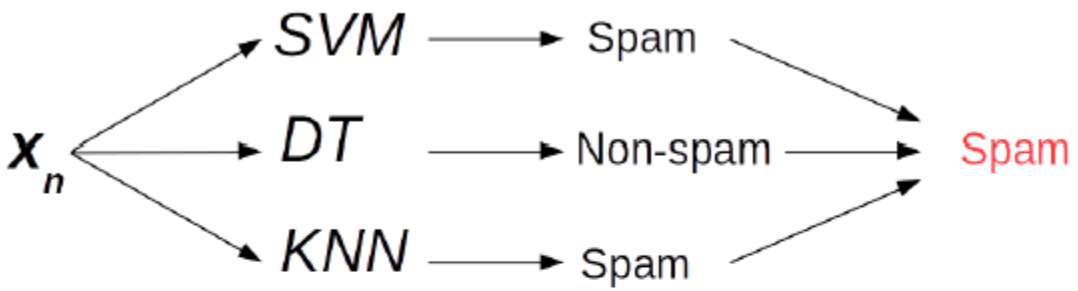
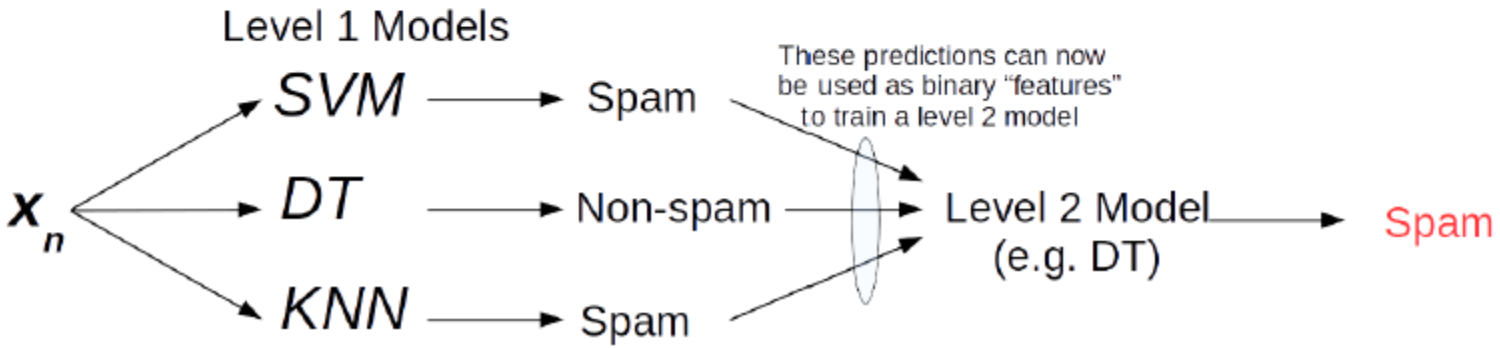
Ensemble Schemes

Approach

- Voting or Averaging of predictions of multiple pre-trained models

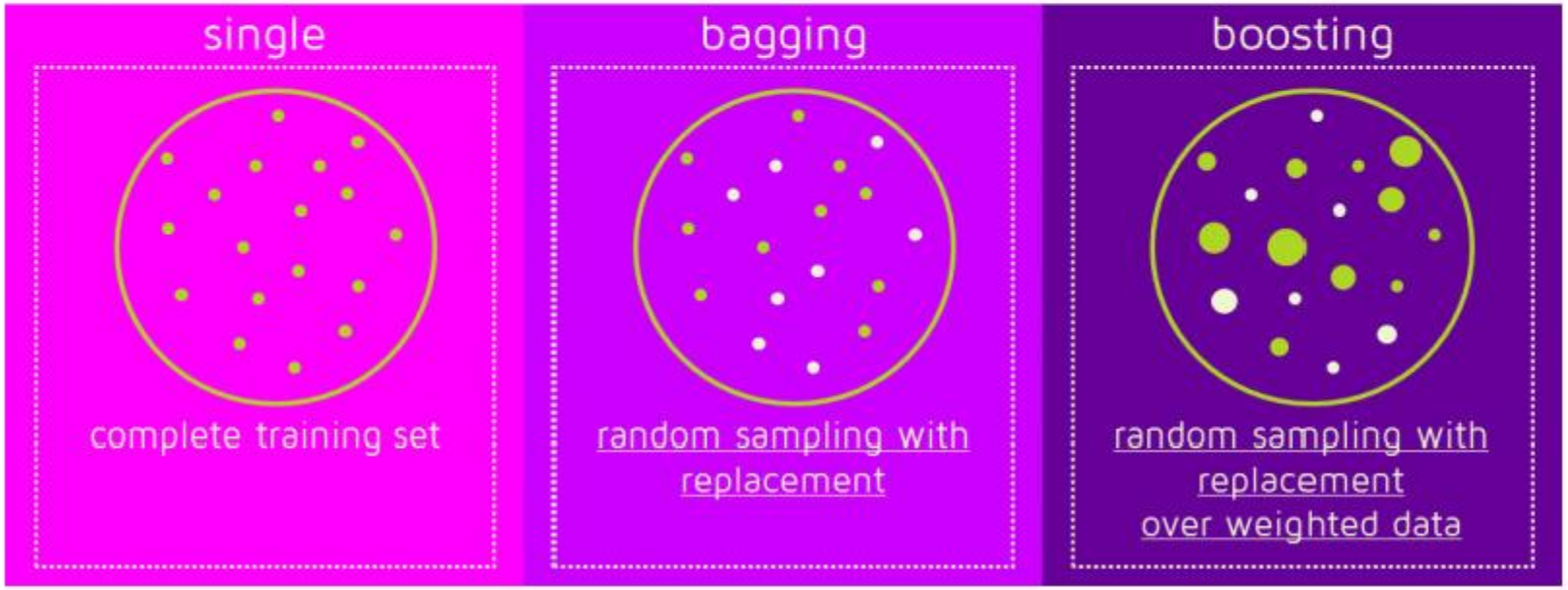


- "Stacking": Use predictions of multiple models as "features" to train a new model and use the new model to make predictions on test data



Bagging

- Instead of training different models on same data, train same model multiple times on different data sets, and "combine" these "different" models
- We can use some simple/weak model as the base model
- How do we get multiple training data sets (in practice, we only have one data set at training time)?

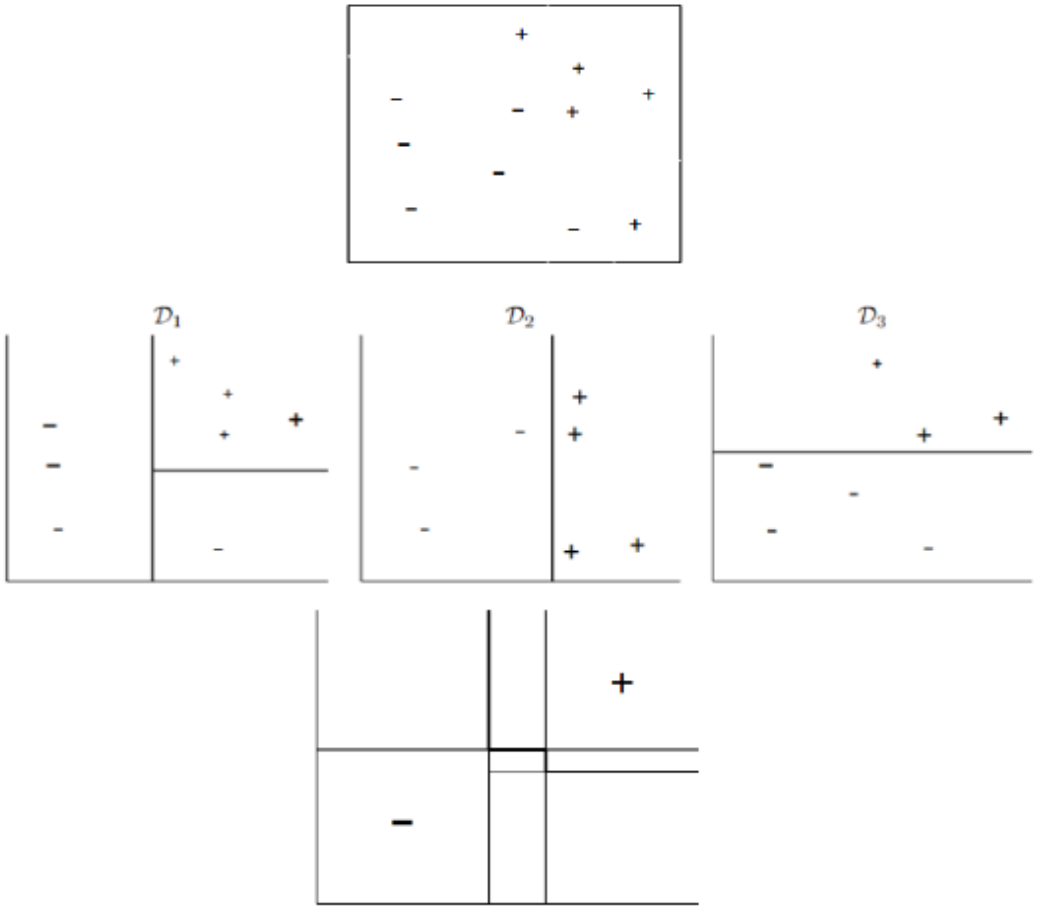


Bagging

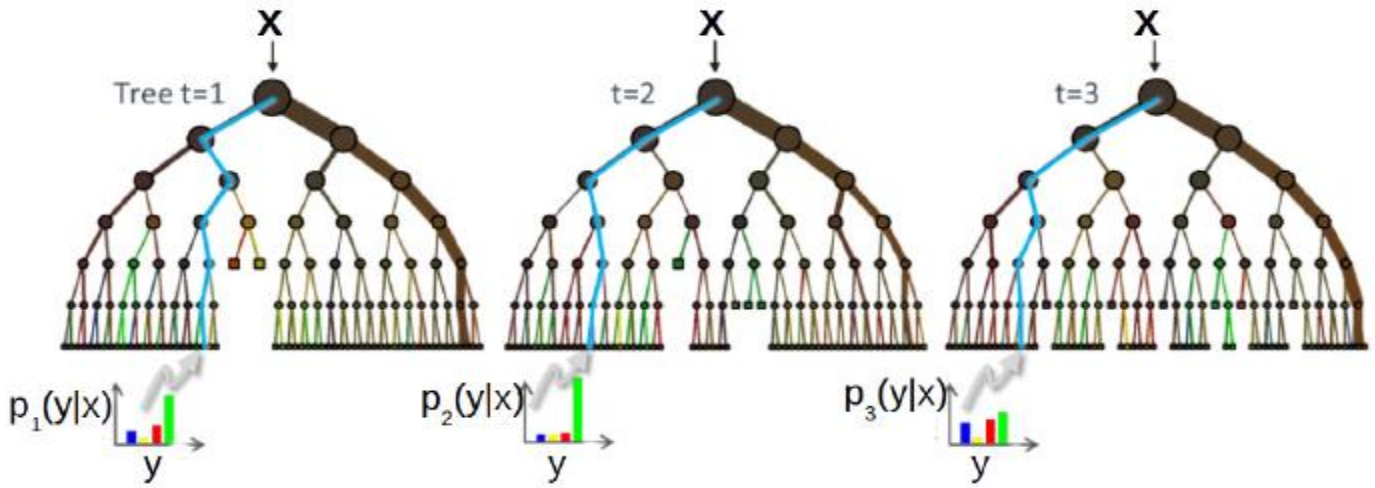
- Bagging stands for Bootstrap Aggregation
- Takes original data set D with N training examples
- Creates M copies $\{\tilde{D}_m\}_{m=1}^{M}$
  - Each $\tilde{D}_m$ is generated from D by sampling with replacement
  - Each data set $\tilde{D}_m$ has the same number of examples as in data set D
  - These data sets are reasonably different from each other (since only about 63% of the original examples appear in any of these data sets)
- Train models $h_1, \ldots, h_M$ using $\tilde{D}_1, \ldots, \tilde{D}_M$, respectively
- Use an averaged model $h = \frac{1}{M} \sum_{m=1}^{M} h_m$ as the final model
- Useful for models with high variance and noisy data

Bagging

Top: Original data, Middle: 3 models (from some model class) learned using three data sets chosen via bootstrapping, Bottom: averaged model



Random forests



- An ensemble of decision tree (DT) classifiers
- Uses bagging on features (each DT will use a random set of features)
  - Given a total of D features, each DT uses $\sqrt{D}$ randomly chosen features
  - Randomly chosen features make the different trees uncorrelated
- All DTs usually have the same depth
- Each DT will split the training data differently at the leaves
- Prediction for a test example votes on/averages predictions from all the DTs

Boosting

- The basic idea
  - Take a weak learning algorithm
    - Only requirement: Should be slightly better than random
  - Turn it into an awesome one by making it focus on difficult cases
- Most boosting algoithms follow these steps:
  1. Train a weak model on some training data
  2. Compute the error of the model on each training example
  3. Give higher importance to examples on which the model made mistakes
  4. Re-train the model using "importance weighted" training examples
  5. Go back to step 2

AdaBoost

- Given: Training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$ with $y_n \in \{-1, +1\}$, $\forall n$
- Initialize weight of each example $(\boldsymbol{x}_n, y_n)$: $D_1(n) = 1/N$, $\forall n$
- For round $t = 1 : T$
  - Learn a weak $h_t(\boldsymbol{x}) \rightarrow \{-1, +1\}$ using training data weighted as per D_t
  - Compute the weighted fraction of errors of h_t on this training data

$$\epsilon_t = \sum_{n=1}^{N} D_t(n) \mathbb{1}[h_t(\boldsymbol{x}_n) \neq y_n]$$

  - Set "importance" of h_t: $\alpha_t = \frac{1}{2} \log(\frac{1-\epsilon_t}{\epsilon_t})$(gets larger as ϵ_t gets smaller)
  - Update the weight of each example

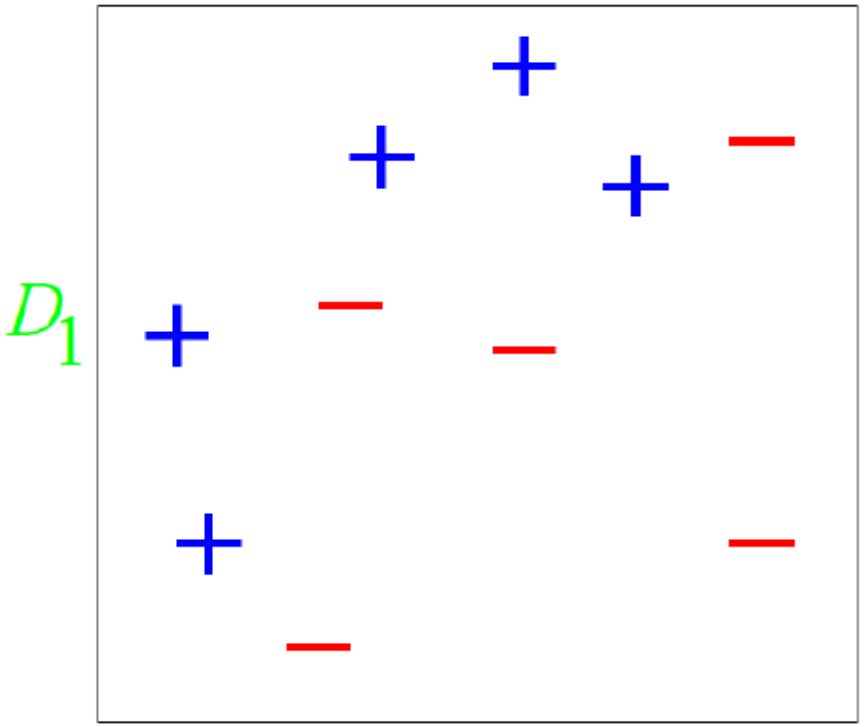
$$\begin{aligned} D_{t+1}(n) &\propto \begin{cases} D_t(n) \times \exp(-\alpha_t) & \text{if } h_t(\boldsymbol{x}_n) = y_n \quad \text{(correct prediction: decrease weight)} \\ D_t(n) \times \exp(\alpha_t) & \text{if } h_t(\boldsymbol{x}_n) \neq y_n \quad \text{(incorrect prediction: increase weight)} \end{cases} \\ &= D_t(n) \exp(-\alpha_t y_n h_t(\boldsymbol{x}_n)) \end{aligned}$$

  - Normalize D_{t+1} so that it sums to 1: $D_{t+1}(n) = \frac{D_{t+1}(n)}{\sum_{m=1}^{N} D_{t+1}(m)}$
- Output the "boosted" final hypothesis $H(\boldsymbol{x}) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}))$

AdaBoost

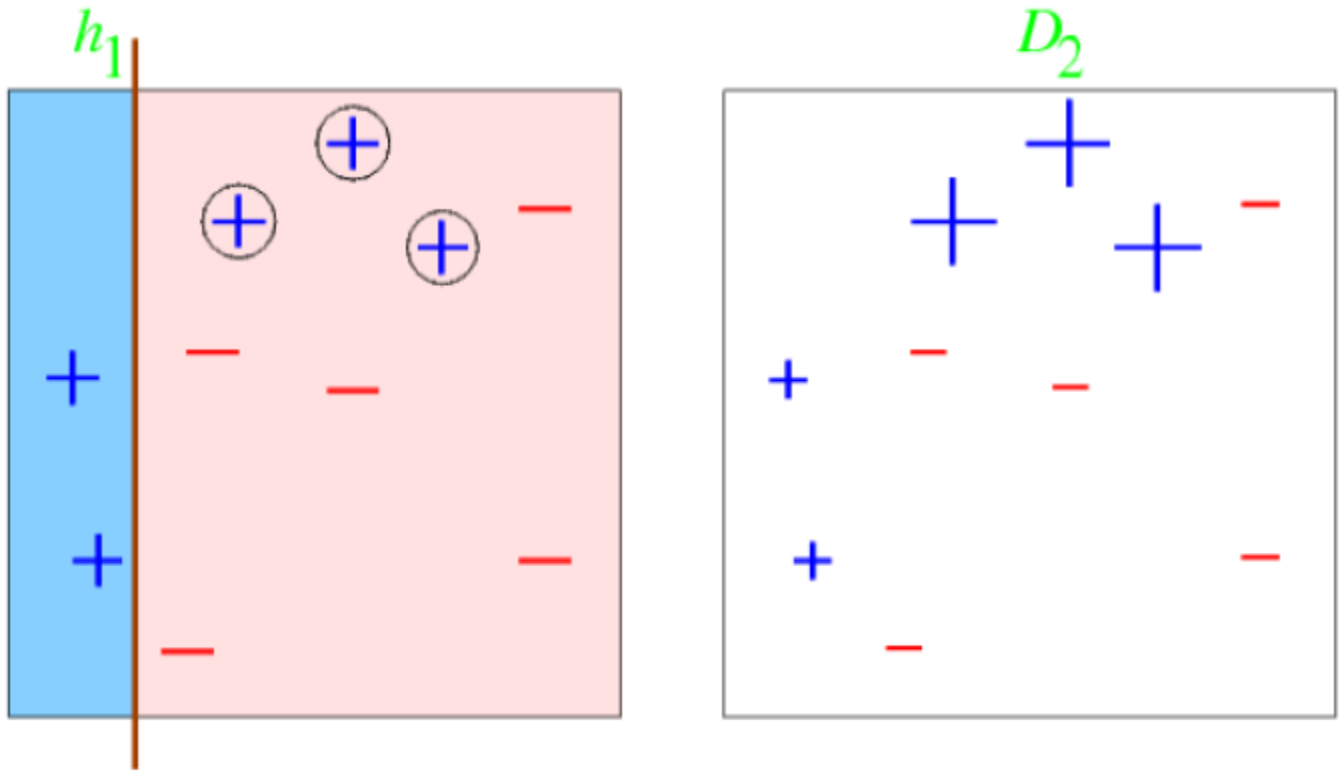
Consider binary classification with 10 training examples

Initial weight distribution D_1 is uniform (each point has equal weight $= 1/10$)



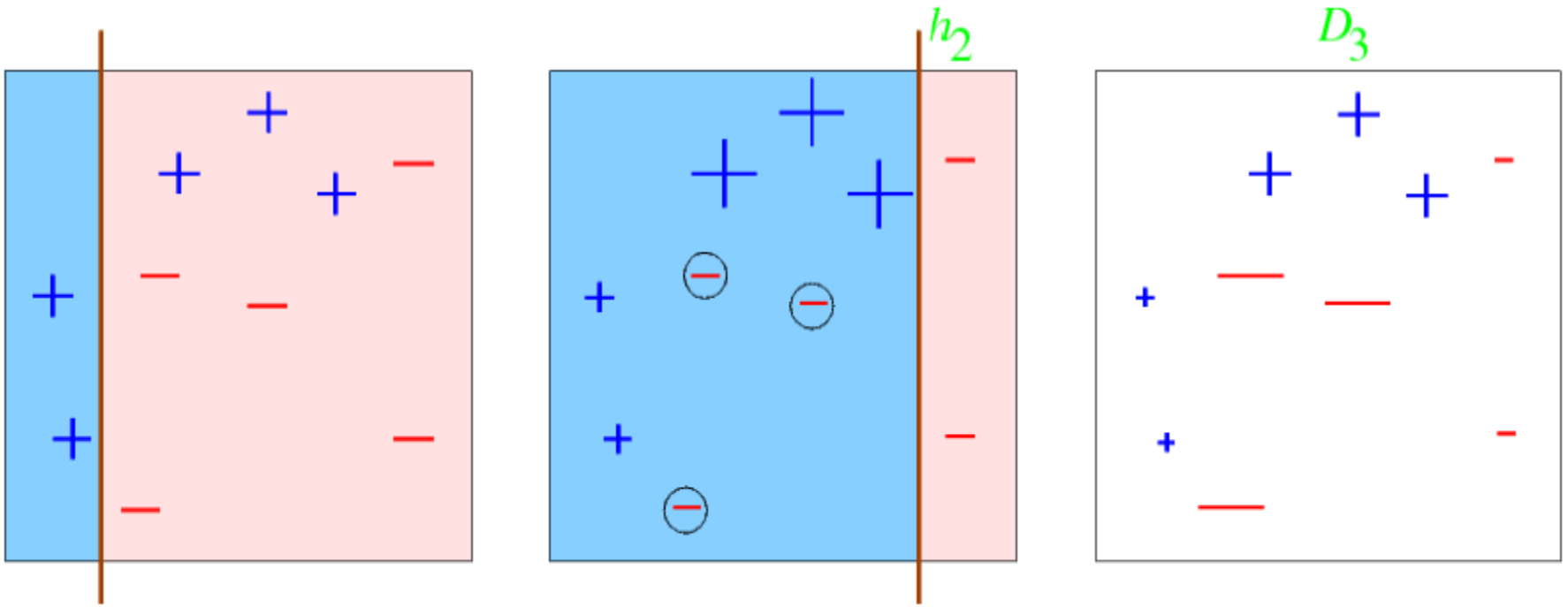
Each of our weak classifers will be an **axis-parallel linear classifier**

AdaBoost



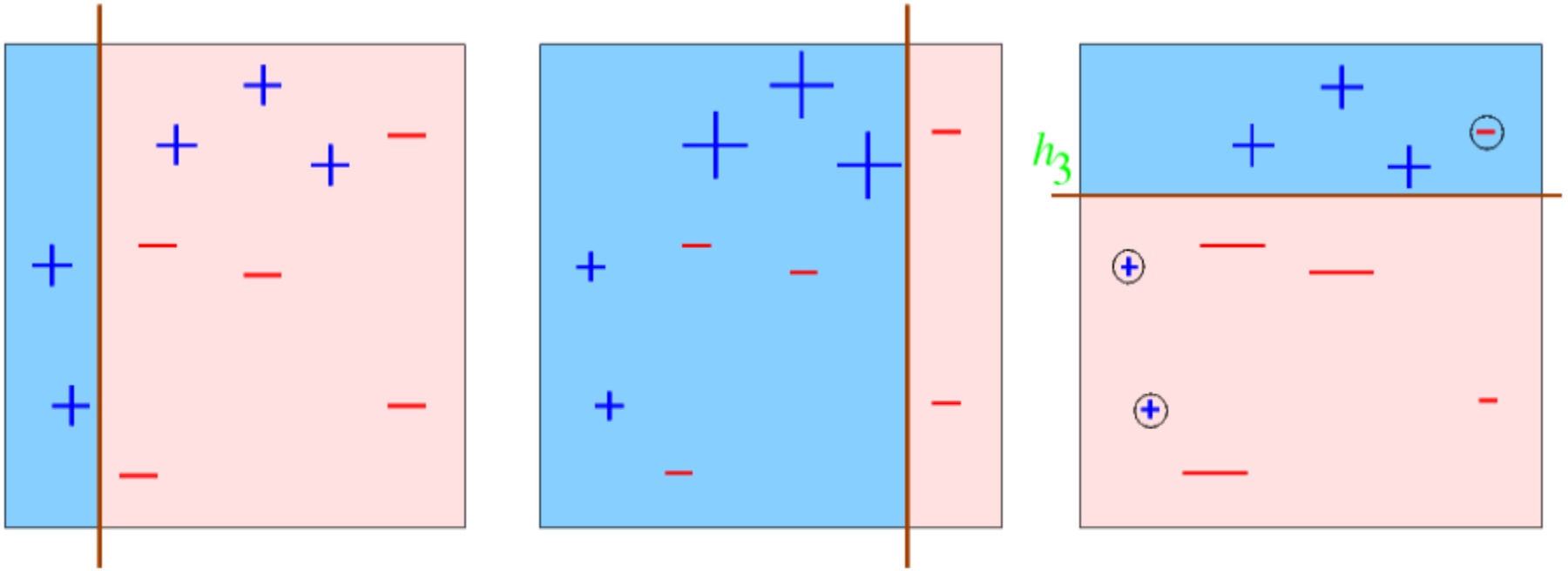
- Error rate of h_1: $\epsilon_1 = 0.3$; weight of h_1: $\alpha_1 = \frac{1}{2}\ln((1-\epsilon_1)/\epsilon_1) = 0.42$
- Each misclassified point upweighted (weight multiplied by $\exp(\alpha_2)$)
- Each correctly classified point downweighted (weight multiplied by $\exp(-\alpha_2)$)

AdaBoost



- Error rate of h_2: $\epsilon_2 = 0.21$; weight of h_2: $\alpha_2 = \frac{1}{2}\ln((1-\epsilon_2)/\epsilon_2) = 0.65$
- Each misclassified point upweighted (weight multiplied by $\exp(\alpha_2)$)
- Each correctly classified point downweighted (weight multiplied by $\exp(-\alpha_2)$)

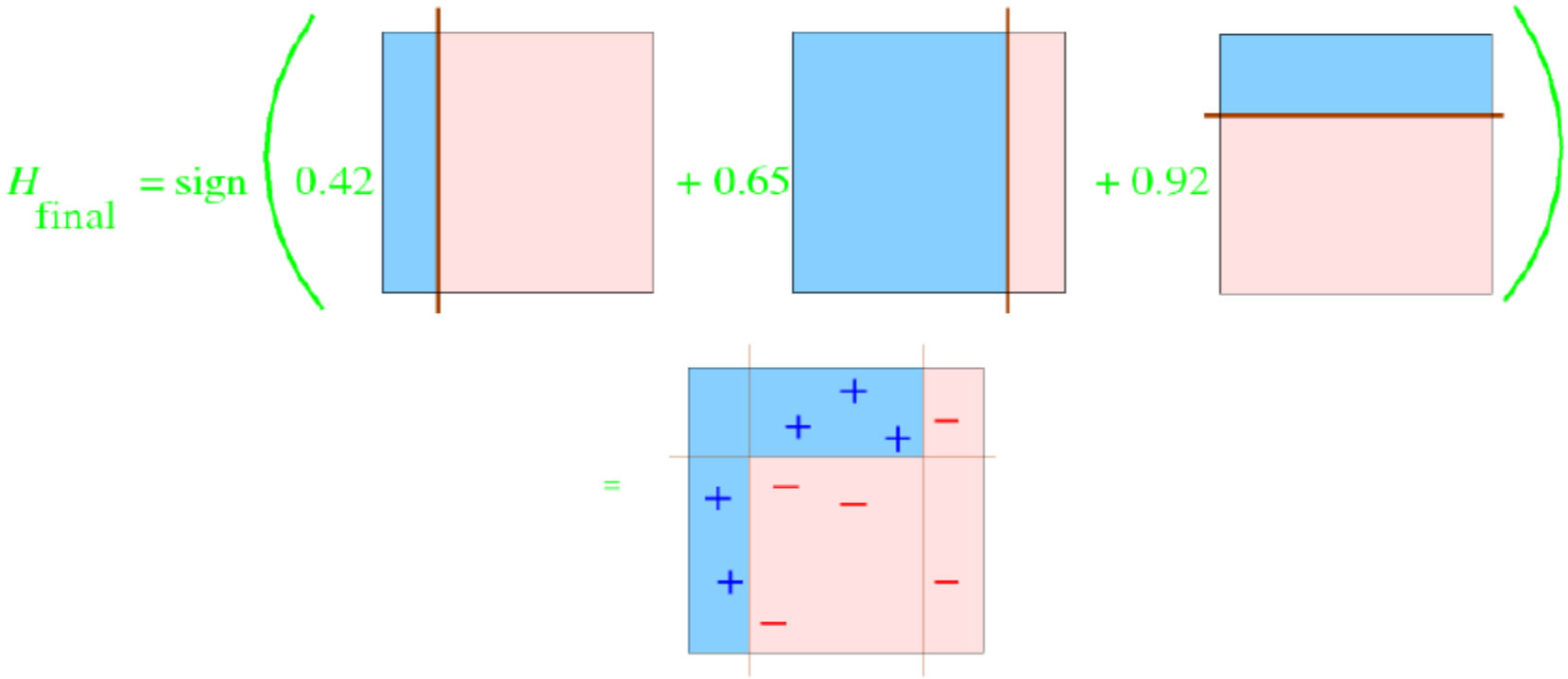
AdaBoost



- Error rate of h_3: $\epsilon_3 = 0.14$; weight of h_3: $\alpha_3 = \frac{1}{2}\ln((1-\epsilon_3)/\epsilon_3) = 0.92$
- Suppose we decide to stop after round 3
- Our ensemble now consists of 3 classifiers: h_1, h_2, h_3

AdaBoost

- Final classifier is a weighted linear combination of all the classifiers
- Classifier h_i gets a weight α_i



- Multiple weak, linear classifiers **combined** to give a strong, nonlinear classifier

Comments

- For AdaBoost, given each model's error $\epsilon_t = 1/2 - \gamma_t$, the training error consistently gets better with rounds

$$\text{train-error}(H_{final}) \leq \exp(-2\sum_{t=1}^{T}\gamma_t^2)$$

- Boosting algorithms can be shown to be minimizing a loss function
  - E.g., AdaBoost has been shown to be minimizing an exponential loss

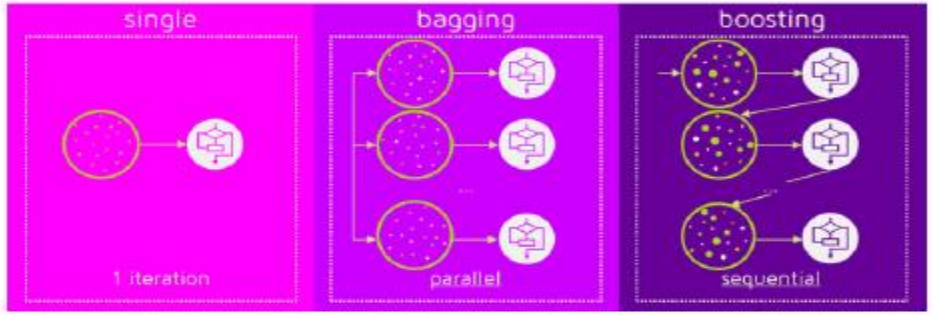
$$\mathcal{L} = \sum_{n=1}^{N} \exp\{-y_n H(\boldsymbol{x}_n)\}$$

where $H(\boldsymbol{x}) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}))$, given weak base classifiers $h_1, \ldots, h_T$

- Boosting in general can perform badly if some examples are outliers

Comparison

- No clear winner; usually depends on the data
- Bagging is computationally more efficient than boosting (note that bagging can train the M models in parallel, boosting can't)

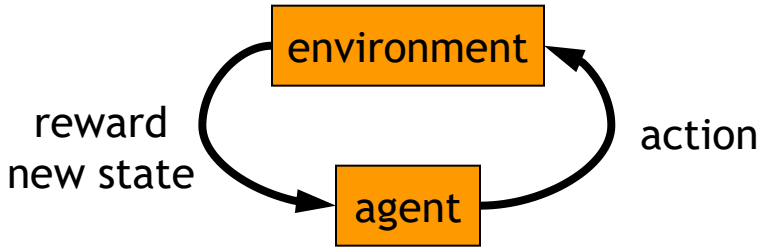


- Both reduce variance (and overfitting) by combining different models
  - The resulting model has higher stability as compared to the individual ones
- Bagging usually can't reduce the bias, boosting can (note that in boosting, the training error steadily decreases)
- Bagging usually performs better than boosting if we don't have a high bias and only want to reduce variance (i.e., if we are overfitting)

Ενισχυτική Μάθηση

Εισαγωγή

- Επίβλεψη μάθησης
  - κατηγοριοποίηση, παλινδρόμηση
- Εκμάθηση χωρίς επίβλεψη
  - συσταδοποίηση
- Ενισχυτική μάθηση
  - πιο γενική από την εποπτευόμενη/μη εποπτευόμενη μάθηση
  - μαθαίνουμε από την αλληλεπίδραση με το περιβάλλον για την επίτευξη ενός στόχου



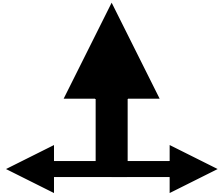
Εισαγωγή

			+1
			-1
START			

actions: UP, DOWN, LEFT, RIGHT

UP

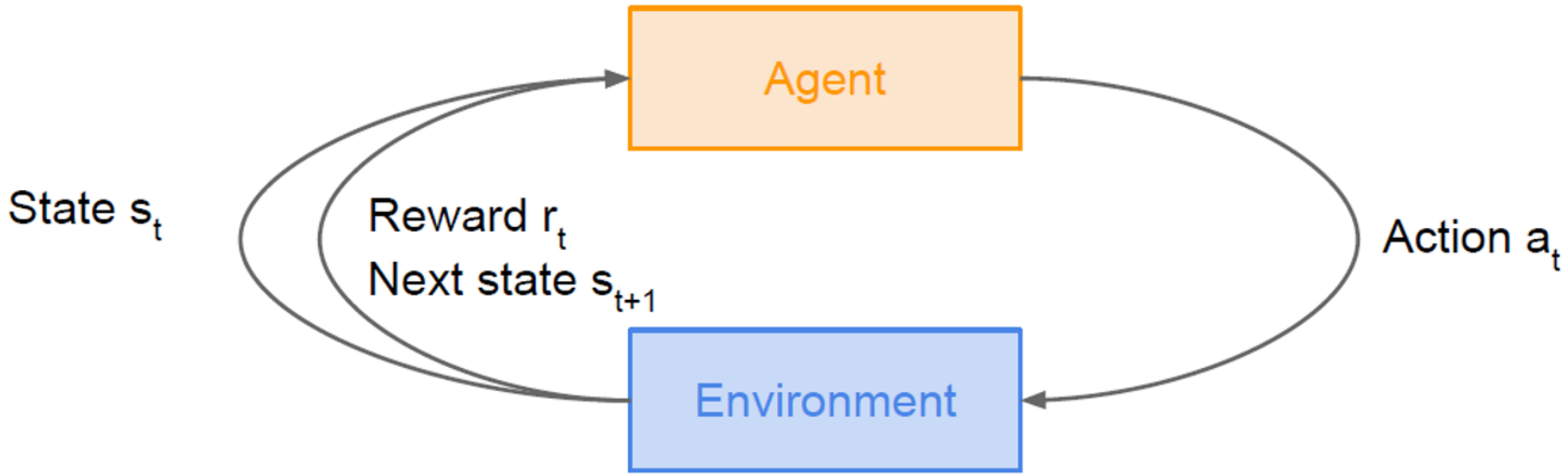
80% move UP
10% move LEFT
10% move RIGHT



- reward +1 at [4,3], -1 at [4,2]
- reward -0.04 for each step

- what's the strategy to achieve max reward?
- what if the actions were deterministic?

Εισαγωγή

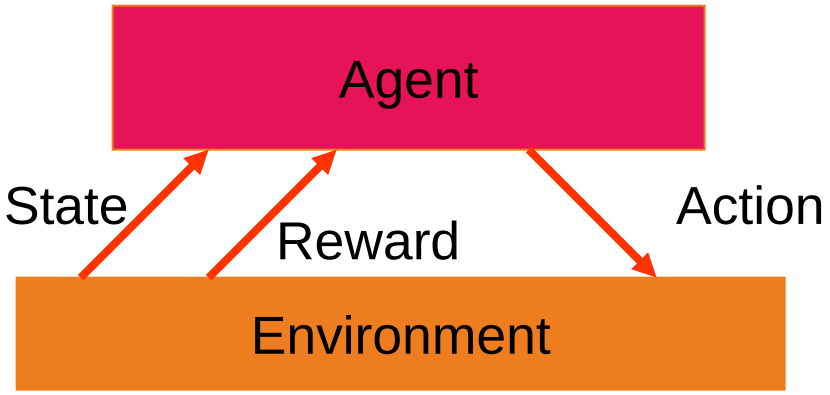

Agent
State s_t
Reward r_t
Next state s_{t+1}
Action a_t
Environment

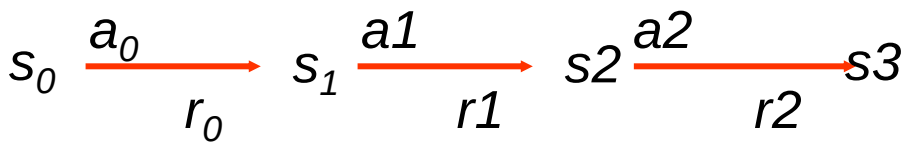
Μοντέλο

- Κάθε αντίληψη(ε) είναι αρκετή για να καθορίσει την κατάσταση (η κατάσταση είναι προσβάσιμη)
- Ο πράκτορας μπορεί να αποσυνθέσει το στοιχείο Ανταμοιβής από μια αντίληψη
- Το καθήκον του πράκτορα: να βρεθεί μια βέλτιστη πολιτική, αντιστοίχιση καταστάσεων σε ενέργειες, που μεγιστοποιούν τη μακροπρόθεσμη μέτρηση της ενίσχυσης
- Σκεφτείτε την ενίσχυση ως ανταμοιβή
- Μπορεί να μοντελοποιηθεί ως μοντέλο MDP!

Μοντέλο

- MDP model <S,T,A,R>





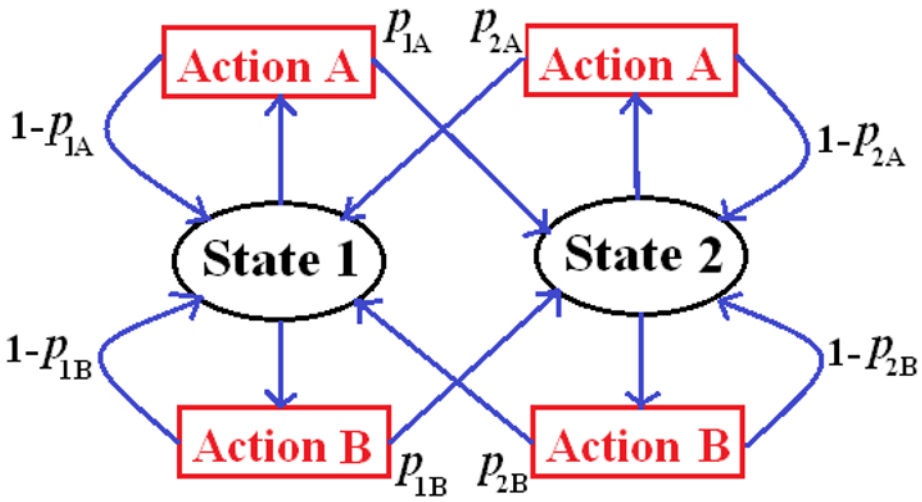
- S– set of states
- A– set of actions
- T(s,a,s') = P(s'|s,a)– the probability of transition from *s* to *s'* given action *a*
- R(s,a)– the expected reward for taking action *a* in state *s*

$$R(s,a) = \sum_{s'} P(s'|s,a) r(s,a,s')$$

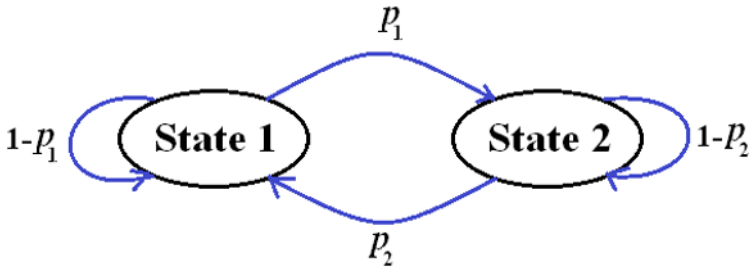
$$R(s,a) = \sum_{s'} T(s,a,s') r(s,a,s')$$

Μοντέλο

Markov Decision Process



Markov Chain



Μοντέλο

- Όμως, δεν γνωρίζουμε τίποτα για το μοντέλο περιβάλλοντος — τη συνάρτηση μετάβασης T(s,a,s')
- Εδώ έρχονται δύο προσεγγίσεις
  - Προσέγγιση βάσει μοντέλου RL:
    - μάθετε το μοντέλο και χρησιμοποιήστε το για να εξαγάγετε τη βέλτιστη πολιτική.
    - π.χ. Προσαρμοστική δυναμική μάθηση (Adaptive Dynamic Learning - ADP).
  - Χωρίς Μοντέλο προσέγγιση RL:
    - εξαγάγετε τη βέλτιστη πολιτική χωρίς να μάθετε το μοντέλο.
    - π.χ. Προσέγγιση χρονικής διαφοράς (Temporal Difference – TD)
    - Το Temporal Difference Learning στοχεύει να προβλέψει έναν συνδυασμό της άμεσης ανταμοιβής και της δικής της πρόβλεψης ανταμοιβής την επόμενη χρονική στιγμή. Στο TD Learning, το σήμα εκπαίδευσης για μια πρόβλεψη είναι μια μελλοντική πρόβλεψη. Αυτή η μέθοδος είναι ένας συνδυασμός της μεθόδου Monte Carlo (MC) και της μεθόδου Δυναμικού Προγραμματισμού (DP).

Μοντέλο

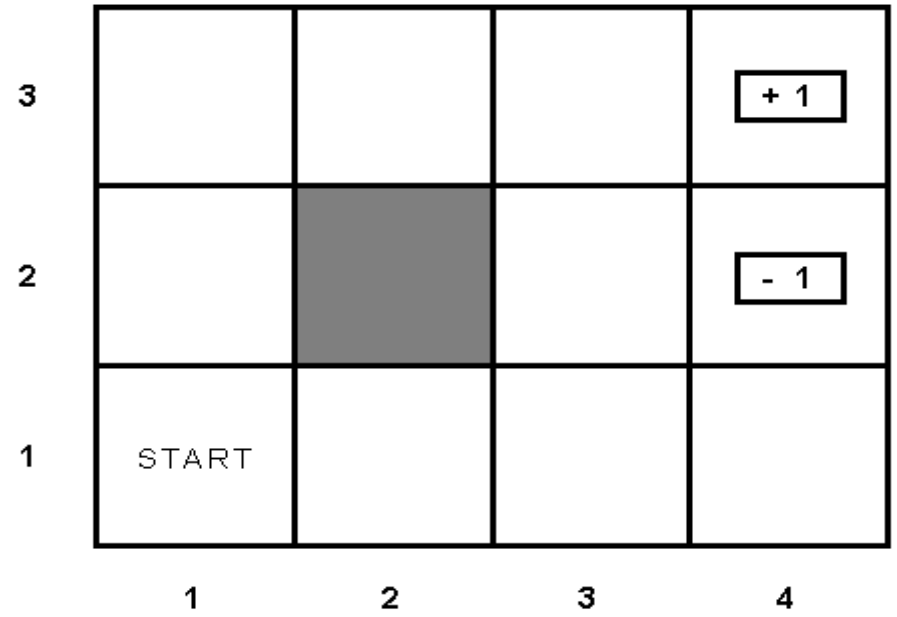
- Παθητική μάθηση
  - Ο πράκτορας υπονοεί ότι παρακολουθεί τον κόσμο να περνάει και προσπαθεί να μάθει τις χρησιμότητα του να βρίσκεται σε διάφορες πολιτείες
- Ενεργητική μάθηση
  - Ο πράκτορας όχι απλώς παρακολουθεί, αλλά και ενεργεί

Μοντέλο

- Ο πράκτορας βλέπει τις ακολουθίες μεταβάσεων κατάστασης και συνδέει τις ανταμοιβές
- Το περιβάλλον δημιουργεί μεταβάσεις καταστάσεων και ο πράκτορας τις αντιλαμβάνεται

  π.χ. (1,1) →(1,2) →(1,3) →(2,3) →(3,3) →(4,3)[**+1**]

  (1,1)→(1,2) →(1,3) →(1,2) →(1,3) →(1,2) →(1,1) →(1,2) →(1,3) →(1,4) →(2,4)[**-1**]
- Βασική ιδέα: ενημέρωση της τιμής χρησιμότητας χρησιμοποιώντας τις δεδομένες ακολουθίες εκπαίδευσης.



Μοντέλο

```
function PASSIVE-RL-AGENT(e) returns an action
  static: U, a table of utility estimates
          N, a table of frequencies for states
          M, a table of transition probabilities from state to state
          percepts, a percept sequence (initially empty)

  add e to percepts
  increment N[STATE[e]]
  U ← UPDATE(U, e, percepts, M, N)
  if TERMINAL?[e] then percepts ← the empty sequence
  return the action Observe
```

Μοντέλο

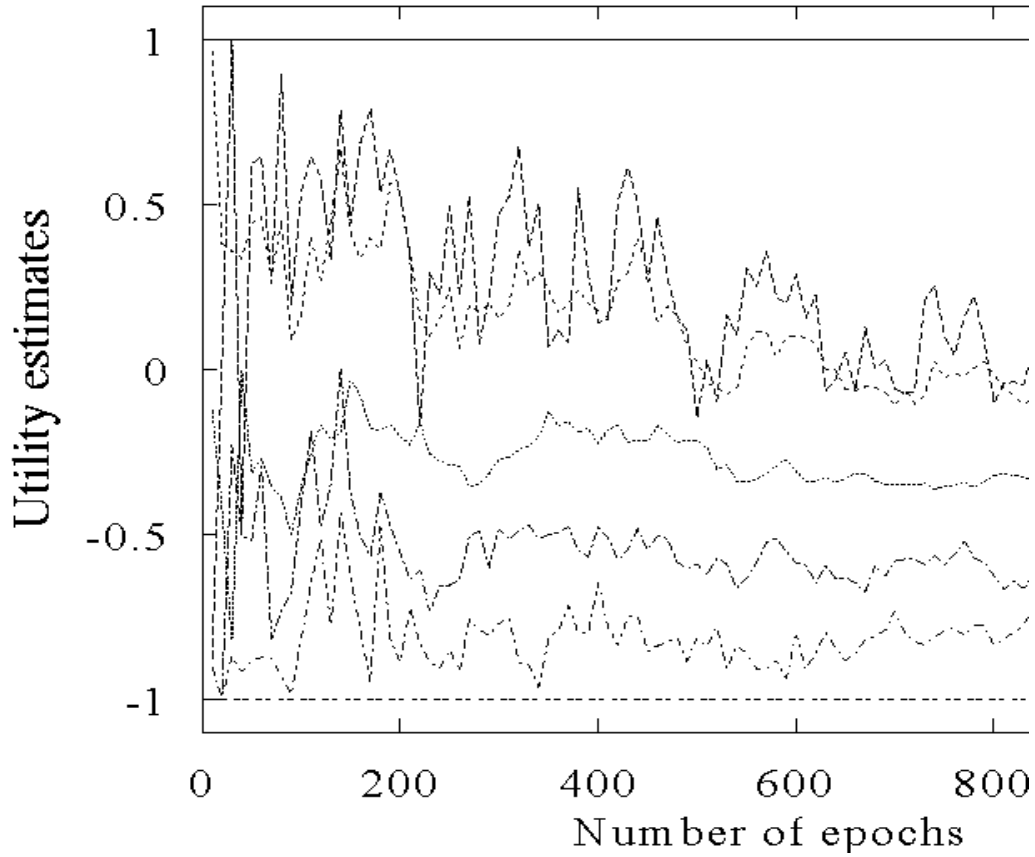
- Επιβράβευση για να πάει σε μια νέα κατάσταση
  - το άθροισμα των ανταμοιβών από αυτήν την κατάσταση μέχρι να επιτευχθεί μια τερματική κατάσταση
- Κλειδί: χρησιμοποιήστε την παρατηρούμενη ανταμοιβή για να πάει η κατάσταση ως άμεση απόδειξη της πραγματικής αναμενόμενης χρησιμότητας αυτής της κατάστασης
- Λειτουργία βοηθητικού προγράμματος μάθησης απευθείας από το παράδειγμα ακολουθίας

Μοντέλο

- Temporal Difference (TD(0)) βασική ιδέα:
  - προσαρμόστε την εκτιμώμενη αξία οφέλους της τρέχουσας κατάστασης με βάση την άμεση ανταμοιβή της και την εκτιμώμενη αξία της επόμενης κατάστασης.
- Ο κανόνας ενημέρωσης οφέλους

$$U(s) = U(s) + \lambda(R(s) + U(s') - U(s))$$

- λ είναι η παράμετρος του ρυθμού εκμάθησης
- Μόνο όταν το λ είναι μια συνάρτηση που μειώνεται καθώς αυξάνεται ο αριθμός των φορών που επισκέφθηκε μια κατάσταση, τότε το U(s) μπορεί να συγκλίνει στη σωστή τιμή.

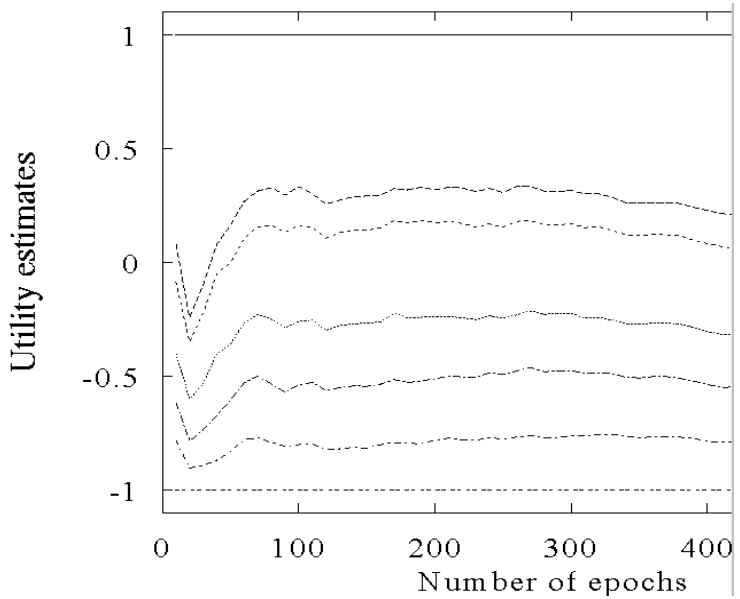


ADP

- Το Adaptive dynamic programming(ADP) είναι διαφορετικό από τη μέθοδο TD
- Το ADP είναι μια προσέγγιση που βασίζεται σε μοντέλο!
- Ο κανόνας ενημέρωσης για παθητική μάθηση

$$U(s) = \sum_{s'} T(s,s')(r(s,s') + \gamma U(s'))$$

- Ωστόσο, σε ένα άγνωστο περιβάλλον, το T δεν δίνεται, ο πράκτορας πρέπει να μάθει ο ίδιος το T από εμπειρίες με το περιβάλλον.



ADP

- Ένας ενεργός πράκτορας πρέπει να εξετάσει
  - τι ενέργειες θα γίνουν;
  - ποια μπορεί να είναι τα αποτελέσματά τους (τόσο για τη μάθηση όσο και για τη λήψη των ανταμοιβών μακροπρόθεσμα);
- Ενημέρωση της εξίσωσης του οφέλους

$$U(s) = \max_a (R(s,a) + \gamma \sum_{s'} T(s,a,s') U(s'))$$

- Κανόνας επιλογής δράσης

$$a = \operatorname*{argmax}_a (R(s,a) + \gamma \sum_{s'} T(s,a,s')\, U(s'))$$

ADP

For each s, initialize *U(s)* , *T(s,a,s')* and *R(s,a)*

Initialize *s* to current state that is perceived

Loop forever

{

Select an action ***a*** and execute it (using current model ***R*** and ***T***) using

$$a = \arg\max_{a}(R(s,a) + \gamma \sum_{s'} T(s,a,s')U(s'))$$

Receive immediate reward ***r*** and observe the new state ***s'***

Using the transition tuple <*s,a,s',r*> to update model ***R*** and ***T*** (see further)

For all the sate ***s***, update ***U(s)*** using the updating rule

$$U(s) = \max_{a}(R(s,a) + \gamma \sum_{s'} T(s,a,s')U(s'))$$

s = s'

}

ADP

- Χρησιμοποιήστε την πλειάδα μετάβασης <s, a, s', r> για να μάθετε T(s,a,s') και R(s,a).
- Αυτό είναι εποπτευόμενη μάθηση!
  - Εφόσον ο πράκτορας μπορεί να πάρει κάθε μετάβαση (s, a, s', r) άμεσα, πάρτε το (s,a)/s' ως παράδειγμα εισόδου/εξόδου της συνάρτησης πιθανότητας μετάβασης T.
  - Διαφορετικές τεχνικές στην εποπτευόμενη μάθηση
  - Χρησιμοποιήστε τα r και T(s,a,s') για να μάθετε R(s,a)

$$R(s,a) = \sum_{s'} T(s,a,s')r$$

ADP

- Πλεονεκτήματα:
  - Ο αλγόριθμος ADP συγκλίνει πολύ πιο γρήγορα από το Temporal Learning.
  - Αυτό συμβαίνει γιατί χρησιμοποιεί τις πληροφορίες από το μοντέλο του περιβάλλοντος.
- Μειονεκτήματα:
  - Δύσκολα για μεγάλο χώρο καταστάσεων
  - Σε κάθε βήμα, ενημερώστε το U για όλες τις καταστάσεις
  - Βελτιώστε το με σάρωση προτεραιοτήτων

Q-Learning

- Ορισμός συνάρτησης Q-value

$$U(s) = \max_a Q(s,a)$$

- Κανόνας ενημέρωσης συνάρτησης τιμής Q

$$U(s) = \max_a (R(s,a) + \gamma \sum_{s'} T(s,a,s') U(s'))$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') U(s')$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

- Βασική ιδέα της μάθησης TD-Q
  - Σε συνδυασμό με προσέγγιση χρονικής διαφοράς
  - Ο κανόνας ενημέρωσης

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

- Κανόνας για να επιλέξετε τη δράση που θα κάνετε

$$a = \arg\max_a Q(s,a)$$

Q-Learning

For each pair (*s, a*), initialize *Q(s,a)*

Observe the current state s

Loop forever

{

Select an action ***a*** and execute it

$$a = \arg\max_{a'} Q(s,a)$$

Receive immediate reward ***r*** and observe the new state ***s'***

Update *Q(s,a)*

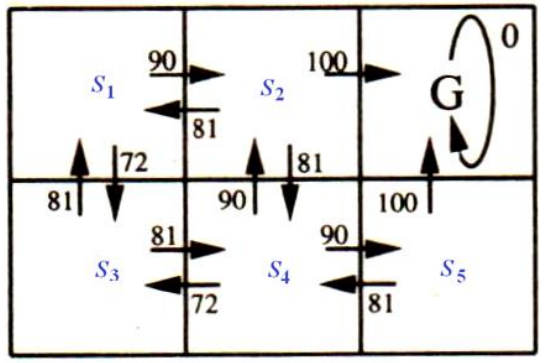
$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

s=s'

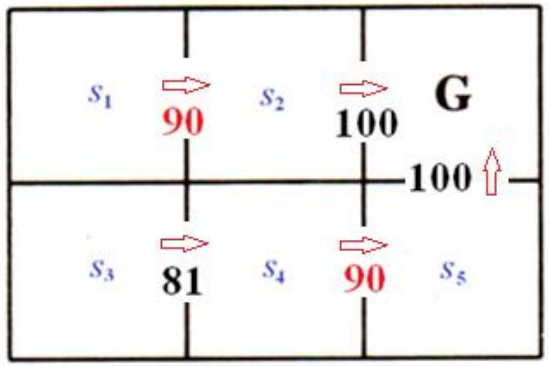
}

Q-Learning

$Q(s,a)$	↑	↓	←	→
s_1	0	72	0	90
s_2	0	81	81	100
s_3	81	0	0	81
s_4	90	0	72	90
s_5	100	0	81	0
G	0	0	0	0



$Q(s, a)$ values



π^*

Exploration

- Μια ενέργεια έχει δύο είδη αποτελεσμάτων
  - Κερδίστε ανταμοιβές στην τρέχουσα πλειάδα εμπειρίας (s,a,s’)
  - Επηρεάστε τις αντιλήψεις που λαμβάνονται, και ως εκ τούτου την ικανότητα του πράκτορα να μαθαίνει

Exploration

- Ανταλλαγή κατά την επιλογή δράσης μεταξύ
  - είναι αμέσως καλό (αντανακλάται στις τρέχουσες εκτιμήσεις χρησιμότητας χρησιμοποιώντας αυτά που μάθαμε)
  - Το μακροπρόθεσμο καλό του (η εξερεύνηση περισσότερων για το περιβάλλον το βοηθά να συμπεριφέρεται βέλτιστα μακροπρόθεσμα)
- Δύο ακραίες προσεγγίσεις
  - Προσέγγιση «παράξενη»: δρα τυχαία, με την ελπίδα ότι τελικά θα εξερευνήσει ολόκληρο το περιβάλλον.
  - «άπληστη» προσέγγιση: δρα για τη μεγιστοποίηση της χρησιμότητάς της χρησιμοποιώντας την τρέχουσα εκτίμηση του μοντέλου
- Ακριβώς όπως ο άνθρωπος στον πραγματικό κόσμο! Οι άνθρωποι πρέπει να αποφασίσουν μεταξύ τους
  - Συνεχίζοντας σε μια άνετη ύπαρξη
  - Ή να χτυπήσετε στο άγνωστο με την ελπίδα να ανακαλύψετε μια νέα και καλύτερη ζωή

Exploration

- Ένα είδος λύσης: ο πράκτορας θα πρέπει να είναι πιο εκκεντρικός όταν έχει ελάχιστη ιδέα για το περιβάλλον και πιο άπληστος όταν έχει ένα μοντέλο που είναι σχεδόν σωστό
  - Σε μια δεδομένη κατάσταση, ο πράκτορας θα πρέπει να δίνει κάποια βαρύτητα σε ενέργειες που δεν έχει δοκιμάσει πολύ συχνά.
  - Ενώ τείνουν να αποφεύγουν ενέργειες που πιστεύεται ότι είναι χαμηλής χρησιμότητας
- Υλοποιείται από τη συνάρτηση εξερεύνησης f(u,n):
  - εκχώρηση υψηλότερης εκτίμησης χρησιμότητας σε σχετικά ανεξερεύνητα ζεύγη καταστάσεων ενεργειών
  - Αλλάξτε τον κανόνα ενημέρωσης της συνάρτησης τιμής σε

$$U^+(s) = \max_a(r(s,a) + \gamma f(\sum_{s'} T(s,a,s')U^+(s'), N(a,s))$$

  - Το U+ υποδηλώνει την αισιόδοξη εκτίμηση της χρησιμότητας

Exploration

- Ένα είδος ορισμού του f(u,n)

$$f(u,n)=\begin{cases} R^{+} & \text{if } n< Ne \\ u & \text{otherwise} \end{cases}$$

- R^{+}είναι μια αισιόδοξη εκτίμηση της καλύτερης δυνατής ανταμοιβής που μπορεί να επιτευχθεί σε οποιαδήποτε κατάσταση
- Ο πράκτορας θα δοκιμάσει κάθε ζευγάρι δράσης-κατάστασης τουλάχιστον Ne φορές
- Ο πράκτορας θα συμπεριφερθεί αρχικά σαν να υπήρχαν υπέροχες ανταμοιβές σκορπισμένες παντού - αισιόδοξος

Exploration

Exploration of unknown states and actions to gather new information

Exploitation of learned states and actions to maximize the cumulative reward

- **ε-greedy search:**

  Explore – with probability ε choose uniformly one action among all possible actions.

  Exploit – with probability 1-ε choose the best action.

  Start with a high ε and gradually decrease it in order initiate exploitation once enough exploration.

Exploration

Choose action *a* according to probability

$$P(a \mid s) = \frac{\exp Q(s,a)}{\sum_{b \in A} \exp Q(s,b)}$$

Move from exploration to exploitation using

$$P(a \mid s) = \frac{\exp\left[Q(s,a)/T\right]}{\sum_{b=1}^{A} \exp\left[Q(s,b)/T\right]}$$

Start with a large T and gradually decrease it.

T large, $P(a \mid s) \approx 1/A$ (constant) $\Rightarrow$exploration

T small, better actions → exploitation.