

Online Learning (PWEA, Greedy)

Lecturer: Kris Kitani

Scribes: Akshay Dharmavaram, Zongyue Zhao

1 Review

1.1 Introduction

In the last lecture, we briefly went over the course logistics, then broke down the problem of robot learning along three axes: (1) whether the learner is exposed to all possible situations (exhaustive vs. sampled); (2) whether the learner receives feedback for all possible actions (instructive vs. evaluative); (3) whether the outcome of the current action affects the next action (sequential vs. one-shot) [3]. In this lecture, we first introduced the concept of online learning, then briefly discussed Prediction with Expert Advice (PWEA). We also analyzed the theoretical upper bound of the number of mistakes a greedy PWEA learner could make, under the assumption of realizability.

1.2 Robot Learning Problems

Robotic systems are used in various situations. Before diving into the specific techniques best used in each situation, we need to characterize the robot learning problem [2]. As demonstrated in Figure 1, we differentiate robot learning problems with the characteristics of feedback the learner receives from the environment.

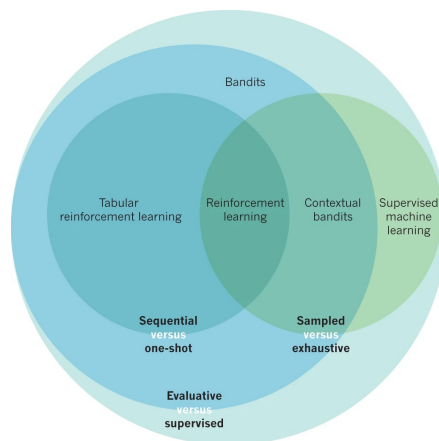


Figure 1: Decisions of machine-learning feedback [3]

The first dimension we examine is whether the learner is exposed to all possible situations. Consider the difference between plain memorization and supervised learning. In the first case, all valid key-solution pairs are presented to the system during training, whereas in the latter case, the training set is merely a sampled subset of all possible situations, and the learner is expected to perform on instances that may never appear during training.

Another example is the difference between Tabular RL and Approximate RL. In tabular RL, because the state and action spaces are small enough for the value functions to be represented in arrays, convergence to optimum is often guaranteed, provided enough occasions of traversals. On the other hand, in approximate RL, the state space's size is either infinite or beyond computational limits. As such, the majority of potential states are not to be examined in training, and those in the training set are not likely to reappear during testing. Hence, in this case, we need to focus on generalization: to utilize feedback from a limited subset of the state space for applications on a broader subset [6].

Mathematically speaking, in Exhaustive problems the learner is given all valid combinations of the states and labels:

$$D_{train} = \{(s, a) \mid \forall s \in \mathcal{S}, \forall a \in \mathcal{A}, (s, a) \text{ valid}\}, D_{test} \subseteq D_{train} \quad (1)$$

In Sampled feedback problems, the learner is only allowed access to samples drawn from the true distribution:

$$D_{train} = \{(s_i, a_i) \mid s_i, a_i \sim \mathcal{D}(s, a), i = 1, \dots, N\}, D_{test} \not\subseteq D_{train} \quad (2)$$

The second dimension is whether we have information for all possible actions (labels) given an arbitrary state. If so, the feedback is said to be instructive (supervised), as opposed to evaluative feedback. In supervised learning, we have a ground truth label for each encountered state. This is equivalent to having a Boolean mask (which yields true only at Ground Truth) on the action (label) space for each state:

$$\forall (s_i, a_i) \in D_{train}, \forall a \in \mathcal{A}(s_i) : \text{Receive Mask}[s_i, a] = (a == a_i) \quad (3)$$

The instructive mask here presents which $a \in A(s)$ is the best.

On the other hand, in RL we rely on either the reward received from executing the current action, or the return (cumulative rewards) corresponding to a sequence of state-actions. At each step, we have at most an numerical value only for the action executed, which is insufficient to determine whether this specific action is the best among the action space.

$$\begin{aligned} \forall (s_i, a_i) \in D_{train} : \text{Receive } R(s_i, a_i) \sim P(r \mid s_i, a_i) \quad (\text{No } R(s_i, \forall a \in A(s_i))) \\ \text{or} \\ \forall \zeta_i \in D_{train} : \text{Receive } G(\zeta_i) \sim P(g \mid \zeta_i) \quad \text{where } \zeta_i = \{(s_t, a_t)\}_{iT+1:(i+1)T} \quad (\text{No } G(\forall \zeta)) \end{aligned} \quad (4)$$

In either case, the reward (or return) only presents how well the action (or sequence) is, but not whether it is the best or the worst. Such a phenomenon demands exploration in search of better actions [6].

The third dimension is whether the outcome of action affects the next state. Consider a problem where the learner repeatedly faces k different actions. After choosing an action, the learner receives a reward that follows a probability distribution conditional to the action. If the probability $R(a)$ is fixed, the problem is called the k -arm bandit. If the probability is non-stationary, i.e., $R \sim R(s, a)$ but $S \perp\!\!\!\perp A$, the problem is called contextual k -arm bandit. Either way, the environment (and the state drawn) must be independent of the actions.

The characteristic described above is called one-shot. On the other hand, the sequential case refers to when states are indeed affected by previous actions. In this case, we need to learn not only the reward distribution $R(s, a)$, but also the system dynamics $P(s' \mid s, a)$ [6].

2 Summary

2.1 Online Learning

Online learning algorithms are primarily used to tackle predictive or sequential decision-making tasks that require learning from a continuous stream of data instances. The pseudo-code for the family of online learning algorithms is summarized in Algorithm 1 below. The update equations on line 7 and 8 state that θ is updated based on how similar the learner’s predictions are to the target distribution.

Algorithm 1 Online Learning

```
1: Initialize Nature                                ▷ black box model that emulates the target distribution
2: Initialize the parameters ( $\theta$ ) of the Online Learner
3: for  $trial = 1, \dots$ , Number of Trials do
4:   for  $t = 1, \dots$ , Sequence Length do
5:      $x^{(t)}, y^{(t)} \leftarrow \text{SAMPLE NATURE}(t)$                                 ▷ Obtain the target input-output pair
6:      $\hat{y}^{(t)} \leftarrow \text{ONLINE LEARNER}(x^{(t)}|\theta)$     ▷ Estimate the output conditioned on the  $x^{(t)}$  and  $\theta$ 
7:      $l \leftarrow \text{LOSS FUNCTION}(y^{(t)}, \hat{y}^{(t)})$         ▷ Calculate the loss of the model
8:      $\theta \leftarrow \text{UPDATE PARAMETERS}(l, \theta)$         ▷ Update the online learner
9:   end for
10: end for
```

The family of online learning algorithms share the following properties:

- The algorithm must make predictions and learn on-the-fly
- Sequence streams are assumed to be uncorrelated
- Require immediate feedback in terms of a loss or a reward conditioned on the recent prediction
- Requires access to the entire state or the sampled state

Moving ahead, the domain of online learning encompasses algorithms such as Predictions with Expert Advice [7], Bandits[8], and Contextual Bandits [4], which all attempt to minimize the loss of the model’s prediction against the known reference provided by the Environment. We can formally interpret this as an optimization problem that addresses the following cumulative loss:

$$\underset{\hat{y}}{\operatorname{argmin}} = \sum_{t=1}^T l(y^{(t)}, \hat{y}^{(t)}) \quad (5)$$

2.2 Online Learning vs. Supervised Learning

Online learning differs from supervised learning mainly in terms of the ability of the model to access data. In a supervised setting [1], the algorithm attempts to approximate a stationary data-distribution, and usually involves a training and testing regime. Whereas in the online setting there is no specific training or testing datasets or stages. The online learning algorithms usually

involve incrementally learning the distribution using the sequences of input and output pairs. Due to the nature of the data collection procedure, online learning algorithms tend to be more agile, and can adapt quickly to shifts in the data-distribution, which can be stochastic, deterministic, or adversarial. Thus, online learning algorithms prove to be the preferred choice for robotic learning algorithms.

As both supervised learning algorithms, and online learning algorithms, learn models that map a given input space to a desired approximation of the output distribution, selecting the appropriate algorithm boils down to identifying the appropriate characteristics that define the problem at hand. The main design choices that need to be considered when deciding between supervised and online learning algorithms are:

- Is the data distribution changing over time?
- Is the evaluative or instructive feedback incremental?
- Is the training dataset too large to process all at once?

2.3 Prediction With Expert Advice (PWEA)

To motivate the use of PWEA, we will present a learning problem involving gamblers and racehorses. In this motivating example, we have a set of expert gamblers who routinely bet on horses. As an outsider, we intend on identifying the best expert from the set of experts and mimic their bets. We start by weighing all the experts equally. Next, we update our weights based on the accuracy of each expert. We penalize the experts who incorrectly predicted the outcome of the horse race, by reducing their share of weights to zero. Then, we reinitialize our distribution of weights by equally distributing the weights among experts who correctly predicted the outcome of the race. This process is repeated iteratively until we converge to the subset of the best experts.

The pseudo-code for the PWEA algorithm is summarized in Algorithm 2 below. The update equations on line 8 state that the set E is updated based on the performance of each expert. Experts that perform poorly are removed from the set, and experts that succeed, remain in the set under consideration. The performance is evaluated by an external black-box entity, represented as Nature in Algorithm 2.

Algorithm 2 Predictions With Expert Advice

- | | |
|---|--|
| 1: Initialize the set of experts E | ▷ these are similar to classes |
| 2: Initialize Nature | ▷ black box model that evaluates each expert |
| 3: for $trial = 1, \dots, \text{Number of trials}$ do | |
| 4: $y_E^{(t)} \leftarrow \text{SAMPLE NATURE}(t)$ | ▷ Evaluate the performance |
| 5: $E \leftarrow \{e y_e = \text{success}\}$ | ▷ Update E conditioned on the performance of each expert |
| 6: end for | |
-

PWEA has the characteristics of an online learning algorithm. It is suitable for data streams that are one-shot in nature. This ascertains that the individual data streams are uncorrelated and do not affect each other. Thus, the order of the dropping of the experts is independent with respect to the final convergence of the algorithm. Thus, the final result is independent of the stochastic ordering of

the streams. PWEA is also instructive in terms of the feedback provided to the learning algorithm. Thus, the online learner receives the fully observable loss based on its predictions. This translates to obtaining the complete information required to evaluate if the expert was indeed right or wrong. The notion of experts in the PWEA framework can be mapped to classes in the supervised learning setting. Finally, the PWEA algorithm is exhaustive in nature, as we have access to the entire action space, which is finite in size. Thus, it is theoretically possible to exhaustively search the action space. However, it should be noted that the action space is exponential with respect to the number of experts.

2.4 Greedy/Consistent Algorithm

The Consistent (Greedy) Algorithm is similar to the PWEA mentioned in the last section; however, the expert selection is slightly modified. The Consistent algorithm does not maintain a set, like the PWEA. Instead, the algorithm maintains a stack, and pop's experts from the stack based on their performance. The algorithm will converge once the best expert has been encountered. It should be noted that the greedy algorithm need not investigate every expert nor empty the stack to converge upon the best expert. Assuming a *best expert* exists, then it will take a maximum of N trials to identify the expert, where N is the number of experts. One drawback of this method is that it can only identify at the most one expert. Thus, if the set of *best experts* comprises of more than one element, this method will fail to identify all experts in the desired set.

Before formally presenting the greedy algorithm, we will introduce the notation required to mathematically ground the algorithm. We define the instance domain χ as the set of all possible outcomes. Intuitively, this can be thought of as the set of all possible *advice* expressed by each expert. We denote the advice of all the experts at a given timestamp t , as a vector $x^{(t)}$, which has the dimensionality based on the size of the number of experts. Next, we define the target domain \mathcal{Y} , which is the space of all outcome values for each expert. We denote the outcome of all the experts at a given timestamp t , as a vector $y^{(t)}$, which has the dimensionality based on the size of the number of experts. Finally, we define the hypothesis class \mathcal{H} , as the set of all selector or indicator functions. Intuitively, this can be thought of as a way to poll a given expert. We can write the relationship between vectors in these three spaces as follows:

$$\hat{y} = h_e(x) = x_e \quad (6)$$

We summarize the notation below:

- $x^{(t)} \in \chi \subset \mathcal{R}^E$
 - χ is the instance domain
 - $x^{(t)}$ is an E -dimensional vector of expert advice
 - $x_e^{(t)}$ is the advice of the expert e at trial t
- $y^{(t)} \in \mathcal{Y} = \{0, 1\}$
 - \mathcal{Y} is the target domain
 - $y_e^{(t)}$ is the outcome of expert e at trial t
- $h \in \mathcal{H} = \{h_1, \dots, h_E\}$

- \mathcal{H} is the hypothesis class
- h is the expert selector function
- h_e is a function that is defined as $\hat{y} = h_e(x) = x_e$

The pseudo-code for the greedy algorithm is summarized in Algorithm 3 below. The update equations on line 7 state that the version space $V^{(t)}$ is updated based on the performance of each expert. Experts that perform poorly are removed, and experts that succeed, remain under consideration.

Algorithm 3 Greedy Algorithm

```

1: Initialize  $V^{(1)} = \mathcal{H}$                                 ▷ Initialize the version space to store all hypothesis
2: for  $t = 1, \dots$ , Number of Trials do
3:    $\text{RECEIVE}(x^{(t)})$                                      ▷ Obtain the expert advice
4:    $h = \text{FIRSTELEMENT}(V^{(t)})$                            ▷ Greedy selection & prediction (pop from stack)
5:    $\hat{y}^{(t)} = h(x^{(t)})$ 
6:    $\text{RECEIVE}(y^{(t)})$                                      ▷ Obtain true outcome
7:    $V^{(t+1)} = \{h \in V^{(t)} : h(x^{(t)}) = y^{(t)}\}$       ▷ Update the consistent hypothesis
8: end for

```

The greedy algorithm delineated in Algorithm 3, details the process of obtaining the best expert. Initially, we initialize the version space to the whole hypothesis space. This can be justified as we don't know which hypothesis is correct, and thus intermittently hold the belief that all possible hypotheses are potential solutions. Next, we iterate over the different trails, and first receive the different expert advice by polling our experts. Next, we select the first expert based on the initialization of the stack. As the algorithm is greedy, we are only concerned with the element on the top of the stack. Next, we use this selection function to obtain our estimate for the outcome. Finally, we obtain our true outcome, and update our version space using the newly acquired information. This update is performed by only retaining the set of hypotheses that are in agreement with our most recent outcome.

The greedy algorithm has a fair set of assumptions, specifically in terms of realizability [5]. We assume that there exists a hypothesis h^* that maps $\chi \rightarrow \mathcal{Y}$.

It is worthwhile noting that the consistent algorithm makes mistakes before identifying the perfect hypothesis. As demonstrated in Fig 2, these mistakes would result in a loss compared to the case where we knew the perfect hypothesis in hindsight. In the next section, the upper bound of the number of such mistakes is discussed.

2.5 Analysis of the Consistent Algorithm

Assume realizability (the perfect hypothesis is in the hypothesis class); it is known that the version space will converge to the set only containing perfect hypotheses, after enough mistakes are made. The question is, is there an upper bound for the number of mistakes the algorithm could make, before it eliminates all imperfect hypotheses?

Theorem 1. *Let $p^{(t)} \in \{1, 0\}$ denote whether the consistent algorithm makes a mistake at an arbitrary timestamp t . Let $M_c(\mathcal{H}) \doteq \sum_{\tau=0}^{t-1} p^{(\tau)}$ represent the total number of mistakes made by the algorithm. Then, $M_c(\mathcal{H}) \leq |\mathcal{H}| - 1$.*




	Larry	Curly	Moe	winner	winnings
					
	1	0	0	0	\$8K
	\$4K	\$4K	\$4K		
	1	1	1	1	\$12K
		\$6K	\$6K		
	1	1	0	0	\$6K
		\$6K	\$6K		
In hindsight, cash in hand would have been:	\$12K	\$24K	\$36K		\$26K
	Larry	Curly	Moe		Our algorithm

Figure 2: Loss in Winnings

Proof. At any timestamp τ , the following relationship holds true:

$$|V^{(\tau)}| \leq |V^{(\tau-1)}| - p^{(\tau-1)} \quad (7)$$

because the algorithm removes at least one (the first) hypothesis from the version space after a mistake was made.

Taking the summation on both sides of the equation 7 from timestamp $\tau = 1$ to any t yields:

$$\forall t : \sum_{\tau=1}^t |V^{(\tau)}| \leq \sum_{\tau=0}^{t-1} |V^{(\tau)}| - \sum_{\tau=0}^{t-1} p^{(\tau)} \quad (8)$$

With basic algebra, equation 8 evaluates to:

$$\forall t : |V^{(t)}| \leq |V^{(0)}| - \sum_{\tau=0}^{t-1} p^{(\tau)} = |\mathcal{H}| - \sum_{\tau=0}^{t-1} p^{(\tau)} \quad (9)$$

Note that the initial version space is identical to the entire hypothesis space \mathcal{H} .

Under the assumption of realizability, it is guaranteed that the version space always contains a perfect hypothesis:

$$\forall t : |V^{(t)}| \geq 1 \quad (10)$$

Combining equation 9 and 10, we bound the size of the version space:

$$\forall t : 1 \leq |V^{(t)}| \leq |\mathcal{H}| - \sum_{\tau=0}^{t-1} p^{(\tau)} \quad (11)$$

Substitute in the notation of $M_c(\mathcal{H})$, the upper bound of total mistakes made by the consistent algorithm is:

$$M_c(\mathcal{H}) \leq |\mathcal{H}| - 1 \quad (12)$$

□

From the derivation above, we conclude the common strategy for bound derivations.

- Define 'potential' function (the size of the version space $|V|$).
- Upper bound the potential function.
- Lower bound the potential functions.
- Combine the bounds.
- Use algebra or approximation to get the performance bound.

It is worth mentioning that online learning theory focuses on the theoretical bounds of algorithm performance. Such guarantees are often vital to the reliability demands in robotic systems.

References

- [1] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168, 2006.
- [2] J. H. Connell and S. Mahadevan. *Robot Learning*. Springer Science & Business Media, Dec. 2012. Google-Books-ID: 11DVBwAAQBAJ.
- [3] M. L. Littman. Reinforcement learning improves behaviour from evaluative feedback. *Nature*, 521(7553):445–451, 2015.
- [4] T. Lu, D. Pál, and M. Pál. Contextual multi-armed bandits. In *Proceedings of the Thirteenth international conference on Artificial Intelligence and Statistics*, pages 485–492. JMLR Workshop and Conference Proceedings, 2010.
- [5] Y. Ohta, H. Maeda, and S. Kodama. Reachability, observability, and realizability of continuous-time positive systems. *SIAM journal on control and optimization*, 22(2):171–180, 1984.
- [6] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass, 1998.
- [7] V. Vovk. A game of prediction with expert advice. *Journal of Computer and System Sciences*, 56(2):153–173, 1998.
- [8] P. Whittle. Multi-armed bandits and the gittins index. *Journal of the Royal Statistical Society: Series B (Methodological)*, 42(2):143–149, 1980.

3 Appendix

We found an image that helps illustrate weighting in a general PWEA:

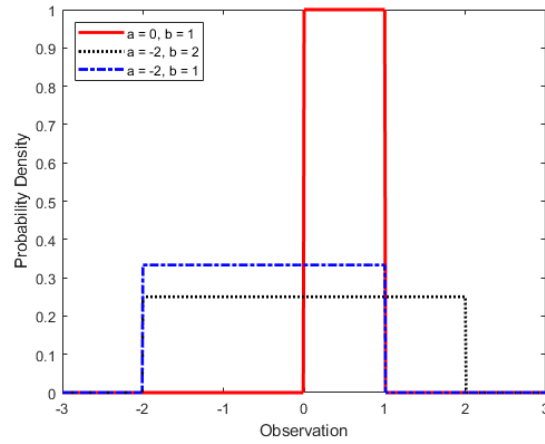


Figure 3: Weighting in PWEA.

Figure 3 above shows the iterative concentration of weights by removing under-performing experts. We start with the dotted black distribution, re-weight our belief to obtain the dotted blue distribution, and finally re-weight again to converge on the red distribution. The red distribution illustrates the convergence and identification of the best expert from the set of experts.