Training examples

large	is_mammal	has_claws	can_fly	can_bark	has_tail	Label
1	0	1	1	0	0	-
0	1	1	1	0	0	-
0	1	1	0	1	1	+
1	1	1	0	1	1	+

A Learning Algorithm

1. Initialize initial hypothesis $h \leftarrow A_1 \wedge \neg A_1 \ldots A_n \wedge \neg A_n$
2. For each positive example $e \in S$ **do**:
   - if the i^{th} boolean attribute $A_i = 0$ in the example, remove A_i from h, otherwise remove $\neg A_i$.
3. Output h as the hypothesis that best approximates the target concept.

Properties of this learning algorithm

Biases:

- **Representational bias**: concepts are describable by purely conjunctive expressions.
- **Algorithmic bias**: keeps track of only the most specific hypothesis consistent with the data.

Analysis of this learning algorithm

- Size of hypothesis space $|F| = 3^n$ (exponential).
- Let $n = 100$. Then $F \approx 10^{47}$.
- How many examples are needed for the algorithm to learn the "dog" concept?
- This algorithm will never converge quickly unless some additional assumptions are made (e.g. examples are drawn from a fixed (unknown) distribution).
- Surprisingly, we can then show that this algorithm can reliably find high accuracy approximations in polynomial time, given m examples, where

$$m = \frac{1}{\epsilon}(n(ln(3) + ln(\frac{1}{\delta}))$$

Mistake-Bounded Model of Concept Learning

- Unlike before, each time the learner receives a training example, it must *predict* the label (positive or negative), before being given the right answer.
- Learner is evaluated in terms of the number of mistakes it makes before converging to the right hypothesis.
- Useful model of online learning (e.g. for web-based datamining).
- **Problem:** How many mistakes will our concept learning algorithm make, before converging to the right hypothesis?
- **Answer:** $n + 1$ (where n is the number of attributes)

Design of a Learning System

Choose

- **Task** (robot navigation, weather prediction,...)
- **Training experience** (scalar feedback, labeled examples,...)
- **Target function** (state $\rightarrow$ value, feature vector $\rightarrow$ label,...)
- **Function representation** (neural nets, decision trees, nearest neighbor,...)
- **Learning method** (backpropagation, c4.5, kernel regression,...)

Example: Weather Prediction

- **Task:** Predict weather in East Lansing next Saturday (Prob(snow)).
- **Training experience:** Database of measurements and final outcome.
- **Target function:** $f : (x_1, \ldots, x_n) \rightarrow [0, 1]$
- **Function representation:** $f(x) = \sum_{i=1}^{n} w(i)x(i)$
- **Learning method:** LMS (Delta rule, Adaline)

Learning Method

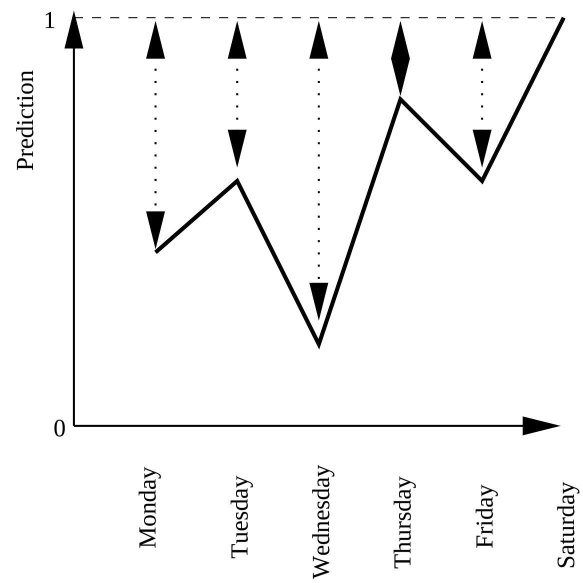
- Let P_t be the prediction on day t and z be the final outcome on Saturday.
- Generalized delta rule:

$$\Delta(w_t) = \alpha(z - P_t)\nabla_w P_t$$

- For linear approximators:

$$\Delta(w_t)(i) = \alpha(z - \sum_{k=1}^{n} w_t(k)x_t(k))x_t(i)$$

Weather prediction: Supervised learning



Sequential Prediction/Decision Problems

- Weather prediction
- Stock market
- Game playing
- Robot navigation
- Manufacturing/scheduling

Weather Prediction revisited

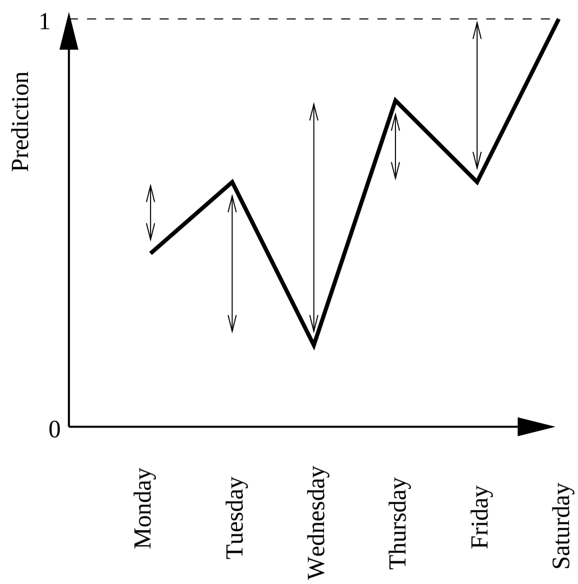
- Error $e_t = z - P_t$ (e.g. Saturday's outcome - Monday's prediction)
- Problem: Cannot learn until final outcome is known!
- How can an agent learn from online experience?
- **Key idea:** Temporal difference learning (Sutton)
- Reexpress error as sum of differences between temporally successive predictions.

Temporal Difference Learning

Error =	Tuesday's prediction - Monday's prediction
+	Wednesday's prediction - Tuesday's prediction
+	Thursday's prediction - Wednesday's prediction
+	Friday's prediction - Thursday's prediction
+	Saturday's outcome - Friday's prediction

- TD(0): $\Delta(w_t) = \alpha(P_{t+1} - P_t)\nabla_w P_t$
- TD(1): $\Delta(w_t) = \alpha(P_{t+1} - P_t)\sum_{k=1}^{t} \nabla_w P_k$
- TD(λ): $\Delta(w_t) = \alpha(P_{t+1} - P_t)\sum_{k=1}^{t} \lambda^{t-k}\nabla_w P_k$

Weather Prediction using TD learning

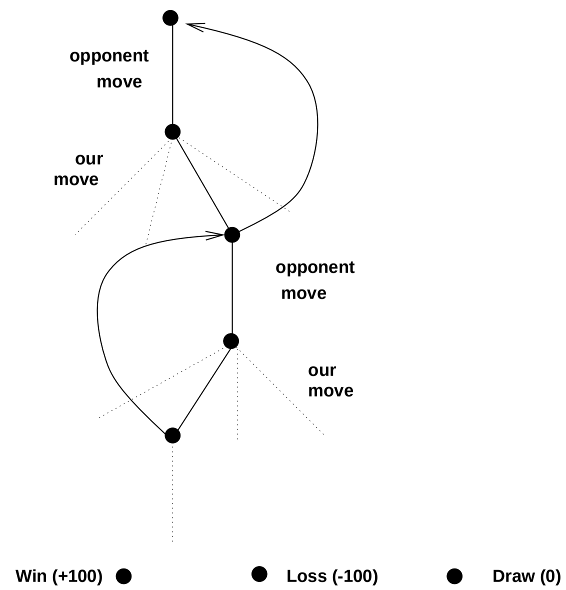


Reinforcement Learning

- **Reward:** Scalar feedback
- **Policy:** what do I do in this state?
- **Value function:** How good is this state (assuming I follow a fixed policy)?
- **Model**: what happens if I do this action?
- **Key idea:** Learn the optimal value function $V*$
  - Model-free (TD(0) or Q-learning)
  - Model-based (Real-time dynamic programming)

Backing Up Future Evaluations

(Samuel)



Other Function Approximators

Decision Trees

- Choose some feature to split on (e.g. humidity)
- Choose some value to split on (e.g. 80%)
- Partition all instances into $\leq 80\%$ and $> 80\%$.
- Repeat until class impurity is minimal

Nearest Neighbor

- Store all instances
- Given a new feature vector, determine "closest" instances using some distance metric
- Assign new vector the class label of the majority of the closest instances

Issues in Choosing Approximators

- Generality of learning algorithm
- Convergence
- Noise immunity and robustness
- Speed
- Incrementality