Latent Surgical Interventions in Residual Neural Networks
Benjamin Donnot, Isabelle Guyon, Zhengying Liu, Marc Schoenauer, Antoine Marot, Patrick Panciatici

To cite this version:
Benjamin Donnot, Isabelle Guyon, Zhengying Liu, Marc Schoenauer, Antoine Marot, et al.. Latent Surgical Interventions in Residual Neural Networks. 2018. <hal-01906170>

HAL Id: hal-01906170
https://hal.archives-ouvertes.fr/hal-01906170
Submitted on 26 Oct 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Latent Surgical Interventions in Residual Neural Networks

Benjamin Donnot†‡, Isabelle Guyon*,‡, Zhengying Liu‡, Marc Schoenauer‡, Antoine Marot†, Patrick Panchiati‡

† ChaLearn, Berkeley, California.
‡ UPSud et Inria TAU, Université Paris-Saclay, France.
† RTE France

Abstract

We propose and study a novel artificial neural network framework, which allows us to model surgical interventions on a physical system. Our approach was developed to predict power flows in power transmission grids, in which high voltage lines are disconnected and re-connected with one-another from time to time, either accidentally or willfully. However, we anticipate a broader applicability. For several exemplary cases, we illustrate by simulation that our methodology permits learning from empirical data to predict the effect of a subset of interventions (elementary interventions) and then generalize to combinations of interventions never seen during training. We verify this property mathematically in the additive perturbation case. In terms of transfer learning, this is equivalent to training on data from a few source domains then, with a zero-shot learning, generalizing to new target domains (super-generalization). Our architecture bears resemblance with the successful ResNets, with the simple modification that interventions are encoded as an addition of units in the neural network. For applications to real historical data, from the French high voltage power transmission company RTE, we evaluate the viability of this technique to rapidly assess curative actions that human operators (dispatchers) could take in emergency situations. Integrated in an overall planning and control system, methods deriving from our approach could allow Transmission System Operators (TSO) to assess in real time many more alternative actions, reaching a better exploration-exploitation tradeoff, compared to presently deployed physical system simulator.

1 Background and motivations

In this paper, we are interested in speeding up the computation of power flows in power transmission grids using artificial neural networks, to emulate slower physical simulators. Key to our approach is the possibility of simulating the effect of actions on the grid topology. Such neural networks may then be used as part of an overall computer-assisted decision process in which human operators (dispatchers) ensure that the power grid is operated in security at all times, namely that the currents flowing in all lines are below certain thresholds (line thermal limits). We describe our application setting for concreteness, but anticipate a broader applicability of the techniques developed in this paper in various domains of physics, chemistry, manufacturing, biomedicine and others, in which some actions can be combined with each other, but running extensive simulations for each possible combination of such actions is computationally untractable.

Electric power generated in production nodes (such as power plants) is transmitted towards consumption nodes in a power grid. The power lines enable this transmission through substations interconnecting them. Each pattern of connections is referred to as a grid topology. This topology is
Electricity must be transported from production nodes (brown circles) to consumption nodes (green circles). They are interconnected through a network (grid) of transmission lines (red lines), connected at substations (red squares). The injections $x = (x_1, x_2, x_3, x_4)$, which include both productions and consumptions, must add up to zero. The way in which lines are interconnected is referred to as grid topology $\tau$. The flows in the red lines $y$ result from the injections and the topology: $y = S(x, \tau)$. At any time, the grid operators (or dispatchers) must make sure that the network is operated in security and no line exceeds its thermal limit (a current flow above which the line might melt). (a) Line $y_4$ goes over its thermal limit 100. (b) A change in topology (splitting of node 6) brings $y_4$ back to its thermal limit.

The space of all possible grid topologies is huge and grows exponentially with the number of substations. For example, the French high-voltage transmission network includes $N \approx 6200$ substations, with more than a dozen possible configurations per substation and therefore more than $10^N$ possible grid topologies. Even if only a small number of those are achievable, the search space is still humongous. In practice, Transmission System Operators (TSOs) limit their dispatchers to choosing from a very limited set of candidate operations on the grid topology. However, operating the grid is becoming increasingly complex for a number of reasons, including the advent of renewable energies (introducing wind and solar power that are little predictable), the globalization of energy markets (also introducing fluctuations of consumptions harder to predict), the growth in consumption and the concurrent limitations on new line construction (to protect the environment). Therefore, it is becoming urgent to optimize more tightly the operation of the grid with more flexibility, by considering a broader range of topological changes operated more frequently, without compromising security.

Using neural networks to emulate the power grid is not new. As early as 1995, [10] proposed to run load flow computation with a neural network. The objective however was to reproduce faithfully the calculation carried out in regular load-flow simulator with a parallelizable neural network implementation, for a fixed topology. Neural networks have also been used for short-term load forecasting (see [7] for a recent example). A competition just ended on a similar topic (https://web.see4c.eu/), in which the participants were challenged to forecast power flows given historical records. The problem we are tackling in this paper is rather different: The goal is to quickly evaluate the flows in all lines for variants of the grid topology (line inter-connection pattern), for given injections (production and consumption, balancing each other). Using one neural network for each topology is unrealistic,
thus one must find a way to use both injections and topology as input to the estimator. A related problem was addressed in [4, 5] where the authors used neural networks to predict power flows under unplanned contingencies (equipment failures, e.g. lines disconnected by a storm). This paper goes beyond the analysis of contingencies and study planned coordinated actions reconfiguring the grid topology to obtain desired effects, using real historical data.

Our method can be seen as a transfer learning approach [11]: we train a neural network to generalize across domains (each topology being a domain). After learning to make predictions in a reference topology, the neural network can adapt with few shot learning [3] to new domains (new topologies). We also demonstrate that our predictive models exhibit a “super-generalization” property that can be seen as a zero shot learning: The neural network generalizes to new domains for which it has not been trained at all. This is made possible by encoding topologies as inputs that modify the neural network architecture. This line of work has been pursued in [8] for one shot learning, in [9] for few shot learning, and in [13] for zero shot learning, although with quite different approaches. Our method combines ideas from residual networks [6], dropout [14], and conditional computation (e.g. 2). The idea behind the conditional computation is to adapt the architecture of a neural network depending on its inputs. Both the weights and the architecture are trained. This conditional computation has been adapted with success by [12] for image classification for example. More recently Minhui Zou et al. [16] show that adding some units inside an already trained neural network goes beyond the analysis of contingencies and study planned coordinated actions reconfiguring the grid topology to obtain desired effects, using real historical data.

2 Proposed methodology

The objective is to approximate a function \( y = S(x, \tau) \) that maps input data \( x \) (e.g. power productions and consumptions - also called injections) to output data \( y \) (e.g. power flows on transmission lines), parameterized by a discrete topology vector \( \tau \), taking values in an intervention space (all possible power-grid topologies e.g. line interconnections, in our example). This section first introduces notations, then describes the proposed neural network architecture, and demonstrates its behavior on a simple example. Finally, a mathematical analysis proves some properties of this architecture.

2.1 Notations and definitions

Let \( x \in \mathcal{X} \subset \mathbb{R}^p \) be the input vector of the system (dim \( x = p \) = number of injections), \( y \in \mathcal{Y} \subset \mathbb{R}^l \) the output vector (dim \( y = l \) = number of lines), and \( \tau \in \{0, 1\}^c \) the action vector (dim \( \tau = c \)). The action space is represented as a binary vector without any particular encoding of (discrete) actions. In particular, unary actions (e.g. single power line disconnection) are not necessarily one-hot encoded and therefore \( c \) is not necessarily equal to the number of unary actions. If one-hot encoding of unary actions is used, then if unary action \( a_1 \) is encoded with \( \tau^{(1)} = (1, 0, 0, \ldots) \), unary action \( a_2 \) is encoded with \( \tau^{(2)} = (0, 1, 0, \ldots) \) and double action \( a_{1,2} \) is encoded with \( \tau^{(12)} = (1, 1, 0, \ldots) \).

In general, double action \( a_{1,2} \) is encoded with \( \tau^{(12)} = \tau^{(1)} \lor \tau^{(2)} \) (element-wise "or" operation). More generally, let \( \tau^I \) be the vector in \( \mathbb{R}^c \) such that \( \tau^I = \bigvee_{i \in I} \tau^{(i)} \). Here \( I \subset \{1, \ldots, u\} \) where \( u \) is the number of unary actions. If one-hot encoding of unary action is used, then \( (\tau^I)_i = 1 \) if \( i \in I \), and 0 otherwise. For any encoding of actions, by convention, \( \tau^0 = (0, 0, \ldots) \) will represent the absence of action, corresponding to the system in its reference topology.

Let \( y = S(x, \tau) \) be the ground truth of the system’s response to input \( x \) in topology \( \tau \). The ground truth here will be given by the physical simulator. In contrast, the approximation made by the neural network will be denoted \( \hat{y} = \hat{N}(x, \tau) \).

Similarly to other learning problems, for any fixed topology \( \tau \), training data pairs \( \{x, y\} \) are drawn i.i.d. according to an unknown probability distribution\(^2\). We call simple generalization the capability of \( \hat{y} = \hat{N}(x, \tau) \) to approximate \( y = S(x, \tau) \) for test inputs \( x \) not pertaining to the training set, and when \( \tau \) values are drawn i.i.d. from a source domain distribution that remains the same in training and test data (this covers in particular the case of a fixed \( \tau \)).

\(^2\)In practice, \( x \) is drawn randomly, but \( S(x, \tau) \) is a deterministic function implementing Kirchhoff’s circuit laws. No noise term is involved in the calculation of \( y \) from \( x \).
Conversely, if **values of \( \tau \) are not drawn similarly in training and test data**, i.e. \( \tau \) is drawn according to a source domain distribution in training data and from a target domain distribution in test data, then we will talk about **super-generalization**. This setting is a particular case of transfer learning from a source domain to a target domain.

### 2.2 Neural network architecture

The proposed Latent Surgical Intervention (LSI) model represented schematically in Figure 2 is given by:

\[
\hat{y} = D(E(x) + d(\underbrace{e(E(x)) \odot \tau}_{\text{some units are masked depending on } \tau}))
\]

where \( E \) and \( e \) (encoders) and \( D \) and \( d \) (decoders) are all differentiable functions (typically implemented as artificial neural networks). The \( \odot \) operation denotes the component-wise multiplication. If the system is in the reference topology \( \tau^0 \), predictions are made according to \( \hat{y} = D(E(x)) \).

Indeed, assuming that \( d(0) = 0 \), for \( \tau^0 = (0, 0, 0, \ldots) \), we have \( d(e(E(x)) \odot \tau) = 0 \), thus the LSI block lets the information flow directly, without modification.

Table 1: **Toy data with additive perturbations.** We illustrate the superiority of LSI over the baseline architecture in a case of additive perturbations: a generative reference system \( S(x, \tau^0) = F(x) \), unary perturbations \( S(x, \tau^{(i)}) = F(x) + \alpha_i \), and binary perturbations \( S(x, \tau^{(ij)}) = F(x) + \alpha_i + \alpha_j \). Here, \( F(x) = [x_1 \cos(x_2), x_1 \sin(x_2)] \). We report test data Mean Squared Error (MSE ± one std) across 20 independently trained neural networks in two settings. **SETTING 1**: 100 training data points in Source 1 and Source 2 domains. **SETTING 2**: only 1 training data point in Source 1 and Source 2 domains. In both cases, 1 000 training points are available in Source 0 domain. The best results are highlighted in **bold**, extreme variance is highlighted in **red**. Test MSE are computed on 1000 points per domain, not included in the training set.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Action (( \tau ))</th>
<th>Data generation</th>
<th><strong>SETTING 1</strong> 100 training points in target 1 and 2</th>
<th><strong>SETTING 2</strong> 1 training point in source 1 and 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs (all domains)</td>
<td>-</td>
<td>( x_1 \sim N(1, 0.1) ) ( x_2 \sim U(0, 5\pi/3) )</td>
<td>0.003 0.0059 0.001 0.0003</td>
<td>0.0010 0.0008 0.0002 0.0001</td>
</tr>
<tr>
<td>Source 0 (reference)</td>
<td>( \tau^0 = (0, 0) )</td>
<td>( y_1 = x_1 \cos(x_2) ) ( y_2 = x_1 \sin(x_2) )</td>
<td>0.003 0.001 ±0.0001</td>
<td>0.0012 0.0003 ±0.0001 ±0.0001</td>
</tr>
<tr>
<td>Source 1</td>
<td>( \tau^{(1)} = (1, 0) )</td>
<td>( y_1 = x_1 \cos(x_2) + 2.5 ) ( y_2 = x_1 \sin(x_2) )</td>
<td>0.005 0.001 ±0.0001</td>
<td>0.01 0.02 ±0.07 ±0.07</td>
</tr>
<tr>
<td>Source 2</td>
<td>( \tau^{(2)} = (0, 1) )</td>
<td>( y_1 = x_1 \cos(x_2) ) ( y_2 = x_1 \sin(x_2) + 2.5 )</td>
<td>0.005 0.001 ±0.0002 ±0.001</td>
<td>2 0.01 ±0.04 ±0.04</td>
</tr>
<tr>
<td>Target</td>
<td>( \tau^{(12)} = (1, 1) )</td>
<td>( y_1 = x_1 \cos(x_2) ) ( y_2 = x_1 \sin(x_2) + 2.5 )</td>
<td>0.6 0.001 ±0.003 ±0.001</td>
<td>5 0.03 ±0.07 ±0.09</td>
</tr>
</tbody>
</table>
2.3 Super-generalization ability of LSI

In this section, we compare and contrast the LSI architecture with a baseline architecture in which the topology information \( \tau \) is entered as additional inputs to the first module \( E(\cdot) \), see Figure 2. The experimental setting and results are described in Table 1 and illustrated in Figure 3. As we can see on this figure, the LSI model is capable of super-generalization: it learns only from Source domain data (many points from Source 0, but only one point from Source 1 and from Source 2); then it generalizes to Target domain data. The results of Table 1 provide more details and show that the baseline model cannot super-generalize similarly to the LSI. Both baseline and LSI successfully learn the training data distribution ("Training error" rows) and successfully generalizes to unseen test data when \( \tau = 0 \) (Source 0 row: this is a standard generalization framework in which test data come from the same distribution as training data). Both methods generalize to Source 1 and Source 2 data when enough training examples are provided (Source 1 and Source 2 rows, SETTING 1). But the baseline method does not generalize as well as the LSI when very little data are available in the other two source domains (Source 1 and Source 2 rows, SETTING 2). In this setting (SETTING 2), we can observe that, for the baseline method, the variance of the error across repeated experiments is very high. This variance is much smaller for the LSI architecture (by almost two orders of magnitude). Low variance in the predictions is a crucial feature in application to power systems.

Finally and most importantly, the "Target" row (last one) reports results in the Target domain for which no data are available at training time. This is where LSI demonstrates its super-generalization capabilities and the baseline method fails in all settings. A more detailed study of this test cases is carried out in the supplementary material.

Our illustrative example was taken from the family of generative models with additive perturbations. Our experimental successes prompted us to investigate whether the super-generalization property of LSI architectures could be mathematically proven, at least in a limited setting. Consider a data generating system \( S(x, \tau) \) satisfying:

\[
S(x, \tau^0) = F(x) \\
S(x, \tau^i) = F(x) + \epsilon_i(x), \quad i = 1, \ldots, c
\]  

(2)

and

\[
S(x, \tau^\mathcal{I}) = F(x) + \sum_{i \in \mathcal{I}} \epsilon_i(x), \quad |\mathcal{I}| \geq 2
\]  

(3)

\(^3\)the learned function depends on the initialization and the mini-batch ordering.
for some (unknown) deterministic functions $F(x), \epsilon_1(x), \ldots, \epsilon_c(x)$. We prove two theorems in supplementary material showing that a LSI architecture with linear submodules $d$ and $D$ exhibits super-generalization in the following sense: if such a LSI architecture $NN(x, \tau)$ can learn from empirical data to make “good” predictions on data triplets $(x, \tau, y)$ coming only from Source domains defined by equations (2), then this network will also be able to make “good” predictions on data coming from Target domains given by equation (3).

In the next section we show experimental success of super-generalization of LSI architectures in non-linear cases, for which we do not have yet mathematical guarantees.

3 Predicting flows in power grids

In this section, we present results for our target application: predicting power flows in power grids. First we test our method on simulated data, and benchmark our proposed LSI architecture with the baseline architecture of Figure 2, as well as with the “DC approximation” (a linearization of the power flow equations widely used in the power system community). Second, we use real data coming from the “Toulouse” area, South West part of the French power grid, to predict the flows in a near real time process for complex interventions given partial information.

3.1 The case 118 benchmark: Synthetic data from a physical simulator

We first conduct controlled experiments on a standard medium-size benchmark from “Matpower” [13], a library commonly used to test power system algorithms [1]. We use case118, a simplified version of the Californian power grid. This test case includes 99 consumptions, 54 productions (dim $x = 153$), and 186 power lines (dim $y = 186$). Topology changes consist in reconfiguring line connections in one or more substations (see Figure 1). A study of the reference grid topology indicates that there are 11 558 possible unary actions (corresponding to single node splitting or merging, compared to the reference topology). We sampled randomly 100 for training (Source domains: $\tau(i) \in T_{Source}$). We sampled 50000 different inputs $x$ in the reference topology ($\tau^0$). For each $\tau(i)$, we sampled 1000 inputs $x$ among these possible changes. We used the Hades2 software to compute the flows $y$ in all cases. This resulted in a training set of 150 000 rows (each row being one triplet $(x, \tau(i), y)$). We created an independent test set of the same size in a similar manner, keeping the same $\tau(i) \in T_{Source}$. We proceeded differently for the Target dataset. We sampled 1500 (Target domains: $\tau(j) \in T_{Target}$) among the 4950 possible binary actions $\tau(j) = \tau(i) \lor \tau(j)$, $\tau(i) \in T_{Train}$. Then, for each of these 1500 $\tau(i)$, we sampled 100 inputs $x$ (with the same distribution as the one used for the training and regular test set). We used the same physical simulator to compute the flows $y$ coming only from Source do-

We compared the proposed LSI method with two benchmarks: the DC approximation, a standard baseline in power systems, which is a linearization of the AC (Alternative Current) non-linear power flow equations, and the baseline neural network architecture (Figure 2) in which $\tau$ is simply an input (referred to as "baseline" network). We optimized the L2 (mean-square) error, using the Adam optimizer from Tensorflow. To make the comparison least favorable to the LSI architecture, all hyper-parameters of the neural network (learning rates, number of units) were optimized by cross-validation for the baseline network architecture and are exposed in the supplementary material.

The results shown in Figure 4 indicate that the LSI method (green curves) performs better than the DC approximation (black dashed line) both for regular generalization and super-generalization. In contrast, the baseline neural network architecture (red curves) does not outperform the DC approximation in the super-generalization case (Figure 4b).

Figure 4a indicates that the LSI architecture may possibly be slightly under-fitting, since it is outperformed by the baseline neural network for regular generalization. This can be explained by the fact that the baseline network has many more available connections to learn from (no unit in the inner

---

4For simplicity, we assumed $l = 1$ and thus all $y$'s are scalar, but the generalization to the case with any $l$ is straightforward. Consistent with the power system application setting, we assumed that data are collected without noise, but our results could be extended to noisy cases.

layer being disabled). Adding more hidden units in the LSI might yield yet better performance; more systematic experiments are under way.

Figure 5 shows that the baseline neural network architecture is not viable: not only does it perform worse than the DC approximation, but its variance is quite high. While it is improving in regular generalization with the number of training epochs, its super-generalization performances get worse.

3.2 Real French ultra high voltage power grid data

We now present results on a part of the French ultra high voltage power grid: the "Toulouse" area with 246 consumptions, 122 productions 387 lines and 192 substations often split in a variable number of nodes. Similarly to the previous artificial data experiments, the inputs \( x \) are injections (productions and consumptions - this time of dimension \( \text{dim} \; x = 387 \)) and the outputs \( y \) are flows (with \( \text{dim} \; y = 387 \)). In this study, \( x \) and \( y \) come from real historical data.

One similarity between real and synthetic data is that \( y \) (the flows) have also been calculated from \( x \) by a physical simulator (not measured). But there are several differences between real and synthetic data. First, for the emulated data we could deliberately chose which topologies to include in \( T^{\text{Source}} \) and \( T^{\text{Target}} \) domains: the emulated data is a controlled environment in which changes in \( \tau \) were our own interventions. In contrast, real data consist of played-back historical situations on which we cannot deliberately intervene. Thus, we used data from 2012 to May 2017 for \( T^{\text{Source}} \) and data from June and July 2017 for \( T^{\text{Target}} \). This favored changes in distribution of \( \tau \).

Another key difference in real data is the actions space. In our synthetic experiments, actions induced changes in line interconnections via nodes splitting and merging (grid topology changes). In real data actual grid states are not precisely labelled. Instead, only information on line outages are available to us. This is problematic because, when power lines are taken out for maintenance or damaged by storms, dispatchers make curative/preventive node splitting actions to guarantee grid security. Unfortunately no records are made of such induced topological changes. Therefore we can only use information on line disconnections as surrogate for actual grid topology complex interventions. This makes the task of the neural network much harder: it must learn the effects of latent topological changes.

In this context \( \text{dim} \; \tau = 387 \), the number of power lines in the power grid. At the time of recording of our historical data, the physical simulator computed \( y \) as a function of \( x \) and the actual topology. Thus, from our perspective, the system \( S \) is a function of \( x, \tau \), and latent factors (unrecorded.
In our experiments, we achieved a speed-up of 300 times using the LSI architecture compared to running the physical simulator on the synthetic dataset (power grid of 118 nodes). Provided the dataset are stored in the computer memory, our experiments on the Toulouse area show this speed up could be as high as 2000 times compared to running one physical simulation. These speeds up were obtained using a single high end Graphical Processing Unit (GPU) Nvidia Titan X. Further works include scaling up our method computationally to the entire French extra high voltage power grid.

Currently, the error obtained on the “Toulouse” dataset on super-generalization is still too high for our model to be used in production. But if this error goes down to the level of error achieved on the regular generalization (see Figure 5a), this process could be implemented for real time operation support. This goal seems achievable considering the results on the synthetic experiments, where the super-generalization error is really close to the regular generalization ones. A better theoretical understanding of our model could help us design a more powerful method. Further work is under way to analyze various cases of data generating models and their associated distortions in an effort to create a benchmark suite to test super-generalization abilities.

Our approach bridges transfer learning and the estimation of effects of interventions in systems emulated by neural networks, a topic that deserves more in depth studies, including researching theoretical guarantees for “super-generalization”. Our empirical evaluation of the super-generalization capabilities of the LSI architecture demonstrates the viability of the method and could inspire research in other application domains.

Table 2: Comparison of datasets and experimental design. We illustrate the super-generalization capabilities of the method. Training is performed simultaneously with data triplets \( \{x, \tau, y\} \) both from the source domain \( \tau = \tau^{(0)} \) and the target domains \( \tau \in T_{\text{Source}} \). Then the model is tested for super-generalization using data triplets \( \{x, \tau, y\} \) from the target domains with \( \tau \in T_{\text{Target}} \). The architecture of Latent Surgical Interventions (LSI) has been evaluated on a number of test cases, as well as the LSI.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Source domain ( \tau = \tau^{(0)} ) lots of training data</th>
<th>Source domains ( \tau = \tau^{(i)} \in T_{\text{Source}} ) few training data for each ( \tau )</th>
<th>Super-generalization ( \tau \in T_{\text{Target}} ) novel ( \tau ) not necessarily in training data</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTE historical data</td>
<td>( \approx 5 ) years of data with ( \tau \approx \tau^{(0)} )</td>
<td>Intermixed ( \tau ) different from ( \tau^{(0)} ), same 5 years</td>
<td>Next month: new ( \tau \approx \tau^{(0)} ) not in training data</td>
</tr>
<tr>
<td>Synthetic dataset (physical simulator)</td>
<td>5000 samples for ( \tau \equiv \tau^{(0)} )</td>
<td>100 samples per ( \tau = \tau^{(i)} )</td>
<td>100 samples per ( \tau = \tau^{(i)} )</td>
</tr>
<tr>
<td>Toy examples (artificial data)</td>
<td>1000 generated for ( \tau \equiv \tau^{(0)} )</td>
<td>1 or 100 sample per ( \tau = [1,0] ) or ( \tau = [0,1] )</td>
<td>1000 samples testing super-generalization to ( \tau = [1,1] )</td>
</tr>
</tbody>
</table>

Figures 5a displays the learning curves obtained on these real data. We arrive at the same conclusions as in the previous subsection: the LSI model is able to learn a similar distribution than the one it is trained on (figure 5a). It can also generalize to unseen grid states better than the reference architecture (figure 5a), which is a critical property for the application. The baseline architecture performs well on data distributed similarly to training data (figure 5a) but does not super-generalize as well as the LSI.

4 Discussion and conclusion

The architecture of Latent Surgical Interventions (LSI) has been evaluated on a number of test cases, and we summarize their experimental design in Table 2. In all cases, training is performed on data triplets \( \{x, \tau, y\} \), for which \( \tau \in T_{\text{Source}} \) belong to source domains. We demonstrated that LSI generalizes not only by approximating well \( y \) for new values of \( x \) when \( \tau \in T_{\text{Source}} \), but also when \( \tau \in T_{\text{Target}} \) (super-generalization). The toy example and the synthetic data were used as a sanity check. It allowed us to verify that the method works in the case of additive superposition. The synthetic power grid example allowed us to measure ourselves against a standard baseline (the DC approximation) on a standard benchmark (the case 118 example of Matpower). The LSI architecture shows very competitive results. Finally, the experiments on real data show real promise.

In our experiments, we achieved a speed-up of \( \approx 300 \) times using the LSI architecture compared to running the physical simulator on the synthetic dataset (power grid of 118 nodes). Provided the dataset are stored in the computer memory, our experiments on the Toulouse area show this speed up could be as high as 2000 times compared to running one physical simulation. These speeds up were obtained using a single high end Graphical Processing Unit (GPU) Nvidia Titan X. Further works include scaling up our method computationally to the entire French extra high voltage power grid.

Currently, the error obtained on the “Toulouse” dataset on super-generalization is still too high for our model to be used in production. But if this error goes down to the level of error achieved on the regular generalization (see Figure 5a), this process could be implemented for real time operation support. This goal seems achievable considering the results on the synthetic experiments, where the super-generalization error is really close to the regular generalization ones. A better theoretical understanding of our model could help us design a more powerful method. Further work is under way to analyze various cases of data generating models and their associated distortions in an effort to create a benchmark suite to test super-generalization abilities.

Our approach bridges transfer learning and the estimation of effects of interventions in systems emulated by neural networks, a topic that deserves more in depth studies, including researching theoretical guarantees for “super-generalization”. Our empirical evaluation of the super-generalization capabilities of the LSI architecture demonstrates the viability of the method and could inspire research in other application domains.
References


