Writeup

Yannik Pitcan
Ram Akella

October 1, 2019

Research Objective

Deep reinforcement learning methods have not been studied for the preventive maintenance problem, not withstanding claims made by firms such as GE. The first goal is to understand how well such deep reinforcement learning methods can help us implicitly predict quantities such as RUL (remaining useful life) and achieve close to optimal repair policies. The second goal is to investigate how we can combine observations about the system process with prior knowledge and for multiple predictor distributions. In particular, we want to understand how to mix the two models to get a better understanding of our system. One is data-based empirical knowledge and the second is either prior knowledge or expertise-based adaptation of empirical knowledge. This will be combined with real-time observations. Currently, the first task is completed with deep reinforcement learning and classical POMDP implementations for the CMAPPS dataset. The focus from now onwards is the second task.

Dynamical System Consideration

Past work on RL for preventive maintenance had promise, but suffered from a few flaws. One of them was the lack of distinguishment between covariates and sensor readings. For example, past work attempted to understand the RUL distribution via fitting a Weibull distribution. But, the fitted parameters (λ and k) were dependent on input covariates. This takes us to our second point that summary values such as RUL (remaining useful life) were used to represent the full distribution, when a set of samples (x_t, y_t) from said distribution may have been preferable. Our current work differs from past work in that it models the data as a dynamical system with latent states and observations and avoids encapsulating knowledge of the distributions by a summary statistic.

POMDP vs MDP

The use of a Partially Observable Markov Decision Process goes in tandem with distinguishing between covariates and sensor readings. In the past work when the preventive maintenance problem was treated as an MDP, it ignored the fact that there may be hidden states (for example, system health) that impacted sensor observations.

Outline

We demonstrate two methods. One will be fitting an HMM to the provided data and then we have a POMDP. After that, we can solve the POMDP using model-based approaches.

The other approach involves using POMDP-RL techniques using deep variational autoencoders .

Using a Modified EM Framework

The question then becomes: "How we can determine these hidden states?

Let $S = \{1, 2, \ldots, X\}$ be the space of underlying states and the space of observations be O. An HMM $\theta = (\pi_0, P, B)$ is parameterized by π_0, P, and B representing the initial state matrix, state transition matrix, and emission matrix respectively. Let $\pi_0 = [P(s_0 = i)]_{i=1}^X$, $P_{ij} = P(s_{t+1} = j | s_t = i)$ and $B_{ij}(\eta) = P_\eta(y_t = j | s_t = i)$.

For estimation, we can use an EM framework, treating the hidden states as the latent states in a Hidden Markov Model with our measurements as the observations. The latent states represent the health of the machine, where 1 is failing condition and M is perfect condition.

Let $\mathcal{Y} = (Y^{(1)}, \ldots, Y^{(D)})$ be the set of D observation sequences, where each $Y^{(i)} = (y_1^{(i)}, \ldots, y_T^{(i)})$ and assume each observation is an iid draw.

The technique we use is to repeat the following steps until convergence ($\theta = (\pi_0, P, B)$):

- Compute $Q(\theta, \theta^s) = \sum_{x \in \mathcal{S}} \log[P(\mathcal{O}, x; \theta)] P(x | \mathcal{O}; \theta^s)$ for the E-step
- Set $\theta^{s+1} = \arg\max_\theta Q(\theta, \theta^s)$ for the M-step.

Let $\theta^s = (\pi_0^s, P^s, B^s)$.

We can rewrite the auxiliary log-likelihood as

$$Q(\theta^s, \theta) = \mathrm{E}\left[\sum_{k=1}^N \sum_{i=1}^X I(x_k = i) \log B_{iy_k}(\eta) + \sum_{k=1}^N \sum_{i=1}^X \sum_{j=1}^X I(x_{k-1} = i, x_k = j) \log P_{ij} | y_{1:N}, \theta^{(n)}\right] + const(\theta^s)$$
$$= \sum_{k=1}^N \sum_{i=1}^X \pi_{k|N}(i) \log B_{iy_k}(\eta) + \sum_{k=1}^N \sum_{i=1}^X \sum_{j=1}^X \pi_{k|N}(i,j) \log P_{ij} + const(\theta^s)$$

In this setting, $\pi_{k|N}(i) = P(x_k = i | y_{1:N}, \theta^s)$ and $\pi_{k|N}(i,j) = P(x_k = i, x_{k+1} = j | y_{1:N}, \theta^s)$. We derive these probabilities in the next subsection.

$$P_{ij} = \frac{\sum_{k=1}^N \pi_{k|N}(i,j)}{\sum_{k=1}^N \pi_{k|N}(i)}$$

is our estimate after maximizing $Q(\theta^s, \theta)$ with respect to the constraint that P is stochastic.

Forward-Backward Filtering

It is not obvious how to compute $\pi_{k|N}$ and $\pi_{k|N}(i,j)$ at a first glance. For this, we use a forward-backward filtering algorithm. In this algorithm, we make two passes through the data. The first pass goes forward in time while the second pass goes backward in time.

$$\pi_{k|N}(i) = P(x_k = i | y_{1:N}) = P(x_k = i | y_{1:k}, y_{k+1:N}) \propto P(y_{k+1:N} | x_k = i) P(x_k = i | y_{1:k}) = \alpha_{1:k}(i) \beta_{k+1:N}(i)$$

where $\alpha_{1:k}(i) = P(x_k = i | y_{1:k})$ and $\beta_{k+1:N}(i) = P(y_{k+1:N} | x_k = i)$. In shorthand notation, we say

$$\pi_{k|N} \propto \alpha_{1:k} \beta_{k+1:N}.$$

We call α and β the forward and backward 'messages' respectively.

Similarly,

$$\pi_{k|N}(i,j) \propto \alpha_{1:k}(i) P_{ij} B_{jy_{k+1}} \beta_{k+1:N}(j).$$

To compute the forward and backward messages, we can do this recursively. For the forward update, we have

$$\begin{aligned}\alpha_{1:k+1} &= P(x_{k+1}|y_{1:k+1}) \\ &= P(x_{k+1}|y_{1:k}, y_{k+1}) \\ &\propto P(y_{k+1}|x_{k+1}, y_{1:k})P(x_{k+1}|y_{1:k}) \\ &\propto P(y_{k+1}|x_{k+1})P(x_{k+1}|y_{1:k})\end{aligned}$$

And

$$\begin{aligned}P(x_{k+1}|y_{1:k+1}) &\propto P(y_{k+1}|x_{k+1}) \sum_{x_k} P(x_{k+1}|x_k, y_{1:k})P(x_k|y_{1:k}) \\ &= P(y_{k+1}|x_{k+1}) \sum_{x_k} P(x_{k+1}|x_k)P(x_k|y_{1:k}) \\ &= P(y_{k+1}|x_{k+1}) \sum_{x_k} P(x_{k+1}|x_k)\alpha_{1:k}.\end{aligned}$$

And, for the backward update, we have

$$\begin{aligned}\beta_{k+1:N} &= P(y_{k+1:N}|x_k) \\ &= \sum_{x_{k+1}} P(y_{k+1:N}|x_k, x_{k+1})P(x_{k+1}|x_k) \\ &= \sum_{x_{k+1}} P(y_{k+1:N}|x_{k+1})P(x_{k+1}|x_k) \\ &= \sum_{x_{k+1}} P(y_{k+1}, y_{k+2:N}|x_{k+1})P(x_{k+1}|x_k) \\ &= \sum_{x_{k+1}} P(y_{k+1}|x_{k+1})P(y_{k+2:N}|x_{k+1})P(x_{k+1}|x_k) = \sum_{x_{k+1}} P(y_{k+1}|x_{k+1})\beta_{k+2:N}P(x_{k+1}|x_k)\end{aligned}$$

Assume we see the observation at the kth step is j, i.e. $y_k = j$. Then $O_k = diag(B_{*j})$, i.e. it is a matrix with diagonal entries given by $P(y_k = j|x_k = i)$. Then in matrix form, we can write the above recursions as

$$\alpha_{1:k+1} \propto O_{k+1}P^\top \alpha_{1:k}$$

and

$$\beta_{k+1:N} = PO_{k+1}\beta_{k+2:N}.$$

Adapting for the initial and end states being fixed

As mentioned before, we must constrain our set of state sequences to those that begin with a "perfect" state and end at a "failing" state. The difference between past work and our work is that we not only constrain the starting state, but also the final state.

There are several ways one can do this. We investigated two methods – the first method involved modifying the M-step calculation for the expected log-likelihood to add an extra constraint in the Lagrangian. The other route we looked at was incorporating the constraint in the forward-backward recursion. We decided to use the latter approach as it was more amenable to a closed form solution.

Since we know the final state will be a failing state, this means we want

$$\alpha_{1:N} = P(x_N|y_{1:N}) = (0, 0, 0, 1),$$

the vector of all zeroes except for a 1 in the final position. This is the final α calculated in the forward recursions traditionally, but assuming O and P are invertible, we see that

$$\alpha_{1:k+1}O_{k+1}^{-1}(P^\top)^{-1} \propto \alpha_{1:k}.$$

	0	1	2	3
0	0.57195	0.428050	0.000000	0.000000
1	0.00000	0.652764	0.347236	0.000000
2	0.00000	0.000000	0.930993	0.069007
3	0.00000	0.000000	0.000000	0.940000

Figure 1:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.360700	0.352035	0.654110	0.250018	0.158376	0.313508	0.674435	0.263660	0.195972	0.370725	0.369717	0.599625	0.628262
1	0.436410	0.418065	0.573899	0.295204	0.185121	0.401567	0.579687	0.317424	0.215156	0.444231	0.430179	0.533618	0.562867
2	0.547470	0.516838	0.453458	0.371906	0.235578	0.543432	0.452842	0.389537	0.255838	0.560099	0.523920	0.416080	0.435070
3	0.691413	0.630099	0.320734	0.470864	0.308480	0.698728	0.296990	0.493328	0.316482	0.694711	0.649091	0.280831	0.279651

Figure 2:

Thus we can work backwards from $\alpha_{1:N}$ to determine the rest of the α terms.

Every time we do the E step in our EM algorithm, we must compute $\alpha_{1:j}$ for $j \in \{1, \ldots, N\}$. To that extent, note that $\alpha_{1:1} = (1, 0, 0, 0)$ and $\alpha_{1:N} = (0, 0, 0, 1)$ as noted previously. Then

$$\alpha_{1:k} \propto \prod_{i=2}^{k} (O_i P^\top) \alpha_{1:1}$$

in the forward direction. And in the backward direction,

$$\alpha_{1:k} \propto \prod_{i=N}^{k} (P^\top)^{-1} O_i^{-1} \alpha_{1:N}.$$

To account for constraints on the final state *and* the start state, we redefine

$$\alpha_{1:k} = \frac{c_1 \prod_{i=N}^{k} (P^\top)^{-1} O_i^{-1} \alpha_{1:N} + c_2 \prod_{i=2}^{k} (O_i P^\top) \alpha_{1:1}}{2},$$

for $1 < k < N$, where c_1 and c_2 are normalizing constants for the corresponding terms in the numerator.

Our backward message updates do not change. These averaged α values along with the β values are then used to compute $\pi_{k|N}(i)$

And in the M-step, we skip parameter updates for P_{ij} when $i < j$ and keep these values fixed at 0. This idea is inspired by Roweis [1].

In summary, we use an altered method of forward-backward filtering to control our first and last states. We don't believe this has been done in the literature and empirically, we also get promising results when using this method.

The following figures show the state transition (Figure 1) and emission matrices. For our emissions, we provide two matrices – one representing the means at each state (Figure 2) and the other with variances (Figure 3).

Now that we have hidden states estimated, we can introduce the POMDP model.

Variational Autoencoders

The prior two approaches involved treating our preventive maintenance problem as a hidden Markov Model. We can use deep learning techniques such as variational autoencoders instead to tackle this

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.009310	0.008882	0.007158	0.005685	0.004610	0.006988	0.007201	0.005570	0.004541	0.008132	0.008022	0.008182
1	0.010611	0.009626	0.007814	0.005592	0.004656	0.008099	0.008251	0.005520	0.004522	0.008816	0.008692	0.008705
2	150.009463	150.008090	150.006493	150.003809	150.002515	150.007039	150.007106	150.003744	150.002358	150.007439	150.007417	150.007657
3	540.005828	540.004954	540.004172	540.002446	540.001798	540.004966	540.004852	540.002472	540.001616	540.005065	540.004553	540.004810

Figure 3:

problem. The intuition behind the VAE approach is we want to maximize the sum of log marginal likelihood terms $\sum_{n=1}^{N} \log p_\theta(o^{(n)})$ for observations $(o^{(n)})_{n=1}^{N}$ where $p_\theta(o) = \int p_\theta(o|s)p_\theta(s)\,ds$. This is intractable in practice so instead we maximize the sum of ELBOs where each ELBO term is a lower bound of the log marginal likelihood.

$$ELBO(\theta, \phi, o) = \mathrm{E}_{q_\phi(s|o)}[\log \frac{p_\theta(o|s)p_\theta(s)}{q_\phi(s|o)}]$$

for a family of encoders $q_\phi(s|o)$ parameterised by ϕ. One can see the analogy between this and the EM approach discussed earlier. We're using a surrogate expression, ELBO, instead of the log marginal distribution.

Deep Variational RL

Lastly, we can incorporate estimation and control by using deep variational reinforcement learning. Fundamentally, instead of learning a generative model that maximizes just the ELBO functions, we also optimize an expected return. We implemented this on CMAPSS data using Tensorflow instead of PyTorch because we needed to modify our implementation to restrict the state sequences from our model estimates. Tensorflow, although more difficult to prototype in, leads itself to customization far easier than Keras or PyTorch.

Detailed Discussion of Two Research Pathways

Currently, the models in place just treat the preventive maintenance problem as an MDP, focusing solely on measurement observations (y_t) while ignoring the potential hidden states (x_t). This leads us to discuss the first research direction.

Path 1: Single Source

This may be split into two parts:

- **Estimation:** System identification – estimate an underlying state and then use model-based approaches to solve for optimal repair strategy to reduce total maintenance cost.
- **Control:** Determining optimal actions without knowledge of underlying states (model-free reinforcement-learning approach)
- **Combining Estimation and Control:** Understanding the benefits of using a deep-RL approach that combines the two.

As before, we create a simulation for a machine running until it reaches failure. The underlying states $(1, 2, 3, 4)$ represent the health level of the machine, with 1 indicating very-low/failing and 4 very-high/perfect condition. A level of 2 denotes low but not failing and 3 moderately high health.

Our state transitions are given by

$$\begin{bmatrix} p_{11} & 0 & 0 & 0 \\ p_{21} & p_{22} & 0 & 0 \\ 0 & p_{32} & p_{33} & 0 \\ 0 & 0 & p_{43} & p_{44} \end{bmatrix}$$

where p_{ij} denotes the probability of the system health moving from level i to j. The above system is for the case where no action is taken. This model assumes gradual degradation.

The actions here are either to repair $a = 1$ or do nothing $a = 0$.

When a repair is done, then

$$\begin{bmatrix} 0 & p'_{12} & p'_{13} & p'_{14} \\ 0 & 0 & p'_{23} & p'_{24} \\ 0 & 0 & 0 & p'_{34} \\ 0 & 0 & 0 & p'_{44} \end{bmatrix}$$

is our transition matrix, where we use p' to denote these are probabilities when we do a repair.

Our observations depend on whether we repair the system or not. Furthermore, the observation space is continuous with the observations distributed by pdfs f_0 and f_1 for actions 'no-repair' and 'repair' respectively.

If we don't repair the system, then define $O(o|s, a = 0) \sim f_0(s)$. Else, if we do repair, then $O(o|s, a = 1) \sim f_1(s)$.

If we use discrete observation buckets, then we have a matrix form

$$O(0) = \begin{bmatrix} NoRepair & 1 & 2 & 3 & 4 \\ 1 & o_{11} & o_{12} & o_{13} & o_{14} \\ 2 & o_{21} & o_{22} & o_{23} & o_{24} \\ 3 & o_{31} & o_{32} & o_{33} & o_{34} \\ 4 & o_{41} & o_{42} & o_{43} & o_{44} \end{bmatrix}$$

and

$$O(1) = \begin{bmatrix} Repair & 1 & 2 & 3 & 4 \\ 1 & o'_{11} & o'_{12} & o'_{13} & o'_{14} \\ 2 & o'_{21} & o'_{22} & o'_{23} & o'_{24} \\ 3 & o'_{31} & o'_{32} & o'_{33} & o'_{34} \\ 4 & o'_{41} & o'_{42} & o'_{43} & o'_{44} \end{bmatrix}$$

Lastly, our reward function takes the following form:

$$\sum_{t=1}^{T} \sum_{s=1}^{4} [c_1(4 - s_t) I(s_t = j) + c_2(s_t) I_M(a_t = 1 | s_t = j)]]$$

where I is an indicator function. $c_1(4 - s_t)$ is the cost of maintenance and $c_2(s_t)$ is the cost of being in a degraded state.

Another way of writing this is as follows

$$R(0) = \begin{bmatrix} NoRepair & Reward \\ 1 & r_1 \\ 2 & r_2 \\ 3 & r_3 \\ 4 & r_4 \end{bmatrix}$$

$$R(1) = \begin{bmatrix} Repair & Reward \\ 1 & r'_1 \\ 2 & r'_2 \\ 3 & r'_3 \\ 4 & r'_4 \end{bmatrix}$$

In our experiments, we use the reward matrix

100	50	0	-50
-50	0	50	100

0	204.525160
1	88.982036
2	-34.290243
3	-171.265281

Figure 4:

For example, the reward we get for repairing in state 2 is 50.

The overarching goal is to use reinforcement learning methods with POMDPs to give an optimal strategy for repairing the system that minimizes the total repair costs and losses due to non-repair.

The two routes are as follows:

- 1. EM based approach for estimation and then determining an optimal control policy using any POMDP solver. For example, we could use a VAE (variational autoencoder) for time series. This doesn't incorporate control, maximizing a likelihood function, and outputs a particle filter model.
- 2. DVRL (deep variational reinforcement learning), which combines both estimation and control. This is because it's optimizing an objective cost function that also depends on model parameters.

We also incorporate knowledge of the start and end states being in perfect and poor condition respectively, which we discuss in the next section.

Current Status 1

So far, we have studied four different POMDP based techniques for modeling the preventive maintenance problem. We implemented value-iteration, Monte-Carlo Tree Search (POMCP), ADRQN, and DVRL on the CMAPSS dataset. After finishing up these implementations, we are currently analyzing how well deep-RL methods such as DVRL and ADRQN performed versus model-based methods. We also incorporate constraints on the start and end states by a method similar to boosting, where we propagate the errors between our true start and end states and our estimates, re-running DVRL until we reach convergence. The next steps are analyzing how two sources of information (prior knowledge with observations) should be combined to determine optimal repair strategies.

Results from Separating Estimation and Control

After doing HMM estimation and POMDP value iteration, we get the utility vector seen in Figure 4.

In practice, we often have some existing knowledge about the system process before seeing measurements. This leads us to the second research path.

Path 2: Combining Multiple Sources

Here, we ask how to combine prior knowledge about the data generating process when determining the optimal repair strategy that minimizes costs. For example, if we have a controlled system generating data, this alters the observed machine measurements as opposed to an uncontrolled system. This is because if the machine is deteriorating, we repair and restore the system back to full health so our measurements seen will be different than if we let the machine fail.

The alternative route would be to identify enough of the distribution based on impact on the objective function and disambiguate.

For now, let us assume the data generating process takes on one of the following forms:

- No prior distribution – our data is generated via bootstrapping. For example, with the C-MAPPS data, we could resample from the trajectories.
- One prior
  - Do we trust this prior distribution? Or should we ignore that and just bootstrap? Or just combine both?
- We have two prior distributions and we would like to understand which one generates our data, or how they interact with each other. For example, we can combine two priors via a "weighted average" (partial update) or fully via a Bayesian update.

These three are the possible ways of combining the two models, which we are investigating:

- One generating model at each timestep
  - In this setting, only one of P_1 or P_2 generates our data.
- Mixture model
- Hierarchical Bayes

Analogously to the single source model, we have another layer of complexity when studying estimation versus control here. Either we could

- Estimate underlying model and then come up with a control policy using POMDP-RL.
- Perform control and estimation at once using a method such as DVRL again.

This section gives one a glimpse of some of the open problems we are working on.

Sequential Testing

The paper `http://stat.columbia.edu/~jcliu/paper/GSPRT_SQA3.pdf` introduces Sequential Probability Ratio Testing (SPRT) for different families of hypotheses.

When we have two families of composite hypotheses, i.e.,

$$H_0 : f \in \{g_\theta : \theta \in \Theta\} \text{ against } H_A : f \in \{h_\gamma : \gamma \in \Gamma\}$$

, we consider a sequential test where we compute

$$L_n = \frac{\sup_{\gamma \in \Gamma} \prod_{i=1}^n h_\gamma(X_i)}{\sup_{\theta \in \Theta} \prod_{i=1}^n g_\theta(X_i)}.$$

If L_n crosses e^A or e^{-B} for constants $A, B > 0$, then we stop sampling. If $L_n > e^A$, then we reject the null.

We could use this to infer which distribution is generating our data, but the research component is understanding how to use this with particle filters. If we use a variational autoencoder to learn an underlying model, the output is a particle filter. Currently there hasn't been work on how to use SPRT with particle filters.

Example: Mixture Model Setting

What we would like to solve is initially the following: If our data is generated from $\epsilon_t P_0 + (1-\epsilon_t)P_1$, where P_0 and P_1 are the two priors, does $\epsilon_t \to 0$ or to a constant over time? Maybe P_0 is one prior generated from bootstrapping and P_1 is the second prior provided or based on data that incorporates domain knowledge.

In this setting, we can think of our $z_t = \{x_t, y_t\}$ pairs coming from data sources P_0 and P_1.

Define $p(z_t|P_0)$ and $p(z_t|P_1)$ to be the probabilities of observing z_t given P_0 and P_1 respectively at time t. Then

$$p(P_j|z_t) = \frac{\epsilon_t p(z_t|P_0)I(j=0) + (1-\epsilon_t)p(z_t|P_1)I(j=1)}{\epsilon_t p(z_t|P_0) + (1-\epsilon_t)p(z_t|P_1)}.$$

But now, we want to understand

$$p(\{P_{j_1}, \ldots, P_{j_T}\}|\{z_1, \ldots, z_T\})$$

where each P_{jt} is either P_0 or P_1.

The simplistic approach to estimate ϵ would be to do a maximization of the above quantity over all assignments of $\{P_{j1}, \ldots, P_{jT}\}$ to the set $\{P_0, P_1\}^T$. This method, however, is computationally intensive and furthermore does not account for time-dependence in our data.

We propose modeling this as a multiarmed bandit problem, where P_0 and P_1 are the two arms. The goal here is to then use concentration inequalities to understand good approximate repair policies. These will be approximate because there is a dependency between actions taken at different timesteps and we are looking at finite-horizon solutions.

Current Status 2

For this, the first priority is studying how we can combine particle filters (outputs of VAE) can be used with SPRT. Other open problems that will be tackled after this are how to get guarantees on learning multiple source distributions using concentration-type inequalities.

References

[1] S. Roweis. Constrained Hidden Markov Models. Advances in neural information processing systems, 2000.