EE XXX

Design of Intelligent SoC Robot

http://ssl.kaist.ac.kr/class/18fall/socrobotdesign

Hoi-Jun Yoo

KAIST

AI for Robot

Outline

-Neural Network for Intelligent SoC Robot-

1. *Introduction of AI for Robot*
2. *Pattern Recognition (PR)*
3. *Convolution Neural Network (CNN)*
4. *Unsupervised Learning*
5. *Reinforcement Learning*
6. *Additional Learning Algorithm*

Outline

-Neural Network for Intelligent SoC Robot-

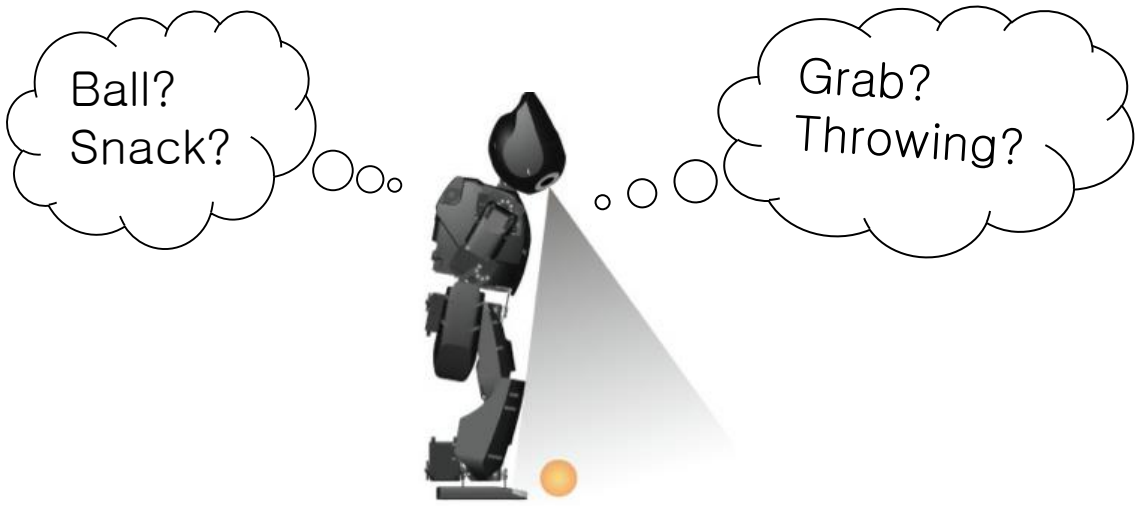
1. *Introduction of AI for Robot*
2. *Pattern Recognition (PR)*
3. *Convolution Neural Network (CNN)*
4. *Unsupervised Learning*
5. *Reinforcement Learning*
6. *Additional Learning Algorithm*

Why AI is required for Robot?

- For automatic understanding of images and video
- Types of CV algorithms for robot
  - Image Processing : Obtaining the image information from digital images
  - Image Recognition : Identifying the types of and numbers of objects from an image, and Learning

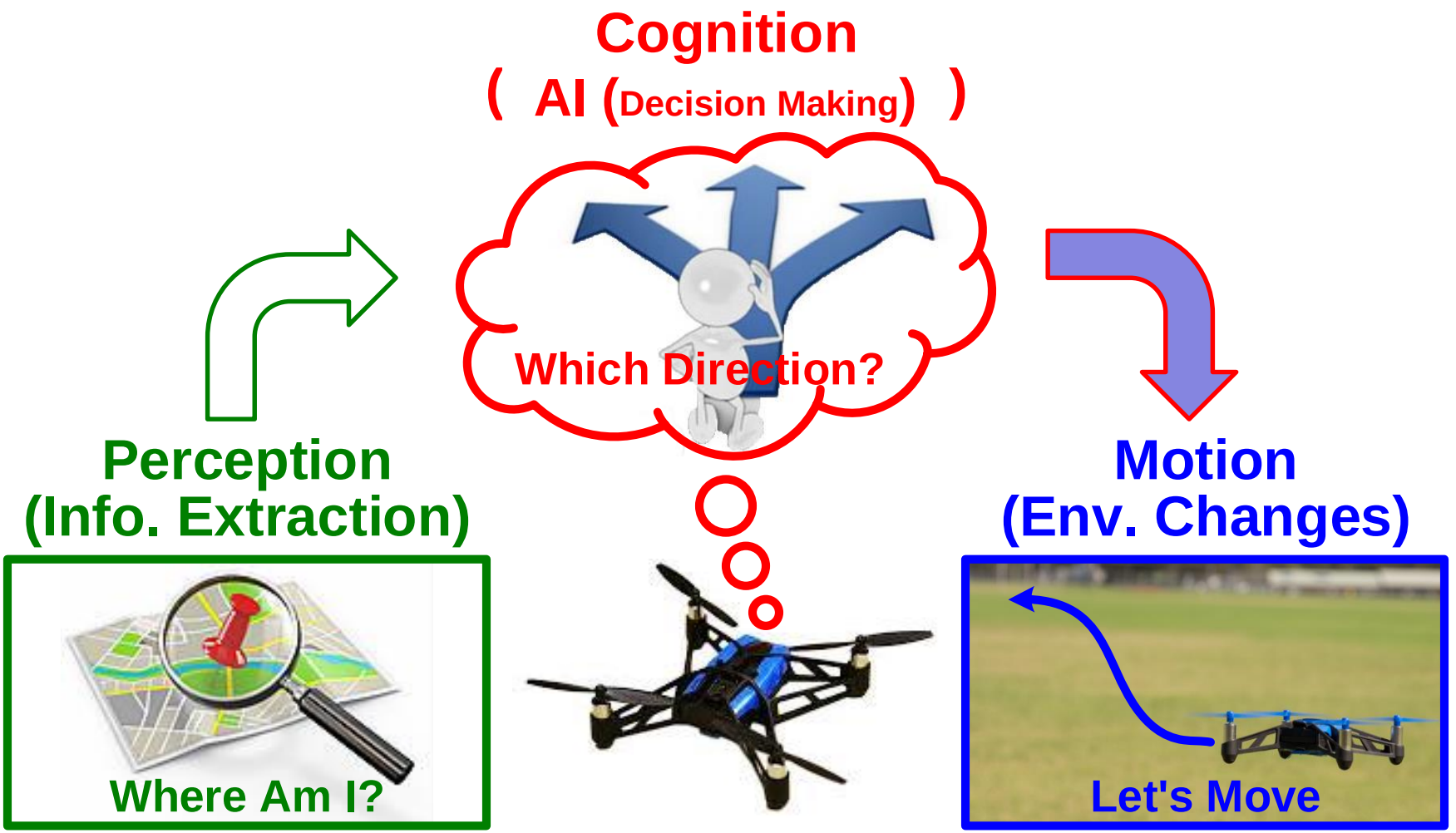
Previous lecture

Today's lecture



Artificial Intelligence for Robots

- Perception-Cognition-Motion Cycle



AI and Machine Learning

- Deep Neural Network
- Classical Conditioning
- Genetic Algorithm
- Reinforcement Learning
- Fuzzy Logic and Probabilistic Approach

Training

- Anything that is simple enough to have some rules that describe the basic behaviors

Eg)

Go forward unless

There is an obstacle on the left → turn right

There is an obstacle on the right →turn left

If you haven't seen anything on the left recently, head left.

Predicting Free Space

1. When a robot is exploring a new environment it may want to head towards free space. This is one way to choose where to go next while the robot will not crash into any objects.

2. If sensors are very noisy, a neural network guide the robot to find free space in a given direction.

3. And the robot can learn the whole thing by itself, without human intervention.

Predicting Free Space

1: Data Collection

The robot has a camera on its head. The robot turns its camera direction by +-30° and so collected a detailed visual data.

After the scan the robot tried to move in a randomly chosen direction as far as it could until an object was detected within 15 cm of the robot.

This was repeated many times.

Predicting Free Space

2: Pre-Processing

Image data (those centered around the front of the robot) were captured and two filters were used to clean up the sensor readings

- **The median filter** that was described in the lecture on image processing
- **A symmetry filter** that matched up sensor readings on either side of the robot and took the lower of the two values.

Predicting Free Space

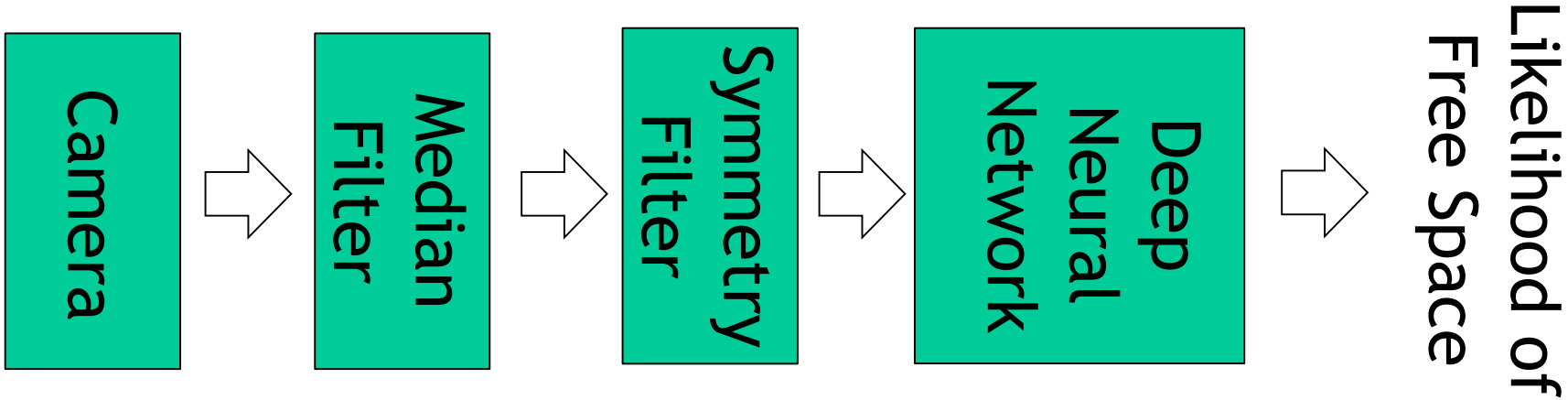
3: Deep Neural Network

A deep neural network with more than 5 neuron layers and > 100 nodes was used. This was trained using back-propagation.

The output of the network was biased so that it was more likely to estimate free space that may not. This made sure that the robot did not miss potentially interesting areas to explore.

Predicting Free Space

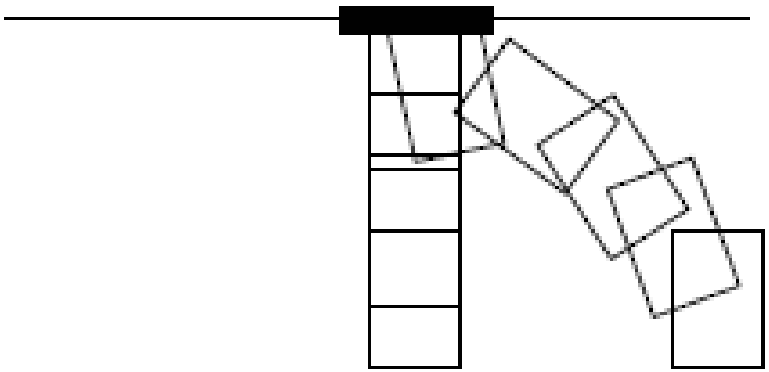
4: The Whole Solution



Homing

When a robot has to ‘dock’, fore example, with a charging station, how can the robot learn to drive into the correct place?

The robot should be able to approach the station from any angle.



Homing

A radial basis function (RBF) neural network is useful for this:

- Learns rapidly
- Can integrate the output of several basis functions

Method:

- Take inputs from the sonar sensors of the robot
- Produce outputs of angle and distance to charger

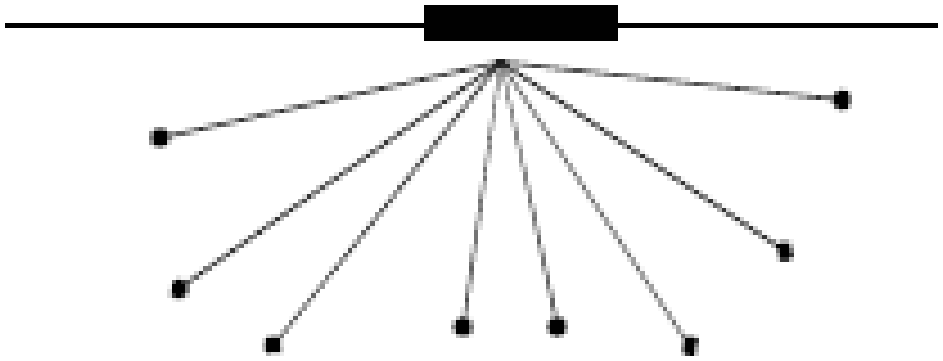
Homing

How can the robot learn this?

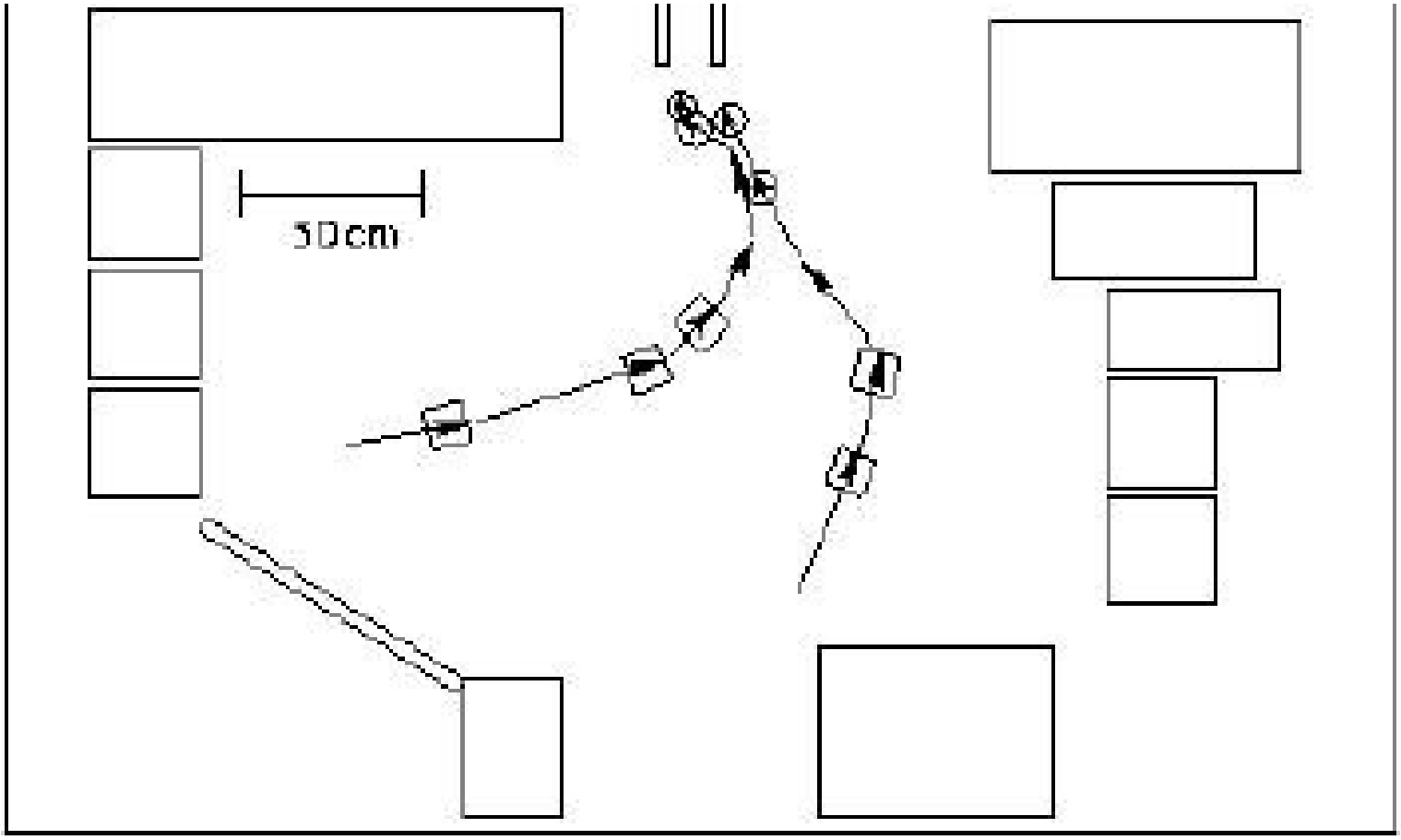
- Start at the charger
- Make lots of smaller (random) movements away from the charger, recording heading and distance travelled
- If the robot is in a place that looks different
  - Add a new RBF (weights = sonar perceptions)
  - Train the output weights to be the inverse heading and distance to get back the charger
- Repeat

Homing

To make this training process efficient the robot can drive out in a fan shape, changing the heading slightly (and the distance travelled) each time.



Homing: Results



Outline

-Neural Network for Intelligent SoC Robot-

1. *Introduction of AI for Robot*
2. ***Pattern Recognition (PR)***
   : What is Pattern Recognition?

3. *Convolution Neural Network (CNN)*
4. *Unsupervised Learning*
5. *Reinforcement Learning*
6. *Additional Learning Algorithm*

Pattern Recognition

- **Computer Vision**
  - Visual inspection, Automatic Target Recognition
  - Face recognition and Surveillance
- **Character recognition**
  - Zip Code, Bank Check
- **Computer aided diagnosis**
  - Medical imaging, EEG/ECG signal analysis
- **Speech recognition**
  - Human Computer Interaction, UI

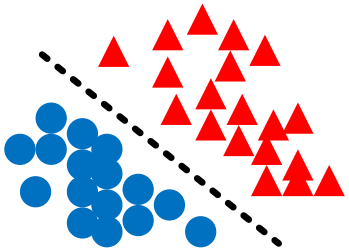
Features and Patterns

- **Feature**
  - Any distinctive aspect or property
  - Feature Vector: Combination of d features represented as a d-dimensional column vector
  - Feature Space: d-dimensional space defined by the feature vector
  - Scatter Plot: Objects are represented as points in feature space

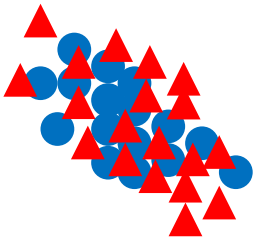
- **Pattern**
  - A composite of features defining an object
  - A pair of variables {x, w} where x is a feature vector, w is a label(the concept behind the observation)

Features

- **Good Feature Vector**
  - Ability to discriminate examples from different classes (Similar feature values for the same class)

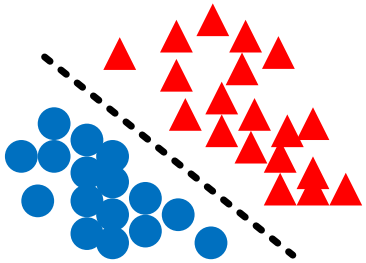


Good Features

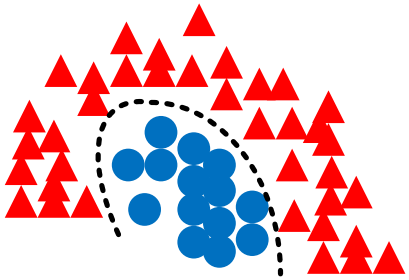


Bad Features

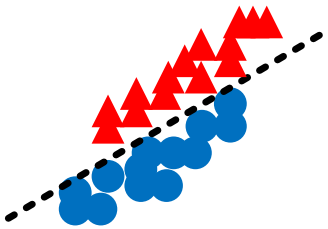
- **Feature Properties**



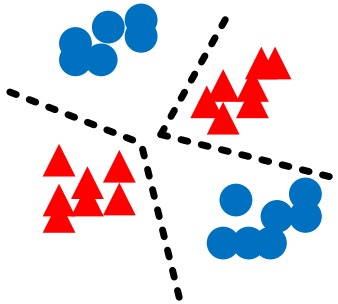
Linear Separability



Non-Linear Separability

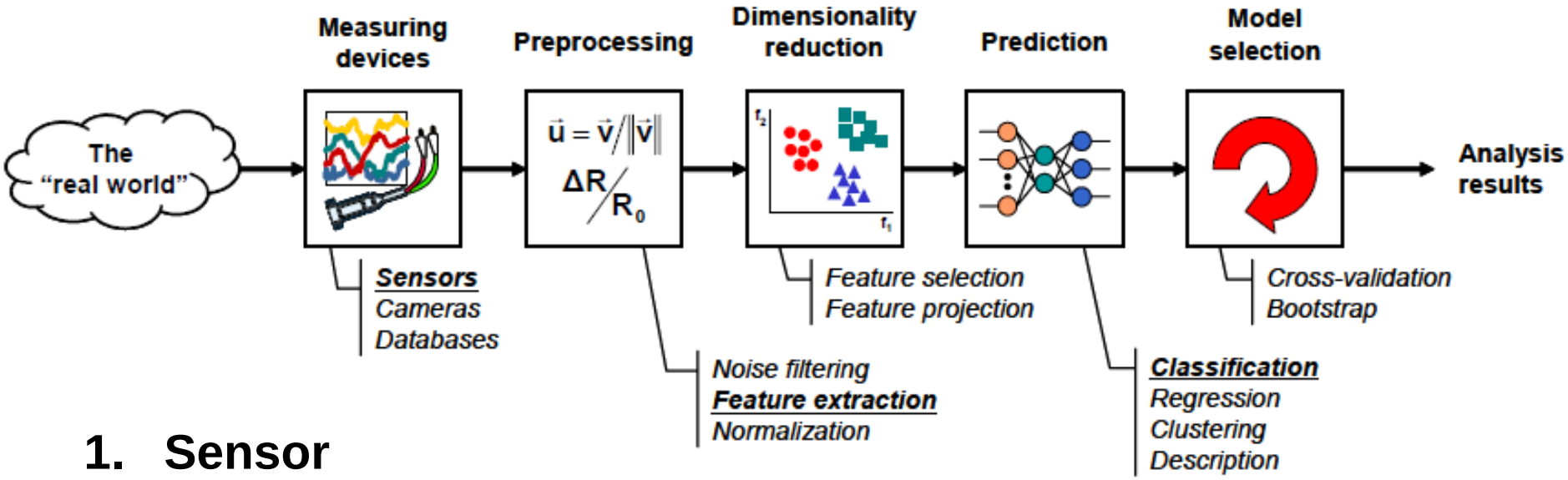


Highly Correlated Feature



Multi-Modal

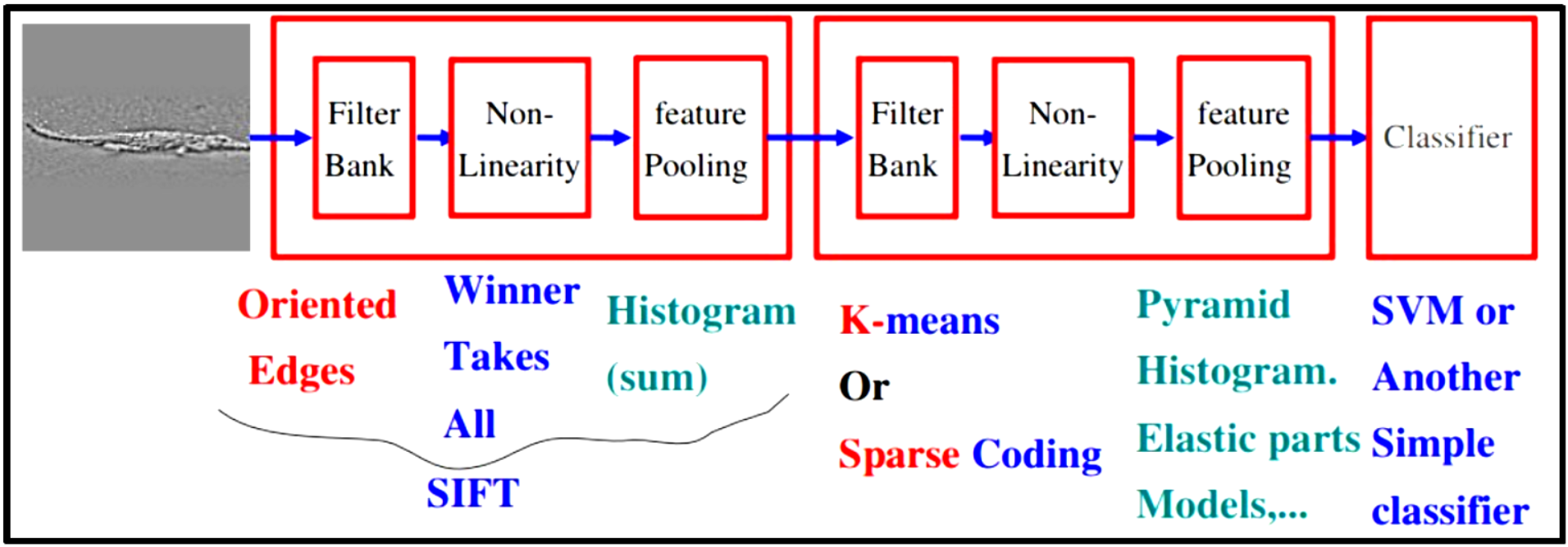
PR Pipeline



1. **Sensor**
2. **Pre-processing**
3. **Feature Extraction** ⟹ **Manual or Automatic (DNN)**
4. **Classification**

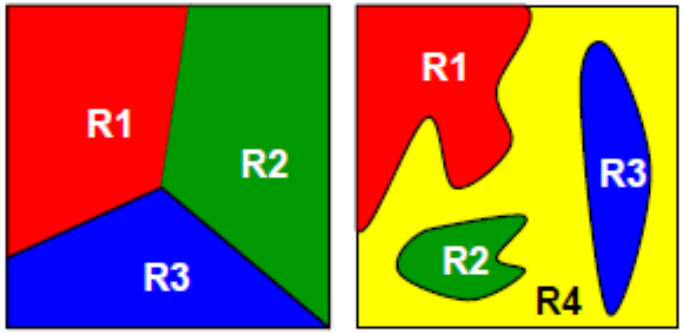
- **Training Set** ⟹ **A set of examples already classified or described**

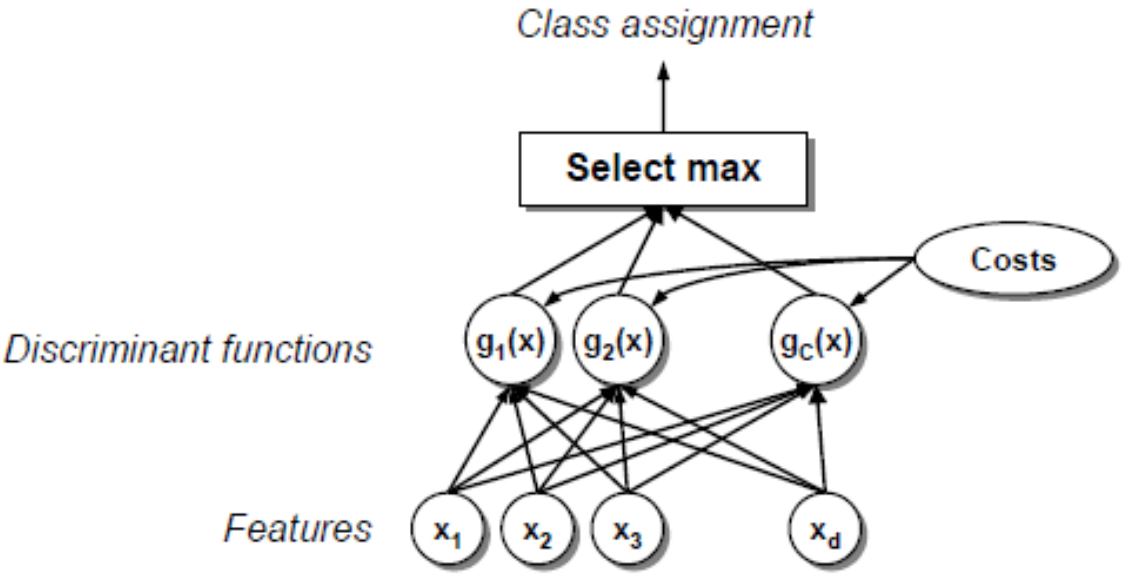
PR System


Filter Bank
Non-Linearity
feature Pooling
Filter Bank
Non-Linearity
feature Pooling
Classifier
Oriented Edges
Winner Takes All
Histogram (sum)
SIFT
K-means Or Sparse Coding
Pyramid Histogram. Elastic parts Models,...
SVM or Another Simple classifier

Classifier

- **Classifier: Partition feature space into class-labeled decision regions**
  - Decision Boundary: Borders between decision regions
  - Classification: Mapping a feature vector to a decision region
- **Classifier: a set of discriminant functions**
  - Classifier: Assign a feature vector x to a class w_i iff $g_i(x) > g_j(x)$

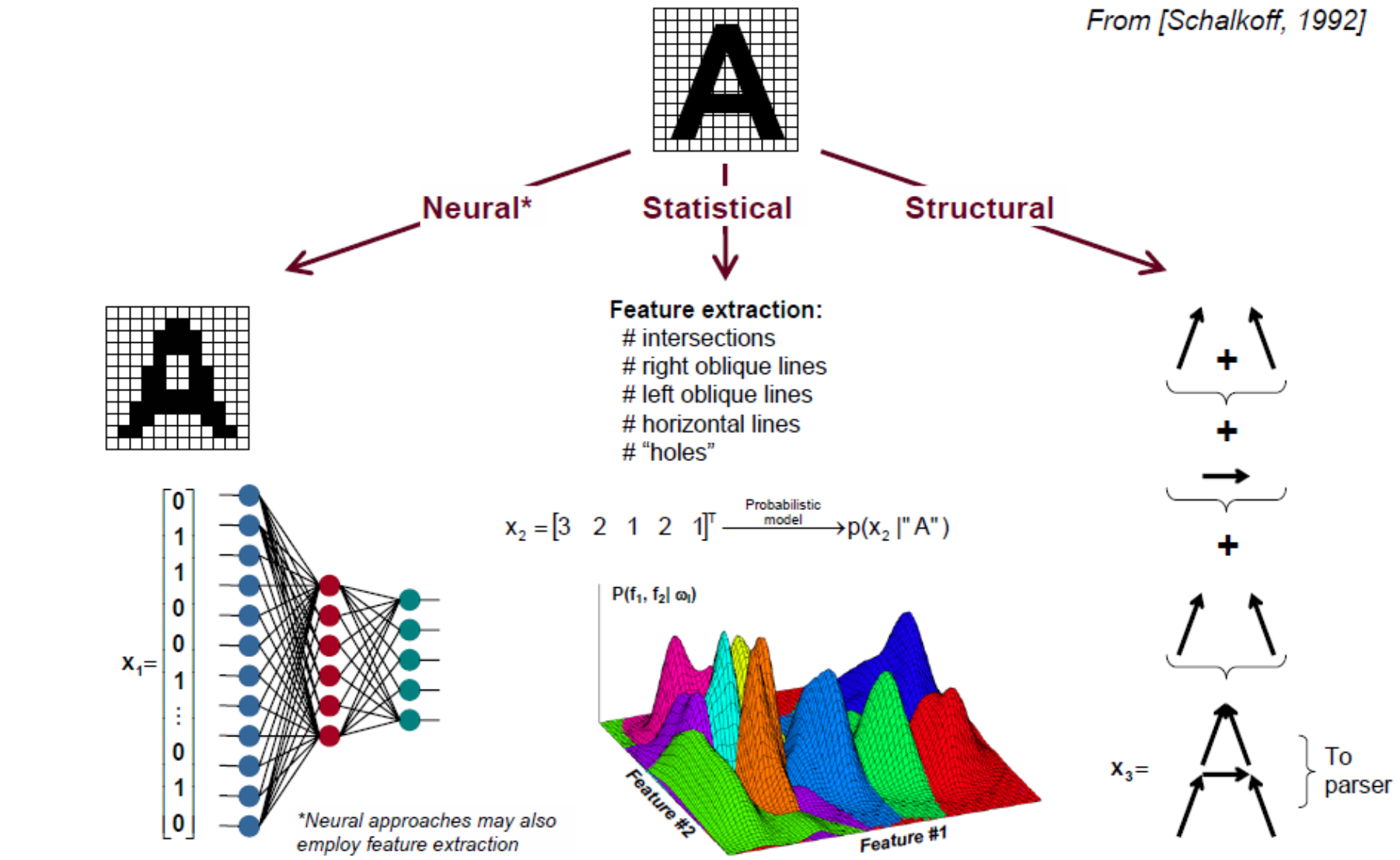




Types of PR

- **Statistical PR**
  - Classified based on n underlying statistical model of features
  - Statistical model is a class-conditional probability $Pr(x|c_i)$, probability of feature vector x given class c_i
- **Neural Network PR**
  - The response of a NN to an input stimuli(Pattern)
  - Trainable, non-algorithmic, black-box
- **Syntactic PR**
  - Classified based on measures of structural similarity
  - Knowledge is “formal grammars or relational descriptions(graphs)
  - Not only for classification but also for description

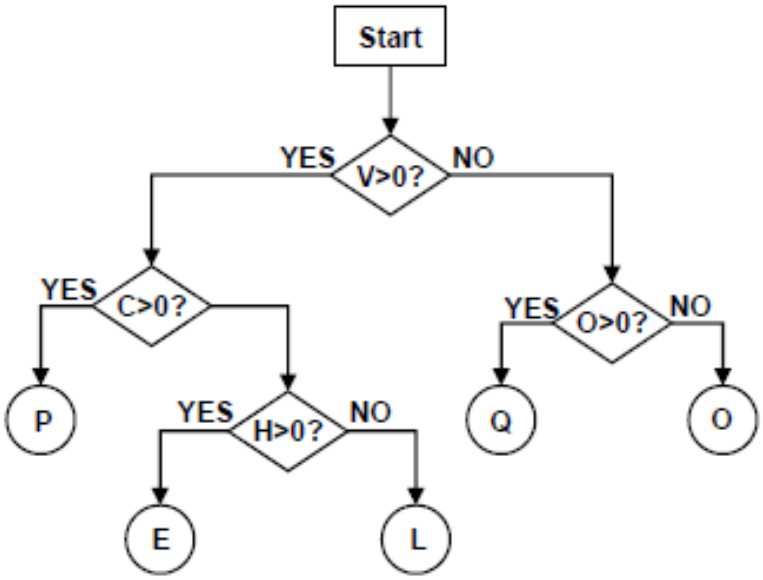
Comparison of PR Types



An Example

- **Recognizing the lletters L, P, O, E, Q**
  - Determine “Features”
  - Design a tree-structured classifier

Character	Features			
	Vertical straight lines	*Horizontal straight lines*	*Oblique straight lines*	*Curved lines*
L	*1*	*1*	*0*	*0*
P	*1*	*0*	*0*	*1*
O	*0*	*0*	*0*	*1*
E	*1*	*3*	*0*	*0*
Q	*0*	*0*	*1*	*1*



PR Design Cycle

1. **Data collection**
   - How many examples are enough?
2. **Feature choice**
   - Requires basic prior knowledge (Garbage in, garbage out)
3. **Model choice**
   - Statistical, Neural and Structural approaches
   - Parameter settings
4. **Training**
   - Supervised, unsupervised and reinforcemenr learning
5. **Evaluation**
   - Overfitting vs generalization

Classifiers

- "Distribution of Inputs" is Known
  - — Bayesian Decision Theory
  - — Linear & Quadratic Classifiers
- "Distribution of Inputs" is Unknown
  - — Parameter Estimation: Assume the density (Gaussian)
    - Maximum Likelihood
    - Bayesian Estimation
  - — Non-parametric Density Estimation: No assumption
    - Kernel Density Estimation
    - Nearest Neighbor

Learning Models

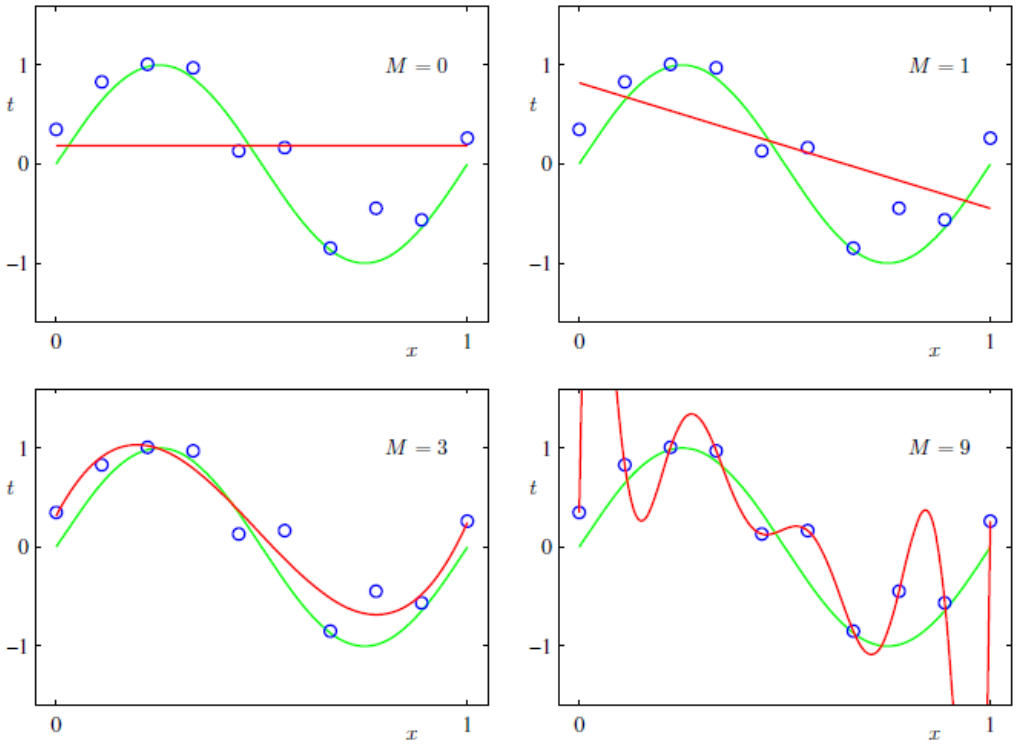
- **Model Based (Feature Based)**
  - **Linear Model → Curse of Dimensionality**
    - — Basis Function independent from inputs
    - — Linear equation of parameter θ
    - — Basis function can be polynomial or trigonometric
  - **Kernel Model**
    - — Basis Function is selected based on inputs
    - — # of parameters is not dependent on the dimension of input, but # of training samples.
  - **Hierarchical Model**
    - — Non-linear Model
    - — Sigmoid base function → Neural Network

Linear Model

$$f_w(x) = \sum_{j=1}^{M} w_j \phi_j(x) = W^T \Phi(x)$$

$$\Phi(x) = (\phi_1(x), \ldots, \phi_M(x))^T \qquad W = (w_1, \ldots, w_M)^T$$

$$\phi(x) = (1, x, x^2, \ldots, x^{M-1})^T$$



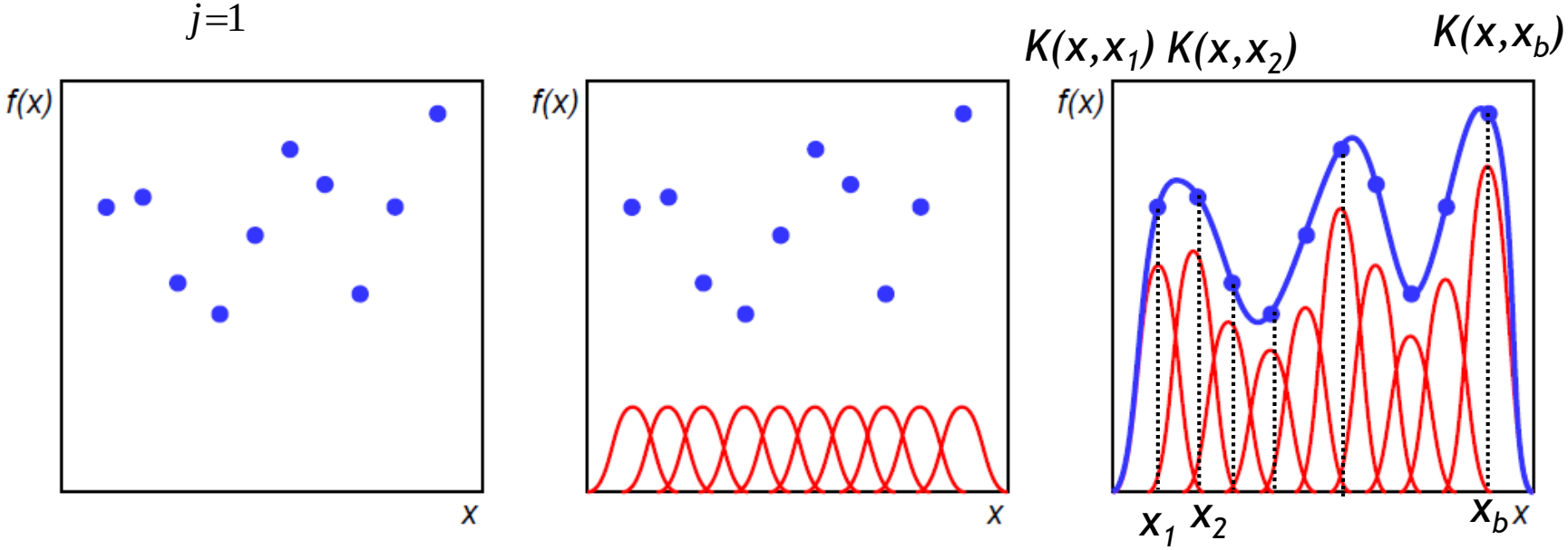
Kernel Model

$$f_\theta = \sum_{j=1}^{n} \theta_j K(x, x_j)$$

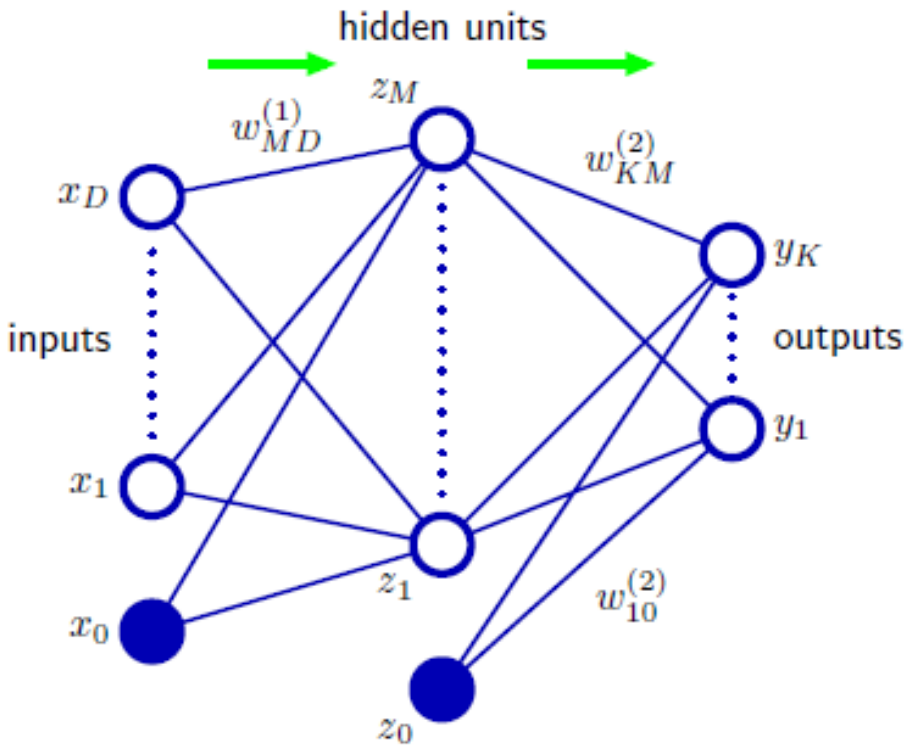
$$K(x, c) = \exp\left(-\frac{\|x - c\|^2}{2h^2}\right)$$

Eg, Gauss Kernel

$$f_\theta = \sum_{j=1}^{n} \theta_j K(x, c_j)$$



Hierarchical Model: Neural Network



$$f_k(x;w) = \sum_{j=1}^{b} \alpha_j \phi(x;w_j)$$

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

Basis function:

$$\phi(x;w_j) = \sigma(a_j)$$

$$= \sigma\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right)$$

$$\sigma(a) = \frac{1}{1+\exp(-a)} \quad \text{or} \quad \tanh(a) \quad \text{or} \quad \phi(x;c,h) = \exp\left(-\frac{\|x-c\|^2}{2h^2}\right)$$

Hierarchical Model: NN

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \quad \text{where} \quad z_j = \sigma(a_j)$$

$$f_k(x; w) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} \sigma\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

$$= \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} \sigma\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i \right) \right) \quad \text{where } x_0 = 1$$

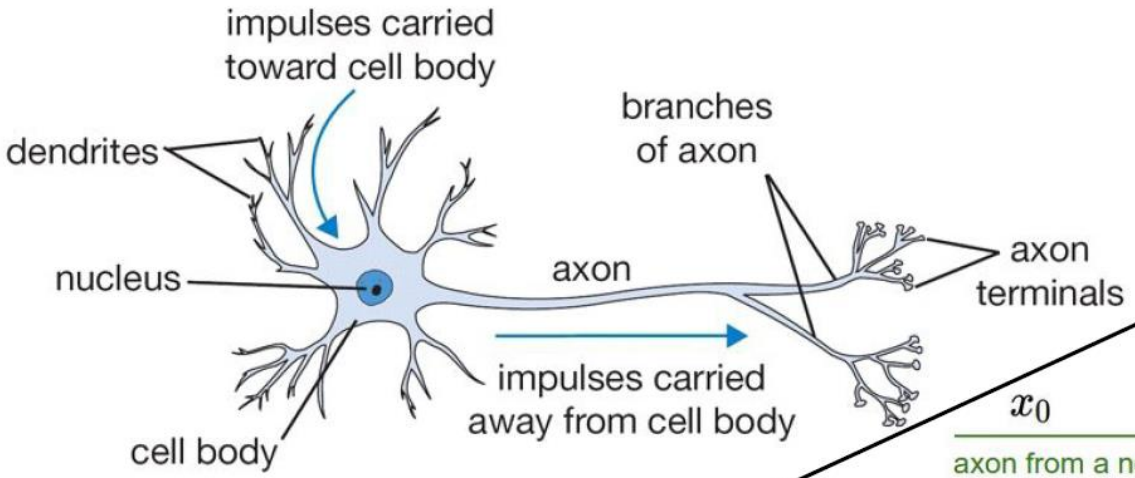
No 1:1 Mapping between function f and parameters α, w, and x_0

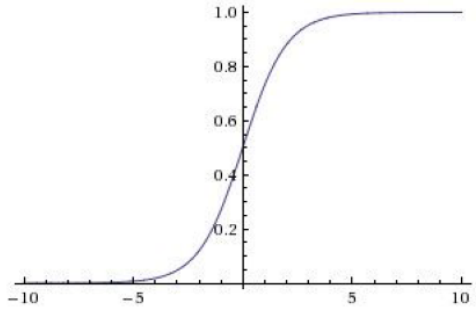
Outline

-Neural Network for Intelligent SoC Robot-

1. *Introduction of AI for Robot*
2. ***Pattern Recognition (PR)***
   : Neural Network – Feature Classifier
3. *Convolution Neural Network (CNN)*
4. *Unsupervised Learning*
5. *Reinforcement Learning*
6. *Additional Learning Algorithm*

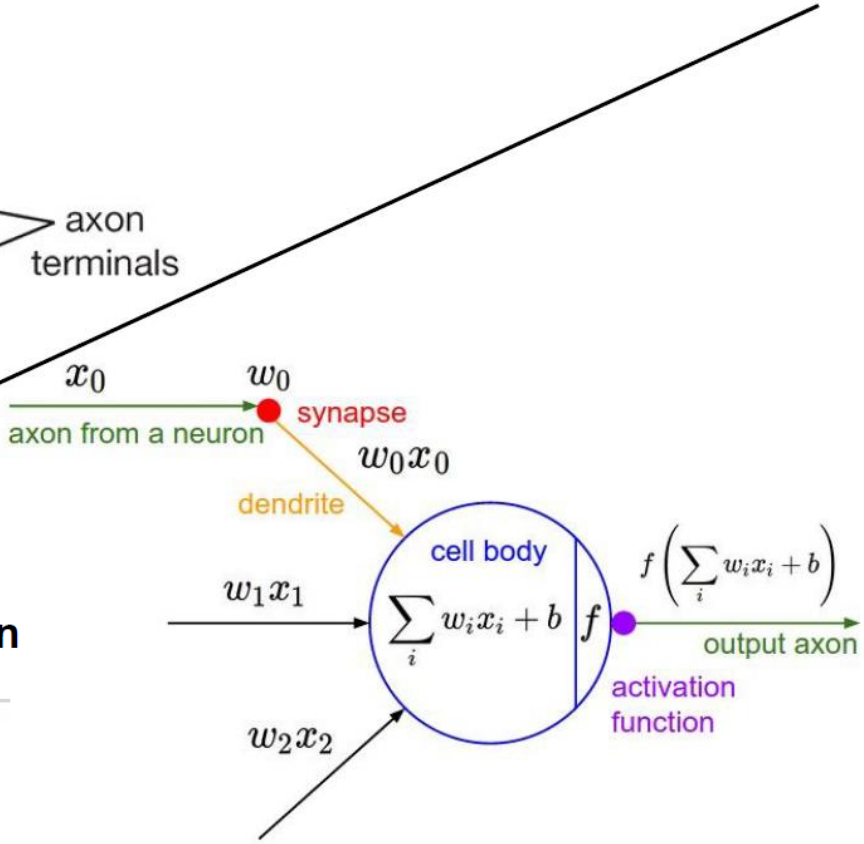
Inspiration from Neurobiology



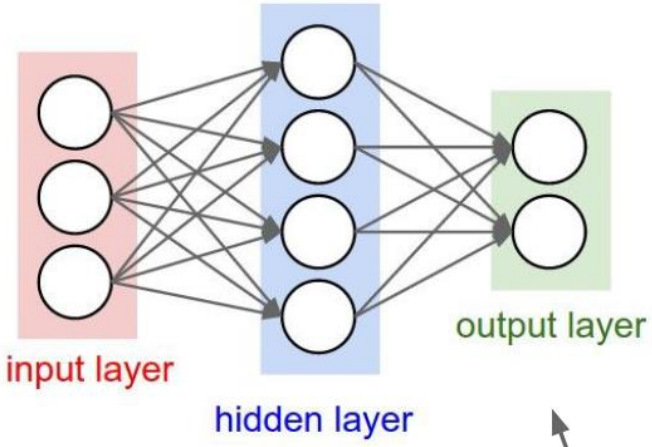


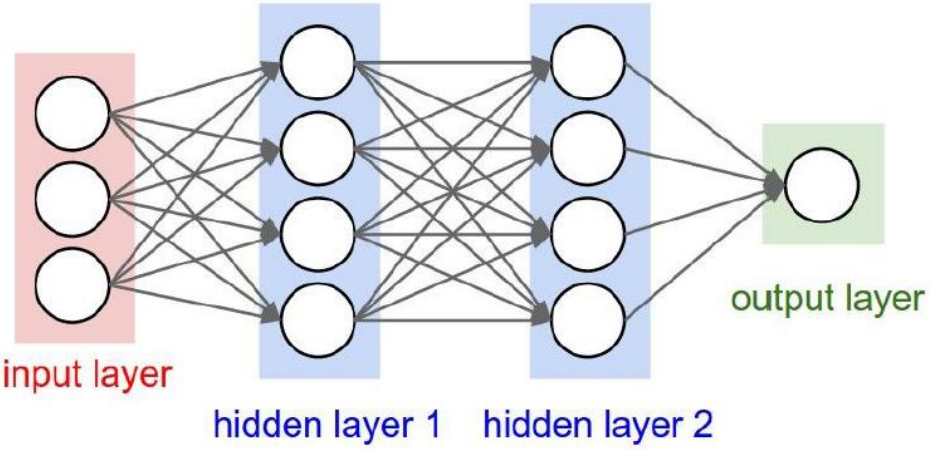
sigmoid activation function

$$\frac{1}{1+e^{-x}}$$



Neural Network Architecture





"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

Outline

-Neural Network for Intelligent SoC Robot-

1. *Introduction of AI for Robot*
2. ***Pattern Recognition (PR)***
   : Learning the Neural Network

3. *Convolution Neural Network (CNN)*
4. *Recurrent Neural Network (RNN)*
5. *Unsupervised Learning*
6. *Reinforcement Learning*
7. *Additional Learning Algorithm*

Perceptron Learning Rule

$$y = \sum_{i=1}^{N} w_i x_i + b$$

Hebbian Learning: $w_i^{new} = w_i^{old} + \eta \cdot y x_i$

Delta Learning: $w_i^{new} = w_i^{old} + \eta \cdot (r - y) x_i$

LSE(Least Square Error) Learning:

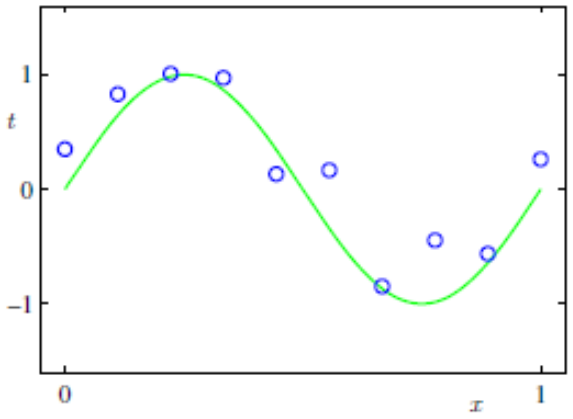
$$L = -\frac{1}{2} \sum_{j=1}^{M} (r_j - y_j)^2$$

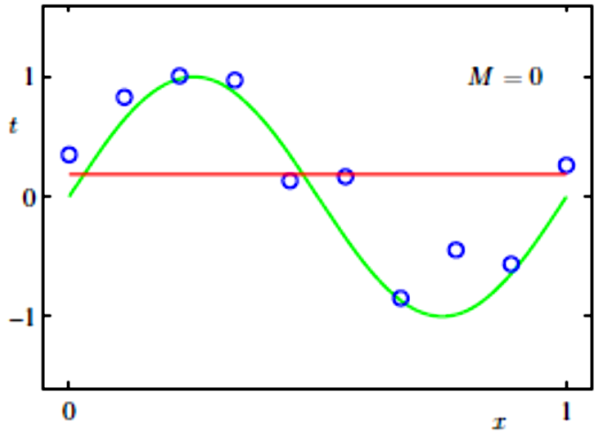
$$w_i^{new} = w_i^{old} + \eta \cdot \frac{\partial L}{\partial w_i} = w_i^{old} + \eta \cdot \frac{\partial L}{\partial y_j} \cdot \frac{\partial y_j}{\partial w_i}$$

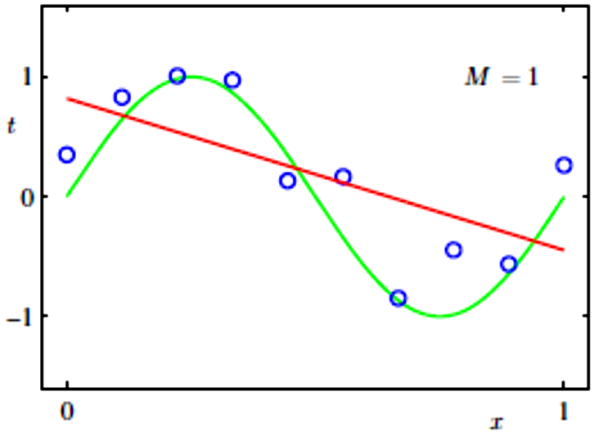
$$= w_i^{old} + \eta \cdot \sum_{j=1}^{M} (r_j - y_j) \cdot \frac{\partial y_j}{\partial w_i}$$

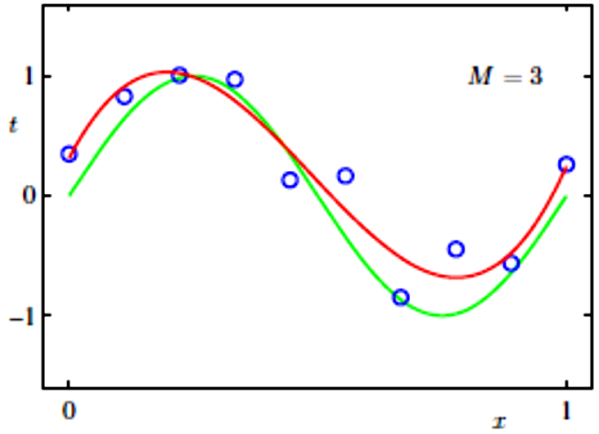
Overfitting and Underfitting

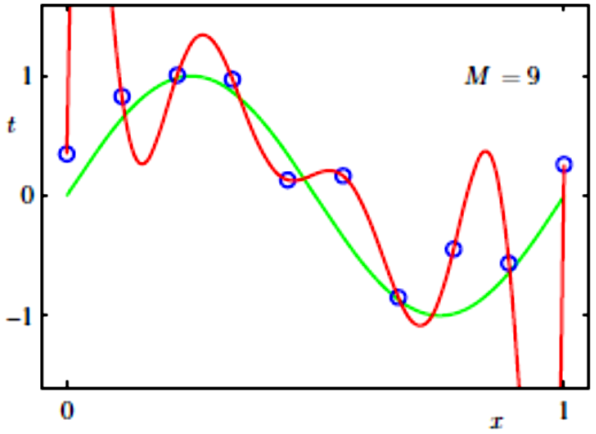
$$y(x,w) = w_0 + w_1 x + w_2 x^2 + ... + w_M x^M = \sum_{j=0}^{M} w_j x^j$$



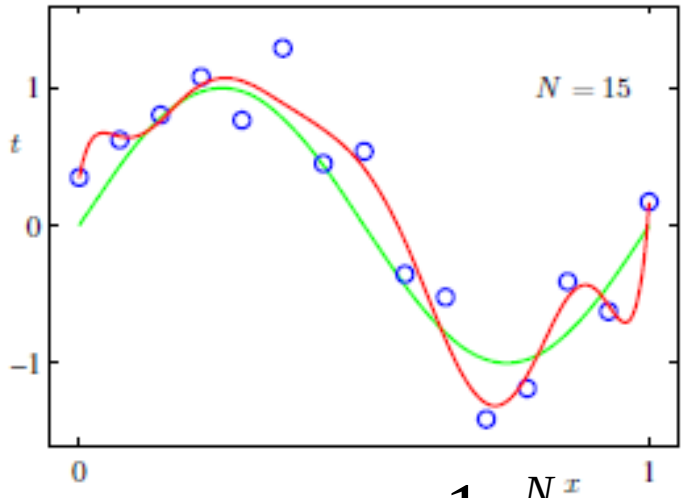


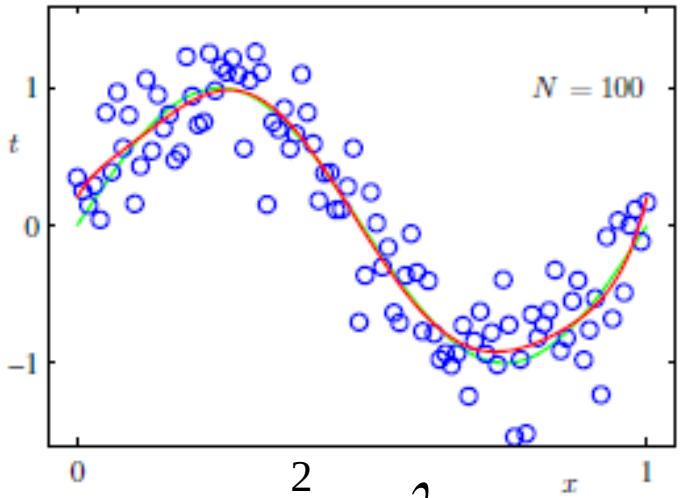




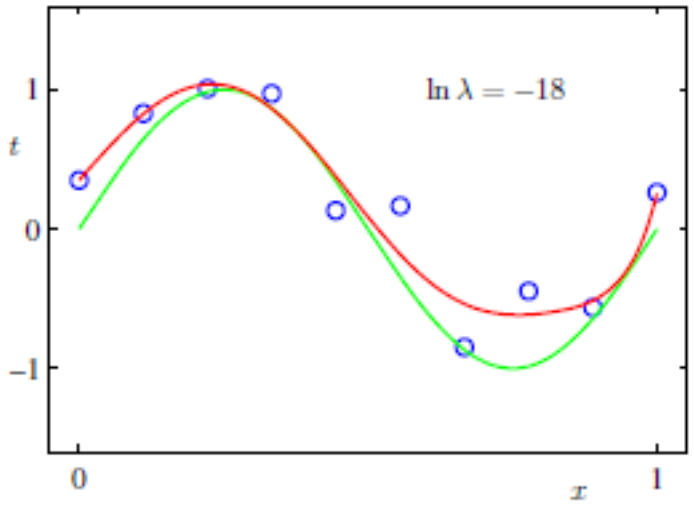


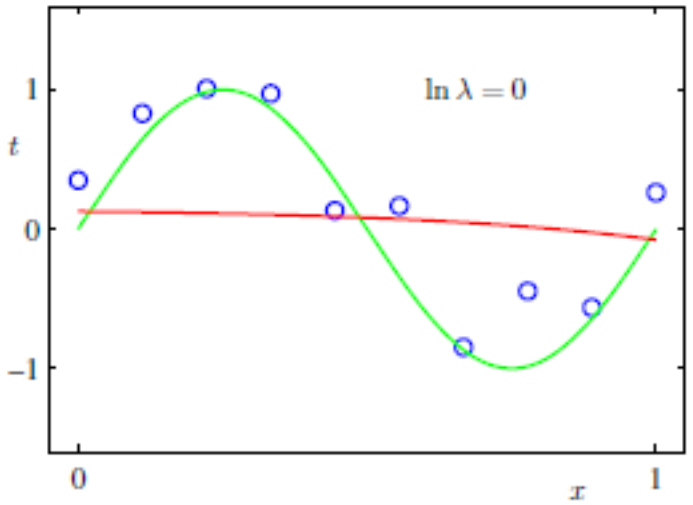
Overfitting Reduction





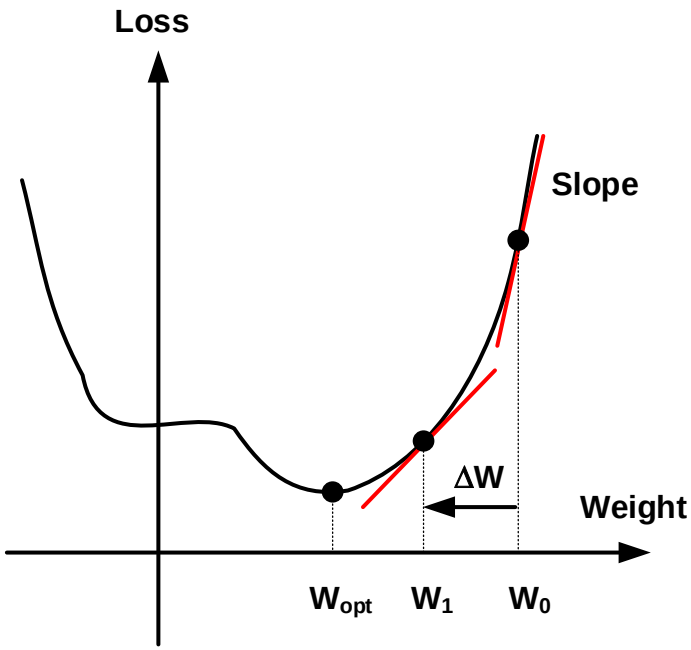
$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2 + \frac{\lambda}{2}\|w\|^2$$





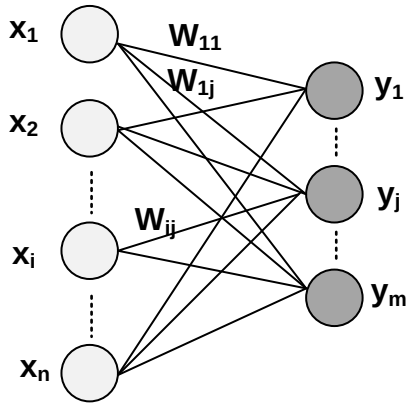
Gradient Descent

- **Numerical gradient: Slow, approximate, easy**
- **Analytic gradient: Fast, exact, error prone**
- **In practice: Derive analytic gradient, check the results with numerical gradient**



$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Learning: Update the Weights



$$\Delta w = -\eta \frac{\partial L}{\partial w}$$

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_i}$$

$$L = -\frac{1}{2}\sum_{j=1}^{M}(r_j - y_j)^2$$

$$u = \sum_{i=1}^{N} w_i x_i + b$$

$$y = f(u)$$

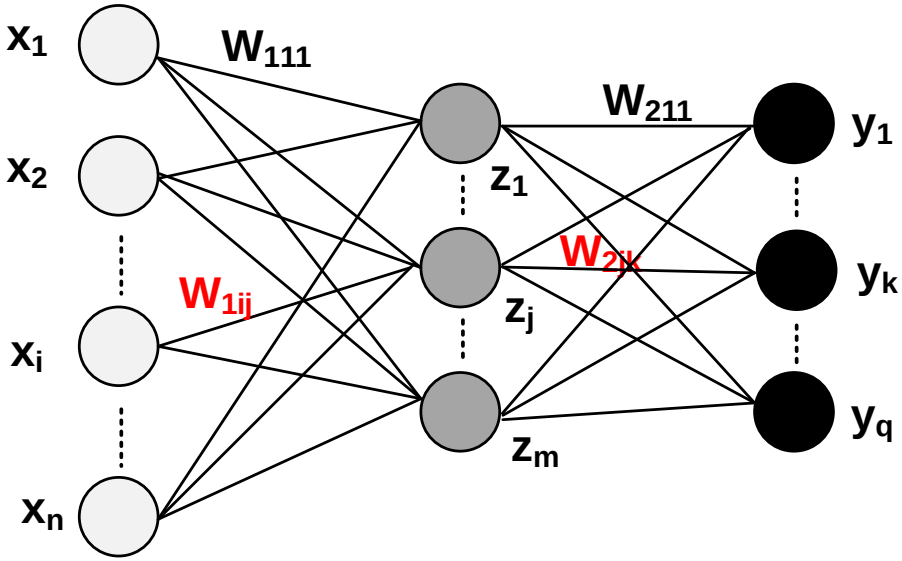
$$\frac{\partial f(u)}{\partial u} = f(u)\{1 - f(u)\}$$

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial w_{ij}} = -(r_j - y_j)\frac{\partial y_j}{\partial w_{ij}}$$

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial u_j}\frac{\partial u_j}{\partial w_{ij}} = -(r_j - y_j)y_j(1 - y_j)x_i$$

$$\Delta w = \eta(r_i - y_i)y_i(1 - y_i)x_i$$

Back-Propagation



Output → Hidden Layer

$$\frac{\partial L}{\partial w_{2jk}} = \frac{\partial L}{\partial y_k}\frac{\partial y}{\partial u_{2k}}\frac{\partial u_{2k}}{\partial w_{2jk}}$$

$$\frac{\partial L}{\partial w_{2ik}} = -(r_k - y_k)y_k(1-y_k)z_j$$

Hidden Layer → Input

$$\frac{\partial L}{\partial w_{1ij}} = \sum_{k=1}^{q}\left[\frac{\partial L}{\partial y_k}\frac{\partial y_k}{\partial u_{2k}}\frac{\partial u_{2k}}{\partial w_{1ij}}\right] \qquad \frac{\partial L}{\partial w_{1ij}} = -\sum_{k=1}^{q}\left[(r_k - y_k)y_k(1-y_k)\frac{\partial u_{2k}}{\partial w_{1ij}}\right]$$

$$\frac{\partial u_{2k}}{\partial w_{1ij}} = \frac{\partial u_{2k}}{\partial z_j}\frac{\partial z_j}{\partial w_{1ij}} \qquad \frac{\partial z_j}{\partial w_{1ij}} = \frac{\partial z_j}{\partial u_{1j}}\frac{\partial u_{1j}}{\partial w_{1ij}} = z_j(1-z_j)x_i$$

$$\Delta w_{1ij} = \eta\sum_{k=1}^{q}[(r_k - y_k)y_k(1-y_k)w_{2jk}]z_j(1-z_j)x_i$$

Back-Propagation

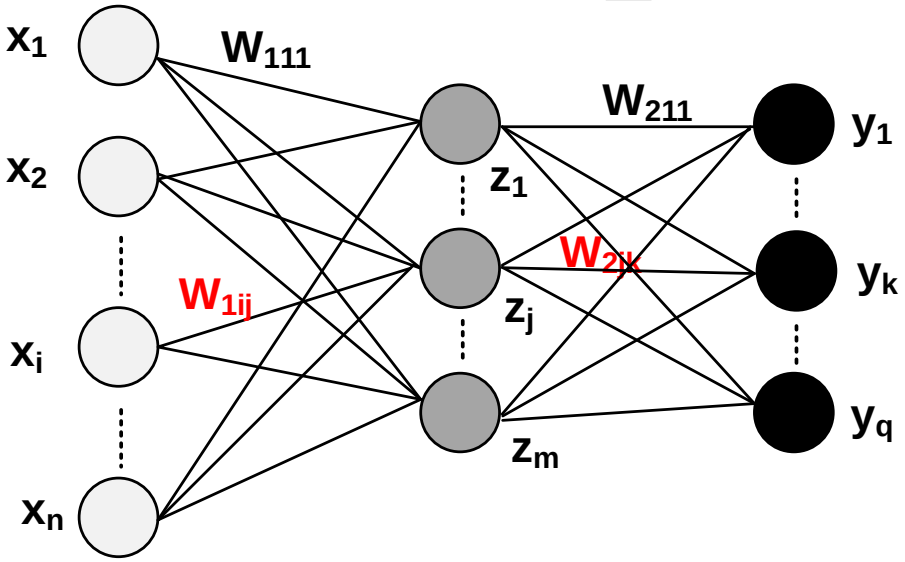
$$\frac{\partial E}{\partial w_{2jk}} = \frac{\partial E}{\partial y_k}\frac{\partial y}{\partial u_{2k}}\frac{\partial u_{2k}}{\partial w_{2jk}}$$

$$\frac{\partial E}{\partial w_{2i1}} = -(r_k - y_k)y_k(1-y_k)z_j$$

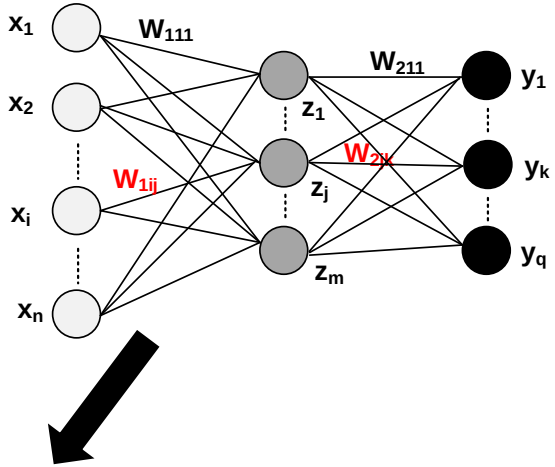
Differential of Error Function

Differential of Activation Function

Differential of Weight



Back-Propagation



$$\frac{\partial E}{\partial w_{1ij}} = \sum_{k=1}^{q} \left[\frac{\partial E}{\partial y_k} \frac{\partial y}{\partial u_{2k}} \frac{\partial u_{2k}}{\partial w_{1ij}} \right]$$

$$= -\sum_{k=1}^{q} \left[(r_k - y_k) y_k (1 - y_k) \frac{\partial u_{2k}}{\partial w_{1ij}} \right]$$

$$\frac{\partial u_{2k}}{\partial w_{1ij}} = \frac{\partial u_{2k}}{\partial z_j} \frac{\partial z_j}{\partial w_{1ij}}$$

$$\frac{\partial z_j}{\partial w_{1ij}} = \frac{\partial z_j}{\partial u_{1j}} \frac{\partial u_{1j}}{\partial w_{1ij}}$$

$$\frac{\partial E}{\partial w_{1ij}} = -\sum_{k=1}^{q} \left[(r_k - y_k) y_k (1 - y_k) w_{2jk} z_j (1 - z_j) x_i \right]$$

Differential of Error Function

Differential of Sigmoid Function

Weight

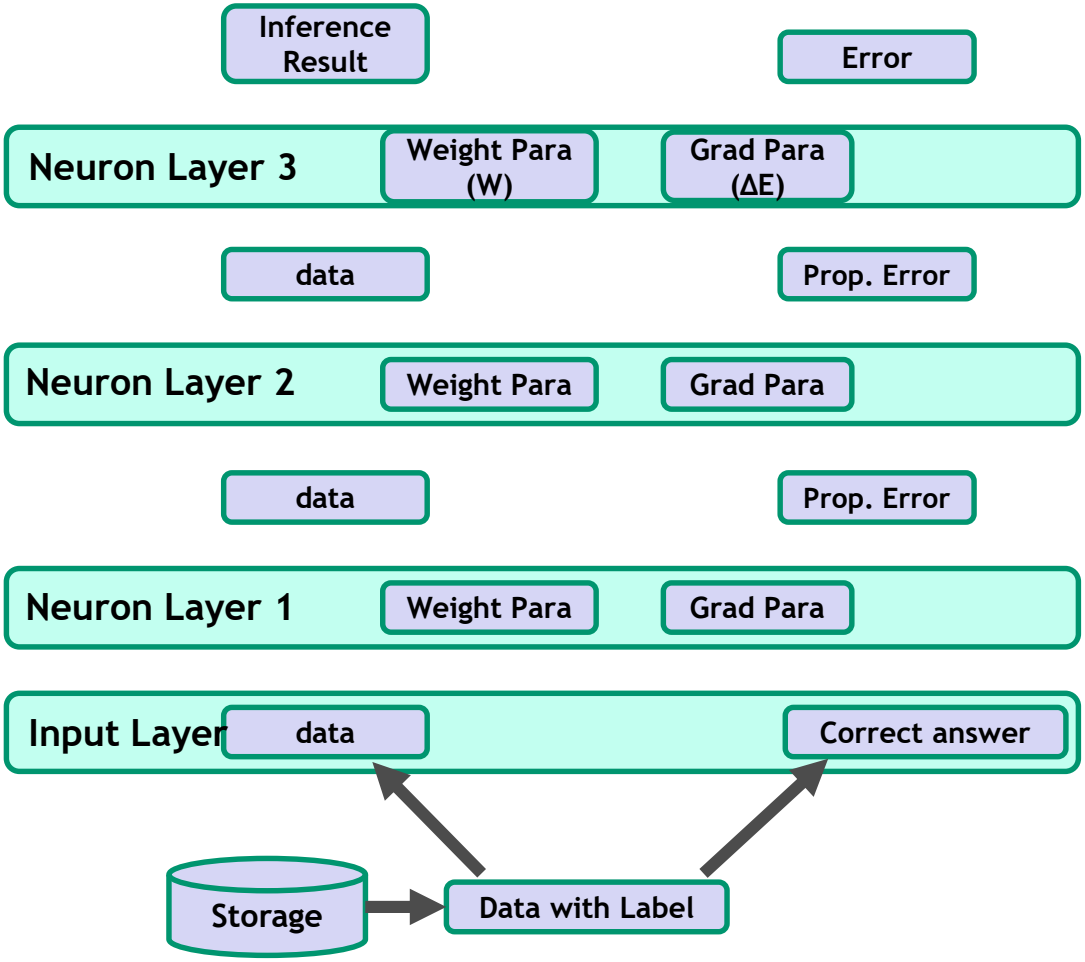
Differential of Sigmoid Function

Input

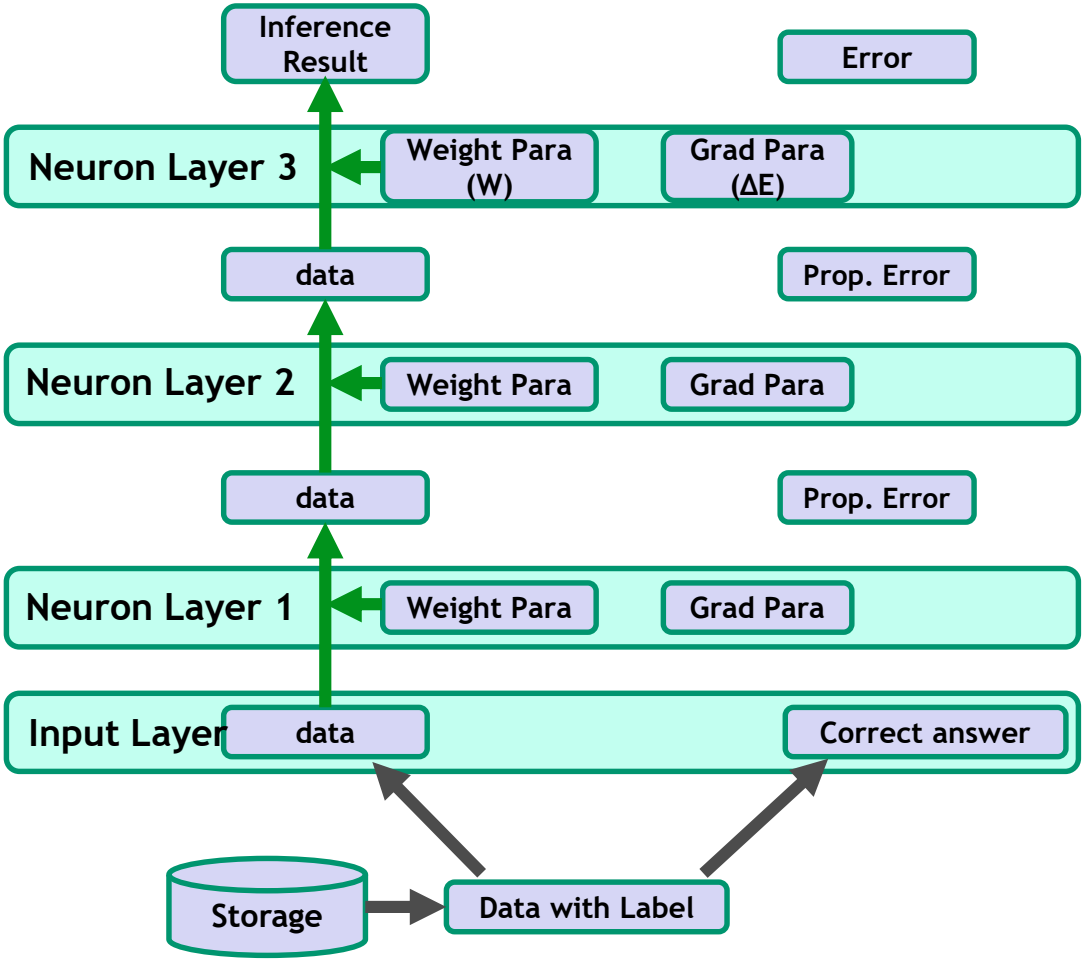
Hidden Layer - Output Layer

Input Layer - Hidden Layer

Learning Cycle

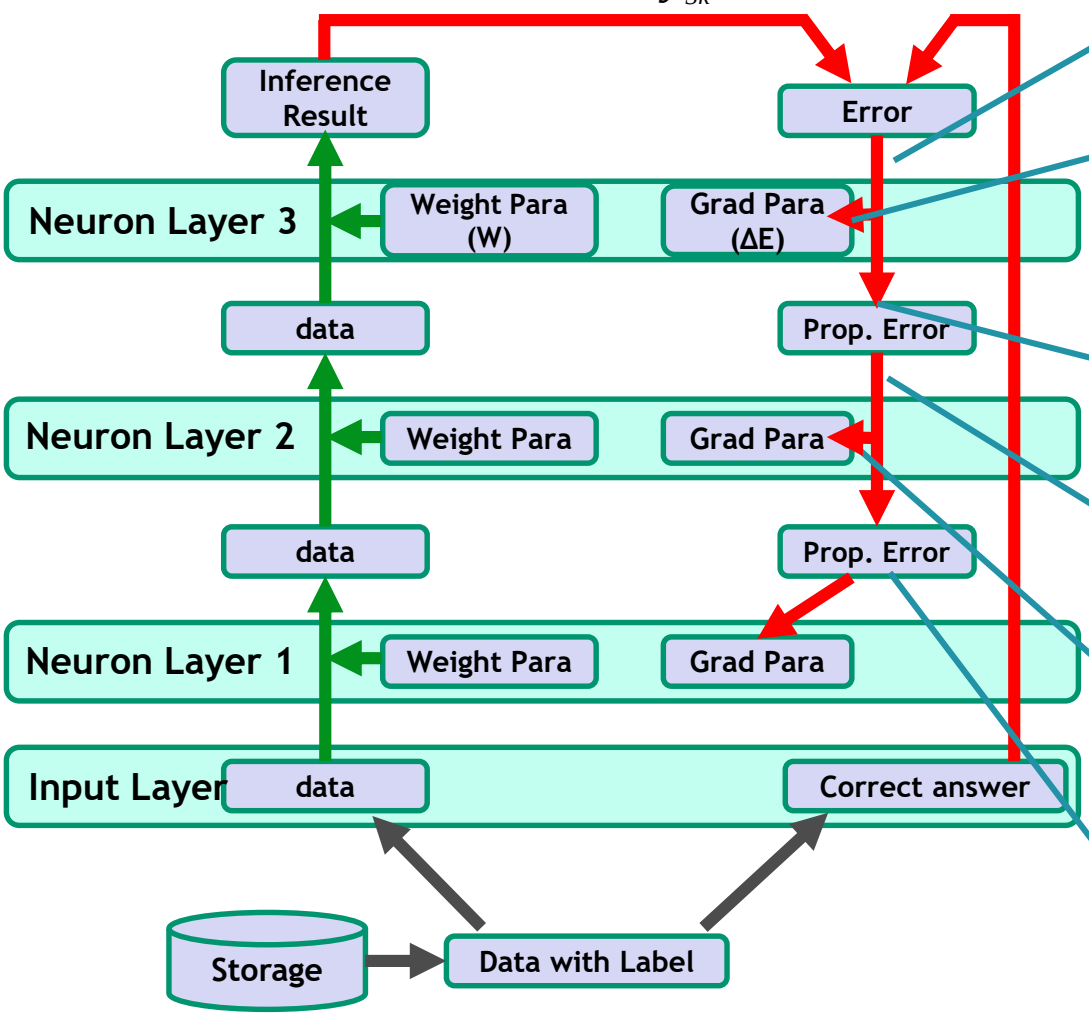


Learning Cycle: Forward Inferencing



Forward

Learning Cycle: Forward Inferencing



$$\frac{\partial E}{\partial y_{3k}} = Correct - Inference = \Delta y$$

$$\delta_{3k} = \Delta y \cdot \frac{\partial y_{3k}}{\partial a_{3k}}$$: Local Gradient

$$\delta_{3k} \cdot \frac{\partial a_{3k}}{\partial w_{3ij}}$$: LG x Input

Forward

BackPropag

$$\sum_{k=1}^{q} \delta_{3k} \cdot \frac{\partial a_{3k}}{\partial w_{2ij}} w_{3jk}$$

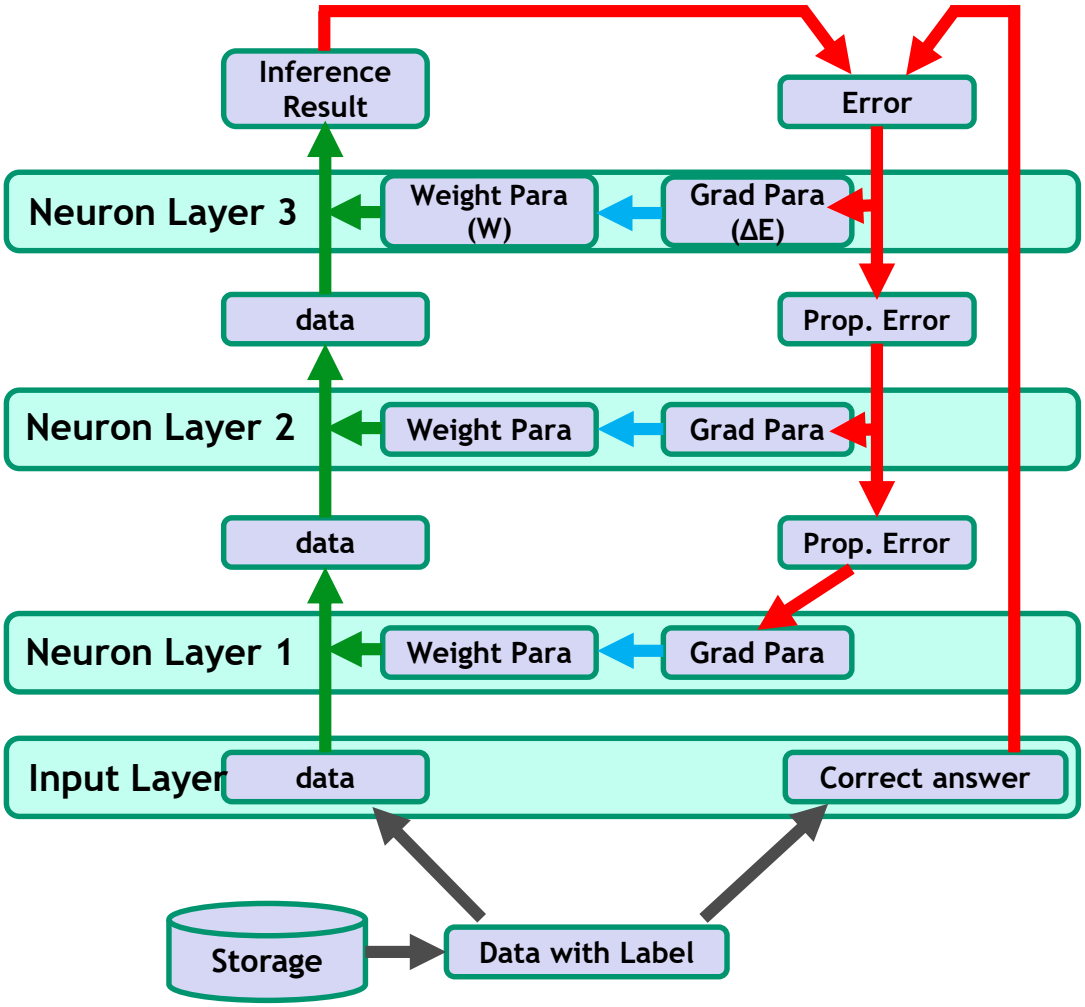
$$\delta_{2j} = \frac{\partial y_{3j}}{\partial a_{2j}} \sum_{k=1}^{q} \delta_{3k} \cdot w_{3kj}$$

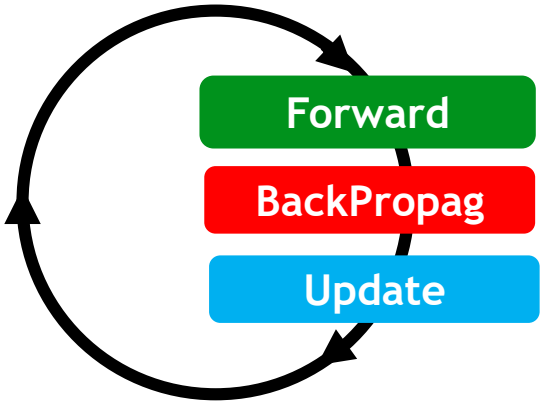
$$\sum_{k=1}^{q} \delta_{3k} \cdot w_{3jk} \cdot \frac{\partial y_2}{\partial w_1} = \delta_{2j} y_{1j}$$

$$y_{2j}(1-y_{2j}) y_{1i}$$

$$\sum_{k=1}^{q} \delta_{2k} \cdot \frac{\partial a_{2k}}{\partial w_{1ij}} w_{2jk}$$

Learning Cycle: Forward Inferencing





Outline

-Neural Network for Intelligent SoC Robot-

1. *Introduction of AI for Robot*
2. ***Pattern Recognition (PR)***
   : Activation Function

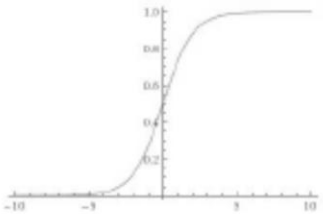
3. *Convolution Neural Network (CNN)*
4. *Unsupervised Learning*
5. *Reinforcement Learning*
6. *Additional Learning Algorithm*

Activation Functions

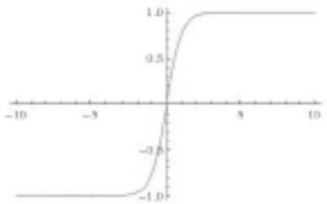
- Non-Linearity

Sigmoid

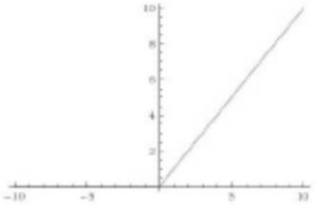
$\sigma(x) = 1/(1+e^{-x})$



tanh tanh(x)

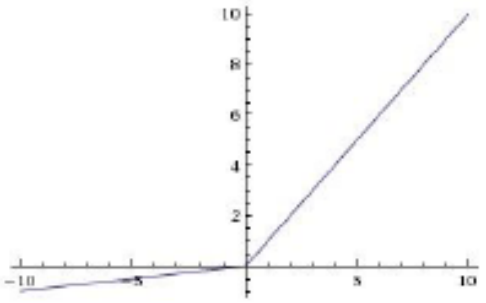


ReLU max(0,x)



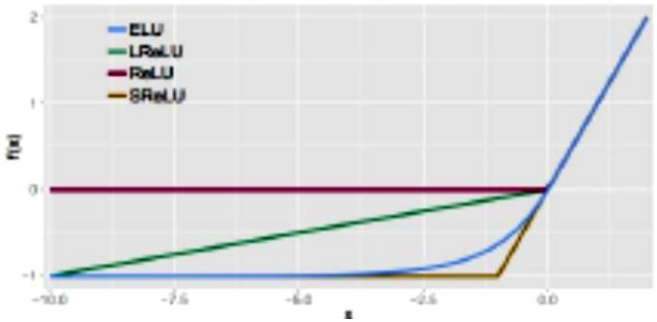
Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

Leaky ReLU
max(0.1x, x)



ELU

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

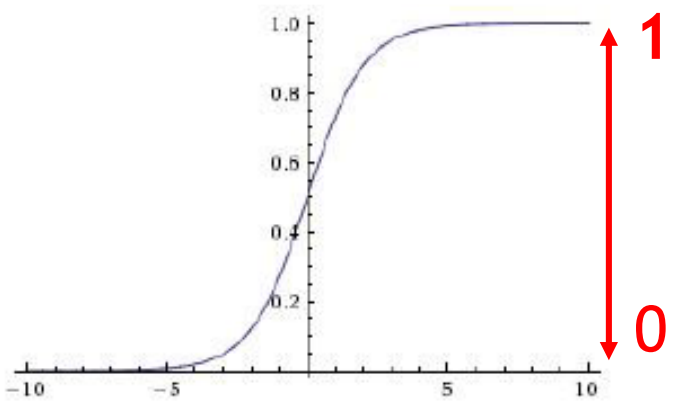


Sigmoid

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

1. Differentiable
2. Output is in [0, 1]
3. Biologically all-or-nothing

$$\sigma'(x) = \sigma(x) \bullet (1-\sigma(x)) = \frac{1}{1+e^{-x}} \bullet \left(1 - \frac{1}{1+e^{-x}}\right) = \frac{e^{-x}}{\left(1+e^{-x}\right)^2}$$

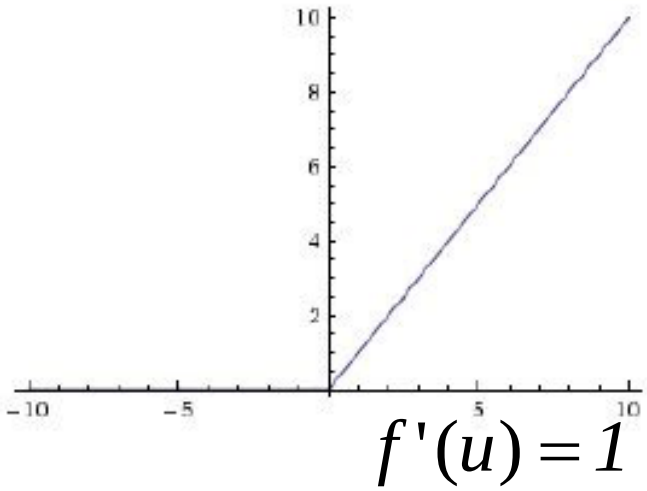


Sigmoid

1. Gradient = 0 at saturated regions
2. Output is not Zero centered
3. Exponential needs complicated calculation

ReLU

f(x) = max(0, x)



$$f'(u) = 1$$

ReLU
(Rectified Linear Unit)

1. No Saturation
2. Efficient Computation
3. Fast Convergence

1. Not zero-centered output
2. Differentiable(?)

,

Outline

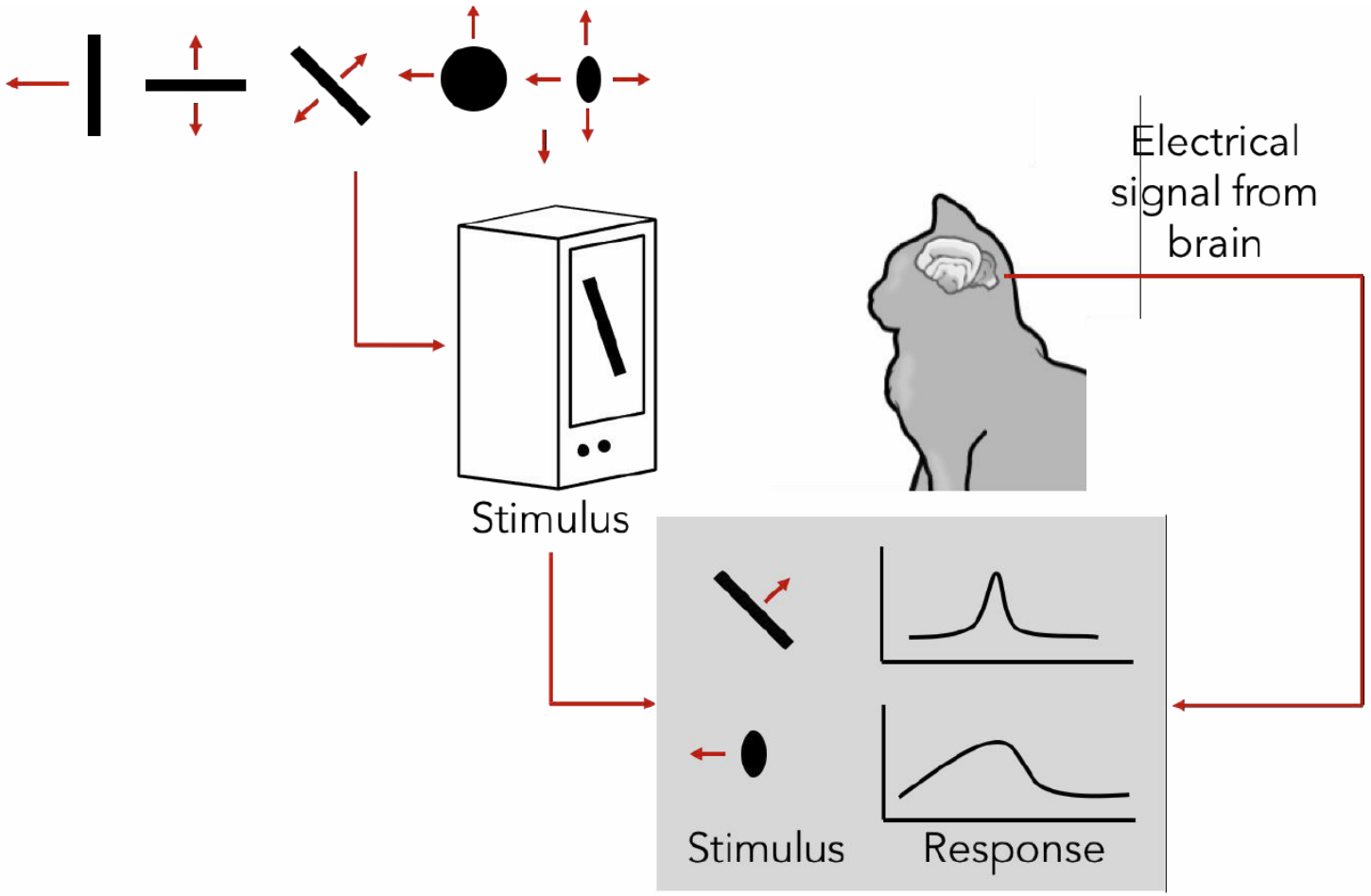
-Neural Network for Intelligent SoC Robot-

1. *Introduction of AI for Robot*
2. *Pattern Recognition (PR)*
3. ***Convolution Neural Network (CNN)***
   : What is CNN?

4. *Unsupervised Learning*
5. *Reinforcement Learning*
6. *Additional Learning Algorithm*

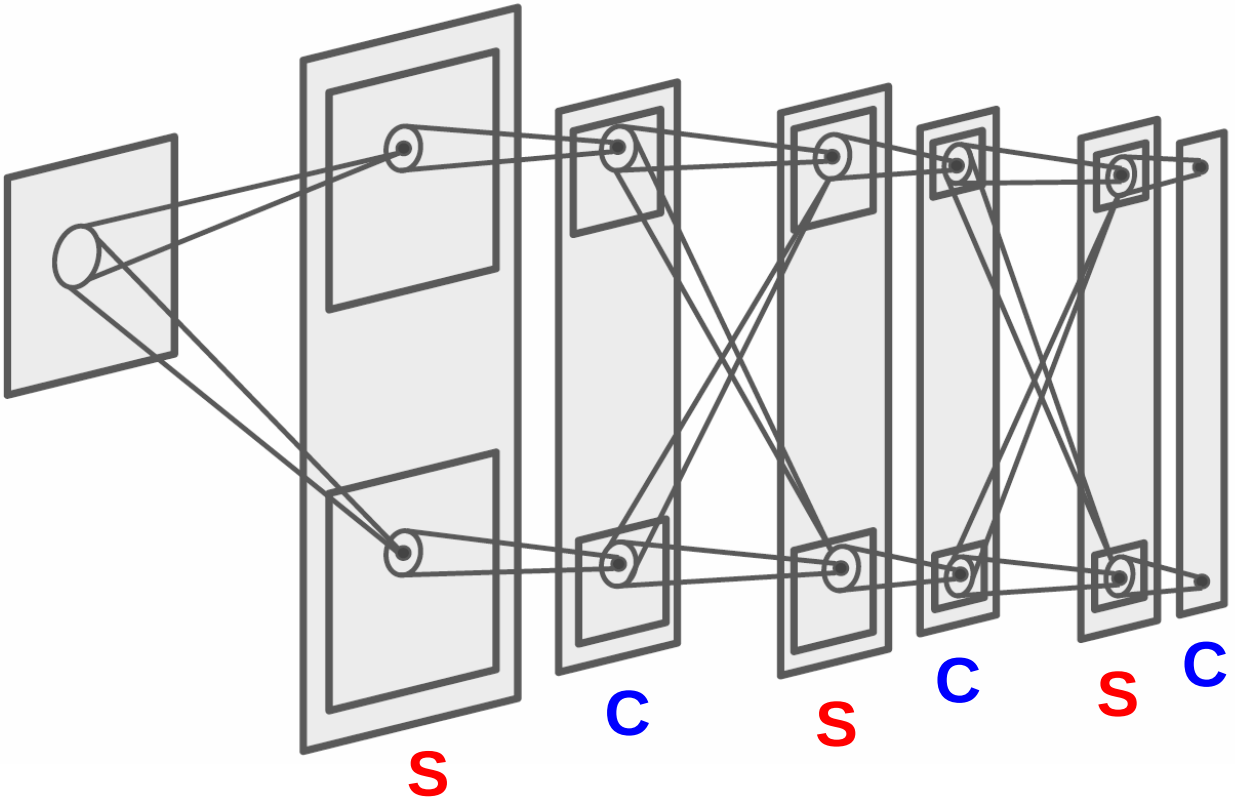
Hubel & Wiesel (1950s~1960s)

- **Experiments on a cat's visual cortex**



Neocognitron - Fukushima (1980)

- Prototype of modern CNN architecture
- Composed of simple & complex cells



Simple Cell
Convolution

Complex Cell
Min Pooling

Applications of CNN

Image Classification

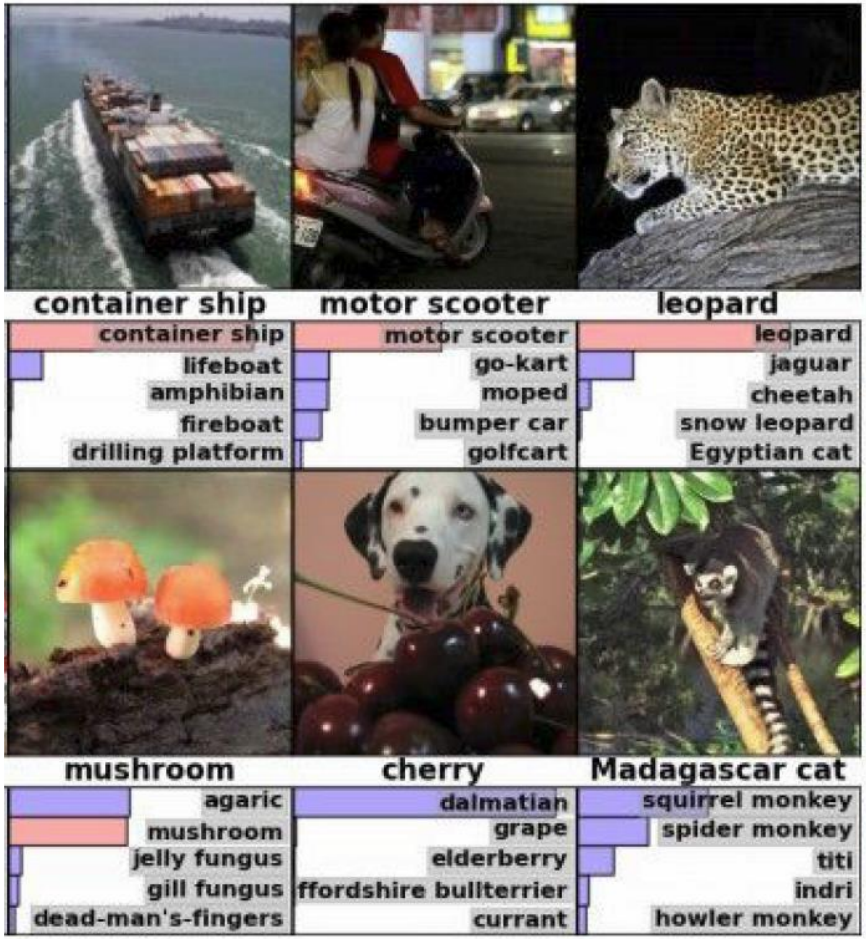
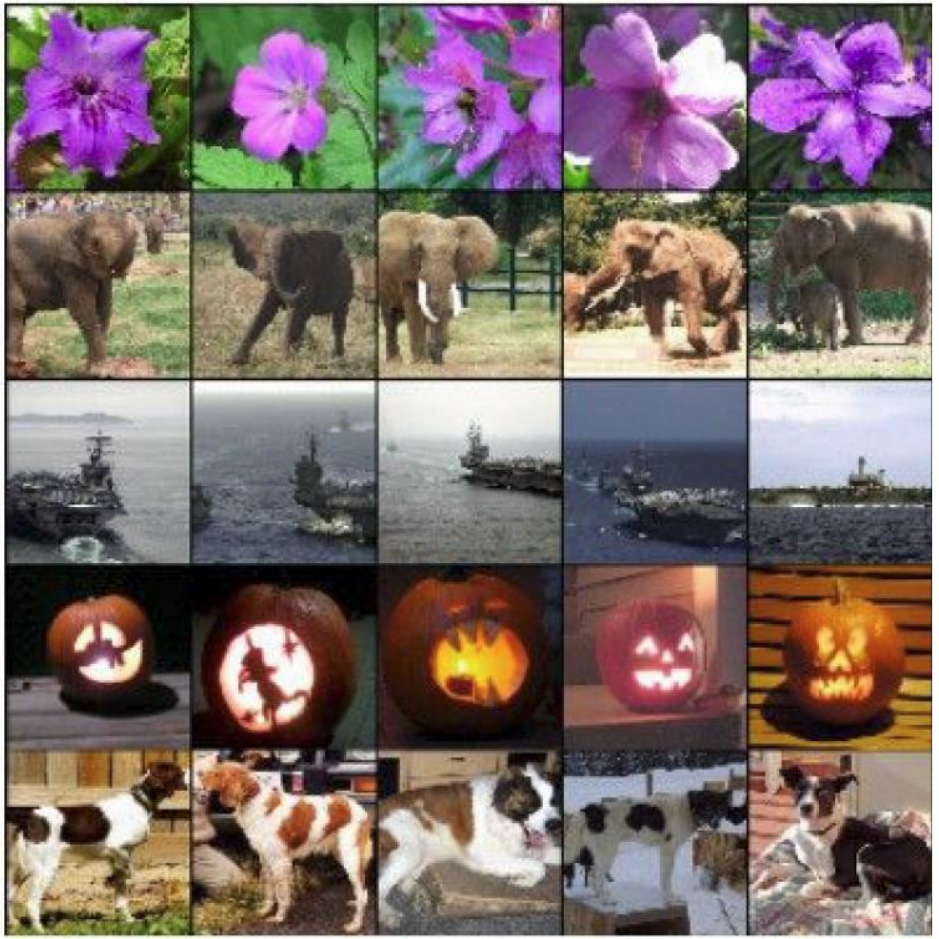


Image Retrieval



Outline

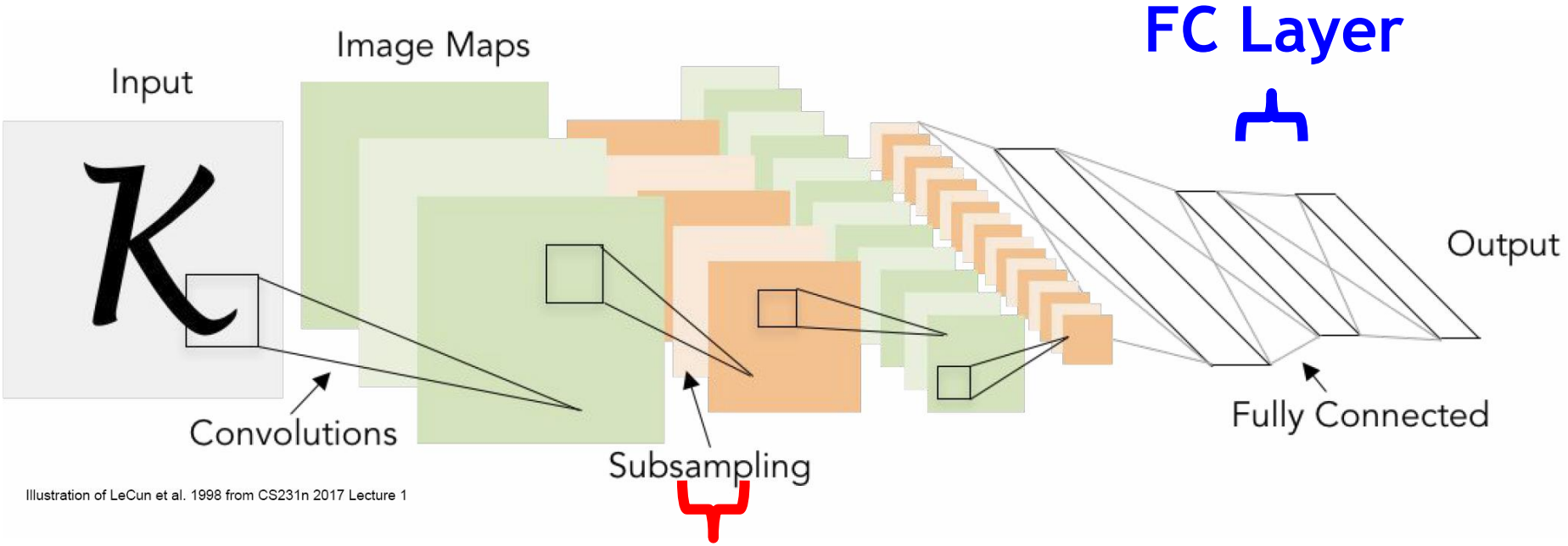
-Neural Network for Intelligent SoC Robot-

1. *Introduction of AI for Robot*
2. *Pattern Recognition (PR)*
3. ***Convolution Neural Network (CNN)***
   : Basic Architecture of CNN

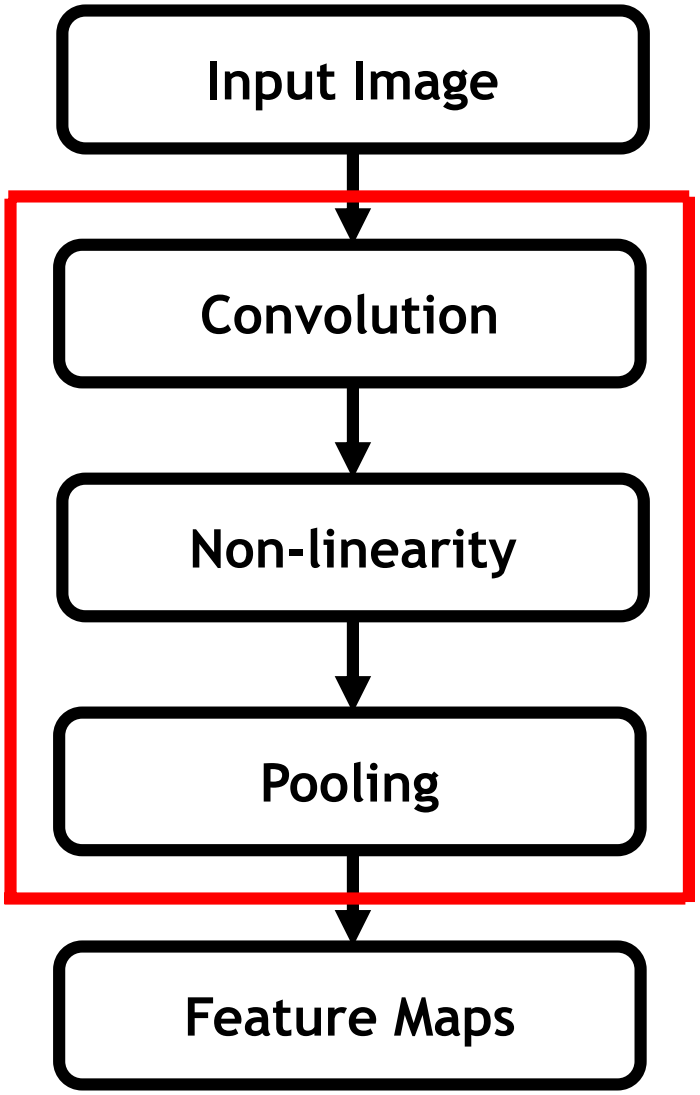
4. *Unsupervised Learning*
5. *Reinforcement Learning*
6. *Additional Learning Algorithm*

Two Basic Types of Layer

- Convolution layer
  → Extract local feature of input images
- Fully connected layer → Classification



Structure of Convolution Layer



- Convolution
- Non-linearararity
  - Sigmoid
  - Tanh
  - ReLU
- Pooling
  - Max
  - Mean

Convolution and Cross-Correlation

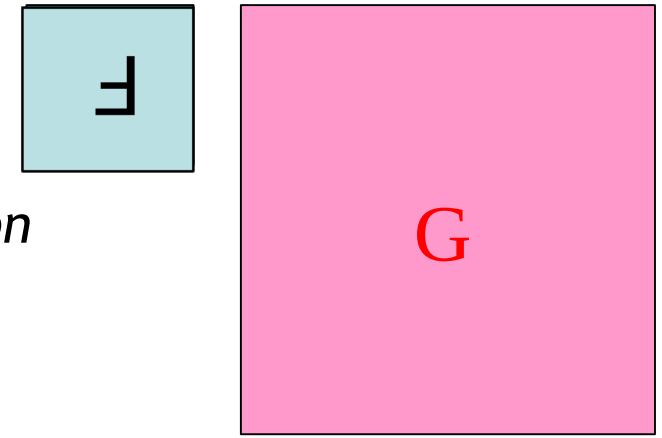
- **Convolution:**
  - **Flip the filter (bottom to top, right to left)**
  - **Then apply cross-correlation**

$$H(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} G(u,v) \cdot F(i-u, j-v)$$

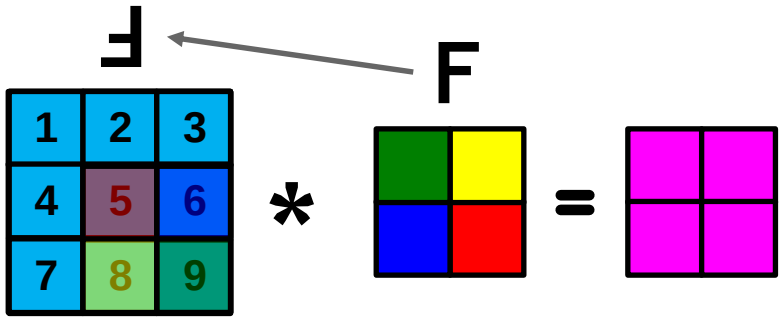
$$H = G * F = G \times \text{F}$$

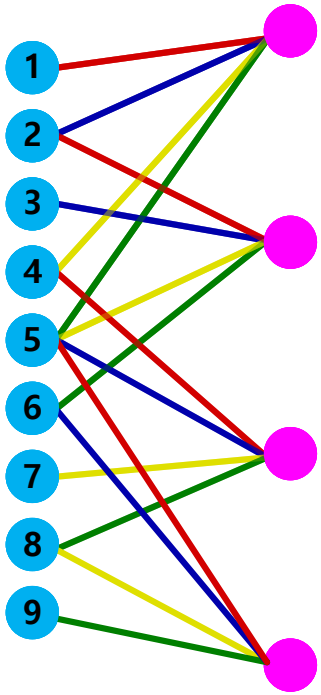
convolution operator

Cross-correlation operator

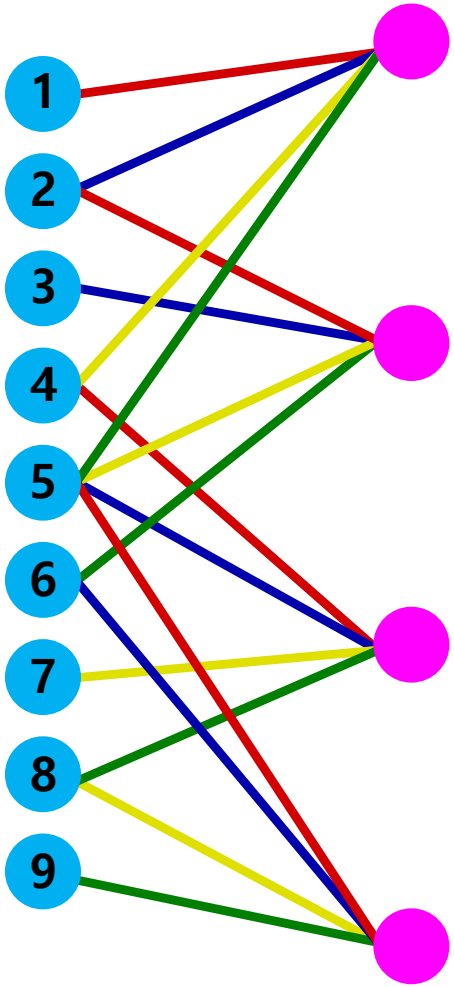


Neuron Connections in CNN



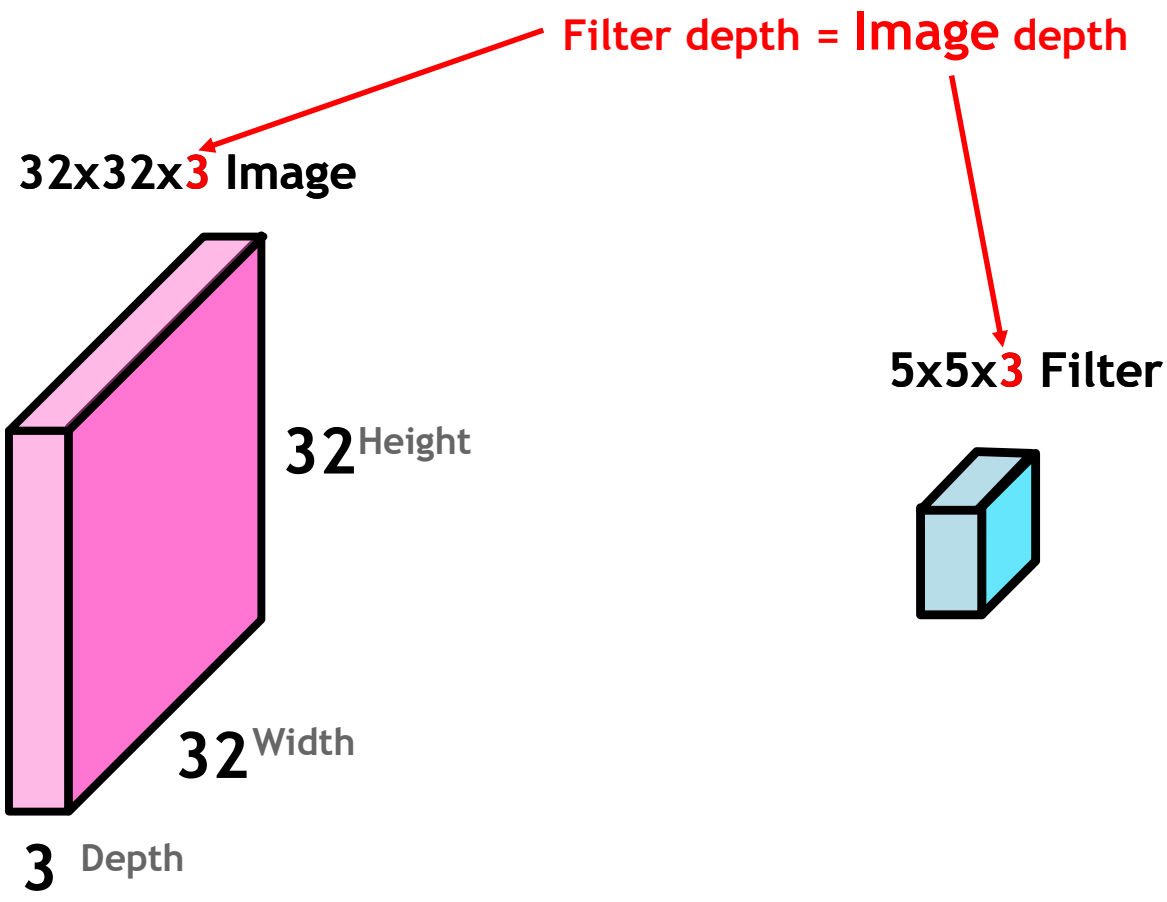


2D Convolution

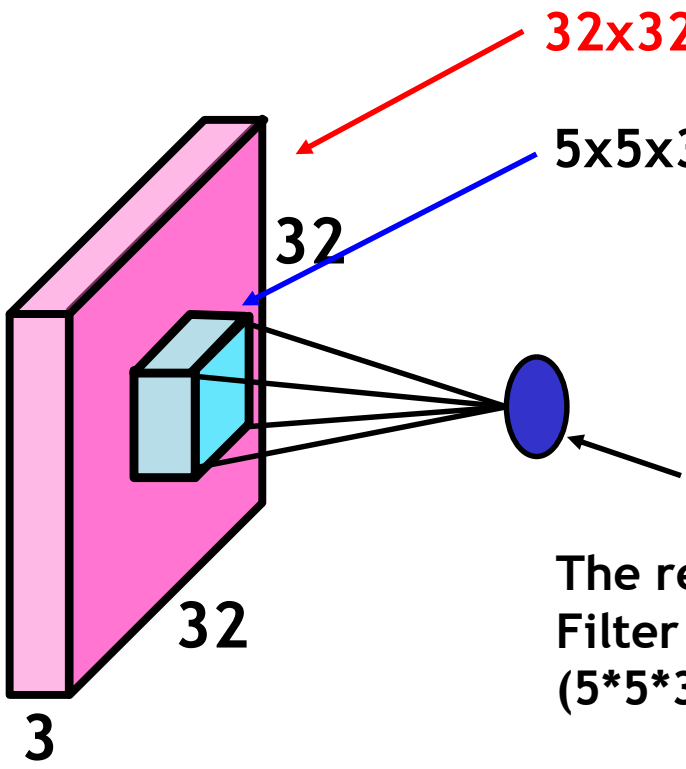


Convolution Layer

Convolution Layer



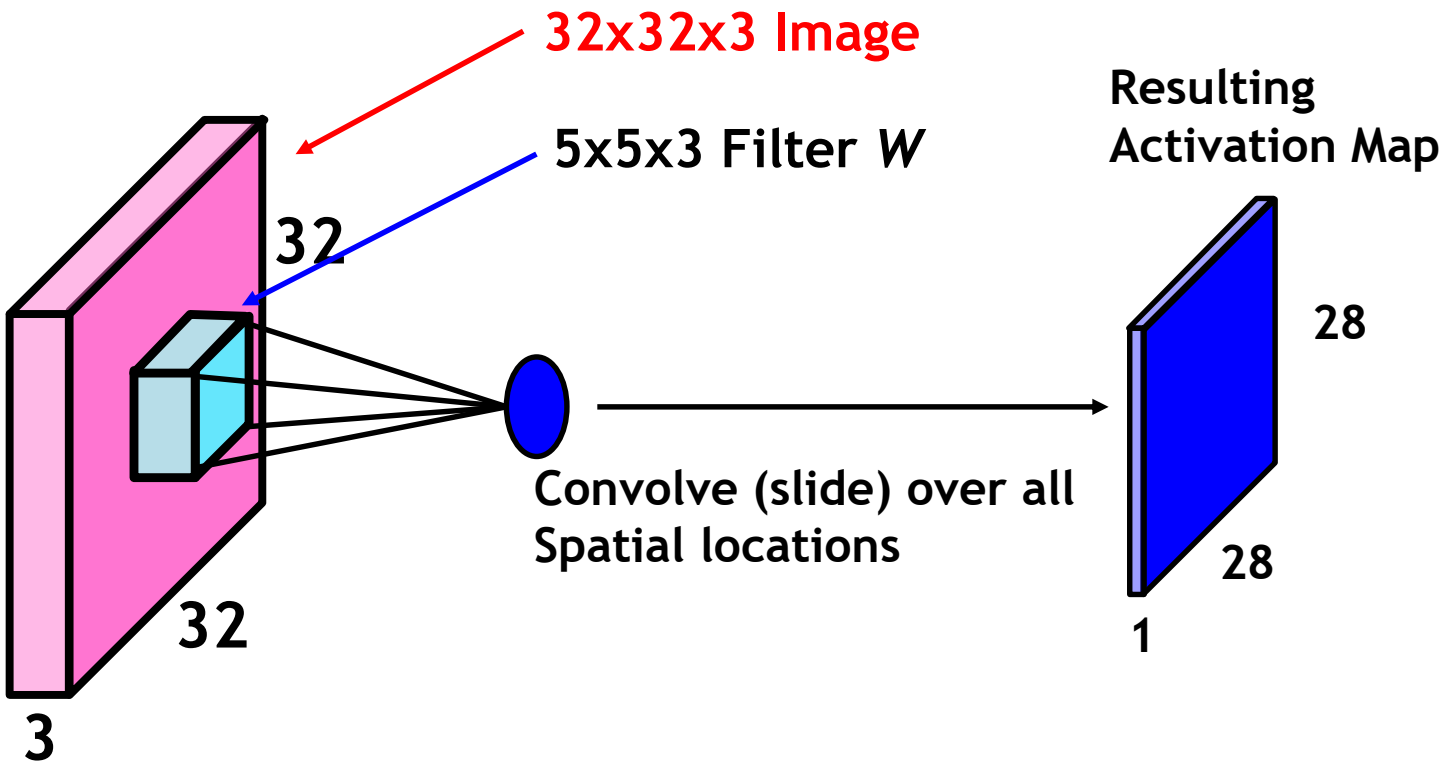
Convolution Operation



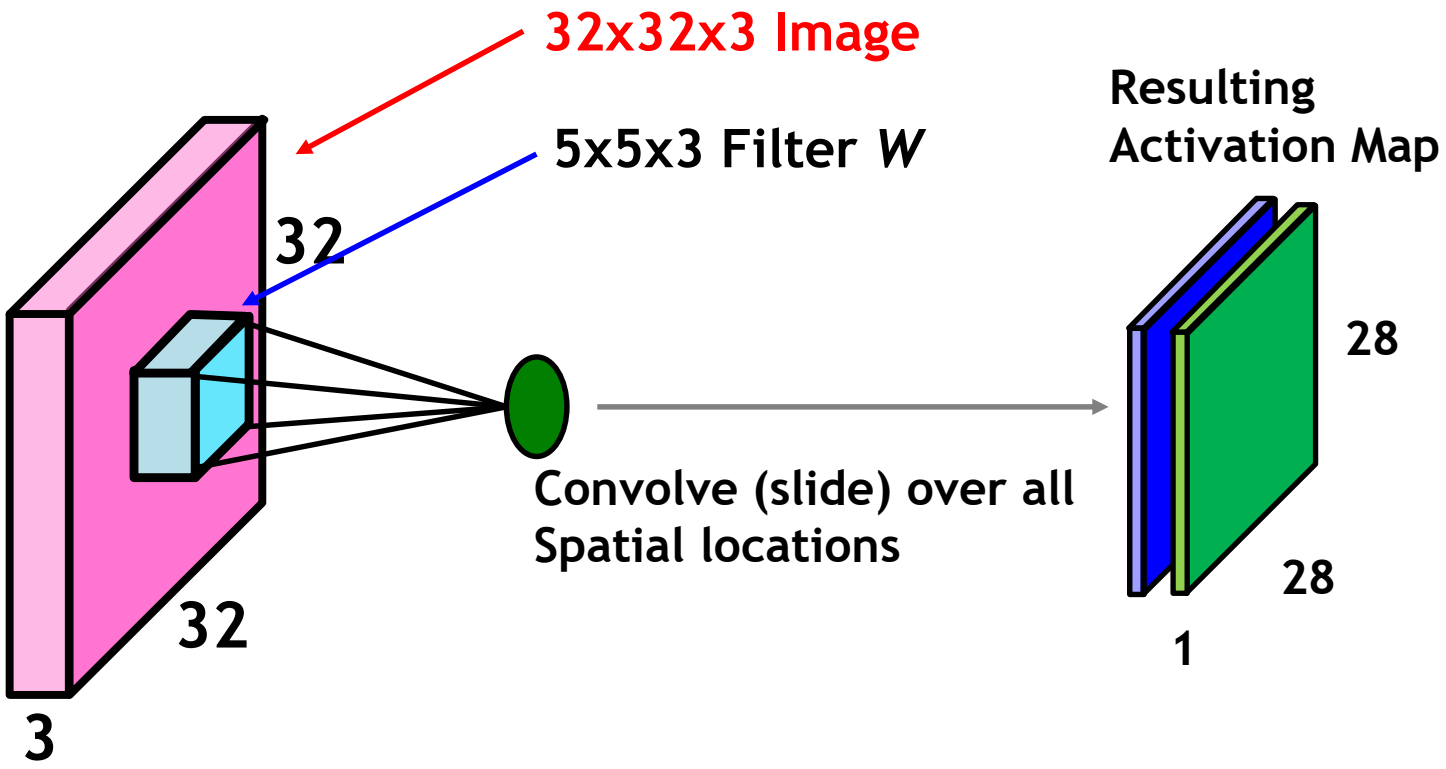
The result of taking a dot product between the Filter and a small 5x5x3 block of the Image (5*5*3=75-dimensional dot product + bias)

$$f = \sum w_i x_i + b$$

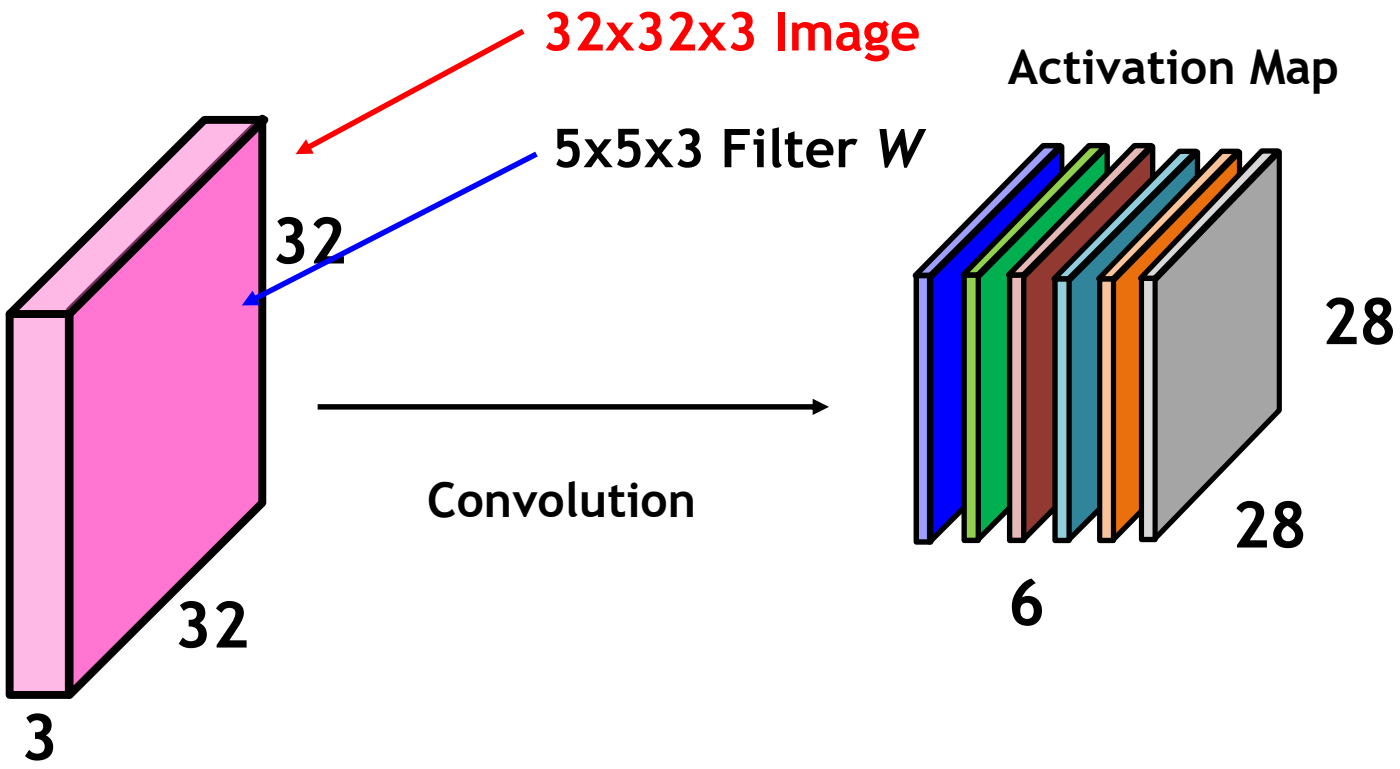
Convolution Operation



Convolution Operation

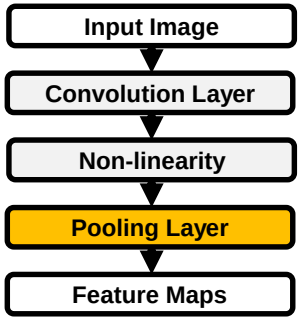


Convolution Operation

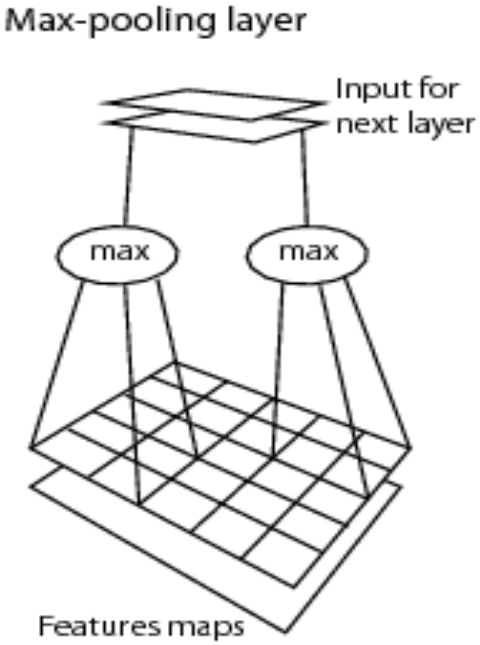


If there are 6 of 5x5 filters,
we have 6 separate activation maps

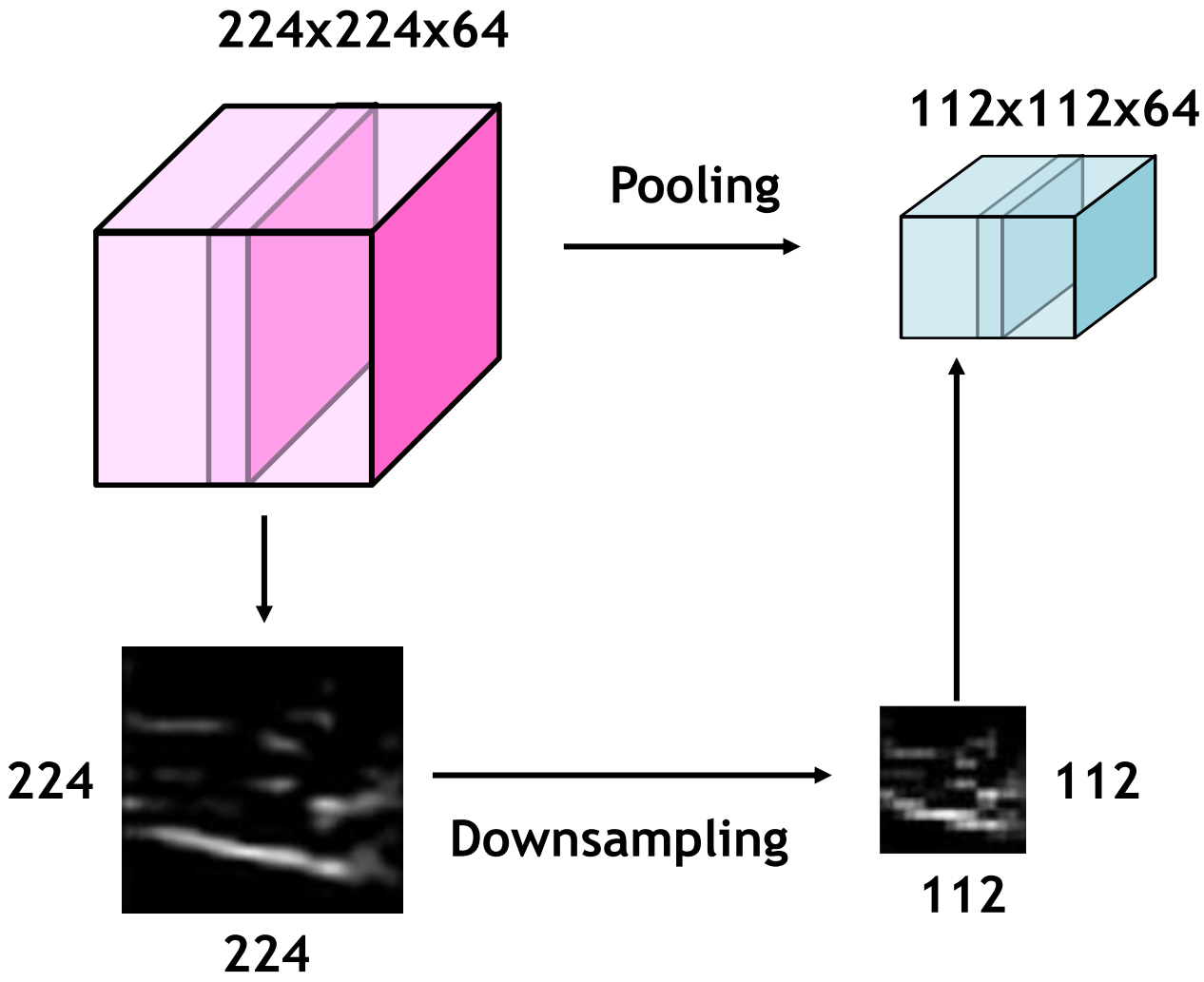
Pooling Operation



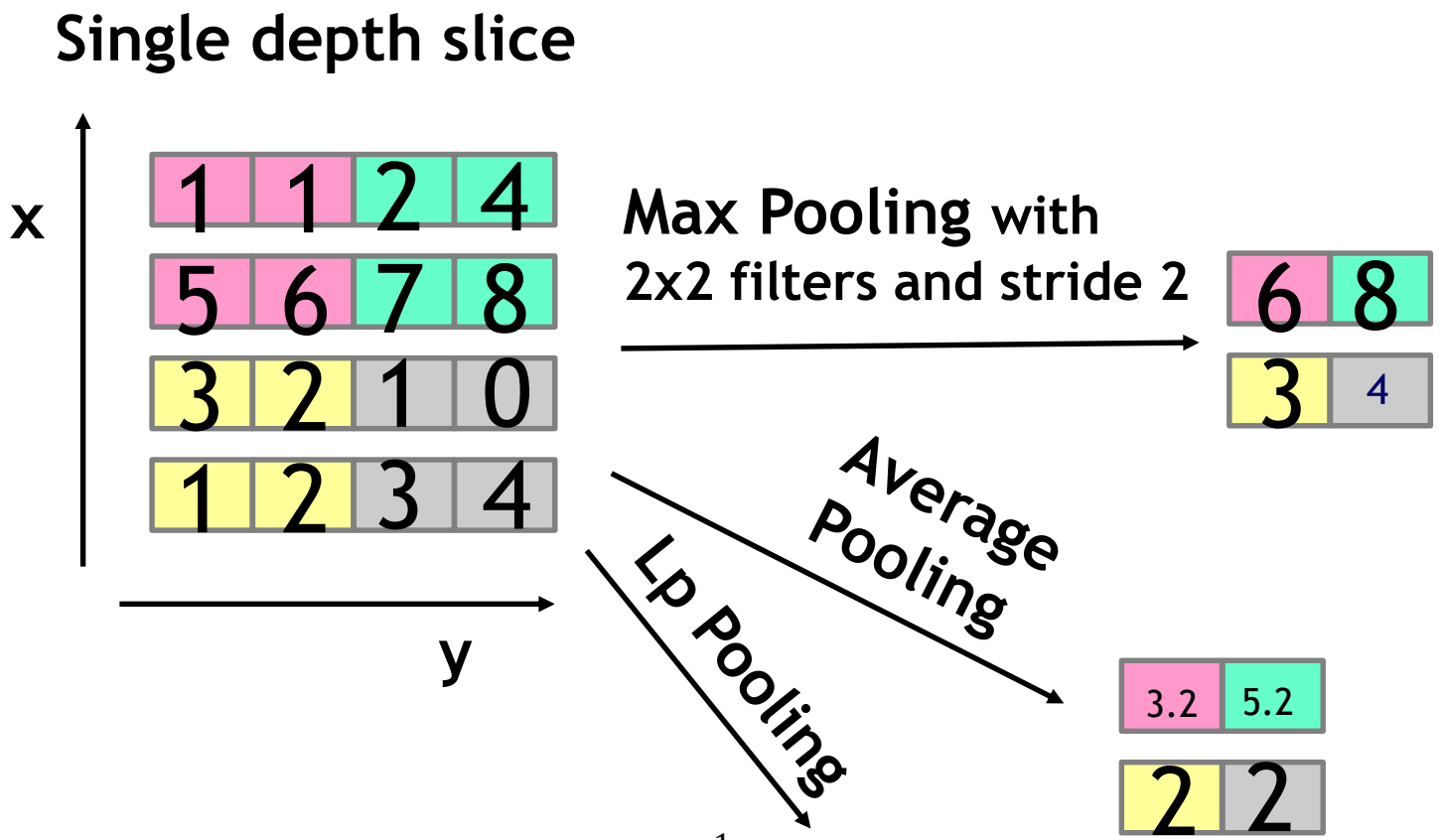
- Select maximum response of inputs
- Reduce computation complexity



Pooling = Down Sampling



Max Pooling

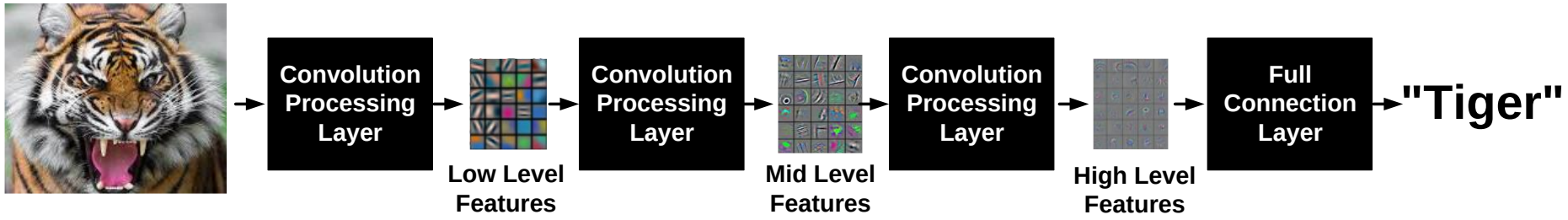


$$f(x_i) = \left(\sum_{j=1}^{n} \sum_{i=1}^{m} I(i,j)^P \cdot G(i,j) \right)^{\frac{1}{P}}$$

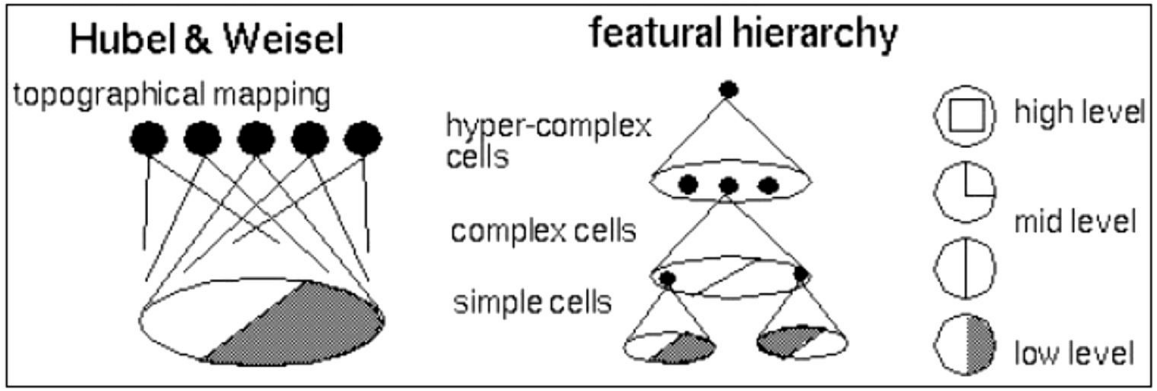
Emphasize the peak

Combination of Convolution Layers

- Extract more complex features through multiple conv. layers

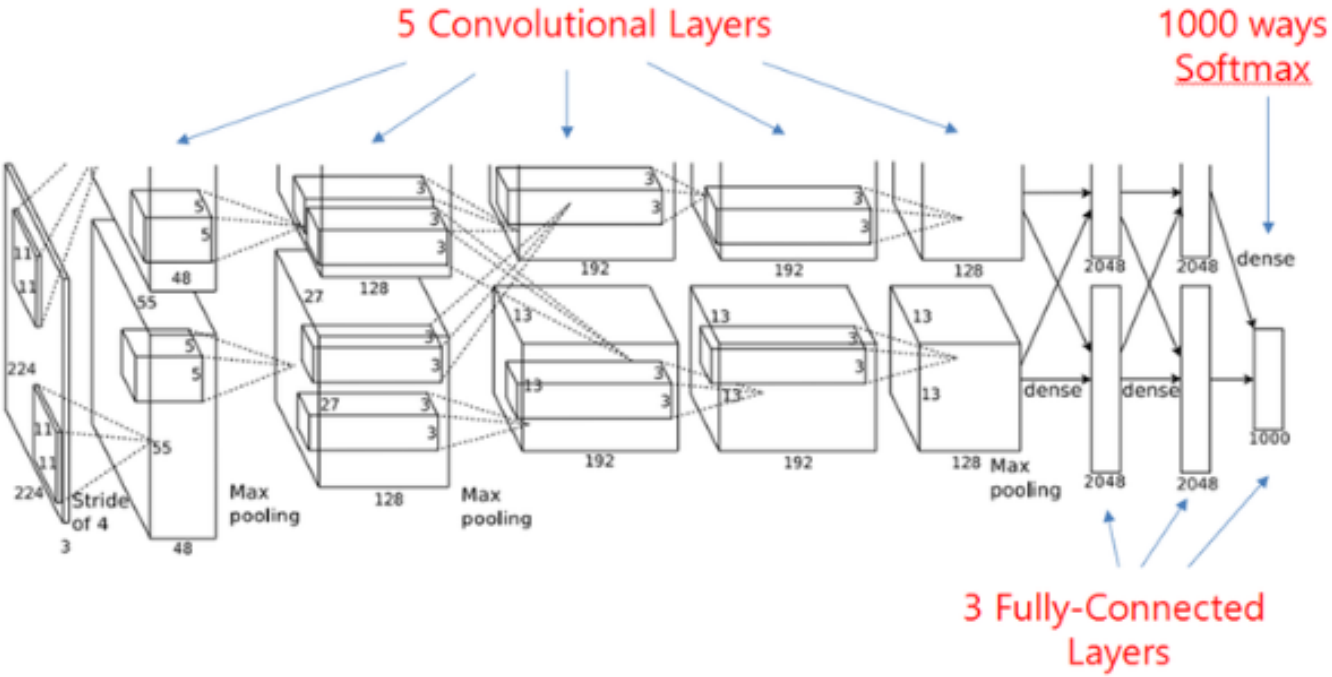


Simple Feature ⟶ Complex Feature



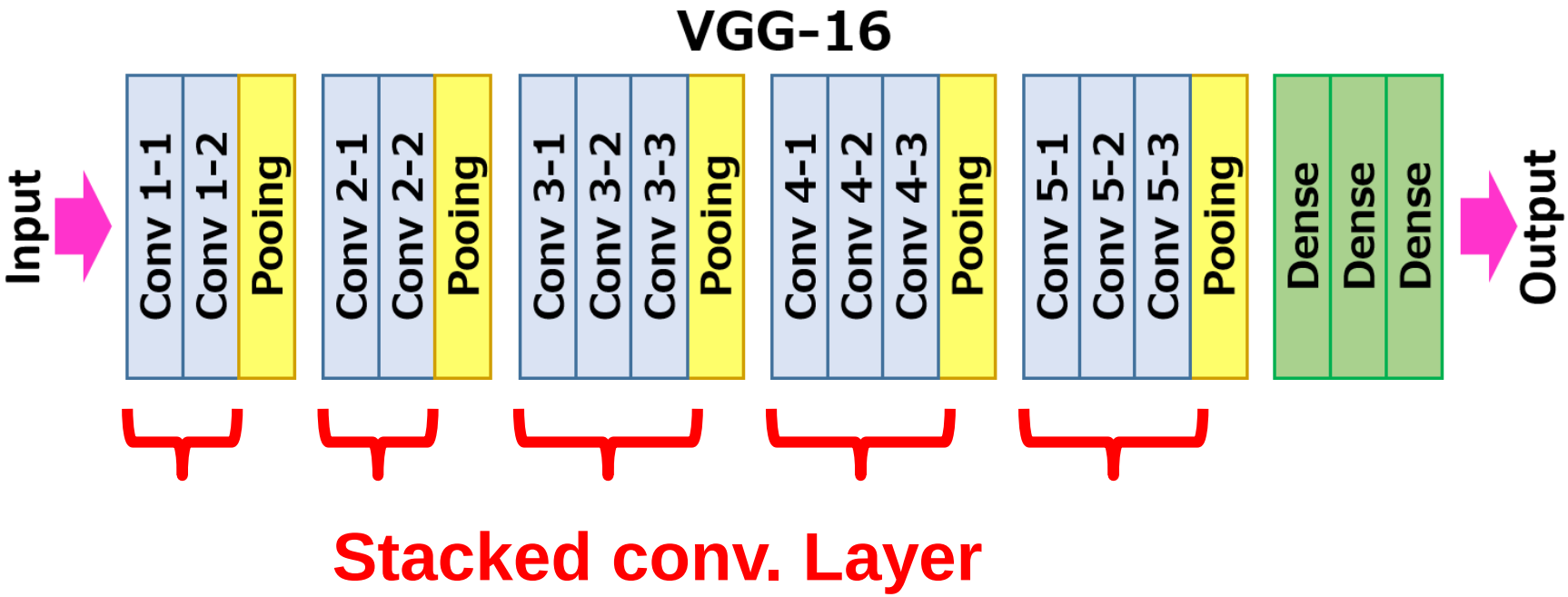
AlexNet (2012)

- **ILSVRC-2012(Image classification) winner**
- **5 Conv. Layers with Pooling layers, 3 FC layers**
- **Utilize ReLU(Rectified Linear Unit) & DropOut**



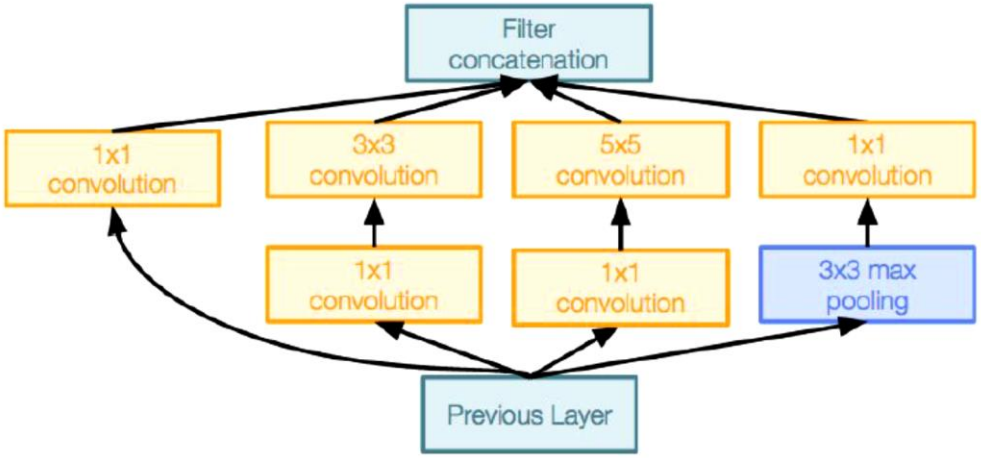
VGGNet (2014)

- **Deeper layers than AlexNet with smaller filters**
- **7x7 filter can substitute by 3 stacked 3x3 filter**

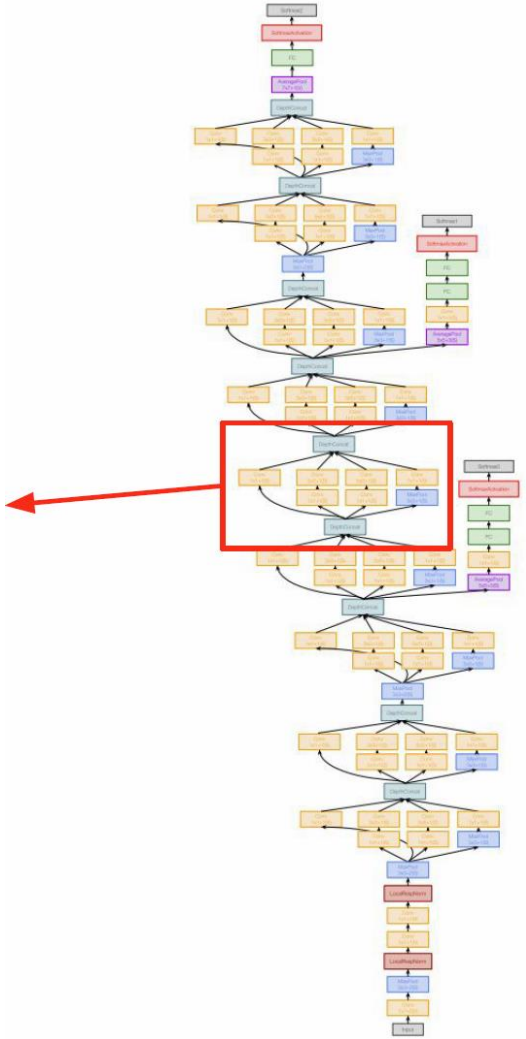


GoogLeNet (2014)

- **22 layers (1 FC layer)**
- **Multiple receptive field**
- **ILSVRC 2014 winner**

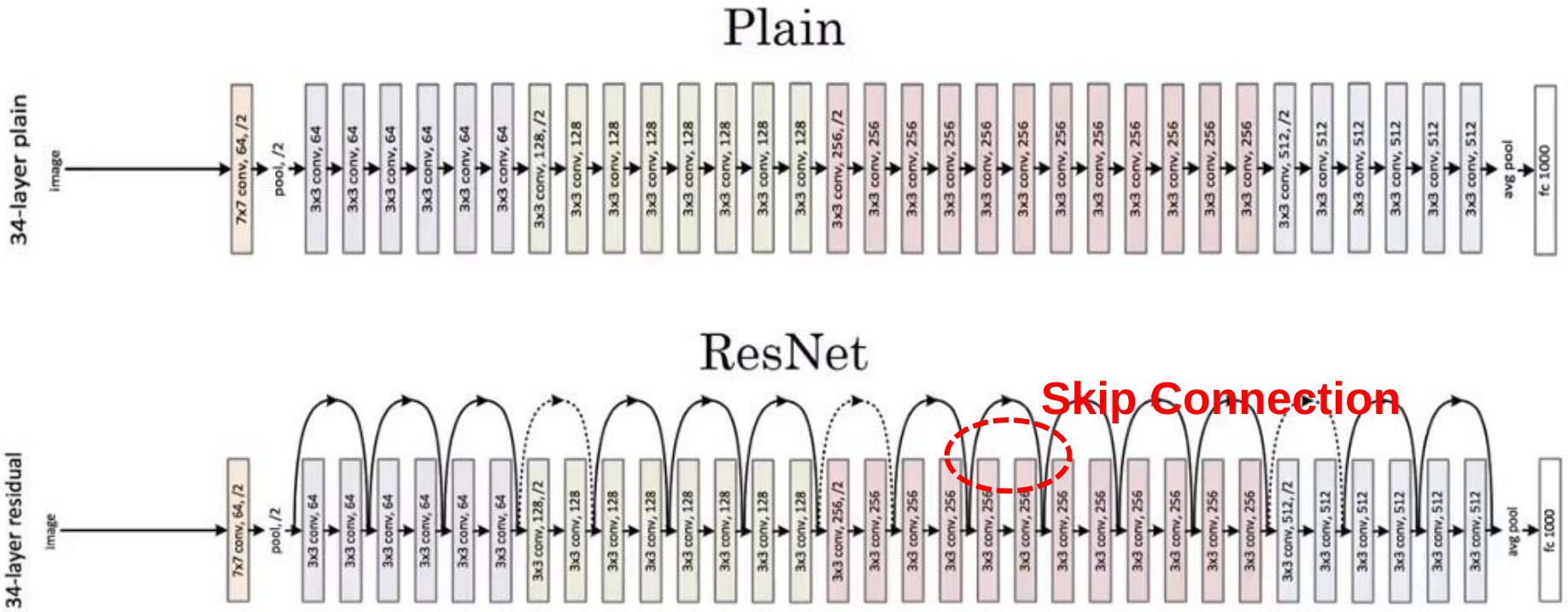


Inception module



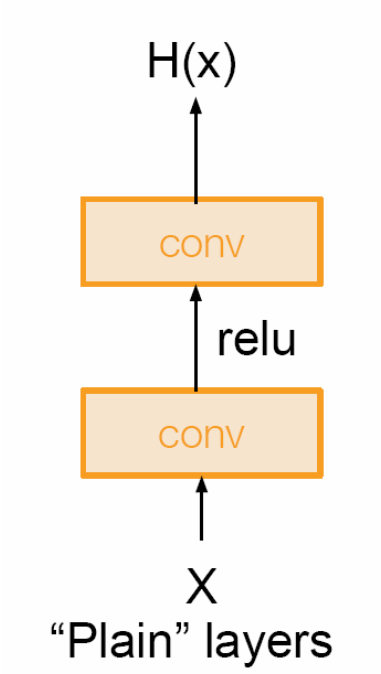
ResNet (2015)

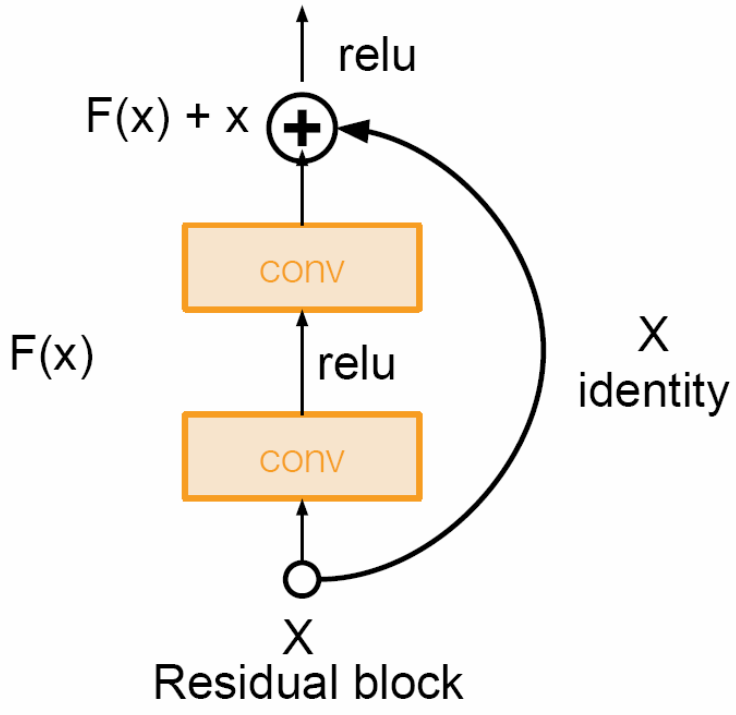
- **Extremely deep layers (152 layers)**
- **ILSVRC 2015 winner**



ResNet (2015)

- **Gradient is vanished through deep plain layers**
- **Add skip connection**
  → **Direct error propagation through this path**



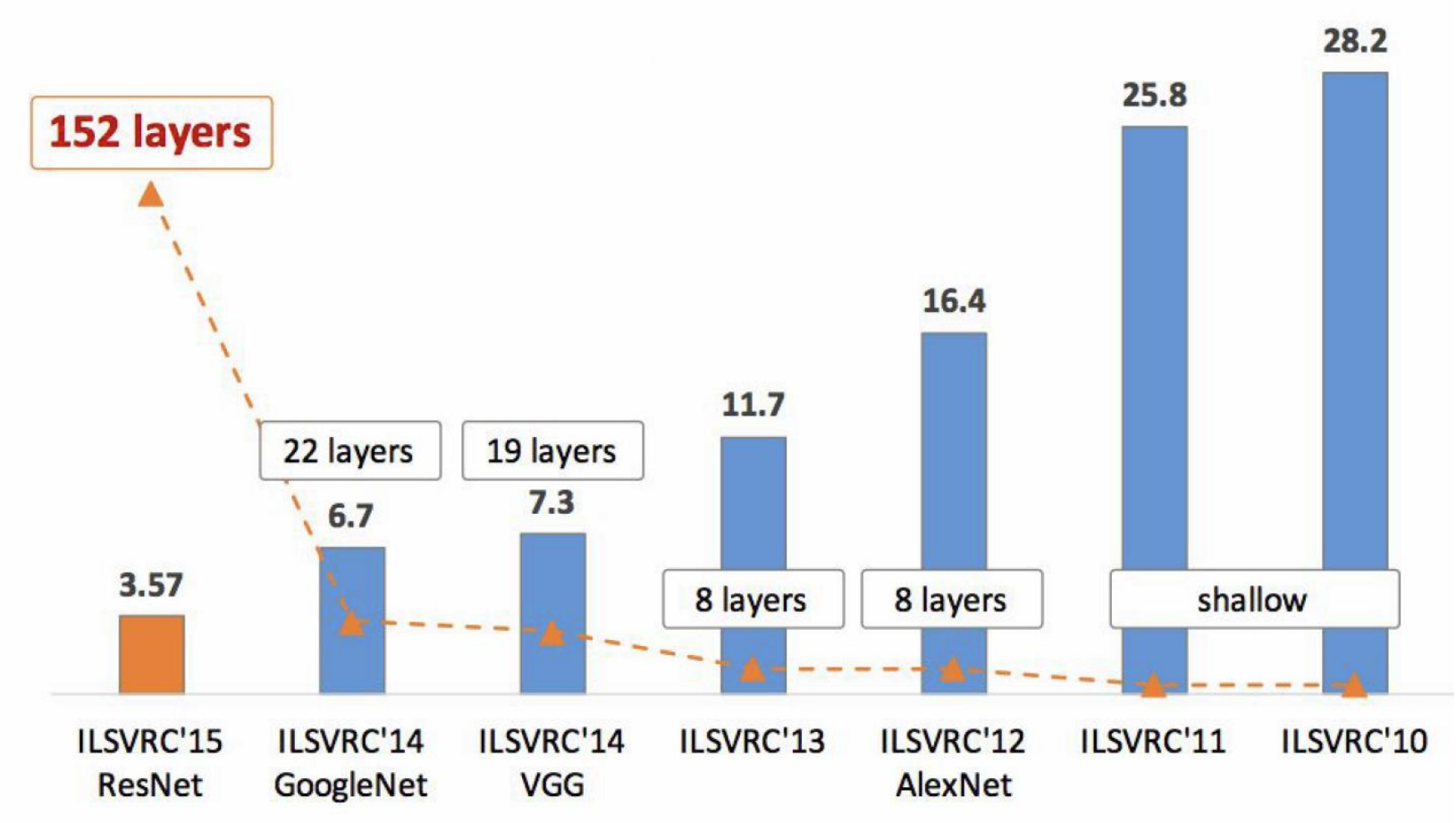


Summary of Popular DNNs

Metrics	LeNet-5	AlexNet	OverFeat (fast)	VGG-16	GoogLeNet (v1)	ResNet-50
Top-5 error	n/a	16.4	14.2	7.4	6.7	5.3
Input Size	28x28	227x227	231x231	224x224	224x224	224x224
# of CONV Layers	**2**	**5**	**5**	**16**	**21**	**49**
Filter Sizes	5	3, 5,11	3, 7	3	1, 3 , 5, 7	1, 3, 7
# of Channels	1, 6	3 - 256	3 - 1024	3 - 512	3 - 1024	3 - 2048
# of Filters	6, 16	96 - 384	96 - 1024	64 - 512	64 - 384	64 - 2048
Stride	1	1, 4	1, 4	1	1, 2	1, 2
# of Weights	26k	2.3M	16M	14.7M	6.0M	23.5M
# of MACs	1.9M	666M	2.67G	15.3G	1.43G	3.86G
# of FC layers	**2**	**3**	**3**	**3**	**1**	**1**
# of Weights	406k	58.6M	130M	124M	1M	2M
# of MACs	405k	58.6M	130M	124M	1M	2M
Total Weights	**431k**	**61M**	**146M**	**138M**	**7M**	**25.5M**
Total MACs	**2.3M**	**724M**	**2.8G**	**15.5G**	**1.43G**	**3.9G**

Trends of CNN Nets

- **The deeper layers, the more accurate**



Outline

-Neural Network for Intelligent SoC Robot-

1. *Introduction of AI for Robot*
2. *Pattern Recognition (PR)*
3. *Convolution Neural Network (CNN)*
4. *Unsupervised Learning*
5. *Reinforcement Learning*
6. *Additional Learning Algorithm*

Outline

-Neural Network for Intelligent SoC Robot-

1. *Introduction of AI for Robot*
2. *Pattern Recognition (PR)*
3. *Convolution Neural Network (CNN)*
4. ***Unsupervised Learning***
   : Basics of Unsupervised Learning
5. *Reinforcement Learning*
6. *Additional Learning Algorithm*

Unsupervised Learning

- No external teaching signal to tell the robot if it has produced the correct response

- Therefore, to learn in this type of problem an internal measure is required and a new type of machine learning is needed

- There are many different machine learning schemes for unsupervised learning

Outline

-Neural Network for Intelligent SoC Robot-

1. *Introduction of AI for Robot*
2. *Pattern Recognition (PR)*
3. *Convolution Neural Network (CNN)*
4. *Recurrent Neural Network (RNN)*
5. ***Unsupervised Learning***
   : Clustering – Nearest Neighbor Search Algorithm
6. *Reinforcement Learning*
7. *Additional Learning Algorithm*

Clustering

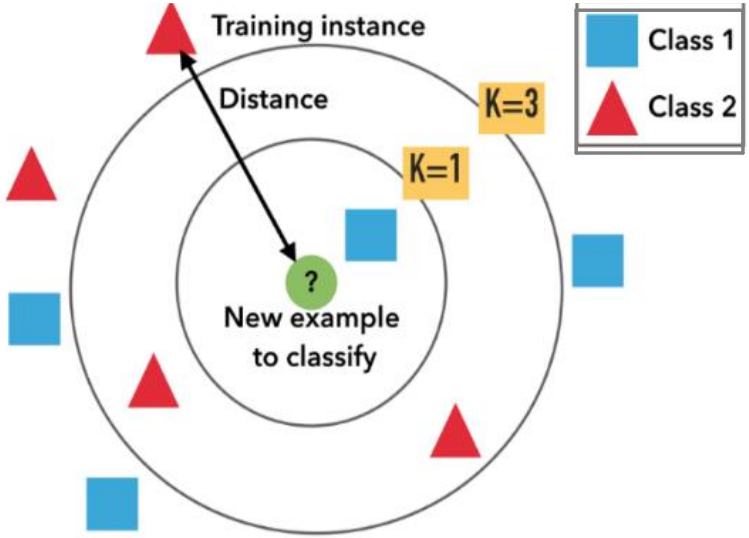
- One thing that the robot may want to do is to cluster similar perceptions together

- This is something that statisticians have done for years

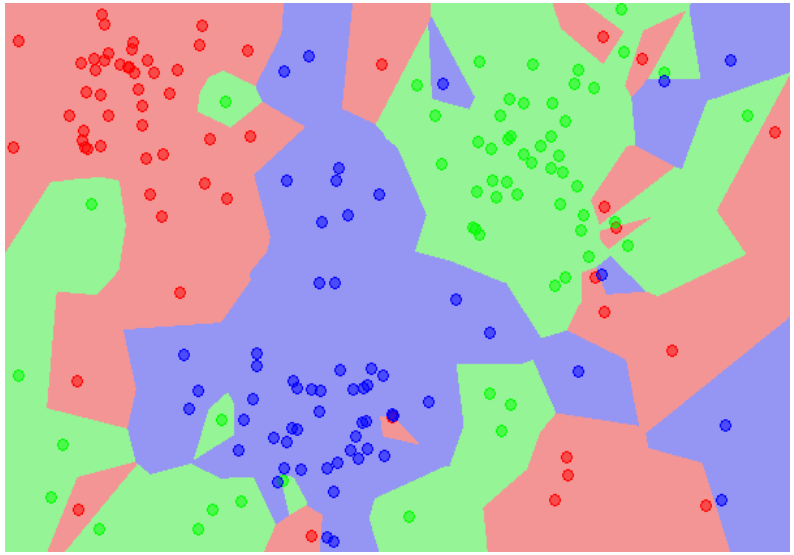
- The simplest method is …

Nearest Neighbor Search (NNS) Algorithm

- One kind of AI classification algorithm
- Find nearest data in database
- Classifying Input data as nearest data's label



<Concept of k-Nearest Neighbor>

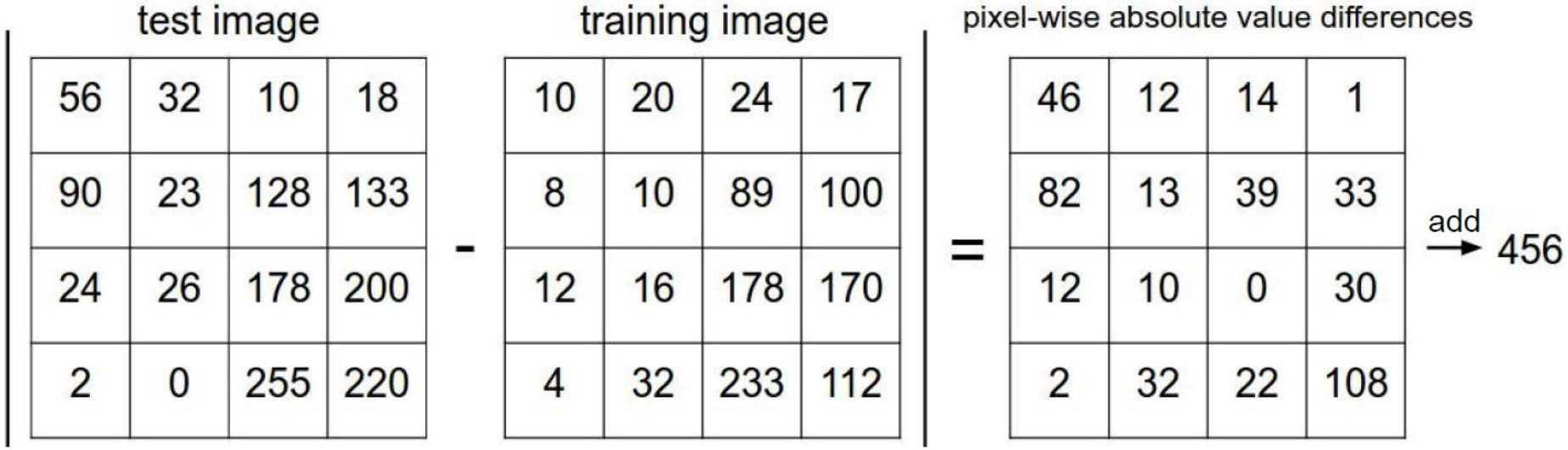


<kNN Classification Map>

Nearest Neighbor

Distance Metric

L1 distance: $$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



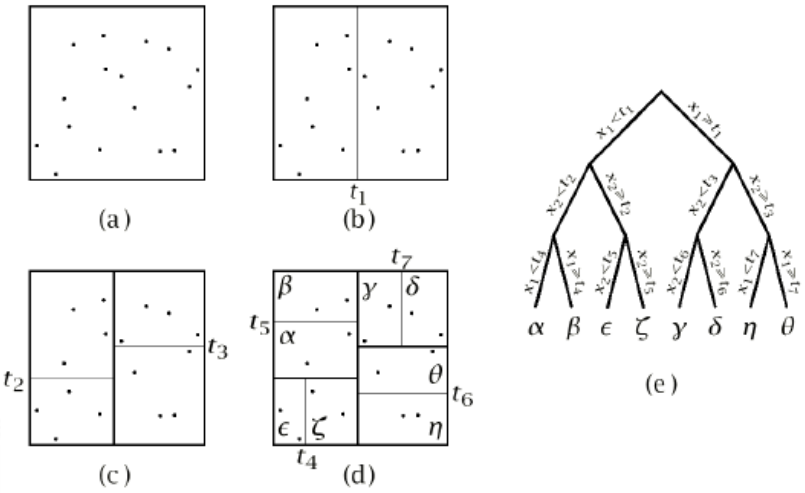
Hyperparameter

- **L1 Manhattan Distance**

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

- **L2 Euclidean Distance**

$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



- **P-Norm**

$$||x||_p = \left(|x_1|^p + \cdots + |x_n|^p\right)^{\frac{1}{p}} \qquad p \geq 1, x \in \mathbb{R}^n$$

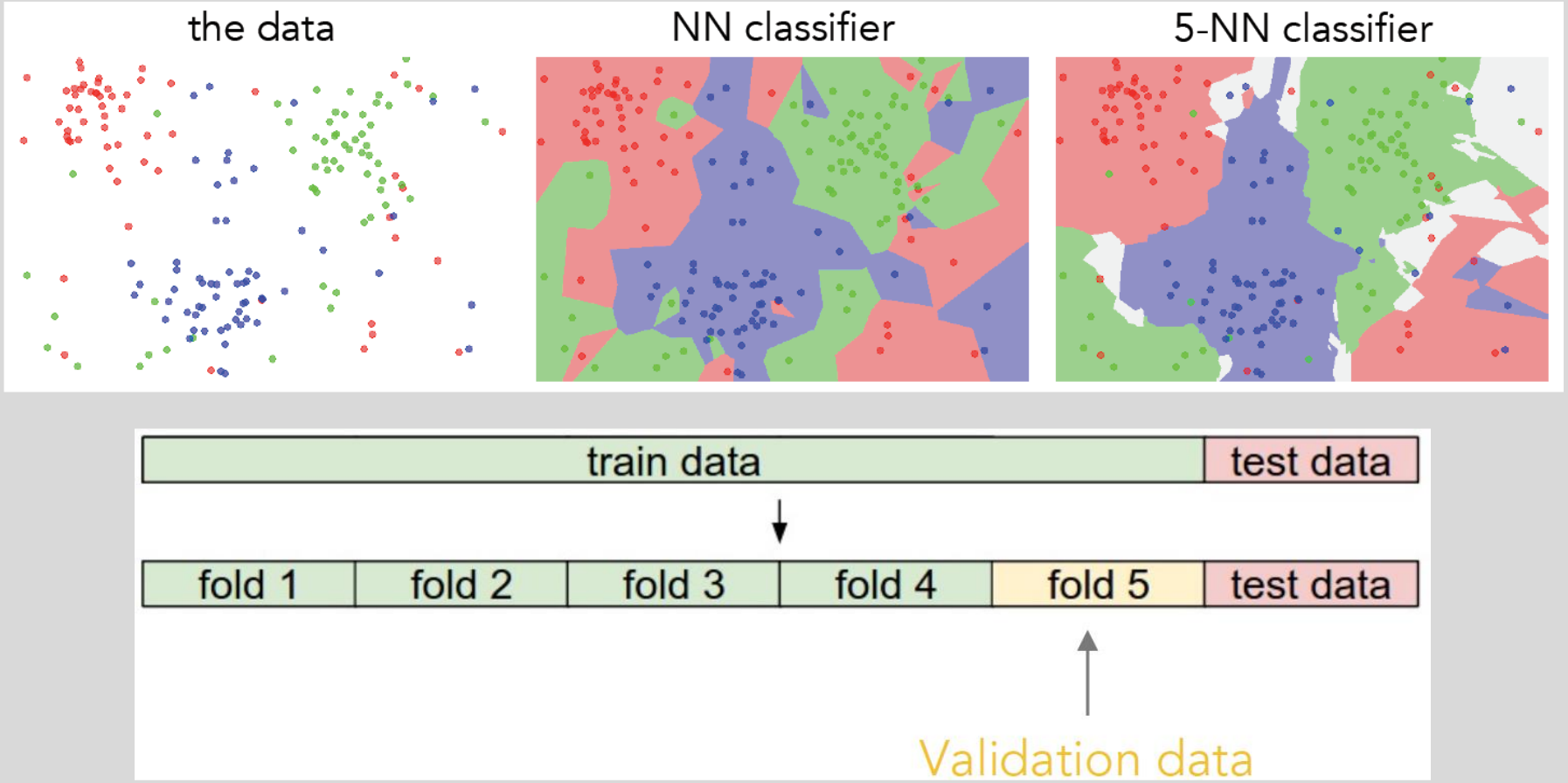
Nearest Neighbor Training and Test

1. **Memorize all of the training data**
2. **For every test input image**
    1. Find the nearest train image by using L1 distance
    2. Predict the label with the obtained nearest training Image

- **Training is simple (Just store data in memory)**
- **Test is complicated N train image x M test image**
- **Approximate NN, k-NN**

Example : K-Nearest Neighbor

- Find the k nearest neighbors and use them to vote the label



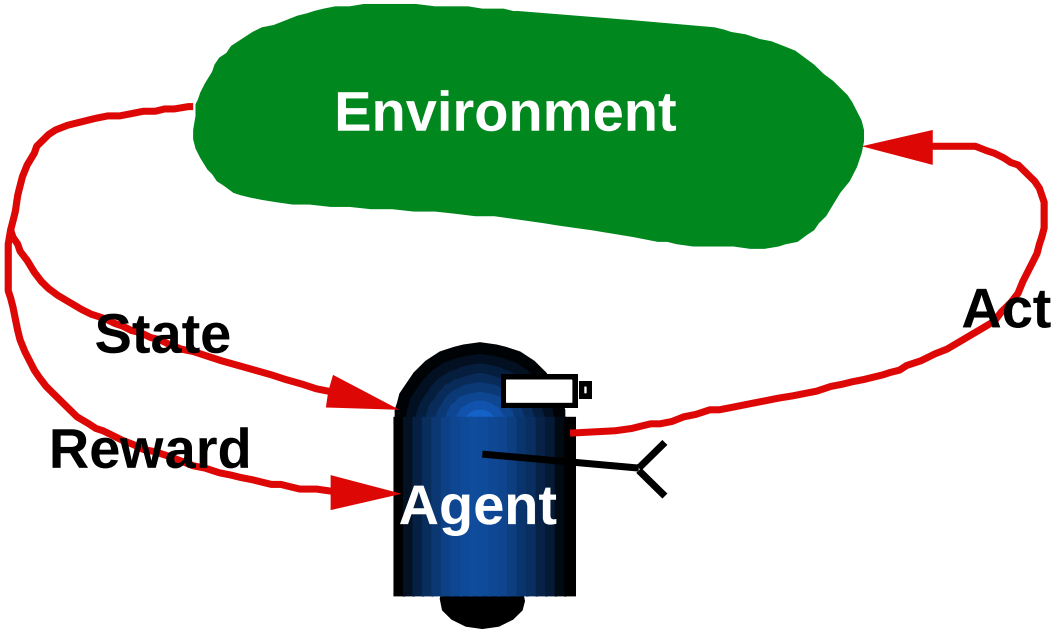
Outline

-Neural Network for Intelligent SoC Robot-

1. *Introduction of AI for Robot*
2. *Pattern Recognition (PR)*
3. *Convolution Neural Network (CNN)*
4. *Recurrent Neural Network (RNN)*
5. *Unsupervised Learning*
6. ***Reinforcement Learning (RL)***
   : Basics of Reinforcement Learning
7. *Additional Learning Algorithm*

State, Action, Reward

- State : State that the agent recognizes
- Action : Behavior from current state
- Reward : Indicate how well agent is doing
  - Ex) Atari game : score
  - Ex) Go : Win/lose



- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}

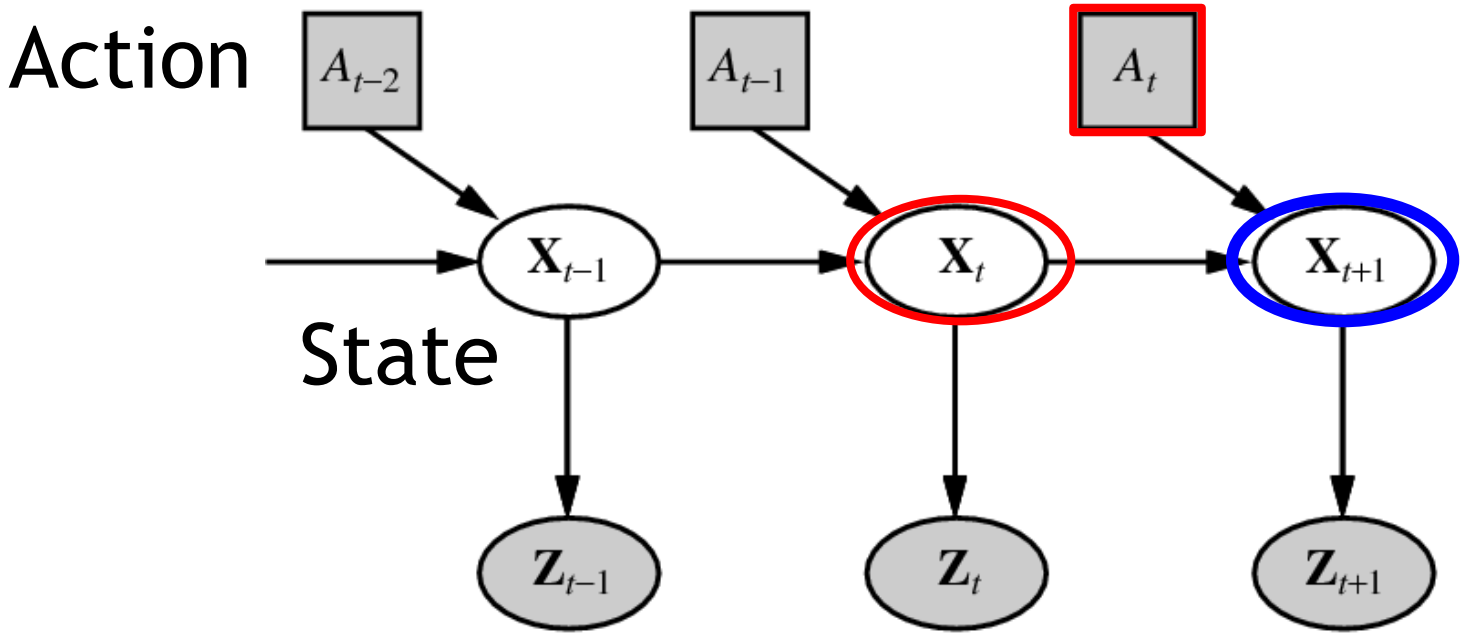
Markov Assumption

$P(s_t \mid a_t, s_{t-1})$: pdf a_t changes the state from s_{t-1} to s_t

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

$$=$$

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$



Optimal Quantities

- The value of a state s:

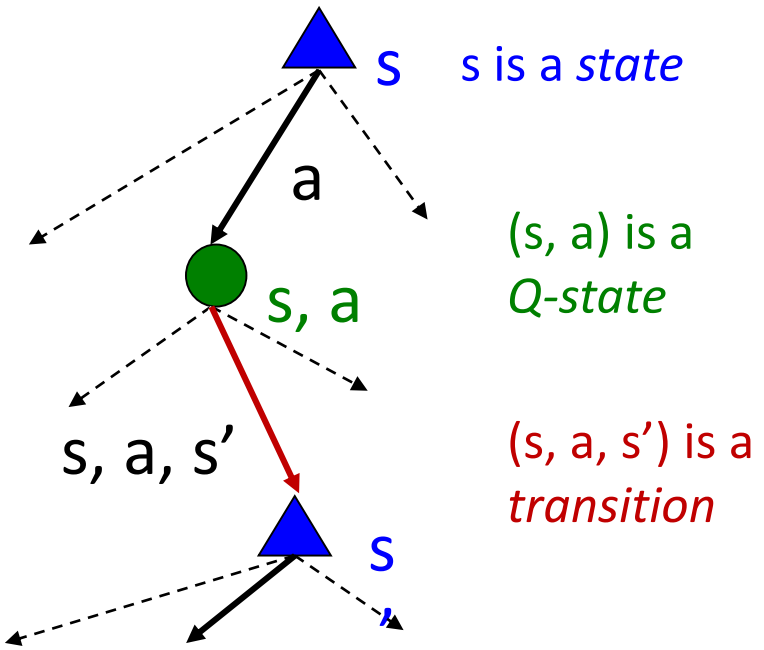
  $V^*(s)$ = expected reward starting in s and acting optimally

- The value of a q-state (s, a):

  $Q^*(s, a)$ = expected reward starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

  $\pi^*(s)$ = optimal action from state s

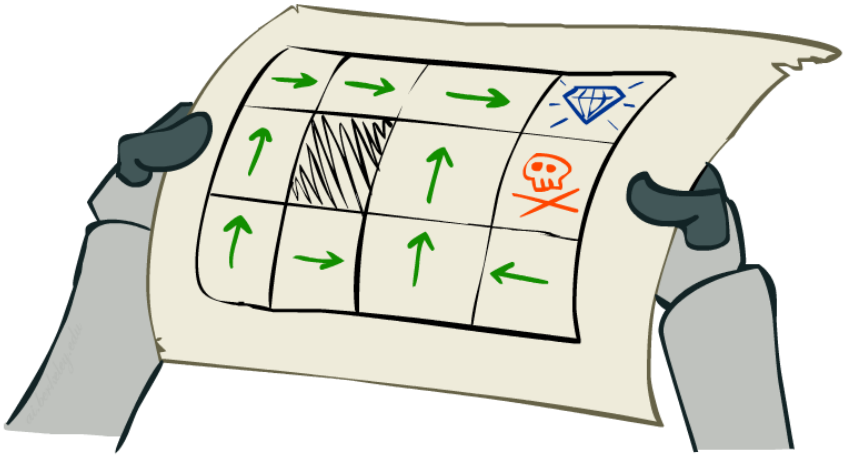


Policies

- We want an optimal plan, or sequence of actions, from start to a goal

- An optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected reward if followed

- Expectimax didn't compute entire policies
  - It computed the action for a single state only



Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

Value function

- Value function : Prediction of future reward

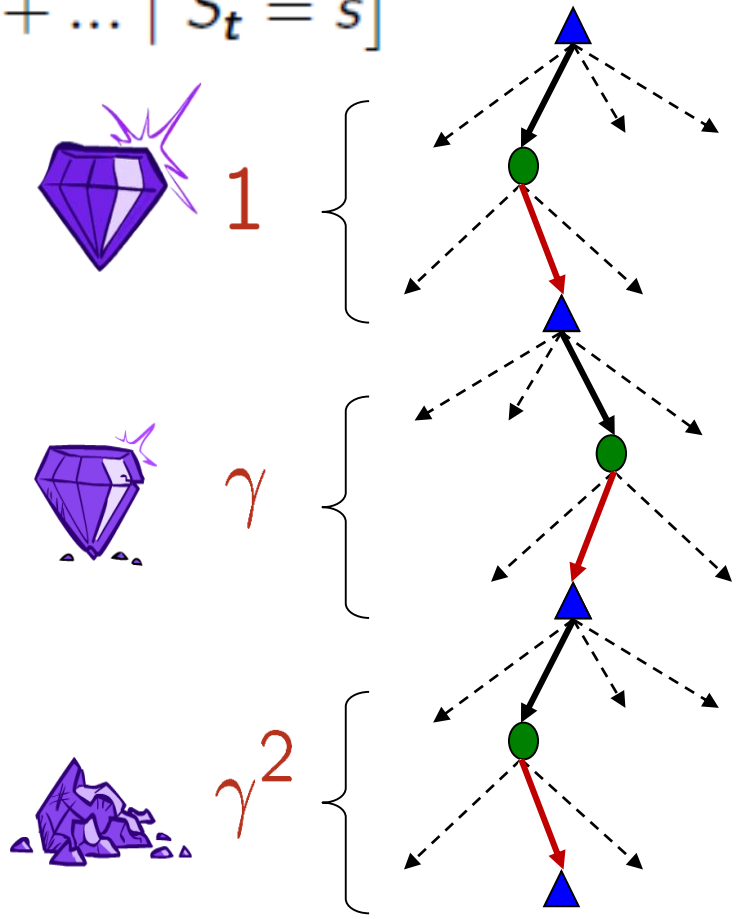
$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \right]$$

- Additive

$$V([r_0, r_1, r_2, \ldots.]) = r_0 + r_1 + r_2 + \cdots$$

- Discounted

$$V([r_0, r_1, r_2, \ldots.]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$$



Outline

-Neural Network for Intelligent SoC Robot-

1. *Introduction of AI for Robot*
2. *Pattern Recognition (PR)*
3. *Convolution Neural Network (CNN)*
4. *Recurrent Neural Network (RNN)*
5. *Unsupervised Learning*
6. ***Reinforcement Learning (RL)***
   : Example - Value-based RL
7. *Additional Learning Algorithm*

Value-based RL

- Value-based reinforcement learning
  - Value function is approximated by Q
  - Policy is directly determined from value function
- Update Q-value at each Q-state:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

  - But can't compute this update without knowing T, R

- Instead, compute average as we go
  - Receive a sample transition (s, a, r, s')
  - This sample suggests

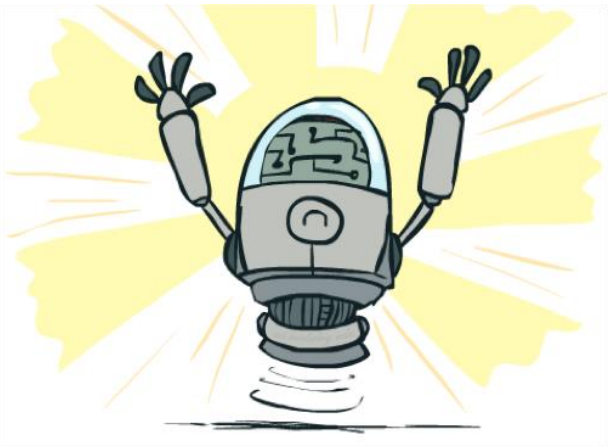
$$Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$$

  - But we want to average over results from (s, a)
  - So keep a running average

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha) \left[r + \gamma \max_{a'} Q(s',a') \right]$$

Q-Learning Properties

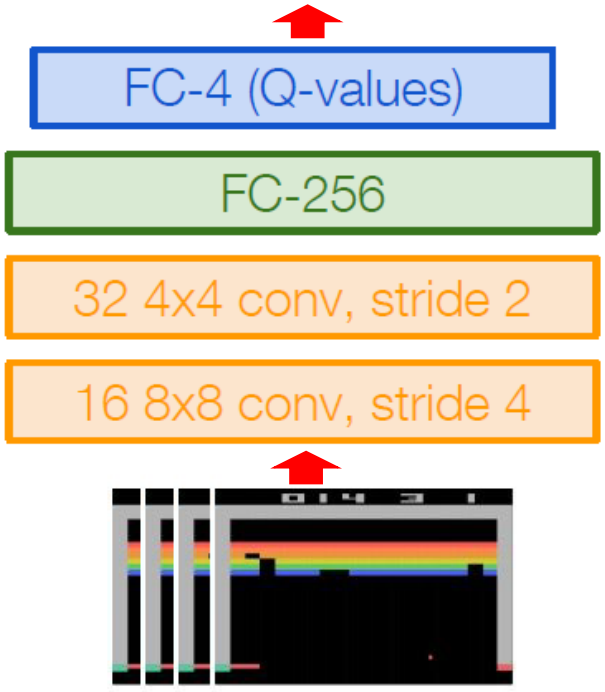
- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!

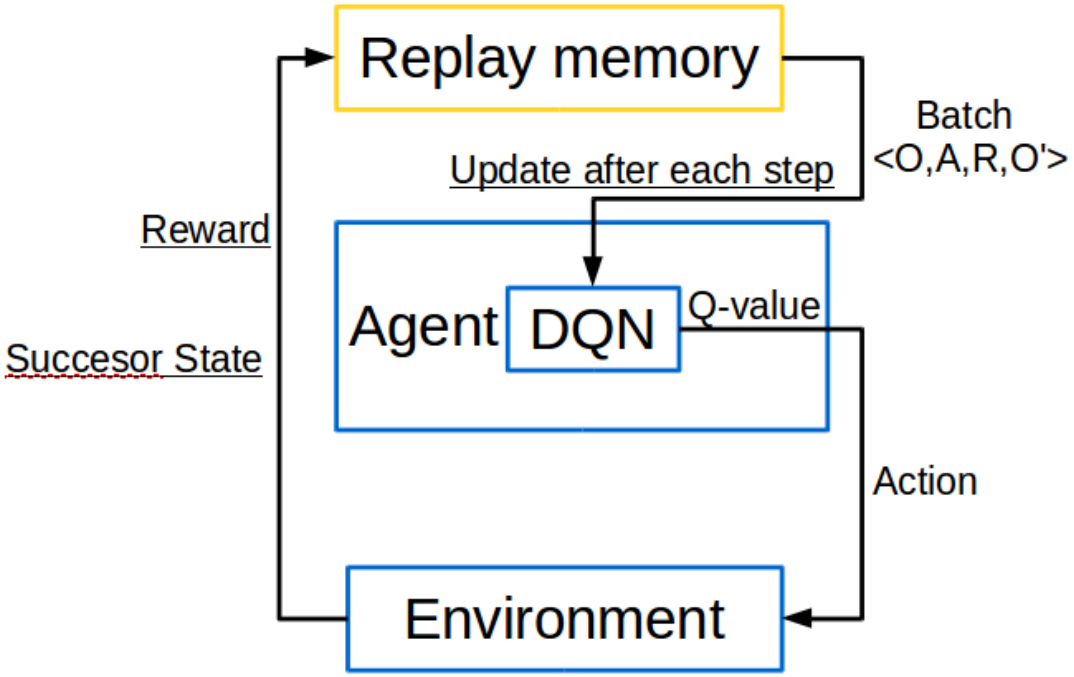


- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions

Value-based RL - Deep Q Learning

- Deep reinforcement learning with CNN
  - Raw pixel as the input data of CNN
  - Function approximator : CNN
  - Experience replay : mini-batch learning with history data





Linear Value Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few numbers

- Disadvantage: states may share features but actually be very different in value!

Approximate Q-Learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

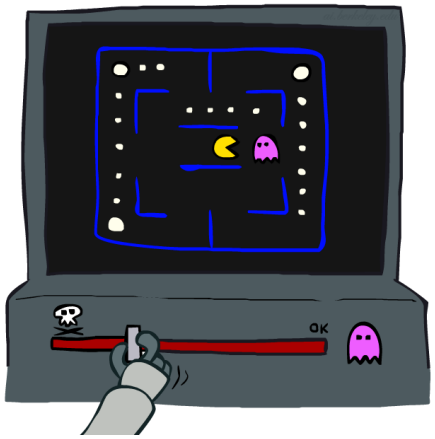
- Q-learning with linear Q-functions:

$$\text{transition} = (s, a, r, s')$$

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$$

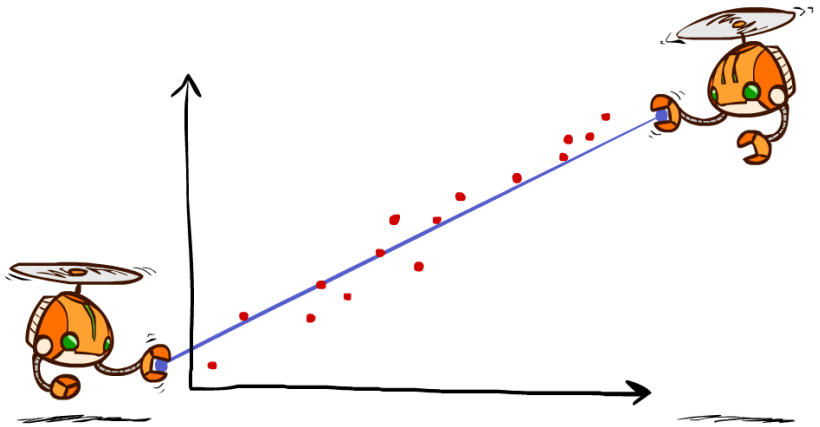
$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}] \quad \text{Exact Q's}$$

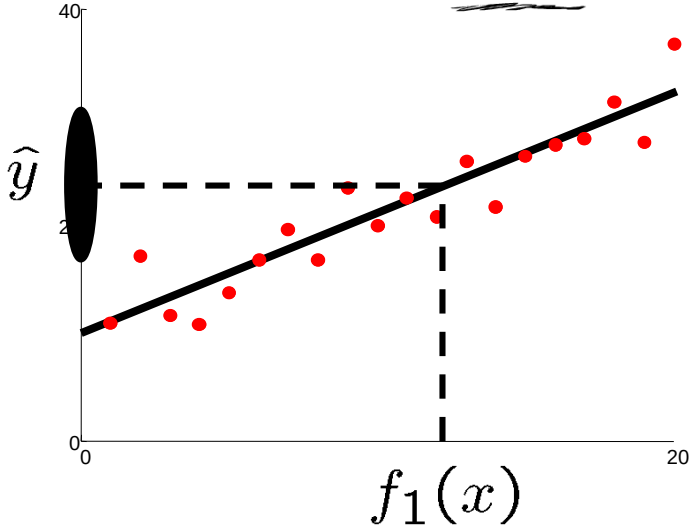
$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a) \quad \text{Approximate Q's}$$

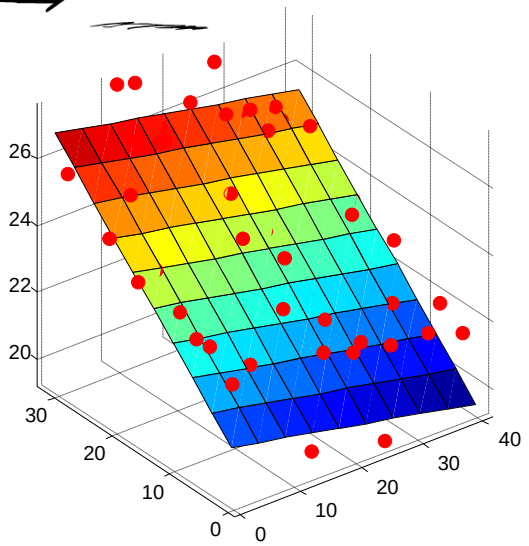


- Intuitive interpretation:
  - Adjust weights of active features
- Formal justification: online least squares

Q-Learning







Prediction:

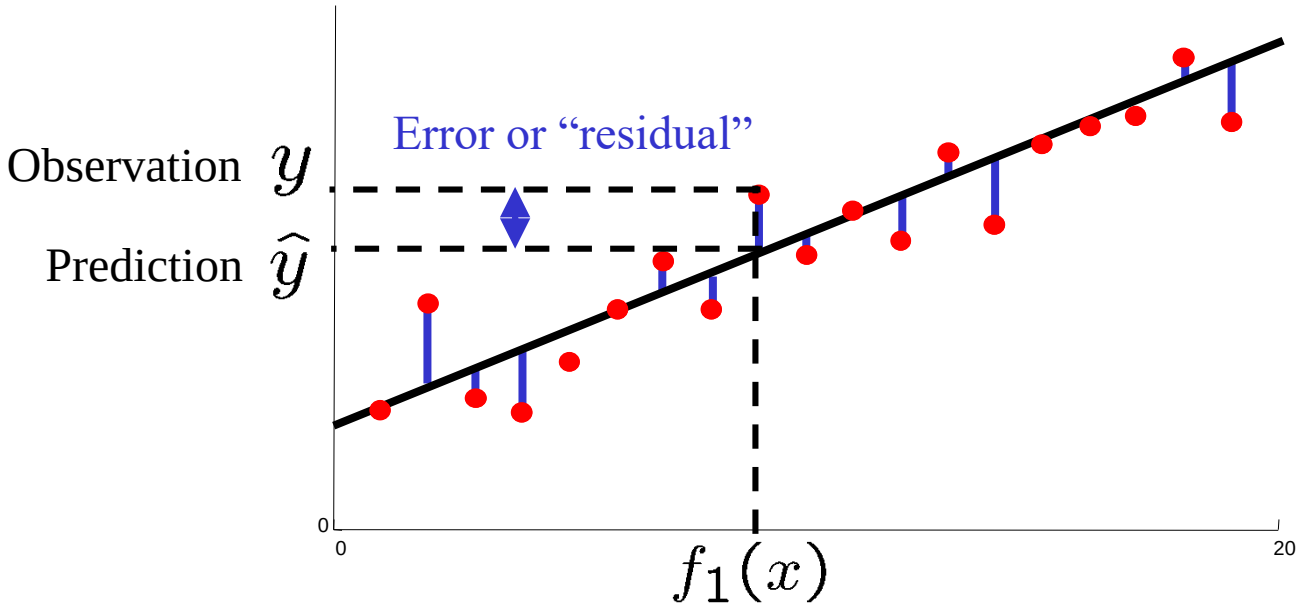
$$\widehat{y} = w_0 + w_1 f_1(x)$$

Prediction:

$$\widehat{y}_i = w_0 + w_1 f_1(x) + w_2 f_2(x)$$

Q-Learning: Least Squares*

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i) \right)^2$$



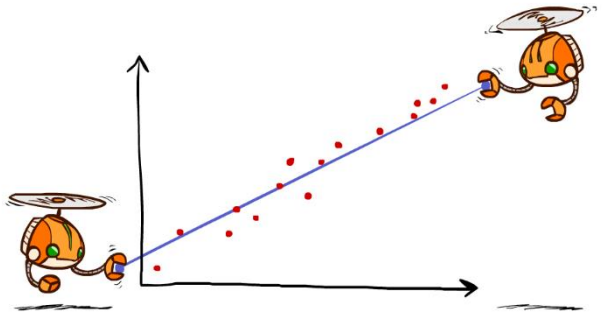
Minimizing Error*

Imagine we had only one point x, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\ \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$



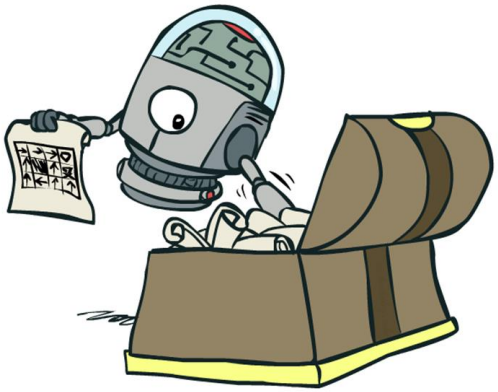
Approximate q update explained:

$$w_m \leftarrow w_m + \alpha\left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

"target" "prediction"

Policy Search

- Problem: often the feature-based policies that work well (win games, maximize expected rewards) aren't the ones that approximate V / Q best
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)
  - Distinction between modeling and prediction again later in the course



- Solution: learn policies that maximize rewards, not the values that predict them

- Policy search: start with an Q-learning then fine-tune by hill climbing on feature weights

Policy Search

- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Fine tune each feature weight up and down and see if your policy is better than before
- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical
- Better methods exploit DNN!

Outline

-Neural Network for Intelligent SoC Robot-

1. *Introduction of AI for Robot*
2. *Pattern Recognition (PR)*
3. *Convolution Neural Network (CNN)*
4. *Recurrent Neural Network (RNN)*
5. *Unsupervised Learning*
6. ***Reinforcement Learning (RL)***
   : Applications of RL
7. *Additional Learning Algorithm*

Applications of RL - Game

Google Deepmind - Alpago

Starcraft AI

Google Deepmind - Atari

- Suitable for using to determine the next behavior for current state
- Easy to use Reward-specific applications such as winning games

Outline

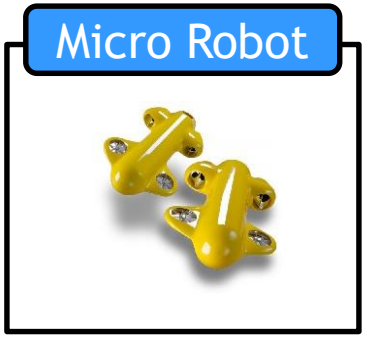
-Neural Network for Intelligent SoC Robot-

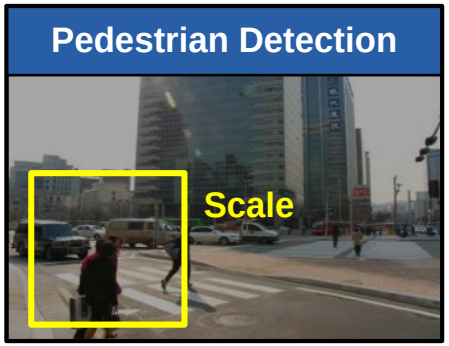
1. *Introduction of AI for Robot*
2. *Pattern Recognition (PR)*
3. *Convolution Neural Network (CNN)*
4. *Recurrent Neural Network (RNN)*
5. *Unsupervised Learning*
6. *Reinforcement Learning*
7. ***Additional Learning Algorithm***
   - ***: (Neuro-) Fuzzy Logic***

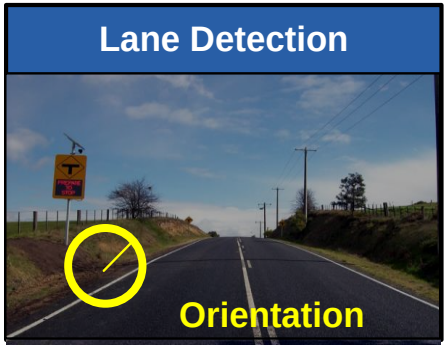
Why do robots need Neuro-fuzzy logic?

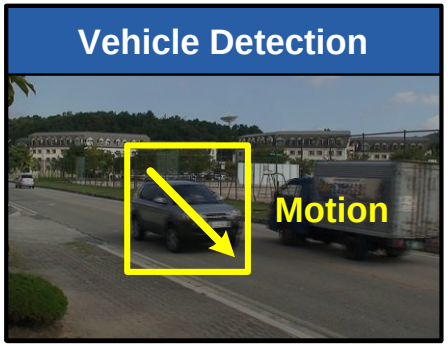
- Real world is not “crisp” but ambiguous.

  ➔**How to include the effect of ambiguity into our model?**





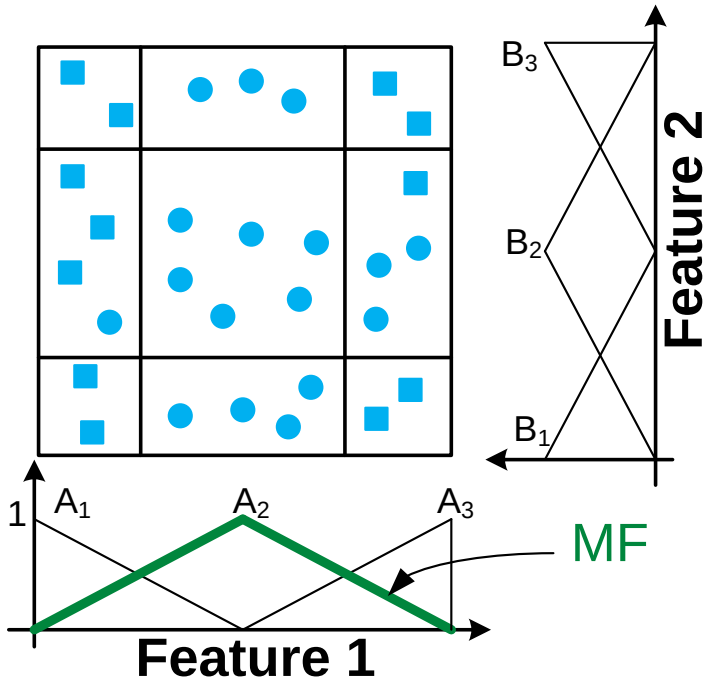




<Applications>

<Functionality for Mobile Robot>

Basic Neuro-Fuzzy Classifier



- Feature Space Partitioning
  - Membership Function (MF)

☺ **Less Complexity**

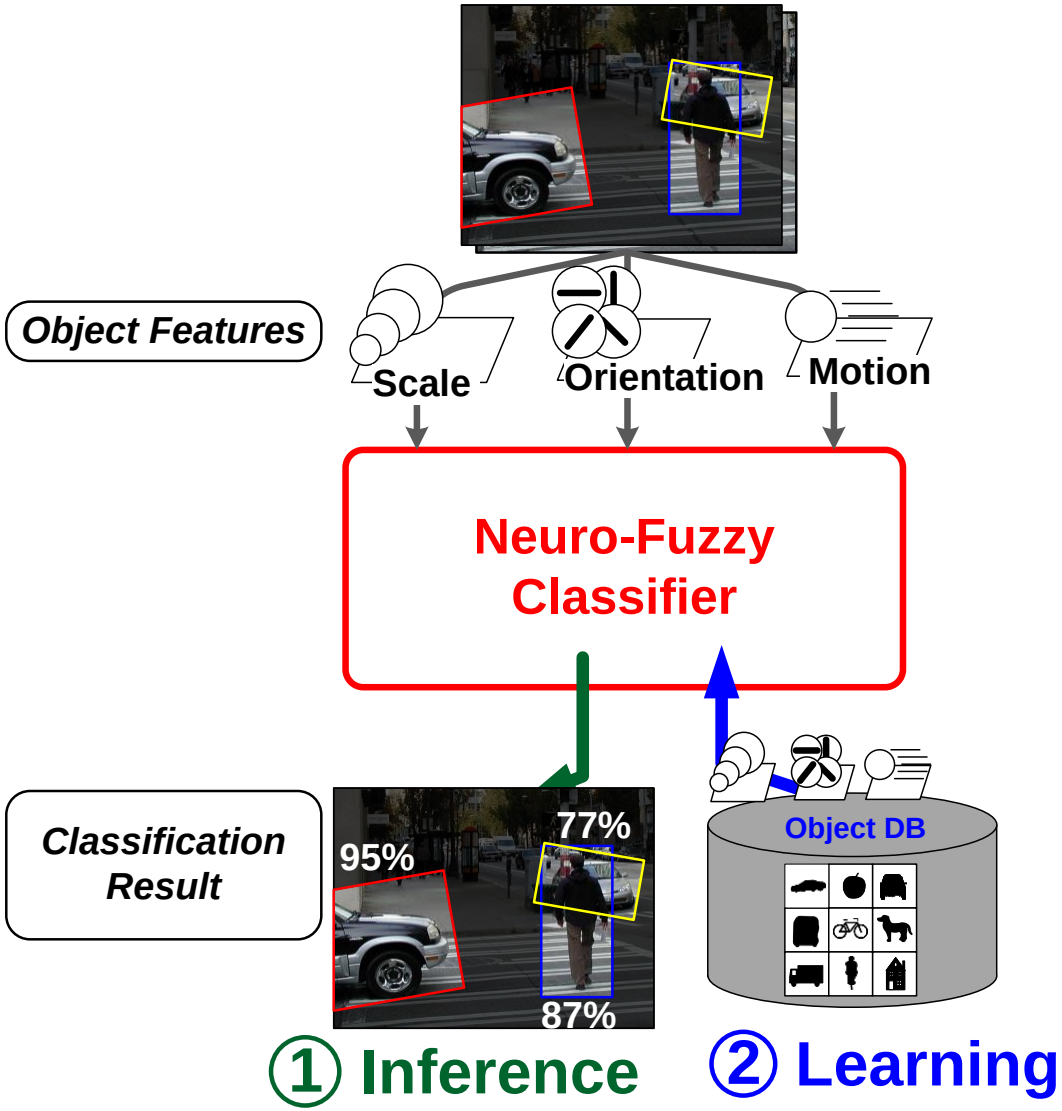
[1] I. Campo, Trans. on Fuzzy Systems,2008

- Limited Inference Structure for One Specific Algorithm[1]

→ **Degraded Accuracy for Other Classification Problems ☹**

→ **Learning process is required!**

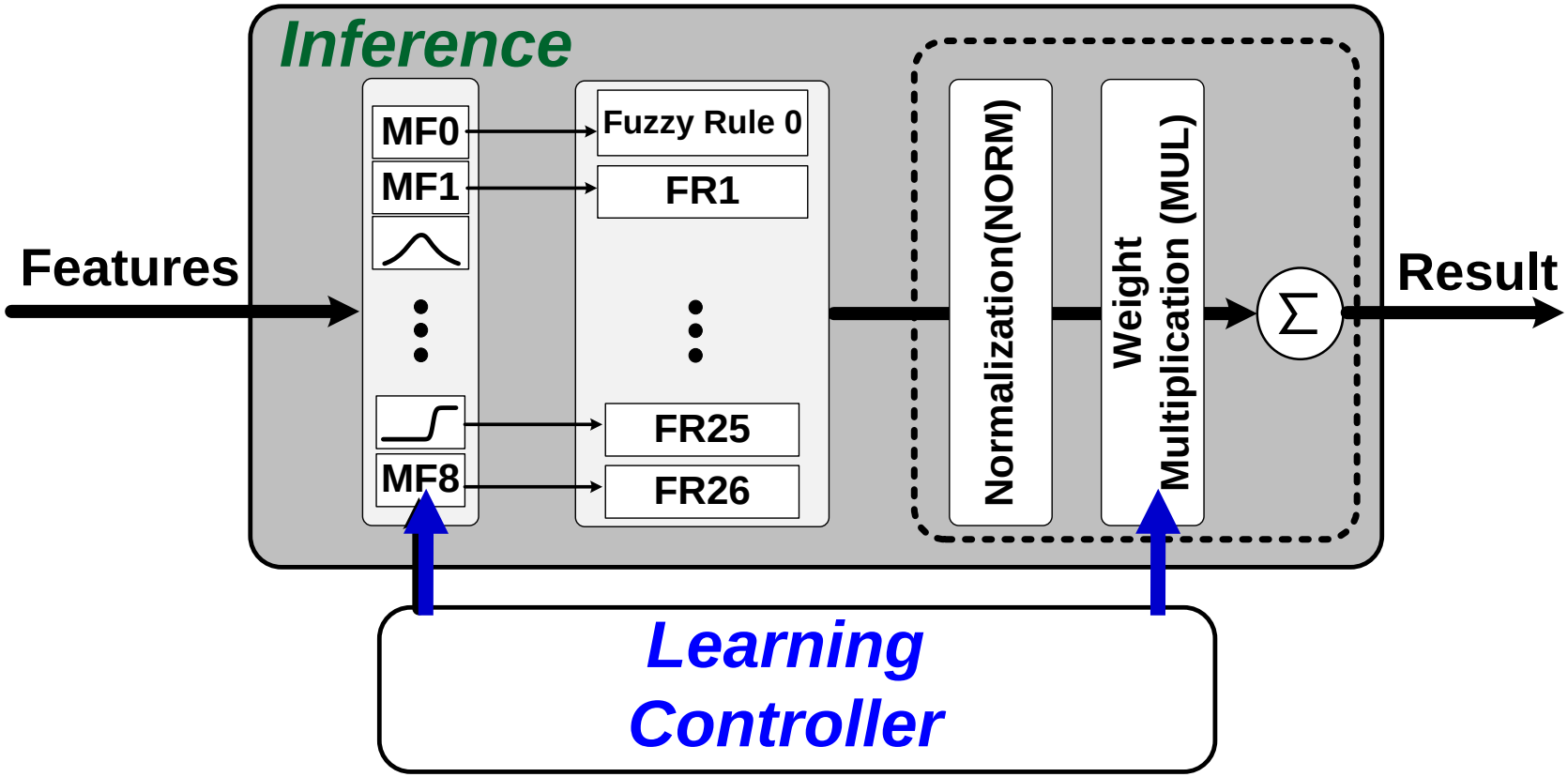
Neuro-Fuzzy Object Classifier



① Inference
- Fuzzy Logic

② Learning
- Neural Network

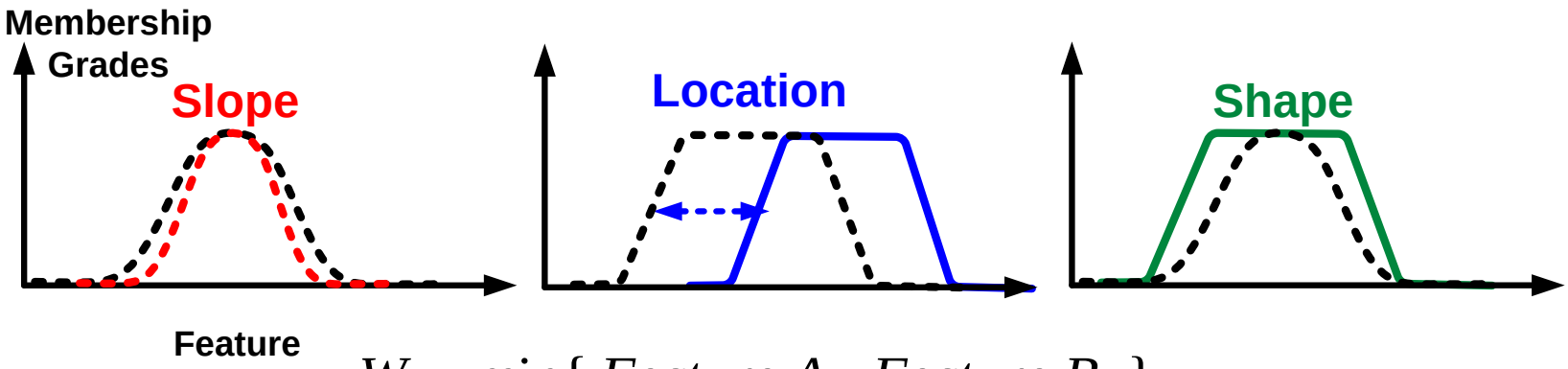
Neuro-Fuzzy Object Classifier



- Object Decision w/ Input Features by Fuzzy Rules
- Object Learning w/ Fuzz/Defuzzification

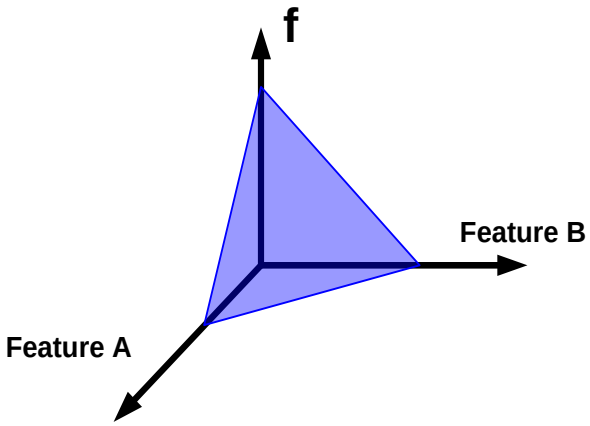
Neuro-Fuzzy Object Classifier

- **Fuzzification (MF)** – On-Line & Off-Line Learning



$$W_1 = \min\{\ Feature\ A_1,\ Feature\ B_1\ \}$$

- **Defuzzification Function** – Off-Line Learning



Inference Output

$$= \bar{W}_1 f_1 + \bar{W}_2 f_2 + \cdots + \bar{W}_{27} f_{27}$$