Boosting, Gradient Boosting

Why Does Grad. Boosting work?

CS1090A Introduction to Data Science

Pavlos Protopapas, Natesh Pillai, Chris Gumb

Jingfei Huang
Sêrtar, Sichuan, China

Outline

- Introduction to boosting
- Gradient Boosting
- **Mathematical Formulation - Gradient Boosting**

Why Does Gradient Boosting Work?

Intuitively, each simple model $T^{(i)}$ that we add to our ensemble T, models the residuals of T.

Thus, with each addition of $T^{(i)}$, the residuals are reduced.

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

Question: How?

Why Does Gradient Boosting Work?

In the previous example, we started off by modeling $T^{(0)}$ that gave us some predictions

$$T^{(0)}(x) = T^{(0)}_{pred}.$$

We then train $T^{(1)}$ on the residuals of $T^{(0)}$ such that $T^{(1)}_{pred} \approx \underbrace{y - T^{(0)}_{pred}.}_{r^{(0)}}$

Finally, we combined the models to form $T \leftarrow T^{(0)} + \lambda T^{(1)}$.

The residuals of this model would be:

$$\begin{aligned} y - T_{pred} &= y - T^{(0)}_{pred} - \lambda T^{(1)}_{pred} \\ &= r^{(0)} - \lambda T^{(1)}_{pred} \end{aligned}$$

Therefore, as we combine more models, the residuals keeps decreasing.

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

Why Does Gradient Boosting Work?

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

If we want to easily reason about convergence and how to choose λ and investigate the effect of λ on the model T, we need a bit more mathematical formalism.

In particular, how can we effectively descend through this optimization via an iterative algorithm?

We need to formulate gradient boosting as a type of **gradient descent**.

Review: A Brief Sketch of Gradient Descent

In optimization, when we wish to minimize a function, called the **objective function (or loss function)**, over a set of variables, we compute the partial derivatives of this function with respect to the variables.

If the partial derivatives are sufficiently simple, one could analytically find a common root - i.e., a point at which all the partial derivatives vanish; this is called a **stationary point.**

If the objective function has the property of being **convex**, then the stationary point is precisely the global minimum.

Review: A Brief Sketch of Gradient Descent

In practice, our objective functions are complicated, and analytically finding a stationary point is intractable.

Instead, we use an iterative method called gradient descent.

1. Initialize the variables at any value

$$x = [x_1, \dots, x_J]$$

2. Take the gradient of the objective function at the current variable values.

$$\nabla_x f(x) = \left[\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x), \dots, \frac{\partial f}{\partial x_J}(x) \right]$$

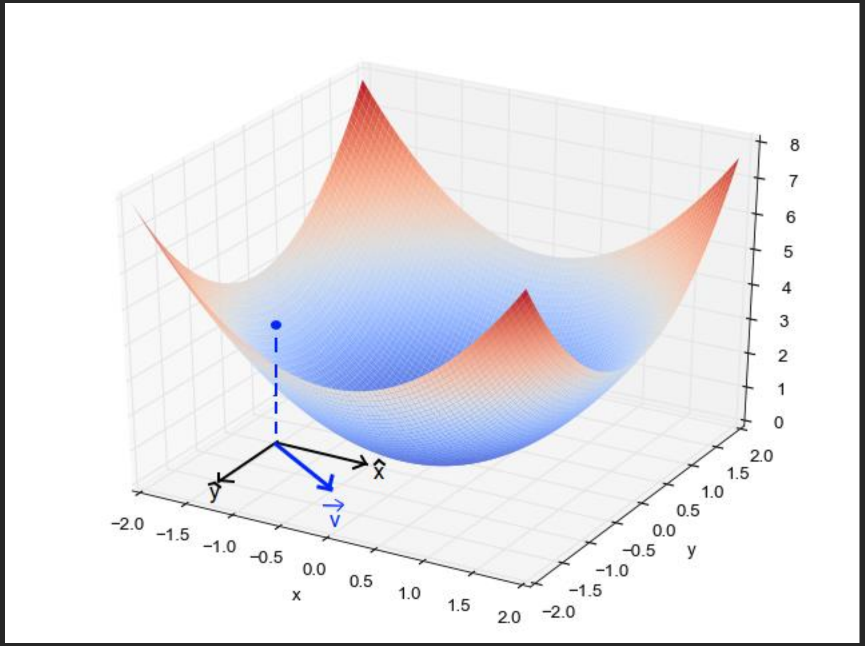
3. Adjust the variables by some negative multiple of the gradient.

$x \leftarrow x - \lambda \nabla f_x(x)$ where λ is the learning rate.

Why Does Gradient Descent Work?

Claim: If the function is convex, this iterative methods will eventually move x close enough to the minimum, for an appropriate choice of λ.

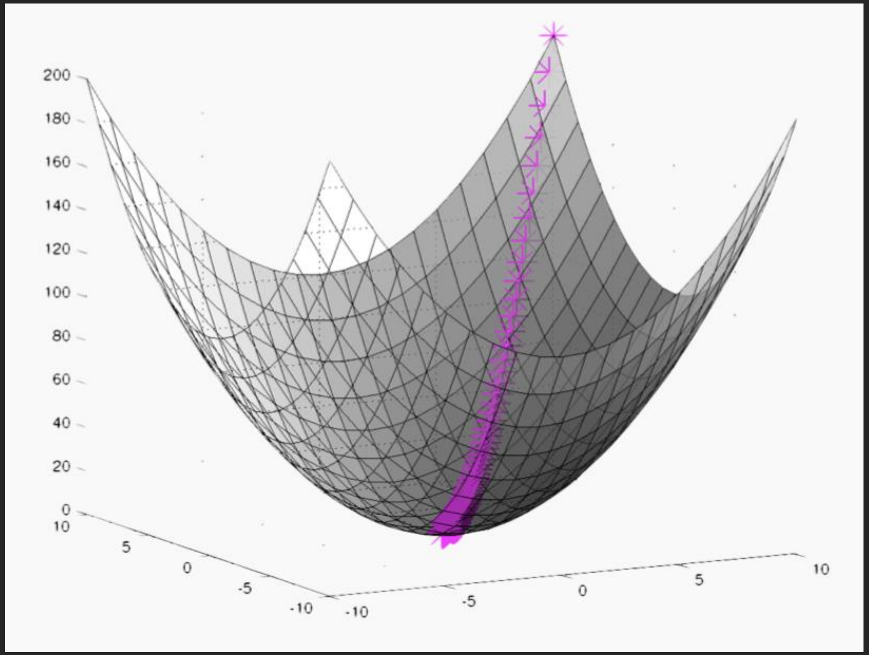
Why does this work? Recall, that as a vector, the gradient at at point gives the direction for the greatest possible rate of increase.



Why Does Gradient Descent Work?

Subtracting a λ multiple of the gradient from x, moves x in the opposite direction of the gradient (hence towards the steepest decline) by a step of size λ.

If f is convex, and we keep taking steps descending on the graph of f, we will eventually reach the minimum.

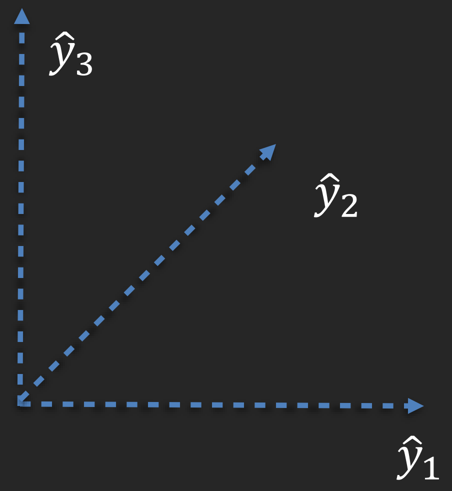


Gradient Boosting as Gradient Descent

Prediction space

Assume for now that we have 3 training examples $\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$.

Let us look at the space of predictions:



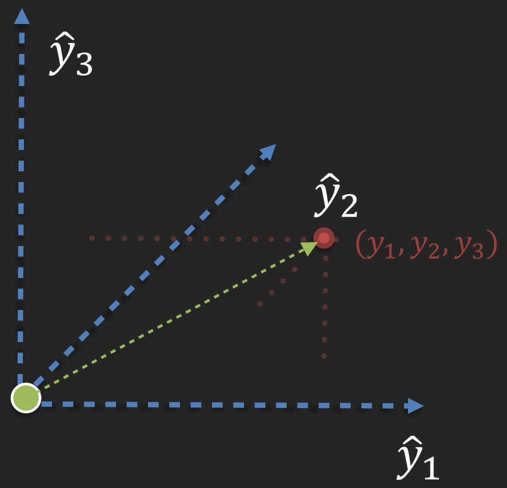
In this space, each point represents our set of predictions $y = (\hat{y}_1, \hat{y}_2, \hat{y}_3)$ for all datapoints. Note that here we have not assumed any relation between x and y whatsoever, we are just optimizing over the space of predictions.

The loss that we are optimizing for is the MSE:

$$L(y) = \frac{1}{3}[(y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + (y_3 - \hat{y}_3)^2]$$

Gradient Boosting as Gradient Descent

$$L(y) = \frac{1}{3}[(y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + (y_3 - \hat{y}_3)^2]$$



Here as the loss function is convex, we already know the unique minimizer $y^* = (y_1, y_2, y_3)$.

So if we start at the origin (or anywhere else), we will end at y^* when we optimize using gradient descent.

Gradient Boosting as Gradient Descent

More generally, we have:

$$\boxed{MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

Treating this as an optimization problem, we can try to directly minimize the MSE with respect to the predictions:

$$\begin{aligned}\nabla_{\hat{y}} MSE &= \left[\frac{\partial MSE}{\partial \hat{y}_1}, \dots, \frac{\partial MSE}{\partial \hat{y}_N}\right] \\ &= -2[y_1 - \hat{y}_1, \dots, y_N - \hat{y}_N] \\ &= -2[r_1, \dots, r_n]\end{aligned}$$

The update step for gradient descent would look like:

$$\boxed{\hat{y}_n \leftarrow \hat{y}_n + \lambda r_n, \qquad n = 1, \dots, N}$$

Gradient Boosting as Gradient Descent (cont.)

There are two reasons why minimizing the MSE with respect to $\hat{y}_n$'s – the predictions - is not interesting:

Minimizing a convex function

- We know where the minimum MSE occurs: $\hat{y}_n = y_n$, for every n.
- Learning sequences of predictions, $\{\hat{y}_n^{(1)} \ldots \hat{y}_n^{(i)} \ldots\}$ does not produce a model. This is because we have, by no means, learned to map predictors to the prediction.

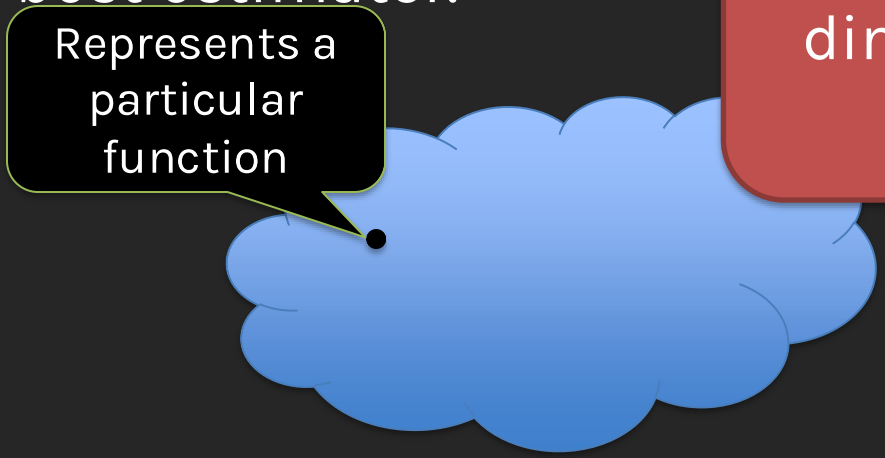
Here "i" is the iteration num (epoch)

Gradient Boosting as Gradient Descent

When trying to view Gradient Boosting as a form of Gradient Descent, we need to look at the function space.

Imagine a space where each point represents a function $f: X \rightarrow y$, then ideally, we want to optim this function space to fi best estimator.

But we cannot do computation on infinite dimensional abstract spaces


Represents a particular function

Abstract function space

Gradient Boosting as Gradient Descent

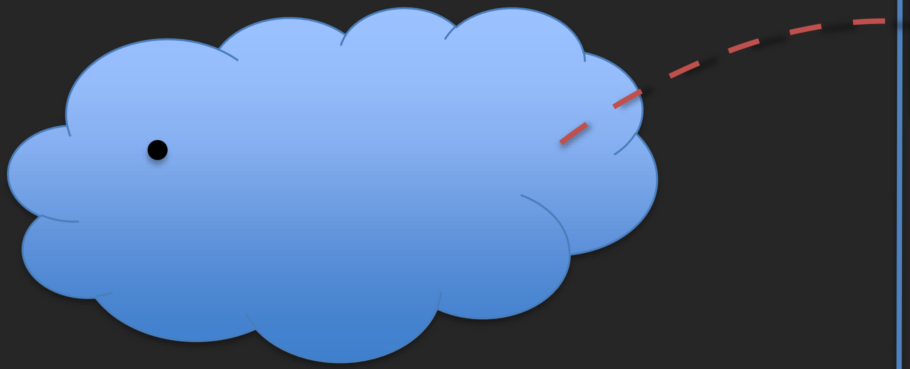
When trying to view Gradient Boosting as a form of Gradient Descent, we need to look at the function space.

Imagine a space where each point represents a function $f: X \rightarrow y$, then ideally, we want to optimize over this function space to find out the best estimator.

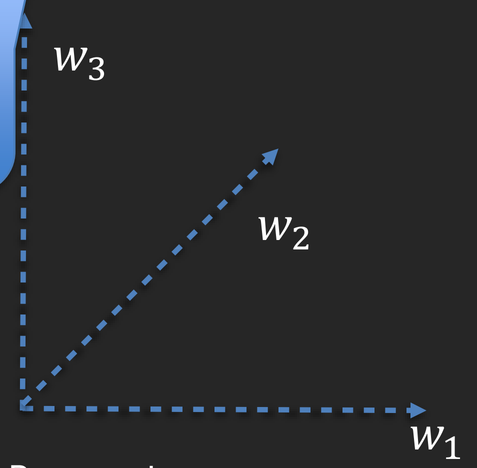
In case of **parametric models**, we optimize our model in the weight space (parameters of our model)

$$\hat{f} \coloneqq [w_1\ w_2\ \ldots\ w_N]$$

Means "$\hat{f}$ is represented by"



Abstract function space



Parameter space

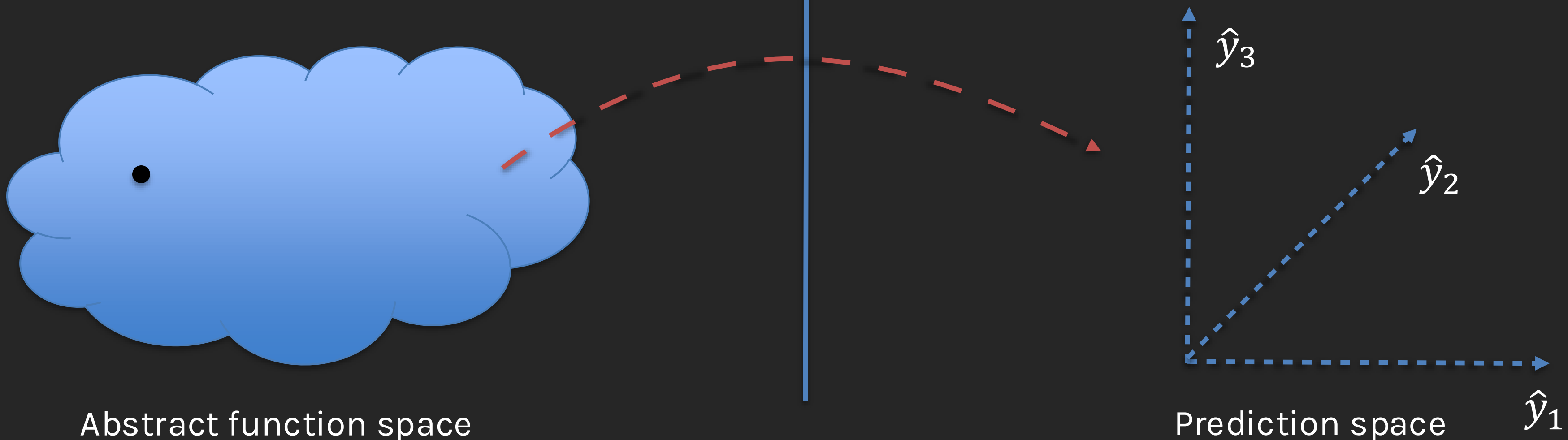
Gradient Boosting as Gradient Descent

When trying to view Gradient Boosting as a form of Gradient Descent, we need to look at the function space.

Imagine a space where each point represents a function $f: X \rightarrow y$, then ideally, we want to optimize over this function space to find out the best estimator.

What about **non-parametric** models? In boosting, we optimize in the prediction space of the model.

$$\hat{f} \coloneqq [\hat{f}(x_1), \hat{f}(x_2), \ldots, \hat{f}(x_n)]$$



Abstract function space

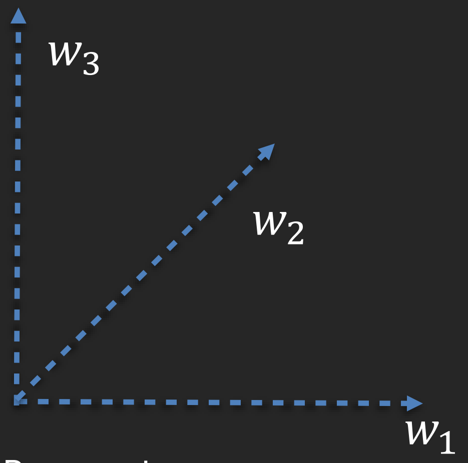
Prediction space

Gradient Boosting as Gradient Descent

In summary, for parametric models (say neural nets), gradient descent is done in the parameter space. Boosting can be thought of as doing gradient descent in the prediction space.

Parametric regime

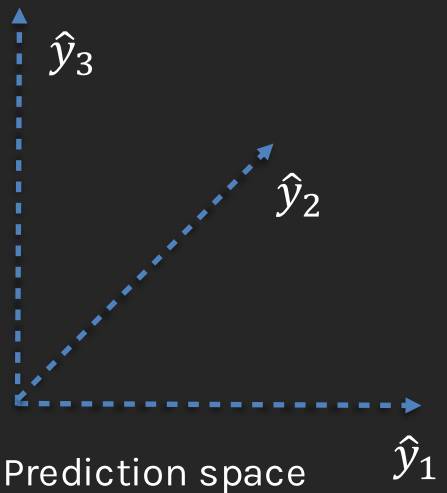
$$\hat{f} := [w_1,\ w_2,\ \ldots,\ w_N]$$



Parameter space

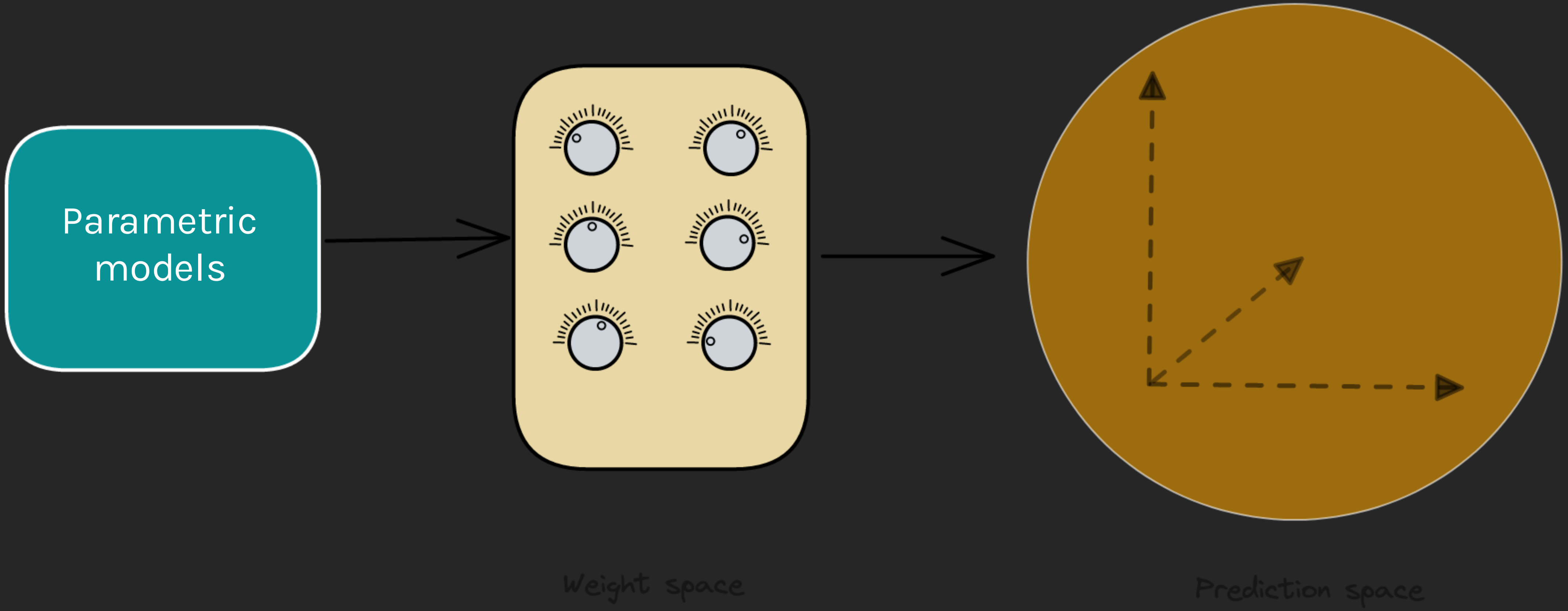
Non-parametric regime

$$\hat{f} := [\hat{f}(x_1)\ , \hat{f}(x_2)\ , \ldots , \hat{f}(x_n)]$$



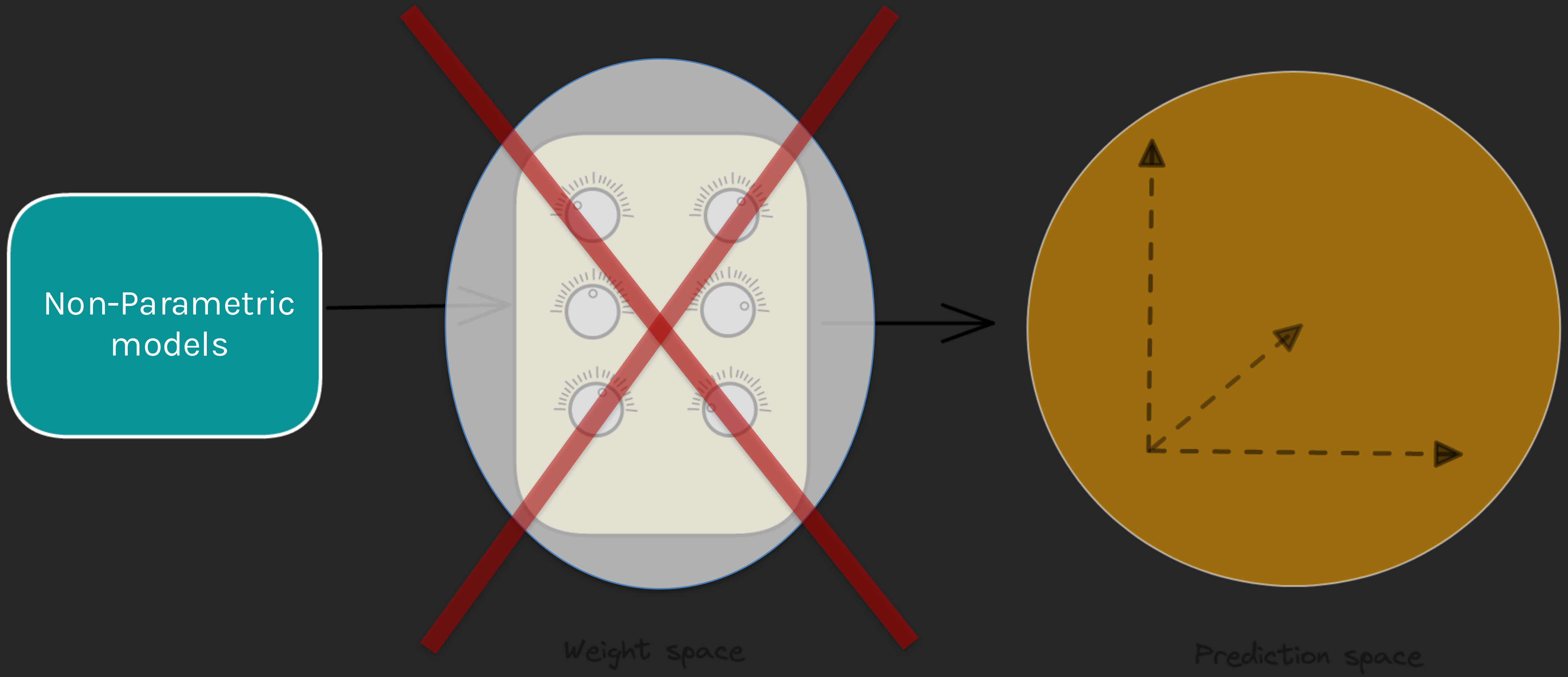
Prediction space

Gradient Boosting as Gradient Descent

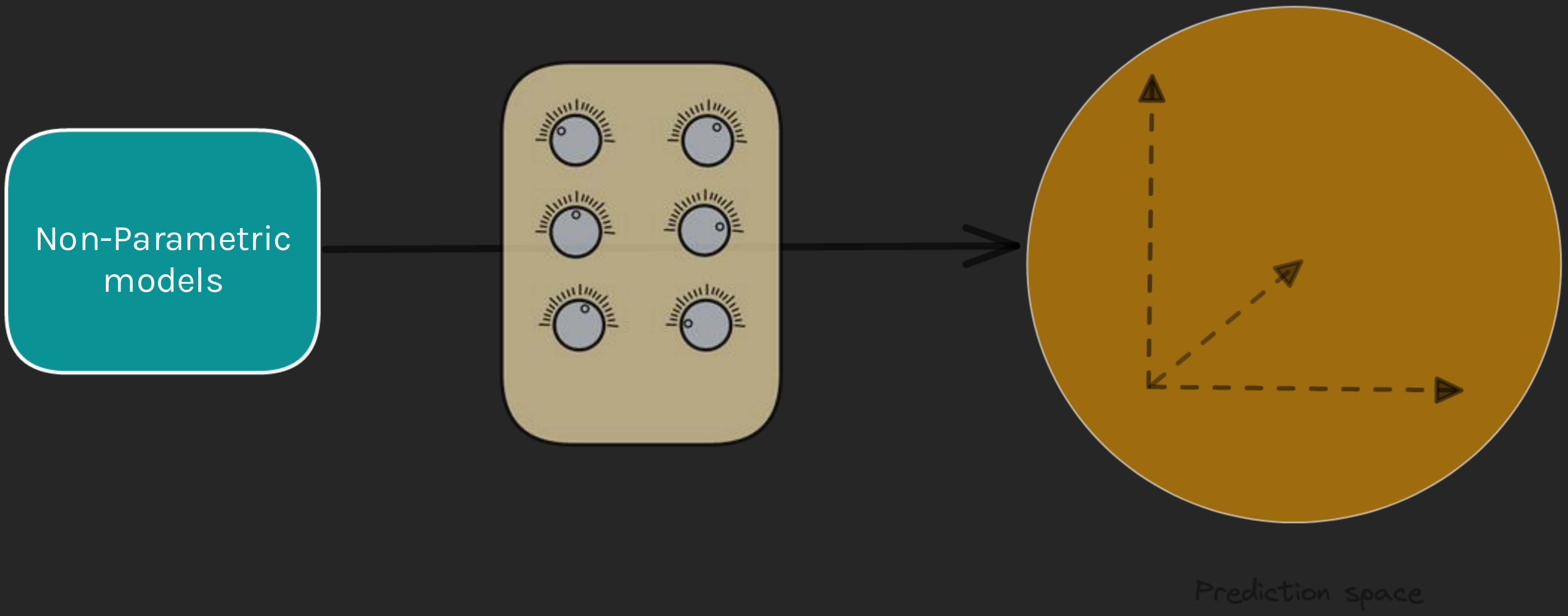


Parametric models

Gradient Boosting as Gradient Descent

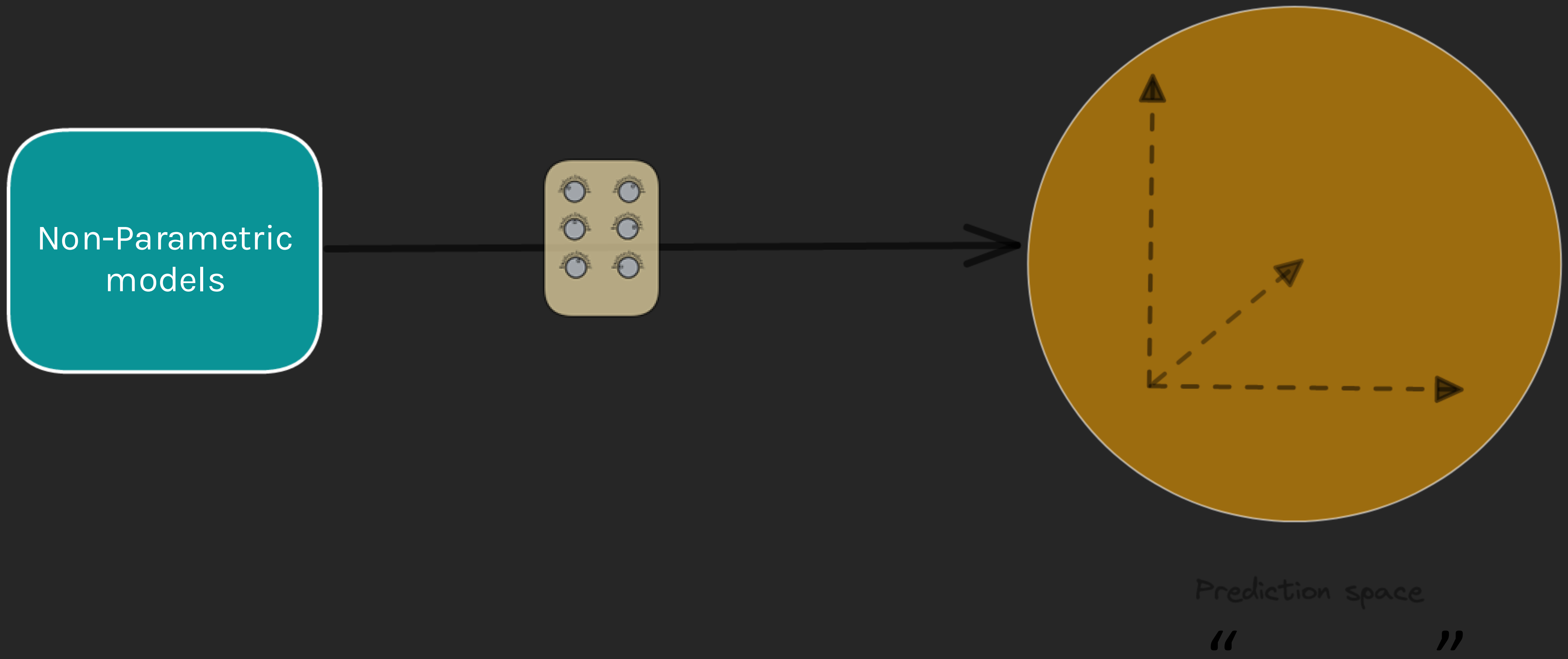


Gradient Boosting as Gradient Descent



Non-Parametric models

Gradient Boosting as Gradient Descent



Non-Parametric models

Gradient Boosting as Gradient Descent

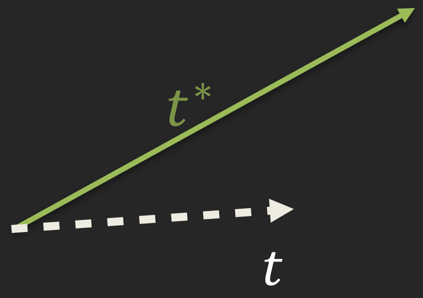
Let our current estimate be $T^{(n)}(x)$. In the prediction space, this function is just a vector of the prediction values. Then the optimal direction to take a step using Gradient Descent is along the negative gradient.

$$
\begin{aligned}
t^* &= -\frac{\partial \mathcal{L}}{\partial T^{(n)}(x)} \\
&= -\left[\frac{\partial \mathcal{L}}{\partial \hat{y}_1}, \frac{\partial \mathcal{L}}{\partial \hat{y}_2}, \ldots, \frac{\partial \mathcal{L}}{\partial \hat{y}_N}\right] \\
&= -[(\hat{y}_1 - y_1)], \ldots, (\hat{y}_N - y_N)] \\
&= 2[r_1, r_2, \ldots, r_N]
\end{aligned}
$$

Hence, the optimal direction to step in along the residuals.

Gradient Boosting as Gradient Descent

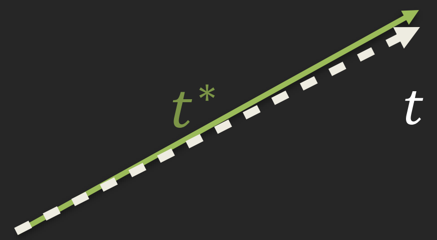
If we are restricting to a simple family of functions, we might not be able to always take a step in this optimal direction t^*. But we want to take as close a step to this direction as possible. Let $\mathbb{H}$ be a simple family of models (say Decision Trees of `max_depth=5`).



Then we need to find some $t \in \mathbb{H}$ such that $t = [t(x_1), t(x_2), \dots, t(x_N)]$ is as close to $t^* = [r_1, r_2, \dots, r_N]$ as possible. Well, this is quite easy to do if we train another model in $\mathbb{H}$ on the training set $\{(x_1, r_1), (x_2, r_2), \dots, (x_N, r_N)\}$.

Gradient Boosting as Gradient Descent

NOTE: If we were **not** restricting to a simple family of models (like using an unrestricted decision tree), we could directly take a step in the direction of the residuals and end up at y^*:

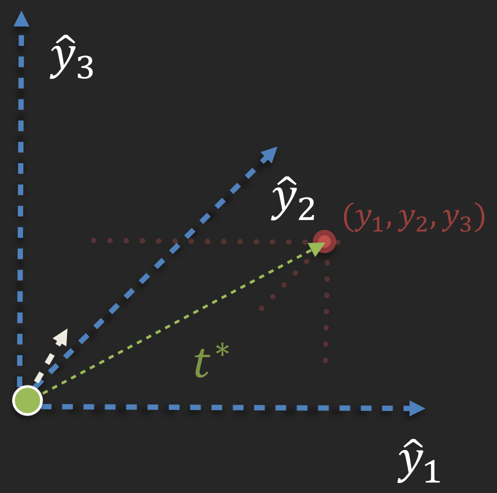


But this would mean we are completely overfitting on the training set. However, our goal is to decrease the bias without increasing the variance.

Gradient Boosting as Gradient Descent

How does this pan out graphically?

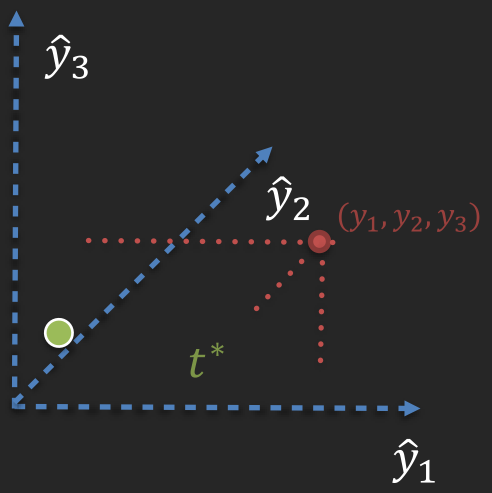
1. Say we start at the origin.
2. The optimal direction is along the residual vector (green line).
3. Because we restrict to a simple family of models, our approximations of the residuals are not exact and hence we take a step close to that direction.



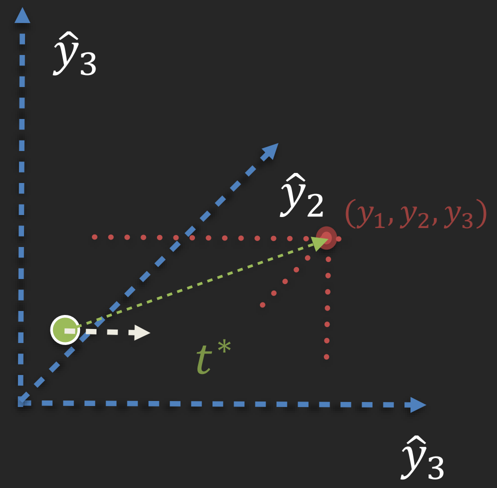
Gradient Boosting as Gradient Descent

How does this pan out graphically?

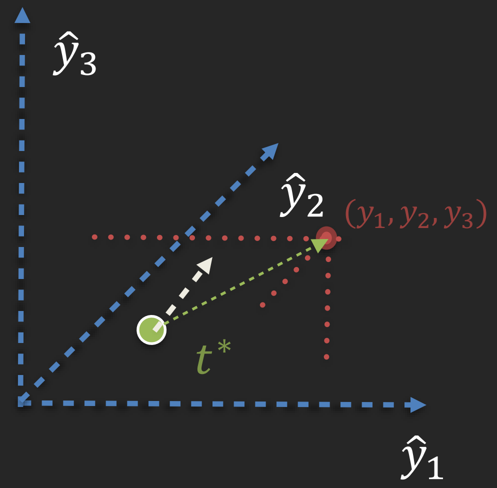
1. Say we start at the origin.
2. The optimal direction is along the residual vector (green line).
3. Because we restrict to a simple family of models, our approximations of the residuals are not exact and hence we take a step close to that direction.



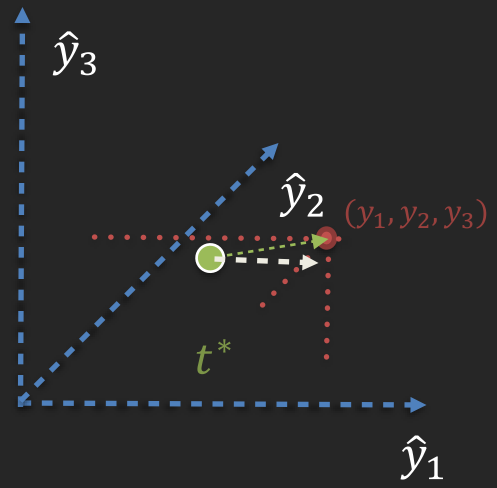
Gradient Boosting as Gradient Descent



Gradient Boosting as Gradient Descent



Gradient Boosting as Gradient Descent



Gradient Boosting as Gradient Descent

Hence instead of using the gradient (the residuals), we use an **approximation** of the gradient that depends on the predictors:

$$\boxed{\hat{y}_n \leftarrow \hat{y}_n + \lambda \hat{r}_n(x_n) \qquad \text{where } n = 1, 2, \dots, N}$$

In gradient boosting, we use a simple model $\hat{r}_n(x_n)$, to approximate the residuals $r_n(x_n)$ in each iteration (which is the negative gradient of the loss in the prediction space).

Technical note: Gradient boosting is descending in a space of models or functions mapping x_n to y_n !

Gradient Boosting as Gradient Descent (cont.)

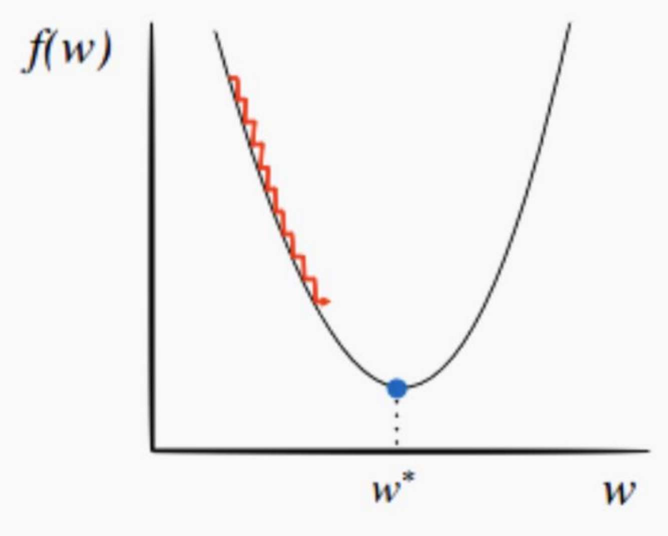
But why do we want to connect gradient boosting to gradient descent?

By making this connection, we can import the massive amount of techniques for studying gradient descent to analyze gradient boosting.

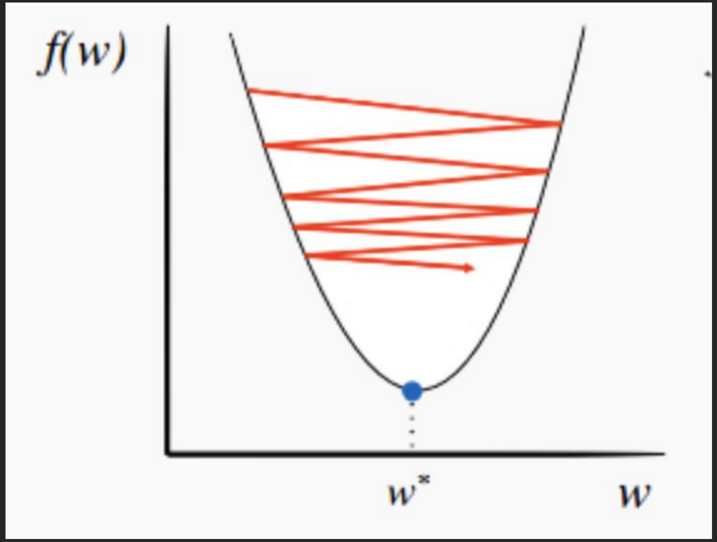
For example, we can easily reason about how to choose the learning rate λ in gradient boosting.

Choosing a Learning Rate

For a constant learning rate λ:



If λ is too small, it takes too many iterations to reach the optimum.



If λ is too large, the algorithm may ‘bounce’ around the optimum and never get sufficiently close.

Choosing a Learning Rate

We will learn this later:
RMSProp and Adam?

Choosing λ:

- If λ is a constant, then it should be tuned through cross validation.
- For better results, use a variable λ. That is, let the value of λ depend on the gradient

$$\lambda = h\left(\left|\left|\nabla f(x)\right|\right|\right)$$

where $\nabla f(x)$ is the magnitude of the gradient. So

- around the optimum, when the gradient is small, λ should be small
- far from the optimum, when the gradient is large, λ should be larger

Termination

Under ideal conditions, gradient descent iteratively approximates and converges to the optimum.

When do we terminate gradient descent?

- We can limit the number of iterations in the descent. But for an arbitrary choice of maximum iterations, we cannot guarantee that we are sufficiently close to the optimum in the end.

- If the descent is stopped when the updates are sufficiently small (e.g., the residuals of T are small), we encounter a new problem: the algorithm may never terminate!

Thank you