Instance Based Learning

- **given**: an instance $x^{(q)}$ to classify
- find the k training-set instances $(\mathbf{x}^{(1)}, y^{(1)}) \ldots (\mathbf{x}^{(k)}, y^{(k)})$ that are most similar to $x^{(q)}$
- return the class value

$$\hat{y} \leftarrow \underset{v \in \text{values}(Y)}{\arg\max} \sum_{i=1}^{k} \delta(v, y^{(i)}) \qquad \delta(a,b) = \begin{cases} 1 \text{ if } a = b \\ 0 \text{ otherwise} \end{cases}$$

incremental deletion

start with all training instances in memory
for each training instance $(x^{(i)}, y^{(i)})$
    if other training instances provide correct classification for $(x^{(i)}, y^{(i)})$
        delete it from the memory

incremental growth

start with an empty memory
for each training instance $(x^{(i)}, y^{(i)})$
    if other training instances in memory **don't** correctly classify $(x^{(i)}, y^{(i)})$
        add it to the memory

Find nearest neighbor in k-d tree

```
NearestNeighbor(instance x^(q))
    PQ = { }                                        // minimizing priority queue
    best_dist = ∞                                   // smallest distance seen so far
    PQ.push(root, 0)
    while PQ is not empty
        (node, bound) = PQ.pop();
        if (bound ≥ best_dist)
            return best_node.instance               // nearest neighbor found
        dist = distance(x^(q), node. instance)
        if (dist < best_dist)
            best_dist = dist
            best_node = node
        if (q[node.feature] – node.threshold > 0)
            PQ.push(node.left, x^(q)[node.feature] – node.threshold)
            PQ.push(node.right, 0)
        else
            PQ.push(node.left, 0)
            PQ.push(node.right, node. threshold - x^(q) [node.feature])
    return best_node. instance
```

Decision Trees

MakeSubtree(set of training instances D)
    C = DetermineCandidateSplits(D)
    if stopping criteria met
        make a leaf node N
        determine class label/probabilities for N
    else
        make an internal node N
        S = FindBestSplit(D, C)
        for each outcome k of S
            D_k = subset of instances that have outcome k
            k^{th} child of N = MakeSubtree(D_k)
    return subtree rooted at N

$$H(Y) = -\sum_y P(y) \log_2 P(y)$$

$$H(Y|X) = \sum_x P(X = x) H(Y|X = x)$$

$$H(Y|X = x) = -\sum_y P(Y = y|X = x) \log_2 P(Y = y|X = x)$$

$$\text{InfoGain}(D, S) = H_D(Y) - H_D(Y|S)$$

$$\text{SplitInfo}(D, S) = -\sum_{k \in \text{outcomes}(S)} \frac{|D_k|}{|D|} \log_2 \left(\frac{D_k}{D} \right)$$

$$\text{GainRatio}(D, S) = \frac{\text{InfoGain}(D, S)}{\text{SplitInfo}(D, S)}$$

$$\sum_{i=1}^{|D|} (y_i - \hat{y_i})^2 = \sum_{L \in \text{leaves}} \sum_{i \in L} (y_i - \hat{y_i})^2$$

Pruning in C4.5

1. split given data into training and *tuning* (*validation*) sets
2. grow a complete tree
3. do until further pruning is harmful
    - evaluate impact on tuning-set accuracy of pruning each node
    - greedily remove the one that most improves tuning-set accuracy

Methodology

$$\text{TPR (recall)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

$$error_D(h) \pm z_\alpha \sqrt{\frac{error_D(h)(1 - error_D(h))}{n}}$$

$$t = \frac{\bar{\delta}}{\sqrt{\frac{1}{n(n-1)} \sum_{i=1}^{n} (\delta_i - \bar{\delta})^2}}$$

Learning curve

given training/test set partition
- for each sample size s on learning curve
    - (optionally) repeat n times
        - randomly select s instances from training set
        - learn model
        - evaluate model on test set to determine accuracy a
        - plot (s, a) or (s, avg. accuracy and error bars)

ROC curve

let $\left(\left(y^{(1)}, c^{(1)}\right) \ldots \left(y^{(m)}, c^{(m)}\right) \right)$ be the test-set instances sorted according to predicted confidence $c^{(i)}$ that each instance is positive
let num_neg, num_pos be the number of negative/positive instances in the test set
$TP = 0$, $FP = 0$
$last\_TP = 0$
for $i = 1$ to m
    // find thresholds where there is a pos instance on high side, neg instance on low side
    if ($i > 1$) and ($c^{(i)} \neq c^{(i-1)}$) and ($y^{(i)}$ == neg) and ($TP > last\_TP$)
        $FPR = FP / num\_neg$, $TPR = TP / num\_pos$
        output (FPR, TPR) coordinate
        $last\_TP = TP$
    if $y^{(i)}$ == pos
        ++TP
    else
        ++FP
$FPR = FP / num\_neg$, $TPR = TP / num\_pos$
output (FPR, TPR) coordinate

Bayes Network

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | \text{Parents}(X_i))$$

$$P(b|j, m) = \frac{P(b, j, m)}{P(j, m)} = \frac{P(b, j, m)}{P(b, j, m) + P(\neg b, j, m)}$$

$$\text{MAP (Laplace):}\ P(X = x) = \frac{n_x + 1}{\sum_{v \in \text{values}(X)} (n_v + 1)}$$

$$\text{m-estimates:}\ P(X = x) = \frac{n_x + p_x m}{\left(\sum_{v \in \text{values}(X)} n_v \right) + m}$$

$$\text{M-step:}\ P(a|b, e) = \frac{E\#(a \wedge b \wedge e)}{E\#(b \wedge e)}$$

$$I(X, Y) = \sum_x \sum_y P(x, y) \log_2 \frac{P(x, y)}{P(x)P(y)}$$

$$P(X_1, \ldots, X_n, Y) = P(Y) \prod_{i=1}^{n} P(X_i|Y)$$

$$P(Y = y|X) = \frac{P(y) \prod_{i=1}^{n} P(x_i|y)}{\sum_{y'} P(y') \prod_{i=1}^{n} P(x_i|y')}$$

$$I(X_i, X_j|Y) = \sum_{x_i} \sum_{x_j} \sum_y P(x_i, x_y, y) \log_2 \frac{P(x_i, x_j|y)}{P(x_i|y)P(x_j|y)}$$

$$P(Y = y|\mathbf{x}) = \frac{P(y)P(\mathbf{x}|y)}{\sum_{y'} P(y')P(\mathbf{x}|y')}$$

MCMC Gibbs Sampling

state ← random assignment to non-evidence variables
for i ← 1 **to** *num-samples* **do**
    for each non-evidence variable x
        sample x according to P(x|*Parents*(x))
        state ← *state* with new value of x
P(F) ← fraction of states in which F is true

EM algorithm

Given:
- data set with some missing values
- model structure, initial model parameters

Repeat until convergence
- *Expectation* (E) step: using current model, compute expectation over missing values
- *Maximization* (M) step: update model parameters with those that maximize probability of the data (MLE or MAP)

Chow-Liu

1. compute weight $I(X_i, X_j)$ of each possible edge (X_i, X_j)
2. find maximum weight spanning tree (MST)
3. assign edge directions in MST

Prim MST

given: graph with vertices V and edges E

$V_{new} \leftarrow \{ v \}$ where v is an arbitrary vertex from V
$E_{new} \leftarrow \{ \}$
repeat until $V_{new} = V$
{
    choose an edge (u, v) in E with max weight where u is in V_{new} and v is not
    add v to V_{new} and (u, v) to E_{new}
}
return V_{new} and E_{new} which represent an MST

Kruskal MST

given: graph with vertices V and edges E

$E_{new} \leftarrow \{ \}$
for each (u, v) in E ordered by weight (from high to low)
{
    remove (u, v) from E
    if adding (u, v) to E_{new} does not create a cycle
        add (u, v) to E_{new}
}
return V and E_{new} which represent an MST

BN Hill Climbing Algorithm

given: data set D, initial network B_0

$i = 0$
$B_{best} \leftarrow B_0$
while stopping criteria not met
{
    for each possible operator application a
    {
        $B_{new} \leftarrow$ apply(a, B_i)
        if score(B_{new}) > score(B_{best})
            $B_{best} \leftarrow B_{new}$
    }
    ++i
    $B_i \leftarrow B_{best}$
}
return B_i

TAN algorithm

1. compute weight $I(X_i, X_j \mid Y)$ for each possible edge (X_i, X_j) between features
2. find maximum weight spanning tree (MST) for graph over $X_1 \ldots X_n$
3. assign edge directions in MST
4. construct a TAN model by adding node for Y and an edge from Y to each X_i

Markov Network

$$P(\mathbf{V}) = \frac{1}{Z} \prod_k \phi_k(\mathbf{V})$$

$$Z = \sum_{\mathbf{v} \in \mathbf{V}} \prod_k \phi_k(\mathbf{v})$$

Neural Network

$$E(\mathbf{w}) = \frac{1}{2} \sum_{d \in D} \left(y^{(d)} - o^{(d)} \right)^2$$

$$E(\mathbf{w}) = \sum_{d \in D} -y^{(d)} \ln \left(o^{(d)} \right) - \left(1 - y^{(d)} \right) \ln \left(1 - o^{(d)} \right)$$

$$E(\mathbf{w}) = -\sum_{d \in D} \sum_{i=1}^{\#class} y_i^{(d)} \ln \left(o_i^{(d)} \right)$$

$$net = w_0 + \sum_i w_i x_i$$

$$f(net) = \frac{1}{1 + e^{-net}} \qquad \text{(Sigmoid)}$$

$$f(net) = \tanh(x) = \frac{2}{1 + e^{-2net}} - 1 \qquad \text{(Tanh)}$$

$$f(net) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \qquad \text{(ReLU)}$$

$$f(net) = \frac{e^{net_j}}{\sum_j e^{net_j}} \qquad \text{(softmax)}$$

$$\Delta \mathbf{w} = -\eta \nabla_{\mathbf{w}} E(\mathbf{w})$$

$$\Delta w_{ji} = -\eta \delta_j o_i$$

$$\delta_j = -\frac{\partial E}{\partial net_j} = \text{e.g. } o_j(1 - o_j) \sum_k \delta_k w_{kj}$$

given: network structure and a training set $D = \left\{ (\mathbf{x}^{(1)}, y^{(1)}) \ldots (\mathbf{x}^{(m)}, y^{(m)}) \right\}$
initialize all weights in $\mathbf{w}$ to small random numbers
until stopping criteria met do
    initialize the error $E(\mathbf{w}) = 0$
    for each $(\mathbf{x}^{(d)}, y^{(d)})$ in the training set
        input $\mathbf{x}^{(d)}$ to the network and compute output $o^{(d)}$
        increment the error $E(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2} \left(y^{(d)} - o^{(d)} \right)^2$
    calculate the gradient

$$\nabla E(\mathbf{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n} \right]$$

    update the weights

$$\Delta \mathbf{w} = -\eta \nabla E(\mathbf{w})$$

Step 1: $v^{(t+1)} = \mu v^{(t)} - \eta \cdot \frac{\partial E}{\partial w}\Big|_{w^{(t)}}$

Interpret v as velocity, $1 - \mu$ as friction. v stores the moving average of the gradients.

Step 2: $w^{(t+1)} = w^{(t)} + v^{(t+1)}$

Step 1: $v^{(t+1)} = \mu v^{(t)} - \eta \cdot \frac{\partial E}{\partial w}\Big|_{w^{(t)} + \mu v^{(t)}}$

Step 2: $w^{(t+1)} = w^{(t)} + v^{(t+1)}$

Step 1: $G^{(t+1)} = G^{(t)} + \left(\frac{\partial E}{\partial w}\Big|_{w^{(t)}}\right)^2$

G is the sum of the square gradient

Step 2: $w^{(t+1)} = w^{(t)} - \frac{\eta}{\sqrt{G^{(t+1)}+10^{-7}}} \odot \frac{\partial E}{\partial w}\Big|_{w^{(t)}}$

Scale the learning rate by $1/\sqrt{G}$ and follow the negative gradient direction

Step 1: $G^{(t+1)} = \gamma G^{(t)} + (1-\gamma) \cdot \left(\frac{\partial E}{\partial w}\Big|_{w^{(t)}}\right)^2$

G is a moving exponential average of the square gradient.

Step 2: $w^{(t+1)} = w^{(t)} - \frac{\eta}{\sqrt{G^{(t+1)}+10^{-7}}} \odot \frac{\partial E}{\partial w}\Big|_{w^{(t)}}$

Scale the learning rate by $1/\sqrt{G}$ and follow the negative gradient direction.

Step 1: $m^{(t+1)} = \beta_1 m^{(t)} + (1-\beta_1) \cdot \frac{\partial E}{\partial w}\Big|_{w^{(t)}}$

m is the exponential moving average of the gradients (similar to speed in the momentum method)

Step 2: $v^{(t+1)} = \beta_2 v^{(t)} + (1-\beta_2) \cdot \left(\frac{\partial E}{\partial w}\Big|_{w^{(t)}}\right)^2$

v is the exponential moving average of the square gradients

Step 3: $w^{(t+1)} = w^{(t)} - \frac{\eta}{\sqrt{v^{(t+1)}+10^{-8}}} \odot m^{(t+1)}$

$$E\left(\theta^{(D)}\right) = -\frac{1}{2}E_{x\sim p_{data}} \log D(\mathbf{x}) - \frac{1}{2}E_z \log\left(1 - D(G(\mathbf{z}))\right)$$

$$E\left(\theta^{(G)}\right) = -E\left(\theta^{(D)}\right)$$

$$\min_G \max_D E_{x\sim p_{data}} \log D(\mathbf{x}) + E_z \log\left(1 - D(G(\mathbf{z}))\right)$$

Linear Models

Ridge regression

$$\hat{\beta} = \left(X^T X - \lambda I\right)^{-1} X^T Y$$

LASSO coordinate descent

repeat until convergence
  for $j = 1 \ldots n$
    $\rho_j = \sum_i x_j^{(i)}\left(y^{(i)} - \sum_{k\neq j} \beta_k x_k^{(i)}\right)$
    $z_j = \sum_i \left(x_j^{(i)}\right)^2$
    $\beta_j = \frac{1}{z_j} S(\rho_j, \lambda)$

$$S(\rho,\lambda) = \begin{cases} \rho - \lambda & \text{if } \rho > \lambda \\ 0 & \text{if } -\lambda \le \rho \le \lambda \\ \rho + \lambda & \text{if } \rho < -\lambda \end{cases}$$

Ensembles

Bagging

learning:
given: learner L, training set $D = \{\ \langle \mathbf{x}^{(1)}, y^{(1)}\rangle \ \ldots\ \langle \mathbf{x}^{(m)}, y^{(m)}\rangle\ \}$
for $i \leftarrow 1$ to T do
  $D_i \leftarrow m$ instances randomly drawn with replacement from D
  $h_i \leftarrow$ model learned using L on D_i

AdaBoost

given: learner L, # stages T, training set $D = \{\ \langle \mathbf{x}^{(1)}, y^{(1)}\rangle \ \ldots\ \langle \mathbf{x}^{(m)}, y^{(m)}\rangle\ \}$

for all i : $w_1(i) \leftarrow 1/m$ // initialize instance weights
for $t \leftarrow 1$ to T do
  for all i : $p_t(i) \leftarrow w_t(i) / (\sum_i w_t(j))$ // normalize weights
  $h_t \leftarrow$ model learned using L on D and p_t
  $\varepsilon_t \leftarrow \sum_i p_t(i)(1 - \delta(h_t(\mathbf{x}^{(i)}), y^{(i)}))$ // calculate weighted error
  if $\varepsilon_t > 0.5$ then
    $T \leftarrow t - 1$
    break
  $\beta_t \leftarrow \varepsilon_t / (1 - \varepsilon_t)$
  for all i where $h_t(\mathbf{x}^{(i)}) = y^{(i)}$ // down-weight correct examples
    $w_{t+1}(i) \leftarrow w_t(i)\,\beta_t$

return:
$h(\mathbf{x}) = \arg\max_y \sum_{t=1}^{T}\left(\log\frac{1}{\beta_t}\right)\delta(h_t(\mathbf{x}), y)$

Random Forest

given: candidate feature splits F,
  training set $D = \{\ \langle \mathbf{x}^{(1)}, y^{(1)}\rangle \ \ldots\ \langle \mathbf{x}^{(m)}, y^{(m)}\rangle\ \}$
for $i \leftarrow 1$ to T do
  $D_i \leftarrow m$ instances randomly drawn with replacement from D
  $h_i \leftarrow$ randomized decision tree learned with F, D_i

randomized decision tree learning:
to select a split at a node
  $R \leftarrow$ randomly select (without replacement) f feature splits from F
    (where $f << |F|$)
  choose the best feature split in R
do not prune trees

Support Vector Machines

$$h(x) = \begin{cases} +1 & \text{if } \mathbf{w}^T\mathbf{x} + b > 0 \\ -1 & \text{o/w} \end{cases}$$

$$\text{margin}_D(h) = \frac{1}{2}\hat{\mathbf{w}}^T(x_+ - x_-) = \frac{1}{||\mathbf{w}||_2}$$

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||_2^2 \qquad \text{(hard-margin)}$$
$$\text{s.t. } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \ge 1\ \forall i$$

$$\min_{\mathbf{w},b,\xi^{(i)}} \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{m}\xi^{(i)} \qquad \text{(soft-margin)}$$
$$\text{s.t. } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \ge 1 - \xi^{(i)},\ \xi^{(i)} \ge 0\ \forall i$$

$$\max_{\alpha_1,\ldots,\alpha_m} \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{j=1}^{m}\sum_{i=1}^{m}\alpha_j\alpha_k y^{(j)}y^{(k)}\left(\mathbf{x}^{(j)}\cdot\mathbf{x}^{(k)}\right)$$
$$\text{s.t. } \alpha_i \ge 0,\ \sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \qquad \text{(h-m dual)}$$

$k(x,z) = (x\cdot z)^d$ (polynomial d)

$k(x,z) = (x\cdot z + 1)^d$ (polynomial up-to d)

$k(x,z) = \exp\left(-\gamma||x-z||^2\right)$ (RBF)

SMO algorithm

- *Input: C, kernel, kernel parameters, ε*
- *Initialize b and all α's to 0*
- *Repeat until KKT satisfied (to within ε):*
  - *Find an example **e1** that violates KKT (prefer unbound examples here, choose randomly among those)*
  - *Choose a second example **e2**. Prefer one to maximize step size (in practice, faster to just maximize $|E_1 - E_2|$). If that fails to result in change, randomly choose unbound example. If that fails, randomly choose example. If that fails, re-choose **e1**.*
  - *Update α_1 and α_2 in one step, compute new threshold b*

$$\begin{aligned} \alpha_i = 0 &\iff y_i u_i \ge 1 \\ 0 < \alpha_i < C &\iff y_i u_i = 1 \\ \alpha_i = C &\iff y_i u_i \le 1 \end{aligned}$$

- Given examples e1 and e2, set

$$\alpha_2^{new} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta}$$

where $\eta = K(x_1,x_1) + K(x_2,x_2) - 2K(x_1,x_2)$

- Clip this value in the natural way:

$$\alpha_2^{new,clipped} = \begin{cases} H & \text{if } \alpha_2^{new} \ge H; \\ \alpha_2^{new} & \text{if } L < \alpha_2^{new} < H; \\ L & \text{if } \alpha_2^{new} \le L. \end{cases}$$

if y1 = y2 then:
$L = \max(0, \alpha_2 + \alpha_1 - C)$
$H = \min(C, \alpha_2 + \alpha_1)$
otherwise:
$L = \max(0, \alpha_2 - \alpha_1)$
$H = \min(C, C + \alpha_2 - \alpha_1)$

- Set $\alpha_1^{new} = \alpha_1 + s(\alpha_2 - \alpha_2^{new,clipped})$ where $s = y_1 y_2$

Learning Theory

$$error_D(h) = P_{x\in D}\left(c(x) \neq h(x)\right)$$

$$error(h) \le error_D(h) + \sqrt{\frac{VC(\log(2m/VC)+1) + \log(4/\delta)}{m}}$$

$$m \ge \frac{1}{\varepsilon}\left(\ln|H| + \ln\left(\frac{1}{\delta}\right)\right) \quad (h \in C)$$

$$m \ge \frac{1}{\varepsilon}\left(4\log_2\left(\frac{2}{\delta}\right) + 8\text{VC-dim}(H)\log_2\left(\frac{13}{\varepsilon}\right)\right)$$

$$m \ge \frac{1}{\varepsilon^2}\left(\ln|H| + \ln\left(\frac{1}{\delta}\right)\right) \quad \text{(agnostic learning)}$$

$$m < \max\left(\frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right), \frac{\text{VC-dim}(C) - 1}{32\varepsilon}\right) \implies \text{w.p. } \delta,\ error_D(h) > \varepsilon$$

$$VC \le \frac{4R^2}{\text{margin}_D(h)^2}$$

Halving algorithm

// initialize the version space to contain all $h \in H$
$VS_1 \leftarrow H$

for $t \leftarrow 1$ to T do
  given training instance $\langle \mathbf{x}_t, c(\mathbf{x}_t)\rangle$
  $h'(\mathbf{x}_t) = \text{MajorityVote}(VS_t, \mathbf{x}_t)$

  // eliminate all wrong h from version space (reduce the size of the VS by at least half on mistakes)
  $VS_{t+1} \leftarrow \{h \in VS_t : h(\mathbf{x}_t) = c(\mathbf{x}_t)\}$

return VS_{t+1}

Weighted majority

given: a set of predictors $A = \{a_1 \ldots a_n\}$, learning rate $0 \le \beta < 1$

  for all i initialize $w_i \leftarrow 1$
  for each training instance $\langle \mathbf{x}, c(\mathbf{x})\rangle$
    initialize q_0 and q_1 to 0
    for each predictor a_i
      if $a_i(\mathbf{x}) = 0$ then $q_0 \leftarrow q_0 + w_i$
      if $a_i(\mathbf{x}) = 1$ then $q_1 \leftarrow q_1 + w_i$
    if $q_1 > q_0$ then $h(\mathbf{x}) = 1$
    else if $q_0 > q_1$ then $h(\mathbf{x}) = 0$
    else if $q_0 = q_1$ then $h(\mathbf{x}) = 0$ or 1 randomly chosen

    for each predictor a_i do
      if $a_i(\mathbf{x}) \neq c(\mathbf{x})$ then $w_i \leftarrow \beta\, w_i$

Bias, Variance, & Overfitting

$$E_D\left[\left(f(x;D) - E[y|x]\right)^2\right] = \left(E_d[f(x;D)] - E[y|x]\right)^2 + E_D\left[\left(f(x;D) - E_D[f(x;D)]\right)^2\right]$$

$$E\left[\left(y - f(x;D)\right)^2\right] = E\left[\left(y - E[y|x]\right)^2 \mid x, D\right] + \left(f(x;D) - E[y|x]\right)^2$$

$h \in H$ overfits if $\exists h' \in H$ s.t. $error(h) > error(h')$ and $error_D(h) < error_D(h')$

Reinforcement Learning

$$V^\pi(s) = \sum_{t=0}^{\infty}\gamma^t E[r_t]$$

$$\pi^* = \arg\max_\pi V^\pi(s) \quad \forall s$$

$$Q(s,a) = E[r(s,a)] + \gamma E_{s'|s,a}[V^*(s')]$$

$$\pi^*(s) = \arg\max_a Q(s,a)$$

$$P(a_i|s) = \frac{c^{\hat{Q}(s,a_i)}}{\sum_j c^{\hat{Q}(s,a_j)}}$$

Q Learning (Deterministic)

for each s, a initialize table entry $\hat{Q}(s,a) \leftarrow 0$
observe current state s
do forever
  select an action a and execute it
  receive immediate reward r
  observe the new state s'
  update table entry
    $\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'}\hat{Q}(s',a')$
  $s \leftarrow s'$

Q Learning (Non-deterministic)

for each s, a initialize table entry $\hat{Q}(s,a) \leftarrow 0$
observe current state s
do forever
  select an action a and execute it
  receive immediate reward r
  observe the new state s'
  update table entry
    $\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n\left[r + \gamma \max_{a'}\hat{Q}_{n-1}(s',a')\right]$
  $s \leftarrow s'$

AlphaZero

Initialize *DNN f_θ*

Repeat Forever
  Play Game
  Update $\boldsymbol{\theta}$

Play Game:
  Repeat Until Win or Lose:
    From current state *S*, perform MCTS
    Estimate move probabilities $\boldsymbol{\pi}$ by MCTS
    Record (*S*, $\boldsymbol{\pi}$) as an example
    Randomly draw next move from $\boldsymbol{\pi}$

Update $\boldsymbol{\theta}$:
  Let *z* be previous game outcome (+1 or -1)
  Sample from last game's examples (*S*, $\boldsymbol{\pi}$, *z*)
  Train DNN f_θ on sample to get new $\boldsymbol{\theta}$