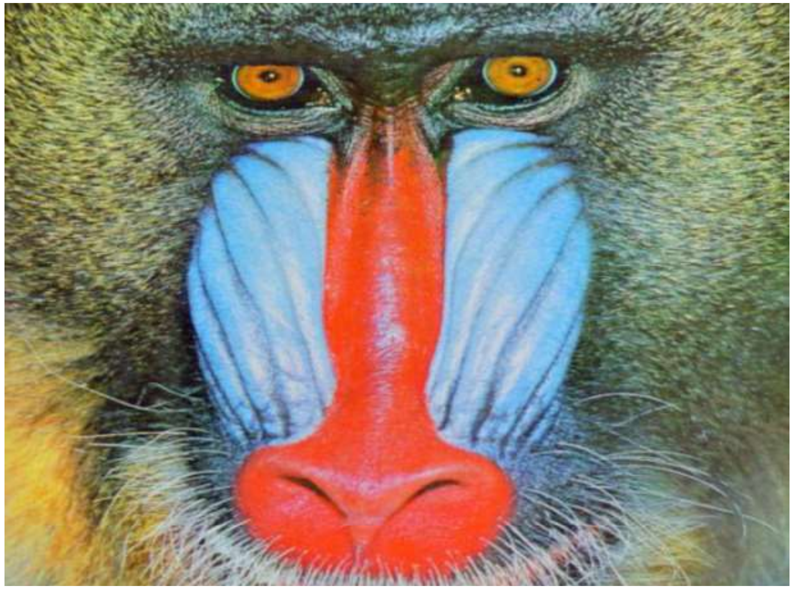
Deep Neural network

Last time

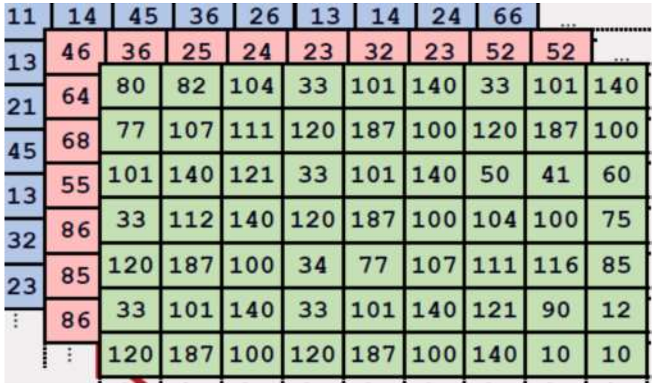
- Difference between linear regression and logistic regression

- How to optimize logistic regression

Preliminariries: Digital image representation



=



Picture element: pixel

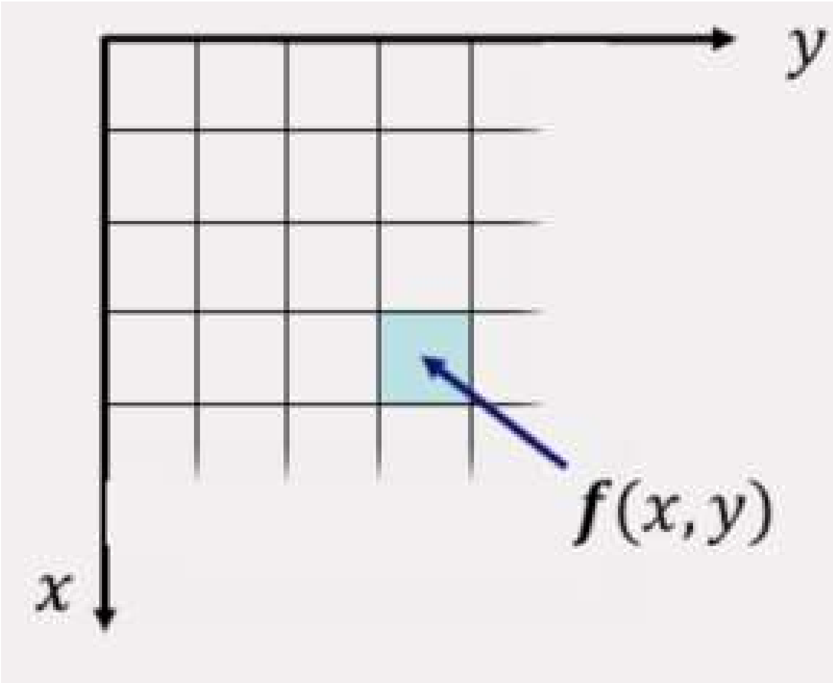
Typically 8-bits per channel [0-255] (UINT)

Preliminairies: Convolution / image filtering

Linear filtering

$$\boldsymbol{w}(x,y) \star \boldsymbol{f}(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \boldsymbol{w}(s,t)\boldsymbol{f}(x-s,y-t)$$

Image

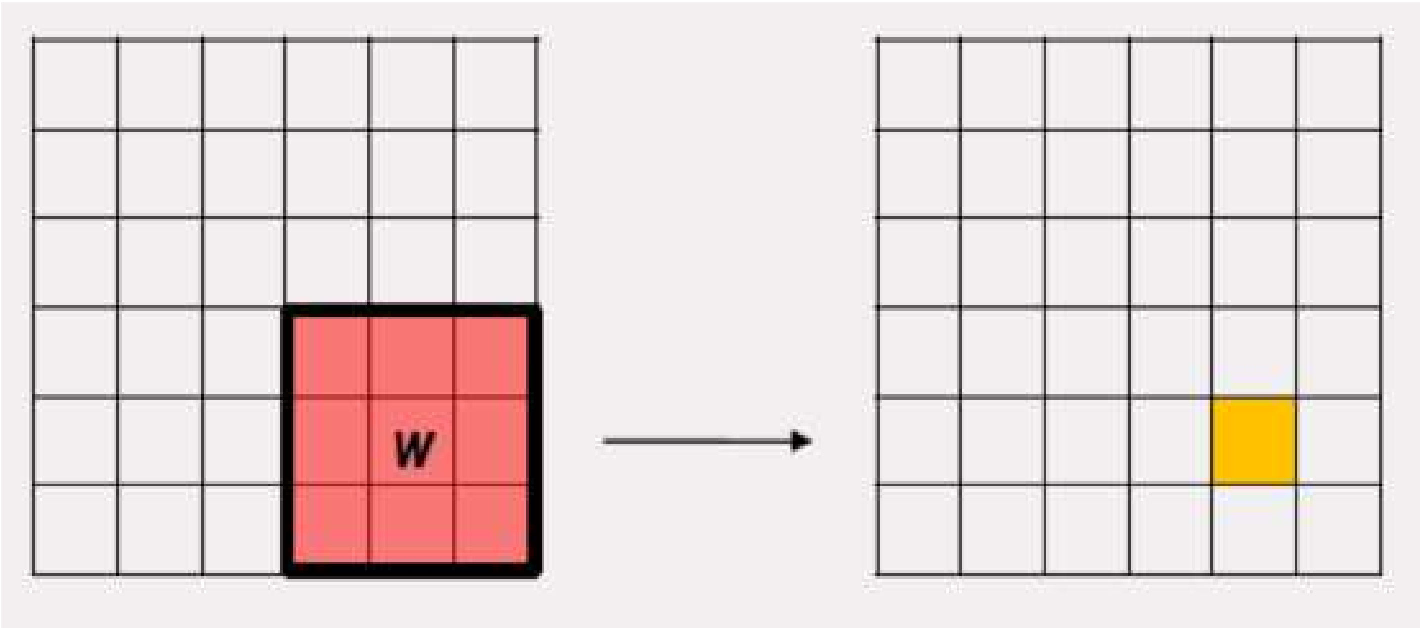


$\boldsymbol{w}(-1,-1)$	$\boldsymbol{w}(-1,0)$	$\boldsymbol{w}(-1,1)$
$\boldsymbol{w}(0,-1)$	$\boldsymbol{w}(0,0)$	$\boldsymbol{w}(0,1)$
$\boldsymbol{w}(1,-1)$	$\boldsymbol{w}(1,0)$	$\boldsymbol{w}(1,1)$

Filter kernel

Preliminairies: Convolution

Linear filtering



How to deal with pixels at the border?

Preliminairies: Convolution

Flip mask w.r.t. signal

$$\boldsymbol{w}(x,y) \star \boldsymbol{f}(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \boldsymbol{w}(s,t)\boldsymbol{f}(x-s,y-t)$$

Image f

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

Kernel w

1	2	3
4	5	6
7	8	9

Zero padded image

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

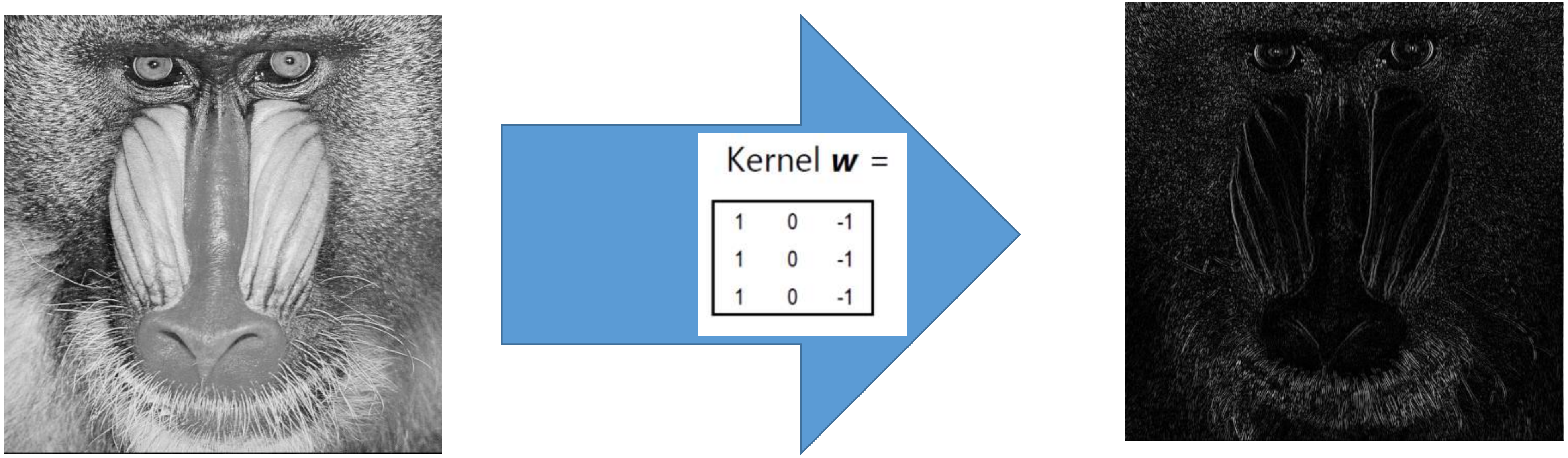
Zero padded image

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	1	2	3	0	0	0
0	0	0	4	5	6	0	0	0
0	0	0	7	8	9	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Cropped result

0	0	0	0	0
0	1	2	3	0
0	4	5	6	0
0	7	8	9	0
0	0	0	0	0

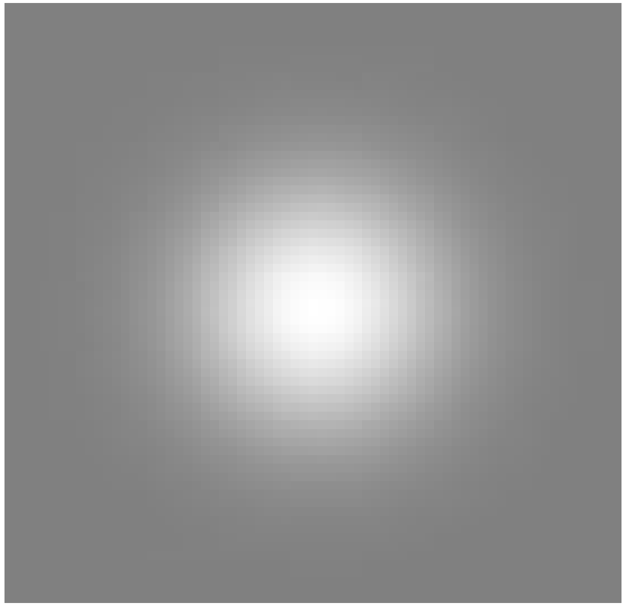
Preliminairies: Convolution

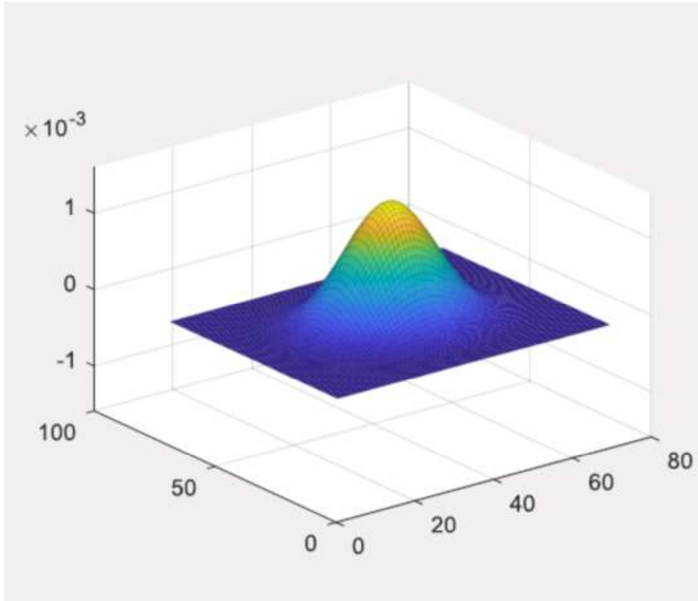


try your self

```
import cv2
import numpy as np

# Create a dummy input image.
canvas = np.zeros((100, 100), dtype=np.uint8)
canvas = cv2.circle(canvas, (50, 50), 20, (255,), -1)

kernel = np.array([[-1, -1, -1],
                   [-1, 4, -1],
                   [-1, -1, -1]])

dst = cv2.filter2D(canvas, -1, kernel)
cv2.imwrite("./filtered.png", dst)
```

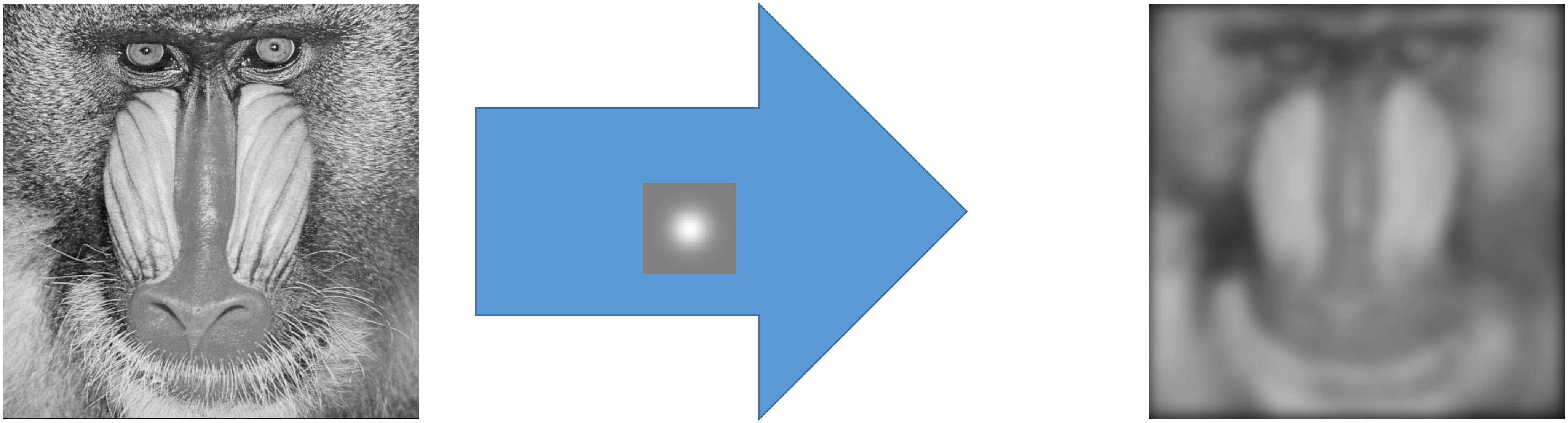
Preliminairies: Convolution



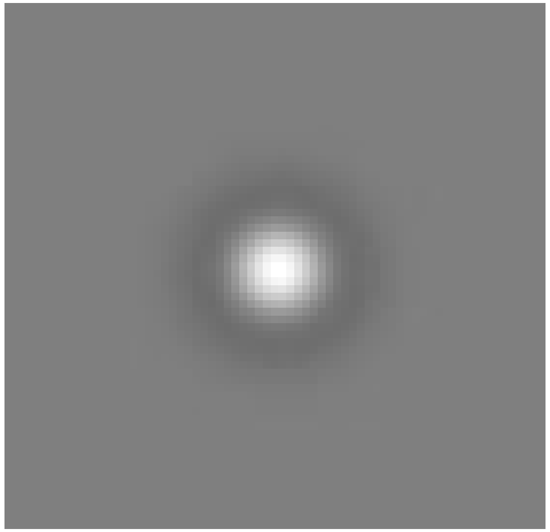


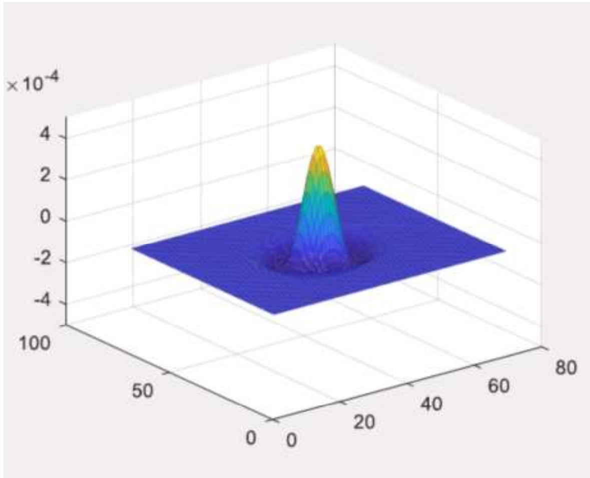
Low-pass filter

Preliminairies: Convolution



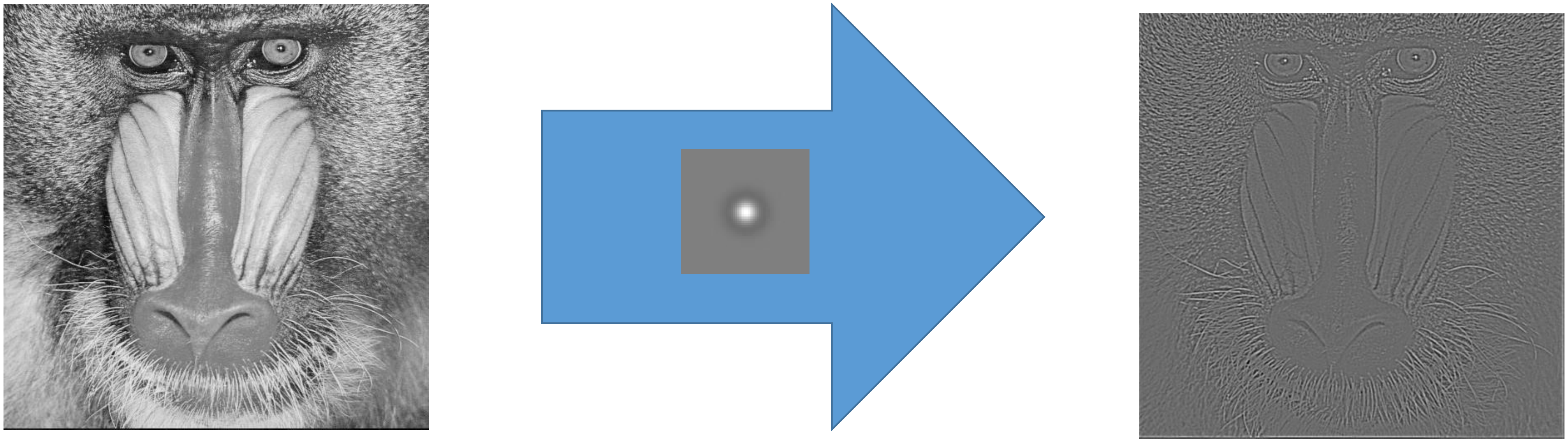
Preliminairies: Convolution



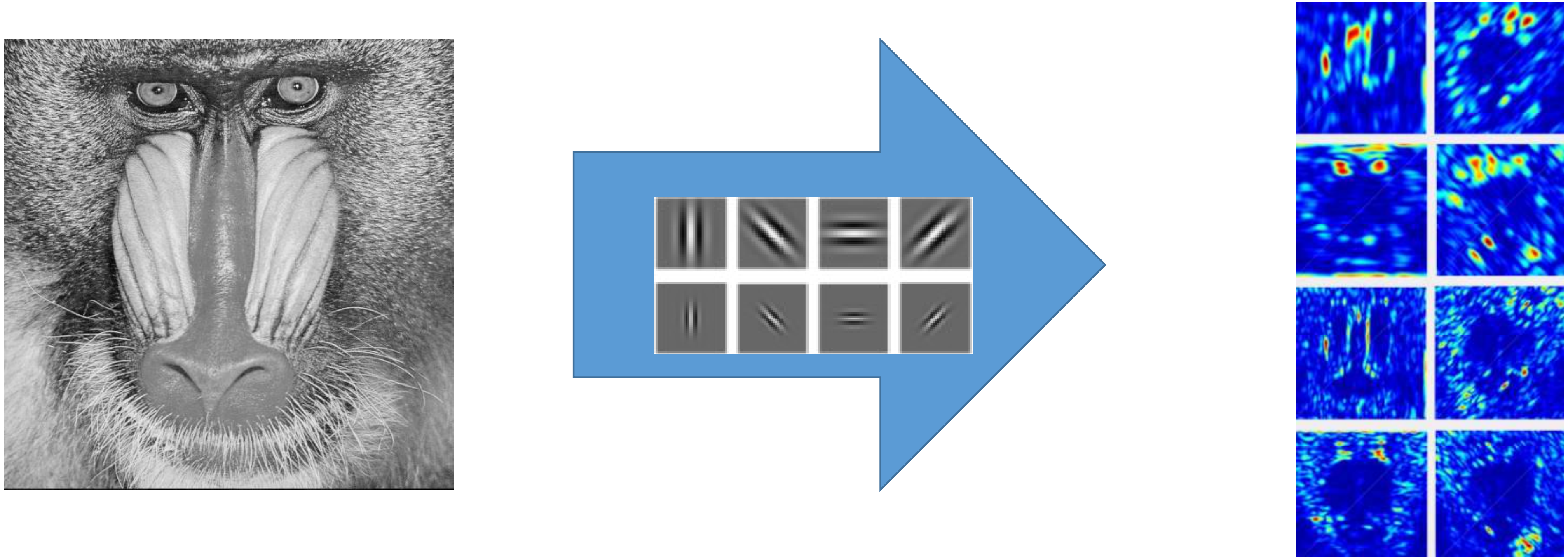


High-pass filter

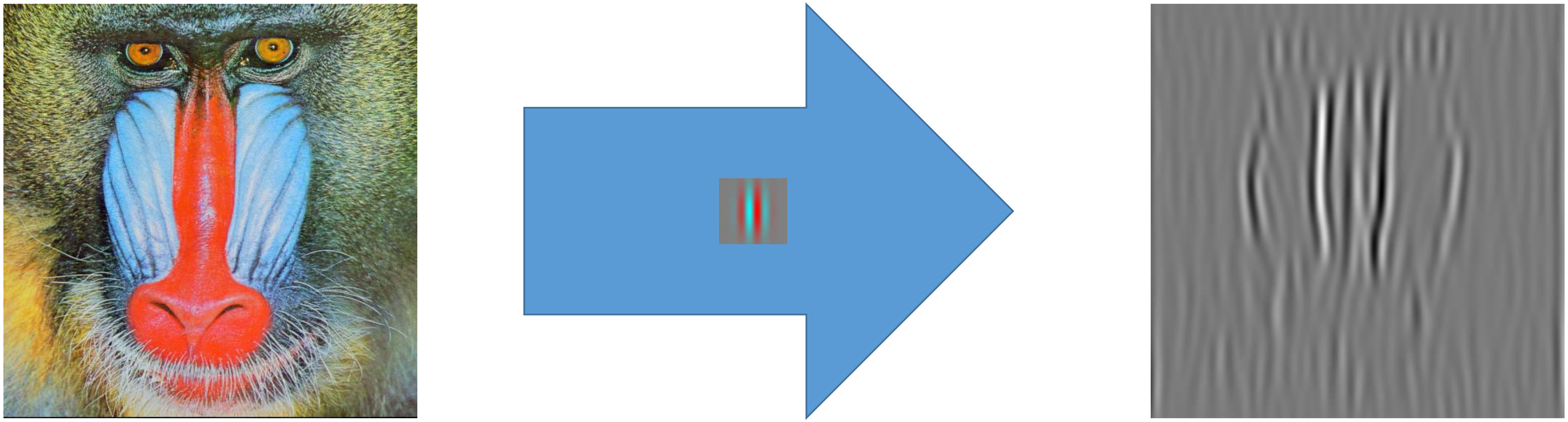
Preliminairies: Convolution



Preliminairies: Convolution

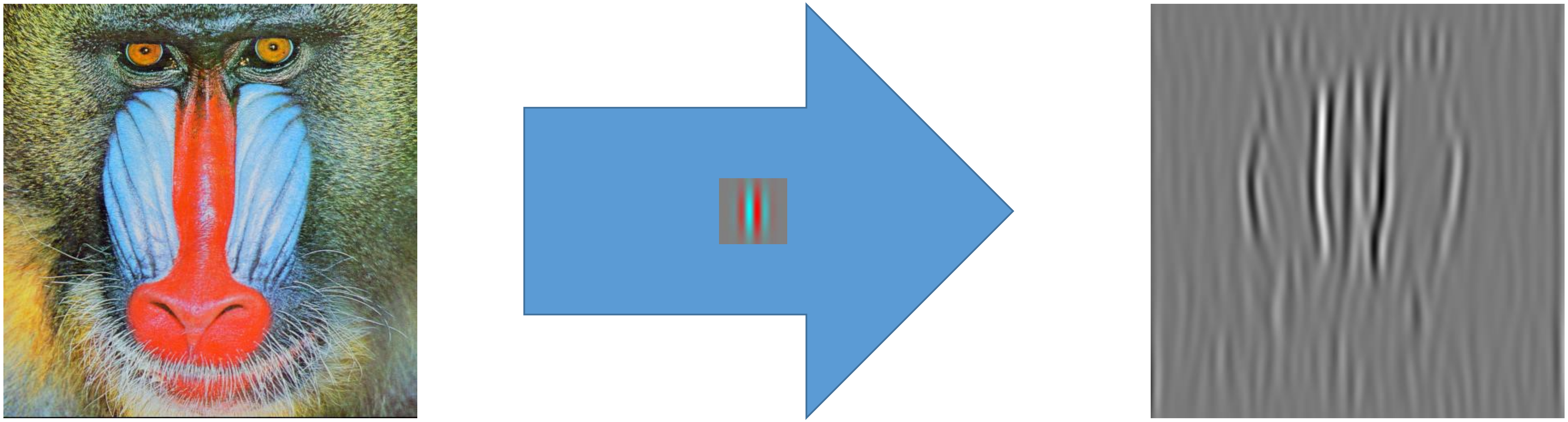


Preliminairies: Convolution



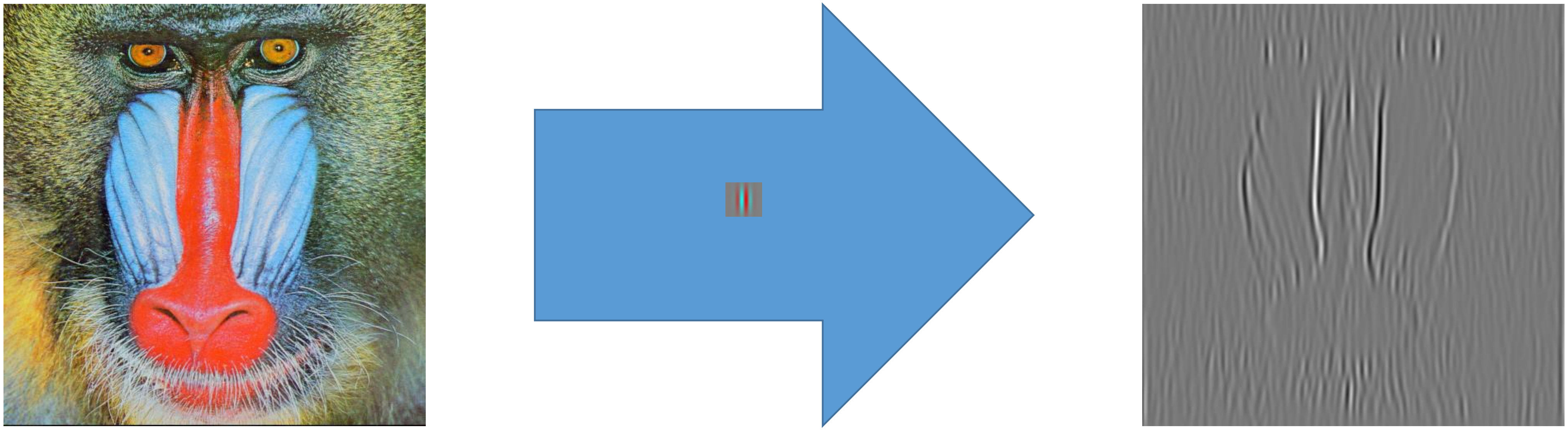
Preliminairies: Convolution

Single filters to find specific color changes

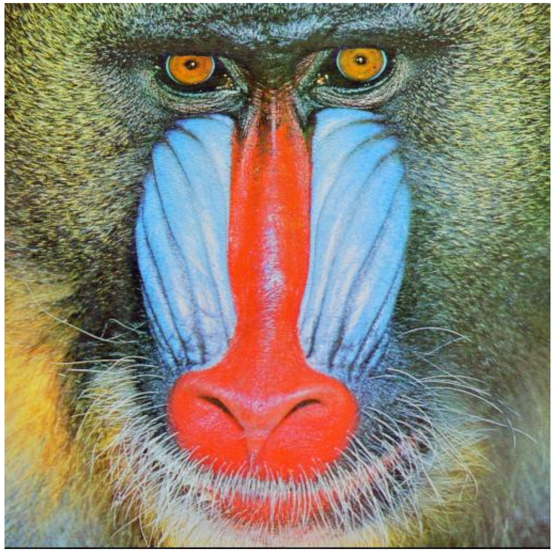


Preliminairies: Convolution

Single filters to find specific color changes



Preliminairies: Convolution



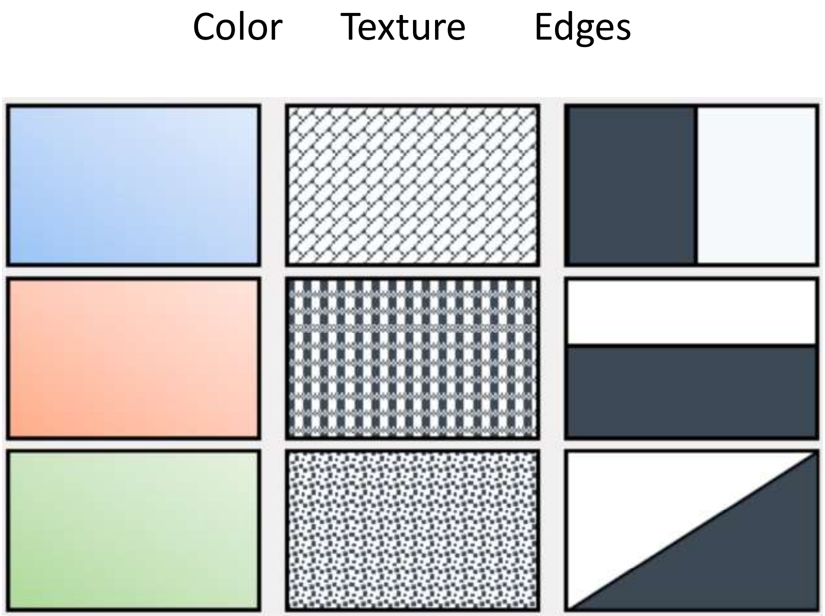


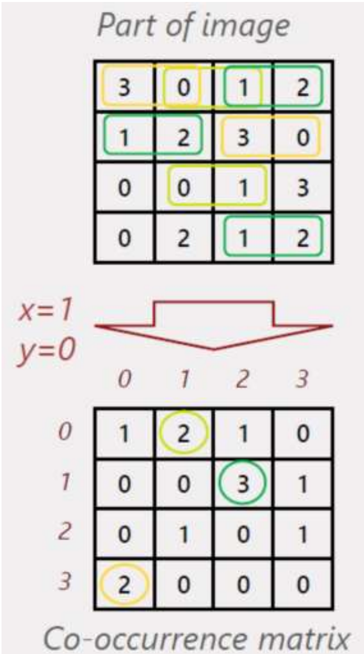
Image features Co-occurrence matrix

- Given a grey-level image I, co-occurrence matrix computes how often pairs of pixels with a specific value and offset occur in the image.

- The offset, $(\Delta x, \Delta y)$, is a position operator that can be applied to any pixel in the image (ignoring edge effects): for instance, (1,2) could indicate "one down, two right".
- An image with p different pixel values will produce a p xp co-occurrence matrix, for the given offset.
- The $(i,j)^{th}$ value of the co-occurrence matrix gives the number of times in the image that the i^{th} and j^{th} pixel values occur in the relation given by the offset.

$$C_{\Delta x,\Delta y}(i,j) = \sum_{x=1}^{n}\sum_{y=1}^{m}\begin{cases}1, & \text{if } I(x,y)=i \text{ and } I(x+\Delta x, y+\Delta y)=j\\ 0, & \text{otherwise}\end{cases}$$

Co-occurrence matrix



Statistical measures

Homogeneity	$\sum_i \sum_j \frac{P(i,j)}{1+\|i-j\|}$
Contrast	$\sum_i \sum_j (i-j)^2 P(i,j)$
Energy	$\sum_i \sum_j P(i,j)^2$
Dissimilarity	$\sum_i \sum_j P(i,j)\|i-j\|$
Entropy	$-\sum_i \sum_j P(i,j) \log(P(i,j)+\varepsilon)$
Correlation	$\sum_i \sum_j \frac{(i-\mu_x)(i-\mu_y)P(i,j)}{\sigma_x \sigma_y}$

$$C_{\Delta x,\Delta y}(i,j) = \sum_{x=1}^{n}\sum_{y=1}^{m} \begin{cases} 1, & \text{if } I(x,y)=i \text{ and } I(x+\Delta x, y+\Delta y)=j \\ 0, & \text{otherwise} \end{cases}$$

Example

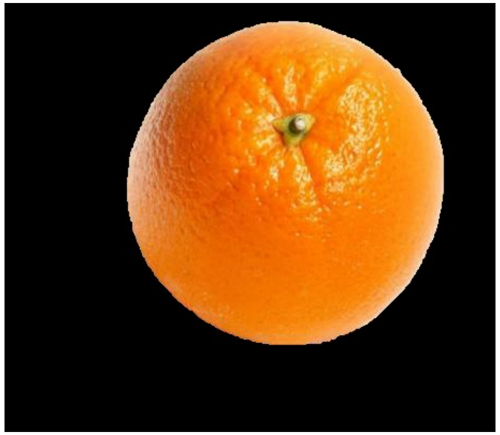
https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_glcm.html

Preliminairies: Image features

- Generally not optimal to work on the raw data
  - Very sensitive to changes in viewpoint, illumination, scaling, rotation, etc.
  - Super high dimensional: curse of dimensionality
- Better idea : use a low dimensional mapping of the original data
  - Summarize the image into a set of descriptive features (lines, corners, colors, texture, …)
  - Enables training relatively simple and robust classification models
- Concept also used in neural networks
  - Use an encoding scheme to obtain a representation in a latent (feature) space
  - Similar to image compression!

Preliminairies: Classification

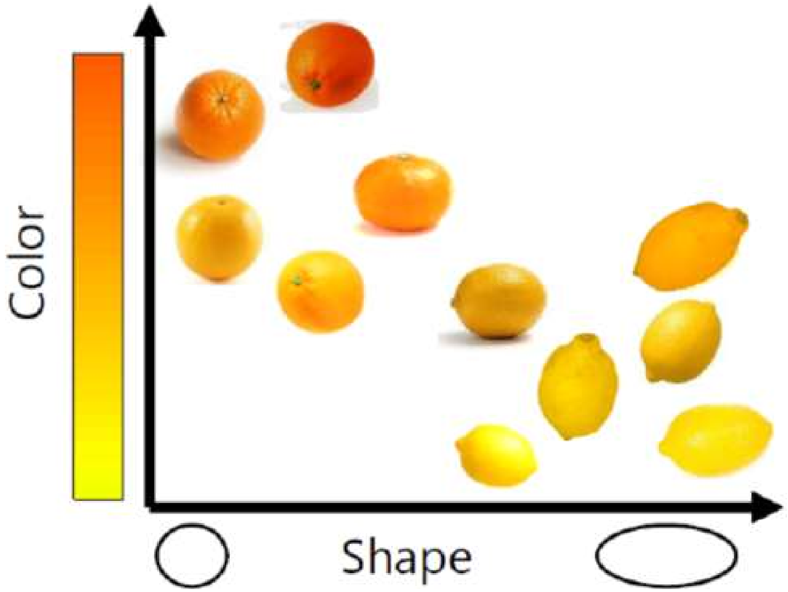
- Linear classification example:separate lemons from oranges

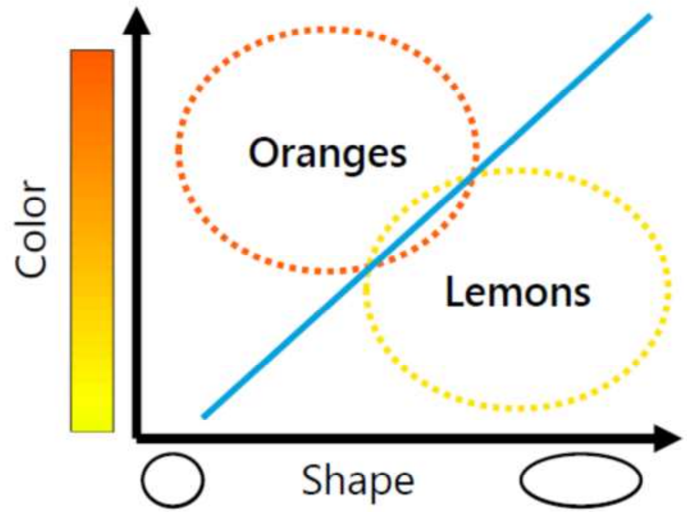


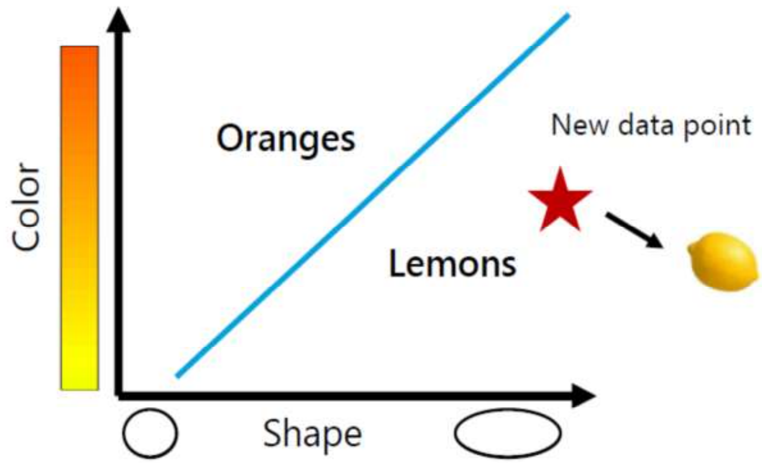
Color:
orange
Shape:
sphere
Diameter:
Diameter:±8 cm
Weigth:
±0.1 kg

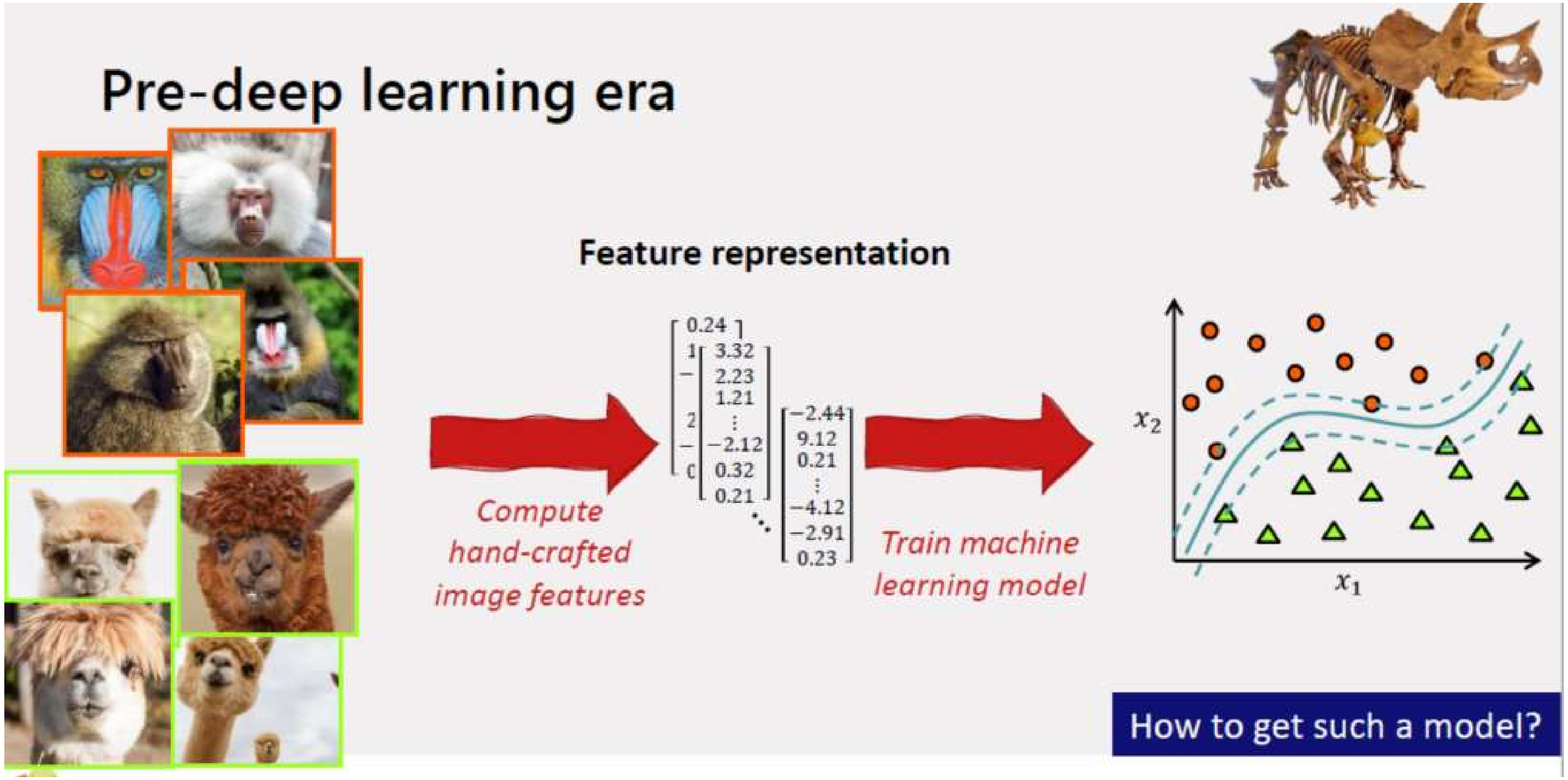
Color:
yellow
Shape:
elipsoid
Diameter:
Diameter:±8 cm
Weigth:
±0.1 kg

→ Use "color" and "shape" as features


Color
Shape

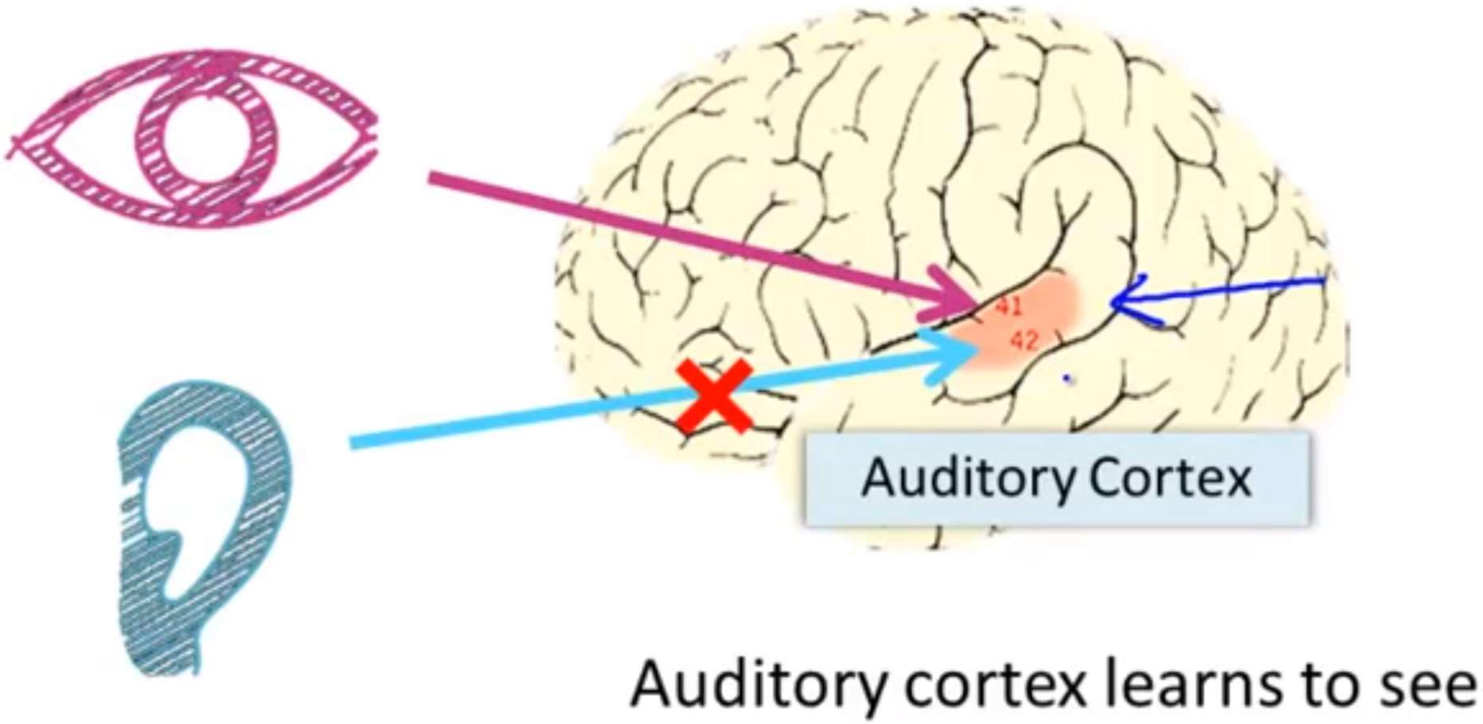

Color
Oranges
Lemons
Shape


Color
Oranges
New data point
Lemons
Shape


Pre-deep learning era
Feature representation
Compute hand-crafted image features
Train machine learning model
x_2
x_1
How to get such a model?

Neural Network

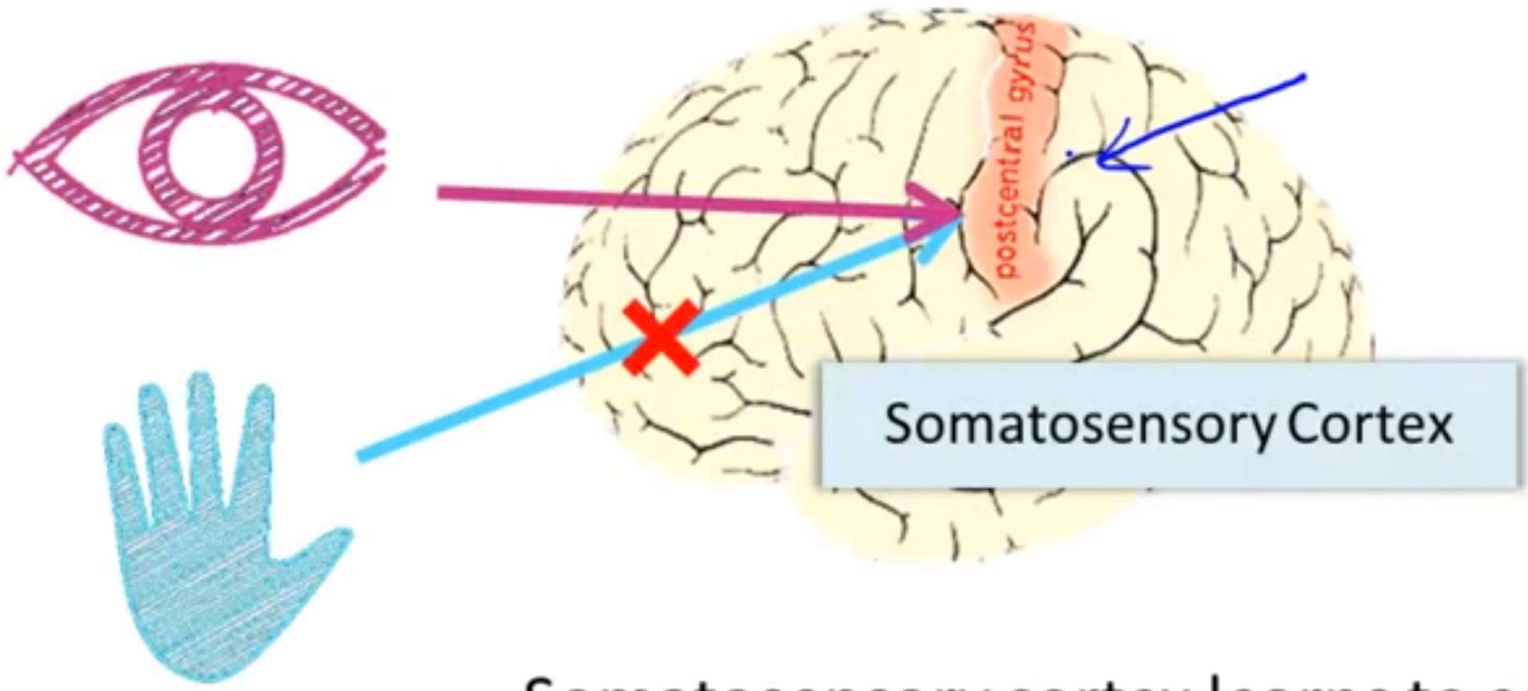
The "one learning algorithm" hypothesis



[Roe et al. 1992]

Slide credit: Andrew Ng

The "one learning algorithm" hypothesis



Somatosensory cortex learns to see

[Metin and Frost 1989]

Slide credit: Andrew Ng

Sensor representations in the brain

Seeing with your tongue

Human echolocation (sonar)

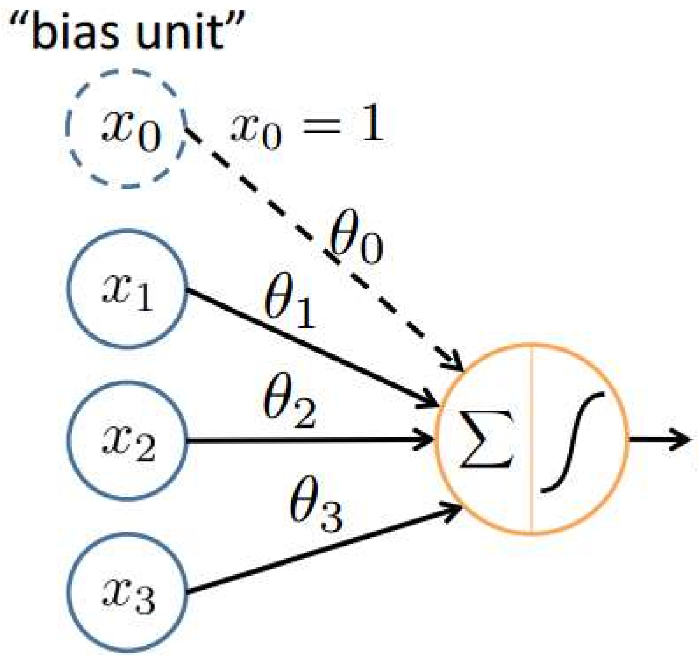
Haptic belt: Direction sense

Implanting a 3rd eye

[BrainPort; Welsh & Blasch, 1997; Nagel et al., 2005; Constantine-Paton & Law, 2009]

Slide credit: Andrew Ng

Neural Network

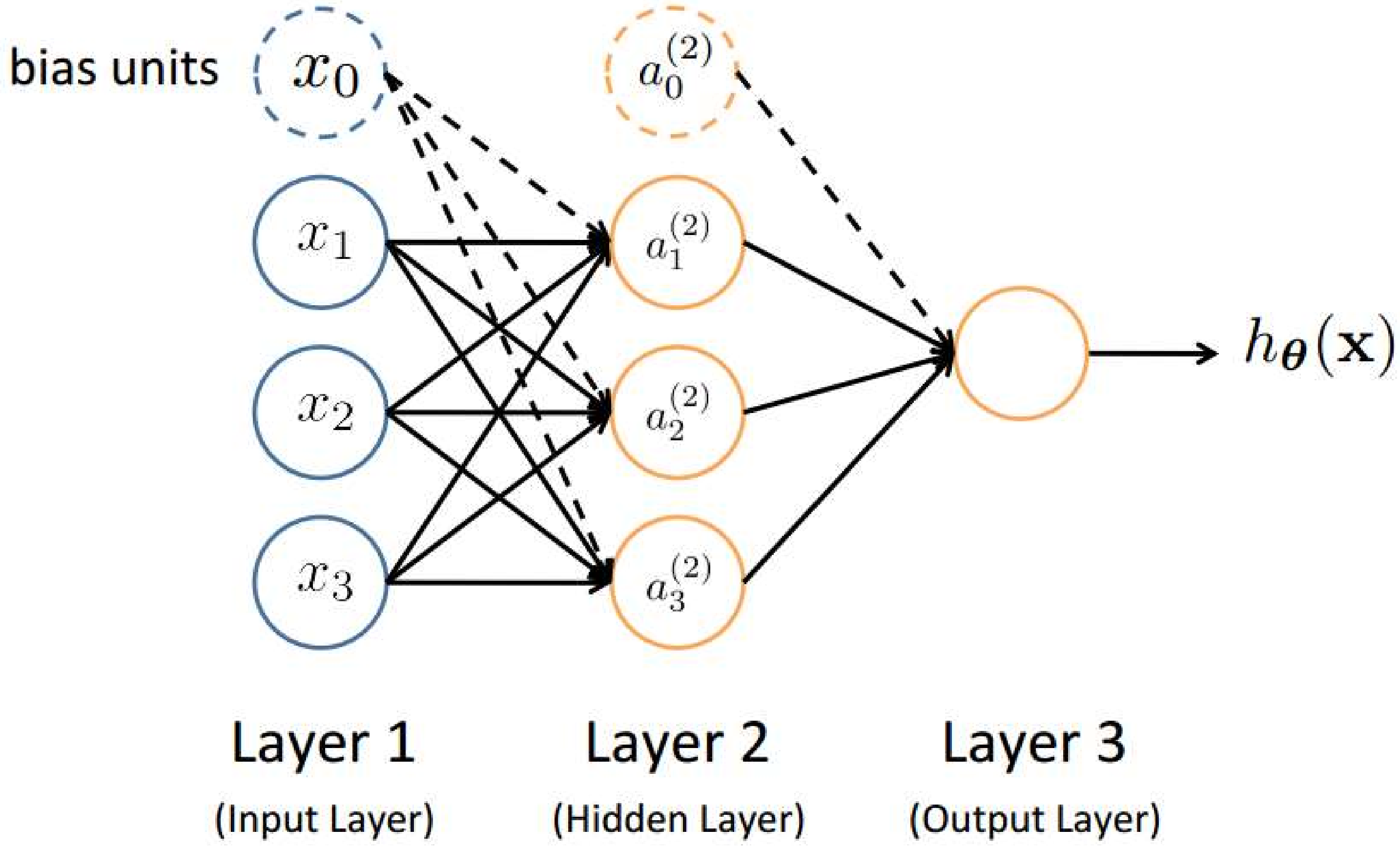


$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^\intercal \mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\intercal \mathbf{x}}}$$

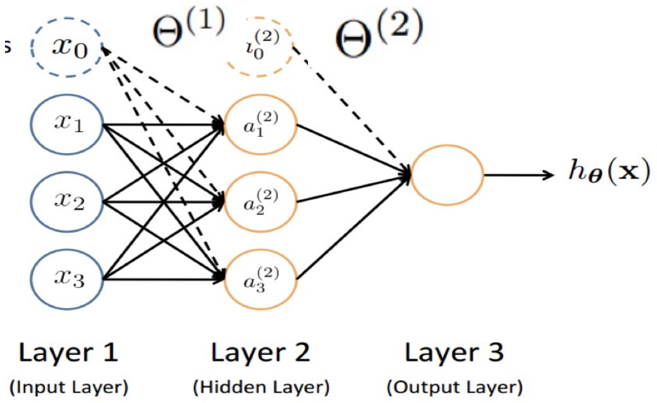
Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

Neural Network (feed forward)



Feed-Forward Process

- Input layer units are features
- Working forward through the network, the **input function** is applied to compute the input value
  - E.g., weighted sum of the input
- The **activation function** transforms this input function into a final value
  - Typically a **nonlinear** function (e.g, **sigmoid**)



$a_i^{(j)} =$ "activation" of unit i in layer j

$\Theta^{(j)} =$ weight matrix controlling function mapping from layer j to layer $j + 1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3\times 4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times 4}$$

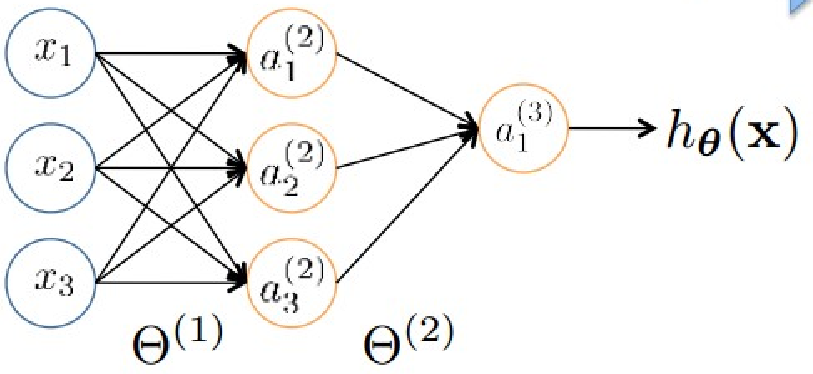
Vector Representation

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right) = g\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3\right) = g\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3\right) = g\left(z_3^{(2)}\right)$$

$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$



Feed-Forward Steps:

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

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Based on slide by Andrew Ng

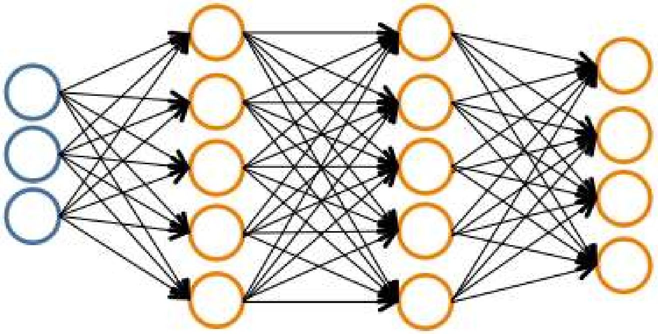
Can extend to multi-class

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

when car

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

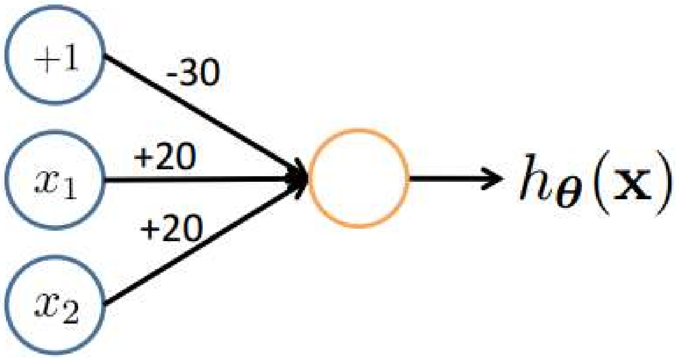
when truck

Why staged predictions?

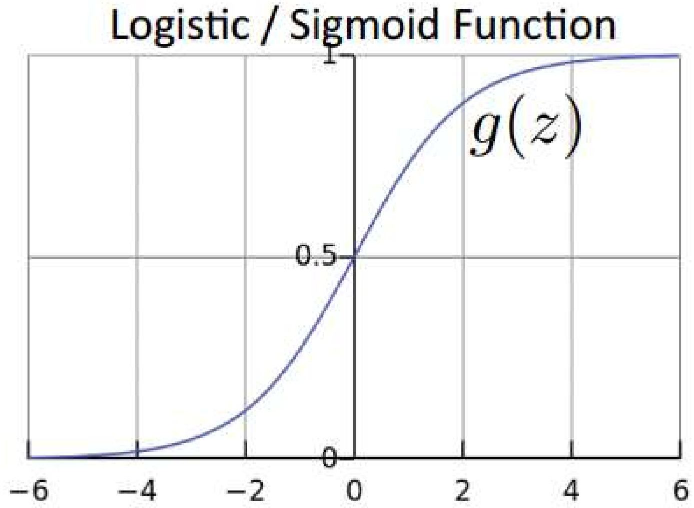
Simple example: AND

$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



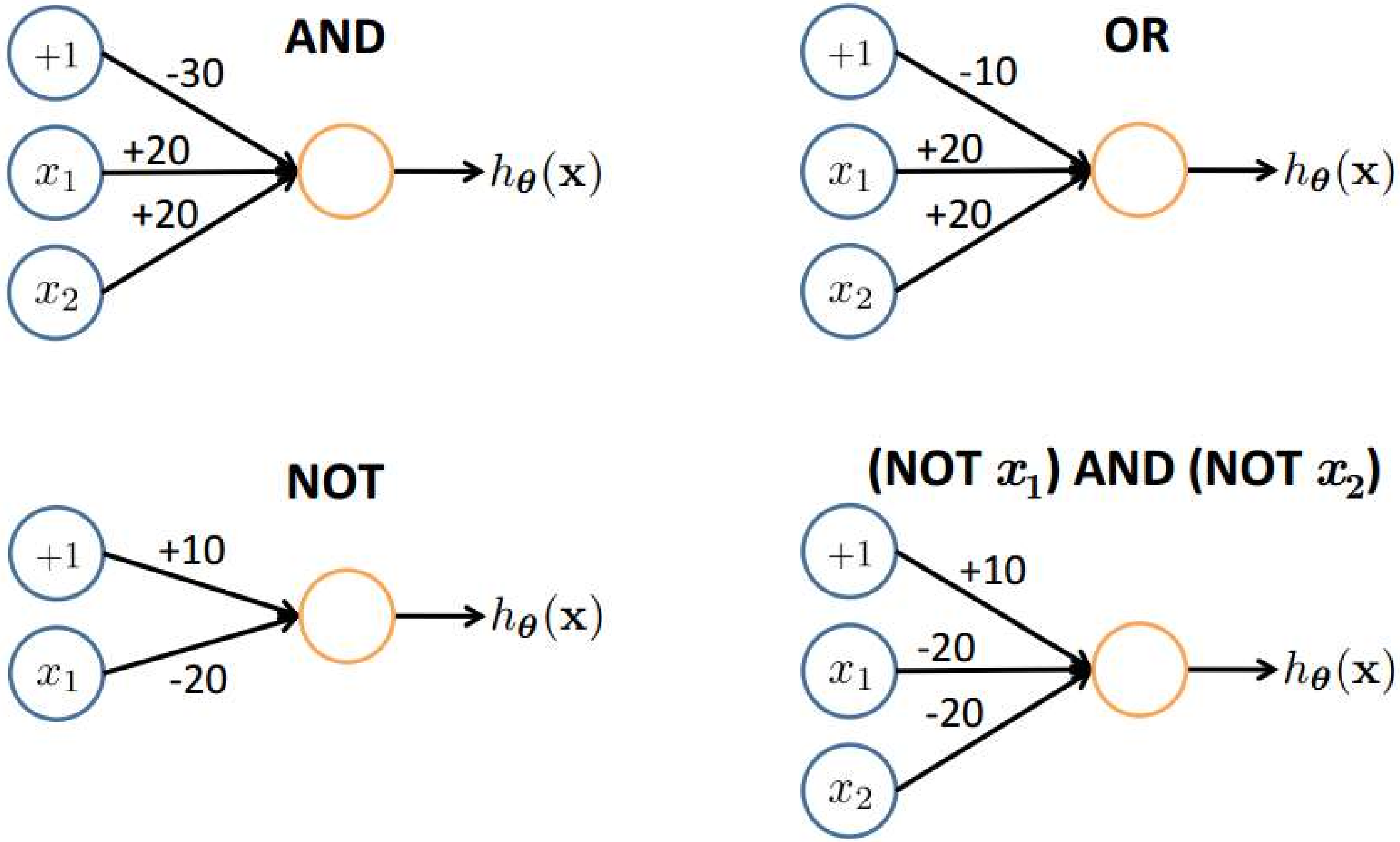
$$h_\Theta(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$



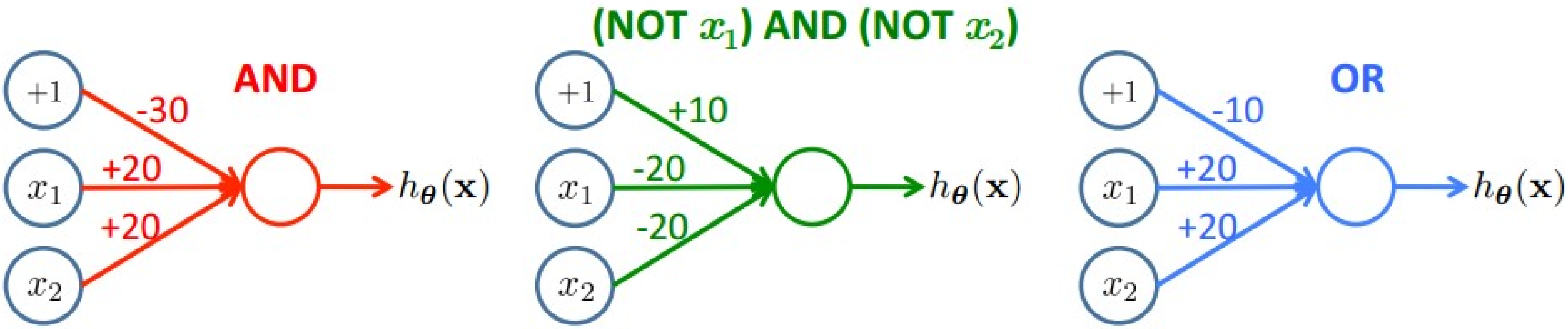
x_1	x_2	$h_\Theta(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

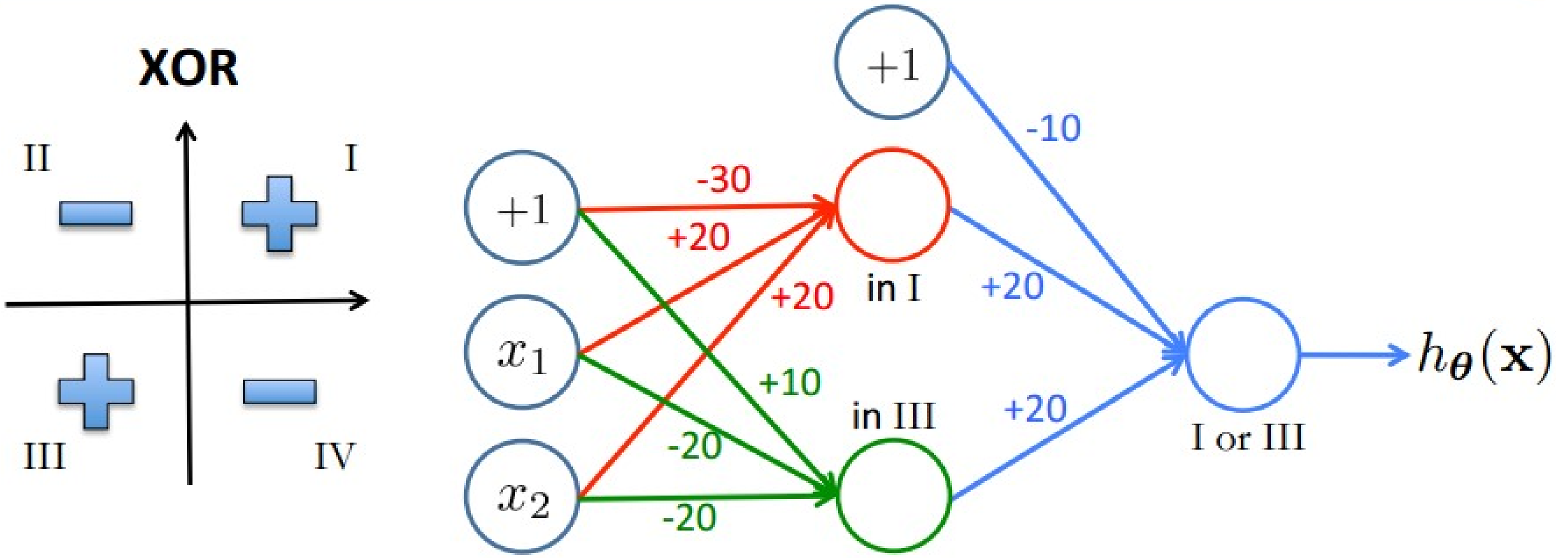
Based on slide and example by Andrew Ng

Representing Boolean Functions

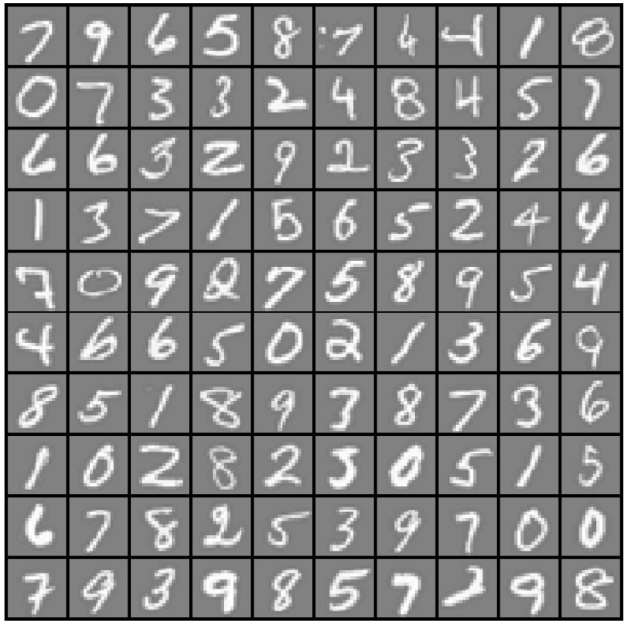


Combining Representations to Create Non--Linear Functions

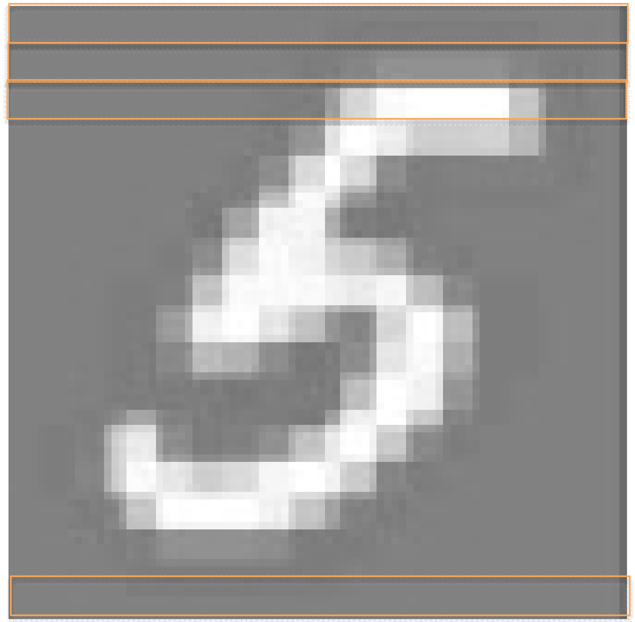




Layering Representations

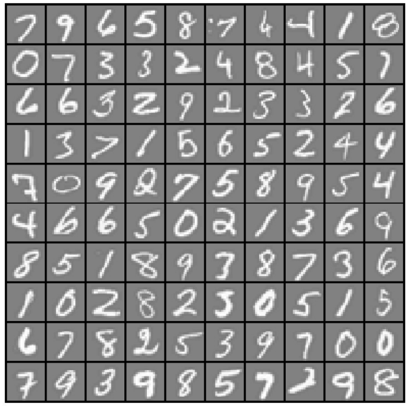


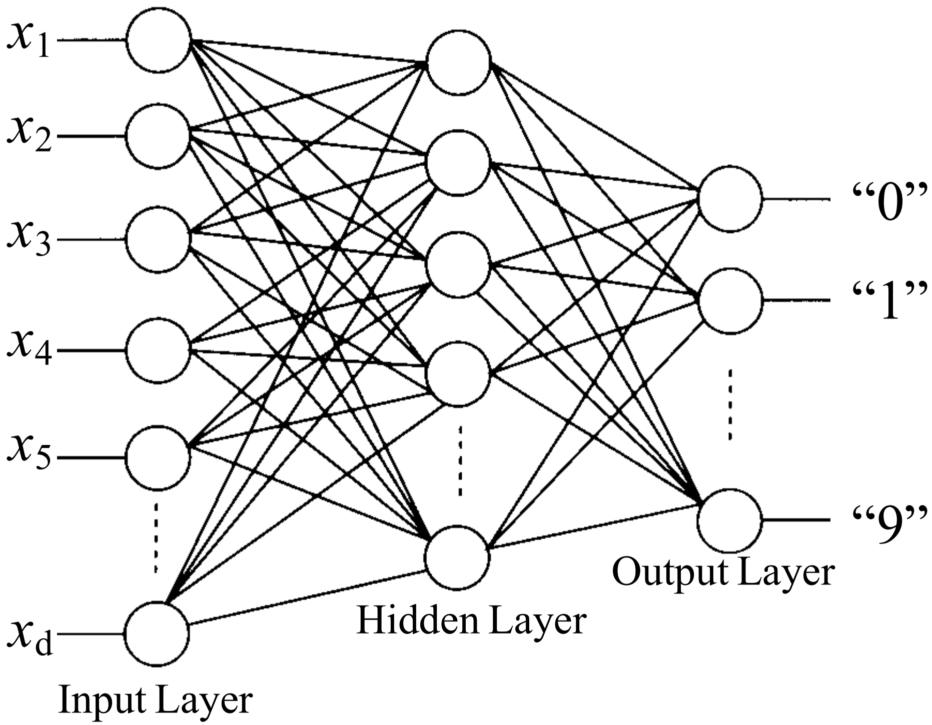
20 × 20 pixel images
$d = 400$ 10 classes

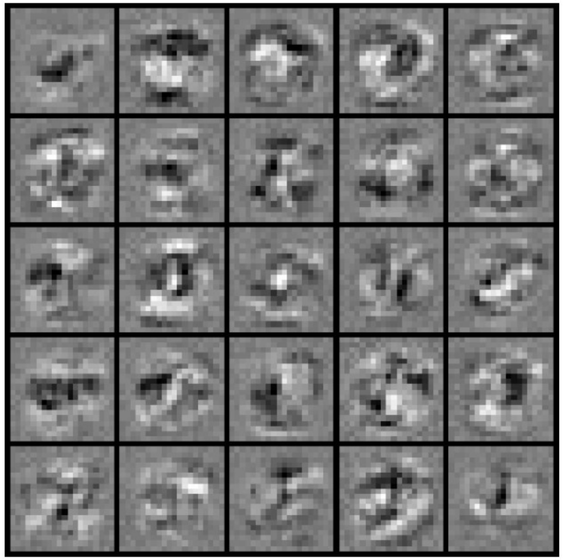


Each image is "unrolled" into a vector x of pixel intensities

Layering Representations







Visualization of Hidden Layer

Stochastic Sub-gradient Descent

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$

Initialize $\boldsymbol{w} \leftarrow \mathbf{0} \in \mathbb{R}^n$

For epoch $1 \ldots T$:

For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:

Update $w \leftarrow w - \eta \nabla f(\theta)$

- Return θ

Recap: Logistic regression

$$\min_{\boldsymbol{\theta}} \quad \frac{\lambda}{2n}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{n}\sum_i \log\left(1 + e^{-\mathrm{y_i}(\boldsymbol{\theta}^\mathrm{T}\mathbf{x}_\mathrm{i})}\right)$$

Let $\mathrm{h}_\theta(x_i) = 1/(1 + e^{-\theta^T x_i})$ (probability $y = 1$ given x_i)

$$\frac{\lambda}{2n}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{n}\sum_i \mathrm{y_i} \log\left(h_\theta(x_i)\right) + (1 - y_i)(\log\left(1 - h_\theta(x_i)\right)$$

Cost Function

$$f(\theta) = J(\theta) + g(\theta), \quad g(\theta) = \gamma\, \theta^T \theta$$

Logistic Regression:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log h_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1-y_i)\log\left(1-h_{\boldsymbol{\theta}}(\mathbf{x}_i)\right)\right] + \frac{\lambda}{2n}\sum_{j=1}^{d}\theta_j^2$$

Neural Network:

$$h_\Theta \in \mathbb{R}^K \qquad (h_\Theta(\mathbf{x}))_i = i^{th}\text{output}$$

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log\left(h_\Theta(\mathbf{x}_i)\right)_k + (1-y_{ik})\log\left(1-(h_\Theta(\mathbf{x}_i))_k\right)\right]$$
$$+ \frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

k^{th} class:	true, predicted
not k^{th} class:	true, predicted

Based on slide by Andrew Ng

Optimizing the Neural Network

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(h_\Theta(\mathbf{x}_i))_k + (1-y_{ik})\log\Big(1-(h_\Theta(\mathbf{x}_i))_k\Big)\right]$$
$$+\frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

Solve via: $\min_{\Theta} J(\Theta)$

$J(\Theta)$ is not convex, so GD on a neural net yields a local optimum

- But, tends to work well in practice

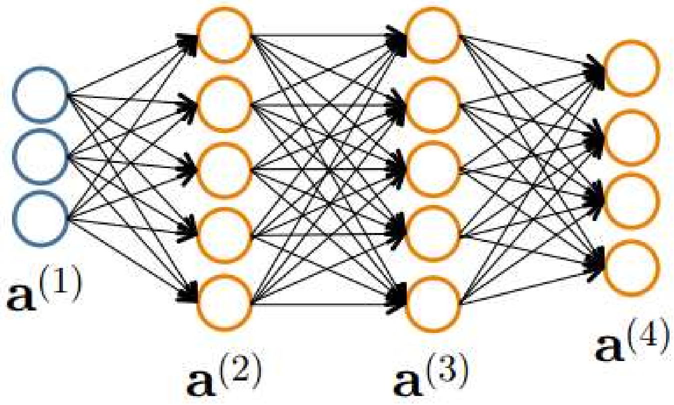
Need code to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta)$

Based on slide by Andrew Ng

Forward Propagation

- Given one labeled training instance $(\mathbf{x},\ y)$:



<u>Forward Propagation</u>

- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $\mathrm{a}_0{}^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $\mathrm{a}_0{}^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = \mathrm{h}_\Theta(\mathbf{x}) = g(\mathbf{z}^{(4)})$

Based on slide by Andrew Ng

Online examples

- https://www.w3schools.com/ai/ai_perceptrons.asp

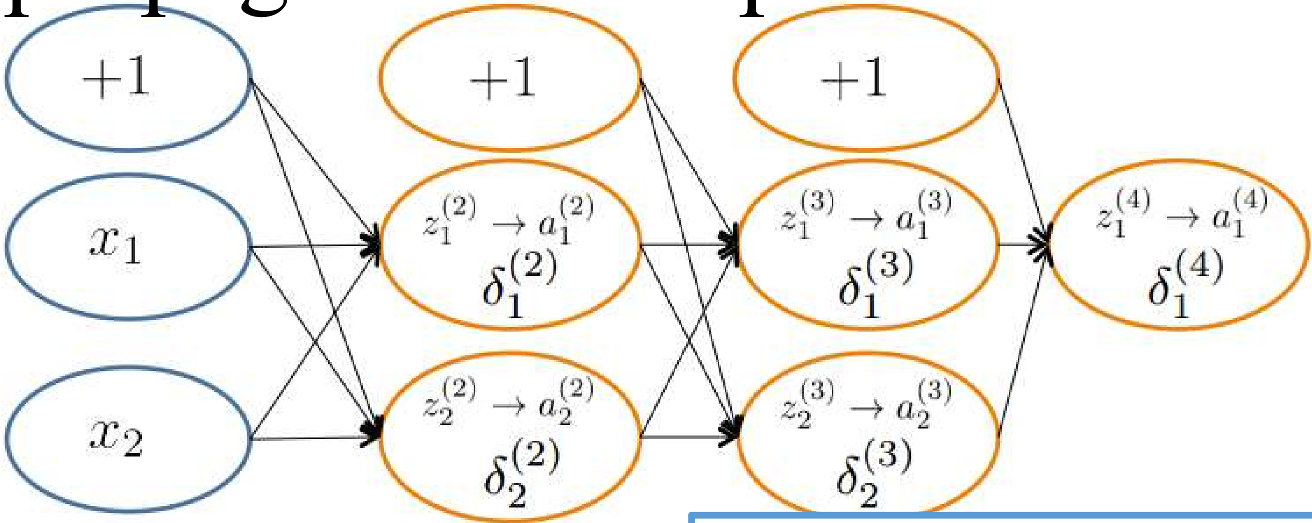
Next time

- Can we make a model for image classification?
- Can we measure the quality of a certain model?
- How can we improve this by learning from data?

- Middle test: 18th Oct 2022

- Final project

Backpropagation: Compute Gradient



$$\boxed{\frac{d}{dt} f(g(t)) = f'(g(t))g'(t) = \frac{df}{dg} \cdot \frac{dg}{dt}}$$

$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

$$\text{where } \text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$$

Based on slide by Andrew Ng