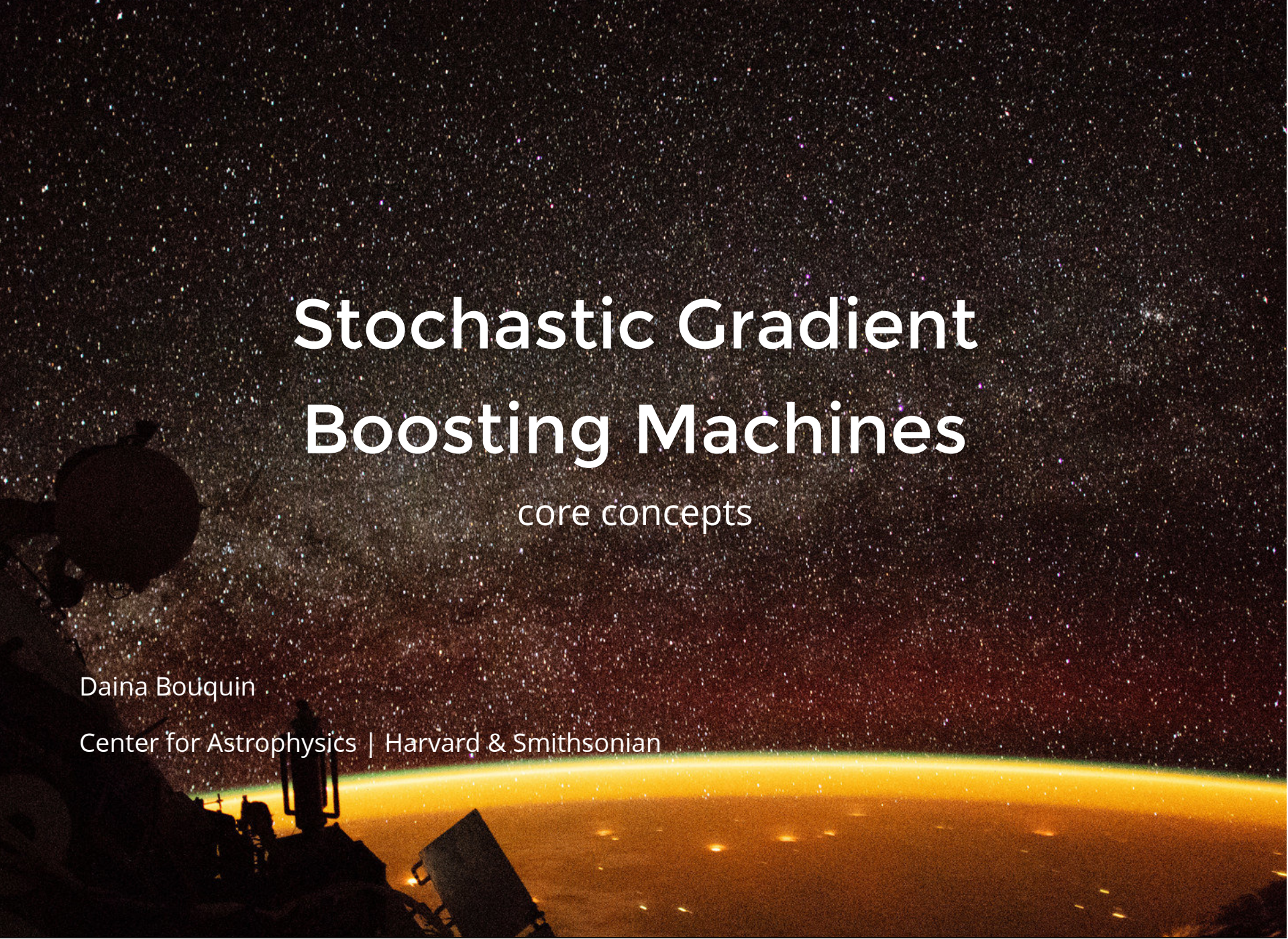


# Stochastic Gradient Boosting Machines

core concepts

Daina Bouquin

Center for Astrophysics | Harvard & Smithsonian

# I'm a librarian

MS Data Analytics

MS Library and Information Science

CAS Data Science

# I don't like the "Black Box" metaphor in machine learning.

It's discouraging and misleading.

Literacy in means being able to critically assess what you're presented with, learning to problem solve and actively contributing to a dialog.

You can't do that if you believe something is unknowable.

"...a deep learning system is not a black box; even the development of such a system need not be a black box. The real challenge, however, is that both of these things are complex, and not necessarily well understood."

Dallas Card

The “black box” metaphor in machine learning

# Stochastic Gradient Boosting Machines: not a black box

## Can a set of weak learners create a single strong learner?

Yes. Boosting algorithms iteratively learn weak classifiers with respect to a distribution and add them to a final strong classifier

**Boosting:**

ML ensemble method/metaheuristic

Helps with bias-variance tradeoff (reduces both)

**metaheuristic** is a higher-level procedure or heuristic designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computational capacity

high bias means you could miss relevant relations between features

**Bias** = error from erroneous assumptions in the learning algorithm

$\therefore$ *underfitting*

**Variance** = sensitivity to small fluctuations in the training set

$\therefore$ *overfitting*

You don't want to model noise

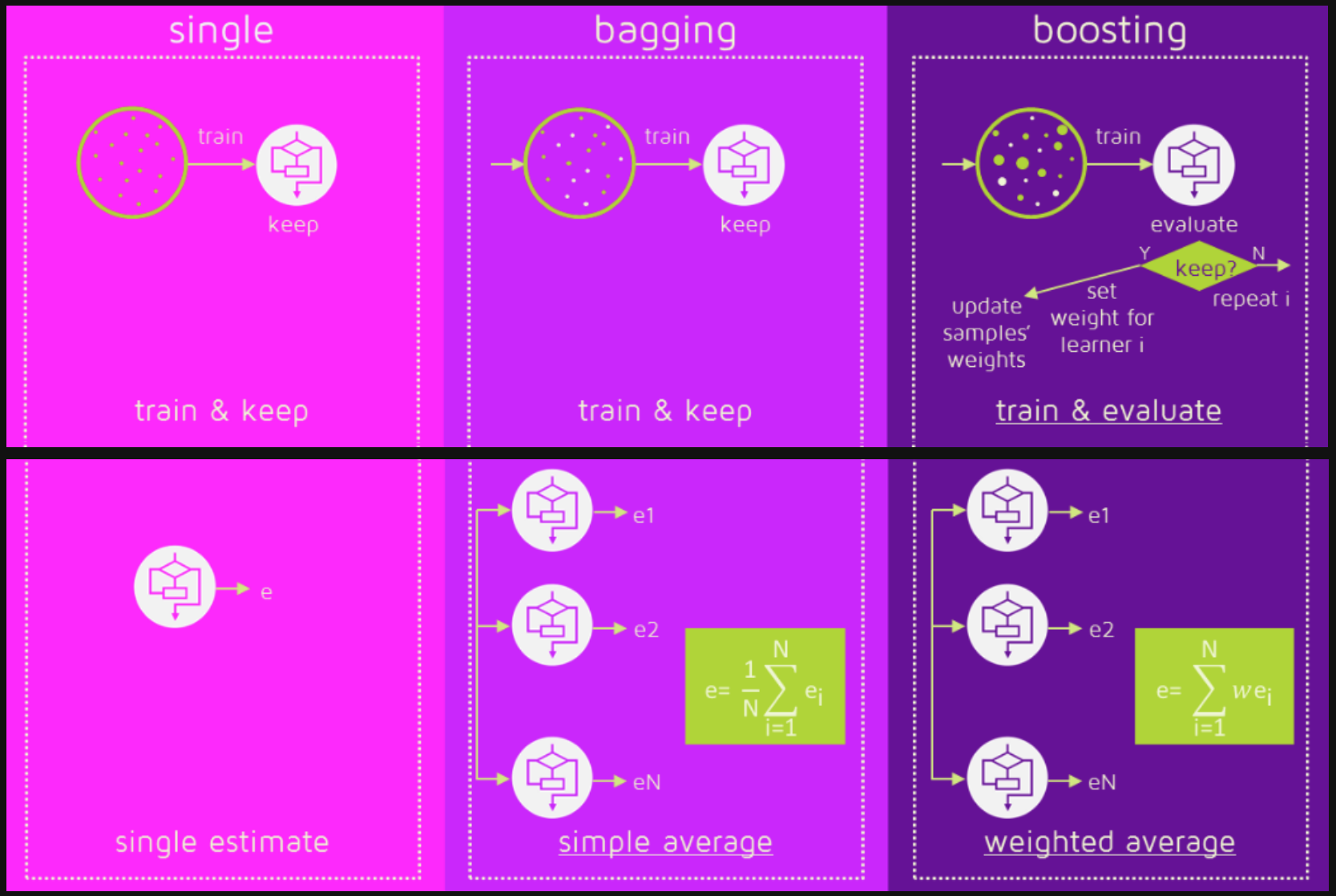
## Boosting algorithms:

Weighted in relation to the weak predictors' accuracy

Weighting decorrelates the predictors by focusing on regions missed by past predictors

New predictors learn from previous predictor mistakes

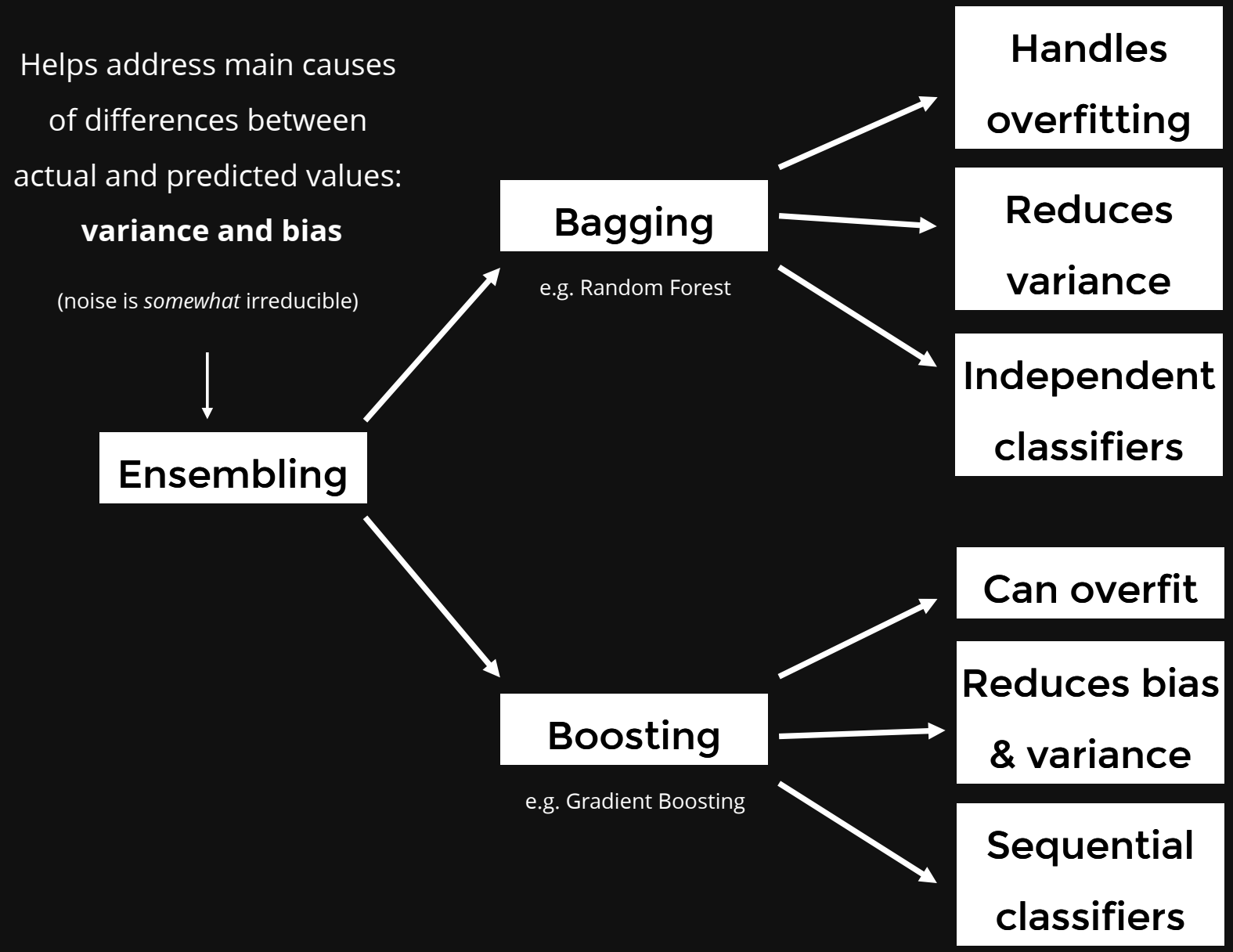
∴ take fewer iterations to converge




https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

Boosting means observations have an unequal probability of appearing in subsequent models

Observations with highest error appear most


Helps address main causes of differences between actual and predicted values: **variance and bias**
(noise is *somewhat* irreducible)
Ensembling
Bagging
e.g. Random Forest
Handles overfitting
Reduces variance
Independent classifiers
Boosting
e.g. Gradient Boosting
Can overfit
Reduces bias & variance
Sequential classifiers

# Boosting with Gradient Discent

gradient descent assuming a convex cost function

Local minimum must be a global minimum

**The point of GD is to **minimize the cost function****

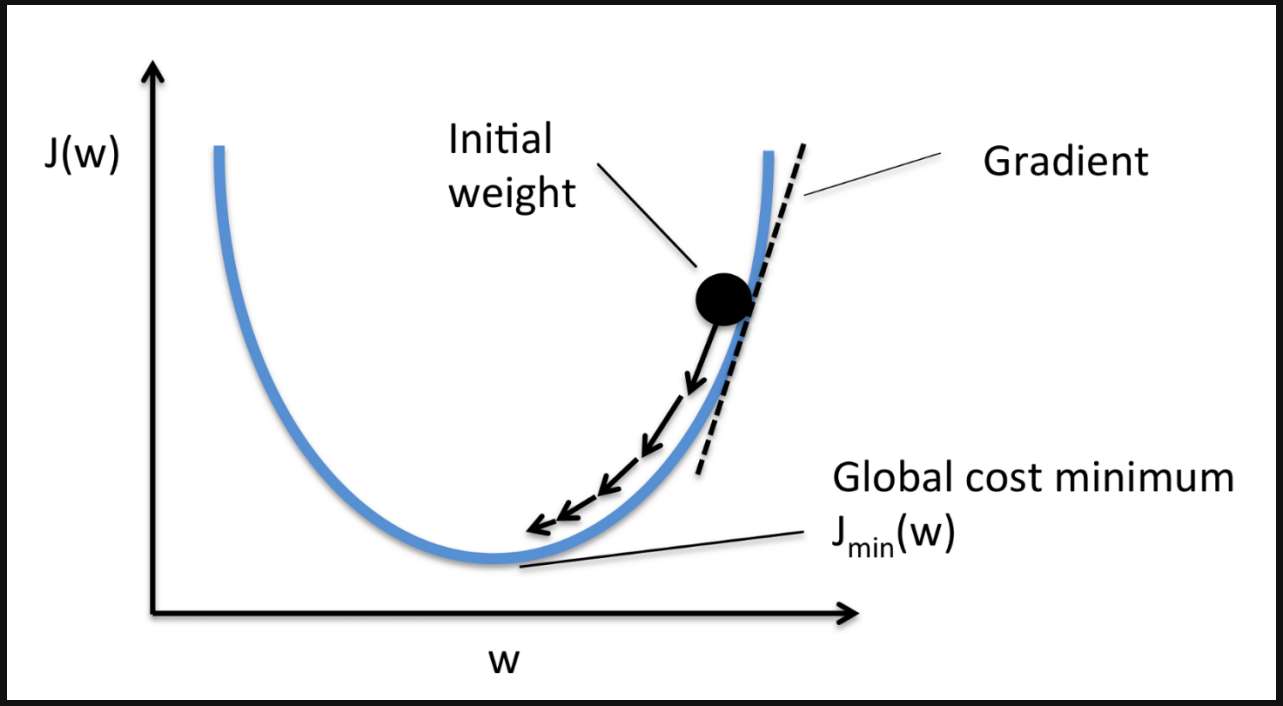
(find the lowest error value/the deepest valley in that function)

Most common cost function is mean squared error

Too much random noise can be an issue with convex optimization.

Non-convex optimization options for boosting exist though  e.g. BrownBoost

If you're worried about local minima check out restarts  (SGDR)




https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

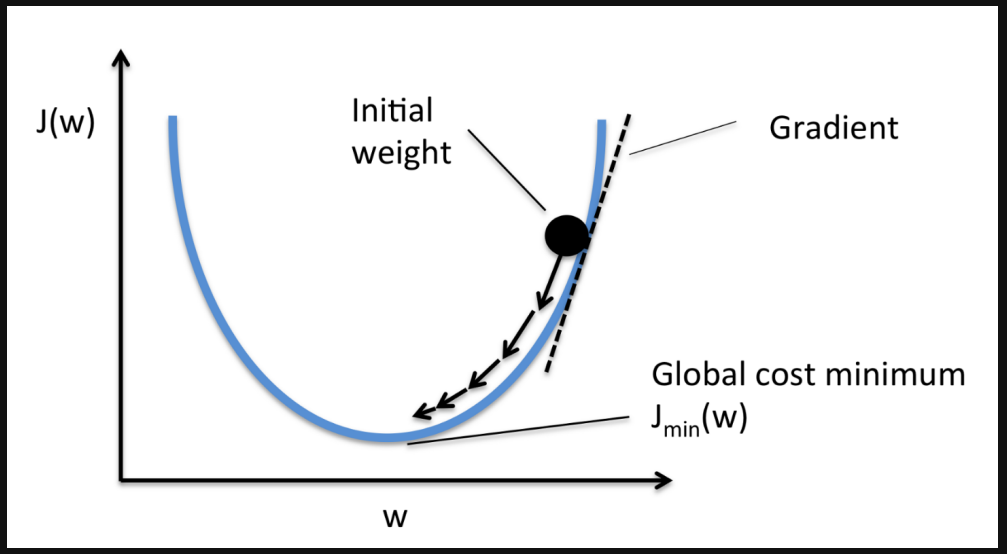
Slope points to the nearest valley

Choice of cost function will affect calculation of the gradient of each weight.

Cost function is for monitoring the error with each training example

The derivative of the cost function with respect to the weight (slope!) is where we shift the weight to minimize the error for that training example

This gives us direction




https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

GD optimizers use a technique called "annealing" to determine the learning rate (how small/large of a step to take) = **α**

if alpha is too large we overshoot the min

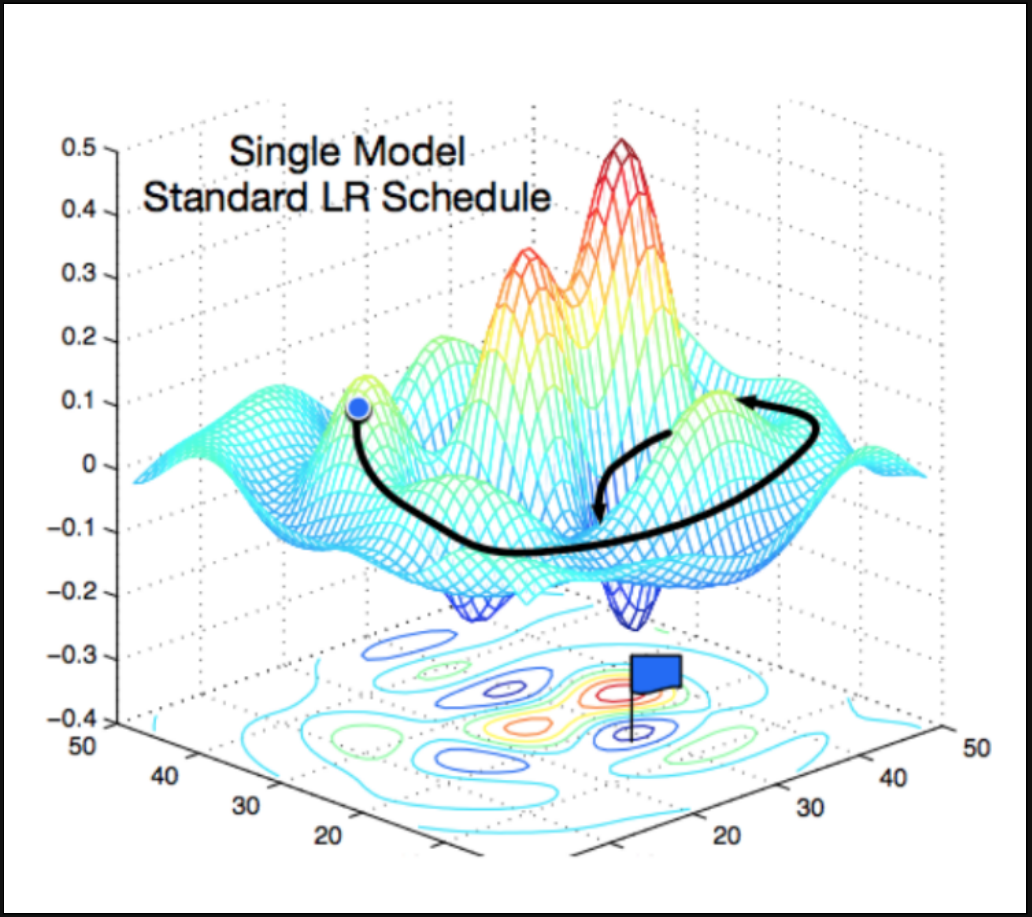
if alpha is too small we take too many iterations to find the min

Theta (weight) should decrease at each iteration

Example:

Black line represents a non linear loss function.

If our parameters are initialized to the blue dot, we need a way to move around parameter space to the lowest point.




https://medium.com/38th-street-studios/exploring-stochastic-gradient-descent-with-restarts-sgdr-fa206c38a74e

# Then just do it stochastically

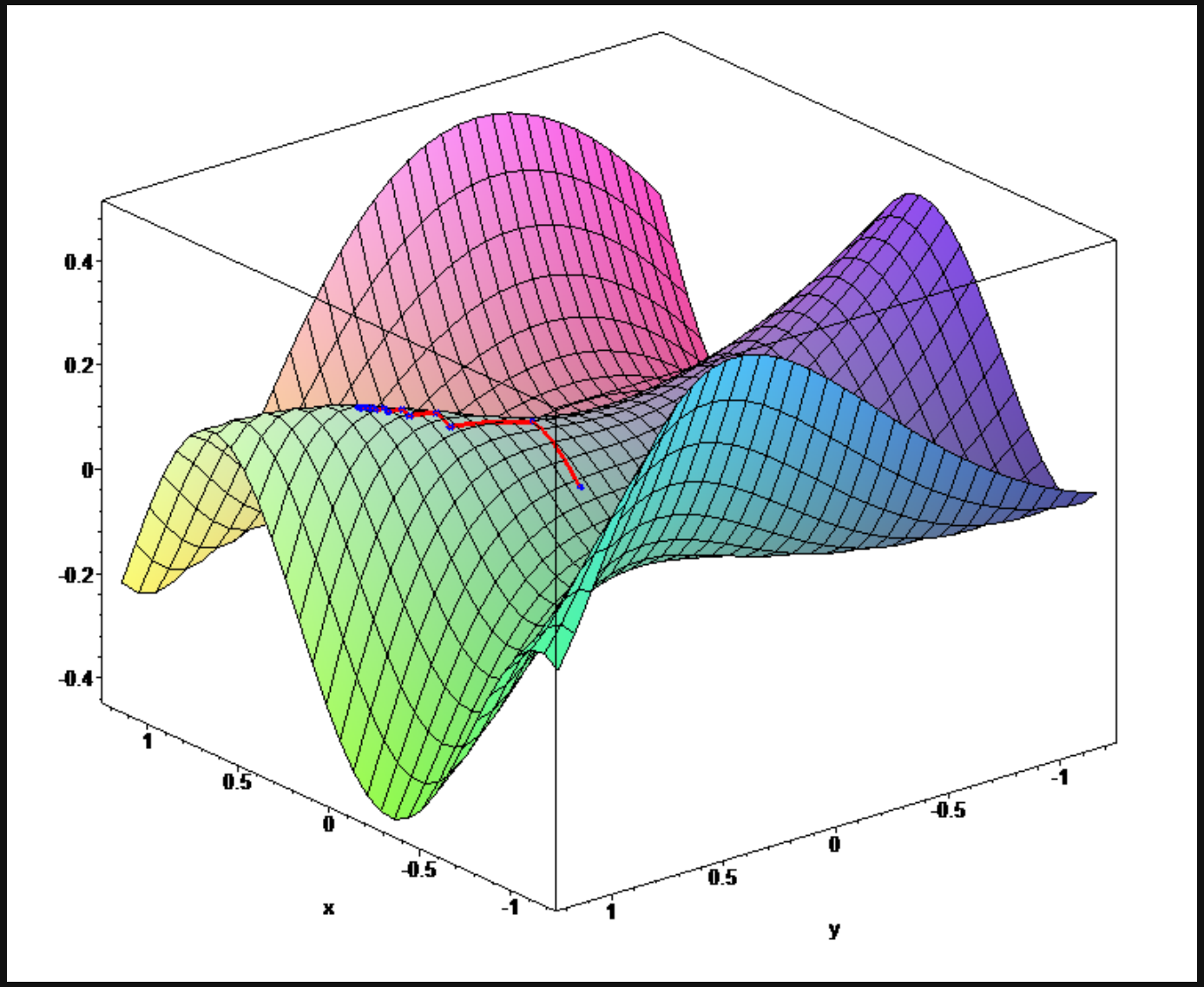
With every GD iteration shuffle the training set and pick a random training example

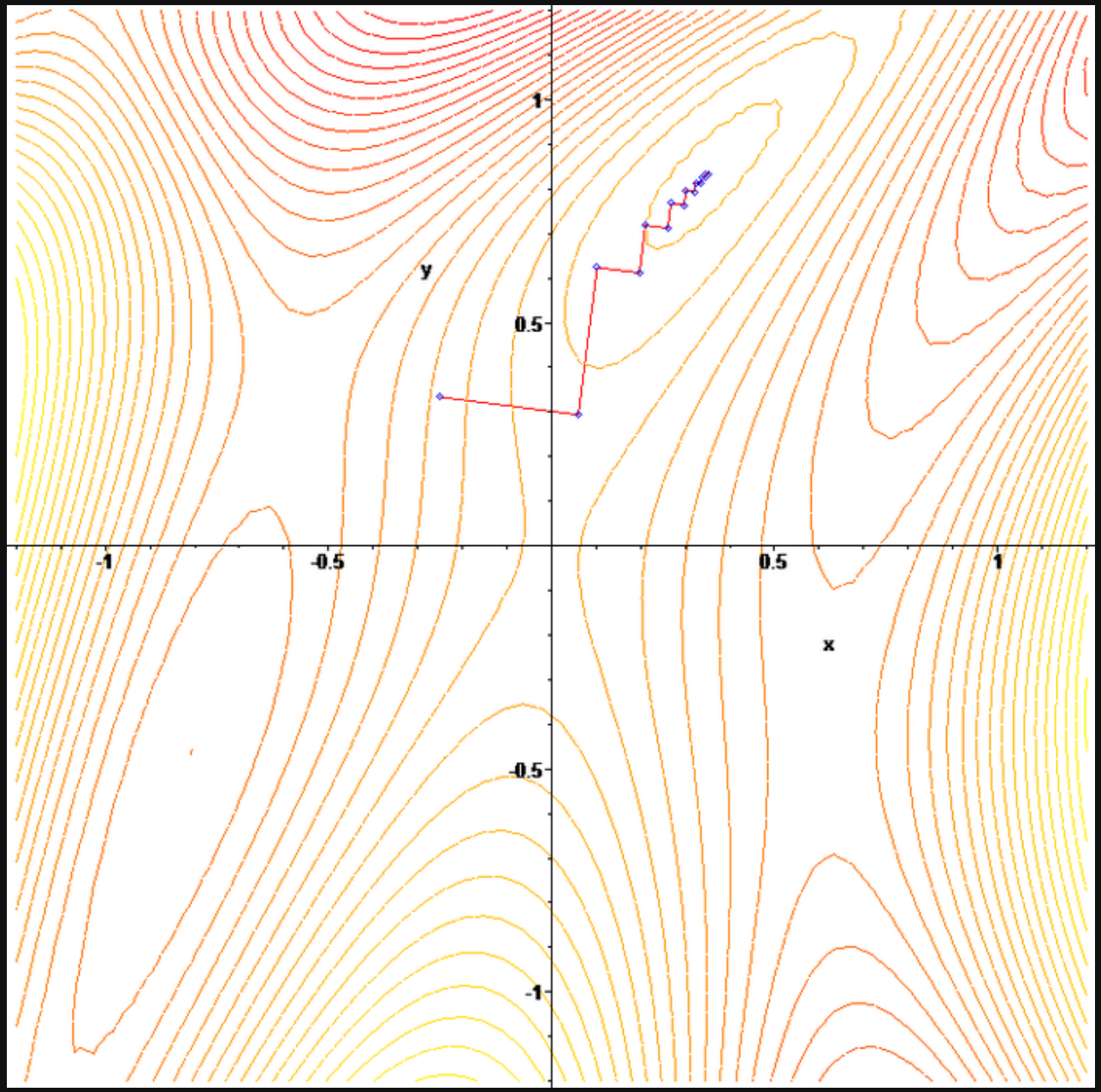
Since you're only using one training example, the path to the minima will be all zig-zag crazy

(Imagine trying to find the fastest way down a hill

only you can't see all of the curves in the hill)

May want to consider mini-batching rather than stochastic approach with very large datasets

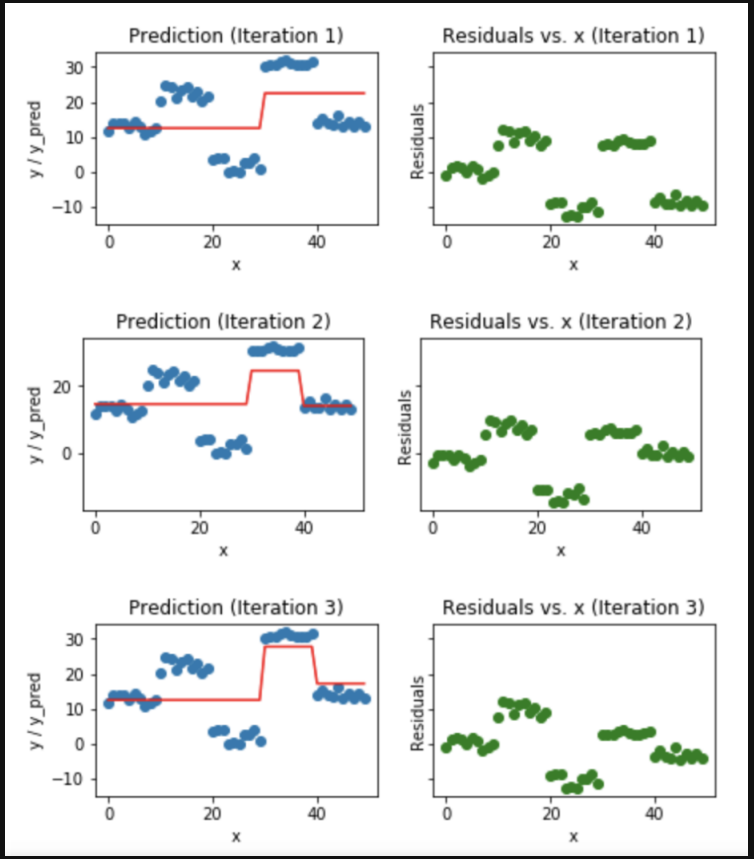

0.4
0.2
0
-0.2
-0.4
1
0.5
0
-0.5
-1
x
1
0.5
0
-0.5
-1
y

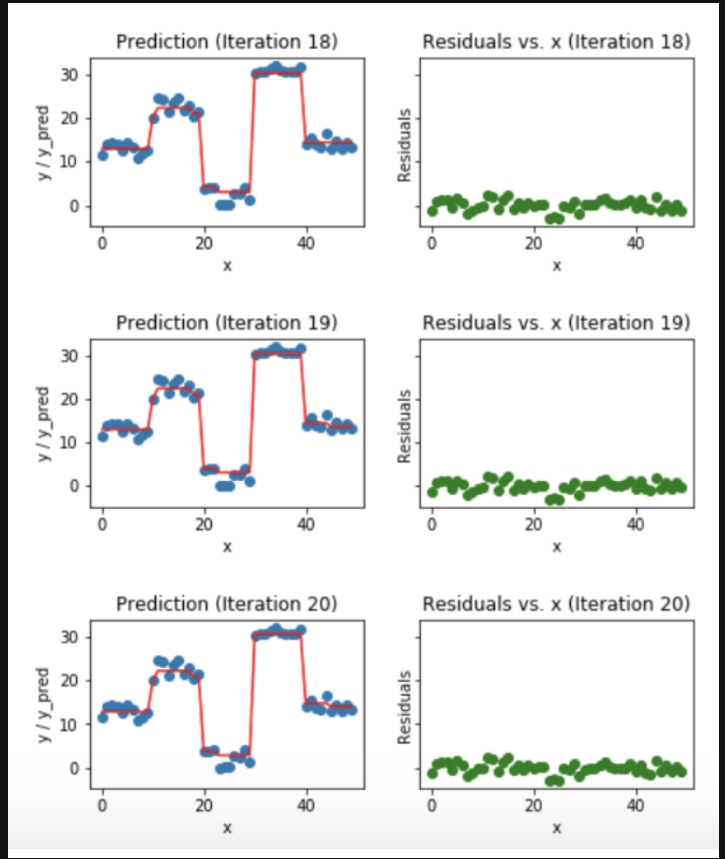

1
y
0.5
-1
-0.5
0.5
1
x
-0.5
-1

## Gradient boosting machine - Linear Regression Example

GBM can be configured to different base learners (e.g. tree, stump, linear model)

https://www.kaggle.com/grroverpr/gradient-boosting-simplified/

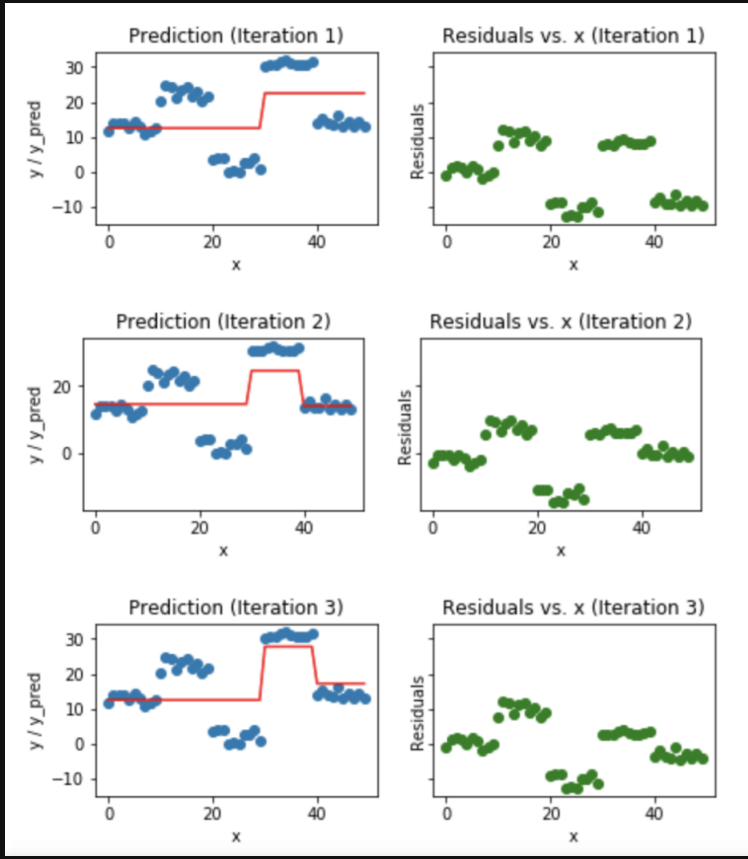


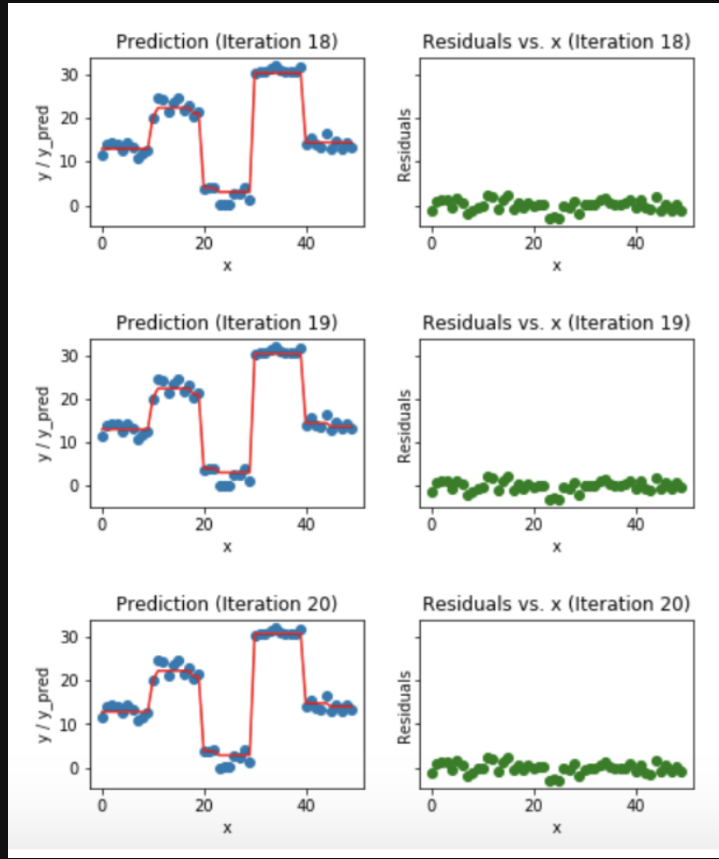




basic assumption: sum of residuals = 0

leverage pattern in residuals  to strengthen weak prediction model until residuals become randomly distributed

if you keep going you risk overfitting




Algorithmically we are minimizing our loss function such that the test loss reaches its minima

Adjusted our predictions using the fit on the residuals and accordingly adjusting value of alpha

We are doing supervised learning here

you can check for overfitting using a
k-fold cross validation

resampling procedure used to evaluate machine learning models on a limited data sample

# Pseudocode for a generic gradient boosting method

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations $M$.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For $m$ = 1 to $M$:

    1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

    2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
    3. Compute multiplier $\gamma_m$ by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

    4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

http://statweb.stanford.edu/~jhf/ftp/trebst.pdf ***MATH***

StackOverflow fixed my problems

https://bit.ly/2FwXUAF

(there are a lot of people who can help you if you're lost)

## Further...

- The probabiliy of GD to get stuck at a saddle is actually 0: arxiv.org/abs/1602.04915
- Presence of saddle points might severly slow GDs progress down: www.jmlr.org/proceedings/papers/v40/Ge15.pdf
- Lots on optimization: https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f
- Tools like H2O are great: http://www.h2o.ai/wp-content/uploads/2018/01/GBM-BOOKLET.pdf
- Learn about ranking: https://pdfs.semanticscholar.org/9b9c/4bf53eb680e2eb26b456c4752a23dafb2d5e.pdf
- Learning rates: https://www.coursera.org/learn/machine-learning/lecture/3iawu/gradient-descent-in-practice-ii-learning-rate
- Original work from 1999: https://statweb.stanford.edu/~jhf/ftp/trebst.pdf

# Please
# make your code citable

https://guides.github.com/activities/citable-code/