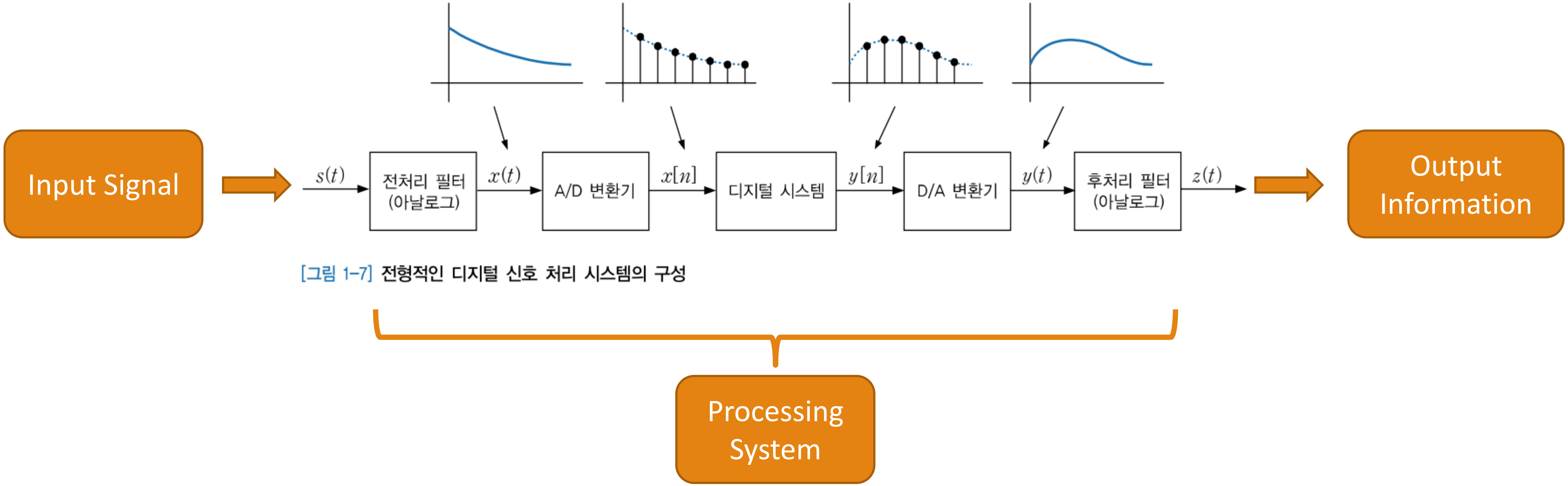
딥러닝을 활용한 **디지털 영상 처리**

Digital Image Processing via Deep Learning

Lecture 1 – Introduction to Image Processing and Classification

Thinking back to Signal Processing

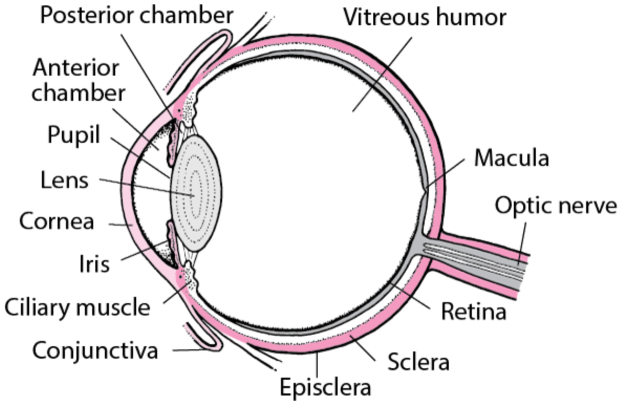


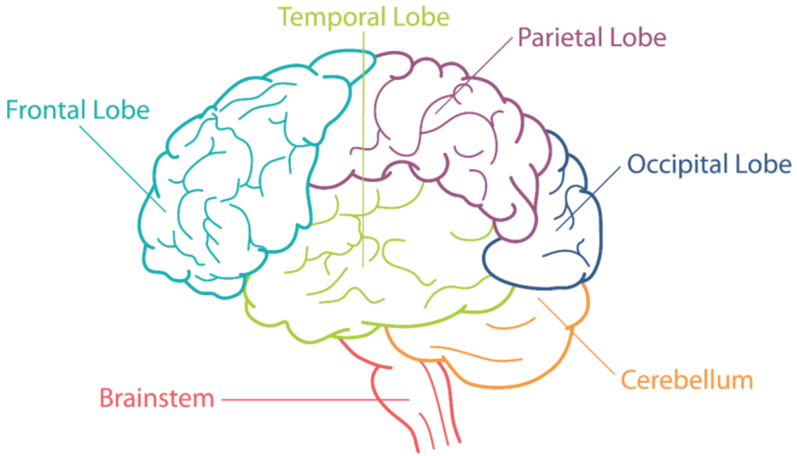
[그림 1-7] 전형적인 디지털 신호 처리 시스템의 구성

How humans see the World

We observe the world via our eyes mostly.
Our eyes process the image as a signal and send it to our brain.
(Similar to pre-processing of a signal)
The brain receives the signal and determines what we are seeing.

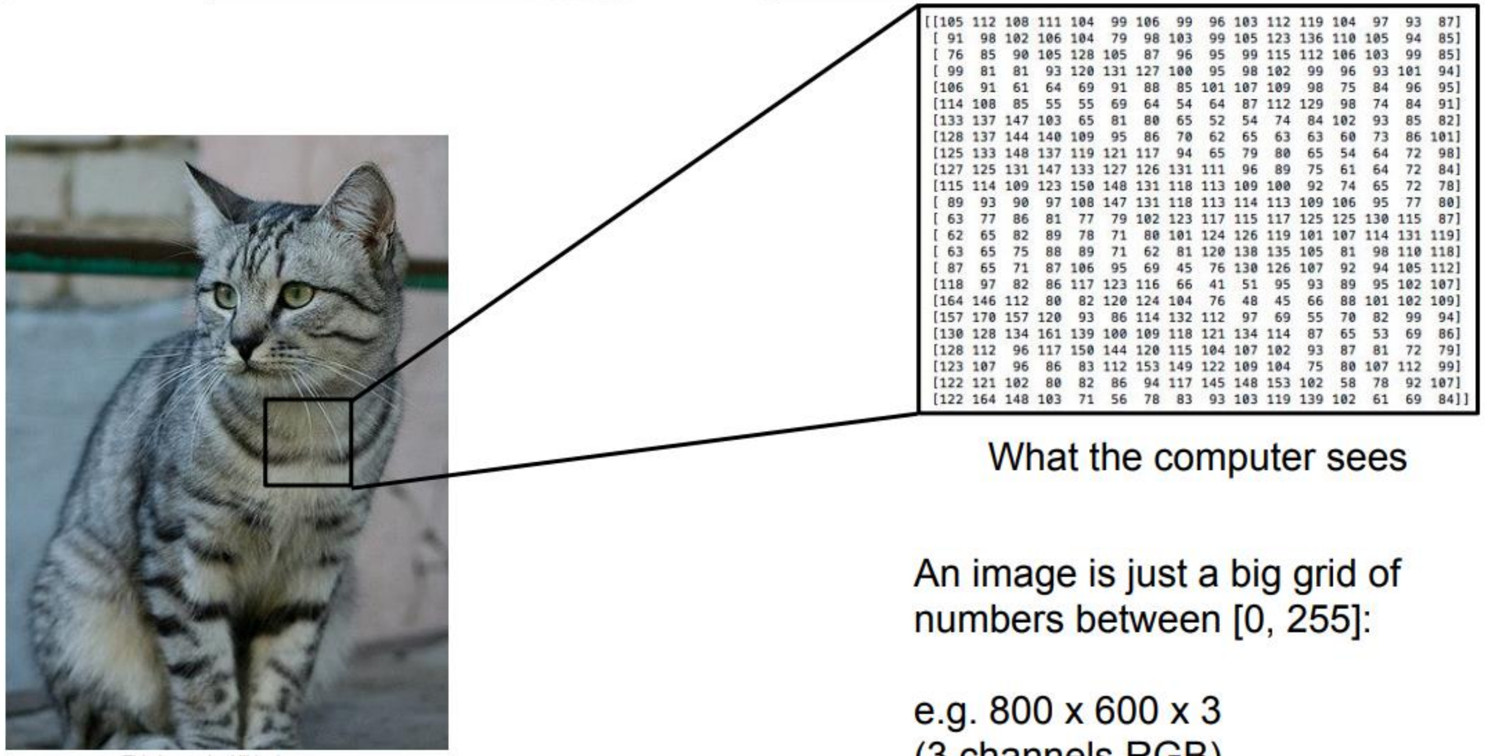
A Cute Cat





Treat Image as a Signal

The computer treats image/images from a video as a matrix filled in with numbers representing grayscale intensity and RGB colour information

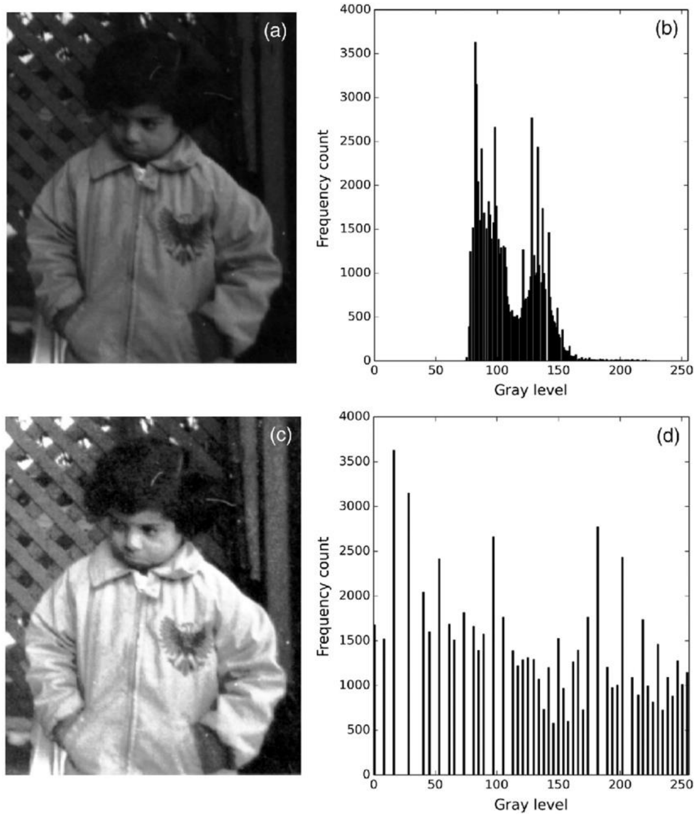


An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Image Processing Techniques

Histogram Equalization



Edge Detection

Feature point extraction

Image Classification

Knowing what is in the image / what the image indicates is a major challenge in computer vision

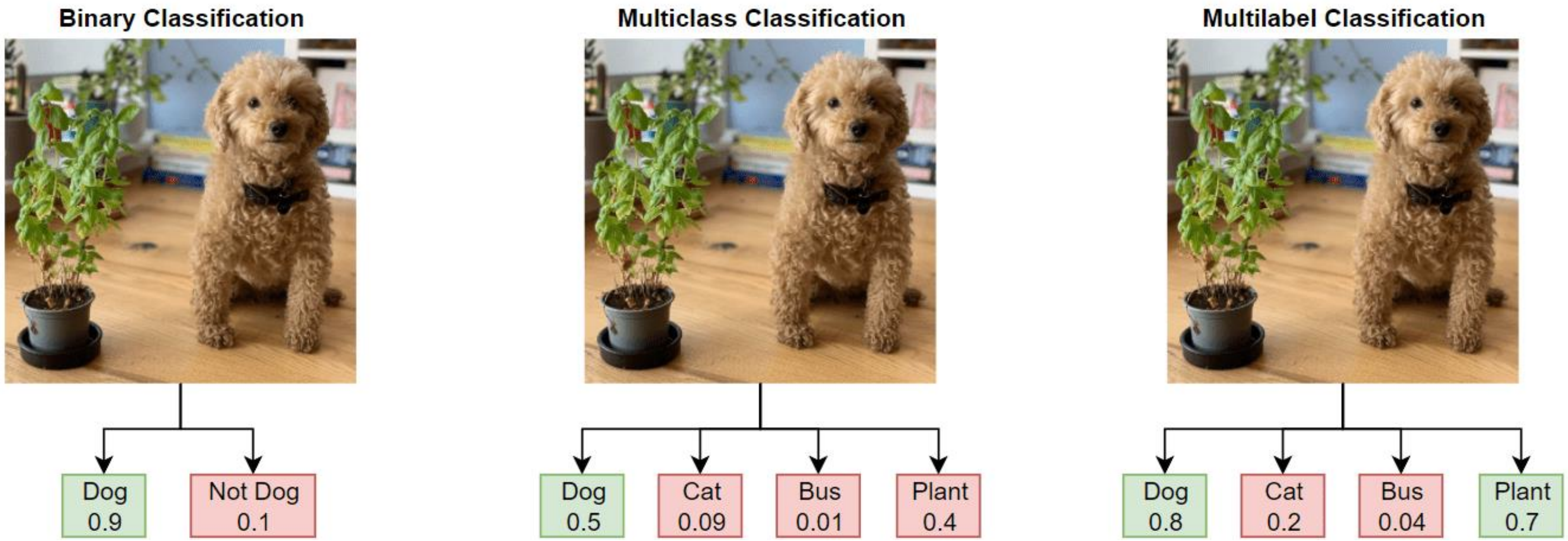


Image Classification

```python
def classify_image(image):
  # Some magic here?
  return class_label
```

Input Image to be classified

Output label

Magic Box

Cat

Image Classification

How can we set up the magic box?

Attempt 1: Find edges of the image and use feature extractor to identify shapes
Drawback: Extremely complex algorithms and set of rules are different for all features.

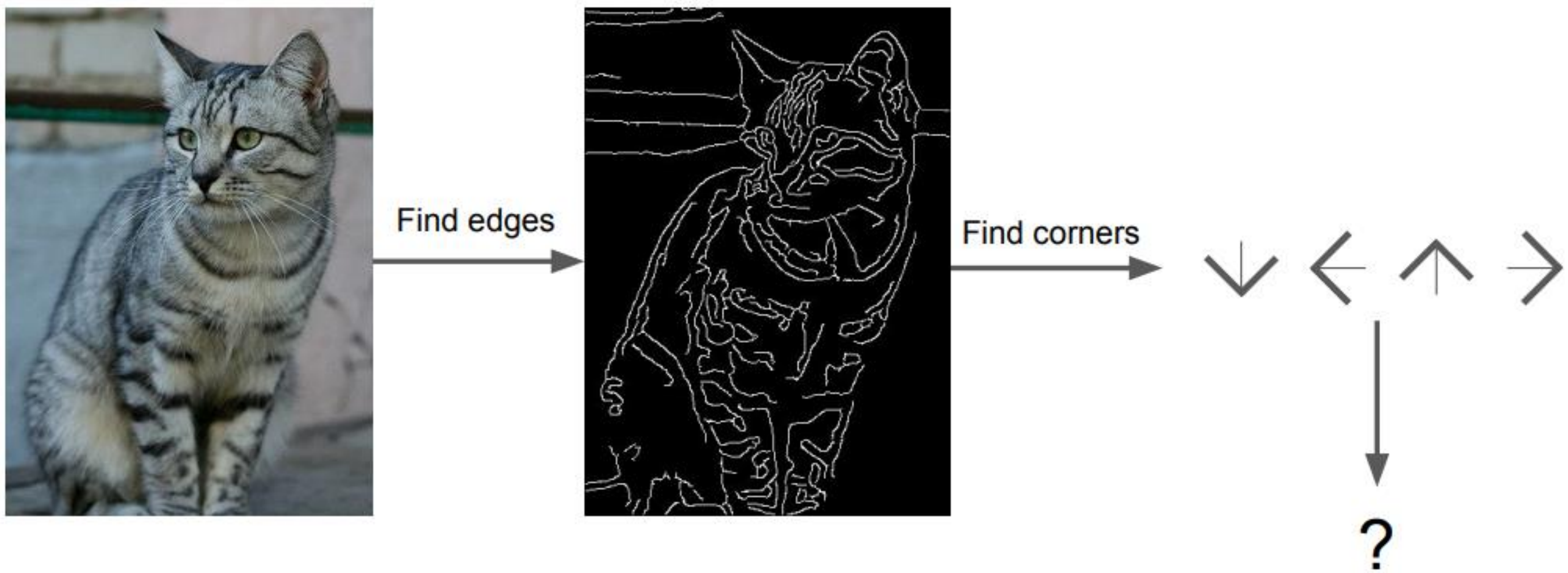


Image Classification

Attempt 2: Data Driven Approach

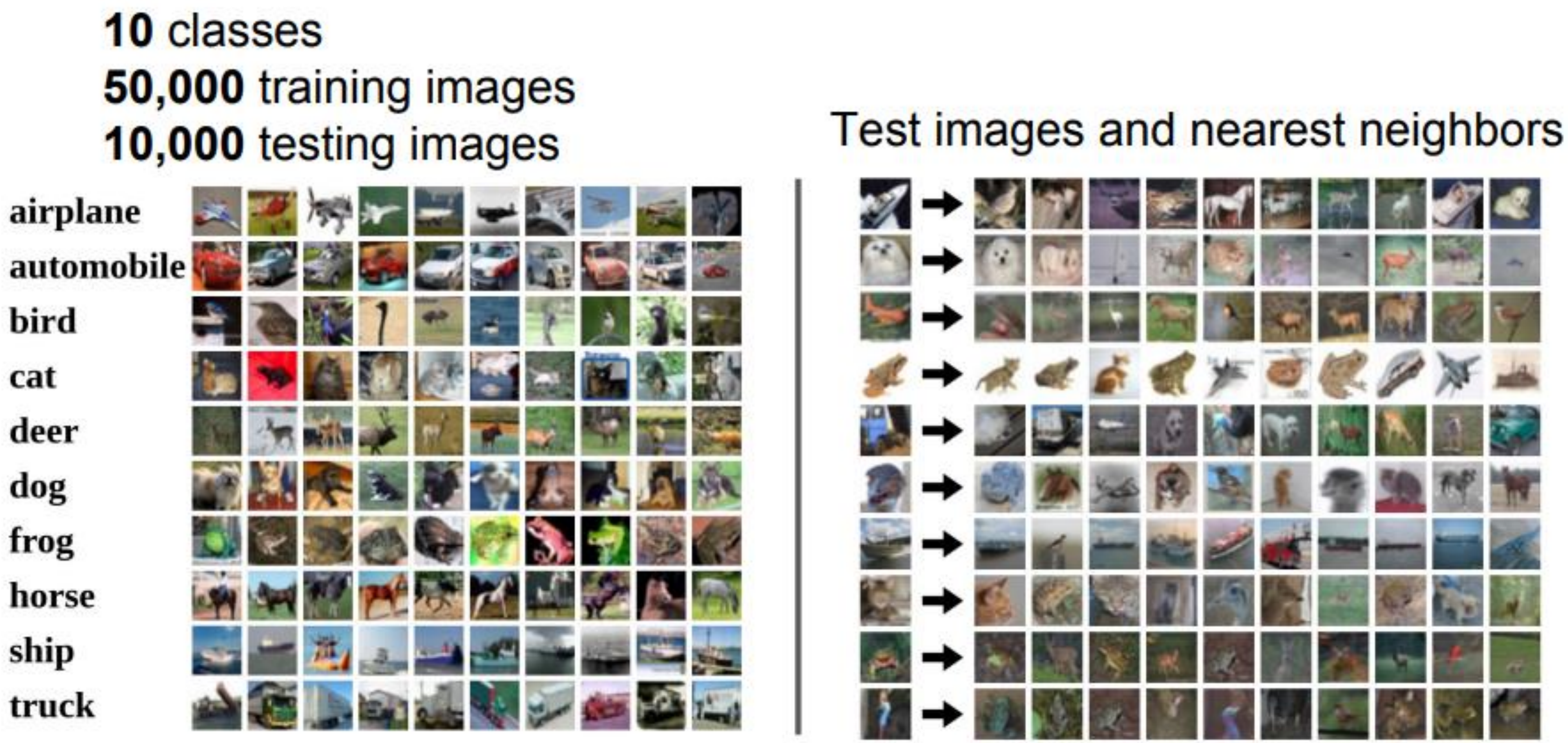


Image Classification: K Nearest Neighbor

K Nearest Neighbor (KNN):

- For training, simply memorise all training data.
- For testing, for each test image, find the closest train image and predict the label of the nearest image

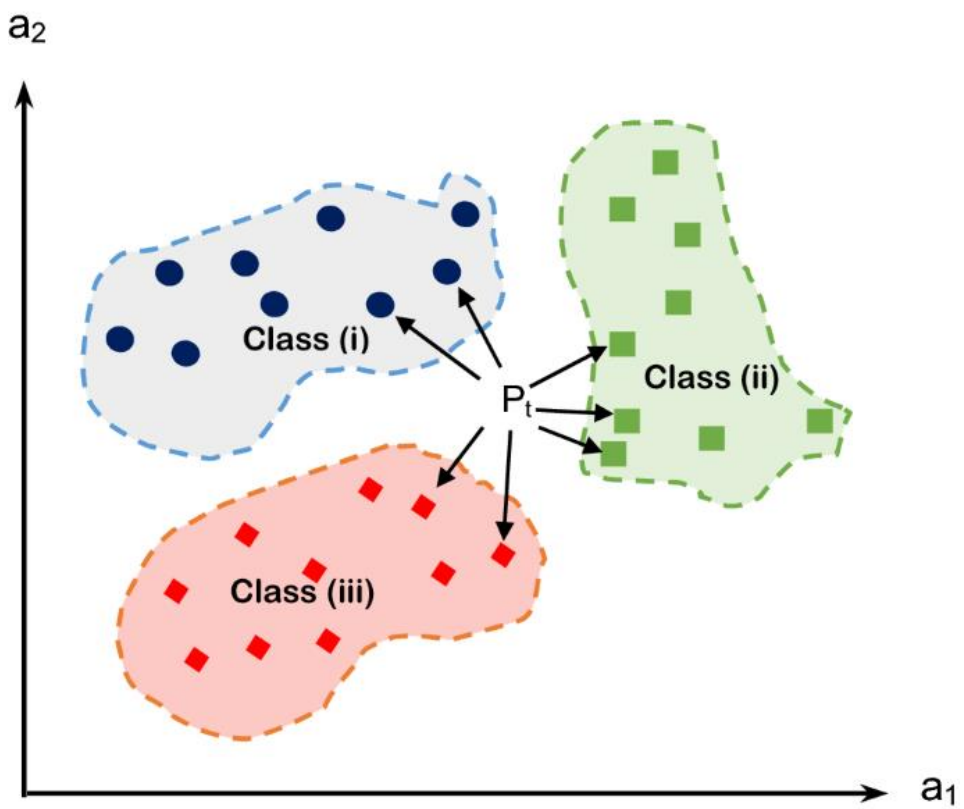


Image Classification: K Nearest Neighbor

L1 Distance: $d_{L1}(I_1, I_2) = \sum_i \left| I_1^i - I_2^i \right|$

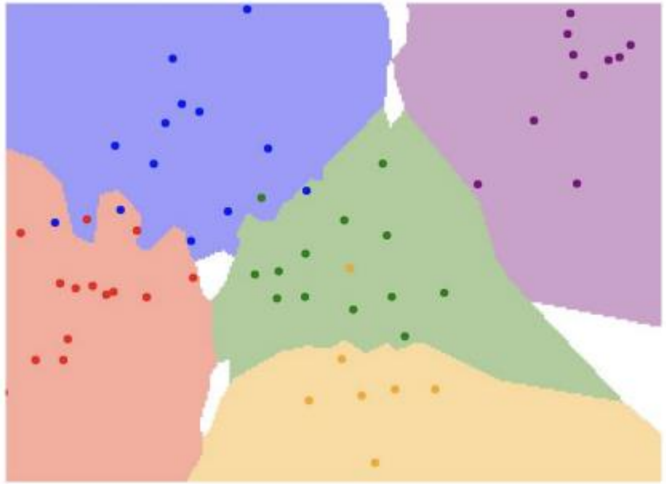
L2 Distance: $d_{L2}(I_1, I_2) = \sqrt{\sum_i \left(I_1^i - I_2^i\right)^2}$, where I_n is a set of images and i indicates for a class.

$$\left| \overset{\text{test image}}{\begin{bmatrix} 56 & 32 & 10 & 18 \\ 90 & 23 & 128 & 133 \\ 24 & 26 & 178 & 200 \\ 2 & 0 & 255 & 220 \end{bmatrix}} - \overset{\text{training image}}{\begin{bmatrix} 10 & 20 & 24 & 17 \\ 8 & 10 & 89 & 100 \\ 12 & 16 & 178 & 170 \\ 4 & 32 & 233 & 112 \end{bmatrix}} \right| = \overset{\text{pixel-wise absolute value differences}}{\begin{bmatrix} 46 & 12 & 14 & 1 \\ 82 & 13 & 39 & 33 \\ 12 & 10 & 0 & 30 \\ 2 & 32 & 22 & 108 \end{bmatrix}} \xrightarrow{\text{add}} 456$$

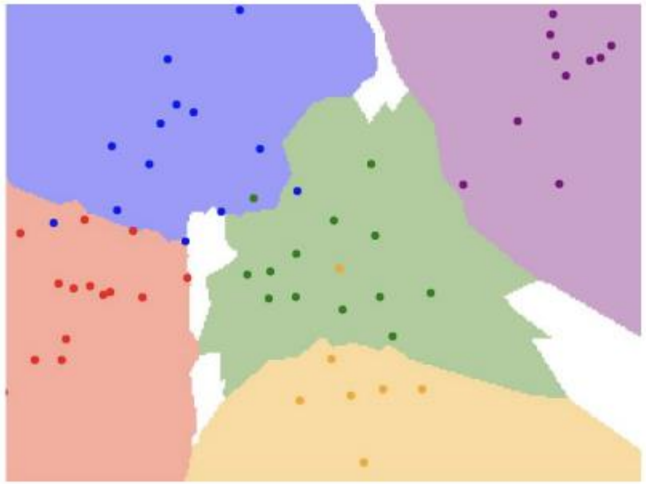
Image Classification: K Nearest Neighbor

KNN takes the majority votes from K nearest points

K = 1



K = 3



K = 5

Try yourself: http://vision.stanford.edu/teaching/cs231n-demos/knn/

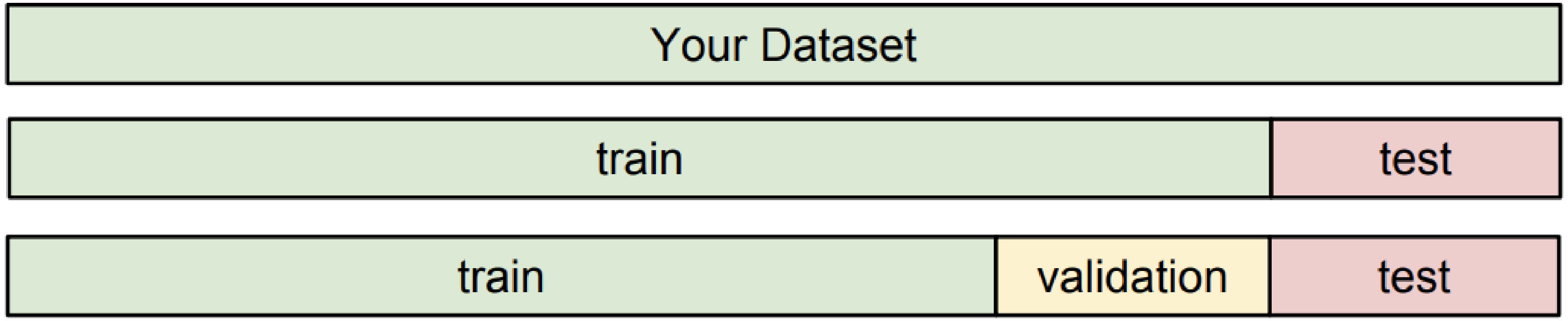
Setting Hyperparameters

What is the best K?
What is the best distance metric to use?

These are hyperparameters: choice about the algorithm that we set rather than learn.

The hyperparameters are very problem-dependent. Must try them all out and see what works the best.

Given a Dataset, we usually leave a portion out for testing.
In addition, we can leave a validation set for selecting appropriate hyperparameters.



Setting Hyperparameters

Cross-Validation: Split the data into folds, try each fold as validation, and average the results.

It is useful for small datasets, but not frequently used in Deep Learning

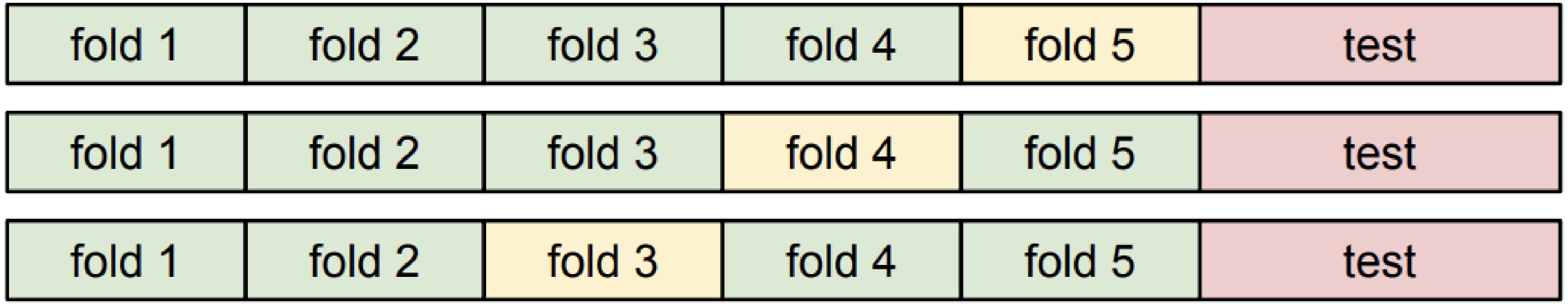


Image Classification

The idea of KNN is good but KNN is actually never used in image classification.

- Very slow at test time: We want an algorithm that may be slow during training but fast during testing. KNN is fast during training since it simply memorises data but slow during testing due to distance computation.
- The Distance metrics on pixels are not informative

(all 3 images have same L2 distance to the one on the left)

Image Classification: Linear Classification

3072x1

$$\boxed{f(x,W)} = \boxed{W}\boxed{x} + \boxed{b}$$

10x1

10x1 10x3072

Image

Array of **32x32x3** numbers (3072 numbers total)

$\longrightarrow f(\mathbf{x},\mathbf{W}) \longrightarrow$ **10** numbers giving class scores

$\uparrow$

W

parameters or weights

Image Classification: Linear Classification

Example with an image with 4 pixels and 3 classes (cat / dog / ship)

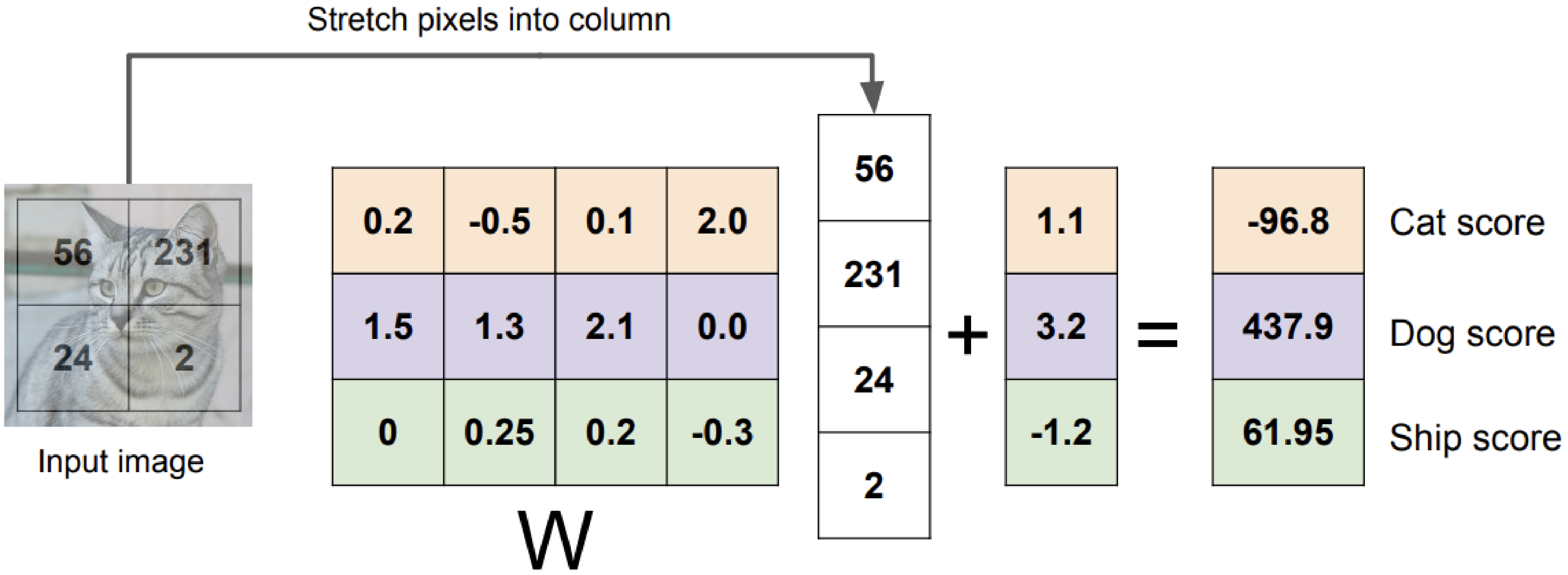
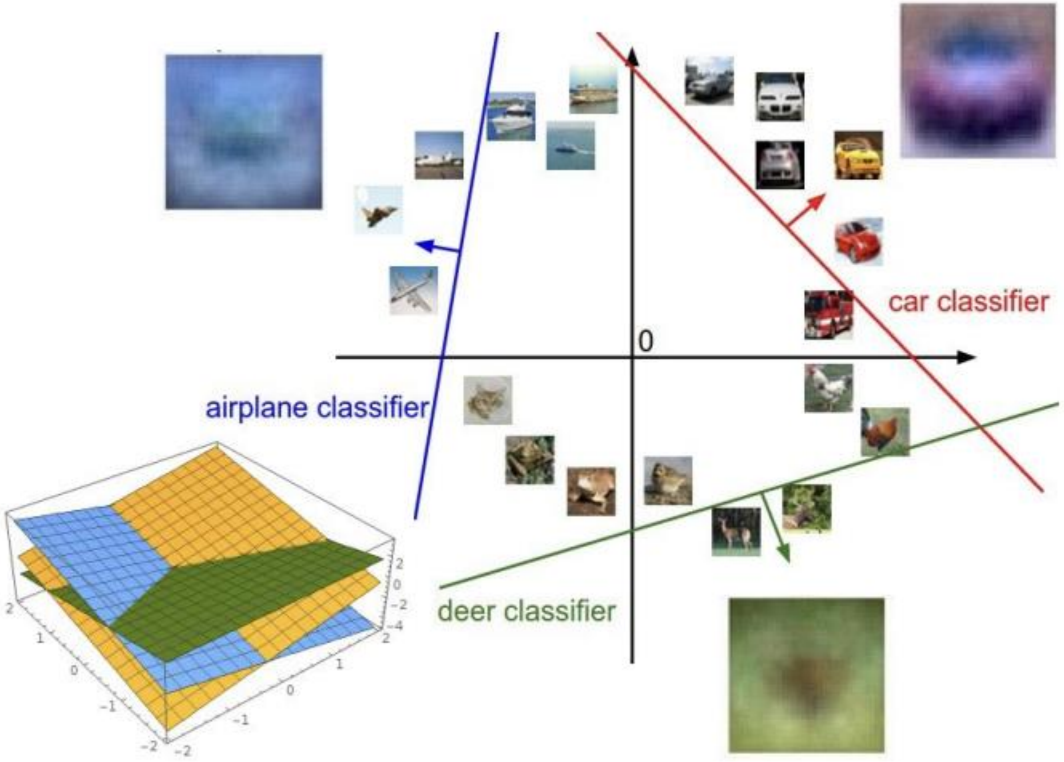


Image Classification: Linear Classification



$$f(x,W) = Wx + b$$

Array of **32x32x3** numbers
(3072 numbers total)

Image Classification: Linear Classification

Example Scores for 3 images



airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

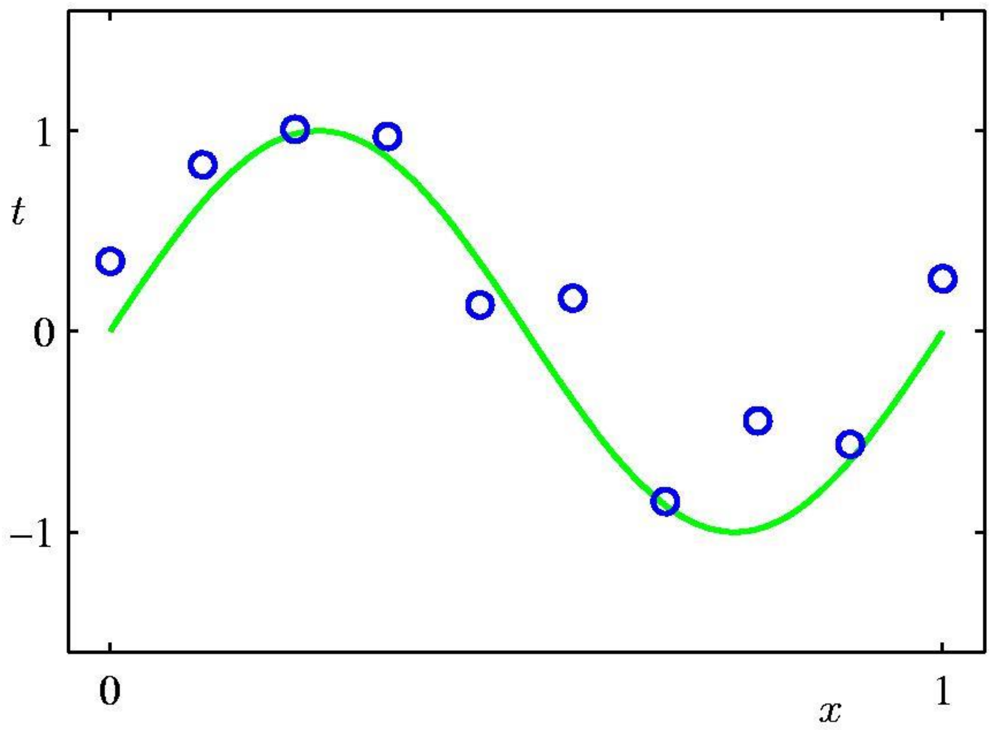
Linear Regression: Polynomial Curve Fitting

Green Curve: function used to generate data $\sin(2\pi x)$
Blue Points: Dataset generated via the green curve with some noise

Goal: Predict a function that fits the blue points well and represents the green curve as closely as possible, assuming we do not know the green curve.

A Polynomial function is common to start with:

$$y(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

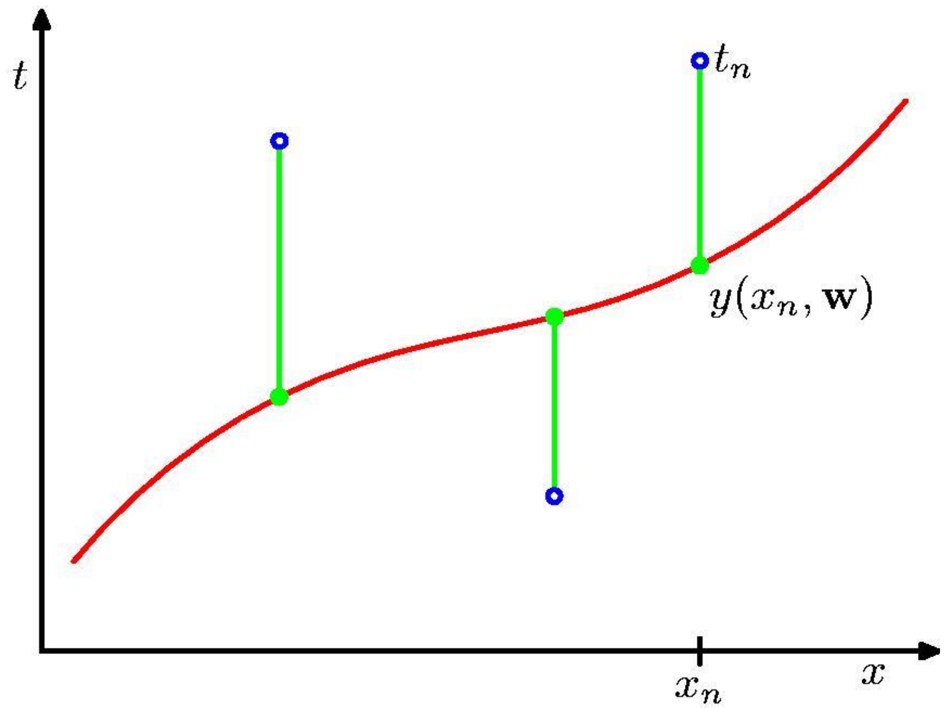


Linear Regression: Polynomial Curve Fitting

We measure and error, the difference between the data value (blue point) and the prediction, y

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

Our objective is to find the weights, ***w***, that minimises the error.



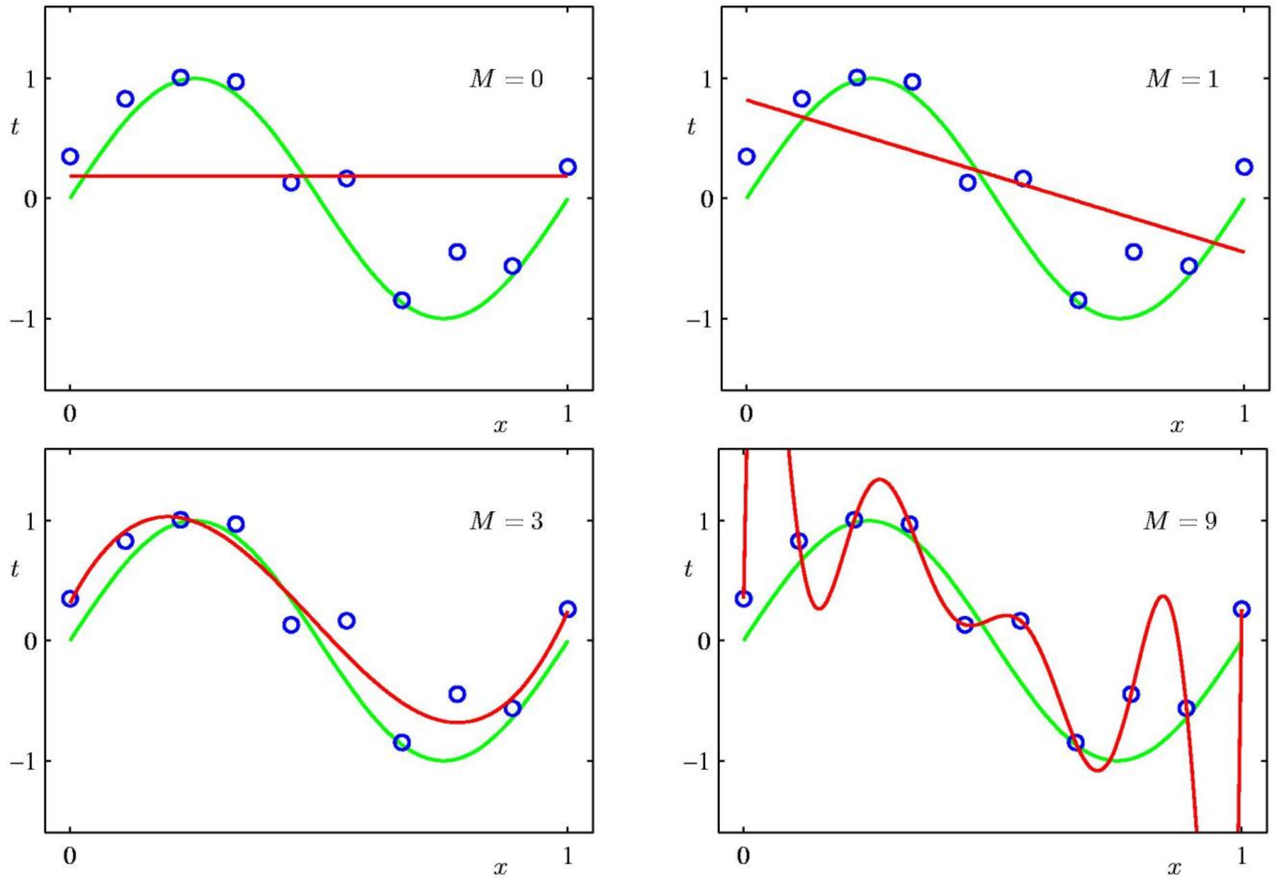
Linear Regression: Polynomial Curve Fitting

The results differ by the value of M chosen, which is the order of the polynomial

Which one seems to fit best?

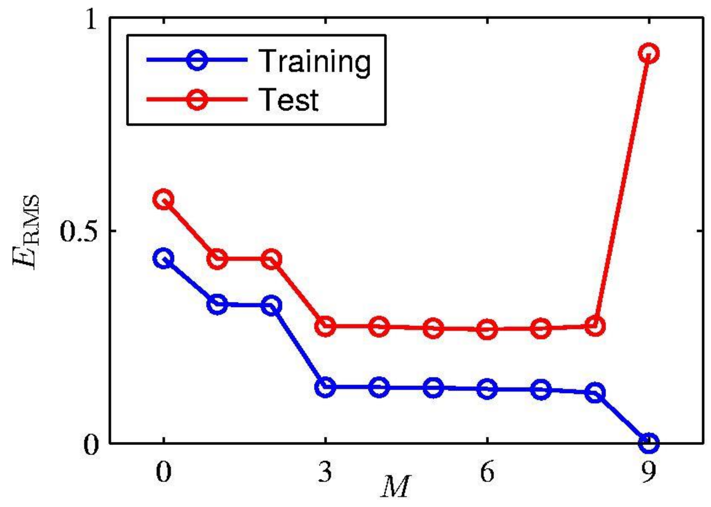
M = 0 & M = 1 is known as **underfitting**. It hardly represents the pattern of data.

M = 9 is known as **overfitting**. The error is 0 but cannot predict the value outside the training set.



Linear Regression: Polynomial Curve Fitting

Overfitting occurs when the order of the polynomial high compared to the number of training data.

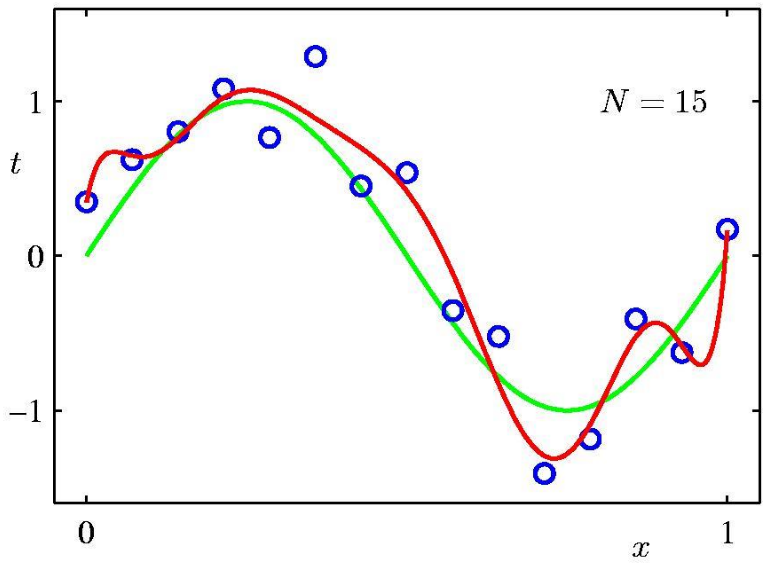


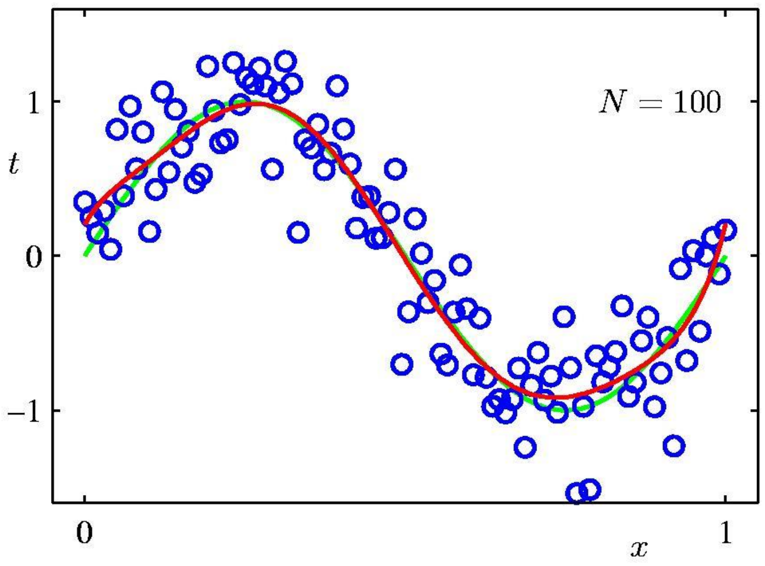
	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

For higher order polynomials, the weights seem to have large absolute values.

Linear Regression: Polynomial Curve Fitting

Solution to Overfitting 1: Increase the number of training dataset





Linear Regression: Polynomial Curve Fitting

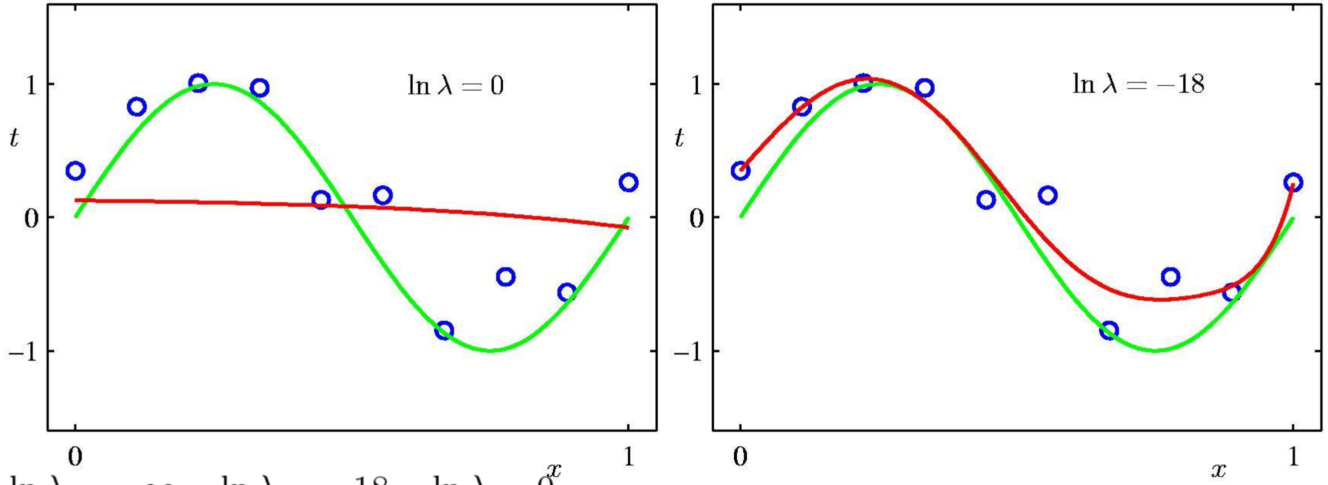
Solution to Overfitting 2: Penalize for large coefficient values

Rewrite the error function including the **regularization term**

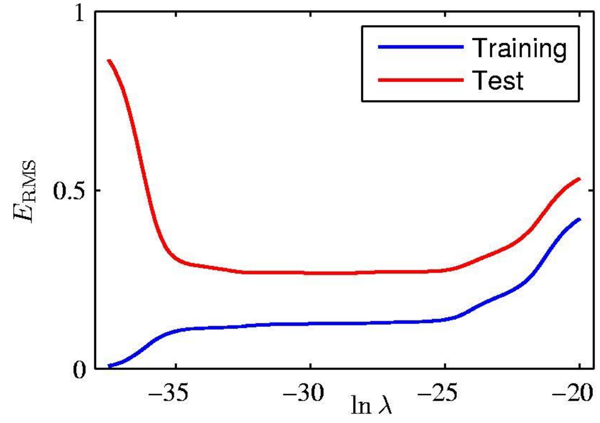
This is called the **ridge regression**

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

$\lambda = 0$ ($\ln(\lambda) = -\infty$)corresponds to no regularization.



	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01



Linear Regression: Polynomial Curve Fitting

How do we find the optimal weights?

$$y(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

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

$$\boldsymbol{w} = argmin_{\boldsymbol{w}} \sum_{n=1}^{N} \{y(x_n, \boldsymbol{w}) - t_n\}^2 = argmin_{\boldsymbol{w}} (\boldsymbol{x}\boldsymbol{w} - \boldsymbol{t})^T (\boldsymbol{x}\boldsymbol{w} - \boldsymbol{t}) \Rightarrow \boxed{\boldsymbol{w} = (\boldsymbol{x}^T \boldsymbol{x})^{-1} \boldsymbol{x}^T \boldsymbol{y}}$$

The complexity of this computation is $O(n^3)$

As the size of the dataset increase, weights become extremely difficult to find.
We must seek more convenient methods to find the weights

Towards Deep Learning

Can Linear Classifier classify all the images? How about non-linear relationships?

How can we find appropriate weights not by heavy computation?

Deep Learning methods remedy (partially in some sense) these shortcomings.

Next Lecture

Perceptron and Introduction to Deep Learning