Building fast proxies for optimization of thermal reservoir simulation

Matthieu Rousset

December 16, 2011

Abstract

With decreasing amounts of conventional energy resources available, production of unconventional oil and gas has become an important new source of energy. Steam-assisted gravity drainage (SAGD) is a widely used process, allowing the production of very heavy oils. The numerical simulation of SAGD, however, is complex and highly CPU/time intensive. Moreover, the economical cost of this process is high and production/design optimization could be very beneficial. The high CPU/time requirements, however, render optimization impractical. In this paper, we aim at the creation of fast proxies for SAGD reservoir simulation, with the objective of maximizing net present value (NPV) with respect to selected design parameters. For two practical examples, we use the commercial reservoir simulator STARS to run a number of training simulations. These results are then processed for the creation of an artificial neural network (ANN) and a suport vector regression (SVR) model. Sensitivity analysis is then performed on some of the proxies parameters in order to assess the quality of the models. The key question of how many training runs are required in order to obtain robust proxies for optimization, is also answered with sensitivity analysis. Finally, optimization is performed using the ANN proxy model.

1 Introduction

When producing oil from the subsurface, engineers often build a detailed geological model of the oil reservoir. This numerical representation of the underlying rocks and fluids is then used to predict the flow behavior, under a given set of controls. The controls usually represent the amount of pressure or flow that is imposed at the producing and injecting wells. An other important input feature is the placement of the wells. In this project, we focus on thermal applications, which require the injection of heat to achieve oil production. Specifically, we consider the steam-assisted gravity drainage (SAGD) process, which involves injecting steam into the subsurface in order to lower the oil viscosity, and allow production by gravity. Simulation of oil production from SAGD is a very difficult task, because it involves reproducing complex physical phenomena and strong nonlinearities (e.g. large variation of fluid properties with temperature). As a result, the simulation of these processes is very time/CPU intensive. Another aspect of this process, is its higher economical cost compared to traditional oil production. This makes it an ideal candidate for optimization, which aims at maximizing the predicted profits - e.g. net present value (NPV) - by finding the optimal design parameters and/or sequence of input controls. Unfortunately, optimization algorithms require running many simulations, which is often impossible with CPU intensive thermal simulations. In this work, we investigate on the creation of fast proxies for thermal reservoir simulations, using an artificial neural network (ANN) and a support vector regression (SVR) model. The proxies are then used to perform optimization, which is of high practical interest.

To the authors knowledge, there have been only a few attempts at the creation of fast proxies for thermal applications. Queipo et al. [1] have reported the use of neural networks to build proxies of the SAGD process, and have used them to perform global optimization of net present value. Although they seem to achieve good optimization results, it is unclear how accurate the proxy models are. Vanegas et al. [2] have developed a semi-analytical proxy model for SAGD, based on the widely used analytical model from Butler. It allows the use of heterogeneous fields of rock permeability but it is still based on a simplified physical description.

2 Machine learning methods

2.1 Artificial neural network (ANN)

ANN is a combination of artificial neurons that tries to mimic the structure of biological neural networks observed in nervous systems. It is composed of input nodes, output nodes and a number of hidden layers, containing artificial neurons. The first layer is called the input layer, containing all the input nodes, and the last layer is called the output layer containing all the output nodes. The number of hidden layers and the artificial neurons in each of these layers is a design parameter that can be varied. Fig. 1 shows an example of ANN with only one hidden layer of 4 artificial neurons.

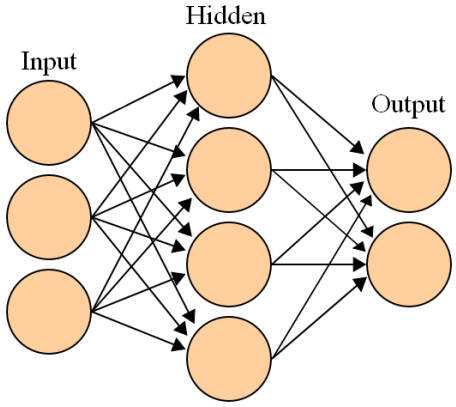

Figure 1: Example of an artificial neural network (source: Wikipedia)

We let u_i^j denote the input of the j^{th} node of the i^{th} layer and v_i^j denote its output. The scalar n_i represents the number of nodes in layer i and k the total number of layers. For the input layer, we have $u_1^j = v_1^j$ and for the output layer, we have $u_k^j = v_k^j$. For each of the hidden layers, we have

$$v_{i+1}^j = \sum_{\ell=1}^{n_i} W_i^{j,\ell} u_i^\ell + b_i^j \tag{1}$$

where $\mathbf{W}_i$ is the weight matrix and $\mathbf{b}_i$ is the bias vector between the i^{th} and the $i+1^{\text{th}}$ layers.

The neurons themselves represent activation functions which can be any differentiable function. In our implementation, we choose the sigmoid function, defined as follows:

$$g(x) = \frac{1}{1+e^{-x}} \tag{2}$$

The neural network is fully defined by:

1. The interconnection pattern between different layers of neurons
2. The activation function that converts a neuron's weighted input to its output activation.
3. The learning process for updating the weights of the interconnections

The first two points are defined by the weight matrix and the bias vector. The last point is achieve by minimizing some kind of error. In this work, we use Matlab neural network toolbox to build an ANN with one hidden layer of sigmoid neurons. Error minimization is achieved by using the Levenberg-Marquart backpropagation algorithm. Starting with a set of randomly generated initial guess for the weights and biases, the algorithm finds the parameters that minimize the error between the desired and the actual output. Note that the number of input nodes is equal to the number of input features and there is only one output node since we are using only one target variable (the NPV).

2.2 Support vector regression (SVR)

First proposed by V. Vapnik et al. [3], SVR is a regression model that aims at predicting the output of a nonlinear function for some given values of the input features. The general idea of SVR is to perform linear regression in a high dimensional feature space where the input data are mapped via a nonlinear function. Similarly to support vector machine, which is used for classification, SVR only consider a small number of training points to perform the prediction because it ignores any training data that is close (within a threshold ϵ) to the model prediction. We include here a bried description of SVR and more details can be found in [3].

In SVR, the goal is to find a function $f(x)$ that has at most ϵ deviation from the actual targets $y^{(i)}$ for all the training data, and at the same time is as flat as possible. In the case where $f(x)$ is a linear function of the form $f(x) =< \omega, x > +b$, the resulting primal optimization problem is given by:

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \\ &\text{subject to} && \begin{cases} y^{(i)} - < \omega, x^{(i)} > -b \le \epsilon + \xi_i \\ < \omega, x^{(i)} > -y^{(i)} + b \le \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases} \end{aligned} \tag{3}$$

Here, $<,>$ represents the inner product of two vectors. The norm $\|\omega\|^2$ measures the flatness of the proxy, and the constraints force the model to approximate all training points within ϵ precision. ξ_i and ξ_i^* are slacks variables introduced to allow

some trade-off between the flatness of the function and the compliance with the ϵ deviation constraints. C represents a weight given to the penalty for violating the constraints.

The corresponding dual problem is given by:

$$\begin{aligned} \max \quad & -\frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i-\alpha_i^*)\left(\alpha_j-\alpha_j^*\right) < x^{(i)}, x^{(j)} > \\ & -\epsilon\sum_{i=1}^{m}(\alpha_i+\alpha_i^*) - \sum_{i=1}^{m} y^{(i)}(\alpha_i-\alpha_i^*) \\ \text{s.t.} \quad & \begin{cases} \sum_{i=1}^{m}(\alpha_i-\alpha_i^*) \\ 0 \le \alpha_i, \alpha_i^* \le C \end{cases} \end{aligned} \tag{4}$$

The optimization problem in Eq. 4 is convex and can be solved using a readily available optimization algorithm. It is also clear from Eq. 4 that SVR can be kernelized by replacing terms of the form $< x, z >$ with a kernel function $K(x, z)$. In this work, we use libSVM [] for our implementation of SVR with a Gaussian kernel function.

3 Numerical examples

3.1 Well placement optimization

An oil reservoir with a net thickness of 38 meters, a width of about 164 meters and a length of 900 meters is represented with a 2D model containing 51×38 grid blocks. Each grid block is rectangular with dimensions 1.6 m by 1 m and the model represents half of the entire reservoir. A pair of SAGD horizontal wells is placed on the symmetry plane. Our goal is to find the best position for the producer and injector wells on this symmetry plane. The two design variables are the distances of the injector and producer wells from the bottom of the reservoir.

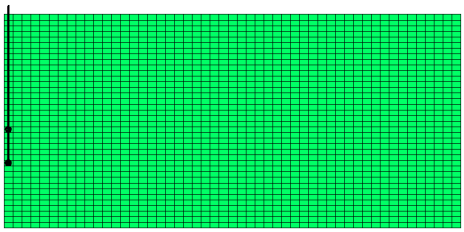

Figure 2: Schematic illustration of reservoir model 1 with the wells represented in black

We use a Matab script to run all possible combinations of input parameters, using the commercial simulator Stars. Since the injector well must always lie above the producer well, this represents 703 possible well placements. The net present value (NPV) is then computed for each configuration from the formula:

$$\text{NPV} = Q_o p_o + Q_g p_g - Q_w p_w - Q_s p_s, \tag{5}$$

where p_o, p_g, p_w and p_s represent the price of oil, price of gas, cost of producing water and the cost of injecting steam respectively. They are set to 80 \$/bbl, 5 \$/MMSCF, 10 \$/bbl and 3 \$/bbl. Oil and gas production are denoted with Q_o and Q_g, water production with Q_w, and steam injection with Q_s. Among the 703 sample points, 50 are picked randomly and labeled as testing samples. The remaining 653 sample points are used for training the models. We first create a number of artificial neural network proxies using subsets of the training samples set of increasing size. We use one hidden layers containing 10 artificial neurons. For each model, we compute the error between the ANN response and the true response using the testing set. Results are displayed in Fig. 3, and show that the error decreases sharply with increasing size of the training set until 100 sample points. Then, it remains close to about 0.05 for training sets containing more sample points. We also checked the sensitivity of an ANN model

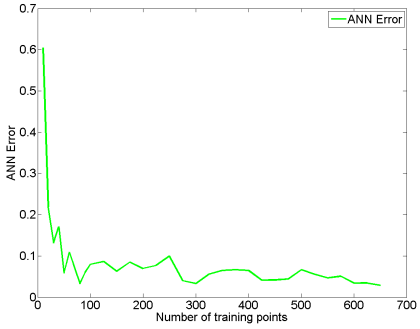

Figure 3: ANN testing error sensitivity to number of training samples

trained with a set of 100 training samples, to the number of hiden layers and number of artificial neurons. For this example, we found that ANN was not very sensitive to these parameters.

We follow the same procedure to build a number of SVR proxies with increasing number of training samples. Results are plotted in Fig. 4. SVR provides a relatively lower testing error with a smaller number of training samples than ANN. The error does not decrease as sharply however, with an increasing number of training points, but it reaches lower values.

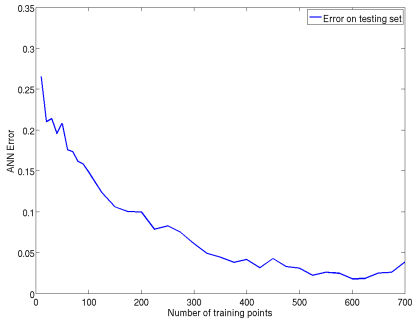

Figure 4: SVR testing error sensitivity to number of training points

We also plot in Fig. 5 and Fig. 6, the sensitivity of an SVR models trained with 200 training points to the value of ϵ and C respectively, as defined in Eq. 3. It is clear from Fig. 5 that epsilon must be chosen to be as small as possible. Indeed, a smaller value of epsilon leads to a smaller testing error. On the other hand,we see from Fig. 6, that the penalty function weight C should be chosen greater than 1 in order to insure minimal testing error.

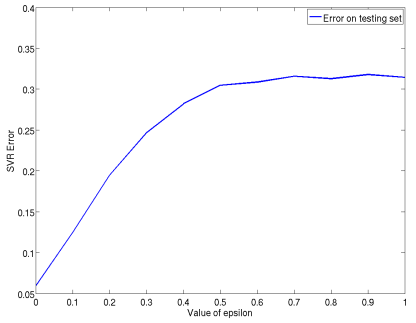

Figure 5: SVR testing error sensitivity to the value of ϵ

Finally we illustrate the use of such proxy models for well placement optimization. We apply a generalized pattern search algorithm where the objective function is the NPV, computed from an ANN proxy model trained with 100 training samples. We show the evolution of the objective function with the number of iterations during the optimization in Fig. 7. The optimal placement found from the optimization is to have the producer well in grid block 15 and the injector well in block 36. This matches exactly the best positions found by exhaustive search.

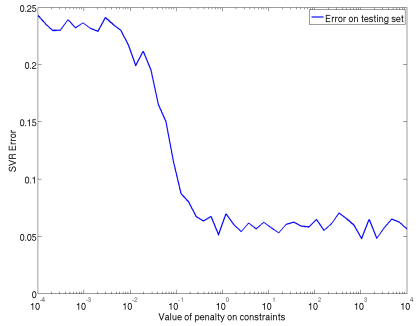

Figure 6: SVR testing error sensitivity to the value of penalty function weight

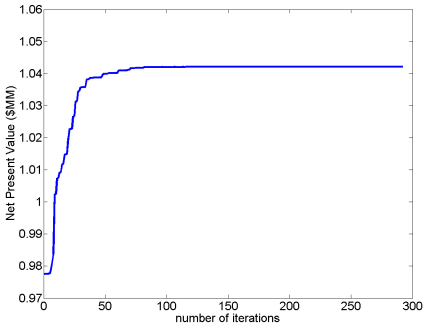

Figure 7: Optimization using Generalized pattern search with an ANN proxy

3.2 Production control

For the same SAGD model as presented in the previous section, we now seek to create a proxy model allowing the optimization of the well controls. For a given position of the wells, we set the pressure for both the producer and injector wells at 5 intervals during the entire production time. So our goal is now to create proxy models that take these 10 parameters as input, and return the net present value as an output. We run 750 simulations using the commercial simulator Stars and label 50 randomly chosen cases as testing samples.

We create a number of artificial neural networks with an increasing number of training samples and plot the testing error for each of these models in Fig. 8. We see that ANN can yield a very low testing error for this problem with less than 200 training samples. We also performed sensitivity analysis for the number of layers and the number of artificial neurons but we didn't find these parameters to be significant to the testing error.

We also build SVR models with increasing numbers of training samples. Results are displayed in Fig. 9. They show that the testing error de-

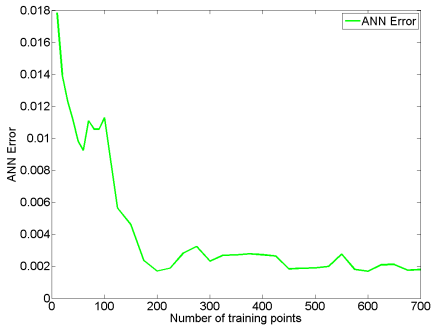

Figure 8: ANN testing error sensitivity to number of training samples

creases very sharply and then becomes stable for more than 10 training samples. This surprisingly sharp decrease of the testing error is probably due to a low sensitivity of the target value (NPV) to many combinations of the input parameters.

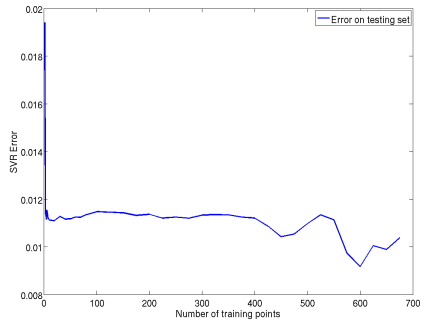

Figure 9: SVR testing error sensitivity to number of training samples

Finally, we illustrate the use of these proxy models by using the generalized pattern search algorithm to perform the optimization of NPV. We plot in Fig. 10 the evolution of the objective value. We verified that the optimized NPV is higher than any NPV found in the 750 simulations performed previously.

4 Conclusions

We used both ANN and SVR to create fast proxy models for complex thermal reservoir simulations. We verified that ANN and SVR can provide robust proxies for the optimization of production controls and well placements. For the two simple cases considered in this work, we found that the number of training simulations required to obtain a small testing error is in the order of 100 to

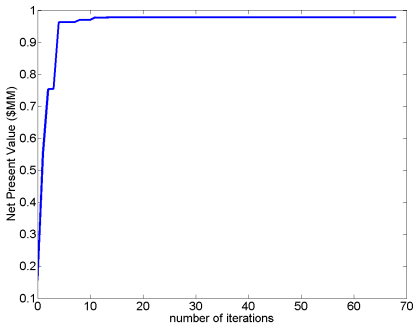

Figure 10: SVR testing error sensitivity to number of training samples

200. We have the used the ANN proxy to perform optimization using the generalized pattern search algorithm. This requires in the order of 100 to 200 simulation runs. In order to obtain a significant runtime speedup using these proxies, the training runs would need to be run in parallel. In concusions, ANN and SVR are promising "black-box" approaches to create fast proxies for optimization when massive parallel computing capability is available.

References

[1] Nestor V. Queipo, Javier V. Goicochea P., and Salvador Pintos. Surrogate Modeling-Based Optimization of SAGD Processes. *SPE*, 2001.

[2] J.W. Vanegas, P. Clayton, V. Deutsch, and L. B. Cunha. Uncertainty assessment of sagd performances using a proxy model based on butler's theory. SPE, 2008.

[3] A. J. Smolatand Bernhard Scholkof. A tutorial on support vector regression. 2003.