Sarsa: On-Policy TD Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-Learning: Off-Policy TD Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$;

Algorithm 1 Simulated annealing optimisation method.

Require: Input(T_0, α, N, T_f)
$T \leftarrow T_0$
$S_{act} \leftarrow$ generate initial solution
while $T \geq T_f$ **do**
  for cont $\leftarrow$ 1 TO $N(T)$ **do**
    $S_{cond} \leftarrow$ Neighbour solution [from (S_{act})]
    $\delta \leftarrow f(S_{cond}) - f(S_{act})$
    if rand(0,1) $< e^{-\delta/T}$ **or** $\delta < 0$ **then**
      $S_{act} \leftarrow S_{cond}$
    end if
  end for
  $T \leftarrow \alpha(T)$
end while
return Best S_{act} visited

Algorithm 2 Tabu search optimisation method.

$s_0 \leftarrow$ generate initial solution
$s_{best} \leftarrow s_0$
tabuList $\leftarrow \{s_0\}$
repeat
  $\{s_1, s_2, ..., s_n\} \leftarrow$ generate neighbourhood from $(s_{best}$
  $s_{candidate} \leftarrow s_1$
  for i $\leftarrow$ 2 TO n **do**
    $\delta \leftarrow f(s_i) - f(s_{candidate})$
    if s_i is not in tabuList **and** $\delta < 0$ **then**
      $s_{candidate} \leftarrow s_i$
    end if
  end for
  $s_{best} \leftarrow s_{candidate}$
  Add $s_{candidate}$ to tabuList
until a termination criterion is satisfied
return s_{best}

Algorithm 3 Genetic algorithm optimisation method.

$t \leftarrow 0$
Initialise $P(t)$ ► initial population
Evaluate $P(t)$
repeat
  Generate offspring $C(t)$ from $P(t)$ ► using crossover and mutation
  Evaluate $C(t)$
  Select $P(t+1)$ from $P(t) \cup C(t)$
  $t \leftarrow t+1$
until a termination criterion is satisfied
return Best individual found from P

- deduction: cause + rule ⇒ effect
- abduction: effect + rule ⇒ cause
- induction: cause + effect ⇒ rule

A production rule has the form

```
if <condition> then <conclusion>
```

P(A v B) = P(A) + P(B) - P(A ^ B)

P(A) = P(A ^ B) + P(A ^ ~B)

$$P(A|B) = \frac{P(A,B)}{P(B)}$$

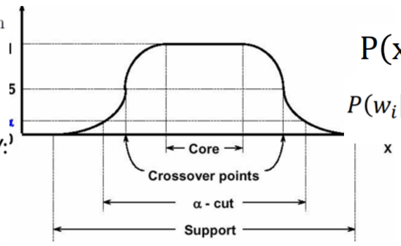
P(A,B,C,D) = P(A|B,C,D)P(B|C,D)P(C|D)P(D)

P(~A|B) = 1 - P(A|B)

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\tilde{\ }A)P(\tilde{\ }A)}$$

$$P(x_1, x_2, \dots, x_n|y) = P(x_1 \mid y) \cdot P(x_2 \mid y) \cdots P($$

$$P(w_i|w_{i-2}, w_{i-1}) = \frac{P(w_i, w_{i-2}, w_{i-1})}{P(w_{i-2}, w_{i-1})} \approx \frac{count(w_i, w_{i-2}, w_{i-1})}{count(w_{i-2}, w_{i-1})}$$



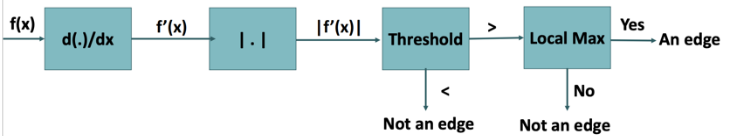
Cumulative Distribution Function (CDF) of pixels in image X and Y:

$$cdf_x(i) = \sum_{j=0}^{i} p_x(j)$$

$cdf_y(i) = (i+1)K$, for $0 \leq i < L$ for some constant K.

So, the transfer function can be defined as:

$$\mathbf{T(i) = round}\left(\frac{\mathbf{cdf_x(i) - cdfmin}}{\mathbf{1 - cdfmin}} \times \mathbf{(L-1)}\right) \xrightarrow{\text{if } cdf_{min}=0} \mathbf{round(cdf_x(i) \times (L-1))}$$



TF-IDF(t, d, D)=TF(t, d)× IDF(t, D)

$$\mathrm{TF(t,d)} = \frac{\text{Number of times term "t" appears in document d}}{\text{Total number of terms in document d}}$$

$$\mathrm{IDF(t,D)} = \mathrm{Log}_{10}\frac{\text{Total number of documents in corpus D}}{\text{Number of documents containing term t}}$$

Algorithm 1 Memory-based explainable reinforcement learning approach with the on-policy method SARSA to compute the probability of success and the number of transitions to the goal state.

Initialize $Q(s,a)$, T_t, T_s, P_s, N_t
for each episode **do**
  Initialize $T_{List}[]$
  Choose an action using $a_t \leftarrow$ SELECTACTION(s_t)
  repeat
    Take action a_t
    Save state-action transition T_{List}.add(s, a)
    $T_t[s][a] \leftarrow T_t[s][a] + 1$
    Observe reward r_{t+1} and next state s_{t+1}
    Choose next action a_{t+1} using softmax action selection method
    $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$
    $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$
  until s is terminal (goal or aversive state)
  if s is goal state **then**
    for each s,a $\in T_{List}$ **do**
      $T_s[s][a] \leftarrow T_s[s][a] + 1$
    end for
  end if
  Compute $P_s \leftarrow T_s/T_t$
  Compute N_t for each s $\in T_{List}$ as pos(s, T_{List}) + 1
end for

Algorithm 2 Explainable reinforcement learning approach to compute the probability of success using the learning-based approach.

Initialize $Q(s,a)$, $\mathbb{P}(s_t,a_t)$
for each episode **do**
  Initialize s_t
  Choose an action a_t from s_t
  repeat
    Take action a_t
    Observe reward r_{t+1} and next state s_{t+1}
    Choose next action a_{t+1} using softmax action selection method
    $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$
    $\mathbb{P}(s_t,a_t) \leftarrow \mathbb{P}(s_t,a_t) + \alpha[\varphi_{t+1} + \mathbb{P}(s_{t+1},a_{t+1}) - \mathbb{P}(s_t,a_t)]$
    $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$
  until s_t is terminal (goal or aversive state)

Algorithm 3 Explainable reinforcement learning approach to compute the probability of success using the introspection-based approach.

Initialize $Q(s,a)$, $\hat{P}_s$
for each episode **do**
  Initialize s_t
  Choose an action a_t from s_t
  repeat
    Take action a_t
    Observe reward r_{t+1} and next state s_{t+1}
    Choose next action a_{t+1} using softmax action selection method
    $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$
    $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$
  until s_t is terminal (goal or aversive state)
  $\hat{P}_s \approx \left[(1-\sigma) \cdot \left(\frac{1}{2} \cdot log_{10}\frac{Q(s_t,a_t)}{R^T} + 1\right)\right]_{\hat{P}_s \geq 0}^{\hat{P}_s \leq 1}$

	Search Algorithms Covered at this Lecture	
Uninformed Search	Breath First Search (BFS)	
	Depth First Search (DFS)	
	Iterative Deepening Search (IDDFS)	calls DFS for different depths
	Uniform Cost Search (UCS)	expand least-cost unexpanded node (按照所有的距离累加排序直到目标)
Informed Searches	Greedy Best-First Search	priority queue（base heuristic)按照h从小到大
	A* Search	g(n) + h(n)（从小到大)

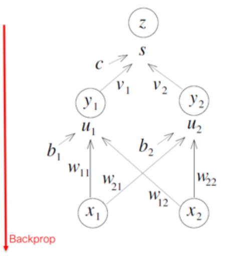
Ensemble Learning, Random Forests:

Select the number of models to build, m
for $i = 1$ *to* m **do**
  Generate a bootstrap sample of the original data
  Train a tree model on this sample
  for *each split* **do**
    Randomly select k ($< P$) of the original predictors
    Select the best predictor among the k predictors and partition the data
  end
  Use typical tree model stopping criteria to determine when a tree is complete (but do not prune)
end

Boosting:

1. Initialize the observation weights $w_i=1/N$, i=1,2,...,N.
2. For m=1 to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$err_m = \frac{\sum_1^N w_i I(y_i \,!= G_m(x_i))}{\sum_1^N w_i}$$
   (c) Compute the influence α_m
   $$\alpha_m = ½ \ln\frac{1-errm}{err_m}$$
   正确：乘 $e^{-\alpha}$
   错误：乘 $e^{+\alpha}$
   (d) Set $w_i^{new\_iteration} = w_i^{old\_iteration}\, e^{\pm\alpha}$, i=1,2,...,N.
3. Output $G(x)=\text{sign}[\sum_{m=1}^{M} \alpha_m G_m(x)]$.

3. Backward pass: Propagate errors back through the network to adjust weights ($E = \frac{1}{2}\sum_i (z_i - t_i)^2$).



Partial derivative:
$$\frac{\partial E}{\partial z} = z - t$$
$$\frac{dz}{ds} = g'(s) = z(1-z)$$
$$\frac{\partial s}{\partial y_1} = v_1$$
$$\frac{\partial y_1}{\partial u_1} = y_1(1-y_1)$$

Then: $w_{ij}^{new\_epoch} \leftarrow w_{ij}^{old\_epoch} - \eta \frac{\partial E}{\partial w_{ij}}$

$$\frac{\partial E}{\partial s} = (z-t)z\,(1-z)$$
$$\frac{\partial E}{\partial v_1} = (z-t)z\,(1-z)\,y_1$$
$$\frac{\partial E}{\partial u_1} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial y_1}\frac{\partial y_1}{\partial u_1} = (z-t)z(1-z)v_1y_1(1-y_1)$$
$$\frac{\partial E}{\partial w_{11}} = (z-t)z\,(1-z)v_1\,y_1\,(1-y_1)\,x_1$$

$X_n = (X - X_{min}) / (X_{max} - X_{min})$; $X_n \in [0,1]$ or

$X_n = 2*(X - X_{min})/(X_{max} - X_{min}) - 1$; $X_n \in [-1,1]$

Rule of thumb: N_h should lead to a number of parameters (weights) N_w that:

$N_w <$ (Number of samples) / 10

The number of weights N_w of a MLP, with N_i neurons in its input layer, a hidden layer with N_h neurons, and N_o neurons in the output layer is:

$N_w = (N_i+1)*N_h+(N_h+1)*N_o$

Machine learning (subfields of AI algorithms)
three types: supervised learning
unsupervised learning
reinforcement learning

① supervised learning:
- Regression (one output, real value)
- Binary classification (two discrete classes [positive / negative])
- Multiclass classification (discrete classes, >2 possible values)

$\boxed{\text{real world input}}$ $\boxed{\text{Model input}}$ $\boxed{\text{Model}}$ $\boxed{\text{Model output}}$ $\boxed{\text{real world output}}$

method: decision tree
- Entropy $H(\langle p_1 \cdots p_n \rangle) = \sum_{i=1}^{n} -P_i \log_2 P_i$
- Minimal error Pruning: $E = 1 - \frac{n+1}{N+k}$ (number in majority class; total items; number of class)

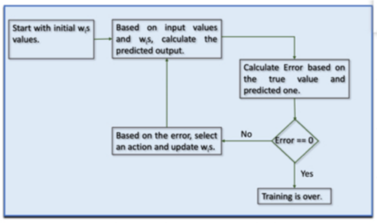
② unsupervised learning
learn about a dataset without labels
clustering: Grouping similar data point together

neural network

mathematical formula: $z = g(s) = w_0 + \sum_i w_i x_i$ (transfer function)
w_0: bias (constant); w_i: weight; x_i: input

perceptron: neuron with step transfer function (like 0, 1)
and ($w_0 = -1.5$, $w_1 = 1$, $w_2 = 1$)
or ($w_0 = -0.5$, $w_1 = 1$, $w_2 = 1$)

learning:
error function: $E = \frac{1}{2}\sum_i (z_i - t_i)^2$
z_i: actual output; t_i: target output



Artificial Neural Network Training Diagram

if $z(s) = \frac{1}{1+e^{-s}}$, $z'(s) = z(1-z)$
$z(s) = \tanh(s)$, $z'(s) = 1 - z^2$

step: Forward pass: list all the formulas
Calculate the error: $E = \frac{1}{2}\sum_i (z_i - t_i)^2$
Backward pass: adjust weight use derivative

Returns

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- $V^\pi(s)$ is the expected value of following policy π in state s
- $V^*(s)$ be the maximum discounted reward obtainable from s

$$Q(s,a) = r(s,a) + \gamma V^*(s')$$

O(1) < O(log n) < O(n) < O(n log n) < O(n²) < O(n³) < O(2ⁿ)