rus

Gradient Descent: When does it work?

- Functions must be “nice”
  - Almost-everywhere-differentiable, preferably smooth, to ensure that gradient steps decrease the objective value.
  - Convex, preferably strongly-convex, to ensure that repeated gradient steps converge to the global minimum.

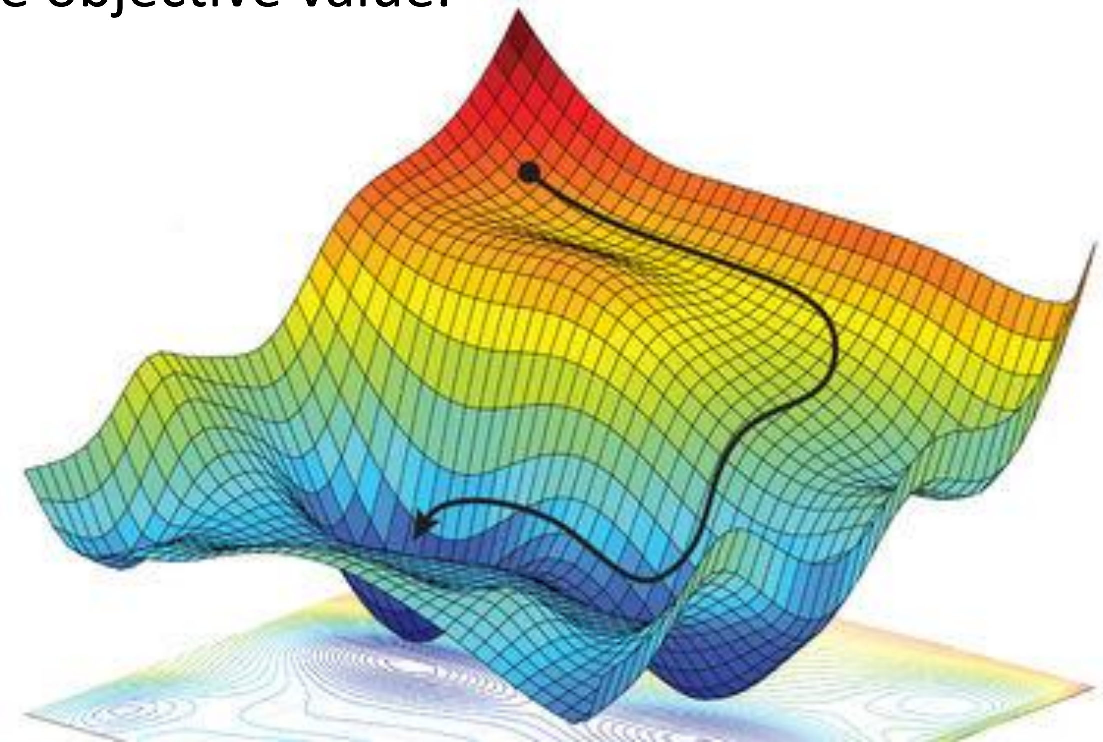


Image from Alexander Amini, Daniela Rus

Gradient Descent: Advanced

- We have been using fixed step size η. However, you can often reach the optimum faster if you decrease your step-size as you iterate.

- Is the gradient direction the best direction to go in?

  - Often, the case is no – you can use second derivative information to get very fast convergence on certain functions. The simplest method is Newton's method.

  - Often requires computing/storing the Hessian ∇f, which is very expensive.

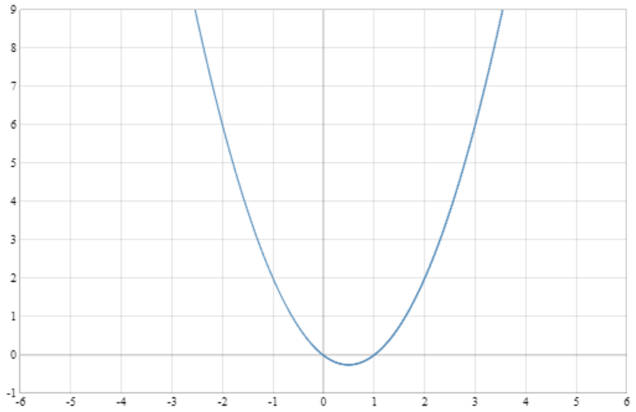
Gradient Descent: Looking ahead

- Deep learning:
  - Often too slow to compute full gradient at every step, so we compute the gradient over one or a few data-points – this is called *(mini-batch) stochastic gradient descent (SGD)*
- Constrained optimization:
  - Sometimes the set we are minimizing over is a subset of $\mathbb{R}^d$. In this case can use *projected gradient descent* – take a gradient step, then project back to the subset. This is used for example in certain formulations of support vector machines, which we will cover next.

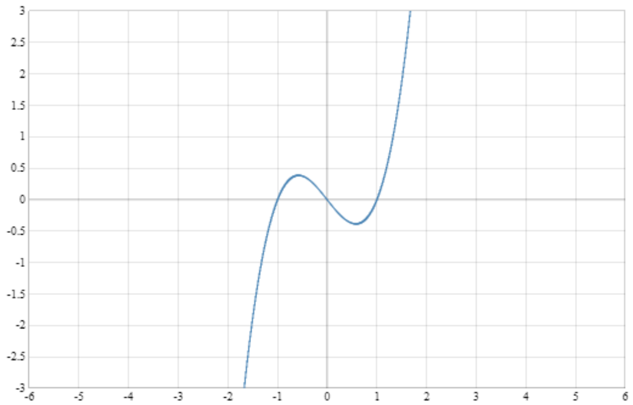
Gradient Descent: Review

- Which of the following are "nice" functions for gradient descent?

  - $f(x) = \frac{1}{2}x^2 - x$

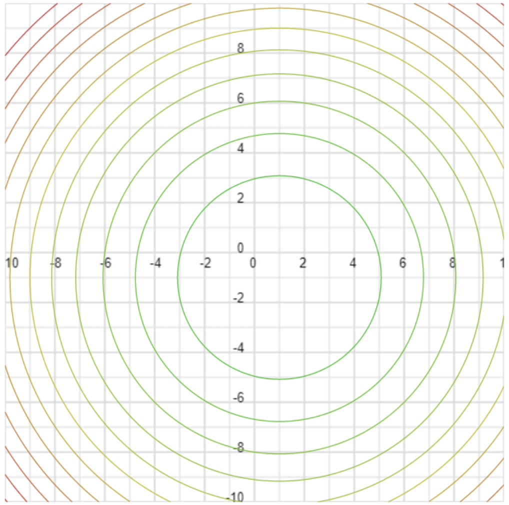


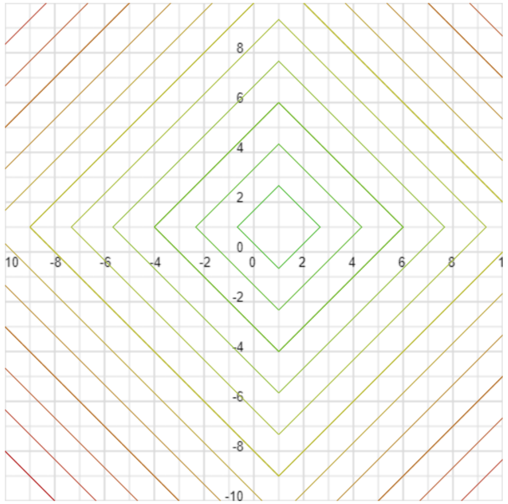
  - $f(x) = x^3 - x$



Gradient Descent: Review

- Which of the following are "nice" functions for gradient descent?





  - $f(x) = \|x - y\|_2^2$ for any $y \in \mathbb{R}^2$

  - $f(x) = \|x - y\|_1$ for any $y \in \mathbb{R}^2$

Gradient Descent: Review

- What changes if we want to solve $\max f(x)$ over $x \in \mathbb{R}^d$?

- Under what conditions can we ensure success?

Kernels: What are they?

Kernels: What are they?

- Intuitively, a kernel function is a generalized way of measuring the similarity between two points:

  - The regular Euclidean inner product computes $x \cdot y$ for $x, y \in \mathbb{R}^d$

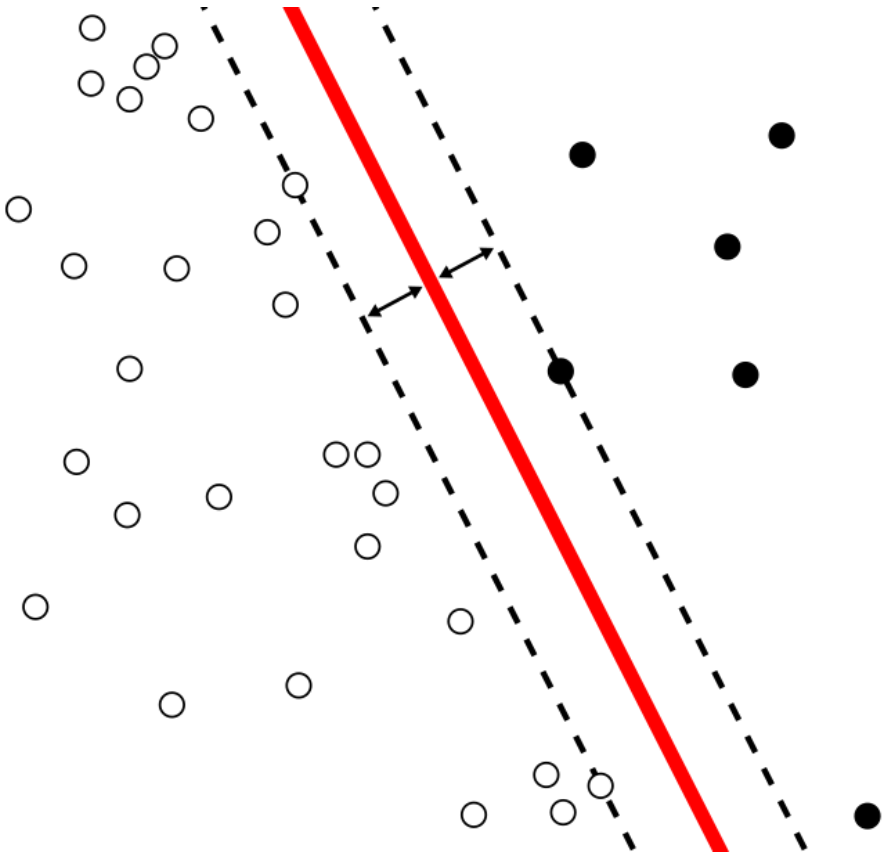
  - A kernel function K computes $K(x, y) = \phi(x) \cdot \phi(y)$ for some implicit mapping ϕ on the space to which x, y belong (note this no longer has to be Euclidean space).

Kernels: Why are they useful?

- Simple algorithms depend on how nicely separated data is in the feature space:

  - Perceptron, support vector machines – depend on margin between classes.

  - Clustering algorithms – depend on clusters being well-separated.



Kernels: Why are they useful?

- However, interesting data is often structured in a way such these properties don't hold in the input space.
- The input space where these properties hold might be hard to compute.

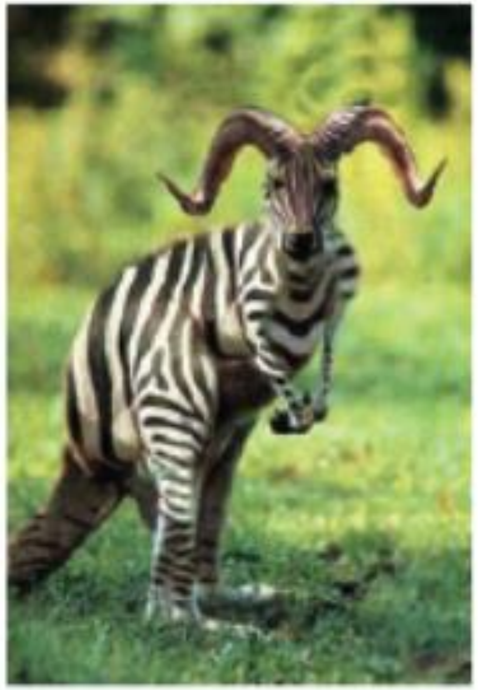
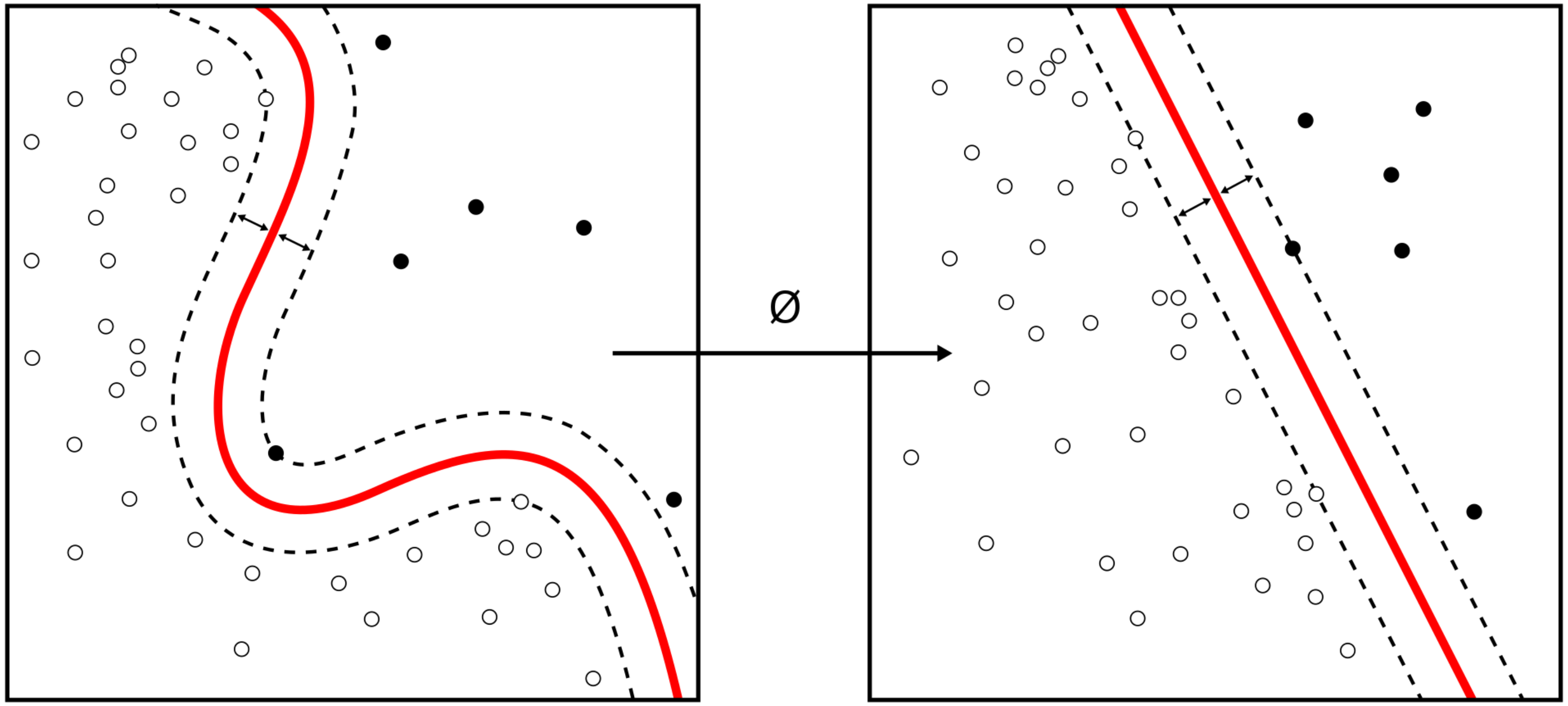


Image from Jia Deng

Kernels: Why are they useful?

- Kernels allow us to have algorithms that with guarantees depending on separability in a different (implicit) space – the space mapped to by the function ϕ – but without having to compute this space.



Kernels: Why can we use them?

Kernels: Why can we use them?

- Many algorithms can be *kernelized* – made to depend only on the inner product between the input points:
    - Perceptron
    - Support vector machines
    - Certain clustering algorithms
    - …

Kernels: Perceptron

- Recall the perceptron algorithm:
  - Start with $w = 0$
  - For $t = 0, \ldots$
    - Get example x_t, y_t
    - Predict $w_t \cdot x_t$
    - Mistake on positive: $w_{t+1} = w_t + x_t$
    - Mistake on negative: $w_{t+1} = w_t - x_t$

Kernels: Perceptron

- Recall the perceptron algorithm:
  - For $t = 0, \dots$
    - Get example x_t, y_t
    - Predict $a_{i_1} K(x_{i_1}, x_t) + \cdots + a_{i_{t-1}} K(x_{i_{t-1}}, x_t)$
    - Mistake on positive: set $a_{i_t} = 1$, store $x_{i_t} = x_t$
    - Mistake on negative: set $a_{i_t} = -1$, store $x_{i_t} = x_t$

Kernels: Advanced

- Kernels allow us to compute decision boundaries over any classes of objects, so long as we can define a kernel function over them:
  - String kernels – used in comp. linguistics to classify text and in comp. biology to classify molecules
  - Gaussian kernels have an implicit mapping to infinite-dimensional space.

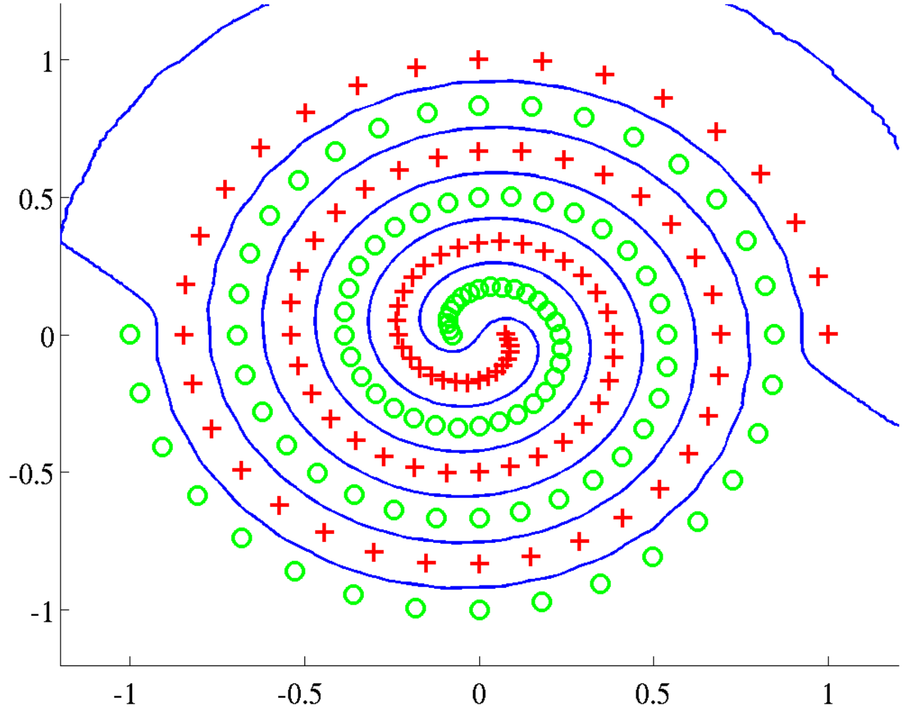


Image from Neil Lawrence

Kernels: Review

- How many mistakes will the kernelized perceptron make (assume the data has margin γ and radius R in the ϕ space)?

Kernels: Review

- How can we show that a function $K(x, y)$ is a kernel?