Pontifícia Universidade Católica
do Rio de Janeiro

Allan Gurwicz

Deep Generative Models for Reservoir Data: An Application in Smart Wells

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–Graduação em Engenharia Elétrica of the Departamento de Engenharia Elétrica of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica.

Advisor : Prof. Marco Aurélio Cavalcanti Pacheco
Co-advisor: Dr. Ana Carolina Alves Abreu
Co-advisor: Dr. Smith Washington Arauco Canchumuni

Rio de Janeiro
January 2020

Pontifícia Universidade Católica
do Rio de Janeiro

Allan Gurwicz

Deep Generative Models for Reservoir Data: An Application in Smart Wells

Dissertation presented to the Programa de Pós–Graduação em Engenharia Elétrica of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica. Approved by the Examination Committee.

Prof. Marco Aurélio Cavalcanti Pacheco
Advisor
Departamento de Engenharia Elétrica – PUC-Rio

Dr. Ana Carolina Alves Abreu
Co-advisor
Departamento de Engenharia Elétrica – PUC-Rio

Dr. Smith Washington Arauco Canchumuni
Co-advisor
Departamento de Engenharia Elétrica – PUC-Rio

Dr. Alexandre Anozé Emerick
CENPES – Petrobras

Dr. Max de Castro Rodrigues
CENPES – Petrobras

Dr. Luciana Faletti Almeida
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca – CEFET-RJ

Dr. Manoela Rabello Kohler
Departamento de Engenharia Elétrica – PUC-Rio

Rio de Janeiro, January 6th, 2020

Allan Gurwicz

Graduated in Petroleum Engineering from the Pontifical Catholic University of Rio de Janeiro in 2017. Works as a researcher at the Applied Computational Intelligence Laboratory (ICA/PUC-Rio), mainly applying Artificial Intelligence to the Oil & Gas industry.

Bibliographic data

Gurwicz, Allan

Deep generative models for reservoir data: an application in smart wells / Allan Gurwicz; advisor: Marco Aurélio Cavalcanti Pacheco; co-advisors: Ana Carolina Alves Abreu, Smith Washington Arauco Canchumuni. – 2020.

v., 84 f: il. color ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica, 2020.

Inclui bibliografia

1. Engenharia Elétrica – Teses. 2. Redes generativas adversariais. 3. Poços inteligentes. 4. Simulação de reservatório. 5. Aprendizado profundo. I. Pacheco, Marco Aurélio Cavalcanti. II. Abreu, Ana Carolina Alves. III. Canchumuni, Smith Washington Arauco. IV. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. V. Título.

CDD: 621.3

Acknowledgments

I would like to extend my deepest gratitude to:

All my family, especially my parents, Nataly and Michel, for encouraging me to pursue this degree and to continue on this path.

Isabela, for all the love and support which keeps me afloat and thriving everyday.

My advisor, Marco Aurélio, and co-advisors, Ana Carolina and Smith. They helped guide and shape this work, and it definitely would not have come to this were not for them.

Everybody at the Applied Computational Intelligence Laboratory, chiefly the Flexwell team. The project challenges and the great day-to-day at the lab was paramount to achieving this.

All members of the jury, for taking the time to look into and evaluate this work.

CNPq, PUC-Rio and Petrobras, for the financial help during this period.

And to all who may, directly or indirectly, use, benefit from or improve on the methodology here created.

Abstract

Gurwicz, Allan; Pacheco, Marco Aurélio Cavalcanti (Advisor); Abreu, Ana Carolina Alves (Co-advisor); Canchumuni, Smith Washington Arauco (Co-advisor). **Deep Generative Models for Reservoir Data: An Application in Smart Wells**. Rio de Janeiro, 2020. 84p. Dissertação de mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Reservoir simulation, which via complex equations emulates flow in reservoir models, is paramount to the Oil & Gas industry. By estimating the behavior of the reservoir given different input conditions, it allows specialists to optimize various parameters in the oilfield project stage. Alas, the computational time needed for simulations is directly correlated to the complexity of the model, which grows exponentially with each passing day as more intricate and detailed reservoir models are needed, seeking better refinement and uncertainty reduction. As such, optimization techniques which could greatly improve the results of field developments may be made unfeasible. This work proposes the use of deep generative models for the generation of reservoir data, which may then be used for multiple purposes. Deep generative models are systems capable of modeling complex data structures, which after robust training are capable of sampling data following the same distribution of the original dataset. The present application focuses on smart wells, a technology for completions which brings about a plethora of advantages, among which the better ability for reservoir monitoring and management, although also carrying a significant increase in project investment. As such, these previously mentioned optimizations turn indispensable as to guarantee the adoption of the technology, along with its maximum possible return. As to make smart well control optimizations viable within a reasonable time frame, generative adversarial networks are here used to sample datasets after a relatively small number of simulated scenarios. These datasets are then used for the training of proxies, algorithms able to substitute the reservoir simulator and considerably speed up optimization methodologies. Case studies were done in both relatively simple and complex industry benchmark models, comparing network architectures and validating each step of the methodology. In the complex model, closest to a real-world scenario, the methodology was able to reduce the proxy error from an average of 18.93%, to 9.71%.

Keywords

Generative Adversarial Networks; Smart Wells; Reservoir Simulation; Deep Learning.

Resumo

Gurwicz, Allan; Pacheco, Marco Aurélio Cavalcanti; Abreu, Ana Carolina Alves; Canchumuni, Smith Washington Arauco. **Modelos Generativos Profundos para Dados de Reservatório: Uma Aplicação em Poços Inteligentes**. Rio de Janeiro, 2020. 84p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Simulação de reservatório, que por meio de equações complexas emula fluxo em modelos de reservatório, é primordial à indústria de Óleo e Gás. Estimando o comportamento do reservatório dadas diferentes condições de entrada, permite que especialistas otimizem diversos parâmetros na etapa de projeto de campos de petróleo. Entretanto, o tempo computacional necessário para simulações está diretamente correlacionado à complexidade do modelo, que cresce exponencialmente a cada dia que se passa, já que modelos mais detalhados são necessários dada a busca por maior refinamento e redução de incertezas. Deste modo, técnicas de otimização que poderiam significativamente melhorar os resultados de desenvolvimentos de campo podem se tornar inviáveis. Este trabalho propõe o uso de modelos generativos profundos para a geração de dados de reservatório, que podem então ser utilizados para múltiplos propósitos. Modelos generativos profundos são sistemas capazes de modelar estruturas de dados complexas, e que após treinamento robusto são capazes de amostrar dados que seguem a distribuição do conjunto de dados original. A presente aplicação foca em poços inteligentes, uma tecnologia de completação que traz diversas vantagens, dentre as quais uma melhor habilidade de monitoramento e gerenciamento de reservatórios, apesar de carregar um aumento significativo no investimento do projeto. Assim, essas otimizações previamente mencionadas se tornam indispensáveis, de forma a garantir a adoção da tecnologia, junto ao seu máximo retorno. De modo a tornar otimizações de controle de poços inteligentes viáveis dentro de um prazo razoável, redes generativas adversariais são aqui usadas para amostrar conjuntos de dados após um número relativamente pequeno de cenários simulados. Esses dados são então utilizados para o treinamento de aproximadores, algoritmos capazes de substituir o simulador de reservatório e acelerar consideravelmente metodologias de otimização. Estudos de caso foram realizados em modelos referência da indústria, tanto relativamente simples quanto complexos, comparando arquiteturas de redes e validando cada passo da metodologia. No modelo complexo, mais próximo de um cenário real, a metodologia foi capaz de reduzir o erro do aproximador de uma média de 18.93%, para 9.71%.

Palavras-chave

Redes Generativas Adversariais; Poços Inteligentes; Simulação de Reservatório; Aprendizado Profundo.

Table of contents

List of figures

List of tables

List of Abbreviations

B-DCGAN – Boundary-Seeking Deep Convolutional Generative Adversarial Network
CAPEX – Capital Expenditure
CNN – Convolutional Neural Network
DCGAN - Deep Convolutional Generative Adversarial Network
GAN – Generative Adversarial Network
LSTM – Long Short-Term Memory
MAE – Mean Absolute Error
MSE – Mean Squared Error
NPV – Net Present Value
NRMSE – Normalized Root Mean Squared Error
OPEX – Operational Expenditure
ReLU – Rectified Linear Units
RMSE – Root Mean Squared Error
RNN – Recurrent Neural Network
tanh – Hyperbolic Tangent
W-DCGAN – Wasserstein Deep Convolutional Generative Adversarial Network

1
Introduction

Field development optimizations are cardinal to the Oil & Gas industry, as to guarantee projects are viable and optimally profitable. While there are numerous areas to be optimized, there is the need in all to evaluate the influence of each possibility in the optimization, in regards to the field as a whole.

Reservoir simulation emerges as a way to model the behavior of the reservoir given different input conditions and geological information, by emulating flow via complex equations. As such, the optimization possibilities may be evaluated without the need for real field implementation.

However, these intricate equations make the process highly computer-intensive, requiring great computing power and encumbering these optimizations. Thus, reservoir simulator proxies arise as a method to draw on the behavior modeling ability of the simulator, while foregoing the need for the intensive computing and overcoming its shortcomings.

Alas, these supervised algorithms still need to be trained on a dataset constituted by a significant amount of simulations, and as such the need for simulations remains crucial in the context of optimizations. As such, studies in the area of simulator substitution are paramount as to ensure faster and more efficient optimizations, in turn ensuring more profitable field developments.

In order to ease the creation and validation of the methodology here proposed, the smart well control optimization area was chosen, as an attempt to diminish the disadvantages of reservoir simulation in a specific optimization medium.

Since first implemented in a field, in 1997, smart wells have increasingly been applied to various oilfield development projects across the globe (Gao et al., 2007). Employed in a myriad of scenarios, ranging from mature to cutting-edge projects, they are not only able to induce viability in non-economic developments, but to increase the value of economic ones.

Improving on conventional wells, they include technologies capable of remote measurement and control (Abreu, 2016). This allows for better reservoir and production management, as well as optimizations aiming for the increase of project net present value (NPV), raises in oil and gas and reduction in water productions, among other common objectives.

Alas, these benefits come associated to a main downside: the considerable increase in investment for the application of the technology. While its use, guided by specialists, was proven enough for overturning this increase, control optimizations become indispensable and lead to even more attractive projects.

1.1 Motivation

There are countless optimization techniques applied to smart well control in the Oil & Gas literature. With different methodologies, they are able to prove the technology is worth using by showcasing these increases and reductions in the aimed for objectives.

The main friction point in the use of these methodologies, and a point shared by all, is the need for expensive objective function calculations. As they mostly rely on reservoir simulation, which is highly computer resource-intensive, many are unfeasible and while theoretically valid, are shy from being routinely applied.

The following subsections illustrate the context in which this work is inserted, and the motivation for its development.

1.1.1 Flexwell

The idea for the present work was born after the need for reduction in the number of simulations was found in the context of the Flexwell project, a Research & Development project developed in a partnership between the Applied Computational Intelligence Laboratory, of PUC-Rio, and Petrobras.

Consisting in the development of a methodology and software capable of estimating the value of the flexibility and information under uncertainty brought by the use of smart wells, the core of the work is based on Abreu (2016).

The software is robust to optimizing smart well controls in certain different manners, as Abreu et al. (2018) describes in detail. A small summary follows, illustrated by Figures 1.1 to 1.4.

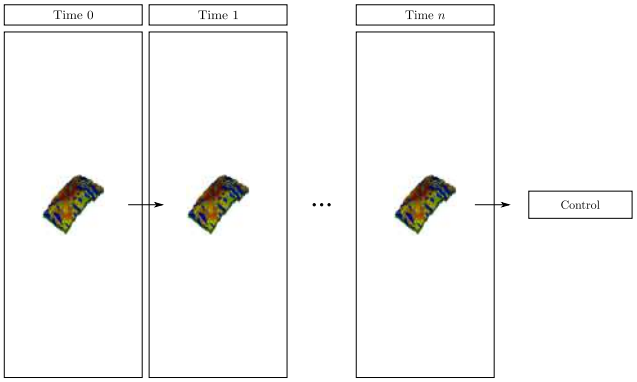

Figure 1.1: Illustration of the optimization without uncertainty.

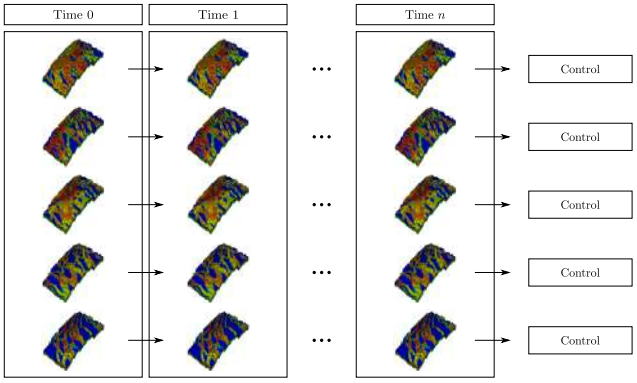

Figure 1.2: Illustration of the optimization under uncertainty with clairvoyance.

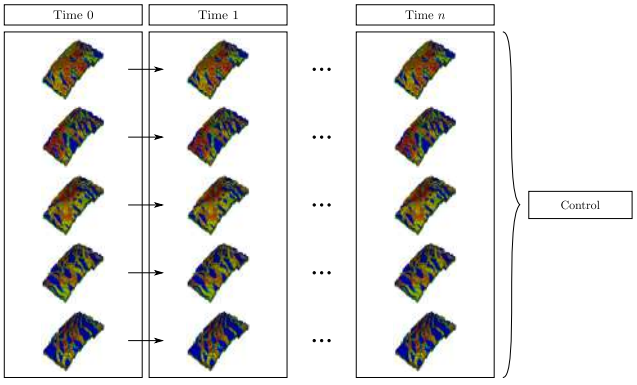

Figure 1.3: Illustration of the optimization under uncertainty without future information.

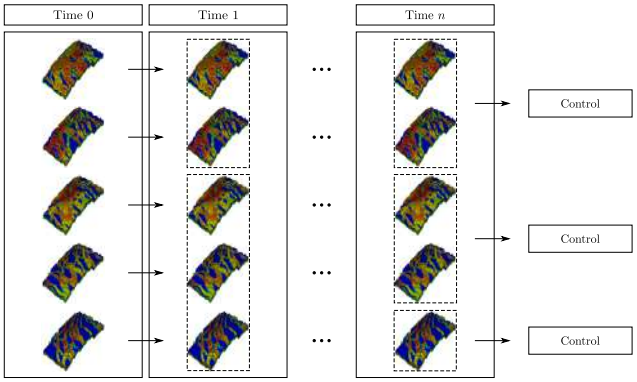

Figure 1.4: Illustration of the optimization under uncertainty considering future information.

– **Optimization without uncertainty**

  This case works with a single reservoir model, assuming that all reservoir properties are true. The optimization seeks the strategy that maximizes the objective function for this single realization.

– **Optimization under uncertainty with clairvoyance**

  Here, the optimization takes into account different geological scenarios, representing the reservoir uncertainty. An individual optimization is done per scenario, and each one has a different optimal control schedule.

– **Optimization under uncertainty without future information**

  While this case considers multiple reservoir realizations, a single optimization is done, seeking to maximize the expected objective function, that is, the average of all objective functions.

– **Optimization under uncertainty considering future information**

  This approach assimilates information acquired on the reservoir through the optimization. This is done by clustering the scenarios in regards to predetermined measurements in certain timesteps, and an optimal control is found for each cluster as a whole.

All these optimization methods are tightly bound by the need for numerous reservoir simulations, corroborating the need, in the context of this project, for reducing the amount of these expensive simulations.

Figure 1.5 illustrates an example of reservoir simulation in this context, where the simulator receives as inputs the control settings, represented by the boxes, besides the geological information, represented by the reservoir model, and outputs production curves.

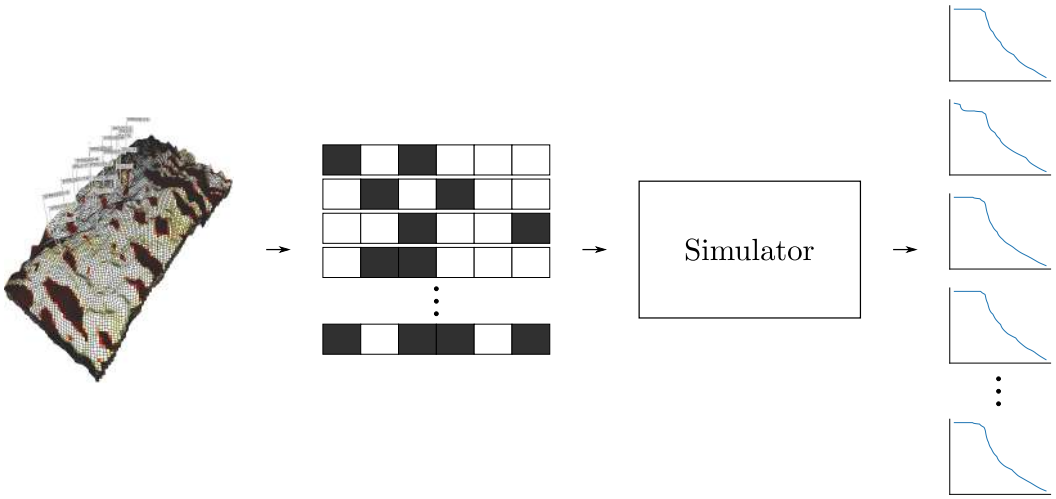

Figure 1.5: Illustration of the reservoir simulation process.

1.1.2 Reservoir Proxies

Aiming for the reduction in the need for simulations, proxy models emerge as a way to mimic the simulator's behavior. While physics-based proxies incorporate the mathematics of fluid flow, data-driven based proxies make no assumptions on the underlying data, using it for the prediction of future data (Aïfa, 2014). There is a plethora of reservoir proxies in the literature, such as the following examples.

Calvette et al. (2019) used long short-term memory networks for the prediction of smart well production data, using the controls and productions as inputs to the proxy. The methodology was tested in two reservoir models, obtaining good results. Kohler (2013) applied multilayer perceptrons to production prediction, with well coordinates and productions as inputs. They used Latin hypercube sampling to create datasets for two case studies, reducing input dimension with principal component analysis and finding good result metrics. Navratil et al. (2019) used an encoder-decoder architecture receiving drilling actions, reservoir properties and productions to predict production rates. The case studies presented low error rates and proved the speedup in relation to conventional simulation. Mohaghegh et al. (2015) introduces a "Surrogate Reservoir Model", which is a so called smart proxy. It receives information on each individual gridblock of the reservoir, and was used for the prediction of oil rates, bottom-hole pressures, gas oil ratios and water cuts. They applied the model to a giant mature oilfield, and not only achieved good results, were able to use the proxy for production optimization. Cao et al. (2016) applied a neural network-based proxy to real field collected production

and pressure data, acceptably predicting future production.

There are plenty more published examples available, with assorted inputs and outputs. Yet, a shared characteristic between the majority is the need for extensive simulations to build the dataset for training the networks, excluding the ones based on real world collected data.

Some of the examples presented try to mitigate this, such as Kohler (2013) which intelligently chooses scenarios for the dataset via Latin hypercube sampling, and Mohaghegh et al. (2015) which, by considering each grid-block individually, can generate considerable datasets with a low amount of simulations.

Figure 1.6 illustrates the simulator proxy training process for the smart well control context, where the simulator is used for the building of a dataset, which is then used for the training of the proxy. Figure 1.7 shows the inference process, where the trained proxy may be used in place of the simulator to return the outputs based on the same inputs the simulator would have received. Both figures mantain the representations of Figure 1.5.

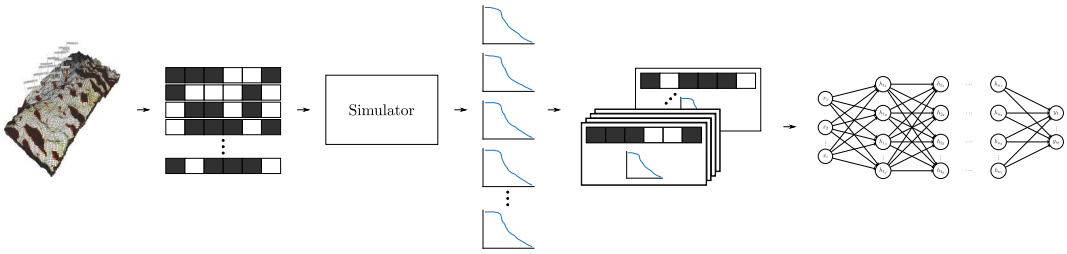

Figure 1.6: Illustration of the proxy training process.

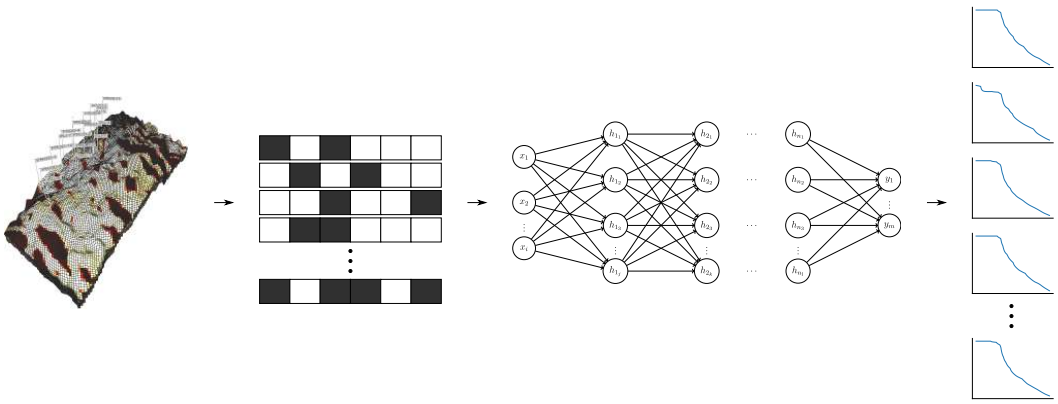

Figure 1.7: Illustration of the proxy inference process.

1.2 Proposed Methodology

While theoretically helping in the reduction of simulations, some ideas are not applicable to all methodologies, relying on the type of the underlying data, or have not been tested as simulator-use reducing methodologies.

A great downside in the use of proxies is the fact that, while reducing simulator use after its construction, a great number of simulations are first needed for training. As such, the total number of simulations might not necessarily be reduced with the use of these techniques.

Deep learning, aiming to ease the objective of allowing computers to learn from input, harnesses hierarchical algorithms as to represent complex concepts in data. It differs from traditional shallow networks in that the amount of stages for learning, where each stage transforms the activation of the network, is high (Schmidhuber, 2015). The recent availability of high computing power allowed the application of these networks to spread, and proved their superiority in a variety of tasks.

Within the context of data generation, there is a great quantity of methods available in the literature. Deep generative models emerge as a way to represent probability distributions over a set of variables, ideally being able to sample from these distributions (Goodfellow et al., 2016). These models are able to capture complex structure in the data, allow for fast sampling, and are computationally viable and scalable (Rezende et al., 2014).

Generative models have been used and validated for images (Goodfellow et al., 2014; Radford et al., 2016), music (Mogren, 2016; Yang et al., 2017) and various time-series data (Esteban et al., 2017; Hartmann et al., 2018; Yahi et al., 2017), among others. In the Oil & Gas domain, there is use in history matching (Canchumuni et al., 2019), the generation of pore and reservoir-scale models (Mosser et al., 2018), and work focusing on geophysics (Mosser et al., 2018) and geology (Dupont et al., 2018).

The present work introduces a methodology which makes use of deep learning and deep generative models to, with a relatively low number of simulations, reduce the need for simulations in the training of reservoir proxies. As such, other optimization methodologies can benefit from the fast objective function calculation of proxies, while not sacrificing computer power for extensive simulations in their training.

A generative adversarial network is proposed as a way to, after initial training, generate scenarios representative of the reservoir in question, mantaining input data diversity. Thus, a smaller number of simulations are needed for the construction of this network, which then generates new scenarios for

training proxies and enabling other optimization methodologies.

The present work proposes the methodology and studies its validity, by applying datasets sampled by the generator to long short-term memory network-based proxies. The work is divided into the following chapters:

- Chapter 2 introduces a background on the petroleum engineering themes related to the application of the present work, including an explanation on smart well technology and control optimization literature.
- Chapter 3 outlines the relevant deep learning concepts, as to better understand the networks and techniques used in this work.
- Chapter 4 describes in detail the proposed methodology.
- Chapter 5 details the application of the methodology to case studies, aiming for its validation and the justification of its use.
- Chapter 6 analyzes and draws conclusions from the results of the previous chapter, then suggesting future work for further improvement of the methodology.

2
Smart Wells

In order to better understand the application of the present work, this chapter aims to provide a background on the relevant petroleum engineering areas and topics.

2.1
Completions

The set of operations and equipments done and used in the well after its drilling, aiming to make it fit and safe for use, is named completions. In this crucial step, a great number of equipments are traditionally installed, in order to optimally, efficiently and safely produce oil and gas from the intended reservoir.

Thomas et al. (2001) lists the step-by-step of a conventional offshore completion:

- Installation of surface equipment, including the wellhead, blowout preventer and the christmas tree.
- Cleaning of the well and insertion of the completion fluid.
- Testing of the cementing.
- Perforation of the well.
- Installation of the production string and tubing, including the chosen artificial lift equipment.

2.2
Smart Completions

While the conventional completion previously described is usually enough for oilfield development, the increase in decline rates and a recent industry push for more complex projects exposed the need for the use of new, innovative technology. This led to the first use of smart completions, in 1997 (Gao et al., 2007).

Since then, the technology has become widespread, being applied in a myriad of fields all around the world. Lots of case studies can be found in

the literature, such as the ones presented in Anderson (2005), from Brazil, Wulandari et al. (2015), from the North Sea, Jeu et al. (2008), from the Gulf of Mexico, Chan et al. (2014), from Malaysia, and Al-Shenqiti et al. (2007), from Saudi Arabia.

Robinson (2003) defines smart completions as "a completion system capable of collecting, transmitting, and analyzing wellbore production, reservoir, and completion-integrity data, then enabling remote action to enhance reservoir control and well-production performance". The following components are present in the intelligent well, enabling these objectives, as per Konopczynski et al. (2003):

- Flow control devices, being able to control production binarily, discretely or in a infinitely variable way.
- Feedthrough isolation packers, which segregate different reservoir zones and allow isolated control.
- Control, communication and power cables, transmitting power and data to and from downhole monitoring and control devices.
- Downhole sensors, used to individually monitor well parameters from all zones of interest, such as flow, pressure, temperature, phase composition and water pH (Glandt, 2005).
- Surface data acquisition and control systems, which acquire, validate, filter and store the real-time data obtained by the sensors.

Figure 2.1 shows a simplified example of a smart well. The different patterns represent different reservoir zones, the crossed boxes illustrate the packers, isolating the flow between zones of interest, the dashed lines represent the perforations, through which the reservoir fluids enter the well, and the red boxes represent the flow control devices, able to control production.

Establishing a link to the figures in Chapter 1, the opening settings for each one of these flow control devices, in all smart wells of the reservoir, compose the control vector given as input to the simulator or proxy.

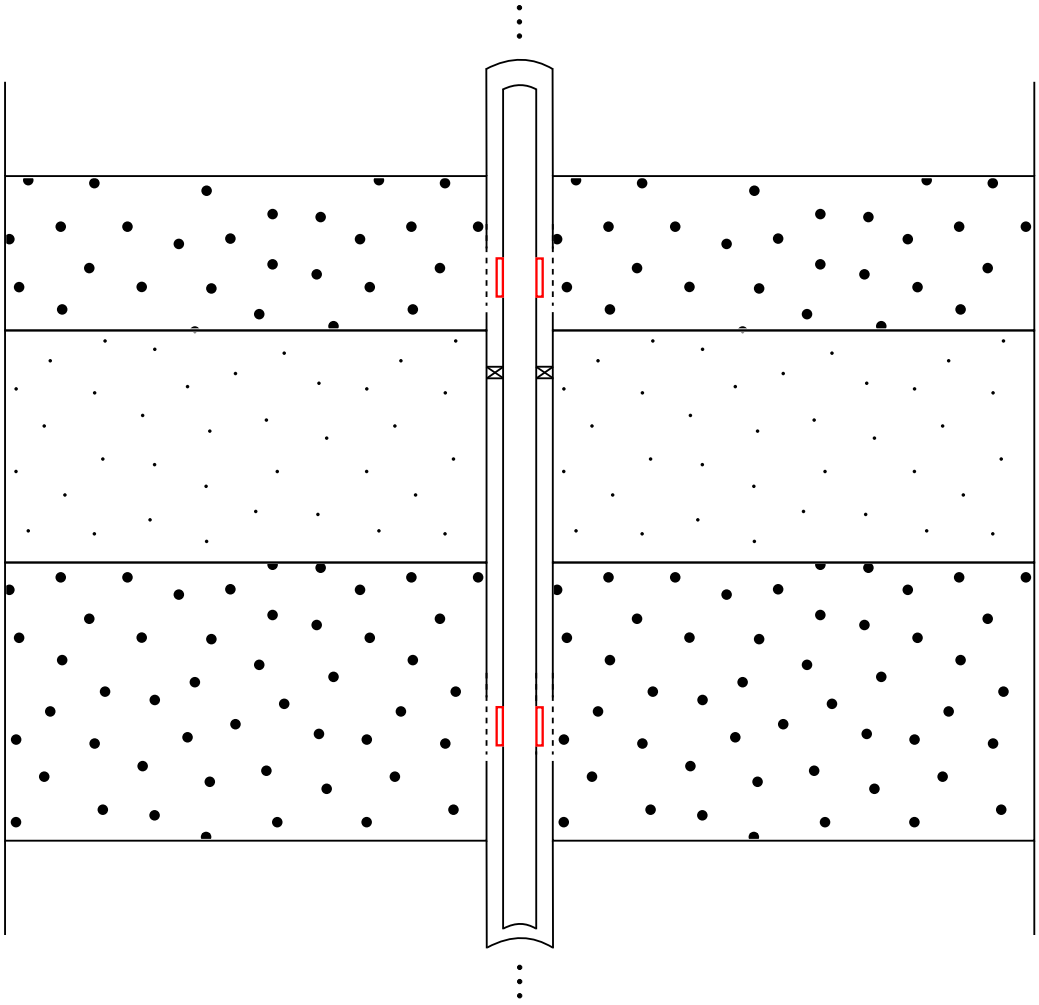

Figure 2.1: Illustration of a smart well.

2.3 Control Optimization

Even though the application of smart wells in a development project incurs in an increase of project cost, in the form of capital and operational expenditures (CAPEX and OPEX), Pari et al. (2009) illustrates how its use may lead to increases in total project net present value (NPV). While this might be enough for the justification of its use, as Schiozer and Silva (2009) shows, the comparision to conventional wells is only fair if done after production strategy optimization, as it leads to even further considerable gains in NPV. There is a great amount of work in smart well control optimization and gain quantification. Some examples follow.

Abreu (2016) developed a methodology for estimating the value of the flexibility given by the technology and seeking the optimal flow control strategy

under uncertainties. A case study was done in the UNISIM-I reservoir (Avansi and Schiozer, 2015), which is based in a Brazilian field, and the methodology was able to increase NPV. Emerick and Portela (2007) optimized valve settings by using direct search methods. Two case studies were done in Brazilian offshore fields, obtaining increases in NPV and oil production. Su and Oliver (2010) used ensemble-based optimization for the valve control aiming for the reduction of water production. Two complex case studies showed that the methodology was successful. Durlofsky and Aziz (2002) applied a gradient-based optimization algorithm, which was able to optimize valve settings, leading to increases in oil recovery, for three case studies. Almeida et al. (2007) used evolutionary algorithms for optimizing the controls. A synthetic reservoir-based case study revealed an increase in NPV and oil production, and a reduction in water production, with the application of the methodology. Ilamah and Waterhouse (2018) employed a proactive optimization approach, using genetic algorithms and local searches, to a case study on a real North Sea field, after representative scenario selection. The resulting control settings induced an increment in oil production for all model realizations, even the ones not considered in the optimization.

2.4 Reservoir Simulation

A common point in the majority of methodologies aiming for smart well control optimization is the need for reservoir simulation, as to calculate objective functions.

Reservoir simulators are numerical flow simulators, which by using the finite difference method to solve complex equations, coupled to information inputted on geology and wells, are able to model the behavior of the reservoir (Rosa et al., 2006).

They receive as input a model of the reservoir, divided into gridblocks, which contain information on properties such as porosity and permeability, among others. The simulator is then responsible for solving a set of equations, such as Darcy's law and material balance equations, to model flow between adjacent blocks, for all gridblocks in the reservoir. This is done in discrete increments, called timesteps, whose division directly influences the accuracy of the simulator (Mattax and Dalton, 1990).

As the present work focuses on data-driven models and only uses reservoir simulation as a means to achieve production results, the simulation concept may be simplified by considering the simulator a black-box. This means that, after receiving the required inputs, it outputs chosen variables, such as oil, gas

and water productions, and little attention needs to be given to how that is accomplished.

The models used for simulation hike in complexity with each passing day, aiming for better representation of real reservoirs with more refined and detailed grids, property distributions and further timestep discretization. While this leads to more accurate output data, computational cost is also greatly increased. As such, many of the optimization methodologies previously described might become unfeasible, depending on the complexity of the reservoir and available computing power (Kohler, 2013).

3
Deep Learning

Deep learning is a subarea of machine learning, which, in turn, is a subarea of artificial intelligence. Machine learning is the subject which focuses on allowing computers to learn from input, i.e., converting experience or training data to expertise or knowledge (Shalev-Shwartz and Ben-David, 2014). Deep learning is, in a simplistic definition, the approach consisting in facilitating this learning by showcasing the world in terms of a hierarchy of concepts, where complex concepts are built out of simpler ones (Goodfellow et al., 2016).

As the world of machine learning is nearly endless, this chapter aims to provide a background focused in the deep learning topics needed for better understanding the created methodology, and the concepts on which it stands.

3.1
Multilayer Perceptrons

The networks we are used to nowadays are made possible by a foundation built all the way back in the middle of the 20th century. McCulloch and Pitts (1943) defined the first artificial neuron, as seen in Figure 3.1.

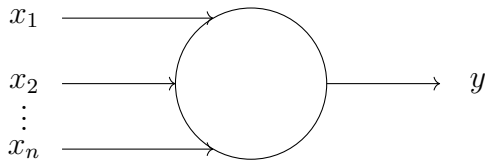

Figure 3.1: Simplified illustration of the McCulloch-Pitts neuron.

It works by applying a sum to the boolean inputs and activating the neuron based on a comparison to a threshold value. That is, if $\sum_{i=1}^{n} x_i \geq \theta$, $y = 1$, where θ is the predefined threshold value. Else, $y = 0$. It also has an inhibitory input, i, which, if on, impedes the neuron from firing.

The model was made closer to what we are used to, nowadays, by the perceptron (Rosenblatt, 1958), as seen in Figure 3.2.

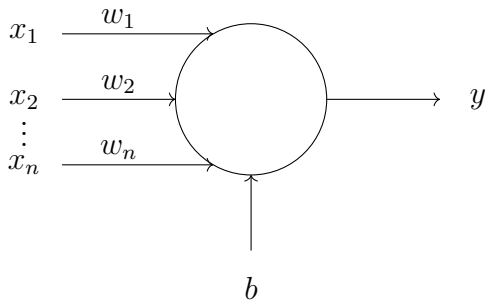

Figure 3.2: Simplified illustration of a neuron in the Rosenblatt perceptron.

The neuron now has w weights and a b bias parameter. It may be used as a linear classifier, where the output y is a function of $\sum_{i=1}^{n} x_i w_i + b$, with a decision boundary given by $\sum_{i=1}^{n} x_i w_i + b = 0$.

Rosenblatt (1958), improving on Hebb (1949), also came up with a learning algorithm, which updates the weights and the bias iteratively.

As to greatly simplify the history of deep learning, in the coming decades, improvements were made to the model:

- Perceptrons can be stacked in a single layer, as to provide multiple outputs.
- These layers of perceptrons can now be stacked, giving origin to the multilayer perceptron.

This model, which is still vastly used nowadays, can be seen in Figure 3.3.

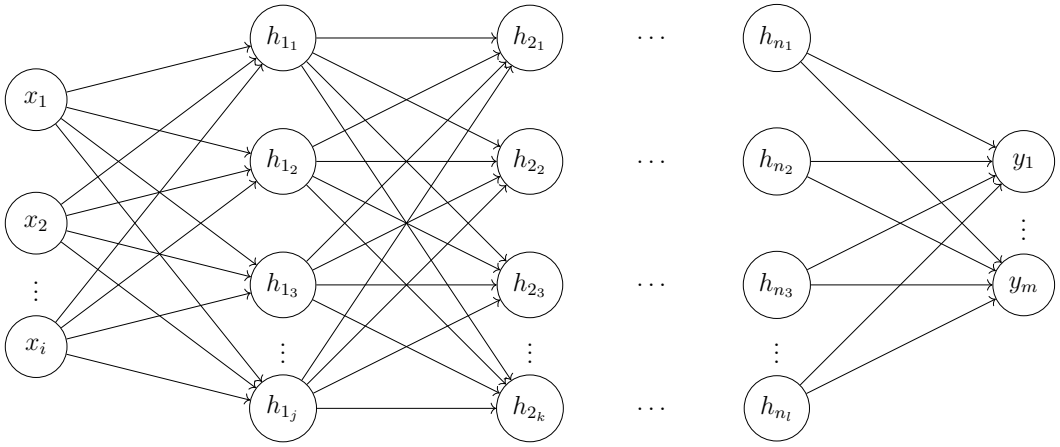

Figure 3.3: Simplified example of a multilayer perceptron.

The output of the y_m neuron, for example, is given by (3-1). The same equation is applicable to the output of all neurons in the network.

$$y_m = f_y\left(\sum_{o=1}^{l} w_{om} h_{n_o} + b_m\right) \tag{3-1}$$

Where w_{om} is the weight of the connection between neuron o of the nth hidden layer and neuron m of the output layer, h_{n_o} is the output of neuron o of the nth hidden layer, b_m is the bias and f_y is the activation function of the output layer.

This activation function is differentiable and usually non-linear, so the network may learn from complex data. Some examples are the sigmoid (σ), hyperbolic tangent (tanh), rectified linear units (ReLU) and leaky ReLU functions, as seen in Figure 3.4.

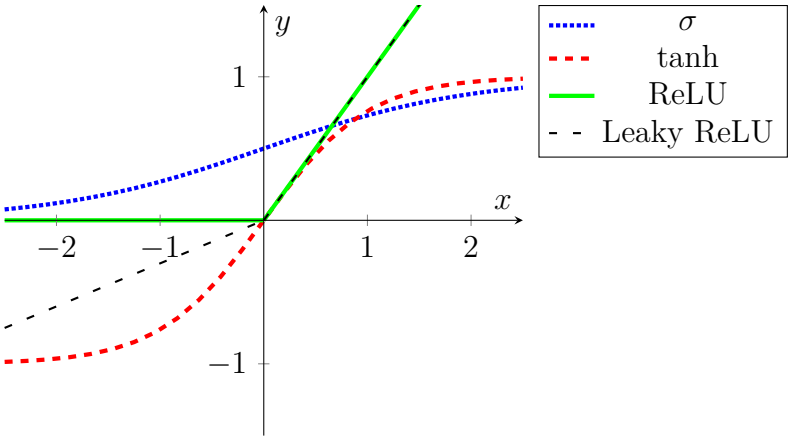

Figure 3.4: Graphs of commonly used activation functions.

3.2 Loss Functions and Optimizers

In order to update the weights and bias in this complex network, new learning rules need to be used. As this work's focus is not on deep learning intricacies, the math behind this section will be skipped in favor of a general explanation.

The main objective of the network, as a supervised learning algorithm, is to act as an universal approximator (Hornik et al., 1989), i.e., correctly giving outputs based on inputs and as per the data observed. As a way to measure the accuracy of this action, cost or loss functions are used to calculate the distance between the output of the network and the true desired output.

Some examples of loss functions are the mean squared error (3-2), mean absolute error (3-3) and crossentropy (3-4), among others.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3-2}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3-3}$$

$$crossentropy\ loss = -\sum_{i=1}^{n}\sum_{j=1}^{k} y_{ij}\ln(\hat{y}_{ij}) \tag{3-4}$$

Where y are the true values, $\hat{y}$ are the outputs of the network, n is the amount of data, and k is the number of existing classes.

Optimizers are used to minimize these functions, updating the weights and biases of the network as to better approximate the data. It is also important to mention backpropagation, that is, while the inputs are fed-forward through the network, the errors are "backpropagated", from the output to the input layers. Some examples of optimizers follow.

- **Gradient Descent**

  Gradient descent updates the parameters of the network based on the partial derivative of the cost funcion in regards to each parameter, and a predefined learning rate. This update is done once for each pass through the whole dataset.

- **Stochastic Gradient Descent**

  This algorithm is different from the previous one in that the parameters are updated once per training example.

- **Mini-Batch Gradient Descent**

  Here, a mini-batch number is defined, and the parameters are updated once per mini-batch of training examples.

- **Adagrad**

  This algorithm, developed by Duchi et al. (2011), introduces an adaptive learning rate. That is, each parameter has a different learning rate, and the updates are done based on its importance.

- **Adadelta and RMSprop**

  Zeiler (2012) and Hinton et al. (2012) respectively came up with these algorithms, which solve a problem that Adagrad has of decreasing learning rates.

- **Adam**

  Kingma and Ba (2014) adds a momentum-like term to the previous algorithms, empirically improving results.

3.3 Convolutional Neural Networks

While Fukushima (1980) is given credit by Schmidhuber (2015) for introducing this architecture, or at least a predecessor of it, the literature usually associates the birth of modern convolutional neural networks (CNNs) to Yann LeCun (LeCun et al., 1989, 1990, 1998).

Typical CNNs are made of convolutional, pooling and fully connected layers, as seen in Figure 3.5.

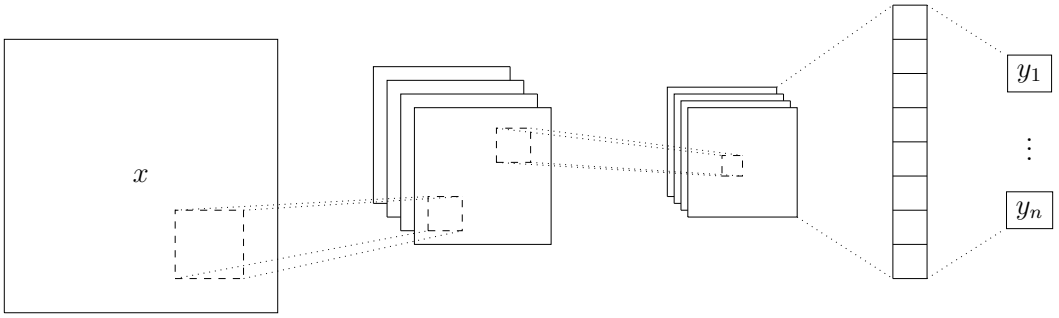

Figure 3.5: Simplified example of a convolutional neural network.

The convolutional layers are responsible for extracting features from the input data, by applying a filter, also called kernel. The kernel strides through the data, performing a Hadamard product, that is, elementwise multiplications, and storing the results on the feature, or activation, maps. This is followed by the application of a non-linearity on the resulting data, such as ReLU or others.

The pooling layers reduce the size of the feature maps, mantaining only the most important information while merging similar features. The most common type is the max pooling, which takes the maximum value in each predefined window from the activation maps.

These layers and operations may be repeated multiple times, as seen in the state-of-the-art (LeCun et al., 1998; Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; He et al., 2015).

The output of the last pooling layer is then fed into a multilayer perceptron, also called a fully connected or dense layer. It may act as a classifier, for example, outputting probabilities for each class, in one of the uses of these networks.

3.4 Recurrent Neural Networks

Recurrent neural networks (RNNs) introduce a time dependency to the architecture. A simple example can be seen in Figure 3.6.

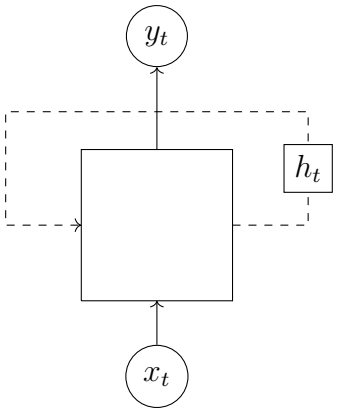

Figure 3.6: Simplified example of a recurrent neural network.

The network is divided in timesteps. At each, the hidden state h_t is updated by (3-5), and the output y_t is given by (3-6).

$$h_t = f_h(w_h h_{t-1} + w_x x_t + b_h) \tag{3-5}$$

$$y_t = f_y(w_y h_t + b_y) \tag{3-6}$$

Where w are the network weights, x are the inputs, b are the biases and f are the activation functions.

This imbues the network with a so called memory, remembering previous inputs to make future predictions.

3.4.1 Long Short-Term Memory Networks

While simple RNNs perform satisfactorily for some time-reliant tasks, their memory is short lived. While previous inputs influence close future timesteps, this is not true for the long-term, due to a problem of exploding or vanishing gradients (Hochreiter, 1998).

As to address this problem, Hochreiter and Schmidhuber (1997), complemented by Gers et al. (1999), came up with the long short-term memory (LSTM) architecture, as seen in Figure 3.7.

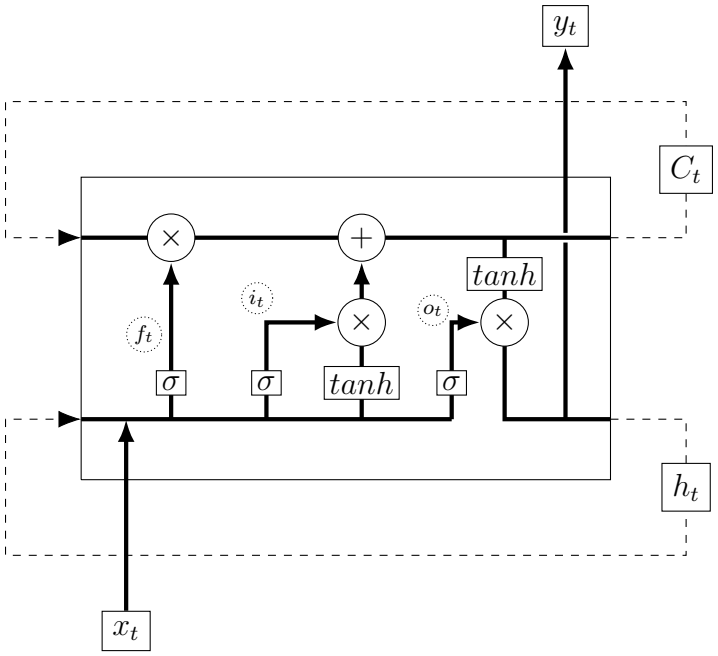

Figure 3.7: Simplified example of a long short-term memory network unit.

The architecture is comprised of a cell state (C), which is updated by gates. The forget gate (f) is responsible for deciding how much of the previous timestep is kept (3-7). The input gate (i) decides which values to update, and creates new values (3-8). Then, the cell state is updated (3-9), and the output gate (o) decides what to output and pass to the next timestep (3-10).

$$f_t = \sigma(w_f \cdot [h_{t-1}, x_t] + b_f) \tag{3-7}$$

$$\begin{aligned} i_t &= \sigma(w_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= tanh(w_c \cdot [h_{t-1}, x_t] + b_C) \end{aligned} \tag{3-8}$$

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t \tag{3-9}$$

$$\begin{aligned} o_t &= \sigma(w_o \cdot [h_{t-1}, x_t] + b_o) \\ y_t &= h_t = o_t tanh(C_t) \end{aligned} \tag{3-10}$$

All of this allows this architecture to handle long time dependencies, improving on the vanilla RNN.

3.5 Generative Adversarial Networks

Generative Adversarial Networks (GANs), conceived by Goodfellow et al. (2014), belong to a class of networks named deep generative models.

The model is composed of two networks, a generator and a discriminator, both multilayer perceptrons in the original formulation. They are pitted against each other, where the generator aims to fool the discriminator into thinking its generated samples are real, while the discriminator aims to correctly classify real and fake samples.

The discriminator D and the generator G play a zero-sum game (3-11). In the original formulation, the proposed value function V is given by (3-12).

$$\min_G \max_D V(D,G) \tag{3-11}$$

$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log(D(x))] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))] \tag{3-12}$$

Where $p_z(z)$ is the known distribution of input noise variables, which are given to the generator that has a distribution p_g, and $p_{data}(x)$ is the original data distribution.

Figure 3.8 illustrates the general structure of the GAN, and the relationship between networks.

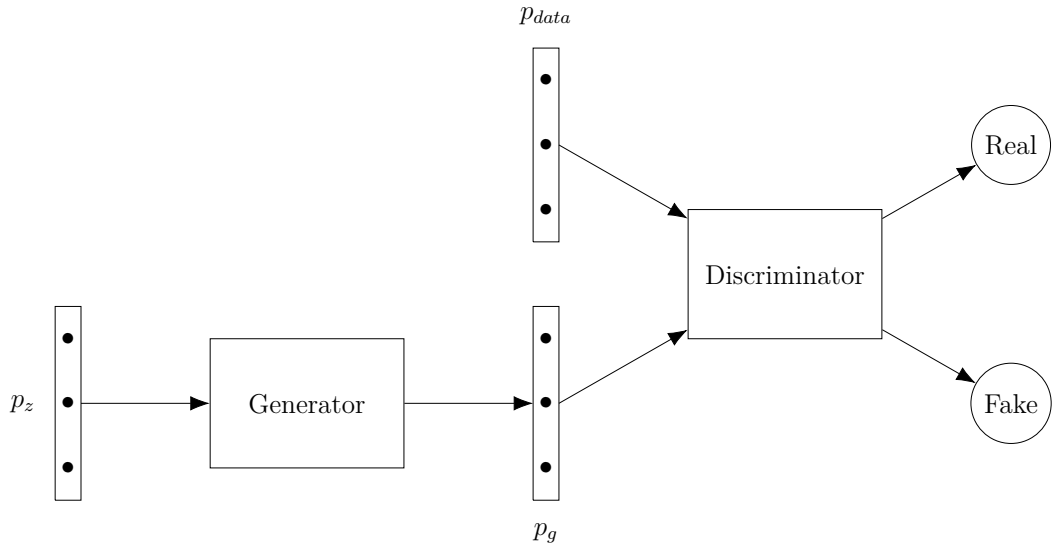

Figure 3.8: Structure of the generative adversarial network.

Goodfellow et al. (2014) proves that $p_g = p_{data}$ is a global optimum and Nash equilibrium of the game, per the following:

– The discriminator aims to maximize, while the generator aims to minimize, $V(G,D) = \int_x p_{data}(x)\log(D(x))dx + \int_z p_z \log(1 - D(G(z)))dz = \int_x p_{data}(x)\log(D(x)) + p_g(x)\log(1 - D(x))dx$;

– As $\frac{a}{a+b}$ is the maximum of $a\log(x) + b\log(1-x)$ in $[0,1]$ for any $(a,b) \in \mathbb{R}^2\backslash\{0,0\}$, $V(G,D)$ is maximum when $D(x) = D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$, for any fixed generator;

– Assuming the optimal discriminator, $V(G, D^*_G(x)) = \mathbb{E}_{x\sim p_{data}}\left[\log\left(\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right)\right] + \mathbb{E}_{x\sim p_g}\left[\log\left(\frac{p_g(x)}{p_{data}(x)+p_g(x)}\right)\right] = \mathbb{E}_{x\sim p_{data}}\left[\log\left(\frac{\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}}{2}\right)\right] + \mathbb{E}_{x\sim p_g}\left[\log\left(\frac{\frac{p_g(x)}{p_{data}(x)+p_g(x)}}{2}\right)\right] - \log(4) = KL\left(p_{data}, \frac{p_{data}+p_g}{2}\right) + KL\left(p_g, \frac{p_{data}+p_g}{2}\right) - \log(4) = 2\cdot JSD(p_{data}, p_g) - \log(4)$, where KL is the Kullback-Leibler divergence and JSD is the Jensen-Shannon divergence.

– As $JSD(a,b) \geq 0\ \forall a,b$ and $JSD(a,b) = 0$ if $a = b$, the global minimum of $V(G, D^*_G) = -\log(4)$ happens when $p_g = p_{data}$, and both discriminator and generator are at their optimum and have no incentives for changing.

The paper states that while theoretically guaranteed to achieve this optimum, in practice this doesn't happen and both networks' losses keep oscillating during training, unable to find the equilibrium.

Other problems that happen in GAN training include mode collapse, where the generator only produces samples from a single mode of the distribution, and vanishing gradients, where the discriminator is so good that the generator cost saturates and it isn't able to learn.

Several methodologies exist in the literature aiming for the mitigation of these problems. Salimans et al. (2016) lists some, including minibatch discrimination, where the discriminator looks at multiple data in combination, one-sided label smoothing, consisting in replacing the targets from 0 and 1 to smoothed values like 0.1 and 0.9, and virtual batch normalization, where the inputs of each mini-batch are normalized.

Other methodologies include alterations to or new cost functions, such as Goodfellow et al. (2014), the original GAN paper, which suggests using the maximization of $\log(D(G(z)))$ as the generator objective.

3.5.1 Deep Convolutional Generative Adversarial Networks

Not only multilayer perceptrons, as in the original paper, may be used in the layers of the generator and discriminator. Radford et al. (2016) proposed the Deep Convolutional GAN (DCGAN) architecture, using convolutional networks.

They also list guidelines for stable training of the network, improving on previous convolutional GANs.

3.5.2 Wasserstein Generative Adversarial Networks

Arjovsky et al. (2017) proposed a new cost function for training GANs, titled Wasserstein distance, or Earth-Mover distance, given by (3-13). It represents the cost of the optimal transport plan of "mass" from distribution p_{data} to distribution p_g.

$$W(p_{data}, p_g) = \inf_{\gamma \in \Pi(p_{data}, p_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[||x - y||\right] \tag{3-13}$$

Where $\gamma(x, y)$ represents the aforementioned transport plans, belonging to the set $\Pi(p_{data}, p_g)$.

They also show that while this calculation may be intractable, the function can also be given by (3-14).

$$W(p_{data}, p_g) = \frac{1}{K} \sup_{||f||_L \leq K} \mathbb{E}_{x\sim p_{data}} \left[f(x)\right] - \mathbb{E}_{x\sim p_g} \left[f(x)\right] \tag{3-14}$$

Where $||f||_L \leq K$ means the function f should be K-Lipschitz continuous, i.e., there exists a real constant K such that $|f(x_1) - f(x_2)| \leq K|x_1 - x_2| \; \forall x_1, x_2 \in \mathbb{R}$.

The authors suggest using (3-15), where the function f_w may be a neural network with weights w.

$$\max_{w \in W} \mathbb{E}_{x\sim p_{data}} \left[f_w(x)\right] - \mathbb{E}_{z\sim p_z} \left[f_w(G(z))\right] \tag{3-15}$$

As such, the discriminator is not responsible for telling real from fake samples apart, but for learning a function for computing the Wasserstein distance between both distributions.

As for the question of enforcing K-Lipschitz continuity, the methodology enforces weight clipping, where the weights of the network are clamped to a compact space W and the continuity is mantained. This also helps with vanishing gradients.

3.5.3 Boundary-Seeking Generative Adversarial Networks

Hjelm et al. (2018) introduced a method for training GANs on discrete data, named Boundary-Seeking GANs. They also aim for improving the stability in continuous data, as per the following.

Recall that Goodfellow et al. (2014) proved that for a given generator G, the optimal discriminator is $D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$.

This equation can be modified to find the true data distribution by knowing the optimal discriminator (3-16):

$$p_{data}(x) = p_g(x)\frac{D^*_G(x)}{1 - D^*_G(x)} \tag{3-16}$$

Generalizing this equation for any discriminator, as assuming by training it gets closer to the optimal one, it can be seen that the generator is optimal when $\frac{D(x)}{1-D(x)} = 1$, that is, $p_g(x) = p_{data}(x)$. This leads to $D(x) = 0.5$, which is the decision boundary, and the discriminator has the same probability of classifying samples as real or fake.

To achieve this, Kristaldi (2017) proposed the following objective for the generator (3-17):

$$\min_G \mathbb{E}_{z\sim p_z(z)}\left[\frac{1}{2}(\log(D(x) - \log(1 - D(x))))^2\right] \tag{3-17}$$

4
Methodology

As introduced in Chaper 1, there is a great need for the reduction in the use of reservoir simulation, as to expedite, and increase feasibility of, smart well control optimizations. This work opted to exploit the advantages of deep learning for this purpose, due to its historical validation in the context of supervised learning. Within deep learning, generative adversarial networks were chosen to generate well controls and its accompanying reservoir data, as to augment existing datasets and increase the robustness of simulator proxies trained on relatively few simulations.

In order to address the challenge of achieving good proxy results without the need for extensive simulations, this work proposes a methodology which harnesses the power of GANs to generate scenarios for augmenting proxy training datasets.

Figure 4.1 illustrates the methodology, from the definition of a reservoir model, its multiple simulation as to create a dataset, the training of a GAN on this data and its use for the sampling of new data, to the use of this final augmented dataset for the training of a reservoir simulator proxy. The following sections detail these steps.

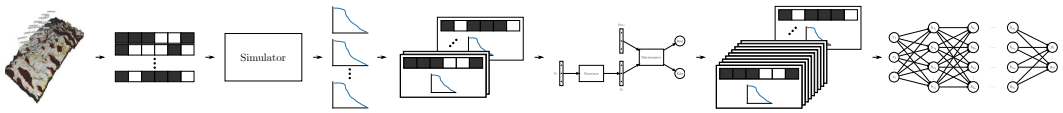

Figure 4.1: Illustration of the proposed methodology.

4.1
Model Selection

The first step is the selection and preparing of a reservoir model, as seen in Figure 4.2. As the focus hereto is on smart wells, the models need to have the valves defined, that is, the smart well controllable zones. This information may come from reservoirs already in production, with smart completions already present, or by specialists, aiming to justify the installation of the technology by proving its positive financial return via optimizations. In this case, this

methodology expedites the testing of several possibilities and combinations of valve settings.

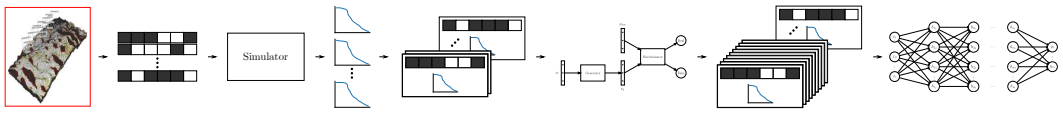

Figure 4.2: Illustration of the model selection step.

4.2 Dataset Building

After its definition, the reservoir model needs to be simulated for the generation of an initial dataset, as shown in Figure 4.3.

While the input parameters are always the smart well controls, this step includes determining which output parameters will be learned and predicted in the future. Some possibilities include production rates, such as oil, water and gas, pressures, such as bottomhole or wellhead, or ratios, such as the gas-oil or the water cut, among many others.

Following parameter choosing, the model should be simulated multiple times with diverse enough input scenarios. An important parameter to be set in this step is the number of simulations to be done. While the use of a great number of scenarios leads to more accurate network predictions, the number should not be as large as the amount of simulations needed for the original objectives, such as optimizations. Testing this parameter is encouraged, as to find the balance between simulation reduction and acceptable prediction errors.

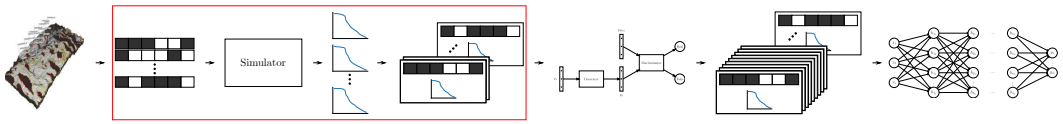

Figure 4.3: Illustration of the dataset building step.

4.3 Generative Adversarial Network Construction and Use

With a robust enough dataset given by the previous step, the GAN can be constructed and trained, as per Figure 4.4. This includes the definition of the architecture to be used in the networks, and cost functions. Hyperparameter and architecture tuning is also encouraged, as to achieve optimal results.

After training, the GAN can now be used for the generation of scenarios, including inputs and outputs. The generator is uncoupled from the network, and with noise provided as input, it can be used to sample multiple scenarios.

While one simulation takes a significant amount of time, the generator can create a whole dataset, with a considerable number of scenarios, in an almost negligible amount of time.

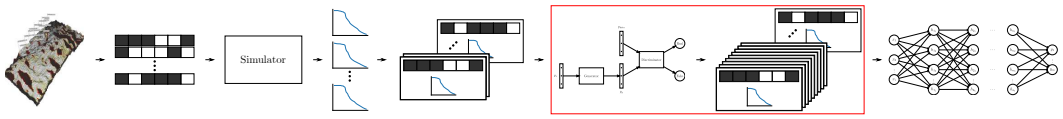

Figure 4.4: Illustration of the generative adversarial network construction and use step.

4.3.1 Generator Evaluation

This work also proposes a methodology for the evaluation of the scenarios generated by the network. While there are several options in the literature for generator evaluation, the data used in the present work has certain characteristics which favor a specific and new result-obtention routine, illustrated by Figure 4.5.

The generator here proposed is used for the sampling of the inputs, which are smart well controls, and the chosen outputs. As the control settings are between 0 and 1, i.e., a fully closed and fully open zone, any generated control in this interval is valid. While reproducing the distribution of the original data may be interesting, depending on the way the original dataset was built, it may not always be needed, as it is interesting to introduce variety in the dataset for future proxy use. Meanwhile, achieving good output results is paramount for accuracy in the proxy. Not only mantaining the distribution of the original data is important, the generated outputs should realistically represent simulator outputs, given the same inputs.

Therefore, in order to guarantee that the error is minimal, that is, that the generator is also correctly representing the simulator, the inputs sampled by the network can be simulated, and its results can then be compared to the generator outputs.

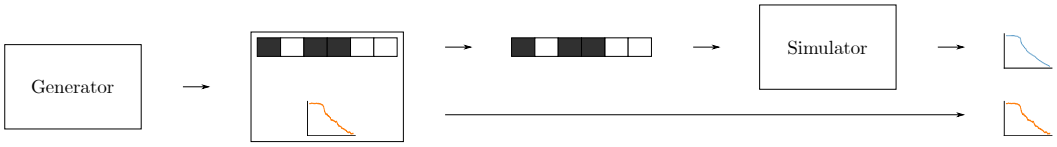

Figure 4.5: Illustration of the proposed result-obtention methodology.

4.4 Proxy Construction, Training and Coupling

After the GAN is trained and the generator is ready for use, the reservoir proxy model can be built, as in Figure 4.6.

This step involves, as in the previous one, the definition of the architecture, including which networks to be used, and several parameters of the model. As before, hyperparameter tuning is also strongly encouraged.

Subsequently, the generator can be coupled to the proxy, with this coupling being the core of the present work. Here, the generator is used for the sampling of several scenarios, creating a new dataset. This is then paired with the previous used one, as to create a final, robust dataset, which may be used for the training of the proxy. This results in a powerful reservoir proxy, which can thus be used to enable several simulation-intensive optimizations.

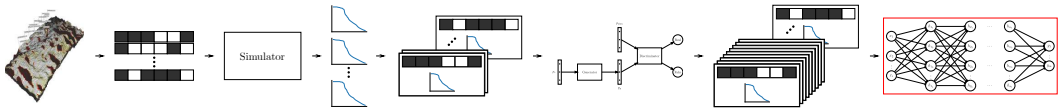

Figure 4.6: Illustration of the proxy construction, training and coupling step.

4.5 Optimization Connection

As mentioned in section 1.1.1, the present work was developed in the context of the optimizations given by the Flexwell project. As such, all the previous steps are inserted in the project workflow, and run in tandem to the optimization.

In the beginning of the optimization, reservoir simulation is needed for objective function calculation of all control possibilities. At a certain point, there are enough simulated scenarios as to begin the GAN training.

In sequence, the trained generator can be used for the sampling of datasets, and the proxy may be trained. It is important to note that these

steps happen simultaneously to the optimization, as to help total optimization time.

After training, the proxy may be used in lieu of the simulator, be it by fully replacing it in the optimization, or only partially. If done via the latter option, it and the GAN may be improved throughout the process, by using the simulated scenarios for further training. Other possibility opened by this option is to check the error of the proxy in each timestep, against the simulated scenarios. As such, information may be gained as to judge whether the proxy and GAN need further training.

As the models here proposed don't take geological information into account as inputs, each one is related to a single reservoir realization. As such, when using one of the optimization possibilities which considers uncertainty, a GAN and proxy need to be built for each geological scenario. This means that even when using scenario clustering, the networks may be used for the whole length of the optimization, as each one accompanies its respective scenario.

5
Case Studies

In order to successfully validate the methodology proposed in this work, three case studies were conducted. This section details the cases, displaying, discussing and analyzing the obtained results.

While the methodology was developed for the optimization context, these case studies were conducted in an isolated manner, with simulations done only for the studies, not given by the necessity of an optimizer.

It is important to note that all simulations done in this work were via CMG's IMEX black-oil reservoir simulator (Computer Modelling Group Ltd., 2017), and all networks were built with the Keras (Chollet et al., 2015) framework, with a TensorFlow (Abadi et al., 2015) backend.

To calculate the errors in the case studies, the root mean squared error (5-1) and the normalized root mean squared error (5-2) metrics were used.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{5-1}$$

$$NRMSE = \frac{RMSE}{y_{max} - y_{min}} \tag{5-2}$$

Where y are the true values, $\hat{y}$ are the values given by the networks and n is the amount of data in question.

5.1
Generative Adversarial Network Validation

The first aspect of the methodology to be tested was the application of GANs to the generation of smart well data, before the connection to other applications. This case study, including the description of the methodology for the generation of smart well data and the results here obtained, was published as Gurwicz et al. (2019).

For this case study, a single realization of the PUNQ-S3 reservoir model was chosen. It is an industry benchmark, first proposed by Floris et al. (2001).

Based on a real field reservoir, it represents a small-size model, with a grid of $19 \times 28 \times 5$ blocks, of which 1761 are active. The field is bound to the east and south by a fault and to the north and west by a strong aquifer, and

contains 11 producer wells. Its production horizon is of 17 years. Figure 5.1 shows the porosity map of the model, including the location of the wells.

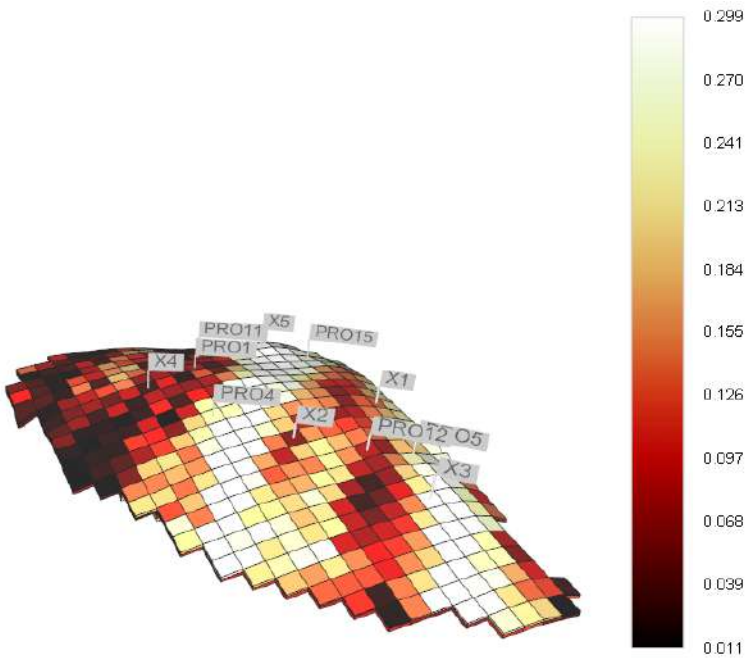

Figure 5.1: Porosity map of the PUNQ-S3 reservoir model.

While the geological model and parameters were kept the same as the original paper (Floris et al., 2001), the suggested historical production data, including the well testing and shut-in periods, was not used for the purposes of this work as it does not necessarily represent the reservoir behavior under normal conditions.

This model was chosen as while it is sufficiently complex and represents a real case, it still has a relatively small simulation runtime, allowing for the testing of several model and network possibilities. As the model is also freely available, the methodology here developed may be compared to other alternatives.

For this initial investigation into the methodology, wells PRO1 and PRO12 were chosen to contemplate smart completions, with flow control valves in all completed zones, i.e., three zones each. This brings the total of controllable valves to six, which were defined to allow for control changes once per month.

While the input parameters for the reservoir simulator are the smart well controls, the output parameters were chosen as the oil and water production rates, in m^3/day, allowing for the perception of changes in reservoir behavior given input changes. The model was then simulated 1000 times with varying

valve settings, as to build a dataset. Each simulated scenario began with all valves open, and then after a random timestep all controls were randomly modified, once or twice, chosen also randomly, per scenario.

The dataset was then normalized to the $[0, 1]$ feature range to ease network learning and reshaped to be of the form [`number of samples, timesteps per sample, features per timestep`], before being fed to the network. In this case, the shape was $[1000, 203, 8]$, for 1000 simulations, 203 timesteps per simulation (one per month for 17 years, minus the first month) and 8 features per timestep (six valve controls plus oil and water productions). In this case study, the output of the generator was kept continuous, and truncated to the second decimal place before evaluation.

As a first test, a conventional GAN model was constructed and trained. Before result evaluation, it was observed that the generated data was stochastic-like, with several spikes. As to better represent the original curves, the Deep Convolutional architecture was henceforth used, which introduced a smoothness to the data, closer in shape to the simulator outputs.

As such, a DCGAN was constructed with an architecture adapted from the one proposed by the original paper (Radford et al., 2016). Figures 5.2 and 5.3 detail the architecture of the networks in this model.

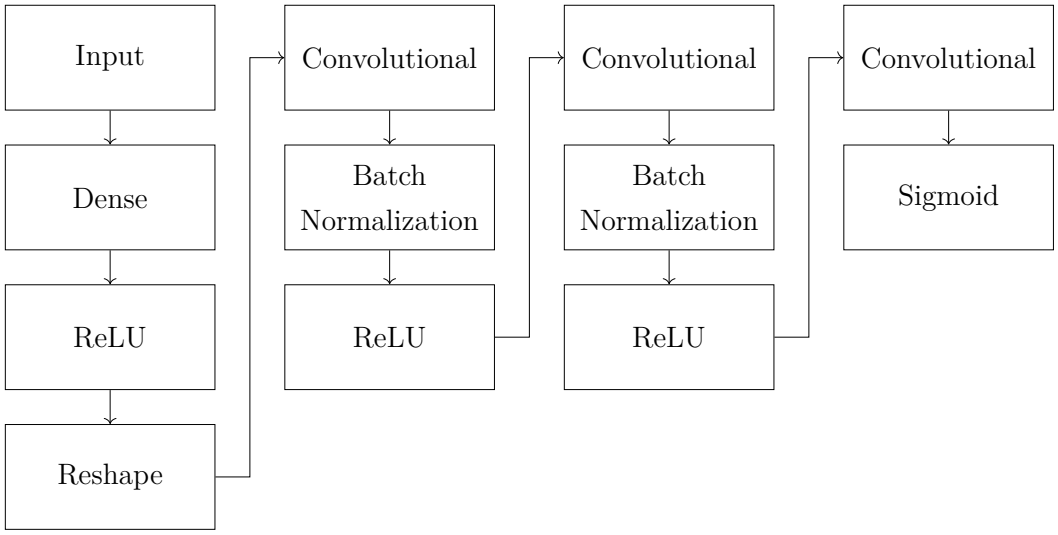

Figure 5.2: Generator architecture.

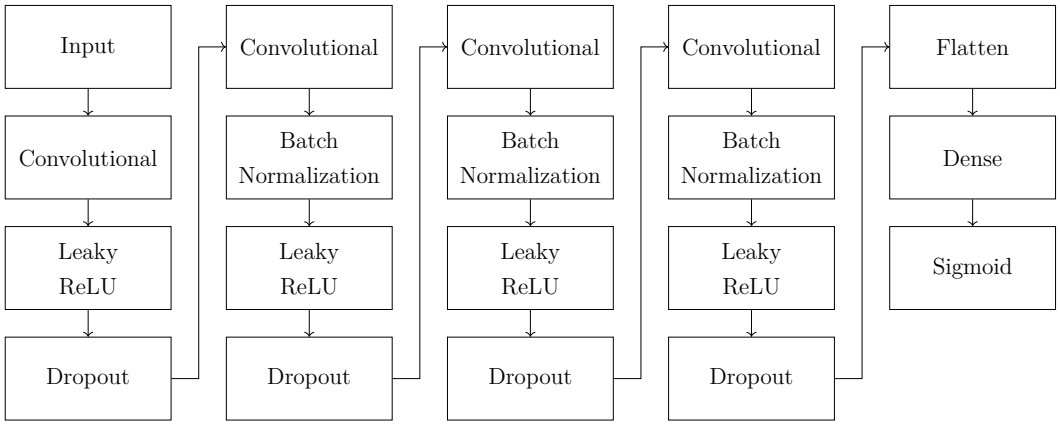

Figure 5.3: Discriminator architecture.

The idea of using convolutional layers, followed by batch normalization and a ReLU activation in the generator, and convolutional layers, followed by batch normalization, a leaky ReLU activation and dropout in the discriminator, was taken from the DCGAN paper.

However, some adaptations were made. The originally used 2D convolutional layers were swapped to 1D, as the data here used is two dimensional, being time series, and not three dimensional, as in images. In the generator, it was chosen to downsample the feature dimension while keeping the other dimension in a fixed size, due to the relatively small complexity of the data. As such, upsampling was not used, as present in common applications of GANs.

Also, as the majority of the data to be generated is in the $[0, 1]$ interval, the sigmoid function was chosen for the output of the generator. This is in contrast to the common use of the hyperbolic tangent, which outputs values in the $[-1, 1]$ interval.

The GAN was then trained adversarially, with the loss functions as proposed in the original GAN paper (Goodfellow et al., 2014) and the Adam optimizer (Kingma and Ba, 2014), and used to sample 128 scenarios, containing both smart well controls and associated oil and water productions. These went through the created results methodology, with Table 5.1 containing the obtained results.

Table 5.1: Results obtained for the DCGAN via the developed routine.

Epochs	RMSE Oil	RMSE Water	NRMSE Oil	NRMSE Water	Average NRMSE
10000	83.26	96.40	6.66%	4.78%	5.72%
20000	100.29	109.88	8.02%	5.45%	6.74%
30000	106.11	107.95	8.49%	5.35%	6.92%

Another test was done using Boundary-Seeking GANs, aiming for better results. While the architecture of the networks remained the same, the loss function proposed by Hjelm et al. (2018) and Kristaldi (2017) was used in the generator. Table 5.2 contains the results of this application.

Table 5.2: Results obtained for the B-DCGAN via the developed routine.

Epochs	RMSE Oil	RMSE Water	NRMSE Oil	NRMSE Water	Average NRMSE
10000	93.20	140.37	7.45%	6.96%	7.21%
20000	59.15	91.90	4.73%	4.56%	4.65%
30000	55.97	84.76	4.48%	4.20%	4.34%

As the one that lead to the best metrics, the B-DCGAN trained on 30000 epochs was chosen to further illustrate results. Figure 5.4 contains a boxplot of the NRMSE of each scenario, for both oil and water productions. The box extends from the lower to the upper quartile values of the data, and the whiskers extend to the 2nd and 98th percentiles. The circles represent outliers, and the middle line, the median. As this plot illustrates, not only is the total NRMSE low, the distribution of the NRMSEs has predominantly low values, with a few high outliers that are still acceptable values.

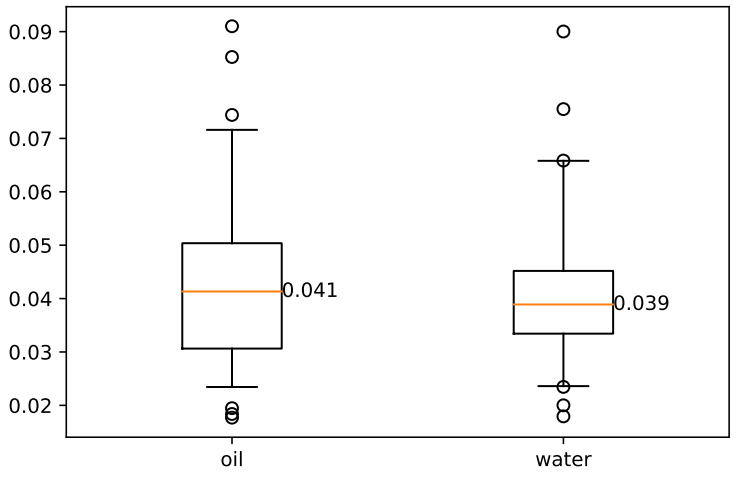

Figure 5.4: Boxplot of the NRMSE.

Figure 5.5 contains boxplots comparing the distribution between the simulated and generated data, given by the concatenation of all timesteps in all scenarios. It was built in a similar fashion to the previous one, with the difference that the whiskers extend to the maximum and minimum values of the data. It shows that the GAN was able to correctly learn the distribution

of the original data, maintaining approximate maximum and minimum values, as well as medians and quartiles.

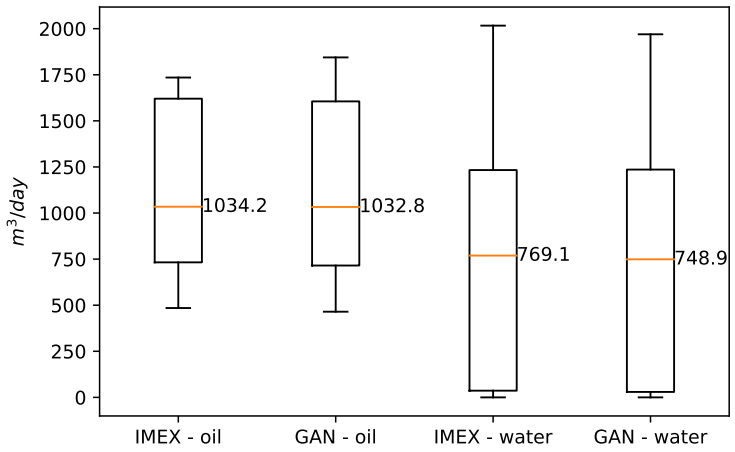

Figure 5.5: Boxplots of the simulated and generated oil and water production data.

Figures 5.6 and 5.7 show examples of generated curves, of a single scenario, compared to the ones simulated with the same valve controls.

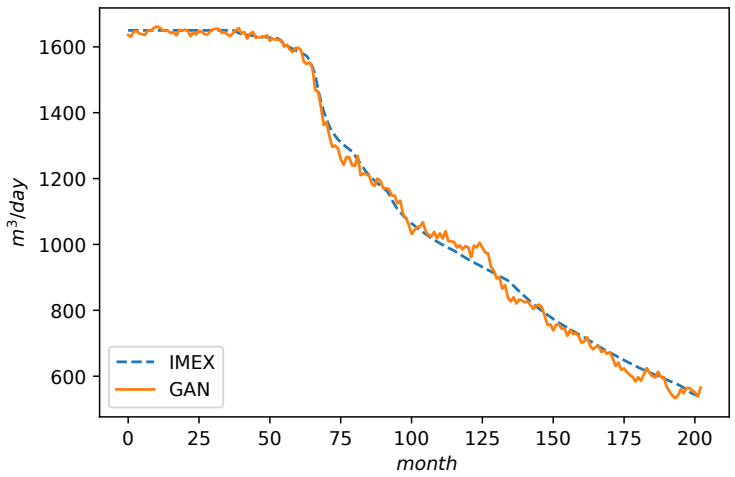

Figure 5.6: Example of a comparison between oil production curves.

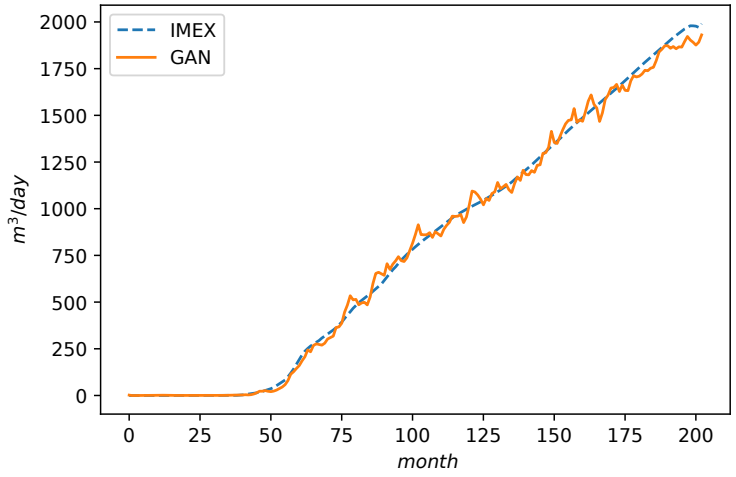

Figure 5.7: Example of a comparison between water production curves.

Figure 5.8 shows a comparison between the valve control data of the original dataset, and the one sampled by the generator. The whiskers extend to the maximum and minimum of the data, and the dotted line represents the mean. These results show that the model was able to replicate the distribution of original valve control data, while increasing diversity, as seen by the shift in the median. This means that the generated strategies have applied control earlier than the original dataset, and is useful as it increases data variety, for future dataset building.

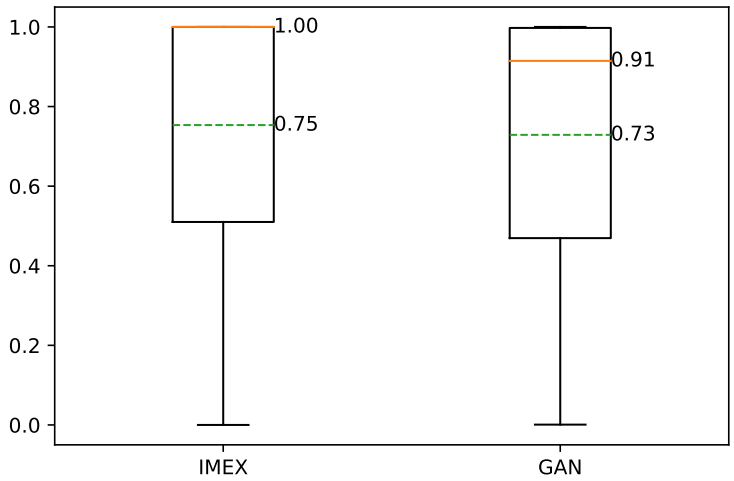

Figure 5.8: Boxplots of the original and generated control data

5.2
Coupling of the Generator to a Reservoir Proxy

While the previous case study validated the ability of a trained generator to correctly sample production curves and generate datasets, the connection to a reservoir proxy is paramount to guarantee that not only the data can be generated, it is useful in the proxy augmentation context.

As such, this case study aims to couple the generator to a simulator proxy, by training the proxy on data sampled by the generator. It also aims to bring the work closer to a real-world use case and to literature recomendations, in contrast to the previous one.

The PUNQ-S3 reservoir model (Floris et al., 2001) was again used, with the same modifications as the first study. This case tested the feasibility of including smart completions in all available zones of all wells, as the total number is modest. As such, the final model contains twenty two valves, one per zone where each well is defined in the original model.

The output parameters were again chosen to be the oil and water production rates, in m^3/day. In this case study, the scenarios no longer began with the valves all open, as it doesn't accurately represent the simulations needed in an optimization. Instead, each scenario started with random valve controls, which were modified once in a random timestep. As such, variety was added to the initial dataset, which no longer is mostly composed of open controls.

Different datasets were built to test the network's response to variations in dataset size, with 100, 1000 and 5000 scenarios. The data was normalized to the $[-1, 1]$ interval and reshaped to $[\texttt{number of scenarios}, 203, 24]$, before being fed to the network. A different generator architecture was here applied, as seen in Figure 5.9.

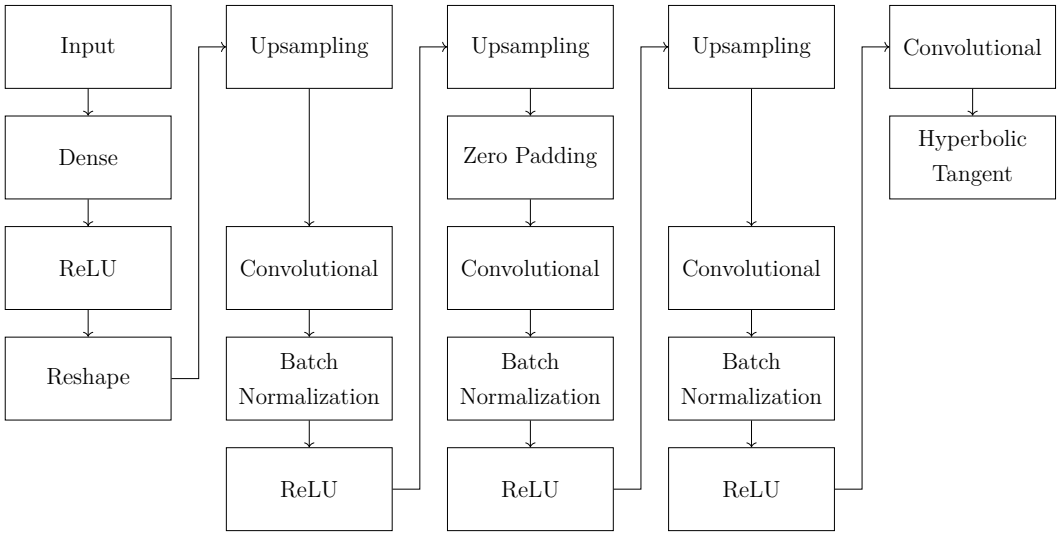

Figure 5.9: New generator architecture.

This more closely resembles the architectures commonly present in the literature, as both timestep and feature dimensions were now modified in shape along the network, with the feature one being downsampled and the timestep one being upsampled, in contrast to what was done in the previous case study. In order to succesfully apply these transformations and achieve the final correct data shape, the convolutional layers' kernel sizes were tuned, and zero padding was added to the data.

Also, the last activation function was changed from sigmoid to the hyperbolic tangent, matching literature recommendation (Radford et al., 2016). While it outputs data in the $[-1, 1]$ interval, as the initial data normalization was also done in this interval, the final data is denormalized to the correct interval.

While the generator still samples values in a continuous manner, the valve controls here generated were truncated to the first decimal place. Although this does not mean the generator is discrete, after this operation the data is only able to fall into ten possibilities.

The GANs were trained adversarially with the loss functions proposed in the original paper (Goodfellow et al., 2014) and the Adam optimizer (Kingma and Ba, 2014), using the different simulated datasets, and used to sample 100 scenarios. Table 5.3 contains the results of the created results methodology, and Figures 5.10 to 5.21 show the boxplots of the distributions of the simulated and generated production data, and the original and generated control data, for the 10000 and 50000 training epoch cases, representing the two extremes of training availability, where the full line represents the median, and the dotted

line, the mean.

Table 5.3: Results obtained for the DCGAN via the developed routine.

Scenarios in the Training Dataset	Epochs	RMSE Oil	RMSE Water	NRMSE Oil	NRMSE Water	Average NRMSE
100	10000	267.48	200.42	18.78%	14.70%	16.74%
	20000	249.63	175.42	18.23%	12.64%	15.44%
	30000	170.19	134.60	12.46%	9.84%	11.15%
	40000	196.15	126.37	15.15%	8.62%	11.89%
	50000	158.96	142.54	11.93%	10.22%	11.08%
1000	10000	195.59	179.70	13.69%	11.89%	12.79%
	20000	175.09	140.36	12.33%	9.75%	11.04%
	30000	199.64	178.45	15.34%	12.14%	13.74%
	40000	152.16	172.71	11.20%	12.52%	11.86%
	50000	171.97	151.70	12.47%	10.71%	11.59%
5000	10000	232.76	201.65	15.50%	13.68%	14.59%
	20000	177.31	164.51	12.74%	11.64%	12.19%
	30000	167.91	144.15	12.31%	10.11%	11.21%
	40000	155.68	140.64	11.54%	9.65%	10.60%
	50000	108.39	94.80	7.97%	6.55%	7.26%

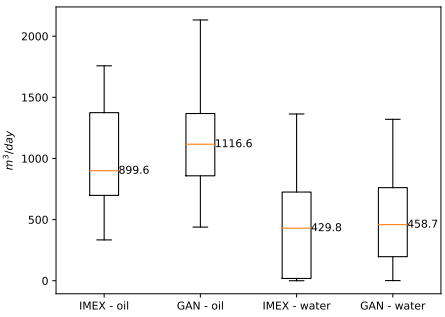

Figure 5.10: Production data boxplots for the DCGAN trained on 100 scenarios for 10000 epochs.

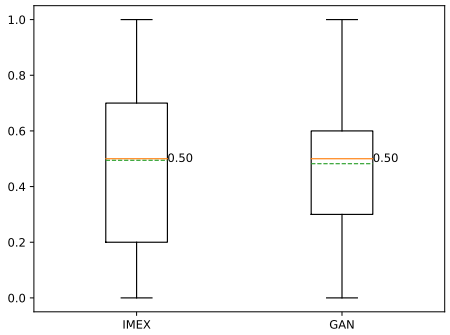

Figure 5.11: Control data boxplots for the DCGAN trained on 100 scenarios for 10000 epochs.

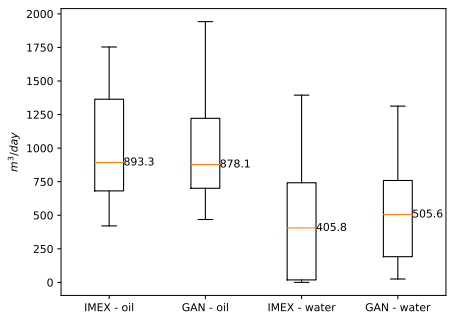

Figure 5.12: Production data boxplots for the DCGAN trained on 100 scenarios for 50000 epochs.

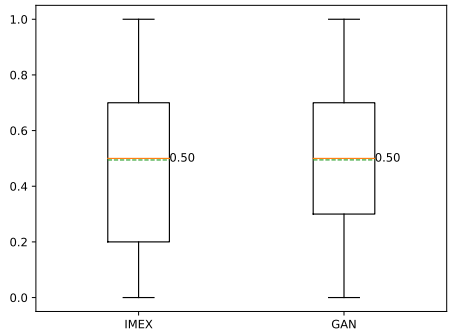

Figure 5.13: Control data boxplots for the DCGAN trained on 100 scenarios for 50000 epochs.

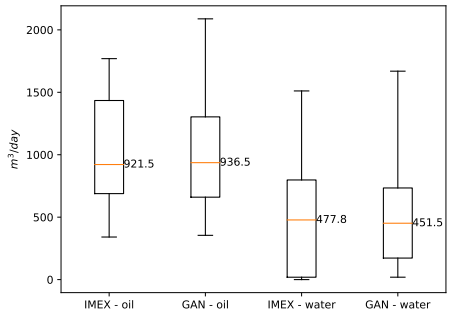

Figure 5.14: Production data boxplots for the DCGAN trained on 1000 scenarios for 10000 epochs.

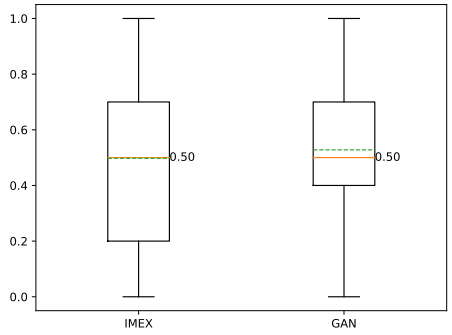

Figure 5.15: Control data boxplots for the DCGAN trained on 1000 scenarios for 10000 epochs.

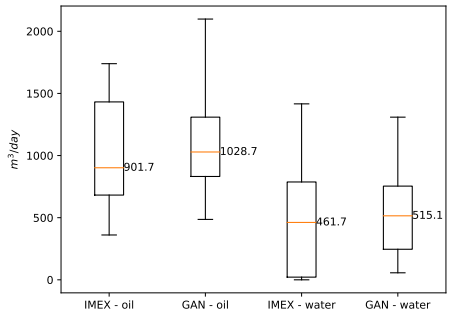

Figure 5.16: Production data boxplots for the DCGAN trained on 1000 scenarios for 50000 epochs.

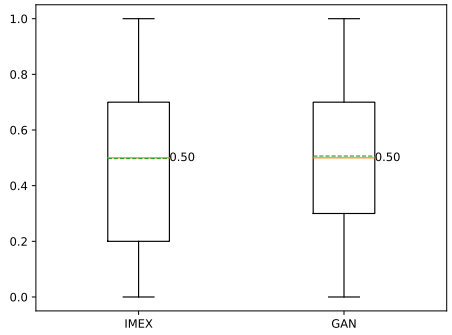

Figure 5.17: Control data boxplots for the DCGAN trained on 1000 scenarios for 50000 epochs.

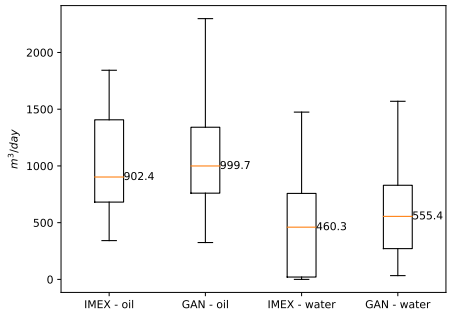

Figure 5.18: Production data boxplots for the DCGAN trained on 5000 scenarios for 10000 epochs.

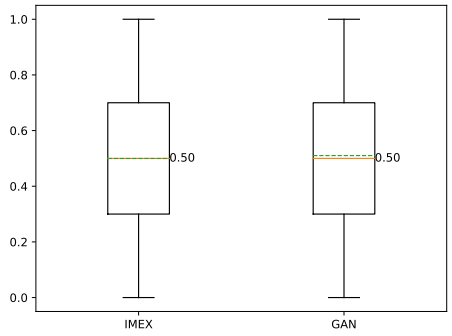

Figure 5.19: Control data boxplots for the DCGAN trained on 5000 scenarios for 10000 epochs.

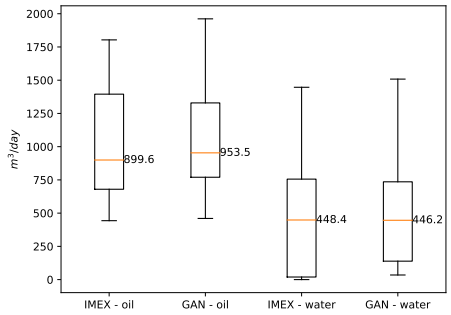

Figure 5.20: Production data boxplots for the DCGAN trained on 5000 scenarios for 50000 epochs.

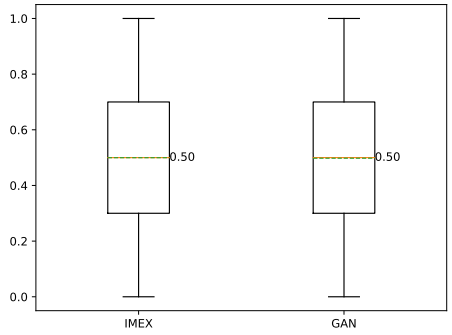

Figure 5.21: Control data boxplots for the DCGAN trained on 5000 scenarios for 50000 epochs.

As before, a Boundary-Seeking GAN was also tested, with the generator loss function proposed in Hjelm et al. (2018) and Kristaldi (2017). Table 5.4 contains the results of this application, and Figures 5.22 to 5.31 show the boxplots of the distributions of the simulated and generated production data, and the original and generated control data, for the 10000 and 50000 training epoch cases, where the full line represents the median, and the dotted line, the mean.

Table 5.4: Results obtained for the B-DCGAN via the developed routine.

Scenarios in the Training Dataset	Epochs	RMSE Oil	RMSE Water	NRMSE Oil	NRMSE Water	Average NRMSE
100	10000	228.22	143.94	16.07%	10.73%	13.40%
	20000	-	-	-	-	-
	30000	-	-	-	-	-
	40000	-	-	-	-	-
	50000	-	-	-	-	-
1000	10000	206.76	210.95	14.65%	14.21%	14.43%
	20000	191.84	163.79	13.34%	11.07%	12.21%
	30000	186.71	196.95	13.52%	13.98%	13.75%
	40000	160.68	145.22	12.36%	10.61%	11.49%
	50000	141.17	119.06	10.37%	8.90%	9.64%
5000	10000	208.65	215.11	14.31%	15.32%	14.82%
	20000	205.40	191.93	15.33%	13.73%	14.53%
	30000	210.65	218.25	15.78%	16.01%	15.90%
	40000	170.01	136.35	12.90%	9.43%	11.17%
	50000	124.13	117.64	8.96%	8.10%	8.53%

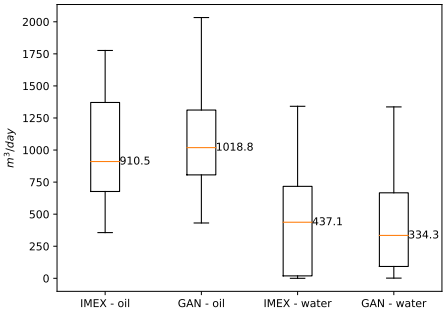

Figure 5.22: Production data boxplots for the B-DCGAN trained on 100 scenarios for 10000 epochs.

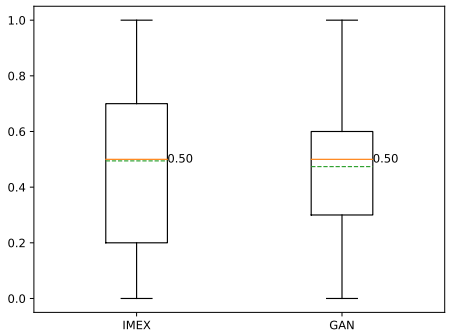

Figure 5.23: Control data boxplots for the B-DCGAN trained on 100 scenarios for 10000 epochs.

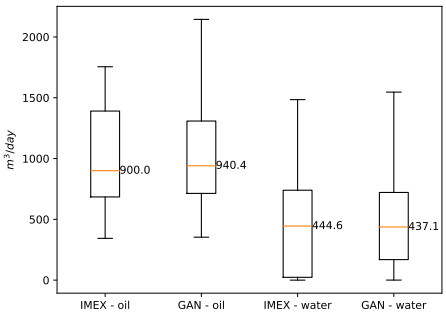

Figure 5.24: Production data boxplots for the B-DCGAN trained on 1000 scenarios for 10000 epochs.

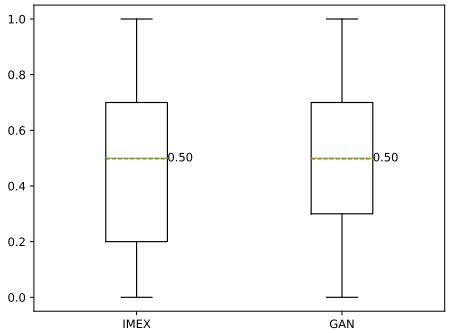

Figure 5.25: Control data boxplots for the B-DCGAN trained on 1000 scenarios for 10000 epochs.

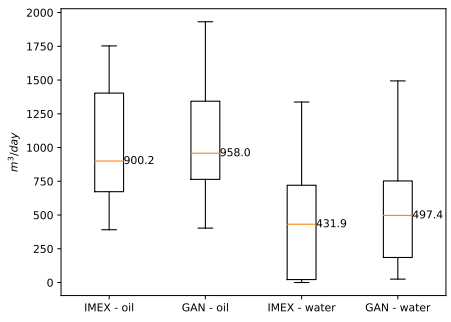

Figure 5.26: Production data boxplots for the B-DCGAN trained on 1000 scenarios for 50000 epochs.

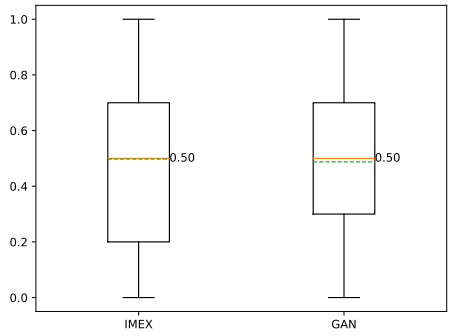

Figure 5.27: Control data boxplots for the B-DCGAN trained on 1000 scenarios for 50000 epochs.

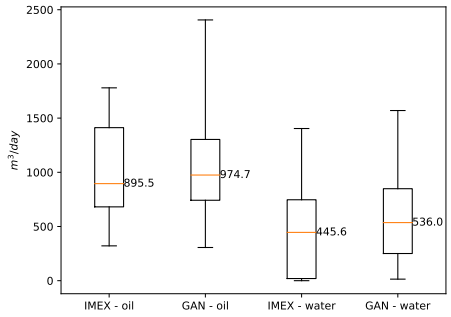

Figure 5.28: Production data boxplots for the B-DCGAN trained on 5000 scenarios for 10000 epochs.

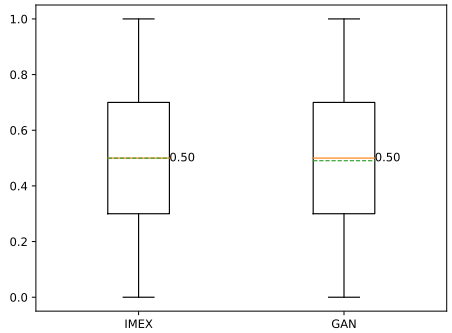

Figure 5.29: Control data boxplots for the B-DCGAN trained on 5000 scenarios for 10000 epochs.

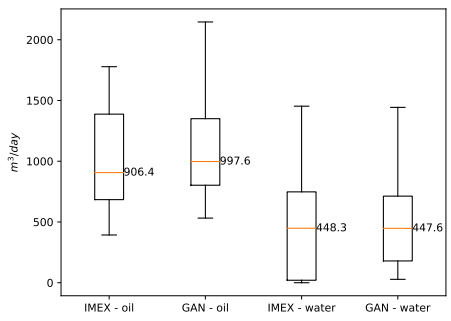

Figure 5.30: Production data boxplots for the B-DCGAN trained on 5000 scenarios for 50000 epochs.

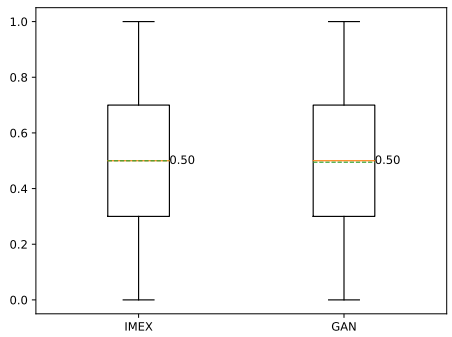

Figure 5.31: Control data boxplots for the B-DCGAN trained on 5000 scenarios for 50000 epochs.

The B-DCGAN trained on 100 scenarios experienced exploding gradients with 20000 or more epochs of training, therefore not generating valid data. A Wasserstein GAN, as proposed in Arjovsky et al. (2017), was then tested

in this study case, as by enforcing weight clipping it tends to avoid the exploding gradients problem. The general architecture of the network was mantained, except for the last layer of the discriminator's activation function being changed from a sigmoid to a linear function as needed for the architecture per the original paper.

The network was then trained adversarially as proposed in Arjovsky et al. (2017), where the disciminator is trained more than the generator and has its weights clipped. The RMSprop (Hinton et al., 2012) optimizer was used, as also proposed in the original paper of this network. Table 5.5 contains the results of this application, and Figures 5.32 to 5.43 show the boxplots of the distributions of the simulated and generated production data, and the original and generated control data, for the 10000 and 50000 training epoch cases, where the full line represents the median, and the dotted line, the mean.

Table 5.5: Results obtained for the W-DCGAN via the developed routine.

Scenarios in the Training Dataset	Epochs	RMSE Oil	RMSE Water	NRMSE Oil	NRMSE Water	Average NRMSE
100	10000	520.08	293.70	31.16%	18.65%	24.91%
	20000	518.95	282.42	34.11%	18.61%	26.36%
	30000	638.43	305.30	42.16%	21.44%	31.80%
	40000	511.63	302.54	31.72%	21.67%	26.70%
	50000	574.93	272.30	36.78%	19.66%	28.22%
1000	10000	211.42	256.94	14.90%	17.71%	16.31%
	20000	290.54	309.56	19.67%	20.64%	20.16%
	30000	226.26	263.74	16.48%	16.59%	16.54%
	40000	256.22	284.92	18.10%	18.91%	18.51%
	50000	228.65	285.06	15.94%	19.95%	17.95%
5000	10000	223.51	268.01	15.91%	18.79%	17.35%
	20000	264.90	243.60	19.30%	16.54%	17.92%
	30000	201.18	217.60	13.86%	15.15%	14.51%
	40000	228.78	229.25	16.68%	14.85%	15.77%
	50000	224.64	215.30	15.87%	14.48%	15.18%

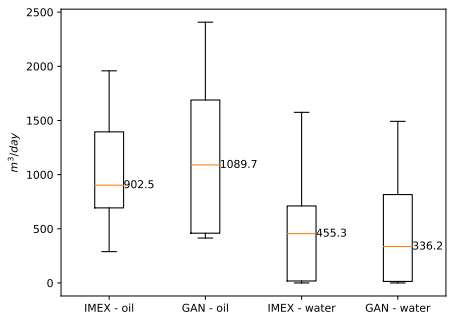

Figure 5.32: Production data boxplots for the W-DCGAN trained on 100 scenarios for 10000 epochs.

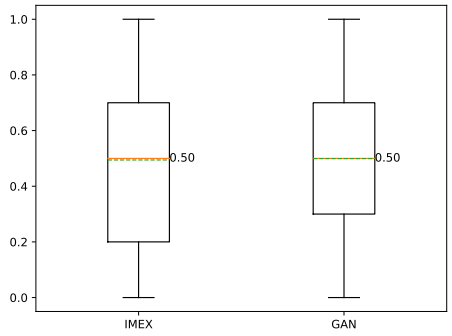

Figure 5.33: Control data boxplots for the W-DCGAN trained on 100 scenarios for 10000 epochs.

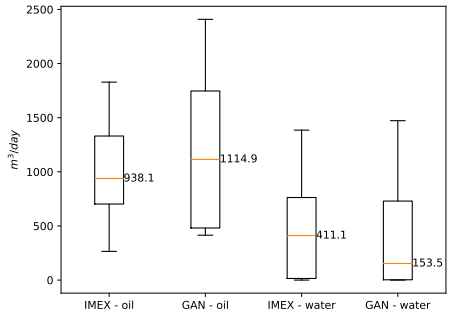

Figure 5.34: Production data boxplots for the W-DCGAN trained on 100 scenarios for 50000 epochs.

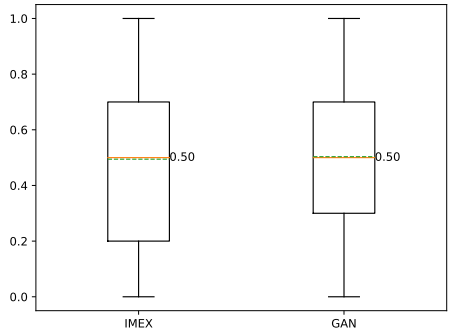

Figure 5.35: Control data boxplots for the W-DCGAN trained on 100 scenarios for 50000 epochs.

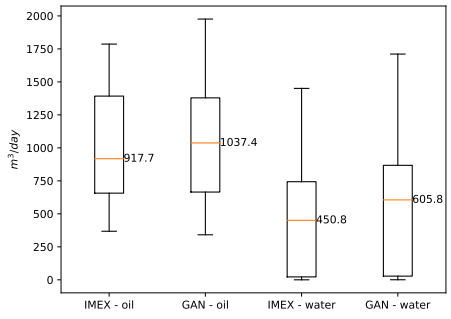

Figure 5.36: Production data boxplots for the W-DCGAN trained on 1000 scenarios for 10000 epochs.

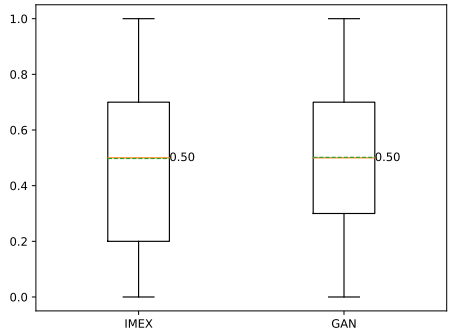

Figure 5.37: Control data boxplots for the W-DCGAN trained on 1000 scenarios for 10000 epochs.

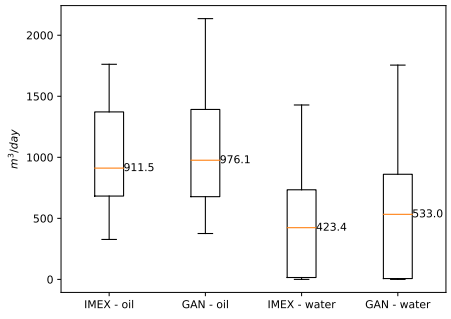

Figure 5.38: Production data boxplots for the W-DCGAN trained on 1000 scenarios for 50000 epochs.

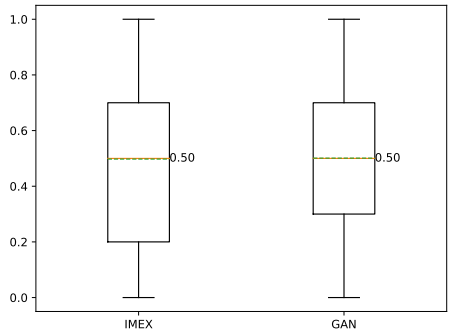

Figure 5.39: Control data boxplots for the W-DCGAN trained on 1000 scenarios for 50000 epochs.

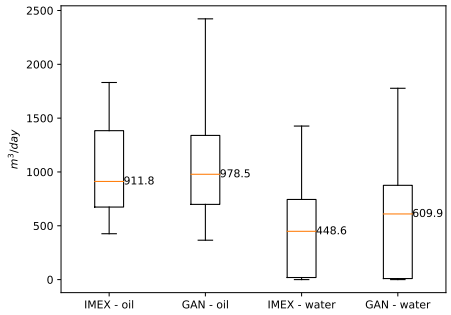

Figure 5.40: Production data boxplots for the W-DCGAN trained on 5000 scenarios for 10000 epochs.

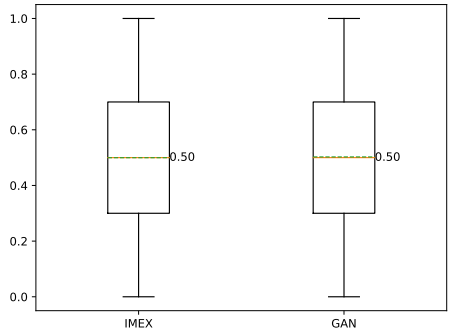

Figure 5.41: Control data boxplots for the W-DCGAN trained on 5000 scenarios for 10000 epochs.

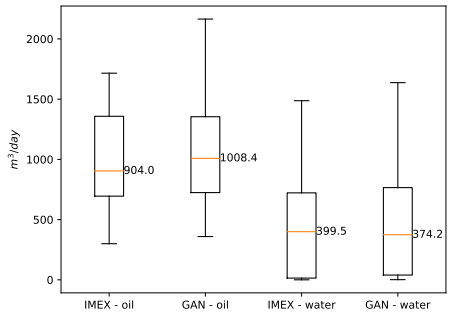

Figure 5.42: Production data boxplots for the W-DCGAN trained on 5000 scenarios for 50000 epochs.

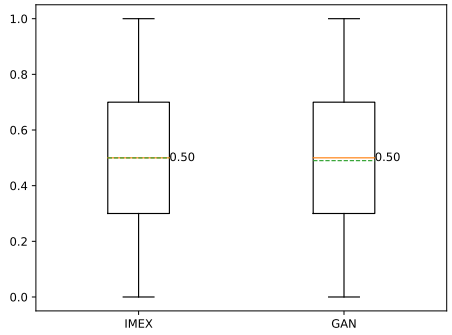

Figure 5.43: Control data boxplots for the W-DCGAN trained on 5000 scenarios for 50000 epochs.

Comparing the three networks' metrics, it can be seen that the DCGAN achieved better results in almost all cases. While better results were expected

from the other networks, this can be explained by the simplicity in this case study, in regards to number of parameters and size of the data.

It is expected that in cases with larger production horizons (where the timestep dimension of the data is higher) and with more controllable valves (where the feature dimension of the data is higher), the DCGAN architecture is surpassed by the others, given their need for more training data to obtain better results. In regards to the Wasserstein architecture, it takes longer to stabilize training in relation to the others. As the present case fixed the number of training epochs for comparison purposes, it was not enough for the stabilization of the W-DCGAN.

With the GANs trained, the next step was the coupling to the reservoir proxy, with the DCGAN chosen for this purpose based on the obtained results. It was decided to test both the influence of the number of scenarios and the number of GAN training epochs in the final results of the proxy. As such, six GANs were chosen: the one trained on 10000 epochs, and the one trained on 50000 epochs, for each number of scenarios.

For each case, the generator was uncoupled from the network, and used for the sampling of new datasets. Two tests were done, with the generation of 5000 and 10000 scenarios, in order to test the influence of the number of sampled scenarios in the proxy.

Each dataset to be fed into the proxy consisted on the samplings of the generator and the original dataset used for training the GAN. The data was normalized to the $[0, 1]$ interval and reshaped to [`number of samples, timesteps per sample, features per timestep`], in this case [`number of sampled scenarios+number of scenarios in the original dataset`, 203, 24].

The data was then randomly shuffled along the first dimension and divided into 80% for training and 20% for validation, before being split into inputs, the valve controls, and outputs, the productions. 100 scenarios were also randomly created, simulated and pre-processed as in the training dataset, in order to act as a test dataset for all cases.

For this case study, a simple LSTM-based proxy was used, with an architecture as shown in Figure 5.44.

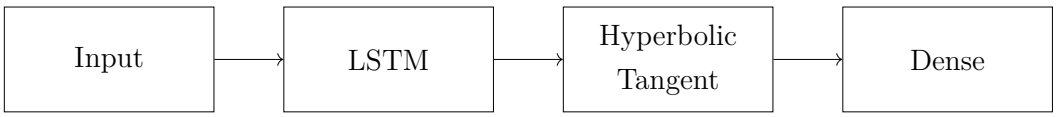

Figure 5.44: Reservoir proxy architecture.

A set of fixed hyperparameters was chosen for all tests of the proxy, as to fairly compare the influence of solely the input dataset. The proxies were trained for 20 epochs with the Adam optimizer (Kingma and Ba, 2014).

Table 5.6 contains the metrics obtained when training the proxy on only the initial datasets, without the use of GANs, and Table 5.7 contains the results of the proxies trained on datasets augmented by the generators. Table 5.8 summarizes the best results of the coupling in each scenario amount case.

Table 5.6: Results obtained for the proxy without the use of GANs.

Scenarios in the Training Dataset	RMSE Oil	RMSE Water	NRMSE Oil	NRMSE Water	Average NRMSE
100	383.85	361.78	19.63%	23.43%	21.53%
1000	111.92	113.97	5.72%	7.38%	6.55%
5000	63.59	53.77	3.25%	3.48%	3.37%

Table 5.7: Results obtained for the proxy with the use of generators.

Generated Scenarios	Scenarios in the Training Dataset	GAN Training Epochs	RMSE Oil	RMSE Water	NRMSE Oil	NRMSE Water	Average NRMSE
5000	100	10000	227.66	173.16	11.64%	11.22%	11.43%
		50000	181.04	171.69	9.26%	11.12%	10.19%
	1000	10000	105.44	110.68	5.39%	7.17%	6.28%
		50000	105.87	104.65	5.41%	6.78%	6.10%
	5000	10000	64.27	70.73	3.29%	4.58%	3.94%
		50000	55.47	49.56	2.84%	3.21%	3.03%
10000	100	10000	267.73	202.02	13.69%	13.09%	13.39%
		50000	143.21	157.34	7.32%	10.19%	8.76%
	1000	10000	107.99	91.25	5.52%	5.91%	5.72%
		50000	89.95	103.67	4.60%	6.72%	5.66%
	5000	10000	91.40	83.44	4.67%	5.40%	5.04%
		50000	54.60	45.98	2.79%	2.98%	2.89%

Table 5.8: Best results of the coupling.

Scenarios in the Training Dataset	GAN Training Epochs	Average NRMSE
100	-	21.53%
	10000	11.43%
	50000	8.76%
1000	-	6.55%
	10000	5.72%
	50000	5.66%
5000	-	3.37%
	10000	3.94%
	50000	2.89%

These results should be analyzed by tier, as seen in Table 5.8. In the first one, where a small number of scenarios was available, the coupling of the generator greatly improved results, going from an average NRMSE of 21.53% to 11.43%, if trained for 10000 epochs, and 8.76%, if trained for 50000 epochs.

In the second tier, with 1000 simulations, the generator mildly improved the proxy, going from an average NRMSE of 6.55% to 5.72%, if trained for 10000 epochs, and 5.66%, if trained for 50000 epochs.

With 5000 simulations, the results worsened if trained for 10000 epochs, with the average NRMSE going from 3.37% to 3.94%. If trained for 50000 epochs, the results improved to 2.89%.

A great comparison lies in the switch between tiers, if one had to choose between the use of the generator or the simulation of more scenarios. With 100 scenarios and a 21.53% average NRMSE, instead of simulating 900 more, to 1000, and achieving a 6.55% metric, one can opt to train the GAN and achieve a 8.76% metric, with only the 100 simulated scenarios.

In regards to the influence of generated scenarios, it can be seen that it is significantly affected by the accuracy of the generator. With the generators trained for 10000 epochs, the use of 5000 generated scenarios led to better results than the use of 10000 scenarios. When trained for more epochs, the use of 10000 sampled scenarios led to better results than 5000. This means that the error is clearly being propagated through the use of the generated scenarios, and the choice of number of scenarios to augment the proxy training dataset should rely on the accuracy of the available generator.

5.3 Validation on a Complex Reservoir Model

In this next case study, the main objective was to apply the routine as described in the previous one, to a more complex and realistic reservoir model. As to achieve this, the OLYMPUS model was selected, as proposed in Fonseca et al. (2018).

This model was developed with the objective of being a benchmark for field development optimization under uncertainty, used for a challenge on finding optimal well controls, optimal well and platform placement, and both in tandem. It is a synthetic model, based on North Sea fields, created to be sufficiently challenging for optimizations, while having realism of properties and uncertainties. It also has a relatively short simulation runtime, allowing for manageable optimization and testing of algorithms and techniques.

It consists on a grid of $118 \times 181 \times 16$ blocks, of which 192750 are active. Each block has approximately 50m $\times$ 50m $\times$ 3m. The reservoir contains four different facies, which model the uncertainties and were regenerated as to provide 50 realizations.

As the present methodology doesn't take uncertainties into account, a single realization was chosen to be used. Figure 5.45 shows the porosity map of the chosen realization, and Figure 5.46 shows the grid top map of the reservoir, highlighting the impermeable shale layer present, which divides it into two zones.

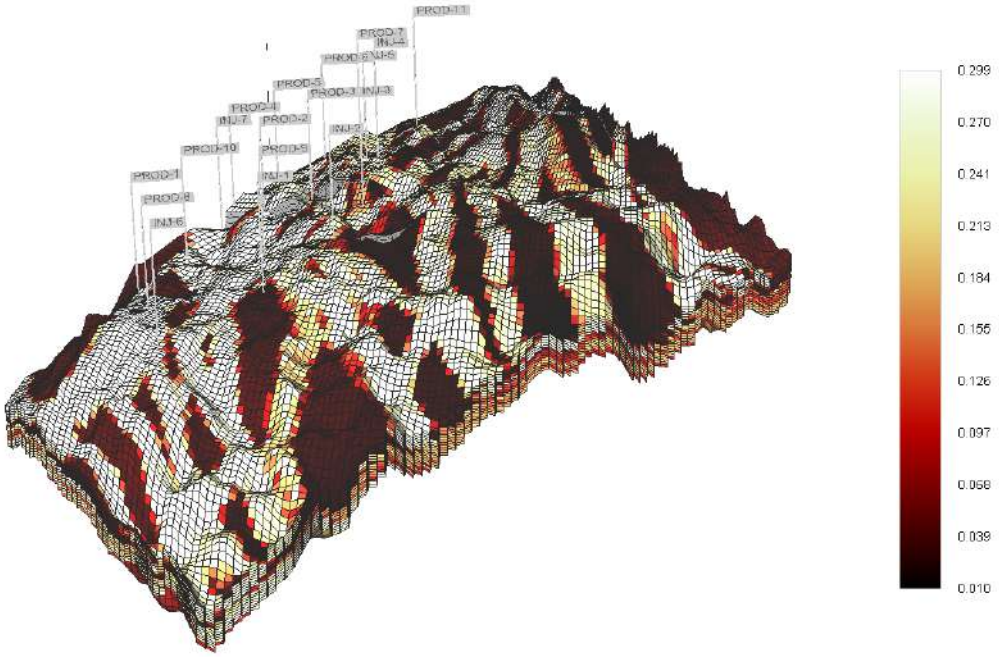

Figure 5.45: Porosity map of a realization of the OLYMPUS reservoir model.

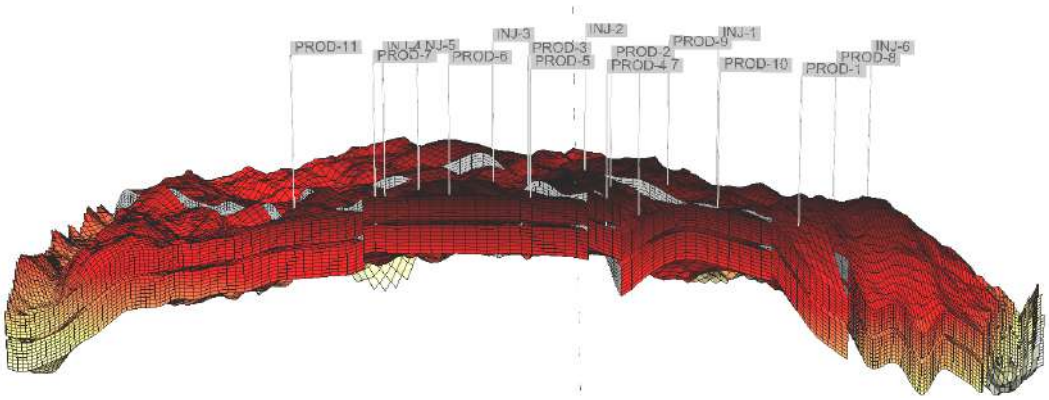

Figure 5.46: Grid top map of the OLYMPUS reservoir model.

This characteristic was used to place the smart completions, and the technology was implanted in all wells that had perforations on both zones. As such, all injector wells plus five producer wells contemplated the technology, with two zones each, the division given by the impermeable layer. This led to a total of twenty four controllable valves. As in the previous case, the valves were defined to be controllable once per month, through the twenty-year reservoir production horizon.

As before, the output parameters were chosen to be the reservoir oil and water production rates, in m^3/day. The inputs were defined by starting with random valve controls, modified once in a random timestep.

As this case more closely resembles a real application scenario, it was opted to do the analysis based on only a dataset of 100 simulations, which more accurately represents what is available on an optimization procedure.

In order to proceed to the network, the data was normalized to the $[-1, 1]$ interval, and reshaped to $[\texttt{number of scenarios}, 239, 26]$, where the second dimension corresponds to a control per month for 20 years, minus the first month, and the third dimension corresponds to the twenty four valves, plus the oil and water production rates.

The DCGAN was chosen for this case study, based on its performance on the previous one, with its architecture kept the same. Figures 5.47 and 5.48 illustrate the network architectures, with the parameters specific to this case study. The vectors underneath each box represent the size of the data when exiting the respective layer, with the "None" element indicating the flexibility in the network to receive data with different amounts of samples. The numbers on the right of certain boxes represent the amount of neurons, in the case of dense layers, and filters, in the case of convolutional layers.

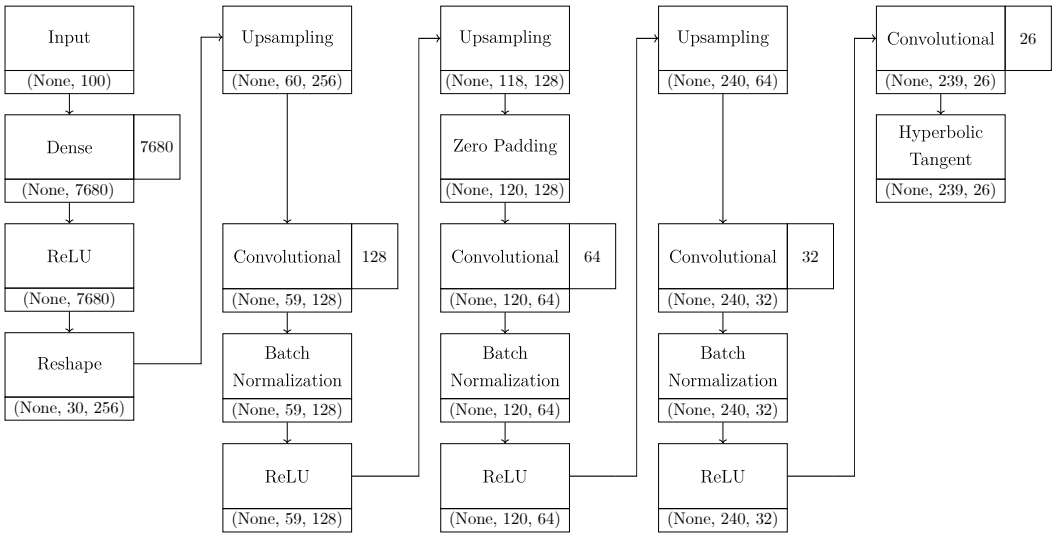

Figure 5.47: Generator architecture with parameters.

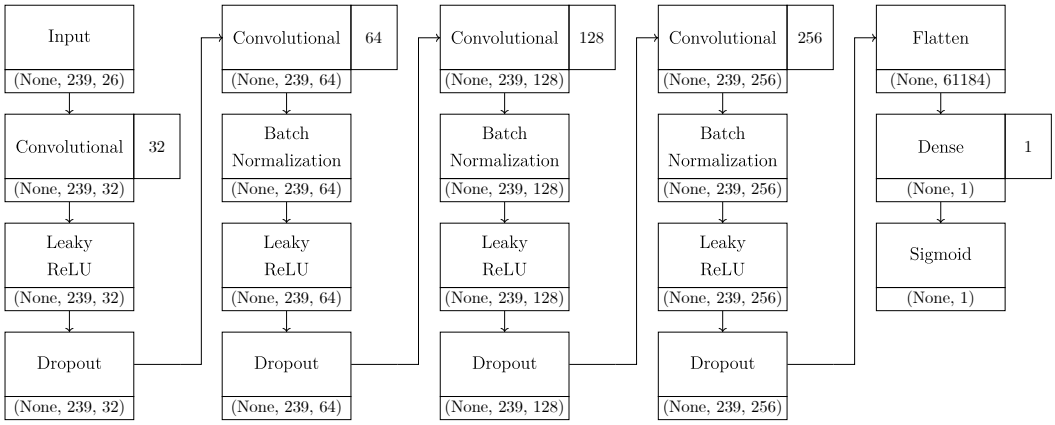

Figure 5.48: Discriminator architecture with parameters.

All layer parameters were kept as the default of the Keras (Chollet et al., 2015) framework, with the exception of a 0.2 slope for the leaky ReLU layers, a 0.25 rate for the dropout layers, and a 0.8 momentum for the batch normalization layers. The Adam optimizer (Kingma and Ba, 2014) was used, with a learning rate of 10^{-5} and a β_1 of 0.5. The parameters were taken from the recommendations in Radford et al. (2016), except for the learning rate, found from experiments.

Differently from the previous application, the results were taken for fewer epoch discretizations, as to test the influence of more extreme changes in the number of epochs. Table 5.9 contains the results of the methodology applied to this case, for the three datasets, after using the generator for the sampling of 100 scenarios. Figures 5.49 to 5.54 show the boxplots of the distributions of the simulated and generated production data, and the original and generated control data, for the 10000, 50000 and 100000 training epoch cases, where the full line represents the median, and the dotted line, the mean.

Table 5.9: Results obtained for the DCGAN via the developed routine.

Scenarios in the Training Dataset	Epochs	RMSE Oil	RMSE Water	NRMSE Oil	NRMSE Water	Average NRMSE
100	10000	766.58	1006.26	16.81%	17.61%	17.21%
	50000	428.36	747.55	9.57%	13.49%	11.53%
	100000	672.30	576.82	14.90%	10.20%	12.55%

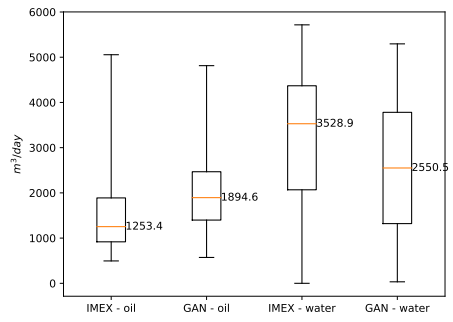

Figure 5.49: Production data boxplots for the DCGAN trained on 100 scenarios for 10000 epochs.

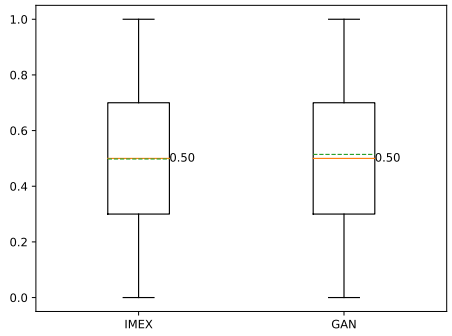

Figure 5.50: Control data boxplots for the DCGAN trained on 100 scenarios for 10000 epochs.

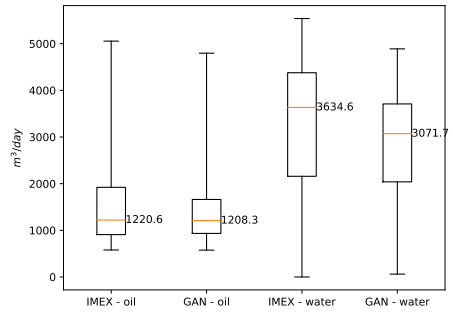

Figure 5.51: Production data boxplots for the DCGAN trained on 100 scenarios for 50000 epochs.

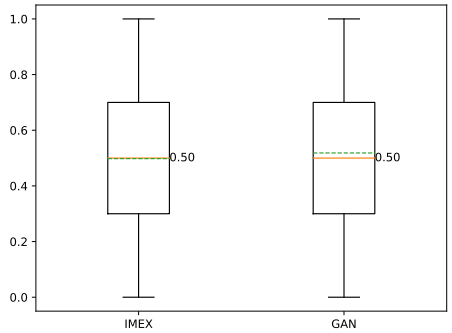

Figure 5.52: Control data boxplots for the DCGAN trained on 100 scenarios for 50000 epochs.

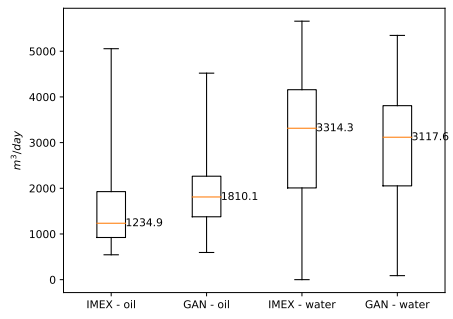

Figure 5.53: Production data boxplots for the DCGAN trained on 100 scenarios for 100000 epochs.

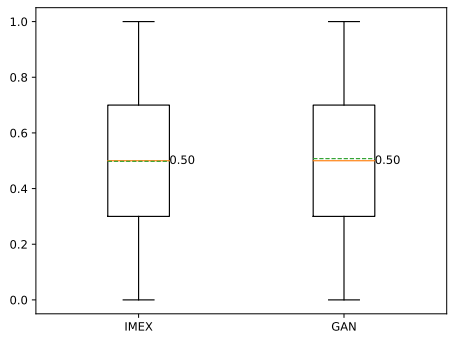

Figure 5.54: Control data boxplots for the DCGAN trained on 100 scenarios for 100000 epochs.

The generators were then uncoupled from the networks, and used for the sampling of new data. Tests were done with the generation of 1000, 5000 and 10000 scenarios.

The data went through the same preprocessing as before, resulting in the [`number of sampled scenarios+number of scenarios in the original dataset`, 239, 26] shape, divided into 80% for training, and 20% for validation. 100 additional scenarios were also simulated, to act as a test dataset.

The proxy was created, maintaining the architecture from the previous case study, as seen in Figure 5.55, which mantains the notations of Figures 5.47 and 5.48. The first "None" element in the vectors indicates the flexibility in the network to receive data with different amounts of samples, and the second indicates the flexibility to receive data with different numbers of timesteps.

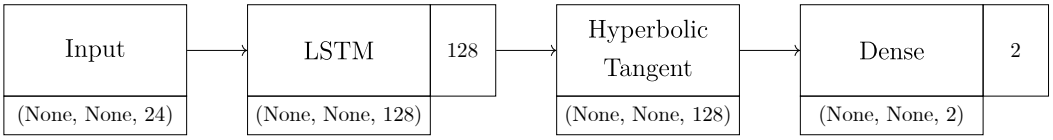

Figure 5.55: Reservoir proxy architecture with parameters.

The proxy was then trained for 20 epochs, with a batch size of 64 and the Adam optimizer (Kingma and Ba, 2014), with the default Keras (Chollet et al., 2015) parameters. Table 5.10 shows the results of the proxy trained on only the simulated dataset, and Table 5.11 shows the results of the proxy trained on the datasets including the generators. Table 5.12 summarizes the best results of the coupling.

Table 5.10: Results obtained for the proxy without the use of GANs.

Scenarios in the Training Dataset	RMSE Oil	RMSE Water	NRMSE Oil	NRMSE Water	Average NRMSE
100	709.19	1288.10	15.58%	22.27%	18.93%

Table 5.11: Results obtained for the proxy with the use of generators.

Scenarios in the Training Dataset	Generated Scenarios	GAN Training Epochs	RMSE Oil	RMSE Water	NRMSE Oil	NRMSE Water	Average NRMSE
100	1000	10000	743.68	1040.43	16.34%	17.98%	17.16%
		50000	453.59	602.62	9.96%	10.42%	10.19%
		100000	569.82	708.88	12.52%	12.25%	12.39%
	5000	10000	557.34	786.73	12.24%	13.60%	12.92%
		50000	386.49	631.87	8.49%	10.92%	9.71%
		100000	528.82	633.40	11.62%	10.95%	11.29%
	10000	10000	548.86	813.71	12.06%	14.07%	13.07%
		50000	397.59	620.15	8.73%	10.72%	9.73%
		100000	530.02	648.62	11.64%	11.21%	11.43%

Table 5.12: Best results of the coupling.

Scenarios in the Training Dataset	GAN Training Epochs	Average NRMSE
100	-	18.93%
	10000	12.92%
	50000	9.71%
	100000	11.29%

The inclusion of the generator improved results in all tested cases, with the most significant one being the GAN trained for 50000 epochs, with 5000 generated scenarios. This lead to a reduction of the error from 18.93% to 9.71%, as seen in Table 5.12.

Even in the case where a large number of epochs is not viable, the error was slightly reduced from 18.93% to 12.92%, with 10000 epochs and 5000 generated scenarios.

Among the chosen training epochs, the best results were always obtained with 50000. This leads to a balance of not too few epochs, as in the case of 10000, or not to many, as the case of 100000. Between the number of generated scenarios, the best results were consistently found with 5000, also highlighting an important balance in the mix of original and generated data.

5.3.1 Time Comparisons

While the performance of the GAN was validated, it is important to guarantee that the time taken to train this additional network is not overbearing in relation to the original simulations. As such, the time taken for each step of this case study is presented in Tables 5.13, 5.14 and 5.15.

The simulations were done in an Intel® Core™ i7-8700 CPU, with 64GB of RAM, and the networks' training in an NVIDIA® GeForce® GTX 1660 GPU, with 1408 CUDA® cores and 6GB of memory.

Table 5.13: Parallel simulation times for the OLYMPUS case study.

Scenarios	Time (minutes)
100	411.04
Average per scenario	4.11

Table 5.14: GAN training times for the OLYMPUS case study.

Epochs	Time (minutes)
10000	21.33
50000	105.97
100000	216.46
Average per epoch	0.13 seconds

Table 5.15: Proxy training times for the OLYMPUS case study.

Simulated Scenarios	Generated Scenarios	Time (seconds)
100	0	6.28
	1000	30.37
	5000	126.77
	10000	251.49

It is important to note that the time needed for the generation of scenarios and proxy inference is trivial in comparison to the simulations and training.

In the context of an optimization, the training of the networks may happen parallel to the simulator. As such, the optimization starts with only simulations. As it progresses and reaches enough simulated scenarios for an initial dataset, the GAN can start training, followed by the proxy. When finished, they may begin to substitute the simulator in the optimization.

100 scenarios might represent, for instance, the amount of simulations needed for one iteration in an optimization algorithm. As the time needed for these simulations is significantly higher than the time needed for the training of the 50000-epoch GAN added to the 5000-generated scenarios proxy, even before the third iteration of the algorithm the reservoir simulator may start to be substituted. In fact, considering the average simulation time per scenario, this network combination could be trained in the same time as $\frac{105.97 \text{ minutes} + 126.77 \text{ seconds}}{4.11 \text{ minutes}} \simeq 26$ simulations, a very small amount in the context of an optimization.

6
Conclusions

Reservoir simulation is one of the pillars in which the Oil & Gas industry stands, as a way to model and predict reservoir behavior is paramount to field developments. As several methodologies aiming for gains in project results heavily rely on it, and the complex equation-dependent simulation process is computationally expensive and time-consuming, ways to obtain faster simulations are always being researched.

Singling out an application of reservoir simulation, smart well control optimizations stand out as each objective function calculation needs a simulation. As these optimizations are essential in justifying the use of this technology, and making the most of it, a way to expedite these processes leads to better and more robust results, within a reasonable time frame.

The present work proposed a novel methodology for the use of generative adversarial networks, a subclass of deep generative models, to generate accurate and diverse smart well reservoir data. This data may then be used in the improvement of simulator proxies, networks able to substitute the simulator and ensure faster optimizations.

The methodology encompasses the selection of the reservoir model, the building of an initial dataset and the construction and use of the GAN. A novel result-evaluation technique is also proposed, by decoupling and simulating the generated input data, then comparing to the generated output data. The trained generator may then be applied to the sampling of new data, used for the training of reservoir proxies, which can in turn be used in control optimizations.

Three case studies were conducted in order to validate the work. The first one used the PUNQ-S3 reservoir model, with smart completions in six total zones, to test the generation of data. A Deep Convolutional GAN was tested, yielding a best average NRMSE of 5.72%, and a Boundary-Seeking Deep Convolutional GAN yielded 4.34%, both trained on a dataset of 1000 simulations.

The second one also used the PUNQ-S3 model, but with smart completions on a total of twenty two valves. This case tested the influence of the number of scenarios in the initial dataset, between 100, 1000 and 5000. A Deep Convolutional GAN obtained a best error of 7.26%, a Boundary-Seeking

Deep Convolutional GAN obtained 8.53%, and a Wasserstein Deep Convolutional GAN obtained 14.51%. The DCGAN was then used for the sampling of new data, and an LSTM-based reservoir proxy was built and trained on these datasets. The coupling of the generator was able to reduce the proxy error from 21.53%, in the case with 100 simulations, to 8.76%. In the case with 1000 simulations, from 6.55% to 5.66%, and in the case with 5000 simulations, from 3.37% to 2.89%.

The final case study was done on the OLYMPUS reservoir model, a complex model built as benchmark for field optimization techniques, where twenty four valves were chosen to be controllable. A DCGAN was trained on 100 simulations, obtaining a best average NRMSE of 11.53%. It was then coupled to an LSTM-based proxy, reducing the NRMSE from 18.93%, to 9.71%. Time comparisons were also done for this case study, finding that the training of the networks takes about the same time as the simulation of 26 scenarios of this reservoir, a very low number in the context of an optimization.

These cases successfully validated the hypothesis that the GANs are not only able to generate realistic data, and to follow the distributions of the original simulations, but to greatly reduce the errors of reservoir simulator proxies. As it was also established that a relatively small number of scenarios is enough for its training, which in turn is relatively fast, their coupling to proxies is able to considerably speed up optimization techniques.

6.1 Future Work

While all proposed objectives were achieved, in the course of this work several points for future investigation and study continuation were found. This section details them, suggesting the extension of this work as to find even better results.

– **Network Architectures**

  While this work tested a few different kinds of GANs, plenty more are available in the literature, the number growing with each passing day. As such, the testing of more types of GANs is encouraged, as to search for ones with better stability, and a good stopping criteria for training. The connection to other kinds of simulator proxies is also advocated for, as to test the reduction in the error of the coupling to other networks.

– **Parameter Tuning**

  Here, all network hyperparameters were kept constant throughout the case studies, as the focus was not on the best possible individual network

results, but on the dialog between networks. A way to significantly improve results is to optimize these parameters, that is, to test different possibilities and heuristics as to yield smaller errors.

– **Output Selection**

In the case studies, the variables chosen as outputs were the reservoir oil and water production rates. Instead of the rates, the cumulative production might yield better network results, as the curves are smoother and less stochastic-like, among other options.

– **Initial Scenario Generation**

In order to generate the initial datasets for training the GANs, the control scenarios were randomly generated. Methodologies for smarter initial control choosing, such as Latin hypercube sampling (McKay et al., 1979), can be tested as to possibly lead to improved results by better covering the possible control space.

– **Data Preprocessing**

The training and validation data split for the proxy was fixed in the present work, done after being randomly shuffled. A potential option to guarantee that the data was correctly divided, that is, that both sets accurately represent the data space, is to apply cross-validation techniques, such as k-fold cross-validation, possibly guaranteeing that the model sees all parts of the data as both training and validation.

– **Generative Models**

While generative adversarial networks were exclusively used in this work, other kinds of deep generative models should be investigated. Examples include variational auto-encoders (Kingma and Welling, 2013), generative stochastic networks (Alain et al., 2016) and variational generative stochastic networks (Bachman and Precup, 2015), among others.

– **Unification of the GAN and Proxy**

In this work, two different networks were used for the final optimization speedup objective. An encouraged line of work is to use the GAN both as data generator, and as proxy. One of the ways proposed is to use an encoder network, usually applied to feature dimension reduction, with the generator network, in order to map data to the generator input latent space. As such, the encoder can be used to receive the smart well controls, mapping them to the input of the generator, which after training could construct the output related specifically to this set of controls.

– **Optimization Connection**

While this work validated the generation of data and proxy augmentation, there is still the need to couple the methodology to an optimization. In doing so, the objective of expediting optimizations may be tested, with the expectation of better results by increasing optimization robustness. This also means that the simulations which compose the initial dataset are done according to the need of the optimization, and not via detached scenario generation techniques. As such, it might also be interesting to check whether the GAN is generating scenarios in the same direction as the optimization leads to, and not guiding the optimization away from its best possible results.

Bibliography

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CORRADO, C. G. S.; DAVIS, A.; DEAN, J.; GHEMAWAT, M.; GOODFELLOW, I.; IRVING, A.; ISARD, M.; JOZEFOWICZ, R.; JIA, Y.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANÉ, D.; SCHUSTER, M.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIÉGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y. ; ZHENG, X.. **TensorFlow: large-scale machine learning on heterogeneous systems**, 2015. https://www.tensorflow.org.

ABREU, A. C. A.. **An approach to value flexibility considering uncertainty and future information: an application to smart wells**. PhD thesis, Department of Electrical Engineering, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, 2016. https://doi.org/10.17771/PUCRio.acad.28331.

ABREU, A. C. A.; BOOTH, R.; PRANGE, M.; BAILEY, W. J.; BERTOLINI, A.; TEIXEIRA, G.; ROMEU, R. K.; EMERICK, A. A.; PACHECO, M. A. C. ; WILKINSON, D.. **A decision support approach to value flexibility considering uncertainty and future information**. Journal of Petroleum Science and Engineering, 2018. https://doi.org/10.1016/j.petrol.2018.03.077.

AÏFA, T.. **Neural network applications to reservoirs: Physics-based models and data models**. Journal of Petroleum Science and Engineering, 2014. https://doi.org/10.1016/j.petrol.2014.10.015.

AL-SHENQITI, M. S.; DASHASH, A. A.; AL-ARNAOUT, I. H.; AL-DRIWEESH, S. M. ; BAKHTEYAR, Z.. **Reduced water production and increased oil production using smart completions and MPFM "case study"**. In: SPE Saudi Arabia Section Technical Symposium, Dhahran, Saudi Arabia, 2007. Society of Petroleum Engineers. https://doi.org/10.2118/110978-MS.

ALAIN, G.; BENGIO, Y.; YAO, L.; YOSINSKI, J.; THIBODEAU-LAUFER, E.; ZHANG, S. ; VINCENT, P.. **GSNs: generative stochastic networks**. Information and Inference: A Journal of the IMA, 5, 2016. https://doi.org/10.1093/imaiai/iaw003.

ALMEIDA, L. F.; TUPAC, Y. J.; PACHECO, M. A. C.; VELLASCO, M. M. B. R. ; LAZO, J. G. L.. **Evolutionary optimization of smart-wells control under technical uncertainties**. In: Latin American & Caribbean Petroleum Engineering Conference, Buenos Aires, Argentina, 2007. Society of Petroleum Engineers. https://doi.org/10.2118/107872-MS.

ANDERSON, A. B.. **Integrating intelligent-well systems into sandface completions for reservoir control in brazilian subsea well**. In: SPE Annual Technical Conference and Exhibition, Dallas, Texas, U.S.A., 2005. Society of Petroleum Engineers. https://doi.org/10.2118/97215-MS.

ARJOVSKY, M.; CHINTALA, S. ; BOTTOU, L.. **Wasserstein generative adversarial networks**. In: Precup, D.; Teh, Y. W., editors, International Conference on Machine Learning (ICML), Sydney, Australia, 2017. PMLR. http://proceedings.mlr.press/v70/arjovsky17a.html.

AVANSI, G. D.; SCHIOZER, D. J.. **UNISIM-I: Synthetic model for reservoir development and management applications**. International Journal of Modeling and Simulation for the Petroleum Industry, 9, 2015.

BACHMAN, P.; PRECUP, D.. **Variational generative stochastic networks with collaborative shaping**. In: International Conference on Machine Learning (ICML), Lille, France, 2015. http://proceedings.mlr.press/v37/bachman15.pdf.

CALVETTE, T.; GURWICZ, A.; ABREU, A. C. ; PACHECO, M. C.. **Forecasting smart well production via deep learning and data driven optimization**. In: Offshore Technology Conference Brasil, Rio de Janeiro, Brazil, 2019. https://doi.org/10.4043/29861-MS.

CANCHUMUNI, S. W. A.; EMERICK, A. A. ; PACHECO, M. A. C.. **Towards a robust parameterization for conditioning facies models using deep variational autoencoders and ensemble smoother**. Computers & Geosciences, 2019. https://doi.org/10.1016/j.cageo.2019.04.006.

CAO, Q.; BANERJEE, R.; GUPTA, S.; LI, J.; ZHOU, W. ; JEYACHANDRA, B.. **Data driven production forecasting using machine learning**. In: SPE Argentina Exploration and Production of Unconventional Resources Symposium, Buenos Aires, Argentina, 2016. https://doi.org/10.2118/180984-MS.

CHAN, K. S.; MASOUDI, R.; KARKOOTI, H.; SHAEDIN, R. ; OTHMAN, M. B.. **Production integrated smart completion benchmark for field redevelopment**. In: International Petroleum Technology Conference, Doha, Qatar, 2014. https://doi.org/10.2523/IPTC-17220-MS.

CHOLLET, F.. **Keras**, 2015. https://keras.io.

COMPUTER MODELLING GROUP LTD. (CMG). **IMEX: three-phase, black-oil reservoir simulator, user's guide**, 2017.

DUCHI, J.; HAZAN, E. ; SINGER, Y.. **Adaptive subgradient methods for online learning and stochastic optimization**. Journal of Machine Learning Research, 12, 2011. http://jmlr.org/papers/v12/duchi11a.html.

DUPONT, E.; ZHANG, T.; TILKE, P.; LIANG, L. ; BAILEY, W.. **Generating realistic geology conditioned on physical measurements with generative adversarial networks**. arXiv e-prints, 2018. https://arxiv.org/abs/1802.03065.

DURLOFSKY, L. J.; AZIZ, K.. **Optimization of smart well control**. In: SPE International Thermal Operations and Heavy Oil Symposium and International Horizontal Well Technology Conference, Calgary, Alberta, Canada, 2002. Society of Petroleum Engineers. https://doi.org/10.2118/79031-MS.

EMERICK, A. A.; PORTELLA, R. C. M.. **Production optimization with intelligent wells**. In: Latin American & Caribbean Petroleum Engineering Conference, Buenos Aires, Argentina, 2007. Society of Petroleum Engineers. https://doi.org/10.2118/107261-MS.

ESTEBAN, C.; HYLAND, S. L. ; RÄTSCH, G.. **Real-valued (medical) time series generation with recurrent conditional GANs**. arXiv e-prints, 2017. https://arxiv.org/abs/1706.02633.

FLORIS, F. J. T.; BUSH, M. D.; CUYPERS, M.; ROGGERO, F. ; SYVERSVEEN, A.-R.. **Methods for quantifying the uncertainty of production forecasts: a comparative study**. Petroleum Geoscience, 7, 2001. https://doi.org/10.1144/petgeo.7.S.S87.

FONSECA, R.-M.; ROSSA, E.; EMERICK, A.; HANEA, R. G. ; JANSEN, J.-D.. **Overview of the olympus field development optimization challenge**. In: 16th European Conference on the Mathematics of Oil Recovery (ECMOR). EAGE, 2018. https://doi.org/10.3997/2214-4609.201802246.

FUKUSHIMA, K.. **Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unnafected by shift in position**. Biological Cybernetics, 36, 1980. https://doi.org/10.1007/BF00344251.

GAO, C. H.; RAJESWARAN, R. T. ; NAKAGAWA, E. Y.. **A literature review on smart well technology**. In: SPE Production and Operations Symposium, Oklahoma City, Oklahoma, U.S.A., 2007. Society of Petroleum Engineers. https://doi.org/10.2118/106011-MS.

GERS, F. A.; SCHMIDHUBER, J. ; CUMMINS, F.. **Learning to forget: continual prediction with LSTM**. In: International Conference on Artificial Neural Networks (ICANN), 1999. https://doi.org/10.1049/cp:19991218.

GLANDT, C. A.. **Reservoir management employing smart wells: A review**. SPE Drilling & Completion, 20, 2005. https://doi.org/10.2118/81107-PA.

GOODFELLOW, I.; POUGET-ABADIE, J.; MIRZA, M.; XU, B.; WARDE-FARLEY, D.; OZAIR, S.; COURVILLE, A. ; BENGIO, Y.. **Generative adversarial nets**. In: Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D. ; Weinberger, K. Q., editors, Advances in Neural Information Processing Systems (NIPS), Montreal, Canada, 2014. Curran Associates, Inc. https://papers.nips.cc/paper/5423-generative-adversarial-nets.

GOODFELLOW, I.; BENGIO, Y. ; COURVILLE, A.. **Deep Learning**. MIT Press, 2016. https://www.deeplearningbook.org.

GURWICZ, A.; CANCHUMUNI, S. A. ; PACHECO, M. A. C.. **Smart well data generation via boundary-seeking deep convolutional generative adversarial networks**. In: Lecture Notes in Computer Science (LNCS), International Conference on Artificial Intelligence and Soft Computing (ICAISC). Springer, 2019. https://doi.org/10.1007/978-3-030-20912-4_7.

HARTMANN, K. G.; SCHIRRMEISTER, R. T. ; BALL, T.. **EEG-GAN: Generative adversarial networks for electroencephalographic (EEG) brain signals**. arXiv e-prints, 2018. https://arxiv.org/abs/1806.01875.

HE, K.; ZHANG, X.; REN, S. ; SUN, J.. **Deep residual learning for image recognition**. arXiv e-prints, 2015. https://arxiv.org/abs/1512.03385.

HEBB, D. O.. **The Organization of Behavior: A Neuropsychological Theory**. John Wiley & Sons, 1949.

HINTON, G.; SRIVASTAVA, N. ; SWERSKY, K.. **rmsprop: Divide the gradient by a running average of its recent magnitude**, 2012. Lecture 6e of the Neural Networks for Machine Learning course. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

HJELM, R. D.; JACOB, A. P.; TRISCHLER, A.; CHE, G.; CHO, K. ; BENGIO, Y.. **Boundary seeking GANs**. In: International Conference on Learning Representations (ICLR), 2018. https://openreview.net/forum?id=rkTS8IZAb.

HOCHREITER, S.. **The vanishing gradient problem during learning recurrent neural nets and problem solutions**. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6, 1998. https://doi.org/10.1142/S0218488598000094.

HOCHREITER, S.; SCHMIDHUBER, J.. **Long short-term memory**. Neural Computation, 9, 1997. https://doi.org/10.1162/neco.1997.9.8.1735.

HORNIK, K.; STINCHCOMBE, M. ; WHITE, H.. **Multilayer feedforward networks are universal approximators**. Neural Networks, 2, 1989. https://doi.org/10.1016/0893-6080(89)90020-8.

ILAMAH, O.; WATERHOUSE, R.. **Field-scale production optimization with intelligent wells**. In: SPE EUROPEC at the EAGE Conference & Exhibition, Copenhagen, Denmark, 2018. Society of Petroleum Engineers. https://doi.org/10.2118/190827-MS.

JEU, S. J.; CUNNINGHAM, W.; SALIES, J. B.; JANNISE, R. C.; BERIDON, B. R.; OUBRE, B. P.; ARNOLD, G. S. ; PUCKETT, C. T.. **Case histories of unique multi-zone intelligent deepwater sand-control completions - how successes were achieved and the associated lessons learned**. In: Offshore Technology Conference, Houston, Texas, U.S.A., 2008. https://doi.org/10.4043/19622-MS.

KINGMA, D. P.; BA, J.. **Adam: A method for stochastic optimization**. arXiv e-prints, 2014. https://arxiv.org/abs/1412.6980.

KINGMA, D. P.; WELLING, M.. **Auto-encoding variational bayes**. arXiv e-prints, 2013. https://arxiv.org/abs/1312.6114.

KOHLER, M. R.. **Neural networks applied to proxies for reservoir and surface integrated simulation**. Master's thesis, Department of Electrical Engineering, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, 2013 (in Portuguese). https://doi.org/10.17771/PUCRio.acad.23258.

KONOPCZYNSKI, M.; AJAYI, A. ; RUSSEL, L.-A.. **Intelligent well completion: Status and opportunities for developing marginal reserves**. In: 27th Annual SPE International Technical Conference and Exhibition, Abuja, Nigeria, 2003. Society of Petroleum Engineers. https://doi.org/10.2118/85676-MS.

KRISTALDI, A.. **Boundary seeking gan**, 2017. https://wiseodd.github.io/techblog/2017/03/07/boundary-seeking-gan/.

KRIZHEVSKY, A.; SUTSKEVER, I. ; HINTON, G.. **ImageNet classification with deep convolutional neural networks**. In: Pereira, F.; Burges, C. J. C.; Bottou, L. ; Weinberger, K. Q., editors, Advances in Neural Information Processing Systems (NIPS), Lake Tahoe, Nevada, U.S.A., 2012. Curran Associates, Inc. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W. ; JACKEL, L. D.. **Backpropagation applied to handwritten zip code recognition**. Neural Computation, 1, 1989. https://doi.org/10.1162/neco.1989.1.4.541.

LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W. ; JACKEL, L. D.. **Handwritten digit recognition with a back-propagation network**. In: Touretzky, D. S., editor, Advances in Neural Information Processing Systems (NIPS), Denver, Colorado, U.S.A., 1990. Morgan Kaufmann. https://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.

LECUN, Y.; BOTTOU, L.; BENGIO, Y. ; HAFFNER, P.. **Gradient-based learning applied to document recognition**. Proceedings of the IEEE, 86, 1998. https://doi.org/10.1109/5.726791.

MATTAX, C. C.; DALTON, R. L.. **Reservoir simulation**. Journal of Petroleum Technology, 42, 1990. https://doi.org/10.2118/20399-PA.

MCCULLOCH, W. S.; PITTS, W.. **A logical calculus of the ideas immanent in nervous activity**. Bulletin of Mathematical Biophysics, 5, 1943. https://doi.org/10.1007/BF02478259.

MCKAY, M. D.; BECKMAN, R. J. ; CONOVER, W. J.. **A comparison of three methods for selecting values of input variables in the analysis of output from a computer code**. Technometrics, 1979. https://doi.org/10.2307/1268522.

MOGREN, O.. **C-RNN-GAN: Continuous recurrent neural networks with adversarial training**. In: Constructive Machine Learning Workshop (CML) at NIPS (Conference on Neural Information Processing Systems), Barcelona, Spain, 2016. https://www.cs.nott.ac.uk/~psztg/cml/2016/papers/CML2016_paper_11.pdf.

MOHAGHEGH, S. D.; ABDULLA, F.; ABDOU, M.; GASKARI, R. ; MAYSAMI, M.. **Smart proxy: An innovative reservoir management tool; case study of a giant mature oilfield in the UAE**. In: Abu Dhabi International Petroleum Exhibition and Conference, Abu Dhabi, U.A.E., 2015. https://doi.org/10.2118/177829-MS.

MOSSER, L. J.; DUBRULE, O. ; BLUNT, M. J.. **Conditioning of three-dimensional generative adversarial networks for pore and reservoir-scale models**. arXiv e-prints, 2018. https://arxiv.org/abs/1802.05622.

MOSSER, L. J.; DUBRULE, O. ; BLUNT, M. J.. **Stochastic seismic waveform inversion using generative adversarial networks as a geological prior**. arXiv e-prints, 2018. https://arxiv.org/abs/1806.03720.

NAVRATIL, J.; KING, A.; RIOS, J.; KOLLIAS, G.; TORRADO, R. ; CODAS, A.. **Accelerating physics-based simulations using neural network proxies: An application in oil reservoir modeling**. arXiv e-prints, 2019. https://arxiv.org/abs/1906.01510.

PARI, M. N.; KABIR, A. H.; MOTAHHARI, S. M. ; BEHROUZ, T.. **Smart well-benefits, types of sensors, challenges, economic consideration, and application in fractured reservoir**. In: SPE Saudi Arabia Section Technical Symposium, Al-Khobar, Saudi Arabia, 2009. Society of Petroleum Engineers. https://doi.org/10.2118/126093-MS.

RADFORD, A.; METZ, L. ; CHINTALA, S.. **Unsupervised representation learning with deep convolutional generative adversarial networks**. In: International Conference on Learning Representations (ICLR), 2016. https://arxiv.org/abs/1511.06434.

REZENDE, D. J.; MOHAMED, S. ; WIERSTRA, D.. **Stochastic backpropagation and approximate inference in deep generative models**. In: International Conference on Machine Learning (ICML), 2014. http://proceedings.mlr.press/v32/rezende14.html.

ROBINSON, M.. **Intelligent well completions**. Journal of Petroleum Technology, 55, 2003. https://doi.org/10.2118/80993-JPT.

ROSA, A. J.; CARVALHO, R. S. ; XAVIER, J. A. D.. **Petroleum Reservoir Engineering**. Petrobras and Interciência, 2006 (in Portuguese). ISBN 8571931356.

ROSENBLATT, F.. **The perceptron: A probabilistic model for information storage and organization in the brain**. Psychological Review, 65, 1958. https://doi.org/10.1037/h0042519.

SALIMANS, T.; GOODFELLOW, I.; ZAREMBA, W.; CHEUNG, V.; RADFORD, A. ; CHEN, X.. **Improved techniques for training GANs**. In: Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I. ; Garnett, R., editors, Advances in Neural Information Processing Systems (NIPS), Barcelona, Spain, 2016. Curran Associates, Inc. http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.

SCHIOZER, D. J.; SILVA, J. P. Q. G.. **Methodology to compare smart and conventional wells**. In: SPE Annual Technical Conference and Exhibition, New Orleans, Louisiana, U.S.A., 2009. Society of Petroleum Engineers. https://doi.org/10.2118/124949-MS.

SCHMIDHUBER, J.. **Deep learning in neural networks: An overview**. Neural Networks, 2015. https://doi.org/10.1016/j.neunet.2014.09.003.

SHALEV-SHWARTZ, S.; BEN-DAVID, S.. **Understanding Machine Learning: From Theory to Algorithms**. Cambridge University Press, 2014. http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning.

SIMONYAN, K.; ZISSERMAN, A.. **Very deep convolutional networks for large-scale image recognition**. arXiv e-prints, 2014. https://arxiv.org/abs/1409.1556.

SU, H.-J.; OLIVER, D. S.. **Smart well production optimization using an ensemble-based method**. SPE Reservoir Evaluation & Engineering, 13, 2010. https://doi.org/10.2118/126072-PA.

THOMAS, J. E.. **Fundamentals of Petroleum Engineering**. Petrobras and Interciência, 2001 (in Portuguese). ISBN 8571930996.

WULANDARI, T.; MAZZUCCHELLI, D.; RASOANAIVO, O.; ZAUGG, E. ; OLIVIER, G.. **Feasibility study of smart completion application in a complex mature field (Dunbar, North Sea)**. In: SPE/IATMI Asia Pacific Oil & Gas Conference and Exhibition, Nusa Dua, Bali, Indonesia, 2015. Society of Petroleum Engineers. https://doi.org/10.2118/176385-MS.

YAHI, A.; VANGURI, R.; ELHADAD, N. ; TATONETTI, N. P.. **Generative adversarial networks for electronic health records: A framework for exploring and evaluating methods for predicting drug-induced**

laboratory test trajectories. In: Machine Learning for Health Workshop (ML4H) at NIPS (Conference on Neural Information Processing Systems), Long Beach, California, U.S.A., 2017. https://arxiv.org/abs/1712.00164.

YANG, L. C.; CHOU, S. Y. ; YANG, Y. H.. **MidiNet: A convolutional generative adversarial network for symbolic-domain music generation**. arXiv e-prints, 2017. https://arxiv.org/abs/1703.10847.

ZEILER, M. D.. **ADADELTA: An adaptive learning rate method**. arXiv e-prints, 2012. https://arxiv.org/abs/1212.5701.