Acta Universitatis Sapientiae
Electrical and Mechanical Engineering, 3 (2011) 51-60

Adaptation of Energy Production to Forecast Values Using External Storage

Sándor KAZI

Department of Computer Science,
Faculty of Electrical Engineering and Informatics,
Budapest University of Technology and Economics, Budapest, Hungary
e-mail: kazi@cs.bme.hu

Manuscript received November 20, 2011; revised December 20, 2011.

Abstract: This paper introduces a real-world optimization task of electrical energy production. Electrical energy production and consumption relies in a large measure on forecasts, the main difference among countries and among sectors is in the person or organization who is supposed to make predictions. In some cases it is the responsibility of the consumer or of the energy producer, in other cases there are specialized companies for hire to make these forecasts (for ex. "*Regelzonenführer*"-s in Austria). This paper considers the problem when the forecast is already given by the producer and they have to predict almost the exact amount of electrical energy to be produced if they want to avoid penalties. The problem in question is from a sector and from a country (wind turbines, Hungary) where the forecast is made by the energy producer and large differences between the predicted and the actually produced values are penalized. The optimization problem is to use a storage facility effectively enough to increase the income of a wind farm by the later submission of previously overproduced electrical energy. This paper introduces this problem in details, and presents solutions for it. The steps to create a reinforcement learning solution for this kind of a stochastic problem are presented besides a simple and effective solution for this exact task. The reinforcement learning solution consists of a modeling and an algorithm application step, and also of the incremental steps to make the solution more effective by specialization.

Keywords: Wind energy, optimization, energy storage, reinforcement learning.

1. Detailed specification

In some countries energy producer companies are obligated to submit a forecast of their next production period. This is a reasonable expectation,

because the buyer (in most cases an electricity service provider) needs to know how much energy can they count on. Because of that, this schedule has to be as accurate as possible. Sometimes the companies are motivated by rewards or penalties according to the accuracy of their predictions. If they apply a storage facility they can balance the over- and underproduction by storing and retrieving energy. This paper is about the strategy of effectively using storage to enhance productivity and increase income by adaptation to the previous prediction and by not being penalized.

The available data of the era is from a wind farm (Mosonszolnok, Hungary; seven turbines), the owners of which have to submit predictions daily for each quarter hour. The data records are historical and consist of a timestamp (interval identifier), a forecast and the produced value.

The environment in question has law enforced buying prices for wind energy. If the difference between the prediction and the production is less than a previously specified threshold (currently that is fifty percent of the forecast), the buying price is approximately 0.1€ per kilowatt-hour. If the produced value is out of the margin the company has to pay a 0.04€ penalty per every kilowatt hour of the difference between the forecast and the production; the buying price remains the same.

Because of the structure of the data, we have to make a conversion from the real life control problem into a discrete time decision problem. To reduce the original optimization task into this kind of a mathematical form we should make assumptions and disregard some components of it.

We disregard the continuance of the quarter hour periods – we assume that decisions have to be made at the end of these periods where the forecast and the produced value are known. At the end of each period we can decide the submission rate of the production (how much should be stored and not submitted) – this is not a real life assumption but a reinforcement learning method designed to cope with this kind of problems can be a good approximation of a continuous time alternative.

We disregard the nature of the storage technology (dissipation and amortization) for this time, to test only the algorithms. There are three causes for this decision: the “continuity assumption” does not really support this kind of information (lifetime also depends on charging speed), this factor can also be considered later by altering the reward function, and the third cause is that it would be another stochastic factor added to the problem. It should be an independent parameter in the decision of whether it is reasonable or not to apply a storage facility, approximate costs can be calculated for these factors. The storage technology was considered only through its boundaries. Corresponding rated power and discharge time parameters belong to a storage which can be either an accumulator (NaS-accumulator for example) or pumped storage

facility (energy stored in the potential energy of water). These parameters are technology-dependent. In a step (quarter hour) the level of the storage can only be changed by the quarter of its rated capacity.

The algorithms consider the forecast to be fixed for each quarter hour and it is also considered out of scope: we assume that the prediction (which is based on meteorological data and personal expertise) is as close as possible. The aim of these algorithms is not to adjust the prediction but to develop a strategy to adjust to the forecasts using storage.

With these assumptions we have a discrete time environment, a decision is needed each step which defines the exact values to sell and to store into or to use up from the storage.

2. Reinforcement learning and modeling

Those who are not familiar with the concept of reinforcement learning can check the online[1] or printed version of an introduction book written by Richard S. Sutton and Andrew G. Barto [1]. This book clarifies the main ideas and methods of the area by the use of both mathematical reasoning and examples.

To apply a reinforcement learning method a Markov decision process is required. In most of the cases the aim of the algorithms is to discover the uncertain parts of the model or to develop a "valuable" strategy.

With the assumptions made in the previous section we have a discrete time stochastic environment which cannot be affected by our decisions, but has its own internal states and transitions among them. If we expand the "environment" with the current state of our storage it can be modeled with a discrete time Markov chain (the Markov-property is present). With actions taken into consideration it implies Markov decision processes (MDP for short) [1]. The "expanded environment" denomination mentioned above also has states, but they can be partly affected by our decisions.

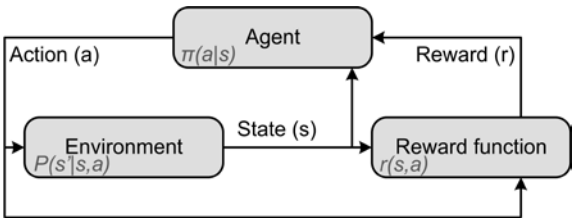

Figure 1: A common notation for a Markov decision process.

[1] An HTML version of the book can be found at the webpage of the author: http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html

It was clarified above, that this problem can be modeled by a Markov decision process. The next step in the modeling phase is to specify the parameters describing the state, the action and the reward (or the reward function). It is clear that if these parameters are given the parts missing from the figure above are the transition probabilities (P) and the policy (π). The policy is driven by the algorithms so it is not a question of modeling, but the probabilities clearly depend on the definition of states and actions.

To specify the Markov decision process we have to assign meaning for the state and action parts. The reward function can be different for each algorithm; therefore it will be defined among the algorithms and in the evaluation section. The state can contain most of the information about the "expanded" environment like: previous and current production and forecast values, current storage level and also forecasts for the upcoming intervals. It is a matter of modeling what we choose to represent a state. In most of the following methods the state contains all the previously mentioned information: it has a backward window for forecasts and production, a forward window for forecasts and contains the actual values of the storage, forecast and produced energy.

It is clear that the action has to represent the choice we make. It can be specified in a few equivalent ways. I chose it to be the one of the most similar ones to the state description: the action is represented by the new storage level (caused by our choice).

After examination a person could notice that in this environment we have more information than in a general case. We can calculate the rewards for all actions that we did not choose – it can be a great deal of information for a learning algorithm and lead to faster convergence.

3. Algorithms

1. Algorithms "by the book" [1]

Algorithm planning can be an iterative process, as it was this time. There were a few less successful algorithms in coping with the above described problem, and a few which can be counted as a success. The first algorithm I used to test the applicability of reinforcement learning on the task was a well-known dynamic programming method [1]. This method had two flaws to fall back on: the accuracy of the transition probabilities, and the fact that it needs final (exit) states or an episodic MDP to calculate the reward backwards. After a conversional step to an episodic MDP (1 episode ~ one or a few days) the only uncertainty is the lack of information about transitions. This information can be gathered runtime, but the granularity (20000 different production values) and the quantity (9000 rows) of the data do not make it possible to gather enough

empirical information about the transition probabilities. Because of these factors I applied quantification in my dynamic programming method, which is compromise, because the margin is stiff, only a difference of 1kWh-s can make a difference.

I also applied an R-learning algorithm [1] on the model, which is an average reward maximization method. The flaw of the model is that it can be used effectively only if the forecast or the margin width is constant. The lack of information about transition probabilities is also present.

2. SARSA algorithms

The methods presented above had their flaws which made them ineffective. The algorithm class I applied next is the SARSA approach which is a different point of view among reinforcement learning methods. It considers state-action-reward-state-action tuples and operates by maintaining a value estimate for state-action pairs (instead of states). This algorithm in its basic form [1] still has the flaw that it needs a good estimate of transition probabilities. We still cannot provide these estimates to be accurate enough because of the quantity of the data. Yet again, quantification is an option, but it still makes the approximate values unreliable.

To improve the SARSA solution by the means of the learning process itself I transformed it into an algorithm using eligibility traces (SARSA(λ)). The eligibility traces are supposed to make the learning process faster by using the information from one step to update more than one value estimates. It is also important to shake off the problem of the unknown probabilities. It is easy to see, that close values represents similar states and actions. This fact calls for exploitation of generalization. Function approximation is a common technique for this kind of task, the linear, gradient-descent SARSA(λ) (for an algorithm using binary features, check [1]) suits these expectations. The effectiveness of linear, gradient-descent function approximation methods depends on the selection method of features.

The method which brought success is a linear, gradient-descent method using Gauss-functions as features. We need a method to select the appropriate feature which divides the state-action space by the features how they can best express the similarity among the state-action pairs. If we divide the space in every dimension for a grid-like Gauss-function placement, we will be stricken by the curse of dimensionality (for n attributes representing the state-action space, we need k^n features to set two values in each dimension). The number of the features is clearly a factor in the runtime of an algorithm, so it is recommended to keep the number of features lower. There are references on the effectiveness of random feature selection [2]. This method chooses a specified number of features randomly from the set of possible values. For the random

selection to be enough, the state and the action representations are also needed to be bounded in every dimension. Luckily, the building blocks of the state and the action representations are bounded variables or a realistic upper and lower bound can be selected. According to the dataset, the upper bound production value is approximately 25000 kWh, but it can be specified as 20000 kWh because with a probability of 0.98 it is also an upper bound. Disregarding very rare events in the feature selection can lead to a better model.

The pseudo-code for the first version of my SARSA(λ) algorithm can be seen on the figure below. The x_i-s are the Gauss functions used for the calculations of φ vectors, I_{acc} is an indicator distribution (equals to 1 in case of "accumulating traces" and to 0 in case of "replacing traces" mode [1]), $\langle \_ , \_ \rangle$ is the scalar product of the two parameters. All other notations have the same meaning as in the original SARSA(λ) variant [1].

Linear, gradient-descent SARSA(λ) (Gauss features)

$n \leftarrow$ number of features
$\vec{\theta} \leftarrow$ any n-parameter gradient vector
$\vec{e} = 0$
for all $\forall i \in [1, n]$ **do**
    $x_i : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}_+$ (Gauss function) $\left(\vec{\phi}(s,a) = \left(x_1(s,a), x_2(s,a), \ldots, x_n(s,a)\right)^T \right)$
end for
$s \leftarrow s_0$
$a \leftarrow \arg\max_{a \in \mathcal{A}(s)} \left\langle \vec{\phi}^T(s,a), \vec{\theta} \right\rangle$
while s is not terminal **do**
    $\vec{e} \leftarrow I_{acc}\vec{e} + \vec{\phi}(s,a)$
    $a \rightarrow r,\ s'$
    $\delta \leftarrow r - \left\langle \vec{\phi}(s,a), \vec{\theta} \right\rangle$
    $a' \leftarrow \arg\max_{a \in \mathcal{A}(s')} \left\langle \vec{\phi}(s',a), \vec{\theta} \right\rangle$
    $\delta \leftarrow \delta + \gamma \left\langle \vec{\phi}(s',a'), \vec{\theta} \right\rangle$
    $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$
    $\vec{e} \leftarrow \gamma\lambda\vec{e}$
    $s \leftarrow s'$
    $a \leftarrow a'$
end while

Figure 2: The pseudo-code of a linear, gradient descent SARSA(λ) solution with eligibility traces and Gauss features.

As already noted there is more information about the process itself than usually. We can calculate the rewards not only for the chosen action but for all of them. If we make the adjustment of eligibility traces according to all of the actions, not only to the chosen action, it can lead to a better approximation. This is a manipulation of the tenth line of the pseudo-code: instead of the addition of one specific (chosen action) φ vector, the added value is the average of all φ vectors for each eligible action.

The feature selection method can also be specialized. One method of specialization is that the distribution used for feature center generation is not a uniform distribution but the distribution of the data or a special distribution which is dense where we want to distinguish states more and sparse where we do not want to. There is another similar method to specialize: we can use taller (and narrower) Gaussian functions in the dimension where it is more important to distinguish states and lower ones otherwise. These two methods can lead to a better feature generation algorithm.

Another modification to upgrade the performance of the algorithm is to avoid penalties by narrowing the set of the actions for each state. If we avoid penalties every time we can, it is a greedy-like minimization of the penalty, and by that a greedy-like maximization of income. The action to choose is still not trivial, so the optimization problem will change, but henceforward needs a solution. This modification can be placed between the twelfth line and the thirteenth line (or we can adjust the model itself) as a step narrowing the set of actions eligible at the current state.

3. Decision tree

The above mentioned greedy-like method had been tested before the success of the SARSA variants. We can call this simple algorithm a decision tree method. Let there be a preferred storage level (c_{pref}), set the new storage level (action) to this level. Now we are going to fit it into the specification (if it is required) by moving it backwards in the direction of the previous storage level. Let the possible action set be the set of all actions possible in all states (from the minimum level, to the maximum). Narrow this action state to the reachable actions: the maximum difference between the actual and the next state of the storage is the quarter of the rated power parameter. Then choose the action from set which is the nearest to the preferred storage level.

4. Evaluation and conclusion

We have the forecasts and real produced values of three months which leaves us with 8736 records having 5506 different forecast and 1672 different production values from the interval between 0 and 24000. The modus of each is

0 (1513 and 1617 times), because of the predicted “windless” intervals and maintenance shutdowns.

Long zero sequences are not really useful for comparison: if we predicted zero we are penalized if we sell any of our production, if we produce zero, than we have to use energy from the storage to occasionally avoid penalty. Both ways, we can get stuck on a full or an empty storage while the zeros are still coming. On the other hand, if the prediction is rarely bad for a long interval (or commute between high and low) then the decision tree method would be absolutely enough to solve the problem.

It’s visible on the forecast and production comparison diagrams, that there is a positive correlation between the two parameters, so it is not impossible to develop a strategy, but it is also a warning of “commute” mentioned above.

The methods described above were tested using three different reward functions. The first type of reward was the exact income which could be negative in case of a large underproduction. The second type of reward was an indicator-like function: 1 if the there are no penalties, -1 otherwise. The third type was similar to the second, but it was multiplied by the forecast value.

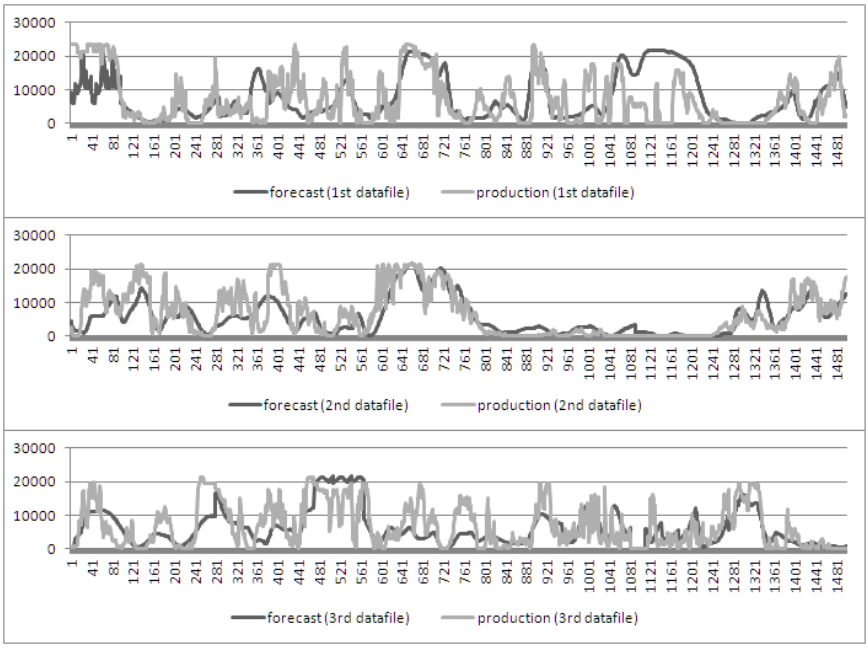

Figure 3: Forecast and production values compared to each other for each test set (horizontal axis represents the number of the interval, the vertical axis is the production or forecast value in kWhs).

But why do they represent the quality of the decision? The answer is trivial for the first one, because it is the income. The second one is not as obvious, and not always represents the real value of a decision, because a penalty can be associated with different values from a large interval. The third one represents the lost value by the penalty – it is twice as the margin size, so it is twice as the value which is sold or not sold on the highest price corresponding to the fact that we are penalized or not (if our prediction was 1000 and we sell 1500 we are not penalized, but if we sell another 500, then we get a penalty for the other 500 too, so we lose an amount directly proportional to the forecast).

More than one test is required to make assumptions, so I created three smaller histories (1500 rows each) and tested the SARSA and decision tree methods I created to cope with the problem. I also tested the "by the book" methods mentioned in the previous section, but the flaws already mentioned made them ineffective. R-learning performed better than dynamic programming (because of that, only R-learning is shown on the diagrams), but it still did not bring success. The biggest problem with them was the loss of accuracy because of the quantification.

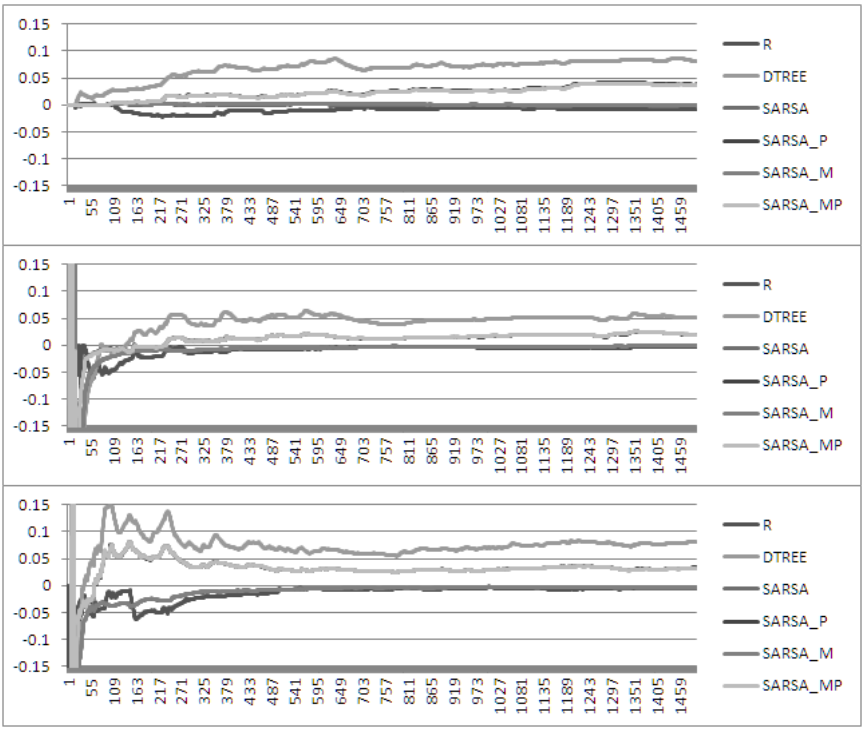

Figure 4: The sum of additional income over time divided by the original value (M stands for "multiple adjustments", P stands for "penalty avoidance").

There are higher values at the begging of two charts, the cause of these is the fact that one penalty avoidance step at the beginning can bring a huge additional income pro rata. All of the lines fall back after that, and stay on the displayed interval.

The best efficiency is presented by the decision tree algorithm, it brings from five to nine percent in addition, which is more than 8000€ for this less than 16 days long interval. Then it is not a surprise that penalty avoidance is also rewarding as a part of a SARSA algorithm: the two reinforcement learning algorithms using this bring approximately half of the success of the decision tree algorithm. The other tree algorithm types are slightly above (the other SARSA variants) or below (R-learning) 0.

Conclusion

The decision tree algorithm is simple, but effective. The SARSA algorithms with this upgrade can reach a reasonable additional income. It is clear that on these datasets the decision tree algorithm is better in this discrete time approach.

To make a real-world application the continuity assumption of the specification section has to be omitted. There are reinforcement learning methods to handle a continuous time problems ([3], [4]) and in these cases this model can be a good discrete model to start at.

References

[1] Sutton, R. S., Barto, A. G., “Reinforcement learning: An introduction”, MIT Press, Cambridge, MA, 1998.

[2] Szepesvári, Cs., “Reinforcement learning: dynamic programming”, University of Alberta, MLSS’08, Kioloa, 2008.

[3] Bradtke, S. J., Duff, M. O., “Reinforcement Learning In Continuous Time and Space”, *Advances in Neural Information Processing Systems, MIT Press*, pp. 393-400, 1994.

[4] Doya, K., “Reinforcement Learning In Continuous Time and Space”, *Neural Computation*, vol. 12, pp. 219-245, 2000.