MIE524 Data Mining

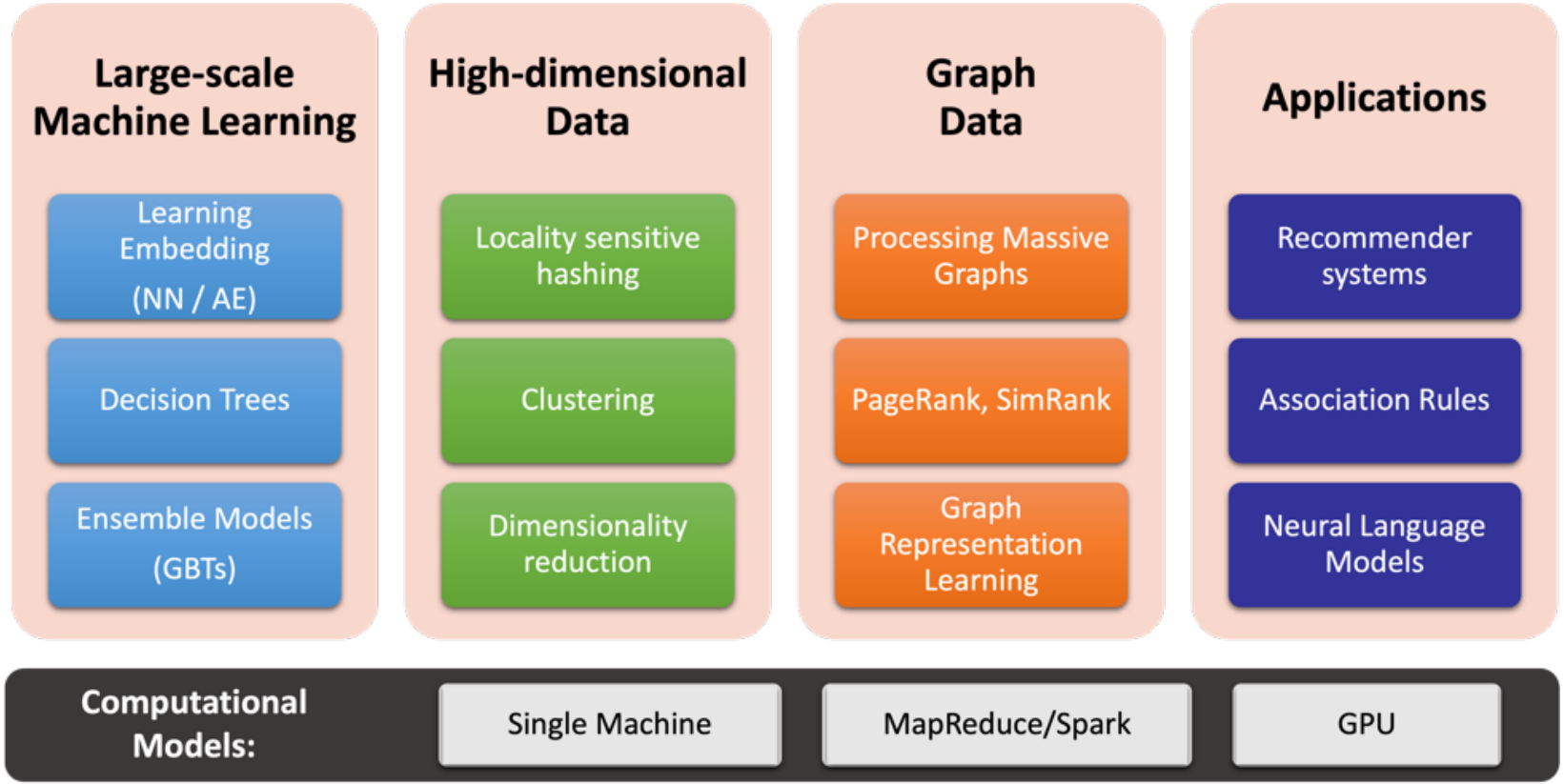
Machine Learning: Ensemble Models

Slides Credits:
Slides from Leskovec, Rajaraman, Ullman (http://www.mmds.org), Leskovec & Ghashami
Cheng Li, Protopapas & Pillai

Announcements

- Quiz 1 during the lab on Thursday Sept 26 (today)
- Midterm date: Thursday Oct 17, 2024 during lecture time

MIE524: Course Topics (Tentative)



Decision Trees: Learning Ensembles

Learning Ensembles

- **Learn multiple trees and combine their predictions**
  - The "wisdom of crowds"
  - A group/ ensemble of base learners that collectively achieve a better final prediction.
  - Decision trees are prone to
    - Overfitting (high variance and low bias) when it hasn't been pruned
    - Underfitting (low variance and high bias) when it's very small, i.e. a decision stump.
  - Ensemble reduces bias or variance, yielding better model performance.

Learning Ensembles

- **There are two main types of ensemble learning:**
  - Bagging and Boosting
- **Bagging (bootstrap aggregation):**
  - Learns multiple trees in parallel over independent samples of the training data
  - **1) Bootstrapping:** Given a dataset, create multiple datasets by sampling data points randomly and with replacement.
  - **2) Parallel training:** Train decision trees on samples independently and in parallel with each other
  - **3) Aggregation:** Depending on the task (i.e. regression or classification), an average or a majority of the predictions are computed for a more accurate estimate.
    - Regression, an average is taken of all the outputs predicted by the individual classifiers; this is known as "soft voting".
    - Classification, the class with the highest majority of votes is accepted; this is "hard voting" or majority voting.

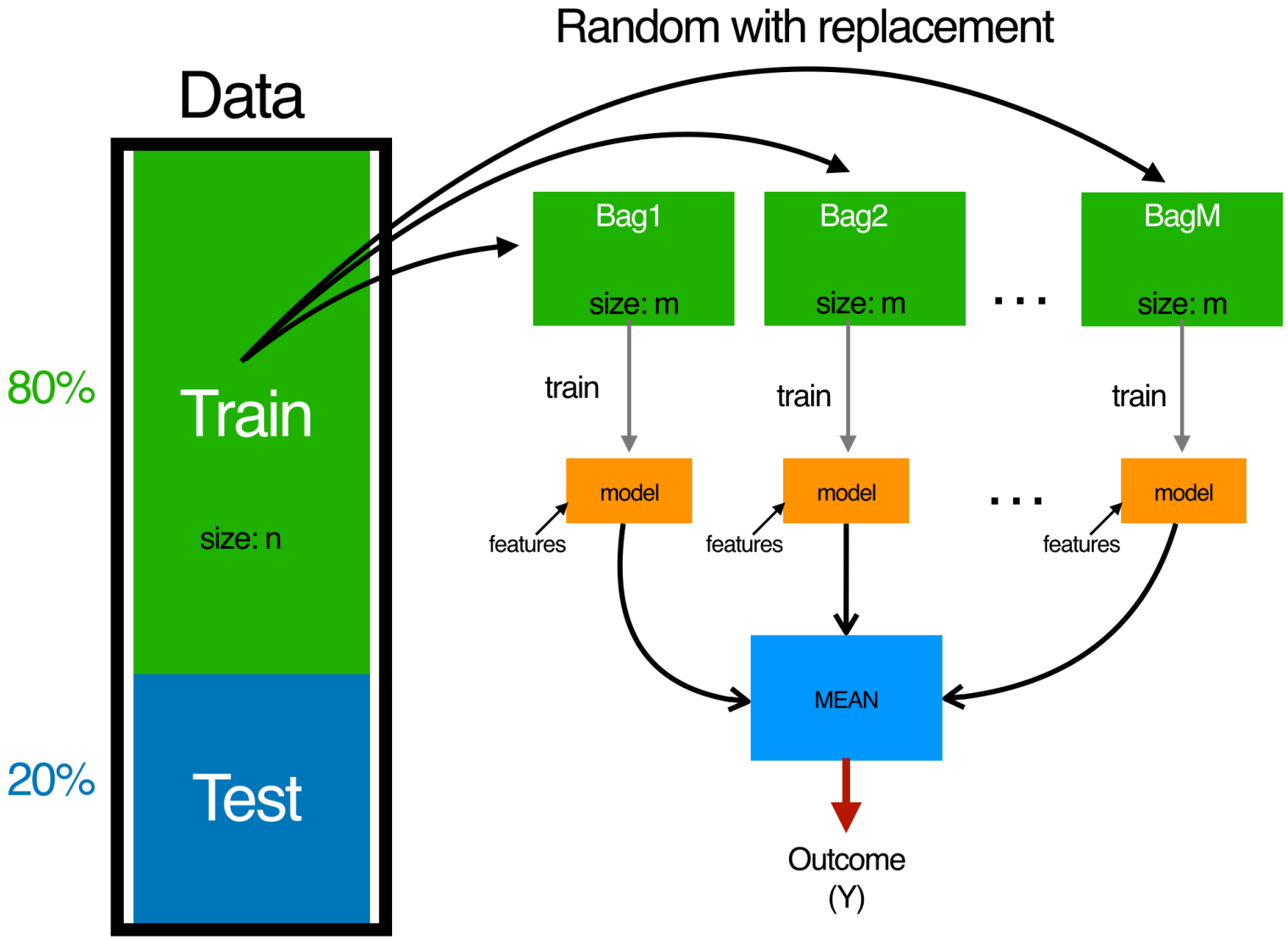
Bootstrap Sampling

Data: $S = \{(x_i, y_i)\}_{i = 1,\ldots,n}$

Bootstrap Sample: subsample of S, drawn **with replacement**

https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

Bootstrap **Aggregating (Bagging)**

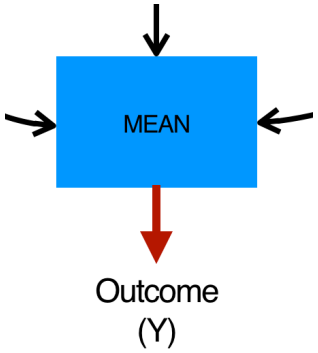


http://www.ub.edu/stat/docencia/EADB/bagging.html

Bagging

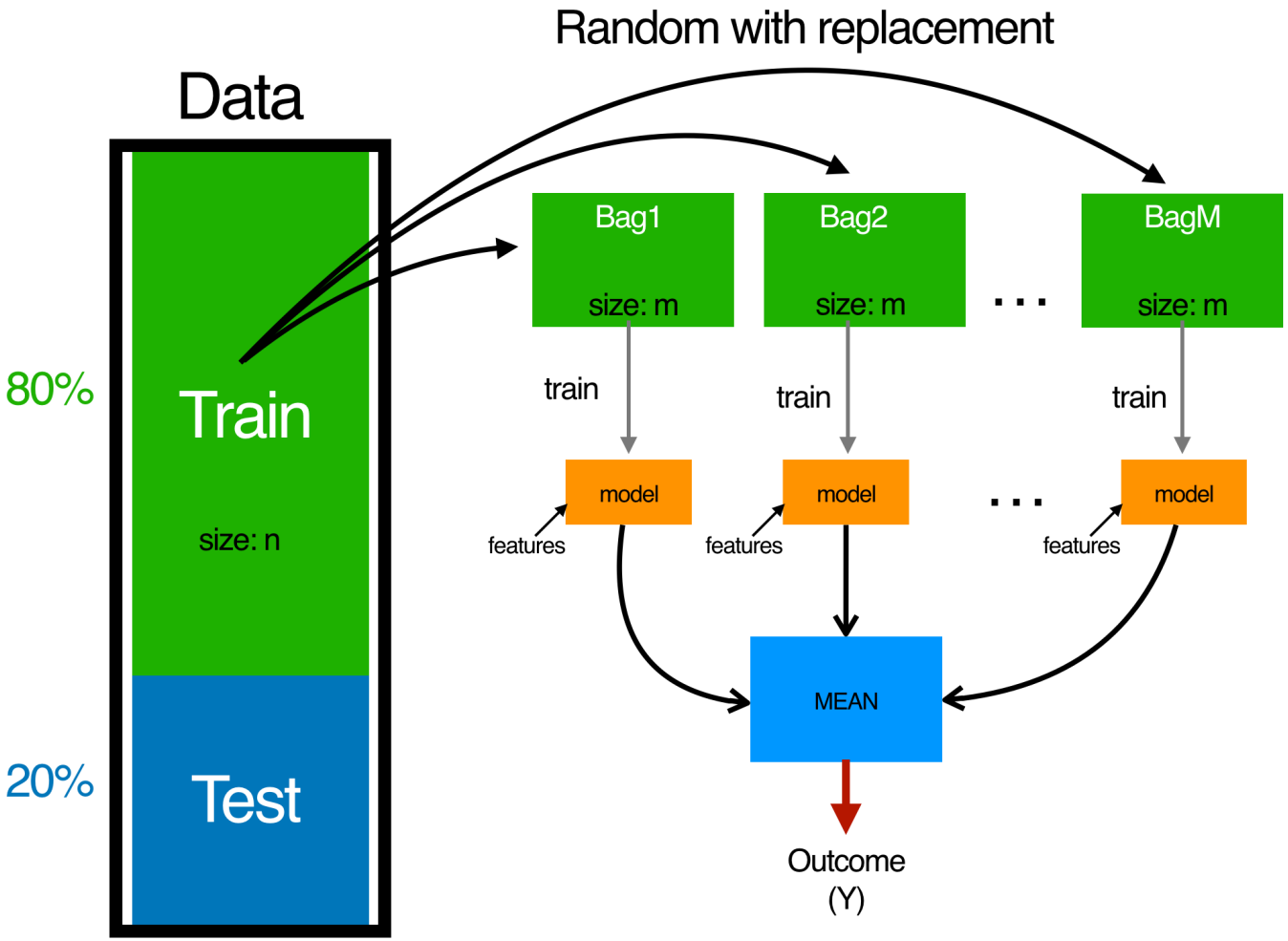
Aggregation:

- **Majority vote!**



Benefits:

- Even if single model overfits, on average they will not



http://www.ub.edu/stat/docencia/EADB/bagging.html

(2) Improvement: Random Forests

- So far we did *instance bagging*.
  - **Decision trees are greedy**
  - They choose which variable to split on using a greedy algorithm that minimizes error.
  - Even with Bagging, the decision trees can have a lot of structural similarities and in turn have high correlation in their predictions.

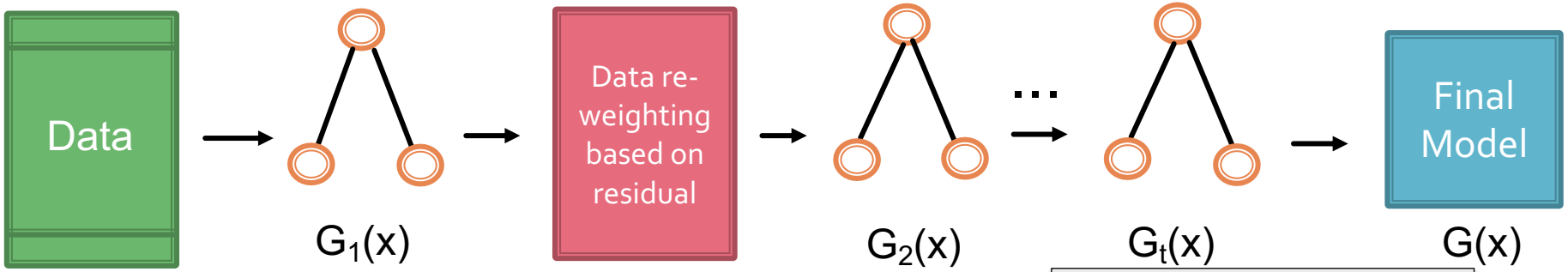
- **Feature Bagging**
  - Pick a random sample of features at each split
  - less correlation among trees
  - **Random forest**

(2) Improvement: Random Forests

- Train a **Bagged Decision Tree**
- But use a modified tree learning algorithm that selects (at each candidate split) **a random subset of features**
  - If we have d features, consider $\sqrt{d}$ random features
- **This is called: Feature bagging**
  - **Benefit:** Breaks correlation between trees
    - If one feature is very strong predictor, then every tree will select it, causing trees to be correlated.
- **Random Forests achieve state-of-the-art results in many classification problems!**

(3): Boosting

- **Boosting:** Another ensemble learning algorithm
  - Combines the outputs of many "weak" classifiers to produce a powerful "committee"
  - Learns multiple trees **sequentially**, each trying to improve upon its predecessor
  - Final classifier is weighted sum of the individual classifiers



Residual: Difference between prediction and ground truth

$$G(x) = \sum_{i=1}^{t} G_i(x)$$

What is Gradient Boosting

Slides modified from Cheng Li

Gradient Boosting = Gradient Descent + Boosting

Gradient Boosting

- Fit an additive model (ensemble) $\sum_t \rho_t h_t(x)$ in a forward stage-wise manner.
- In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
- In Gradient Boosting, "shortcomings" are identified by gradients.
- gradients tell us how to improve our model.

Gradient Boosting for Regression

Slides by Cheng Li

Let's play a game...
You are given $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, and the task is to fit a model $F(x)$ to minimize square loss.
Suppose your friend wants to help you and gives you a model F.
You check his model and find the model is good but not perfect.
There are some mistakes: $F(x_1) = 0.8$, while $y_1 = 0.9$, and $F(x_2) = 1.4$ while $y_2 = 1.3$... How can you improve this model?
Rule of the game:

- You are not allowed to remove anything from F or change any parameter in F.
- You can add an additional model (regression tree) h to F, so the new prediction will be $F(x) + h(x)$.

Gradient Boosting for Regression

Simple solution:

You wish to improve the model such that

$$F(x_1) + h(x_1) = y_1$$
$$F(x_2) + h(x_2) = y_2$$
$$\ldots$$
$$F(x_n) + h(x_n) = y_n$$

Simple solution:

Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$
$$h(x_2) = y_2 - F(x_2)$$
$$\ldots$$
$$h(x_n) = y_n - F(x_n)$$

Gradient Boosting for Regression

Simple solution:

Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$
$$h(x_2) = y_2 - F(x_2)$$
$$\ldots$$
$$h(x_n) = y_n - F(x_n)$$

Can any regression tree h achieve this goal perfectly?

Gradient Boosting for Regression

Simple solution:

Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$
$$h(x_2) = y_2 - F(x_2)$$
$$\ldots$$
$$h(x_n) = y_n - F(x_n)$$

Can any regression tree h achieve this goal perfectly?
Maybe not....
But some regression tree might be able to do this approximately.
How?
Just fit a regression tree h to data
$(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), ..., (x_n, y_n - F(x_n))$
Congratulations, you get a better model!

Gradient Boosting for Regression

Simple solution:

$y_i - F(x_i)$ are called **residuals**. These are the parts that existing model F cannot do well.

The role of h is to compensate the shortcoming of existing model F.

If the new model $F + h$ is still not satisfactory, we can add another regression tree...

How is this related to gradient descent?

Gradient Boosting for Regression

Slides by Cheng Li

Gradient Descent

Minimize a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$

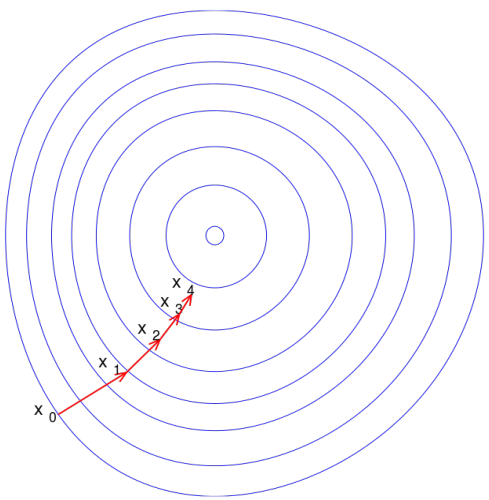


Figure: Gradient Descent. Source: `http://en.wikipedia.org/wiki/Gradient_descent`

Gradient Boosting for Regression

Slides modified from Cheng Li

How is this related to gradient descent?

- Loss function $L\big(y, F(x)\big) = \sum_{x_i}[½(y_i - F(x_i))^2]$
- We want to minimize the loss by adjusting the $F(x_1), F(x_2), \ldots$
- $F(x_i)$ are just numbers (predictions). We can consider them as parameters (that need to be adjusted) and compute the derivatives
  - $\frac{\partial L}{\partial F(x_i)} = \frac{\partial \sum_{x_i}[½(y_i - F(x_i))^2]}{\partial F(x_i)} = \frac{\partial ½(y_i - F(x_i))^2}{\partial F(x_i)} = \frac{\partial ½(y_i{}^2 - 2y_iF(x_i) + F(x_i)^2)}{\partial F(x_i)}$
  - $= \frac{\partial (½y^2 - yF(x_i) + ½F(x_i)^2)}{\partial F(x_i)} = \mathrm{F(x_i)} - \mathrm{y_i}$
- So we can interpret the residuals as *negative* gradients:
  - $\mathrm{y_i} - \mathrm{F(x_i)} = -\frac{\partial L}{\partial F(x_i)}$

Gradient Boosting for Regression

Slides by Cheng Li

How is this related to gradient descent?

$$F(x_i) := F(x_i) + h(x_i)$$
$$F(x_i) := F(x_i) + y_i - F(x_i)$$
$$F(x_i) := F(x_i) - 1\frac{\partial J}{\partial F(x_i)}$$
$$\theta_i := \theta_i - \rho\frac{\partial J}{\partial \theta_i}$$

Gradient Boosting for Regression

How is this related to gradient descent?

For regression with **square loss**,

$$\textit{residual} \Leftrightarrow \textit{negative gradient}$$

$$\textit{fit } h \textit{ to residual} \Leftrightarrow \textit{fit } h \textit{ to negative gradient}$$

$$\textit{update } F \textit{ based on residual} \Leftrightarrow \textit{update } F \textit{ based on negative gradient}$$

Gradient Boosting for Regression

How is this related to gradient descent?

For regression with **square loss**,

$$\textit{residual} \Leftrightarrow \textit{negative gradient}$$

$$\textit{fit } h \textit{ to residual} \Leftrightarrow \textit{fit } h \textit{ to negative gradient}$$

$$\textit{update } F \textit{ based on residual} \Leftrightarrow \textit{update } F \textit{ based on negative gradient}$$

So we are actually updating our model using **gradient descent**!

Gradient Boosting for Regression

How is this related to gradient descent?

For regression with **square loss**,

$$\textit{residual} \Leftrightarrow \textit{negative gradient}$$

$$\textit{fit } h \textit{ to residual} \Leftrightarrow \textit{fit } h \textit{ to negative gradient}$$

$$\textit{update } F \textit{ based on residual} \Leftrightarrow \textit{update } F \textit{ based on negative gradient}$$

So we are actually updating our model using **gradient descent**!
It turns out that the concept of **gradients** is more general and useful than the concept of **residuals**. So from now on, let's stick with gradients. The reason will be explained later.

Gradient Boosting for Regression

Slides modified from Cheng Li

Regression with square Loss

Let us summarize the algorithm we just derived using the concept of gradients. Negative gradient:

$$-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

start with an initial model, say, $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$
iterate until converge:

- calculate negative gradients $-g(x_i)$
- fit a regression tree h to negative gradients $-g(x_i)$
- $F := F + \rho h$, where ρ is the learning rate

The benefit of formulating this algorithm using gradients is that it allows us to consider other loss functions and derive the corresponding algorithms in the same way.

Gradient Boosting for Regression

Slides by Cheng Li

Loss Functions for Regression Problem

Why do we need to consider other loss functions? Isn't square loss good enough?

Gradient Boosting for Regression

Loss Functions for Regression Problem

Square loss is:

- ✓ Easy to deal with mathematically
- × Not robust to outliers
  Outliers are heavily punished because the error is squared.
  Example:

y_i	0.5	1.2	2	5*
$F(x_i)$	0.6	1.4	1.5	1.7
$L = (y - F)^2/2$	0.005	0.02	0.125	5.445

Consequence?

Gradient Boosting for Regression

Slides by Cheng Li

Loss Functions for Regression Problem

Square loss is:

- ✓ Easy to deal with mathematically
- × Not robust to outliers

  Outliers are heavily punished because the error is squared.

  Example:

y_i	0.5	1.2	2	5*
$F(x_i)$	0.6	1.4	1.5	1.7
$L = (y - F)^2/2$	0.005	0.02	0.125	5.445

  Consequence?

  Pay too much attention to outliers. Try hard to incorporate outliers into the model. Degrade the overall performance.

Gradient Boosting for Regression

Loss Functions for Regression Problem

- Absolute loss (more robust to outliers)

$$L(y, F) = |y - F|$$

- Huber loss (more robust to outliers)

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \leq \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta \end{cases}$$

Gradient Boosting for Regression

Slides by Cheng Li

Loss Functions for Regression Problem

- Absolute loss (more robust to outliers)

$$L(y, F) = |y - F|$$

- Huber loss (more robust to outliers)

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \leq \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta \end{cases}$$

y_i	0.5	1.2	2	5*
$F(x_i)$	0.6	1.4	1.5	1.7
Square loss	0.005	0.02	0.125	5.445
Absolute loss	0.1	0.2	0.5	3.3
Huber loss($\delta = 0.5$)	0.005	0.02	0.125	1.525

Gradient Boosting for Regression

Slides by Cheng Li

Other Loss Functions [Hastie, Tibshirani, Friedman, 2017]

TABLE 10.2. *Gradients for commonly used loss functions.*

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \leq \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha\text{th-quantile}\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Gradient Boosting: the algorithm

1. Fit a simple model $T^{(0)}$ on the training data

$$\{(x_1, y_1), \ldots, (x_N, y_N)\}$$

Set $T \leftarrow T^{(0)}$.

Compute the residuals $\{r_1, \ldots, r_N\}$ for T.

2. Fit a simple model, $T^{(1)}$, to the current ***residuals***, i.e. train using

$$\{(x_1, r_1), \ldots, (x_N, r_N)\}$$

3. Set $T \leftarrow T + \lambda T^{(1)}$

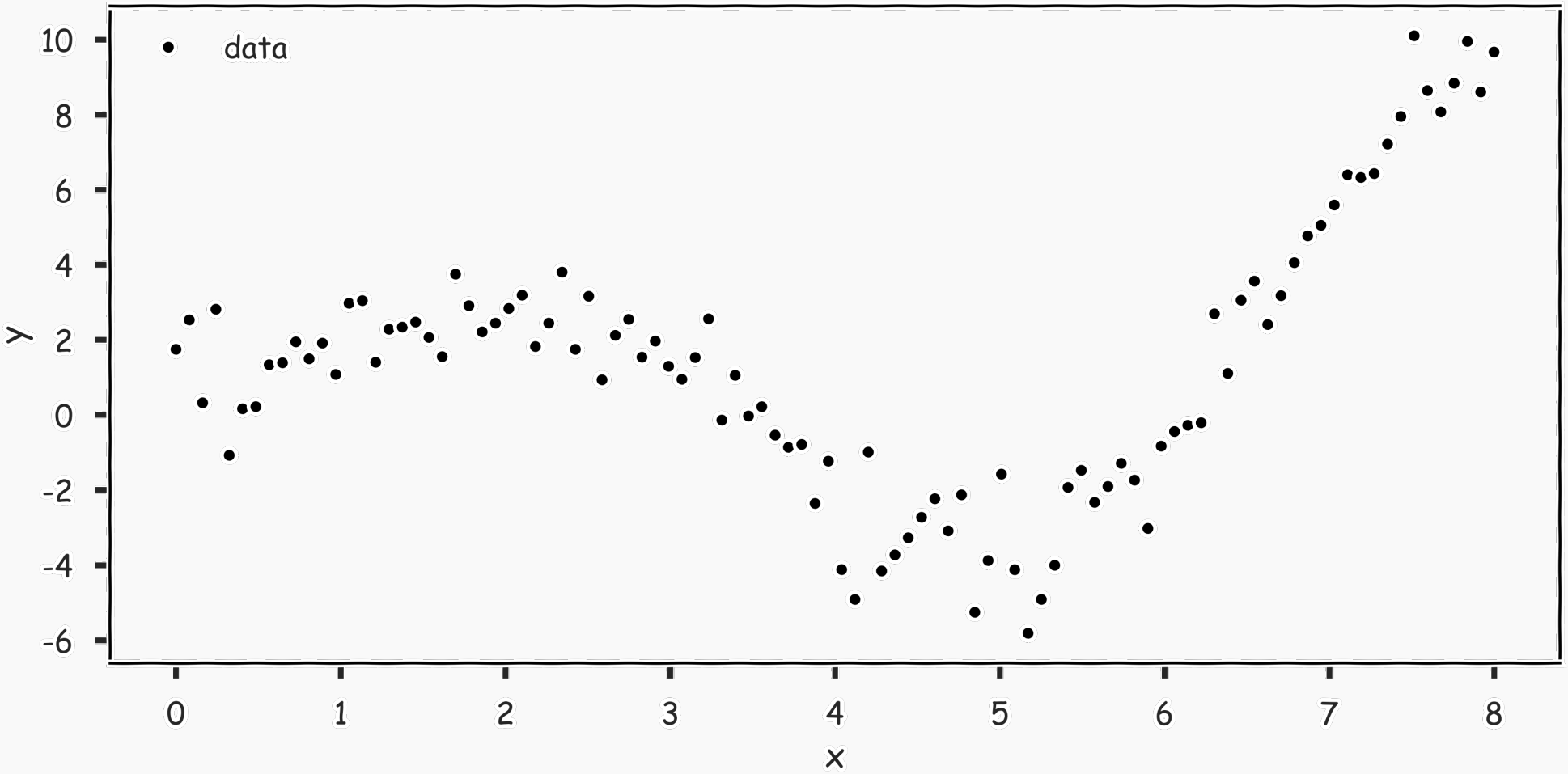
4. Compute residuals, set $r_n \leftarrow r_n - \lambda T^i(x_n), \ n = 1, \ldots, N$

5. Repeat steps 2-4 until ***stopping*** condition met.

where λ is a constant called the ***learning rate***.

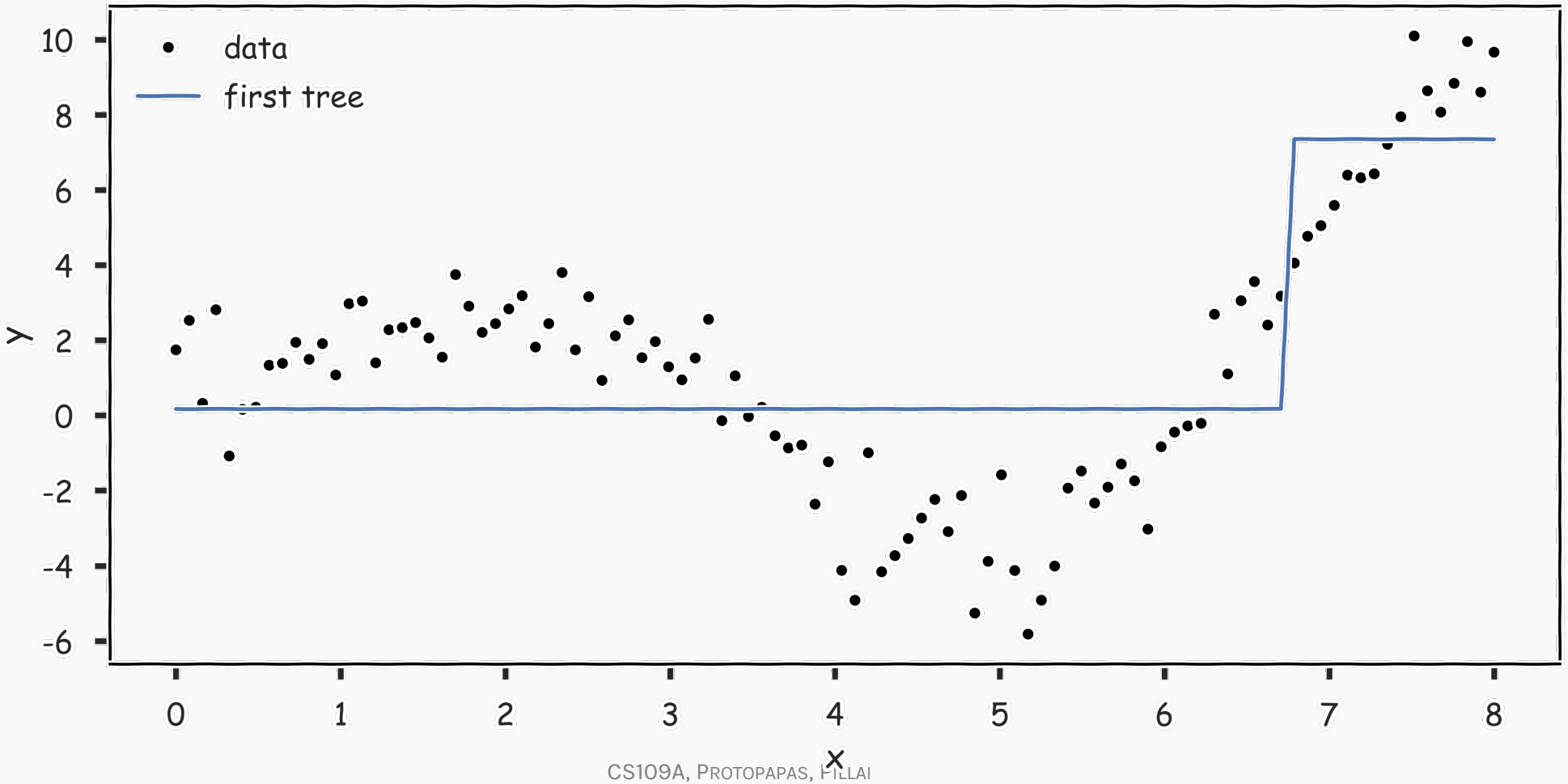
Gradient Boosting: illustration

training data: $\{(x_1, y_1), \ldots, (x_N, y_N)\}$



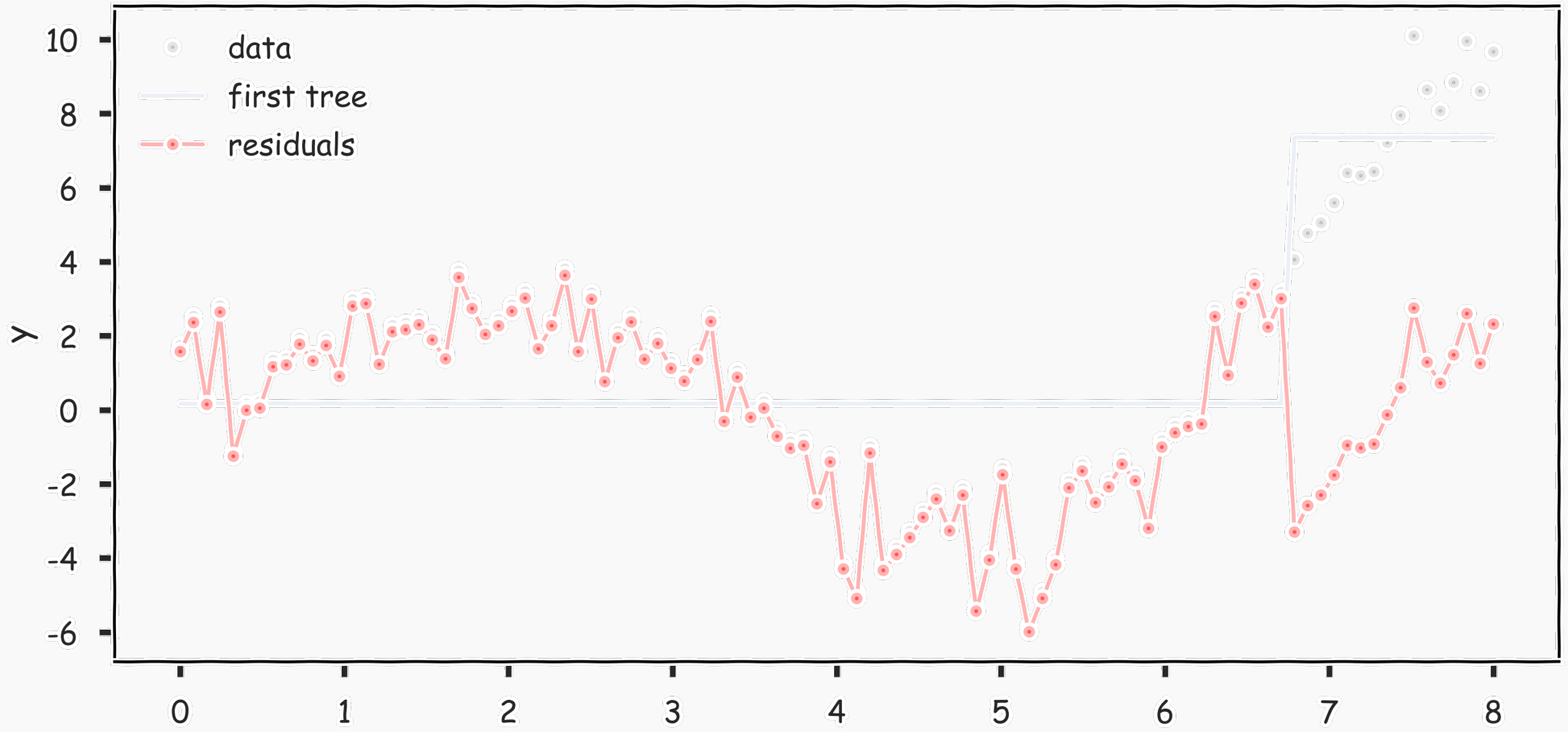
Gradient Boosting: illustration

Fit a simple model $T^{(0)}$



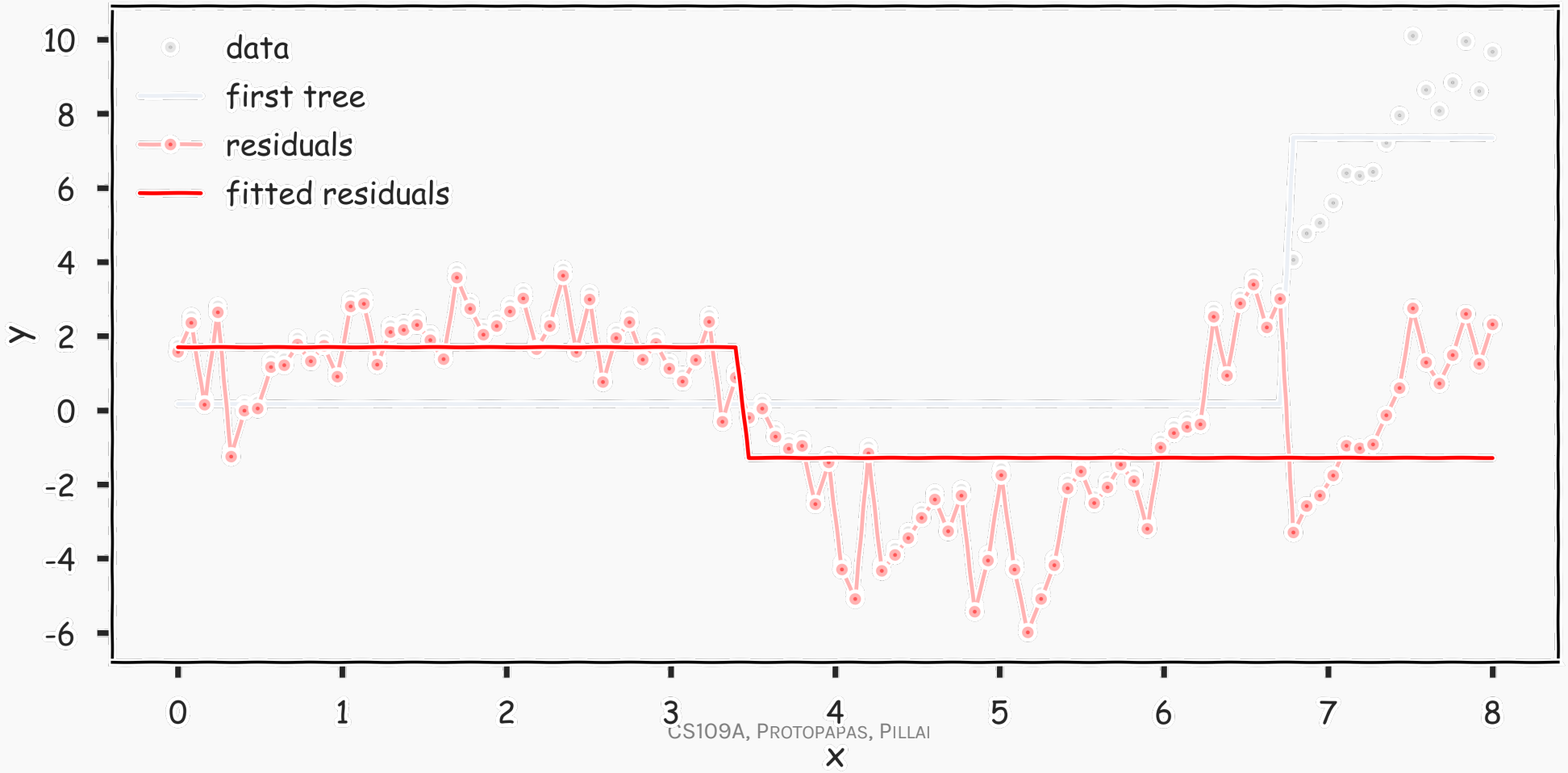
Gradient Boosting: illustration

Compute the residuals $\{r_1, \ldots, r_N\}$ for T.



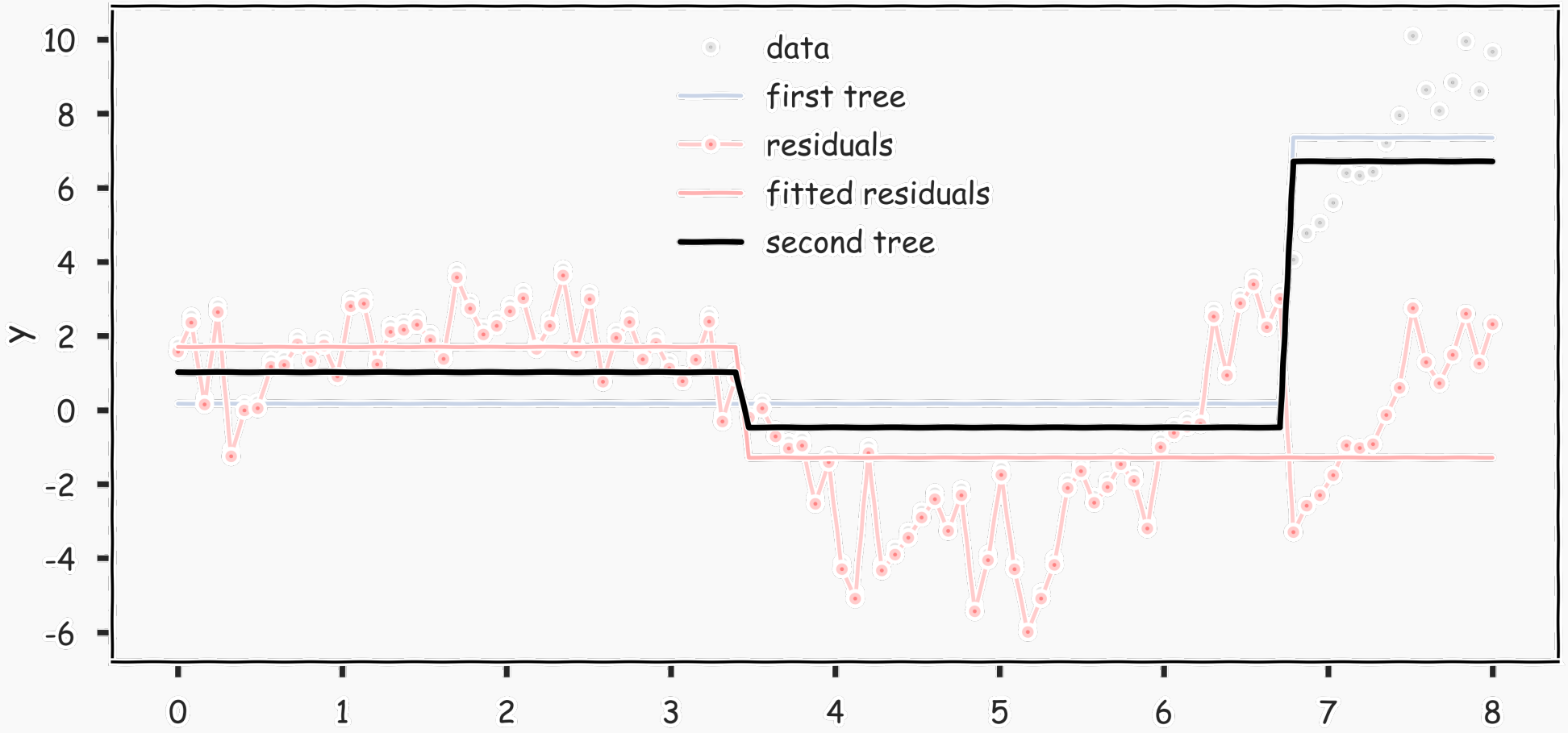
Gradient Boosting: illustration

train using: $\{(x_1, r_1), \ldots, (x_N, r_N)\}$



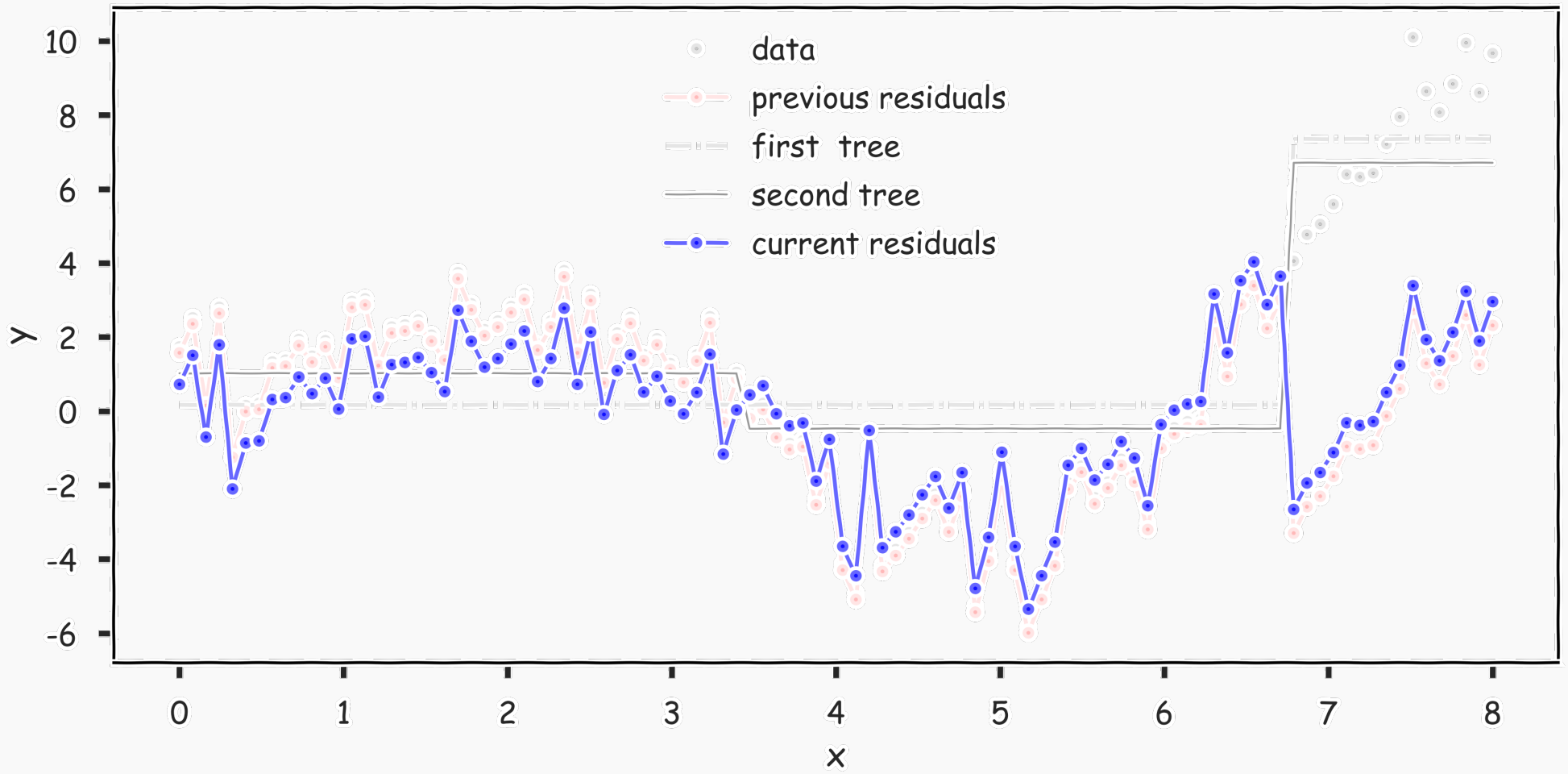
Gradient Boosting: illustration

Set $T \leftarrow T + \lambda T^{(1)}$

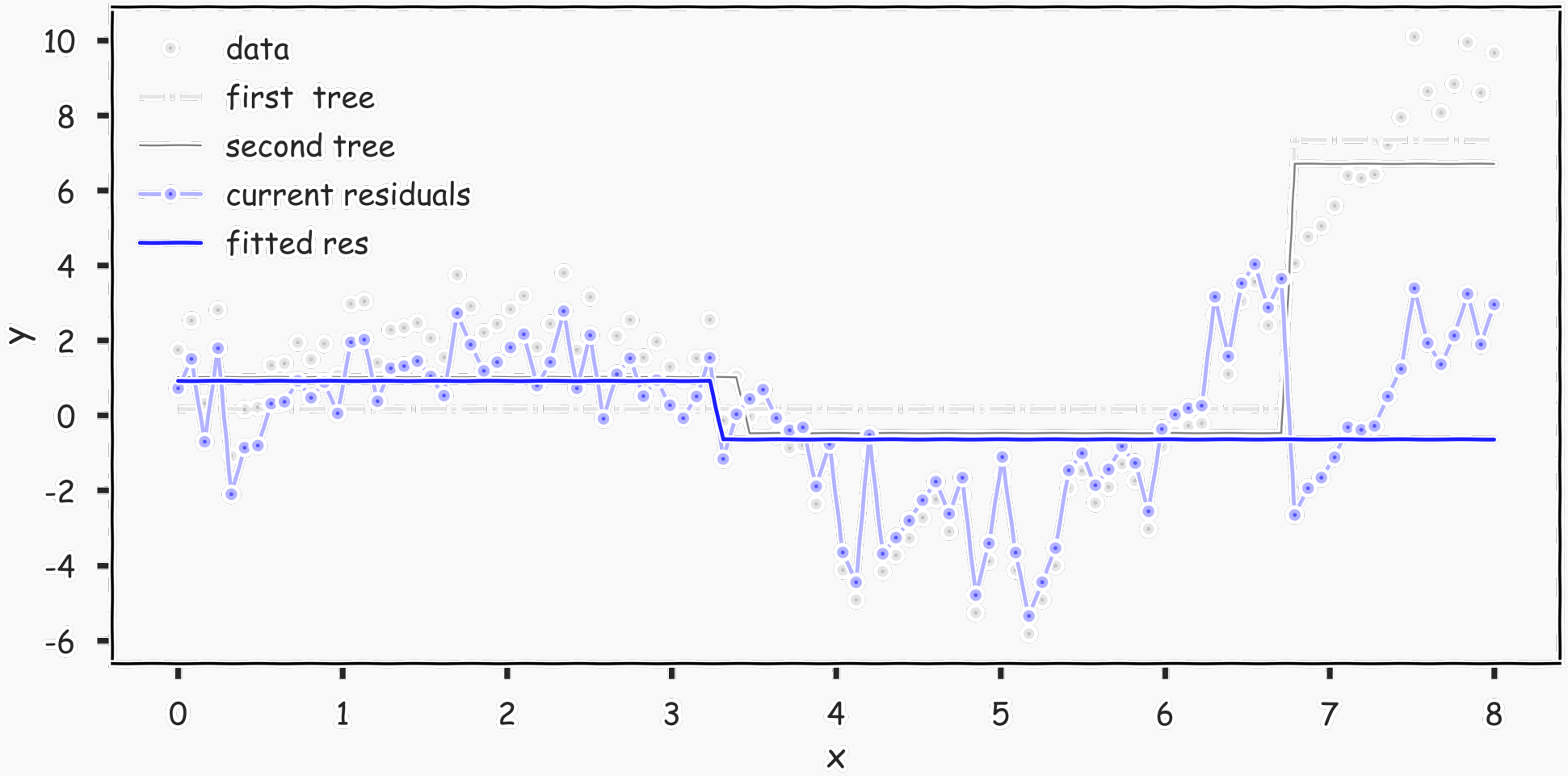


Gradient Boosting: illustration

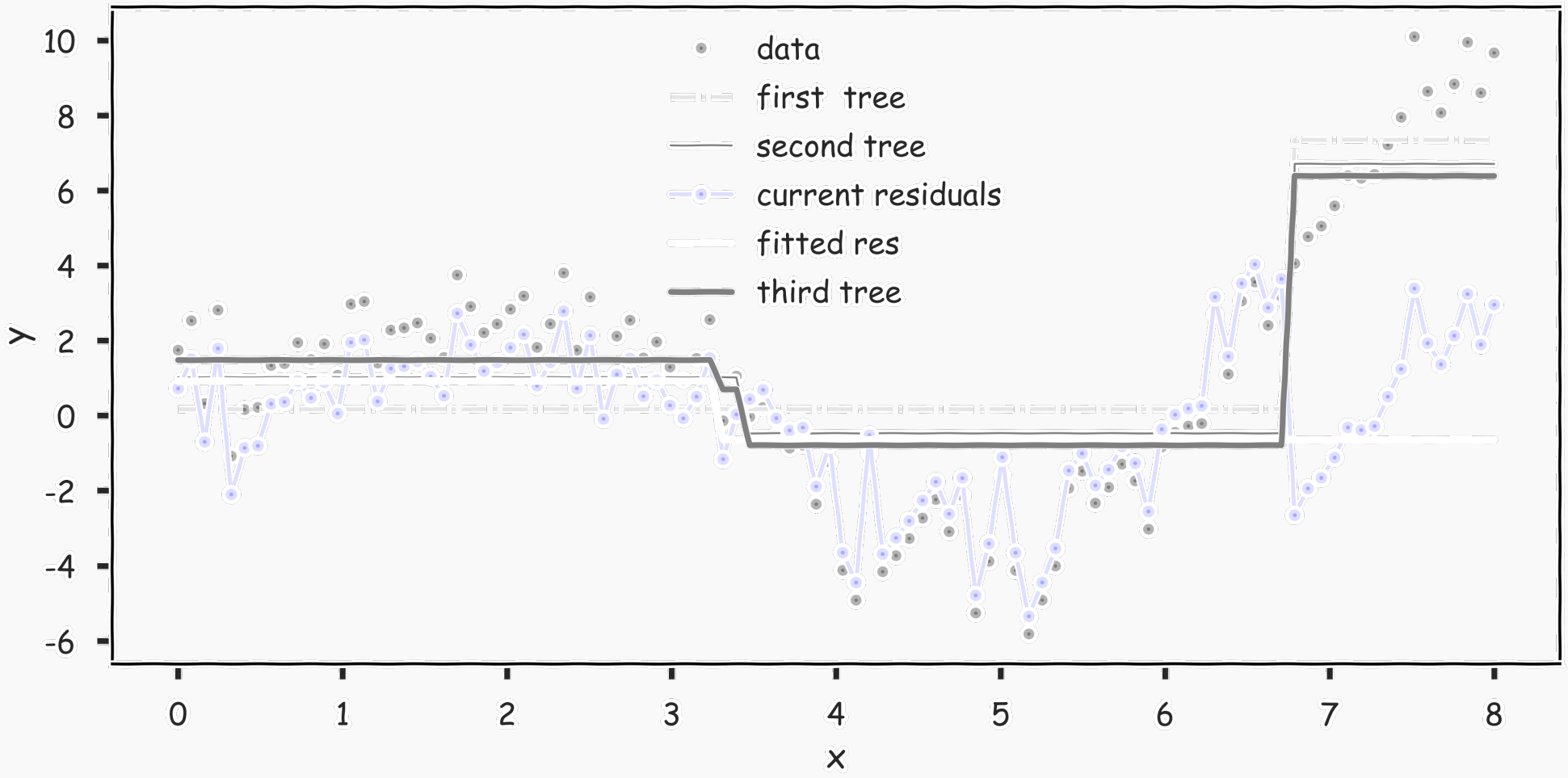
$$r_n \leftarrow r_n - \lambda T^i(x_n)$$



Gradient Boosting: illustration



Gradient Boosting: illustration



Gradient Boosting for Regression

Slides by Cheng Li

Summary

- Fit an additive model $F = \sum_t \rho_t h_t$ in a forward stage-wise manner.
- In each stage, introduce a new regression tree h to compensate the shortcomings of existing model.
- The "shortcomings" are identified by negative gradients.
- For any loss function, we can derive a gradient boosting algorithm.
- Absolute loss and Huber loss are more robust to outliers than square loss.

Things not covered

How to choose a proper learning rate for each gradient boosting algorithm. See [Friedman, 2001]

XGBoost

- XGBoost: e**X**treme **G**radient **B**oosting
  - A highly scalable implementation of gradient boosted decision trees with regularization

  Widely used by data scientists and provides state-of-the-art results on many problems!

- System optimizations:
  - Parallel tree constructions using column block structure
  - Distributed Computing for training very large models using a cluster of machines.
  - Out-of-Core Computing for very large datasets that don't fit into memory.