# Computer Vision

CS308 Autumn

Feng Zheng

SUSTech CS Vision Intelligence


南方科技大学

SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY


# Content

- Brief Review
- Neural Networks
- Convolutional Neural Networks

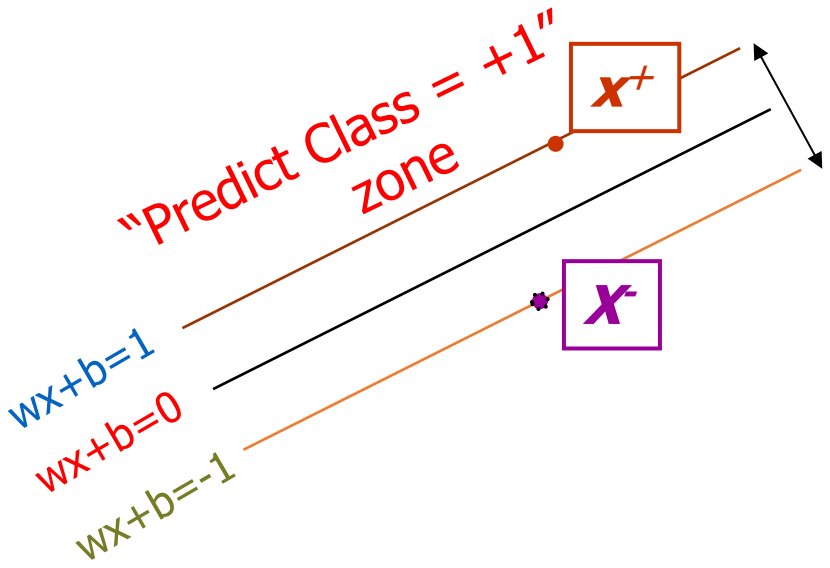
# Brief Review

# Linear SVM Mathematically

What we know:

- $w \cdot x^+ + b = +1$
- $w \cdot x^- + b = -1$
- $w \cdot (x^+ - x^-) = 2$

$$M = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

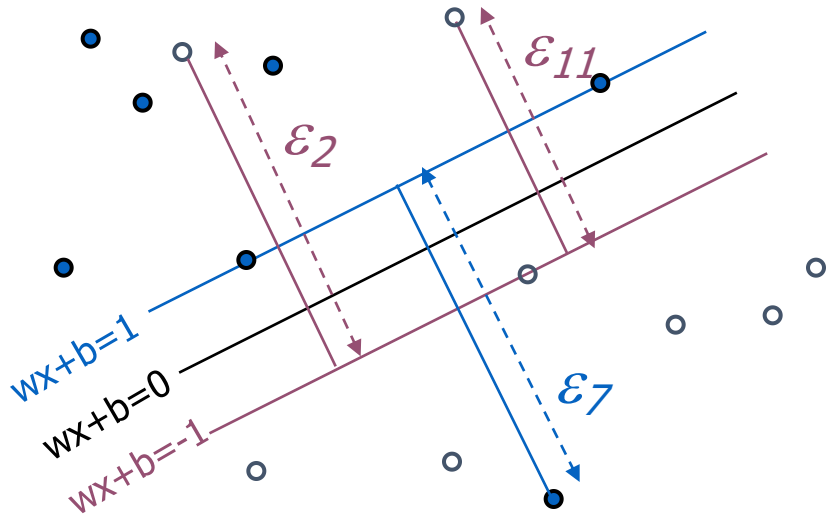



**M**=Margin Width

# Soft Margin Classification

- *Slack variables* $\xi_i$ can be added to allow misclassification of difficult or noisy examples.
- What should our quadratic optimization criterion be?

$$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$




# Non-linear SVMs Mathematically

- Dual problem formulation:

**Find $\alpha_1 \ldots \alpha_N$ such that**
**$Q(\alpha) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j K(x_i, x_j)$ is maximized and**
**(1) $\Sigma\alpha_i y_i = 0$**
**(2) $\alpha_i \geq 0$ for all $\alpha_i$**

- The solution is:

$$f(x) = \Sigma\alpha_i y_i K(x_i, x_j) + b$$

- Optimization techniques for finding $\alpha_i$'s remain the same!

# Neural Networks


# Logistic Regression

- **Data:** Inputs are continuous vectors of length $K$. Outputs are discrete $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$ where $\mathbf{x} \in \mathbb{R}^K$ and $y \in \{0, 1\}$
- **Model:** Logistic function applied to dot product of parameters with input vector

$$p_{\boldsymbol{\theta}}(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$

  - **Prediction:** Output is the most probable class.

$$\hat{y} = \underset{y \in \{0,1\}}{\operatorname{argmax}}\, p_{\boldsymbol{\theta}}(y|\mathbf{x})$$

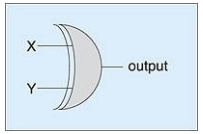
- **Learning:** Finds the parameters that minimize some objective function.

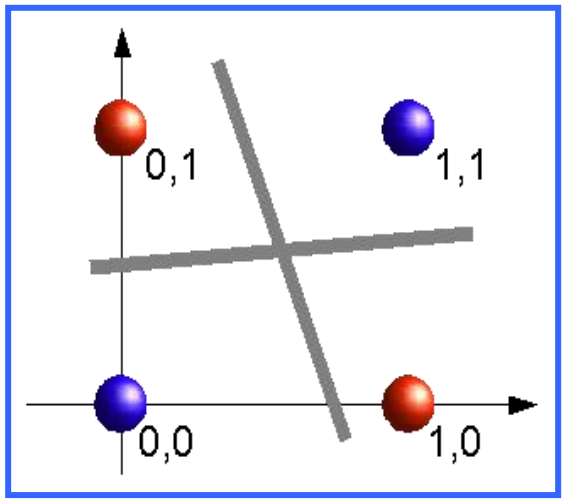
$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$$

# Learning Highly Non-Linear Functions

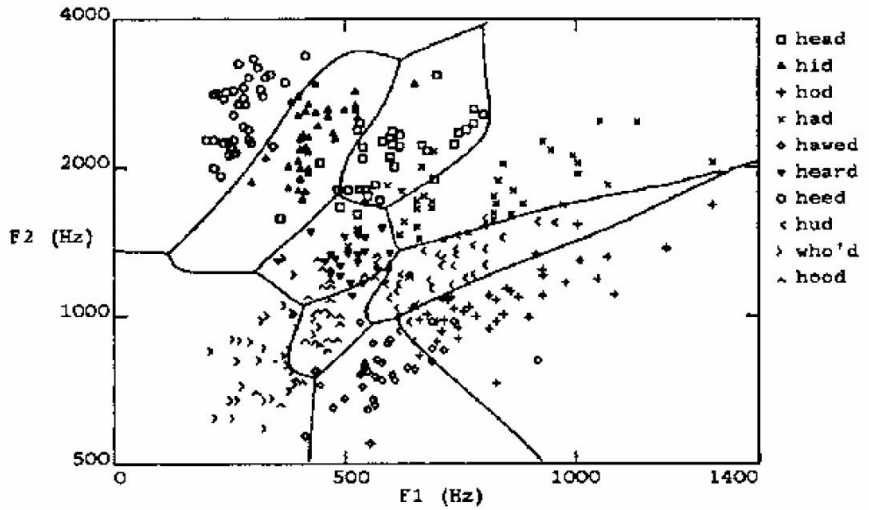
- $f: X \rightarrow Y$
  - $f$ might be non-linear function
  - $X$ (vector of) continuous and/or discrete variables
  - $Y$ (vector of) continuous and/or discrete variables

The XOR gate



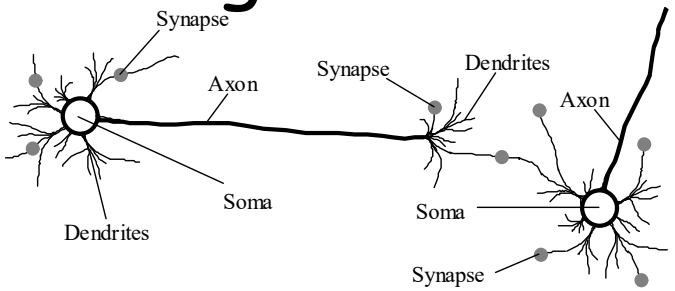



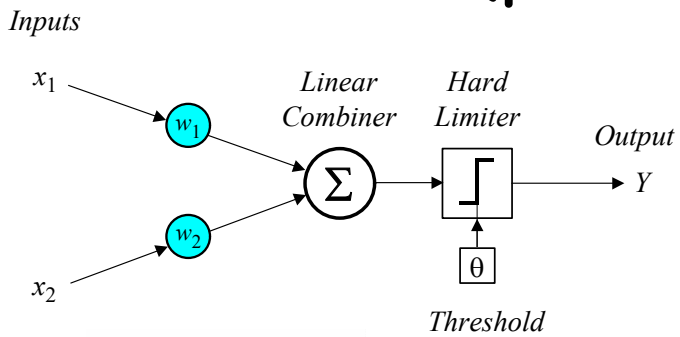

Speech recognition



# Perceptron and Neural Nets

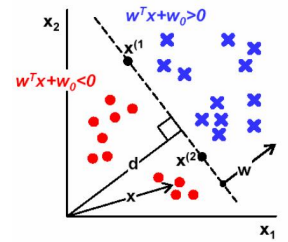
- From biological neuron to artificial neuron (perceptron)



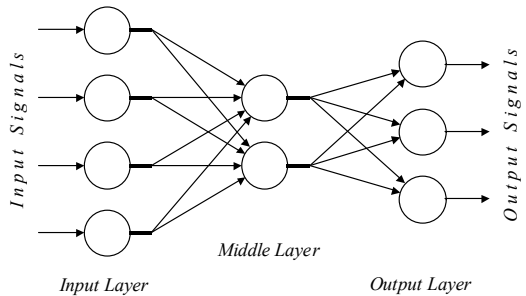




- Activation function

$$Z = \sum_{i=1}^{n} X_i W_i \qquad Y = \begin{cases} +1, \text{ if } Z \geq \omega_0 \\ -1, \text{ if } Z < \omega_0 \end{cases}$$

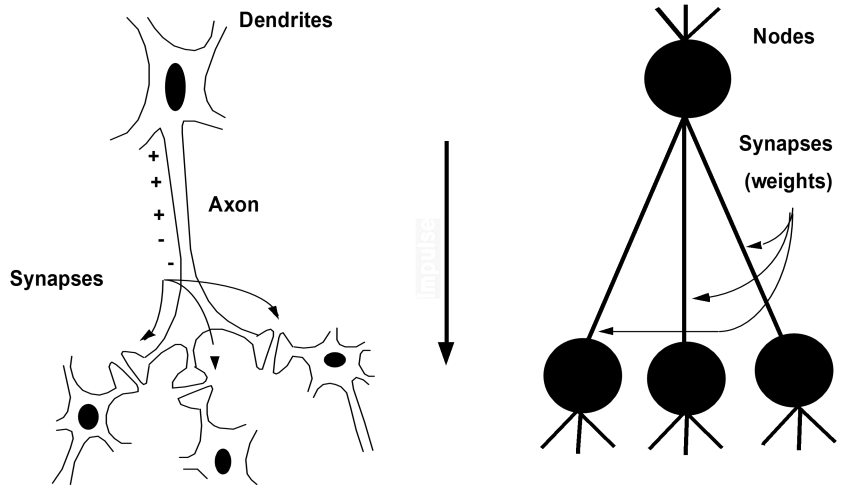



- Artificial neuron networks (Connectionist)
  - Supervised learning
  - Gradient descent



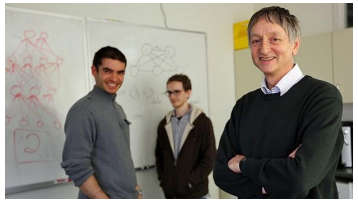
# Connectionist Models

- Consider humans:
  - Neuron switching time ~ 0.001 second
  - Number of neurons ~ $10^{10}$
  - Connections per neuron ~ $10^{4-5}$
  - Scene recognition time ~ 0.1 second
  - 100 inference steps doesn't seem like enough
    → much parallel computation
- Properties of artificial neural nets (ANN)
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed processes



# Motivation: Why is everyone talking about Deep Learning?

- Because a lot of money is invested in it…
  - DeepMind: Acquired by Google for **$400 million**
  - DNNResearch: **Three person startup** (including Geoff Hinton) acquired by Google for unknown price tag
  - Enlitic, Ersatz, MetaMind, Nervana, Skylab: Deep Learning startups commanding **millions of VC dollars**
  - **SENSETIME**: Xiaoou Tang, Xu Li
- Because it made the **front page** of the New York Times



MetaMind

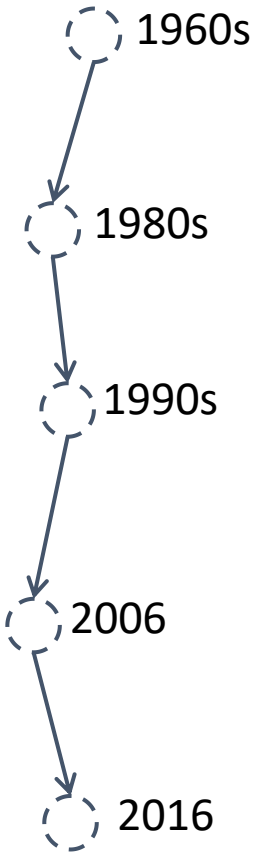
The New York Times

# Motivation: Why is everyone talking about Deep Learning?

- Deep learning:
  - Has won numerous pattern recognition competitions
  - Does so with minimal feature engineering
- This wasn't always the case!

Since 1980s: Form of models hasn't changed much, but lots of new tricks…

- More hidden units
- Better (online) optimization
- Faster computers (CPUs and GPUs)
- New nonlinear functions (ReLUs)



# Background: A Recipe for Machine Learning

- Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$$

*Face* *Face* *Not a face*



- Choose each of these:
  - Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

  - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

**Examples**: Linear regression, Logistic regression, Neural Network

**Examples**: Mean-squared error, Cross Entropy

# Background: A Recipe for Machine Learning

- Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

# of samples

- Train with SGD: (take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Learning rate

Gradients

**Backpropagation** can compute this gradient!

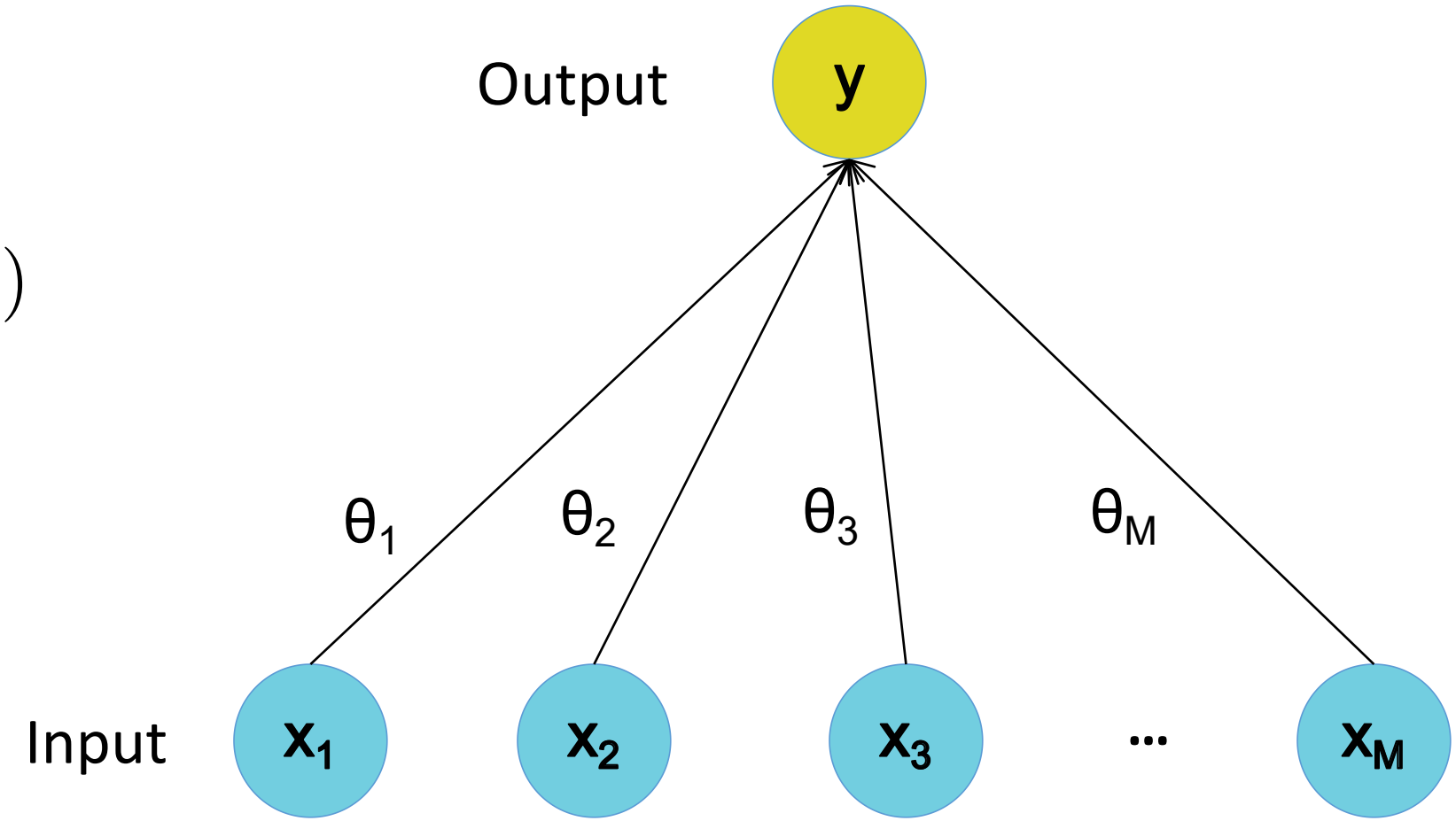
And it's a **special case of a more general algorithm** called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!

# Decision Functions: Linear Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = a$

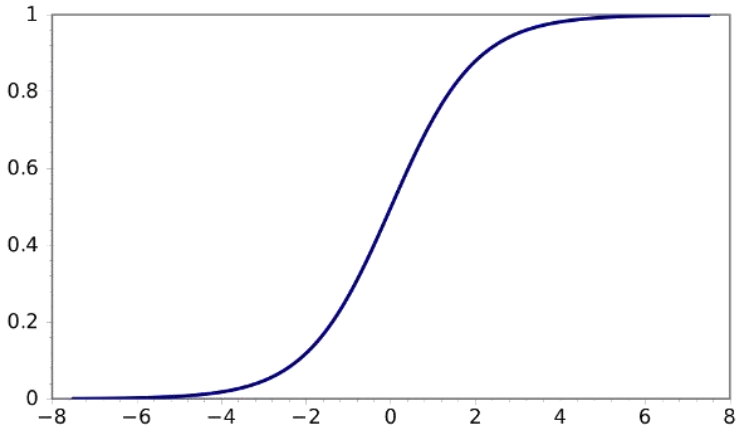
Linear

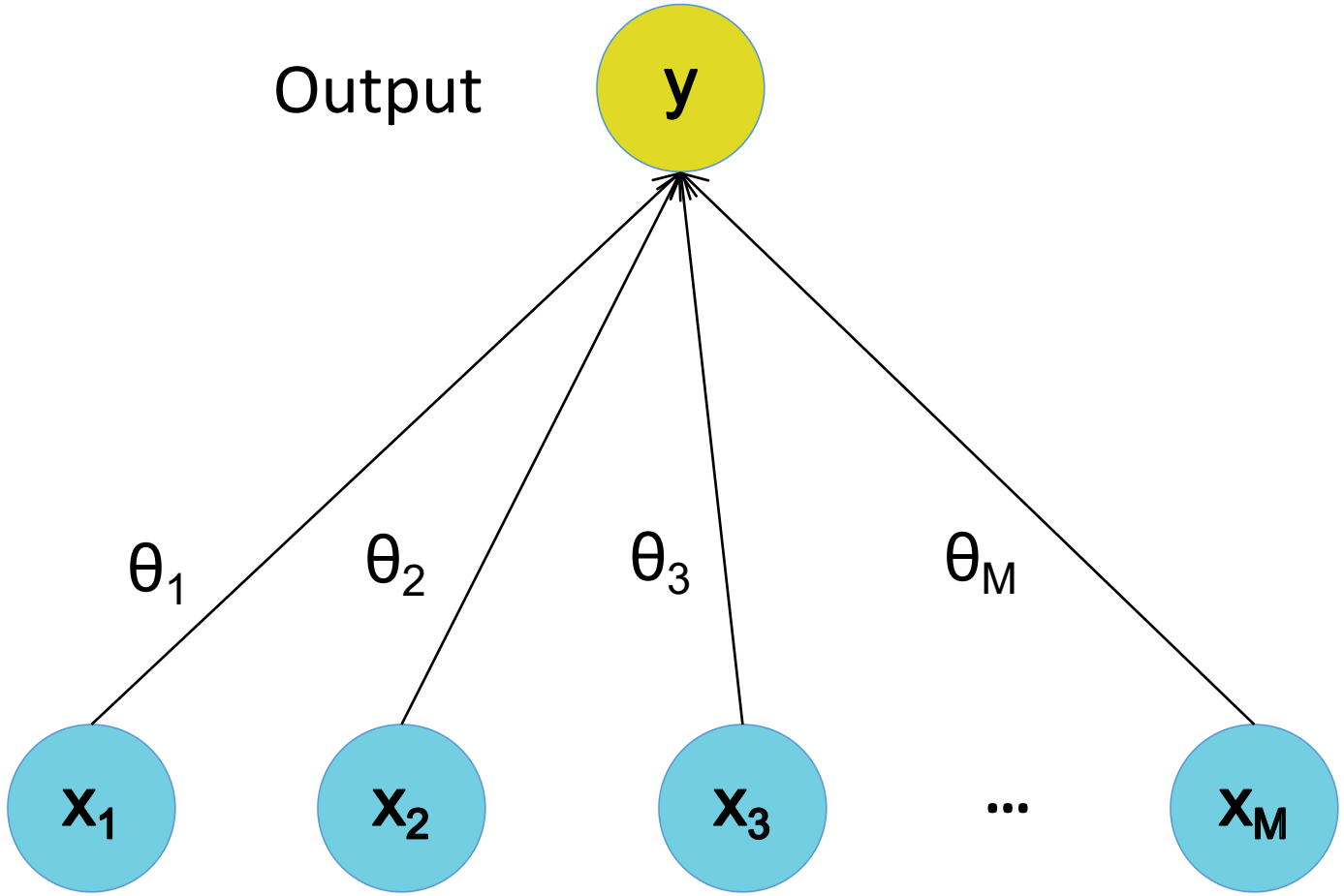


# Decision Functions: Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$

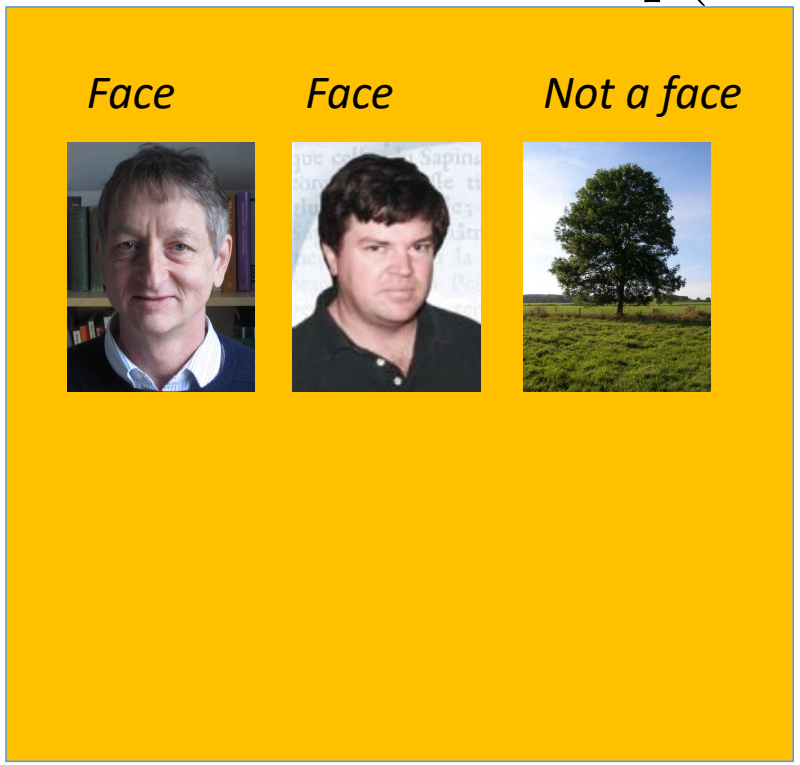


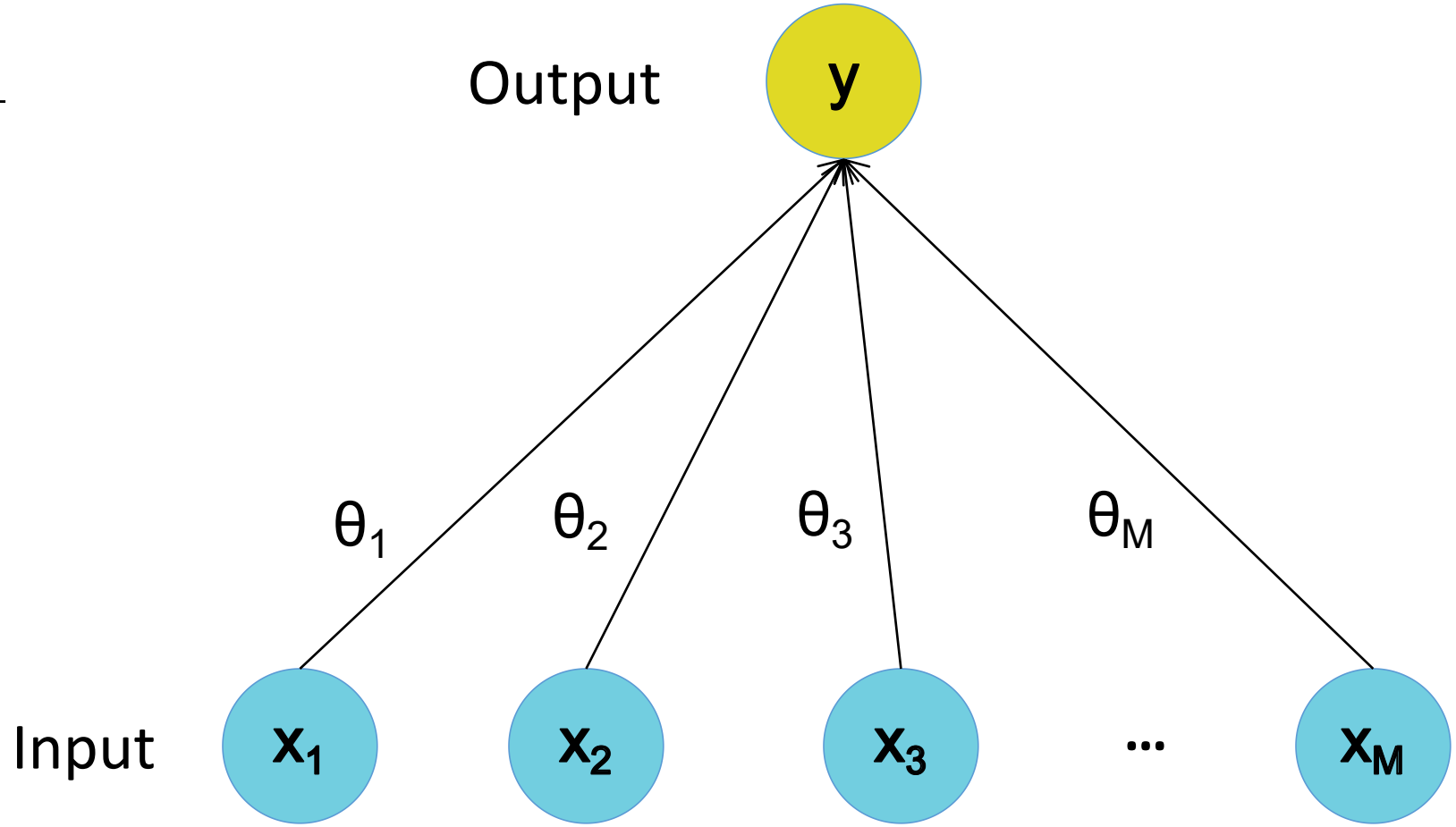



# Decision Functions: Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$

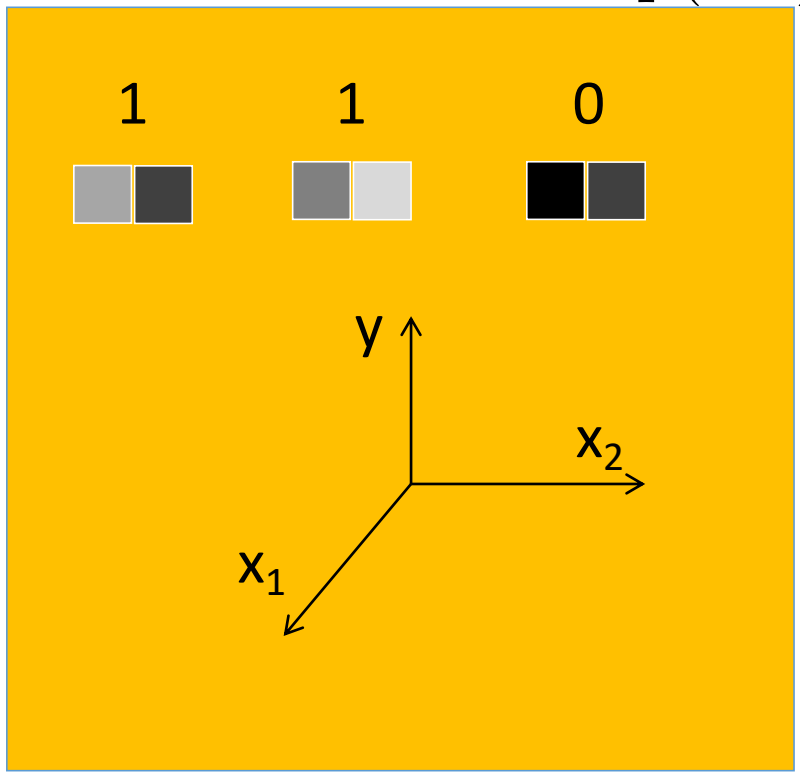


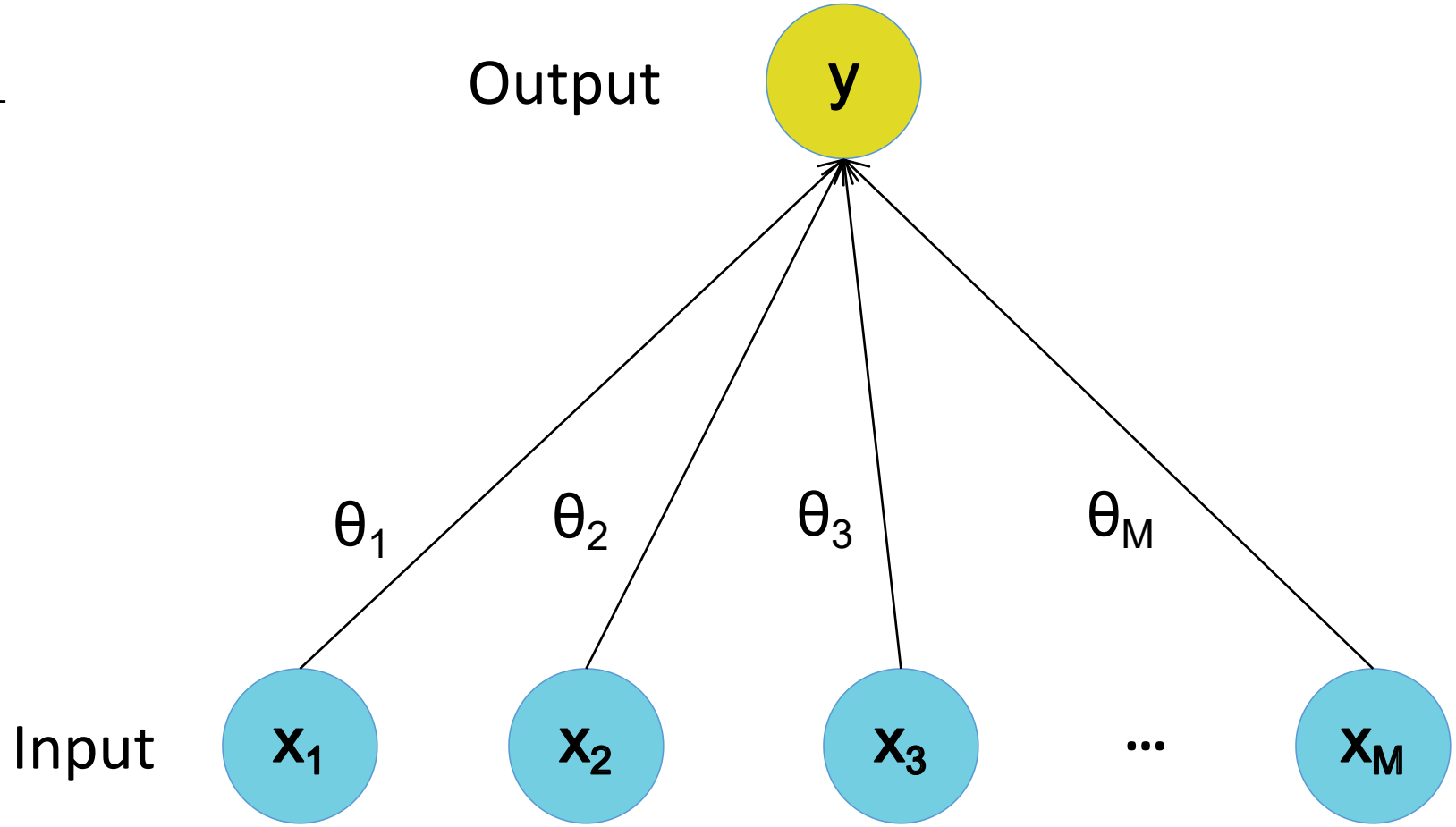



# Decision Functions: Logistic Regression

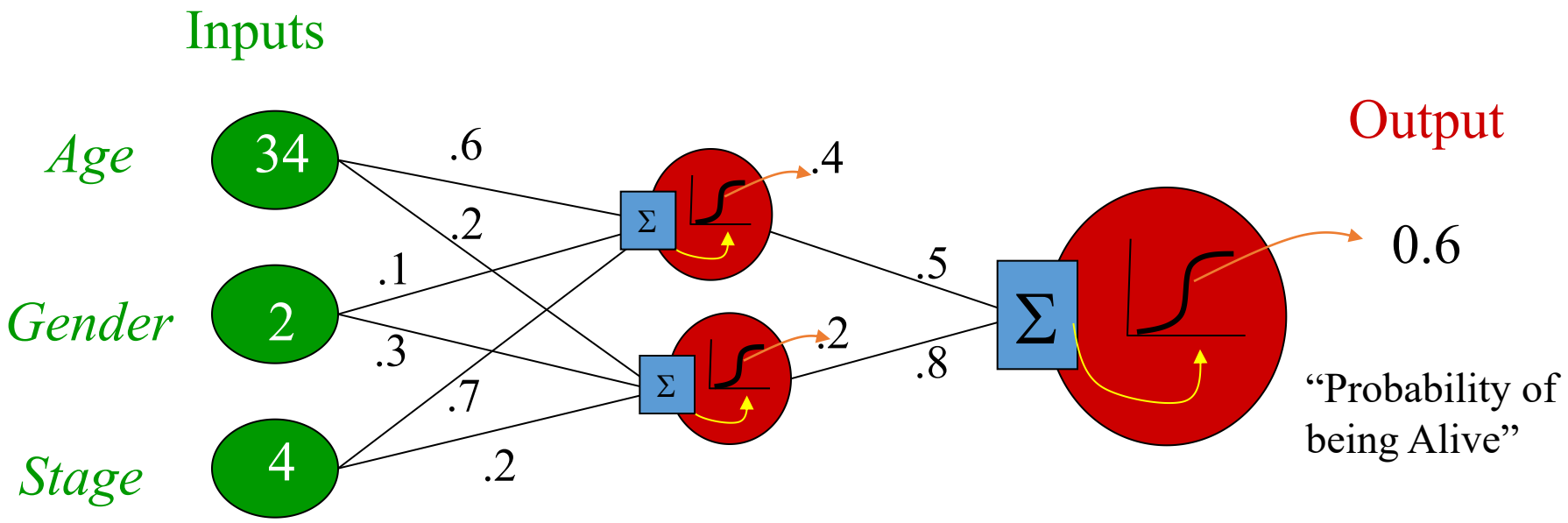
$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$



# Neural Network Model




*Independent variables*

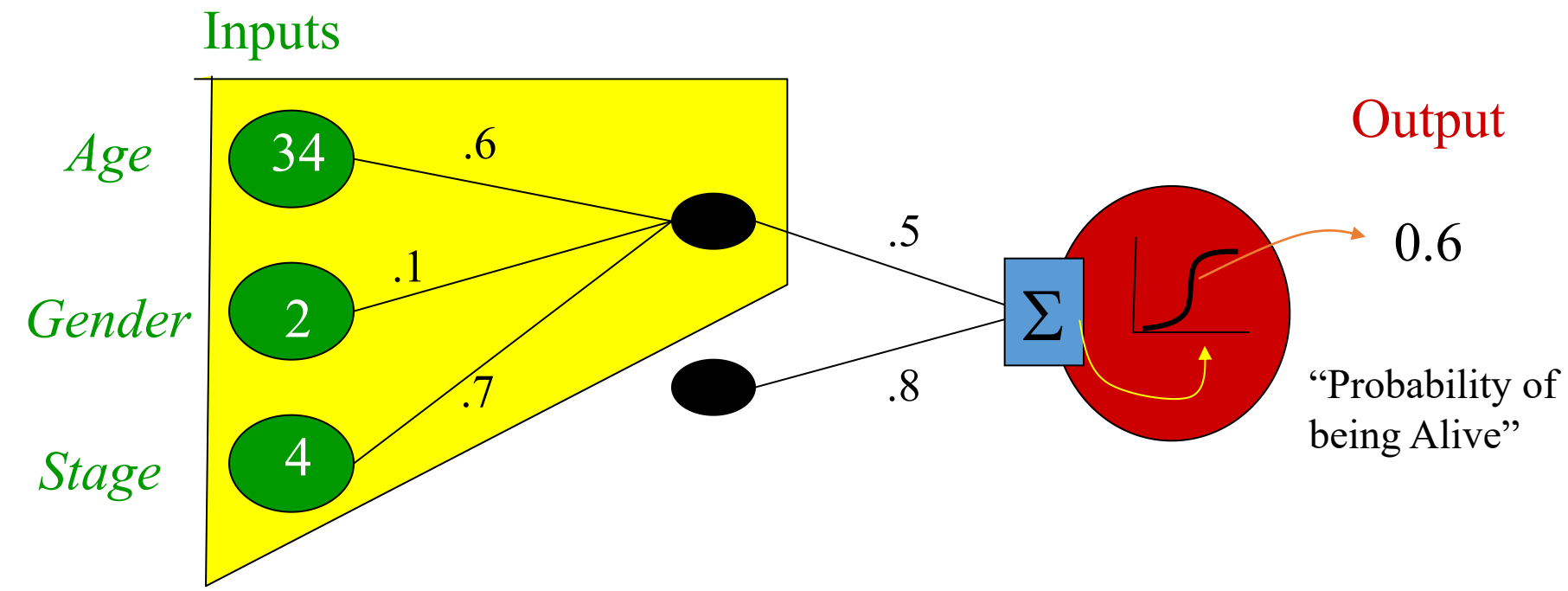
Weights

Hidden Layer

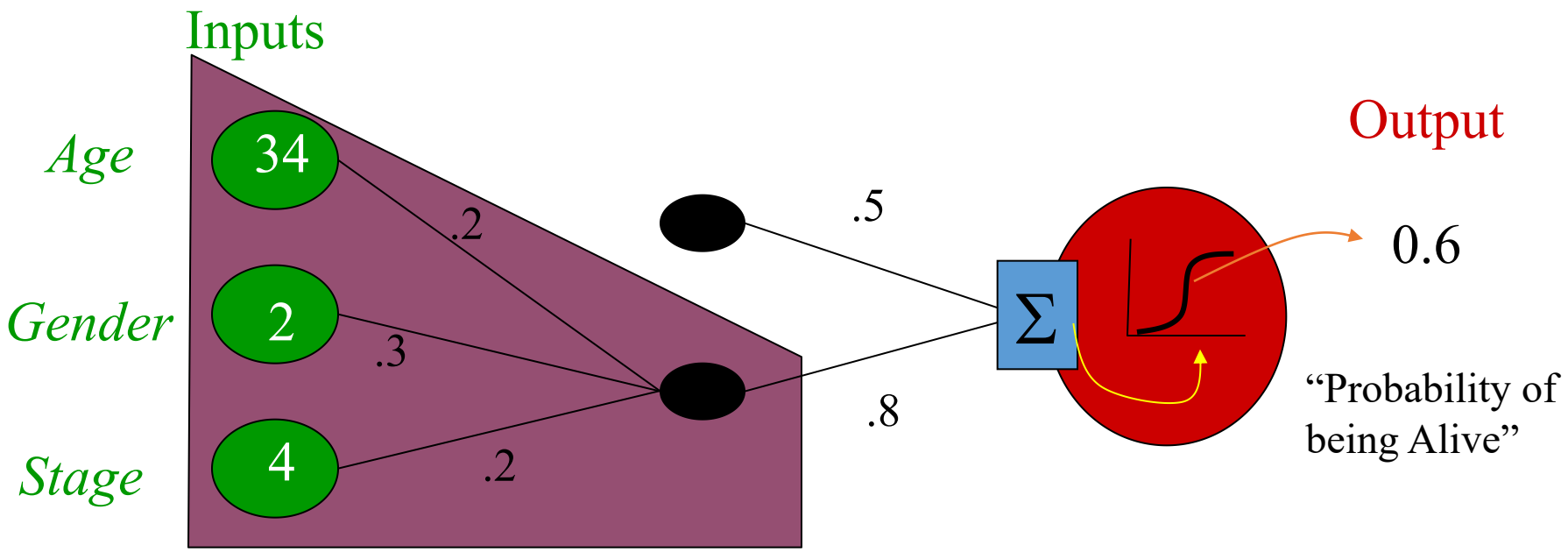
Weights

*Dependent variable Prediction*

# Neural Network Model



# Neural Network Model




*Independent variables*

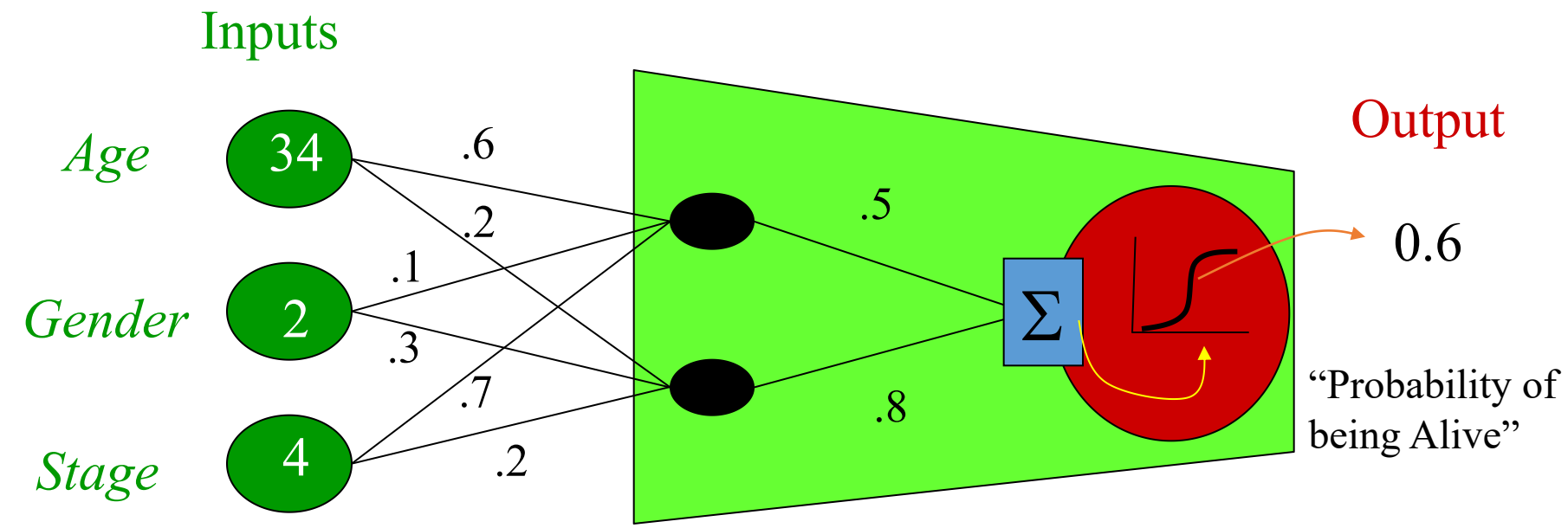
Weights

Hidden Layer

Weights

*Dependent variable Prediction*

# Neural Network Model




*Independent variables*

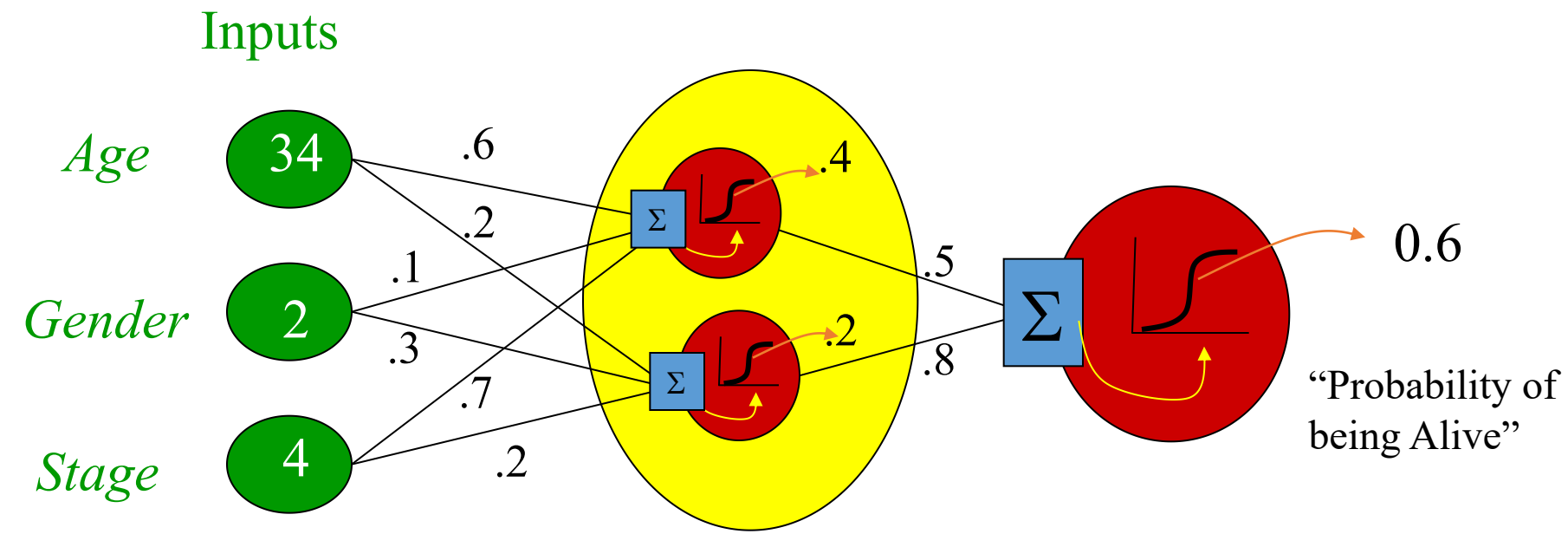
Weights

Hidden Layer

Weights

*Dependent variable Prediction*

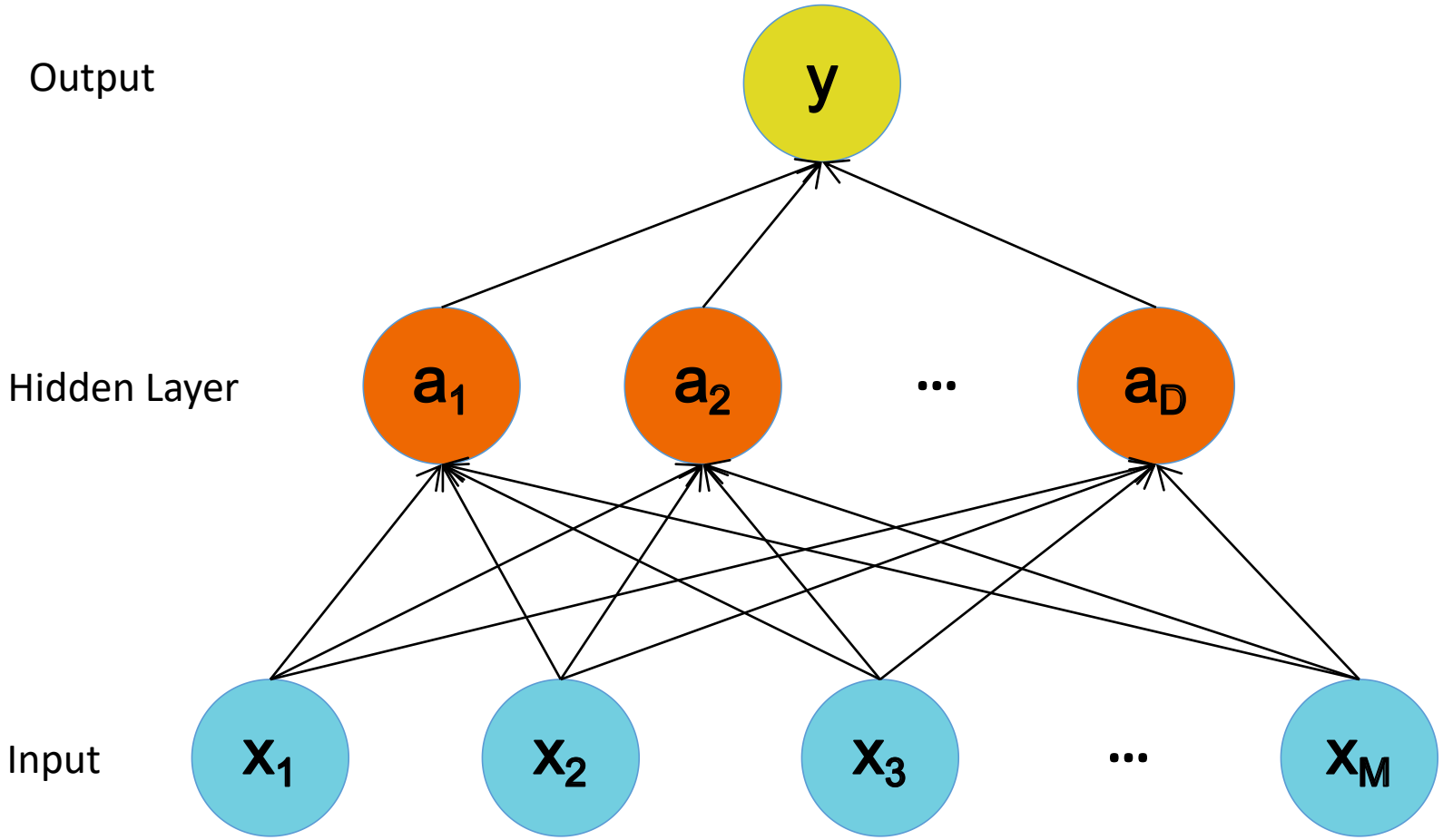
# Neural Network Model



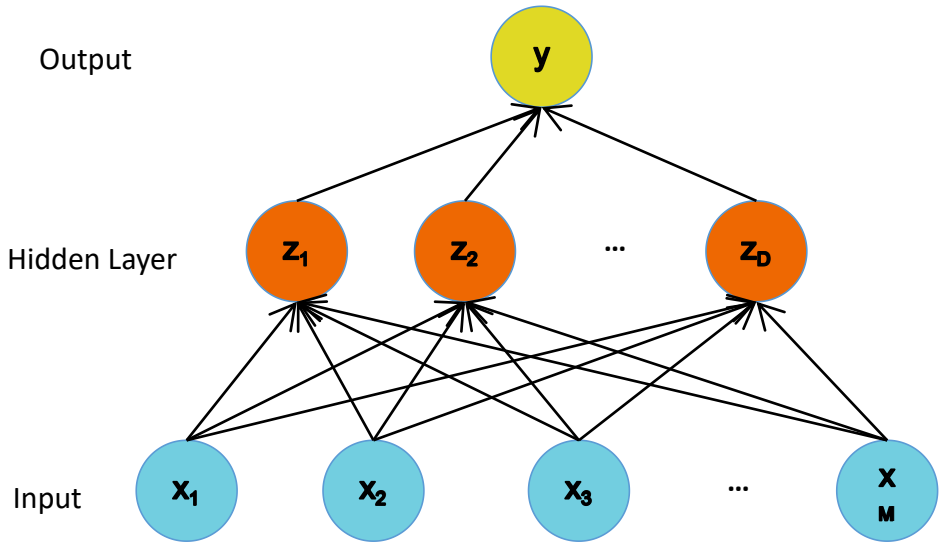

*Independent variables* | Weights | Hidden Layer | Weights | *Dependent variable Prediction*

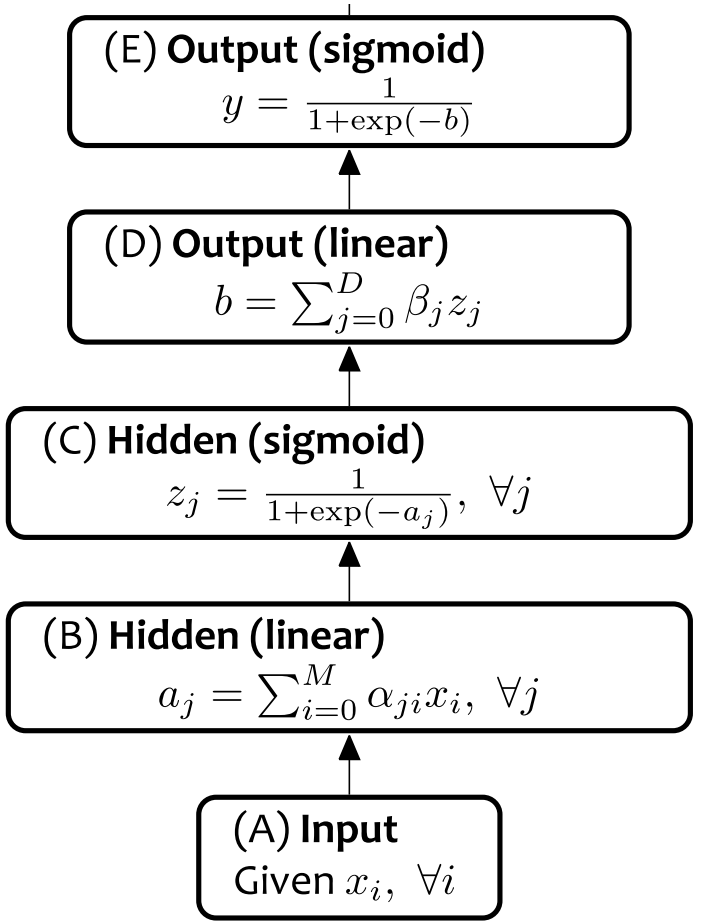
What happens if there is no non-linear activation functions?

# Decision Functions: Neural Network

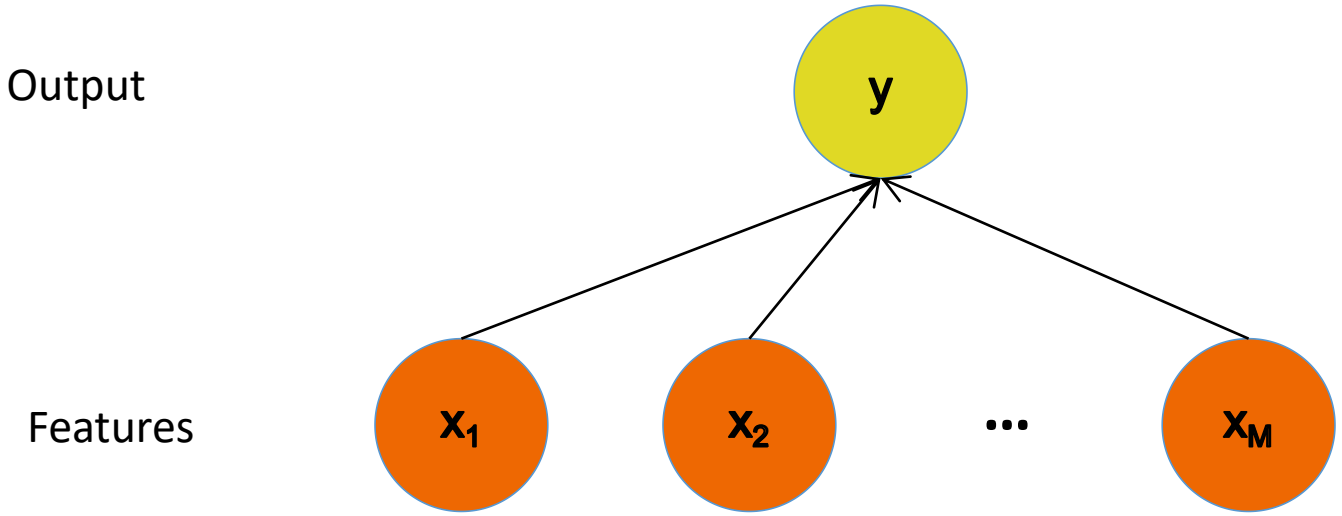


# Decision Functions: Neural Network

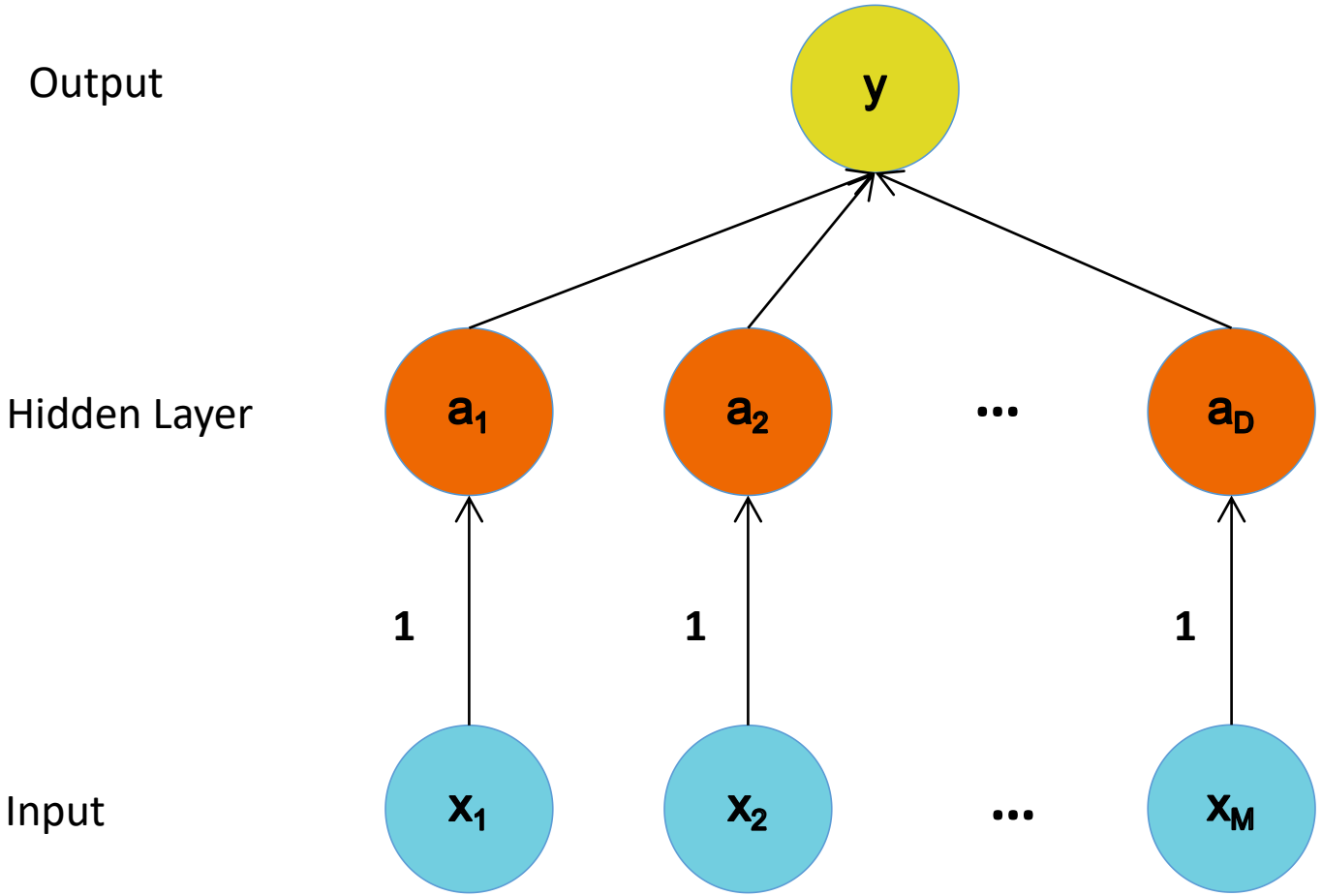


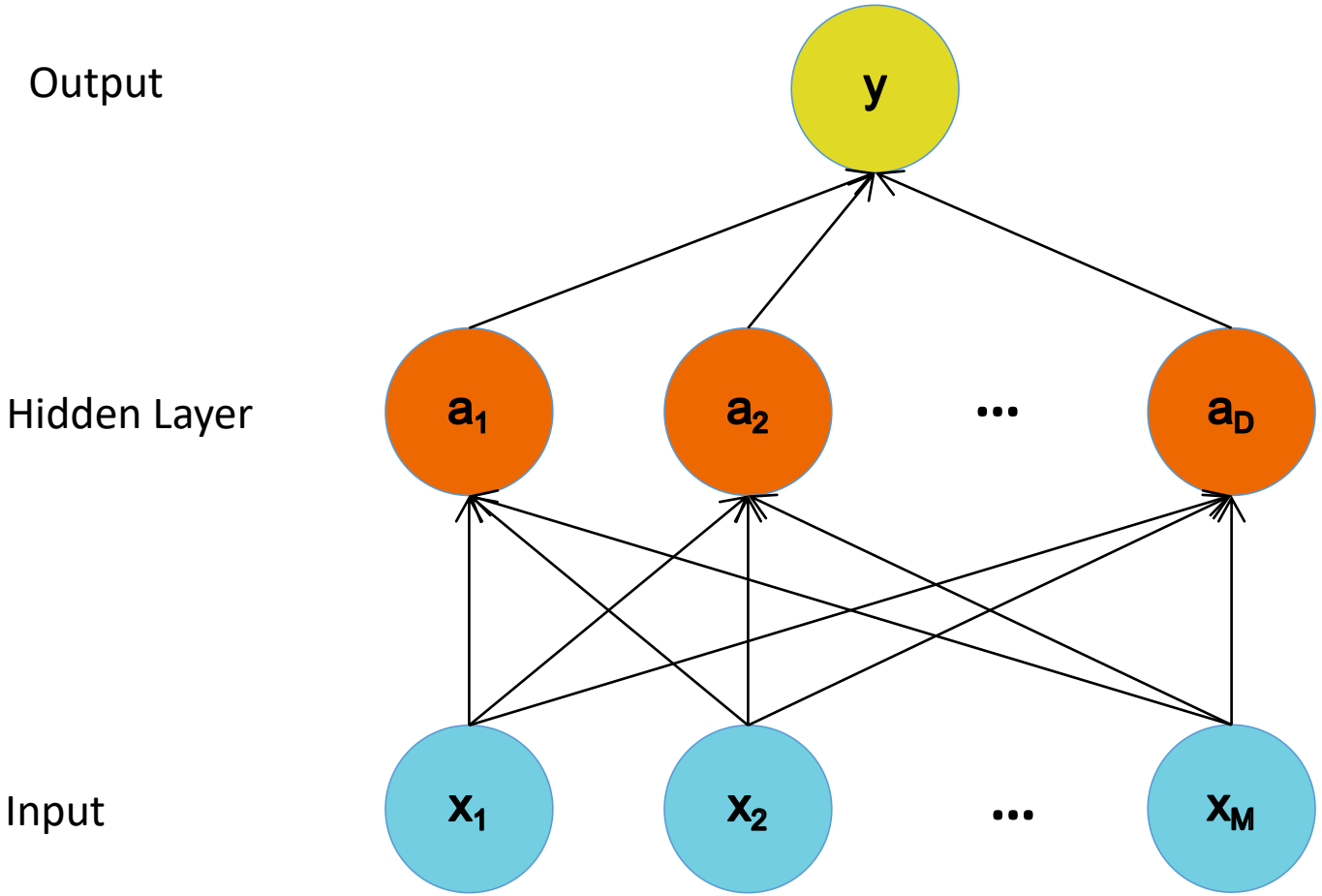
# Building a Neural Net



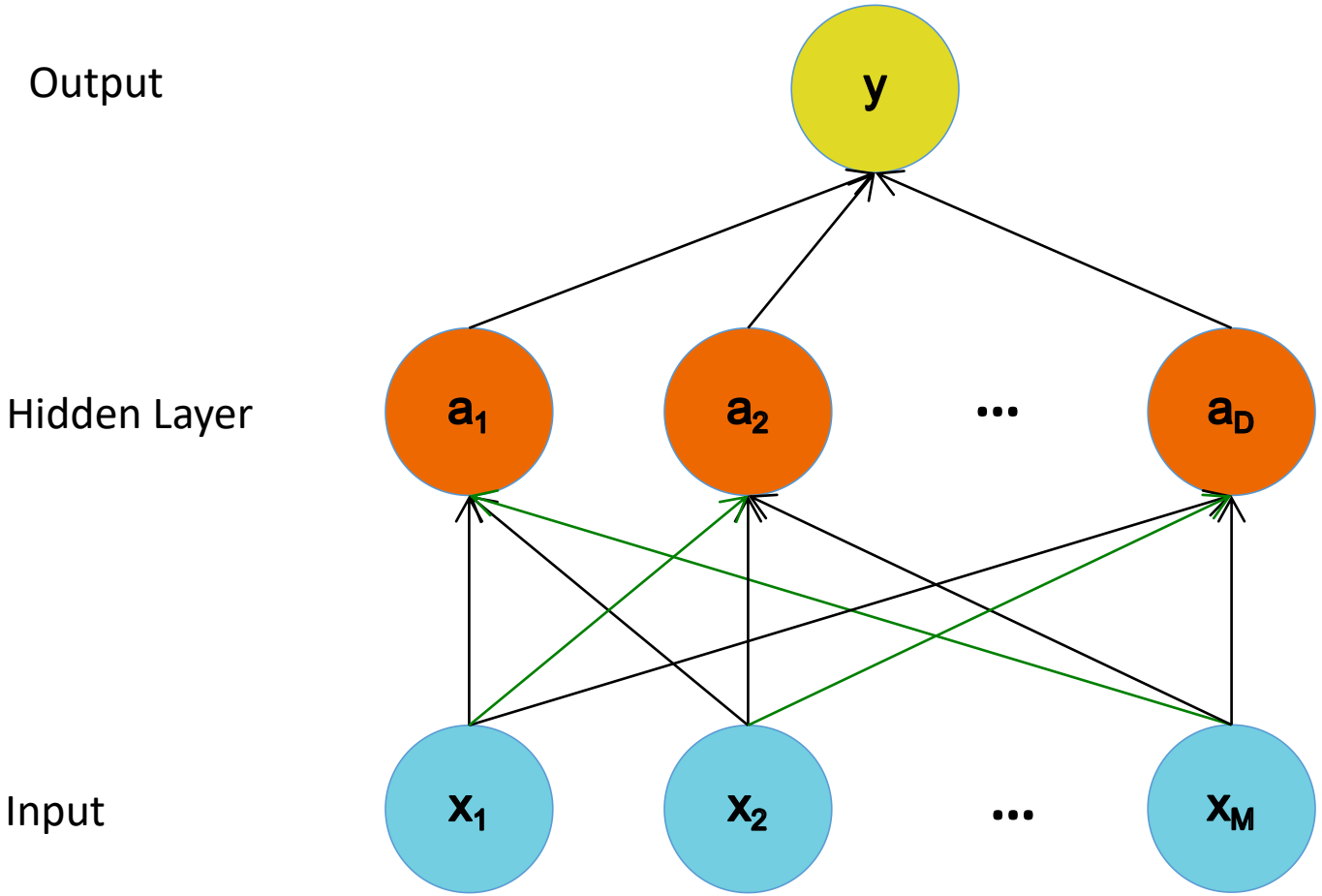
# Building a Neural Net



# Building a Neural Net

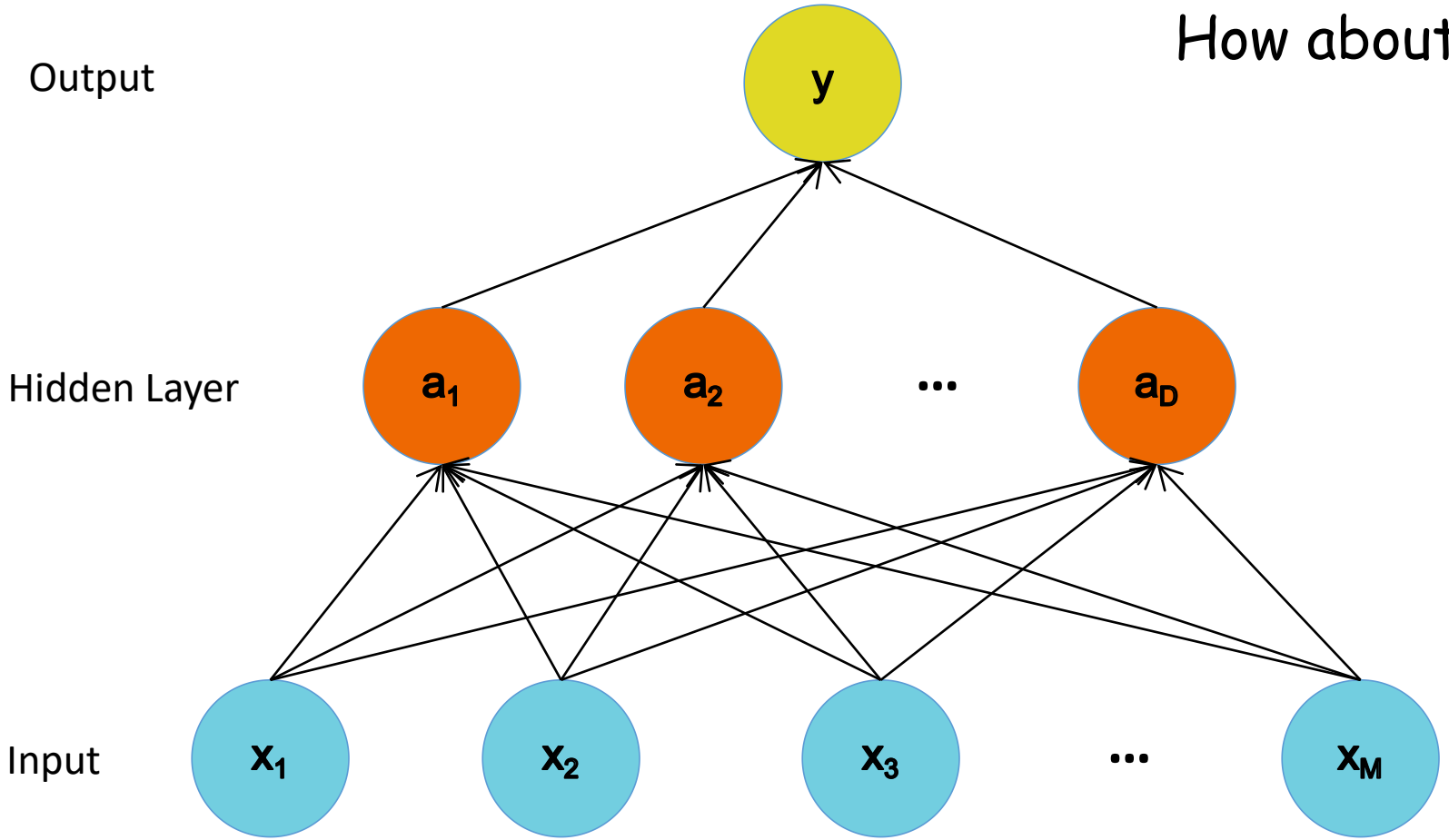


# Building a Neural Net




D = M

# Building a Neural Net

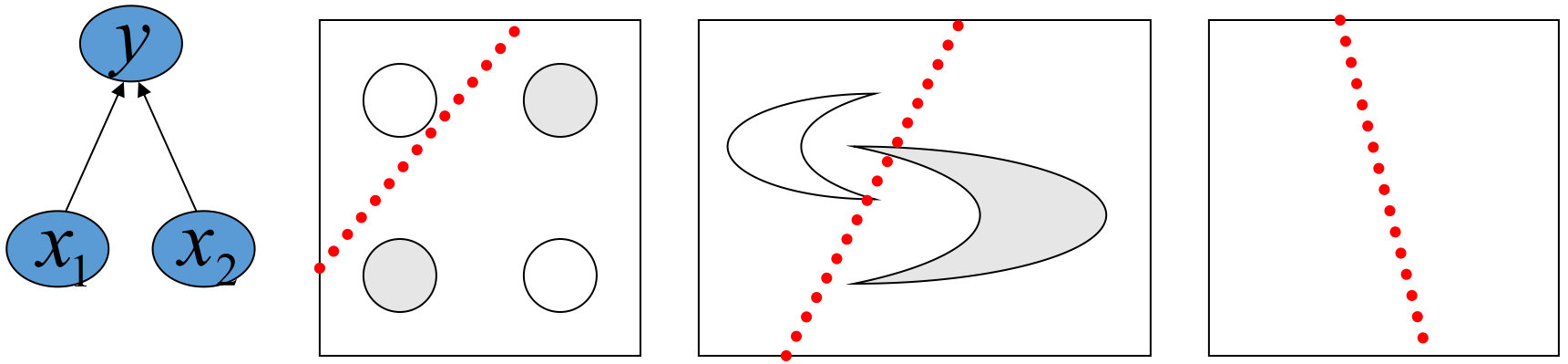



How about multiple outputs?

D < M

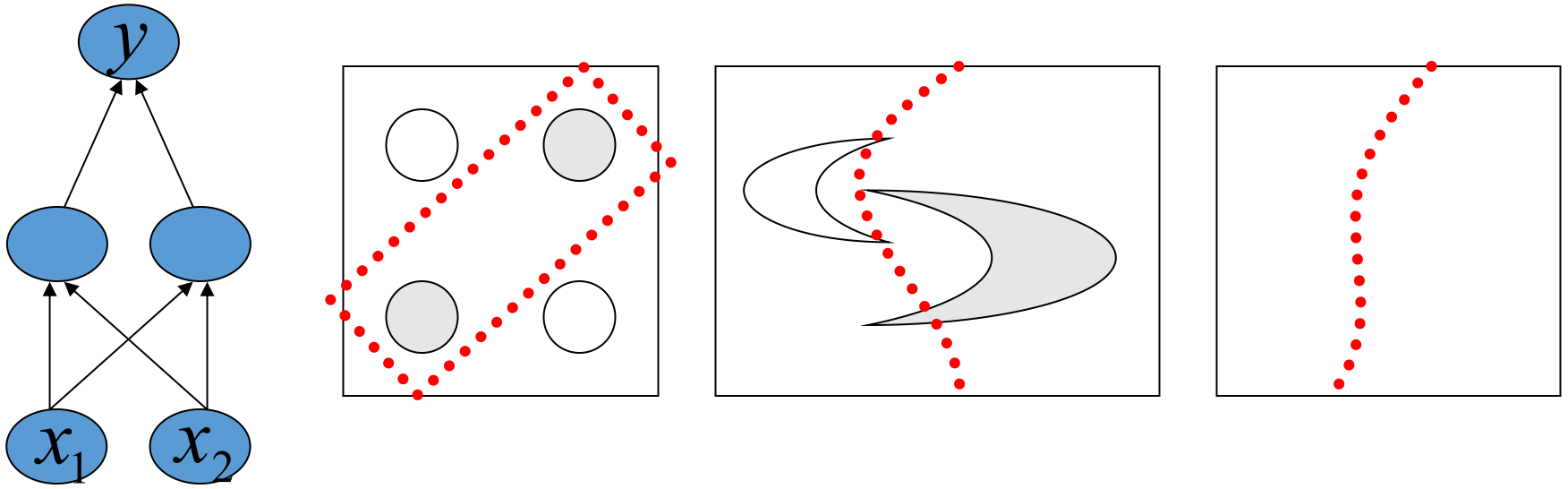
# Decision Boundary

- 0 hidden layers: linear classifier
  - Hyperplanes
  - 1st layer to get a number of half planes (as in single layer perceptron)



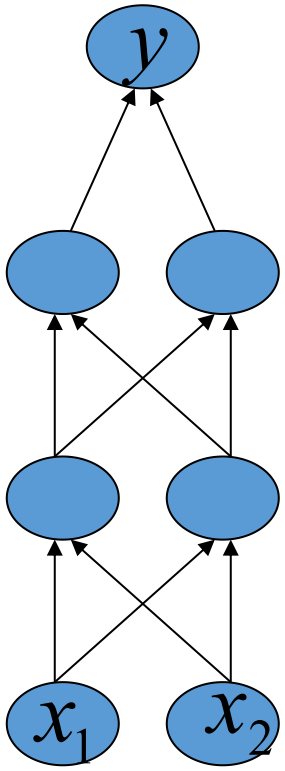
# Decision Boundary

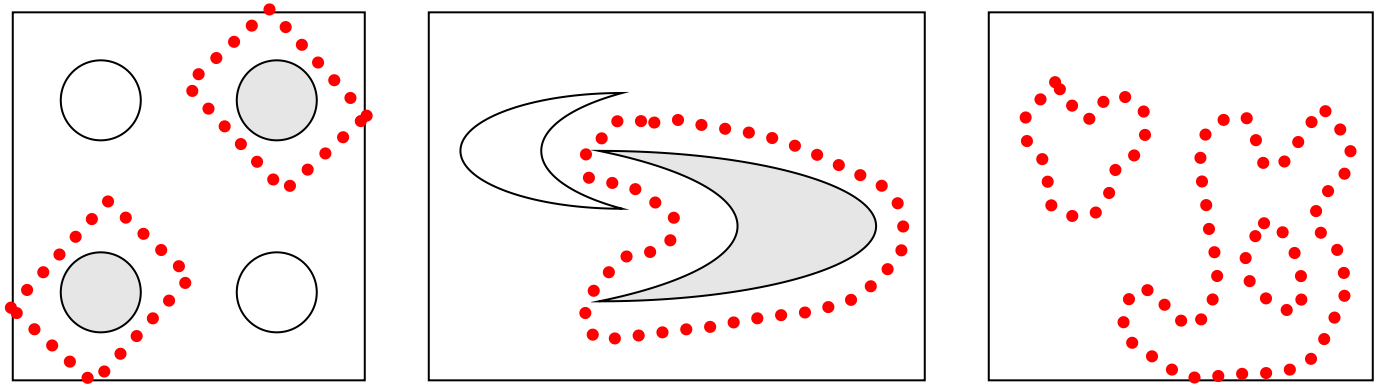
- 1 hidden layer
  - Boundary of convex region (open or closed)
  - 2nd layer to AND the planes together to get a convex region.



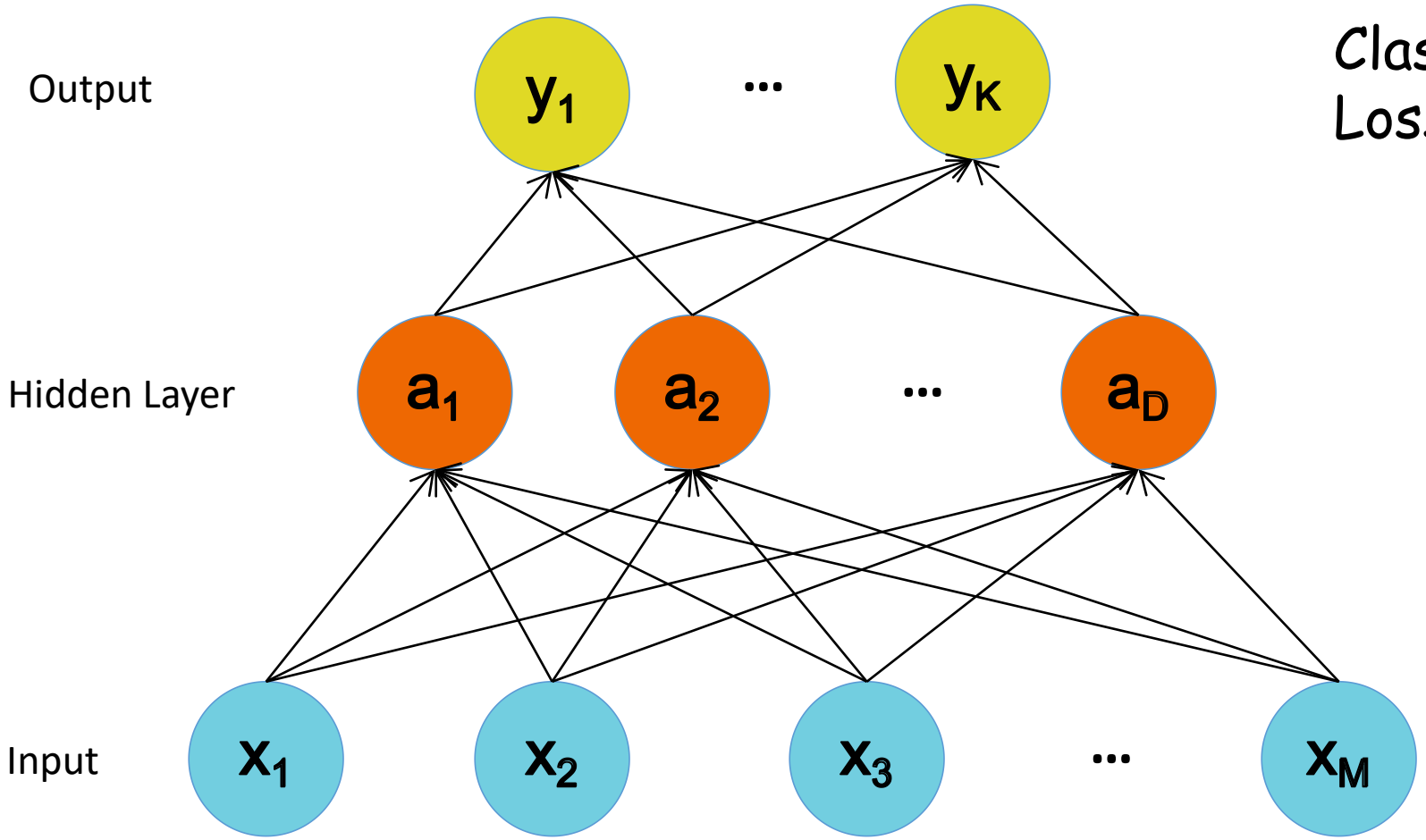
# Decision Boundary

- 2 hidden layers
  - Combinations of convex regions
  - 3rd layer to perform logical operations on these convex regions



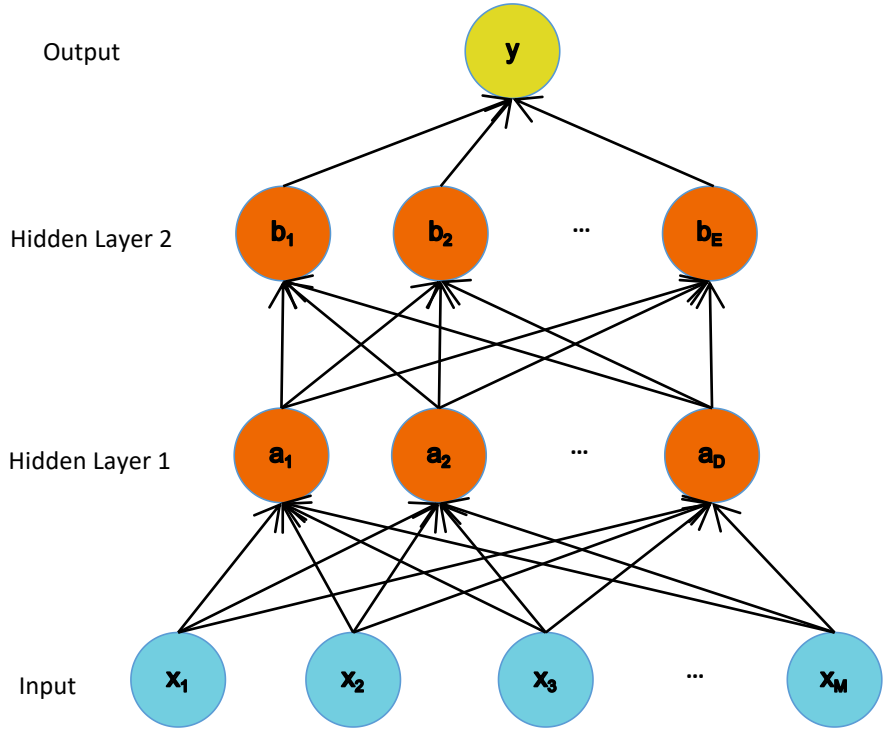
# Decision Functions: Multi-Class Output




Class code: one vs. rest
Loss: cross entropy

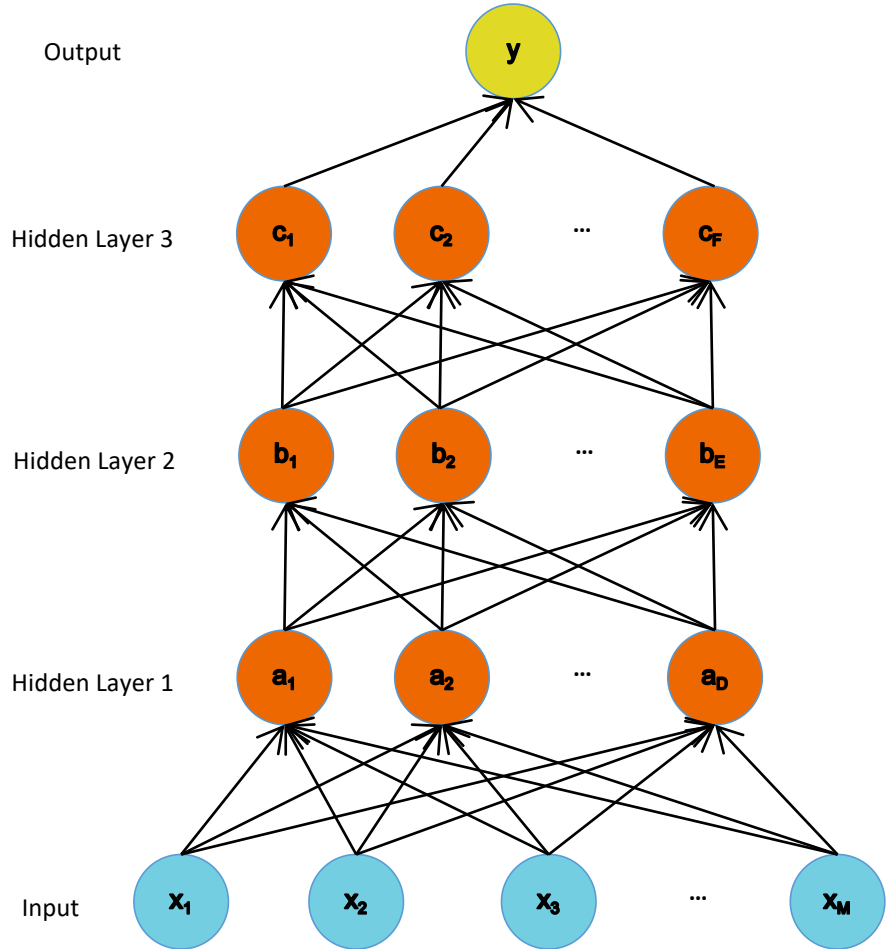
# Decision Functions: Deeper Networks

- Making the neural networks deeper



# Decision Functions: Deeper Networks

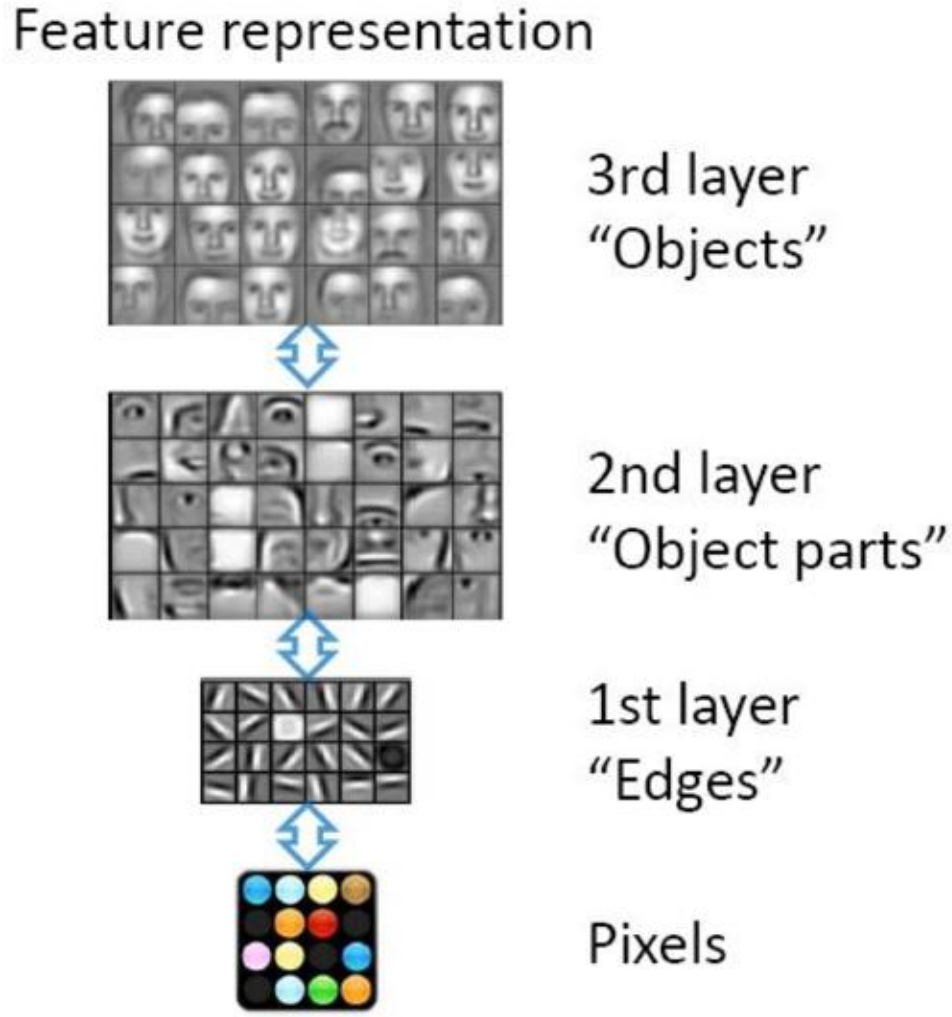
- Making the neural networks deeper and deeper



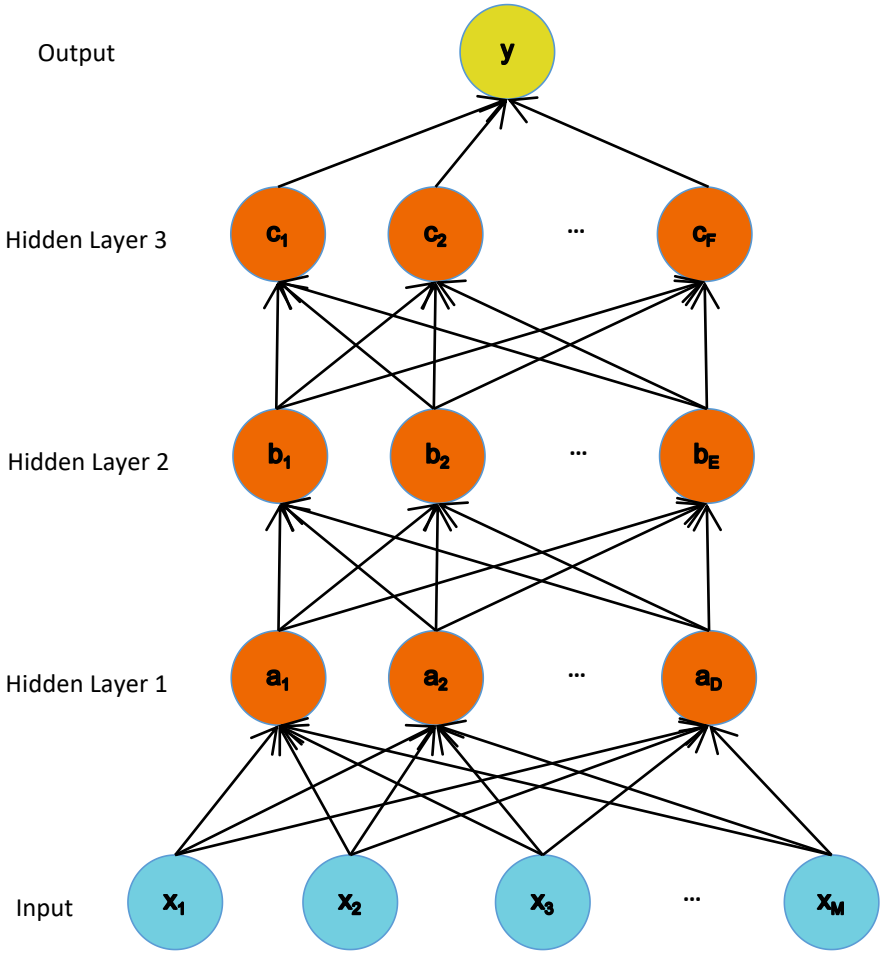
# Why deeper: Decision Functions - Different Levels of Abstraction

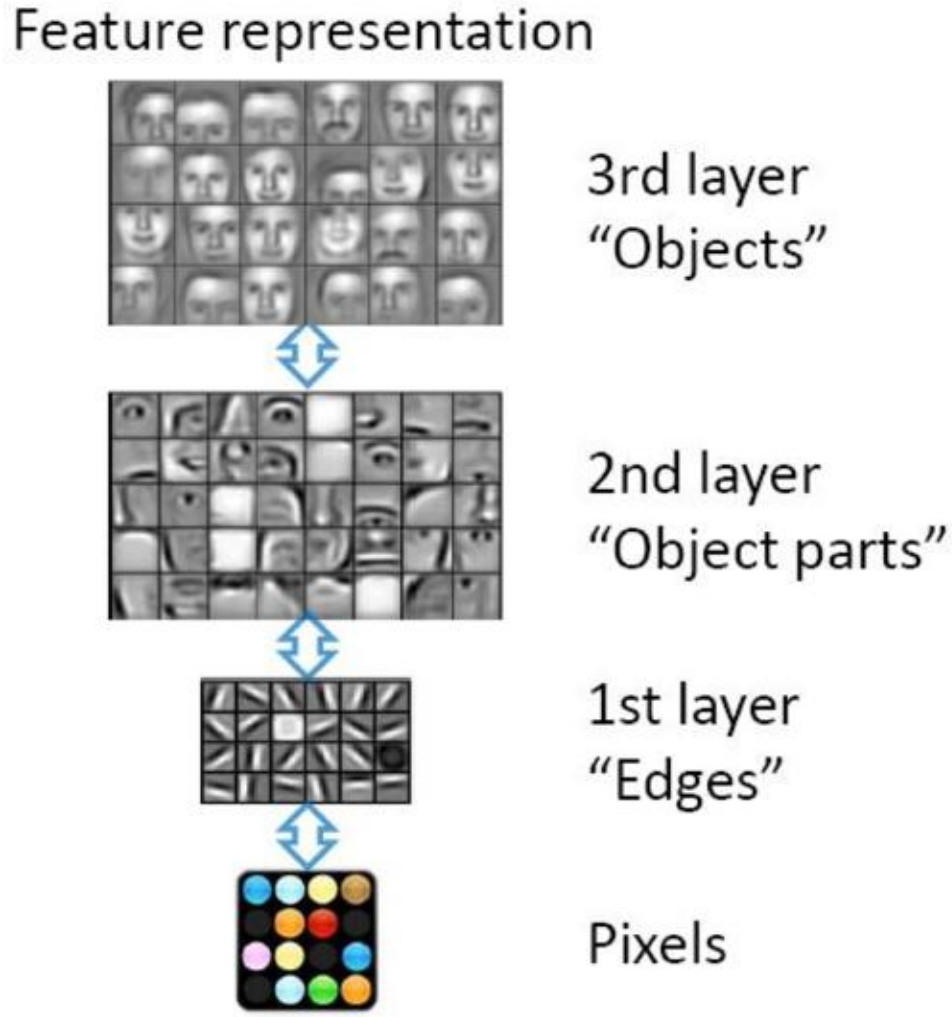
- We don't know the "right" levels of abstraction
- So let the model figure it out!

- Face Recognition:
  - Deep Network can build up increasingly higher levels of abstraction
  - Lines, parts, regions



# Decision Functions: Different Levels of Abstraction



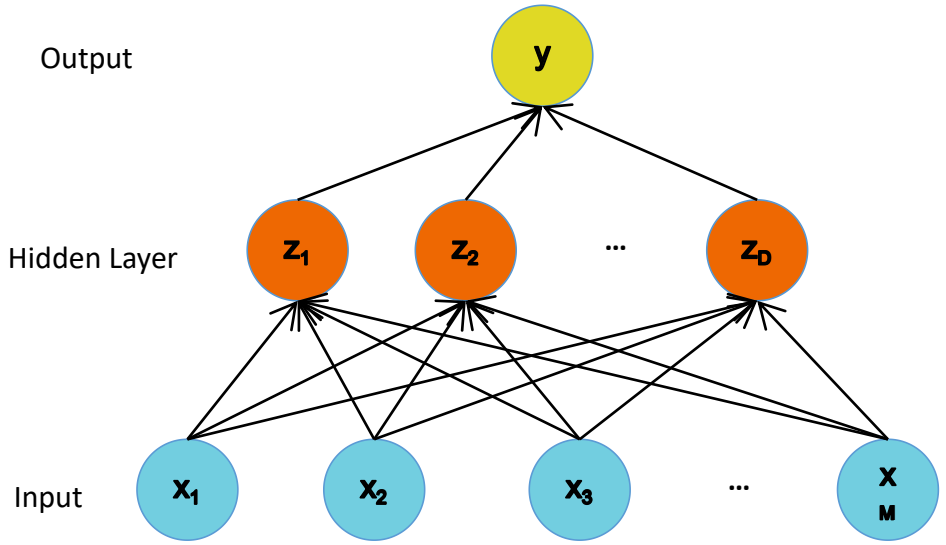
# Architectures

# Neural Network Architectures

- Even for a basic Neural Network, there are many design decisions to make:
  - # of hidden layers (depth)
  - # of units per hidden layer (width)
  - Type of activation function (nonlinearity)
  - Form of objective function
- Nowadays: auto ML for neural architecture search

# Activation Functions

- Neural Network with sigmoid activation functions




(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$

(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

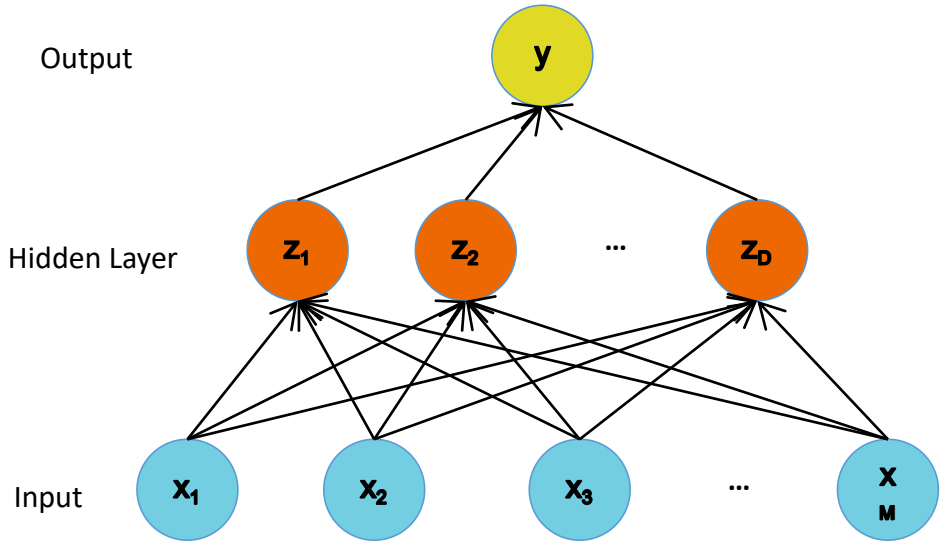
(C) **Hidden (sigmoid)**
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

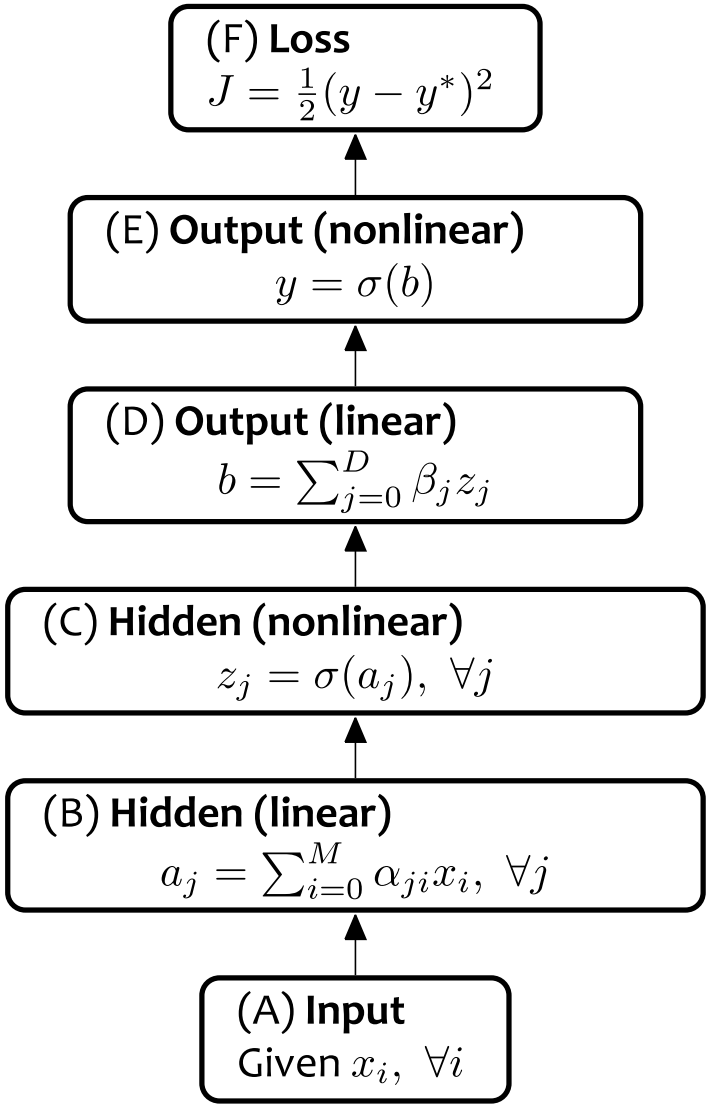
(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) **Input**
Given $x_i, \ \forall i$

# Activation Functions

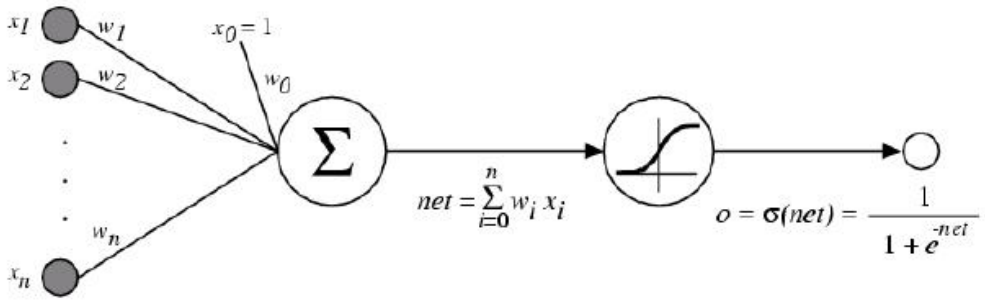
- Neural Network with arbitrary nonlinear activation functions



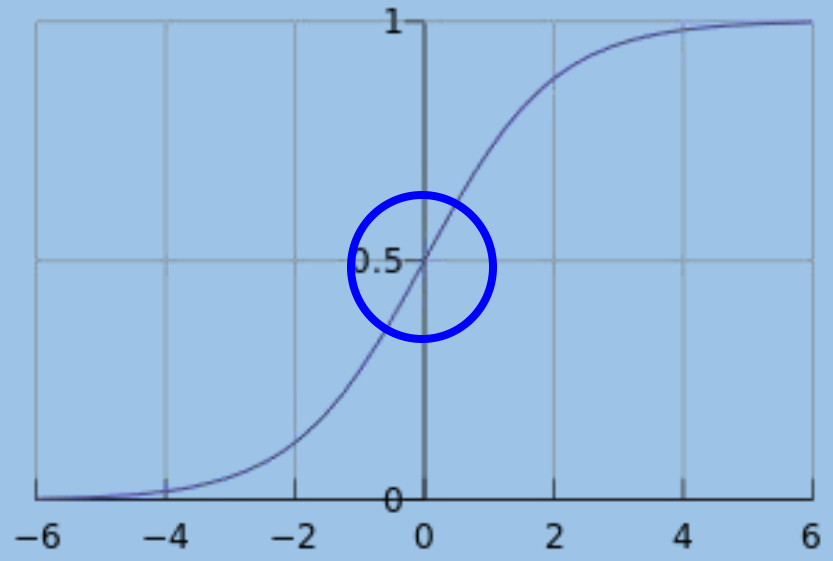
# Activation Functions

- So far, we've assumed that the activation function (nonlinearity) is always the sigmoid function…




## Sigmoid / Logistic Function

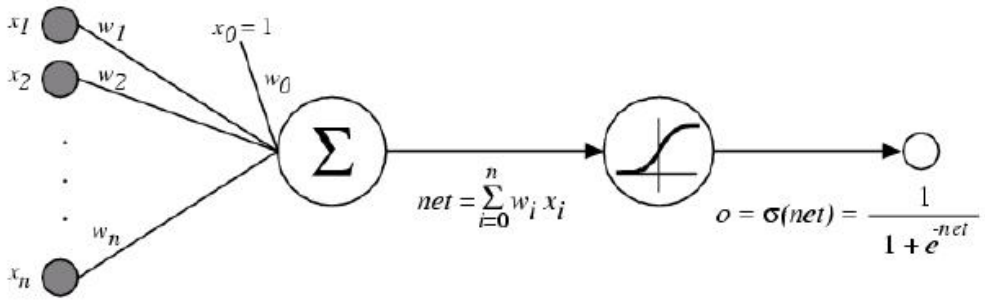
$$\text{logistic}(u) \equiv \frac{1}{1 + e^{-u}}$$

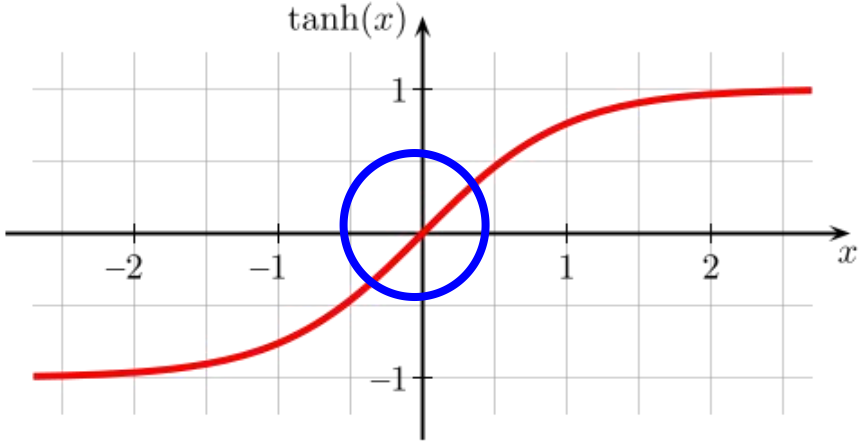


# Activation Functions

- A new change: modifying the nonlinearity
  - The logistic is not widely used in modern Artificial Neural Networks

Alternate 1: tanh



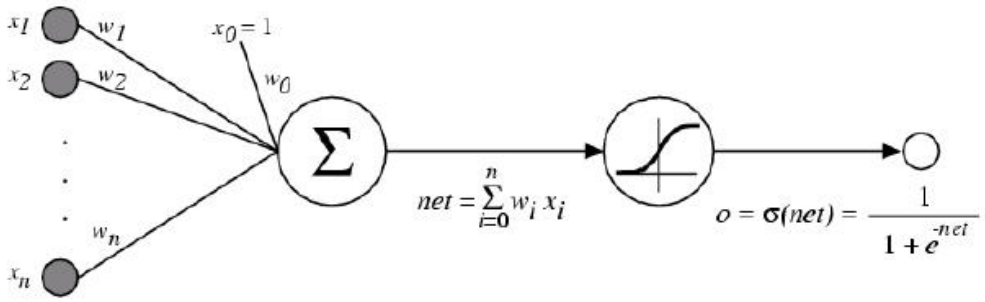



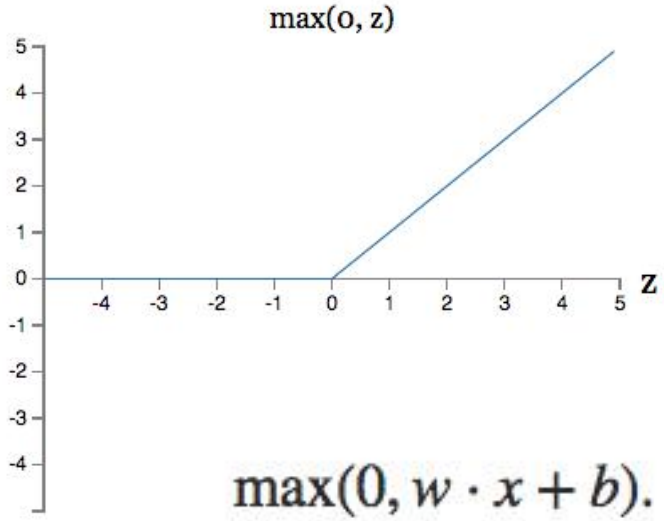

Like logistic function but shifted to range [-1, +1]

# Activation Functions

- A new change: modifying the nonlinearity
  - reLU often used in vision tasks

Alternate 2: rectified linear unit



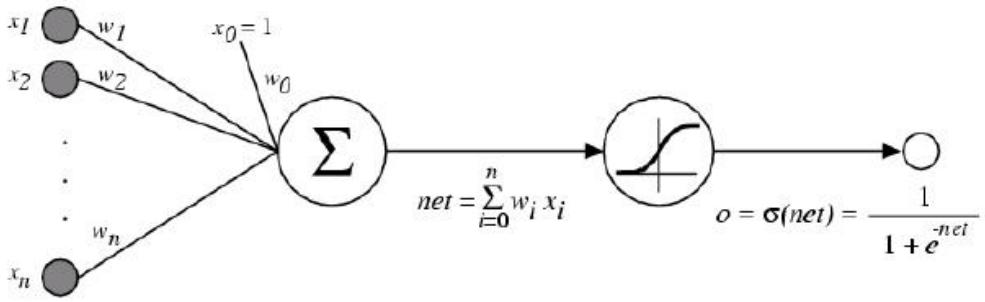



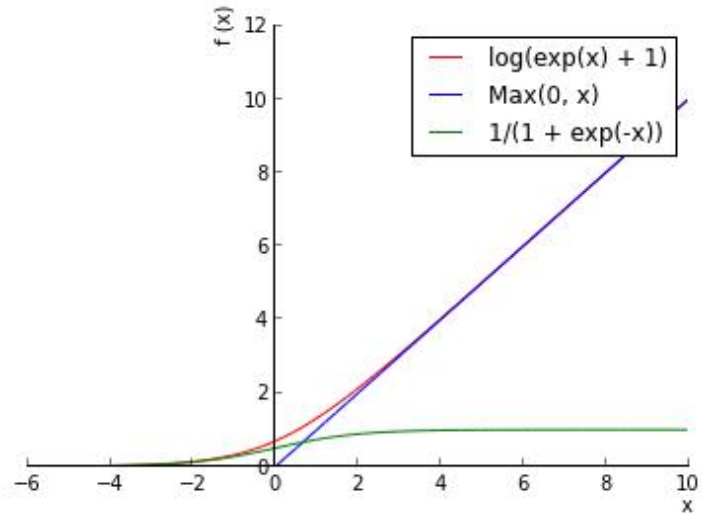

Linear with a cutoff at zero
(Implementation: clip the gradient when you pass zero)

# Activation Functions

- A new change: modifying the nonlinearity
  - reLU often used in vision tasks

Alternate 2: rectified linear unit



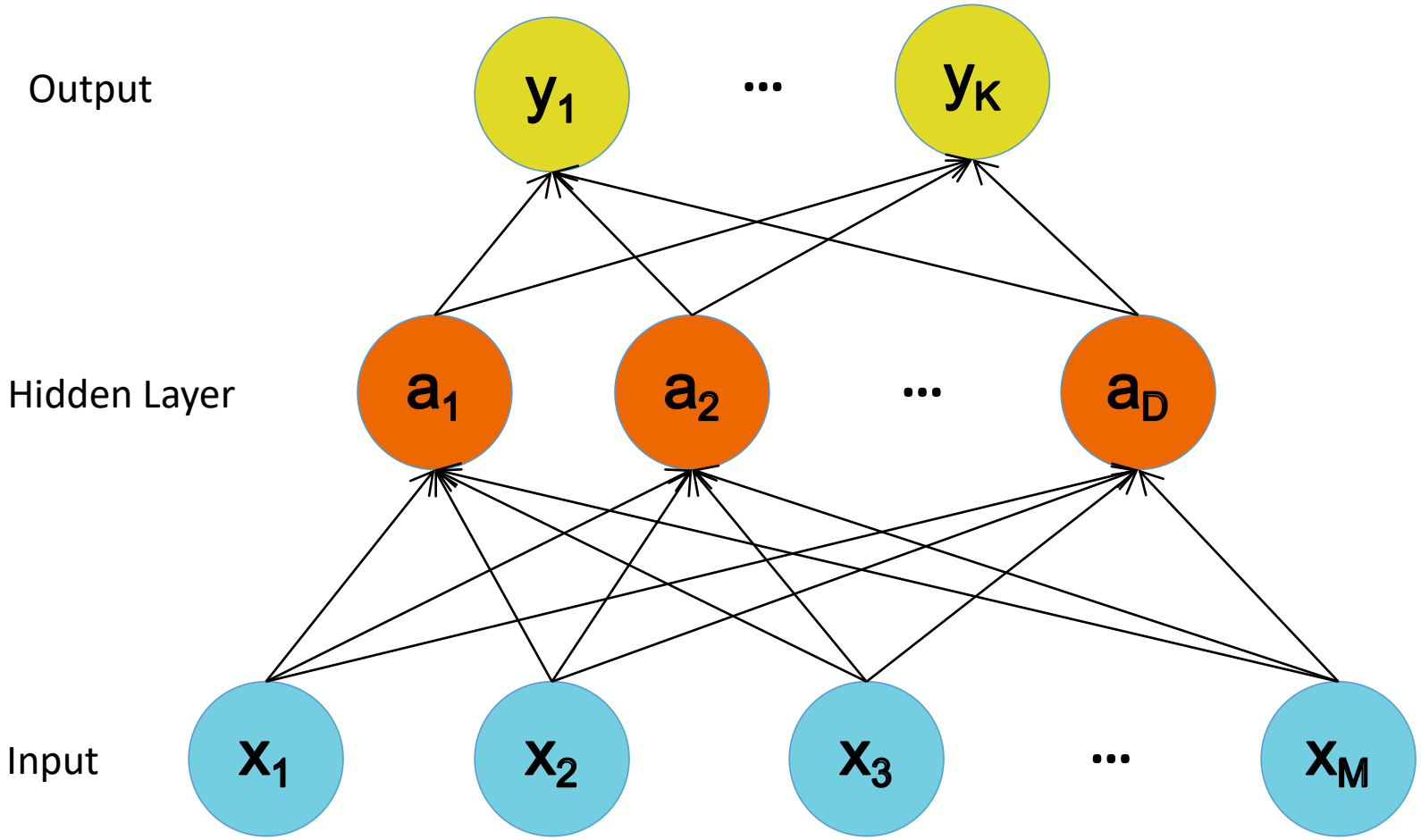

Soft version: log(exp(x)+1)
Doesn't saturate (at one end)
Sparsifies outputs
Helps with vanishing gradient


# Objective Functions for NNs

- Regression:
  - Use the same objective as Linear Regression
  - Quadratic loss (i.e. mean squared error)
- Classification:
  - Use the same objective as Logistic Regression
  - Cross-entropy (i.e. negative log likelihood)
  - This requires probabilities, so we add an additional "softmax" layer at the end of our network

| | Forward | Backward |
|---|---|---|
| Quadratic | $J = \frac{1}{2}(y - y^*)^2$ | $\frac{dJ}{dy} = y - y^*$ |
| Cross Entropy | $J = y^* \log(y) + (1 - y^*) \log(1 - y)$ | $\frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{y - 1}$ |

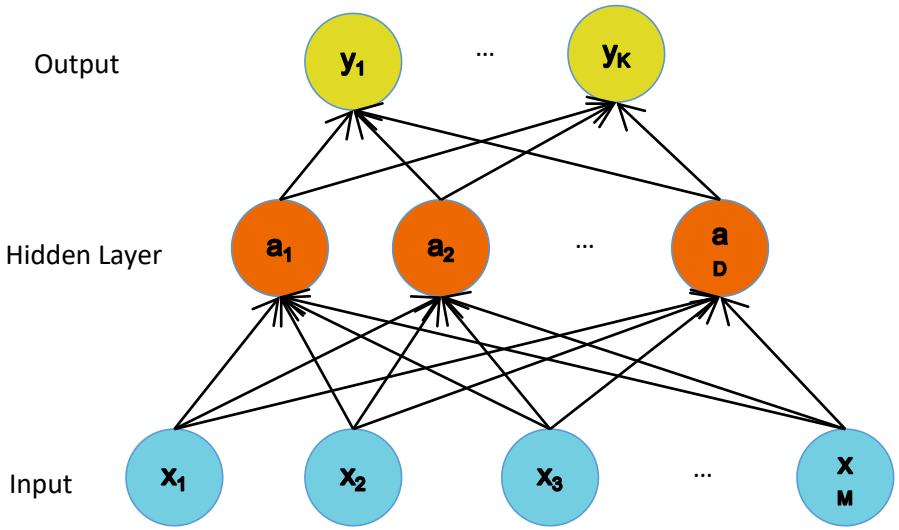
# Multi-Class Output

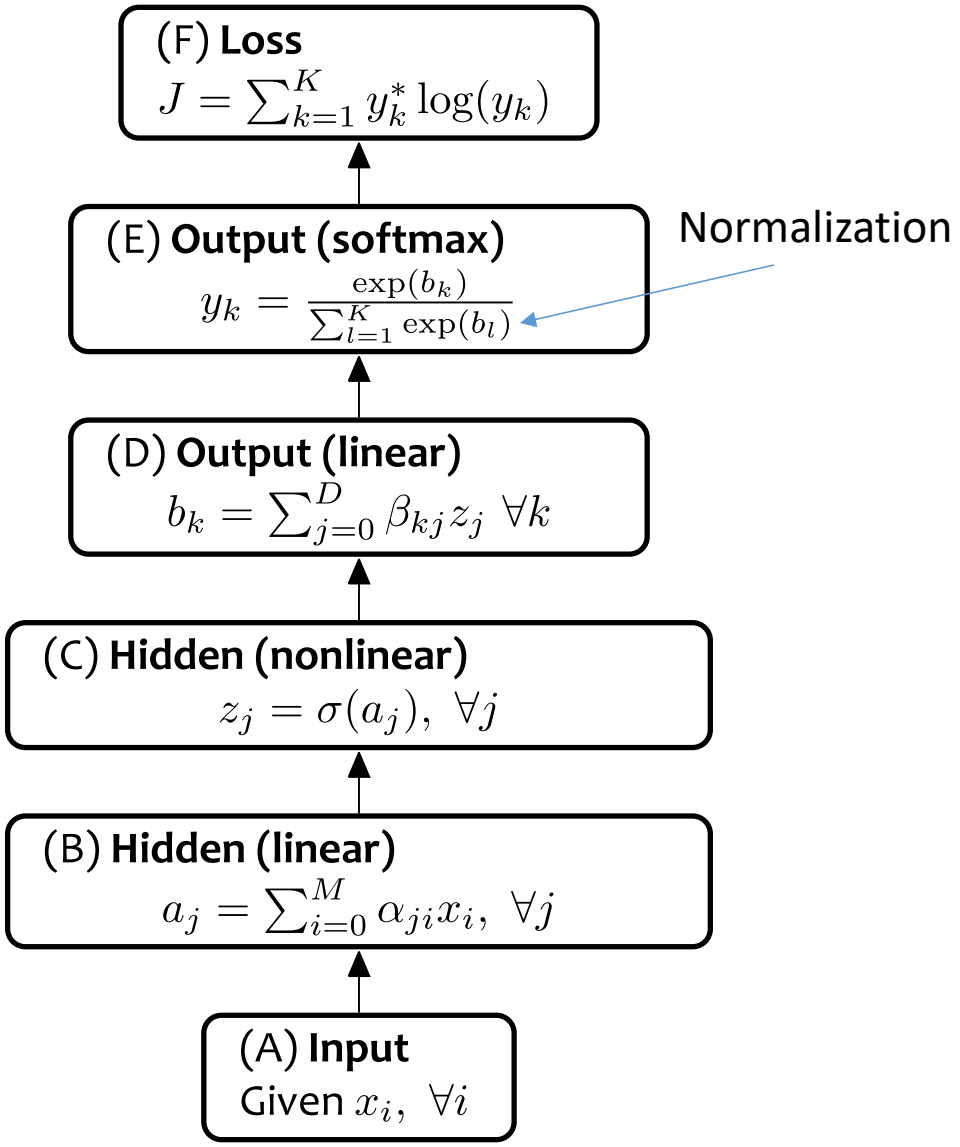


# Multi-Class Output

- Softmax:

$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}$$



# Cross-entropy vs. Quadratic loss

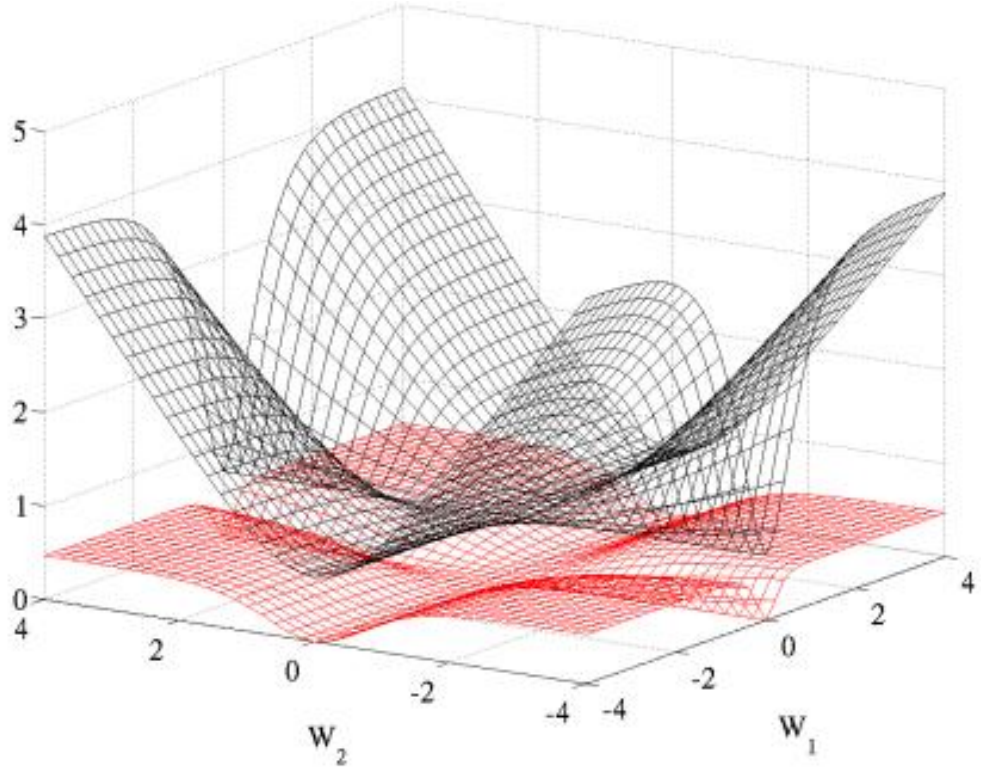



Figure 5: *Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, $W_1$ respectively on the first layer and $W_2$ on the second, output layer.*

# Background: A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$$

2. Choose each of these:

- Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

- Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

# Backpropagation


# Training: Backpropagation

- **Question 1:**
  When can we compute the gradients of the parameters of an arbitrary neural network?

- **Question 2:**
  When can we make the gradient computation efficient?

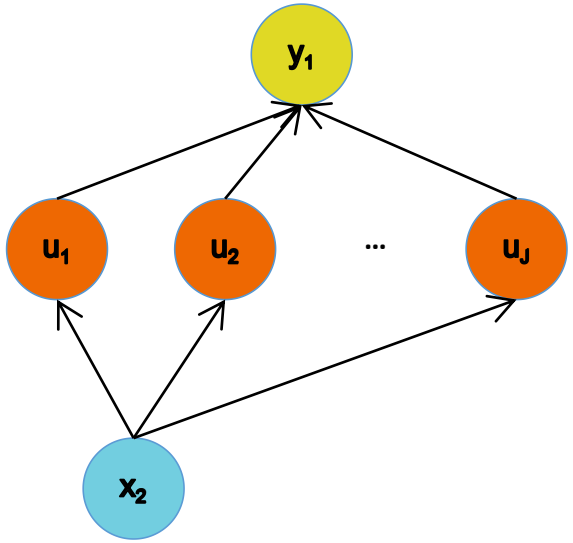
# Training: Chain Rule

- **Backpropagation** is just **repeated** application of the **chain rule**

**Given:** $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$.

**Chain Rule:**

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$



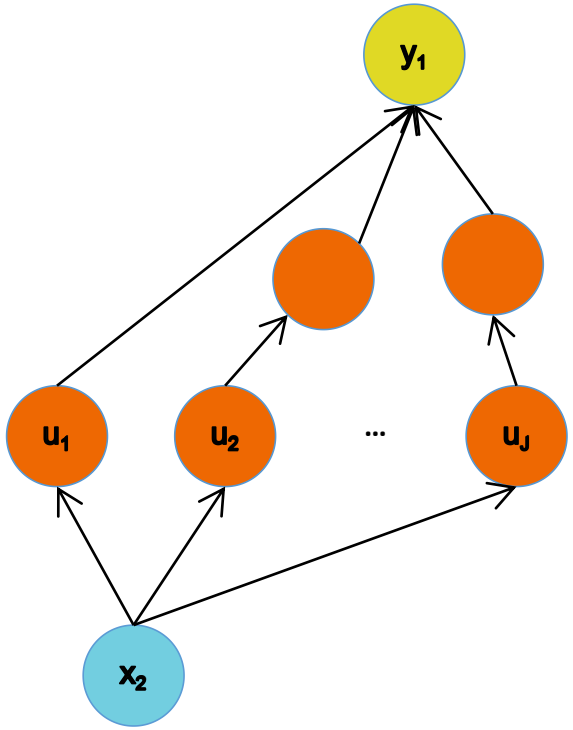
# Training: Chain Rule

- **Backpropagation** is just **repeated** application of the **chain rule**

**Given:** $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$.

**Chain Rule:**

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$



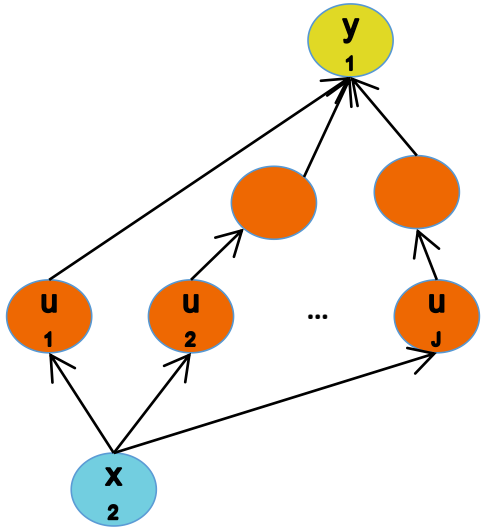
# Training: Chain Rule

- Backpropagation:
  - **Instantiate the computation as a directed acyclic graph**, where each intermediate quantity is a node
  - **At each node, store**
    - the quantity computed in the forward pass
    - the **partial derivative** of the goal with respect to that node's intermediate quantity
  - Initialize all partial derivatives to 0.
  - Visit each node in reverse topological order. At each node, add its contribution to the partial derivatives of its parents
- This algorithm is also called **automatic differentiation in the reverse-mode**

**Given:** $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$.

**Chain Rule:**

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$



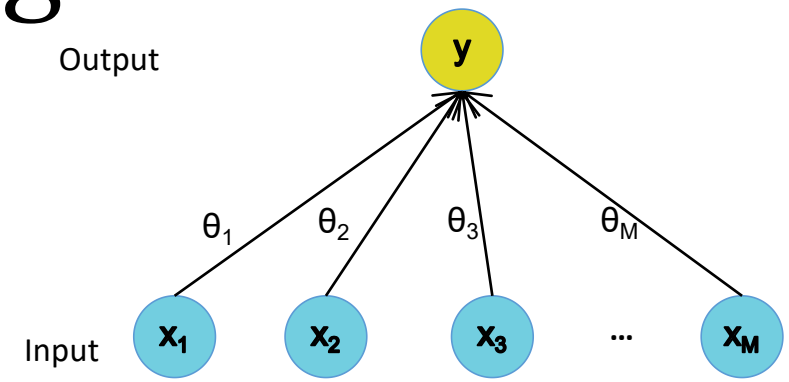

# Training: Backpropagation

**Simple Example:** The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

| Forward | Backward |
|---|---|
| $J = cos(u)$ | $\frac{dJ}{du}$ += $-sin(u)$ |
| $u = u_1 + u_2$ | $\frac{dJ}{du_1}$ += $\frac{dJ}{du}\frac{du}{du_1}$, $\frac{du}{du_1} = 1$ $\quad$ $\frac{dJ}{du_2}$ += $\frac{dJ}{du}\frac{du}{du_2}$, $\frac{du}{du_2} = 1$ |
| $u_1 = sin(t)$ | $\frac{dJ}{dt}$ += $\frac{dJ}{du_1}\frac{du_1}{dt}$, $\frac{du_1}{dt} = \cos(t)$ |
| $u_2 = 3t$ | $\frac{dJ}{dt}$ += $\frac{dJ}{du_2}\frac{du_2}{dt}$, $\frac{du_2}{dt} = 3$ |
| $t = x^2$ | $\frac{dJ}{dx}$ += $\frac{dJ}{dt}\frac{dt}{dx}$, $\frac{dt}{dx} = 2x$ |

# Training: Backpropagation

- Case 1: Logistic Regression




Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backward

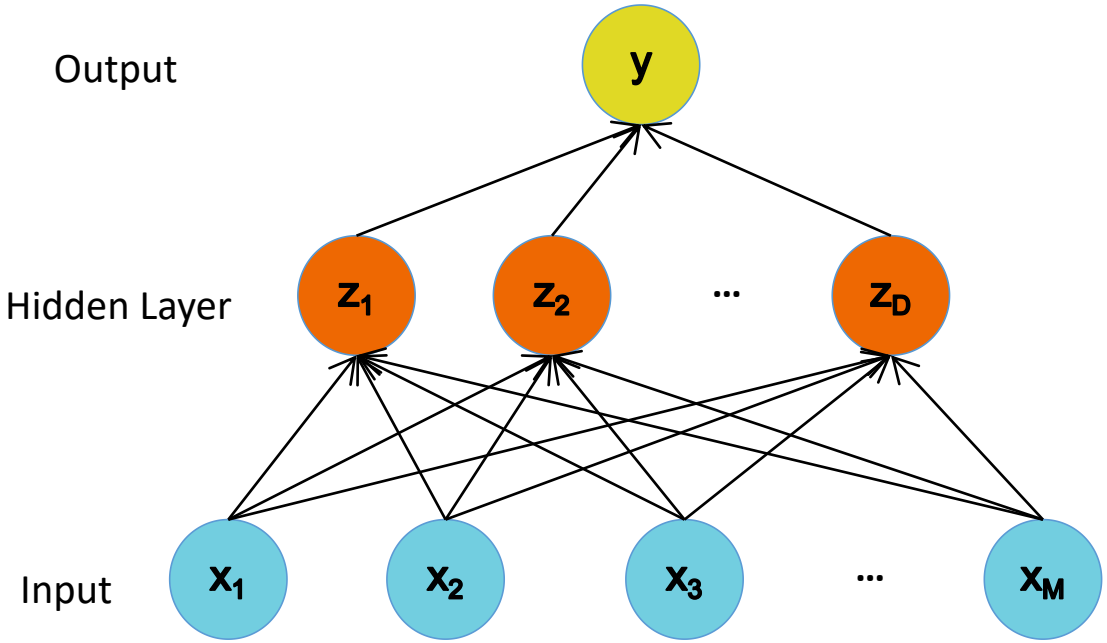
$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{da} = \frac{dJ}{dy}\frac{dy}{da}, \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

$$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \frac{da}{d\theta_j} = x_j$$

$$\frac{dJ}{dx_j} = \frac{dJ}{da}\frac{da}{dx_j}, \frac{da}{dx_j} = \theta_j$$

# Training: Backpropagation




(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$

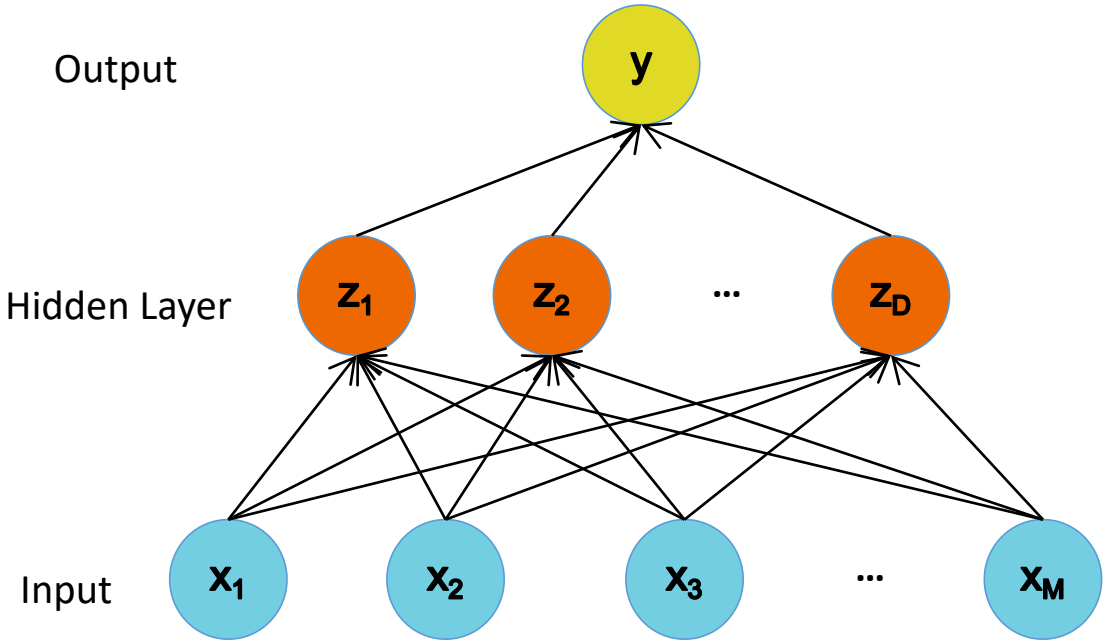
(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

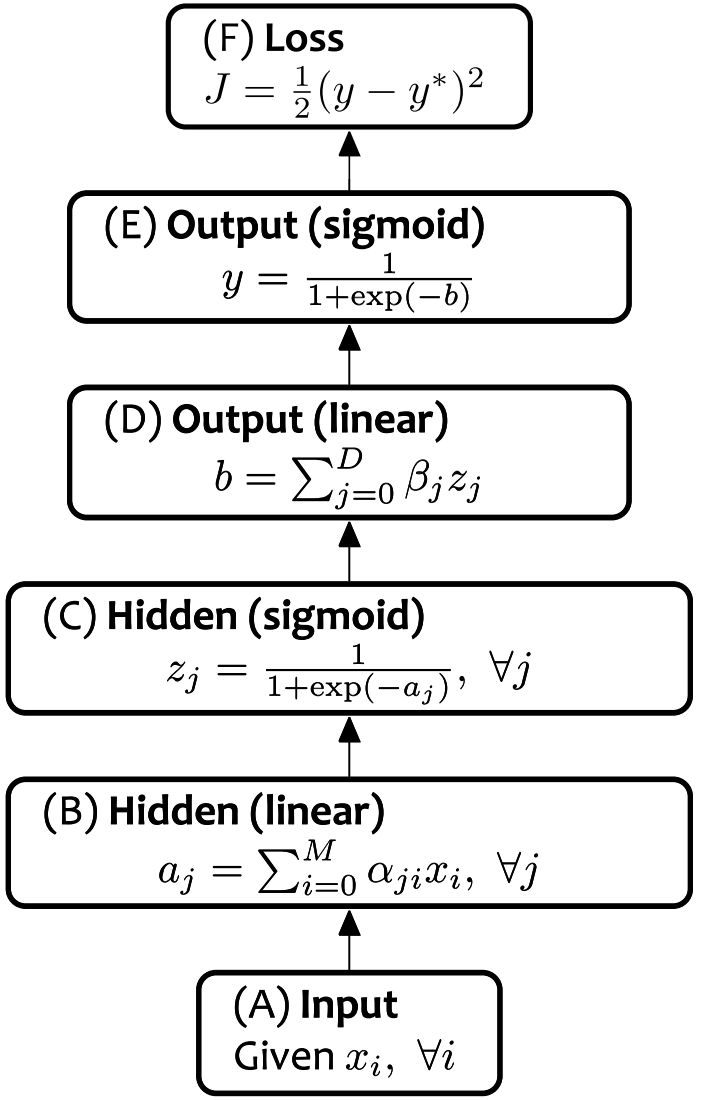
(C) **Hidden (sigmoid)**
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) **Input**
Given $x_i, \ \forall i$

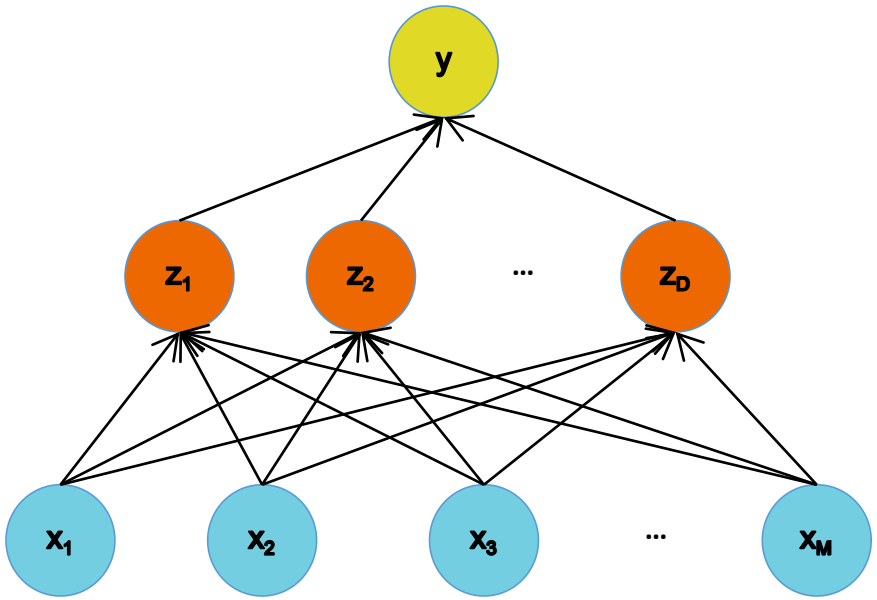
# Training: Backpropagation



# Training: Backpropagation

- Case 2: Neural Network




Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$$


# Training: Backpropagation

- Case 2: Neural Network

| | Forward | Backward |
|---|---|---|
| Module 5 | $J = y^* \log y + (1-y^*)\log(1-y)$ | $\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1-y^*)}{y-1}$ |
| Module 4 | $y = \frac{1}{1+\exp(-b)}$ | $\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b)+1)^2}$ |
| Module 3 | $b = \sum_{j=0}^{D} \beta_j z_j$ | $\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \frac{db}{d\beta_j} = z_j$<br>$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \frac{db}{dz_j} = \beta_j$ |
| Module 2 | $z_j = \frac{1}{1+\exp(-a_j)}$ | $\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j)+1)^2}$ |
| Module 1 | $a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$ | $\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$<br>$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$ |


# Summary

## 1. Neural Networks

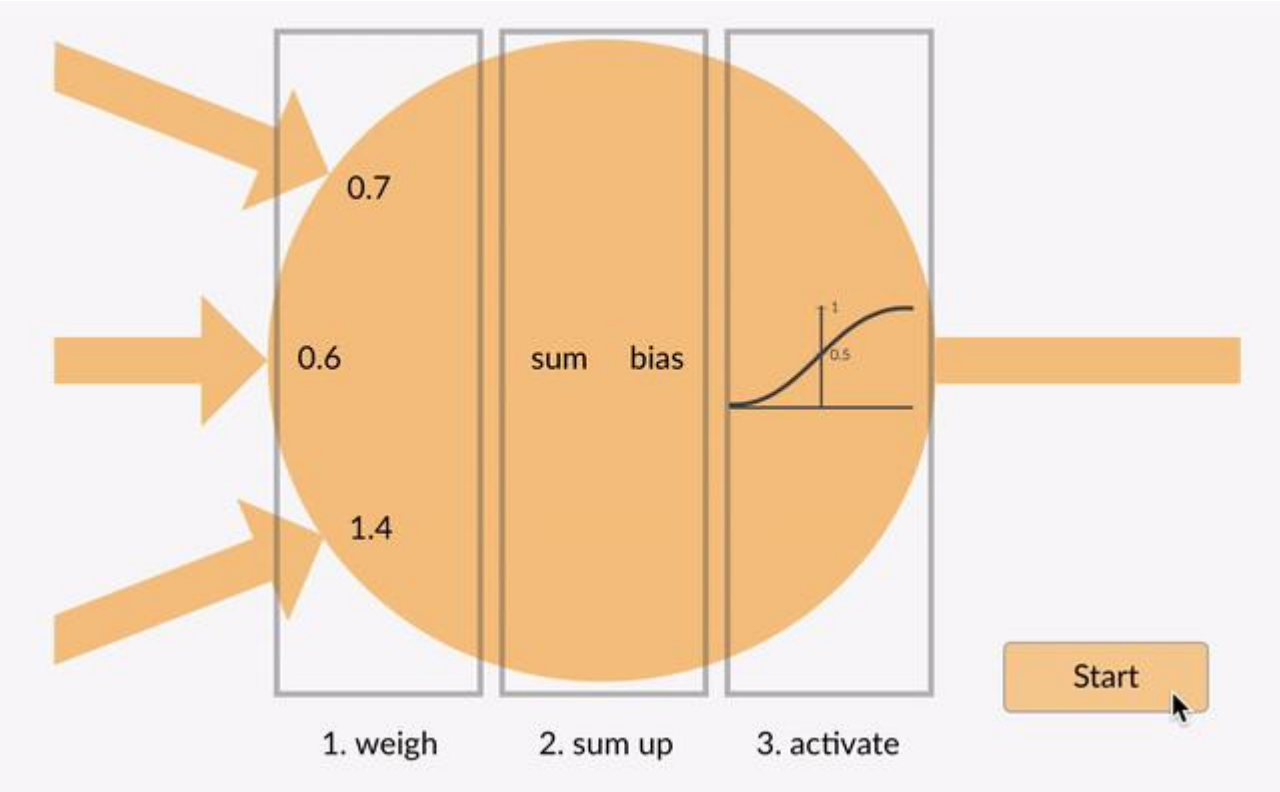
- Provide a way of learning features
- Are highly nonlinear prediction functions
- (Can be) a highly parallel network of logistic regression classifiers
- Discover useful hidden representations of the input

## 2. Backpropagation

- Provides an efficient way to compute gradients
- Is a special case of reverse-mode automatic differentiation

# Summary: ANN

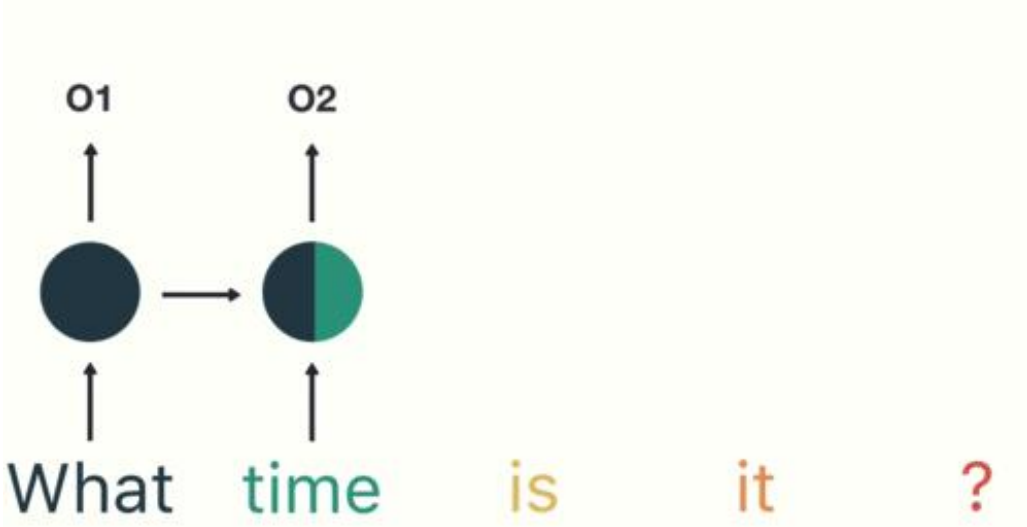
- Artificial Neural Network (ANN)
  - ➢ Challenges
    - ✓ ANN loses the spatial features of an image
    - ✓ ANN cannot capture sequential information in the input data
    - ✓ One common problem in all these neural networks is the Vanishing and Exploding Gradient

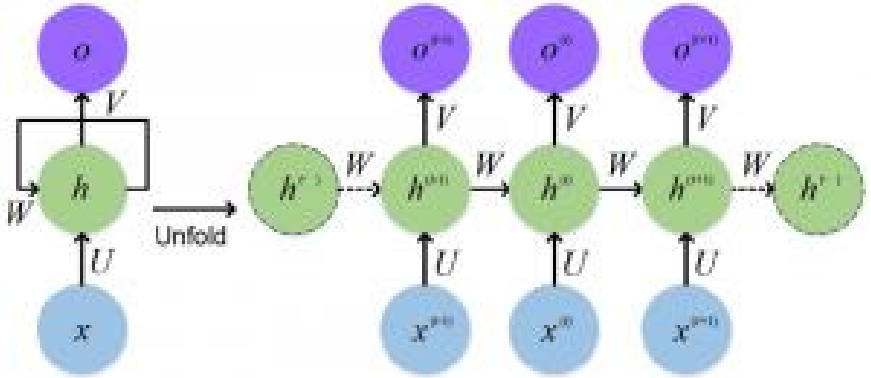


# Summary: RNN

- Recurrent Neural Network (RNN)
  - ➢ Time Series data
  - ➢ Text data
  - ➢ Audio data

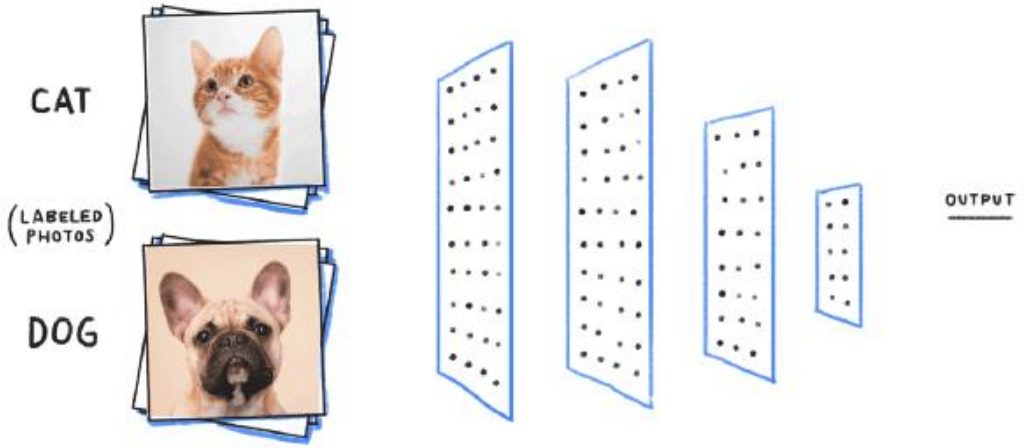
  - ➢ Challenges
    - ✓ Suffer from the vanishing and exploding gradient problem



# Summary: CNN

- Convolution Neural Network (CNN)
  - ➢ The building blocks of CNNs are filters a.k.a. kernels

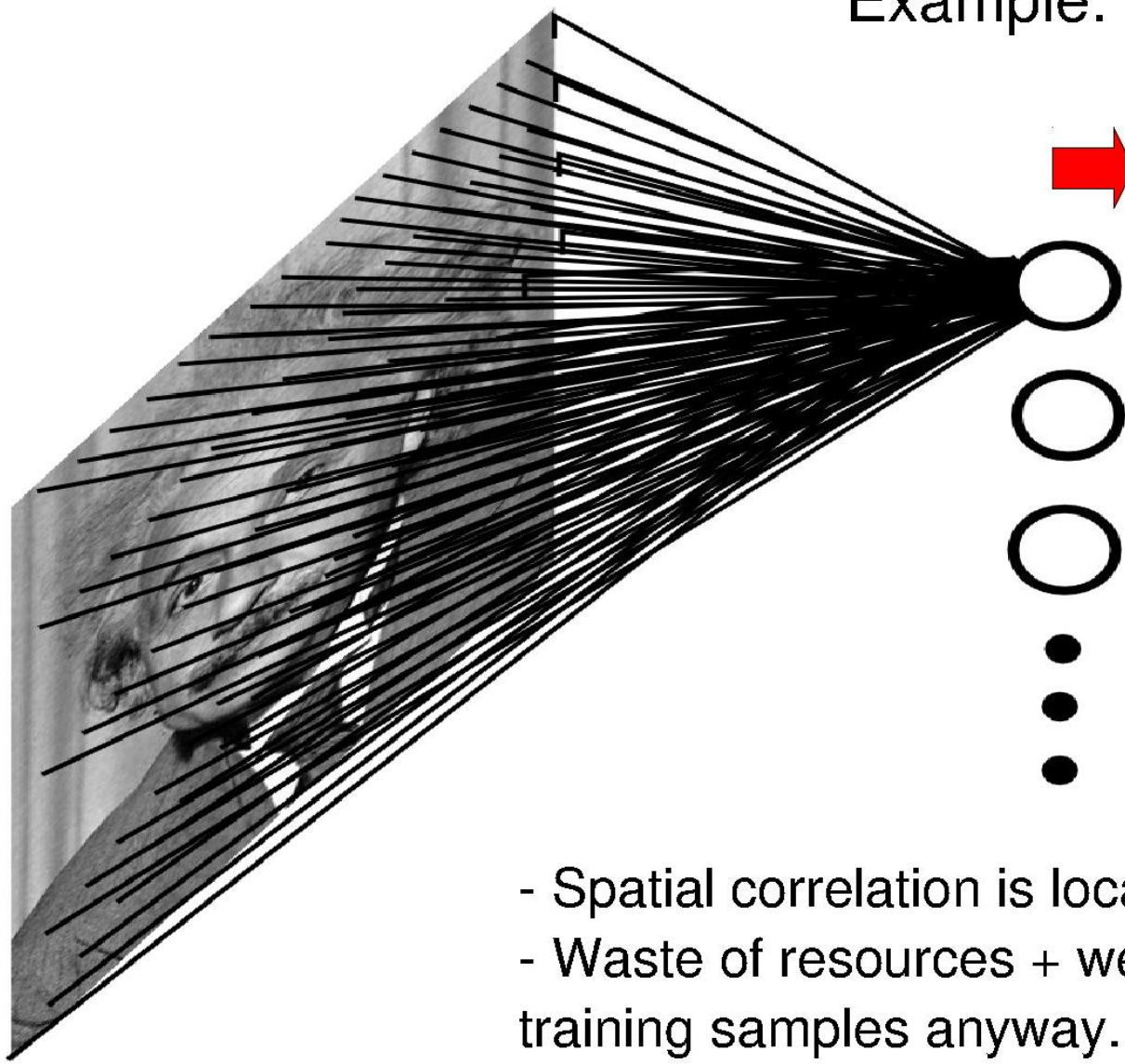



- Comparing the Different Types of Neural Networks (MLP(ANN) vs. RNN vs. CNN)

| | MLP | RNN | CNN |
|---|---|---|---|
| Data | Tabular data | Sequence data (Time Series,Text, Audio) | Image data |
| Recurrent connections | No | Yes | No |
| Parameter sharing | No | Yes | Yes |
| Spatial relationship | No | No | Yes |
| Vanishing & Exploding Gradient | Yes | Yes | Yes |

# Convolutional Neural Networks

# Fully Connected Layer



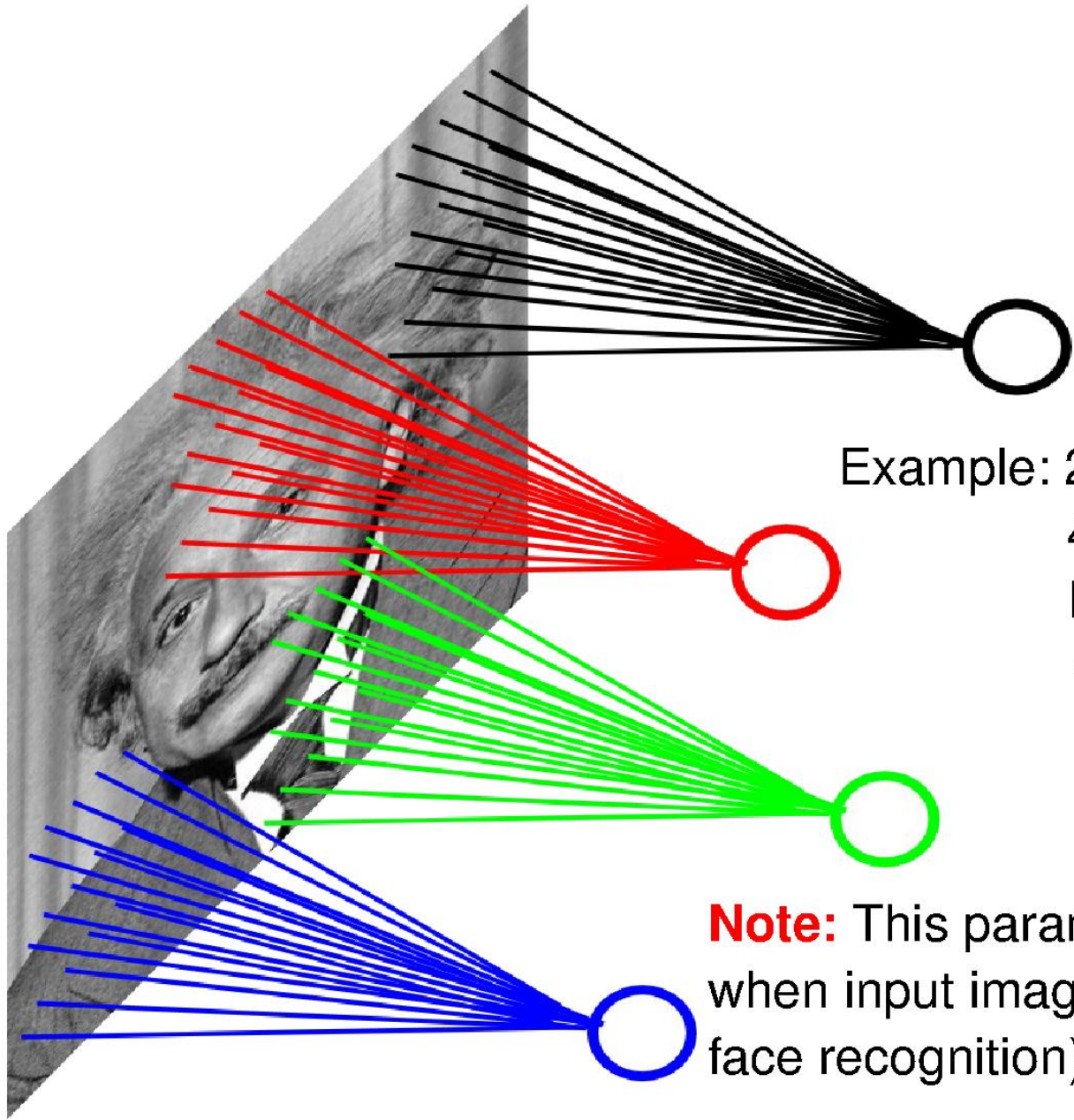
Example: 200x200 image

40K hidden units

**~2B parameters**!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

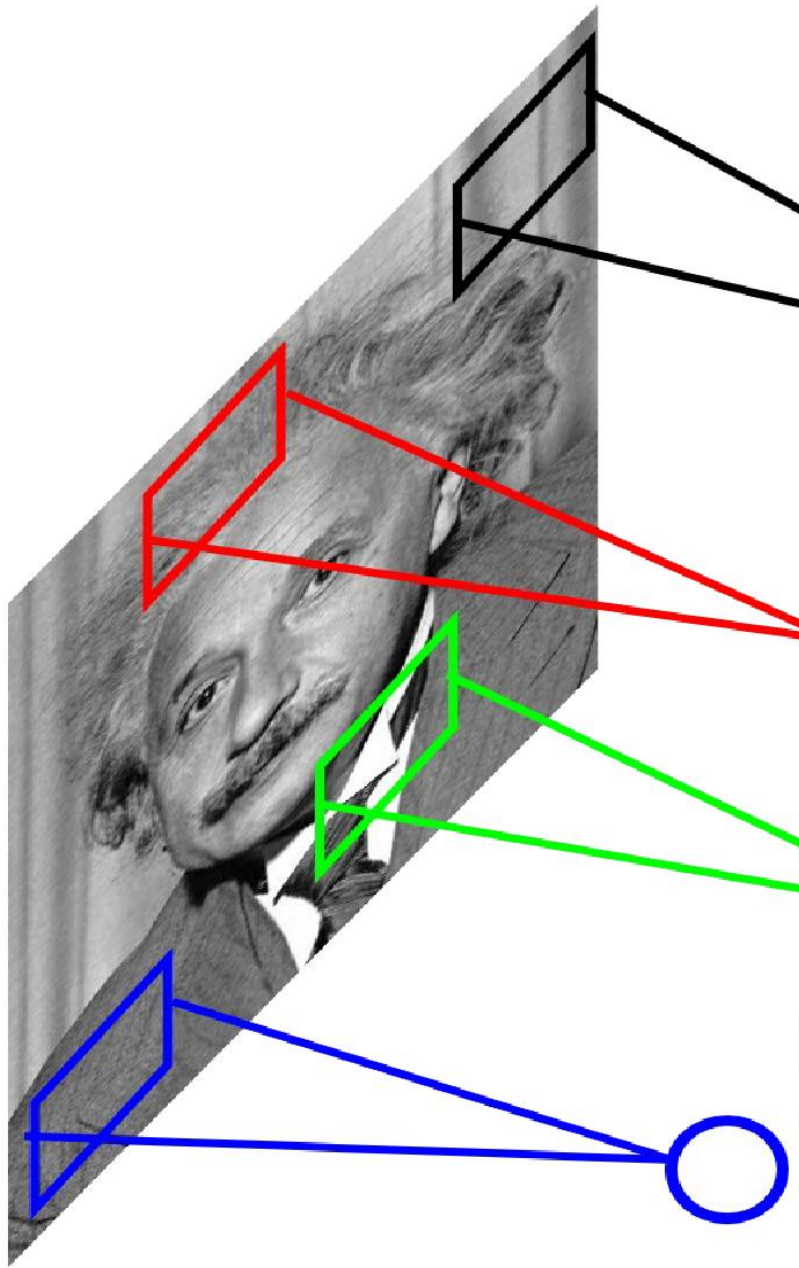
# Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

**Note:** This parameterization is good when input image is registered (e.g., face recognition).

# Locally Connected Layer

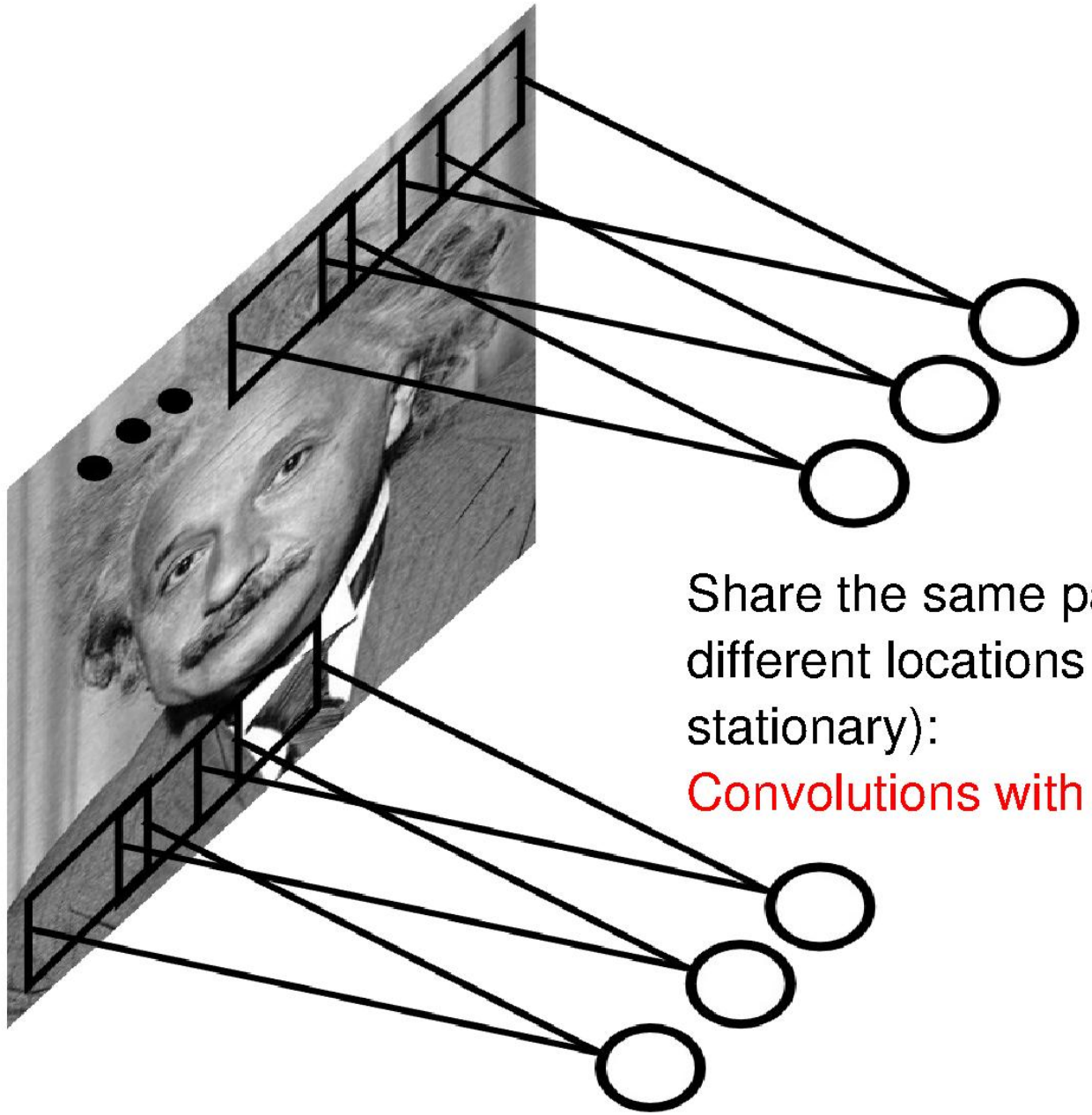


**STATIONARITY?** Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

**Note:** This parameterization is good when input image is registered (e.g., face recognition).

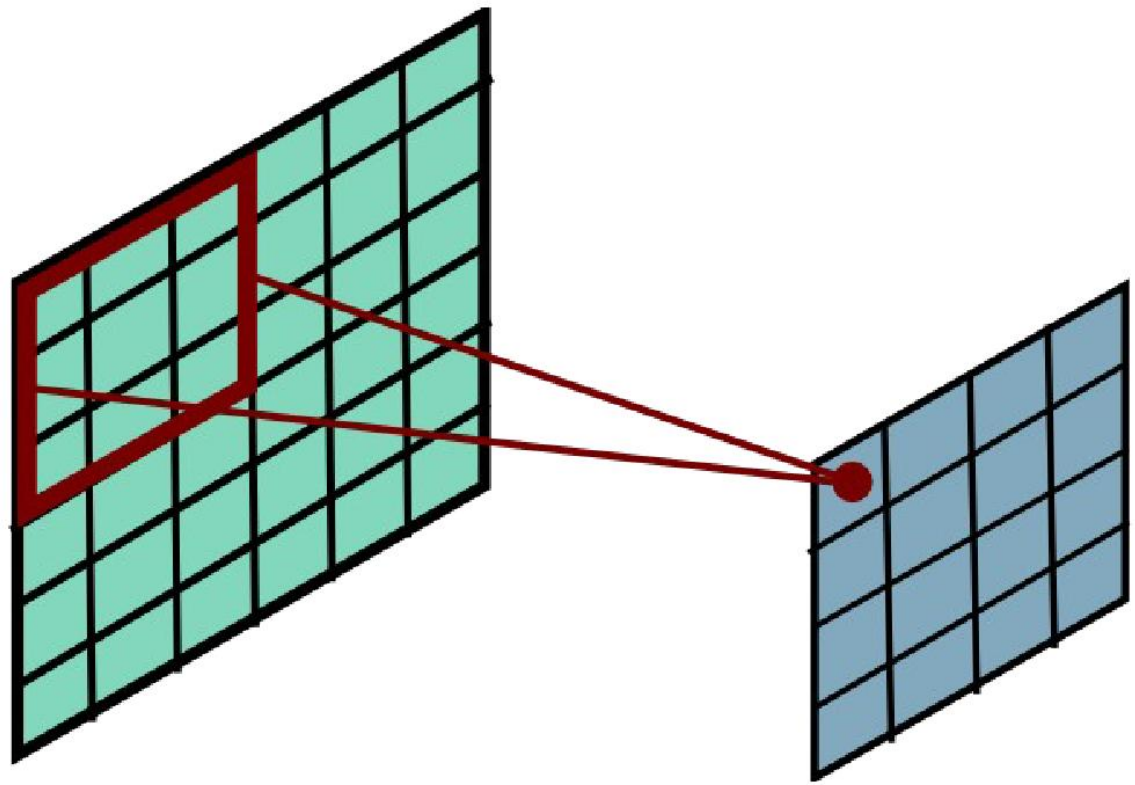
# Convolutional Layer



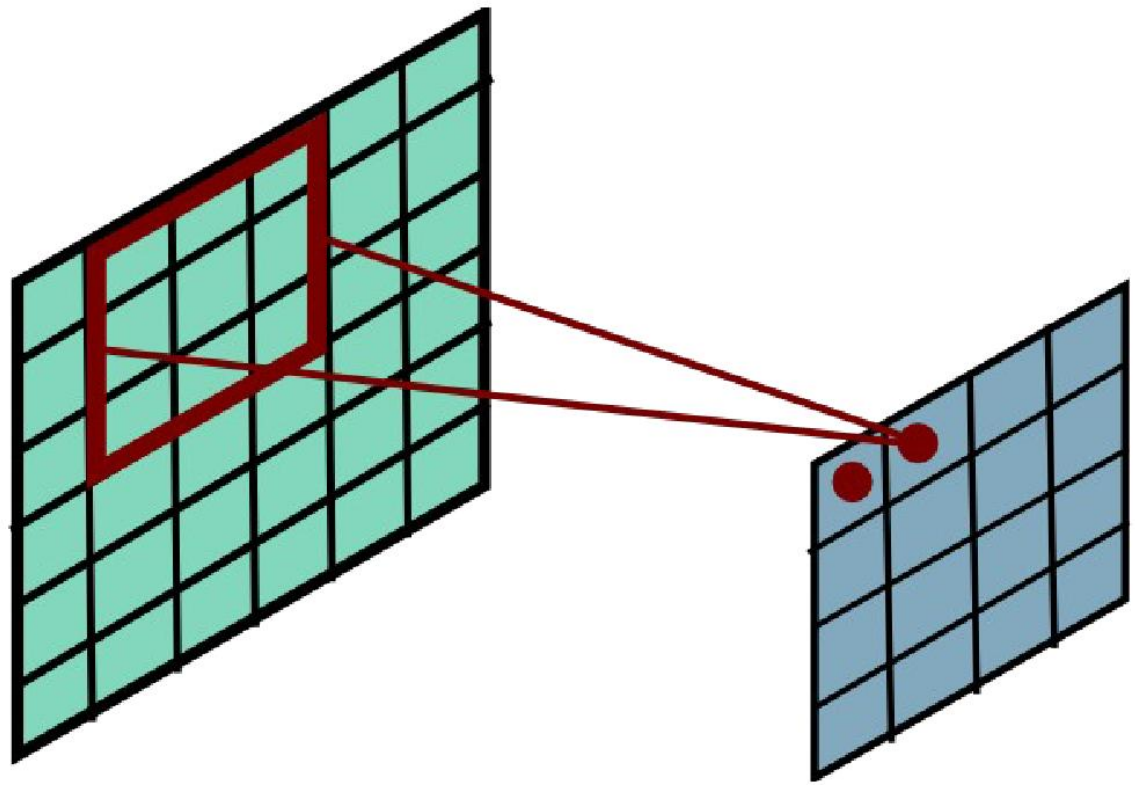
Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

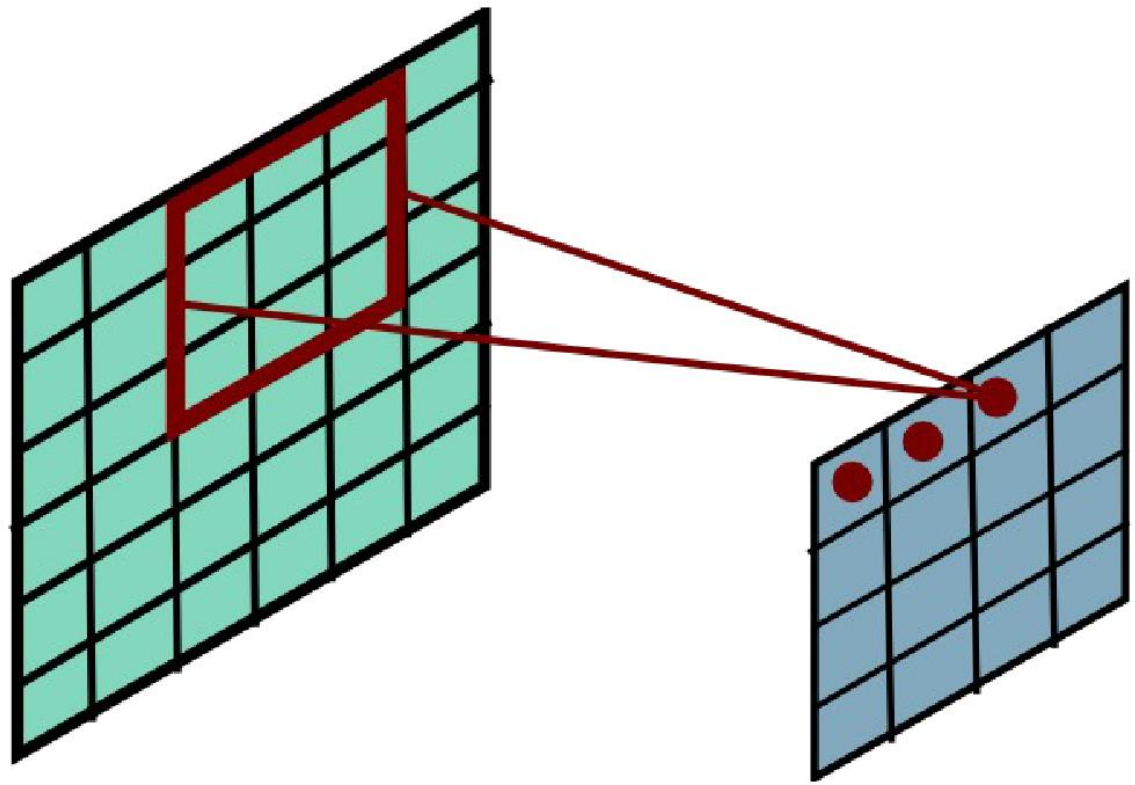
# Convolutional Layer



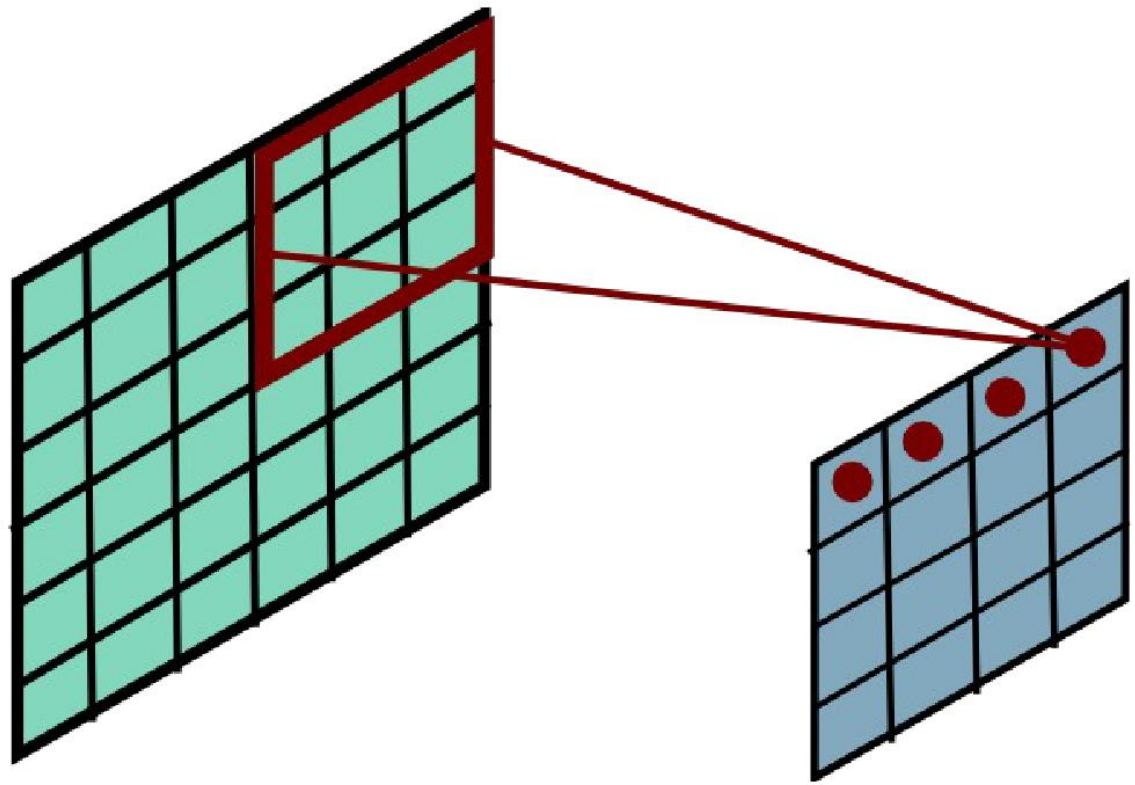
# Convolutional Layer



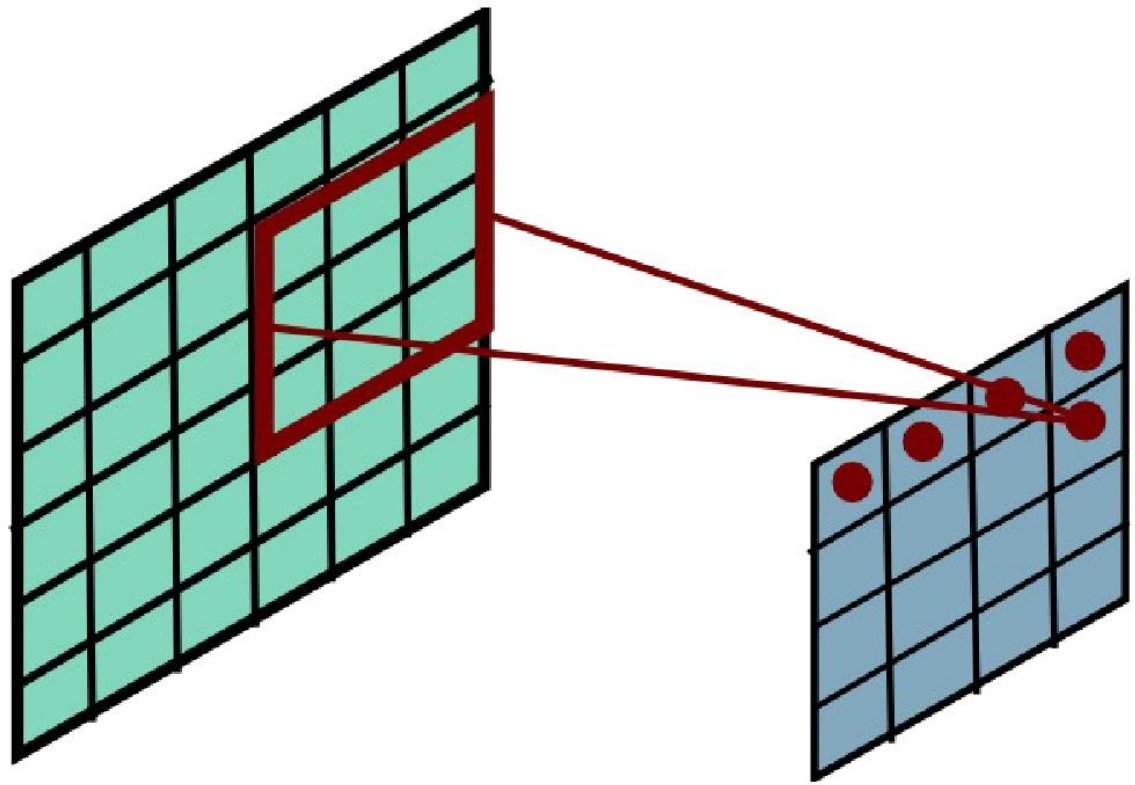
# Convolutional Layer



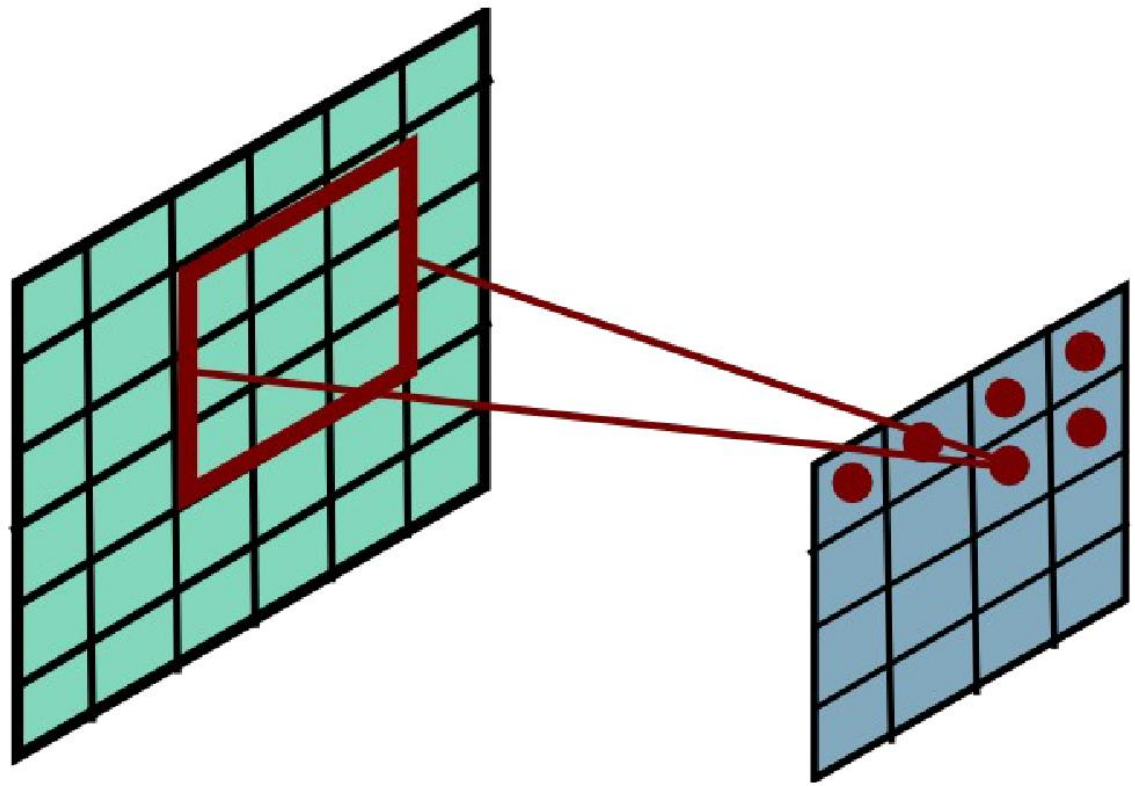
# Convolutional Layer



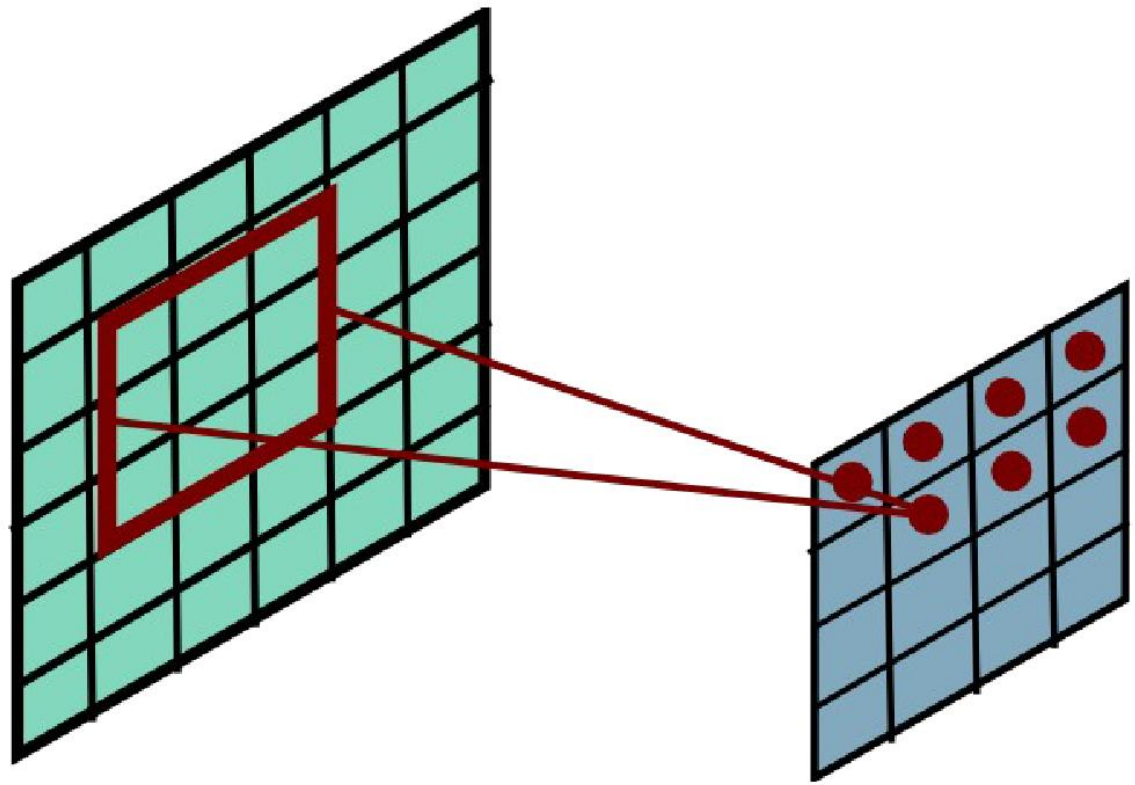
# Convolutional Layer



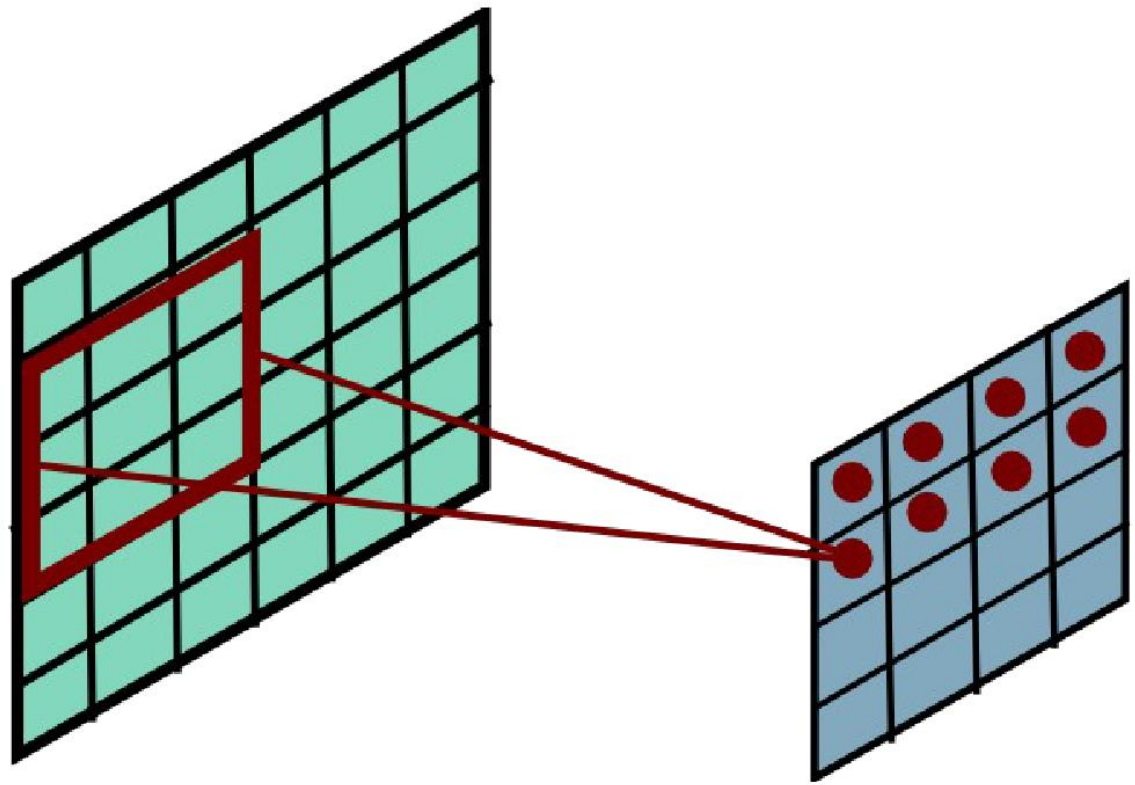
# Convolutional Layer



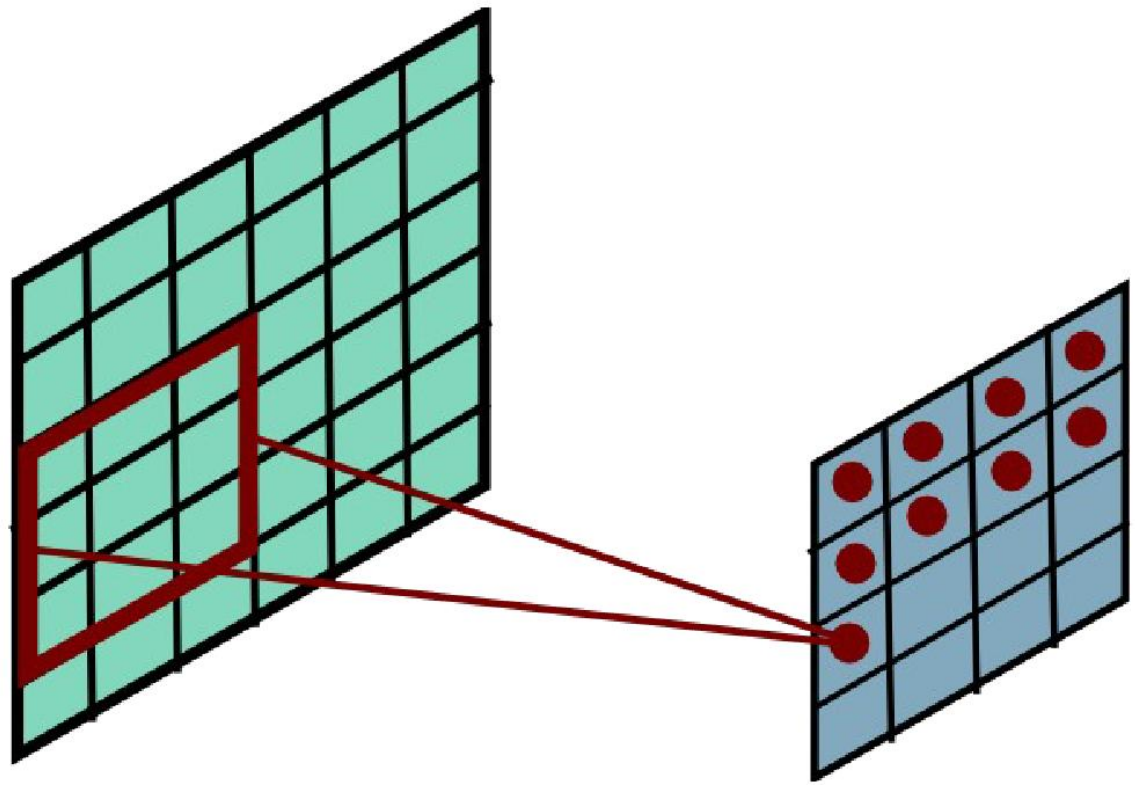
# Convolutional Layer



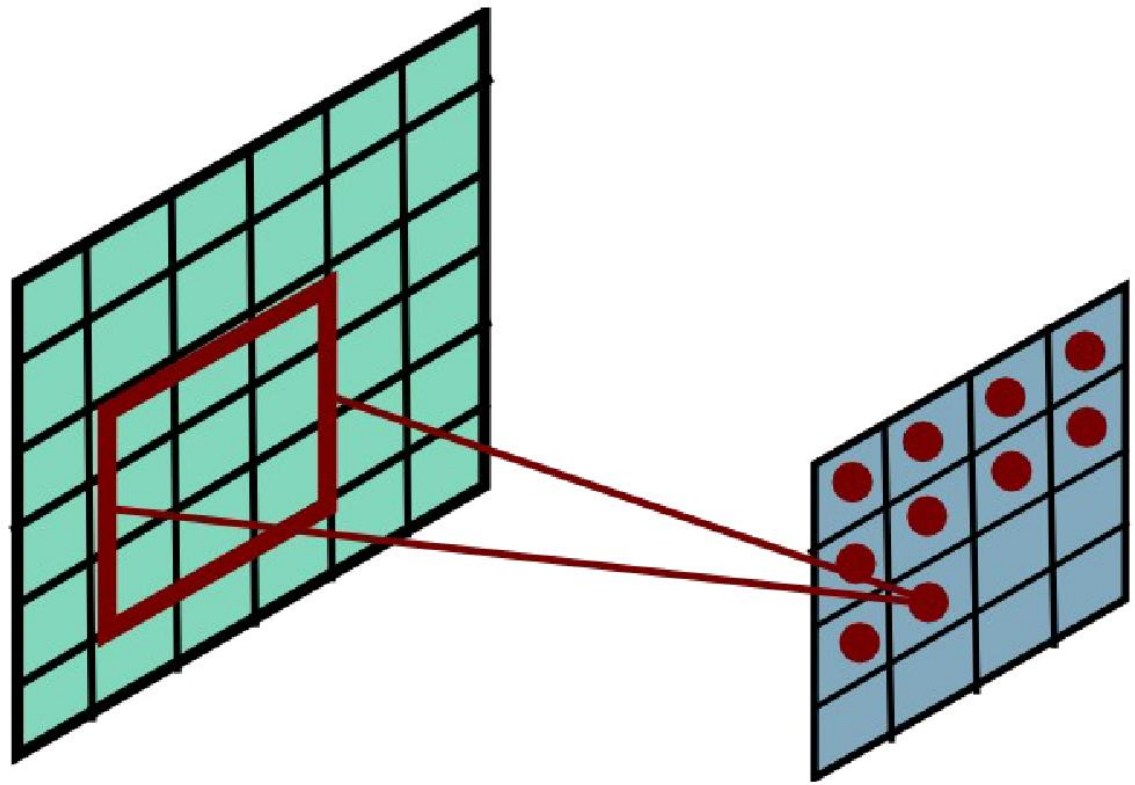
# Convolutional Layer



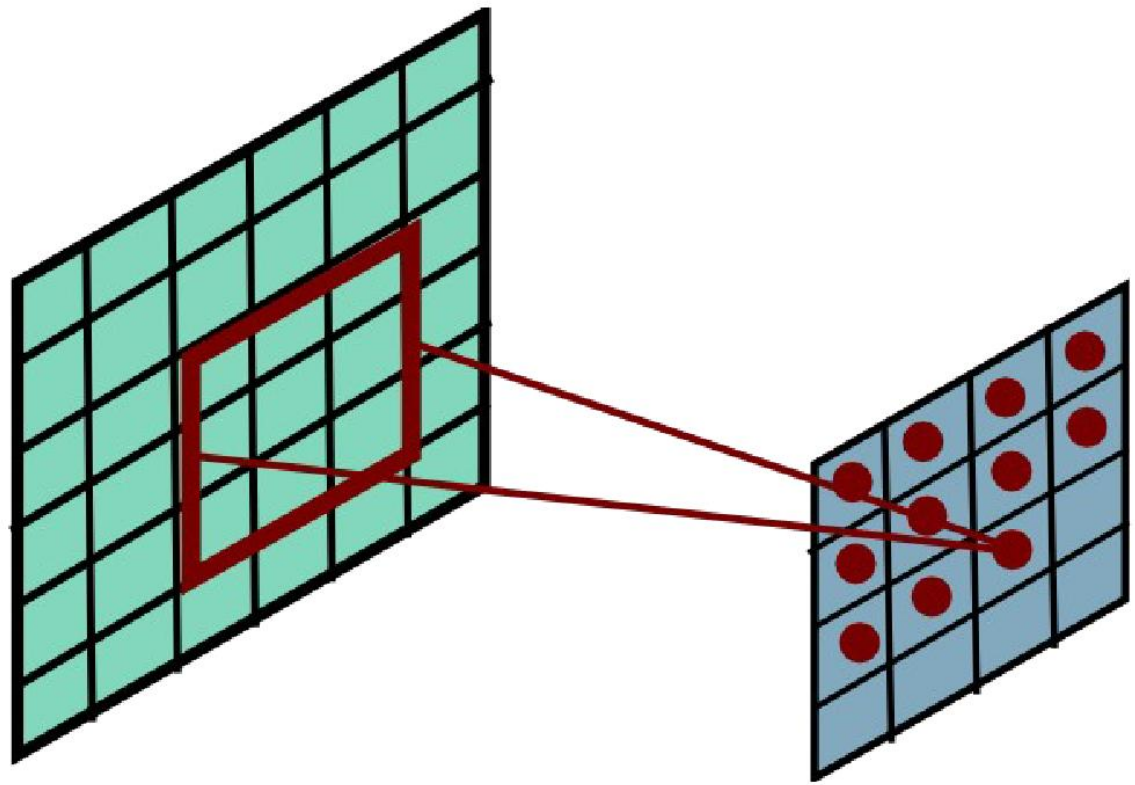
# Convolutional Layer



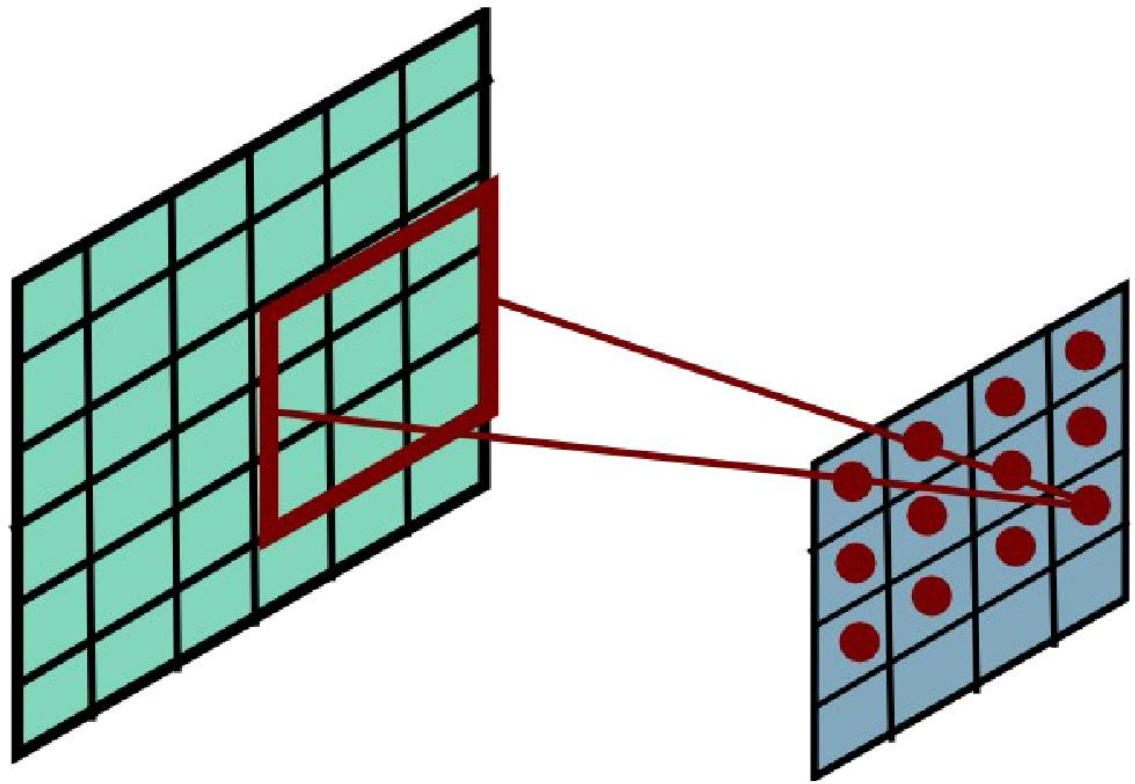
# Convolutional Layer



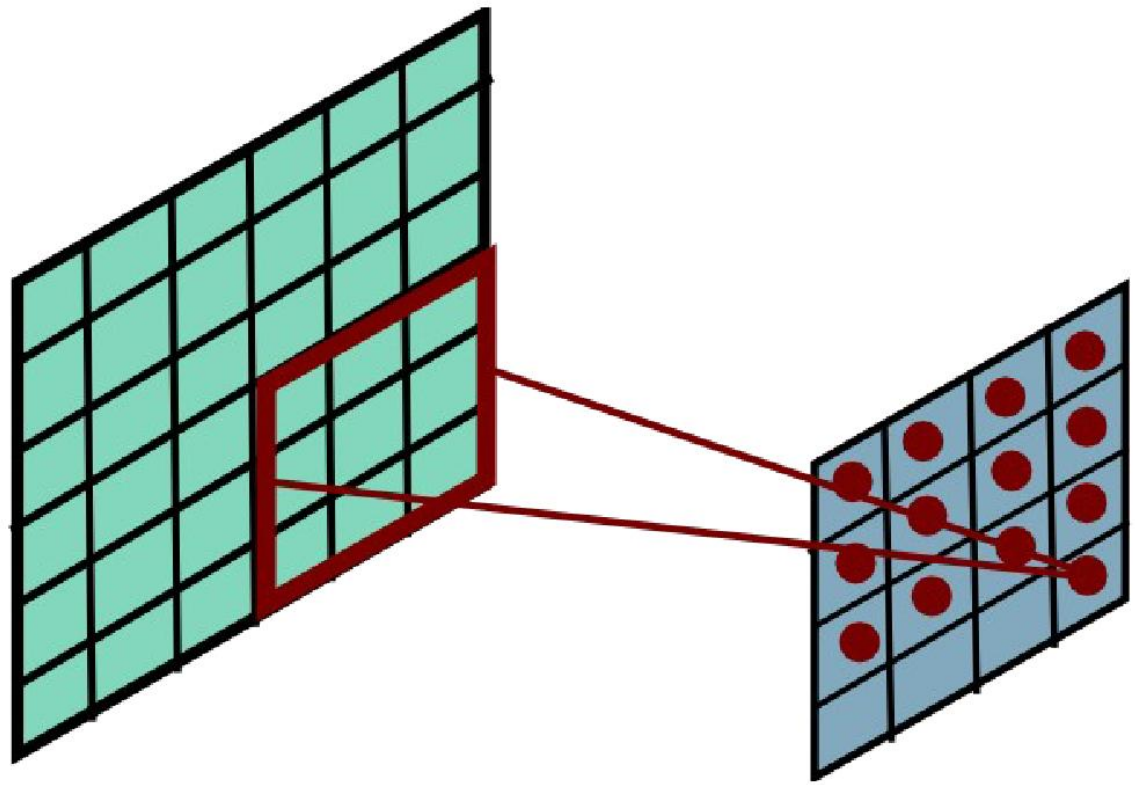
# Convolutional Layer



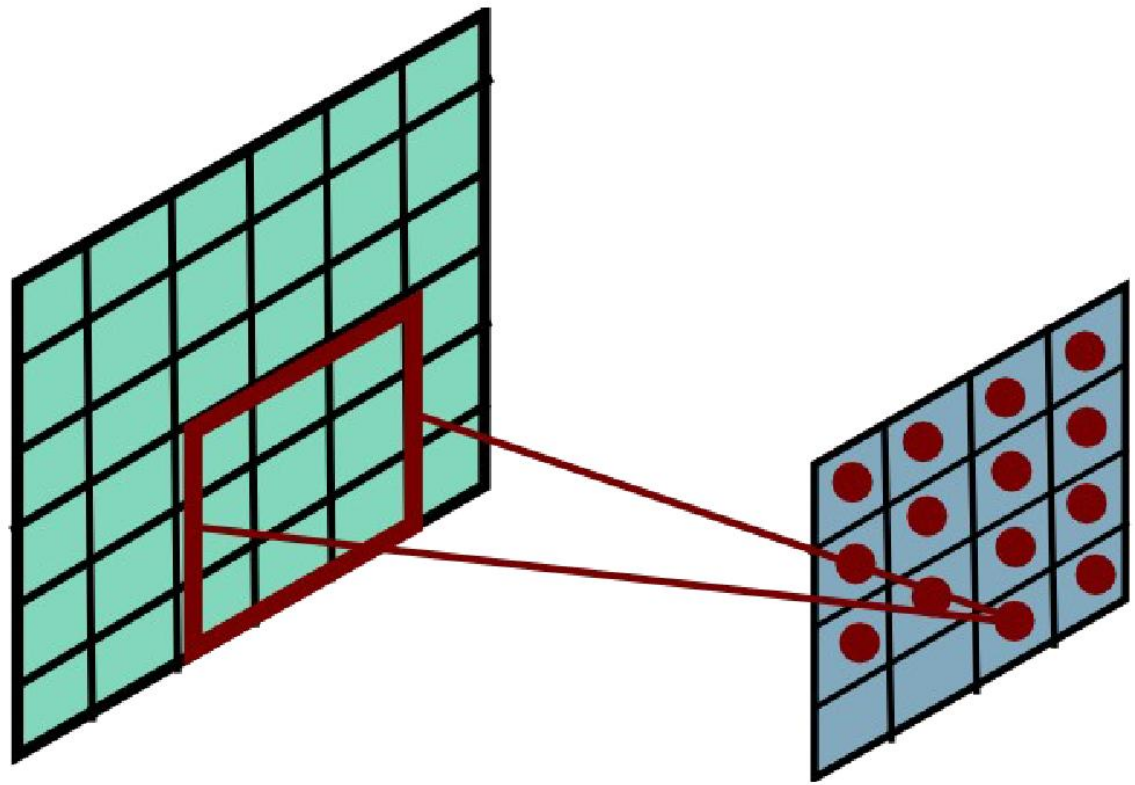
# Convolutional Layer



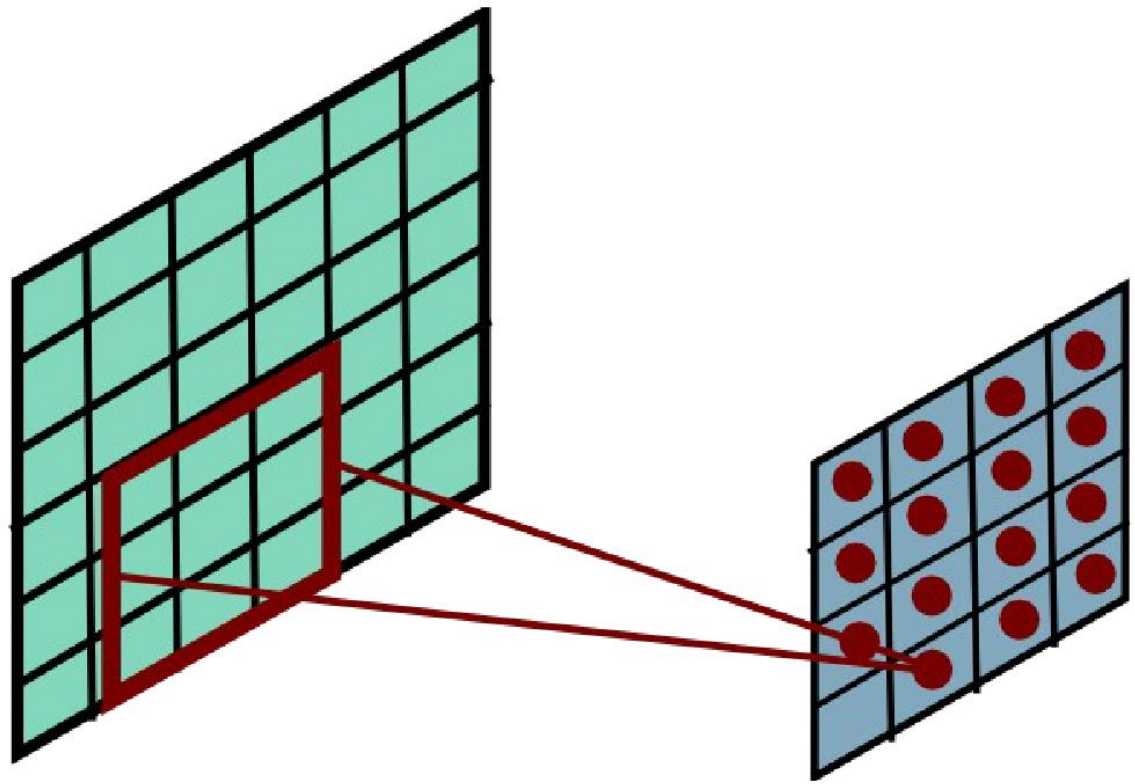
# Convolutional Layer



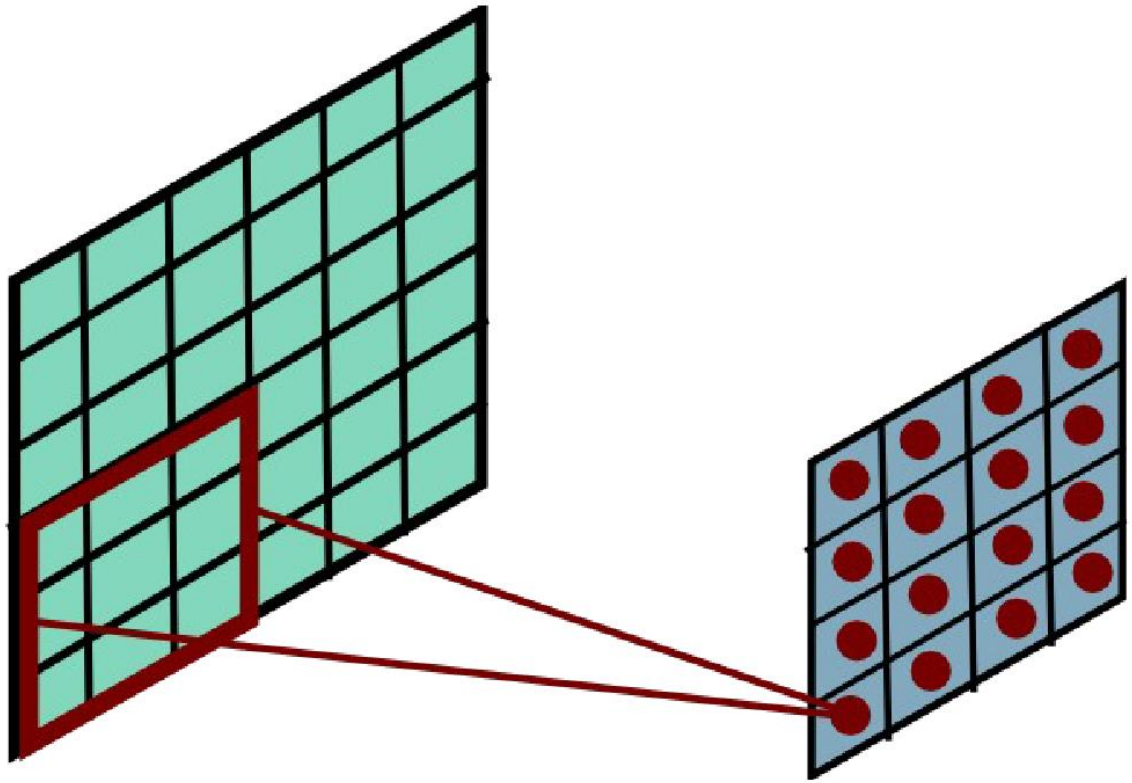
# Convolutional Layer



# Convolutional Layer

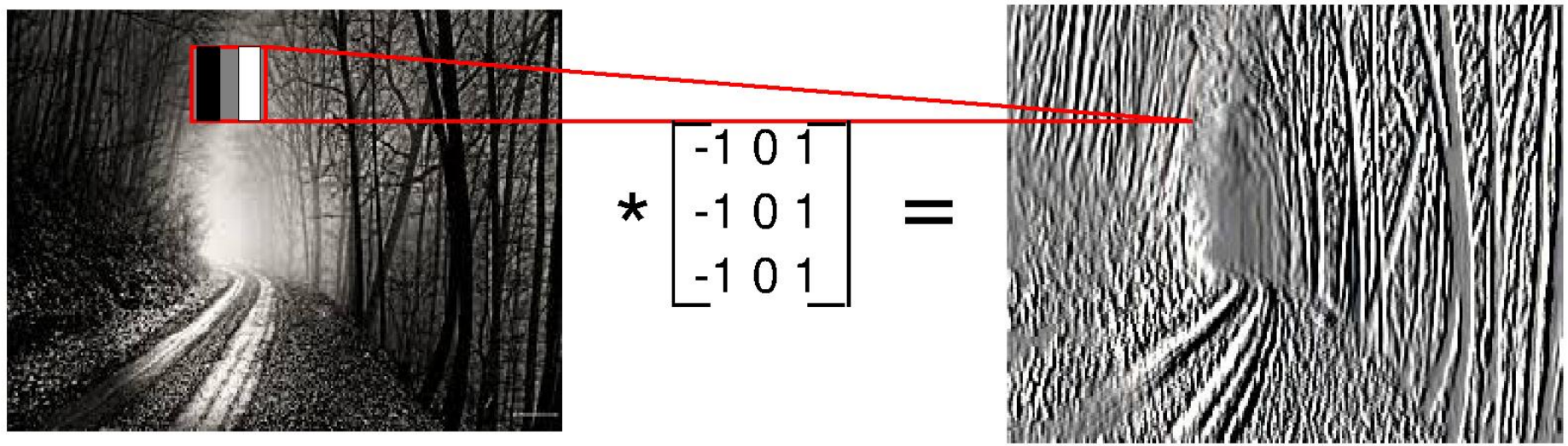


# Convolutional Layer

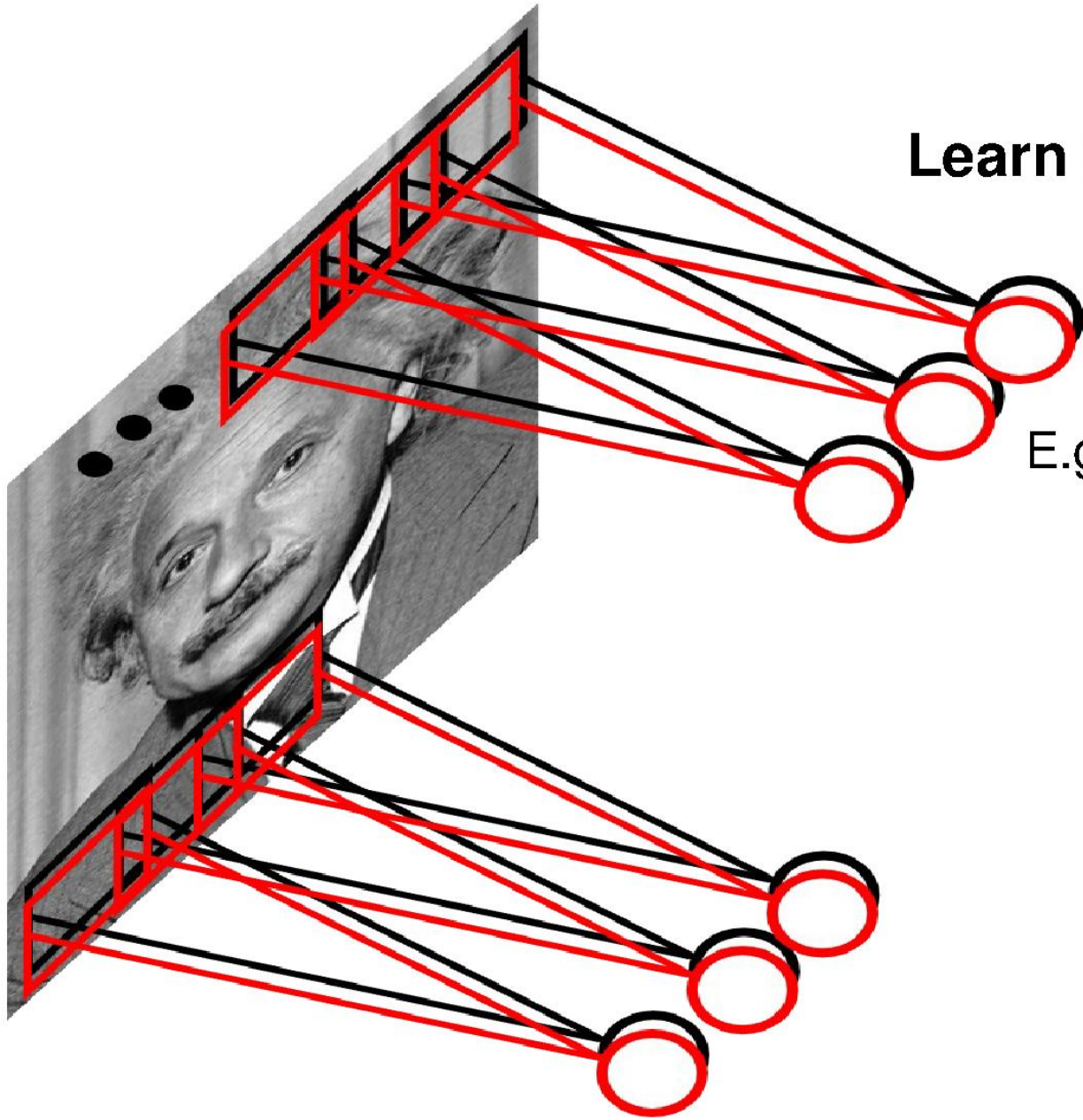


(Input size width — filter size +  2*Padding) / stride )+ 1
= output width of convolutional layer

# Convolutional Layer



# Convolutional Layer



**Learn** multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

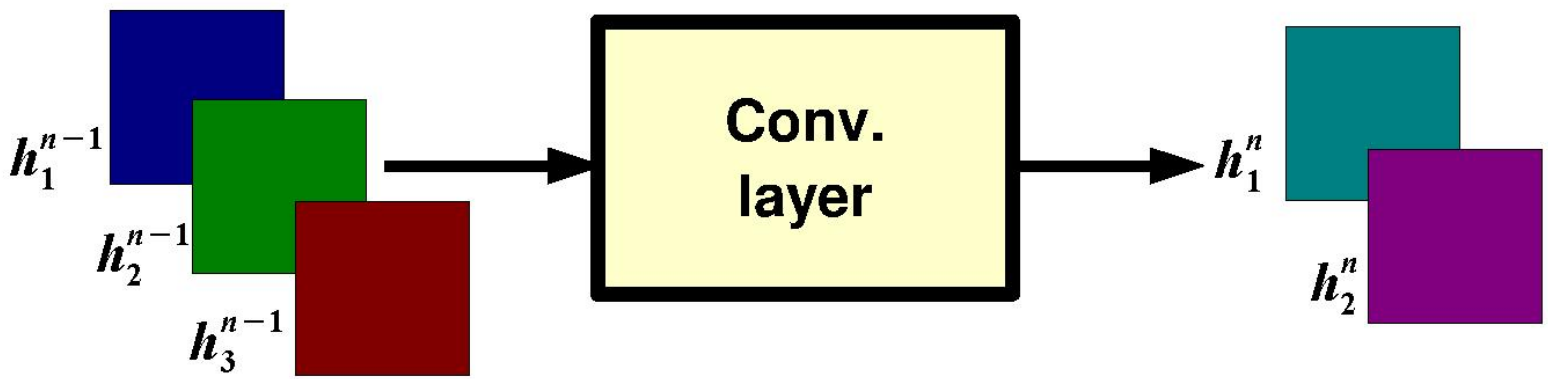
# Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output feature map

input feature map

kernel



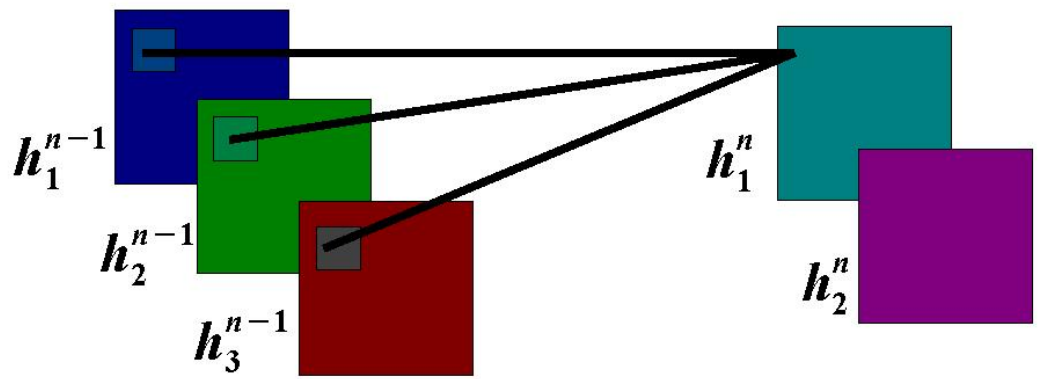
# Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output feature map

input feature map

kernel



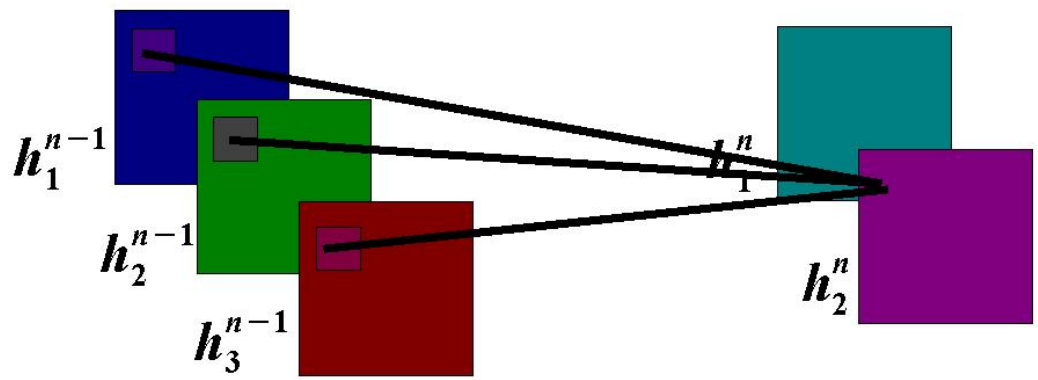
# Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output feature map

input feature map

kernel

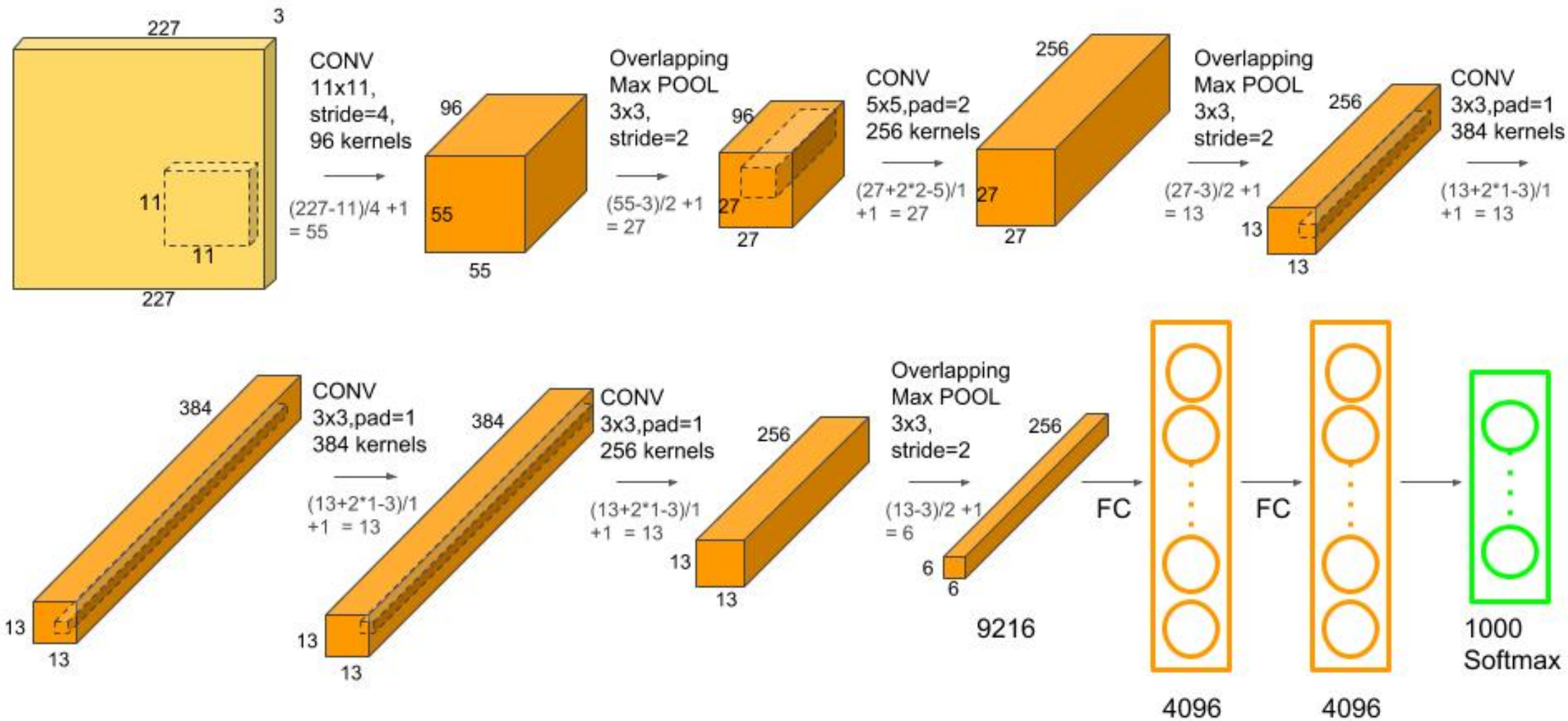


# Deep Convolutional Neural Networks-AlexNet

- The highlights of the paper
  - Use Relu instead of Tanh to add non-linearity. It accelerates the speed by 6 times at the same accuracy.
  - Use dropout instead of regularization to deal with overfitting. However the training time is doubled with the dropout rate of 0.5.
  - Overlap pooling to reduce the size of network. It reduces the top-1 and top-5 error rates by 0.4% and 0.3%, respectively.
- Properties
  - It has 60 million parameters and 650,000 neurons and took five to six days to train on two GTX 580 3GB GPUs
  - It contains 5 convolutional layers and 3 fully connected layers.
  - The image size in the following architecture chart should be 227 * 227

https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

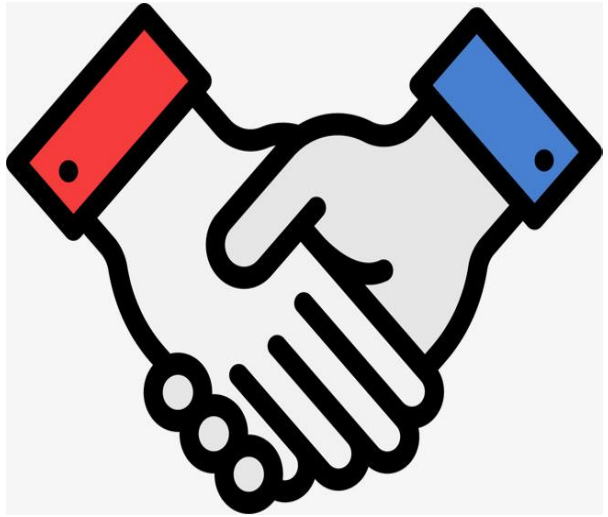
# Deep Convolutional Neural Networks



# Conclusions


# Conclusion

- Neural Networks
- Convolutional Neural Networks



# Thanks

zhengf@sustc.edu.cn