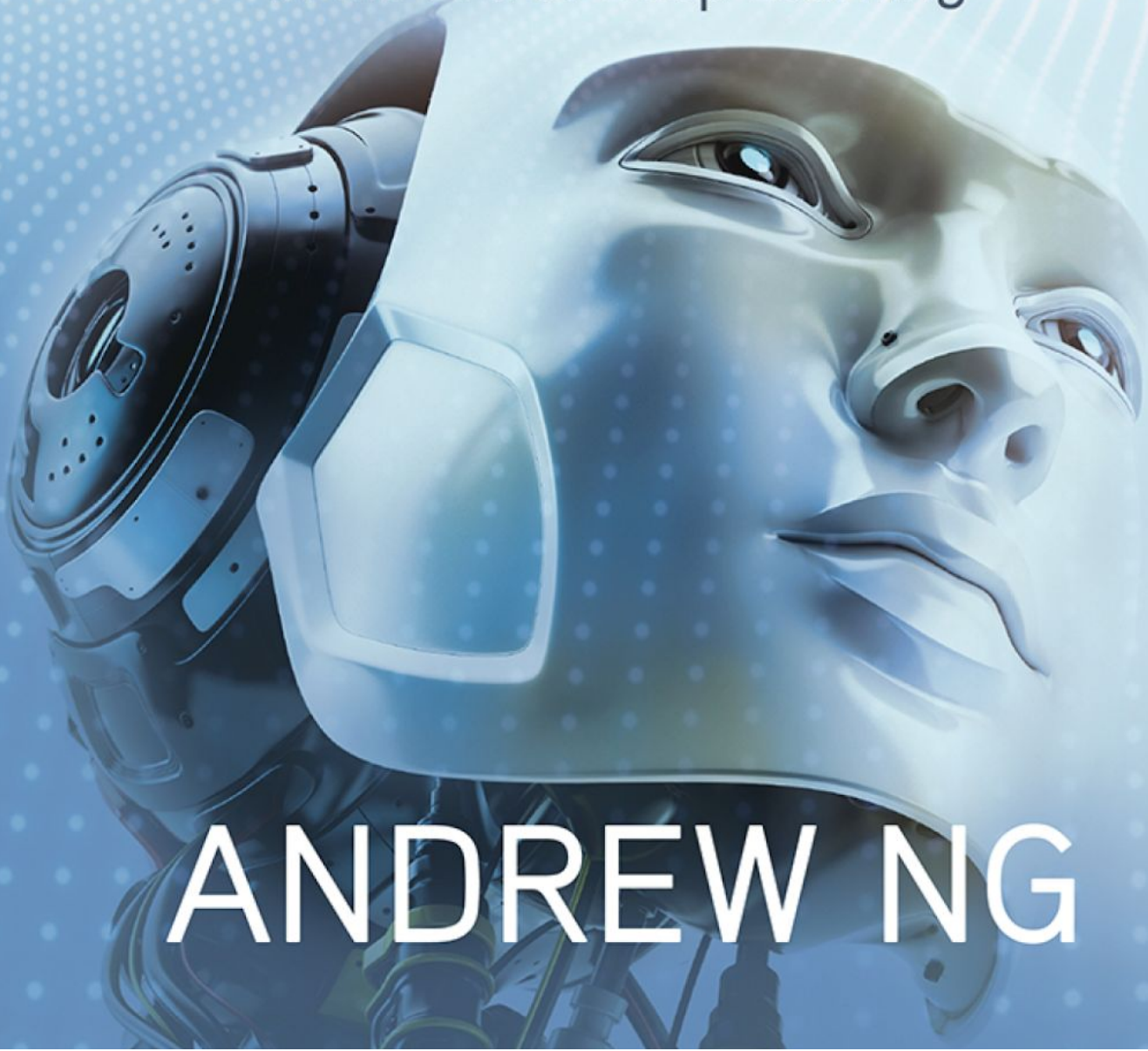
Draft Version

MACHINE LEARNING YEARNING

Technical Strategy for AI Engineers, In the Era of Deep Learning



ANDREW NG

deeplearning.ai

Machine Learning Yearning is a
deeplearning.ai project.

Debugging inference algorithms

44 The Optimization Verification test

Suppose you are building a speech recognition system. Your system works by inputting an audio clip A, and computing some $\text{Score}_A(S)$ for each possible output sentence S. For example, you might try to estimate $\text{Score}_A(S) = P(S|A)$, the probability that the correct output transcription is the sentence S, given that the input audio was A.

Given a way to compute $\text{Score}_A(S)$, you still have to find the English sentence S that maximizes it:

$$\text{Output} = \arg \max_S \text{Score}_A(S)$$

How do you compute the "arg max" above? If the English language has 50,000 words, then there are $(50{,}000)^N$ possible sentences of length N—far too many to exhaustively enumerate. So, you need to apply an approximate search algorithm, to try to find the value of S that optimizes (maximizes) $\text{Score}_A(S)$. One example search algorithm is "beam search," which keeps only K top candidates during the search process. (For the purposes of this chapter, you don't need to understand the details of beam search.) Algorithms like this are not guaranteed to find the value of S that maximizes $\text{Score}_A(S)$.

Suppose that an audio clip A records someone saying "I love machine learning." But instead of outputting the correct transcription, your system outputs the incorrect "I love robots." There are now two possibilities for what went wrong:

1. **Search algorithm problem**. The approximate search algorithm (beam search) failed to find the value of S that maximizes $\text{Score}_A(S)$.
2. **Objective (scoring function) problem.** Our estimates for $\text{Score}_A(S) = P(S|A)$ were inaccurate. In particular, our choice of $\text{Score}_A(S)$ failed to recognize that "I love machine learning" is the correct transcription.

Depending on which of these was the cause of the failure, you should prioritize your efforts very differently. If #1 was the problem, you should work on improving the search algorithm. If #2 was the problem, you should work on the learning algorithm that estimates $\text{Score}_A(S)$.

Facing this situation, some researchers will randomly decide to work on the search algorithm; others will randomly work on a better way to learn values for $\text{Score}_A(S)$. But unless you know which of these is the underlying cause of the error, your efforts could be wasted. How can you decide more systematically what to work on?

Let S_{out} be the output transcription ("I love robots"). Let S* be the correct transcription ("I love machine learning"). In order to understand whether #1 or #2 above is the problem, you can perform the **Optimization Verification test**: First, compute $Score_A(S^*)$ and $Score_A(S_{out})$. Then check whether $Score_A(S^*) > Score_A(S_{out})$. There are two possibilities:

Case 1: $Score_A(S^*) > Score_A(S_{out})$

In this case, your learning algorithm has correctly given S* a higher score than S_{out}. Nevertheless, our approximate search algorithm chose S_{out} rather than S*. This tells you that your approximate search algorithm is failing to choose the value of S that maximizes $Score_A(S)$. In this case, the Optimization Verification test tells you that you have a search algorithm problem and should focus on that. For example, you could try increasing the beam width of beam search.

Case 2: $Score_A(S^*) \leq Score_A(S_{out})$

In this case, you know that the way you're computing $Score_A(.)$ is at fault: It is failing to give a strictly higher score to the correct output S^* than the incorrect S_{out}. The Optimization Verification test tells you that you have an objective (scoring) function problem. Thus, you should focus on improving how you learn or approximate $Score_A(S)$ for different sentences S.

Our discussion has focused on a single example. To apply the Optimization Verification test in practice, you should examine the errors in your dev set. For each error, you would test whether $Score_A(S^*) > Score_A(S_{out})$. Each dev example for which this inequality holds will get marked as an error caused by the optimization algorithm. Each example for which this does not hold ($Score_A(S^*) \leq Score_A(S_{out})$) gets counted as a mistake due to the way you're computing $Score_A(.)$.

For example, suppose you find that 95% of the errors were due to the scoring function $Score_A(.)$, and only 5% due to the optimization algorithm. Now you know that no matter how much you improve your optimization procedure, you would realistically eliminate only ~5% of our errors. Thus, you should instead focus on improving how you estimate $Score_A(.)$.

45 General form of Optimization Verification test

You can apply the Optimization Verification test when, given some input x, you know how to compute $\text{Score}_x(y)$ that indicates how good a response y is to an input x. Furthermore, you are using an approximate algorithm to try to find $\arg\max_y \text{Score}_x(y)$, but suspect that the search algorithm is sometimes failing to find the maximum. In our previous speech recognition example, x=*A* was an audio clip, and y=*S* was the output transcript.

Suppose y* is the "correct" output but the algorithm instead outputs y_{out}. Then the key test is to measure whether $\text{Score}_x(y^*) > \text{Score}_x(y_{out})$. If this inequality holds, then we blame the optimization algorithm for the mistake. Refer to the previous chapter to make sure you understand the logic behind this. Otherwise, we blame the computation of $\text{Score}_x(y)$.

Let's look at one more example. Suppose you are building a Chinese-to-English machine translation system. Your system works by inputting a Chinese sentence *C*, and computing some $\text{Score}_C(E)$ for each possible translation *E*. For example, you might use $\text{Score}_C(E) = P(E|C)$, the probability of the translation being E given that the input sentence was *C*.

Your algorithm translates sentences by trying to compute:

$$\text{Output} = \arg\max_E \text{Score}_C(E)$$

However, the set of all possible English sentences *E* is too large, so you rely on a heuristic search algorithm.

Suppose your algorithm outputs an incorrect translation E_{out} rather than some correct translation *E**. Then the Optimization Verification test would ask you to compute whether $\text{Score}_C(E^*) > \text{Score}_C(E_{out})$. If this inequality holds, then the $\text{Score}_C(.)$ correctly recognized E* as a superior output to E_{out}; thus, you would attribute this error to the approximate search algorithm. Otherwise, you attribute this error to the computation of $\text{Score}_C(.)$.

It is a very common "design pattern" in AI to first learn an approximate scoring function $\text{Score}_x(.)$, then use an approximate maximization algorithm. If you are able to spot this pattern, you will be able to use the Optimization Verification test to understand your source of errors.

46 Reinforcement learning example

Suppose you are using machine learning to teach a helicopter to fly complex maneuvers. Here is a time-lapse photo of a computer-controller helicopter executing a landing with the engine turned off.

This is called an “autorotation” maneuver. It allows helicopters to land even if their engine unexpectedly fails. Human pilots practice this maneuver as part of their training. Your goal is to use a learning algorithm to fly the helicopter through a trajectory *T* that ends in a safe landing.

To apply reinforcement learning, you have to develop a “Reward function” *R*(.) that gives a score measuring how good each possible trajectory *T* is. For example, if *T* results in the helicopter crashing, then perhaps the reward is *R(T)* = -1,000—a huge negative reward. A trajectory *T* resulting in a safe landing might result in a positive *R(T)* with the exact value depending on how smooth the landing was. The reward function *R*(.) is typically chosen by hand to quantify how desirable different trajectories *T* are. It has to trade off how bumpy the landing was, whether the helicopter landed in exactly the desired spot, how rough the ride down was for passengers, and so on. It is not easy to design good reward functions.

Given a reward function *R(T),* the job of the reinforcement learning algorithm is to control the helicopter so that it achieves $\max_T R(T)$. However, reinforcement learning algorithms make many approximations and may not succeed in achieving this maximization.

Suppose you have picked some reward *R(.)* and have run your learning algorithm. However, its performance appears far worse than your human pilot—the landings are bumpier and seem less safe than what a human pilot achieves. How can you tell if the fault is with the reinforcement learning algorithm—which is trying to carry out a trajectory that achieves $\max_T R(T)$—or if the fault is with the reward function—which is trying to measure as well as specify the ideal tradeoff between ride bumpiness and accuracy of landing spot?

To apply the Optimization Verification test, let T_{human} be the trajectory achieved by the human pilot, and let T_{out} be the trajectory achieved by the algorithm. According to our description above, T_{human} is a superior trajectory to T_{out}. Thus, the key test is the following: Does it hold true that $R(T_{\text{human}}) > R(T_{\text{out}})$?

Case 1: If this inequality holds, then the reward function $R(.)$ is correctly rating T_{human} as superior to T_{out}. But our reinforcement learning algorithm is finding the inferior $T_{\text{out.}}$ This suggests that working on improving our reinforcement learning algorithm is worthwhile.

Case 2: The inequality does not hold: $R(T_{\text{human}}) \leq R(T_{\text{out}})$. This means $R(.)$ assigns a worse score to T_{human} even though it is the superior trajectory. You should work on improving $R(.)$ to better capture the tradeoffs that correspond to a good landing.

Many machine learning applications have this "pattern" of optimizing an approximate scoring function $\text{Score}_x(.)$ using an approximate search algorithm. Sometimes, there is no specified input x, so this reduces to just Score(.). In our example above, the scoring function was the reward function $\text{Score}(T)=\text{R}(T)$, and the optimization algorithm was the reinforcement learning algorithm trying to execute a good trajectory T.

One difference between this and earlier examples is that, rather than comparing to an "optimal" output, you were instead comparing to human-level performance T_{human}.We assumed T_{human} is pretty good, even if not optimal. In general, so long as you have some y* (in this example, T_{human}) that is a superior output to the performance of your current learning algorithm—even if it is not the "optimal" output—then the Optimization Verification test can indicate whether it is more promising to improve the optimization algorithm or the scoring function.