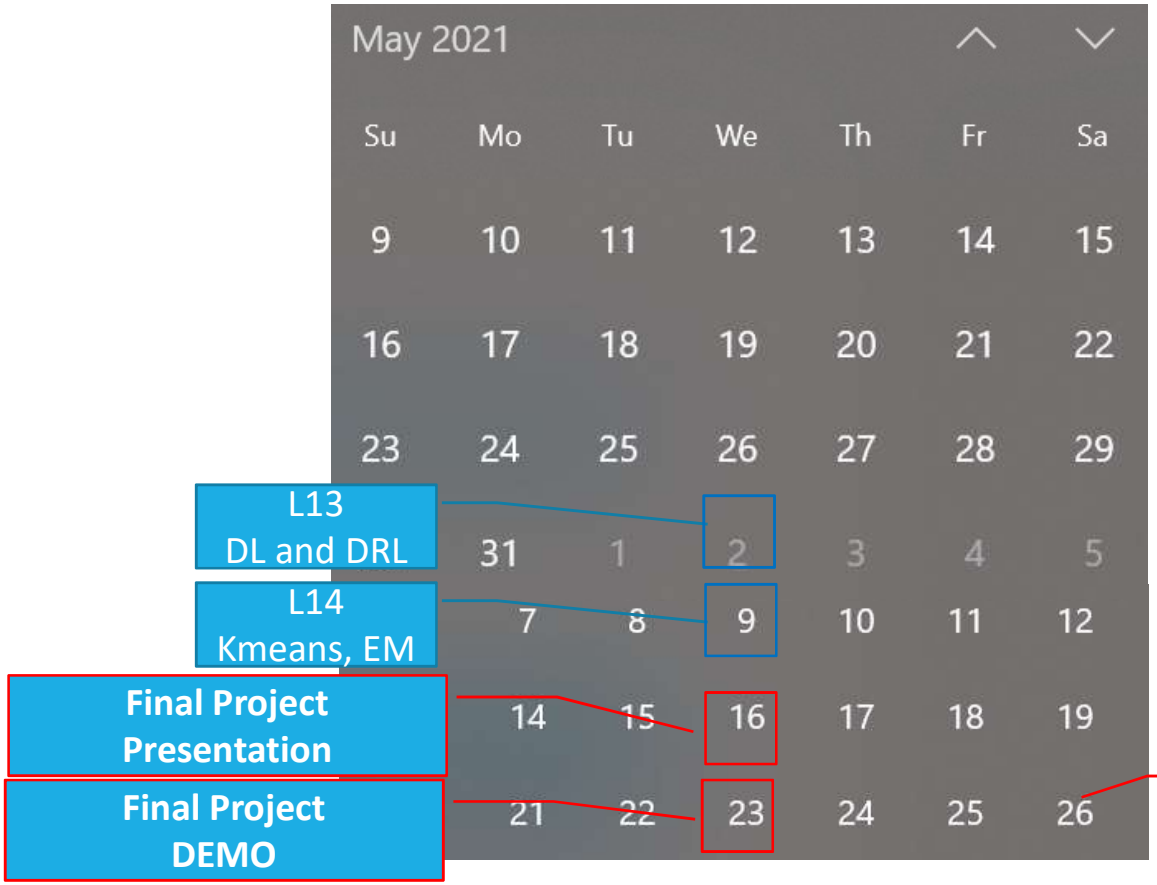
Deep Learning and Deep Reinforcement Learning

KUO-SHIH TSENG
DEPARTMENT OF MATHEMATICS
NATIONAL CENTRAL UNIVERSITY, TAIWAN

2021/06/02

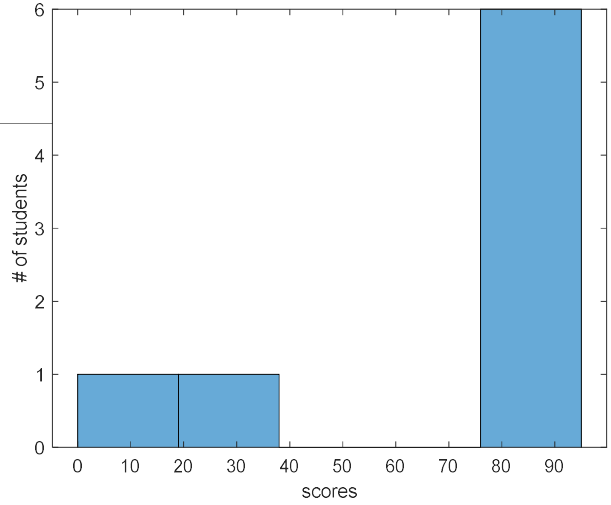
Course Announcement

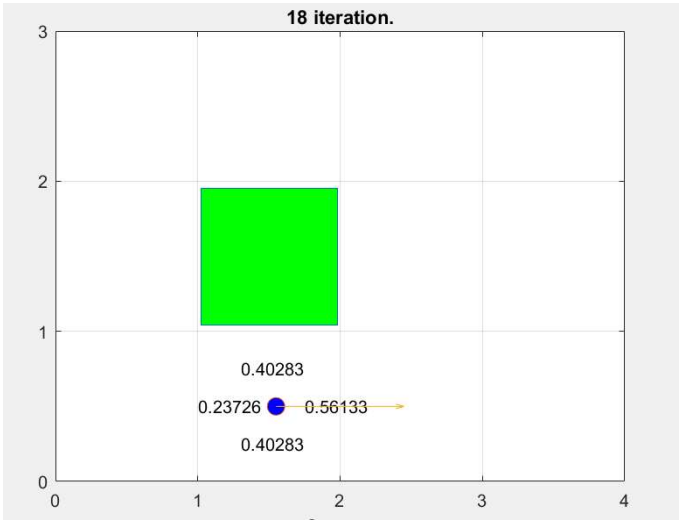


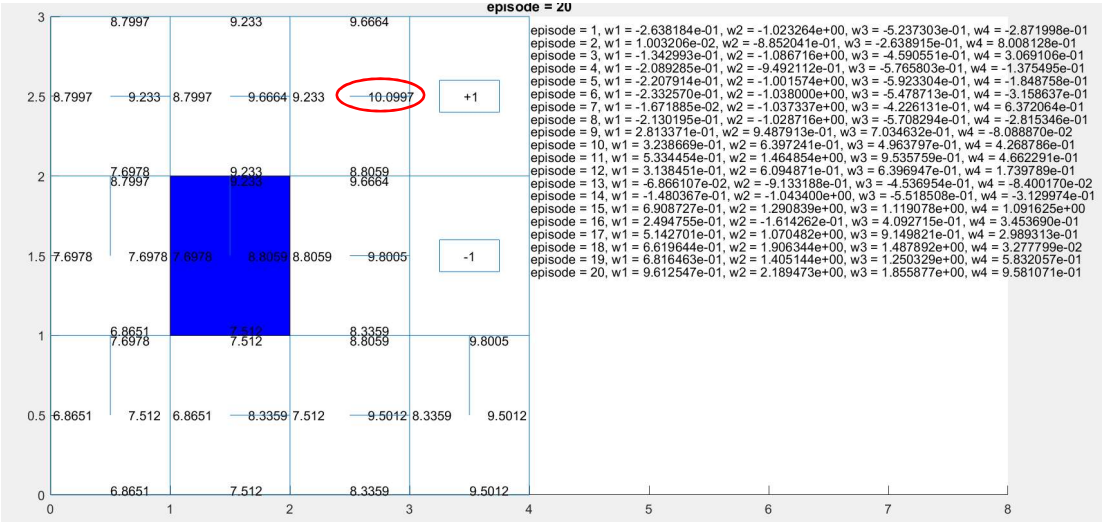
Please take your time on your project!
There are only 3 weeks!
We will have presentation and demo via google meet.

Course Announcement

- HW3



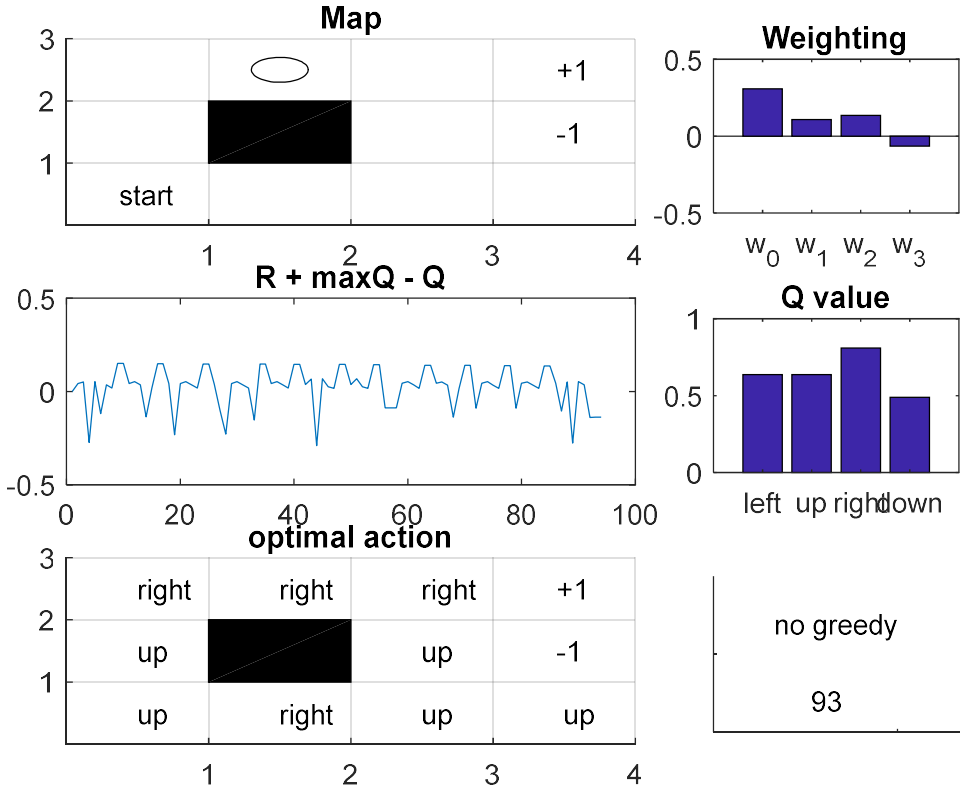




Course Announcement

- HW3
  - P1: Q-learning
  - P2: Leg detection
  - P3: Q-learning to LRTA*

$$w_i \leftarrow w_i + \alpha\left[R(\mathrm{s},\mathrm{a}) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] f_i$$



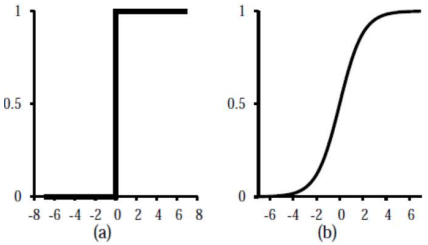
Results from: Bing-Xian Lu

Course Announcement

- HW3 : Adaboost>NB>LR~=Perceptron

Naive Bayse :	Perceptron :	Logistic :	Adaboost :
Train	Train	Train	Train
TP = 114 | FN = 2	TP = 0 | FN = 116	TP = 0 | FN = 116	TP = 96 | FN = 20
FP = 620 | TN = 771	FP = 0 | TN = 1391	FP = 0 | TN = 1391	FP = 106 | TN = 1285
Accuracy : 0.5873	Accuracy : 0.9230	Accuracy : 0.9230	Accuracy : 0.9164
Precision : 0.1553	Precision : NaN	Precision : NaN	Precision : 0.4752
Recall : 0.9828	Recall : 0.0000	Recall : 0.0000	Recall : 0.8276
Test	Test	Test	Test
TP = 114 | FN = 1	TP = 0 | FN = 115	TP = 0 | FN = 115	TP = 90 | FN = 25
FP = 648 | TN = 766	FP = 0 | TN = 1414	FP = 0 | TN = 1414	FP = 122 | TN = 1292
Accuracy : 0.5755	Accuracy : 0.9248	Accuracy : 0.9248	Accuracy : 0.9039
Precision : 0.1496	Precision : NaN	Precision : NaN	Precision : 0.4245
Recall : 0.9913	Recall : 0.0000	Recall : 0.0000	Recall : 0.7826

$$Accuracy = (TP + TN) / N$$
$$\Pr ecision = TP / (TP + FP)$$
$$\mathrm{Re}\, call = TP / (TP + FN)$$

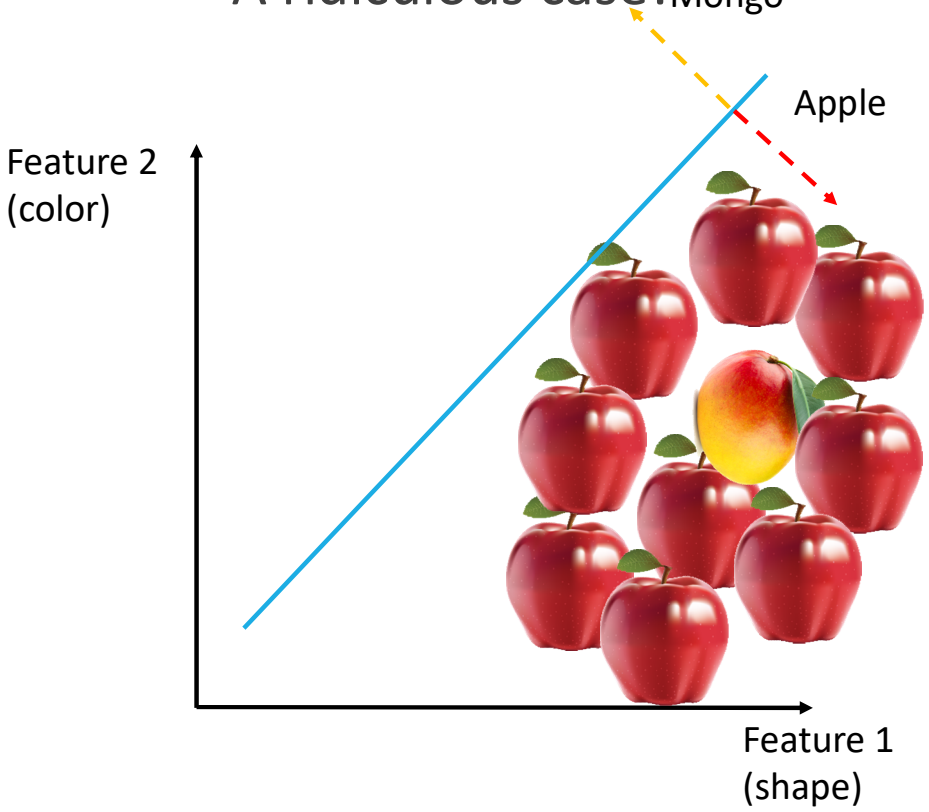


$$L(\mathbf{w}) = (y - h_{\mathbf{w}}(\mathbf{x}))^2$$

Results from:
Yu-Chung Tsai

Course Announcement

- A ridiculous case!



		True value: T	True value: N
Predicted value	T	True Positive (0)	False Positive(0)
	N	False Negative(1)	True Negative(9)

$$Accuracy = (TP + TN) / N$$
$$\Pr ecision = TP / (TP + FP)$$
$$\mathrm{Re}\, call = TP / (TP + FN)$$

$$Accuracy = \frac{0+9}{10} = 0.9$$
$$Precision = \frac{0}{0+0} = \infty$$
$$Recall = \frac{0}{0+1} = 0$$

Course Announcement

- Just guess all of data are “not legs”!
- You got high accuracy!

Predicted value	True value T	True value N
T	True Positive (0)	False Positive(0)
N	False Negative(116)	True Negative(1391)

Training Data:
TP=0 FP=0 TN=1391 FN=116
Accuracy=0.92 precision=infinity Recall=0

$$Accuracy = (TP + TN) / N$$
$$\Pr ecision = TP / (TP + FP)$$
$$\mathrm{Re}call = TP / (TP + FN)$$

$$Accuracy = \frac{0 + 1391}{1391 + 116} = 0.92$$
$$Precision = \frac{0}{0 + 0} = \infty$$
$$Recall = \frac{0}{0 + 116} = 0$$

How to improve it?

$$L(\mathbf{w}) = (y - h_{\mathbf{w}}(\mathbf{x}))^2$$

Course Announcement

- Project meeting: (6/2, 6/9. 6/16)
  - 13:30~13:45: 陳宇揚
  - 13:45~14:00: 李家妤
  - 14:00~14:15: 陳羽暉 蔡沐霖 高文顯
  - 14:15~14:30: 邱韋翔
  - 14:30~14:45: 林寶德
  - 14:45~15:00: 張軒旗

Course Announcement

- Final project:
- You should work with your partner **everyday**!
- The recommended schedule is as follows:
- 6/2~6/9:
  - You should know the MDP for your project
  - You should collect data from your robot/gazebo
  - **Each** team member writes the code (e.g., Q-learning and feature extractions) and test the code using the data (**parallel**).
  - Integrate the code.
- 6/9~6/16:
  - Integrate the code.
  - Let the robot **learn**.
    - Tune features
    - Tune parameters

Course Announcement

- How to learn programming?
  - Taking a programming course (with hand-on training)
  - Project-oriented training (with Google)
  - Learning from experts (leading engineers)

- How to improve your programming skills
  - Keep working on projects
  - Building your debugging log
  - Building your library

Outline

- Neural Network
- Multilayer Perceptron
- Weighted Filter
- Convolution Neural Network
- EX: MNIST
- Deep Reinforcement Learning
- EX: Atari with DQN
- EX: Turtlebot3 with DQN

Outline

[Problem solving]

Search problems

Adversarial Search

[Perception and Uncertainty]

Bayes Theorem

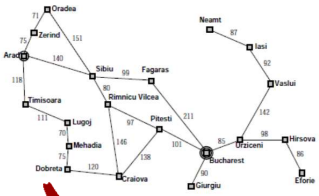
Bayes Filter and Smoothing

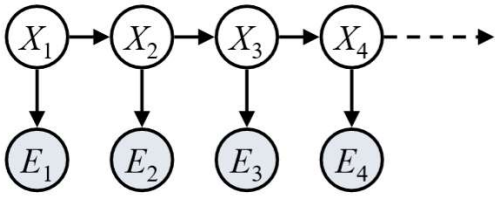
[Learning and Decision-making]

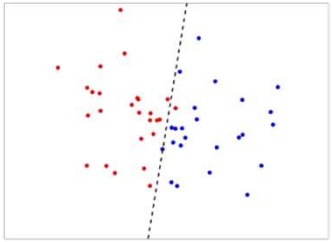
Supervised learning

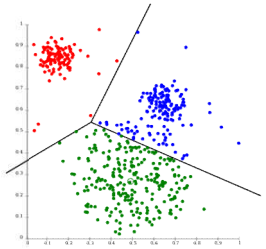
Unsupervised learning

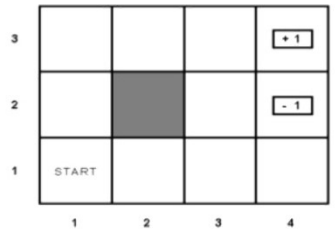
Reinforcement learning



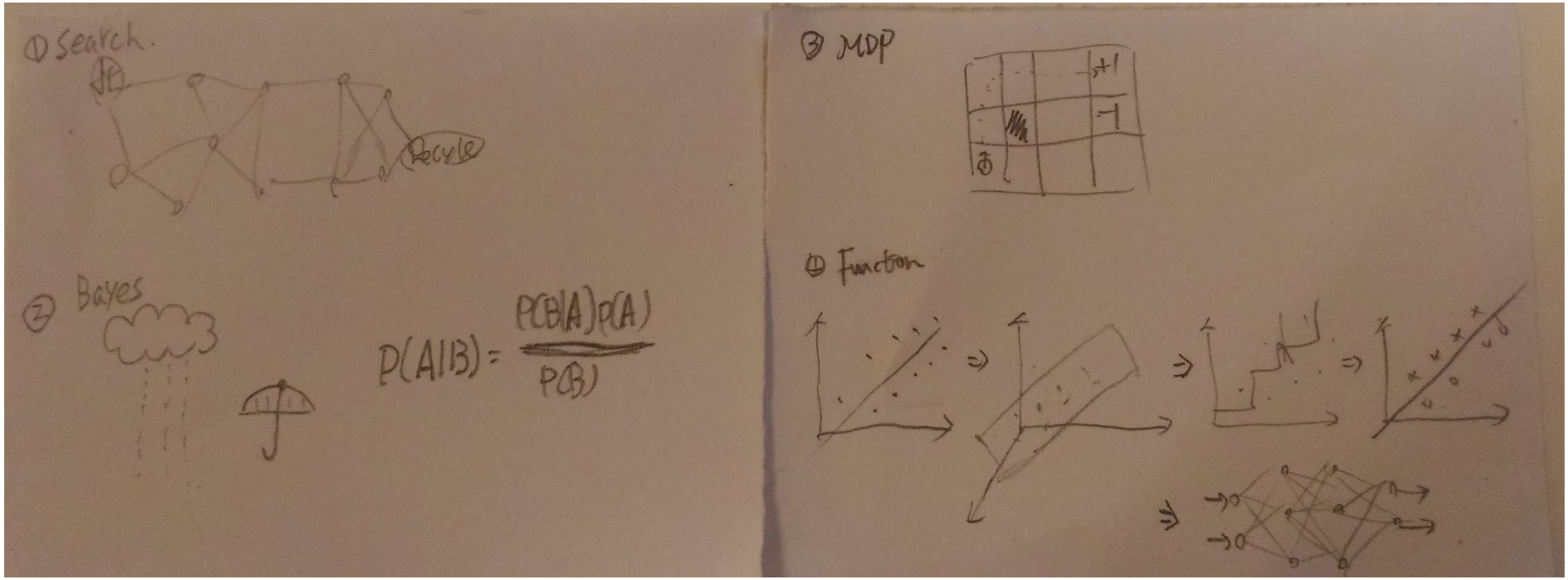








Outline

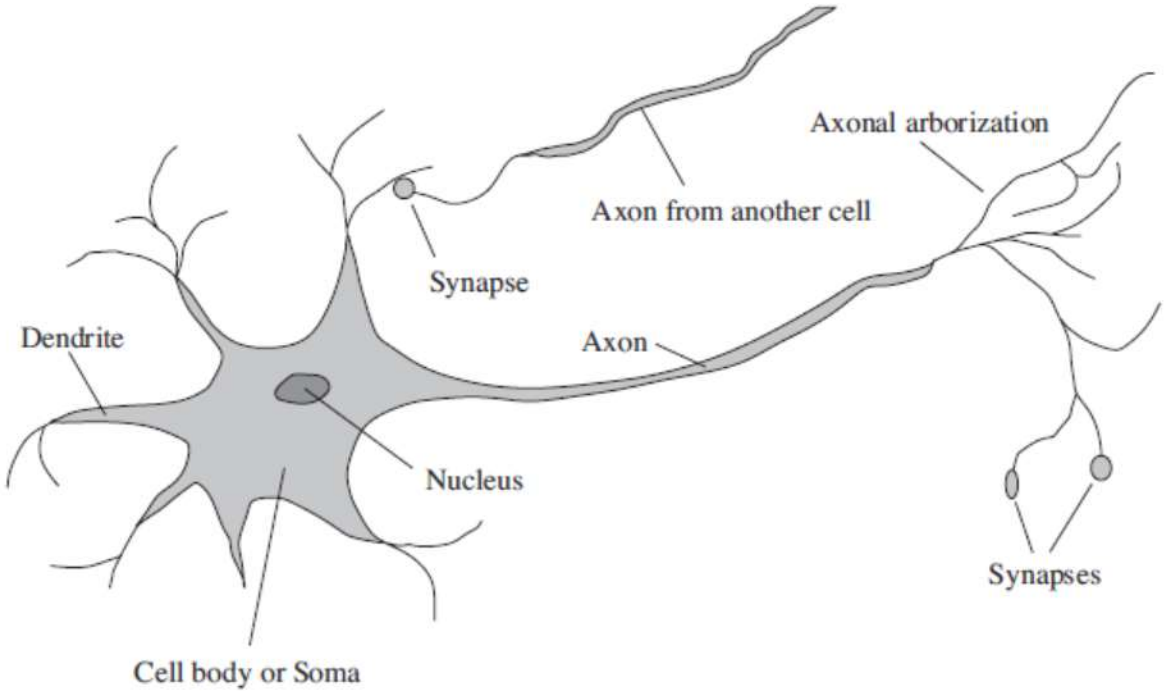


Need

- Belief of Neural Networks
- https://www.youtube.com/watch?v=l9RWTMNnvi4

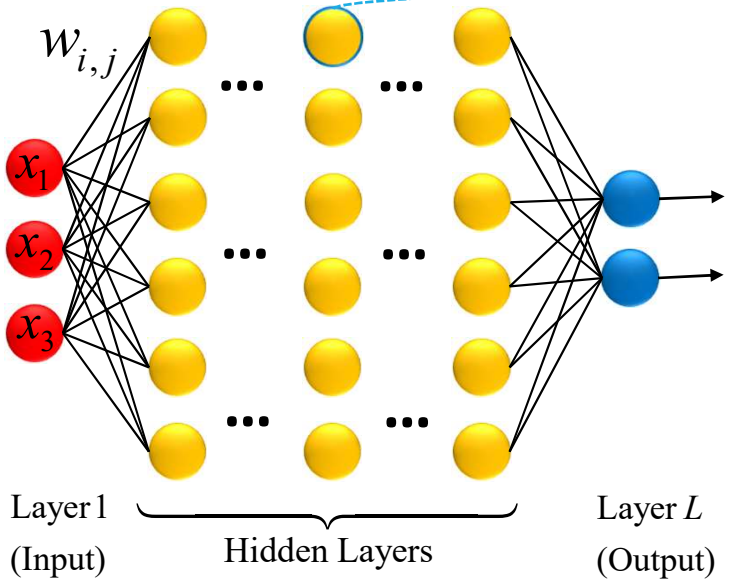
Neural networks

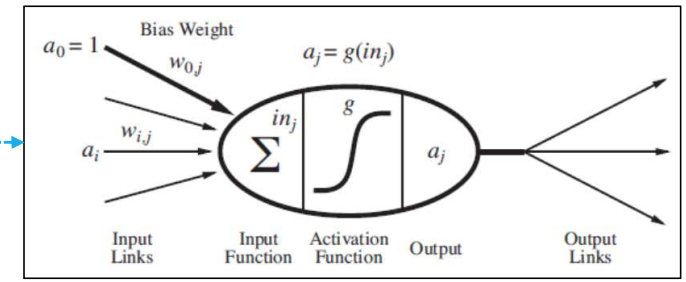
- The concept of neural network (NN) is to simulate human brains' neuron.

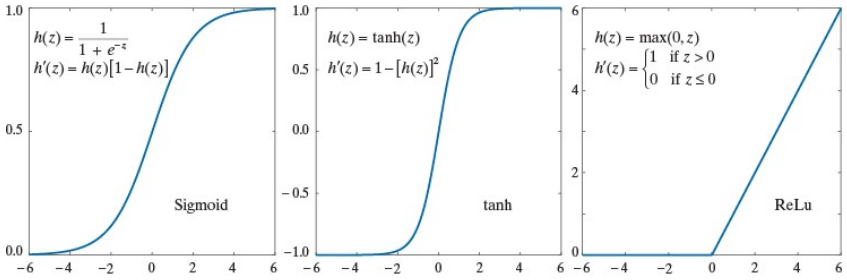


Neural networks

- AI researchers use computers to simulate neuros. Given the input, the NN should automatically tune its weighting for desired output. (Connectionism/bionicsism)

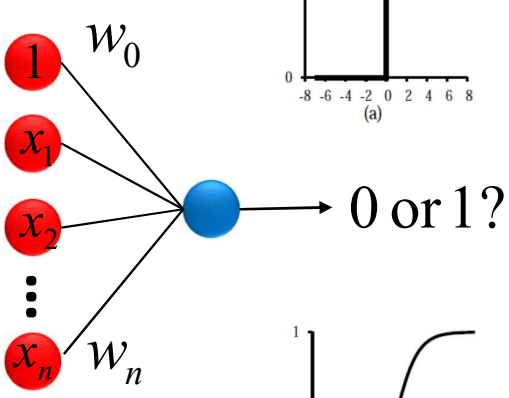




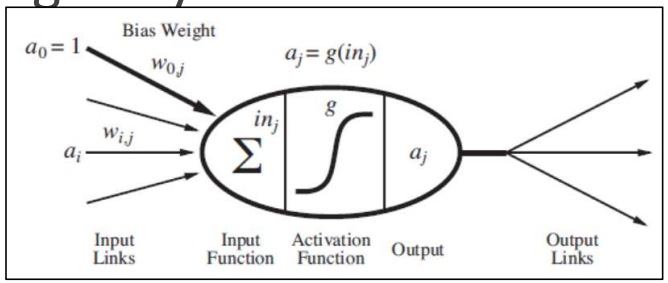


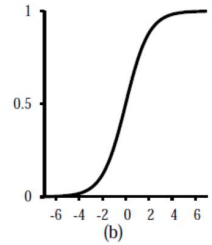
Neural networks

- Recall the special case of NN, a single-layer NN (no hidden layer).
- Perceptron and logistic regression are single-layer NNs.



$$h_W(\mathbf{x}) = \begin{cases} 1 \text{ if } \mathbf{wx} \geq 0 \\ 0 \text{ if } \mathbf{wx} < 0 \end{cases}$$

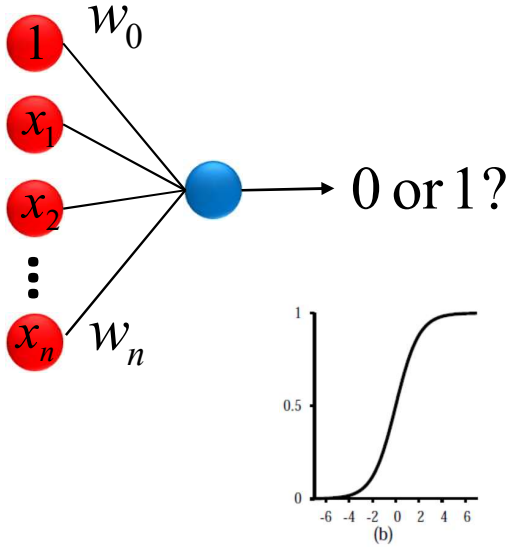




$$h(x) = \text{logistic}(\mathbf{wx}) = \frac{1}{1+e^{-\mathbf{wx}}}$$

Neural networks

- There are two steps of NN.
  - Prediction (forward propagation): Predict the output based on weighting, input data and NN structure.
  - Update (backward propagation): Update the weighting based on the output labels, input data and NN structure.



$$h(x) = \text{logistic}(\mathbf{wx}) = \frac{1}{1+e^{-\mathbf{wx}}}$$

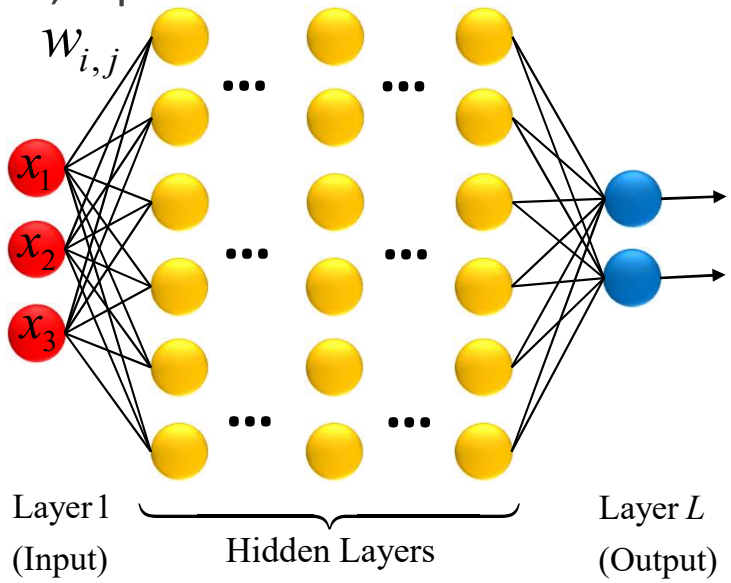
Forward propagation

$$w_j \leftarrow w_j + \alpha(y - h_W(\mathbf{x}))h_W(\mathbf{x})(1 - h_W(\mathbf{x}))x_j$$

Backward propagation

Neural networks

- There are two steps of NN.
  - Prediction (forward propagation): Predict the output based on weighting, input data and NN structure.
  - Update (backward propagation): Update the weighting based on the output labels, input data and NN structure.



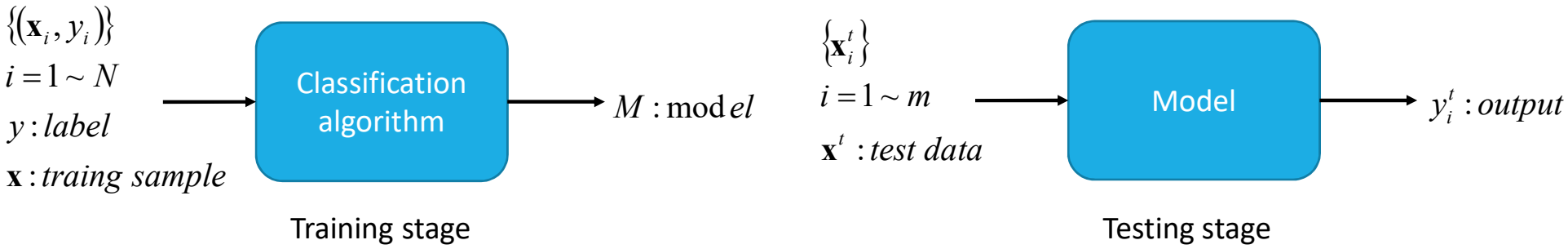
Forward and backward propagations are easy for a single layer NN. How about multilayer NN?

Outline

- Neural Network
- **Multilayer Perceptron**
- Weighted Filter
- Convolution Neural Network
- EX: MNIST
- Deep Reinforcement Learning
- EX: Atari with DQN
- EX: Turtlebot3 with DQN

Multilayer Perceptron (MLP)

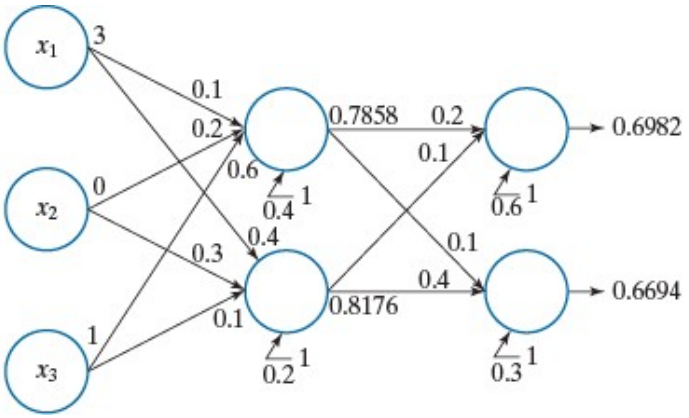
- Prediction (forward propagation): Predict the output based on weighting, input data and NN structure. ➔ Testing
- Update (backward propagation): Update the weighting based on the output labels, input data and NN structure. ➔ Training



Multilayer Perceptron (MLP)

Assume there is a 3 - layer NN (as Fig. shows).
The activation function of each node is logistic function
Find the NN ouput?

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix}$$



$$z_1(2) = W_{1,1}X = (0.1)(3) + (0.2)(0) + (0.6)(1) + (1)(0.4) = 1.3$$

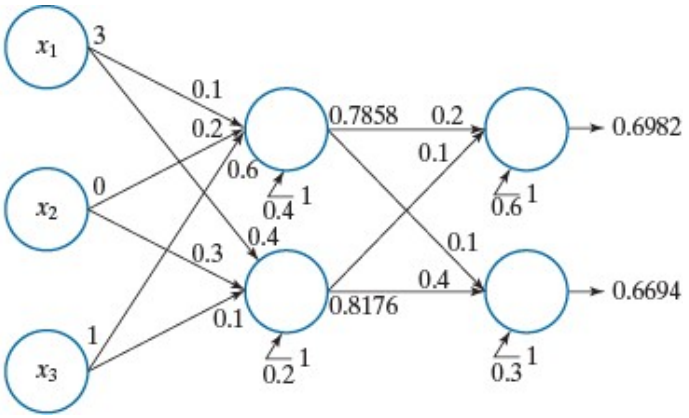
$$a_1(2) = h(z_1(2)) = \frac{1}{1+e^{-1.3}} = 0.7858$$

$$z_2(2) = W_{1,2}X = (0.4)(3) + (0.3)(0) + (0.1)(1) + (1)(0.2) = 1.5$$

$$a_2(2) = h(z_2(2)) = \frac{1}{1+e^{-1.5}} = 0.8176$$

Multilayer Perceptron (MLP)

Assume there is a 3 - layer NN (as Fig. shows).
The activation function of each node is logistic function
Find the NN ouput?



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix}$$

+bias

$$z_1(3) = W_{2,1}A(2) = (0.2)(0.7858) + (0.1)(0.8176) + (0.6)(1) = 0.8389$$

$$a_1(3) = h(z_1(3)) = \frac{1}{1+e^{-0.8389}} = 0.6982$$

$$z_2(3) = W_{2,2}A(2) = (0.1)(0.7858) + (0.4)(0.8176) + (0.3)(1) = 0.7056$$

$$a_2(3) = h(z_2(3)) = \frac{1}{1+e^{-0.7056}} = 0.6694$$

$\because a_1(3) > a_2(3), \therefore$ the data is classified as c_1

Multilayer Perceptron (MLP)

Assume there is a 3 - layer NN (as Fig. shows).
The activation function of each node is logistic function
Find the NN ouput?

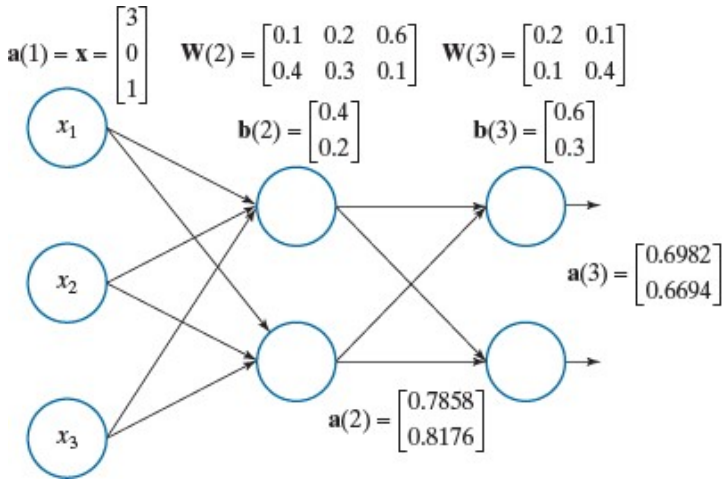
$$a(1) = [3 \quad 0 \quad 1]^T$$

$$z(2) = W(2)a(1) + b(2) = \begin{bmatrix} 0.1 & 0.2 & 0.6 \\ 0.4 & 0.3 & 0.1 \end{bmatrix} \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.4 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 1.3 \\ 1.5 \end{bmatrix}$$

$$a(2) = h[z(2)] = \begin{bmatrix} h(1.3) \\ h(1.5) \end{bmatrix} = \begin{bmatrix} 0.7858 \\ 0.8176 \end{bmatrix}$$

$$z(3) = W(3)a(2) + b(3) = \begin{bmatrix} 0.2 & 0.1 \\ 0.1 & 0.4 \end{bmatrix} \begin{bmatrix} 0.7858 \\ 0.8176 \end{bmatrix} + \begin{bmatrix} 0.6 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 0.8389 \\ 0.7056 \end{bmatrix}$$

$$a(3) = h[z(3)] = \begin{bmatrix} h(0.7858) \\ h(0.8176) \end{bmatrix} = \begin{bmatrix} 0.6982 \\ 0.6694 \end{bmatrix}$$

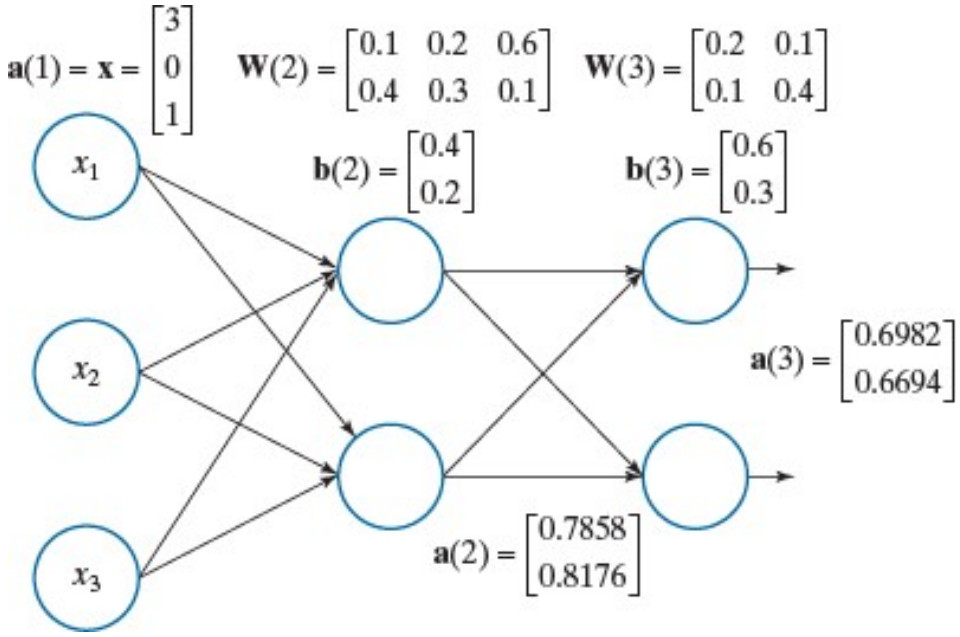


Matrix form

Multilayer Perceptron (MLP)

- Tensorflow code

```
X = tf.Variable([[3.0,0.0,1.0]])
W = tf.Variable([[0.1,0.4 ],
                 [0.2, 0.3 ],
                 [0.6, 0.1 ]])
b = tf.Variable([[0.4,0.2]])

XWb=tf.matmul(X,W)+b
y=tf.nn.sigmoid(tf.matmul(X,W)+b)

with tf.Session() as sess:
    init = tf.global_variables_initializer()
    sess.run(init)
    print('XWb:')
    print(sess.run(XWb))
    print('y:')
    print(sess.run(y ))
```



Multilayer Perceptron (MLP)

- Prediction (forward propagation): Predict the output based on weighting, input data and NN structure.

$[Input]X$: data input, y : label

$\mathbf{w} \leftarrow \mathbf{0}$

$A(1) = X$; // Input layer

for $l = 2{:}L$ //feedforward

$Z(l) = W(l)A(l\text{-}1) + B(l)$;

$A(l) = h(Z(l))$;

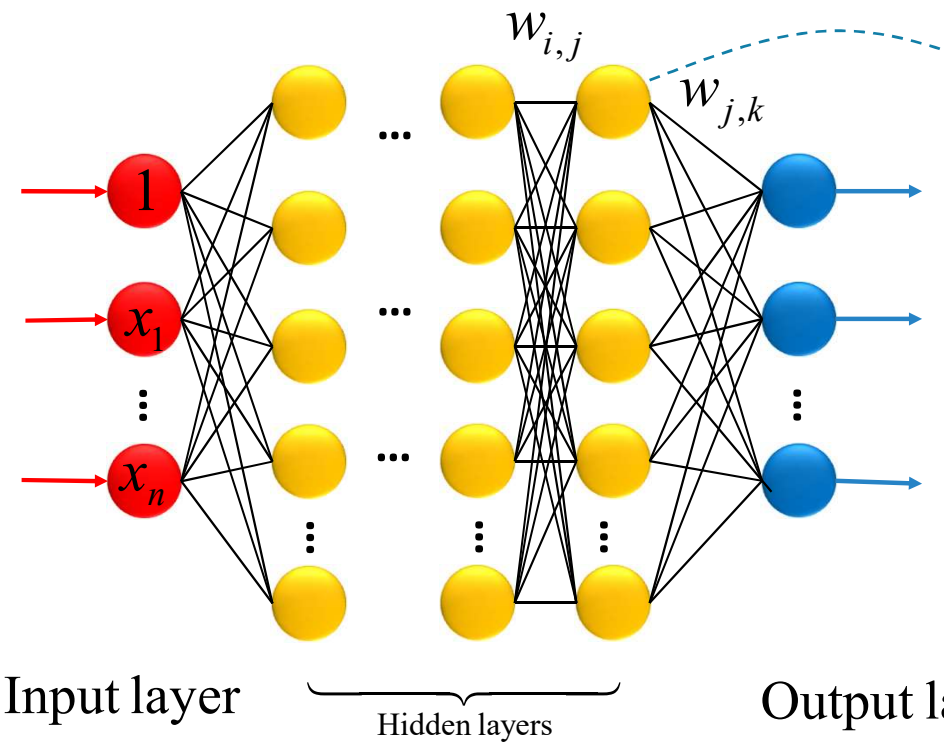
end

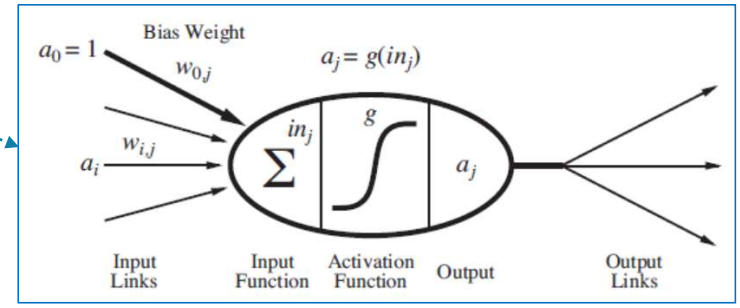
Return A(L)

```
def layer(output_dim,input_dim,inputs, activation=None):
    W = tf.Variable(tf.random_normal([input_dim, output_dim]))
    b = tf.Variable(tf.random_normal([1, output_dim]))
    XWb = tf.matmul(inputs, W) + b
    if activation is None:
        outputs = XWb
    else:
        outputs = activation(XWb)
    return outputs
```

Multilayer Perceptron

- Update (Backward): Update weighting based on input training data.



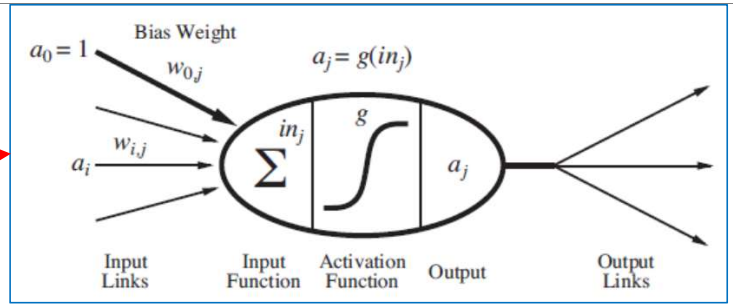


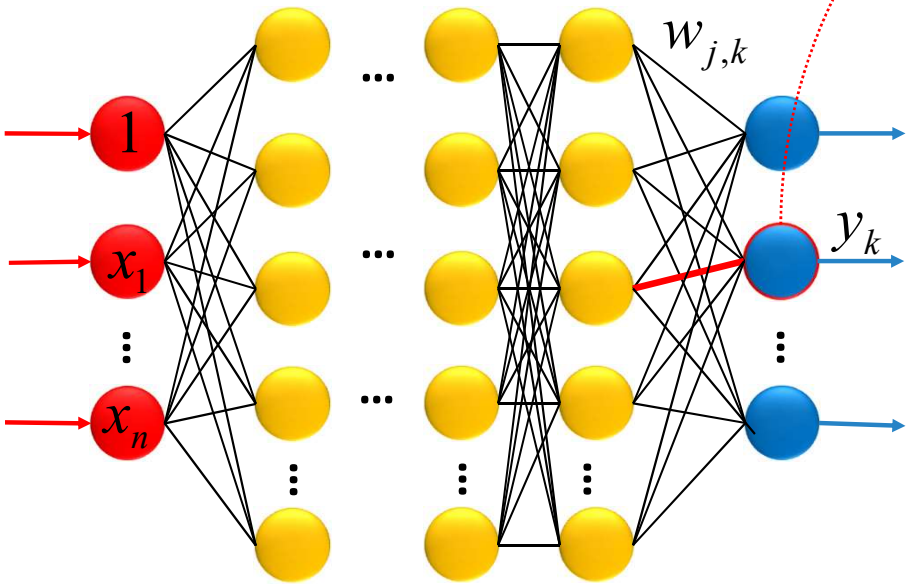
$w_{i,j}$: The weighting of

i^{th} input node to j^{th} output node

Multilayer Perceptron

- j to k node weighting





$$\frac{\partial Loss_k}{\partial w_{j,k}} = -2(y_k - a_k)\frac{\partial a_k}{\partial w_{j,k}} = -2(y_k - a_k)\frac{\partial g(in_k)}{\partial w_{j,k}}$$

$$= -2(y_k - a_k)g'(in_k)\frac{\partial in_k}{\partial w_{j,k}}$$

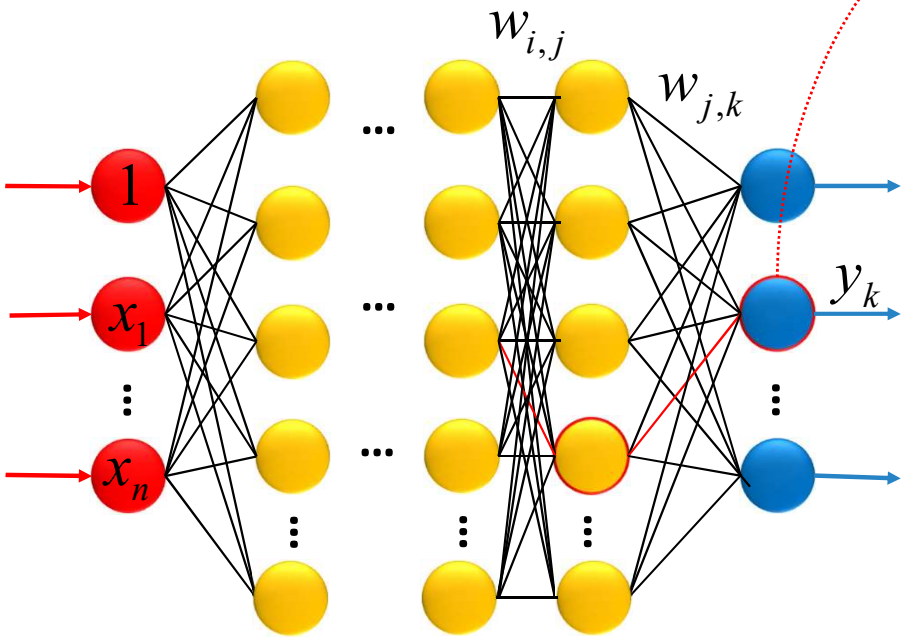
$$= -2(y_k - a_k)g'(in_k)\frac{\partial\left(\sum_j w_{j,k}a_j\right)}{\partial w_{j,k}}$$

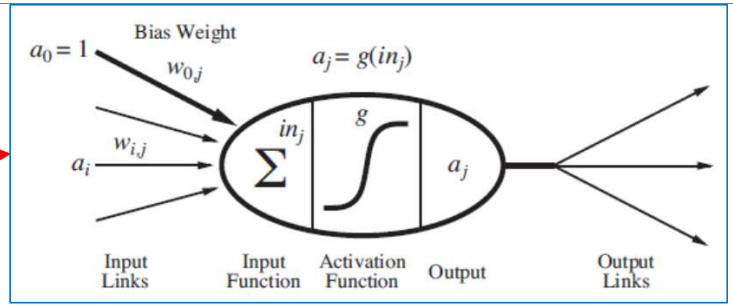
$$= -\underbrace{2(y_k - a_k)g'(in_k)}_{\Delta_k}a_j$$

$$w_{j,k} \leftarrow w_{j,k} + \alpha a_j \Delta_k$$

Multilayer Perceptron

- i to j node weighting





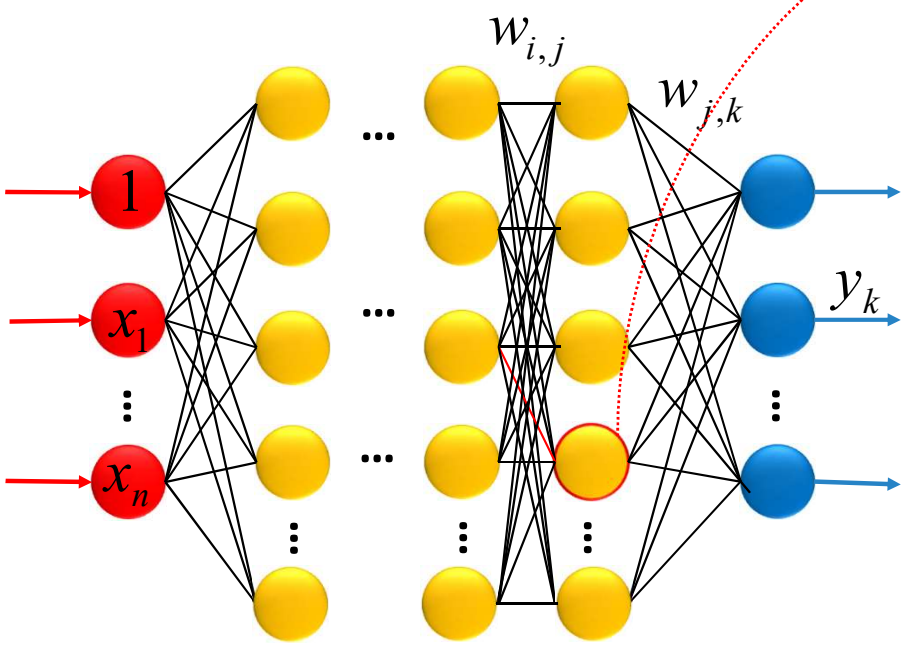
$$\frac{\partial Loss_k}{\partial w_{i,j}} = -2(y_k - a_k)\frac{\partial a_k}{\partial w_{i,j}} = -2(y_k - a_k)\frac{\partial g(in_k)}{\partial w_{i,j}}$$

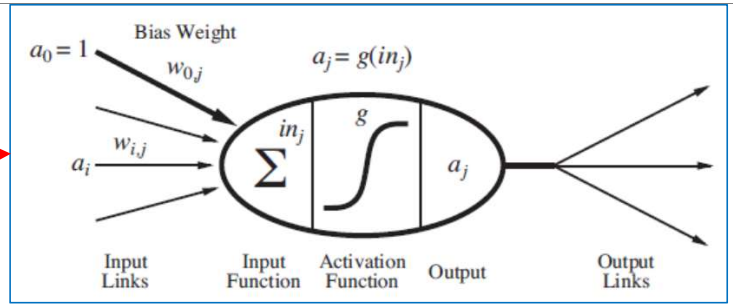
$$= -2(y_k - a_k)g'(in_k)\frac{\partial g(in_k)}{\partial w_{i,j}} = -2\Delta_k \frac{\partial\left(\sum_j w_{j,k}a_j\right)}{\partial w_{i,j}}$$

$$= -2\Delta_k w_{j,k}\frac{\partial a_j}{\partial w_{i,j}} = -2\Delta_k w_{j,k}\frac{\partial g(in_j)}{\partial w_{i,j}}$$

Multilayer Perceptron

- i to j node weighting





$$\frac{\partial Loss_k}{\partial w_{i,j}} = -2\Delta_k w_{j,k} \frac{\partial g(in_j)}{\partial w_{i,j}}$$

$$= -2\Delta_k w_{j,k} g'(in_j) \frac{\partial in_j}{\partial w_{i,j}} = -2\Delta_k w_{j,k} g'(in_j) \frac{\partial \left(\sum_i w_{i,j} a_i \right)}{\partial w_{i,j}}$$

$$= -2\Delta_k w_{j,k} g'(in_j) a_i = -a_i \Delta_j$$

$$w_{i,j} \leftarrow w_{i,j} + \alpha a_i \Delta_j$$

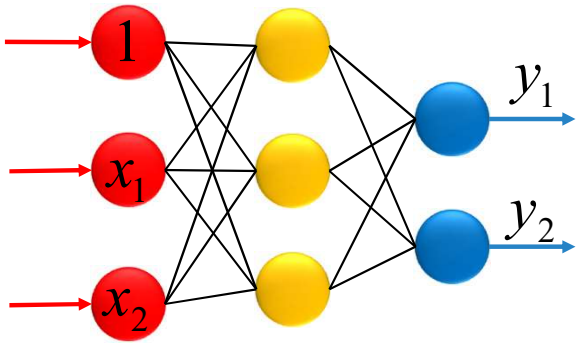
Multilayer Perceptron

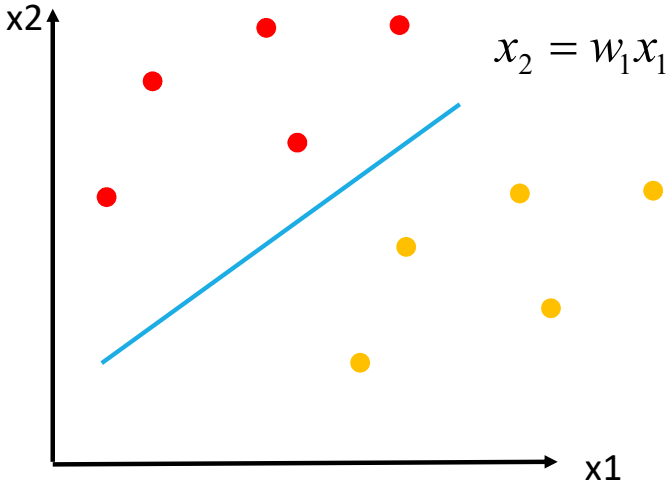
- Illustration of MLP (linear → logistic regression → MLP)
  - Nonlinear functions
  - Feature extraction

$$y = \mathbf{x}^T \mathbf{w}$$

$$w_2 x_2 + w_1 x_1 + w_0 = 0$$

$$x_2 + \frac{w_1}{w_2} x_1 + \frac{w_0}{w_2} > 0 \rightarrow class1$$



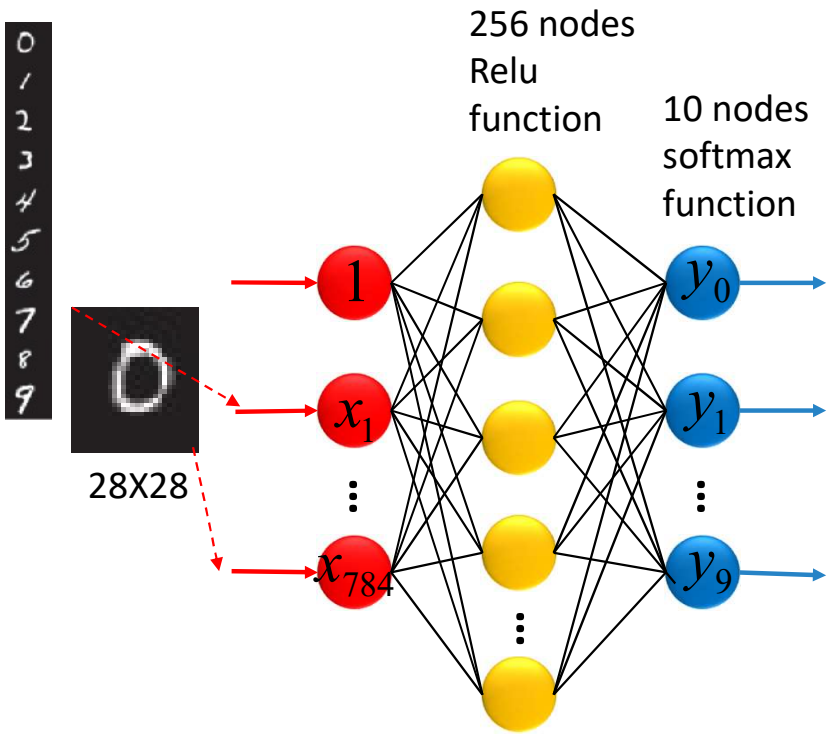


```
function BACK-PROP-LEARNING(examples, network) returns a neural network
  inputs: examples, a set of examples, each with input vector x and output vector y
          network, a multilayer network with L layers, weights w_{i,j}, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  for each weight w_{i,j} in network do
      w_{i,j} ← a small random number

  repeat
      for each example (x, y) in examples do
          /* Propagate the inputs forward to compute the outputs */
          for each node i in the input layer do
              a_i ← x_i
          for ℓ = 2 to L do
              for each node j in layer ℓ do
                  in_j ← Σ_i w_{i,j} a_i
                  a_j ← g(in_j)
          /* Propagate deltas backward from output layer to input layer */
          for each node j in the output layer do
              Δ[j] ← g'(in_j) × (y_j − a_j)
          for ℓ = L − 1 to 1 do
              for each node i in layer ℓ do
                  Δ[i] ← g'(in_i) Σ_j w_{i,j} Δ[j]
          /* Update every weight in network using deltas */
          for each weight w_{i,j} in network do
              w_{i,j} ← w_{i,j} + α × a_i × Δ[j]
  until some stopping criterion is satisfied
  return network
```

It's easy to implement it via Tensorflow!

MLP for MNIST

- MLP for MINST



http://yann.lecun.com/exdb/mnist/

According to y0~y9, MLP predicts the output.

The final accuracy is 94%

However, it doesn't make sense to take each pixel and each weight of MLP for image classification
It takes too much memory and computation!

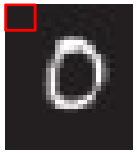
We should use meaningful image features instead of raw data.

Outline

- Neural Network
- Multilayer Perceptron
- **Weighted Filter**
- Convolution Neural Network
- EX: MNIST
- Deep Reinforcement Learning
- EX: Atari with DQN
- EX: Turtlebot3 with DQN

Weighted Filter

- In image processing, researchers adopt some filters to extract features (e.g., edge), which are useful for detection.



z1	z2	z3
z4	z5	z6
z7	z8	z9

✖

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

$$y = \sum_i z_i w_i$$

−1	−2	−1
0	0	0
1	2	1

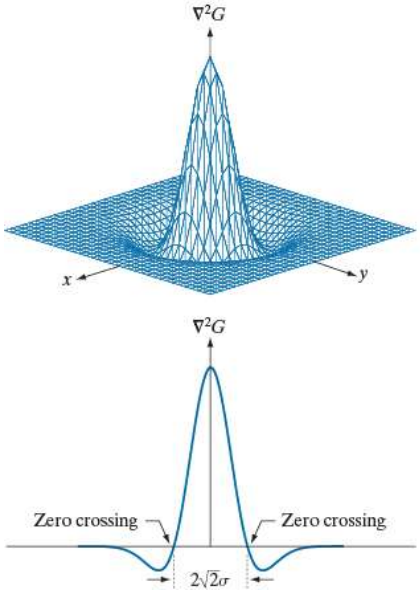
−1	0	1
−2	0	2
−1	0	1

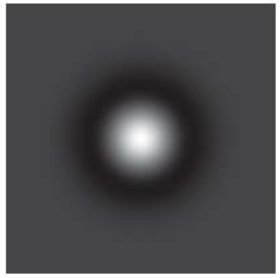
Sobel

Sobel is one kind of filters for edge detection.

Weighted Filter

- Gaussian filter is also popular for edge detection. It has parameters for different effects.

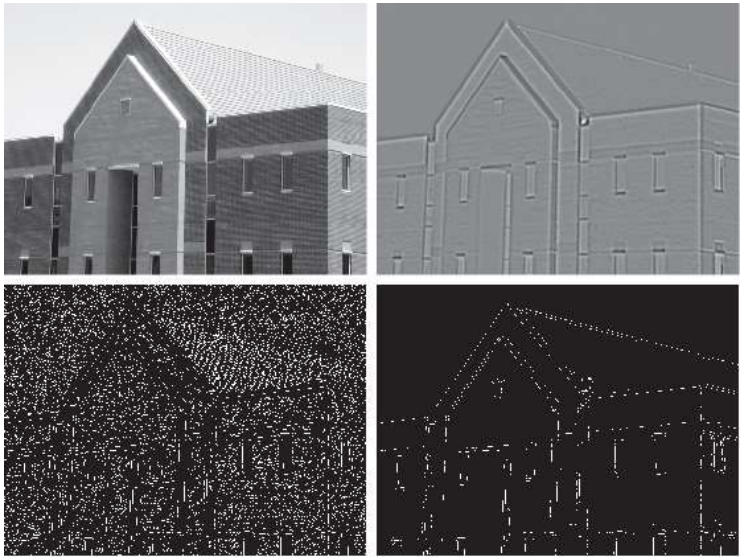




0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

a b
c d

FIGURE 10.22
(a) Image of size 834 × 1114 pixels, with intensity values scaled to the range [0, 1]. (b) Result of Steps 1 and 2 of the Marr-Hildreth algorithm using $\sigma = 4$ and $n = 25$. (c) Zero crossings of (b) using a threshold of 0 (note the closed-loop edges). (d) Zero crossings found using a threshold equal to 4% of the maximum value of the image in (b). Note the thin edges.



How about learning the weighing of filters from training data via neural networks?

Outline

- Neural Network
- Multilayer Perceptron
- Weighted Filter
- **Convolution Neural Network**
- EX: MNIST
- Deep Reinforcement Learning
- EX: Atari with DQN
- EX: Turtlebot3 with DQN

Convolution Neural Network

- Convolution Neural Network (CNN) was proposed by Prof. Yann LeCun. The major concept is to learn the weighting of filters from data.

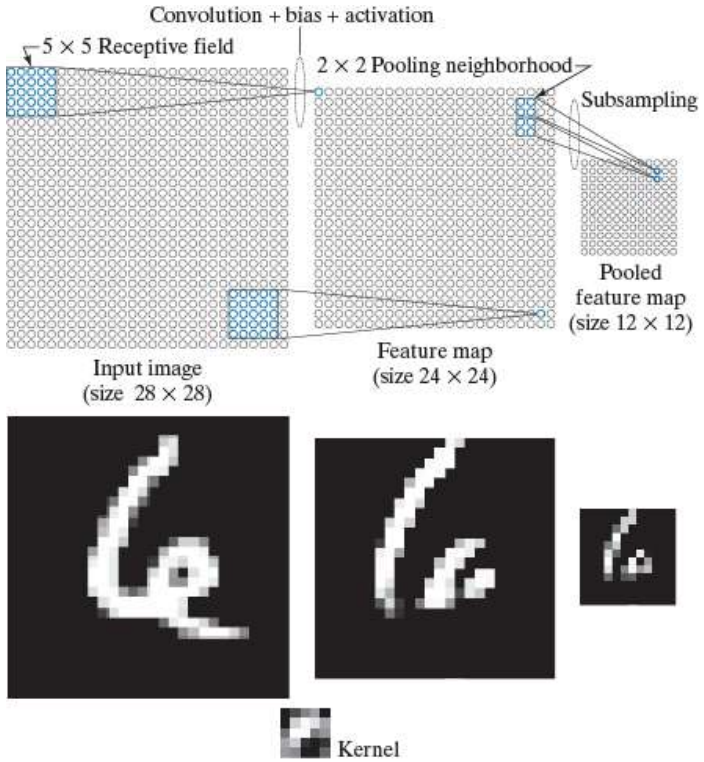
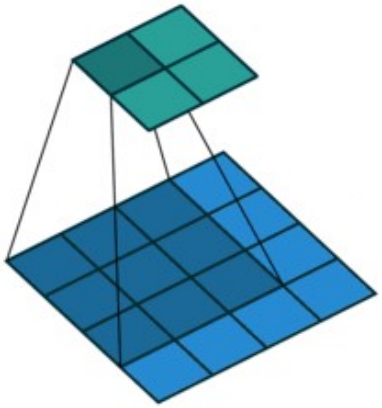


FIGURE 12.41
Top row: How the sizes of receptive fields and pooling neighborhoods affect the sizes of feature maps and pooled feature maps.
Bottom row: An image example. This figure is explained in more detail in Example 12.17. (Image courtesy of NIST.)

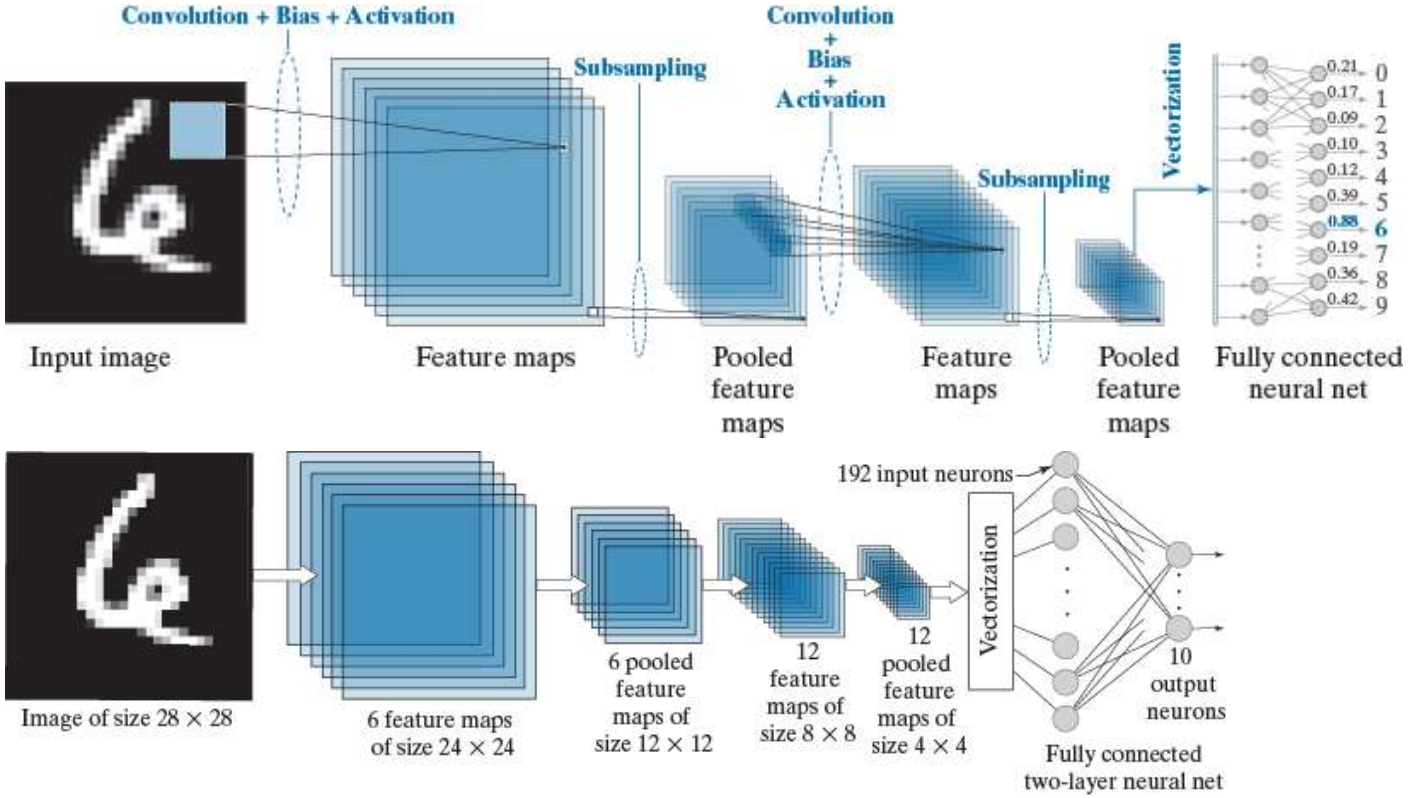
Convolution



Credit from Hsuan-Chi Chang
Ref: https://github.com/vdumoulin/conv_arithmetic

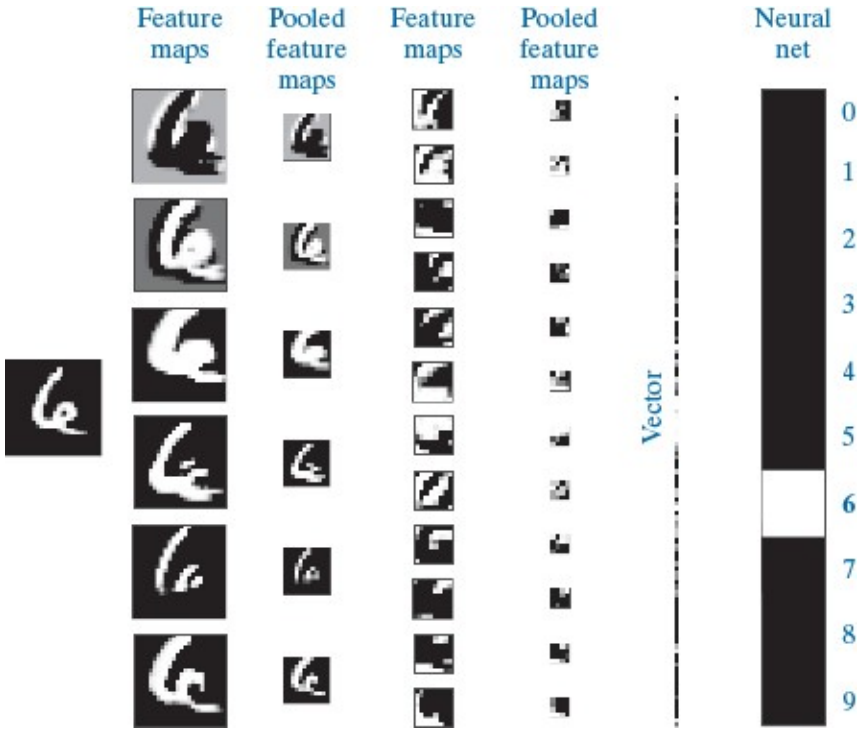
Convolution Neural Network

- CNN for MNIST



Convolution Neural Network

- The images for each layers



The final accuracy is ~99%

However, it still takes a lot of computation. Hence, CNN usually needs GPUs to speed up the training stages (backpropogation).

Outline

- Neural Network
- Multilayer Perceptron
- Weighted Filter
- Convolution Neural Network
- EX: MNIST
- **Deep Reinforcement Learning**
- EX: Atari with DQN
- EX: Turtlebot3 with DQN

Deep Reinforcement Learning

- Since the successes in deep learning of image classification, researcher in RL adopted NN to learn Q functions, so called deep Q-network (DQN).
- In Q-learning, the robot tries to learn Q-functions. Based on Q-functions, the robot can find the optimal decisions.
- We mentioned two approaches for learning Q-functions.
  - Tubular Q-learning
  - Function based Q-learning
- DQN is to learn Q-function via MLP (nonparametric approximation).

Deep Reinforcement Learning

- Why DRL is the future of AI?

1. DL is a nonlinear function approximation
2. DL is a nonparametric approximation (no hand crafting any more!)
3. DL can deal with raw data input (e.g., images or voices)
4. DL can output any functions (e.g., +1/-1 or continuous values)
5. DL is an end-to-end solution for RL

Deep Reinforcement Learning

- Q-learning
- The formulation of Q-learning is the same. The key is to adopt NN for function approximation

$$Q(s,a) = \sum_i w_i f_i(s,a)$$

w : weighting vector

f : features

Linear regression

$$Q(s,a,W) = h(WX)$$

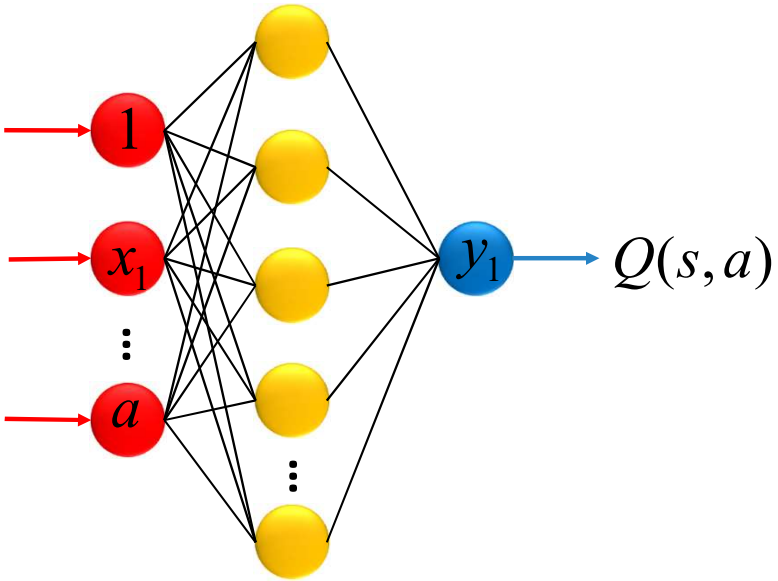
W : weighting vector

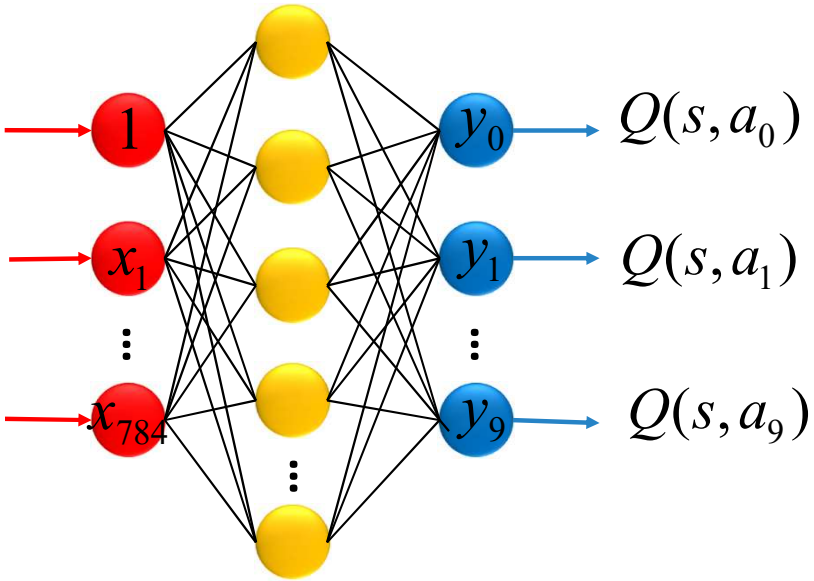
h : activation function

Nonlinear regression

Deep Reinforcement Learning

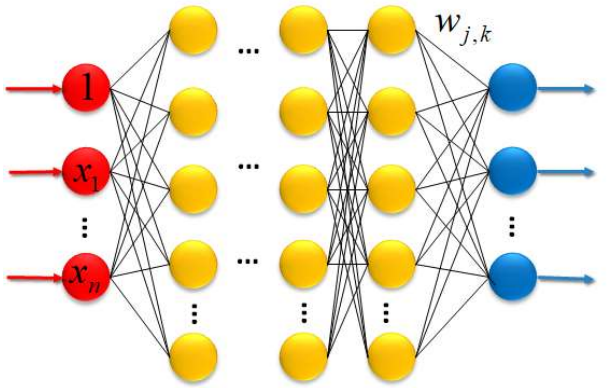
- Q-learning
- Which NN structure is better for Q-learning?





Deep Reinforcement Learning

- What you need to do?
  - Define the state and action
  - Assign reward
  - Design a NN structure



$[GIVEN]$ $[Find]$

$S : state$ $Q(s,a) : \text{action} - \text{utility function (Q function)}$

$A : action$

$R(s,a) : reward$

$\gamma : discount$

$\alpha : \text{learning rate}$

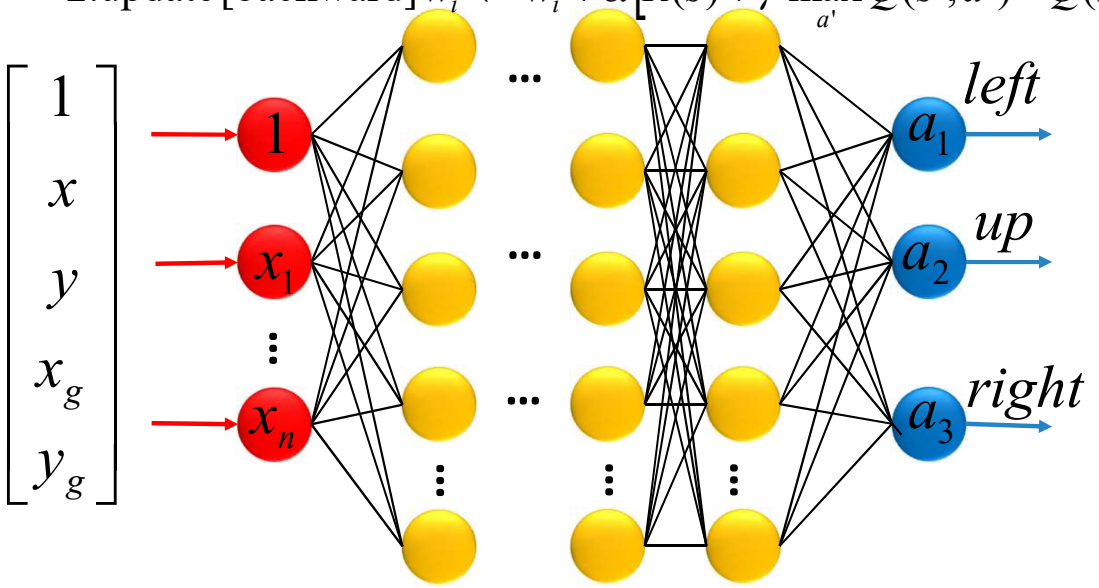
Deep Reinforcement Learning

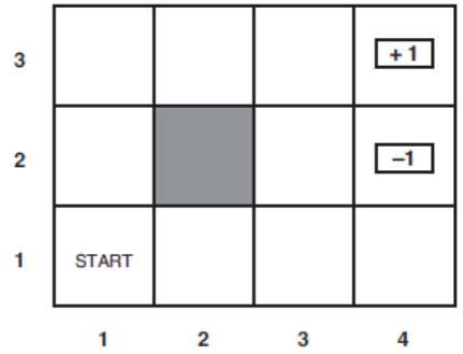
- What you need to do?

1. run action based on $Q(s,a,W)$ [forward]or exploration

and then get (s_i, a_i, s_i', r_i)

2.update [backward] $w_i \leftarrow w_i + \alpha\left[R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]\frac{\partial Q(s,a)}{\partial w_i}$





However, this approach is not stable!

1. Correlations between samples
2. The estimated Q is changing

To solve these issues, DeepMind proposed "experience replay."

Deep Reinforcement Learning

- What you need to do?

1. update MLP parameters : $W' \leftarrow W$

2. run N actions based on $Q(s,a,W)$ [forward]or exploration

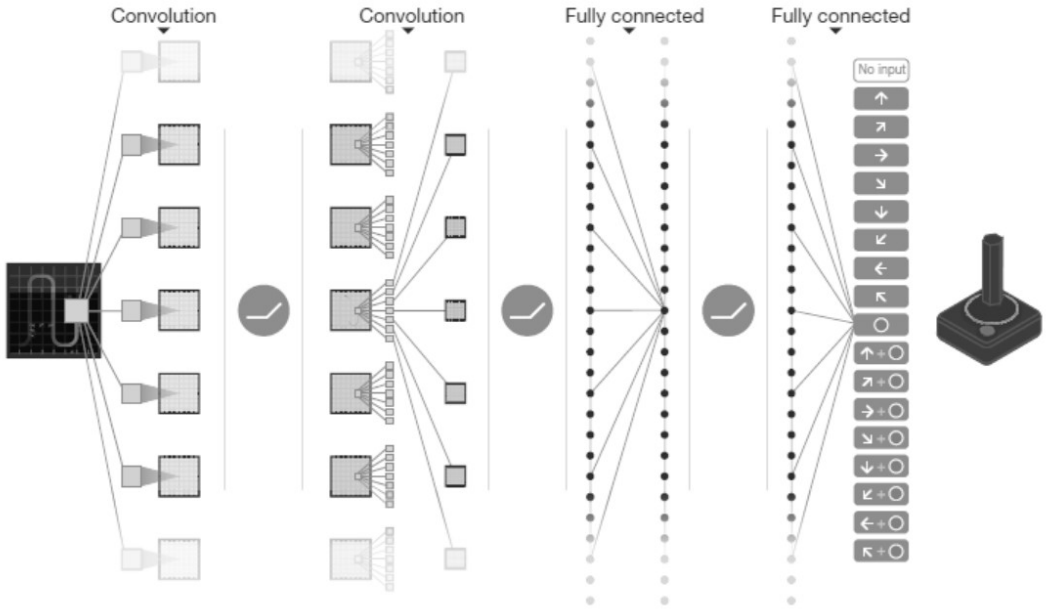
and then get (s_i, a_i, s_i', r_i). Add these data to replay buffer(B).

3. sample K (s_j, a_j, s_j', r_j) from B uniformly.

4.update [backward] $w_i \leftarrow w_i + \alpha \left[R(s) + \gamma \max_{a'} Q(s', a', W') - Q(s, a, W) \right] \frac{\partial Q(s, a, W)}{\partial w_i}$

DQN for Atari

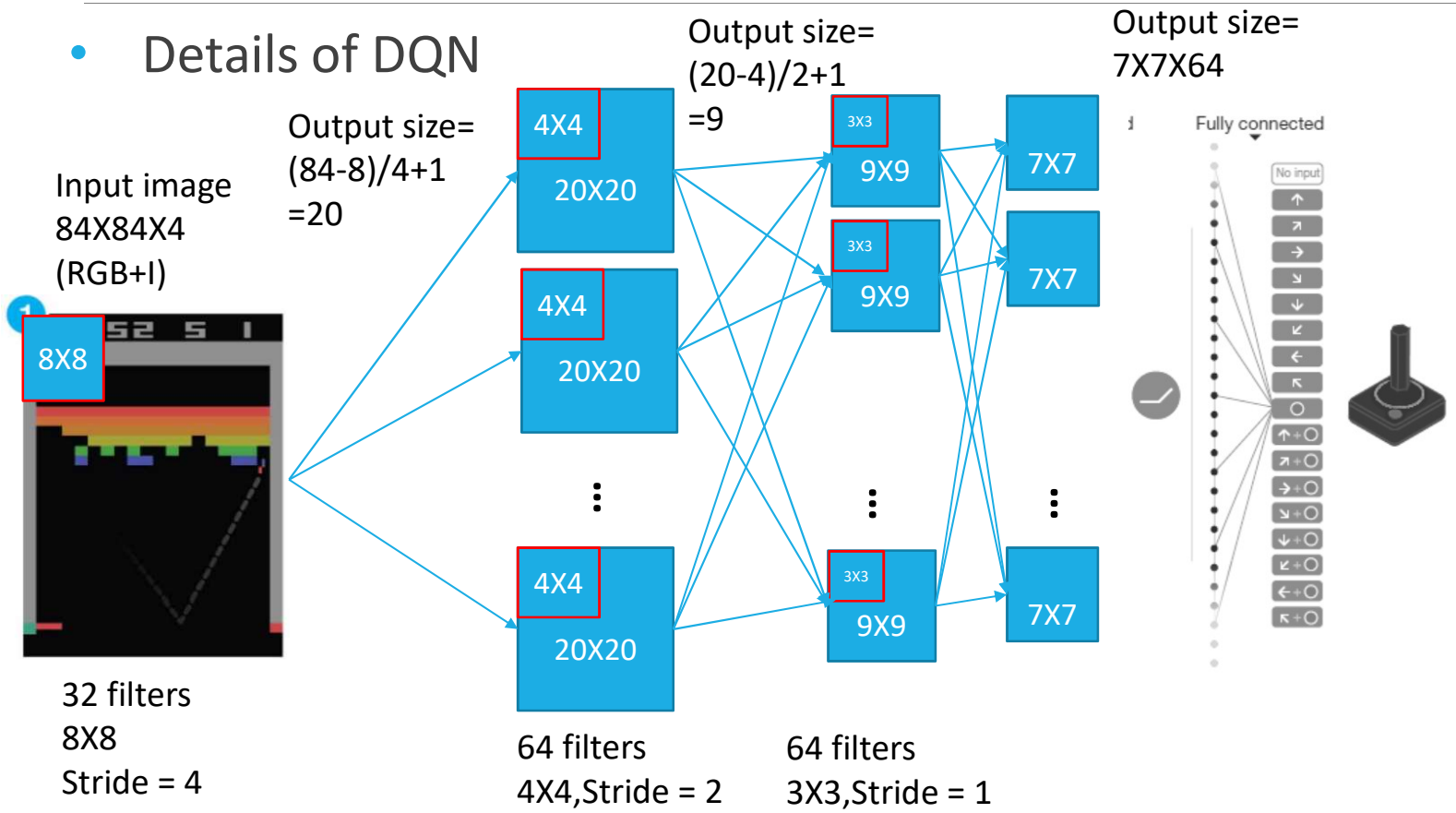
- DeepMind proposed a DQN for playing 48 kinds of Atari games. The robot's performance is better than human players in 29 games!



Mnih et al., "Human-level control through deep reinforcement learning," Nature, 2015.

DQN for Atari

- Details of DQN

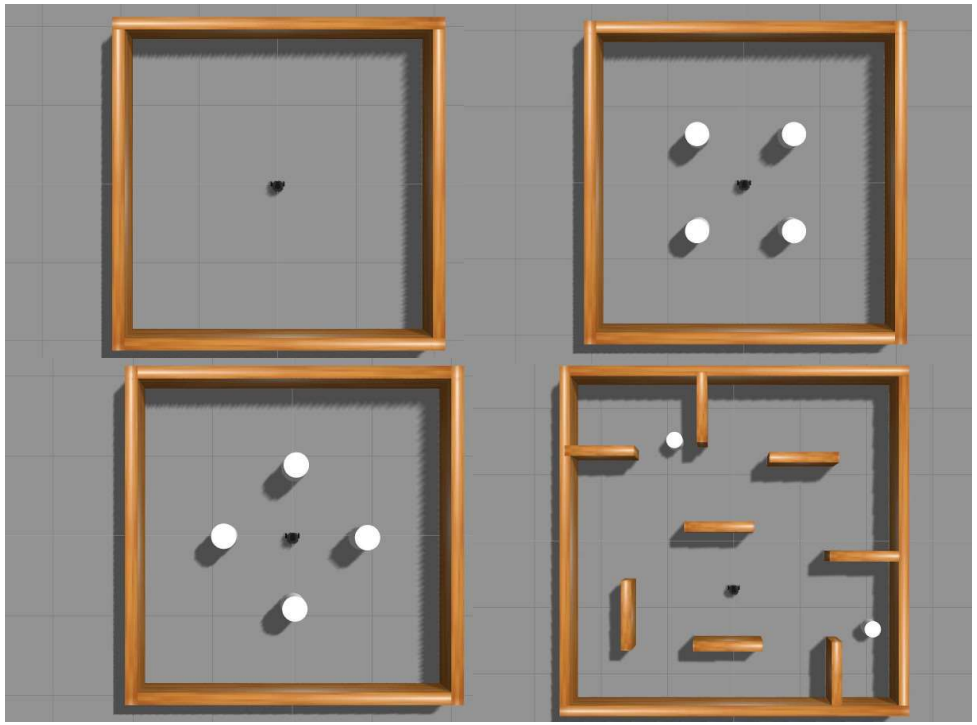


Video

https://www.slideshare.net/onghaoyi/distributed-deep-qlearning

Turtlebot3 with DQN

- Goal: Learn to arrive destination without hitting anything.



Example from https://github.com/ROBOTIS-GIT/turtlebot3_machine_learning.git

Turtlebot3 with DQN

- Goal: Learn to arrive destination without hitting anything.
- Sensors Input: The robot has a laser distance sensor (LDS) and encoder, which gets robot position.
- Actuator output: The robot can move via rotating two wheels.
- Let's define the MDP and then design the MLP.

$[GIVEN]$

$S : state$

$A : action$

$R(s,a) : reward$

$\gamma : discount$

$\alpha : \text{learning rate}$

$[Find]$

$Q(s,a) : \text{action} - \text{utility function (Q function)}$

Turtlebot3 with DQN

- Let's define the MDP and then design the MLP.

$[GIVEN]$

$S : state$

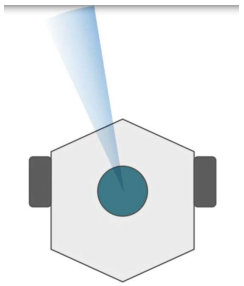
$A : action$

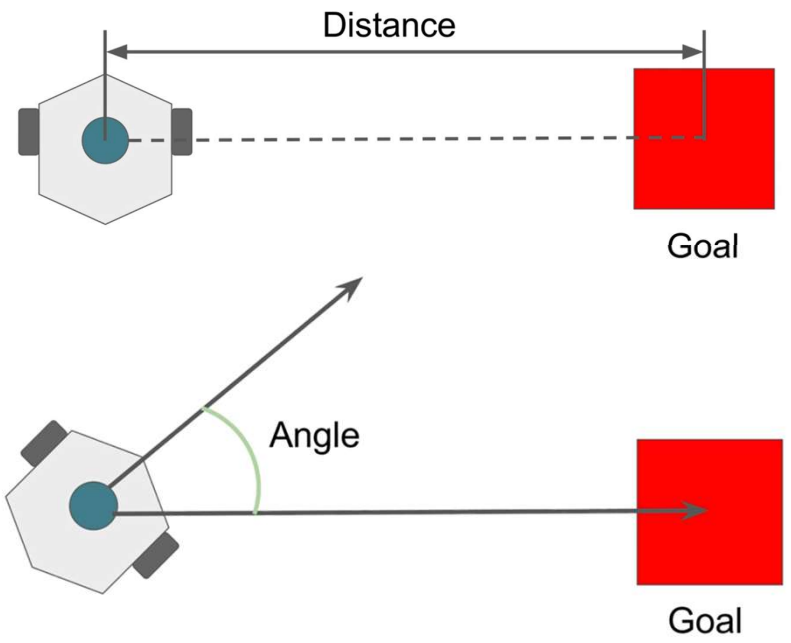
$R(s,a) : reward$

$\gamma : discount$

$\alpha : \text{learning rate}$

$$S = \begin{bmatrix} \text{LDS}_{(24)} \\ \text{Distance} \\ \text{Angle} \end{bmatrix}$$





Turtlebot3 with DQN

- Let's define the MDP and then design the MLP.

$[GIVEN]$

$S : state$

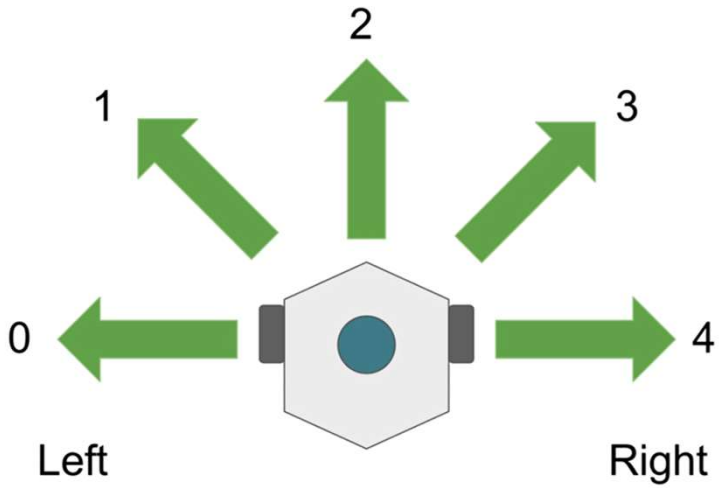
$A : action$

$R(s,a) : reward$

$\gamma : discount$

$\alpha : \text{learning rate}$

$A = \{0,1,2,3,4\}$



0.15 m/s of linear velocity

Action	Angular velocity(rad/s)
0	-1.5
1	-0.75
2	0
3	0.75
4	1.5

Turtlebot3 with DQN

- Let's define the MDP and then design the MLP.

[*GIVEN*]

$S : state$

$A : action$

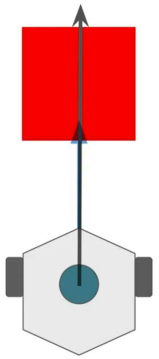
$R(s,a) : reward$

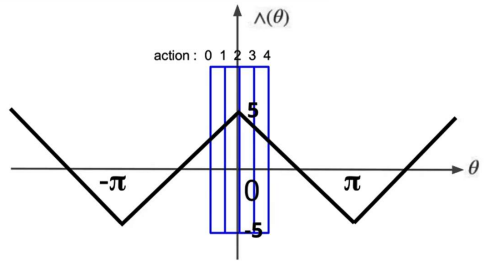
$\gamma : discount$

$\alpha : \text{learning rate}$

$$
\begin{aligned}
&\text{if } -\frac{1}{2}\pi < \theta < \frac{1}{2}\pi, \\
&\quad R_\theta \geq 0 \\
&\text{else} \\
&\quad R_\theta < 0 \\
&\\
&\text{if } D_c < D_g, \\
&\quad R_d > 2 \\
&\text{else} \\
&\quad 1 < R_d \leq 2 \\
&\\
&R = R_\theta \cdot R_d
\end{aligned}
$$

θ : Angle from TurtleBot3 to Goal
D_c : Current distance from Goal
D_g : Absolute distance from Goal
R_θ : Reward from θ
R_d : Reward from distance
R : Reward





$\theta = \frac{\pi}{2} + action \cdot \frac{\pi}{8} + \phi$ $\quad \phi$: Yaw of Turtlebot3

$R_\theta = 5 \cdot (1 - \wedge(\theta))$ $\quad \theta$: Angle from goal

R_θ : Reward from angle

Turtlebot3 with DQN

- Let's define the MDP and then design the MLP.

$[GIVEN]$

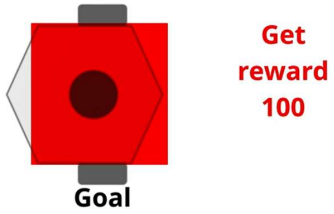
$S : state$

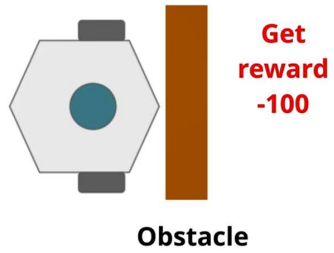
$A : action$

$R(s,a) : reward$

$\gamma : discount$

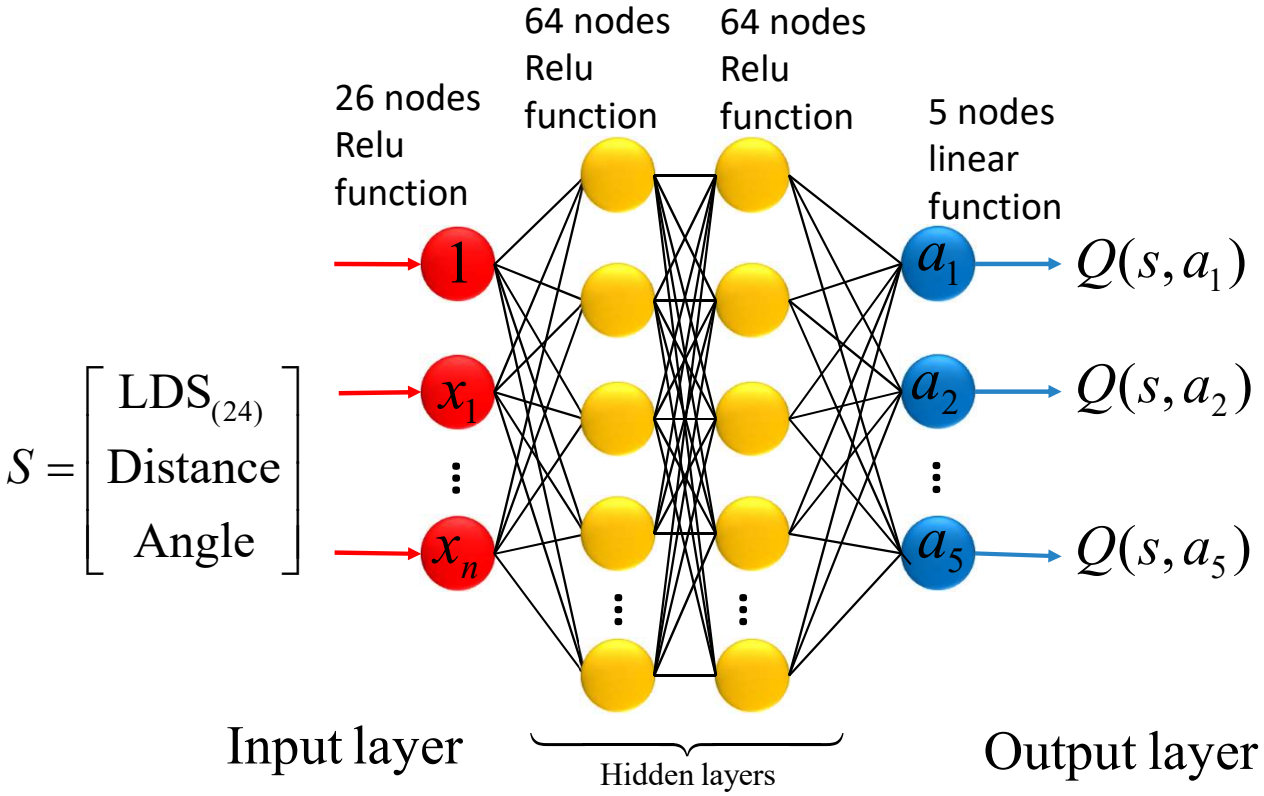
$\alpha : \text{learning rate}$





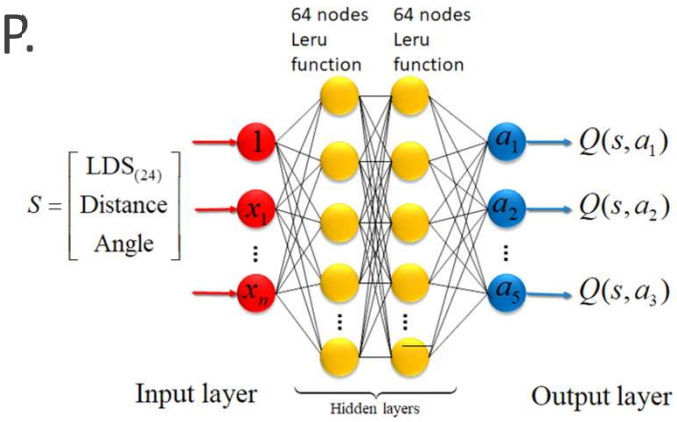
Turtlebot3 with DQN

- Let's define the MDP and then design the MLP.



Turtlebot3 with DQN

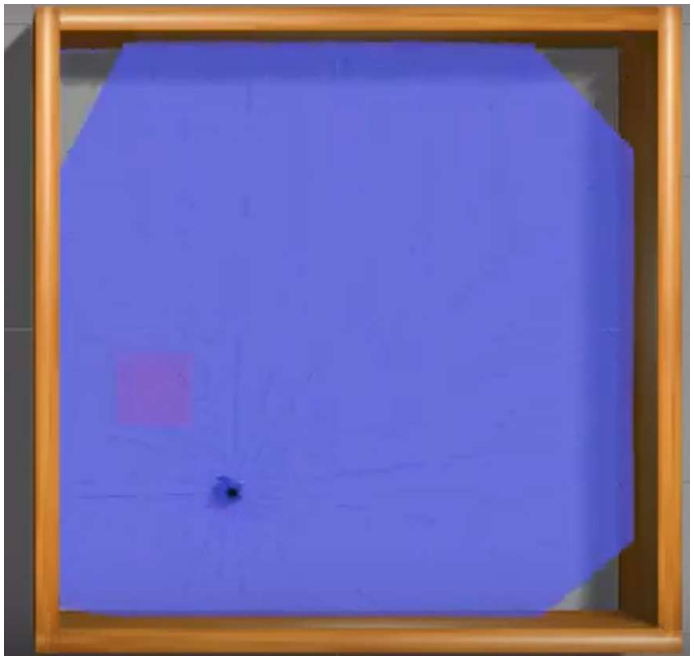
- Let's define the MDP and then design the MLP.




```
model.add(Dense(64, input_shape=(self.state_size,), activation='relu', kernel_initializer='lecun_uniform'))
model.add(Dense(64, activation='relu', kernel_initializer='lecun_uniform'))
model.add(Dropout(dropout))
model.add(Dense(self.action_size, kernel_initializer='lecun_uniform'))
model.add(Activation('linear'))
model.compile(loss='mse', optimizer=RMSprop(lr=self.learning_rate, rho=0.9, epsilon=1e-06))
```

Turtlebot3 with DQN

- Experiments



https://www.youtube.com/watch?v=5uIZU8PCHT8&feature=youtu.be

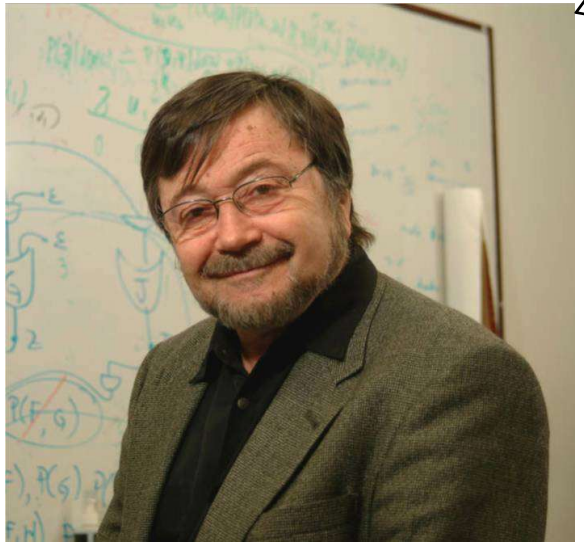
Conclusion

- DL and DRL
- ☺:
  - Image pattern input
  - End-to-end approaches
- ☹:
  - Convergence and optimality (multiple layers → non convex)
  - Overfitting (too many parameters. it can be alleviated by L1 and dropout)
  - Large training samples
- Some researchers claim that GP is better than DL.
- DL can approximate GP [1]

[1] Greg Yang, "Tensor Programs I: Wide Feedforward or Recurrent Neural Networks of Any Architecture are Gaussian Processes," NIPS, 2019.

Conclusion

All the impressive achievements of deep learning amount to just curve fitting.



Judea Pearl, Turing Award winner (invention of Bayesian networks)

CSAIL - MIT ·
Ladies, if he:
- requires lots of supervision
- yet always wants more power
- can't explain decisions - optimizes for the average outcome
- dismisses problems as edge cases
- forgets things catastrophically

He's not your man, he's a deep neural network.
credit: Alex J. Champandard

Conclusion

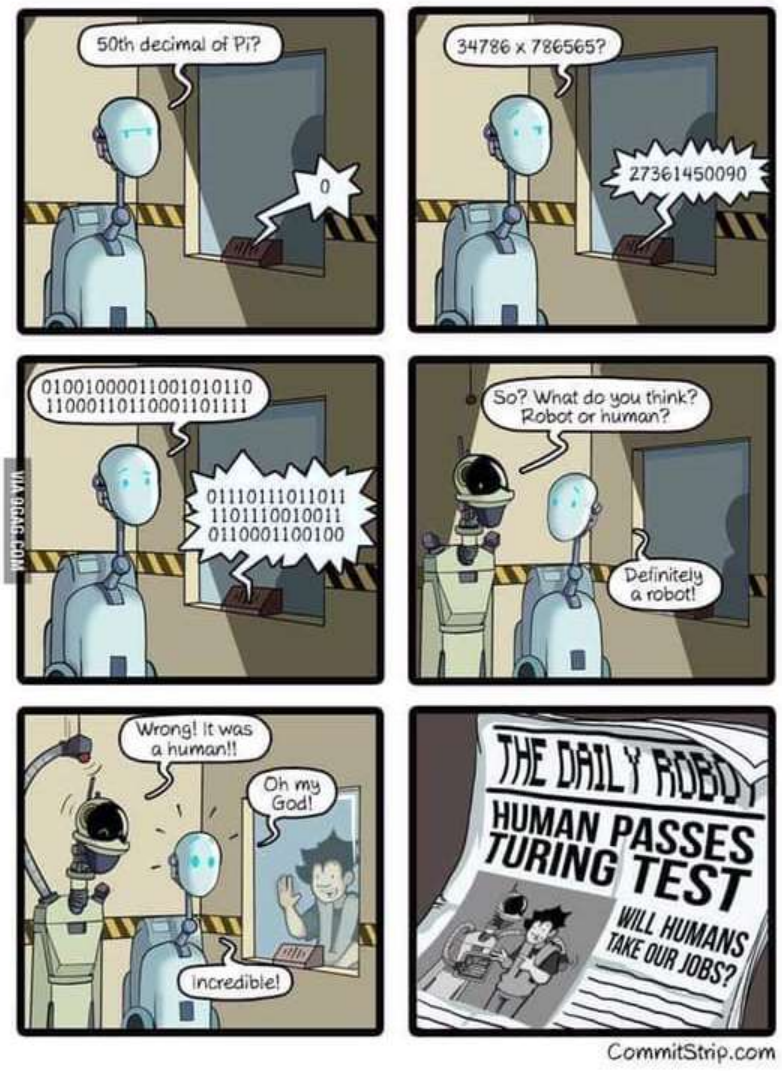
- It’s difficult for DL to play these games
- These games are not just action games! The deep neural networks cannot feel horrible or have nightmares.

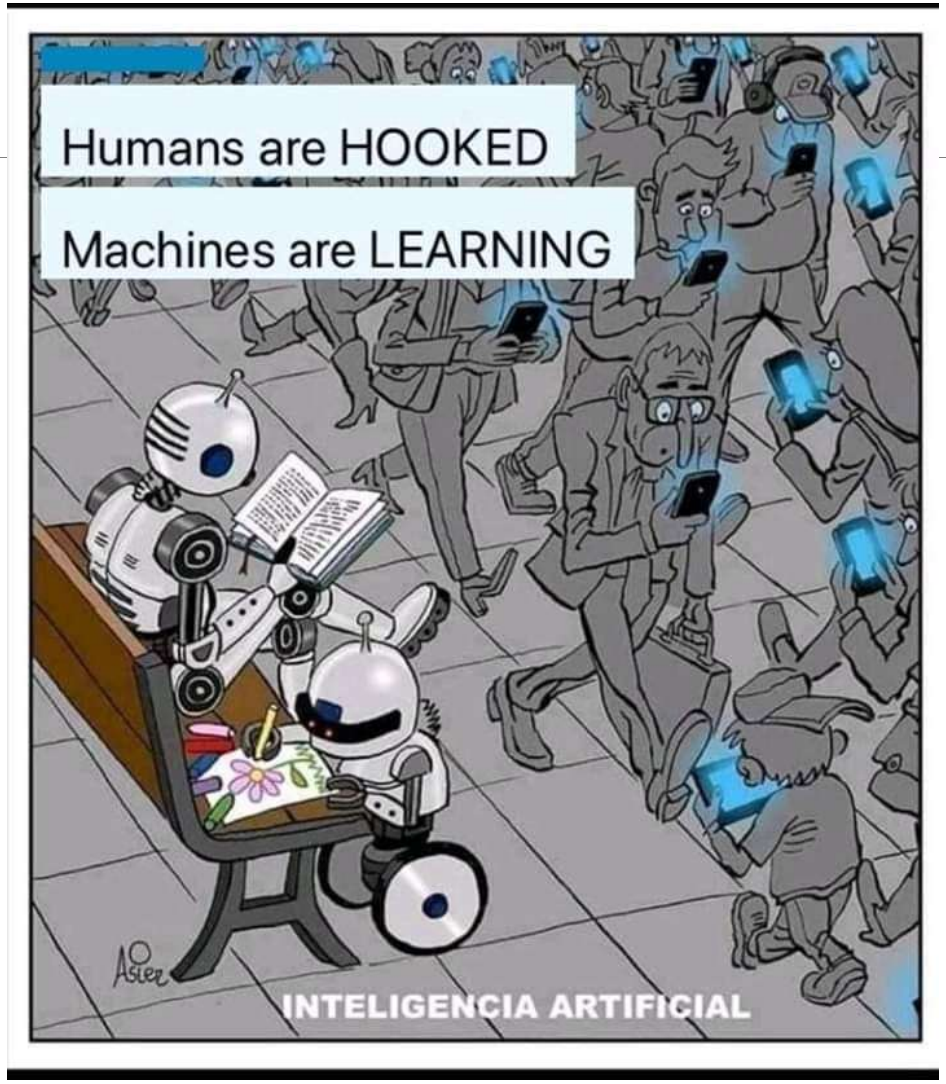
https://en.wikipedia.org/wiki/Drakkhen

https://en.wikipedia.org/wiki/Detention_(video_game)

Conclusion

- As we mentioned, AI is to challenge Gods! Humans try to create new creatures.
- Deep reinforcement learning is the most promising AI approach so far.
- However, both “deep” and “reinforcement learning” were inspired by humans and animals.


50th decimal of Pi?
0
34786 x 786565?
27361450090
01001000011001010110
11000110110001101111
01110111011011
1101110010011
0110001100100
So? What do you think? Robot or human?
Definitely a robot!
Wrong! It was a human!!
Oh my God!
Incredible!
THE DAILY ROBOT
HUMAN PASSES TURING TEST
WILL HUMANS TAKE OUR JOBS?
CommitStrip.com


Humans are HOOKED
Machines are LEARNING
INTELIGENCIA ARTIFICIAL

Q&A