UNIVERSITY AT ALBANY
STATE UNIVERSITY OF NEW YORK

CSI 436/536 (Spring 2026)
Machine Learning
Lecture 9: Linear Regression

Chong Liu
Department of Computer Science

Feb 23, 2026

Recap: Loss and Gradient Descent

- 0-1 loss in linear classifier
  - Hard to optimize!

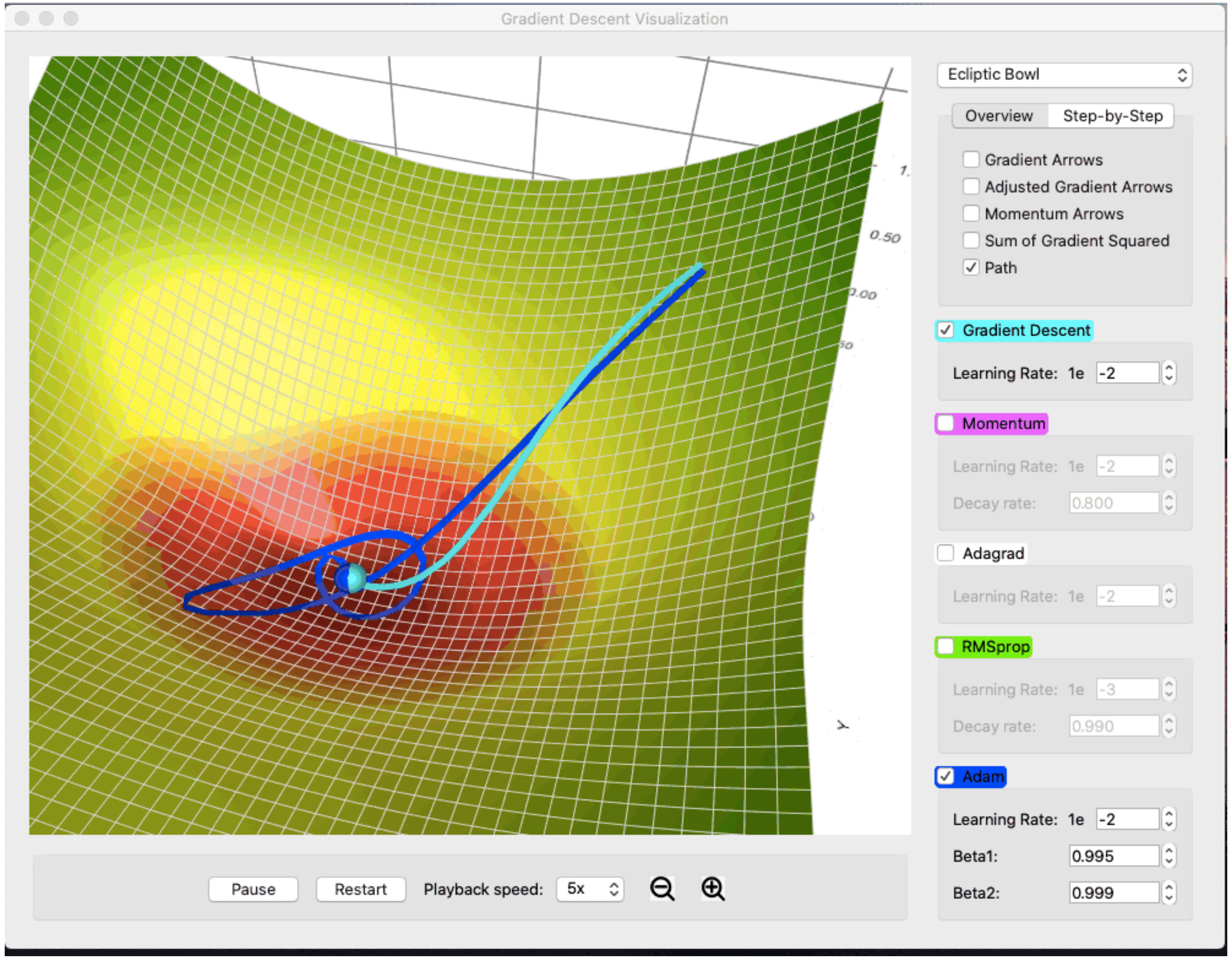
$$\min_{w \in \mathbb{R}^d} \mathrm{Error}(w) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(h_w(x_i) \neq y_i)$$

- Surrogate loss
  - Easy to optimize (continuous, convex, differentiable)
  - Examples: squared loss, logistic loss, exponential loss, ...
- Gradient Descent (GD)

$$\theta_{t+1} = \theta_t - \eta_t \nabla f(\theta_t)$$

Recap: Gradient Descent Demo in 2-D

- An excellent demo tool:
  - https://github.com/lilipads/gradient_descent_viz



Today

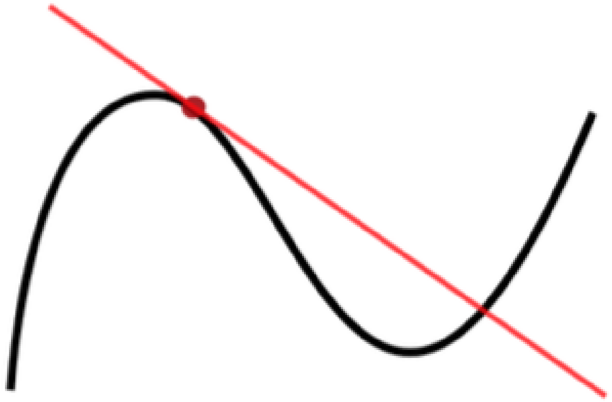
- Stochastic Gradient Descent (SGD)

- Linear regression

- Use SGD to solve linear regression problem!

Recap: What is the “gradient” of a multivariate function?

- We use differentiation to compute derivatives of functions in Calculus.



  - Example $f(x, y) = 3x^2 + xy$, $\frac{\partial f(x,y)}{\partial x} = 6x + y$, $\frac{\partial f(x,y)}{\partial y} = x$.
- In many machine learning problems, the objective involves a function that takes a vector of variables as input, e.g., $f(w) = w^T x$ where $w \in \mathbb{R}^d$.
- How to take derivatives on such functions?

Gradient of logistic loss for learning a linear classifier

- The function to minimize is

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i \cdot x_i^T w))$$

- In-class exercise: Calculate the gradient of loss function w.r.t *w*

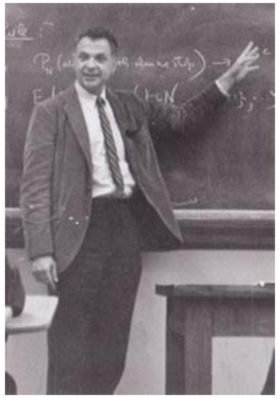
$$\nabla f(w) = \frac{1}{n} \sum_{i=1}^{n} \frac{\exp(-y_i \cdot x_i^T w)}{1 + \exp(-y_i \cdot x_i^T w)} (-y_i x_i)$$

Hint:
- Apply the chain rule.
- d log(x) / dx = 1/x
- d exp(x) / dx = exp(x)

Drawback: Gradient Descent (GD) uses all data to do one update.

Key question: Is there an efficient way to optimize loss function?

Stochastic Gradient Descent (Robbins-Monro 1951)



Herbert Robbins
1915 - 2001

- Gradient descent

$$\theta_{t+1} = \theta_t - \eta_t \nabla f(\theta_t)$$

- Stochastic gradient descent
  - Using a stochastic approximation of the gradient:

$$\theta_{t+1} = \theta_t - \eta_t \hat{\nabla} f(\theta_t)$$

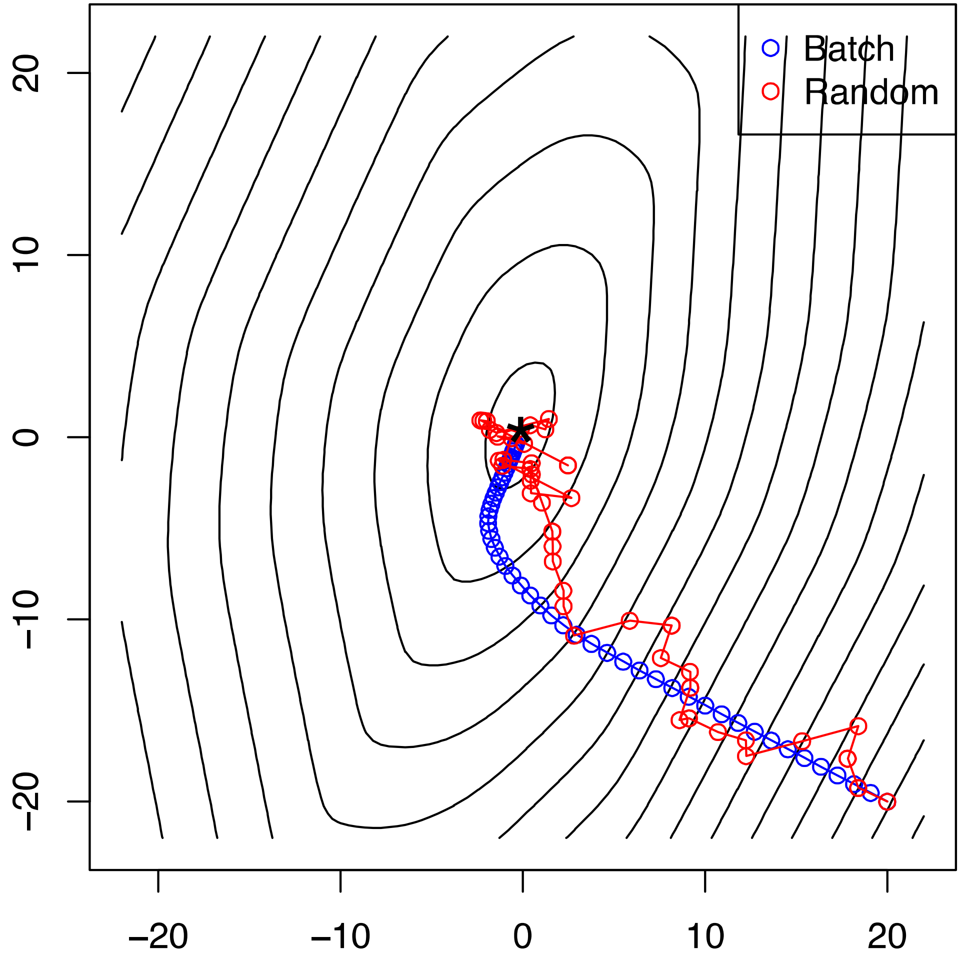
A natural choice of SGD in machine learning

- Recall that

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell(\theta, (x_i, y_i))$$

- SGD samples a data point i uniformly at random while GD uses all data!
  - Use $\nabla_\theta \ell(\theta, (x_i, y_i))$

Illustration of GD vs SGD



Time complexity:

GD: $O(nd * \text{n\_iterations})$
SGD: $O(d * \text{n\_iterations})$

Intuition of the SGD algorithm on the "Spam Filter" example

$$\nabla \ell(w,(x_i,y_i)) = \underbrace{\frac{\exp(-y_i \cdot x_i^T w)}{1+\exp(-y_i \cdot x_i^T w)}}_{\text{Scalar}>0}\left(\boxed{-y_i x_i}\right)$$

Scalar > 0:
≈ 0 if the prediction is correct (no update)
≈ 1 otherwise (update)

Vector of dimension d: provides the direction of the gradient

Given an email example [1, -1, 0.0375, 80] where 0.0375 is proportion of misspelled words.
Its y = 1 means spam.

How will the SGD update change the weight vector?

$$\theta_{t+1} = \theta_t - \eta_t \hat{\nabla} f(\theta_t)$$

If you make a mistake, move the weight towards the direction such that you will be less likely to make the same mistake in the future.

How to choose the step sizes / learning rates?

- In practice:
  - Use cross-validation on validation dataset.
  - Fixed learning rate for SGD is usually fine.
  - If it diverges, decrease the learning rate.

The power of SGD

- Extremely general:
  - Specify an end-to-end differentiable score function
    - E.g., a huge neural network.
- Extremely simple:
  - A few lines of code
- Extremely scalable
  - Just a few pass of the data, no need to store the data

Checkpoint

- Learning a linear classifier:
  - It’s hard to directly optimize 0-1 loss
  - Find a surrogate loss
    - Continuous
    - Convex
    - Differentiable

- Gradient descent
  - Calculating gradient / making sense of gradient
  - Improving GD with Stochastic Gradient Descent

Linear regression example: Housing price

- Case study:
  - 8 features:

```
- MedInc        median income in block group
- HouseAge      median house age in block group
- AveRooms      average number of rooms per household
- AveBedrms     average number of bedrooms per household
- Population    block group population
- AveOccup      average number of household members
- Latitude      block group latitude
- Longitude     block group longitude
```

  - 1 label: house price

- Discussion: What are they?
  - Feature space (input set)
  - Label space (output set)
  - Linear model
  - Performance metric
  - Loss function

Regression for different problems

- Prediction problem
  - How well can one predict label y?
    - In housing price example: how well can one predict price given a house?

- Estimation / inference problem
  - How well can one estimate the true function?
    - In housing price example: how well can one learn the price generating function?

Two problems of supervised learning

	Classification		Regression
	Binary classification	Multi-class classification	
Feature space	$\mathbb{R}^d$	$\mathbb{R}^d$	$\mathbb{R}^d$
Label space	{-1, 1}	{1, 2, 3, …, K}	$\mathbb{R}$
Performance metric	Classification error (0-1 loss) for test data	Classification error (0-1 loss) for test data	Mean Square Error
Popular surrogate loss (for training)	Logistic loss / exponential loss / square loss	Multiclass logistic loss (Cross-Entropy loss)	Square loss

The objective function for learning linear regression under square loss

- $\hat{w} = \text{argmin}_w \frac{1}{n}\sum_{i=1}^{n}(x_i^T w - y_i)^2 = \text{argmin}_w \|Xw - y\|_2^2$
  - aka: Ordinary Least Square (OLS)
- In-class exercise: Solve this optimization problem using Direct Solver (setting gradient to 0).

Derive the SGD algorithm

- Problem:
- $\widehat{w} = \mathrm{argmin}_w \frac{1}{n}\sum_{i=1}^{n}(x_i^T w - y_i)^2 = \mathrm{argmin}_w \|Xw - y\|_2^2$

- Step 1: Calculate the gradient of the square loss

- Step 2: Write the SGD update rule

Checkpoint: How to solve linear regression?

- Challenges:
  - We don’t have access to future data for prediction!
  - We also don’t have access to ground truth
- By solving an optimization problem that minimizes the loss function on the **training data**, and hope that it generalizes.
  - We can verify if it generalizes or not using hold-out / cross-validation …
- The least square optimization problem using square loss function:

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} (f(x_i) - y_i)^2$$

Checkpoint: Linear regression

- Stochastic Gradient Descent (SGD)
  - Using a stochastic approximation of the gradient:

$$\theta_{t+1} = \theta_t - \eta_t \hat{\nabla} f(\theta_t)$$

  - Calculated by one data point randomly sampled from dataset
- Linear regression
  - $\widehat{w} = \mathrm{argmin}_w \frac{1}{n} \sum_{i=1}^{n} (x_i^T w - y_i)^2 = \mathrm{argmin}_w \|Xw - y\|_2^2$
  - Direct solver: $\widehat{w} = (X^T X)^{-1} X^T y$
  - GD: $w \leftarrow w - 2\eta X^T (Xw - y)$
  - SGD: $w \leftarrow w - 2\eta x_i \ \ ({x_i}^T w - y_i)$