# Linear Regression and Classification

Tripp Deep Learning F23

Sources include Bishop, *Pattern Recognition & Machine Learning*

# TODAY'S GOAL

By the end of the class, you should know the difference between regression and classification, understand how some of the key linear methods work, and understand some of their limitations.

# Summary

1. Regression and classification are prediction of continuous and categorical values, respectively
2. Linear regression and classification use linear functions of the inputs
3. Regression and classification are supervised learning problems
4. Linear regression typically minimizes squared-error loss
5. One can have too many or too few basis functions
6. One can have too many or too few inputs

# Summary

7. Ridge regression reduces weights to reduce overfitting
8. A linear discriminant uses a linear boundary to separate examples into different categories
9. Perceptrons adjust the boundary to correct classification errors
10. Logistic regression categorizes inputs probabilistically
11. Linear classifiers can implement Boolean logic, except for XOR
12. If there are multiple categories, it's best to make a predictor for each and choose the one with the highest pre-threshold output
13. The number of basis functions needed to tile a space depends exponentially on the dimension of the space

# REGRESSION AND CLASSIFICATION ARE PREDICTION OF CONTINUOUS AND CATEGORICAL VALUES, RESPECTIVELY

# Regression

- Given a vector $\boldsymbol{x}$, what is the associated scalar $y$?
- Examples:
  - Given scores for app clarity, familiarity, responsiveness, consistency, and aesthetics, how will users rate a phone app overall?
  - Given video from a moving camera, what is the camera's speed of motion?

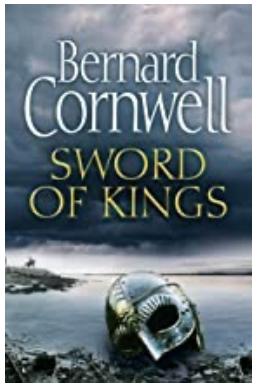
# Classification

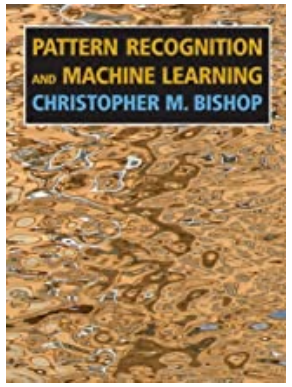
- Given a vector **x**, to which group does it belong?
- Examples:
  - Given an ECG signal, has the person had a heart attack?
  - Given an image, what kind of thing is the image of?

# Regression vs. Classification

- Both may use the same inputs, e.g., suppose you want to judge a book by its cover

**Examples of Inputs**






Machine Learning

**Potential Inferences**

Fiction or non-fiction?

How many hours would it take to read?

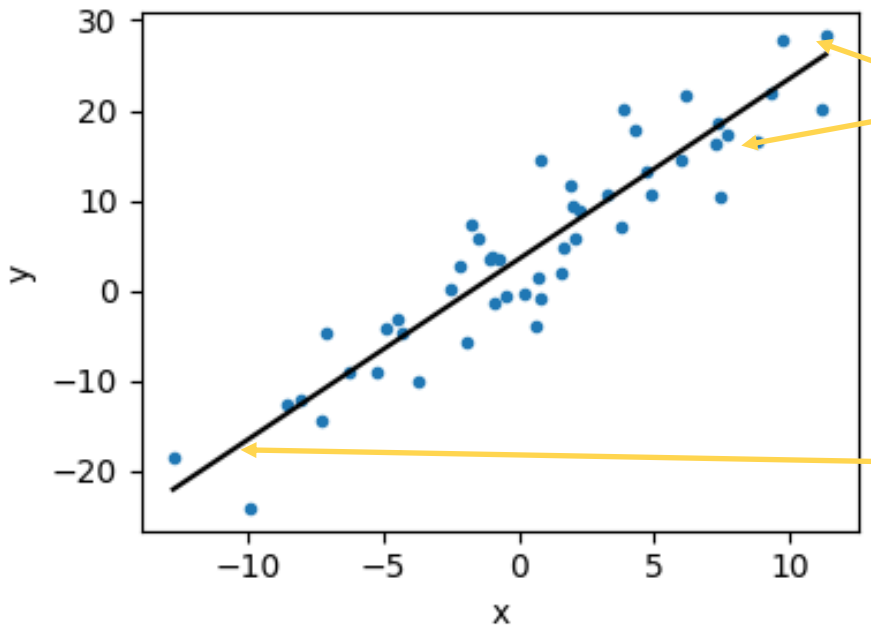
# LINEAR REGRESSION AND CLASSIFICATION USE LINEAR FUNCTIONS OF THE INPUTS

# Linear regression

- In **linear** regression, the prediction $\hat{y}$ is a linear (or affine) function of the inputs $\boldsymbol{x}$:

$$\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + b$$

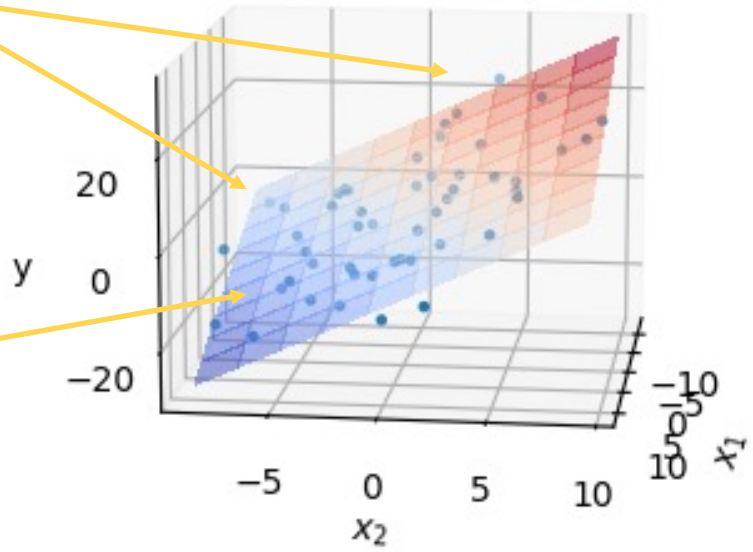
One-dimensional $\boldsymbol{x}$




Data points

Linear fit

Two-dimensional $\boldsymbol{x}$






UNIVERSITY OF WATERLOO

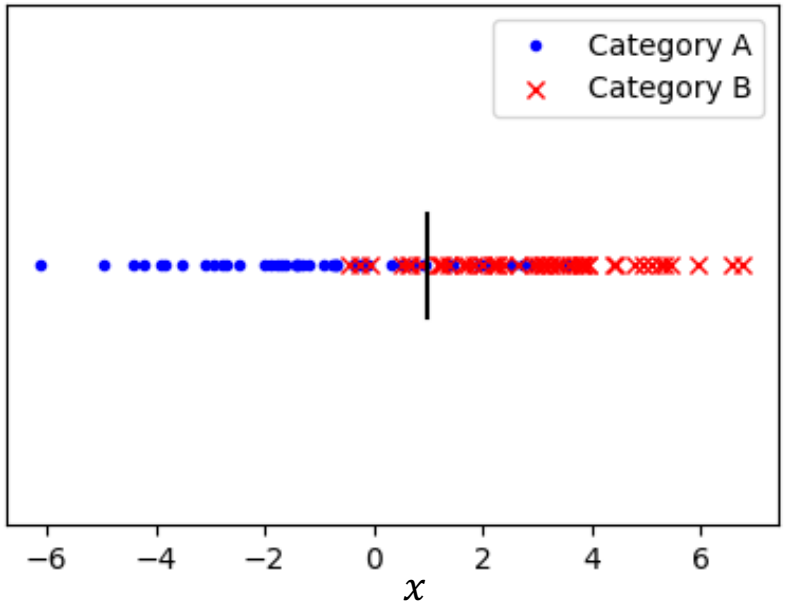
# Linear classification

- **Linear** classification uses a linear decision boundary:

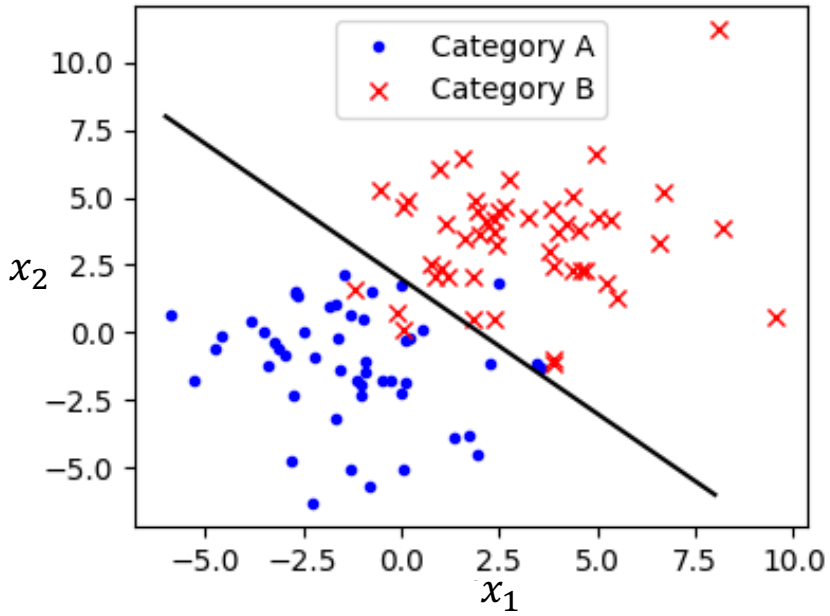
$$\hat{y} = \begin{cases} 0, & \boldsymbol{w}^T\boldsymbol{x} + b < 0 \\ 1, & \boldsymbol{w}^T\boldsymbol{x} + b \geq 0 \end{cases}$$

- If **x** is D-dimensional, then the decision boundary is a (D-1)-dimensional hyperplane

One-dimensional $x$




Two-dimensional $\boldsymbol{x}$

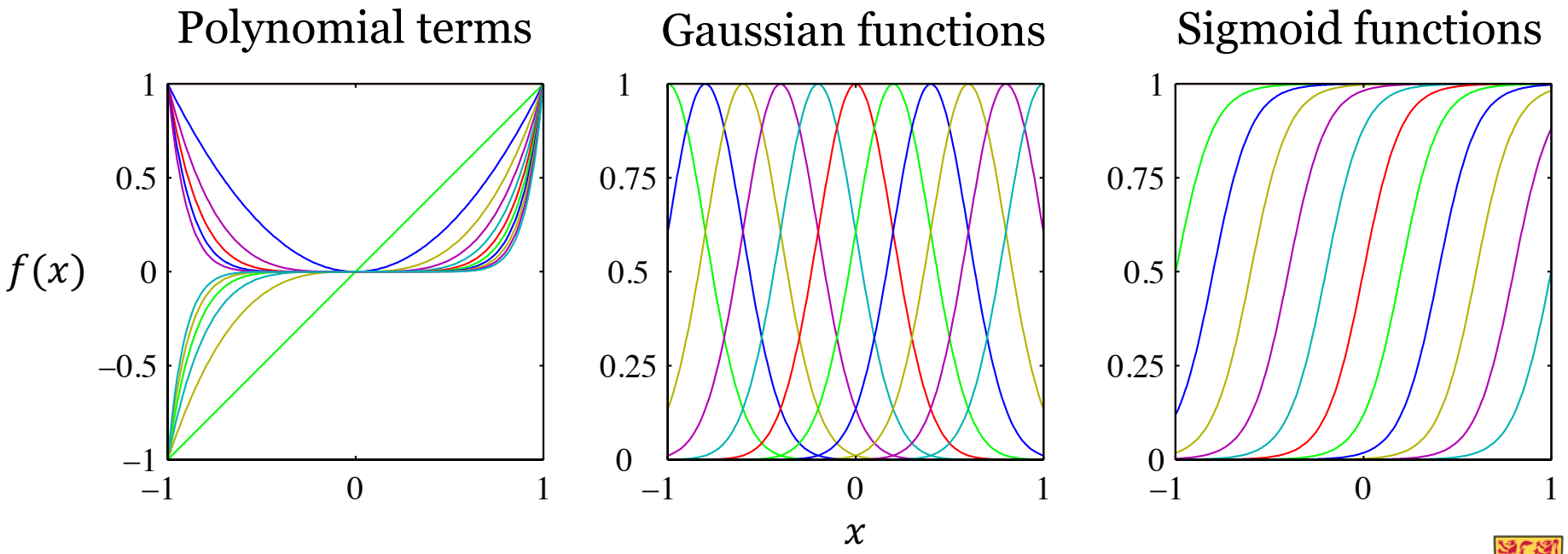


# Nonlinear predictions via nonlinear basis functions

- The independent variables $\boldsymbol{x}$ may consist of:
  - Data (e.g., measurements, observations, etc.)
  - "Features" of data, things that we can calculate (e.g., standard deviation or peak-to-peak range or mean slope of a signal) that may facilitate prediction
  - Nonlinear functions of either of those
- Nonlinear basis functions:
  - Allow approximations and decision boundaries that are not linear
  - Do not change the methods aside from the additional step of calculating the functions; the methods are still linear, but with respect to $\boldsymbol{f}(\boldsymbol{x})$
  - Regression becomes $\hat{y} = \boldsymbol{w}^T \boldsymbol{f}(\boldsymbol{x}) + b$
  - Classification becomes $\hat{y} = 1$ if $\boldsymbol{w}^T \boldsymbol{f}(\boldsymbol{x}) + b > 0$

# Nonlinear predictions via nonlinear basis functions

- This works best when $\boldsymbol{x}$ has few dimensions
- Example: If polynomial terms (i.e., $1, x, x^2, x^3, \ldots$) are used as basis functions, linear regression finds a polynomial that approximates the training data






UNIVERSITY OF WATERLOO

# Problem Scale

- Various linear methods work well with input dimensions up to thousands, tens of thousands, or millions depending on the solver

# REGRESSION AND CLASSIFICATION ARE SUPERVISED LEARNING PROBLEMS

# Supervised Learning

- Definition: Learning a function that approximates examples of input-output pairs
- Normally the function has a generic form with parameters $\theta$
- E.g., $y = \boldsymbol{w}^T \boldsymbol{x} + b$, where $\boldsymbol{w}$ and $b$ are the parameters; $\theta = (\boldsymbol{w}, b)$
- The differences between predictions (outputs of the function) $\hat{y}$ and actual/target values $y$ are summarized with a loss function
- The parameters are set to minimize the loss

Training Data

| Inputs | Targets |
|---|---|
| $\boldsymbol{x}_1$ | $y_1$ |
| $\boldsymbol{x}_2$ | $y_2$ |
| $\boldsymbol{x}_3$ | $y_3$ |
| $\boldsymbol{x}_4$ | $y_4$ |
| $\vdots$ | $\vdots$ |
| $\boldsymbol{x}_n$ | $y_n$ |

# LINEAR REGRESSION TYPICALLY MINIMIZES SQUARED-ERROR LOSS

# Least squares estimation

- The sum-squared error is,

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

- Minimizing this is the same as minimizing the mean-squared error (either can be used)
- This loss has a single minimum that can be found analytically

# Analytical solution

- Suppose we have a dataset of $(\boldsymbol{x}_i, y_i)$ pairs, and we want to fit a model $\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + b$
- To simplify, we will absorb $b$ into $\boldsymbol{w}$
  - $\boldsymbol{w} \leftarrow [\boldsymbol{w}\ b]$ and $\boldsymbol{x} \leftarrow [\boldsymbol{x}\ 1]$
  - Then $\hat{y} = \boldsymbol{w}^T \boldsymbol{x}$
- In matrix-vector form the loss is,

$$L = \|X\boldsymbol{w} - \boldsymbol{y}\|^2$$

where the $i^{th}$ row of $X$ is $\boldsymbol{x_i^T}$ and the $i^{th}$ entry of $\boldsymbol{y}$ is $y_i$

# Analytical solution

The loss can be written

$$\begin{aligned} L &= \|X\boldsymbol{w} - \boldsymbol{y}\|^2 \\ &= (X\boldsymbol{w} - \boldsymbol{y})^T (X\boldsymbol{w} - \boldsymbol{y}) \\ &= \boldsymbol{w}^T X^T X\boldsymbol{w} - \boldsymbol{w}^T X^T \boldsymbol{y} - \boldsymbol{y}^T X\boldsymbol{w} + \boldsymbol{y}^T \boldsymbol{y} \end{aligned}$$

The gradient of the loss with respect to the parameters is,

$$\frac{\partial L}{\partial \boldsymbol{w}} = 2X^T X\boldsymbol{w} - 2X^T \boldsymbol{y}$$

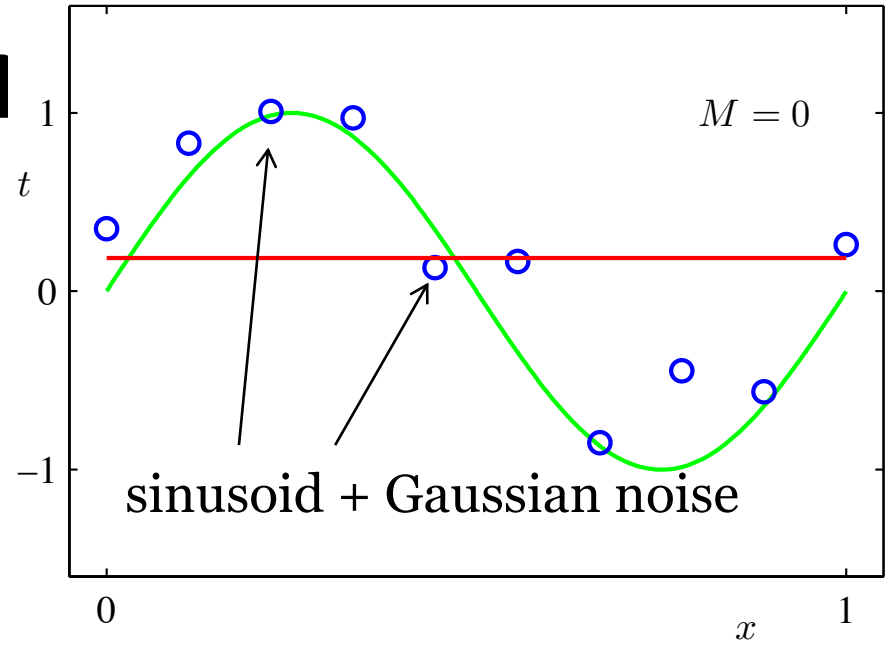
Setting this to zero,

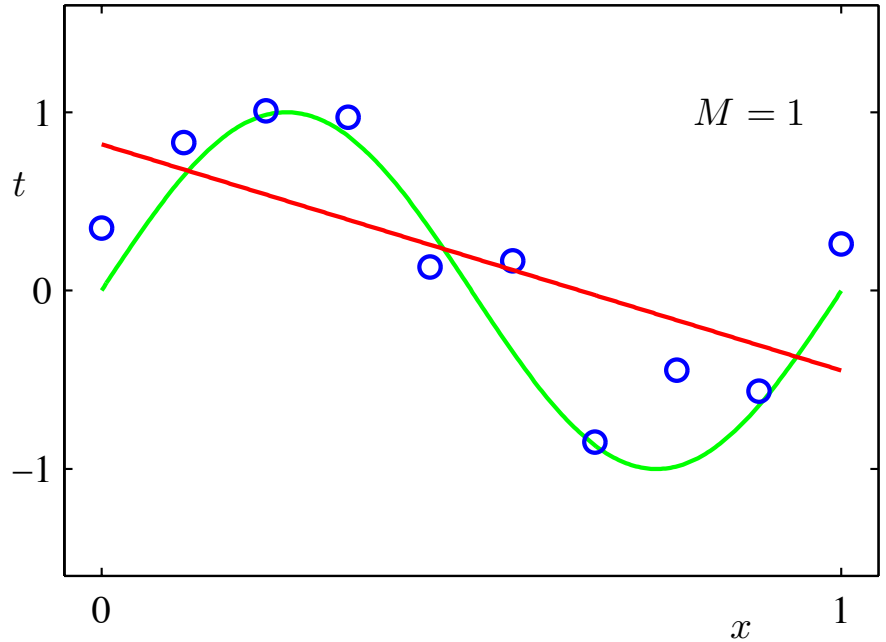
$$\boldsymbol{w}^* = (X^T X)^{-1} X^T \boldsymbol{y}$$

# ONE CAN HAVE TOO MANY OR TOO FEW BASIS FUNCTIONS

# Underfitting in Polynomial Regression

- The plots show
  - An unknown true function (green)
  - Noisy samples of this function (blue)
  - Optimal polynomial approximations of orders 0 and 1
- These low-order polynomials are not flexible enough to approximate the function (green)
- This is an example of underfitting, which can happen if a model is not complex enough

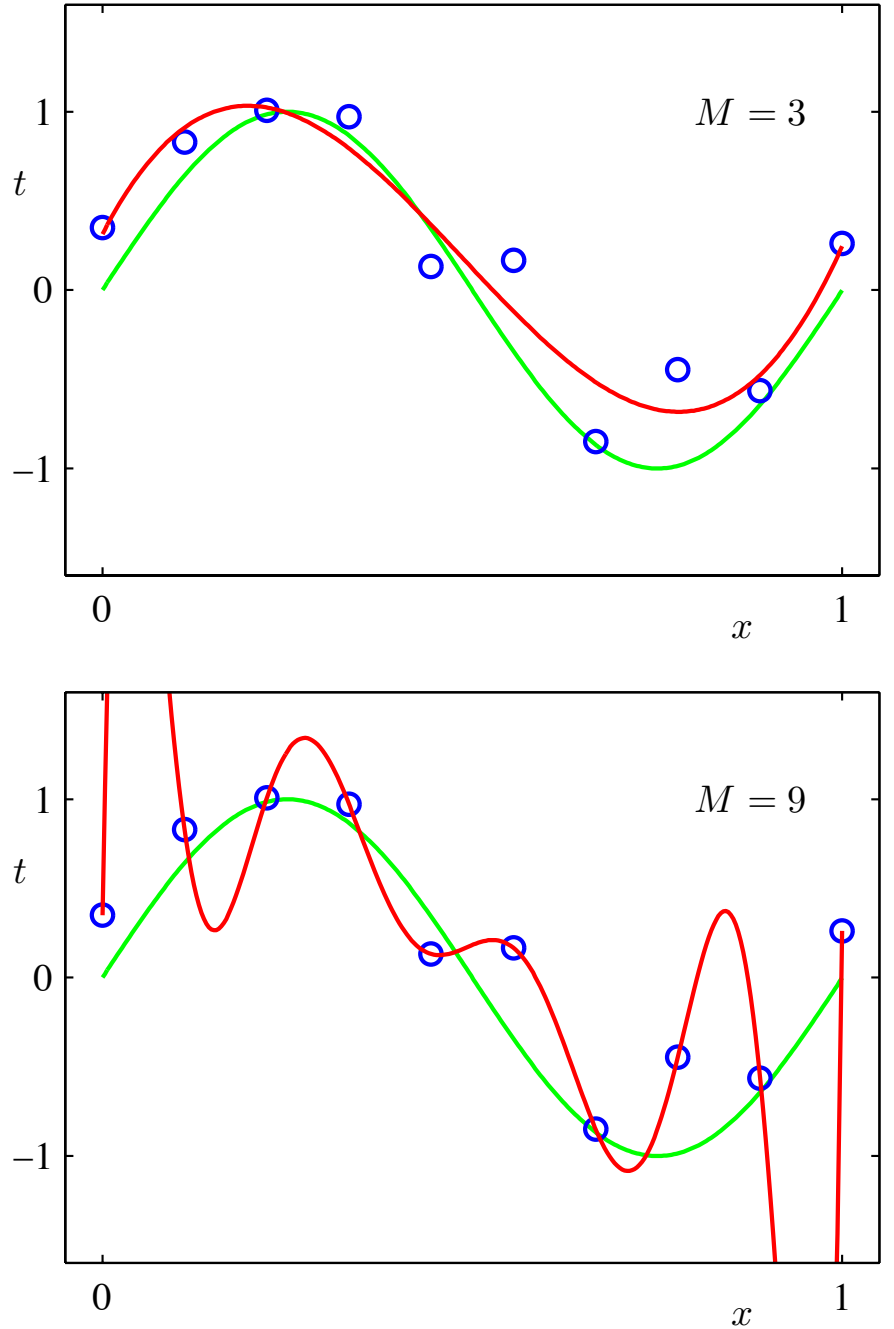



Bishop, Pattern Recognition & Machine Learning

# Overfitting in Polynomial Regression

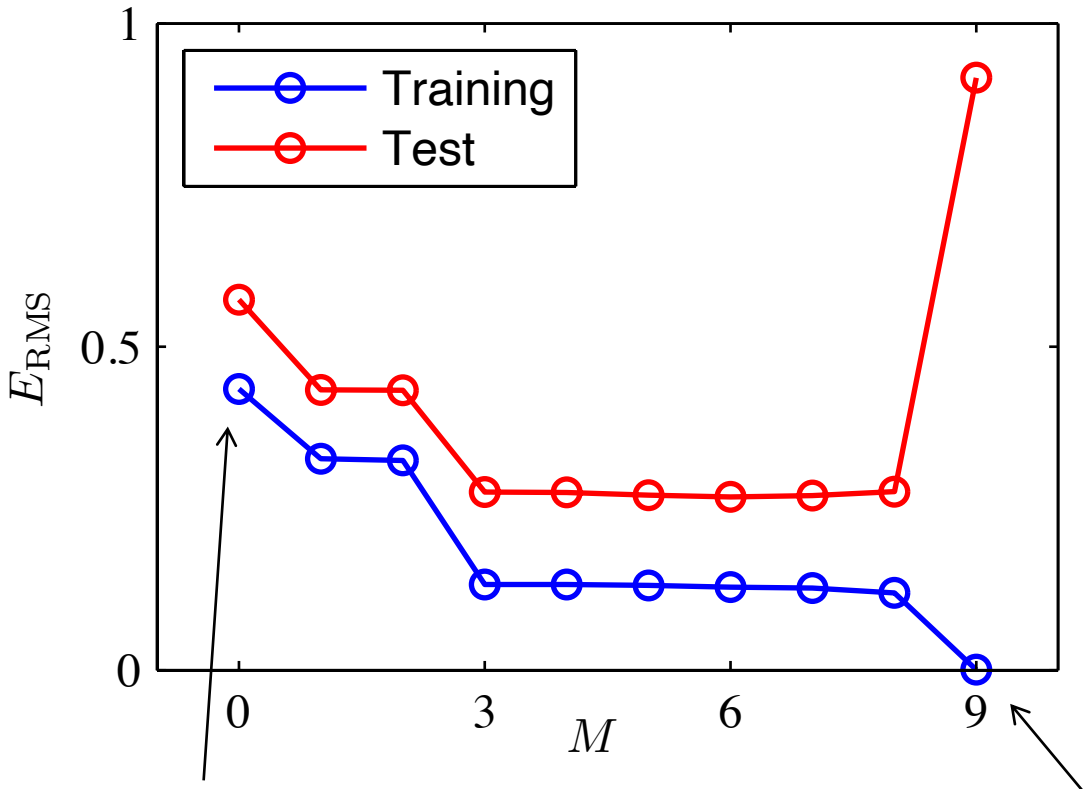
- A third-order polynomial produces a reasonable approximation of the sinusoid despite the noise
- A ninth-order polynomial is too sophisticated for the task
- It fits the examples very well but wouldn't fit new random examples
- This is an example of over-fitting, which can happen when the model is too flexible




Bishop, Pattern Recognition & Machine Learning

# Model Selection

- Choosing an effective polynomial order is an example of model selection
  - Less complex models risk underfitting
  - More complex models risk overfitting
- Recall we are only interested in performance on examples that are not used in training




New random samples

Original random samples

Not flexible enough to approximate underlying sine function (underfitting)

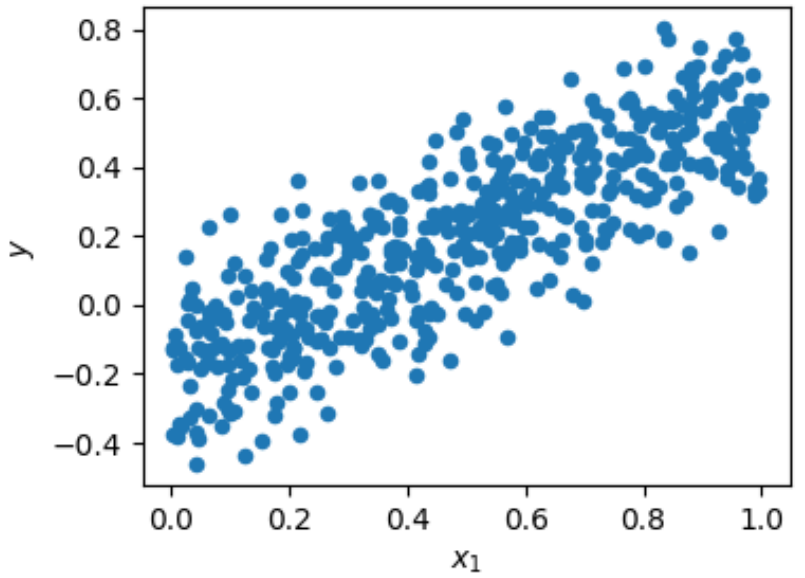
So flexible it approximates the random noise as well as the underlying sine function (overfitting)

# ONE CAN HAVE TOO MANY OR TOO FEW INPUTS

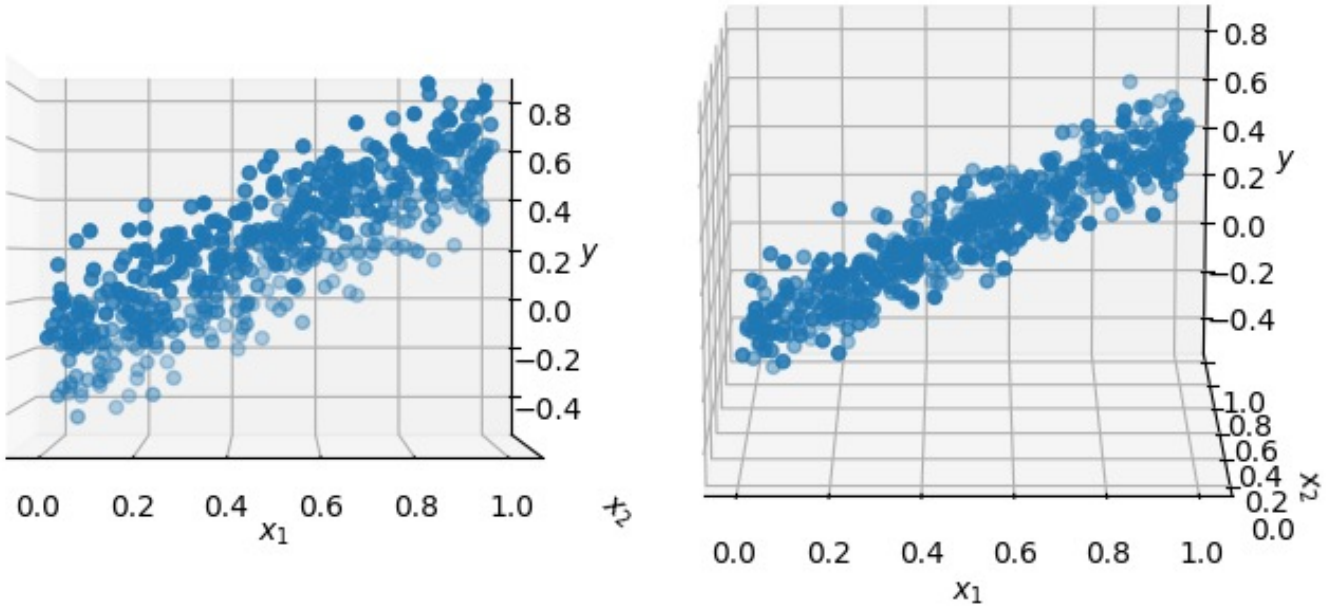
# Feature Selection

- The model will not perform well if the inputs don't contain sufficient information to predict the output
- Example: House price can be predicted better from year, city, and # bedrooms than from just year and # bedrooms
- Example: In these plots, $y$ can be better predicted from $x_1$ and $x_2$ than from $x_1$ alone

$y$ vs. $x_1$ (not enough information)



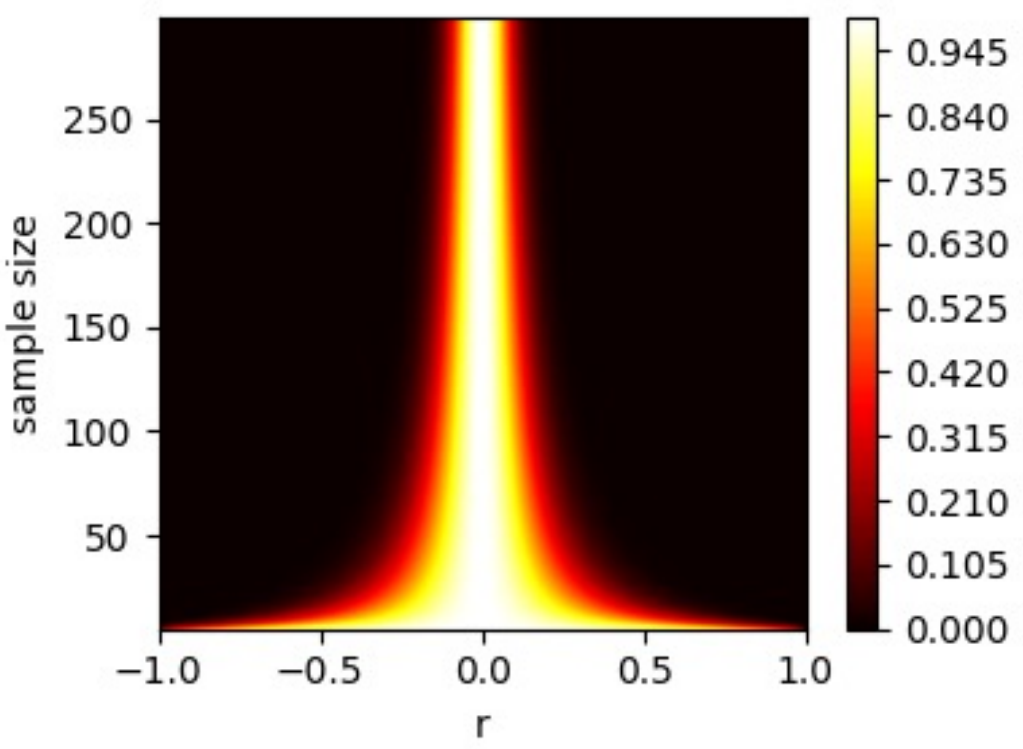

$y$ vs. $x_1$ and $x_2$ (more information)



# Feature Selection

- The model may not generalize well if some inputs have a weak relationship with the output
- Example: Predicting house price from year, city, # bedrooms, 3rd letter of street name, and temperature deviation vs. seasonal average one year prior to sale may involve spurious correlations
- This is because the *sample* correlation of two uncorrelated random variables (e.g., target and irrelevant input) is not generally zero
  - More broadly distributed in small datasets
  - If there are greater numbers of inputs, some of them are more likely to have spurious correlations with the target

Probability densities over sample correlations of uncorrelated variables (plotted as fraction of peak at each sample size)



# RIDGE REGRESSION REDUCES WEIGHTS TO REDUCE OVERFITTING

# Weight magnitudes

- Weights typically have greater magnitudes when there are more basis functions
- These weights are from the polynomial regression example with different orders

| $M=0$ | $M=1$ | $M=3$ | $M=9$ |
|---|---|---|---|
| $w^* = [.19]$ | $\begin{bmatrix} 0.82 \\ -1.27 \end{bmatrix}$ | $\begin{bmatrix} 0.31 \\ 7.99 \\ -25.43 \\ 17.37 \end{bmatrix}$ | $\begin{bmatrix} .35 \\ 232.37 \\ -5321.83 \\ 48568.31 \\ -231639.30 \\ 640042.26 \\ -1061800.52 \\ 1042400.18 \\ -557682.99 \\ 125201.43 \end{bmatrix}$ |

# Weight magnitudes

- This can be understood in terms of the equation for optimal weights,

$$\boldsymbol{w}^* = (X^T X)^{-1} X^T \boldsymbol{y}$$

- This involves inverting the matrix, $X^T X$
- Greater similarity between basis functions (different columns of $X$) leads to smaller eigenvalues, which leads to larger eigenvalues in the inverse

# Ridge regression

- This can be rectified by penalizing large weights within the loss function,

$$\tilde{L}(\boldsymbol{w}) = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 + \lambda \boldsymbol{w}^T \boldsymbol{w}$$

prediction errors are bad

weights are bad

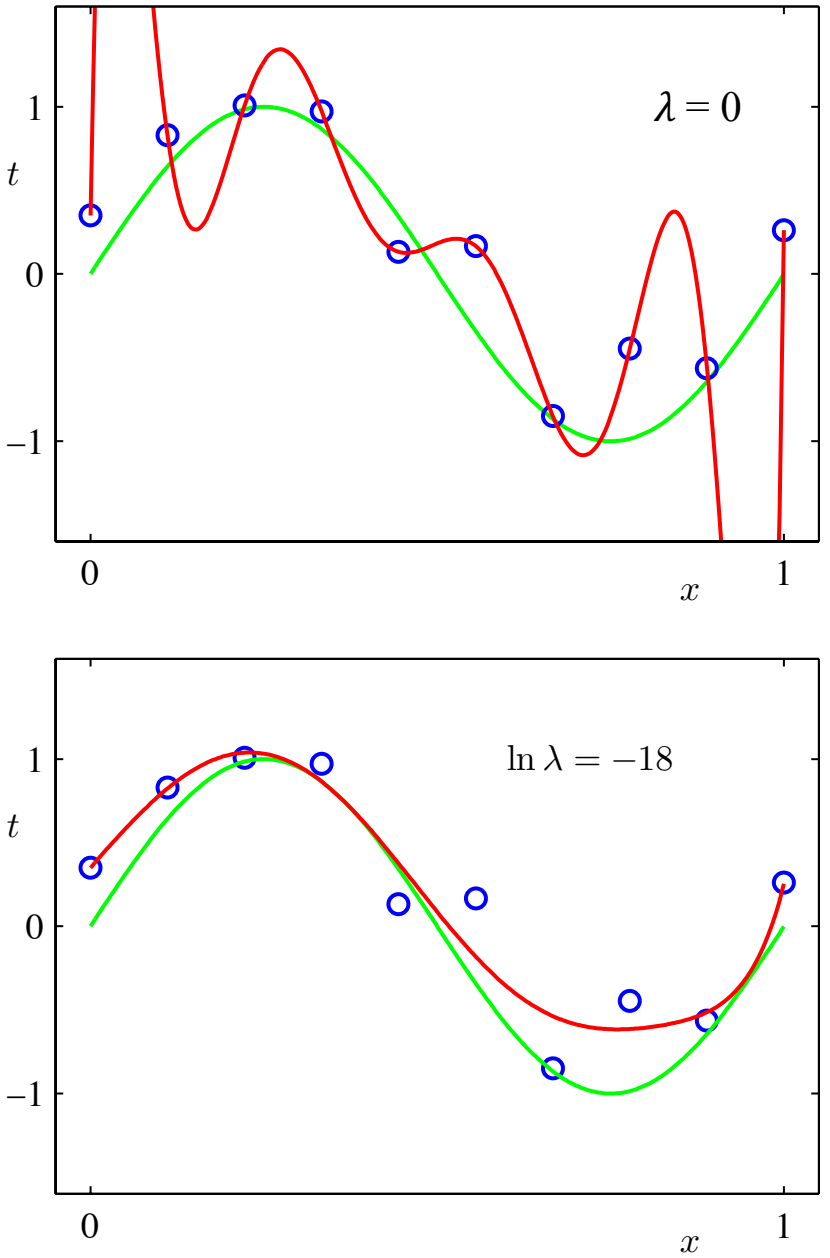
- The optimal weights are then,

$$\boldsymbol{w}^* = (\lambda I + X^T X)^{-1} X^T \boldsymbol{y}$$

- The new entries along the diagonal prevent small eigenvalues
- This is an example of parameter-norm regularization, which is also used in deep networks (we will return to this)

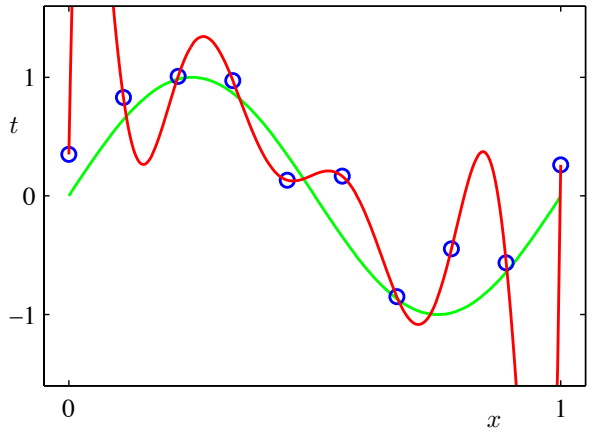
# Ridge regression

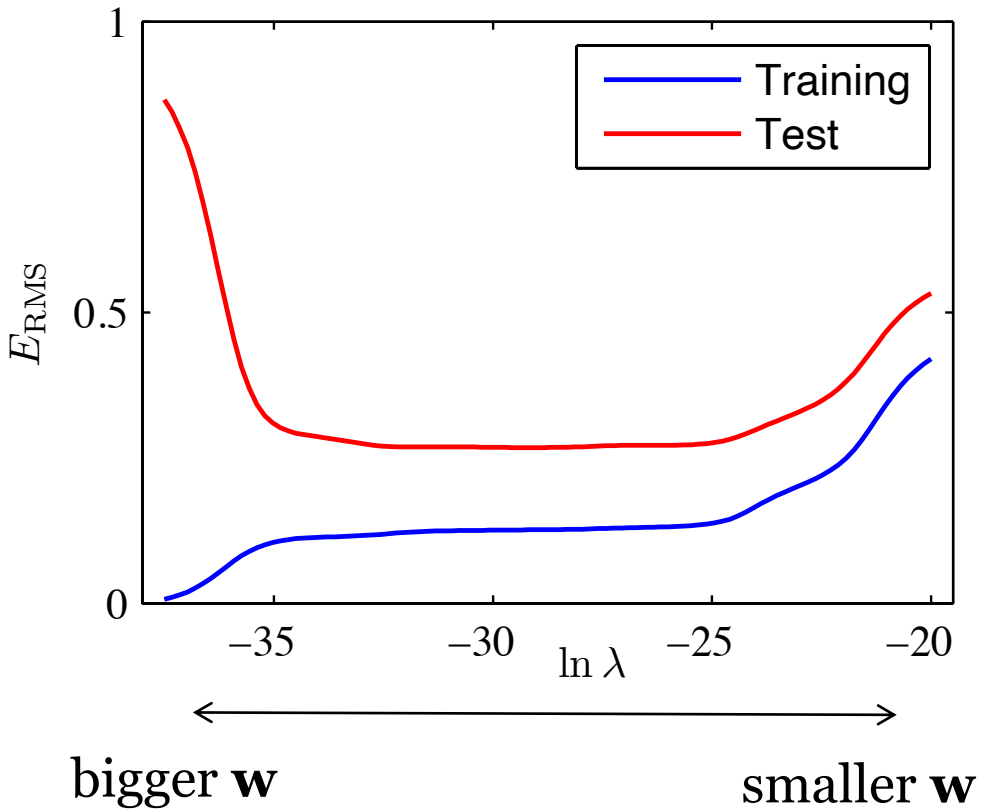
- These examples show 9th-order polynomial regression with and without a penalty on the weights
- The weight penalty allows use of a more complex model without overfitting

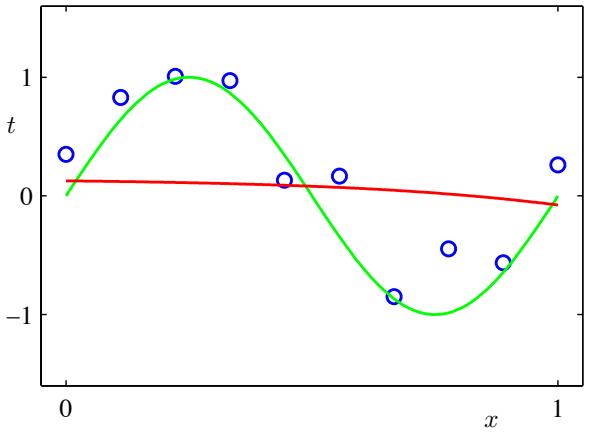



Bishop, Pattern Recognition & Machine Learning

# Ridge regression



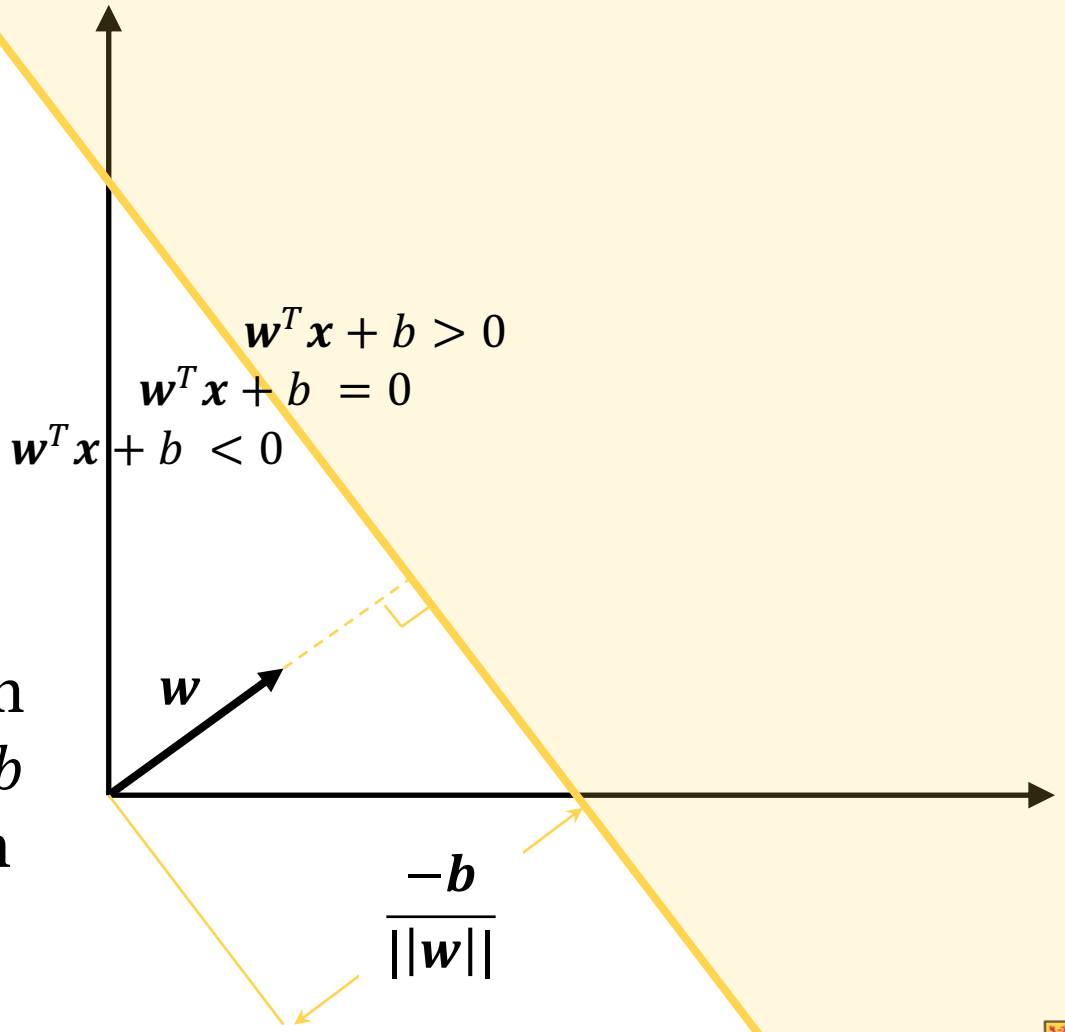



# A LINEAR DISCRIMINANT USES A LINEAR BOUNDARY TO SEPARATE EXAMPLES INTO CATEGORIES

# Linear Discriminant

$$\hat{y} = \begin{cases} 0, & \boldsymbol{w}^T\boldsymbol{x} + b < 0 \\ 1, & \boldsymbol{w}^T\boldsymbol{x} + b \geq 0 \end{cases}$$




$\boldsymbol{w}$ determines the orientation of the decision surface, and $b$ determines its distance from the origin.

# Ways to Find a Linear Discriminant

- Use a linear regression approach with targets -1 and 1 for different categories, minimize squared error
- Fisher's Linear Discriminant
  - Finds vector that maximizes ratio of between-class variance to within-class variance
- Linear support vector machines
- Perceptron learning rule
- Logistic regression

These have an analytical solution, but they are sensitive to outliers

We will focus on these because they are most closely related to deep learning

# PERCEPTRONS ADJUST THE BOUNDARY TO CORRECT CLASSIFICATION ERRORS

# Perceptron

Target values $y \in \{-1,1\}$

Perceptron function $\hat{y}(\boldsymbol{x}) = f(\boldsymbol{w}^T \boldsymbol{x})$

Here again the bias is absorbed into the weights and 1 is appended to the inputs.

$$\mathrm{f(a)} = \begin{cases} 1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

Loss function ("perceptron criterion") ignores correct classifications

$$\mathrm{L}(\boldsymbol{w}) = -\sum_{n \in M} \boldsymbol{w}^T \boldsymbol{x}_n y_n$$

misclassified inputs

# Perceptron learning rule

- Loop through examples and update $\boldsymbol{w}$ to reduce error on the $n^{th}$ misclassified example:

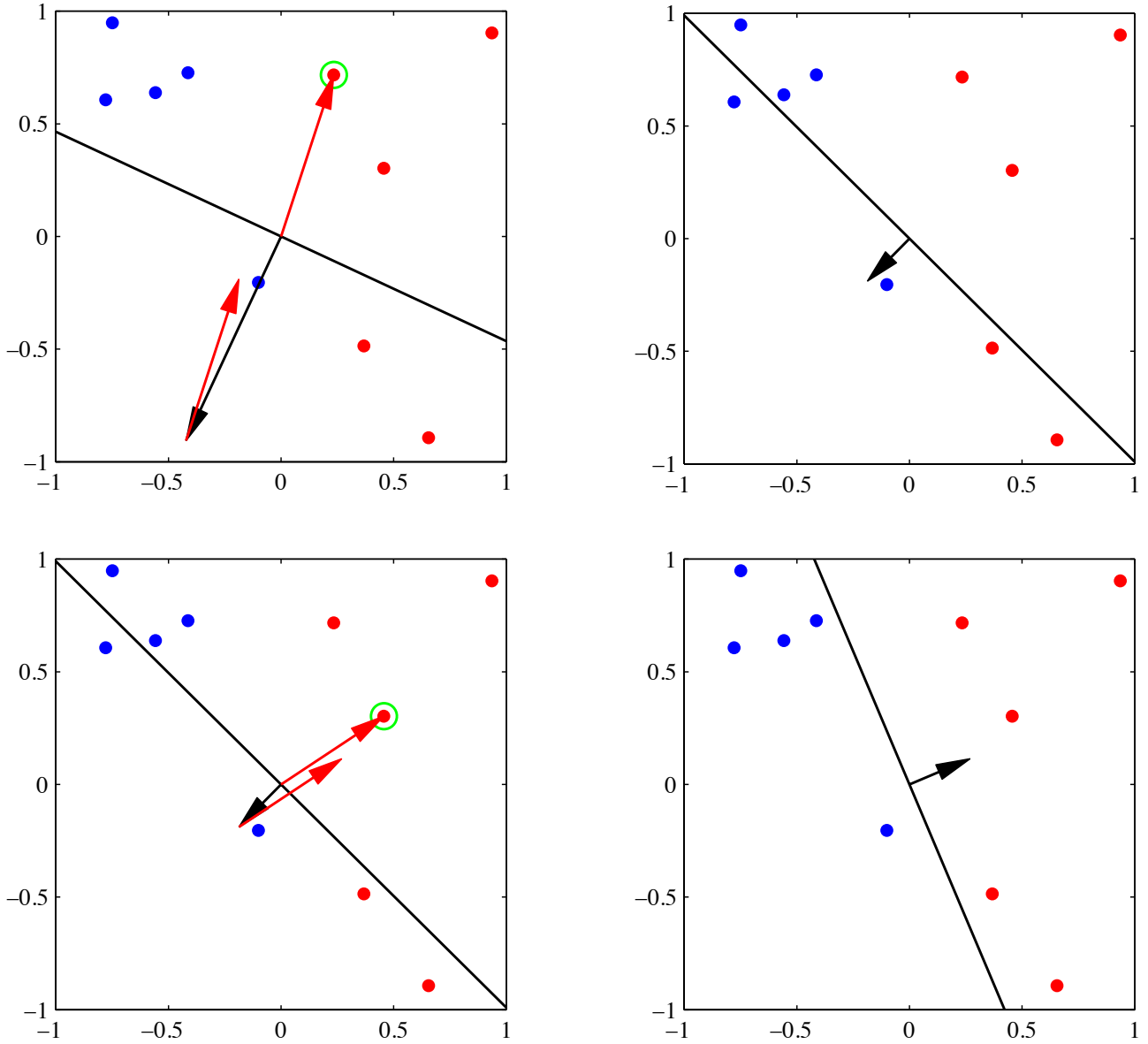
$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{\tau} - \eta \nabla L(\boldsymbol{w}) = \boldsymbol{w}^{\tau} + \eta \boldsymbol{x}_n y_n,$$

  where $\tau$ is the step number and $\eta$ is a learning rate

- This is a variation of gradient descent that uses the gradient of the loss for a single example at a time
- Only the direction of the weight vector is important (not the magnitude) so the learning rate $\eta$ is not very important, can be set to 1
- Correctly classifies linearly separable examples, but does not terminate if the examples are not linearly separable, and the solution may not generalize well

# Perceptron learning example

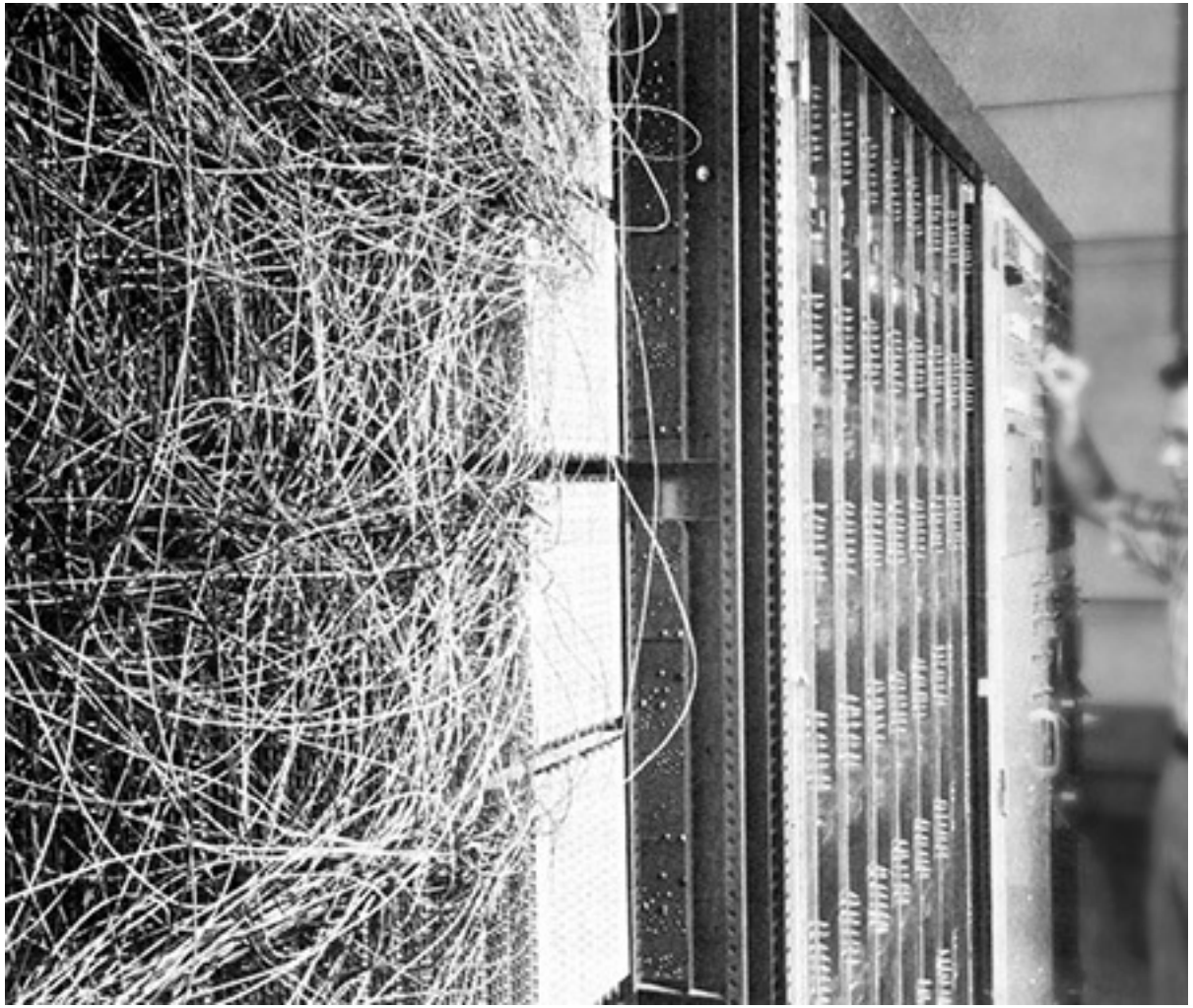
- Here red dots have target $y = 1$; blue have target $y = -1$
- The black vector is the current $\boldsymbol{w}$ and the red arrow is the currently selected $\boldsymbol{x}$
- We want $\boldsymbol{w}$ to point toward the red dots
- Pick $\boldsymbol{x}$ one at a time; if it's misclassified add $\boldsymbol{x}y$ to $\boldsymbol{w}$




Bishop, Pattern Recognition & Machine Learning

# Perceptron hardware

- Developed in 1958 for image recognition with a 400-pixel camera



http://www.rutherfordjournal.org/article040101.html

# LOGISTIC REGRESSION CATEGORIZES INPUTS PROBABILISTICALLY

# Logistic regression overview

- Models the probability that a vector $\boldsymbol{x}$ belongs to a certain category as,

$$p(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x} + b}}$$

- This is learned from training examples with targets $y = 0$ (not in the category) and $y = 1$ (in the category)
- The parameters $\boldsymbol{w}$ and $b$ are found through iterative minimization of a loss
- This isn't a classifier on its own, but it just needs a threshold such as $p(\boldsymbol{x}) = 0.5$

# Logits

- The odds of an outcome are $p/(1-p)$, where $p$ is the probability
- In the equation, $p(\boldsymbol{x}) = \left(1 + e^{-\boldsymbol{w}^T\boldsymbol{x}+b}\right)^{-1}$ the input, $\boldsymbol{w}^T\boldsymbol{x} + b$, is the log-odds of $\boldsymbol{x}$ belonging to the category
- This quantity is called the "logit", which is short for "logistic unit" (the name of the units of log-odds)
- The logistic function, $\left(1 + e^{-l}\right)^{-1}$, maps logit in $[-\infty, \infty]$ to probability in [0,1]
- Note,

$$p = \frac{1}{1 + e^{-\ln\frac{p}{1-p}}} = \frac{1}{1 + \frac{1-p}{p}} = \frac{1}{\left(\frac{1}{p}\right)}$$

# Log loss function

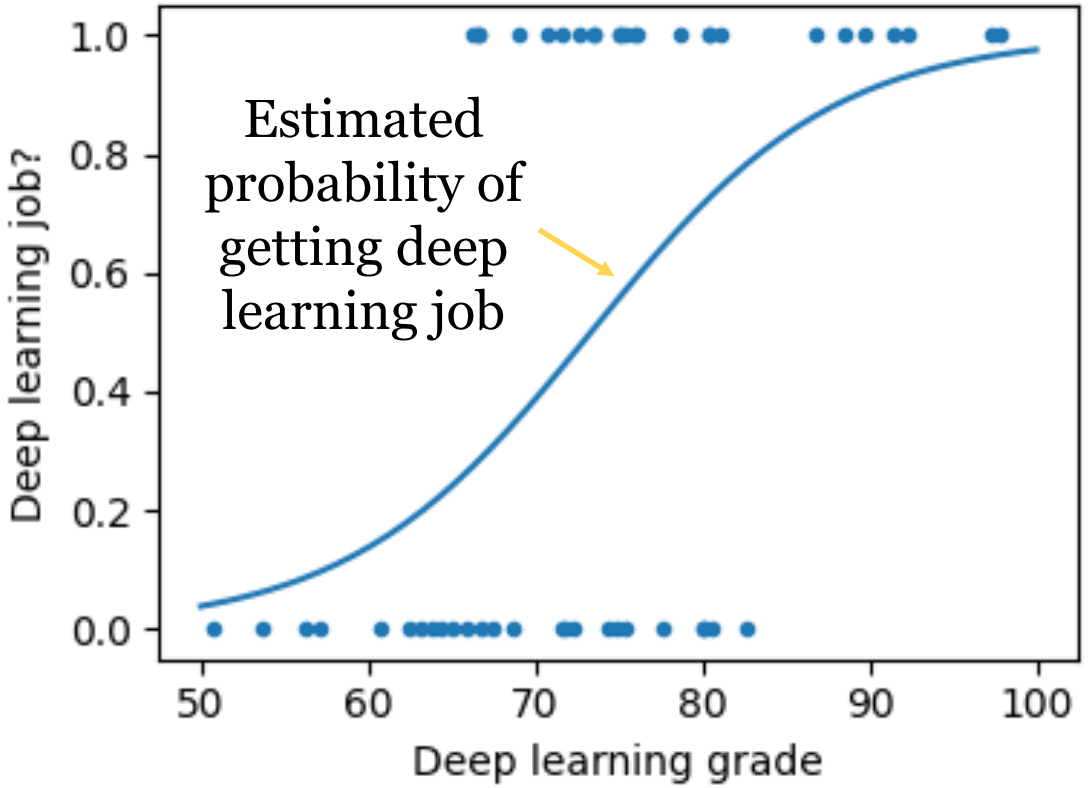
- Loss for $k^{th}$ example is:
  - $L_k = -\ln p(\boldsymbol{x}_k)$ if $y_k = 1$ and
  - $L_k = -\ln(1 - p(\boldsymbol{x}_k))$ if $y_k = 0$
- This can also be expressed as:

$$L_k = -y_k \ln p(\boldsymbol{x}_k) - (1 - y_k) \ln(1 - p(\boldsymbol{x}_k))$$

- This is also called the binary cross-entropy loss

# Example

- Hypothetical examples of whether people who took a deep learning course got a deep learning job (1) or not (0)

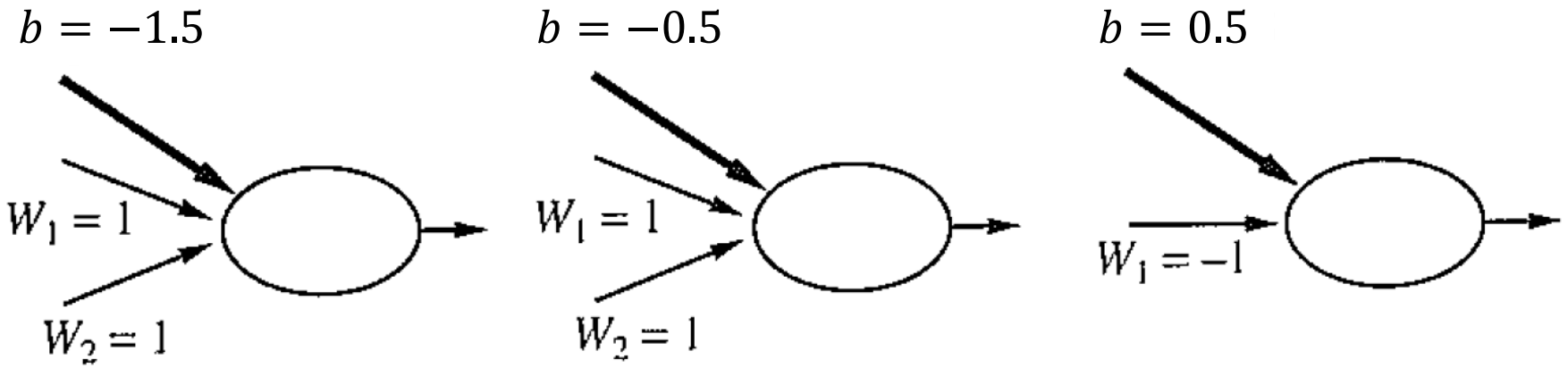


# LINEAR CLASSIFIERS CAN IMPLEMENT BOOLEAN LOGIC, EXCEPT FOR XOR

# Logic operations with threshold units

- What operations do these networks perform given that

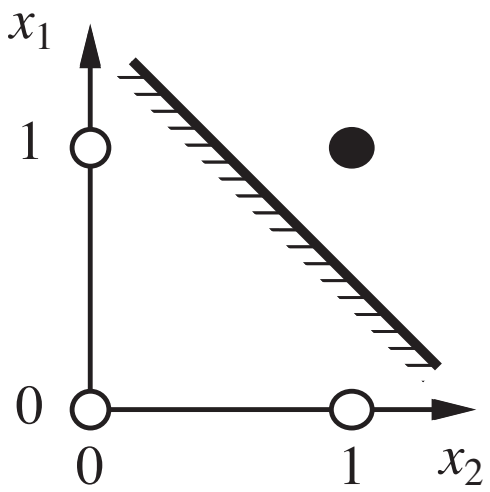
$$x_i \in \{0,1\} \qquad y = \begin{cases} 1 \ \ if \ \boldsymbol{w}^T \boldsymbol{x} + b > 0 \\ 0 \ \ otherwise \end{cases}$$



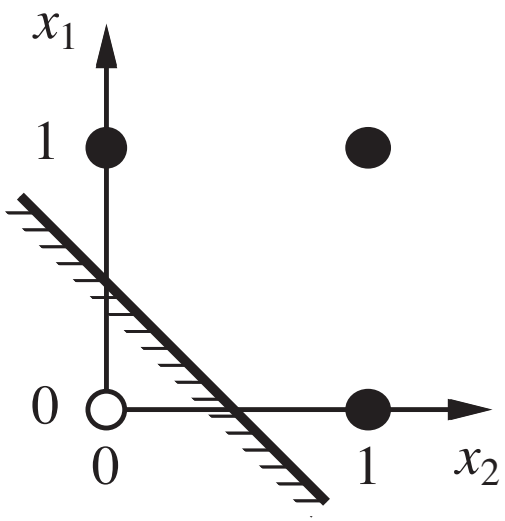

Russell & Norvig, Artificial Intelligence

# Logic operations in threshold units

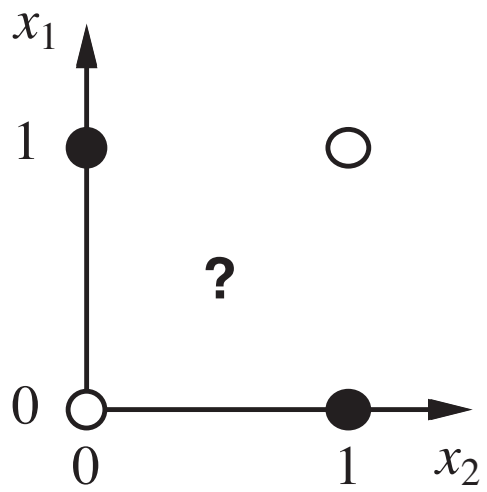
- A linear model can only distinguish linearly separable groups of potential inputs




(a) $x_1$ **and** $x_2$




(b) $x_1$ **or** $x_2$




(c) $x_1$ **xor** $x_2$

Russell & Norvig,
Artificial Intelligence

# IF THERE ARE MULTIPLE CATEGORIES, IT'S BEST TO MAKE A PREDICTOR FOR EACH AND CHOOSE THE ONE WITH THE HIGHEST PRE-THRESHOLD OUTPUT

# Exclusive categories vs. independent categories

- We're talking here about multi-class problems, or multinomial classification
  - In this type of problem, a given example can belong to one of several categories (e.g., fish, bird, frog, snake) rather than just two (e.g., living, non-living) as in binary classification
- You may instead have multiple binary classification problems
  - For example, you may want to know whether someone has each of several different diseases; they may have more than one (e.g., both Parkinson's disease and Lyme disease) or none
  - In this case you just need multiple binary classifiers

# Ways to distinguish between multiple categories

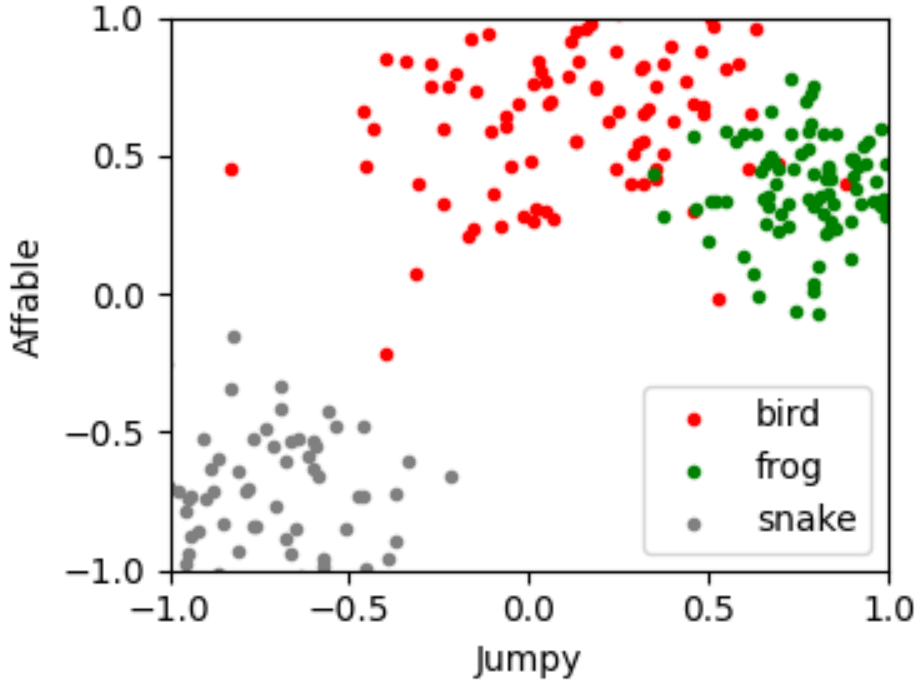
1. Use a separate binary member/non-member classifier for each category and hope only one is positive
2. Use a separate classifier to distinguish between each pair of possibilities and hope one is positive more than the others

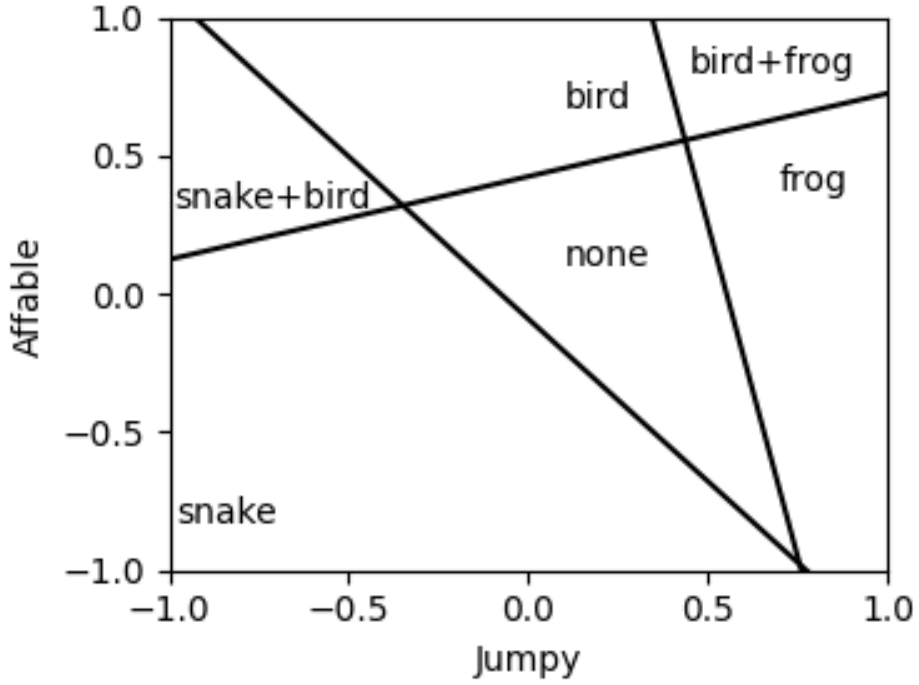
Problematic

3. Learn a binary member/non-member classifier for each category and adopt the category of the classifier with the highest output

# Problem with using independent classifiers

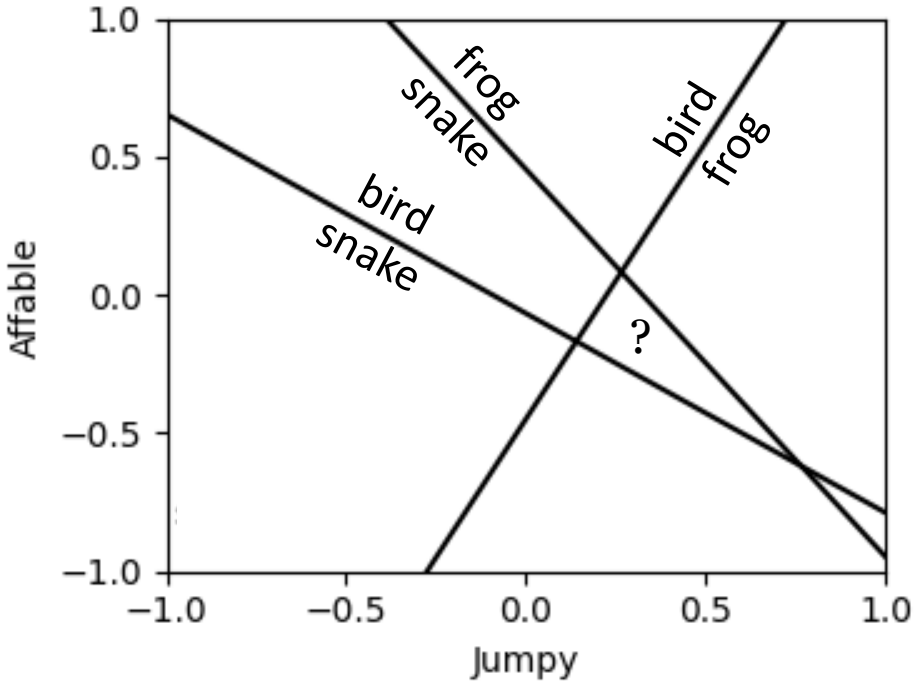
- Unclear what to do if:
  - Multiple classifiers return a positive result
  - No classifiers return a positive result



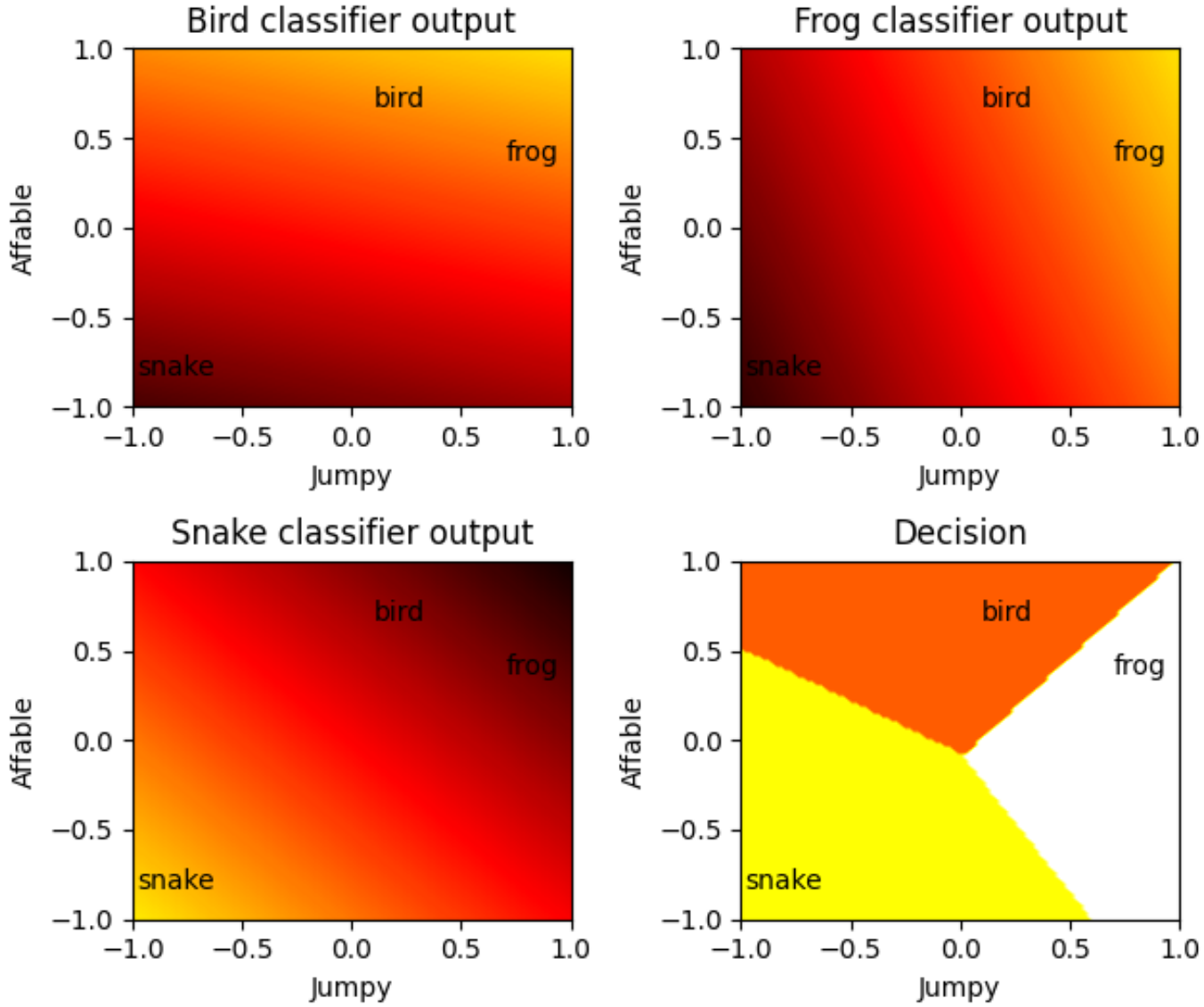
# Problem with using two-way classifiers

- This approach can also create ambiguous regions of the input space
- For example, in the area marked "?" each classifier wins one contest and loses the other
- Furthermore, with $n$ categories, $O(n^2)$ classifiers are needed



# Better to use class of classifier with the highest output

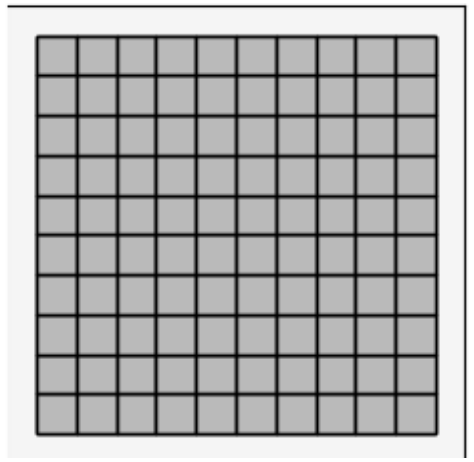
- Train multiple one-vs-rest classifiers
- This approach does not produce ambiguous regions (except with exactly equal outputs)

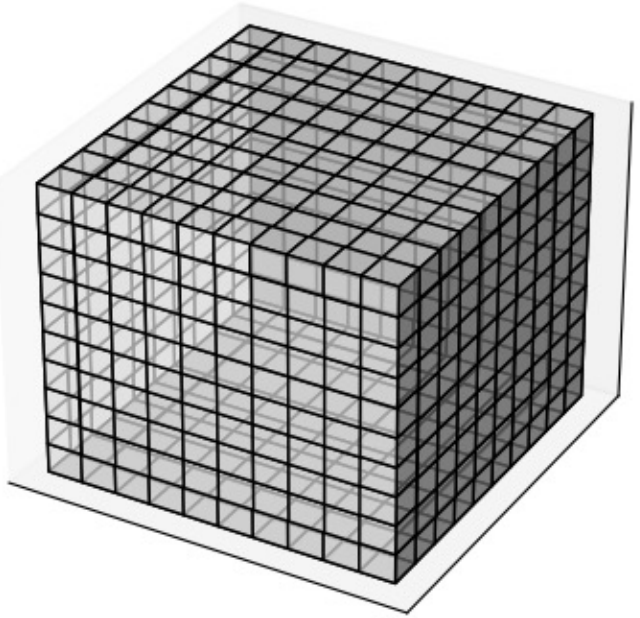


# THE NUMBER OF BASIS FUNCTIONS NEEDED TO TILE A SPACE DEPENDS EXPONENTIALLY ON THE DIMENSION OF THE SPACE

# Curse of dimensionality

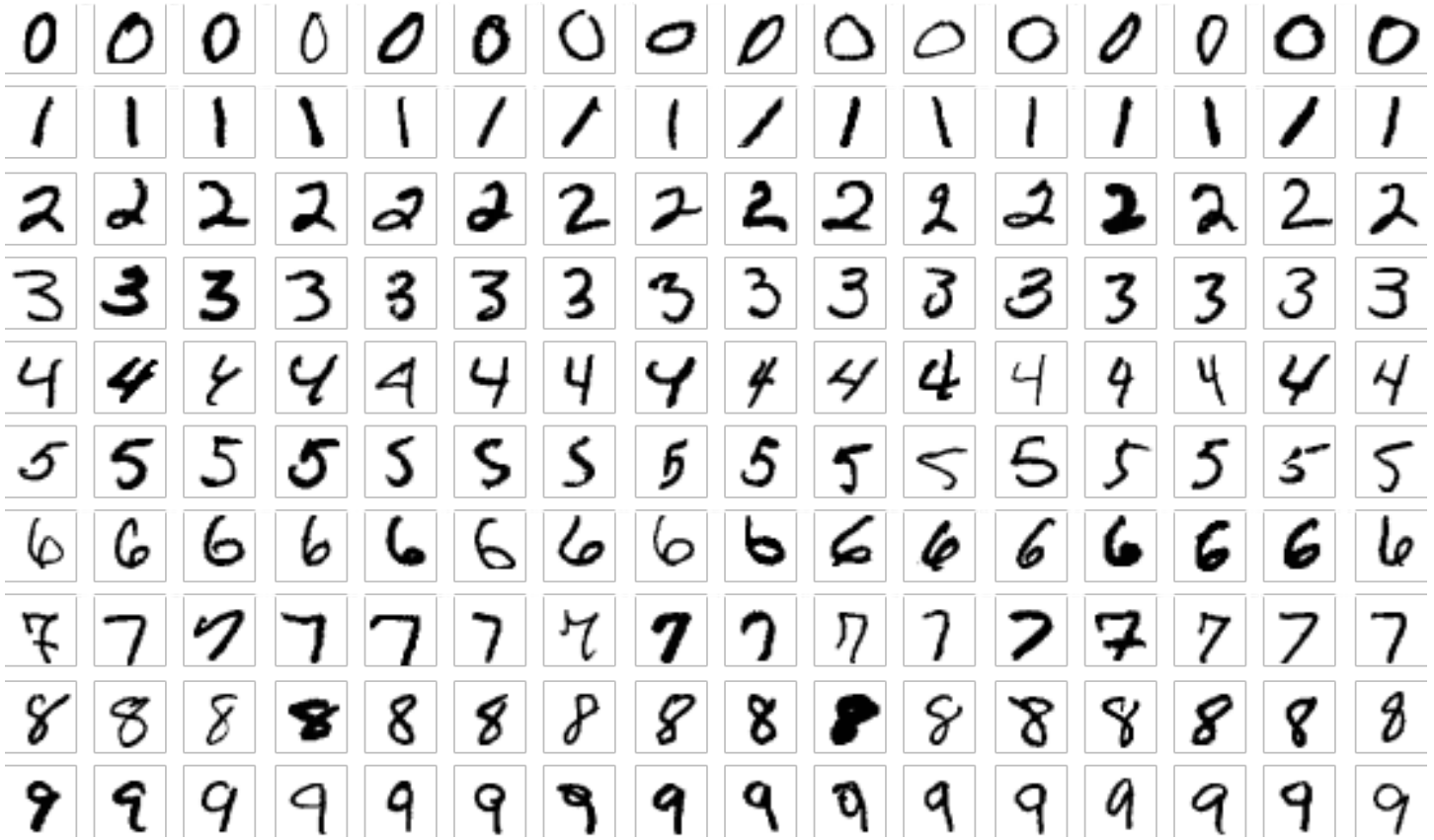
- We have seen that linear methods are useful for two kinds of problems:
  - Finding linear functions of (or boundaries in) high-dimensional spaces
  - Finding nonlinear functions of (or boundaries in) low-dimensional spaces
- Using these methods for nonlinear functions of a high-dimensional space would require tiling the space with basis functions
- This is not practical because the number of fixed basis functions needed grows exponentially with the dimension





# Curse of dimensionality

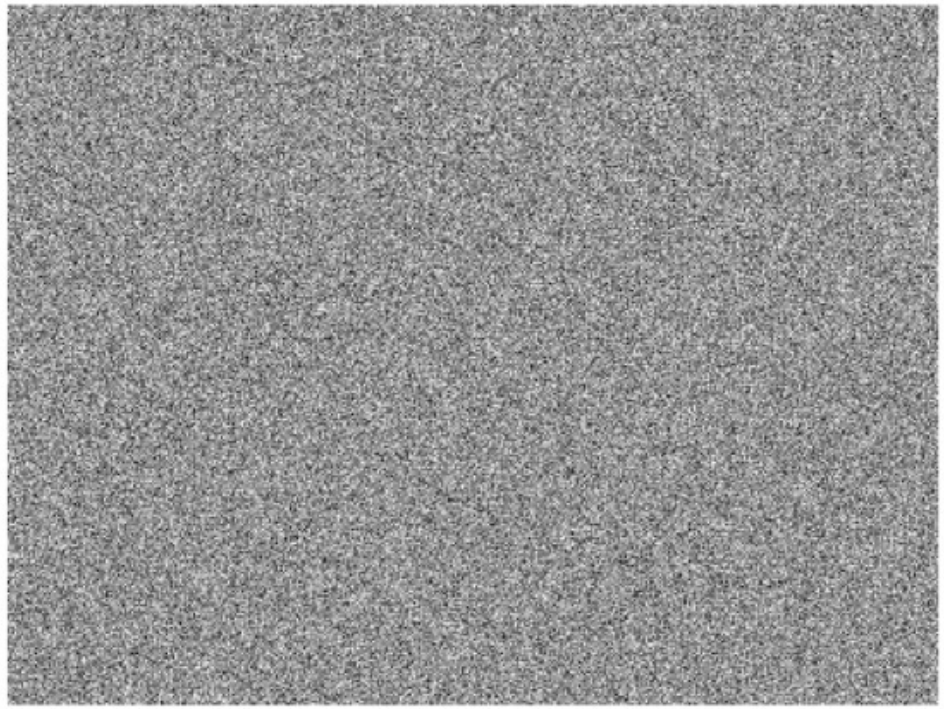
Example: MNIST handwritten digits (simple problem) has $28^2 = 784$D inputs



https://commons.wikimedia.org/wiki/File:MnistExamplesModified.png

# We don’t care about the whole space

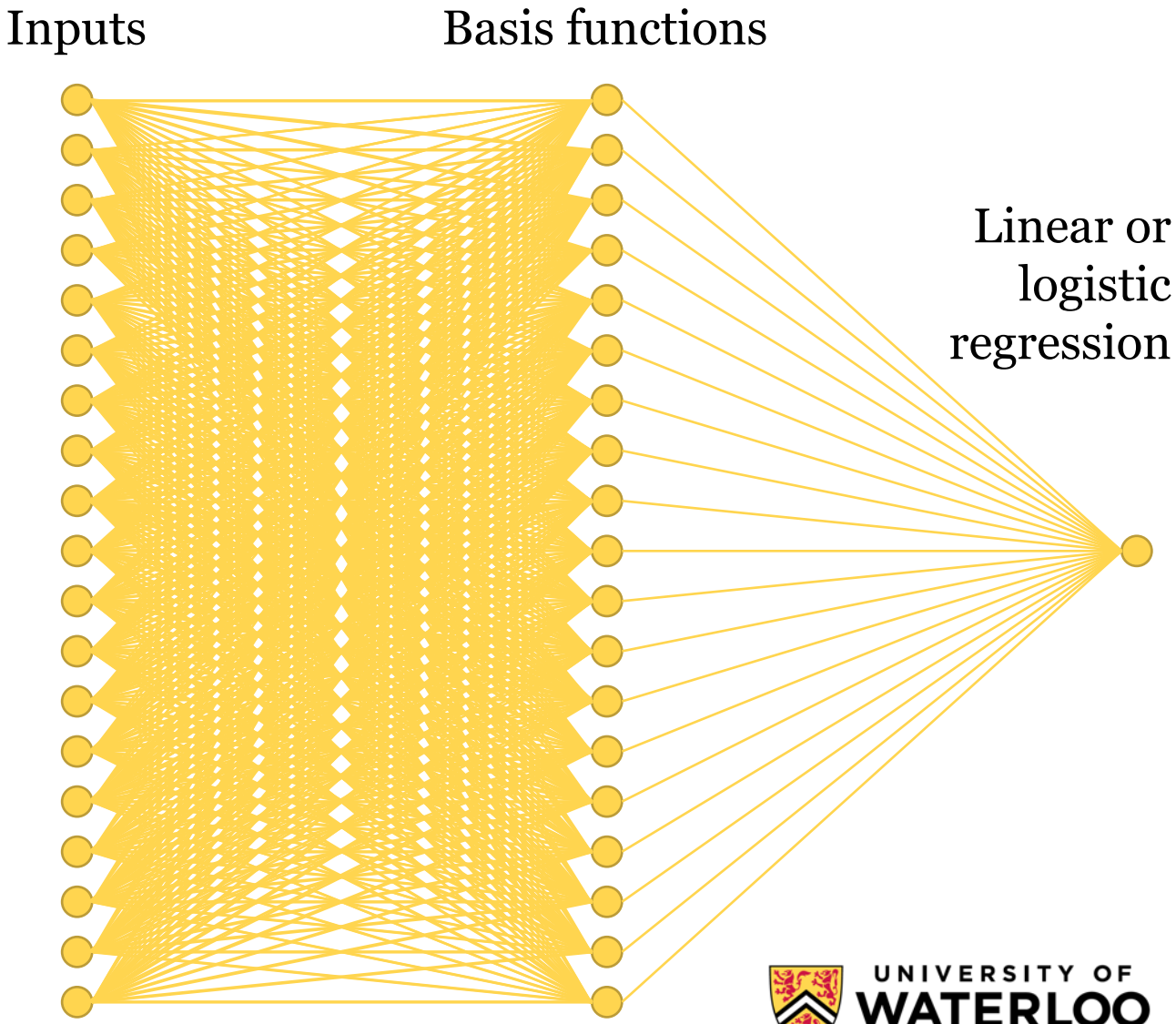
Images of independent random samples look like this



Goodfellow, Bengio & Courville (2015) Deep Learning

# Curse of dimensionality

- Multi-layer neural networks take advantage of this by adapting basis functions so that
  - Regions of variation correspond to regions over which input typically varies
  - Directions of variation correspond to directions over which output typically varies



# Summary

1. Regression and classification are prediction of continuous and categorical values, respectively
2. Linear regression and classification use linear functions of the inputs
3. Regression and classification are supervised learning problems
4. Linear regression typically minimizes squared-error loss
5. One can have too many or too few basis functions
6. One can have too many or too few inputs

# Summary

7. Ridge regression reduces weights to reduce overfitting
8. A linear discriminant uses a linear boundary to separate examples into different categories
9. Perceptrons adjust the boundary to correct classification errors
10. Logistic regression categorizes inputs probabilistically
11. Linear classifiers can implement Boolean logic, except for XOR
12. If there are multiple categories, it's best to make a predictor for each and choose the one with the highest pre-threshold output
13. The number of basis functions needed to tile a space depends exponentially on the dimension of the space

# References


- Bishop (2006) *Pattern Recognition & Machine Learning*, Springer
- Goodfellow, Bengio & Courville (2016) *Deep Learning*, MIT Press
- Russel & Norvig (2002) *Artificial Intelligence: A Modern Approach*, 2nd Ed., Pearson