

Week 6 Discussion

CS 188.2

Topics

- Machine Learning Overview
 - Supervised
 - Regression
 - Classification
 - Unsupervised
 - ML Algorithms
- K-fold Overview
- Back Propagation

Outline

What is machine learning?

Supervised learning: regression

“Non-linear” regression, overfitting, and model selection

Supervised learning: classification

Other machine learning algorithms

Unsupervised learning

Outline

What is machine learning?

Supervised learning: regression

“Non-linear” regression, overfitting, and model selection

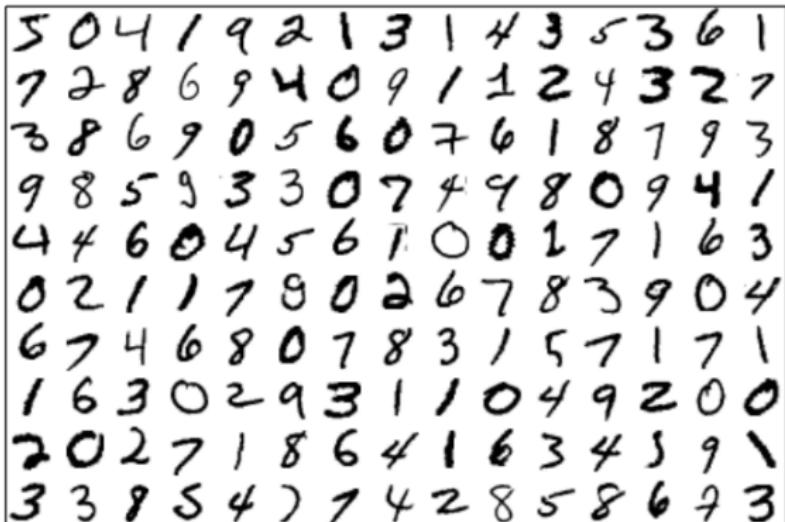
Supervised learning: classification

Other machine learning algorithms

Unsupervised learning

Introduction: digit classification

The task: write a program that, given a 28x28 grayscale image of a digit, outputs the string representation



Digits from MNIST dataset

(<http://yann.lecun.com/exdb/mnist/>)

One approach: try to write a program by hand that uses your a priori knowledge of digits to properly classify the images

Alternative method (machine learning): collect a bunch of images and their corresponding digits, write a program that uses this data to build its own method for classifying images

(More precisely, this is a subset of machine learning called *supervised learning*)

A supervised learning pipeline

Training Data

$(\text{2}, 2)$

$(\text{0}, 0)$

$(\text{8}, 8)$

$(\text{5}, 5)$

\vdots

A supervised learning pipeline

Training Data

(**2** , 2)
(**0** , 0)
(**8** , 8)
(**5** , 5)
⋮

Machine Learning

→ Hypothesis
function
 h_{θ}

A supervised learning pipeline

Training Data

$(\text{2}, 2)$
 $(\text{0}, 0)$
 $(\text{8}, 8)$
 $(\text{5}, 5)$
⋮

Machine Learning

→ Hypothesis
function
 h_{θ}

Deployment

Prediction = $h_{\theta}(\text{2})$

Prediction = $h_{\theta}(\text{5})$

⋮

Unsupervised learning

Training Data

$$\begin{pmatrix} 2 \\ 0 \\ 8 \\ 5 \end{pmatrix}$$

:

Machine Learning

→ Hypothesis
function
 h_{θ}

Deployment

$$\text{Prediction} = h_{\theta} \begin{pmatrix} 2 \end{pmatrix}$$

$$\text{Prediction} = h_{\theta} \begin{pmatrix} 5 \end{pmatrix}$$

:

Outline

What is machine learning?

Supervised learning: regression

“Non-linear” regression, overfitting, and model selection

Supervised learning: classification

Other machine learning algorithms

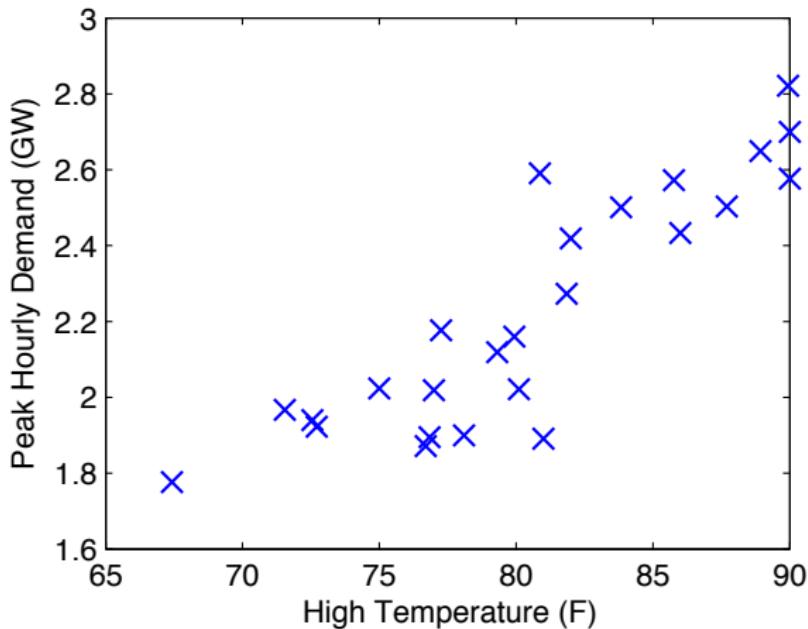
Unsupervised learning

A simple example: predicting electricity use

What will peak power consumption be in the Pittsburgh area tomorrow?

Collect data of past high temperatures and peak demands

High Temperature (F)	Peak Demand (GW)
76.7	1.87
72.7	1.92
71.5	1.96
86.0	2.43
90.0	2.69
87.7	2.50
:	:



Several days of peak demand vs. high temperature in Pittsburgh

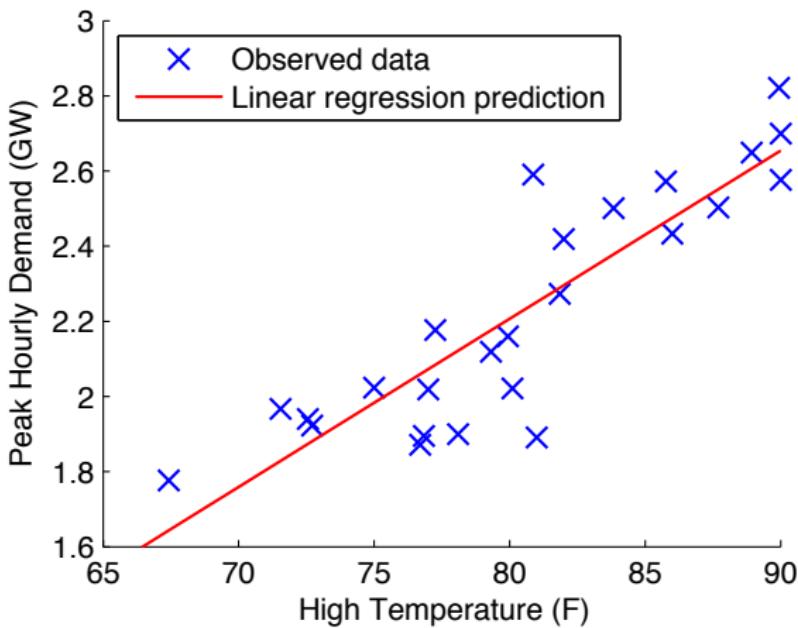
Hypothesize model

$$\text{Peak demand} \approx \theta_1 \cdot (\text{High temperature}) + \theta_2$$

for some numbers θ_1 and θ_2

Then, given a forecast of tomorrow's high temperature, we can predict the likely peak demand by plugging it into our model

Equivalent to “drawing a line through the data”



Notation

Input features: $x^{(i)} \in \mathbb{R}^n$, $i = 1, \dots, m$

- E.g.: $x^{(i)} \in \mathbb{R}^2 = \begin{bmatrix} \text{high temperature for day } i \\ 1 \end{bmatrix}$

Output: $y^{(i)} \in \mathbb{R}$ (*regression task*)

- E.g.: $y^{(i)} \in \mathbb{R} = \{\text{peak demand for day } i\}$

Model Parameters: $\theta \in \mathbb{R}^n$

Hypothesis function: $h_\theta(x) : \mathbb{R}^n \rightarrow \mathbb{R}$

- Hypothesis function: $h_\theta(x)$ returns a *prediction* of the output y , e.g. *linear regression*

$$h_\theta(x) = x^T \theta = \sum_{i=1}^n x_i \theta_i$$

Loss functions

How do we measure how “good” a hypothesis is on the training data?

Typically done by introducing a *loss function*

$$\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}_+$$

Intuitively, this function outputs a “small” value if $h_\theta(x)$ is “close” to y , a large value if it is “far” from y

E.g., for regression, squared loss

$$\ell(h_\theta(x), y) = (h_\theta(x) - y)^2$$

The canonical machine learning problem

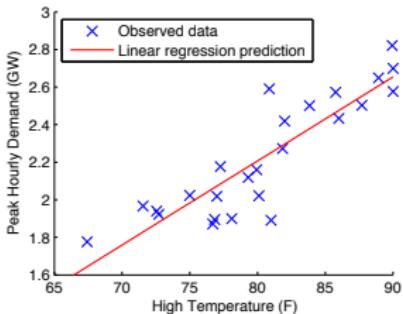
Given a collection of input features and outputs $(x^{(i)}, y^{(i)})$, $i = 1, \dots, m$, and a hypothesis function h_θ , find parameters θ that minimize the sum of losses

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^m \ell\left(h_\theta(x^{(i)}), y^{(i)}\right)$$

Virtually all learning algorithms can be described in this form, we just need to specify three things:

1. The hypothesis class: h_θ
2. The loss function: ℓ
3. The algorithm for solving the optimization problem (often approximately)

Return to power demand forecasting



Linear hypothesis class: $h_\theta(x) = x^T \theta$

Squared loss function: $\ell(h_\theta(y), y) = (h_\theta(x) - y)^2$

Resulting optimization problem

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^m \ell\left(h_\theta(x^{(i)}), y^{(i)}\right) \equiv \underset{\theta}{\text{minimize}} \sum_{i=1}^m \left(x^{(i)T} \theta - y^{(i)}\right)^2$$

Linear regression

Gradient descent to solve optimization problem

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^m \left(x^{(i)T} \theta - y^{(i)} \right)^2$$

Gradient is given by

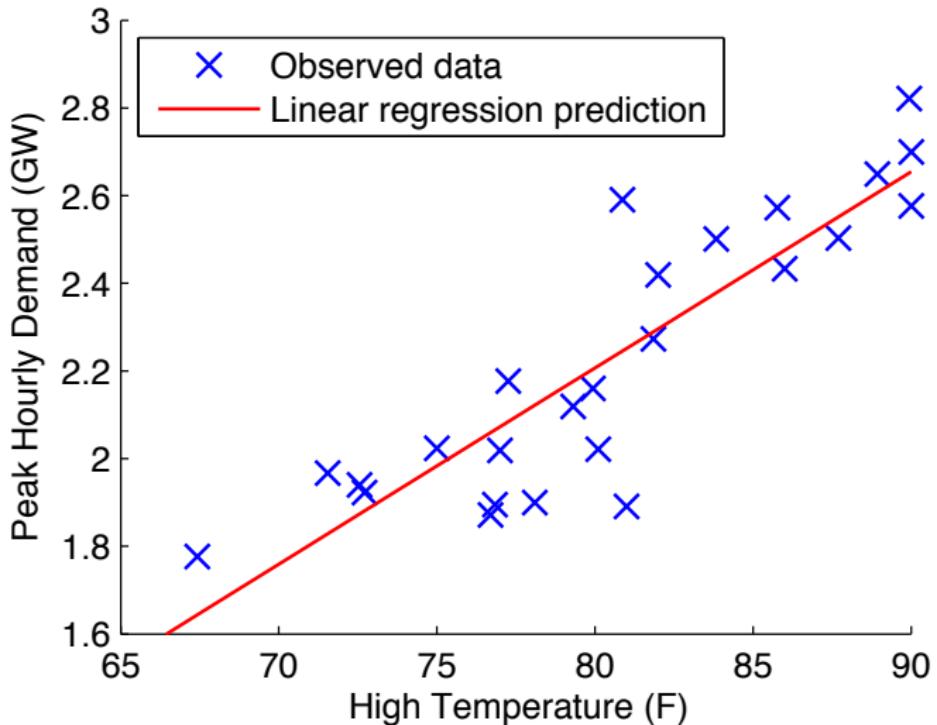
$$\begin{aligned} \nabla_{\theta} \sum_{i=1}^m \left(x^{(i)T} \theta - y^{(i)} \right)^2 &= \sum_{i=1}^m \nabla_{\theta} \left(x^{(i)T} \theta - y^{(i)} \right)^2 \\ &= 2 \sum_{i=1}^m x^{(i)} \left(x^{(i)T} \theta - y^{(i)} \right) \end{aligned}$$

Gradient descent, repeat: $\theta \leftarrow \theta - \alpha \sum_{i=1}^m x^{(i)} \left(x^{(i)T} \theta - y^{(i)} \right)$

In this case, we can also directly solve for $\nabla_{\theta}f(\theta) = 0$

$$\begin{aligned} & \sum_{i=1}^m x^{(i)} \left(x^{(i)T} \theta^* - y^{(i)} \right) = 0 \\ \implies & \left(\sum_{i=1}^m x^{(i)} x^{(i)T} \right) \theta^* = \sum_{i=1}^m x^{(i)} y^{(i)} \\ \implies & \theta^* = \left(\sum_{i=1}^m x^{(i)} x^{(i)T} \right)^{-1} \left(\sum_{i=1}^m x^{(i)} y^{(i)} \right) \end{aligned}$$

Squared loss is one of the few cases that such direct solutions are possible, usually need to resort to gradient descent or other methods



Alternative loss functions

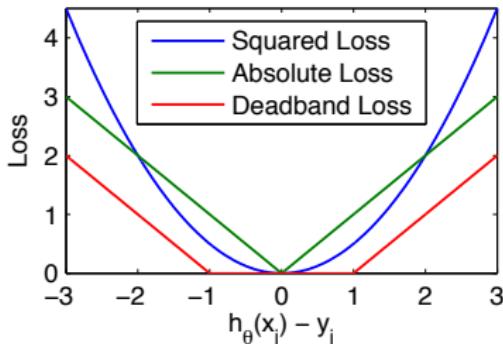
Why did we choose the squared loss function

$$\ell(h_\theta(x), y) = (h_\theta(x) - y)^2?$$

Some other alternatives

Absolute loss: $\ell(h_\theta(x), y) = |h_\theta(x) - y|$

Deadband loss: $\ell(h_\theta(x), y) = \max\{0, |h_\theta(x) - y| - \epsilon\}, \epsilon \in \mathbb{R}_+$

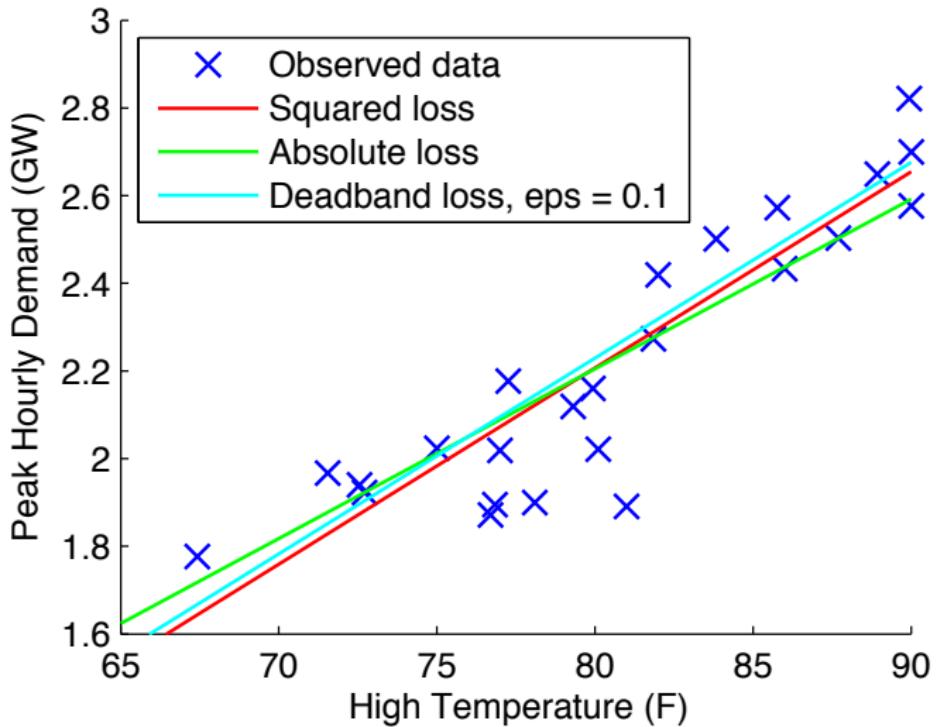


For these loss functions, no closed-form expression for θ^* , but
(sub)gradient descent can still be very effective

E.g., for absolute loss and linear hypothesis class

$$\text{Repeat : } \theta \leftarrow \theta - \alpha \sum_{i=1}^m x^{(i)} \text{sign} \left(x^{(i)T} \theta - y^{(i)} \right)$$

Can also solve for nonsmooth losses using constrained optimization
(and libraries like cvxpy)



Probabilistic interpretation

Suppose that the each output y in our data really *is* equal to the hypothesis function for that example $h_\theta(x)$, just corrupted by Gaussian noise ϵ

$$y = h_\theta(x) + \epsilon$$

The probability density of a Gaussian variable given by

$$p(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right)$$

Substituting terms, we can use this expression to write the probability of y given x (parameterized by θ)

$$p(y|x; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(h_\theta(x) - y)^2}{2\sigma^2}\right)$$

Consider the joint probability of *all* training data (assuming samples are independent and identically distributed)

$$p(y^{(1)}, \dots, y^{(m)} | x^{(1)}, \dots, x^{(m)}; \theta) = \prod_{i=1}^m p(y^{(i)} | x^{(i)}; \theta)$$

Find the parameters that θ maximize the probability of the data

$$\begin{aligned} \underset{\theta}{\text{maximize}} \prod_{i=1}^m p(y^{(i)} | x^{(i)}; \theta) &\equiv \underset{\theta}{\text{minimize}} - \sum_{i=1}^m \log p(y^{(i)} | x^{(i)}; \theta) \\ &\equiv \underset{\theta}{\text{minimize}} \sum_{i=1}^m \left(\log(\sqrt{2\pi}\sigma) + \frac{1}{2\sigma^2} (h_{\theta}(x^{(i)}) - y^{(i)})^2 \right) \\ &\equiv \underset{\theta}{\text{minimize}} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \end{aligned}$$

This is a procedure known as *maximum likelihood estimation*, a common statistical technique

Note that we still just pushed the question of “which loss” to “which distribution”

- But some distributions, like Gaussian, may have reasonable empirical or theoretical justifications for certain problems

Stochastic gradient descent

As mentioned, the optimization problems we deal with in machine learning are of the form

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^m \ell(h_{\theta}(x^{(i)}), y^{(i)})$$

Procedurally, gradient descent then takes the form:

function $\theta = \text{Gradient-Descent}(\{(x^{(i)}, y^{(i)})\}, h_{\theta}, \ell, \alpha)$

 Initialize: $\theta \leftarrow 0$

Repeat until convergence

$g \leftarrow 0$

For $i = 1, \dots, m$:

$g \leftarrow g + \nabla_{\theta} \ell(h_{\theta}(x^{(i)}), y^{(i)})$

$\theta \leftarrow \theta - \alpha g$

return θ

If the number of samples m is large, computing a *single* gradient step is costly

An alternative approach, stochastic gradient descent (SGD), update the parameters based upon gradient each sample:

function $\theta = \text{SGD}(\{(x^{(i)}, y^{(i)})\}, h_\theta, \ell, \alpha)$

 Initialize: $\theta \leftarrow 0$

Repeat until convergence

For $i = 1, \dots, m$:

$\theta \leftarrow \theta - \alpha \nabla_{\theta} \ell(h_{\theta}(x^{(i)}), y^{(i)})$

return θ

Can be viewed as taking many more steps along *noisy* estimates of the gradient, and often converges to a “good” parameter value after relatively few passes over the data set

Outline

What is machine learning?

Supervised learning: regression

“Non-linear” regression, overfitting, and model selection

Supervised learning: classification

Other machine learning algorithms

Unsupervised learning

Classification problems

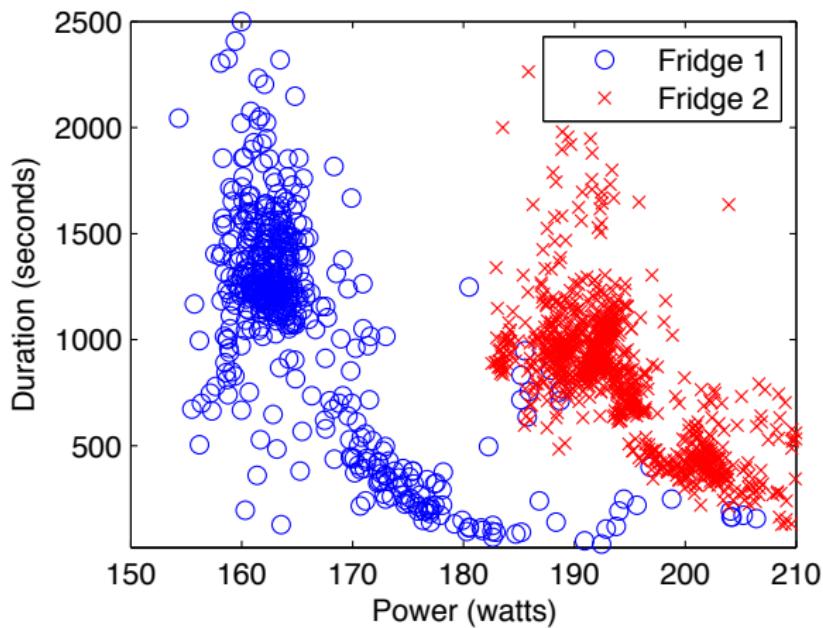
Sometimes we want to predict discrete outputs rather than continuous

Is the email spam or not? (YES/NO)

What digit is in this image? (0/1/2/3/4/5/6/7/8/9)

Example: classifying household appliances

Differentiate between two refrigerators using their power consumption signatures



Notation

Input features: $x^{(i)} \in \mathbb{R}^n$, $i = 1, \dots, m$

- E.g.: $x^{(i)} \in \mathbb{R}^3 = (\text{Duration } i, \text{Power } i, 1)$

Output: $y^{(i)} \in \{-1, +1\}$ (binary classification task)

- E.g.: $y^{(i)}$ = Is it fridge 1?

Model Parameters: $\theta \in \mathbb{R}^n$

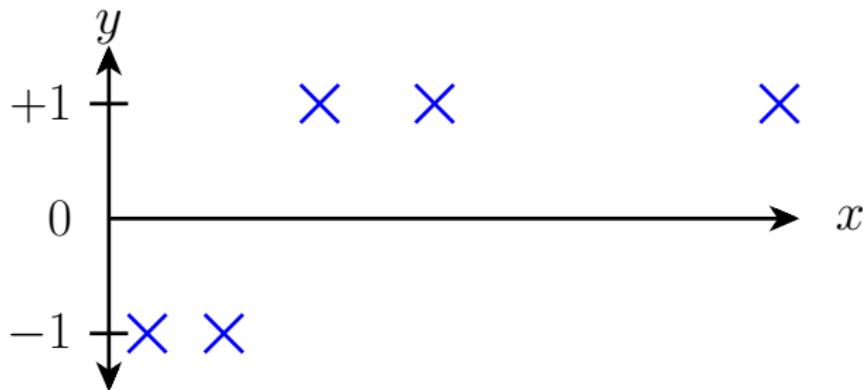
Hypothesis function: $h_\theta(x) : \mathbb{R}^n \rightarrow \mathbb{R}$

- Returns *continuous* prediction of the output y , where the value indicates how “confident” we are that the example is -1 or $+1$; $\text{sign}(h_\theta(x))$ is the actual binary prediction
- Again, we will focus initially on *linear predictors* $h_\theta(x) = x^T \theta$

Loss functions

Loss function $\ell : \mathbb{R} \times \{-1, +1\} \rightarrow \mathbb{R}_+$

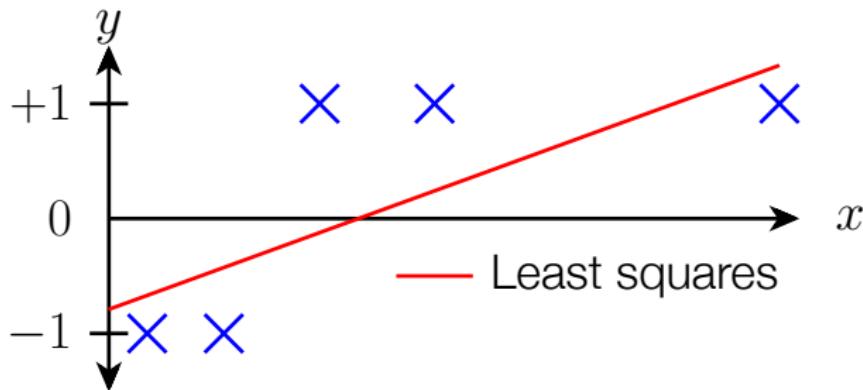
Do we need a different loss function?



Loss functions

Loss function $\ell : \mathbb{R} \times \{-1, +1\} \rightarrow \mathbb{R}_+$

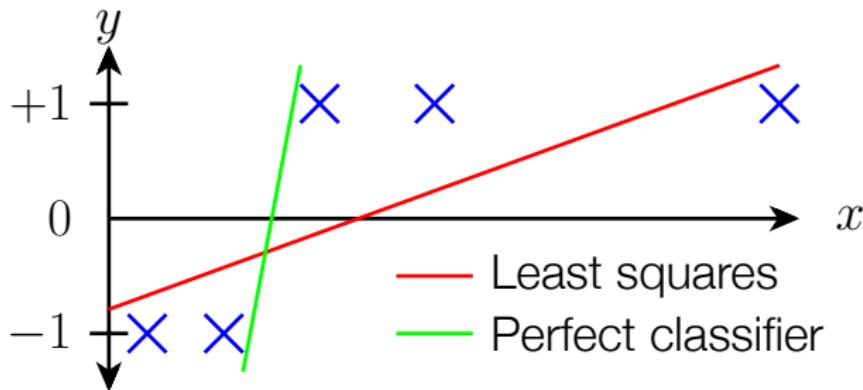
Do we need a different loss function?



Loss functions

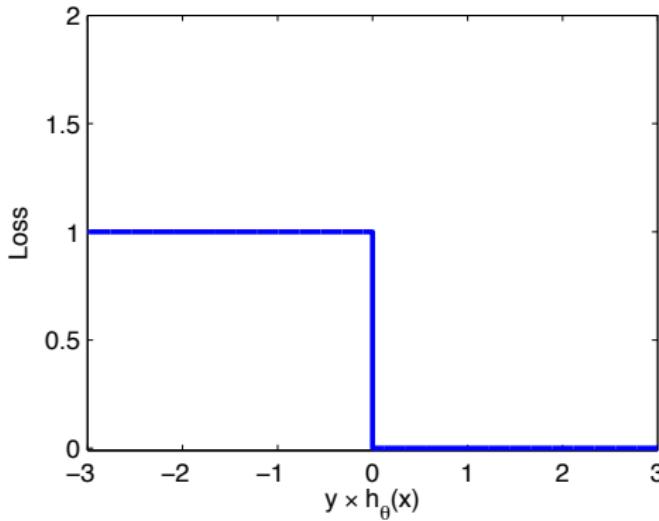
Loss function $\ell : \mathbb{R} \times \{-1, +1\} \rightarrow \mathbb{R}_+$

Do we need a different loss function?



The simplest loss (0/1 loss, accuracy): count the number of mistakes we make

$$\ell(h_\theta(x), y) = \begin{cases} 1 & \text{if } y \neq \text{sign}(h_\theta(x)) \\ 0 & \text{otherwise} \end{cases}$$
$$= \mathbf{1}\{y \cdot h_\theta(x) \leq 0\}$$



Unfortunately, minimizing sum of 0/1 losses leads to a hard optimization problem

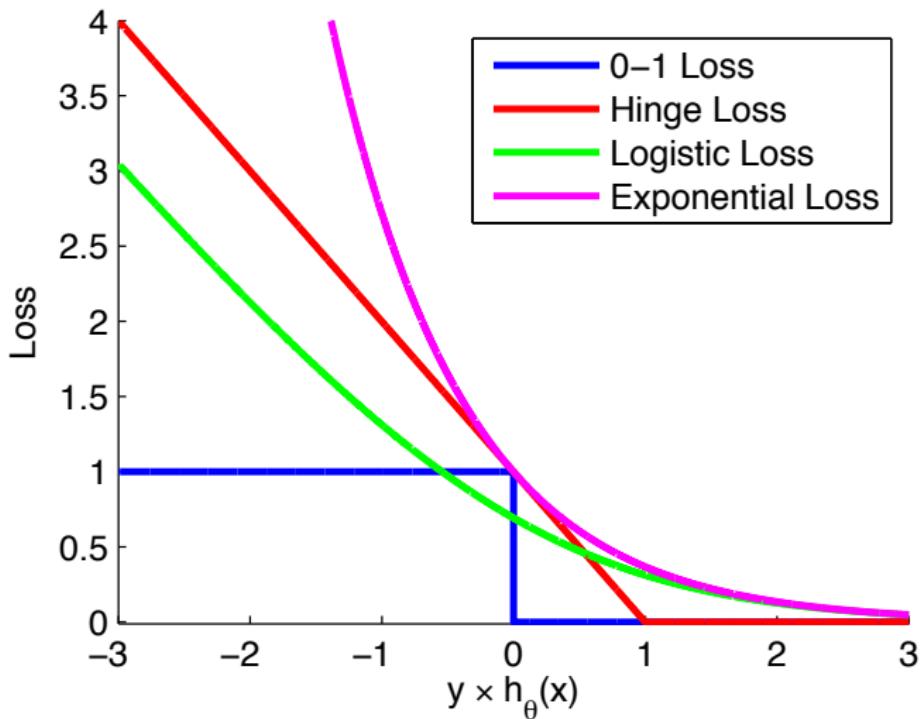
Because of this, a whole range of alternative “approximations” to 0/1 loss are used instead

$$\text{Hinge loss: } \ell(h_\theta(x), y) = \max\{1 - y \cdot h_\theta(x), 0\}$$

$$\text{Squared hinge loss: } \ell(h_\theta(x), y) = \max\{1 - y \cdot h_\theta(x), 0\}^2$$

$$\text{Logistic loss: } \ell(h_\theta(x), y) = \log(1 + e^{-y \cdot h_\theta(x)})$$

$$\text{Exponential loss: } \ell(h_\theta(x), y) = e^{-y \cdot h_\theta(x)}$$



Common loss functions for classification

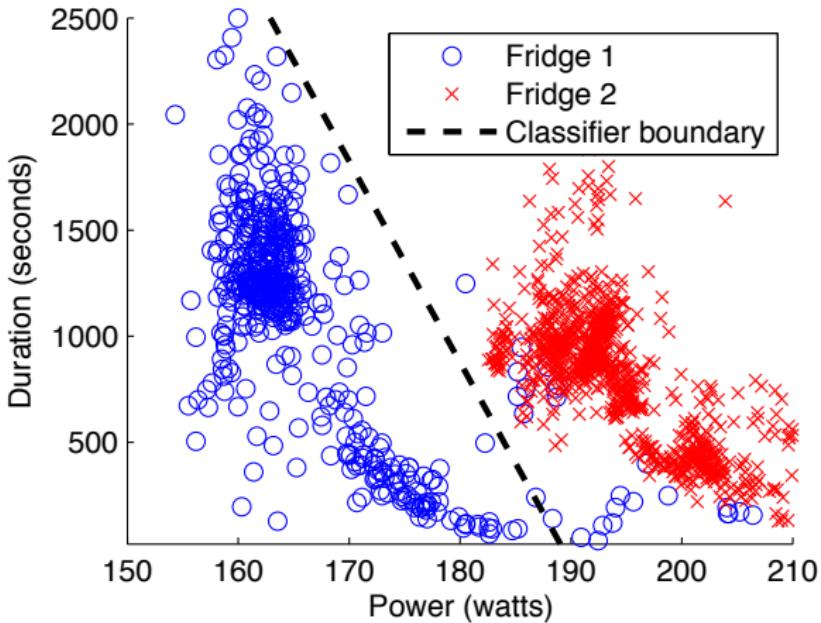
Support vector machines

Support vector machine is just regularized hinge loss and linear prediction (caveat, also common to use “kernel” hypothesis function, more later)

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^m \max\{1 - y^{(i)} \cdot x^{(i)T} \theta, 0\} + \lambda \sum_{i=1}^n \theta_i^2$$

Gradient descent update, repeat:

$$\theta := \theta - \alpha \left(- \sum_{i=1}^m y^{(i)} x^{(i)} \mathbf{1}\{y^{(i)} \cdot x^{(i)T} \theta < 1\} + 2\lambda \sum_{i=1}^n \theta_i \right)$$



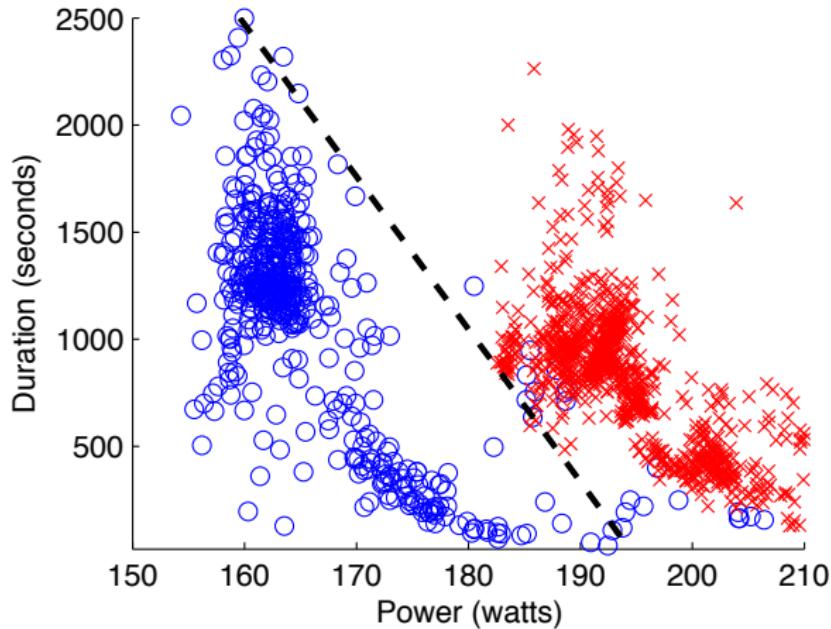
Classification boundary of support vector machine

Logistic regression

Logistic regression uses logistic loss

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^m \log(1 + e^{-y \cdot x^{(i)T} \theta}) + \lambda \sum_{i=1}^n \theta_i^2$$

Again, gradient descent is a reasonable algorithm (can you derive an equation for the gradient?)



Classification boundary of logistic regression

Probabilistic interpretation of logistic regression

Like least squares, logistic regression has a probabilistic interpretation

For binary classification problem, suppose that

$$p(y|x; \theta) = \frac{1}{1 + \exp(-y \cdot h_{\theta}(x))}$$

and for each data point $x^{(i)}$, $y^{(i)}$ is sampled randomly from this distribution

Then

$$\begin{aligned} & \underset{\theta}{\text{minimize}} - \sum_{i=1}^m \log p(y^{(i)}|x^{(i)}; \theta) \\ & \equiv \underset{\theta}{\text{minimize}} \log \left(1 + \exp \left(-y^{(i)} \cdot h_{\theta}(x^{(i)}) \right) \right) \end{aligned}$$

Multi-class classification

When classification is not binary $y \in 0, 1, \dots, k$ (i.e., classifying digit images), a common approach is “one-vs-all” method

Create a new set of y 's for the binary classification problem “is the label of this example equal to j ”

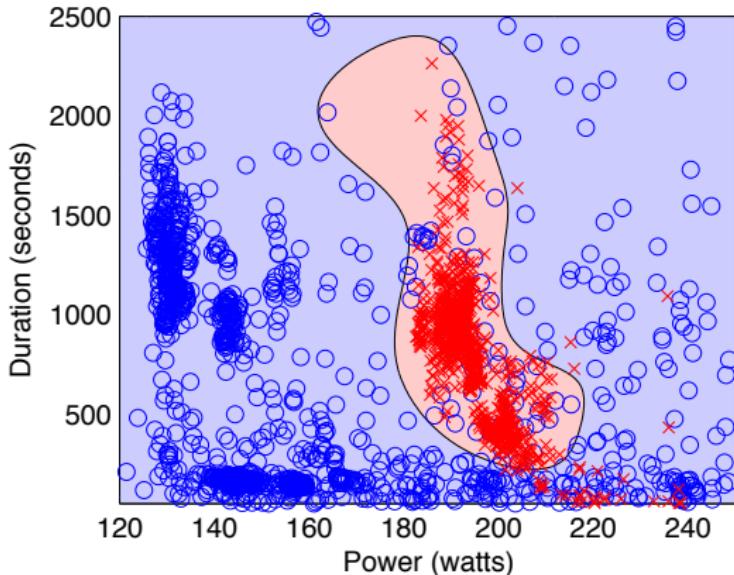
$$\hat{y}^{(i)} = \begin{cases} 1 & \text{if } y^{(i)} = j \\ -1 & \text{otherwise} \end{cases}$$

and solve for the corresponding parameter θ^j

For input x , classify according to the hypothesis with the highest confidence: $\operatorname{argmax}_j h_{\theta^j}(x)$

Non-linear classification

Same exact approach as in the regression case: use non-linear features of input to capture non-linear decision boundaries



Classification boundary of support vector machine using non-linear features

Outline

What is machine learning?

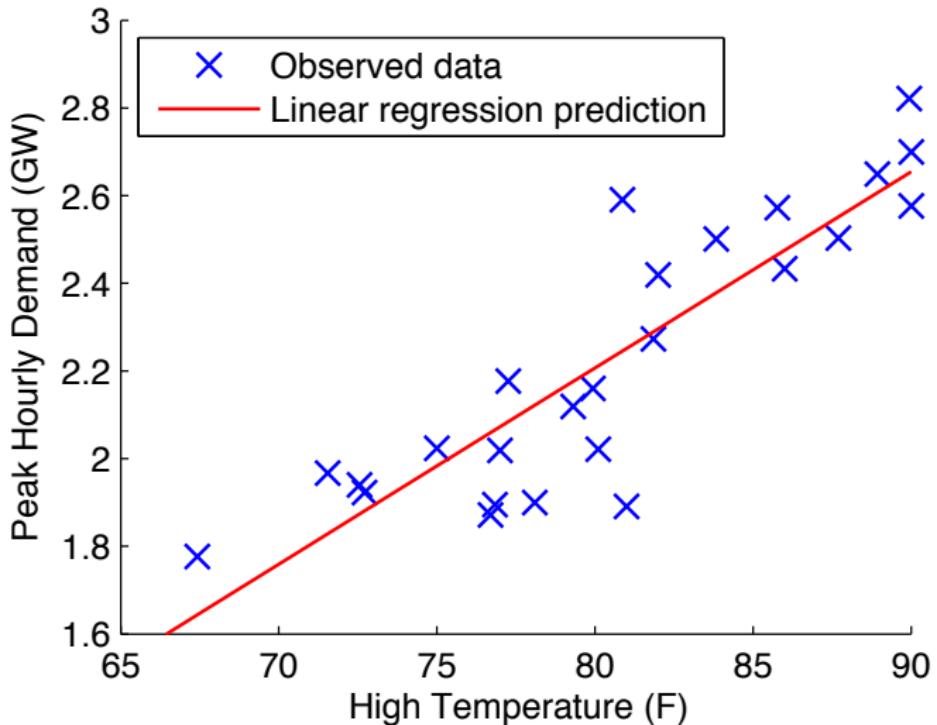
Supervised learning: regression

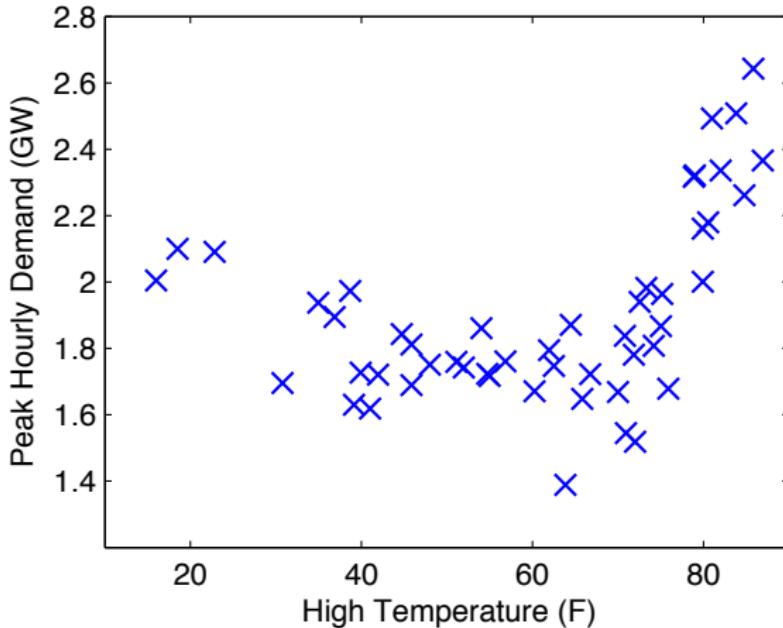
“Non-linear” regression, overfitting, and model selection

Supervised learning: classification

Other machine learning algorithms

Unsupervised learning





Several days of peak demand vs. high temperature in Pittsburgh over all months

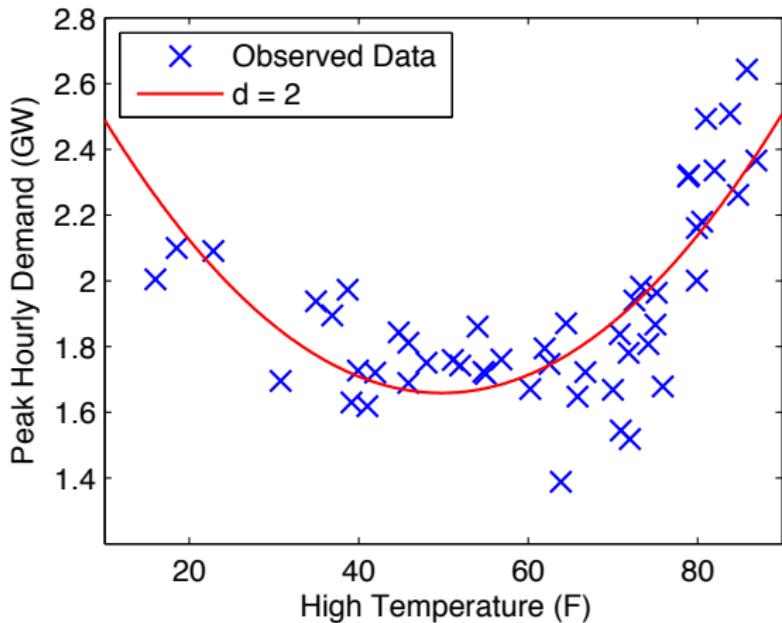
Overfitting

Though they may seem limited, linear hypothesis classes are very powerful, since the input features can themselves include non-linear features of data

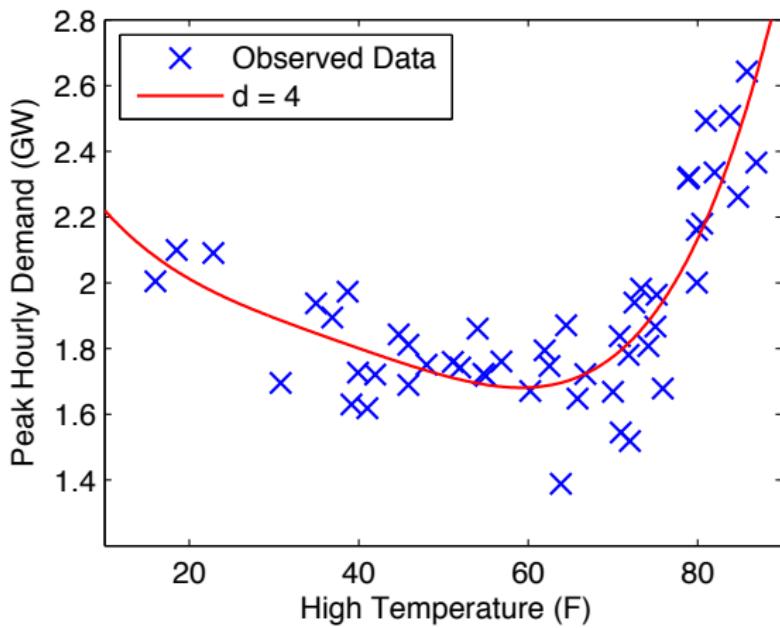
$$x^{(i)} \in \mathbb{R}^3 = \begin{bmatrix} (\text{high temperature for day } i)^2 \\ \text{high temperature for day } i \\ 1 \end{bmatrix}$$

In this case, $h_{\theta}(x) = x^T \theta$ will be a non-linear function of “original” data (i.e., predicted peak power is a non-linear function of high temperature)

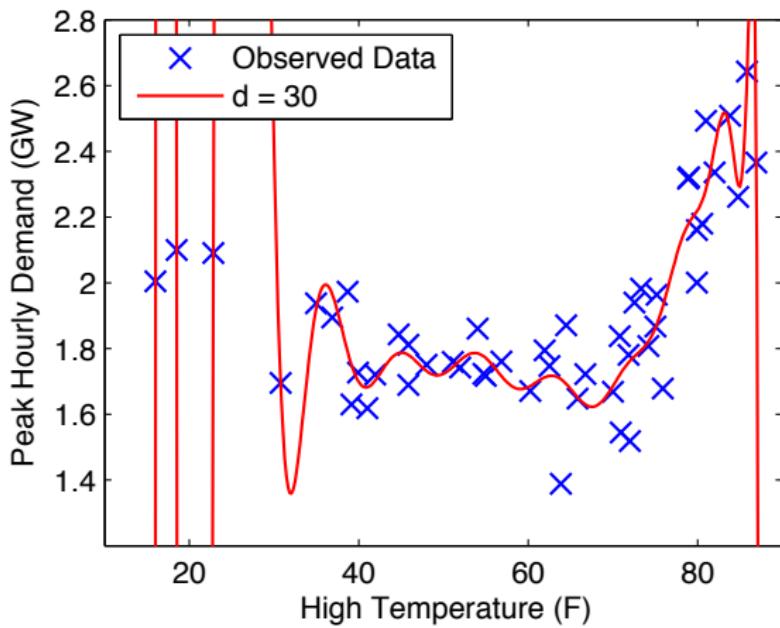
Same solution method as before, gradient descent or (for squared loss) analytical solution



Linear regression with second degree polynomial features



Linear regression with fourth degree polynomial features



Linear regression with 30th degree polynomial features

Training and validation loss

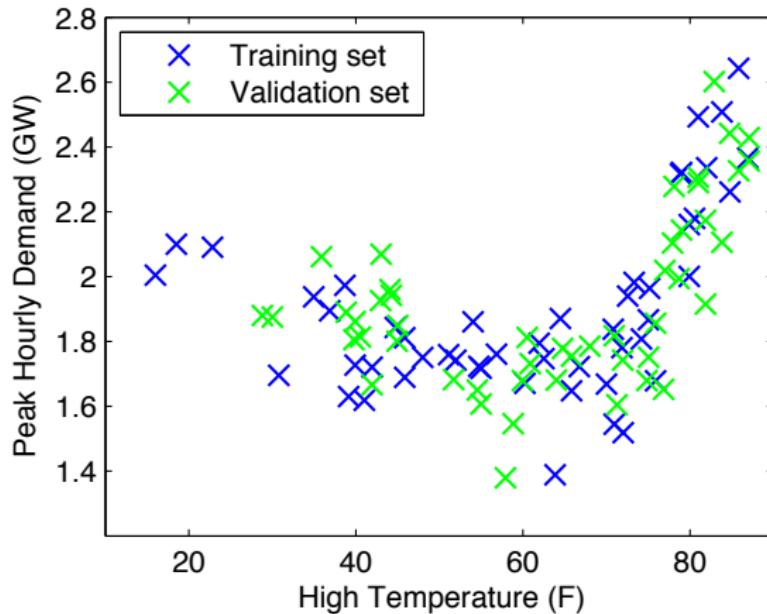
Fundamental problem: we are looking for parameters that optimize

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^m \ell(h_{\theta}(x^{(i)}), y^{(i)})$$

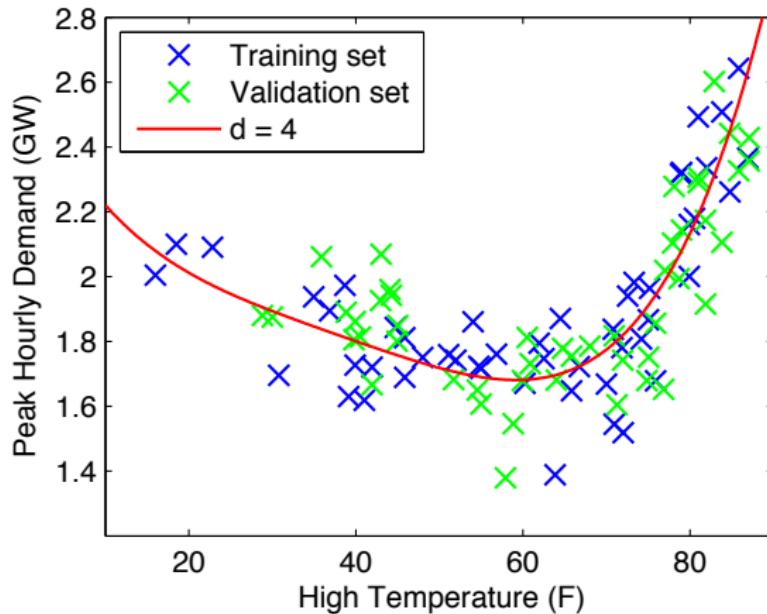
but what we really care about is loss of prediction on *new* examples (x', y') (also called *generalization error*)

Divide data into *training set* (used to find parameters for a fixed hypothesis class h_{θ}), and *validation set* (used to choose hypothesis class)

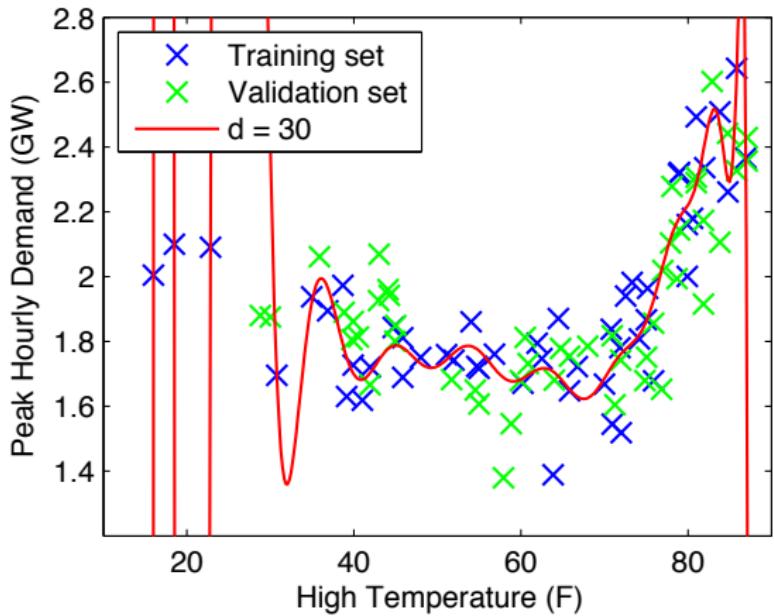
- (Slightly abusing notation here, we're going to wrap the "degree" of the input features into the hypothesis class h_{θ})



Training set and validation set

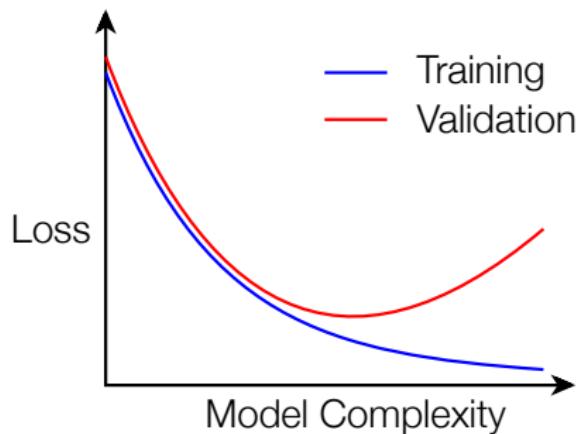


Training set and validation set, fourth degree polynomial

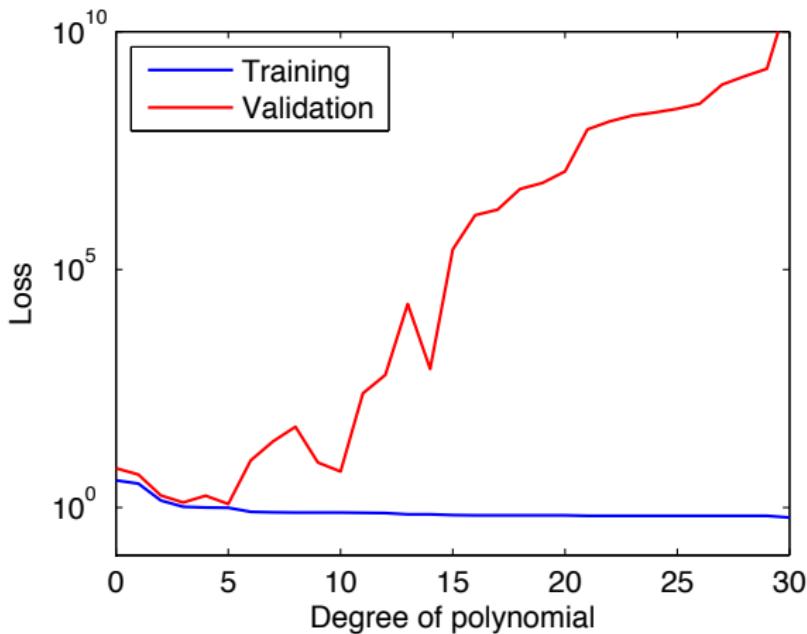


Training set and validation set, 30th degree polynomial

General intuition for training and validation loss



We would like to choose hypothesis class that is at the “sweet spot” of minimizing validation loss



Training and validation loss on peak demand prediction

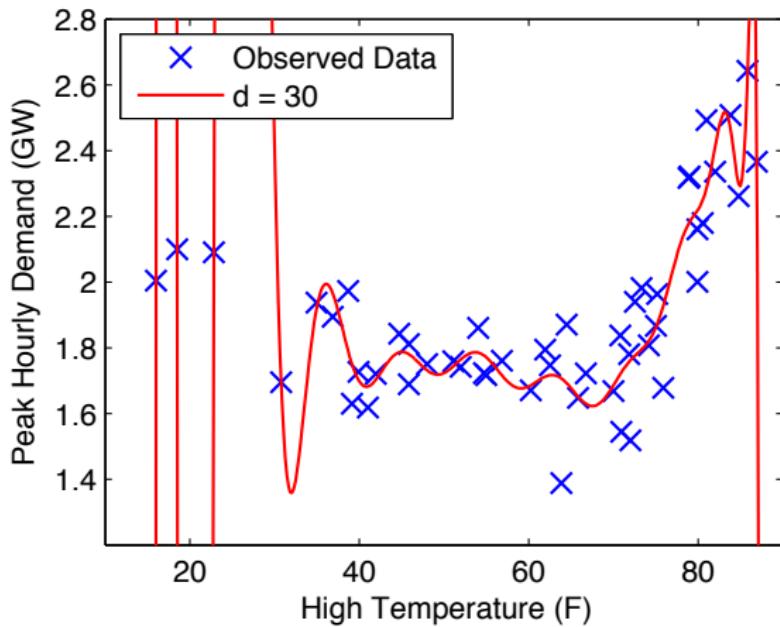
Model complexity and regularization

A number of different ways to control “model complexity”

An obvious one we have just seen: keep the number of features (number of parameters) low

A less obvious method: keep the *magnitude* of the parameters small

Intuition: a 30th degree polynomial that passes exactly through many of the data points requires very large entries in θ

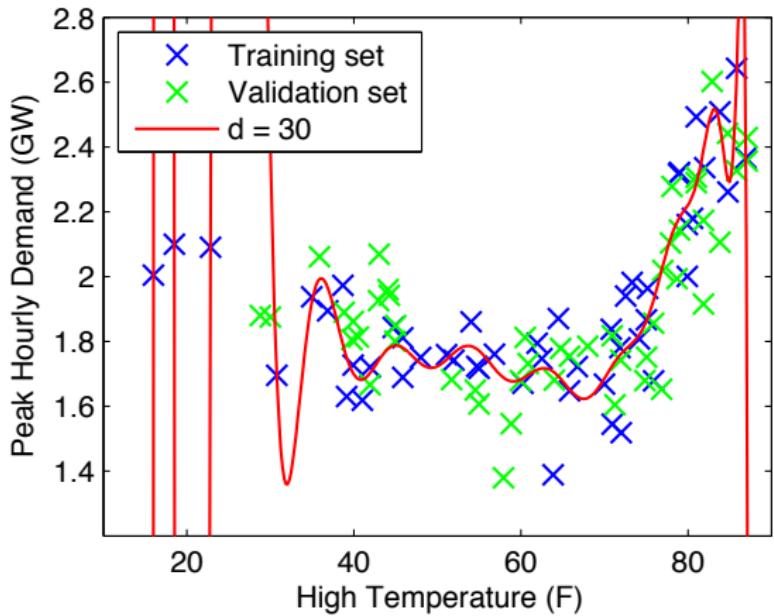


We can directly prevent large entries in θ by penalizing the magnitude of its entries

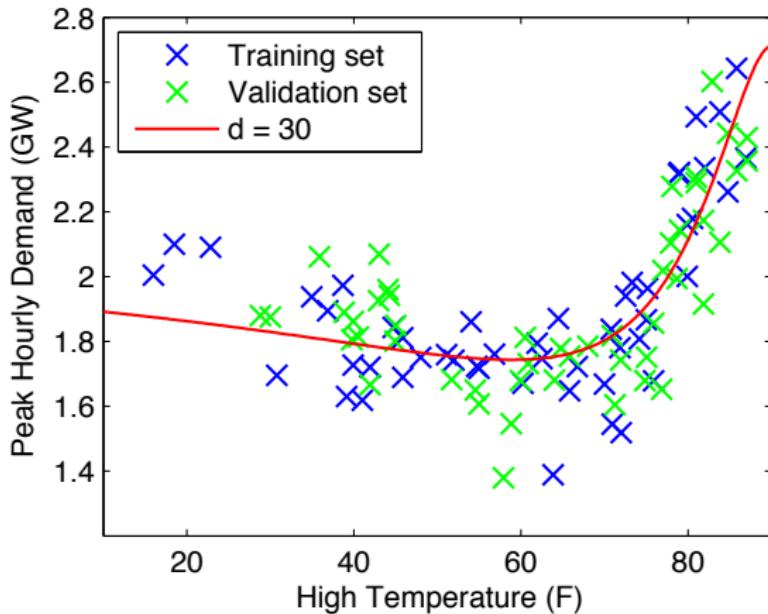
Leads to *regularized loss minimization* problem

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^m \ell\left(h_{\theta}(x^{(i)}), y^{(i)}\right) + \lambda \sum_{i=1}^n \theta_i^2$$

where $\lambda \in \mathbb{R}_+$ is a *regularization parameter* that weights the relative penalties of the size of θ and the loss



Degree 30 polynomial, with $\lambda = 0$ (unregularized)



Degree 30 polynomial, with $\lambda = 1$

Evaluating ML algorithms

The proper way to evaluate an ML algorithm:

1. Break all data into training/testing sets (e.g., 70%/30%)
2. Break training set into training/validation set (e.g., 70%/30% again)
3. Choose hyperparameters using validation set
4. (Optional) Once we have selected hyperparameters, retrain using all the training set
5. Evaluate performance on the testing set

Outline

What is machine learning?

Supervised learning: regression

“Non-linear” regression, overfitting, and model selection

Supervised learning: classification

Other machine learning algorithms

Unsupervised learning

Kernel methods

Kernel methods are a very popular approach to non-linear classification, though they are still “linear” in some sense

$$h_{\theta}(x) = \sum_{i=1}^m \theta_i K(x, x^{(i)})$$

where $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ is a *kernel function* that measures the similarity between x and $x^{(i)}$ (larger values for more similar)

For certain K , can be interpreted as working in a high dimensional feature space without explicitly forming features

Still linear in θ , can use many of the same algorithms as before

Important: $\theta \in \mathbb{R}^m$, as many parameters as examples
(*nonparametric* approach)

Nearest neighbor methods

Predict output based upon closest example in training set

$$h_{\theta}(x) = y^{(\operatorname{argmin}_i \|x - x^{(i)}\|^2)}$$

$$\text{where } \|x\|^2 = \sum_{i=1}^n x_i^2$$

Can also average over k closest examples: k -nearest neighbor

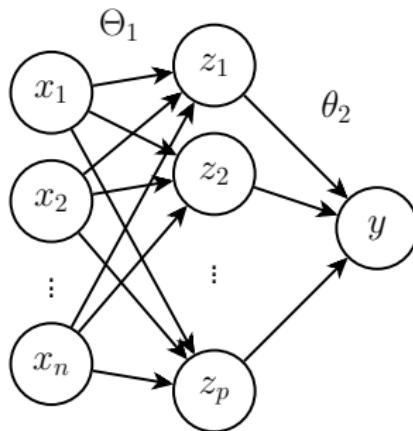
Requires no separate “training” phase, but (like kernel methods) it is nonparametric, requires that we keep around all the data

Neural networks

Non-linear hypothesis class

$$h_{\theta}(x) = \sigma(\theta_2^T \sigma(\Theta_1^T x))$$

for a 2-layer network, where $\theta = \{\Theta_1 \in \mathbb{R}^{n \times p}, \theta_2 \in \mathbb{R}^p$ and $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is a sigmoid function $\sigma(z) = 1/(1 + \exp(-z))$ (applied elementwise to vector)

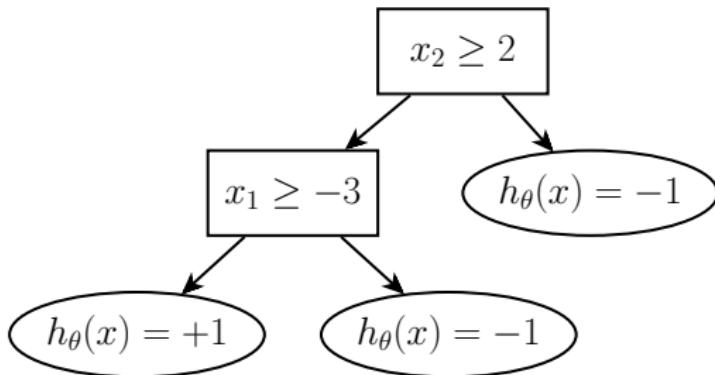


Non-convex optimization, but smooth (gradient and similar methods can work very well)

Some major recent success stories in speech recognition, image classification

Decision trees

Hypothesis class partitions space into different regions



Can also have linear predictors (regression or classification) at the leaves

Greedy training find nodes that best separate data into distinct classes

Ensemble methods

Combine a number of different hypotheses

$$h_{\theta}(x) = \sum_{i=1}^k \theta_i \text{sign}(h_i(x))$$

Popular instances

- Random forests: ensemble of decision trees built from different subsets of training data
- Boosting: iteratively train multiple classifiers/regressors on reweighted examples based upon performance of the previous hypothesis

Outline

What is machine learning?

Supervised learning: regression

“Non-linear” regression, overfitting, and model selection

Supervised learning: classification

Other machine learning algorithms

Unsupervised learning

Supervised learning

Training Data

$(2, 2)$
 $(0, 0)$
 $(8, 8)$
 $(5, 5)$

:

Machine Learning

→ Hypothesis
function
 h_{θ}

Deployment

Prediction = $h_{\theta}(2)$

Prediction = $h_{\theta}(5)$

:

Unsupervised learning

Training Data

$\begin{pmatrix} 2 \\ 0 \\ 8 \\ 5 \\ \vdots \end{pmatrix}$

Machine Learning

→ Hypothesis
function
 h_{θ}

Deployment

Prediction = $h_{\theta} \begin{pmatrix} 2 \end{pmatrix}$

Prediction = $h_{\theta} \begin{pmatrix} 5 \end{pmatrix}$

⋮

Problem setting

Input features: $x^{(i)} \in \mathbb{R}^n$, $i = 1, \dots, m$

Model parameters: $\theta \in \mathbb{R}^k$

How do we specify a hypothesis class or loss function without outputs?

One way to interpret many unsupervised learning algorithms is that they try to “re-create” the input using a limited hypothesis class

Hypothesis function: $h_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^n$

- Want $h_\theta(x^{(i)}) \approx x^{(i)}$ for all training data

Loss function: $\ell : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}_+$

- E.g., $\ell(h_\theta(x), x) = \|h_\theta(x) - x\|^2$

In order to prevent the trivial solution $h_\theta(x) = x$, we need to restrict the class of allowable functions h_θ

k-means

Parameters are a set of k “centers” in the data

$$\theta = \{\mu^{(1)}, \dots, \mu^{(k)}\}, \mu^{(i)} \in \mathbb{R}^k$$

Hypothesis class picks the closest center

$$h_\theta(x) = \mu^{\left(\operatorname{argmin}_i \|x - \mu^{(i)}\|^2\right)}$$

With this framework, training looks the same as supervised learning

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^m \|x^{(i)} - h_\theta(x^{(i)})\|^2$$

Not a convex problem, but can solve by iteratively finding the closest $\mu^{(i)}$ for each example, then setting $\mu^{(i)}$ to be the mean of all examples assigned to it

Principal component analysis

Parameters are two matrices that reduce the effective dimension of the data, $\theta = \{\Theta_1 \in \mathbb{R}^{n \times k}, \Theta_2 \in \mathbb{R}^{k \times n}\}$ with $k < n$

Hypothesis class $h_\theta(x) = \Theta_1 \Theta_2 x$

Interpretation: to reconstruct data, $\Theta_2 x \in \mathbb{R}^k$ needs to preserve “most” of the information in x (dimensionality reduction)

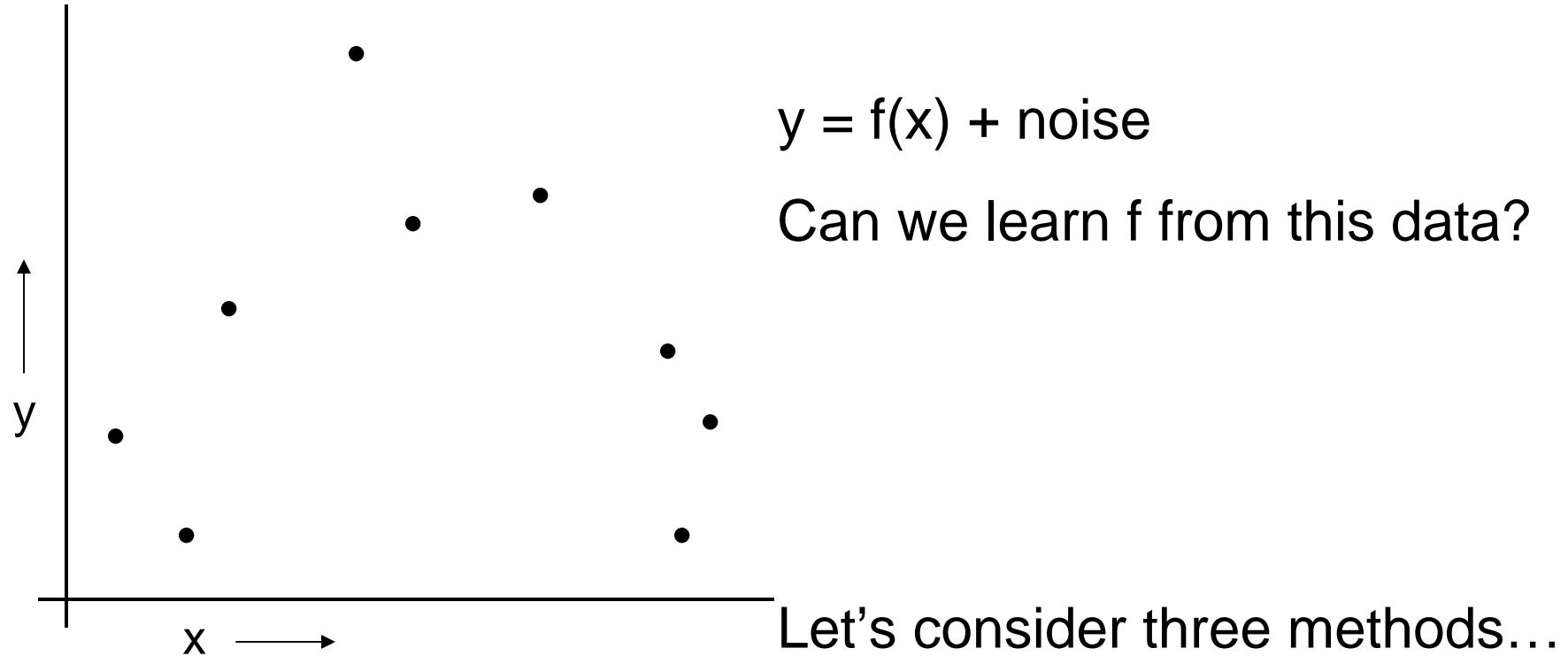
Minimizing loss

$$\underset{\Theta_1, \Theta_2}{\text{minimize}} \sum_{i=1}^m \|x^{(i)} - \Theta_1 \Theta_2 x^{(i)}\|_2^2$$

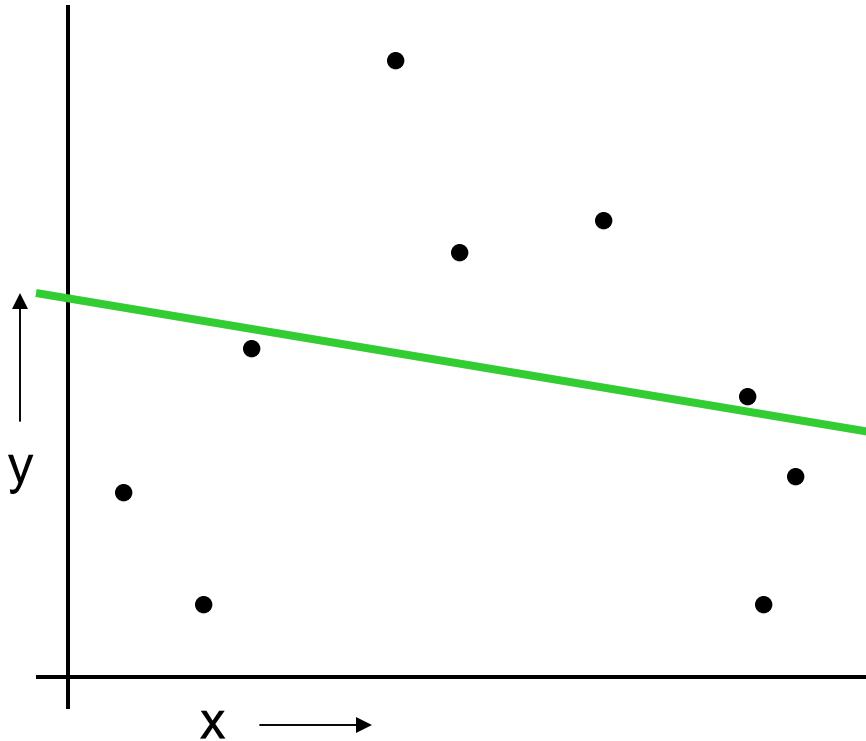
is not a convex problem, but can be solved (exactly) via an eigenvalue decomposition

K fold Cross Validation

A Regression Problem



Linear Regression



Linear Regression

Univariate Linear regression with a constant term:

X	Y
3	7
1	3
:	:


$$\mathbf{X} = \begin{pmatrix} 3 \\ 1 \\ \vdots \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} 7 \\ 3 \\ \vdots \end{pmatrix}$$
$$\mathbf{x}_1 = (3) \dots \quad \mathbf{y}_1 = 7 \dots$$

Originally discussed in the previous Andrew Lecture: "Neural Nets"

Linear Regression

Univariate Linear regression with a constant term:

X	Y
3	7
1	3
:	:

$$z = \begin{pmatrix} 1 & 3 \\ 1 & 1 \\ \vdots & \end{pmatrix}$$

$$z_1 = (1, 3) \dots$$

$$z_k = (1, x_k)$$

$$x = \begin{pmatrix} 3 \\ 1 \\ \vdots \end{pmatrix}$$

$$y = \begin{pmatrix} 7 \\ 3 \\ \vdots \end{pmatrix}$$

$$y = \begin{pmatrix} 7 \\ 3 \\ \vdots \end{pmatrix}$$

$$y_1 = 7 \dots$$

Linear Regression

Univariate Linear regression with a constant term:

X	Y
3	7
1	3
:	:

$$Z = \begin{bmatrix} 1 & 3 \\ 1 & 1 \\ \vdots & \vdots \end{bmatrix}$$

$$z_1 = (1, 3) \dots$$

$$z_k = (1, x_k)$$

$$X = \begin{bmatrix} 3 \\ 1 \\ \vdots \end{bmatrix}$$

$$y = \begin{bmatrix} 7 \\ 3 \\ \vdots \end{bmatrix}$$

$$y_1 = 7 \dots$$

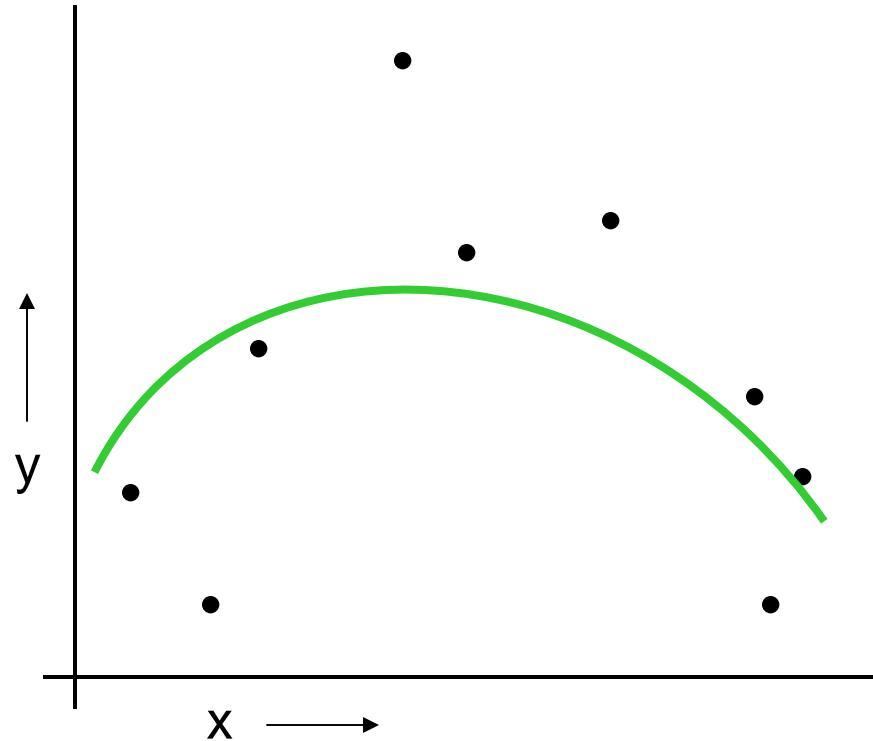
$$y = \begin{bmatrix} 7 \\ 3 \\ \vdots \end{bmatrix}$$

$$y_1 = 7 \dots$$

$$\beta = (Z^T Z)^{-1} (Z^T y)$$

$$y^{est} = \beta_0 + \beta_1 x$$

Quadratic Regression



Quadratic Regression

X	Y
3	7
1	3
.	.

$Z =$

1	3	9
1	1	1
:		

$$\mathbf{z} = (1, x, x^2)$$

$$\mathbf{x} = \begin{bmatrix} 3 \\ 1 \\ \vdots \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 7 \\ 3 \\ \vdots \end{bmatrix}$$

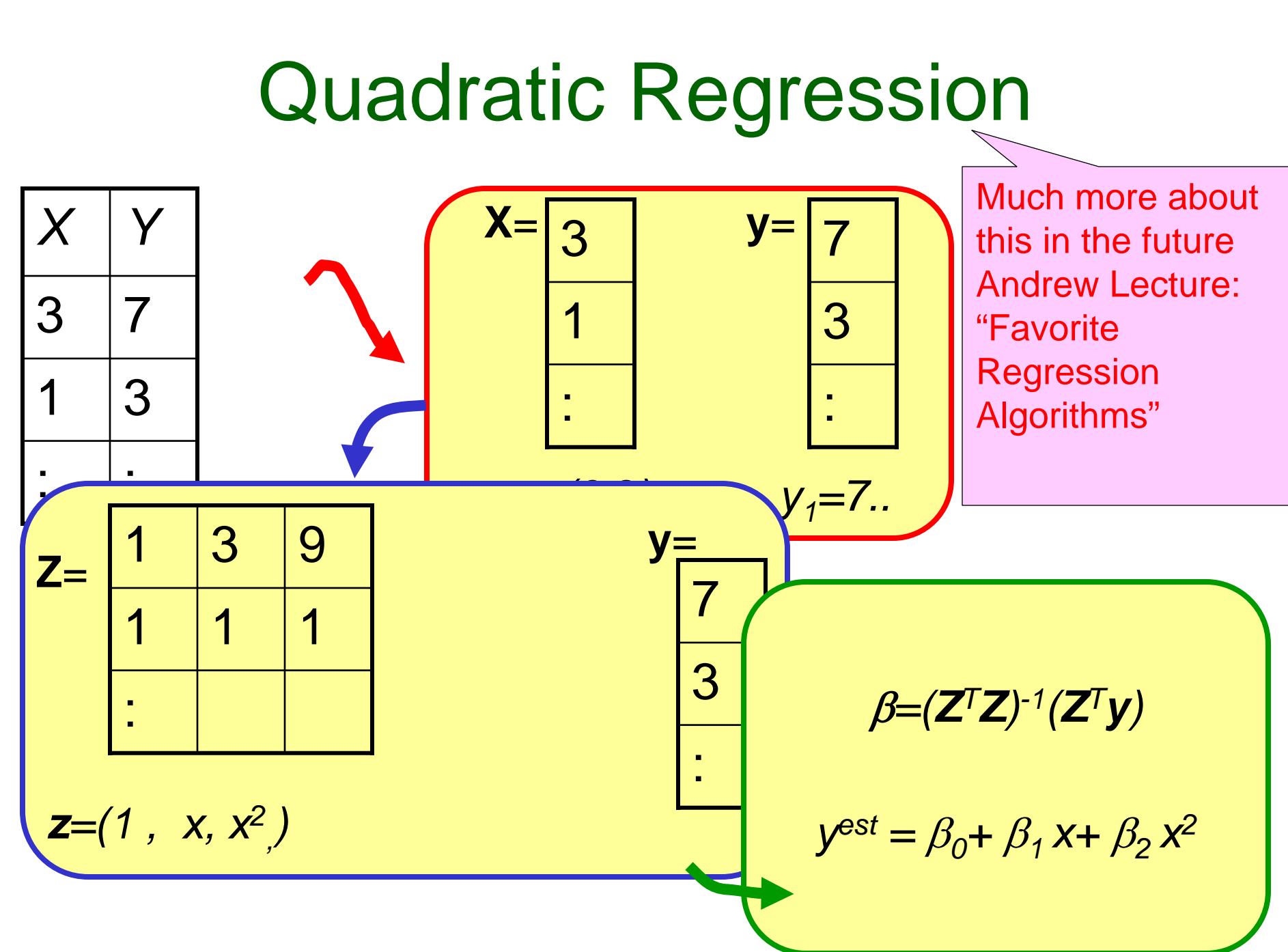
$$y_1 = 7..$$

$$\mathbf{y} = \begin{bmatrix} 7 \\ 3 \\ \vdots \end{bmatrix}$$

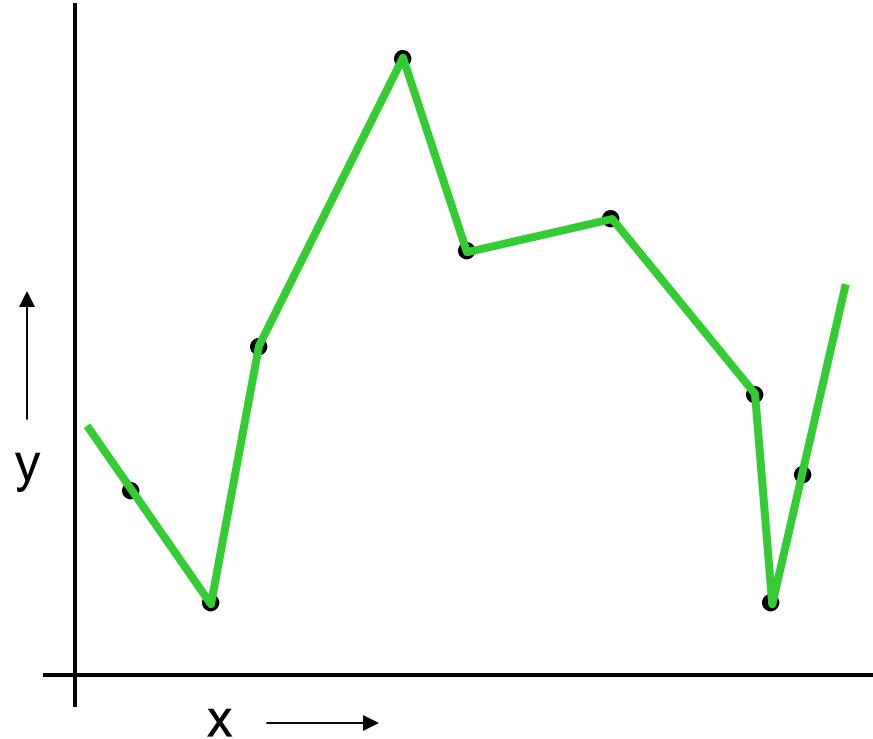
$$\beta = (\mathbf{Z}^T \mathbf{Z})^{-1} (\mathbf{Z}^T \mathbf{y})$$

$$y^{\text{est}} = \beta_0 + \beta_1 x + \beta_2 x^2$$

Much more about
this in the future
Andrew Lecture:
“Favorite
Regression
Algorithms”

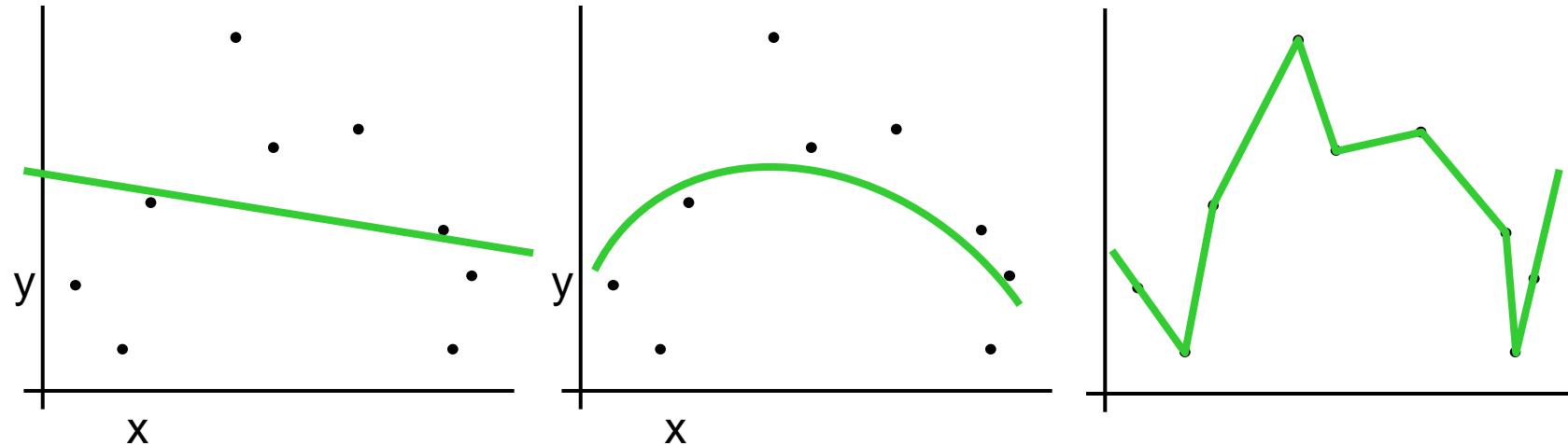


Join-the-dots



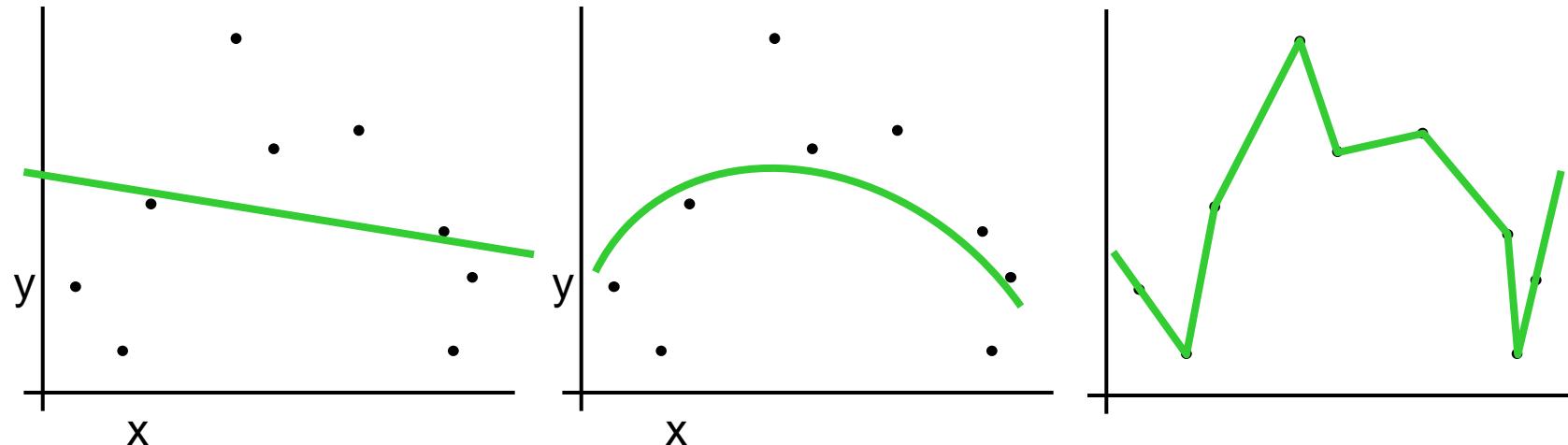
Also known as **piecewise
linear nonparametric
regression** if that makes
you feel better

Which is best?



Why not choose the method with the best fit to the data?

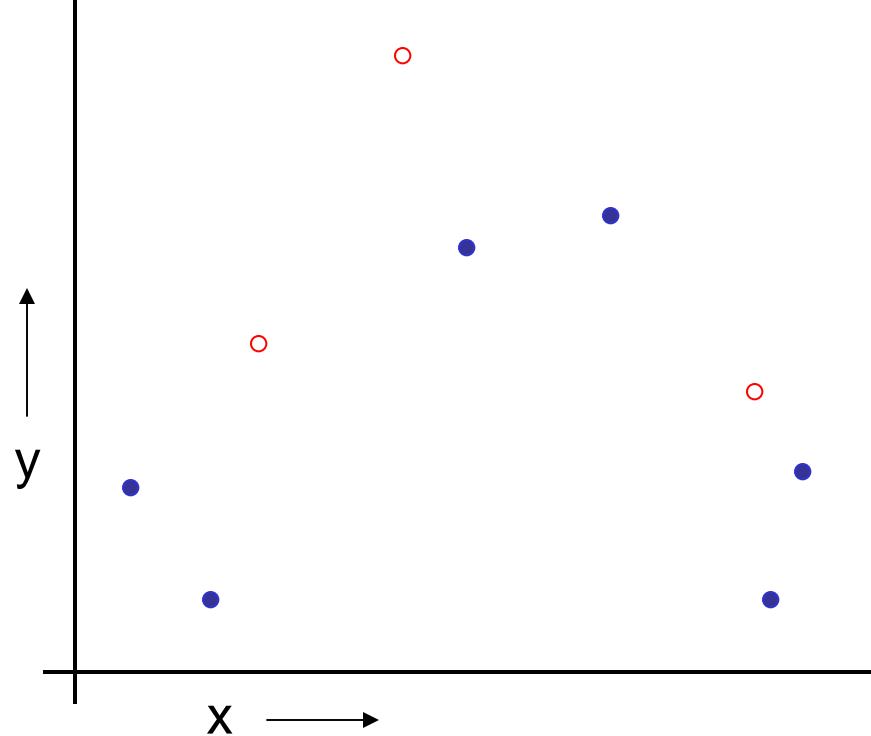
What do we really want?



Why not choose the method with the best fit to the data?

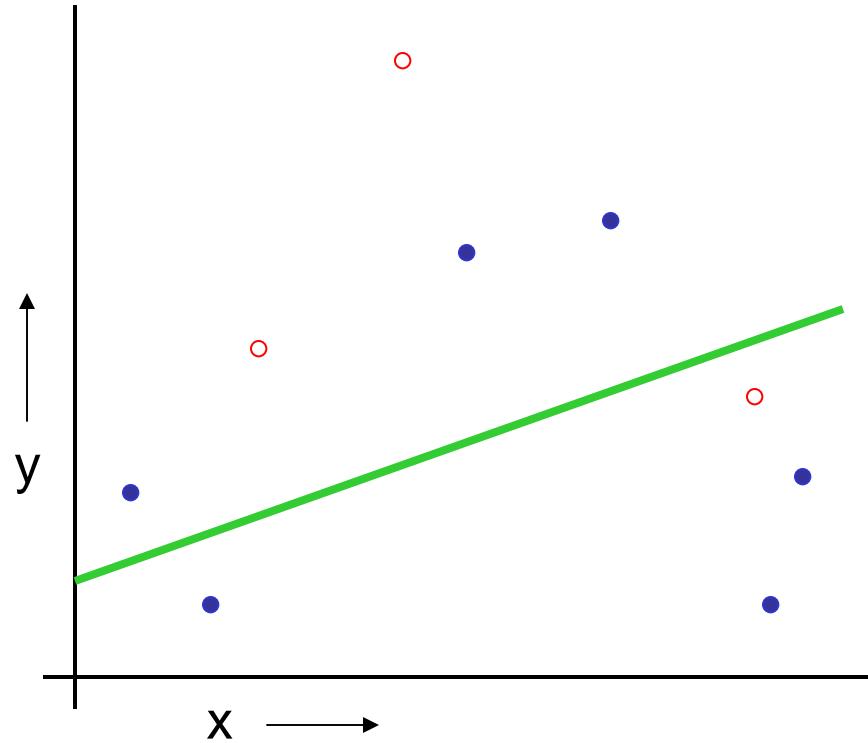
“How well are you going to predict future data drawn from the same distribution?”

The test set method



1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a **training set**

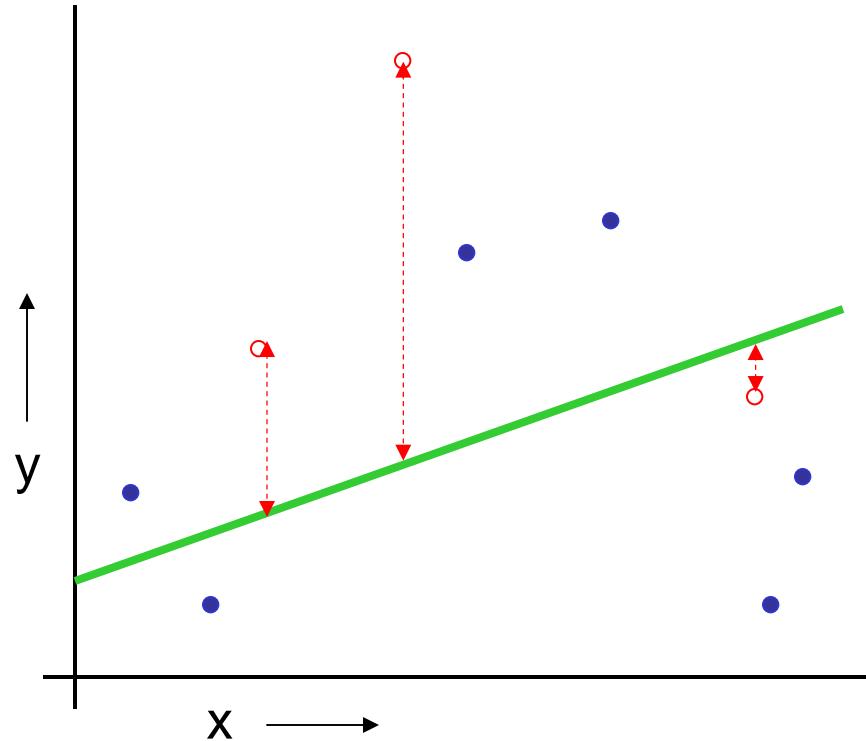
The test set method



1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a **training set**
3. Perform your regression on the training set

(Linear regression example)

The test set method

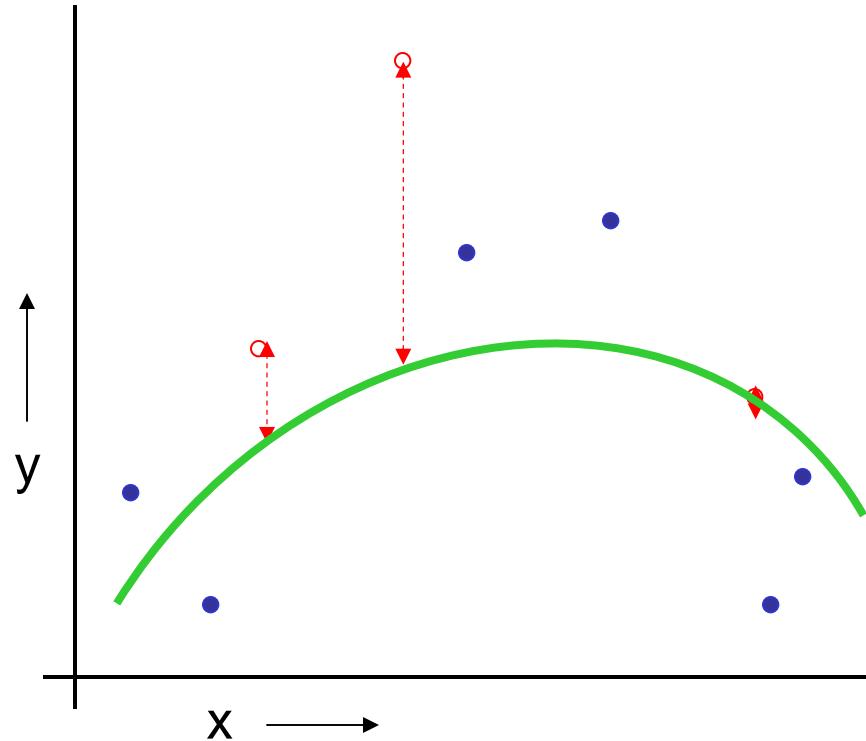


(Linear regression example)

Mean Squared Error = 2.4

1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a **training set**
3. Perform your regression on the training set
4. Estimate your future performance with the test set

The test set method

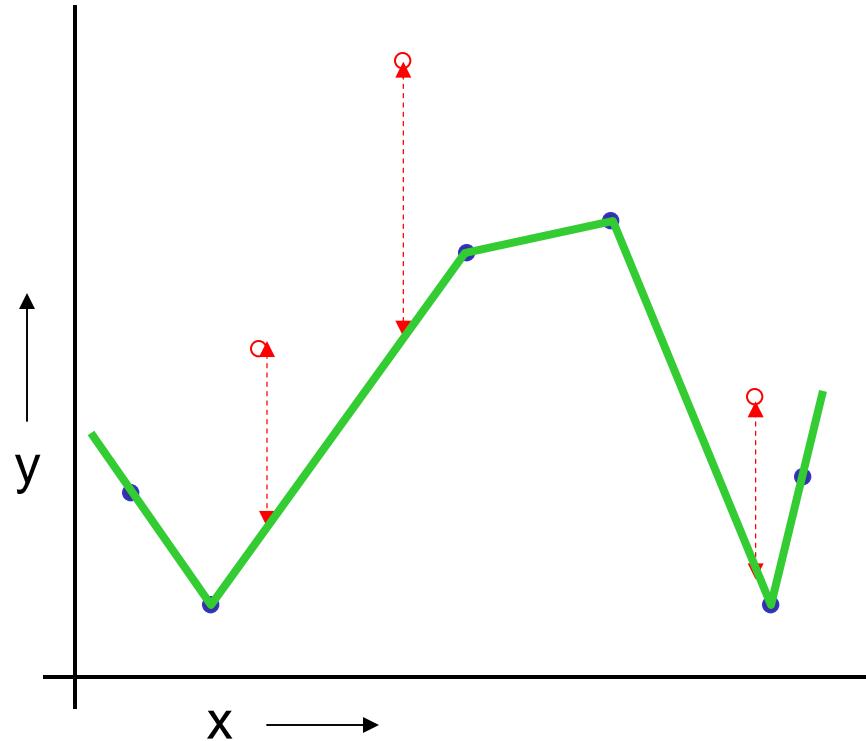


(Quadratic regression example)

Mean Squared Error = 0.9

1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a **training set**
3. Perform your regression on the training set
4. Estimate your future performance with the test set

The test set method



(Join the dots example)

Mean Squared Error = 2.2

1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a training set
3. Perform your regression on the training set
4. Estimate your future performance with the test set

The test set method

Good news:

- Very very simple
- Can then simply choose the method with the best test-set score

Bad news:

- What's the downside?

The test set method

Good news:

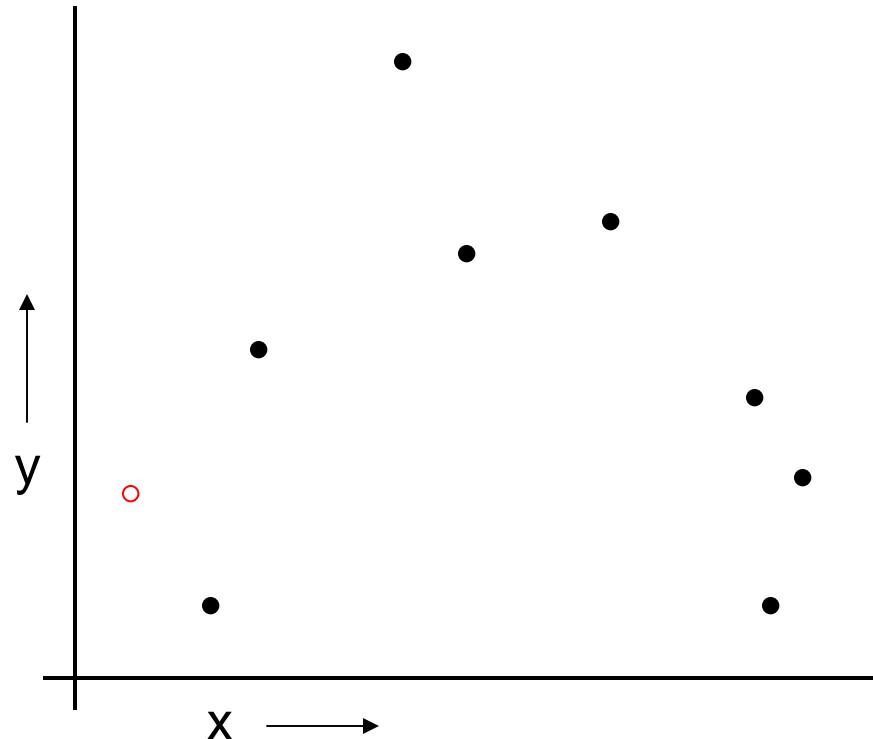
- Very very simple
- Can then simply choose the method with the best test-set score

Bad news:

- Wastes data: we get an estimate of the best method to apply to 30% less data
- If we don't have much data, our test-set might just be lucky or unlucky

We say the “test-set estimator of performance has high variance”

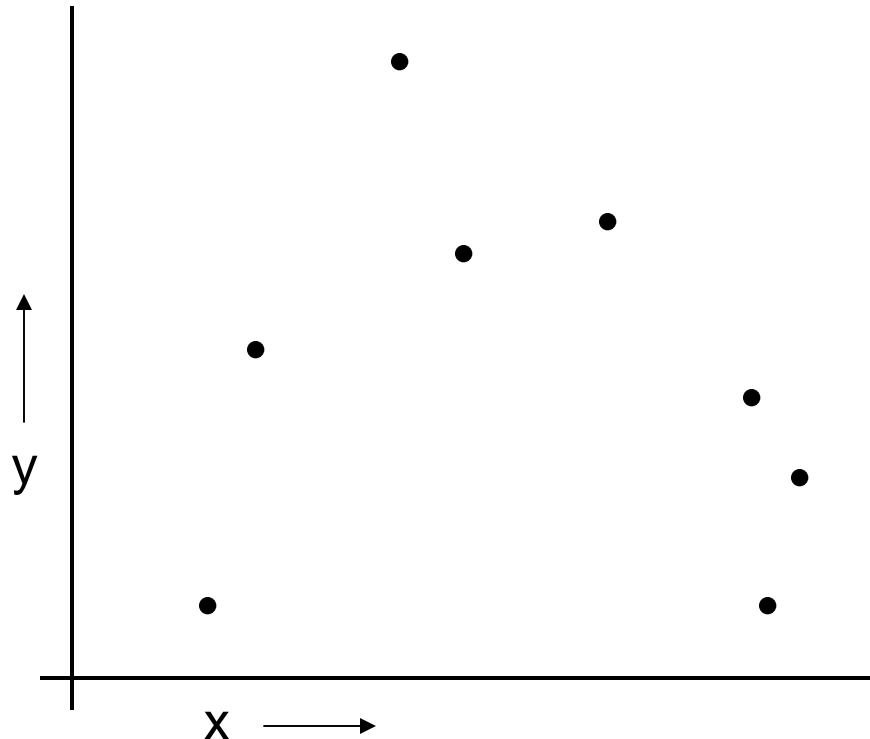
LOOCV (Leave-one-out Cross Validation)



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record

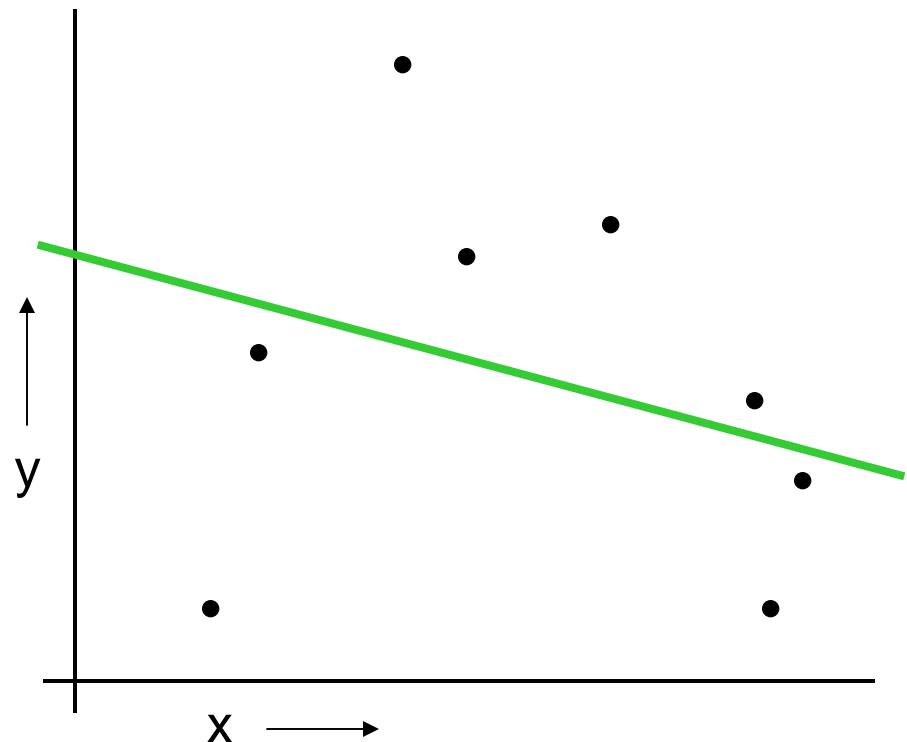
LOOCV (Leave-one-out Cross Validation)



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset

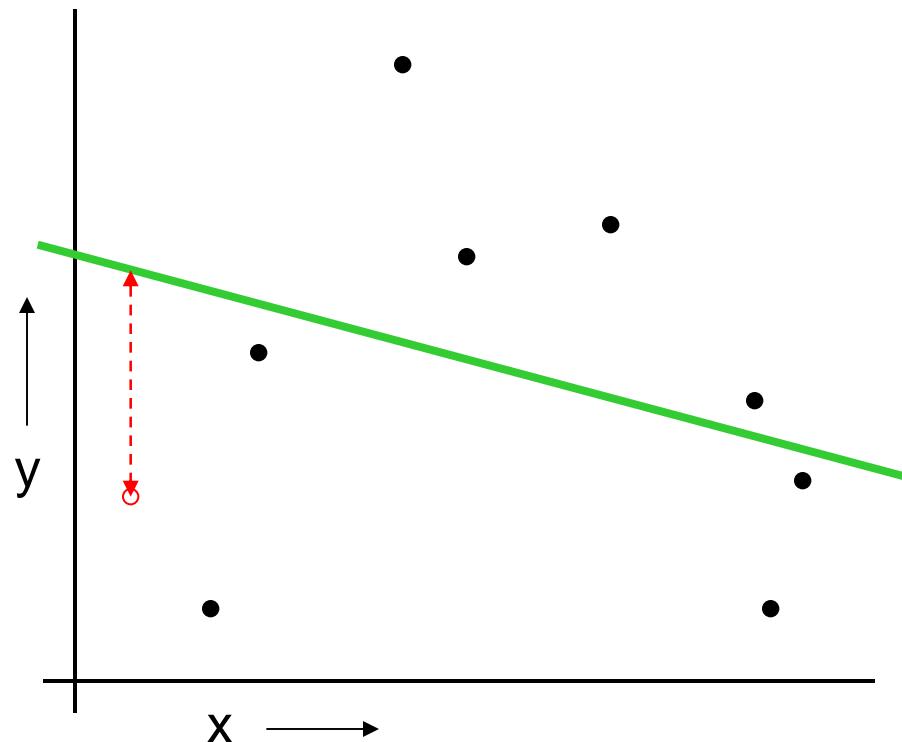
LOOCV (Leave-one-out Cross Validation)



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints

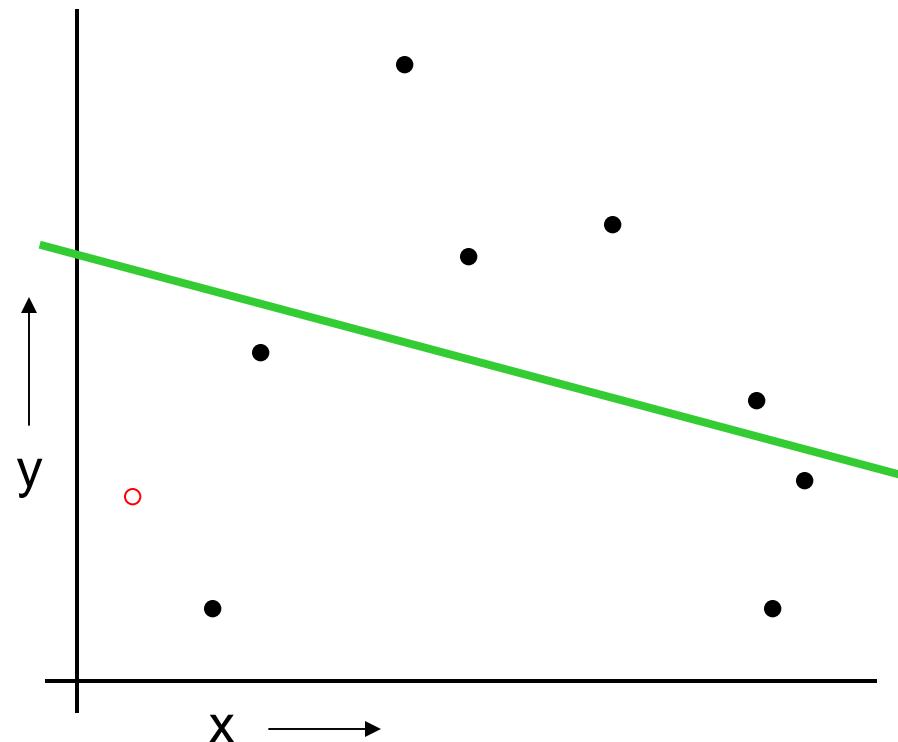
LOOCV (Leave-one-out Cross Validation)



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

LOOCV (Leave-one-out Cross Validation)

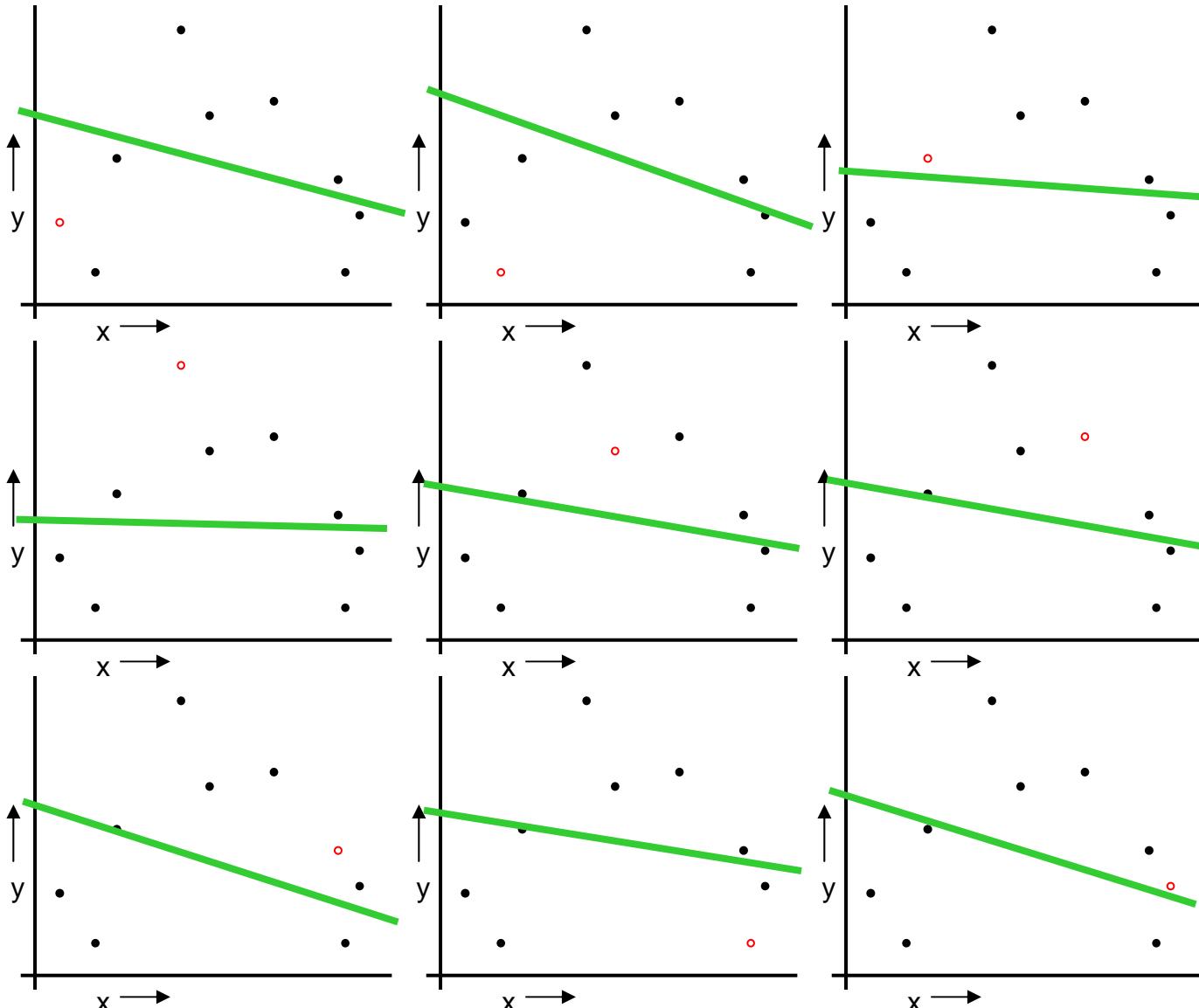


For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

When you've done all points,
report the mean error.

LOOCV (Leave-one-out Cross Validation)



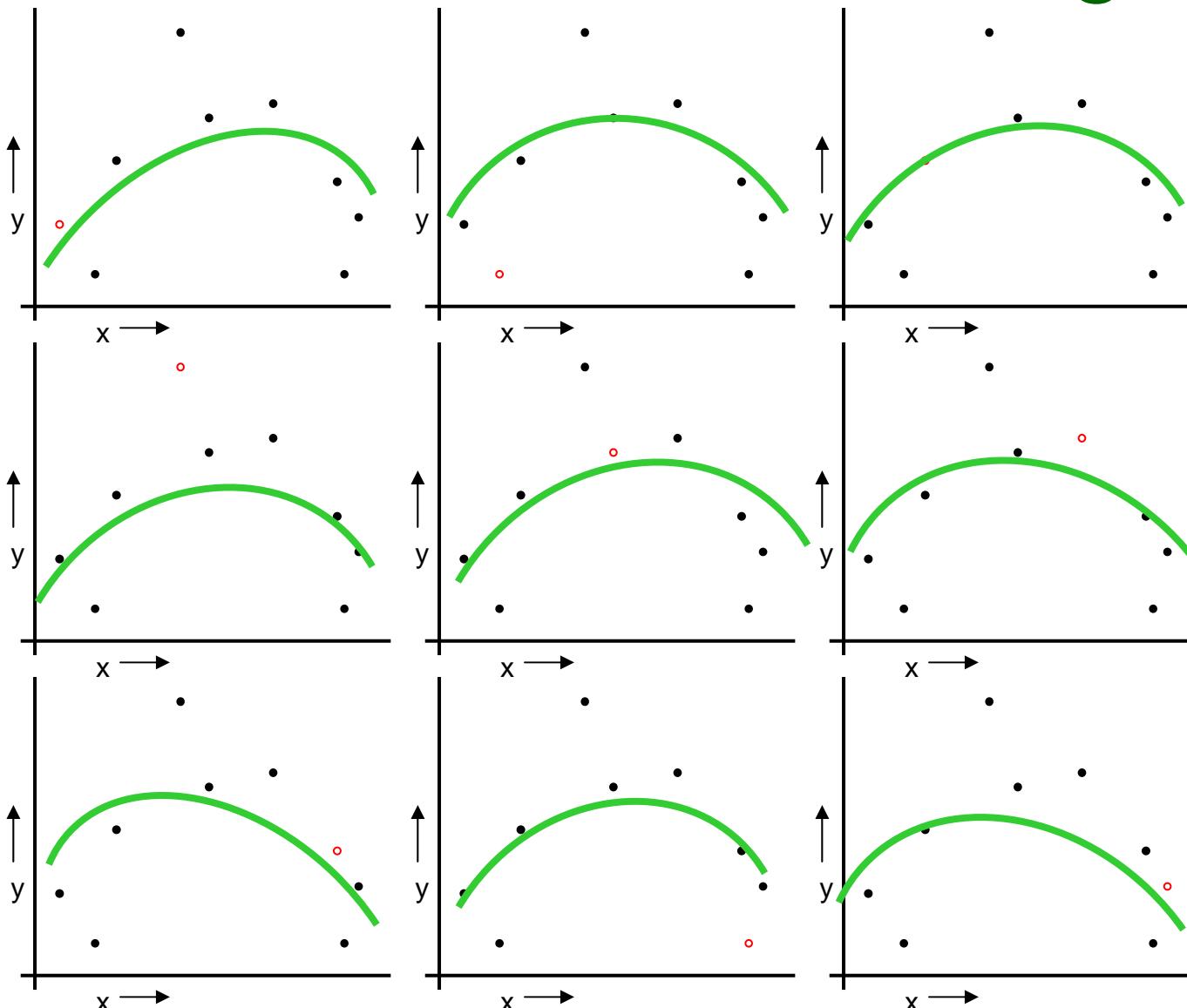
For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$$MSE_{LOOCV} = 2.12$$

LOOCV for Quadratic Regression



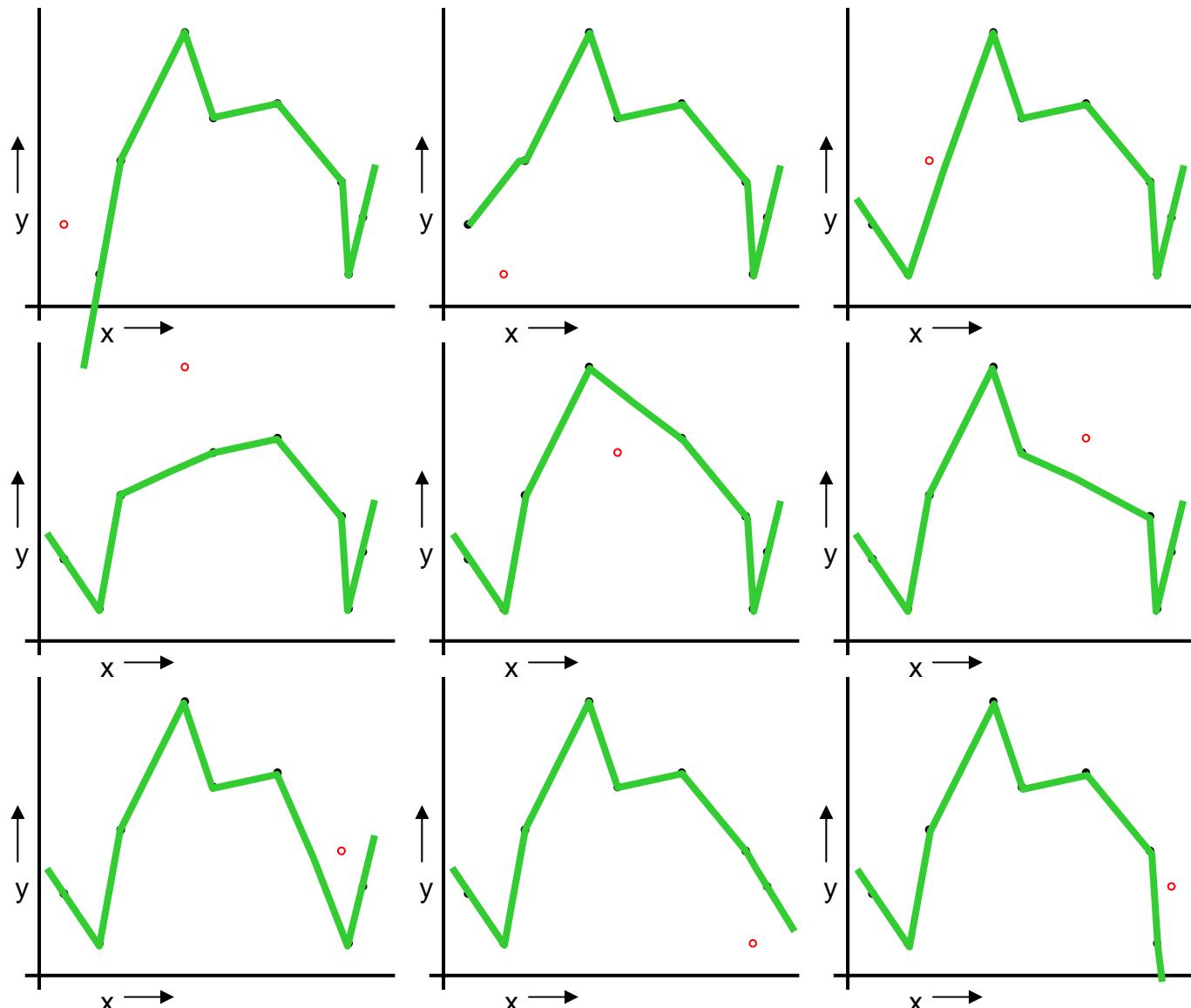
For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$$MSE_{LOOCV} = 0.962$$

LOOCV for Join The Dots



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$$MSE_{LOOCV} = 3.33$$

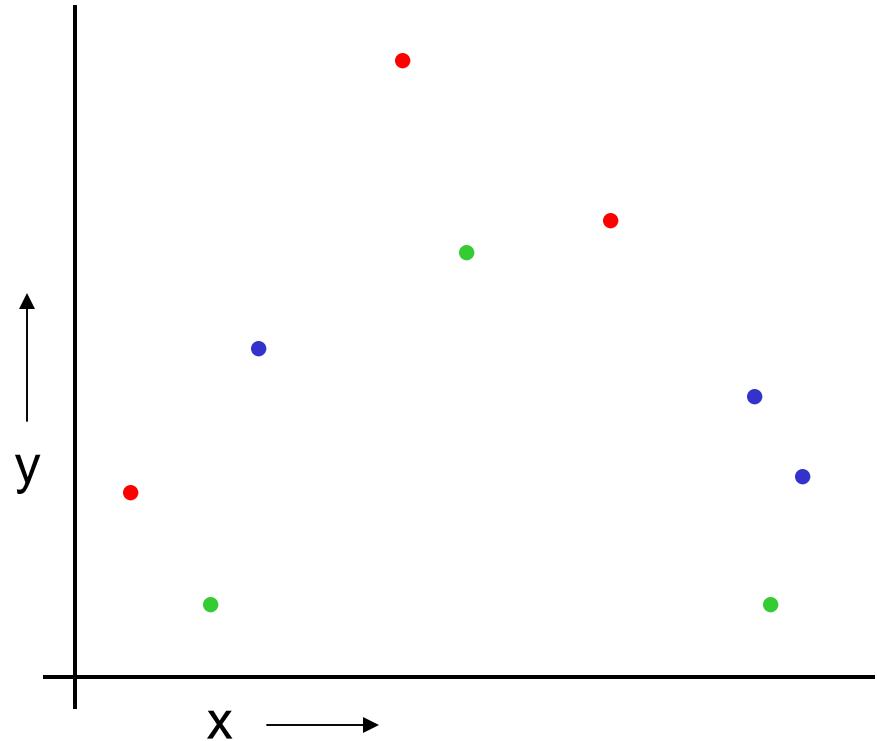
Which kind of Cross Validation?

	Downside	Upside
Test-set	Variance: unreliable estimate of future performance	Cheap
Leave-one-out	Expensive. Has some weird behavior	Doesn't waste data

..can we get the best of both worlds?

k-fold Cross Validation

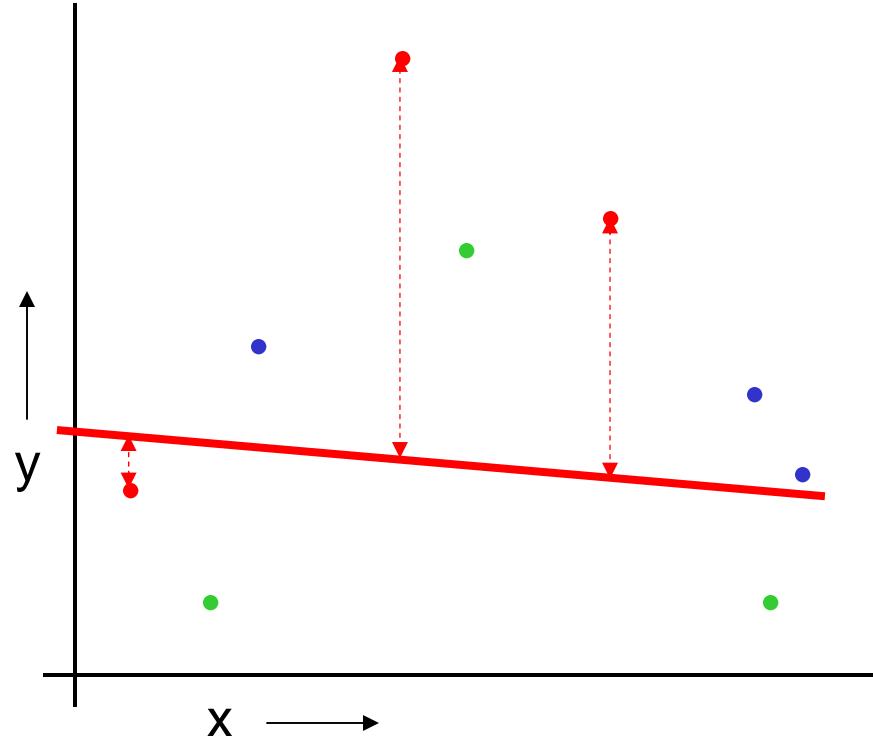
Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)



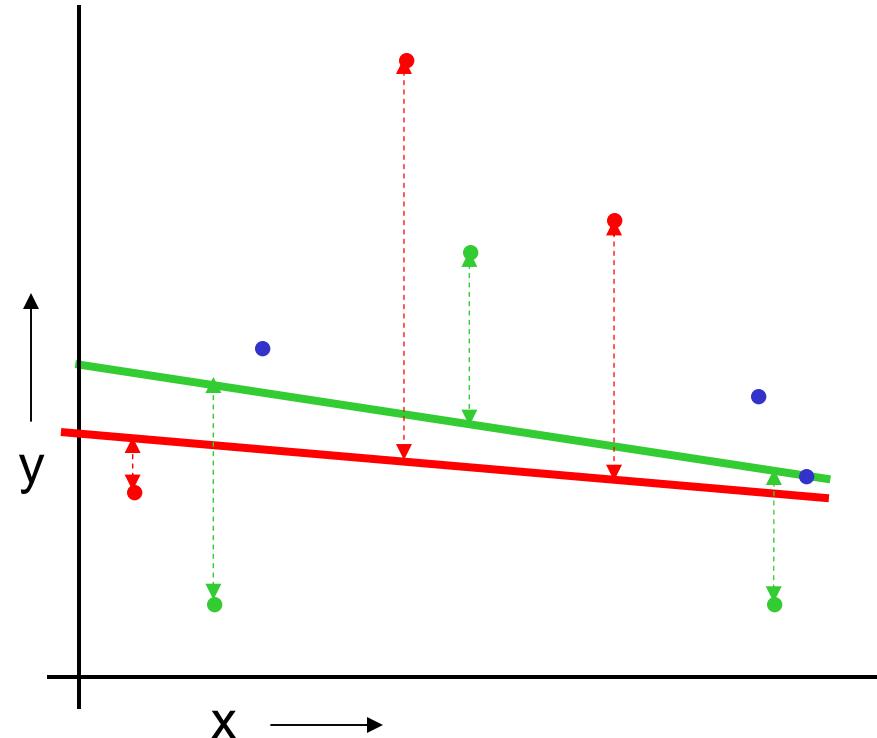
k-fold Cross Validation

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.



k-fold Cross Validation

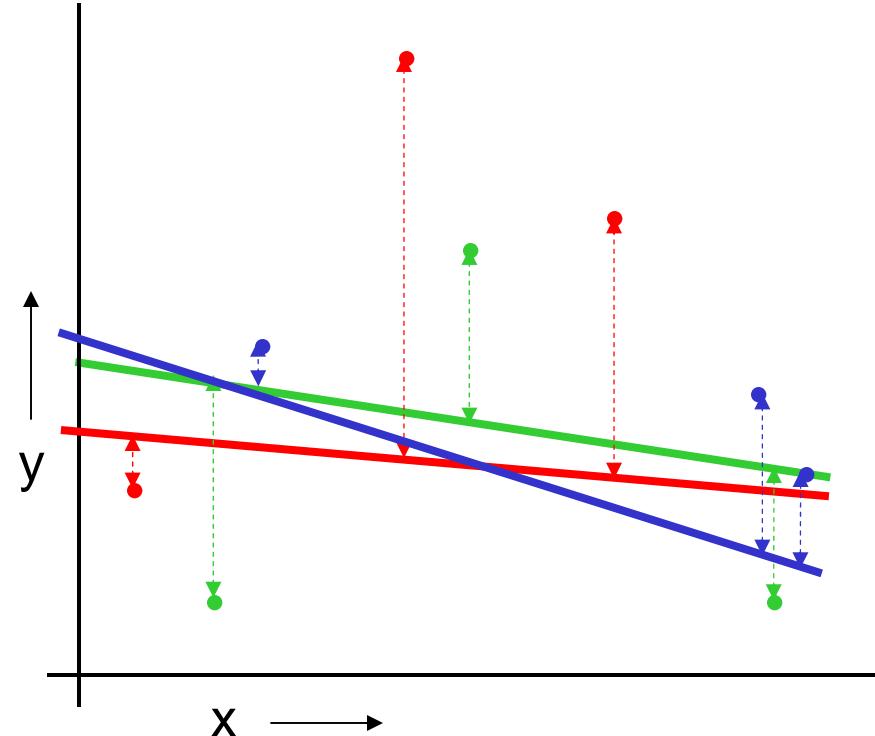


Randomly break the dataset into k partitions (in our example we'll have $k=3$ partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

k-fold Cross Validation



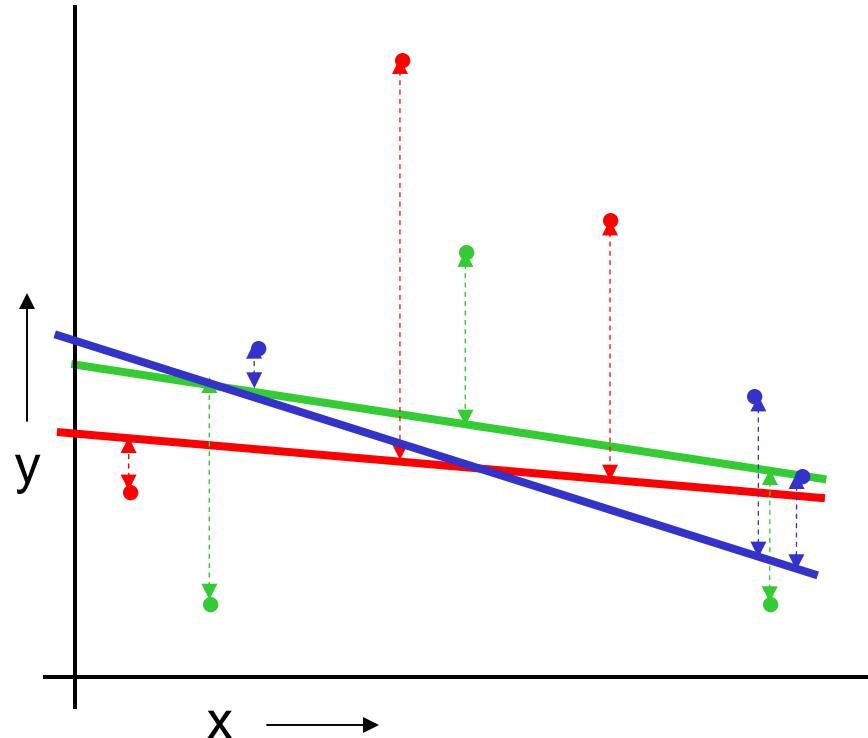
Randomly break the dataset into k partitions (in our example we'll have $k=3$ partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

k-fold Cross Validation



Linear Regression

$$MSE_{3FOLD}=2.05$$

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

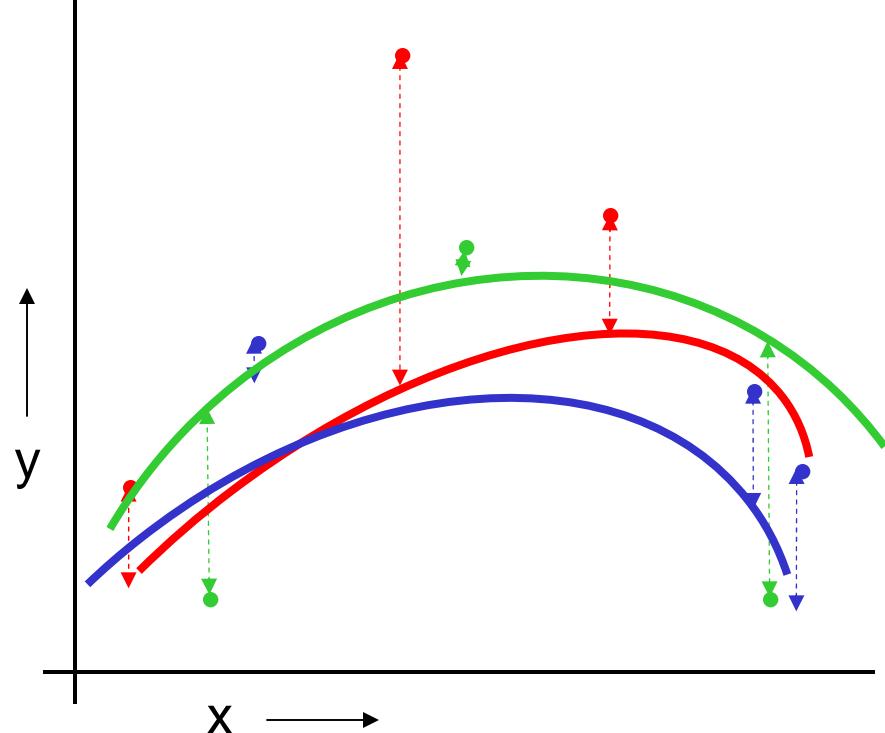
For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

k-fold Cross Validation



Quadratic Regression

$$MSE_{3FOLD}=1.11$$

Randomly break the dataset into k partitions (in our example we'll have $k=3$ partitions colored Red Green and Blue)

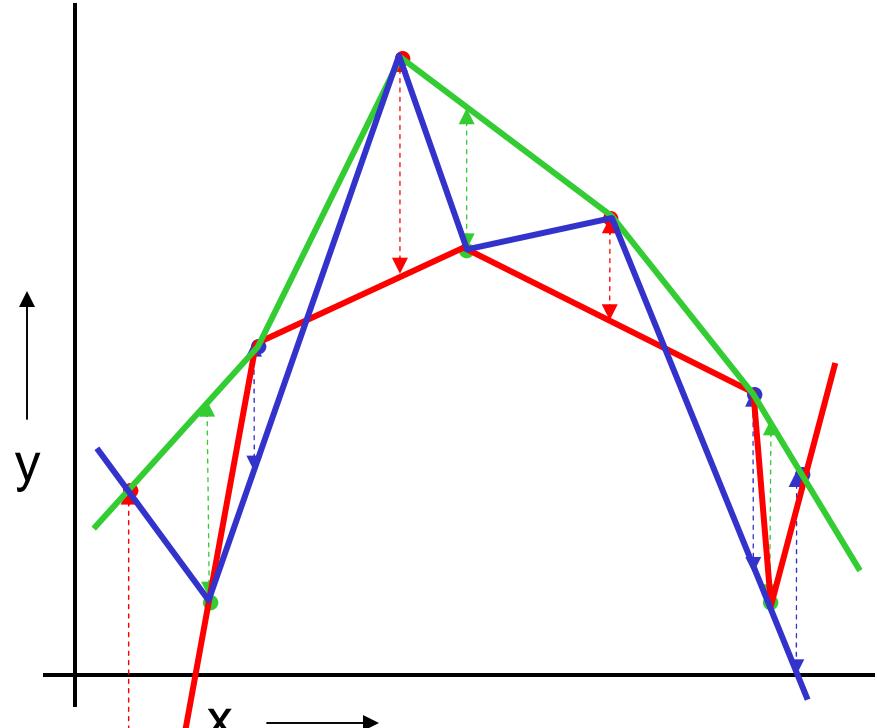
For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

k-fold Cross Validation



Joint-the-dots

$$MSE_{3FOLD}=2.93$$

Randomly break the dataset into k partitions (in our example we'll have $k=3$ partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

Which kind of Cross Validation?

	Downside	Upside
Test-set	Variance: unreliable estimate of future performance	Cheap
Leave-one-out	Expensive. Has some weird behavior	Doesn't waste data
10-fold	Wastes 10% of the data. 10 times more expensive than test set	Only wastes 10%. Only 10 times more expensive instead of R times.
3-fold	Wastier than 10-fold. Expensivier than test set	Slightly better than test-set
R-fold	Identical to Leave-one-out	

Backpropagation

Chain rule

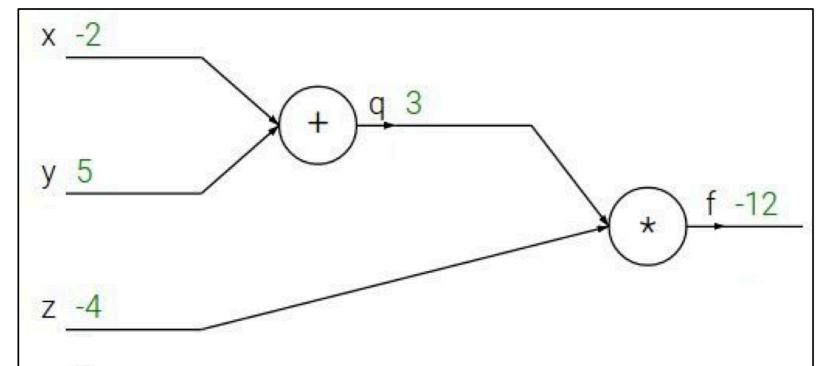
$$f(x, y, z) = (x + y)z$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

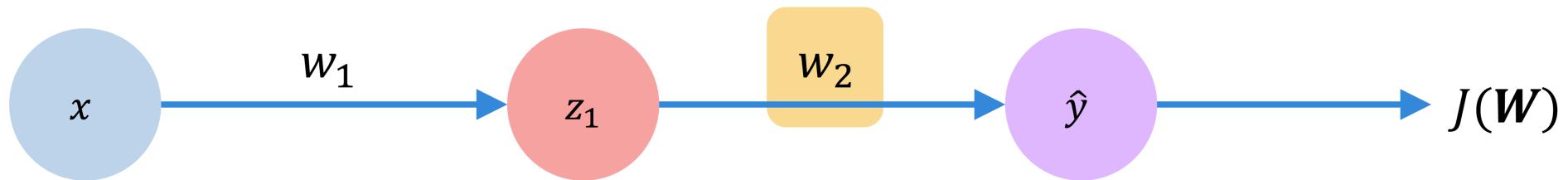


Computational graph

directed graph where the nodes correspond to **operations** or **variables**.
Variables can feed their value into operations, and operations can feed their output into other operations.

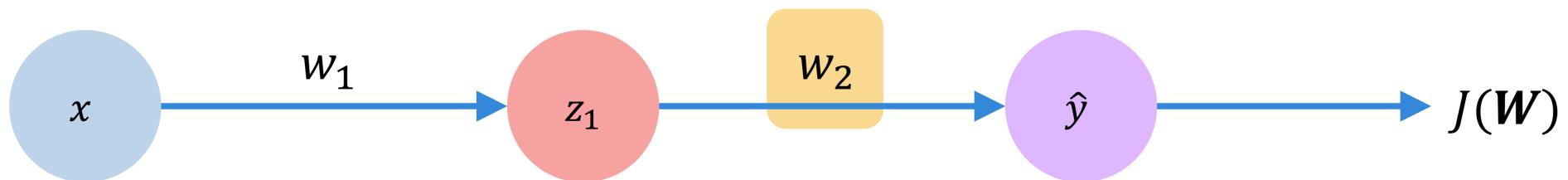
Backpropagation

How do we compute the gradient



How does a small change in one weight (ex. w_2) affect the final loss $J(\mathbf{W})$?

Backpropagation

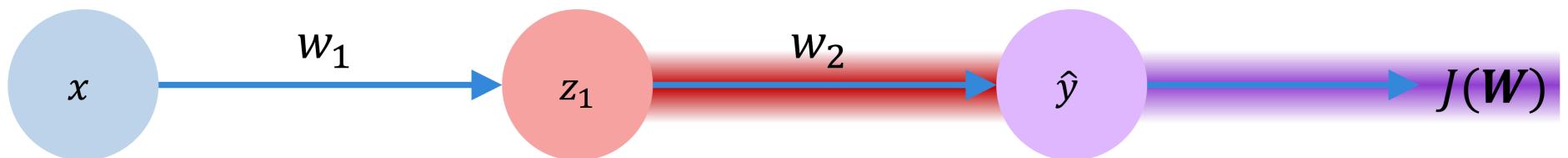


$$\frac{\partial J(\mathbf{W})}{\partial w_2} =$$

Let's use the chain rule!

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)} \quad \hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$$

Backpropagation

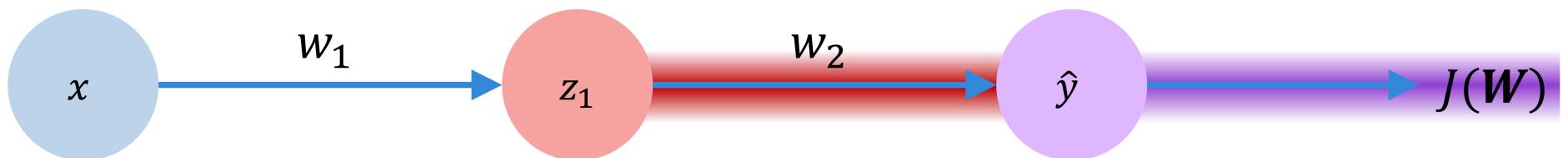


$$\frac{\partial J(\mathbf{W})}{\partial w_2} = \underbrace{\frac{\partial J(\mathbf{W})}{\partial \hat{y}}}_{\text{purple bar}} * \underbrace{\frac{\partial \hat{y}}{\partial w_2}}_{\text{red bar}}$$

Yellow horizontal bar at the bottom.

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)} \quad \hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$$

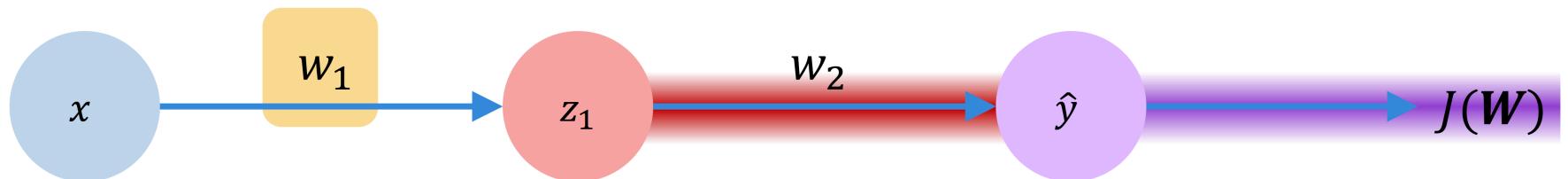
Backpropagation



$$\frac{\partial J(\mathbf{W})}{\partial w_2} = \underline{\frac{\partial J(\mathbf{W})}{\partial \hat{y}}} * \underline{\frac{\partial \hat{y}}{\partial w_2}}$$

$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)}$ $\hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$

Backpropagation



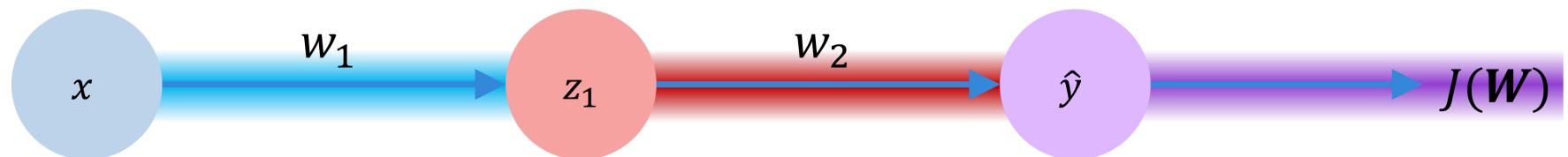
$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

Apply chain rule!

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)} \quad \hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$$

Backpropagation



$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

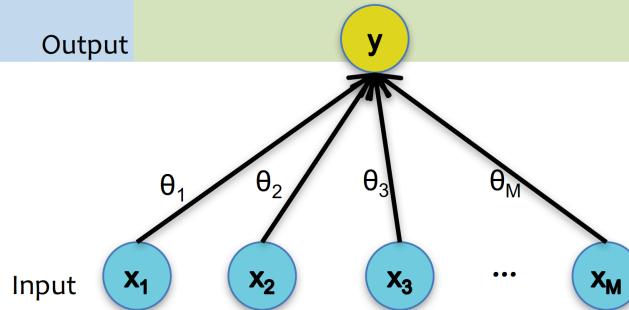
— — —

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)} \quad \hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$$

Training

Backpropagation

**Case 1:
Logistic
Regression**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

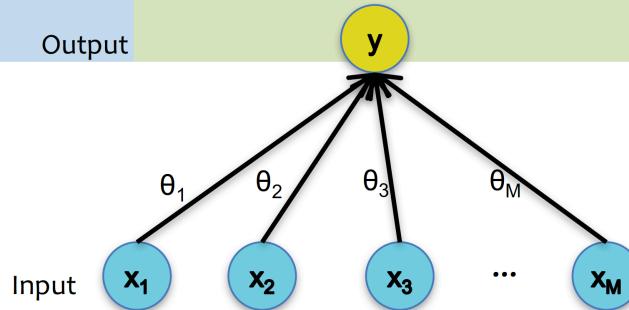
$$a = \sum_{j=0}^D \theta_j x_j$$

Backward

Training

Backpropagation

**Case 1:
Logistic
Regression**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^D \theta_j x_j$$

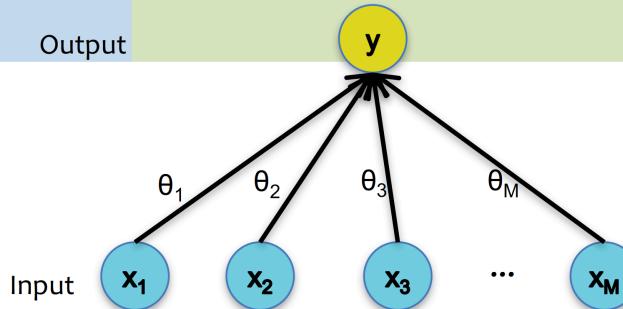
Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

Training

Backpropagation

**Case 1:
Logistic
Regression**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^D \theta_j x_j$$

Backward

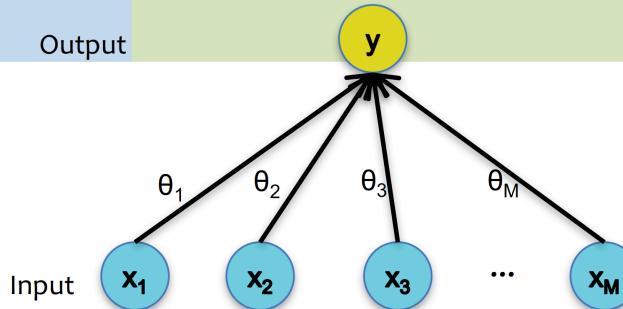
$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

Training

Backpropagation

**Case 1:
Logistic
Regression**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^D \theta_j x_j$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

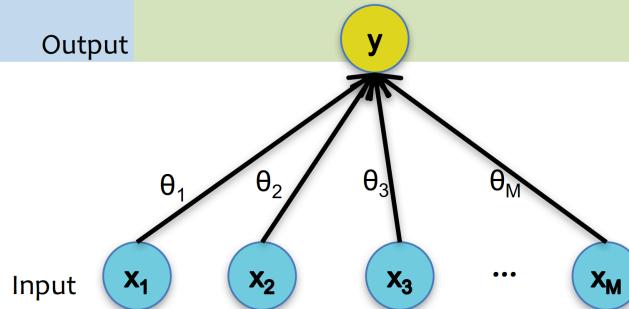
$$\frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

$$\frac{dJ}{d\theta_j} = \frac{dJ}{da} \frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j$$

Training

Backpropagation

**Case 1:
Logistic
Regression**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^D \theta_j x_j$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

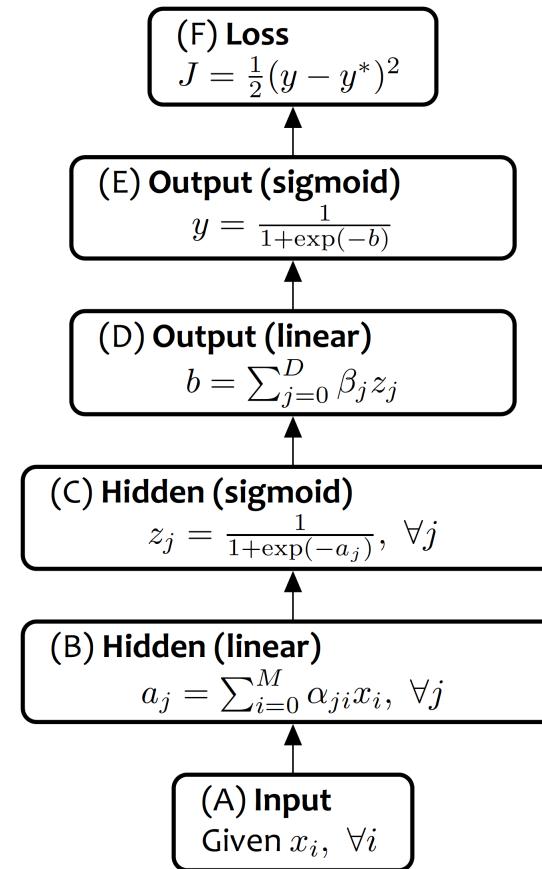
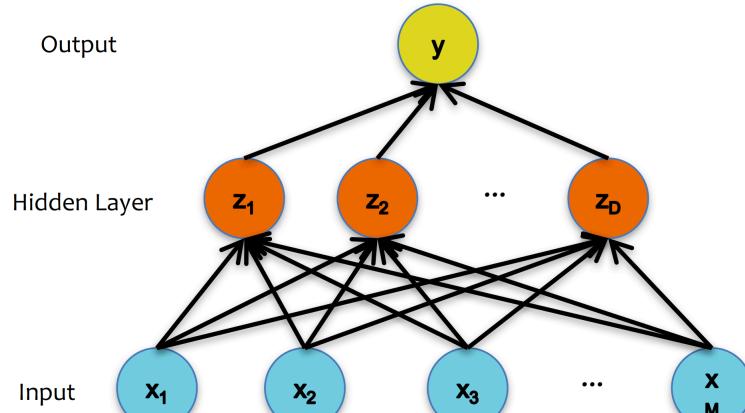
$$\frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

$$\frac{dJ}{d\theta_j} = \frac{dJ}{da} \frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j$$

$$\frac{dJ}{dx_j} = \frac{dJ}{da} \frac{da}{dx_j}, \quad \frac{da}{dx_j} = \theta_j$$

Training

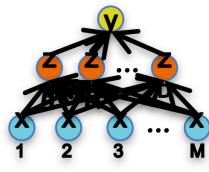
Backpropagation



Training

Backpropagation

Case 2:
**Neural
Network**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

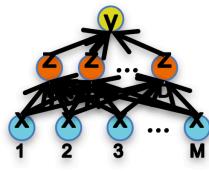
$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

Training

Backpropagation

**Case 2:
Neural
Network**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

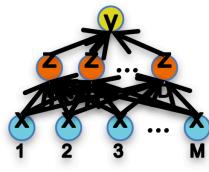
Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

Training

Backpropagation

**Case 2:
Neural
Network**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

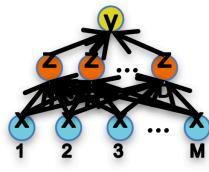
$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

Training

Backpropagation

**Case 2:
Neural
Network**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

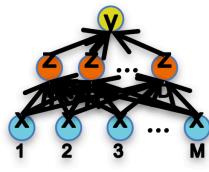
$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

Training

Backpropagation

Case 2: Neural Network



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

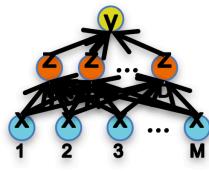
$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

Training

Backpropagation

**Case 2:
Neural
Network**



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j} \frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \sum_{j=0}^D \alpha_{ji}$$

Work out an example

<https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>

Thank you!