



Just a Bit More Elemental Learning Theory

Alexander G. Ororbia II
Introduction to Machine Learning
CSCI-335
9/28//2026

$$\operatorname{argmax}_r \sum \operatorname{onehot}(t^{(r)})$$

Sum up one-hot encodings and then find index of max count value

The K-NN Model

Algorithm:

$k = K = 1$

1. Find example $(\mathbf{x}^*, t^*)$ (from the stored training set) closest to the test instance $\mathbf{x}$. That is:

$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}^{(i)} \in \text{train. set}} \text{distance}(\mathbf{x}^{(i)}, \mathbf{x})$$

2. Output $y = t^*$

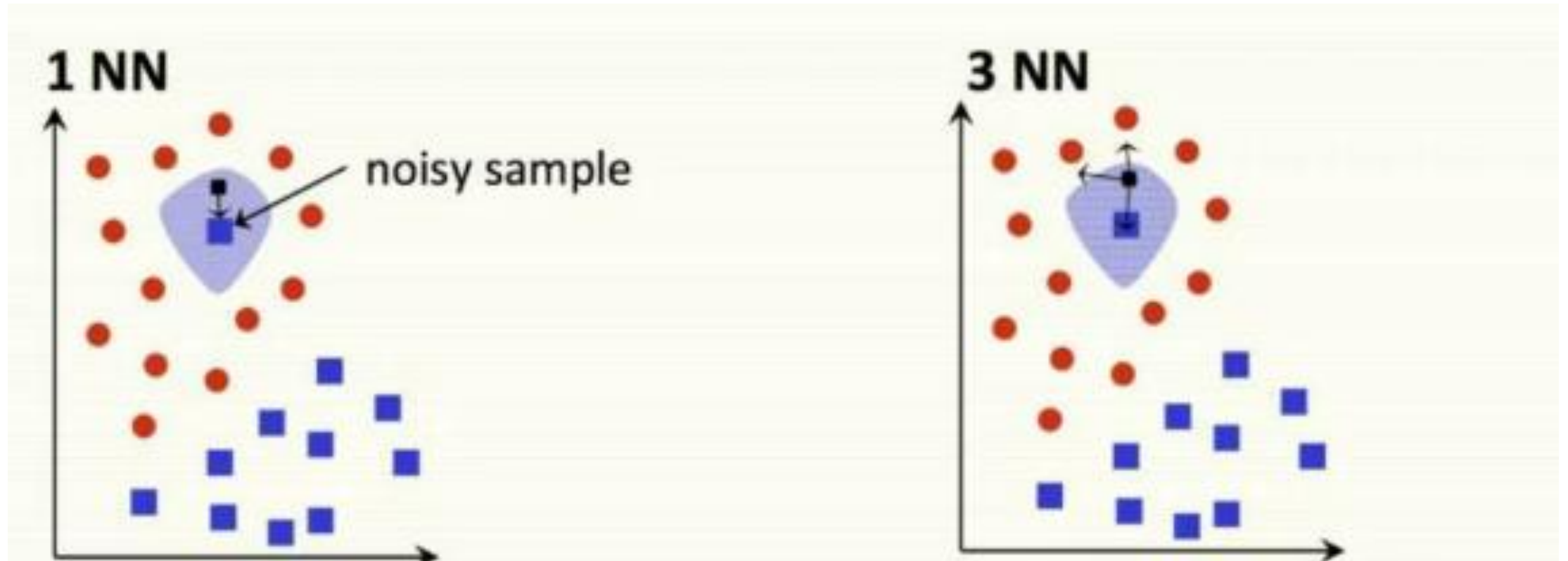
Algorithm (kNN):

1. Find k examples $\{\mathbf{x}^{(i)}, t^{(i)}\}$ closest to the test instance $\mathbf{x}$
2. Classification output is majority class

$$y = \operatorname{argmax}_{t^{(z)}} \sum_{r=1}^k \delta(t^{(z)}, t^{(r)})$$

Over $z \in Z$ classes

(Lazy) Instance-Based Learning with K-NN



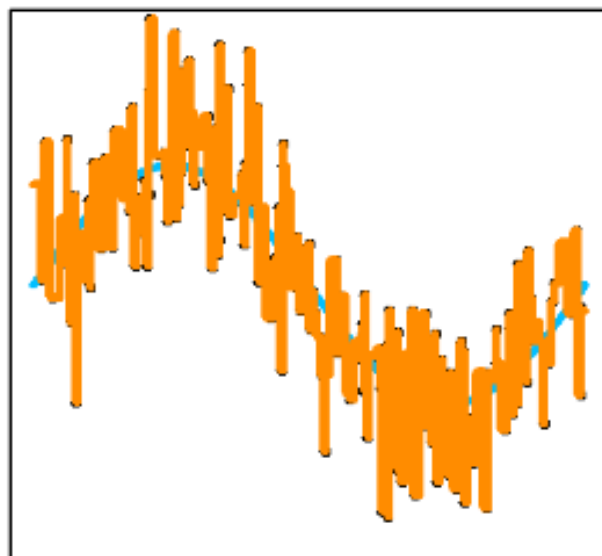
Every sample in blue shaded zone will be misclassified as **blue** class

Every sample in blue shaded zone will be misclassified as **red** class

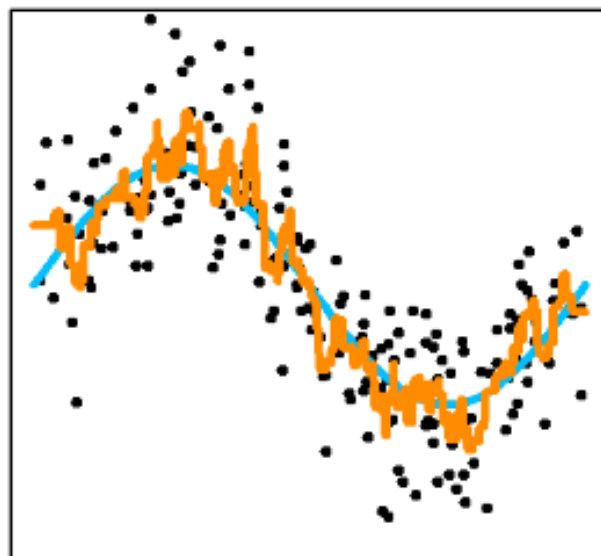
A Note on K -NN Complexity/Capacity

- For K -NN & non-parametrics, data samples are “parameters”
- For K -NN, K controls model “complexity”/capacity, under N data samples
 - K is inversely proportional to capacity
 - Higher K lowers complexity, lower K raises complexity
- $K = 1$ (most complex)
 - Every data sample has a “role” in model prediction (“learns” N different patterns)
 - Overfitting (zero training error, high test error; low bias, high variance)
- $K = N$ (least complex)
 - Model can only make one prediction (an average; line); “globs” data
 - Underfitting (high training error, high test error; high bias, low variance)

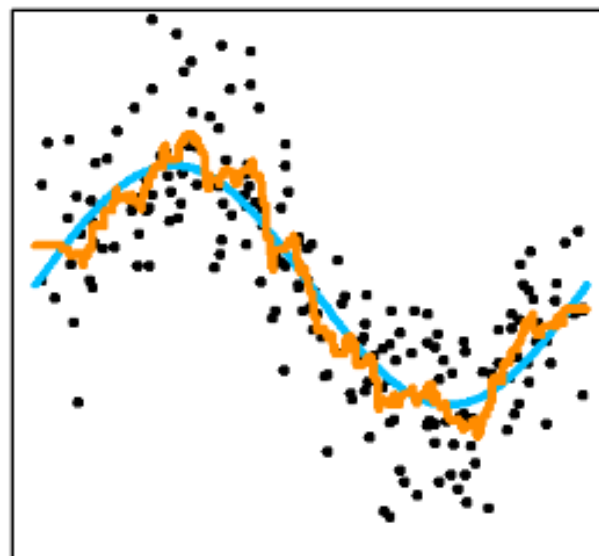
$K = 1$



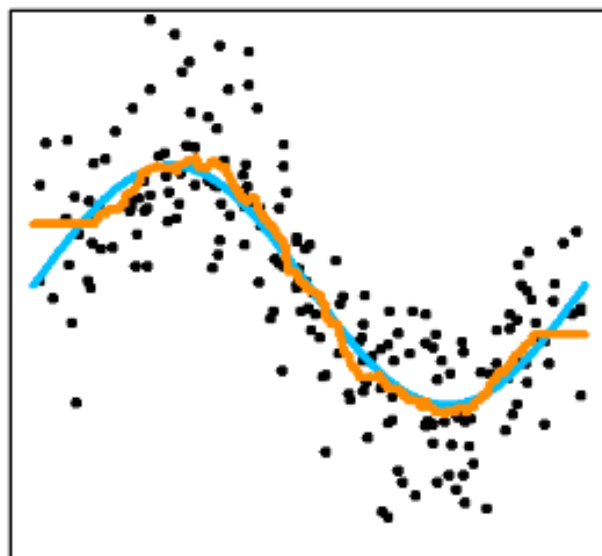
$K = 5$



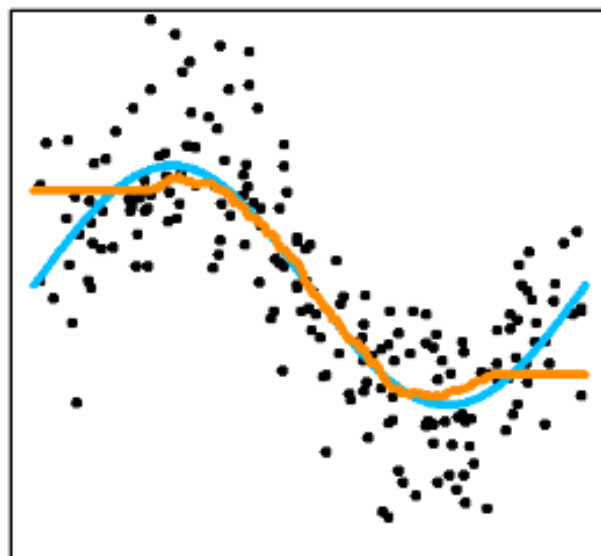
$K = 10$



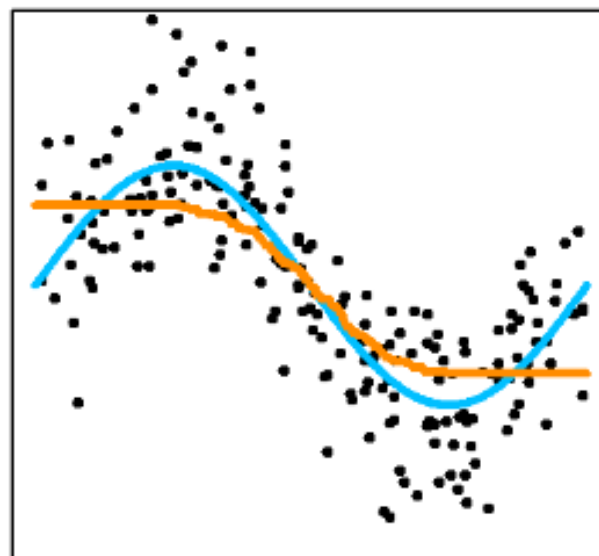
$K = 33$



$K = 66$



$K = 100$



Effect of Training Set Size

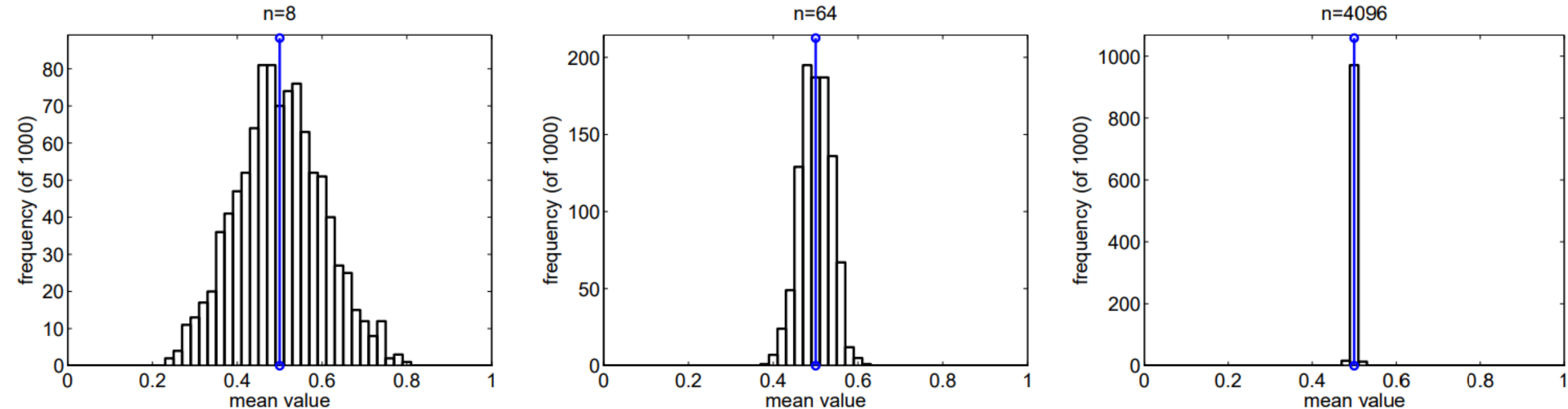
- Expected generalization error can never increase as the no of training examples increases
- For nonparametric models, more data yields better generalization until best possible error is achieved

How to Reduce Variance?

- Choose a simpler classifier
 - Occam's Razor: *Among competing hypotheses, the one with the fewest assumptions should be selected.*
- Regularize the parameters
 - Apply penalties / constraints
 - Go Bayesian – apply prior distributions over parameters
- Get more training data
 - **BIG** data (some theory justifies this!)

The Law of Large Numbers

- The (strong) law of large numbers
 - In limit of infinite data, or $N \rightarrow \infty$, sample mean converges to expected value $S_n \rightarrow \mu$ of data distribution
 - Intuitively, with more data, our estimate of a quantity/moment (like the mean), estimate becomes more *accurate*
- A partial justification for big data!

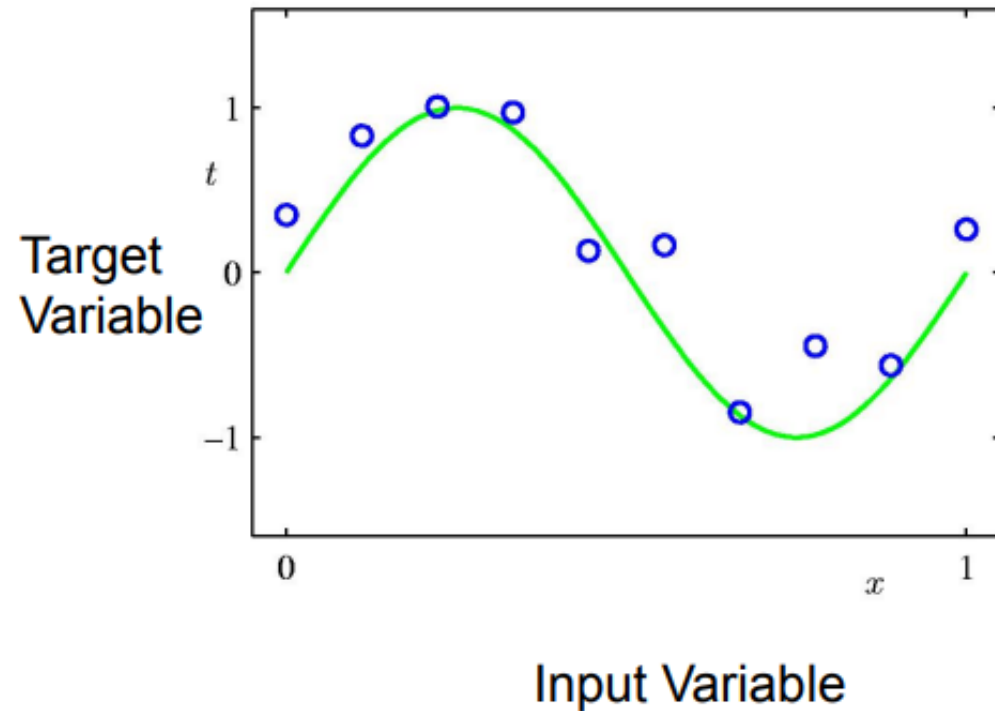


Simple Regression Problem

- Begin discussion on ML by introducing a simple regression problem
 - It motivates a no. of key concepts
- Problem:
 - Observe Real-valued input variable x
 - Use x to predict value of target variable t
- We consider an artificial example using synthetically generated data
 - Because we know the process that generated the data, it can be used for comparison against a learned model

Synthetic Data for Regression

- Data generated from the function $\sin(2\pi x)$
 - Where x is the input
- Random noise in target values



Input values $\{x_n\}$ generated uniformly in range $(0,1)$. Corresponding target values $\{t_n\}$ Obtained by first computing corresponding values of $\sin\{2\pi x\}$ then adding random noise with a Gaussian distribution with std dev 0.3

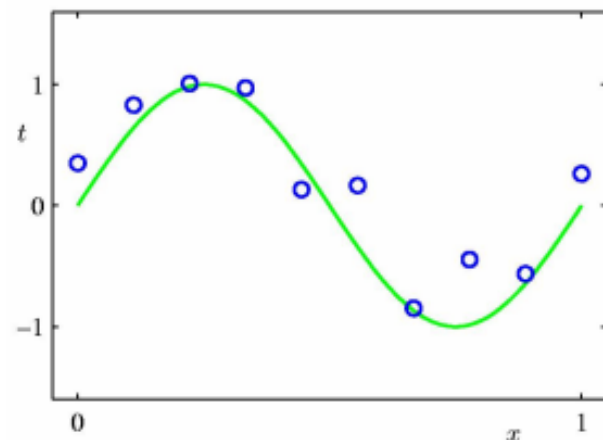
Training Set

- N observations of x

$$\mathbf{x} = (x_1, \dots, x_N)^T$$

$$\mathbf{t} = (t_1, \dots, t_N)^T$$

- Goal is to exploit training set to predict value $\hat{t}$ for some new value $\hat{x}$
- Inherently a difficult problem
- Probability theory provides framework for expressing uncertainty in a precise, quantitative manner
- Decision theory allows us to make a prediction that is optimal according to appropriate criteria



A Simple Approach to Curve Fitting

Representation

- Fit the data using a *polynomial function*

$$y(x, \mathbf{w}) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M = \sum_{j=0}^M w_jx^j$$

– where M is the order of the polynomial

- Is higher value of M better? We'll see shortly!
- Coefficients $w_0, \dots, w_M$ are collectively denoted by vector $\mathbf{w}$
- It is a nonlinear function of x , but a linear function of the unknown parameters

(Still a linear model; linear in parameter space)

Error Function

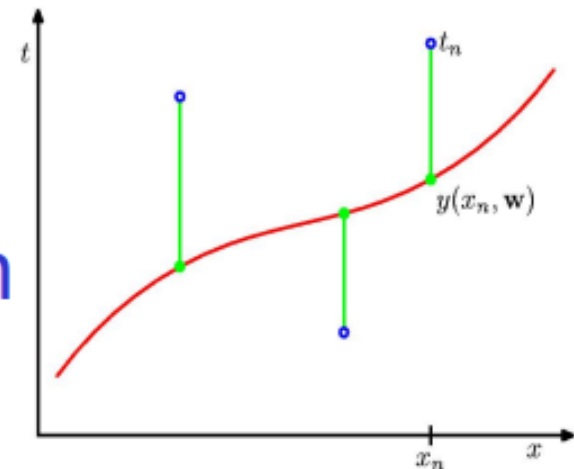
- We can obtain a fit by minimizing an error function
 - Sum of squares of the errors between the predictions $y(x_n, \mathbf{w})$ for each data point x_n and target value t_n

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2$$

- Factor $\frac{1}{2}$ included for later convenience

- Solve by choosing value of $\mathbf{w}$ for which small as possible

Red line is best polynomial fit



Polynomial Curve Fitting

– With a single input variable x

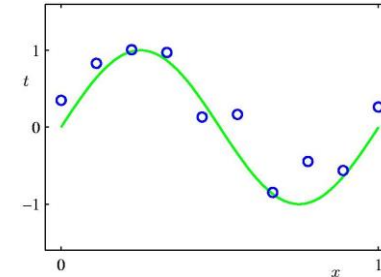
– $y(x, \mathbf{w}) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M$

M = order of polynomial,

x^j denotes x raised to power j ,

Coefficients $w_0, \dots, w_M$ are collectively denoted by vector $\mathbf{w}$

$$= \sum_{j=0}^M w_j x^j$$



Training data set
 $N=10$, Input x , target t

– **Task:** Learn $\mathbf{w}$ from training data $D = \{(x_i, t_i)\}, i = 1, \dots, N$

- Can be done by minimizing an error function that minimizes misfit between $y(x, \mathbf{w})$ for any given $\mathbf{w}$ and training data
- One simple choice of error function is sum of squares of error (SSE) between predictions $y(x_n, \mathbf{w})$ for each data point x_n and corresponding target values t_n so that we minimize:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2$$

- It is zero when function $y(x, \mathbf{w})$ passes exactly through each training data point

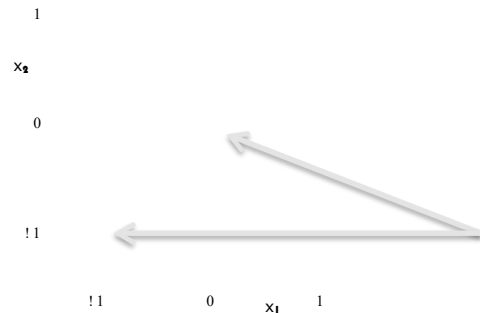
On Basis Functions

- In many applications, we apply some form of fixed-preprocessing, or feature extraction, to the original data variables
- If the original variables comprise the vector $\mathbf{x}$, then the features can be expressed in terms of basis functions $\{\phi_j(\mathbf{x})\}$
 - By using nonlinear basis functions we allow the function $y(\mathbf{x}, \mathbf{w})$ to be a nonlinear function of the input vector $\mathbf{x}$
 - They are linear functions of parameters (gives them simple analytical properties), yet are nonlinear wrt input variables

Fixed Basis Functions

Although we use linear (classification) models
Linear-separability in *feature* space *does not* imply
linear-separability in *input* space

Original Input Space (x_1, x_2)

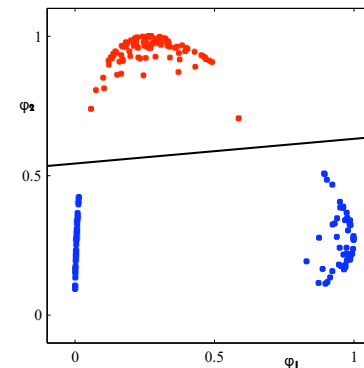


not linearly separable

Nonlinear transformation
of inputs using vector of
basis functions $\phi(\mathbf{x})$

Basis functions are
Gaussian with centers
shown by crosses and
green contours

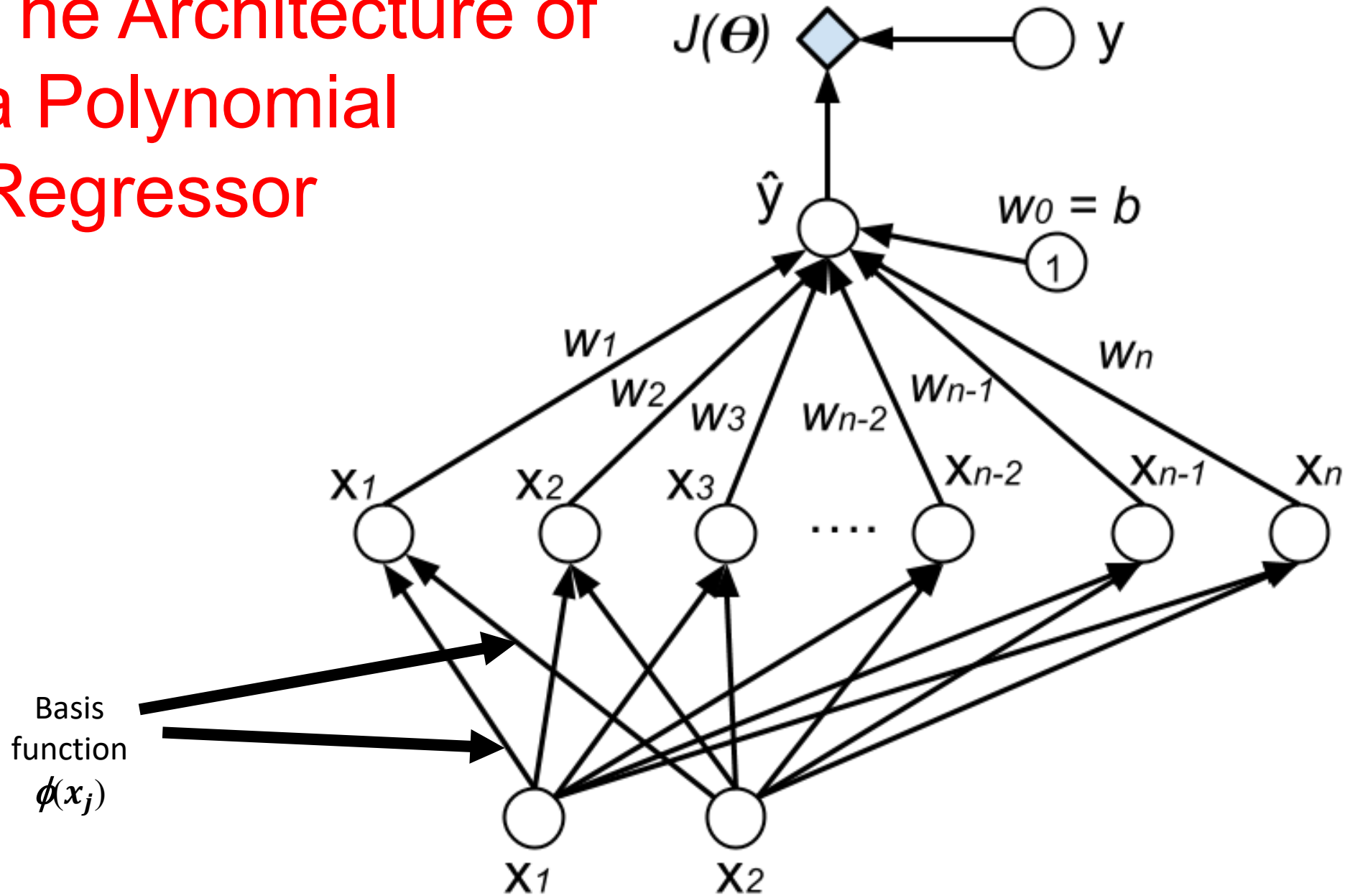
Feature Space (ϕ_1, ϕ_2)



linearly separable

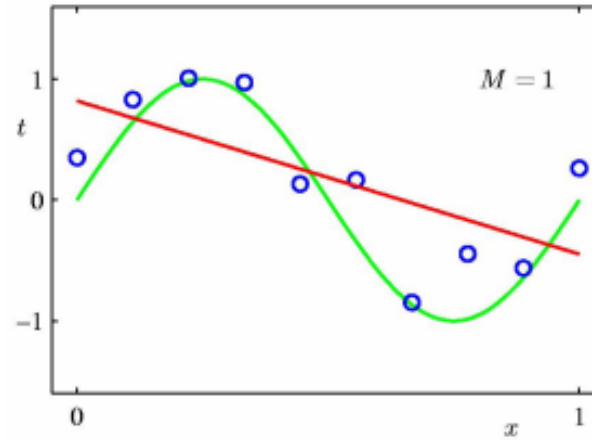
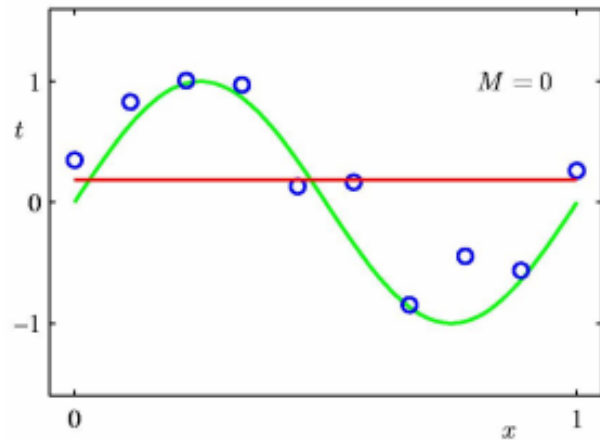
Basis functions with increased dimensionality often used

The Architecture of a Polynomial Regressor

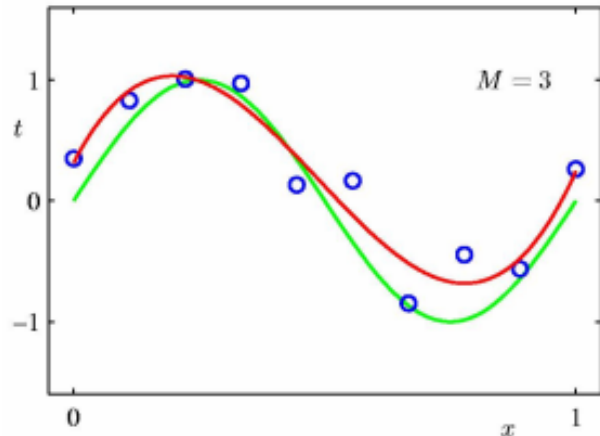


Choosing the Order of M

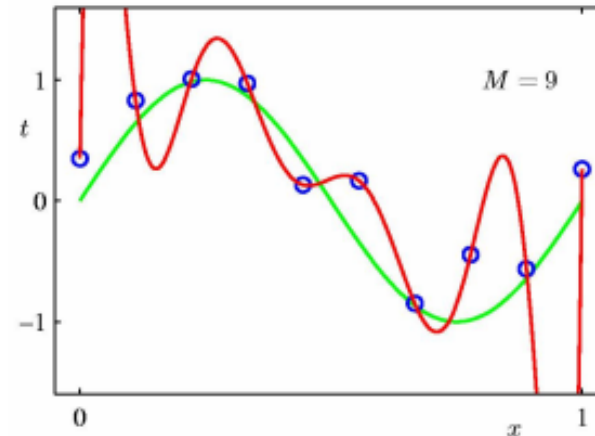
- Model Comparison or Model Selection
- Red lines are best fits with
 - $M = 0, 1, 3, 9$ and $N=10$



← Poor representations of $\sin(2\pi x)$



← Best Fit to $\sin(2\pi x)$

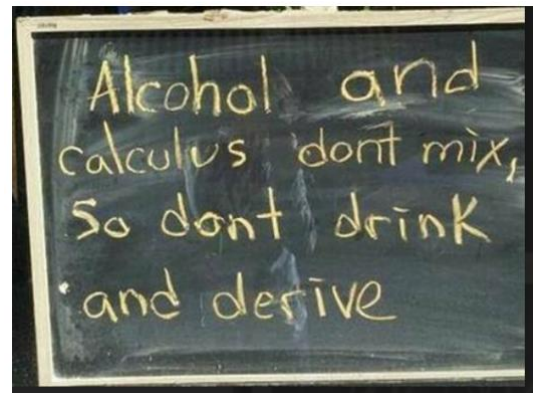


Over Fit
Poor representation of $\sin(2\pi x)$

Computational Bottleneck

- A recurring problem in machine learning:
 - Large training sets are necessary for good generalization
 - *BUT* large training sets are also computationally expensive to use
- Stochastic gradient descent (SGD) is an extension of gradient descent (GD) that offers a solution
 - Moreover, it is a vehicle for generalization beyond training set
 - Expectation may be approximated using small set of samples (we will also later refer to these sets as “mini-batches” → mini-batch GD)

Optimizing Linear Least Squares



Suppose we want to find the value of $\mathbf{x}$ that minimizes

$$f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{b}\|_2^2.$$

There are specialized linear algebra algorithms that can solve this problem efficiently. However, we can also explore how to solve it using gradient-based optimization as a simple example of how these techniques work.

First, we need to obtain the gradient:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \mathbf{A}^\top (\mathbf{Ax} - \mathbf{b}) = \mathbf{A}^\top \mathbf{Ax} - \mathbf{A}^\top \mathbf{b}.$$

Algorithm 4.1 An algorithm to minimize $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{b}\|_2^2$ with respect to $\mathbf{x}$ using gradient descent, starting from an arbitrary value of $\mathbf{x}$.

Set the step size (ϵ) and tolerance (δ) to small, positive numbers.

while $\|\mathbf{A}^\top \mathbf{Ax} - \mathbf{A}^\top \mathbf{b}\|_2 > \delta$ **do**

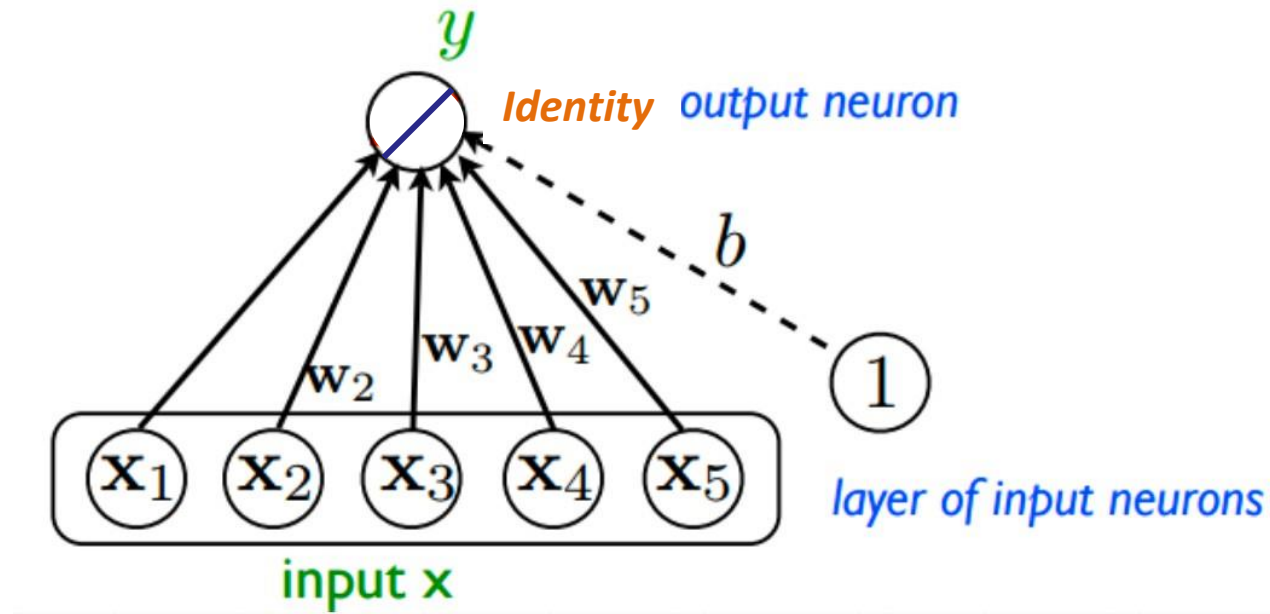
$\mathbf{x} \leftarrow \mathbf{x} - \epsilon (\mathbf{A}^\top \mathbf{Ax} - \mathbf{A}^\top \mathbf{b})$

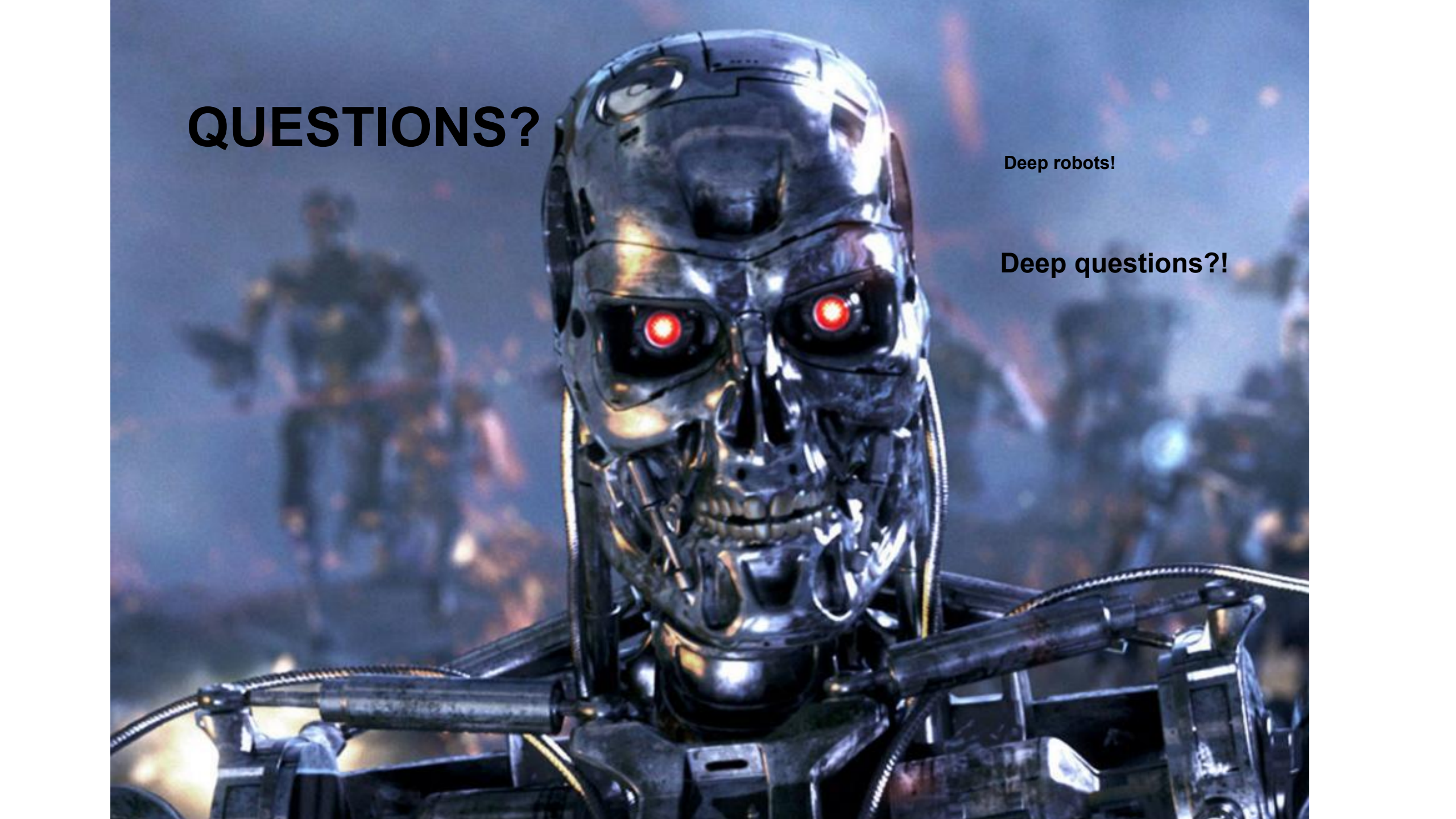
end while

Insight of SGD

- Gradient is an expectation $\nabla \ln p(y | X, \vartheta, \theta) = \theta \sum_{i=1}^m \{y^{(i)} - \vartheta^T \mathbf{x}^{(i)}\} \mathbf{x}^{(i)T}$
 - Expectation may be approximated using small set of samples
- In each step of SGD we can sample a *minibatch* of examples $B = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m')}\}$
 - drawn uniformly from the training set
 - Minibatch size m' is typically chosen to be small: 1 to a hundred
 - Crucially m' is held fixed even if sample set is in billions
 - We may fit a training set with billions of examples using updates computed on only a hundred examples

Next Time: A Treatment of Linear Regression





QUESTIONS?

Deep robots!

Deep questions?!