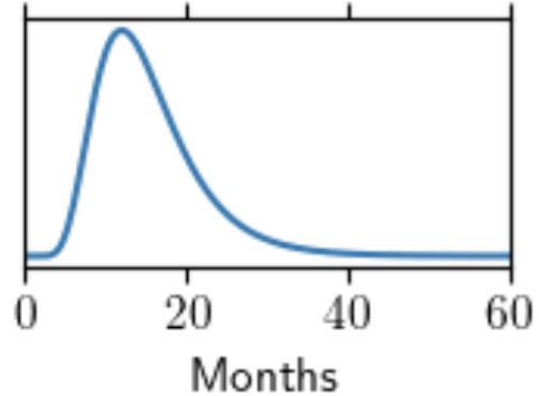


NGBoost: Natural Gradient Boosting for Probabilistic Prediction

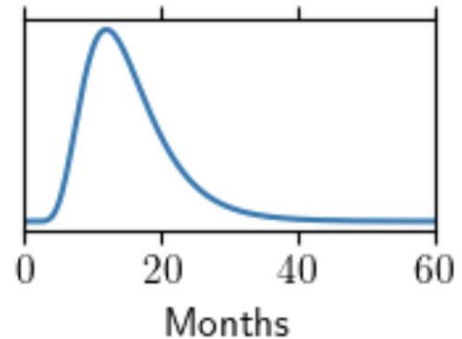
Authors: Tony Duan, Anand Avati, Daisy Ding, Khanh K. Thai,
Sanjay Basu, Andrew Ng, Alejandro Schuler

Presented By: Chella Thiyagarajan N (ME17B179)

Probabilistic Prediction

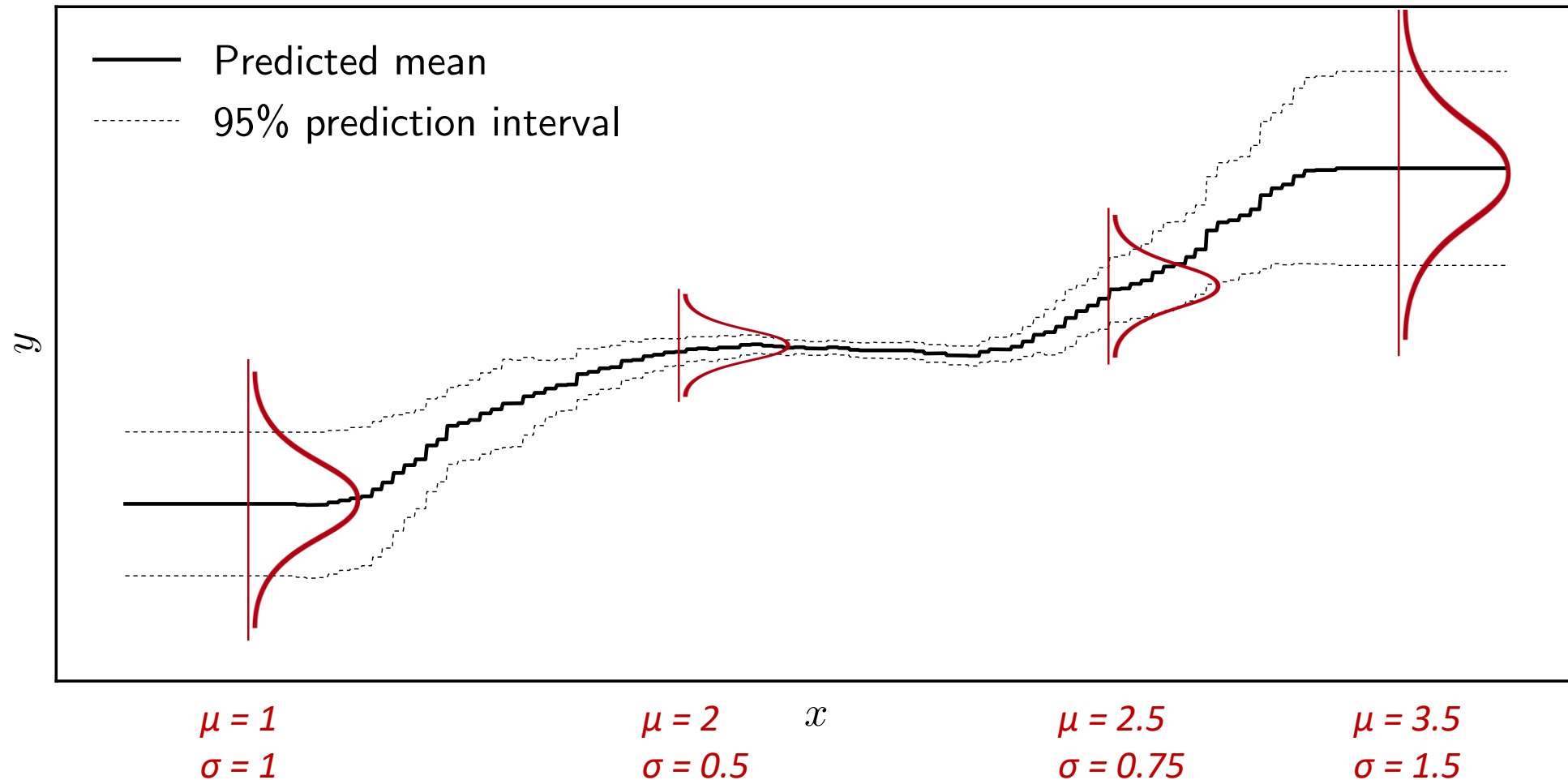
Question	Point Prediction (No uncertainty estimate)	Probabilistic Prediction (Uncertainty is implicit)
How long will this patient live?	11.3 months	
$X=x$	$E[Y X=x]$	$P(Y X=x)$

Probabilistic prediction is already standard for *classification*

	Question	Point Prediction (No uncertainty estimate)	Probabilistic Prediction (Uncertainty is implicit)
Regression	How long will this patient live?	11.3 months	
Classification	Will this patient get sepsis during this hospitalization?	no	$P(\text{no}) = 0.87$ $P(\text{yes}) = 0.13$

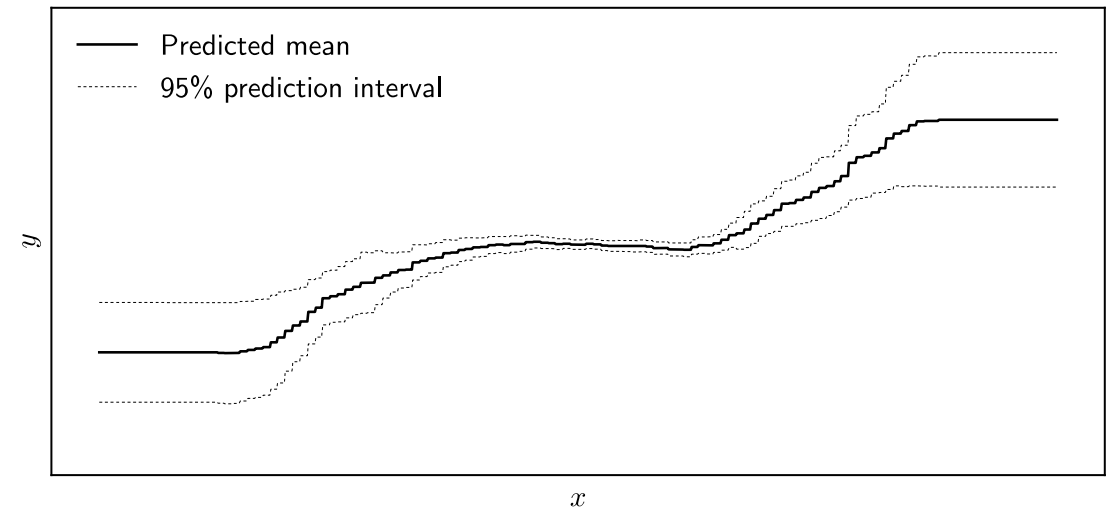
Assume $P(Y/X=x)$ has some **parametric form**

$$Y|X = x \sim \mathcal{N}(\mu(x), \sigma(x))$$

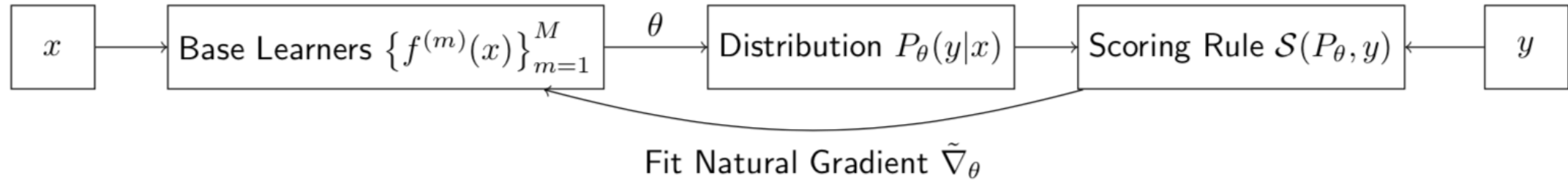


Existing probabilistic methods for *regression* are slow, inflexible, don't scale, and hard to use

- Post-hoc variance
 - Assumes homoscedasticity
- GAMLSS
 - Linear mean model
- Bayesian methods
 - MCMC needed for complex models
- Bayesian deep learning
 - Requires expertise



Outline of our approach



Pick a **scoring rule** to compare probabilistic predictions to ground truth

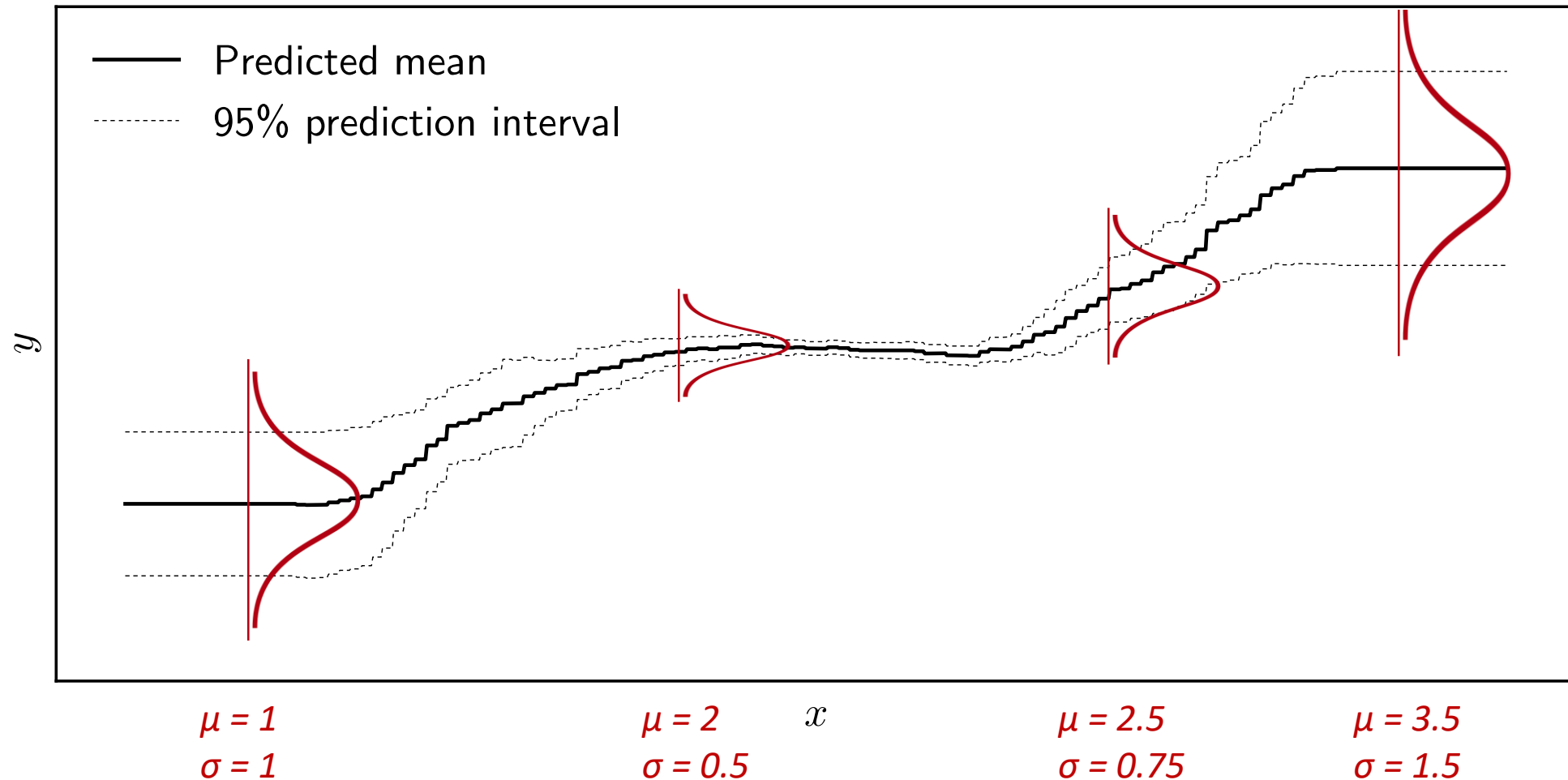
Point Prediction	Loss Function	$L(\hat{y}, y)$
Probabilistic Prediction	Scoring Rule	$S(\hat{P}, y)$

Example scoring rule: negative log-likelihood

$$S(\hat{P}, y) = -\log \hat{P}(y)$$

Assume $P(Y/X=x)$ has some **parametric form**

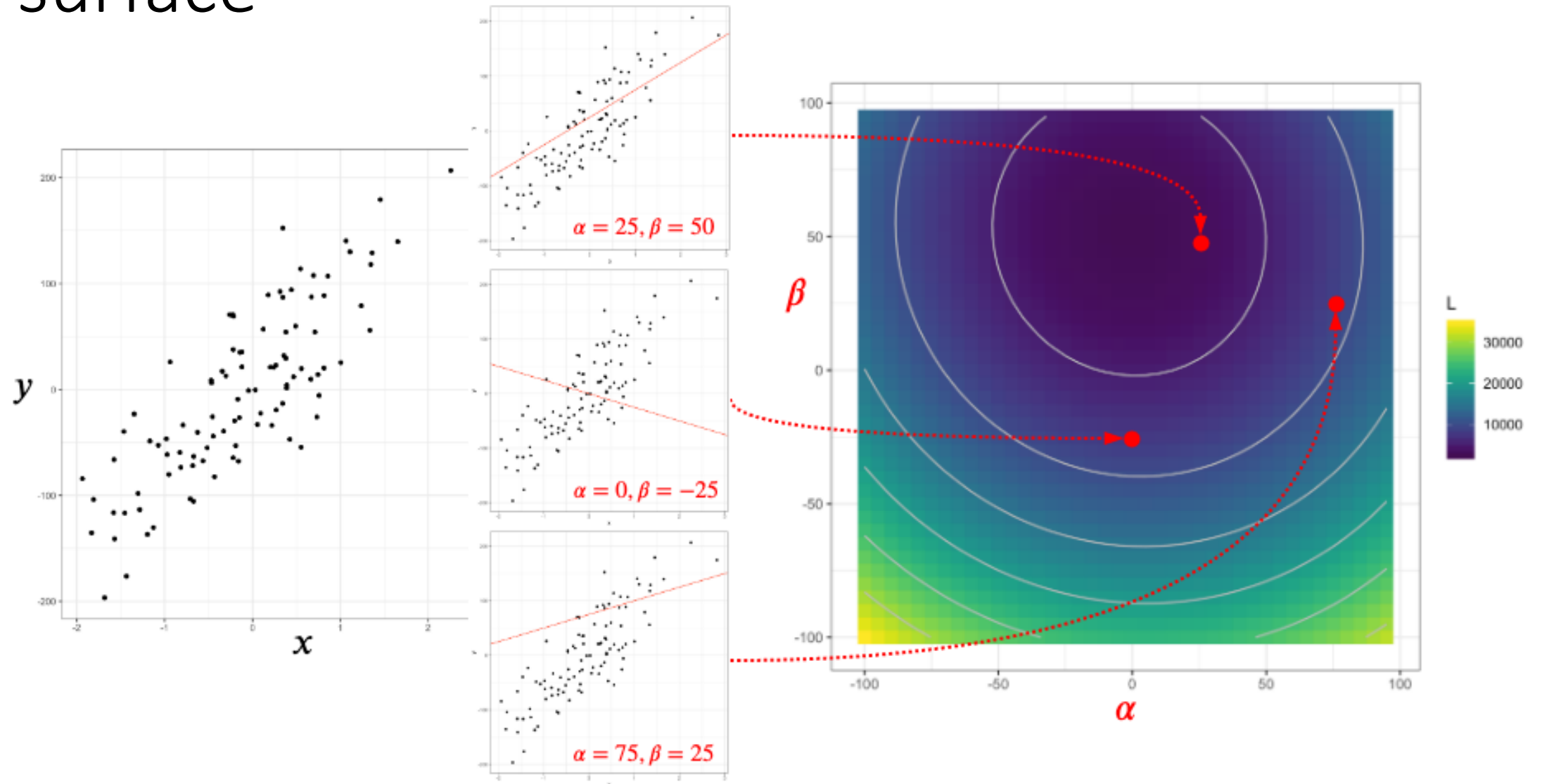
$$Y|X = x \sim \mathcal{N}(\mu(x), \sigma(x))$$



Use **gradient boosting** to estimate the parameters as a function of x

1. Review gradient descent for parametric models
2. Understand how boosting is gradient descent in functional space with an approximated gradient
3. Understand how gradient boosting can be used to fit probabilistic regression models

Gradient descent for a parametric model: the loss surface

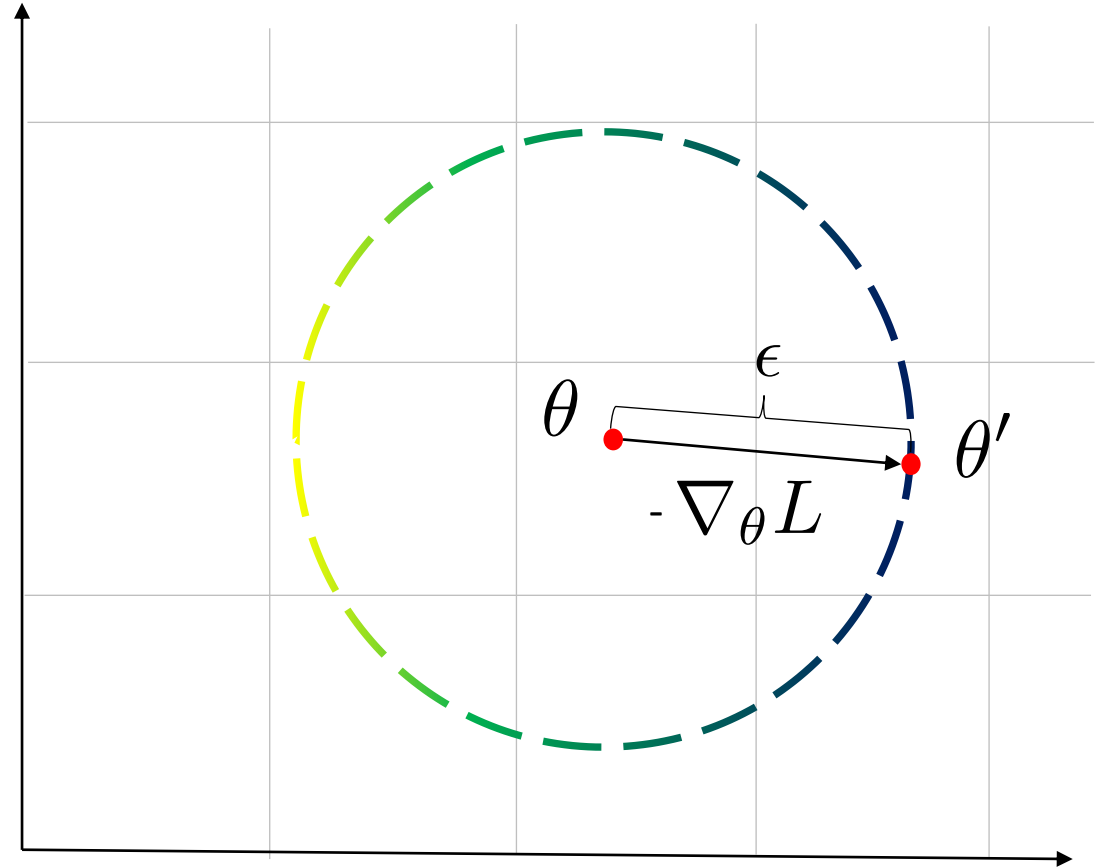


Gradient descent for a parametric model: the gradient

$$\nabla_{\theta} L = \lim_{\epsilon \rightarrow 0} \operatorname{argmax}_{d(\theta, \theta') < \epsilon} L(\theta')$$

- In other words, the derivative we know how to compute from calculus happens to tell us which way to adjust the parameters

$$\theta_{b+1} = \theta_b - \epsilon \nabla_{\theta} L$$

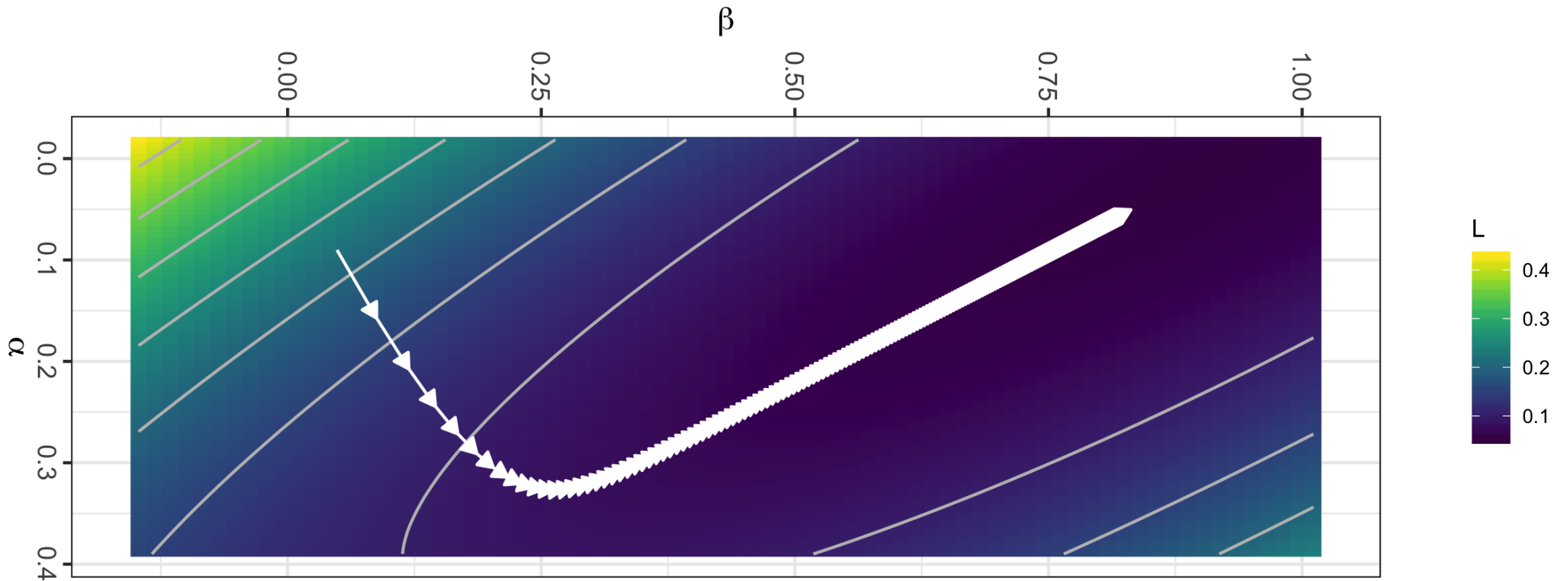


More generally,

$$\begin{aligned}\hat{y}_i &= f(x_i, \theta_1, \theta_2, \dots, \theta_p) \\ L(y, \hat{y}) &= \sum_i l(y_i, \hat{y}_i)\end{aligned}$$

$$\nabla_{\theta} L = \begin{bmatrix} \frac{dL}{d\theta_1} \\ \frac{dL}{d\theta_2} \\ \vdots \\ \frac{dL}{d\theta_p} \end{bmatrix} = \begin{bmatrix} \sum_i \frac{dL}{d\hat{y}_i} \frac{d\hat{y}_i}{d\theta_1} \\ \sum_i \frac{dL}{d\hat{y}_i} \frac{d\hat{y}_i}{d\theta_2} \\ \vdots \\ \sum_i \frac{dL}{d\hat{y}_i} \frac{d\hat{y}_i}{d\theta_p} \end{bmatrix}$$

Gradient descent for a parametric model



Assuming a parametric form is restrictive

$$\hat{y}_i = f(x_i, \theta_1, \theta_2, \dots, \theta_p)$$

Boosting idea: the prediction itself is the parameter

$$\hat{y}_i = \theta(x_i)$$

The gradient is even easier to derive now. For example, with squared-error loss,

$$L(y, \hat{y}) = \sum_i (y_i - \theta(x_i))^2$$

$$\frac{dL}{d\theta(x_i)} = 2(y_i - \theta(x_i))$$

The gradient is even easier to derive now. For example, with squared-error loss,

$$L(y, \hat{y}) = \sum_i (y_i - \theta(x_i))^2$$

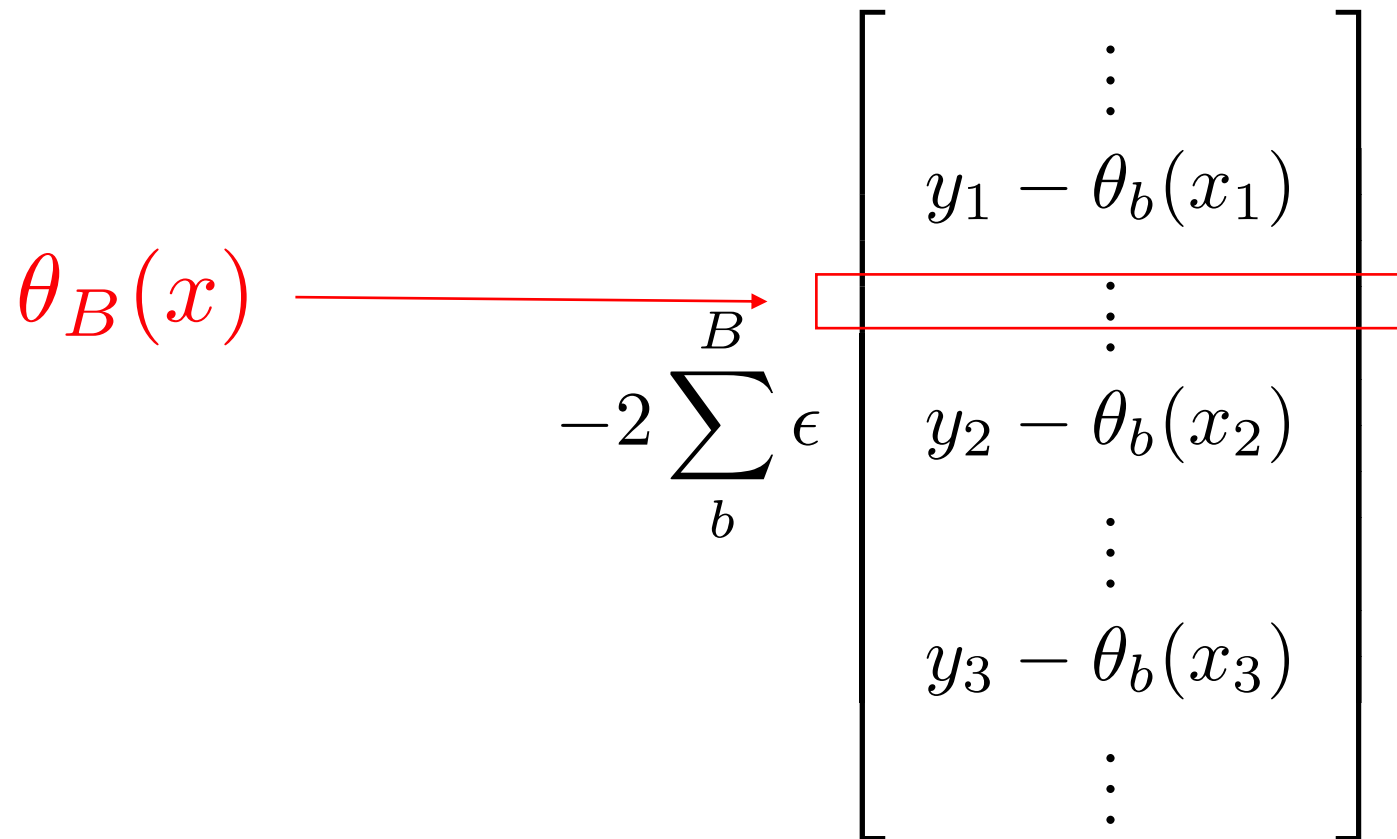
$$\frac{dL}{d\theta(x_i)} = 2(y_i - \theta(x_i))$$

$$\nabla_{\theta} L = 2 \begin{bmatrix} \vdots \\ y_1 - \theta(x_1) \\ \vdots \\ y_2 - \theta(x_2) \\ \vdots \\ y_3 - \theta(x_3) \\ \vdots \end{bmatrix}$$

If we start with a model $\theta_0(x) = 0$ for all x and take B steps, the final model is

$$-2 \sum_b^B \epsilon \begin{bmatrix} \vdots \\ y_1 - \theta_b(x_1) \\ \vdots \\ y_2 - \theta_b(x_2) \\ \vdots \\ y_3 - \theta_b(x_3) \\ \vdots \end{bmatrix}$$

What happens when we try and predict at a new value of x ?

$$\theta_B(x) \xrightarrow{-2 \sum_b^B \epsilon} \begin{bmatrix} \vdots \\ y_1 - \theta_b(x_1) \\ \vdots \\ y_2 - \theta_b(x_2) \\ \vdots \\ y_3 - \theta_b(x_3) \\ \vdots \end{bmatrix}$$


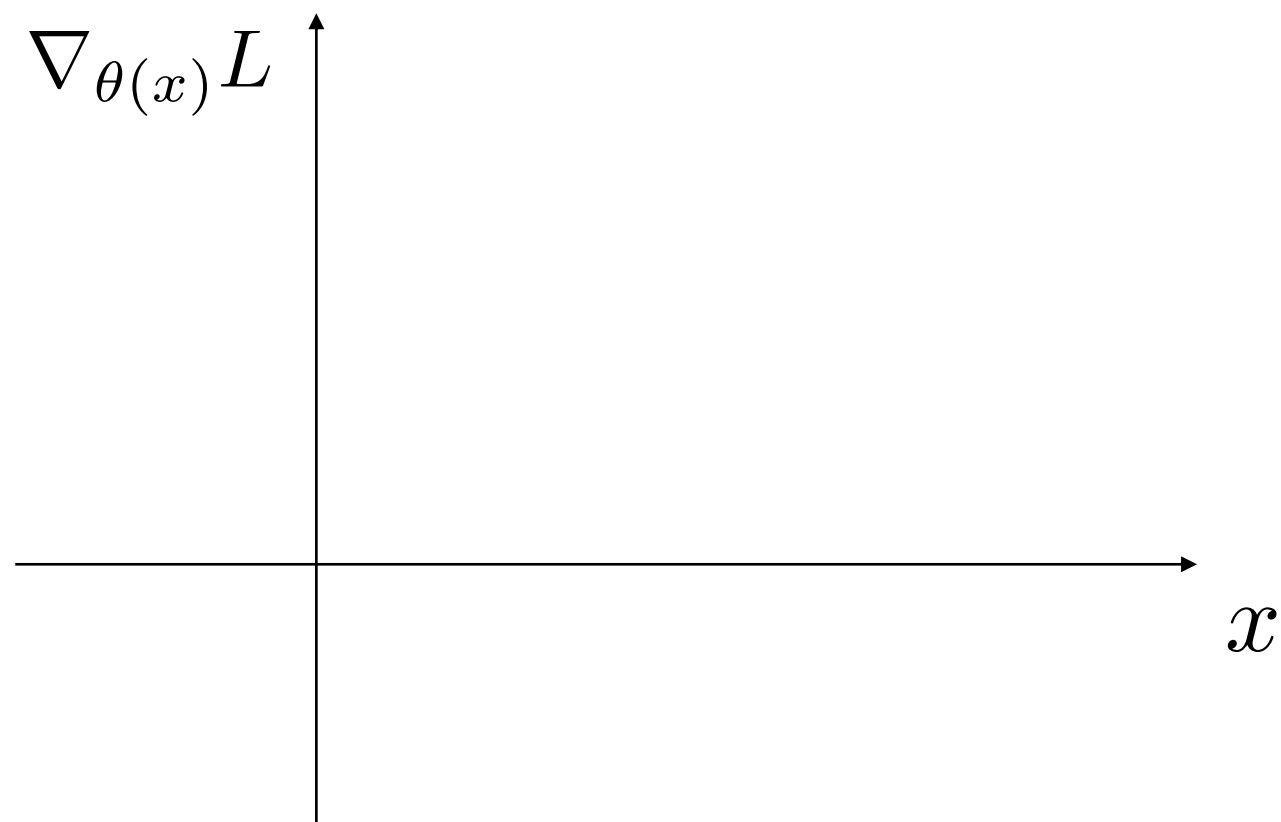
- We never saw y here, so we can't compute what the gradient would be at any iteration!

Gaps in the gradient

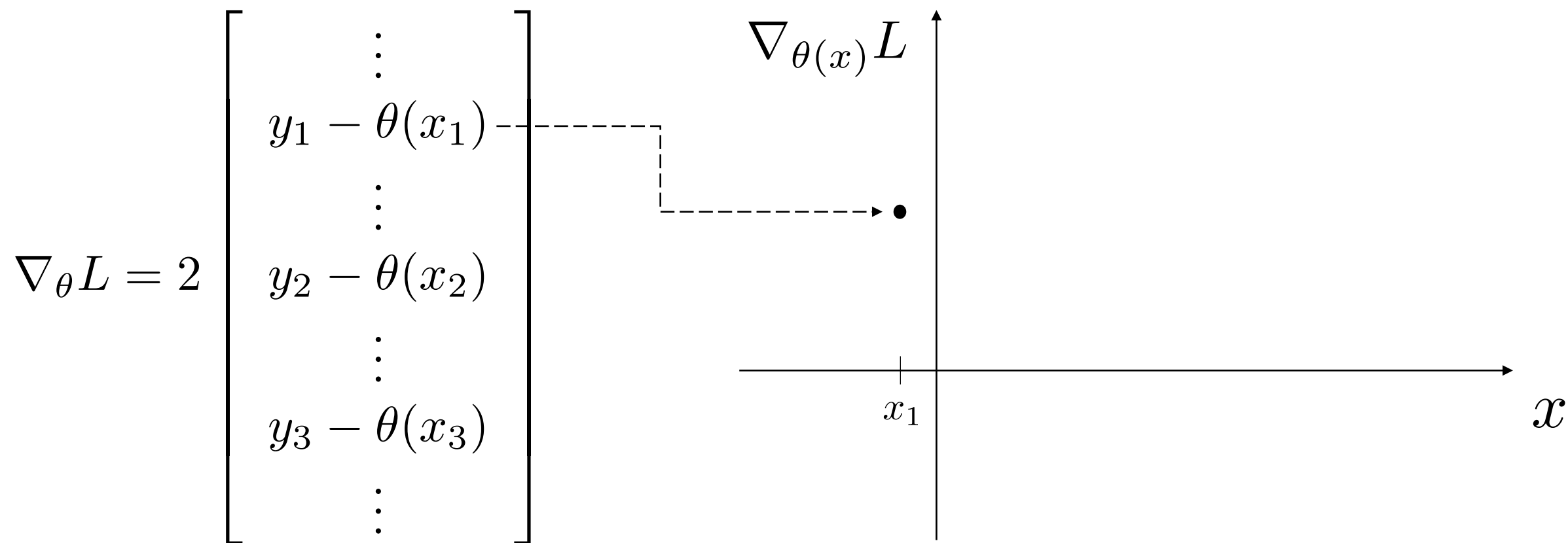
$$\nabla_{\theta} L = 2 \begin{bmatrix} \vdots \\ y_1 - \theta(x_1) \\ \vdots \\ y_2 - \theta(x_2) \\ \vdots \\ y_3 - \theta(x_3) \\ \vdots \end{bmatrix}$$

Gaps in the gradient

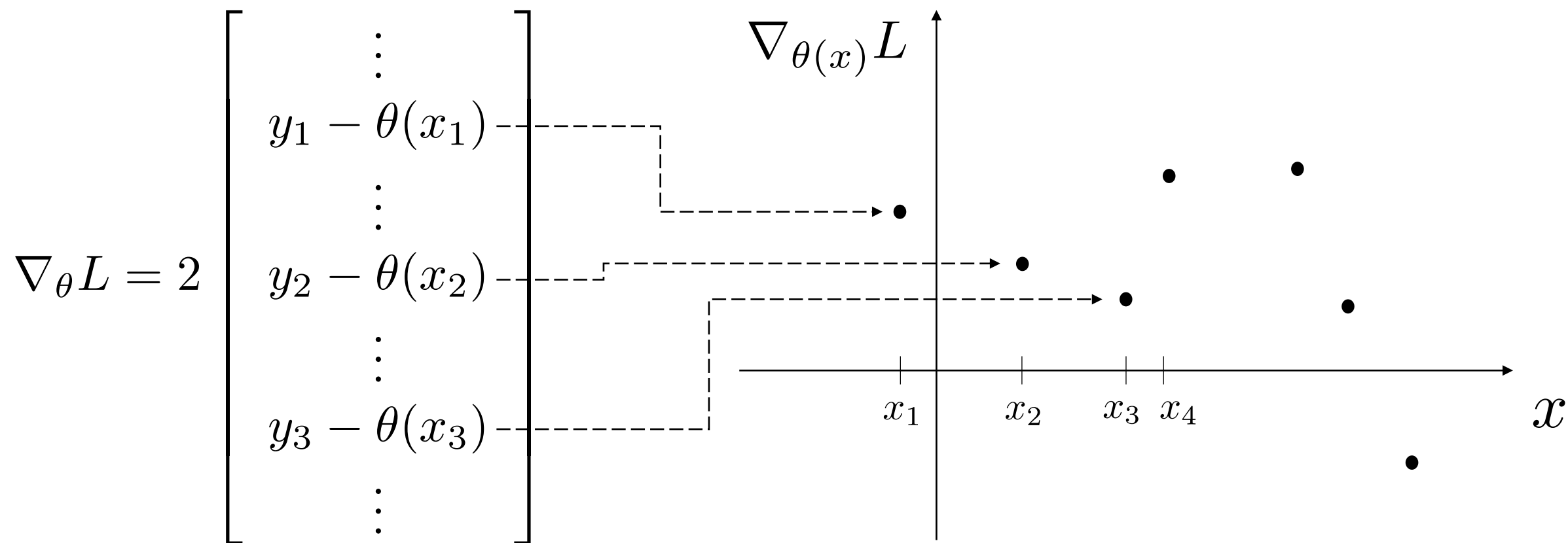
$$\nabla_{\theta} L = 2 \begin{bmatrix} \vdots \\ y_1 - \theta(x_1) \\ \vdots \\ y_2 - \theta(x_2) \\ \vdots \\ y_3 - \theta(x_3) \\ \vdots \end{bmatrix}$$



Gaps in the gradient

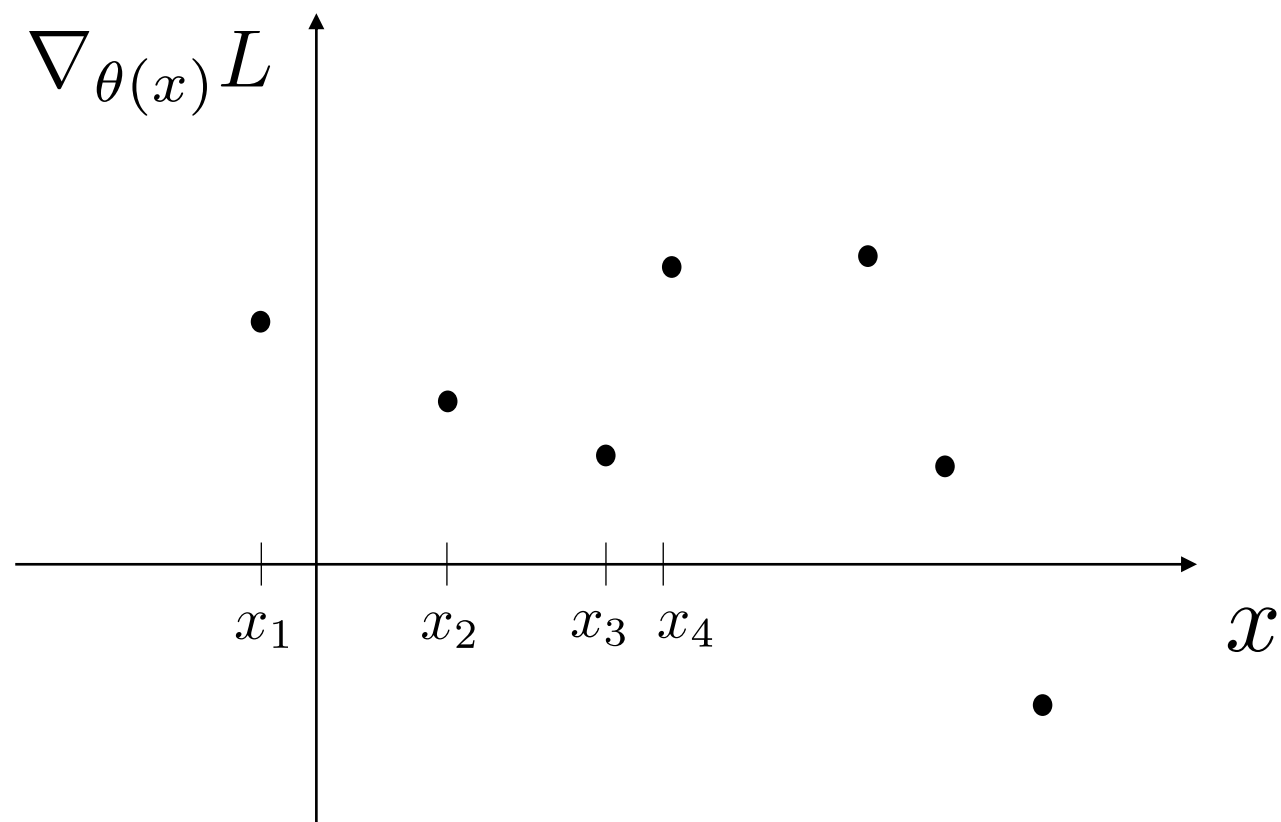


Gaps in the gradient



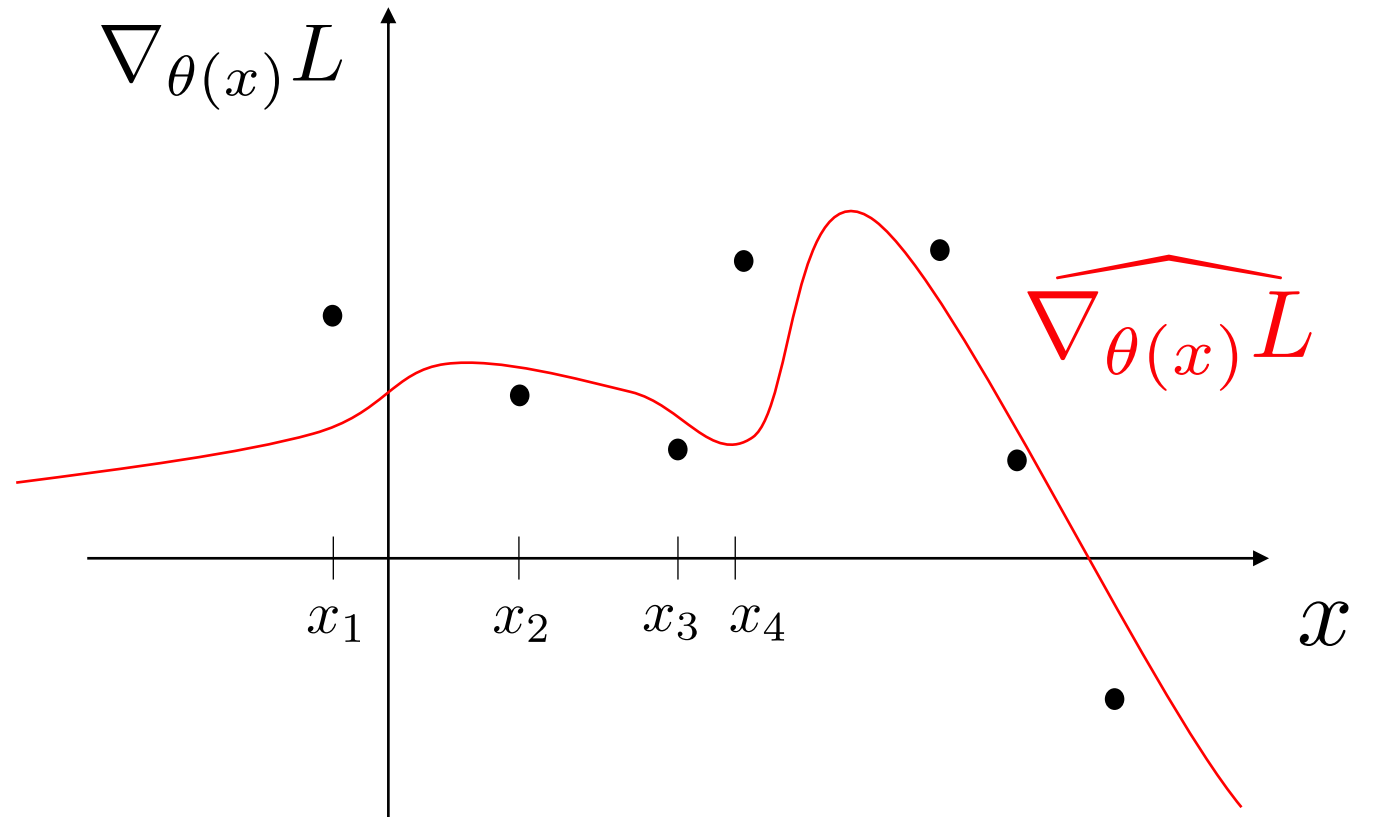
Gaps in the gradient

$$\nabla_{\theta} L = 2 \begin{bmatrix} \vdots \\ y_1 - \theta(x_1) \\ \vdots \\ y_2 - \theta(x_2) \\ \vdots \\ y_3 - \theta(x_3) \\ \vdots \end{bmatrix}$$



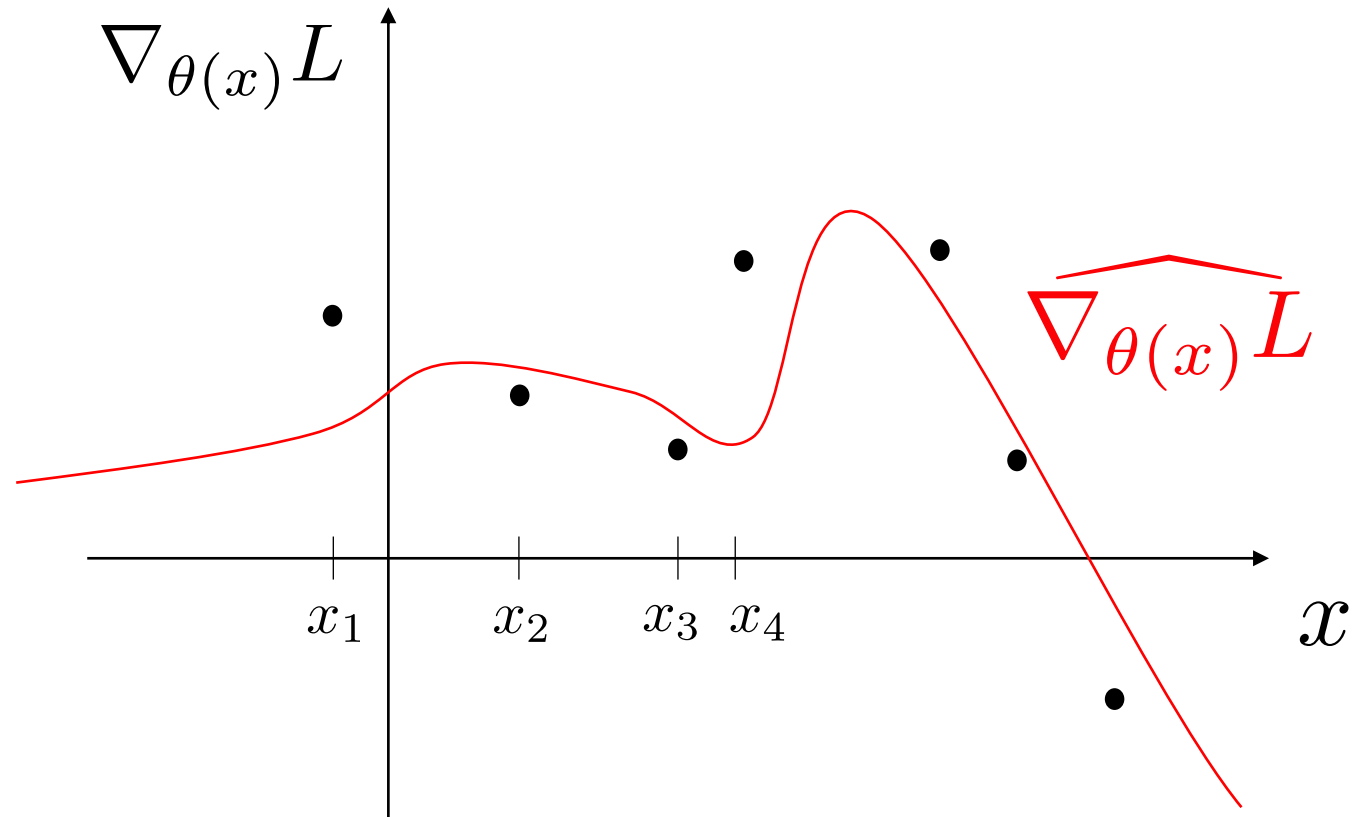
Boosting idea: fill in the gaps with a regression model (*base learner*)

$$\nabla_{\theta} L = 2 \begin{bmatrix} \vdots \\ y_1 - \theta(x_1) \\ \vdots \\ y_2 - \theta(x_2) \\ \vdots \\ y_3 - \theta(x_3) \\ \vdots \end{bmatrix}$$



Boosting idea: fill in the gaps with a regression model (*base learner*)

- Must do this at each step



Taking B steps starting at θ , our model is

$$\theta_B(x) = \sum_b^B \epsilon \widehat{\nabla_{\theta_b(x)} L}$$

- To predict at a new x , sum up the predictions of each of the B regressors at that x

NGBoost idea: we can apply this strategy to fit probabilistic regression models

$$L(y, \theta(x)) \rightarrow S(y, P_{\theta(x)})$$

$$\nabla_{\theta} L \rightarrow \nabla_{\theta} S$$

Example: NLL score and Normal distribution

$$P(y) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{y - \mu}{\sigma} \right)^2}$$

$$\begin{aligned} S(P, y) &= -\log P(y) \\ &= \log(\sigma^2) - \frac{(y - \mu)^2}{2\sigma^2} \end{aligned}$$

So instead of boosting one function, we need to boost as many as there are parameters

$$\begin{bmatrix} \theta_{1B}(x) \\ \theta_{2B}(x) \\ \vdots \\ \theta_{pB}(x) \end{bmatrix} = \begin{bmatrix} \sum_b^B \nabla_{\theta_{1b}(x)} S \\ \sum_b^B \nabla_{\theta_{2b}(x)} S \\ \vdots \\ \sum_b^B \nabla_{\theta_{pb}(x)} S \end{bmatrix}$$

- At each boosting iteration, we fit p models, one per parameter

At the end of this, we have, for any x , a predicted distribution of y

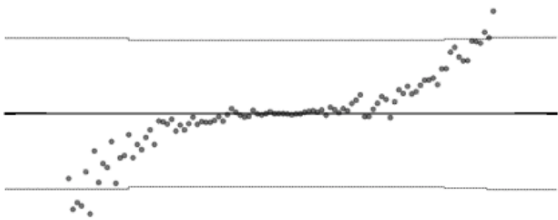
$$P(y|X = x) = P \begin{bmatrix} \theta_{1B}(x) \\ \theta_{2B}(x) \\ \vdots \\ \theta_{pB}(x) \end{bmatrix} (y)$$

Outline of our approach

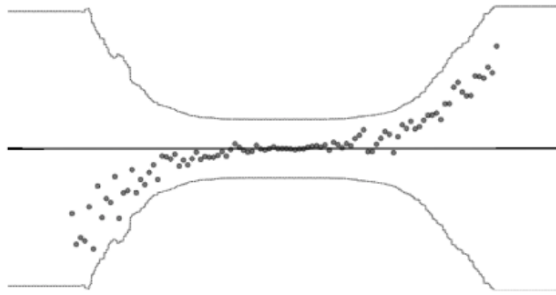
1. Pick a scoring rule to grade our estimate of $P(Y/X=x)$
2. Assume that $P(Y/X=x)$ has some **parametric form**
3. Fit the parameters $\theta(x)$ as a function of x using **gradient boosting**
4. Use the **natural gradient** to correct the training dynamics of this approach

Our boosting approach gives us probabilistic prediction, but performs poorly

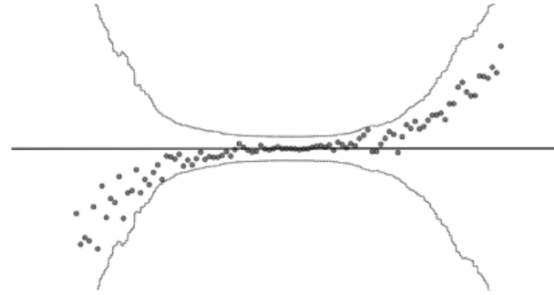
(a) 0% fit



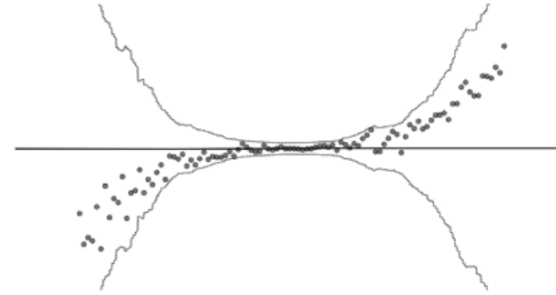
(b) 33% fit



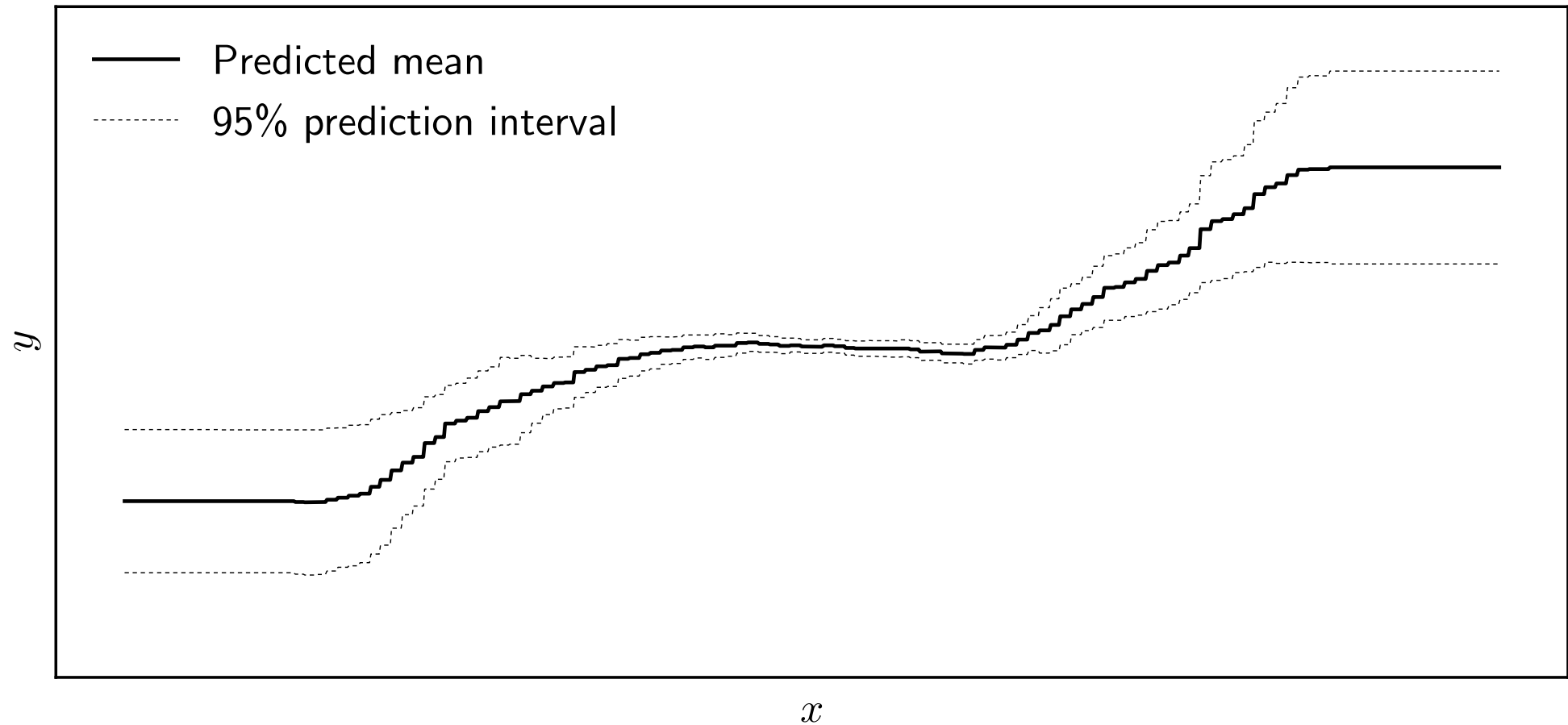
(c) 67% fit



(d) 100% fit



Ideal probabilistic regression



Why?

- We relaxed an unrealistic assumption, so we expect performance to increase, not decrease



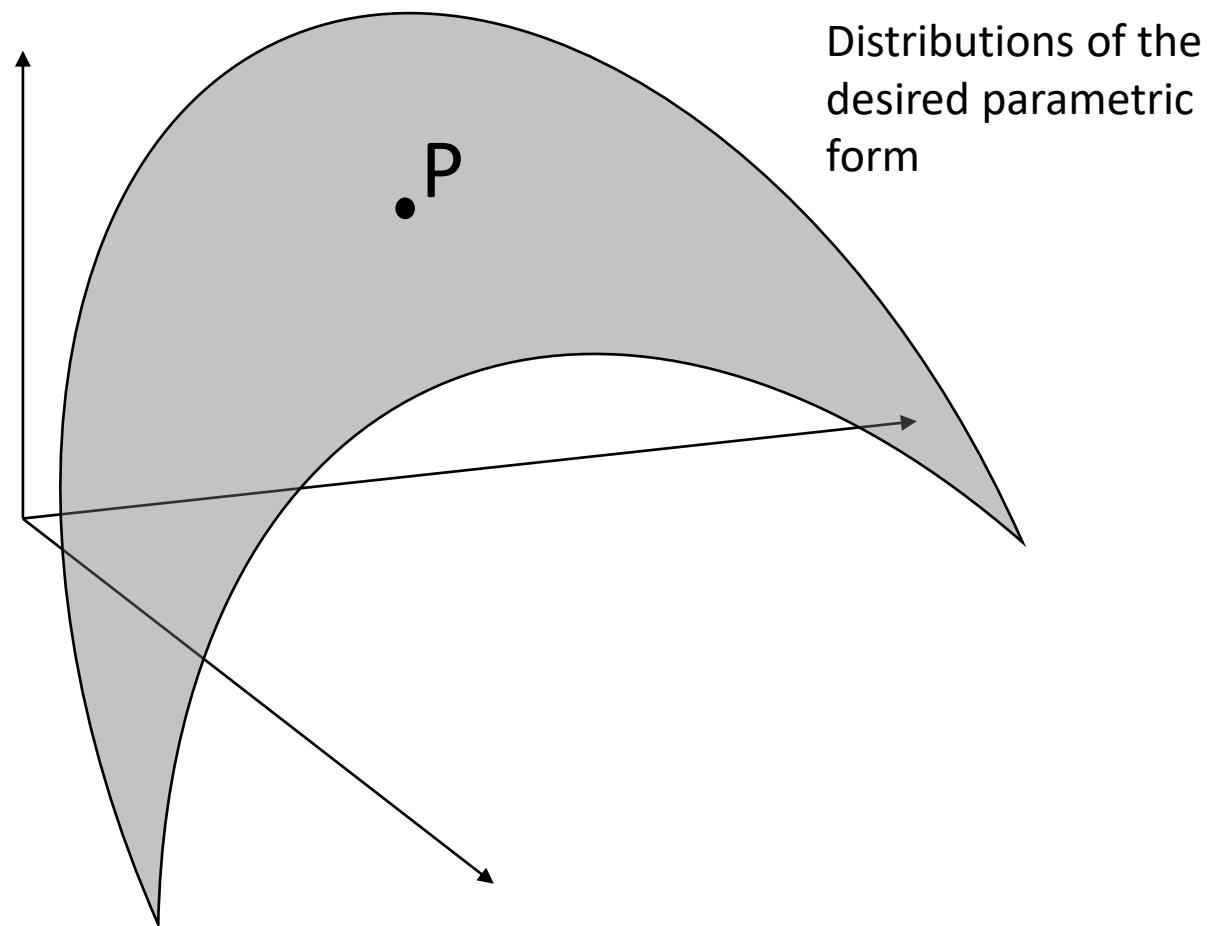
- However, this picture suggests that the algorithm is failing to adjust the mean (in this example).
- There is a better optimum, but we can't find it

Outline of our approach

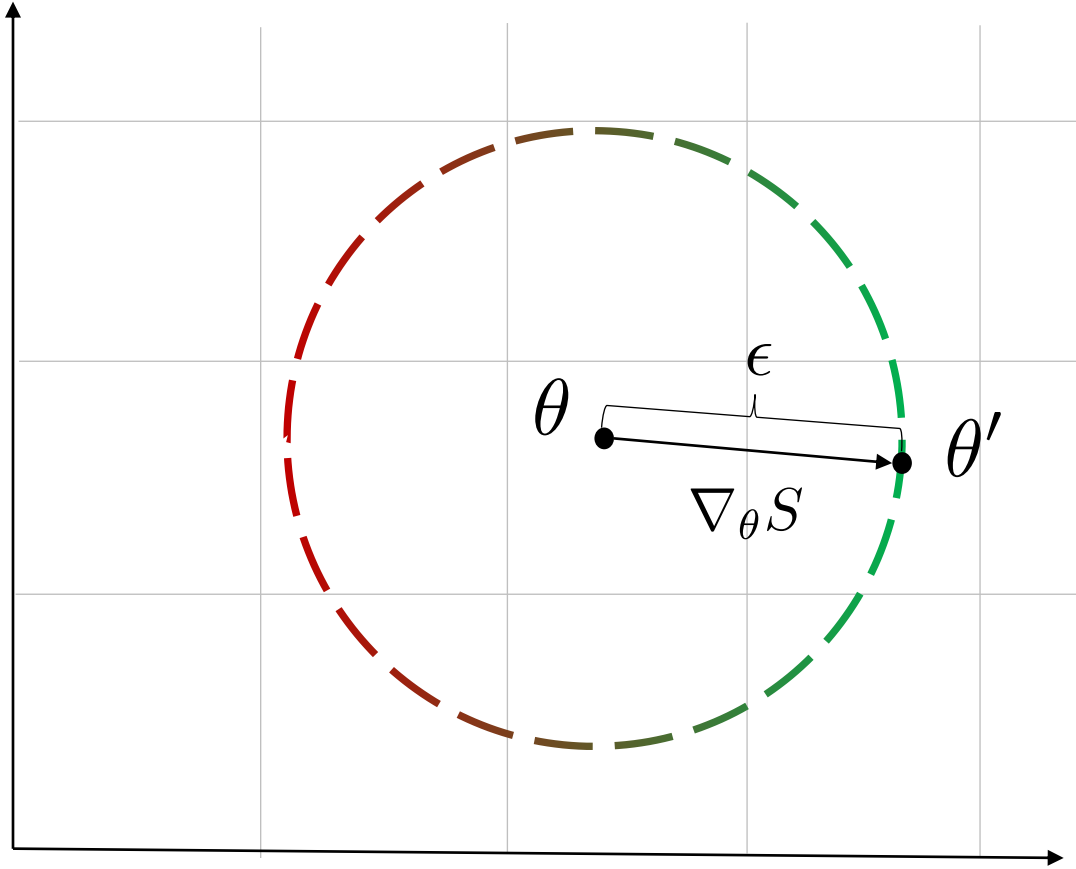
1. Pick a scoring rule to grade our estimate of $P(Y/X=x)$
2. Assume that $P(Y/X=x)$ has some **parametric form**
3. Fit the parameters $\theta(x)$ as a function of x using **gradient boosting**
4. Use the **natural gradient** to correct the training dynamics of this approach

Stepping back: the space of distributions

$P(Y/X)$

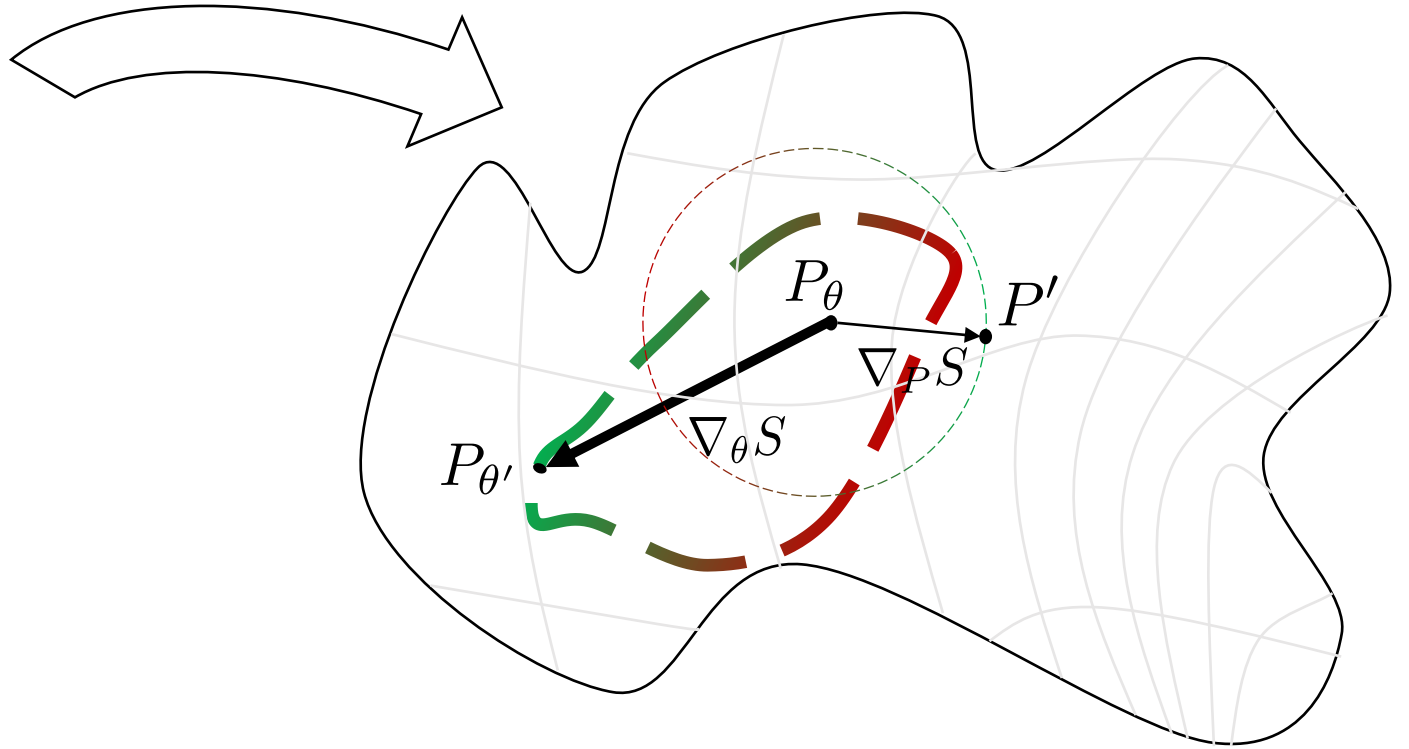
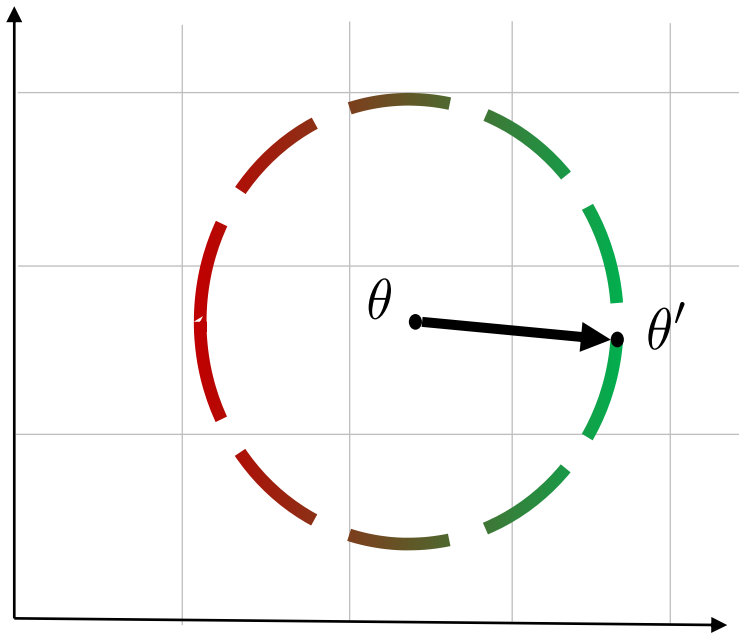


What we typically do: gradient descent in the parameter space



$$\nabla_{\theta} S \propto \lim_{\epsilon \rightarrow 0} \operatorname{argmax}_{d(\theta, \theta') < \epsilon} S(P_{\theta'})$$

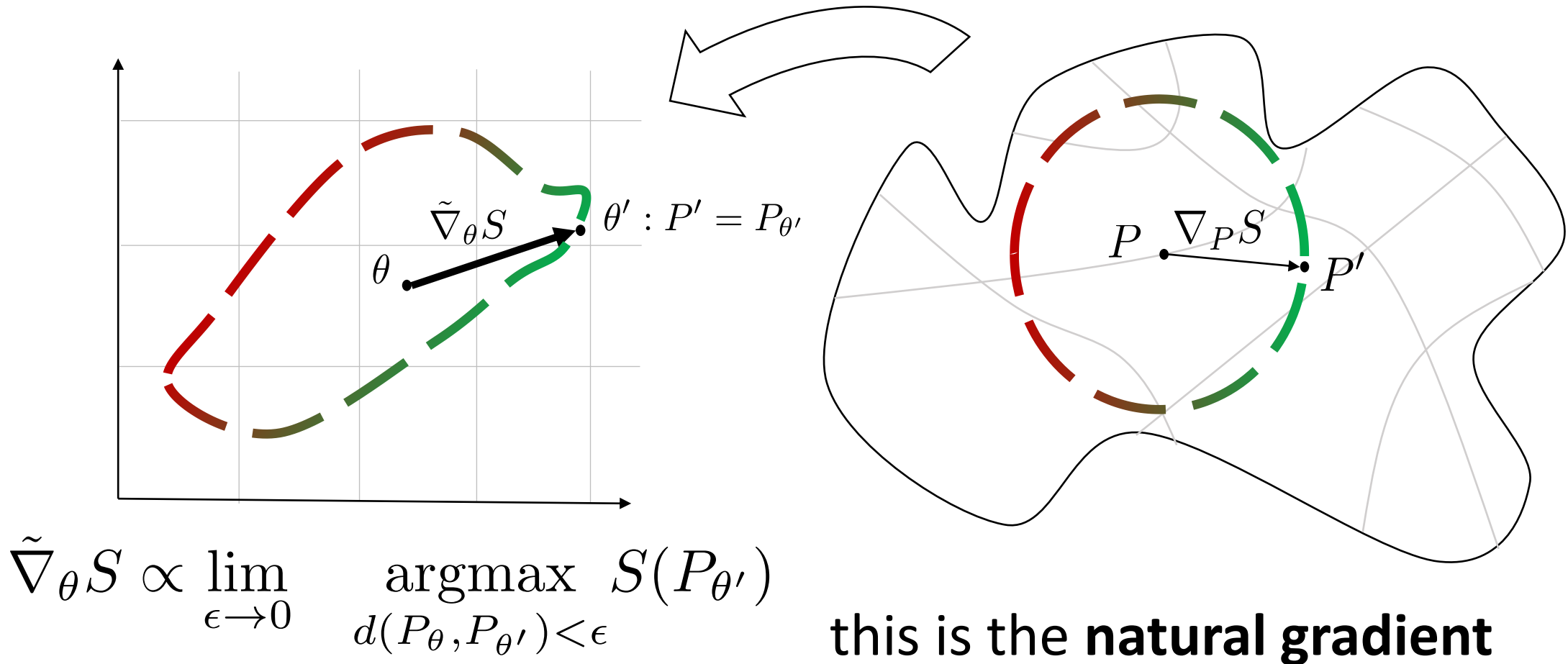
Gradient descent in the parameter space is not gradient descent in the distribution space because **distances** don't correspond



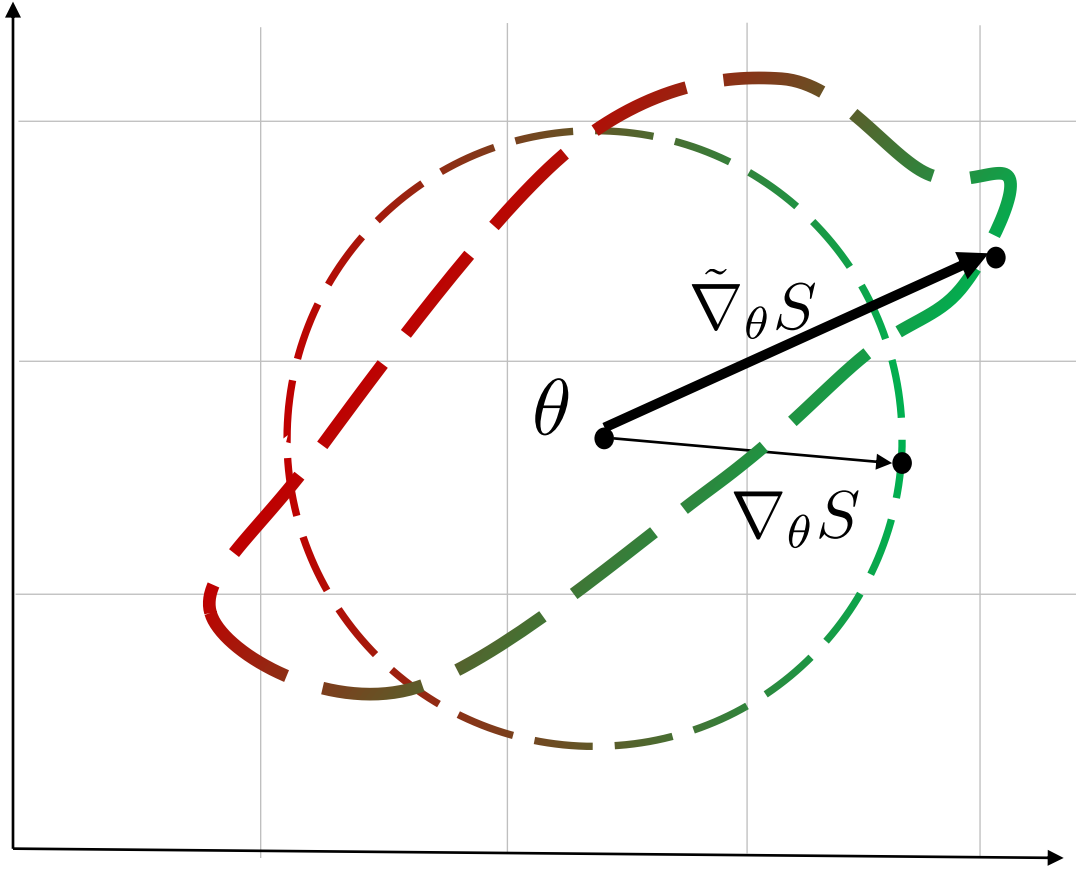
$$\nabla_{\theta} S \propto \lim_{\epsilon \rightarrow 0} \operatorname{argmax}_{d(\theta, \theta') < \epsilon} S(P_{\theta'})$$

$$\nabla_P S \propto \lim_{\epsilon \rightarrow 0} \operatorname{argmax}_{d(P, P') < \epsilon} S(P')$$

Idea: do gradient descent in the distribution by searching parameters in the transformed region



We can conveniently compute the natural gradient by applying a transformation to the gradient



$$\tilde{\nabla}_{\theta} S = I(\theta)^{-1} \nabla_{\theta} S$$

- $I(\theta)$ is the *Riemannian metric* of the space of distributions
- It depends on the parametric form chosen and the score function
- If the score is NLL, this is the Fisher Information

Why? To solve this optimization (for NLL)...

$$\tilde{\nabla}_{\theta} S \propto \lim_{\epsilon \rightarrow 0} \operatorname{argmax}_{d(P_{\theta}, P_{\theta'}) < \epsilon} S(P_{\theta'})$$

- Use KL divergence as the distance in distribution space
 - Not arbitrary- every scoring rule induces its own divergence
- Take the 2nd order Taylor expansion to see that KL divergence between distributions with parameters θ and θ' is approximated by

$$(\theta - \theta')^T I(\theta) (\theta - \theta')$$

- Write the optimization in Lagrangian form, set derivative to zero, gives:

$$\tilde{\nabla}_{\theta} S = I(\theta)^{-1} \nabla_{\theta} S$$

NGBoost is our multiparameter boosting approach with a natural gradient update

Algorithm 1 NGBoost for probabilistic prediction

Data: Dataset $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$.

Input: Boosting iterations M , Learning rate η , Probability distribution with parameter θ , Proper scoring rule \mathcal{S} , Base learner f .

Output: Scalings and base learners $\{\rho^{(m)}, f^{(m)}\}_{m=1}^M$.

$\theta^{(0)} \leftarrow \arg \min_{\theta} \sum_{i=1}^n \mathcal{S}(\theta, y_i) \quad \triangleright$ initialize to marginal

for $m \leftarrow 1, \dots, M$ **do**

for $i \leftarrow 1, \dots, n$ **do**

$g_i^{(m)} \leftarrow \mathcal{I}_{\mathcal{S}} \left(\theta_i^{(m-1)} \right)^{-1} \nabla_{\theta} \mathcal{S} \left(\theta_i^{(m-1)}, y_i \right)$

end

$f^{(m)} \leftarrow \text{fit} \left(\left\{ x_i, g_i^{(m)} \right\}_{i=1}^n \right)$

$\rho^{(m)} \leftarrow \arg \min_{\rho} \sum_{i=1}^n \mathcal{S} \left(\theta_i^{(m-1)} - \rho \cdot f^{(m)}(x_i), y_i \right)$

for $i \leftarrow 1, \dots, n$ **do**

$\theta_i^{(m)} \leftarrow \theta_i^{(m-1)} - \eta \left(\rho^{(m)} \cdot f^{(m)}(x_i) \right)$

end

end

... by improving the training dynamics

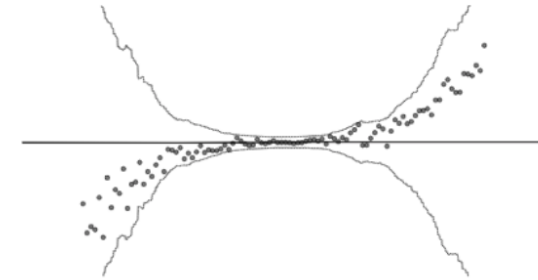
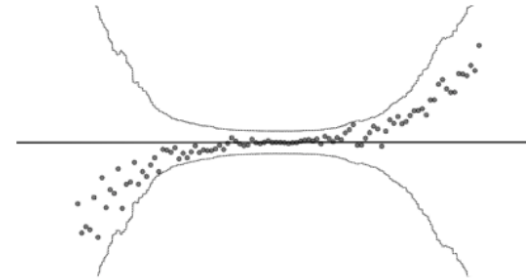
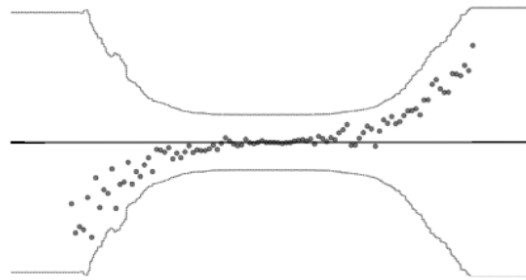
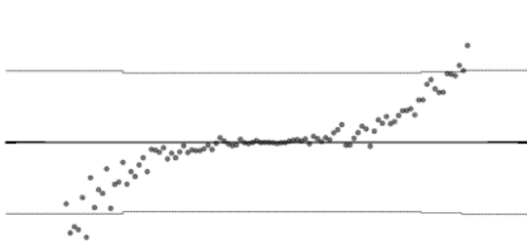
(a) 0% fit

(b) 33% fit

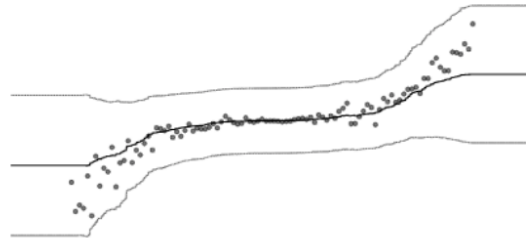
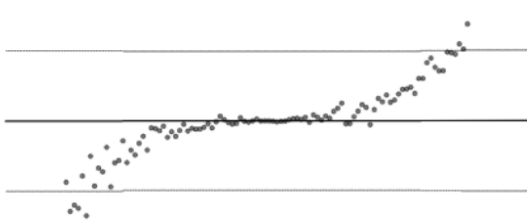
(c) 67% fit

(d) 100% fit

Ordinary



Natural



The result is equal or better performance than state-of-the art probabilistic prediction methods

Dataset	N	NLL				
		NGBoost	MC dropout	Deep Ensembles	Concrete Dropout	Gaussian Process
Boston	506	2.43 ± 0.15	2.46 ± 0.25	2.41 ± 0.25	2.72 ± 0.01	2.37 ± 0.24
Concrete	1030	3.04 ± 0.17	3.04 ± 0.09	3.06 ± 0.18	3.51 ± 0.00	3.03 ± 0.11
Energy	768	0.60 ± 0.45	1.99 ± 0.09	1.38 ± 0.22	2.30 ± 0.00	0.66 ± 0.17
Kin8nm	8192	-0.49 ± 0.02	-0.95 ± 0.03	-1.20 ± 0.02	-0.65 ± 0.00	-1.11 ± 0.03
Naval	11934	-5.34 ± 0.04	-3.80 ± 0.05	-5.63 ± 0.05	-5.87 ± 0.05	-4.98 ± 0.02
Power	9568	2.79 ± 0.11	2.80 ± 0.05	2.79 ± 0.04	2.75 ± 0.01	2.81 ± 0.05
Protein	45730	2.81 ± 0.03	2.89 ± 0.01	2.83 ± 0.02	2.81 ± 0.00	2.89 ± 0.02
Wine	1588	0.91 ± 0.06	0.93 ± 0.06	0.94 ± 0.12	1.70 ± 0.00	0.95 ± 0.06
Yacht	308	0.20 ± 0.26	1.55 ± 0.12	1.18 ± 0.21	1.75 ± 0.00	0.10 ± 0.26
Year MSD	515345	$3.43 \pm \text{NA}$	$3.59 \pm \text{NA}$	$3.35 \pm \text{NA}$	$\text{NA} \pm \text{NA}$	$\text{NA} \pm \text{NA}$

(same experimental setup as previously described)

But unlike other methods, NGBBoost is **fast**, **flexible**, **scalable**, and **easy to use**

- No MCMC
- Few restrictive assumptions
- Works for regression, classification, survival, etc.
- Scales like gradient boosting
- Scikit-learn compatible

```
from ngboost import NGBRegressor

from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X, Y = load_boston(True)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)

ngb = NGBRegressor().fit(X_train, Y_train)
Y_preds = ngb.predict(X_test)
Y_dists = ngb.pred_dist(X_test)

# test Mean Squared Error
test_MSE = mean_squared_error(Y_preds, Y_test)
print('Test MSE', test_MSE)

# test Negative Log Likelihood
test_NLL = -Y_dists.logpdf(Y_test.flatten()).mean()
print('Test NLL', test_NLL)
```