Latent Surgical Interventions in Residual Neural Networks

Benjamin Donnot, Isabelle Guyon, Zhengying Liu, Marc Schoenauer, Antoine Marot, Patrick Panciatici

Latent Surgical Interventions in Residual Neural Networks

Benjamin Donnot[†,*] **Isabelle Guyon**[•‡], **Zhengying Liu**[‡],
Marc Schoenauer[‡], **Antoine Marot**[†], **Patrick Panciatici**[†]
• 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 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.

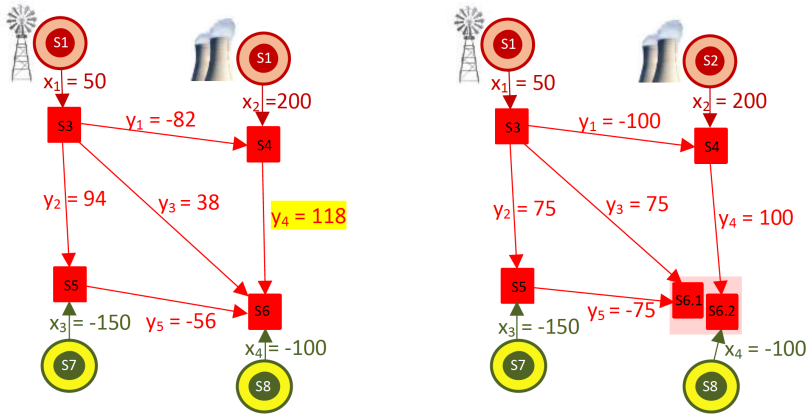
1 Background and motivations

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

[*]Benjamin Donnot corresponding authors: benjamin.donnot@inria.com

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.



(a) (b)

Figure 1: **Mini example of power transmission grid.** 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 $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$, which include both productions and consumptions, must add up to zero. The way in which lines are interconnected is referred to as grid topology $\boldsymbol{\tau}$. The flows in the red lines $\boldsymbol{y}$ result from the injections and the topology: $\boldsymbol{y} = S(\boldsymbol{x}, \boldsymbol{\tau})$. 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.

constantly changing around a reference topology, due to different events such as random equipment failures and planned maintenance operations. In the toy example of Figure 1 we make bold simplifying assumptions for illustrative purposes (constant voltage, DC currents, no losses). Thus, power flows indicated on the lines are proportional to currents. Here we picked an arbitrary number of units and assumed that the thermal limit in all red transmission lines was 100. Positive currents flow in the direction of the arrows (and negative currents in the opposite direction). At each substation, the sum of all incoming currents must equal the sum of outgoing currents. In Figure 1-a, line y_4 goes over its thermal limit of 100. A corrective action in Figure 1-b (node splitting) brings it back below its thermal limit. The object of this paper is to allow dispatchers to quickly assess the effectiveness of a large number of such candidate topology changes as preventive or corrective actions.

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 is enough to bias its predictions. This work is based on the same idea: the Latent Surgical Interventions (LSI) consist in adding units at the heart of a residual neural network. Our contribution is to propose and study this novel simple architecture while demonstrating its effectiveness in application to power flow calculations in grids with varying topology.

2 Proposed methodology

The objective is to approximate a function $\boldsymbol{y} = S(\boldsymbol{x}, \boldsymbol{\tau})$ that maps input data $\boldsymbol{x}$ (*e.g.* power productions and consumptions - also called injections) to output data $\boldsymbol{y}$ (*e.g.* power flows on transmission lines), parameterized by a discrete topology vector $\boldsymbol{\tau}$, 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 $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^p$ be the input vector of the system (dim $\boldsymbol{x} = p =$ number of injections), $\boldsymbol{y} \in \mathcal{Y} \subset \mathbb{R}^l$ the output vector (dim $\boldsymbol{y} = l =$ number of lines), and $\boldsymbol{\tau} \in \{0,1\}^c$ the action vector (dim $\boldsymbol{\tau} = c$). The action space is represented as 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 $\boldsymbol{\tau}^{(1)} = (1, 0, 0, \dots)$, unary action a_2 is encoded with $\boldsymbol{\tau}^{(2)} = (0, 1, 0, \dots)$ and double action $a_{1,2}$ is encoded with $\boldsymbol{\tau}^{(12)} = (1, 1, 0, \dots)$. In general, double action $a_{1,2}$ is encoded with $\boldsymbol{\tau}^{(12)} = \boldsymbol{\tau}^{(1)} \vee \boldsymbol{\tau}^{(2)}$ (element-wise "or" operation). More generally, let $\boldsymbol{\tau}^{\mathcal{I}}$ be the vector in $\mathbb{R}^c$ such that $\boldsymbol{\tau}^{\mathcal{I}} = \bigvee_{i \in \mathcal{I}} \boldsymbol{\tau}^{(i)}$. Here $\mathcal{I} \subset \{1, \dots, u\}$ where u is the number of *u*nary actions. If one-hot encoding of unary action is used, then $(\boldsymbol{\tau}^{\mathcal{I}})_i = 1$ if $i \in \mathcal{I}$, and 0 otherwise. For any encoding of actions, by convention, $\boldsymbol{\tau}^{\emptyset} = (0, 0, \dots)$ will represent the absence of action, corresponding to the system in its reference topology.

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

Similarly to other learning problems, for any fixed topology $\boldsymbol{\tau}$, training data pairs $\{\boldsymbol{x}, \boldsymbol{y}\}$ are drawn *i.i.d.* according to an unknown probability distribution[2]. We call **simple generalization** the capability of $\hat{\boldsymbol{y}} = NN(\boldsymbol{x}, \boldsymbol{\tau})$ to approximate $\boldsymbol{y} = S(\boldsymbol{x}, \boldsymbol{\tau})$ for test inputs $\boldsymbol{x}$ not pertaining to the training set, and when $\boldsymbol{\tau}$ **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 $\boldsymbol{\tau}$).

[2]In practice, $\boldsymbol{x}$ is drawn randomly, but $S(\boldsymbol{x}, \boldsymbol{\tau})$ is a deterministic function implementing Kirchhoff's circuit laws. No noise term is involved in the calculation of $\boldsymbol{y}$ from $\boldsymbol{x}$.

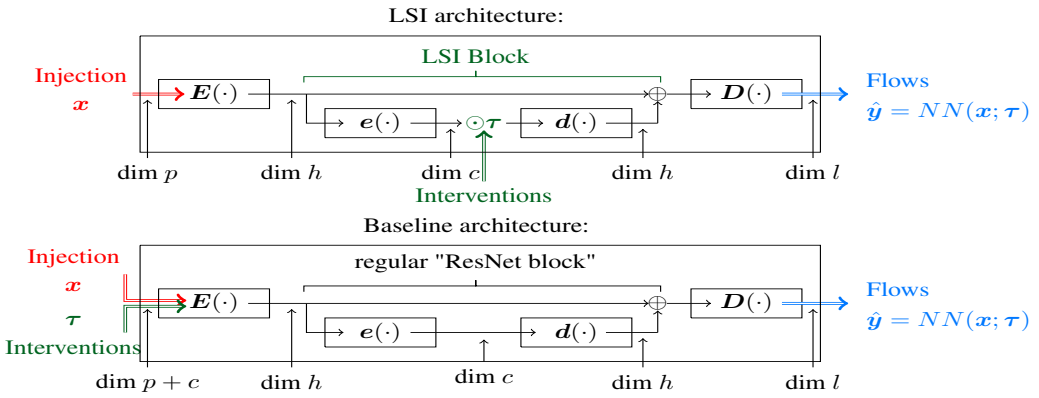


Figure 2: **LSI and baseline architecture**. The main novelty of the LSI architecture is that interventions are introduced by performing an element-wise multiplication $\odot$ with a binary vector $\boldsymbol{\tau}$, which has the effect of selectively activating or inactivating units. The baseline architecture includes instead a standard residual network block [6]. Interventions are introduced as additional inputs to the first block. In both architectures, multiple ResNet or LSI blocks may be stacked.

Conversely, if **values of $\boldsymbol{\tau}$ are not drawn similarly in training and test data**, *i.e.* $\boldsymbol{\tau}$ 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

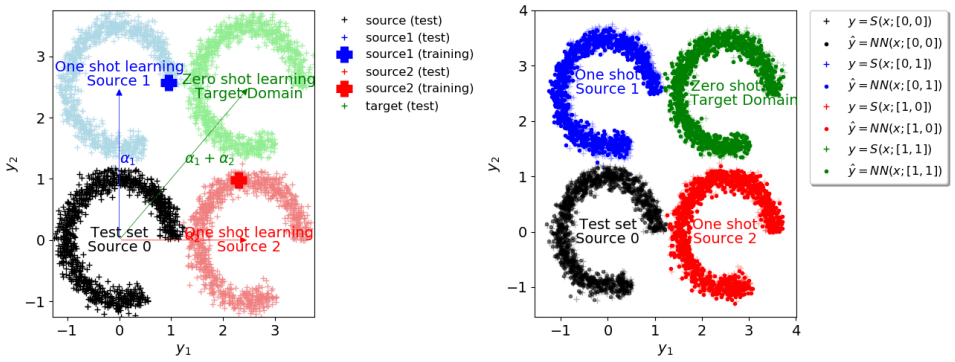
The proposed Latent Surgical Intervention (LSI) model represented schematically in Figure 2 (top) is given by:

$$\hat{\boldsymbol{y}} = \boldsymbol{D}(\boldsymbol{E}(\boldsymbol{x}) + \boldsymbol{d}(\underbrace{\boldsymbol{e}(\boldsymbol{E}(\boldsymbol{x})) \odot \boldsymbol{\tau}}_{\text{some units are masked depending on } \boldsymbol{\tau}})) \tag{1}$$

where $\boldsymbol{E}$ and $\boldsymbol{e}$ (encoders) and $\boldsymbol{D}$ and $\boldsymbol{d}$ (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 $\boldsymbol{\tau}^{\emptyset}$, predictions are made according to $\hat{\boldsymbol{y}} = \boldsymbol{D}(\boldsymbol{E}(\boldsymbol{x}))$. Indeed, assuming that $d(\mathbf{0}) = \mathbf{0}$, for $\boldsymbol{\tau}^{\emptyset} = (0, 0, 0, \dots)$, we have $\boldsymbol{d}(\boldsymbol{e}(\boldsymbol{E}(\boldsymbol{x})) \odot \boldsymbol{\tau}) = \mathbf{0}$, 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 $S(\boldsymbol{x}, \boldsymbol{\tau}^{\emptyset}) = F(\boldsymbol{x})$, unary perturbations $S(\boldsymbol{x}, \boldsymbol{\tau}^{(i)}) = F(\boldsymbol{x}) + \alpha_i$, and binary perturbations $S(\boldsymbol{x}, \boldsymbol{\tau}^{(ij)}) = F(\boldsymbol{x}) + \alpha_i + \alpha_j$. Here, $F(\boldsymbol{x}) = [x_1 \cos(x_2), x_1 \sin(x_2)]$ and $\alpha_i = 2.5$. We report test data Mean Squared Error (MSE $\pm$ 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.

Domain	Action ($\boldsymbol{\tau}$)	Data generation	**SETTING 1** 100 training points in target 1 and 2		**SETTING 2** (see Figure 3) 1 training point in source 1 and 2	
			BASELINE	**LSI**	BASELINE	**LSI**
Inputs (all domains)	-	$x_1 \sim \mathcal{N}(1, 0.1)$ $x_2 \sim \mathcal{U}(0, 5\pi/3)$	0.003 $\pm$0.001	**0.00059** $\pm$0.00009	0.0010 $\pm$0.0006	**0.0003** $\pm$0.0001
Source 0 (reference)	$\boldsymbol{\tau}^{\emptyset} = (0, 0)$	$y_1 = x_1 \cos(x_2)$ $y_2 = x_1 \sin(x_2)$	0.003 $\pm$0.001	**0.001** $\pm$0.001	0.0012 $\pm$0.0007	**0.0003** $\pm$0.0002
Source 1	$\boldsymbol{\tau}^{(1)} = (1, 0)$	$y_1 = x_1 \cos(x_2) + 2.5$ $y_2 = x_1 \sin(x_2)$	0.005 $\pm$0.003	**0.001** $\pm$0.001	4 $\pm$2	**0.02** $\pm$0.07
Source 2	$\boldsymbol{\tau}^{(2)} = (0, 1)$	$y_1 = x_1 \cos(x_2)$ $y_2 = x_1 \sin(x_2) + 2.5$	0.005 $\pm$0.002	**0.001** $\pm$0.001	2 $\pm$2	**0.01** $\pm$0.04
Target	$\boldsymbol{\tau}^{(12)} = (1, 1)$	$y_1 = x_1 \cos(x_2) + 2.5$ $y_2 = x_1 \sin(x_2) + 2.5$	0.6 $\pm$0.5	**0.001** $\pm$0.001	5 $\pm$7	**0.03** $\pm$0.09



(a) Training examples overlaid on ground truth (b) Test ex. predictions overlaid on ground truth

Figure 3: **Toy example of transfer learning with LSI**: The Source domains are color-coded as {black, red, blue } and the Target domain as green. The data, drawn according to Table 1 (Setting 2), are from a bi-variate regression problem of which we show only the values of $\boldsymbol{y}$ for various color-coded values of $\boldsymbol{\tau}$; the values of $\boldsymbol{x}$ are drawn randomly and identically in training and test data. (a) Training data include all black crosses, a **single** red point, a **single** blue point, and **NO** green point at all. (b) The LSI neural network generalizes to new test data from all source domains (including those for which it has seen a single red or blue training example) and to the Target green domain (for which it has seen no training example at all).

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 $\boldsymbol{\tau}$ is entered as additional inputs to the first module $E(.)$, 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 $\boldsymbol{\tau} = 0$ (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[3]. 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 $S(\boldsymbol{x}, \boldsymbol{\tau})$ satisfying :

$$\begin{cases} S(\boldsymbol{x}, \boldsymbol{\tau}^{\emptyset}) = F(\boldsymbol{x}) \\ S(\boldsymbol{x}, \boldsymbol{\tau}^{\{i\}}) = F(\boldsymbol{x}) + \epsilon_i(\boldsymbol{x}), \quad i = 1, \ldots, c \end{cases} \tag{2}$$

and

$$S(\boldsymbol{x}, \boldsymbol{\tau}^{\mathcal{I}}) = F(\boldsymbol{x}) + \sum_{i \in \mathcal{I}} \epsilon_i(\boldsymbol{x}), \quad |\mathcal{I}| \geq 2 \tag{3}$$

[3]the learned function depends on the initialization and the mini-batch ordering.

for some (unknown) deterministic functions $F(\boldsymbol{x}), \epsilon_1(\boldsymbol{x}), \ldots, \epsilon_c(\boldsymbol{x})$. We prove two theorems in supplementary material[4] showing that a LSI architecture with linear submodules $\boldsymbol{d}$ and $\boldsymbol{D}$ exhibits super-generalization in the following sense: if such a LSI architecture $NN(\boldsymbol{x}, \boldsymbol{\tau})$ can learn from empirical data to make "good" predictions on data triplets $(\boldsymbol{x}, \boldsymbol{\tau}, y)$ 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" [15], 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 $\boldsymbol{x}$ = 153), and 186 power lines (dim $\boldsymbol{y}$ = 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: $\tau^{(i)} \in \mathcal{T}^{Source}$). We sampled 50000 different inputs $\boldsymbol{x}$ in the reference topology ($\tau^{\emptyset}$). For each $\boldsymbol{\tau}^{(i)}$, we sampled 1000 inputs $\boldsymbol{x}$ among these possible changes. We used the Hades2 software[5] to compute the flows $\boldsymbol{y}$ in all cases. This resulted in a training set of 150 000 rows (each row being one triplet $(\boldsymbol{x}, \boldsymbol{\tau}^{(i)}, \boldsymbol{y})$). We created an independent test set of the same size in a similar manner, keeping the same $\tau^{(i)} \in \mathcal{T}^{Source}$. We proceeded differently for the Target dataset. We sampled 1500 (Target domains: $\boldsymbol{\tau}^{(ij)} \in \mathcal{T}^{Target}$) among the 4950 possible binary actions $\boldsymbol{\tau}^{(ij)} = \tau^{(i)} \vee \tau^{(j)}$, $\tau^{(i)}$ and $\tau^{(j)} \in \mathcal{T}^{train}$. Then, for each of these 1500 $\boldsymbol{\tau}^{(ij)}$, we sampled 100 inputs $\boldsymbol{x}$ (with the same distribution as the one used for the training and regular test set). We used the same physical simulator to compute the $\boldsymbol{y}$ from the $\boldsymbol{x}$ and the $\boldsymbol{\tau}$. The super-generalization set counts then 150 000 rows, corresponding to 150 000 different triplets $(\boldsymbol{x}, \boldsymbol{\tau}^{(ij)}, \boldsymbol{y})$.

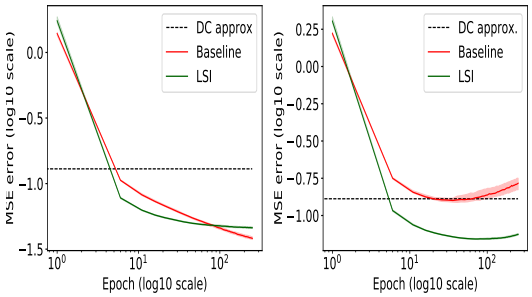
We compared the proposed LSI method with two benchmarks: the DC approximation, a standard baseline in power systems, which is a linearization of the AC (Alternative Current) non-linear powerflow equations, and the baseline neural network architecture (Figure 2) in which $\boldsymbol{\tau}$ is simply 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

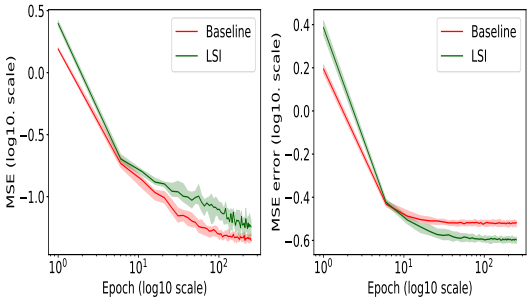
[4]For simplicity, we assumed $l = 1$ and thus all y's are scalar, but the generalization to the case with any l is straightforward. Consistent with the power system application setting, we assumed that data are collected without noise, but our results could be extended to noisy cases.

[5]Freeware available at `http://www.rte.itesla-pst.org/`.



(a) Regular generalization. (b) Super-generalization.

Figure 4: **Synthetic data on a 118 node grid.** We show the MSE error in Amperes on a log scale as a function of training epochs. Neural networks are trained with 15000 injections, for the reference topology $\boldsymbol{\tau}^{\emptyset}$ and unary changes $\boldsymbol{\tau}^{(i)}$. (a) **Regular generalization.** Test injections for **unary changes** $\boldsymbol{\tau}^{(i)}$. (b) **Super-generalization.** Test injections for **binary changes** $\boldsymbol{\tau}^{(ij)}$. Error bars represents the [20%, 80%] intervals, computed on 30 independently trained model.



(a) Regular generalization. (b) Super-generalization.

Figure 5: **Real data from the ultra high voltage power grid.** We show the MSE error in Amperes on a log scale as a function of training epochs. The neural network in both cases is **trained from data until May 2017** with real data. (a) **Regular generalization.** Test set made of randomly sampled data in the same time period as training data. (b) **Super-generalization.** Test set made of the months of June and July 2017.

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

Figure 4b 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 $\boldsymbol{x}$ are injections (productions and consumptions - this time of dimension dim $\boldsymbol{x} = 368$) and the outputs $\boldsymbol{y}$ are flows (with dim$\boldsymbol{y} = 387$). In this study, $\boldsymbol{x}$ and $\boldsymbol{y}$ come from real historical data.

One similarity between real and synthetic data is that $\boldsymbol{y}$ (the flows) have also been calculated from $\boldsymbol{x}$ 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 $\mathcal{T}^{Source}$ and $\mathcal{T}^{Target}$ domains: the emulated data is a controlled environment in which changes in $\boldsymbol{\tau}$ 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 $\mathcal{T}^{Source}$ and data from June and July 2017 for $\mathcal{T}^{Target}$. This favored changes in distribution of $\boldsymbol{\tau}$.

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 dim $\boldsymbol{\tau} = 387$, the number of power lines in the power grid. At the time of recording of our historical data, the physical simulator computed $\boldsymbol{y}$ as a function of $\boldsymbol{x}$ and the actual topology. Thus, from our perspective, the system S is a function of $\boldsymbol{x}$, $\boldsymbol{\tau}$, and latent factors (unrecorded

Table 2: **Comparison of datasets and experimental design.** We illustrate the super-generalization capabilities of the method. Training is performed simultaneously with data triplets $\{\boldsymbol{x}, \boldsymbol{\tau}, \boldsymbol{y}\}$ both from the source domain $\boldsymbol{\tau} = \boldsymbol{\tau}^{(\emptyset)}$ and the target domains $\boldsymbol{\tau} \in \mathcal{T}^{Source}$. Then the model is tested for super-generalization using data triplets $\{\boldsymbol{x}, \boldsymbol{\tau}, \boldsymbol{y}\}$ from the target domains with $\boldsymbol{\tau} \in \mathcal{T}^{Target}$.

Data type	Source domain $\boldsymbol{\tau} = \boldsymbol{\tau}^{(\emptyset)}$ lots of training data	Source domains $\boldsymbol{\tau} = \boldsymbol{\tau}^{(i)} \in \mathcal{T}^{Source}$ few training data for each $\boldsymbol{\tau}$	Super-generalization $\boldsymbol{\tau} \in \mathcal{T}^{Target}$ novel $\boldsymbol{\tau}$ not necessarily in training data
RTE historical data	$\approx$ 5 years of data with $\boldsymbol{\tau} \simeq \boldsymbol{\tau}^{(\emptyset)}$	Intermixed $\boldsymbol{\tau}$ different from $\boldsymbol{\tau}^{(\emptyset)}$, same 5 years	Next month : new $\boldsymbol{\tau} \simeq \boldsymbol{\tau}^{(\emptyset)}$ not in training data
Synthetic dataset (physical simulator)	5000 samples for $\boldsymbol{\tau} \equiv \boldsymbol{\tau}^{(\emptyset)}$	100 samples per $\boldsymbol{\tau} = \boldsymbol{\tau}^{(i)}$	100 samples per $\boldsymbol{\tau} = \boldsymbol{\tau}^{(ij)}$
Toy examples (artificial data)	1000 generated for $\boldsymbol{\tau} \equiv \boldsymbol{\tau}^{(\emptyset)}$	1 or 100 sample per $\boldsymbol{\tau} = [1, 0]$ or $\boldsymbol{\tau} = [0, 1]$	1000 samples testing super-generalization to $\boldsymbol{\tau} = [1, 1]$

dispatcher interventions). This unfortunate loss of information on exact grid topology interventions makes it impossible for us to compare our method to the DC approximation: computing this approximation requires a full description of the topology.

Figure 5 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 5b), 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 $(\boldsymbol{x}, \boldsymbol{\tau}, \boldsymbol{y})$, for which $\boldsymbol{\tau} \in \mathcal{T}^{Source}$ belong to source domains. We demonstrated that LSI generalizes not only by approximating well $\boldsymbol{y}$ for new values of $\boldsymbol{x}$ when $\boldsymbol{\tau} \in \mathcal{T}^{Source}$, but also when $\boldsymbol{\tau} \in \mathcal{T}^{Target}$ (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 case118 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 ≈ 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

[1] O Alsac and B Stott. Optimal load flow with steady-state security. *IEEE transactions on power apparatus and systems*, (3):745–751, 1974.

[2] Y. Bengio and et al. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432*, 2013.

[3] Rich Caruana. Learning many related tasks at the same time with backpropagation. In *Advances in neural information processing systems*, pages 657–664, 1995.

[4] Benjamin Donnot, Isabelle Guyon, Marc @bullet, Antoine Marot, and Patrick Panciatici. Fast power system security analysis with guided dropout. April 2018.

[5] Benjamin Donnot, Isabelle Guyon, Marc Schoenauer, Antoine Marot, and Patrick Panciatici. Anticipating contingengies in power grids using fast neural net screening. In *IEEE WCCI 2018*, Rio de Janeiro, Brazil, July 2018.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[7] T. Hossen, S. J. Plathottam, R. K. Angamuthu, P. Ranganathan, and H. Salehfar. Short-term load forecasting using deep neural networks (dnn). In *2017 North American Power Symposium (NAPS)*, pages 1–6, Sept 2017.

[8] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

[9] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[10] T.T. Nguyen. Neural network load-flow. *IEE Proceedings - Generation, Transmission and Distribution*, 142:51–58(7), January 1995.

[11] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knoweledge and Data Engineering*, 22(10):1345–1359, October 2010.

[12] N. Shazeer and et al. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv:1701.06538*, 2017.

[13] Richard Socher, Milind Ganjoo, Christopher D. Manning, and Andrew Y. Ng. Zero-shot learning through cross-modal transfer. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, pages 935–943, USA, 2013. Curran Associates Inc.

[14] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[15] R. D. Zimmerman and et al. Matpower. *IEEE Trans. on Power Systems*, pages 12–19, 2011.

[16] Minhui Zou, Yang Shi, Chengliang Wang, Fangyu Li, WenZhan Song, and Yu Wang. Potrojan: powerful neural-level trojan designs in deep learning models. *arXiv preprint arXiv:1802.03043*, 2018.