Inverse Problem Solving Approach Using Deep Network Trained by GAN Simulated Data

Sergei Krylov\textsuperscript{1} and Vladimir Krylov\textsuperscript{2}
\textsuperscript{1}Information Systems and Technologies, 403, National Research University Higher School of Econ, B. Pecherskaya, 25, Nizhny Novgorod, 603093, Russia
\textsuperscript{2}Applied Mathematics and Informatics, 403, National Research University Higher School of Econ, B. Pecherskaya, 25, Nizhny Novgorod, 603093, Russia. Tel.: 9648690004
E-mail: vkrylov@hse.ru (Vladimir Krylov)

Abstract. In this paper authors deal with tasks of reliably recover a hidden multi-dimensional model parameter from indirect process observations. Such task is known as inverse problem. There are a lot of inverse problems that have practical value, for example in seismic wave propagation, low-dose tomography. To solve many of these problems in a practical style, this article proposes an approach based on the many simulations of the corresponding forward problem and using the set of simulation data as the training dataset. Most of physical processes have computer models that generate precise results. The existing simulators provide ways to predict process output by input parameters. A difficulty in solving of most inverse problems is that the solution is sensitive to variations in data, which is referred to as ill-posedness. From broad spectrum of methods to overcome ill-posedness authors use machine learning model trained on special simulated data. The paper describes the deep network model using some regularization. The key idea is to use Generative Adversarial Network (GAN) to generate correct input parameters values and support the unique existence. This network is trained by parameter examples that are real solutions of inverse problem. The small manually built dataset transforms to infinite dataset automatically by GAN. The augmented dataset feeds the simulator to get output data to train deep learning network. The network has regularization layers to support stability. The paper describes details of this model using deep augmentation to solve inverse problems on the easy example: the task of throwing a heavy ball at an angle to the horizon, taking into account the force of friction against air.

1. Introduction

Currently deep learning methods are effectively used to solve a growing range of applied problems in which it is necessary to build real-time data processing systems. These days most of the applications of deep neural networks are aimed at searching in the input data for hidden patterns indicating the features of the data presented. These are the well-known tasks of classification, predictive parameter estimation (regression), clustering, etc. In all these cases, either explicit training with a teacher (supervised) is used, or implicit training takes place, when the result is unsupervised and it is interpreted by a person. In the latter case, when the interpretation seems to be unproductive, the teaching methodology changes, that is, implicit learning is performed with the participation of the teacher. Many tasks solved by neural networks are originated by the need to interpret the processes observed by sensors. In cases where the result of the interpretation should be some action that affects the process, the main approach is the reinforcement learning.
algorithms that allow you to change the reaction to the process, considering the data from some sensors as signs of encouragement or penalty. Numerous examples of such tasks are rule-based games with a partner. The training of deep neural networks to win is very effective when the network plays game with itself using random variations in behavior. Such learning loop can be considered as learning during the game with a partner simulator, the role of which is played by the second instance of the learning network.

2. Motivation and Background

2.1. Specifics of the task

In this work, the authors pay attention to specific tasks which have some processes that are observed using sensors. However, unlike the tasks of influencing the course of the process, we are interested in assessing the set of conditions that determine the process. For example, let there is data describing seismic vibrations of the soil at a certain set of points. The task is to determine the coordinates of the source of seismic activity and its parameters, for example, power. To train a deep neural network to determine the coordinates and power of a seismic source from the readings of the sensor grid, using the traditional approach, it is necessary to record many episodes of seismic activity. If it is necessary to take into account local terrain features, this is usually problematic approach. So in practice, instead of the trained model, a theoretical model of restoration of the seismic source from the recorded oscillations of the surrounding soil is used. Such a problem in physics is called the inverse problem of seismic wave propagation [1].

There are many inverse problems that have significant practical value: restoring the boundaries and characteristics of spatial objects from scattered electromagnetic or X-ray waves in microwave and fluoroscopy, determining the parameters of horizontal targets using a scattered electromagnetic field, determining the initial concentrations of substances in the course of a chemical reaction, and many others. Each of the well-known inverse problems was studied by mathematicians and analysis of their works indicates that, in general, these problems are more complex and less studied than direct problems. Obtaining solution equations relies on the derivation of equations to find the minimum of some complex optimization problem, which must be solved again for each version of the set of parameters. Therefore, the study and search for the possibilities of solving the inverse problem by a trained deep neural network is of undoubted interest for many applications where finding a solution requires high speed actions. These are the tasks of navigating moving objects in water, underground, radio-vision in heavily filled volumes.

2.2. Specifics of inverse problem

From an applied viewpoint inverse problems are concerned with determining causes from desired or observed effects. It is common to formalize this as solving an operator equation:

$$ y = (x_{true}) + \varepsilon; \quad x_{true} \in X $$

(1)

Here, X (model parameter space) and Y (data space) are vector spaces with appropriate topologies and whose elements represent possible model parameters and data, respectively. The $: X \to Y$ (forward operator) is a known continuous operator that maps a model parameters to data in absence of observation noise and $\varepsilon \in Y$ is a sample of a $Y$-valued random variable modeling the observation noise. Formally, the inverse problem for the above task can be written as

$$ x = I(y) $$

(2)

The finding of inverse operator $I: Y \to X$ requires taking into account of error $\varepsilon$ because for most inverse problems, the smoothness of the operator leads to Hadamard ill-posedness. The solution must be sought so that the conditions of uniqueness and stability are satisfied. There are many ways to solve ill-posed problems; usually, they all come down to constructing a solution
as an optimization problem while setting some additional restrictions on the resulting solutions that are not directly related to the original direct problem. These methods are known under the general name of regularization methods. The most famous and popular for solving inverse problems in physics is classical Tikhonov regularization. It was introduced by A.N. Tikhonov. For solving ill-posed inverse problems it can [2] be stated in the form

$$ R_\lambda (y) = \arg \min_{x \in X} \left\{ \frac{1}{2} \| A(x) - y \|^2 + \lambda S(x) \right\} $$

Other known methods are:

- Approximate analytic inversion
- Iterative methods with early stopping
- Discretization as regularization
- Variational methods

2.3. Using neural networks to solve inverse problem
A known theorem on the approximating ability of a trained neural network defines the possibility of approximating any continuous nonlinear function $I : \mathbb{R}^n \to \mathbb{R}^m$ by input-output mapping of some multi layer neural network [3]. This feature of neural networks has been investigated by several authors. There is established convergence analysis for the proposed NETT (Network Tikhonov) approach to inverse problems [4]. NETT considers data consistent solutions having small value of a regularizer defined by a trained neural network. To derive the convergence and convergence rates results the paper introduces a new framework based on the absolute Bregman distance. There is the study [5]. This paper proposed a framework for solving inverse problems for differential equations based on neural networks and automatic differentiation. Neural networks are used to approximate hidden fields. Authors analyze the source of errors in the framework and derive an error estimate for a model diffusion equation problem. Besides, there is proposed a way for sensitivity analysis, utilizing the automatic differentiation mechanism embedded in the framework. It frees people from the tedious and error-prone process of deriving the gradients.

2.4. State analysis and problem highlighting
An analysis of the above and other works in the field of the use of neural networks for solving inverse problems shows that in many cases of classical inverse problems and of applied inverse problems, the use of a neural network to obtain a solution is an effective approach. The architecture of the networks constructed by the researchers is able to search for regularized solutions that fully satisfy the requirements. However, the fundamental limitation of the use of neural networks for solving ill-posed tasks is the difficulty of training. Deep learning networks require training datasets of large sizes, while the data in them must satisfy the requirements of stability and uniqueness for the corresponding inverse problem. This condition is very rarely obtained in real experiments. It is possible to train a somewhat complex deep network that solves the inverse problem well in exceptional cases only. Moreover, it will require numerous tricks that are highly dependent on a specific task.

2.5. Simulation based experiments
Currently, almost all applied direct problems can be effectively modeled. There are many simulators of physical processes in which we can calculate the readings of sensors at any point in space under any imaginable real conditions for the creation of influences and the location of objects and the boundaries of various media filling space. For example, the task of obtaining sensor records for a given seismic period is very accurately solved for an environment of high complexity and the presence of many interacting objects. If we consider this task as a direct
one, then the inverse problem will be tasked of determining the location and parameters of seismic disturbance from the readings of sensors located at various remote points. Recently, similar problems in a simpler form have appeared in robotics called reverse kinematics [6]. It is easy to understand that a direct kinematic problem for any practically interesting case can always be modeled with any degree of accuracy. This also applies to many dynamics problems. However, sometimes completely unexplored inverse problems appear. There is an example with an electronic nose [7]. Air (gas mixture) is drawn into a confined space, in which several quartz microbalances are placed, which make it possible to register the sedimentation of individual molecules. Microbalance readings are recorded in real time. The task is to determine the molecules of which substances and in what concentration were present in the “sniffed” gas from these time series. The formalization of the solution to this problem is unknown, but the corresponding direct problem can be simulated for any concentration of various molecules. Building and training a neural network as the main processor of the electronic nose is a useful and important task.

3. Method
Using the direct task simulator to generate a training dataset for the inverse problem seems at first glance obvious. By setting various parameters that determine the results of the simulation, we can get any necessary number of training examples. Let’s use the above notation. Each experiment by simulator can be described as a solution to a direct problem with given parameters: \( (x_i) = y_i + \varepsilon \)

Going through examples, we can get an arbitrarily large set: \( (x_i, y_i + \varepsilon) \) It can be used as a training dataset. However, despite the obviousness of this approach, it encounters difficulties associated with ill-posed of inverse problems.

3.1. Parameters generation
For most inverse problems the topology of the set of solutions \( X \) is that, parameters \( x \) having a large distance between themselves can have very close correspondent \( y \) values in the sense of metrics of multidimensional space \( ||x_i - x_j|| \gg ||y_i - y_j|| \): \( \mathbb{R}^m \supset \mathbb{X} \) An attempt to evaluate the element by the values of a close element leads to an unacceptable error. Regularization methods impose special restrictions on the search for solutions, narrowing the admissible set \( X \) to a specially chosen manifold corresponding to the real admissible parameters of the problem. If we use the principles of statistical regularization, \( X \) must be described by a probability distribution for the points included in it. So the set of pairs of the training dataset should be built on such a set of parameters, the distribution of which coincides with the distribution. The authors consider the generation of a training dataset on a statistical ensemble of parameters estimated in the inverse problem with the distribution of the set of these parameters corresponding to the real world as the key idea of this work. Unfortunately, in practical tasks, the presence of a statistically significant set of parameters with the real world distribution is unrealistic. Usually they have some very limited set of parameters describing real tasks, the volume of which is not enough to create a full-fledged training dataset for deep learning. We propose to generate a statistical ensemble of parameters using the Generative Adversarial Network (GAN). The available real-life data set of limited size must be used as a set of TrueData to pre-train the GAN.

So, the first module of the general architecture to solve an abstract inverse problem is the Generator of the Ensemble of Parameters. A set of real data of possible parameters is used by the generator \( D_{true} = \{x_i_{true}\} \). The random Gaussian noise generator generates sequence \( N_z = \{z_i\} \). The GAN generator output is a sequence of parameters having an asymptotic distribution close to real data: \( GAN(D_{true}; z_i) \rightarrow x_i \)
3.2. Physical World Simulator

The second module of architecture of training the neural network to solve the inverse problem is the Simulator of the Physical World. Its purpose is to provide solutions of direct problem. The simulator must accurately reproduce the readings of the sensors corresponding to the given parameters and laws of the real world. The simulator can take into account both the processes connecting the parameters of the medium perturbation with the reaction of the medium, and the transformation of this reaction into specific sensor signals, depending on the design and principles of the sensors. In any case, the simulator would be implemented as a software package for numerically solving the phenomenological equations of the real world.

For example, it can be a solver of systems of partial differential equations (PDE) or a solver that uses the finite element method (FEM) for multi-media simulation [8]. The simulator can be a separate, very complex software package, both with open source and a proprietary product. Its main task in our notation is to solve a direct problem. Let’s name this module as RWS - Real World Simulator. Its function is to obtain sensor readings (module output) according to the specified parameters (module input): \( RWS(x_i) \rightarrow y_i \)

3.3. DeepNetwork Training Architecture

The trained neural network is the target third module of the described architecture. We cannot name the best types of deep neural networks to solve inverse problems. The network can be classified as regression network, since it must predict the tensor of the parameter data by the tensor of data from the sensors. Typical for this purpose is the use of a multi layer perceptron (MLP). Practice shows that the choice of model is largely determined by the specifics of the problem and it requires many models for comparison. We are confident that at this stage, the usage of AutoML libraries would be a perfectly acceptable recommendation.

Unfortunately, the imperfection of AutoKeras did not allow us to include the results of its application in this paper. The main requirement for the structure of the learning network is the presence of deep regularization. We have included in the number of network layers the regularizers available as part of Keras. Regularizers allow applying penalties on layer parameters or layer activity during optimization. These penalties are incorporated in the loss function that the network optimizes. The penalties are applied on a per-layer basis. In the learning process, the TDN deep neural network module maps data batch from sensors to a data set - estimated parameters.

\[ TDN(\{y_i\}) \rightarrow \{x_i\} \]

The process of network coefficients training is performed by algorithms based on stochastic gradient descent. The amount of loss is monitored at each input data batch. The criterion of stopping training on a pack is a decrease in the rate of decrease of the loss function. After that, the GAN starts to generate the next packet of parameters, RWS generates the next packet of sensor data and the process is repeated until the value of the loss function is stabilized. The quality assessment of the trained model is carried out by the method of k-fold cross-validation. In general, the iterative learning process is determined by the main cycle of the following form:

\[ GAN(D_{true}; z_i) \rightarrow x_i; RWS(x_i) \rightarrow y_i; \]

\[ TDN(\{y_i\}) \rightarrow \{x_i\} \]

We illustrate the detailed operation of the developed model in the next section. Here we restrict ourselves to the representation of diagram on Fig. 1.

4. Experiments

A detailed architecture is illustrated below with an example. The choice of the example of the inverse problem is based on the priority of didactic clarity and the ability of the reader to view all stages of the method independently. The authors considered this to be more important than
Figure 1. Deep neural network training model. Legend: z – Gauss noise generator; E – True Data array, G – GAN generator; D – GAN discriminator; S – Real World Simulator; N – target deep network; L – loss function computing module

the external practical significance of the solved problem. It is possible to show the solution of the electronic nose or vision problems by measuring the static electric field, which is much more interesting from the point of view of the result. However, it would be necessary to use extremely complex physical simulators, to train a complex deep network for many parameters. We choose a simple, familiar from school, but well visualized two-dimensional inverse problem: finding the initial velocity and angle of throwing of a heavy ball at an angle to the horizon from the measured throwing distance and angle of incidence of the ball. Analytically this problem can be solved if neglect air friction, but we will show how this problem is solved using a deep neural network with known air friction and a given ball mass.

4.1. Mathematical model of relations between the parameters of the throw and the measured distance and angle of incidence

Let a ball of mass m be thrown from the ground at an angle alpha with an initial velocity v_0. Friction against air is determined by the coefficient of friction k, the acceleration of gravity – g. As we know, in accordance with Newton’s laws, the motion of a ball in a flat earth coordinate system can be described with high accuracy by a system of differential equations:

\[
\begin{align*}
\frac{dv_x}{dt} &= -k/mv_y \sqrt{v_x^2 + v_y^2} \\
\frac{dv_y}{dt} &= -g - k/mv_y \sqrt{v_x^2 + v_y^2} \\
x(t+1) &= x(t) + \frac{v_x(t) + v_x(t+1)}{2} \Delta t; \\
y(t+1) &= y(t) + \frac{v_y(t) + v_y(t+1)}{2} \Delta t;
\end{align*}
\]

We will use a numerical simulation of the ball flight process in the form of a system of difference equations:

\[
\begin{align*}
v_x(t+1) &= v_x(t) - k/mv_x(t)\sqrt{v_x^2(t) + v_y(t)^2} \Delta t; \\
v_y(t+1) &= v_y(t) - g - k/mv_y(t)\sqrt{v_x^2(t) + v_y(t)^2} \Delta t
\end{align*}
\]

Thus, we can find the coordinates and the velocity vector of the ball at any time. In particular, we can determine the coordinate x = dist, in which the ball will fall to the ground (y = 0) and
at what angle \( \phi \) the velocity vector will be directed at this point. This task is a direct task. In
the following figure, you can see what the simulated ball trajectory looks like Fig. 2.

![Ball trajectory](image)

**Figure 2.** The trajectory connecting the initial angle and velocity with the throw distance and
the angle of incidence of the ball.

To describe the inverse problem, we assume that we know the pair \((\text{dist}, \phi)\) as the readings
of some sensors, and we need to find the velocity vector at the initial point \(x = 0, y = 0\), defined
as a pair \((v_0, \alpha)\). If there is no friction of the ball against the air \((k = 0)\), then the equations
of motion of the ball are easily integrated and one can obtain

\[
\text{dist} = \frac{v_0^2 \sin(2\alpha)}{g}; \quad \phi = \alpha
\]

Then the inverse problem is solved analytically as well

\[
v_0 = \sqrt{\frac{g \times \text{dist}/\sin(2\phi)}{\sin(2\alpha)}}; \quad \phi = \alpha
\]

This inverse problem has all features of ill-posedness. At small angles of incidence of the ball, the
error in measuring this angle affects the error in determining the initial velocity. Calculations
shows that variance of the estimate for the initial velocity depends on the value of the measured
angle of incidence. If we have constant variance of the error in measuring the angle of \(10^{-3}\) so
the variance of initial velocity can be equal 2.5 and much more if the angle is less than 0.1 rad.
(See Fig. 2.)

The influence of measurement errors on the estimation error will be of a similar nature for
our air friction model. Therefore, the regularization of the solution of the inverse problem should
include training on examples that do not use (very unlikely use) data corresponding to small
(less than 0.1 rad) dip angles. Of course, these angles cannot also exceed a value of about 1.57
(less than a right angle). We formed a special dataset corresponding to typical real values of the
throw parameters and named it TrueData.csv.
4.2. Data Deep Augmentation
In this paper, we use deep data augmentation to obtain the correct training datasets of the required sizes. To generate training batches of any size we use the Generative Adversarial Network – GAN – trained on real initial vectors. To do this, we use the simple Vanilla GAN architecture, focused on the generation of two-dimensional vectors. Training a GAN is very fast. The following figures illustrate how the statistical properties of the generated set of vectors relate the properties of real data. The red points are data of the real dataset, and the blue points are data generated by GAN. We use the calculation of the divergence of the Kullback-Leibler to quantify. The Fig. 3 shows distribution of 2D true data and generated by GAN fake data.

To compare how well the generator adjusts to the distribution of the initial angle parameter, we present here the histograms of the true and fake data set. The next figure 5 shows the histograms of true data and generated data by trained GAN.

After training completion, the trained generator() model of the GAN generator block can be used in the training architecture of the deep network to solve inverse problems. Figure 6 shows what a fragment of a pack of vectors of initial ball throw parameters looks like. In general, the generated variants correspond to the intuitive requirements of real data.

4.3. Deep network training for Inverse problem solving
The deep neural network architecture to solve the inverse problem considered here was investigated using a large generated training dataset and AutoKeras library. The task belongs to the regression class and the training is carried out according to the criterion of proximity of the network output values from the training dataset and those predicted by the network. Studies have shown that training one common neural network with an input dimension of two and an output dimension of two is significantly worse than training two separate networks with one

![Solution variance (v. 0) vs. data value](image)

**Figure 3.** Ill-posedness of inverse problem illustration. This is the dependence of the variance of the solution on the value of the source data.
**Figure 4.** Distribution of true data (red) and generated by GAN fake data (blue). Kulbak-Lebler divergence is equal 6.65

**Figure 5.** Comparison of histograms for real (TrueData.csv) and fake (generated) data array
Figure 6. The first 100 generated vectors of initial velocity parameter generated by our GAN output. The best results were obtained using two neural networks with 3 full layers and a ReLU activation function. We used this fact when organizing the training method proposed in the work with cyclic augmentation and deep regularization. So, in order to train to solve the inverse problem the neural network architecture includes 2 dense layers with an activation function of the ReLU type and regularization layers. The network is trained on a batch of a fixed size, but with constantly changing content. After training, the model can be used to solve the inverse problem on the current measurement data of the throwing range and angle of incidence - the inference mode. In practical application, the issue of the quality of the constructed model is important. As an estimate of the accuracy of the depth trained by cyclic augmentation, we use the R-Square metric. The typical results we got are R-Square = 0.9588. The following figures show the correspondence of the parameter predicted by the model with its true value.

The estimation of other parameter – initial velocity is slightly worse.

5. Conclusions
The study is a sort of analog of self-learning methods for networks like AlphaZero. We put the training data generation in the loop of a solution to the problem. The difference is that not the same network is used at each step, but physical world simulator generates all the necessary properties to test the network and to continue training until the network is perfect or its capabilities exhausted. The approach described in the article can be applied to a wide range of problems in which the construction of a neural network is required to restore model parameters from measured indirect data. In the near future, we will be ready to present a more meaningful application of our approach using the inverse physical problem for an electric field as an example. From the point of view of the internal mechanisms of the above method, one of the areas of work will be related to the inclusion of AutoML models directly in the augmented
**Figure 7.** Quality of inverse problem solution for initial angle – R-Square = 0.993

**Figure 8.** Quality of inverse problem solution for initial angle – R-Square = 0.949
learning cycle.

**Acknowledgments**
The article is dedicated to the memory of Dmitry Ponomarev, who gave impetus to the appearance of this work.

**References**
[1] Neto M, Duarte F and Neto-CNPq S 2013 An Introduction to Inverse Problems with Applications (Springer)
[2] Tikhonov A N 1963 *Dokl. Akad. Nauk SSSR* **151** 501–504
[3] Kratsios A 2020 *arXiv* 1910.03344 [stat.ML].
[4] Li H, Schwab J and Antholzer S 2018 *arXiv* 1803.00092 [math.NA].
[5] Xu K and Darve E 2019 *arXiv* 1901.07758 [math.NA]
[6] RN J 2007 Inverse Kinematics *Theory of Applied Robotics* (Boston: Springer)
[7] Krylov V V 2018
[8] Suli E 2012