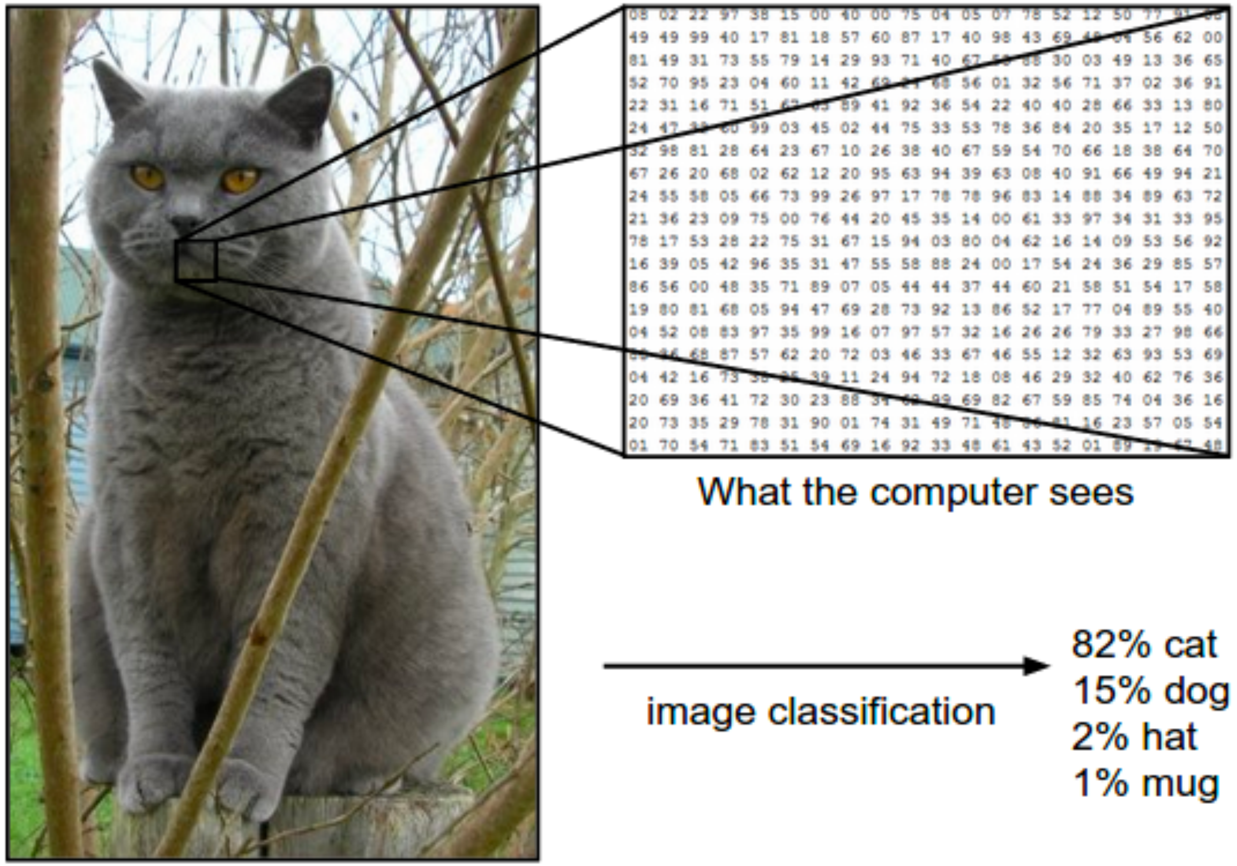
Stochastic Gradient Descent

Spring 2020

Image Classification



Linear model

- Score function
  - Maps raw data to class scores
- Loss function
  - Measures how well predicted classes agree with ground truth labels
    - Multiclass Support Vector Machine loss (SVM loss)
    - Softmax classifier (cross-entropy loss)
- Learning
  - Find parameters of score function that minimize loss function
    - Multiclass Support Vector Machine loss (SVM loss)

Recall: Linear model with SVM loss

- Score function
  - Maps raw data to class scores

$$f(x_i, W) = Wx_i$$

- Loss function
  - Measures how well predicted classes agree with ground truth labels

$$L = \frac{1}{N}\sum_i \sum_{j \neq y_i} [\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta)] + \lambda \sum_k \sum_l W_{k,l}^2$$

Today

- Learning model parameters with Stochastic Gradient Descent that minimize loss

- Later
  - Different score functions: deep networks
  - Same loss functions and learning algorithm

Outline

- Visualizing the loss function
- Optimization
  - Random search
  - Random local search
  - Gradient descent
    - Mini-batch gradient descent

Visualizing SVM loss function

- Difficult to visualize fully
  - CIFAR-10 a linear classifier weight matrix is of size [10 x 3073] for a total of 30,730 parameters

- Can gain intuition by visualizing along rays (1 dimension) or planes (2 dimensions)

Visualizing in 1-D

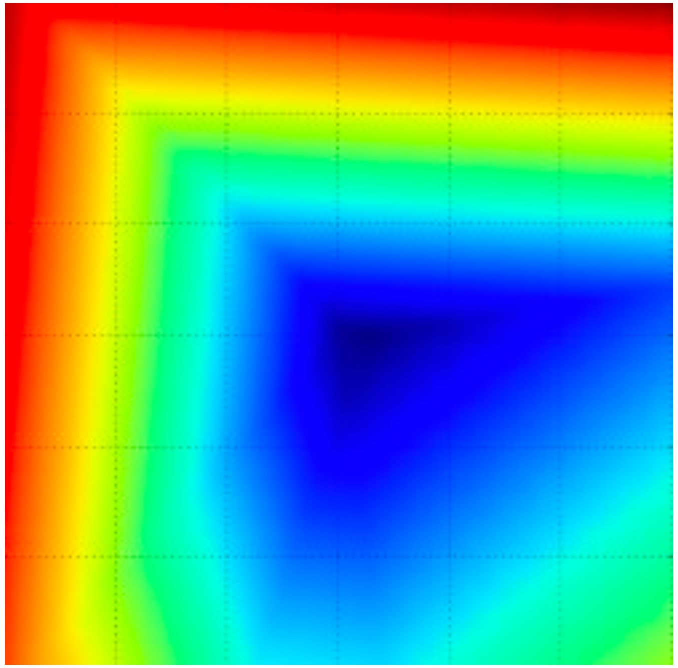
- Generate random weight matrix W
- Generate random direction W_1
- Compute loss along this direction $L(W + aW_1)$

Loss for single example

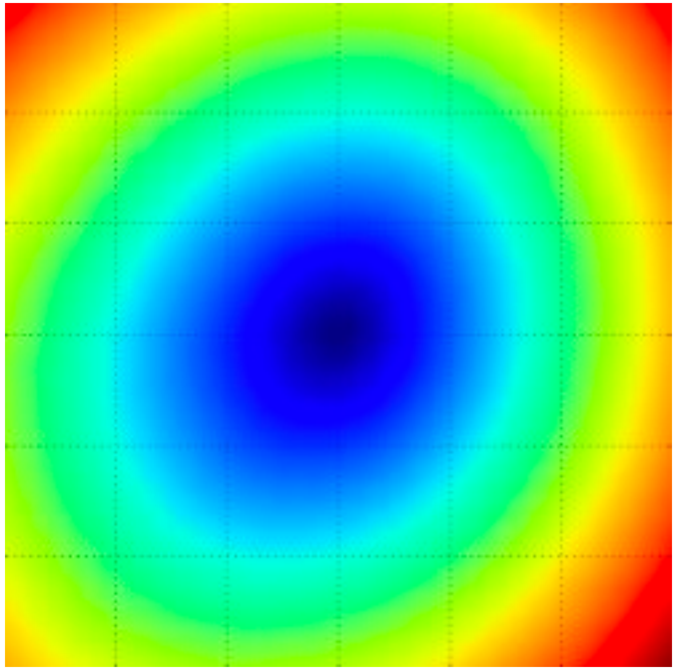
Where is the minima?

Visualizing in 2-D

- Compute loss along plane $L(W + aW_1 + bW_2)$



Loss for single example



Average loss for 100 examples
(convex function)

How do we find weights that minimize loss?

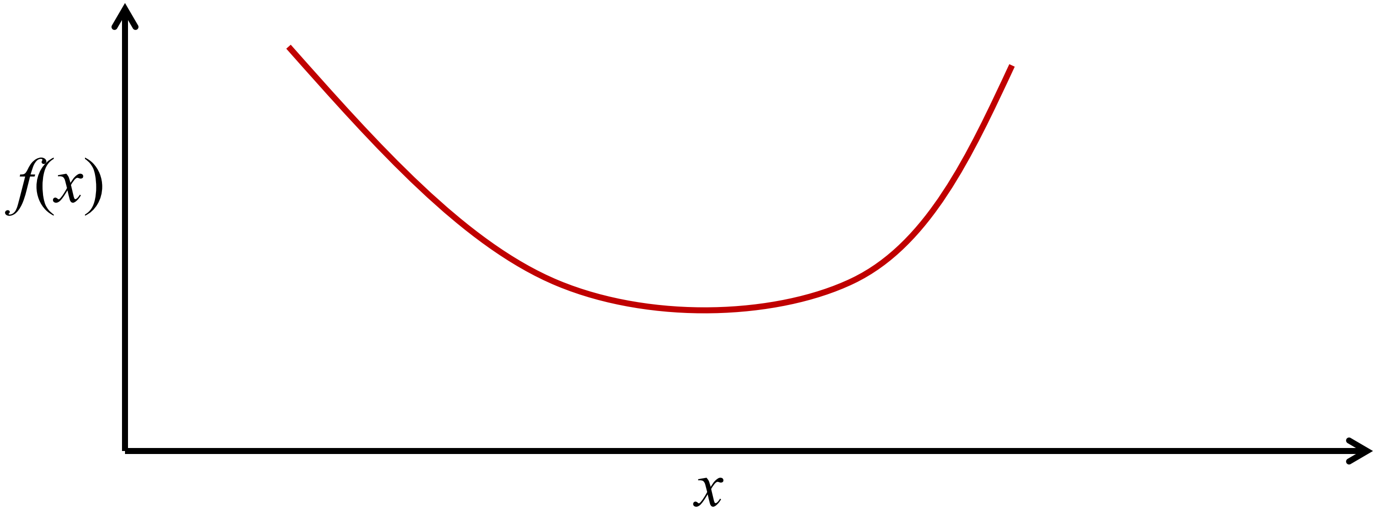
- Random search
  - Try many random weight matrices and pick the best one
  - Performance: poor

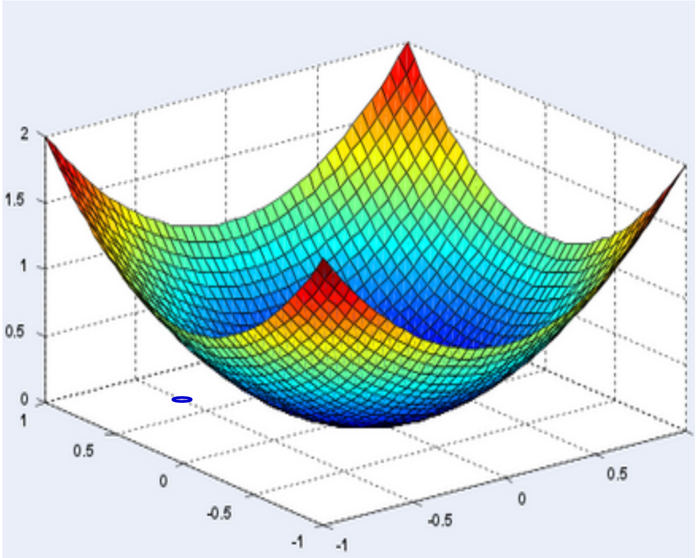
- Random local search
  - Start with random weight matrix
  - Try many local perturbations, pick the best one, and iterate
  - Performance: better but still quite poor

- Useful idea: iterative refinement of weight matrix

Optimization basics

The problem of optimization





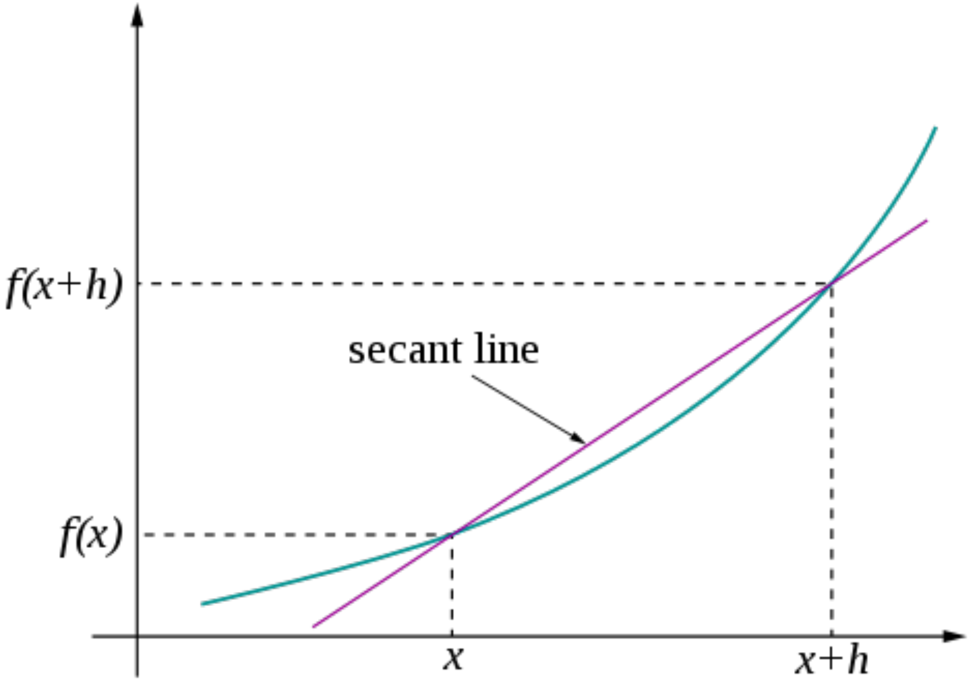
Find the value of $\boldsymbol{x}$ where $\mathbf{f}(\boldsymbol{x})$ is minimum

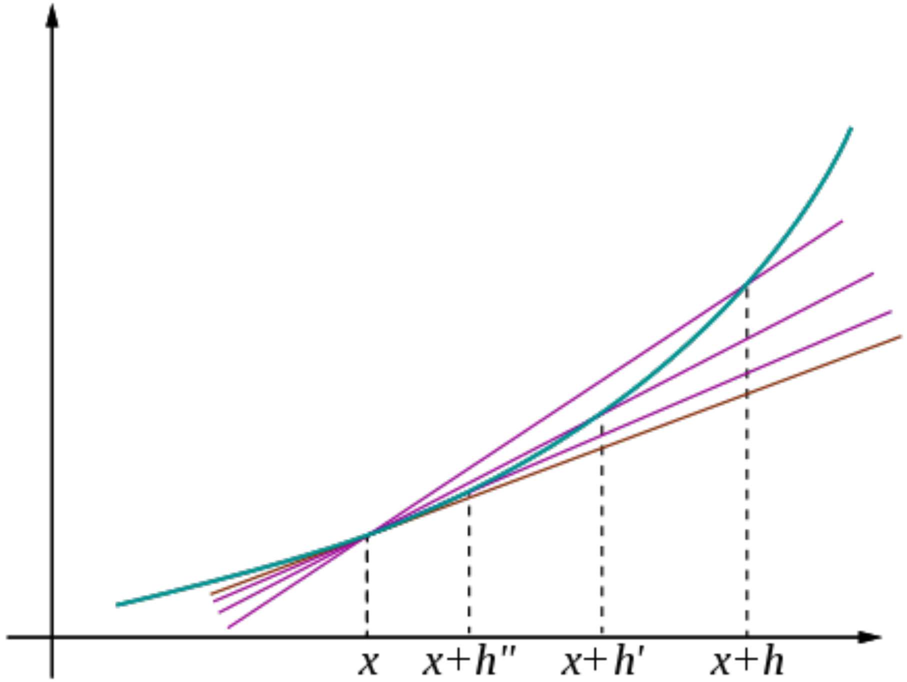
Our setting: $\boldsymbol{x}$ represents weights, $\mathbf{f}(\boldsymbol{x})$ represents loss function

In two stages

- Function of single variable
- Function of multiple variables

Derivative of a function of single variable





$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

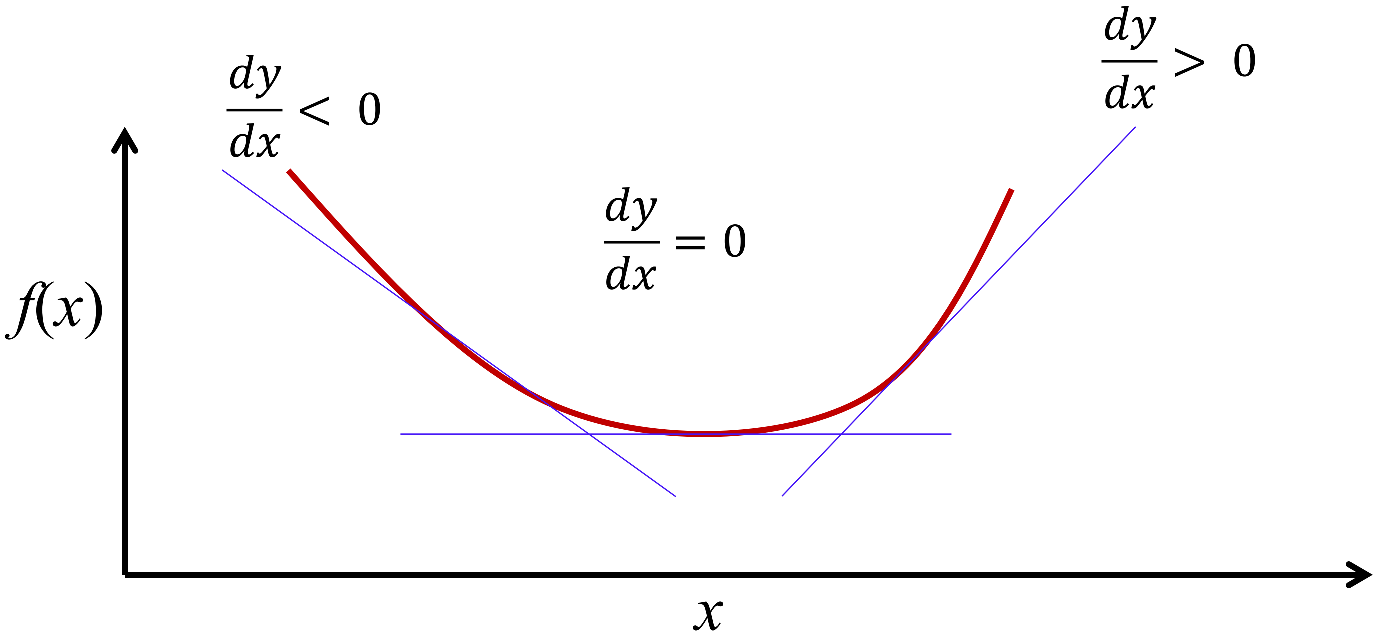
Derivatives

$$\frac{d}{dx}(x^2) = 2x$$

$$\frac{d}{dx}(e^x) = e^x$$

$$\frac{d}{dx}(ln\ x) = \frac{1}{x}\ if\, x > 0$$

Finding minima

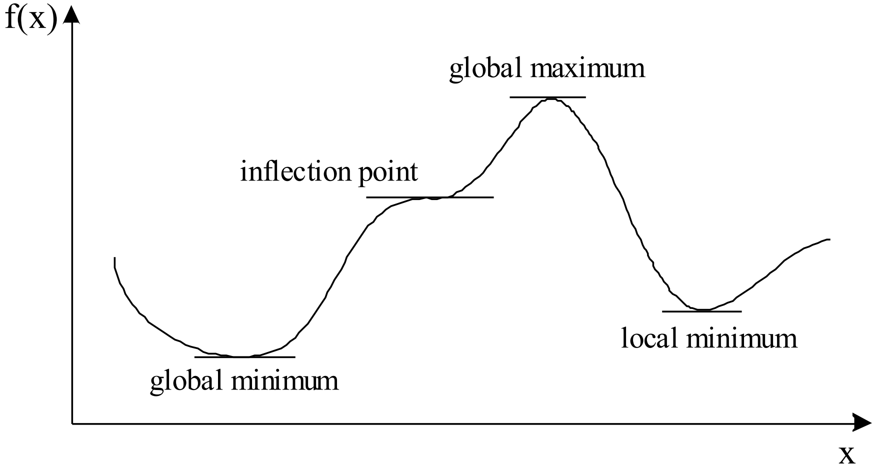


Increase x if derivative negative, decrease if positive

i.e., take step in direction opposite to sign of derivative

(key idea of gradient descent)

Doesn’t always work



- Theoretical and empirical evidence that gradient descent works quite well for deep networks

In two stages

- Function of single variable
- Function of multiple variables

Partial derivatives

The **partial derivative** of an n-ary function $f(x_1,...,x_n)$ in the direction x_i at the point $(a_1,...,a_n)$ is defined to be:

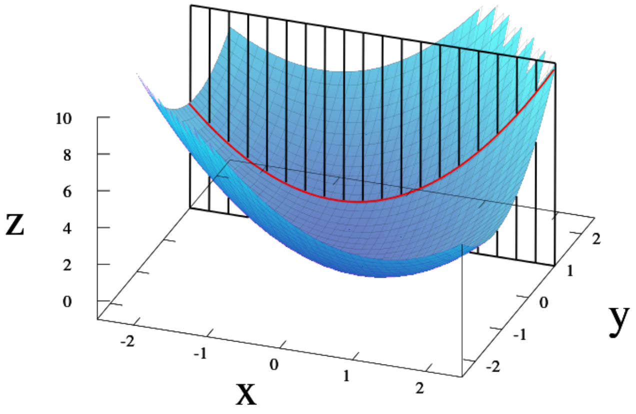
$$\frac{\partial f}{\partial x_i}(a_1,\ldots,a_n) = \lim_{h\to 0}\frac{f(a_1,\ldots,a_i+h,\ldots,a_n) - f(a_1,\ldots,a_i,\ldots,a_n)}{h}.$$

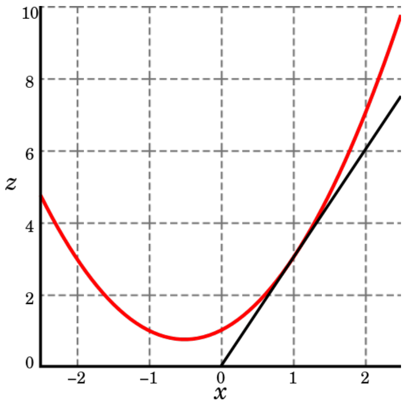
Partial derivative example

$$z = f(x, y) = x^2 + xy + y^2.$$

$$\frac{\partial z}{\partial x} = 2x + y.$$

At (1, 1), the slope is 3



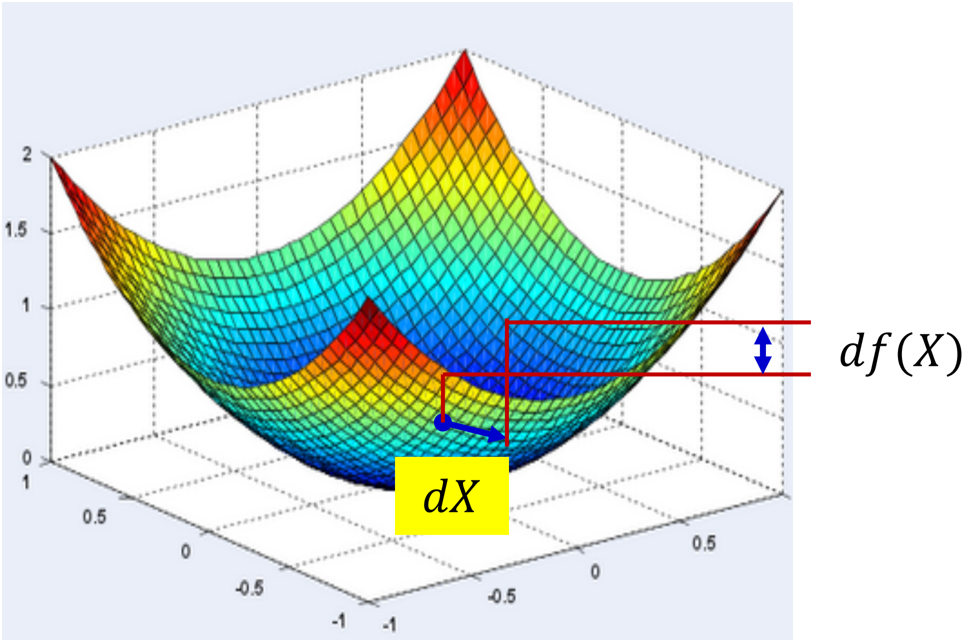


The gradient of a scalar function

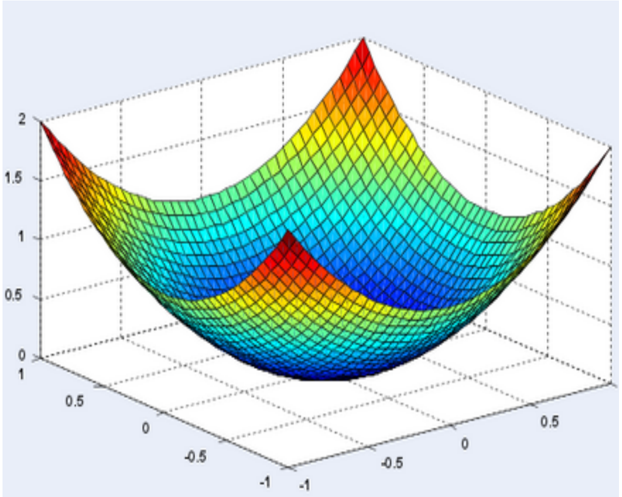
- The *gradient* $\nabla f(X)$ of a scalar function $f(X)$ of a multi-variate input X is a multiplicative factor that gives us the change in $f(X)$ for tiny variations in X

$$df(X) = \nabla f(X) dX$$

$$\nabla f(X) = df(X)/dX$$



Gradients of scalar functions with multi-variate inputs



- Consider $f(X) = f(x_1, x_2, \ldots, x_n)$

- $\nabla f(X) = \left[\frac{\partial f(X)}{\partial x_1} \quad \frac{\partial f(X)}{\partial x_2} \quad \cdots \quad \frac{\partial f(X)}{\partial x_n} \right]$

Computing gradients analytically

$$f(x, y) = x + y \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$

Computing gradients analytically

$$f(x,y) = xy \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}] = [y, x]$$

Derivatives measure sensitivity

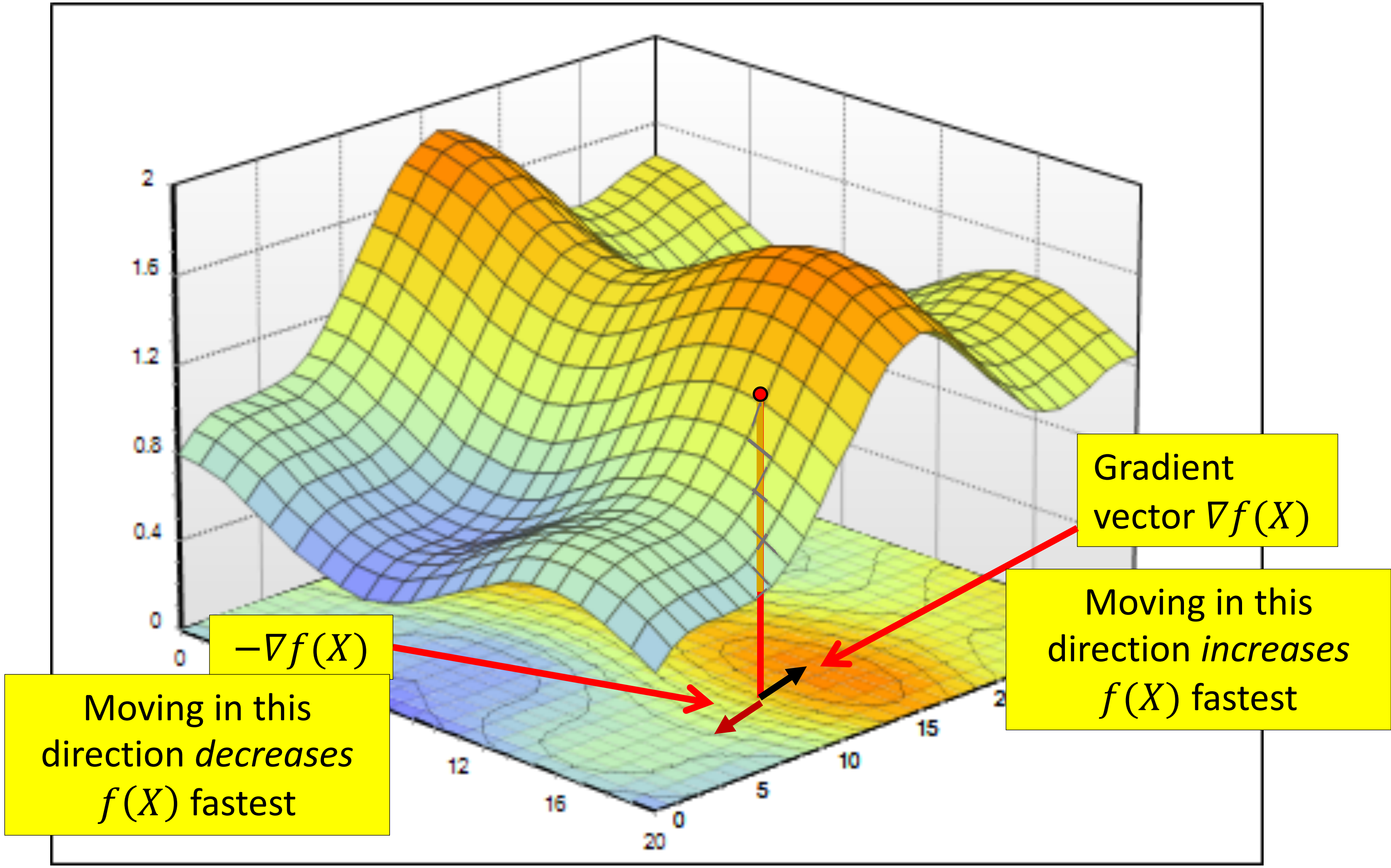
$$f(x, y) = xy$$

$$x = 4, y = -3 \qquad f(x, y) = -12 \qquad \frac{\partial f}{\partial x} = -3$$

If we were to increase x by a tiny amount, the effect on the whole expression would be to decrease it (due to the negative sign), and by three times that amount.

Finding minima

Take step in direction opposite to sign of gradient



Gradient descent algorithm

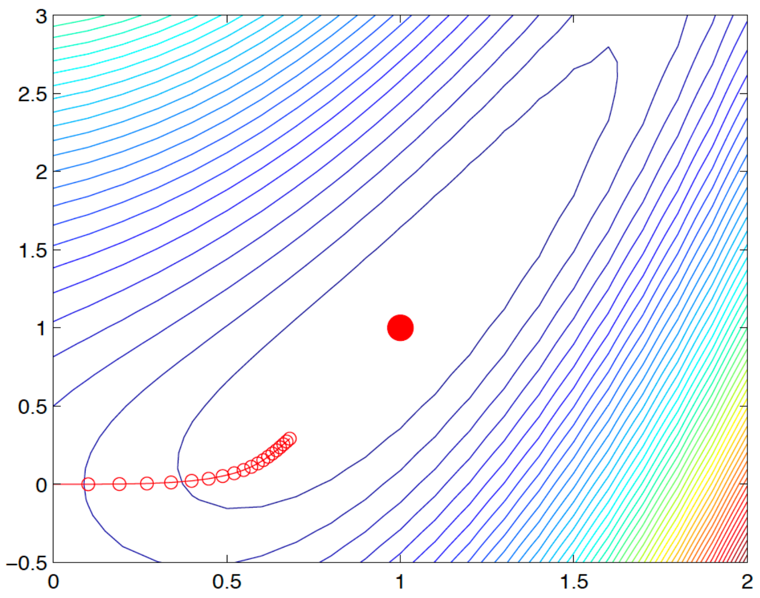
- Initialize:
  - x^0
  - $k = 0$
- Do
  - $x^{k+1} = x^k - \eta^k \nabla f(x^k)$
  - $k = k + 1$
- Until $\left|f(x^k) - f(x^{k-1})\right| \leq \varepsilon$

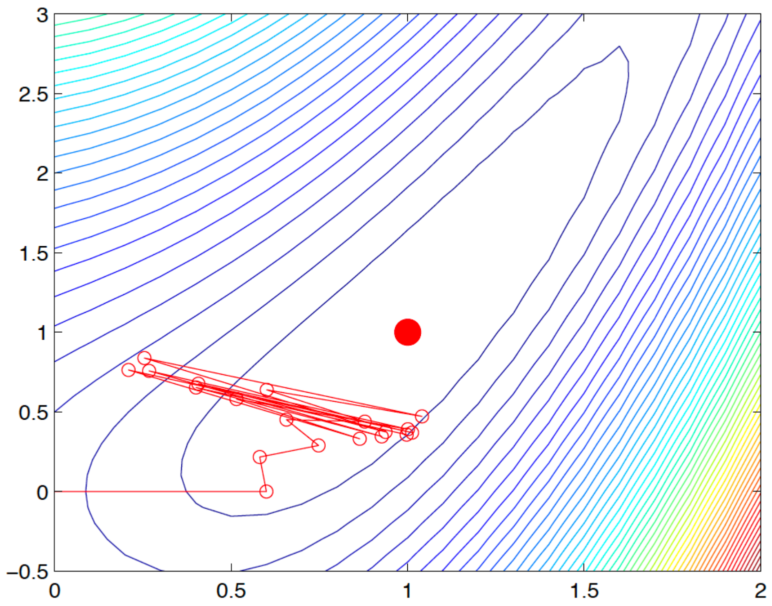
Average gradient across all training examples

$$f(x^k) = \frac{1}{N} \Sigma_{i=1}^{N} f_i(x^k)$$

$$\nabla f(x^k) = \frac{1}{N} \Sigma_{i=1}^{N} \nabla f_i(x^k)$$

Step size affects convergence of gradient descent





Murphy, Machine Learning, Fig 8.2

Gradient descent algorithm

- Initialize:
  - x^0
  - $k = 0$
- Do
  - $x^{k+1} = x^k - \eta^k \nabla f(x^k)$
  - $k = k + 1$
- Until $\left|f(x^k) - f(x^{k-1})\right| \leq \varepsilon$

Average gradient across all training examples

Challenge: Not scalable for very large data sets

Challenge to discuss later: How to choose step size?

Mini-batch gradient descent

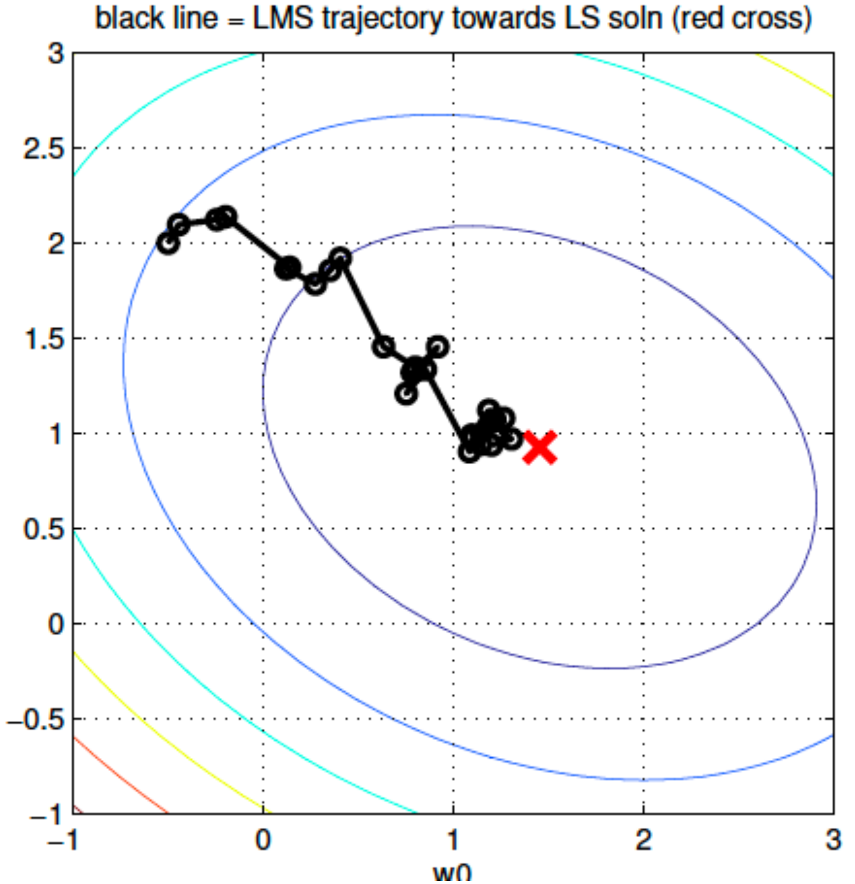
- Initialize:
  - x^0
  - $k = 0$
- Do
  - $x^{k+1} = x^k - \eta^k \nabla f(x^k)$
  - $k = k + 1$
- Until $\left|f(x^k) - f(x^{k-1})\right| \leq \varepsilon$

Average gradient over small batches of training examples (e.g., sample of 256 examples)

Faster convergence

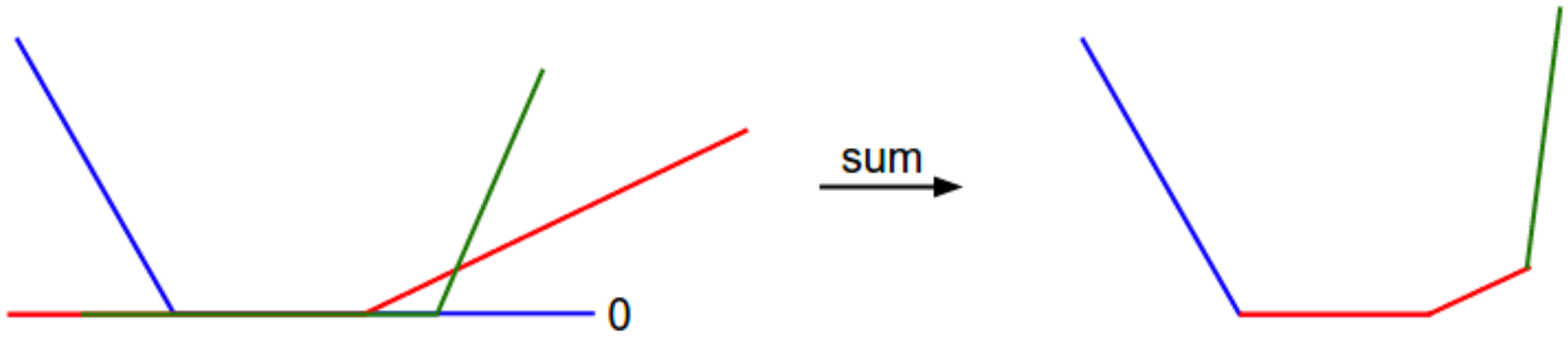
Special case: Stochastic or online gradient descent → use single training example in each update step

Stochastic gradient descent convergence



Murphy, Machine Learning, Fig 8.8

SVM loss visualization



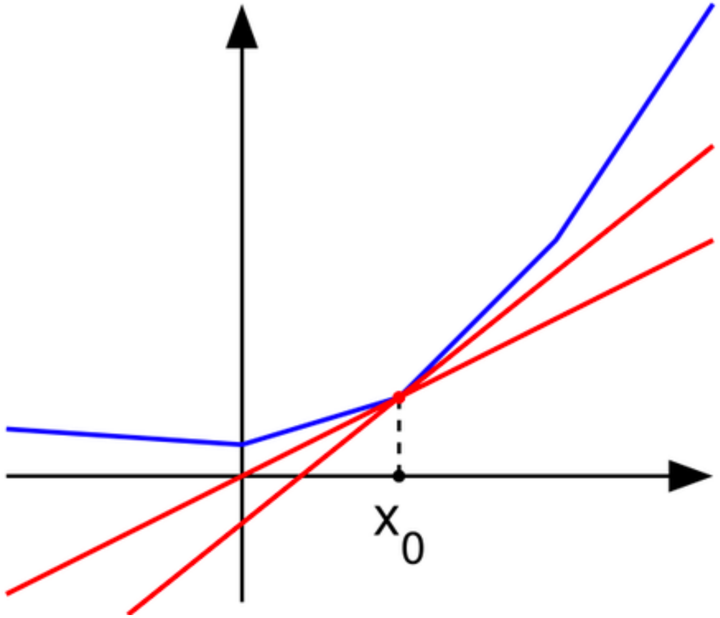
Challenge: Gradient does not exist

Computing subgradients analytically

The set of subderivatives at x_0 for a convex function is a nonempty closed interval [*a*, *b*], where *a* and *b* are the one-sided limits:

$$a = \lim_{x \to x_0^-} \frac{f(x) - f(x_0)}{x - x_0}$$

$$b = \lim_{x \to x_0^+} \frac{f(x) - f(x_0)}{x - x_0}$$



Computing subgradients analytically

$$f(x, y) = \max(x, y) \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = \mathcal{I}(x >= y) \qquad \frac{\partial f}{\partial y} = \mathcal{I}(y >= x)$$

The (sub)gradient is 1 on the input that is larger and 0 on the other input

Subgradient of SVM loss

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$

$$\nabla_{w_{y_i}} L_i = - \left(\sum_{j \neq y_i} \mathcal{I}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right) x_i$$

Number of classes that didn't meet the desired margin

$$\nabla_{w_j} L_i = \mathcal{I}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) x_i$$

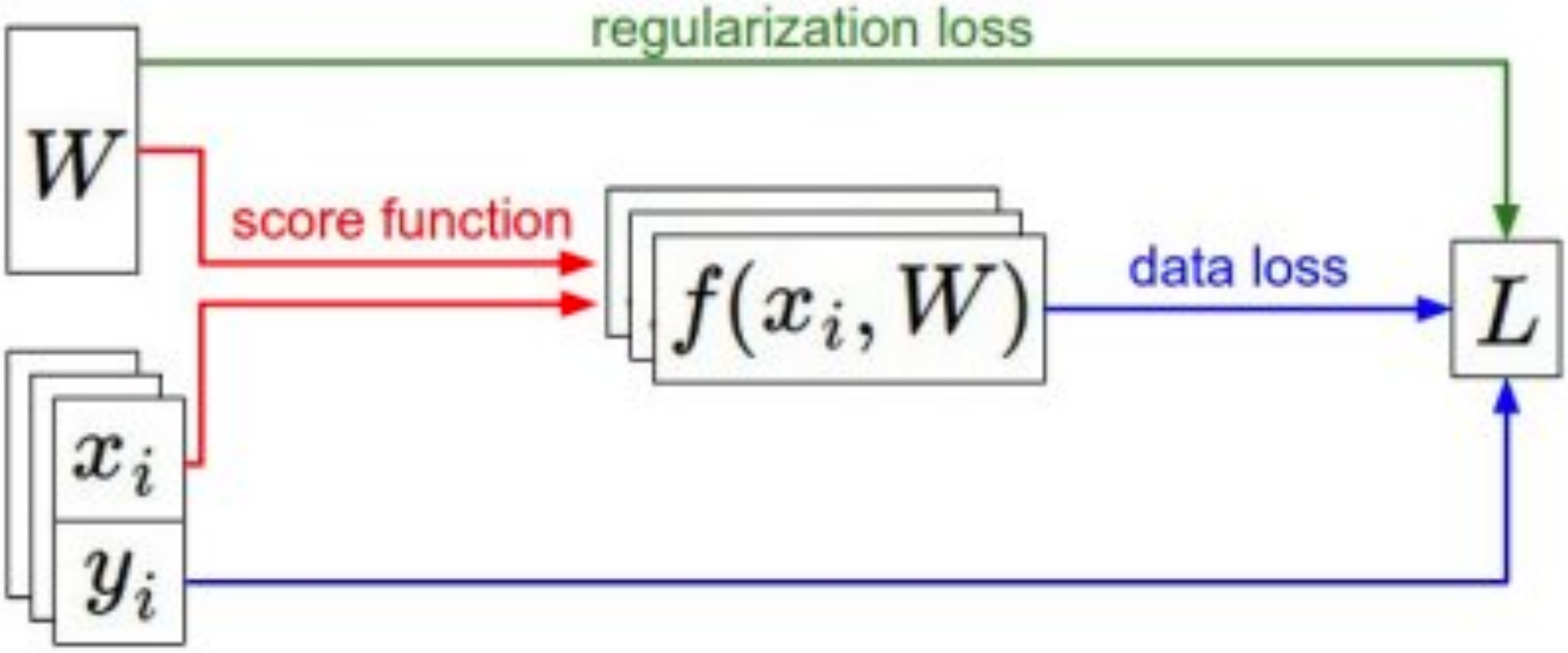
j-th class didn't meet the desired margin

Review derivatives

- Please review rules for computing derivatives and partial derivatives of functions, including the chain rule
  - https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives

- You will need to use them in HW1!

Summary



Acknowledgment

- Based on material from
  - Stanford CS231n http://cs231n.github.io/
  - CMU 11-785 Course
  - Spring 2019 Course