ENGN6528-PMS-DL

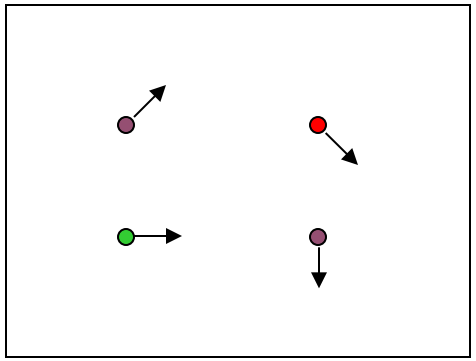
Announcements

- CLab-3 Due on May 23rd, 23:59, 2021
- CLAB-2 Marks will be released by this weekend.
- Release practice questions for your final exam soon.
- Tutorial video on CLAB-3 (same as ours) from ENGN4528 on wattle.

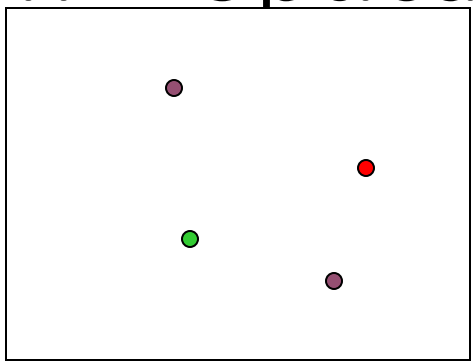
Outline

- Review: Optical Flow, Shape from Shading
- Photometric Stereo
- Introduction to Deep Learning

Review: Problem definition - optical flow



$H(x, y)$



$I(x, y)$

- How to estimate pixel motion from image H to image I?
  - Solve pixel correspondence problem
    - given a pixel in H, look for nearby pixels of the same color in I

Key assumptions

- **color constancy:** a point in H looks the same in I
  - For grayscale images, this is **brightness constancy**
- **small motion**: points do not move very far

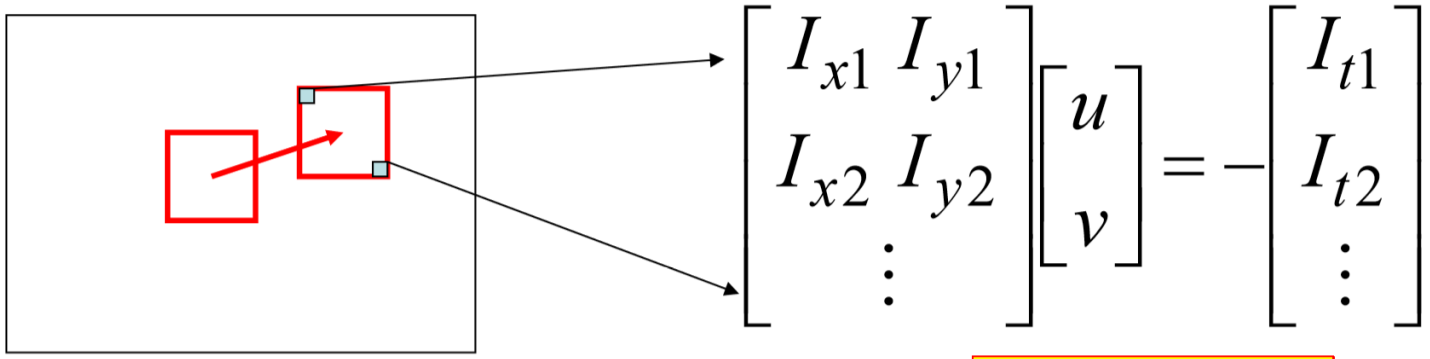
This is called the **optical flow** problem

Credit: K. Grauman

Local Patch Constant Motion: Lucas–Kanade optical flow algorithm

$$I_x u + I_y v = -I_t \quad \Rightarrow \quad \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -I_t$$

Assume constant motion (u,v) in small neighborhood



$$\begin{bmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_{t1} \\ I_{t2} \\ \vdots \end{bmatrix}$$

$$A\vec{u} = b$$

Lucas–Kanade Optical Flow Algorithm

Goal: Minimize $\left\| A\vec{\mathrm{u}} - b \right\|^2$

Method: Least-Squares

$$A\vec{\mathrm{u}} = b$$

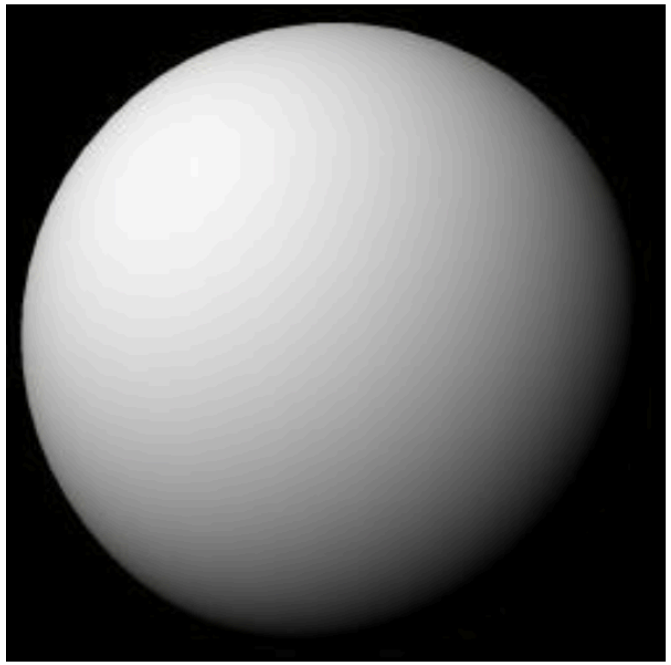
$$\Downarrow$$

$$\underbrace{A^T A}_{2\text{x}2}\ \underbrace{\vec{\mathrm{u}}}_{2\text{x}1} = \underbrace{A^T b}_{2\text{x}1}$$

$$\Downarrow$$

$$\vec{\mathrm{u}} = \left(A^T A\right)^{-1} A^T b$$

Review- ’Shape from Shading’



$$I = k_d \mathbf{N} \cdot \mathbf{L}$$

Assume k_d is 1 for now.

What can we measure from one image?

- $\cos^{-1}(I)$ is the angle between N and L

Solve ’Shape from Shading’

- We have

$$R(p,q) = \frac{1 + p_s p + q_s q}{\sqrt{1 + p^2 + q^2}\sqrt{1 + {p_s}^2 + {q_s}^2}}$$

- Discretize: end up with one nonlinear PDE equation per pixel
- For N pixels, → *N* equations in 2*N* unknowns…

Solve ‘Shape from Shading’

- Solution: add smoothness constraint; add initial boundary conditions.
- Form an energy minimization problem:
- Given observed E(x,y), find {(p,q)} that minimize energy

$$\mathcal{E} = \int \left(\left(I(x,y) - R(p,q) \right)^2 + \lambda \left(p_x^{\,2} + p_y^{\,2} + q_x^{\,2} + q_y^{\,2} \right) \right) dx\, dy$$

- Regularization: minimize combination of difference between observed image and ‘rendering by R(p,q)’, surface curvature

Photometric stereo

Two-view SFM

Key Idea: use feature motion to understand shape

Photometric Stereo

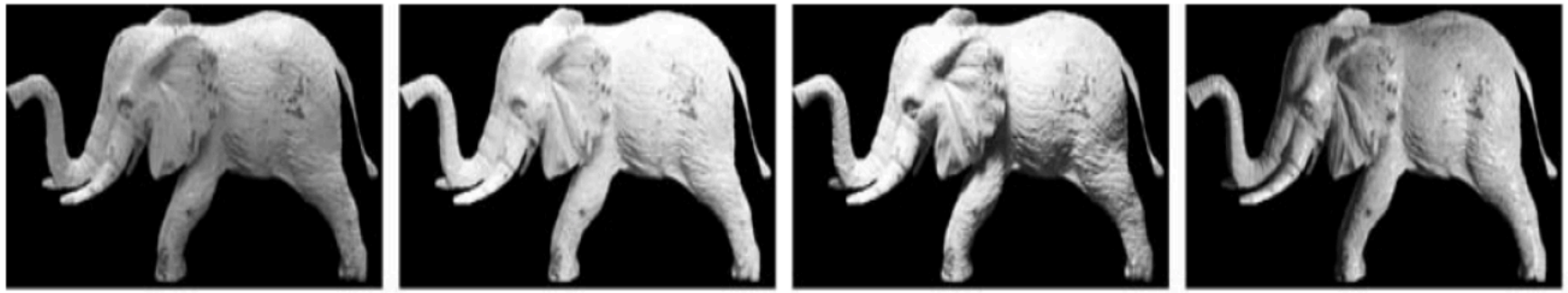
Key Idea: use pixel brightness to understand shape

Photometric Stereo

Key Idea: use pixel brightness to understand shape

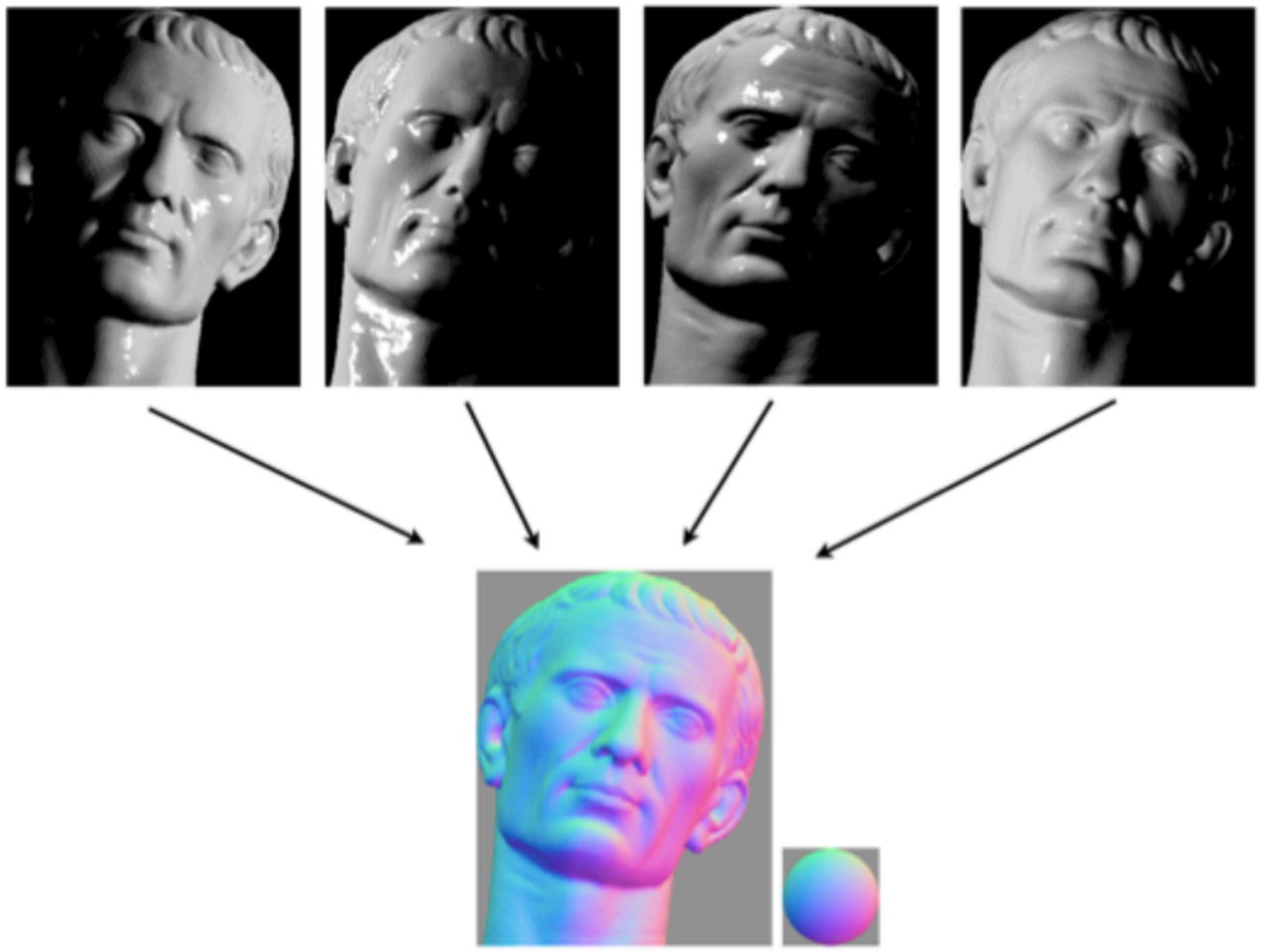
Photometric Stereo

- Use multiple images under different light sources
  - Photometric stereo

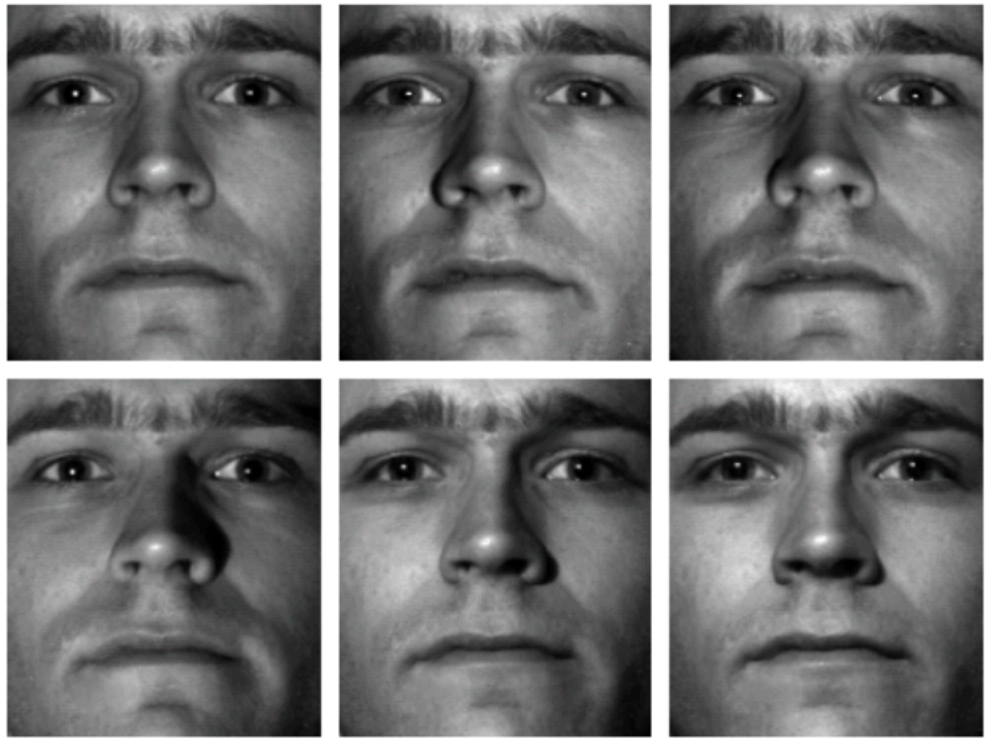


Images from
R. Basri et al. "Photometric Stereo with General, Unknown Lighting", IJCV

Photometric Stereo

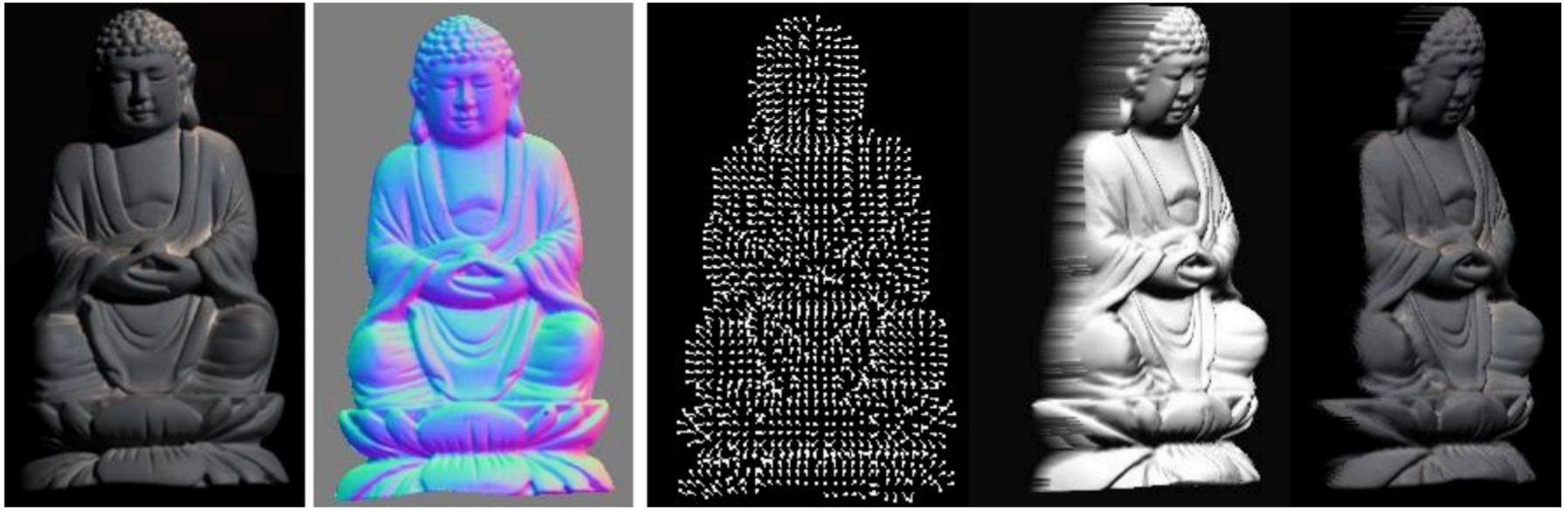


Photometric Stereo



from Athos Georghiades

Photometric Stereo



Input
(1 of 12)

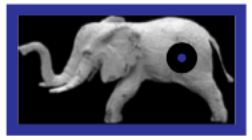
Normals (RGB
colormap)

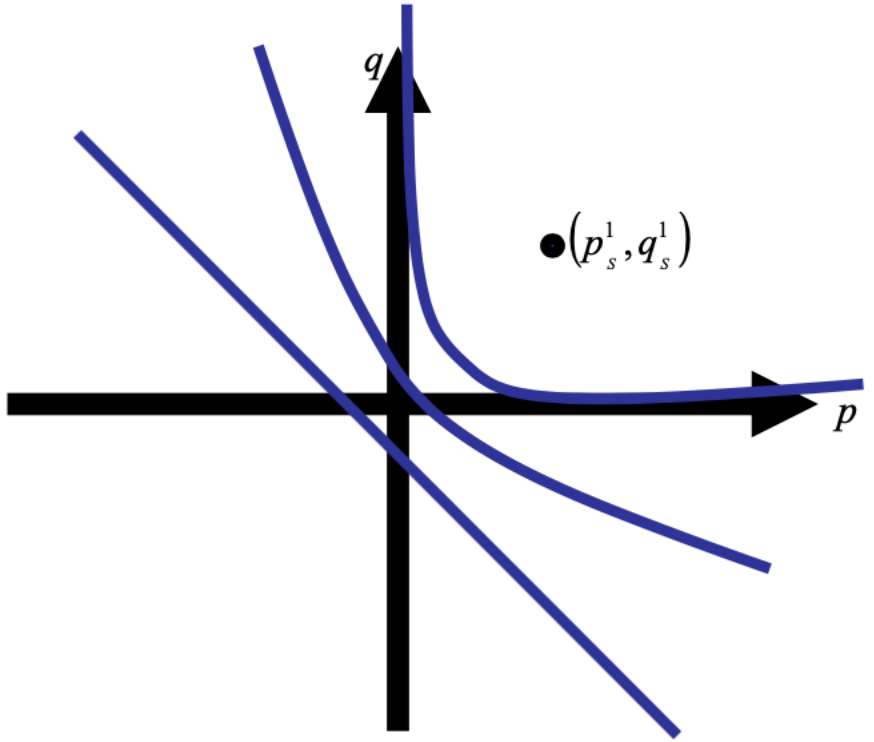
Normals (vectors)

Shaded 3D
rendering

Textured 3D
rendering

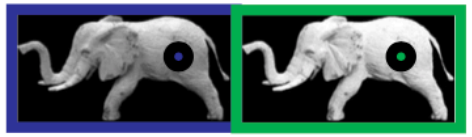
Derivation: Photometric Stereo





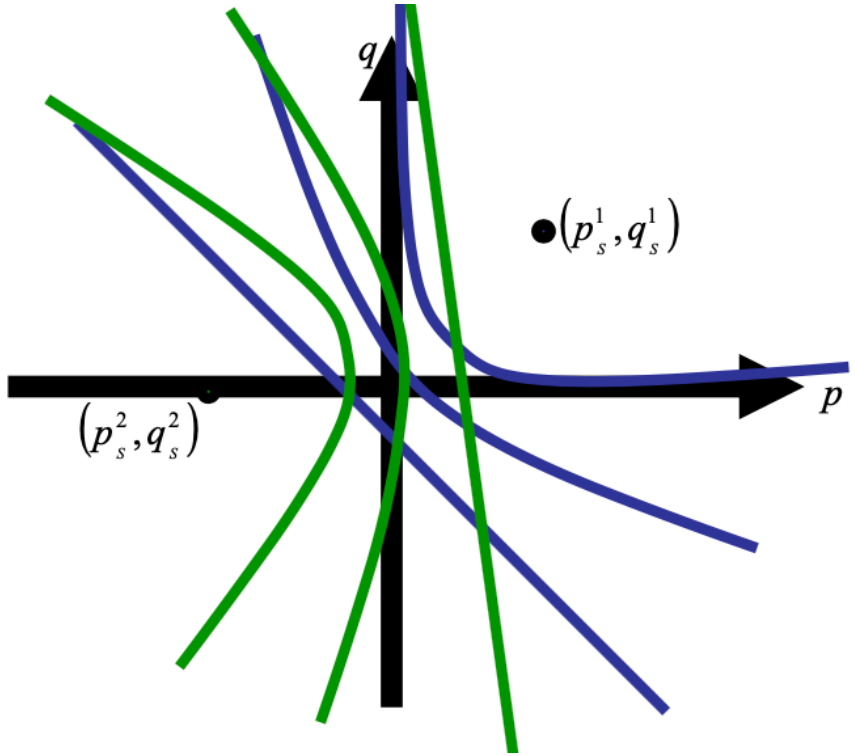
$$I = k_d \mathbf{N} \cdot \mathbf{L}$$

Derivation: Photometric Stereo



p, q: surface normal 表面法线，表示方向
ps, qs: 光源方向

$$R(p,q) = \frac{1 + p_s p + q_s q}{\sqrt{1 + p^2 + q^2}\sqrt{1 + {p_s}^2 + {q_s}^2}}$$

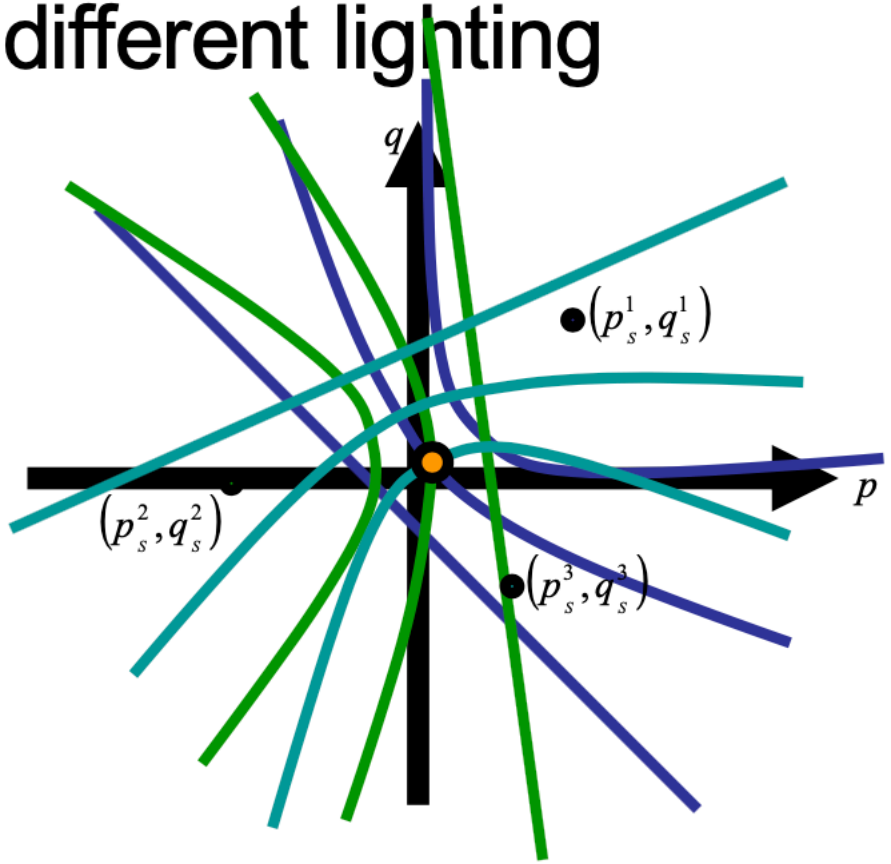


每一条曲线代表给定的(ps, qs)下有相同的亮度

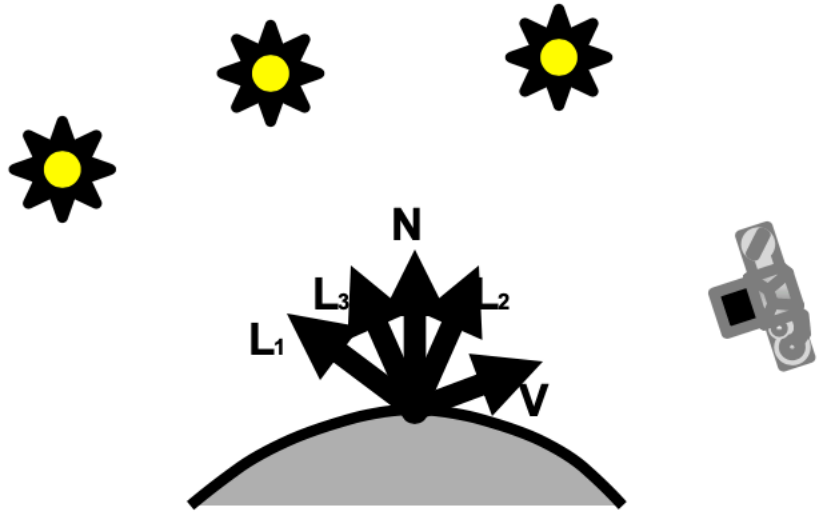
Derivation: Photometric Stereo

- Take several pictures of same object under same viewpoint with different lighting

通过3张照片的交点确定平面的方向



Multiple Images: Algebraic derivation



$$I_1 = k_d \mathbf{N} \cdot \mathbf{L}_1$$
$$I_2 = k_d \mathbf{N} \cdot \mathbf{L}_2$$
$$I_3 = k_d \mathbf{N} \cdot \mathbf{L}_3$$

Write this as a matrix equation:

$$\begin{bmatrix} I_1 & I_2 & I_3 \end{bmatrix} = k_d \mathbf{N}^T \begin{bmatrix} \mathbf{L}_1 & \mathbf{L}_2 & \mathbf{L}_3 \end{bmatrix}$$

Solving the Algebraic Equation

$$\underbrace{\begin{bmatrix} I_1 & I_2 & I_3 \end{bmatrix}}_{\substack{\mathbf{I} \\ 1\times 3}} = \underbrace{k_d \mathbf{N}^T}_{\substack{\mathbf{G} \\ 1\times 3}} \underbrace{\begin{bmatrix} \mathbf{L_1} & \mathbf{L_2} & \mathbf{L_3} \end{bmatrix}}_{\substack{\mathcal{L} \\ 3\times 3}}$$

$$\mathbf{G} = \mathbf{I}\mathbf{L}^{-1}$$

$$k_d = \|\mathbf{G}\|$$

$$\mathbf{N} = \frac{1}{k_d}\mathbf{G}$$

Solving the Algebraic Equation

$$\underbrace{\begin{bmatrix} I_1 & I_2 & I_3 \end{bmatrix}}_{\mathbf{I}} = \underbrace{k_d \mathbf{N}^T}_{\mathbf{G}} \underbrace{\begin{bmatrix} \mathbf{L_1} & \mathbf{L_2} & \mathbf{L_3} \end{bmatrix}}_{\mathcal{L}}$$

$\mathbf{I}$: 1×3 $\quad$ $\mathbf{G}$: 1×3 $\quad$ $\mathcal{L}$: 3×3

$$\mathbf{G} = \mathbf{I}\mathbf{L}^{-1}$$

- When is L nonsingular (invertible)?
  - >= 3 light directions are linearly independent, or:
  - All light direction vectors cannot lie in a plane.
- What if we have more than one pixel?
  - Stack them all into one big system.

More than Three Lights

$$\begin{bmatrix} I_1 & \dots & I_n \end{bmatrix} = k_d \mathbf{N}^T \begin{bmatrix} \mathbf{L_1} & \dots & \mathbf{L_n} \end{bmatrix}$$

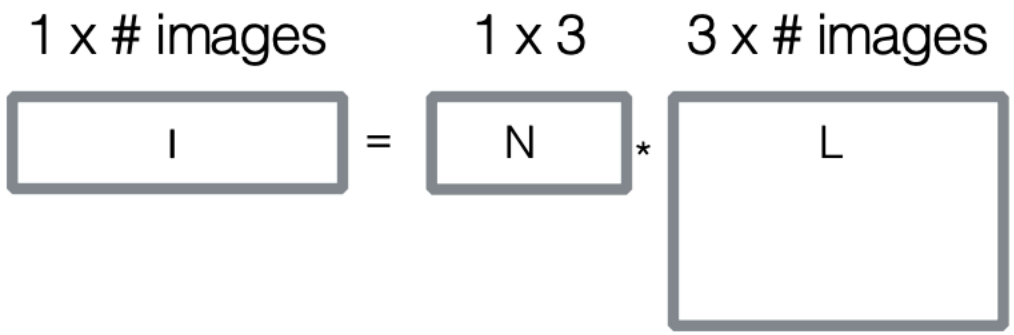
- Solve using least squares (normal equations):

$$\begin{aligned} \mathbf{I} &= \mathbf{GL} \\ \mathbf{IL}^\mathbf{T} &= \mathbf{GLL}^\mathbf{T} \\ \mathbf{G} &= (\mathbf{IL}^\mathbf{T})(\mathbf{LL}^\mathbf{T})^{-1} \end{aligned}$$

- Equivalently use SVD
- Given G, solve for N and k_d as before.

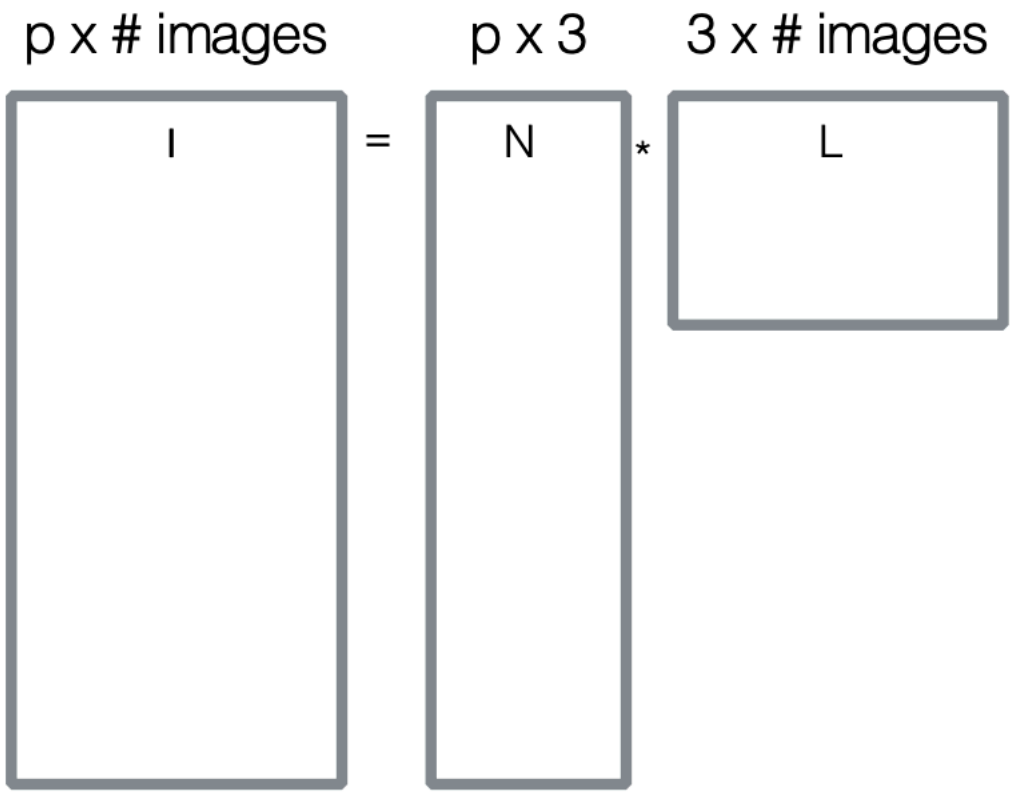
More than one pixel

One pixel case:



More than one pixel

Stack all pixels into one system:

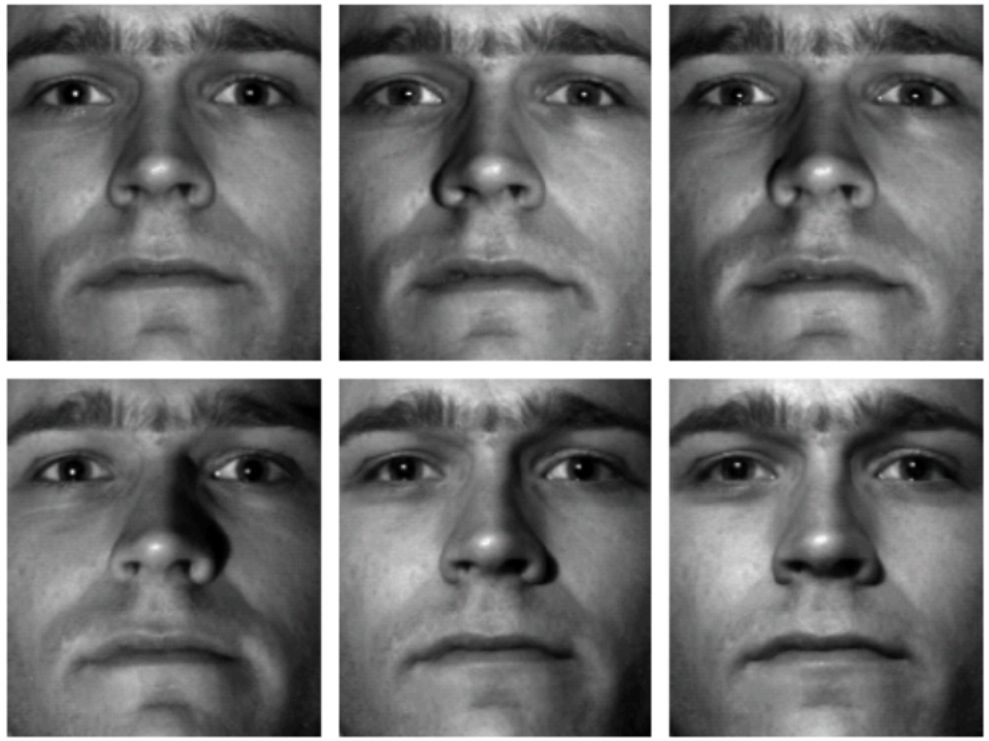


Solve as before.

Solve depth

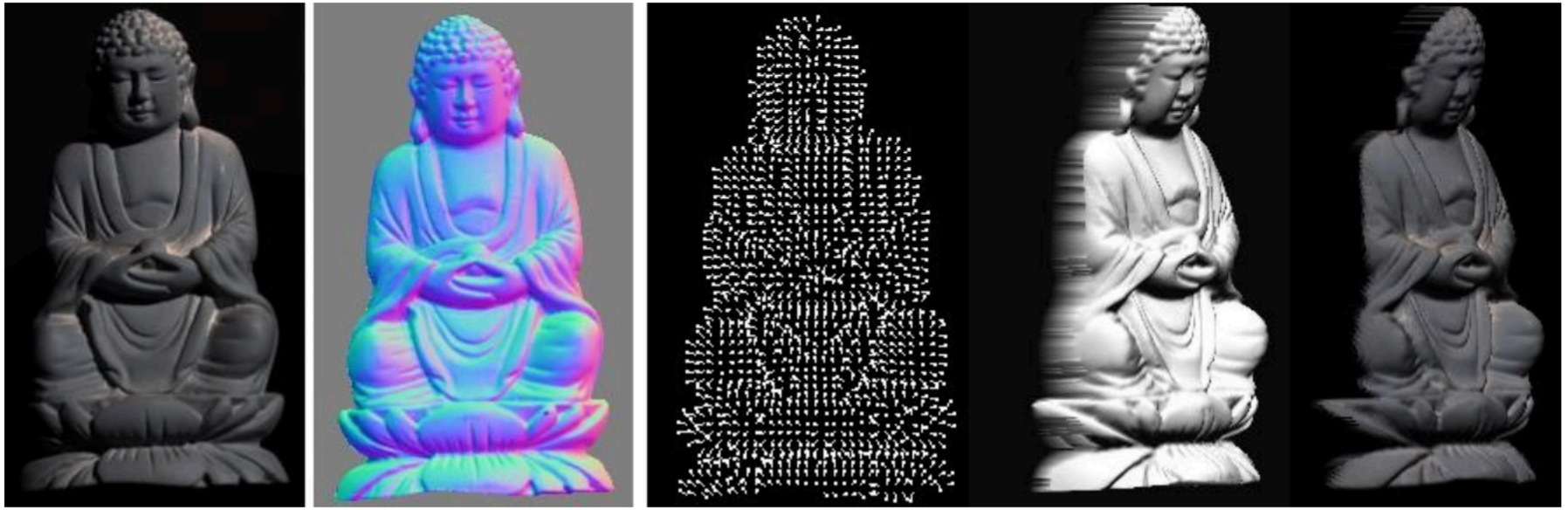
- Given the normal map estimated from more than one images, we could derive the depth map based on integration of normal in the image.

Photometric Stereo



from Athos Georghiades

Photometric Stereo



Input
(1 of 12)

Normals (RGB
colormap)

Normals (vectors)

Shaded 3D
rendering

Textured 3D
rendering

Youtube Demo

- https://www.youtube.com/watch?v=tsLTq3MuXNI

Summary

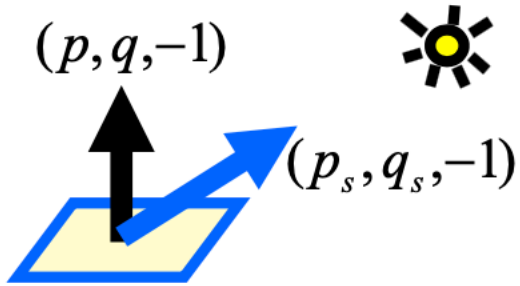
Image formation

Lambertian reflectance

$$E(\mathbf{x}) = \rho L_i(\mathbf{x}, \theta_i, \phi_i) \cos\theta_i = \rho(\mathbf{n} \bullet \mathbf{l}_i)$$

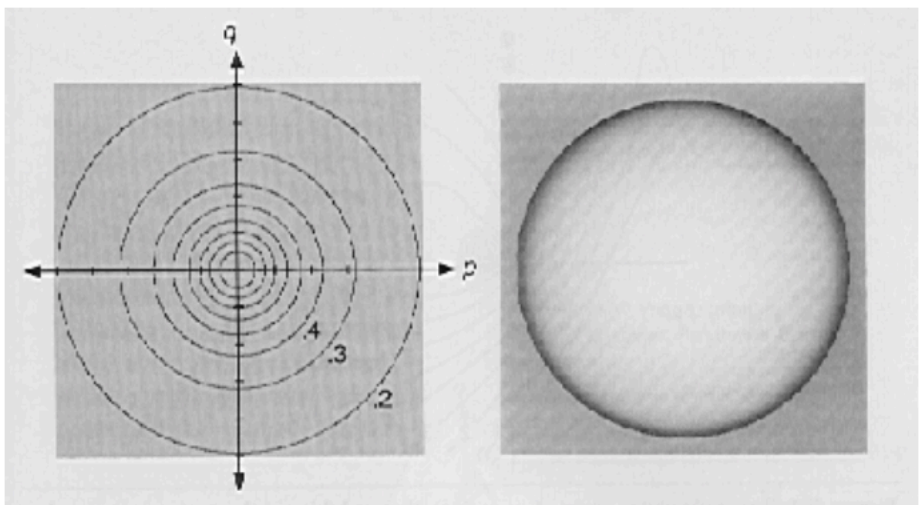
Fixing light, albedo, we can express reflectance only as function of normal.

Lambertian reflectance map

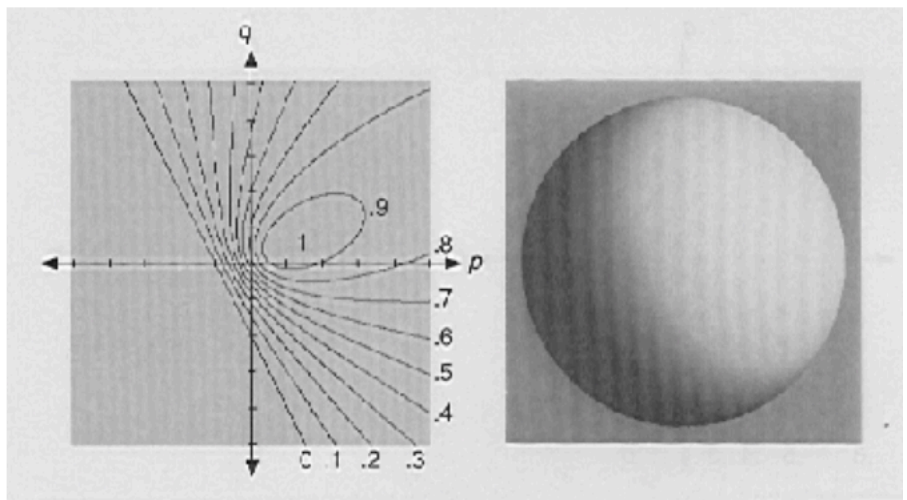


$$E(p,q) = L\rho \frac{1 + pp_s + qq_s}{\sqrt{1+p^2+q^2}\sqrt{1+p_s^{\,2}+q_s^{\,2}}}$$

Local surface orientation that produces equivalent intensities are quadratic conic sections contours in gradient space



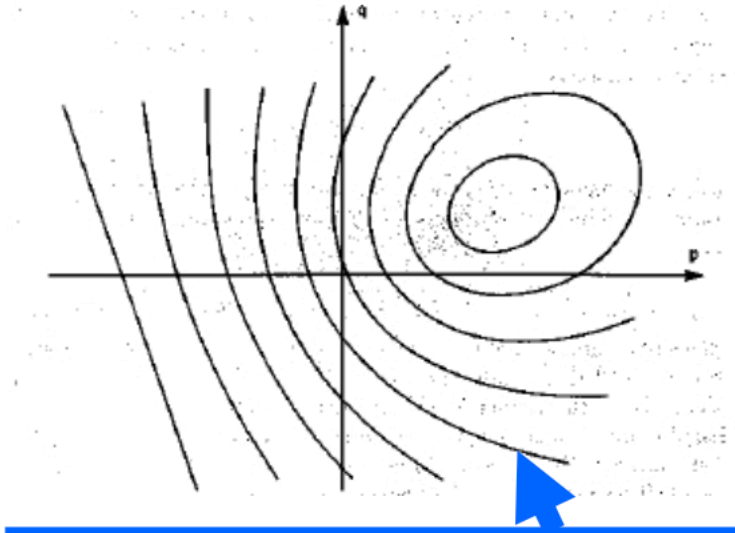
$p_s=0, q_s=0$



$p_s=-2, q_s=-1$

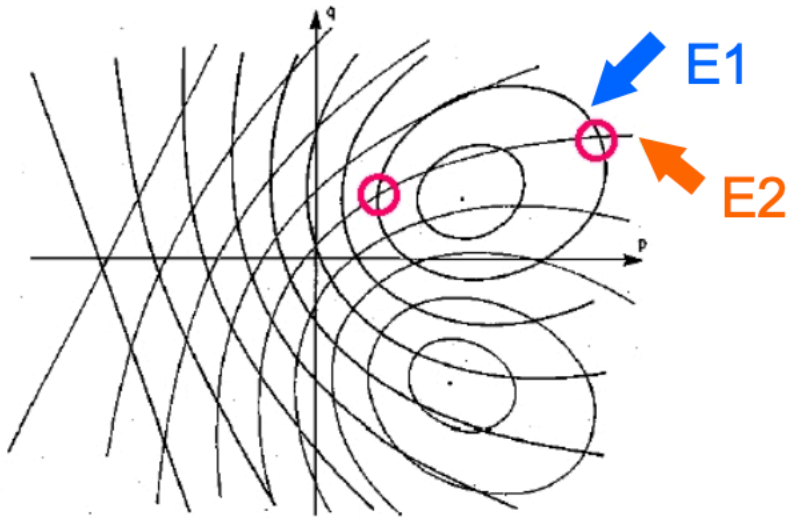
Photometric stereo

One image =one light direction



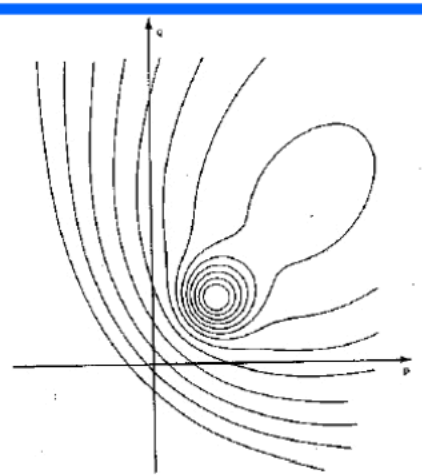
Radiance of one pixel constrains the normal to a curve

Two images = two light directions



A third image disambiguates between the two.

Normal = intersection of 3 curves



Specular reflectance

Photometric stereo

[Birkbeck]

One image, one light direction

$$I(\mathbf{x}) = B(\mathbf{x}) = \rho(\mathbf{x})\mathbf{n}(\mathbf{x}) \bullet \mathbf{l}_i$$

n images, n light directions

$$\begin{bmatrix} \mathbf{l}_1^T \\ \mathbf{l}_2^T \\ \vdots \\ \mathbf{l}_n^T \end{bmatrix} \rho(\mathbf{x})\mathbf{n}(\mathbf{x}) = \begin{bmatrix} I_1^T(\mathbf{x}) \\ I_2^T(\mathbf{x}) \\ \vdots \\ I_n^T(\mathbf{x}) \end{bmatrix}; \quad \underbrace{A\rho(\mathbf{x})\mathbf{n}(\mathbf{x})}_{\mathbf{b}(\mathbf{x})} = I(\mathbf{x})$$

Given: n>=3 images with different known light dir. (infinite light)

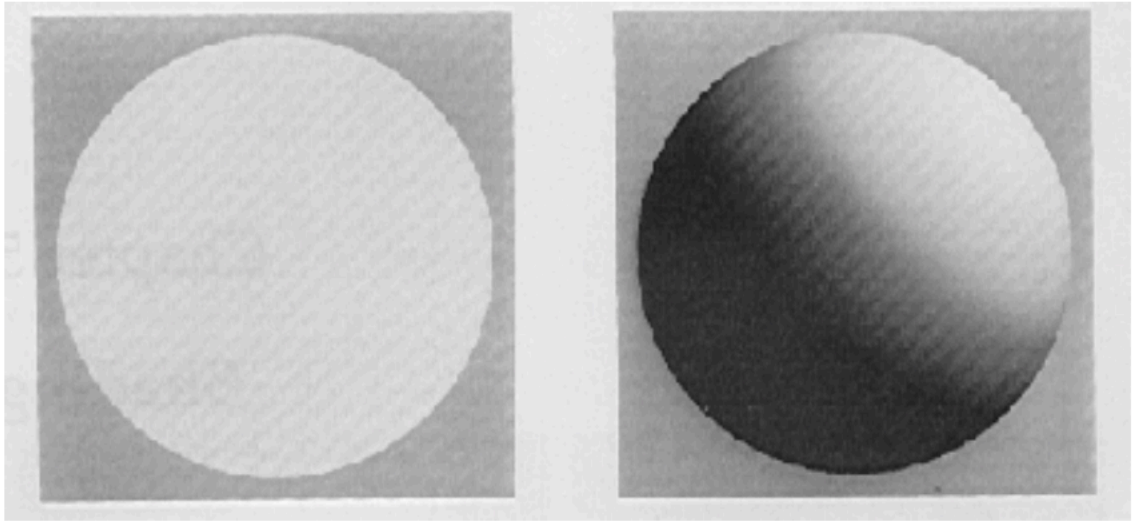
Assume: Lambertain object
orthograhic camera
ignore shadows, interreflections

Recover $\mathbf{b}(\mathbf{x}) = \rho(\mathbf{x})\mathbf{n}(\mathbf{x})$

Albedo = magnitude $|\mathbf{b}(\mathbf{x})|$

Normal = normalized $\dfrac{\mathbf{b}(\mathbf{x})}{|\mathbf{b}(\mathbf{x})|}$

Shape from Photometric measurements



Shading reveals 3D shape geometry

Shape from Shading

One image

Known light direction

Known BRDF (unit albedo)

Ill-posed : additional constraints (intagrability …)

[Horn]

Photometric Stereo

Several images, different lights

Unknown Lambertian BRDF

1. Known lights
2. Unknown lights

[Silver 80, Woodman 81]

Reconstruct normals
Integrate surface

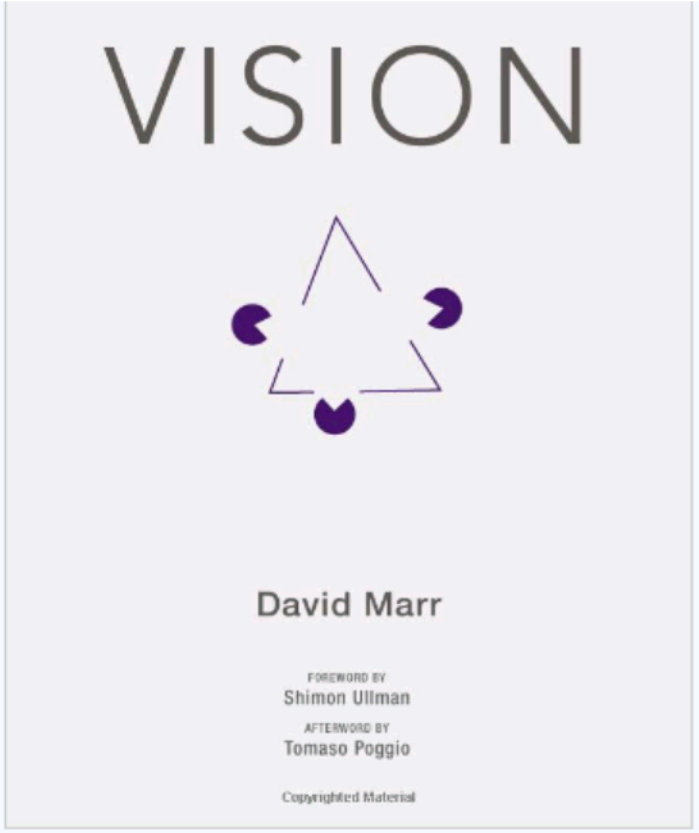
Readings

- Chapter 13.1.1 Computer Vision: Algorithms and Applications (2nd Edition)

Introduction to Deep Learning

Three levels of vision processing

- Low-level vision:
  - image processing, denoising, filtering, image restoration, low-level feature extraction.
- Mid-level vision & 3D Vision:
  - image analysis, image segmentation, contour extraction, perceptual organization, 3D information recovery.
- High-level vision:
  - visual detection, recognition, understanding, semantic labelling, activity, event detection.



Why Deep Learning?

- So far, the best framework in solving various high-level vision problems. (previously SVM, kernel method, decision tree/forest, ...)
- A significant number of online resources and open course materials about deep learning and neural networks are available on the internet.
- ANU also has a few other courses which are all about deep learning – **ENGN8536: Advanced Topics in Mechatronics**
  COMP6xxxx: Neural and Bio-inspired Computing
- In this class, we focus on the fundamentals and applications of DL **in the context of computer vision.**

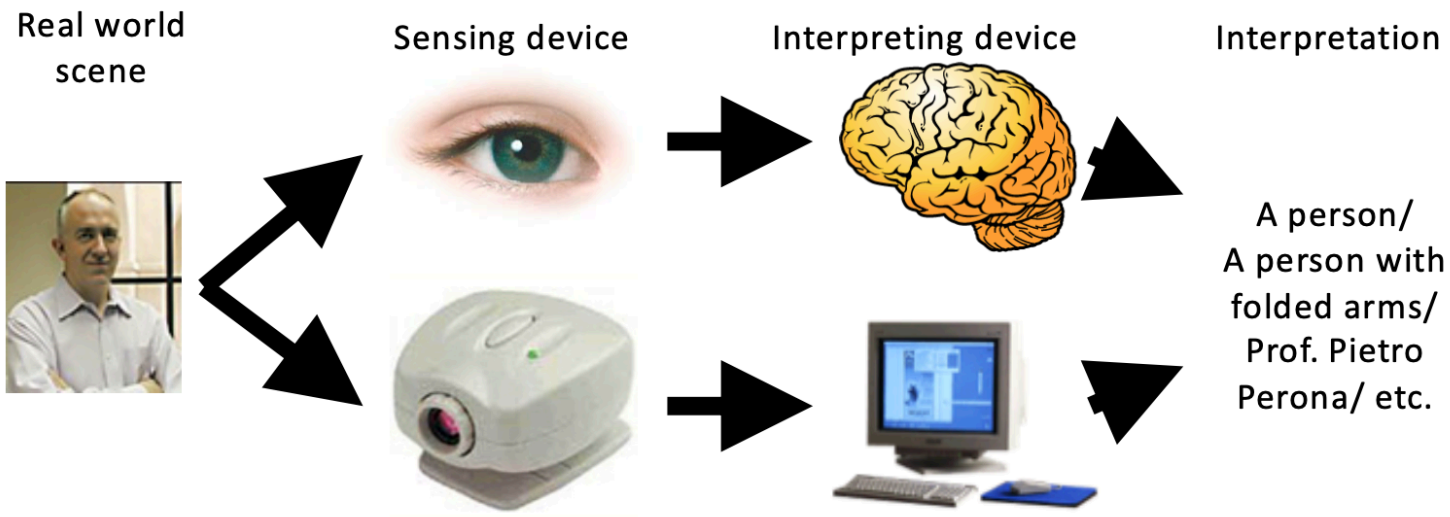
Outline

- **Fundamental** of artificial neural networks and DL**.**
  – Basic concepts of neural networks
  – Basic computation of deep neural networks, its architectures and training.

Goal of Computer Vision

- Make a computer to see and to understand images

- We know it is physically/biologically possible – we do it every day effortlessly!

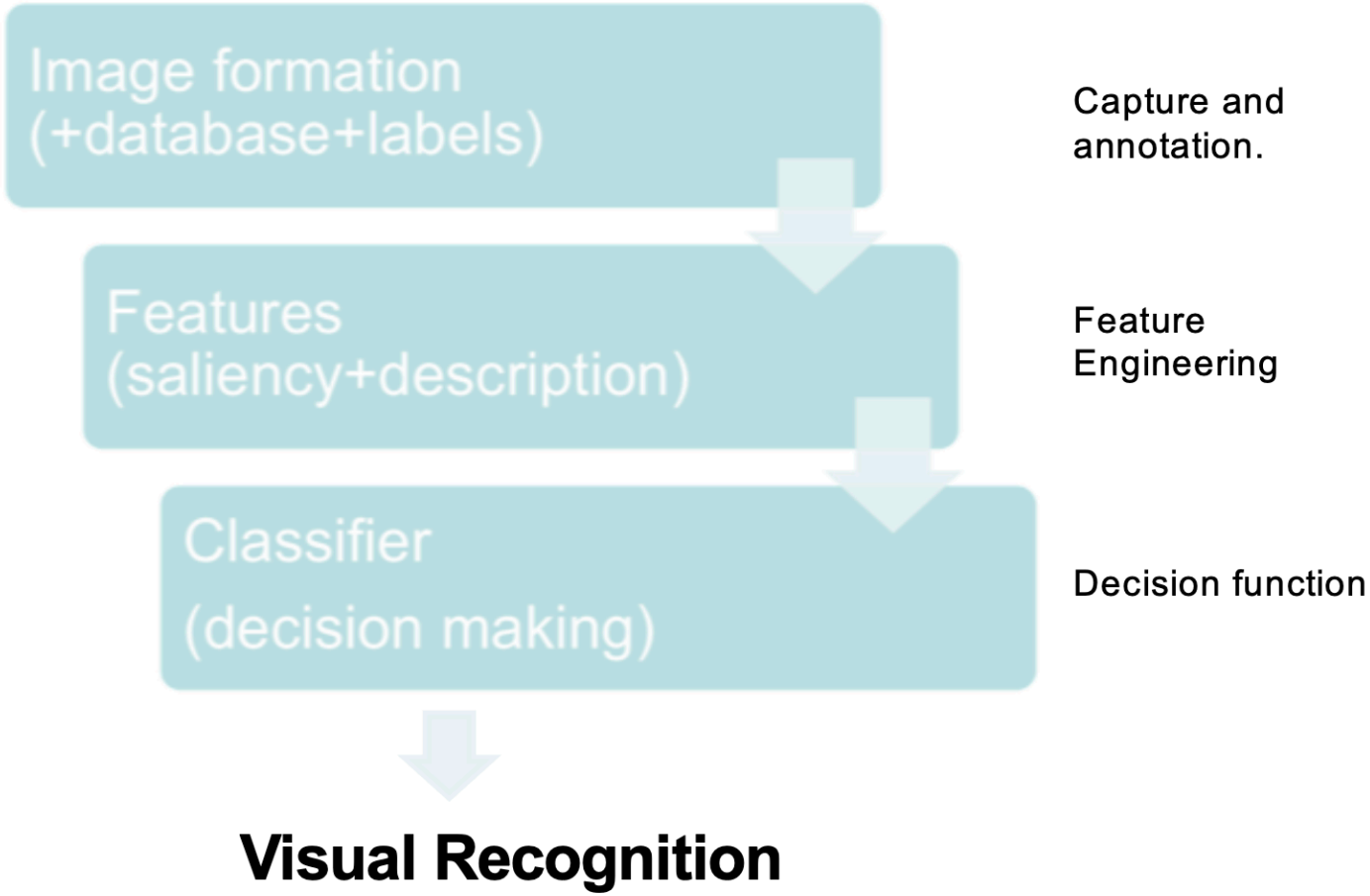


Fundamentals of ANN & DL

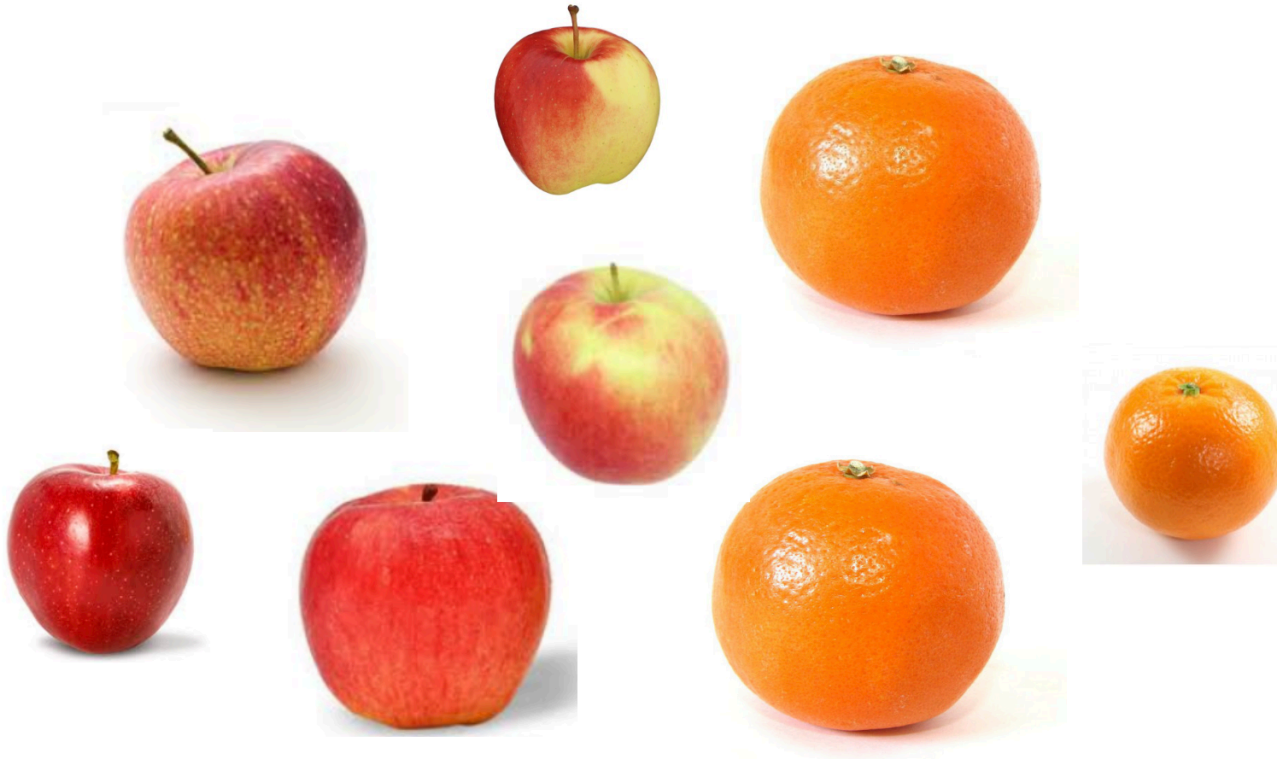
- Basic concepts for visual recognition
  - Binary classifier

- Fundamentals of ANN(**artificial neural networks**)
  - its architecture
  - Its computation
- Deep Learning

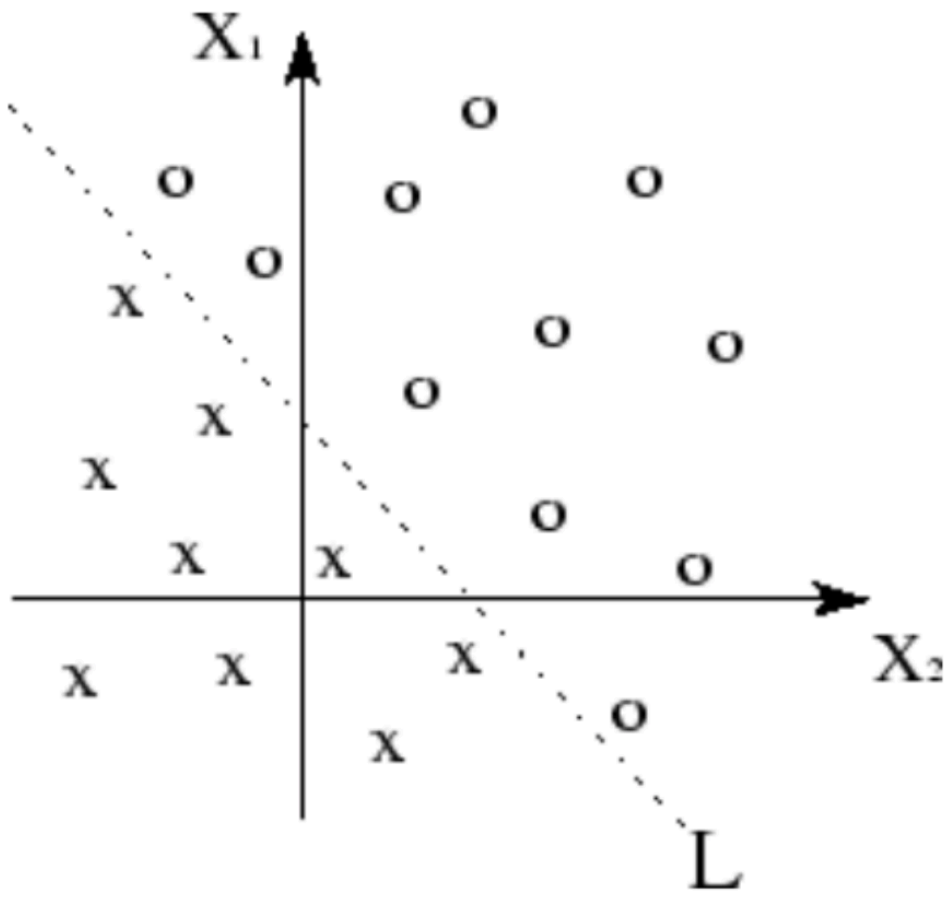
Visual Recognition Basics



Example: Apple and Orange Classification

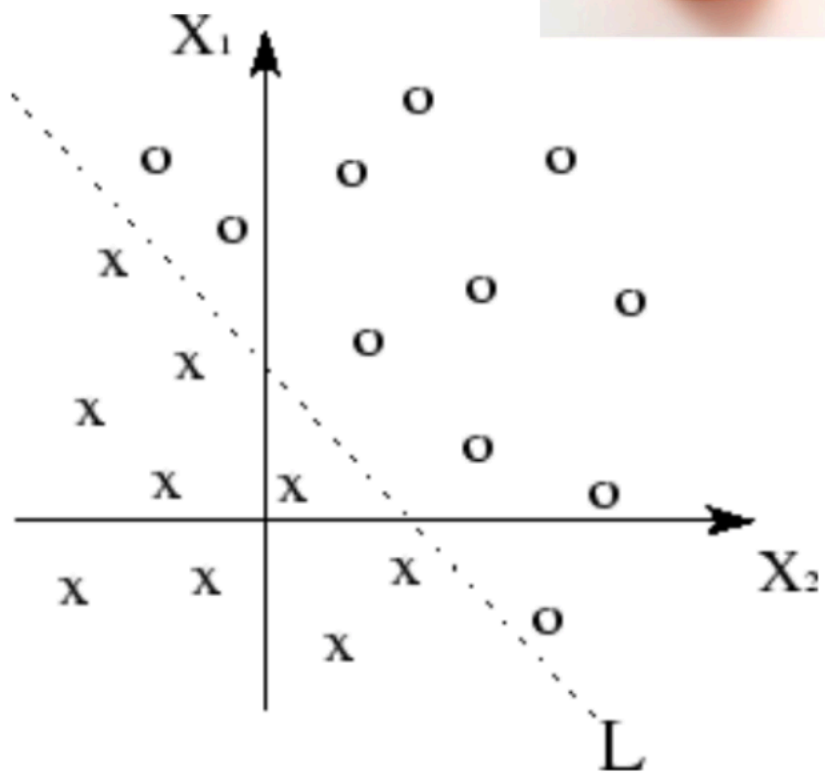


Linear separability



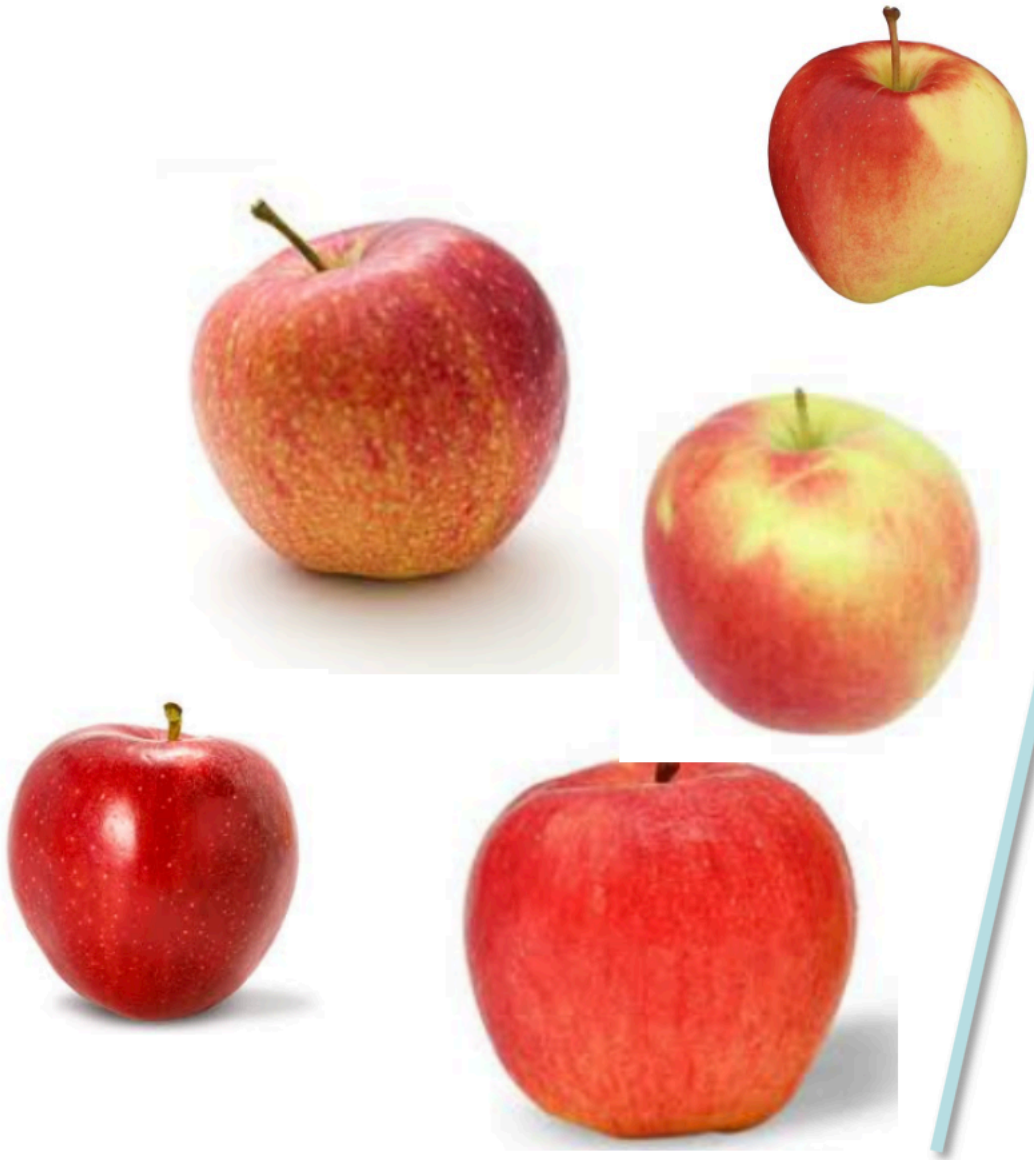
Linearly separable data
+ linear classifer = good.

Kawaguchi

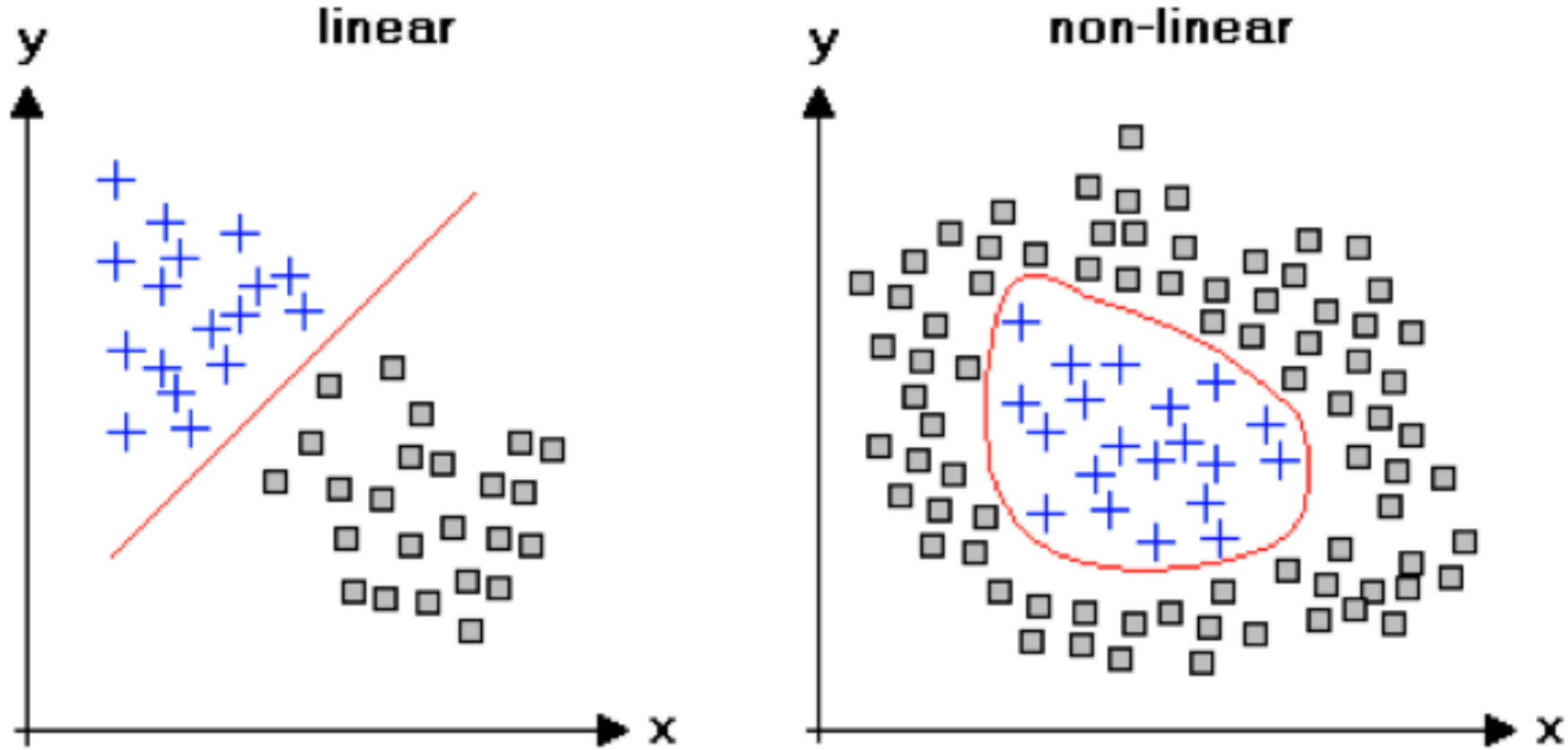


Linearly separable data
+ linear classifer = good.

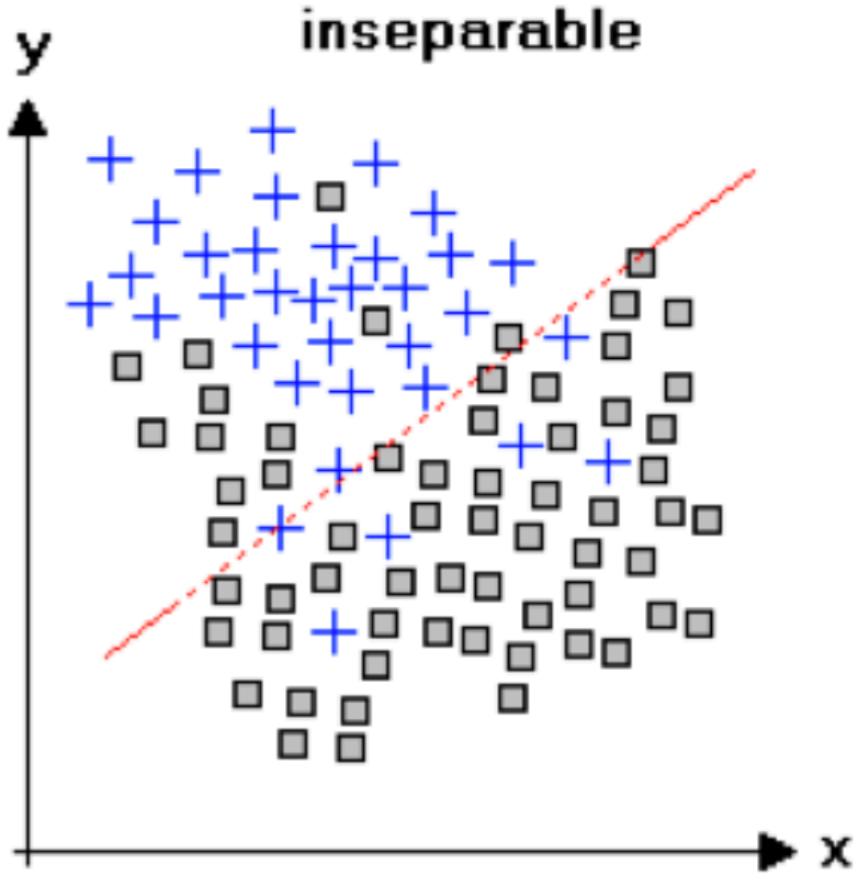
Shape



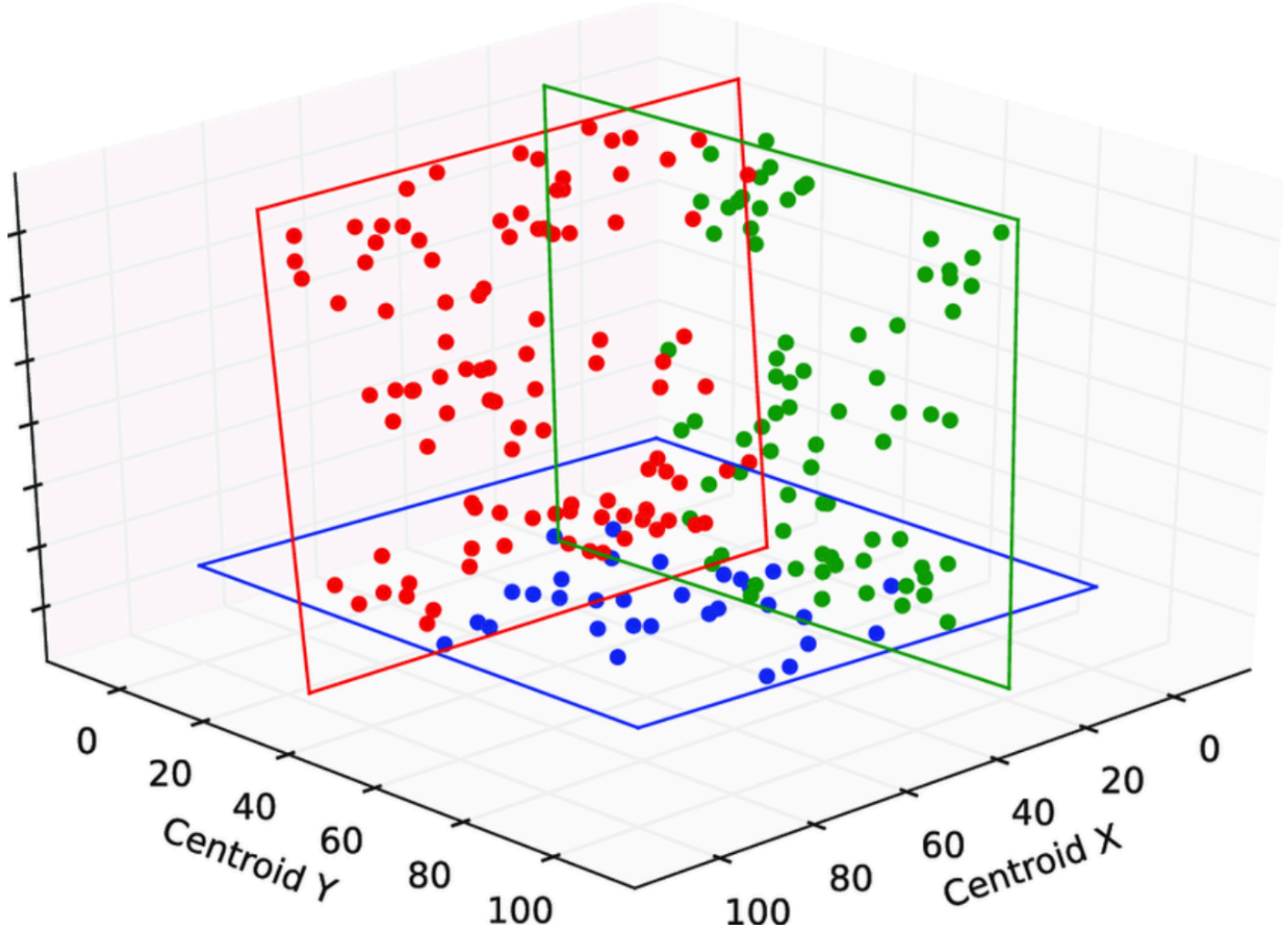
Non-linearly separable case



Inseparable case

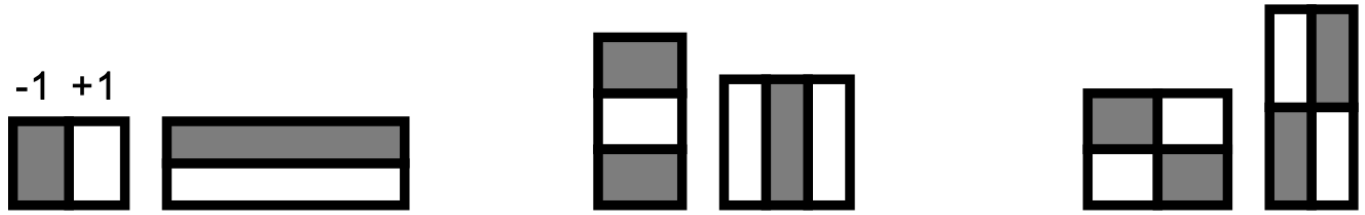


Higher dimension feature space



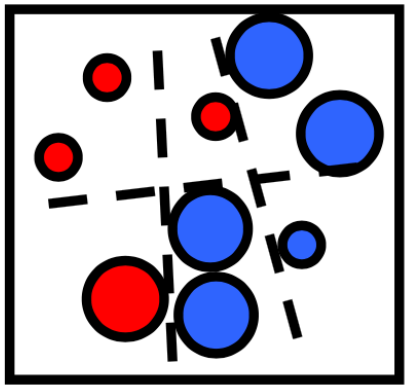
Reminder: Viola-Jones Face Detector

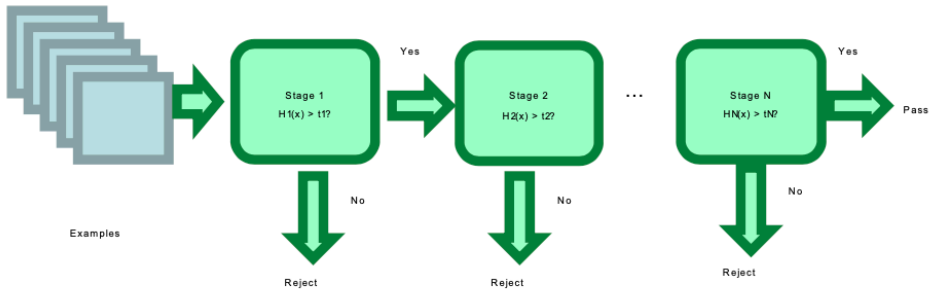
Combine *thousands* of 'weak classifiers'



Two-rectangle features Three-rectangle features Etc.

Learn how to combine in cascade with boosting

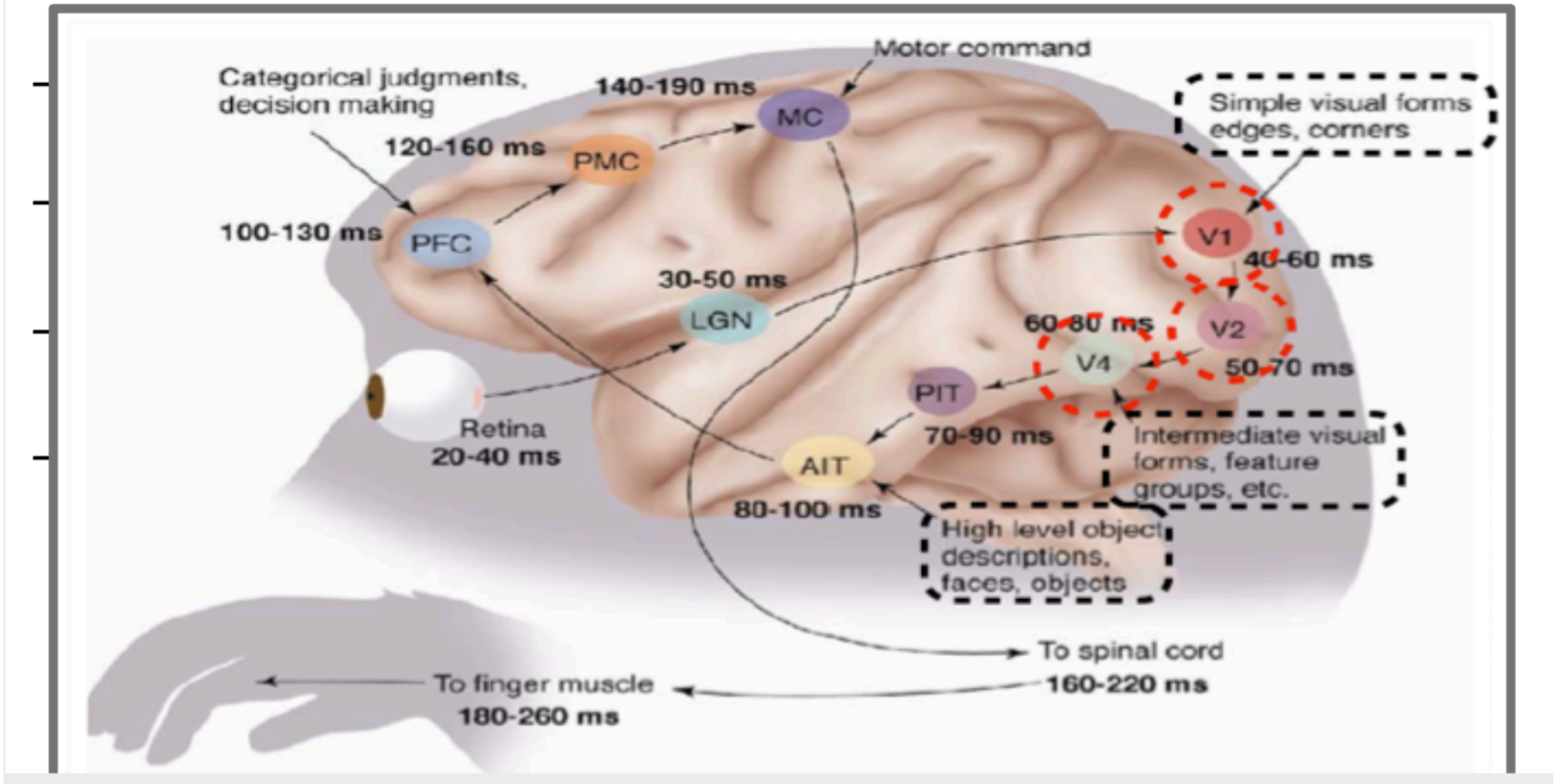




Fundamentals for Artificial Intelligence network

Biological Inspiration: Biological Neural System

It is estimated the number of neurons in the human brain (cortex) to be around 10^10 or 10^11.



Why don’t we just copy biological visual/neural system?

•People tried, but not succeeded. We don’t yet have a sufficient understanding of how our own visual system works

•100 billion neurons used in visual cortex.

•By contrast, latest CPUs have only one billion transistors.

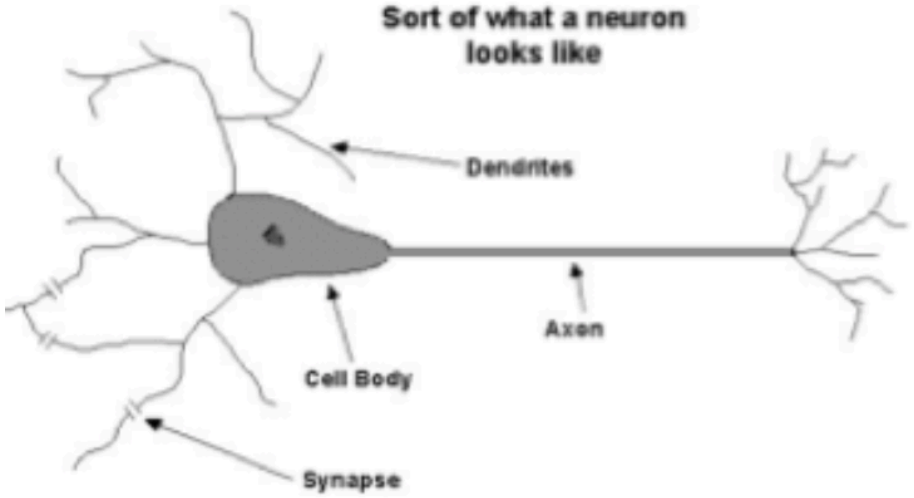
•Very different architectures:
- Brain is slow but parallel
- Computer is fast but mainly serial

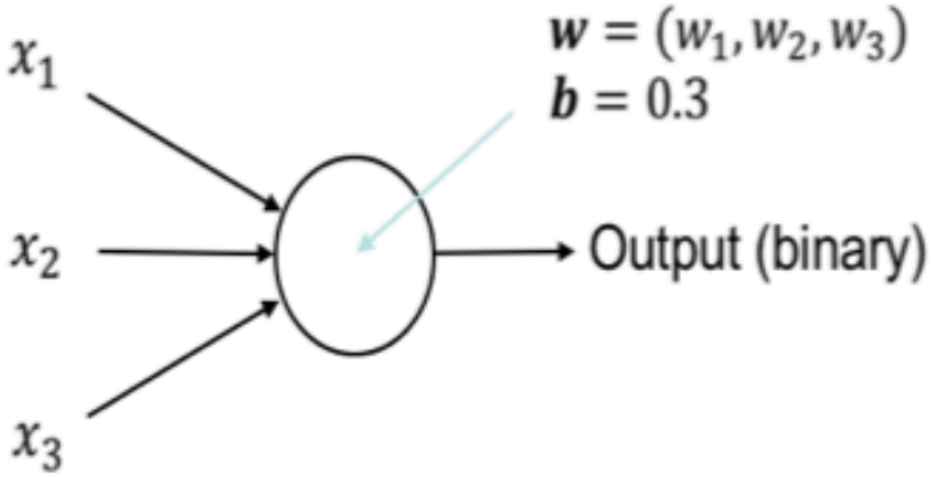
•Bird vs Airplane
- Same underlying aerodynamic principles
- Very different hardware

Biological neurons

Inspiration: Biological Neurons

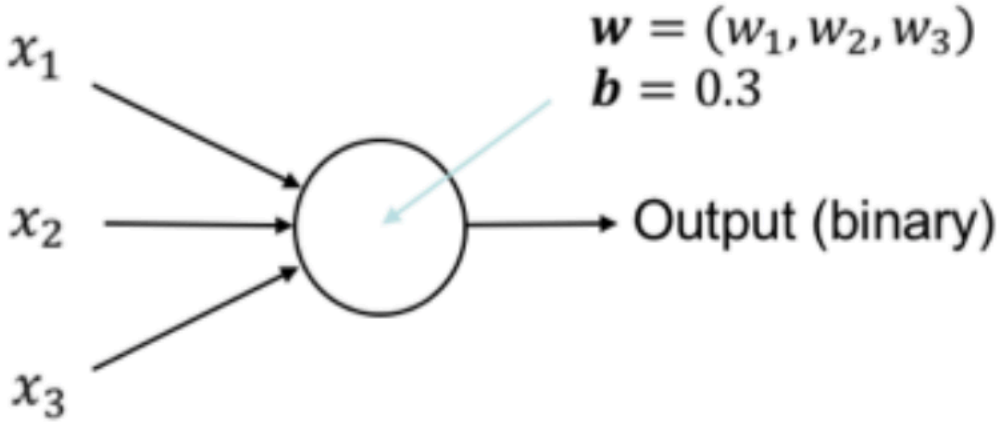
- Neurons
  - accept information from multiple inputs,
  - transmit (process and relay) information to other neurons.
- Multiply inputs by weights along edges
- Apply some **activation function** to the output of each node





Perceptron is the simplest neuron acting as a linear classifier

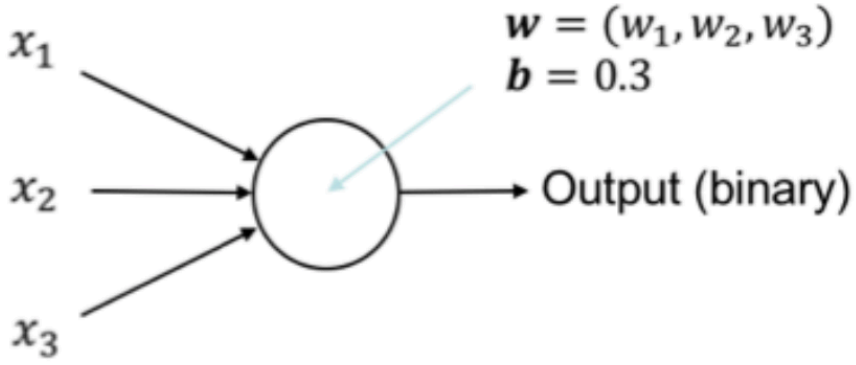
- Linear classifier – vector of weights *w* and a 'bias' *b*



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \qquad w \cdot x \equiv \sum_j w_j x_j.$$

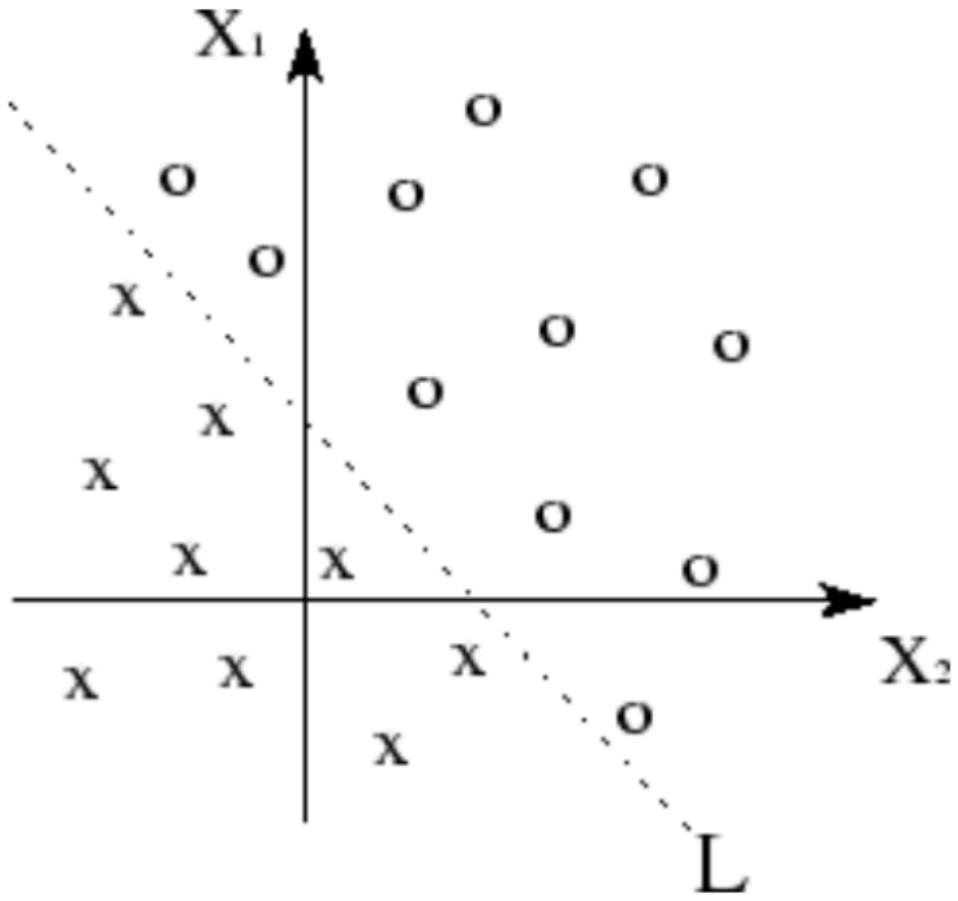
Perceptron as linear classifier

- Basic building block for composition is **perceptron** (Rosenblatt c.1960)
- Linear neuron classifier – vector of weights *w* and a 'bias' *b*



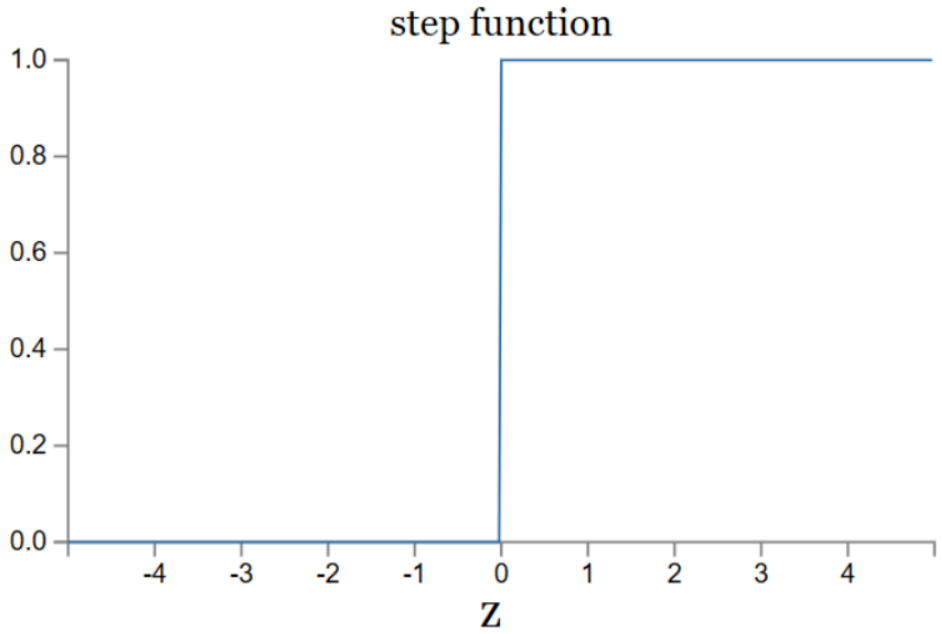
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \qquad w \cdot x \equiv \sum_j w_j x_j$$

Linear Classifier



Perceptron’s activation function

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



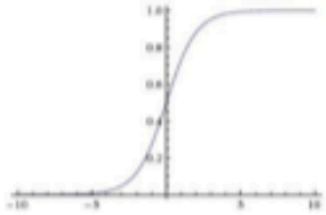
In the context of neural networks, a perceptron is an artificial neuron using the step function as the activation function.

Other types of activation function

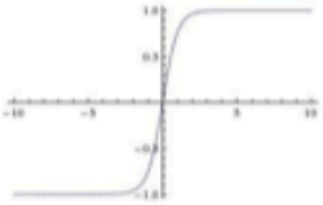
Activation Functions

Sigmoid

$$\sigma(x) = 1/(1+e^{-x})$$

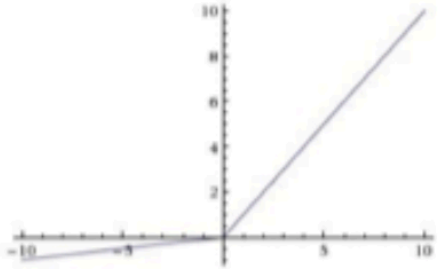


tanh tanh(x)



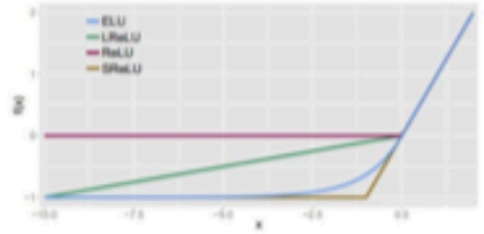
ReLU max(0,x)

Leaky ReLU
max(0.1x, x)



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$



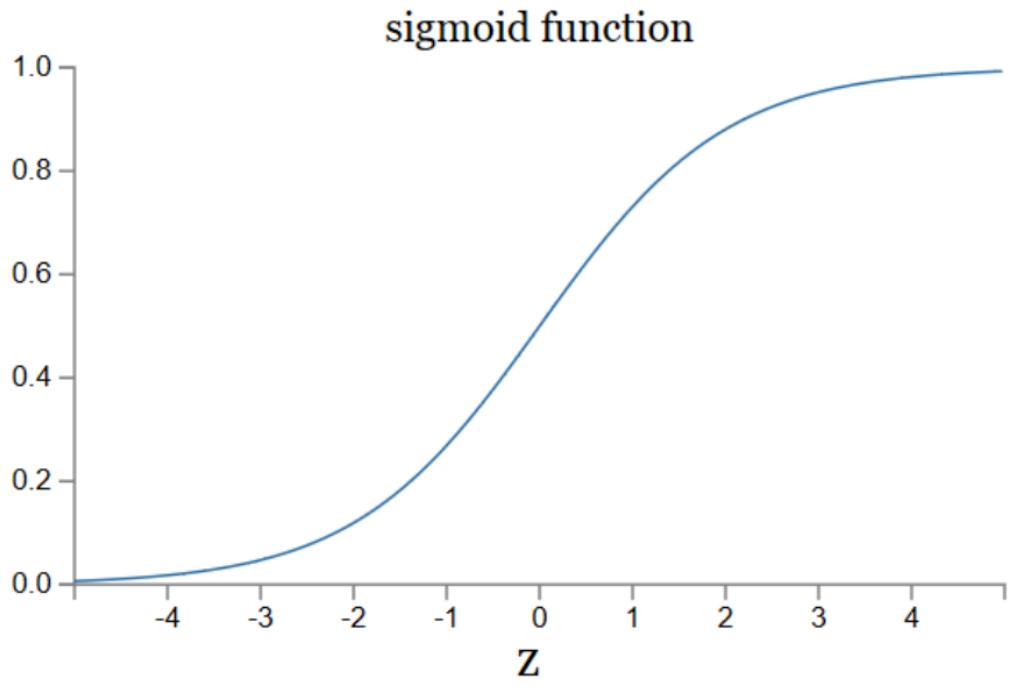
Slide from CS231n: stanford

Sigmoid non-linear activation

- We're going to introduce non-linear activation function to transform the features.

$$\sigma(w \cdot x + b)$$

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}.$$

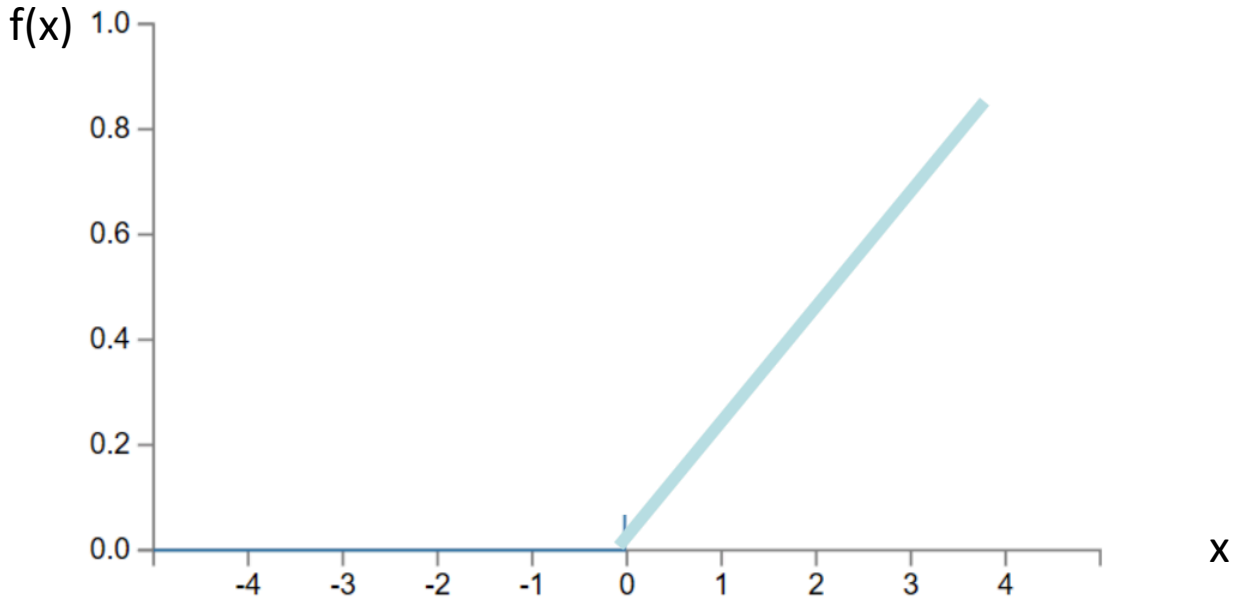


Nielsen

Another non-linear activation function: Rectified Linear Unit (ReLU)

- ReLU

$$f(x) = \max(0, x)$$



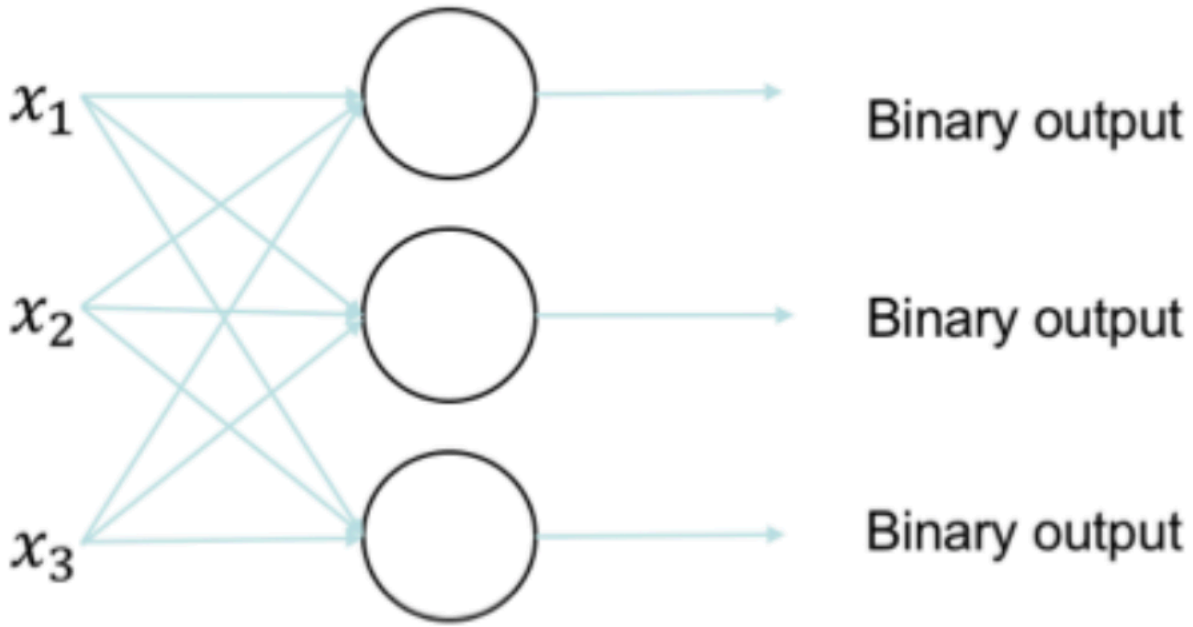
Binary Classifying an image

- Each pixel of the image would be an input.
- So, for a 28 x 28 image, we vectorize.
- **x** = 1 x 784

- **w** is a vector of weights for each pixel, 784 x 1
- b is a scalar bias per perceptron

- result = **xw** + b -> (1x784) x (784x1) + b = (1x1)+b

Add more neurons -> Multiclass neural networks



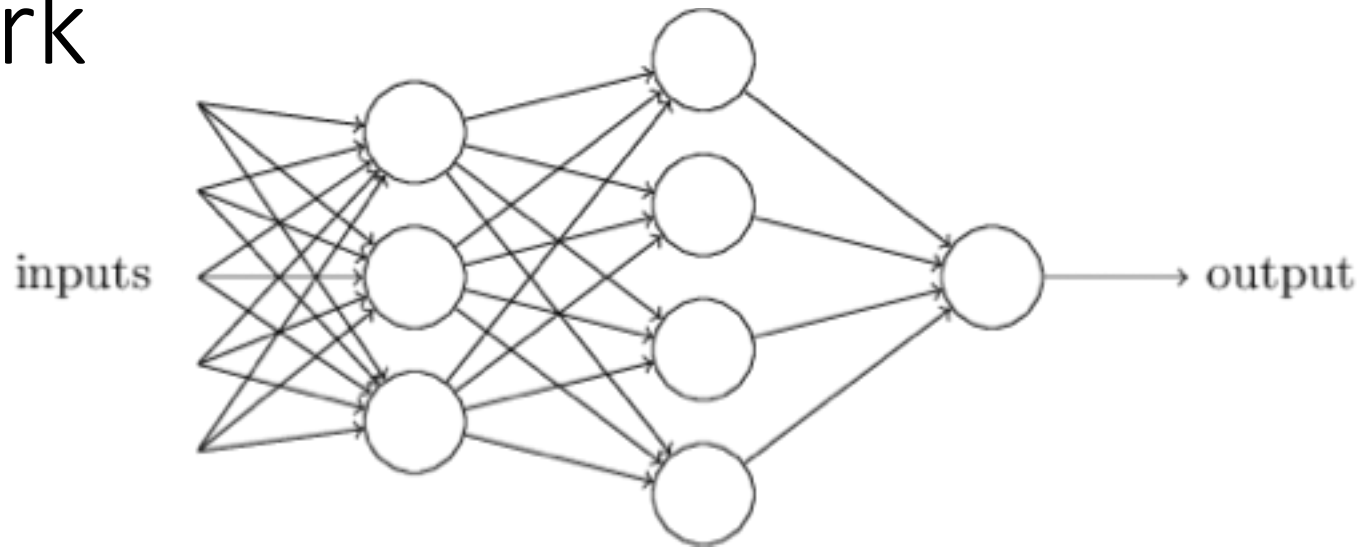
Multi-class classification

- Each pixel of the image would be an input node.
- So, for a 28x28 image, we vectorize the input image as a **x**=1x784 for

#(classes) = e.g., 10 ,in MNIST digit recognition

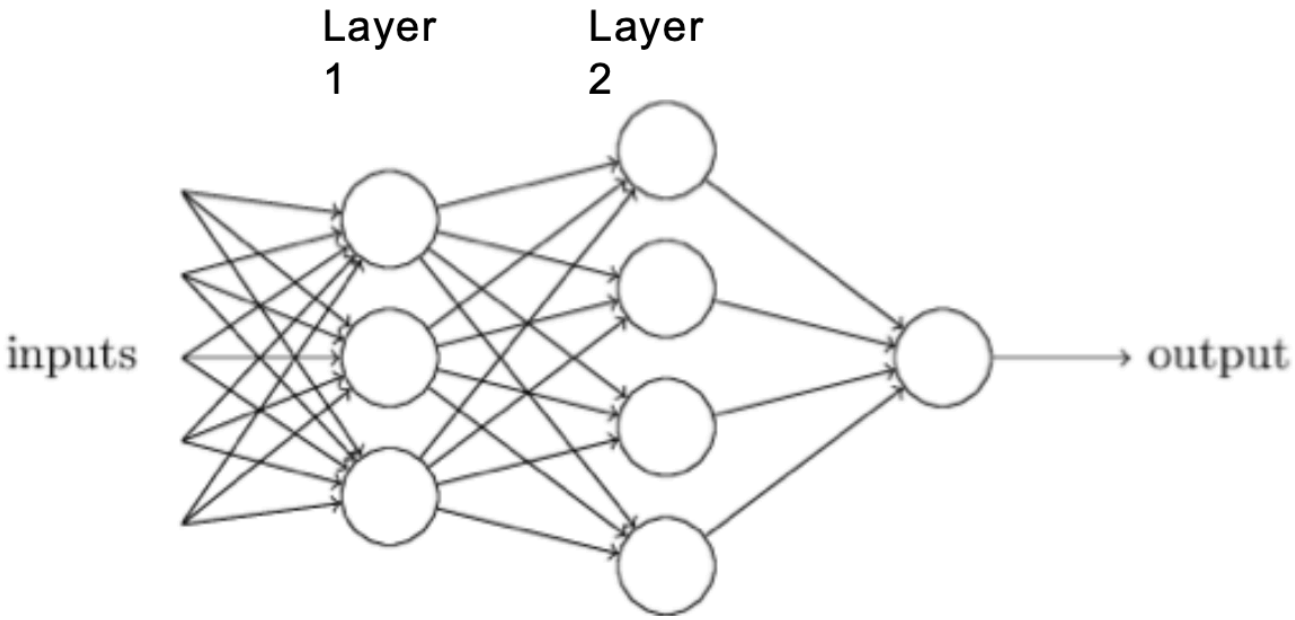
- **W** is a matrix of weights for each pixel/each perceptron – **W** = 784 x 10 (10-class classification)
- **b** is a bias *per perceptron* (vector of biases); (1 x 10)
- result=**xW**+**b** ->(1x784)x(784x10)+**b** -> (1 x 10) + (1 x 10) = 10-D output vector

Add more layers -> Multiple Layer neural network



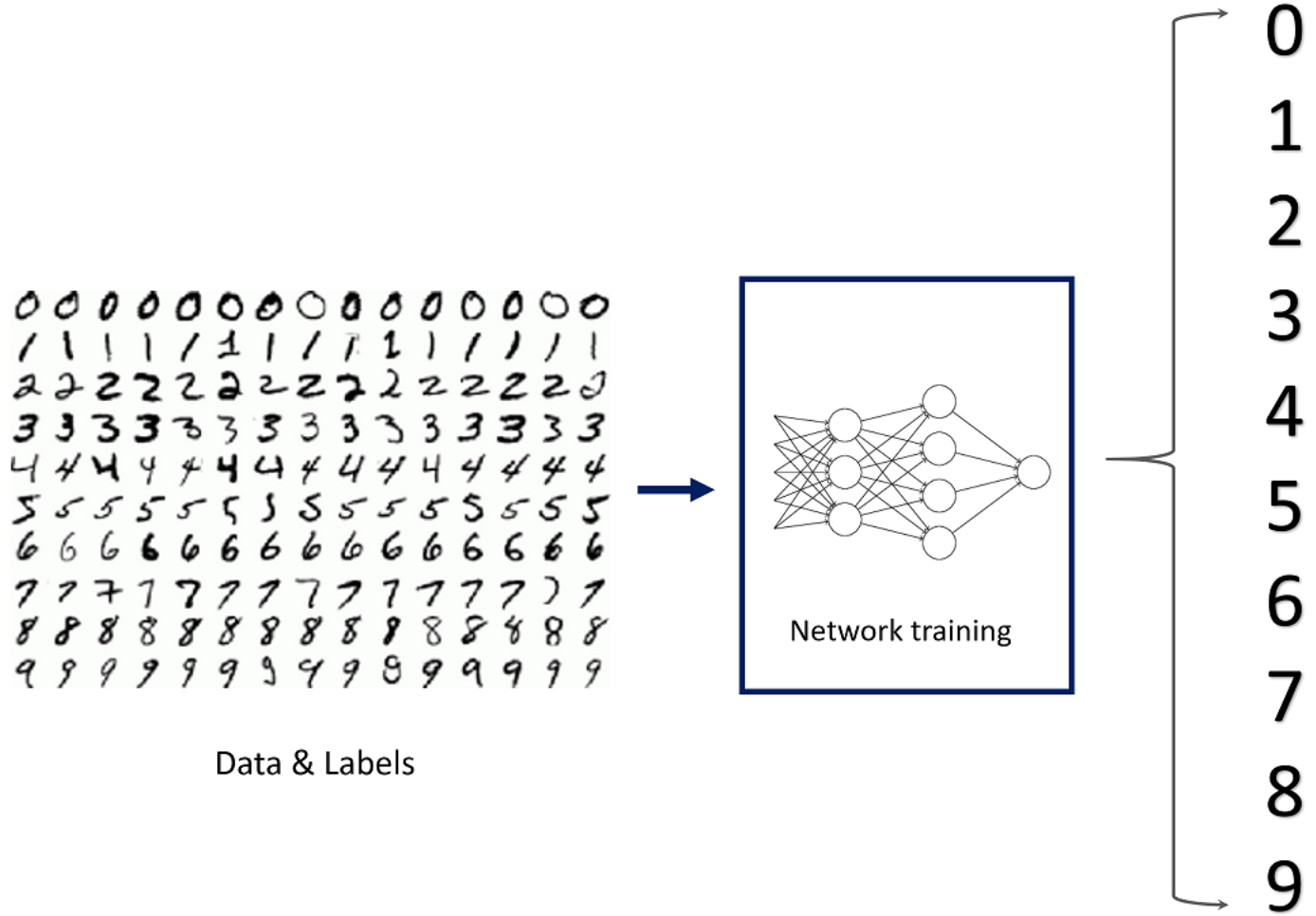
- Attempt to represent complex functions as compositions of smaller functions.
- Outputs from one perception are fed into inputs of another perceptron.

Composition

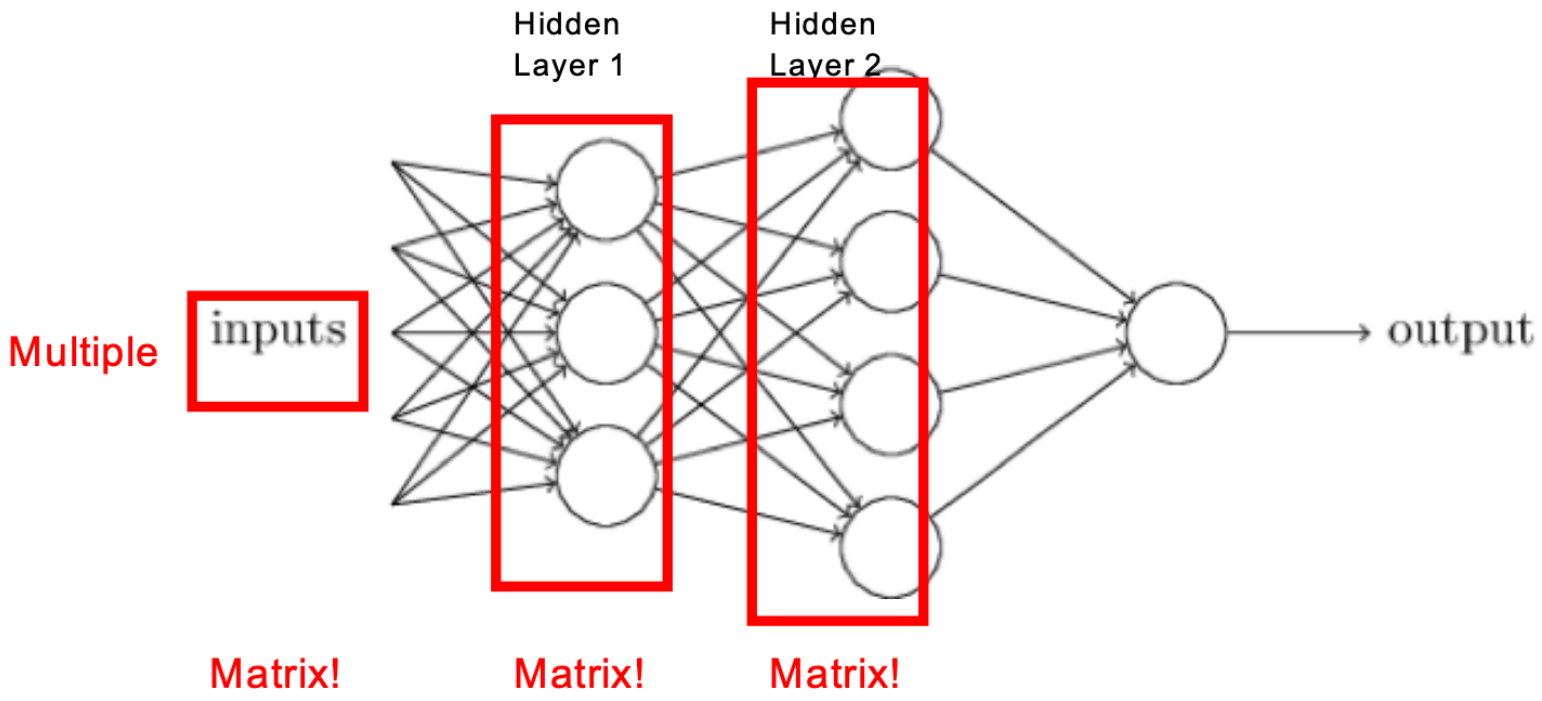


Sets of layers and the connections (weights) between them define the *network architecture.*

Application: Multi-class classification



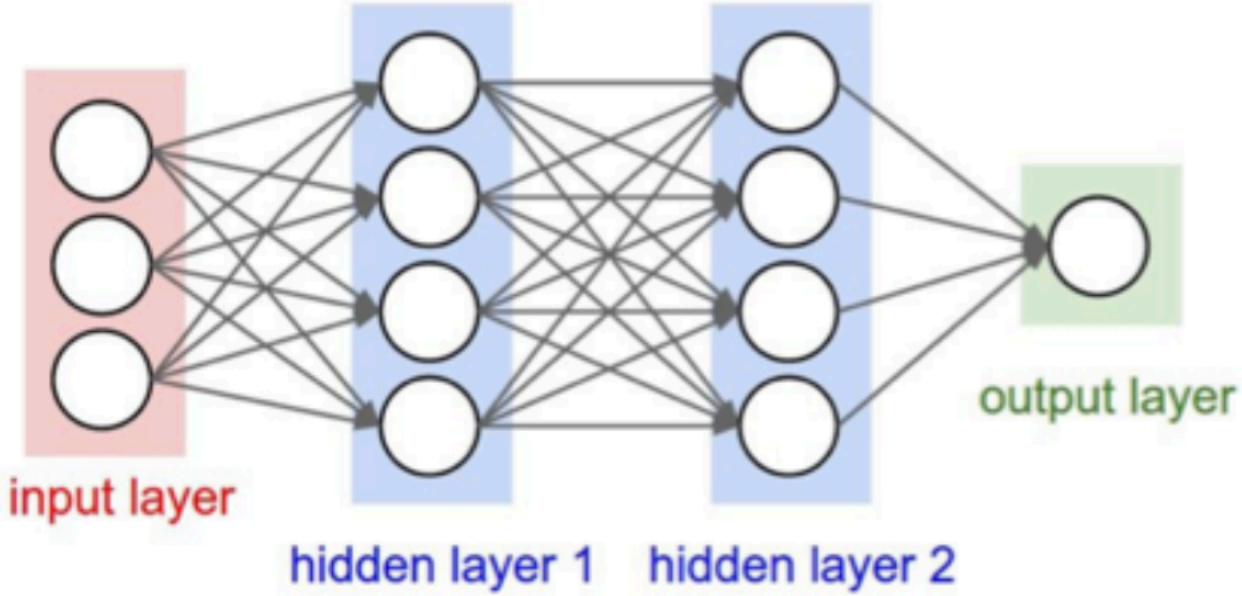
Parallel computation



It's all just matrix-vector multiplications !

GPUs -> special hardware for fast/large matrix multiplication.

Parallel Computation



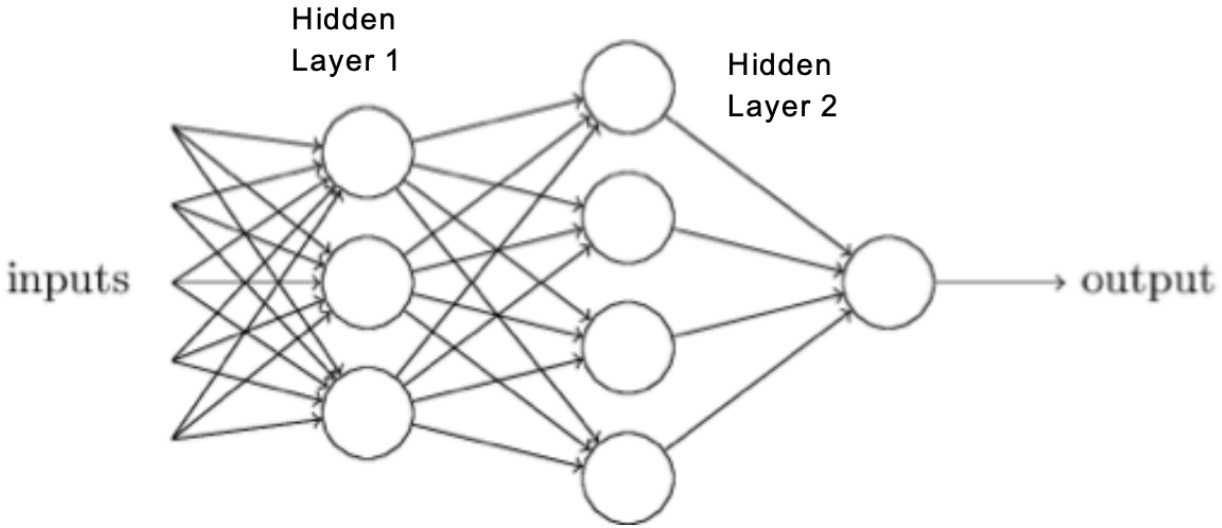

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Problems with using all-linear functions in the hidden layers

- We have formed multiple layer or chains of linear functions.
- We know that linear function of linear functions is a linear function, i.e. they can be reduced:
  - $g = f(h(x))$
  - Eg. $O_1 = xw_1+b_1$; $O_2 = o_1w_2+b_2$; $=>O_2 = (xw_1+b_1)w_2+b_2=xw_1w_2+(b_1w_2+b_2)$

Our multi-layer composition of functions is really just a single linear function : (

Multi-Layer Perceptron (MLP)

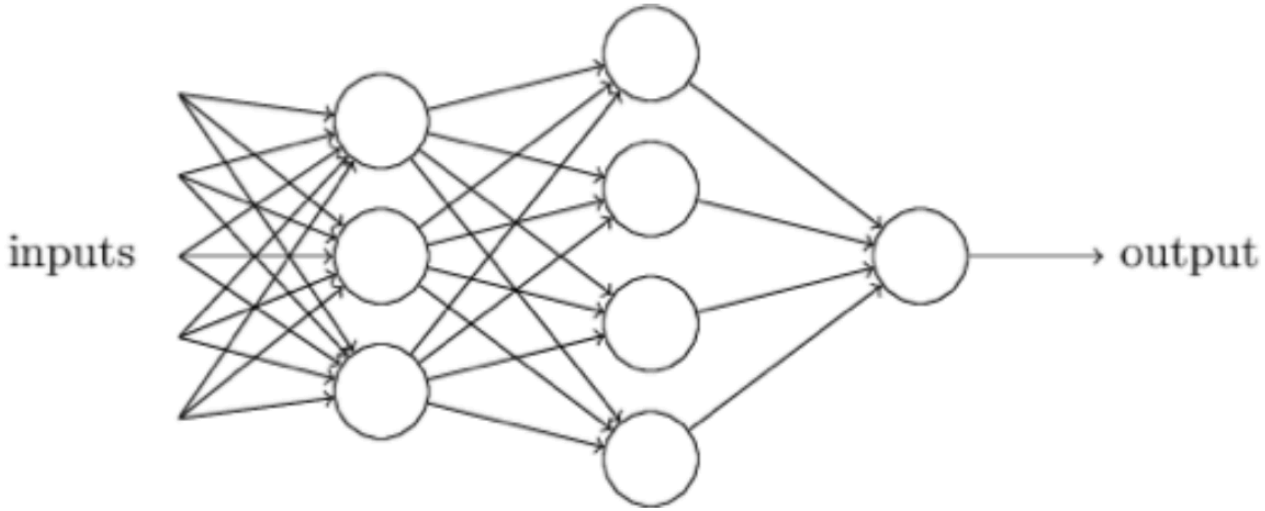


- Attempt to represent complex functions as compositions of smaller functions.
- Outputs from one perception are fed into inputs of another perceptron.
- Layers that are in between the input and the output are called *hidden layers*, because we are going to *learn* their weights via an optimization process.

Nielsen

Multi-Layer Perceptron (MLP)

- …is a '*fully connected*' neural network with non-linear activation functions.



- '*Feed-forward*' neural network

Nielson

Mathematical Theorem for MLP

- The application of MLP is grounded in theory.
- **Universal Approximation Theorem.**

With three (or more) layers and enough parameters, MLP can approximate any function.

Proof(Michael Nielson):

http://neuralnetworksanddeeplearning.com/chap4.html

If a 3-layer network can approximate any function …given enough parameters … then why should we need to go deeper ?

- Intuitively, composition is efficient because it allows *reuse*.
- Empirically, deeper networks do a better job than shallow networks at learning certain hierarchies of knowledge.

Why go **deeper** ?

- A three-layer network is already a “universal approximator”,
- Only if the number of neurons in the hidden layer is sufficiently many.
- The identification of the weights is troublesome (possible convergence into wrong local minima)
- Deep neural networks
  - Have many levels of non-linearity
  - Compactly represent highly non-linear functions
  - Layered structure facilitates convergence to a good optimum
  - **There are many local optima of approximately same (and good) quality**

Interpretation

Question: What does a hidden unit do?

Answer: It can be thought of as a classifier or feature detector.

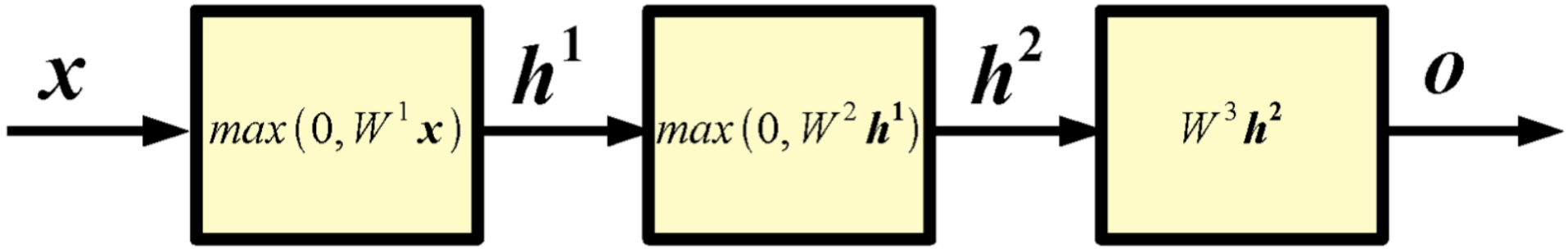
Question: How many layers? How many hidden units?

Answer: Cross-validation or hyper-parameter search methods are the answer. In general, the wider and the deeper the network the more complicated the mapping.

Question: How do I set the weight matrices?

Answer: Weight matrices and biases are learned.
First, we need to define a measure of quality of the current mapping.
Then, we need to define a procedure to adjust the parameters.

Neural Networks: example



x input

h^1 1-st layer hidden units

h^2 2-nd layer hidden units

o output

Example of a 2 hidden layer neural network (or 4 layer network, counting also input and output).

Supervised Learning

$\{(\boldsymbol{x}^i, y^i), i = 1 \dots P\}$ training dataset

$\boldsymbol{x}^i$ i-th input training example

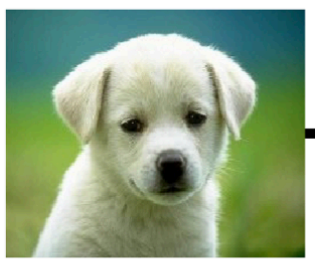
y^i i-th target label

P number of training examples

Goal: predict the target label of unseen inputs.

Supervised Learning Example

Classification



“dog”

classification

Denoising

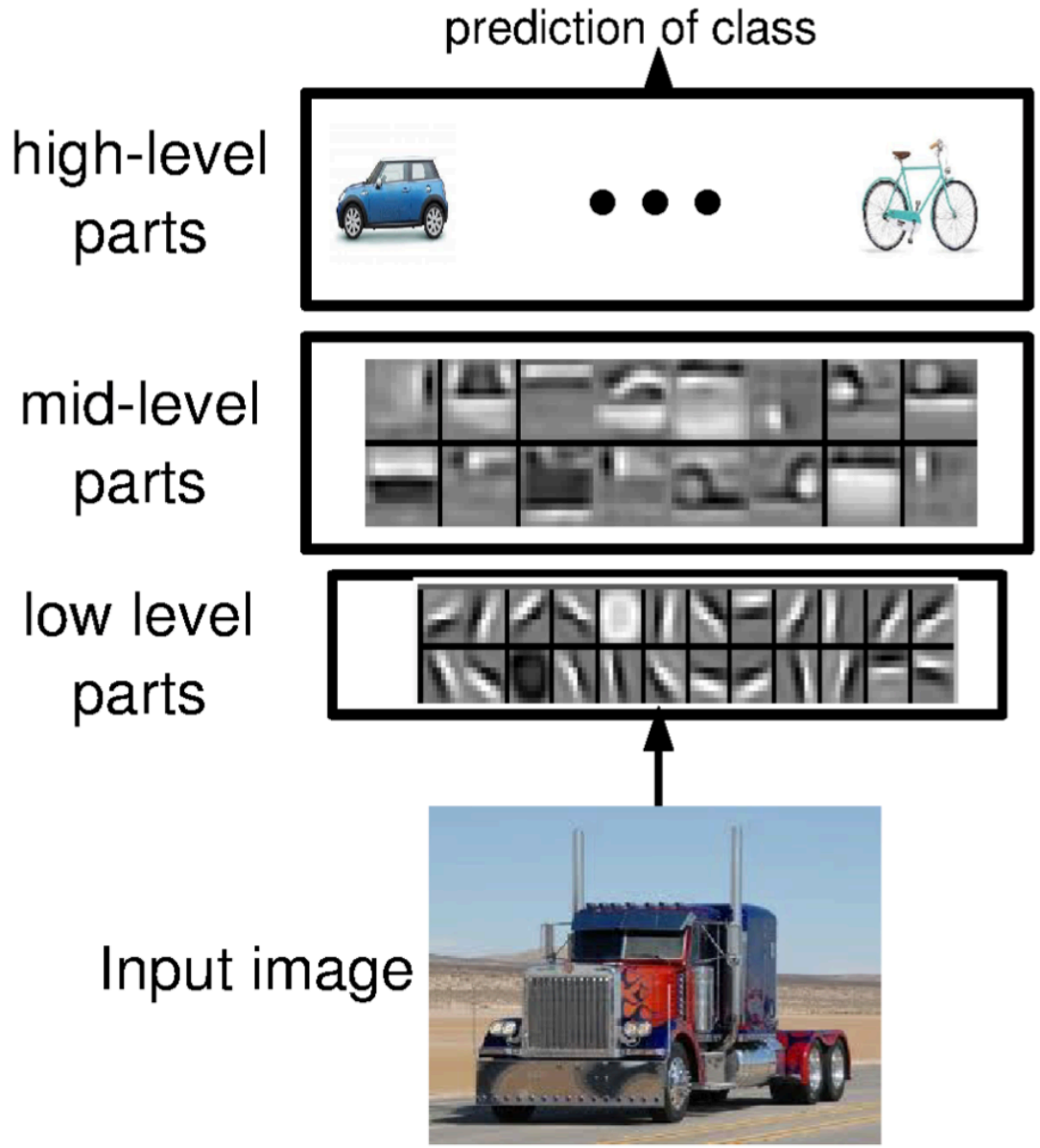
regression

OCR

“2 3 4 5”

structured prediction

Interpretation

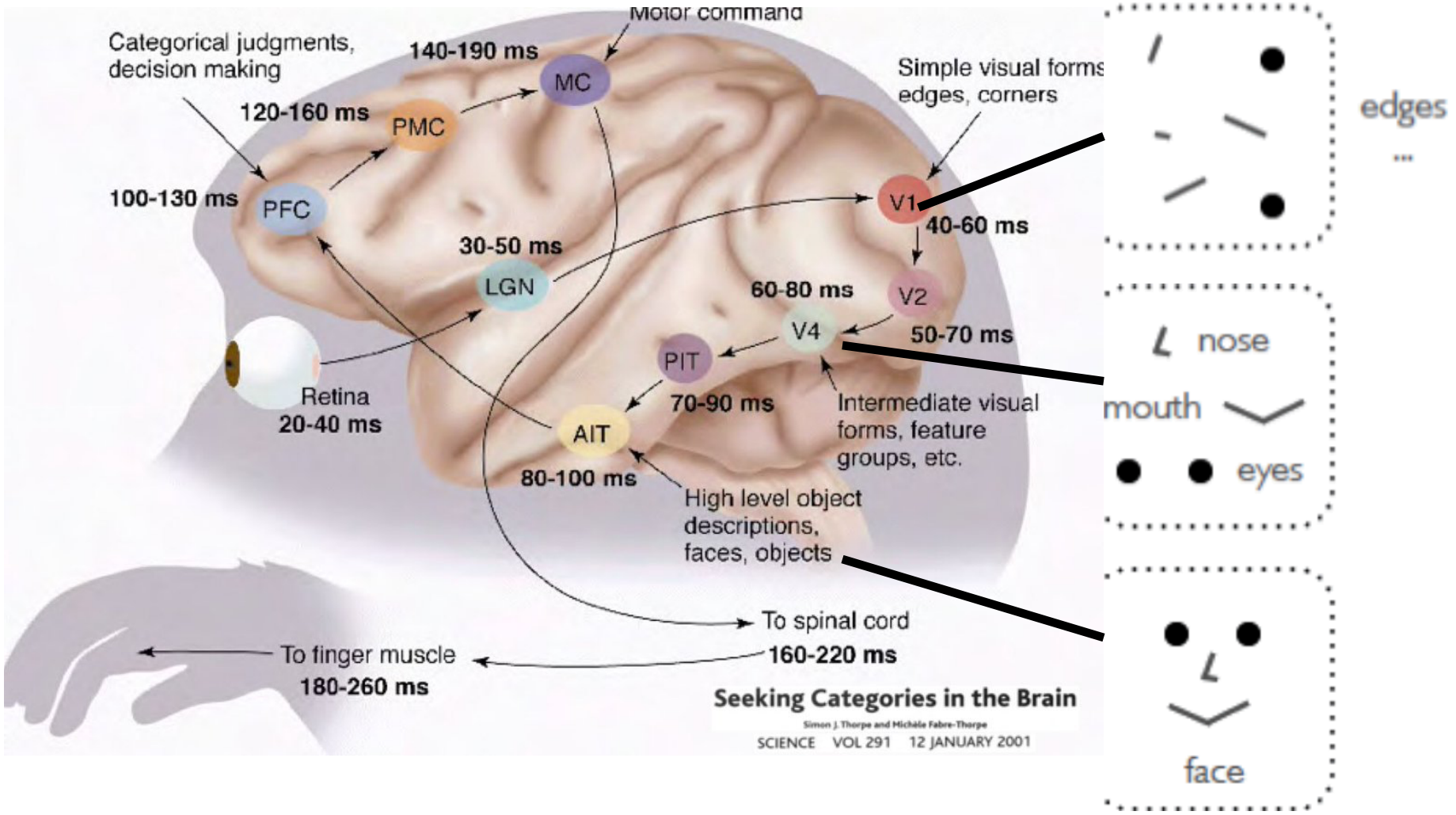


- distributed representations
- feature sharing
- compositionality

NOTE: Not actually the weights; a demonstrative visualization!

Lee et al. "Convolutional DBN's ..." ICML 2009

Biological inspiration: Human Neural Systems

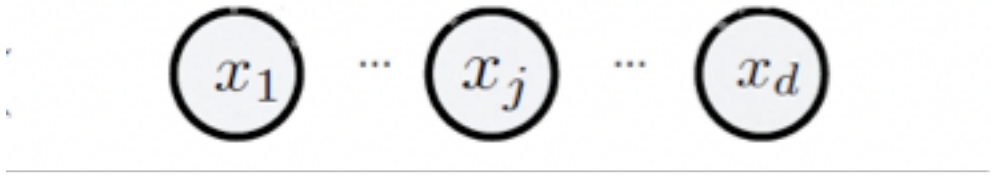


Multilayer Neural Networks

Forward Computation

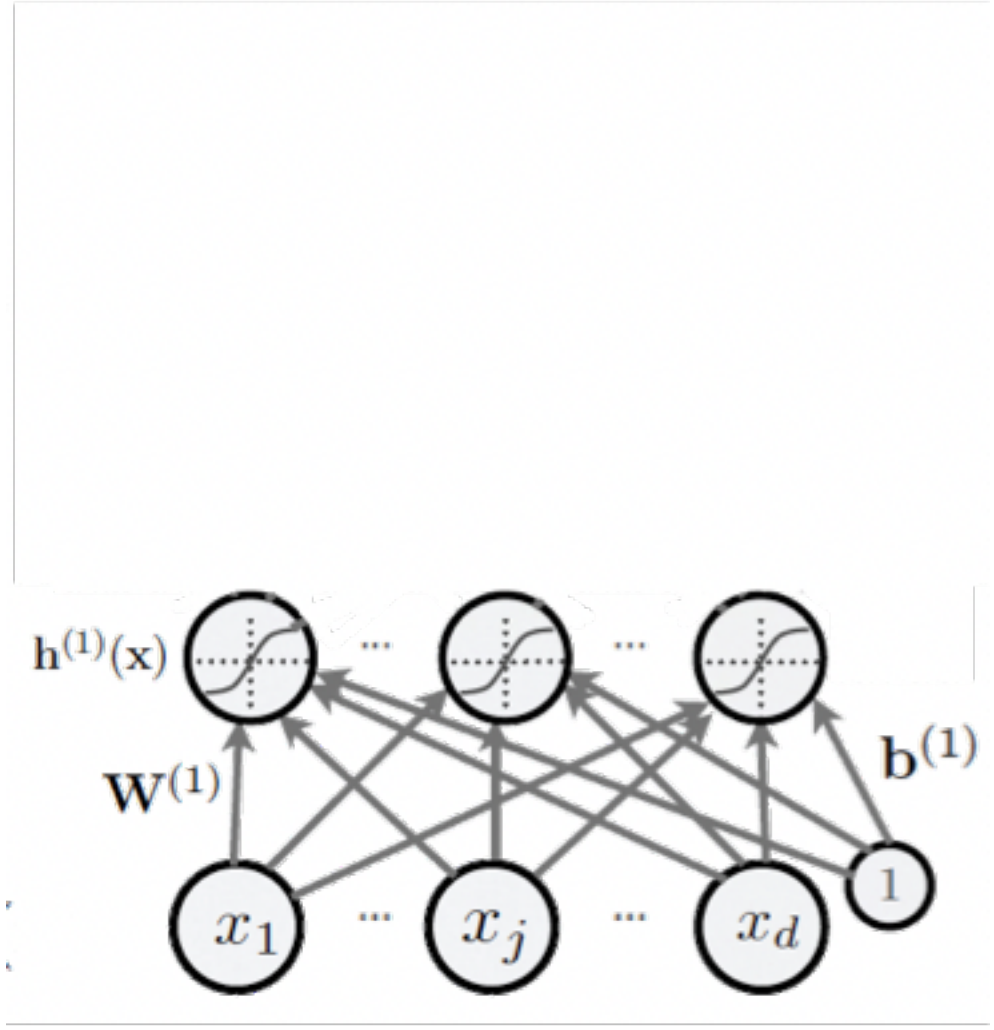
Multilayer Neural Network

- Forward



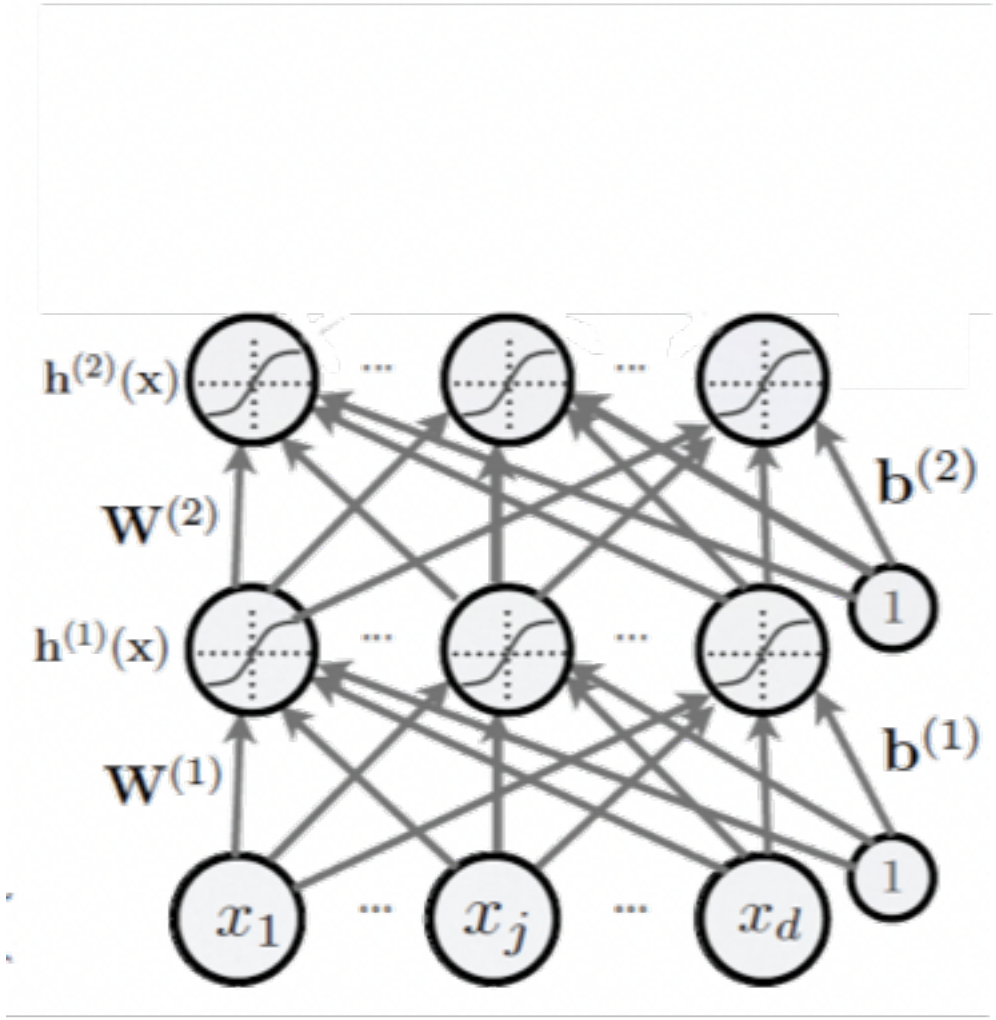
Multilayer Neural Network

- Forward



Multilayer Neural Network

- Forward



Multilayer Neural Network

- Could have *L* hidden layers:
  - Layer pre-activation for k>0:

$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

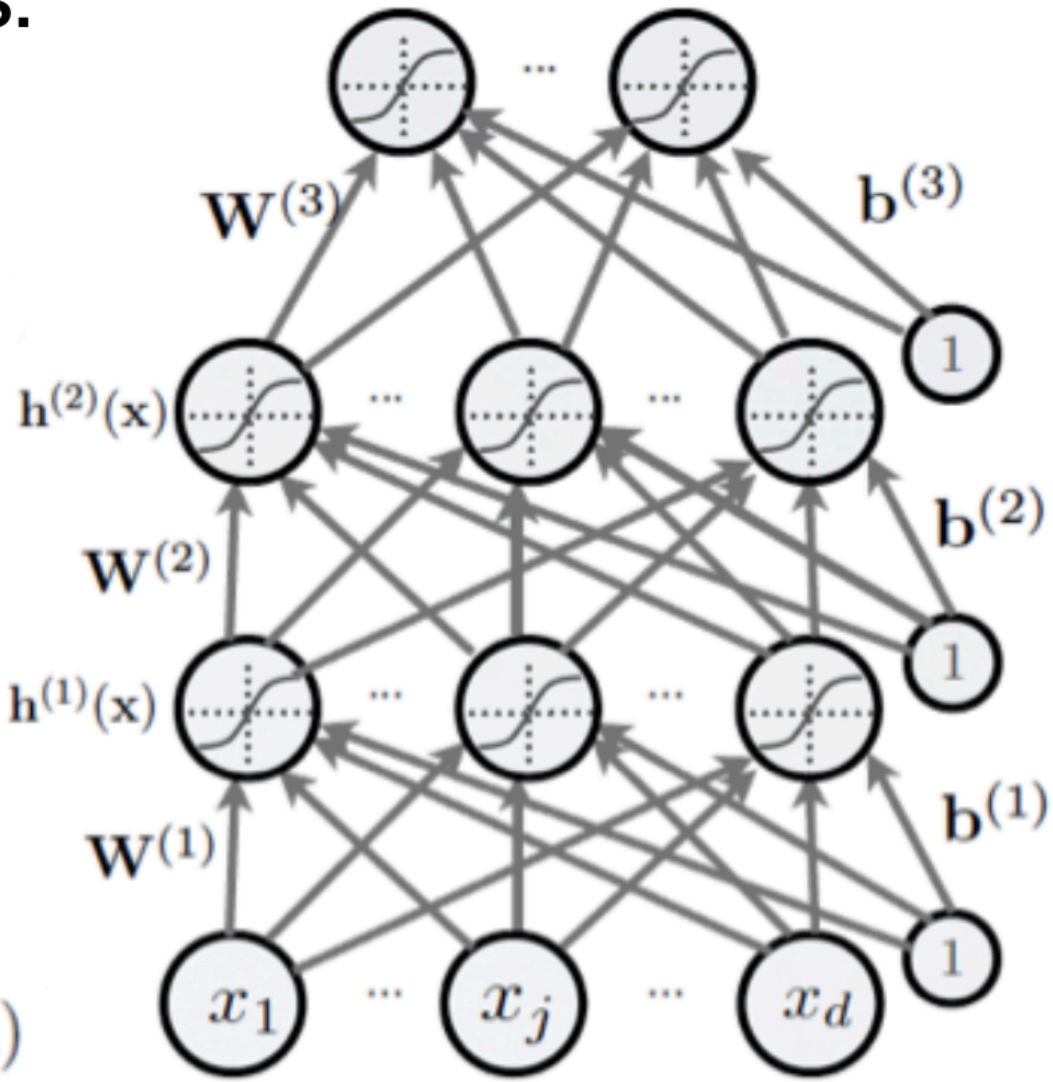
$$(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$$

  - Hidden layer activation (*k* from 1 to *L*):

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

  - Output layer activation (*k*=*L*+1):

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



Reference

- Chapter 6 Deep feedforward networks, Deep Learning, https://www.deeplearningbook.org/