



**HAL**  
open science

# LEAP Nets for System Identification and Application to Power Systems

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

► **To cite this version:**

Balthazar Donon, Benjamin Donnot, Isabelle Guyon, Zhengying Liu, Antoine Marot, et al.. LEAP Nets for System Identification and Application to Power Systems. *Neurocomputing*, 2020, 416, pp.316-327. 10.1016/j.neucom.2019.12.135 . hal-02422708

**HAL Id: hal-02422708**

<https://inria.hal.science/hal-02422708v1>

Submitted on 23 Mar 2024

**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.



Distributed under a Creative Commons Attribution 4.0 International License

# LEAP Nets for System Identification and Application to Power Systems

B. Donon<sup>††1</sup>, B. Donnot<sup>†‡</sup>, I. Guyon<sup>•‡</sup>, Z. Liu<sup>‡</sup>,  
A. Marot<sup>†</sup>, P. Panciatici<sup>†</sup>, M. Schoenauer<sup>‡</sup>

• ChaLearn, USA. ‡ UPSud/Inria, U. Paris-Saclay, France. † RTE France

---

## Abstract

Using neural network modeling, we address the problem *system identification* for continuous multivariate systems, whose structures vary around an operating point. Structural changes in the system are of combinatorial nature, and some of them may be very rare; they may be *actionable* for the purpose of controlling the system. Although our ultimate goal is both *system identification and control*, we only address the problem of identification in this paper. We propose and study a novel neural network architecture called LEAP net, for Latent Encoding of Atypical Perturbation. Our method maps system structure changes to neural net structure changes, using structural actionable variables. We demonstrate empirically that LEAP nets can be trained with a natural observational distribution, very concentrated around a “reference” operating point of the system, and yet generalize to rare (or unseen) structural changes. We validate the generalization properties of LEAP nets theoretically in particular cases. We apply our technique to power transmission grids, in which high voltage lines are disconnected and re-connected with one-another from time to time, either accidentally or willfully. We discuss extensions of our approach to actionable variables, which are continuous (instead of discrete, in the case of our application) and make connections between our problem setting, transfer learning, causal inference, and reinforcement learning.

*Keywords:* System Identification, Latent Space, Residual Networks, LEAP Net, Power Systems

---

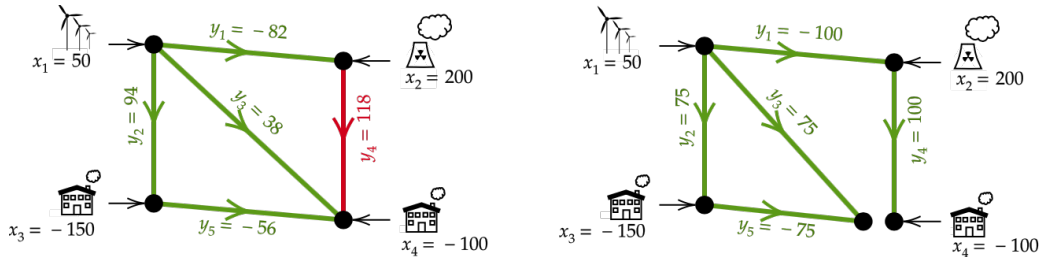


Figure 1: Electricity is transported from production nodes (top) to consumption nodes (bottom), through lines (green and red edges) connected at substations (black circles), forming a transmission *grid* of a given topology  $\tau$ . Injections  $\mathbf{x} = (x_1, x_2, x_3, x_4)$  (production or consumption) add up to zero. Grid operators (a.k.a. *dispatchers*) should maintain current flows  $\mathbf{y} = S(\mathbf{x}; \tau)$  below thermal limits. Left: Line  $\mathbf{y}_4$  goes over its thermal limit 100. Right: A change in topology (splitting of 10.1007/3-540-45356-3 bottom right node) brings  $\mathbf{y}_4$  back to its thermal limit.

## 1. Introduction

Neural networks have been used for identification and control since the 1990s[16]. They present the distinctive advantage that gradients can be back-propagated both through the model of the system and through the controller, if both are neural networks. In this paper, we tackle a problem of **power system identification**, with the long term goal of improving the automatic control of such systems (although the control part is not addressed in this paper). Our intermediate goal, tackled in this paper, is to develop a neural network model of the power grid, which allows us to replace *computationally costly physical simulators* in the computation of power flows, following other pioneering work [18, 13, 7, 6]. Keeping our “control objective” in mind, we adopt an approach allowing us to simulate the effect of planned coordinated actions on the grid topology (as opposed to accidental suffered changes). In particular, we can simulate the effect of changes that rarely or never occur in observational data.

Power systems have the objective of transporting electricity over long distances from production sites (*e.g.* power plants, windmills, or solar cells) to consumers, using a power transmission grid. The grid is made of power lines that are interconnected at substations. At all times, the grid must be maintained in “security”, namely no lines should operate over its “thermal limit”. Indeed, if a line overheats, it may be damaged, melt and/or cause fire or break. To prevent this to happen, lines are automatically put out of service if their thermal limit is exceeded for some period of time. Unfortunately,

24 when a line goes out of service, this results in a surcharge of lines remaining  
25 in service, increasing their risk of overheating. This may result in a cascading  
26 failure (also called black-out). **Our main charter is thus to prevent**  
27 **black-outs.**

28 Although rare, situations of crisis can occur when several line suddenly  
29 break *e.g.* due to a thunderstorm, or if fluctuations of power production  
30 or consumption are unpredictable. Highly skilled engineers (dispatchers)  
31 constantly monitor the grid to make sure it operates in security. However,  
32 operating the grid is becoming increasingly complex because of the advent  
33 of less predictable renewable energies, the globalization of energy markets,  
34 growth in consumption and concurrent limitations on new line construction.  
35 Therefore, it is becoming urgent to optimize more tightly the grid operation.  
36 This will require computationally efficient simulators of power flow to evalu-  
37 ate the effect of candidate actions. Even in the absence of fully automatic  
38 control, an effective neural network emulator of a power system would be  
39 very useful in routine operations: this would allow dispatchers to quickly run  
40 many simulations to evaluate the outcome of various preventive or curative  
41 actions. Although many types of actions can be considered, in this paper,  
42 we limit ourselves to one particular type of action: **reconfiguration of the**  
43 **grid by splitting or merging nodes at substations.**

44 Figure 1 illustrates the problem setting on a small example. A line going  
45 over its thermal limit will be put out of service. Hence, the grid must be  
46 reconfigured quickly to re-balance current flows before that happens. Should  
47 this happen, more lines might go over their thermal limit, which could result  
48 in a cascading failure (black-out). The figure shows a solution consisting of  
49 a node splitting in a substation. We call that a grid topology change. The  
50 space of possible grid topologies grows exponentially with the number of  
51 substations (nodes in the grid). For example, the French high-voltage trans-  
52 mission grid includes  $N \approx 6200$  substations, with more than a dozen possible  
53 configurations per substation and thus  $\gtrsim 10^N$  possible grid topologies. Even  
54 if only a small number of those are achievable, the search space is still humon-  
55 gous. Our challenge is to devise a neural architecture and a training method  
56 such that, using a training dataset illustrating only a few grid topologies,  
57 **power flows can still be accurately predicted for topologies never**  
58 **seen before.**

59 In this paper, the approach that we present uses a neural network to  
60 emulate the power grid  $\mathbf{y} = S(\mathbf{x}; \boldsymbol{\tau})$  in which the inputs  $\mathbf{x}$  are so-called  
61 “injections” (productions and consumptions) and the outputs  $\mathbf{y}$  are the power

62 flows on all the connecting lines of the grid. The system is parameterized  
 63 by structural parameter vector  $\tau$  encoding various grid topologies, which  
 64 includes our **actionable parameters**. We use  $\tau$  to encode changes in our  
 65 neural network architecture.

66 The rest of the paper is organized as follows. In Section 2 we motivate  
 67 our neural network architectural choices. In Section 3 we develop a math-  
 68 ematical formulation of our proposed approach and perform a theoretical  
 69 analysis. In Section 4 we experimentally validate our method on both syn-  
 70 thetic and real world data in our application domain. Finally, in Section 5  
 71 we propose and discuss various extensions of LEAP nets and make connec-  
 72 tions with other settings, including transfer learning, causal modeling, and  
 73 reinforcement learning.

74 This paper is an extended version of our work [5] presented at ESANN  
 75 2019 in Bruges, Belgium. The additional contributions of this version in-  
 76 clude: a new framework in terms of system identification, a toy example  
 77 to better illustrate the problem, the corresponding toy experiment and a  
 78 comparison with 3 baseline methods, an enhanced mathematical formalism  
 79 and two theorems, new figures illustrating of the behavior of LEAP nets,  
 80 new experiments on a standard benchmark power grid, and a new discussion  
 81 section.

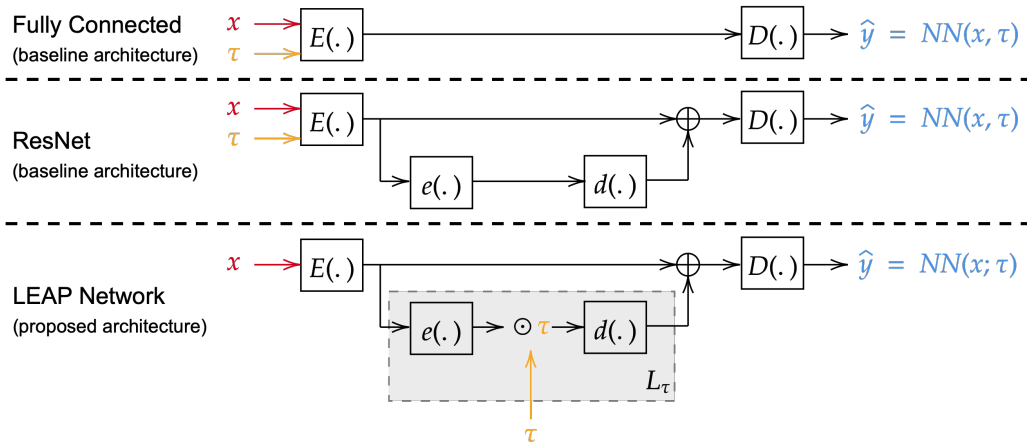


Figure 2: **Baselines and LEAP architectures:** Top: Fully Connected architecture with both  $x$  and  $\tau$  as inputs. Middle: ResNet [12] architecture, with both  $x$  and  $\tau$  as input. Bottom: Proposed LEAP net:  $\tau$  intervenes in the latent embedding space. The effect is to make a “leap” in latent space. See Section 3 for details.

## 82 2. Architectural design

83 In this section, we motivate the design of the LEAP net architecture  
84 (our proposed approach) with considerations of training data imbalance, and  
85 illustrate its benefits on a toy example.

### 86 2.1. Problem setting: system identification

87 We recall that **system identification** is a problem quite distinct to that  
88 of predictive modeling. Much of *machine learning* addresses the problem  
89 of predictive modeling in which both training and test data samples are  
90 drawn from the **same** (unknown) distribution, usually independently and  
91 identically (i.i.d. data). Under these conditions, if a predictive model makes  
92 poor predictions in depleted regions of input space, it affects only weakly its  
93 generalization performance. In contrast, we are here concerned with a sys-  
94 tem identification task, in which we want to be able to **make predictions**  
95 **uniformly well for all possible values of actionable parameters** (or  
96 at least all possible permitted values), to minimize the risk of making ill-  
97 informed decision, which may have catastrophic effects. The problem is par-  
98 ticularly acute when the natural observational distribution of training data,  
99 used to fit our model (our neural network) is very concentrated around an  
100 operating point. This is the case in our power system application: most of  
101 the historical data are recorded for a so-called “reference topology”  $\tau^\theta$  (or in  
102 its close neighborhood), for a large variety of injections  $\mathbf{x}$ . In contrast, very  
103 few examples of other topologies (each for few injection values) are avail-  
104 able for training. This calls for an architecture, which learns principally the  
105 mapping  $\mathbf{y} = S(\mathbf{x}; \tau^\theta)$ , and then learns corrections accounting for deviations  
106 from  $\tau^\theta$ . This is why we have been inspired by the ResNet [11] architecture,  
107 see Figure 2. The main branch, consisting of encoder  $E(\cdot)$  and decoder  $D(\cdot)$ ,  
108 principally learns  $\mathbf{y} = S(\mathbf{x}; \tau^\theta)$ ; meanwhile, the insert, consisting of encoder  
109  $e(\cdot)$  and decoder  $d(\cdot)$ , learns deviations from  $\tau^\theta$ . We will come back to this  
110 in more details in the Section 3.

111 In this paper, **actionable parameters**, encoded by the discrete struc-  
112 tural parameters  $\tau$ , and other *non actionable* variables, encoded as contin-  
113 uous variables  $\mathbf{x}$ , play different roles. We are mostly interested in keeping  
114 the same distribution of  $\mathbf{x}$  between training and test data<sup>2</sup>, but re-balancing

---

<sup>2</sup>although in some cases of practical interest, the distribution of  $\mathbf{x}$  may also drift.

115 the distribution of  $\tau$  in test data, as previously motivated. Changes in dis-  
 116 tribution between training and test data may call for an analysis in terms  
 117 of *transfer learning* (e.g. [19]) or even *causal modeling* (e.g. [21]), since we  
 118 are talking of “actions” deliberately performed by external operators at test  
 119 time, hence of a “manipulated distribution”. However, for simplicity, we  
 120 prefer casting our problem in the framework of machine learning from *i.i.d.*  
 121 data. We postpone making connections with transfer learning and causal  
 122 modeling to the discussion section (Section 5). For the purpose of this pa-  
 123 per, as stated before, in order to be deployed into production as part of a  
 124 computer-assisted human decision system, our neural network model must  
 125 **make predictions uniformly well on a designated distribution sup-**  
 126 **port of  $\tau$** . Assuming that this support is included in the support of the  
 127 observational distribution, we can re-cast the problem as a regular machine  
 128 learning problem, with *i.i.d.* training and test data drawn for the same obser-  
 129 vational distribution. To that end, at test time we use an objective function  
 130 that re-weighs the samples to attain the desired importance for each action-  
 131 able parameter  $\tau$ . Equivalently, we can re-balance the test distribution by  
 132 importance sampling (e.g. [17]) and use an un-weighted objective function.  
 133 In what follows, we will interchangeably adopt either viewpoint, depending  
 134 on which one is conceptually or practically more convenient.

135 There are other obvious departures from the *i.i.d.* framework in the set-  
 136 ting of power systems. Triplets  $\{\mathbf{x}, \tau, \mathbf{y}\}$  are not independent of one another,  
 137 they are organized in a time series. We can assume that  $\mathbf{y}$  is independent of  
 138 the rest of the data samples given  $\mathbf{x}$  and  $\tau$ , reflecting the fact that the sys-  
 139 tem, *i.e.* the power grid and its physical laws, remains unchanged over time.  
 140 What changes over time though is the distribution of  $\mathbf{x}$  (and that of  $\tau$ ), due to  
 141 daily, weekly, or seasonal variations, various weather imponderables, changes  
 142 in technology, and other disruptions. Thus the *i.i.d.* assumption poorly char-  
 143 acterizes the way in which power networks are operated. But, even though,  
 144 in many ways, there is a time component to our problem, we chose here to  
 145 ignore it, and treat the problem more simply. We assume a double random  
 146 process in which a month is first drawn at random (*i.i.d.* from a distribution  
 147 of months that does not change from year to year) and within that month a  
 148 “load flow”  $\{\mathbf{x}, \tau, \mathbf{y}\}$  is drawn at random (*i.i.d.* from a distribution of load  
 149 flows that occur within that month). So, everything considered, we bring  
 150 ourselves back to an *i.i.d.* setting.

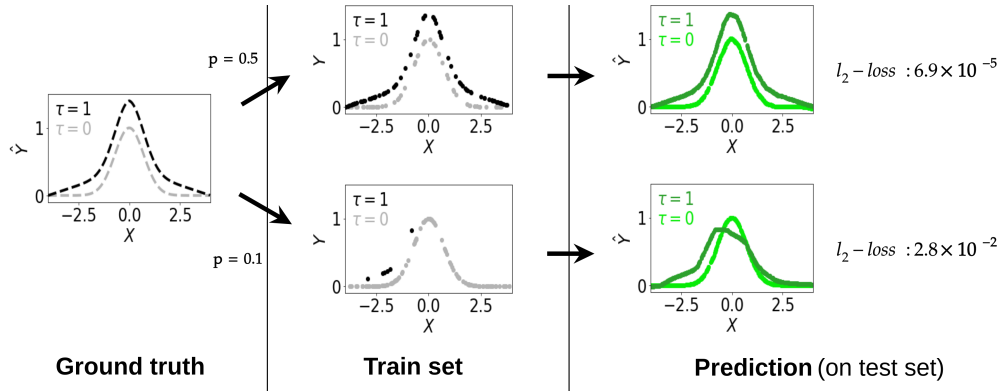


Figure 3: **Impact of data distribution imbalance on prediction accuracy (toy example)**. Left: system to be identified, see Equation 1. Top: Balanced training data *w.r.t*  $\tau$  ( $p = 0.5$ ). A standard fully connected neural network has no problem learning, reaching  $l_2$ -loss =  $6.9 \times 10^{-5}$ . Bottom: Imbalanced training data,  $p = 0.1$ . A fully connected neural net cannot learn well the impact of  $\tau$ :  $l_2$ -loss =  $2.8 \times 10^{-2}$ .

151 *2.2. A toy example*

In this section, we give a first motivational toy example to show how data imbalance may hamper system identification. In this example, the system to be identified, instead of being a complex power grid, is just defined by:

$$y = S(x; \tau) = e^{-x^2} - 0.1 \times \tau \times (|x| - 4) \quad (1)$$

152 Here,  $x$ ,  $\tau$ , and  $y$  are just scalar values, but they play the same role  
 153 as described before. Input values  $x$  are continuous. They are drawn for a  
 154 uniform distribution  $x \sim U(-4, 4)$ . Structural parameters  $\tau$  are discrete and  
 155 have binary values 0 or 1 drawn from a Bernoulli distribution  $\tau \sim B(p)$  with  
 156 probability  $p$  of drawing a 1 and  $(1 - p)$  of drawing a 0.

157 It is equivalent for our analysis to assume either that we draw training  
 158 and test data *i.i.d.* from the same distribution, then re-weigh the distribution  
 159 of  $\tau$  at test time in our objective function to make it uniform, or to assume  
 160 that test data are re-balanced, *i.e* that  $\tau \sim B(0.5)$  in test data. We adopt the  
 161 latter angle in Figure 3 to illustrate how a distributional imbalance due to a  
 162 small value of  $p$  affects the quality of predictions for a vanilla fully connected  
 163 neural network. Here  $\tau = \tau^0 = 0$  can be thought of as the indicator of the  
 164 “reference” situation most frequently encountered, while  $\tau = 1$  is a less  
 165 typical situation, hence getting very few training samples, if  $p$  is small.

166 As a preview of how our LEAP net solution addresses this problem, we  
 167 compare four neural networks with associated training strategies:



- 168 • *LEAP Net*, our proposed architecture as detailed in section 3 with  $\tau$   
169 injected in the latent space.<sup>3</sup>
- 170 • *Fully Connected* Multi-Layer Perceptron (MLP) with  $\tau$  as an input.<sup>4</sup>
- 171 • *ResNet* where the leap block is replaced by a standard residual block,  
172 closer in morphology to LEAP Net, but with  $\tau$  still as an input.<sup>5</sup>
- 173 • *ResNet + Importance Sampling*. Same ResNet architecture, but during  
174 training the data points are weighted to compensate for the imbalance  
175 with regards to  $\tau$  [3]. The idea is to give samples with  $\tau = 0$  a weight of  
176  $\frac{1}{1-p}$ , and samples with  $\tau = 1$  a weight of  $\frac{1}{p}$ , so that the same emphasis  
177 is put on the two cases.<sup>6</sup>

178 For each of those four modeling strategies, and for each selected value of  
179  $p$ , we trained 100 model instances using the Adam optimizer, with default  
180 parameters except for the learning rate that has been set to  $1.10^{-3}$  and  
181 minibatches of 32 points. All instances are trained for 2000 epochs, ensuring  
182 that all architectures have fully converged. The training of one model takes  
183 approximately 5 minutes on a single Nvidia Titan Xp GPU. In all cases, the  
184 training set counts  $N = 128$  data points drawn according to:

- 185 •  $x \sim \mathcal{U}(-4, 4)$ : the uniform distribution between -4 and 4
- 186 •  $\tau \sim \mathcal{B}(p)$ : the Bernoulli law with parameter  $p$ :  $\mathbb{P}(\tau = 1) = p$

187 In Figure 4, we observe that the more imbalanced the distribution (smaller  
188 values of  $p$ ), the larger the loss function. A too large dissimilarity between

---

<sup>3</sup>The encoder  $E(\cdot)$  has 5 hidden layers of 10 neurons,  $e(\cdot)$  and  $d(\cdot)$  are single layer neural nets, and the decoder  $D(\cdot)$  is composed of 1 hidden layer of 10 neurons. All layers in  $E$  and  $D$  use "ReLU" non-linearity. Hyperparameters tuning made us chose different values for the depths of  $E$  and  $D$ , which in our interpretation reflects the fact that the modification caused by  $\tau$  is "closer to the output  $y$  than to the input  $x$ ". This interpretation is based on our current understanding of what the LEAP net does, but should be further substantiated in future work.

<sup>4</sup>The MLP architecture consists of 6 hidden layers each having 10 neurons and with a "ReLU" activation function. It has as many hidden layers as the LEAP architecture.

<sup>5</sup>We note that this architecture has slightly more trainable parameters than the LEAP Net architecture, which is to its advantage. The same parameters as for the LEAP are used.

<sup>6</sup>The same parameters as for the LEAP are used.

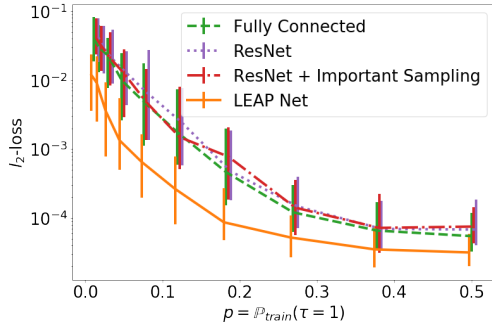


Figure 4: **Toy problem experiment. Evolution of the  $l_2$ -loss with respect to training data imbalance.** We vary parameter  $p$ , which is the probability of  $\tau = 1$  in the training set (*i.e.* training data imbalance). For each of the 4 types of models and for each value of  $p$ , we trained 100 model instances. We show the median  $l_2$ -loss and the 20% – 80% percentile error bars (horizontally shifted to improve readability). Our proposed LEAP net is consistently more resilient to training data imbalance compared to the other models.

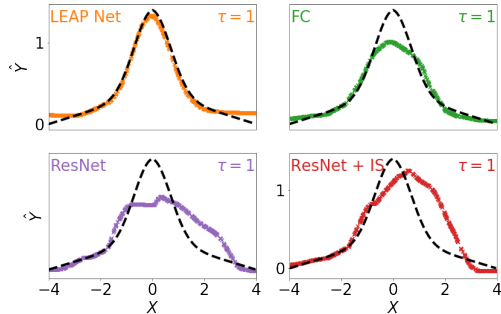


Figure 5: **Toy problem experiment. Predictions for  $\tau = 1$  when  $p = 0.1$ .** For each of the four classes of models, we trained one model instance, in the setting of Figure 3, bottom. The ground truth for  $\tau = 1$  is shown as a black dashed line and the corresponding prediction results on re-weighted test data as colored dots. Our proposed LEAP net qualitatively outperforms other methods, preserving better the shape of the output.

189 training and test distributions is detrimental to the ability of our neural  
 190 network model to learn the actual impact of parameter  $\tau$ . Figure 5 shows  
 191 typical predictions for one instance of each of the 4 types of models  
 192  $p = 0.1$ . The LEAP makes better predictions compared to other models.  
 193 Despite being trained with little training data, the LEAP net is able to  
 194 generalize from the “reference” case  $\tau = 0$  for which it has many values of  $x$   
 195 to the other case  $\tau = 1$  for which it has only a few samples. The shape of  
 196 the curves are not preserved in all the other tested models.

### 197 3. Mathematical formalism

198 In this section, we introduce a mathematical framework to analyze in  
 199 more details the proposed LEAP architecture represented in Figure 2.

#### 200 3.1. Notations and definitions

201 Our objective is to approximate a function  $\mathbf{y} = S(\mathbf{x}, \boldsymbol{\tau})$  that maps input  
 202 data  $\mathbf{x}$  (*e.g.* power production and consumption) to output data  $\mathbf{y}$  (*e.g.* power  
 203 flows), parameterized by a discrete “grid topology vector”  $\boldsymbol{\tau}$ , taking values in

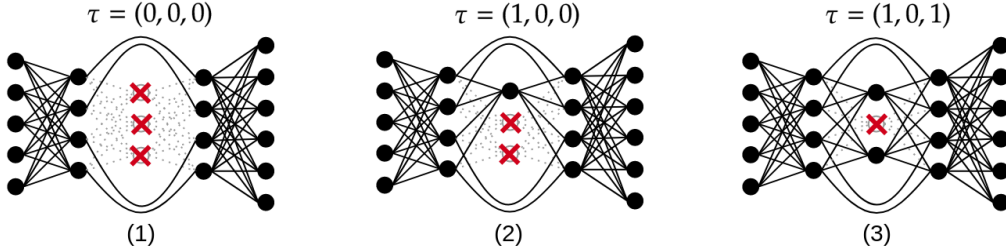


Figure 6: **Example of encoding of  $\tau$  in a LEAP net architecture.** This figure shows another view of the LEAP architecture of Figure 2. Black dots are neurons. The encoder and decoder  $E$  and  $D$  are represented by the densely connected layers on either side of the network. The direct connection between  $E$  and  $D$  is represented by curved lines bypassing the central hidden layer. The interventions in latent space (hidden layer), performed by  $\tau$ , are represented by red crosses. We assume that *unary* topological changes in our system are one-hot encoded (other choices are possible). Crossed neurons are deactivated. **(1)** Architecture for  $\tau = \tau^\emptyset = (0, 0, 0)$ , the reference configuration. **(2)** Architecture for  $\tau = (1, 0, 0)$  creating a path through the central hidden layer. **(3)** Architecture for  $\tau = (1, 0, 1)$  combining two paths and therefore super-imposing effects.

204 a parameter space of potentially actionable variables (all possible power-grid  
 205 topologies *e.g.* line interconnections). Triplets  $\{\mathbf{x}, \boldsymbol{\tau}, \mathbf{y}\}$  are drawn *i.i.d.* ac-  
 206 cording to an unknown probability distribution. In our application setting,  
 207  $\mathbf{x}$  is drawn randomly, but  $S(\cdot, \cdot)$  is a deterministic function implementing  
 208 Kirchhoff’s circuit laws, calculated by a physical solver that we wish to ap-  
 209 proximate.

210 We call **simple generalization** the capability of a neural net  $\hat{\mathbf{y}} =$   
 211  $NN(\mathbf{x}, \boldsymbol{\tau})$  to approximate  $\mathbf{y} = S(\mathbf{x}, \boldsymbol{\tau})$  for test inputs  $\mathbf{x}$  not pertaining to the  
 212 training set, when  $\boldsymbol{\tau}$  values are drawn *i.i.d.* from a distribution that remains  
 213 the same in training and test data (this includes the case of a fixed  $\boldsymbol{\tau}$ ). We  
 214 call **super-generalization** the capability of a neural net  $\hat{\mathbf{y}} = NN(\mathbf{x}, \boldsymbol{\tau})$  to  
 215 approximate  $\mathbf{y} = S(\mathbf{x}, \boldsymbol{\tau})$  when the natural observational distribution of  $\boldsymbol{\tau}$  is  
 216 very imbalanced, with a peak around a reference topology  $\tau^\emptyset$ . We re-weigh  
 217 samples in our objective function to re-balance test data to measure the  
 218 correctness of system identification in regions of  $\boldsymbol{\tau}$  space that are allowable  
 219 actions but are under-represented in the natural distribution. Equivalently,  
 220 the test set distribution is made uniform on the support of allowable actions.

221 The structural parameter vector  $\boldsymbol{\tau}$  represents topological changes in the  
 222 electricity grid, in our application setting. This prompted us to make partic-  
 223 ular choices for its encoding, although other choices are possible. One

224 particularity of our application related to power systems is that we have one  
 225 *primary* “reference” configuration (corresponding to a reference grid topol-  
 226 ogy  $\boldsymbol{\tau}^\theta = (0, 0, 0, \dots)$ ), around which *small* variations are made. This is a  
 227 generic scenario in industry for systems that operate around nominal condi-  
 228 tions, thus we anticipate that our method could be extended to other similar  
 229 situations. In our application scenario, we can easily get a lot of training data  
 230 in the reference topology (corresponding to the typical way in which the grid  
 231 is operated). We have comparably very little data available for training from  
 232 other topologies corresponding to unary changes  $\boldsymbol{\tau}^i = (0, 0, 1, \dots)$  (a sin-  
 233 gle 1 at position  $i$ ). Finally, we have extremely scarce data or no data at  
 234 all available for training from double changes  $\boldsymbol{\tau}^{ij}$ , or higher order changes.  
 235 This motivates our architectural design and how we encode  $\boldsymbol{\tau}$  in the neural  
 236 network architecture (see, Figure 6).

### 237 3.2. Analysis and interpretation of the architecture

238 Our proposed Latent Encoding of Atypical Perturbations network, or  
 239 LEAP net (Figure 2), is composed of three parts: An Encoder  $\mathbf{E}$ , learning  
 240 an embedding of the input data  $\mathbf{x}$ ; a Decoder  $\mathbf{D}$ , learning how to perform  
 241 the required task within this latent representation; and a Latent module  $\mathbf{L}_\tau$ ,  
 242 placed between the  $\mathbf{E}$  and  $\mathbf{D}$  where  $\boldsymbol{\tau}$  intervenes. The overall architecture  
 243 is given by:

$$\mathbf{L}_\tau : \mathbf{h} \rightarrow \mathbf{d}(\mathbf{e}(\mathbf{h}) \odot \boldsymbol{\tau}) \quad (2)$$

$$\hat{\mathbf{y}} = \mathbf{D} \circ (\mathbf{I} + \mathbf{L}_\tau) \circ \mathbf{E}(\mathbf{x}) \quad (3)$$

244 where  $\mathbf{E}$  and  $\mathbf{e}$  (encoders) and  $\mathbf{D}$  and  $\mathbf{d}$  (decoders) are all differentiable func-  
 245 tions (typically implemented as artificial neural networks). The  $\odot$  operation  
 246 denotes the component-wise multiplication and  $\circ$  the function composition.  
 247 If the system is in the reference topology  $\boldsymbol{\tau}^\theta$ , predictions are made according  
 248 to  $\hat{\mathbf{y}} = \mathbf{D} \circ \mathbf{E}(\mathbf{x})$ . A typical way in which we train LEAP nets is to use a  
 249 lot of training data in the reference topology  $\boldsymbol{\tau}^\theta$ , very few examples for each  
 250 of the unary changes  $\boldsymbol{\tau}^i$ , and we expect the network to generalize to target  
 251 domains corresponding to double  $\boldsymbol{\tau}^{ij}$  or higher level changes<sup>7</sup>.

252 Our LEAP net architecture proceeds in the following way : we first embed  
 253  $\mathbf{x}$  in a latent space by applying  $\mathbf{E}$ . Then, based on  $\boldsymbol{\tau}$  and the location of  $\mathbf{E}(\mathbf{x})$

---

<sup>7</sup> $\boldsymbol{\tau}^{ij}$  is defined as a vector that is 1 in both its  $i$ -th and  $j$ -th elements, and 0 everywhere else

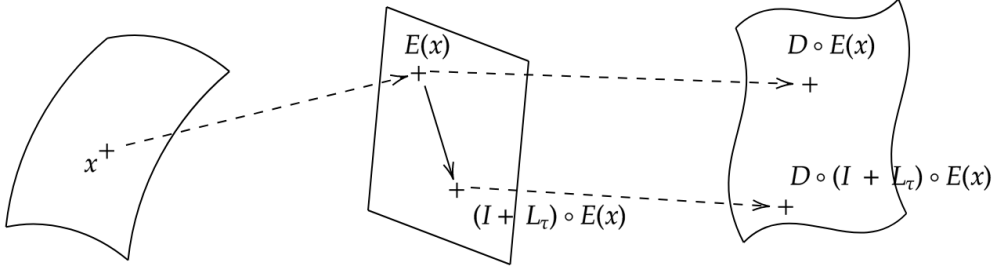


Figure 7: **Leap in latent space changes prediction  $\hat{y}$ .** We schematically illustrate how input  $\mathbf{x}$  is transformed by encoder  $\mathbf{E}$  into  $\mathbf{E}(\mathbf{x})$ , then “moves” in latent space by application of vector  $\mathbf{L}_\tau \circ \mathbf{E}(\mathbf{x})$ , depending both on  $\mathbf{E}(\mathbf{x})$  (*i.e.* the initial location in the latent space), and  $\tau \neq \tau^\theta$ . It is then decoded by  $\mathbf{D}$  into  $\hat{\mathbf{y}} = \mathbf{D} \circ (\mathbf{I} + \mathbf{L}_\tau) \circ \mathbf{E}(\mathbf{x})$ . We also represent at the top the “path” that  $\mathbf{x}$  would have followed if  $\tau = \tau^\theta$  (*i.e.* in the reference configuration).

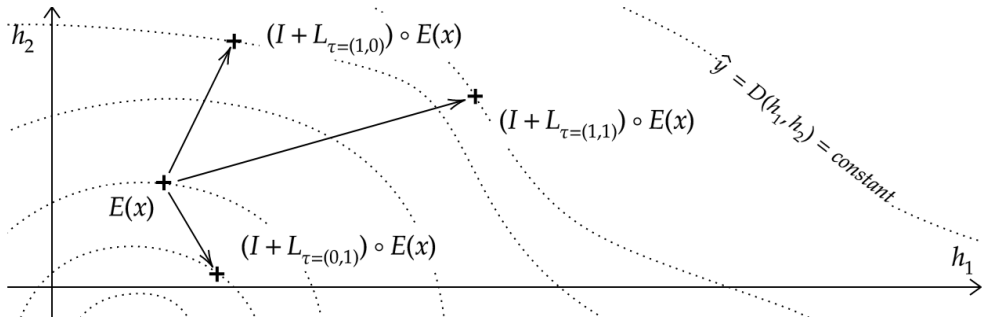


Figure 8: **Various leaps in latent space.** We schematically visualize latent space after having applied encoder  $\mathbf{E}$ . Here the output of  $\mathbf{E}$  is only two-dimensional:  $\mathbf{h} = (h_1, h_2)$ , and  $\tau$  also has only 2 components (its dimension is unrelated to that of  $\mathbf{h}$ ). The final output is one-dimensional. For a given  $\mathbf{x}$  we can visualize the 2D location of  $\mathbf{E}(\mathbf{x})$ . Within this latent space we can see the results of the latent leaps for  $\tau = (1, 0)$ ,  $(0, 1)$ , or  $(1, 1)$ . One should keep in mind that the latent leap for  $\tau = (1, 1)$  is not just the sum of the latent leap for  $\tau = (1, 0)$  and  $\tau = (0, 1)$ : it can be a **combination of conditions with non-linear effects**. Moreover, the direction and norm of each of those leaps depends on the initial location within the latent space  $\mathbf{E}(\mathbf{x})$ . The dashed lines are contour lines of constant output  $\hat{\mathbf{y}}$  defined as  $\{\mathbf{h} | \mathbf{D}(\mathbf{h}) = cste\}$

254 within the latent space, we compute the corresponding leap  $\mathbf{L}_\tau \circ \mathbf{E}(\mathbf{x})$ .<sup>8</sup> Then

<sup>8</sup>During our experiments, we observed that adding a bias in the Leap block was detrimental to its super generalisation ability. Our interpretation of this phenomenon is that by doing that, the LEAP block is never actually deactivated (*i.e.* multiplied by 0), which

255 we decode the signal by applying  $\mathbf{D}$ . Those latent leaps contain information  
256 about how much the system actually deviates from the reference state, and  
257 in which direction. Hence, in our architecture  $\mathbf{L}_\tau$  only needs to learn to  
258 modulate the system response around its nominal value, disregarding of the  
259 system complexity, encoded in  $\mathbf{E}(\cdot)$  and  $\mathbf{D}(\cdot)$ . This process is schematically  
260 illustrated in Figures 7 and 8.

261 While our architecture is inspired by both ResNet and Dropout, the un-  
262 derlying concepts are quite different:

- 263 • ResNet was developed to tackle problems of vanishing gradients in deep  
264 architectures, providing a shorter path between shallow and deep layers.  
265 In contrast, our use a single ResNet block aims at modeling a system  
266 changing around an operating point.
- 267 • Dropout is a regularization technique that randomly deactivates neu-  
268 rons during training in order to make the neural network more resilient.  
269 In contrast, our network alterations encode specific topological changes  
270 of the system being modeled.

271 In the experimental section, we compare our LEAP network to a ResNet ar-  
272 chitecture, in which topology changes are provided as an input rather than as  
273 a change in network structure, as a control experiment. We cannot compare  
274 with dropout because random disconnections would not implement specific  
275 changes in topology.

### 276 3.3. Theoretical analysis of super-generalization

277 In this section we formally prove the **super-generalization** capability  
278 of LEAP nets, when modeling systems with additive perturbation. It is  
279 important to note that LEAP nets are not limited to modeling additive  
280 perturbations. This simple theoretical analysis can be thought of as a “sanity  
281 check”. It is easy to construct cases in which LEAP nets can fail to super-  
282 generalize. In our experimental section we will show various empirical cases  
283 of super-generalization in our application setting of power system, not limited  
284 to additive perturbations.

---

causes gradient to “flow through it” for every training sample. The LEAP block is thus never specialized in adjusting the latent space conditionally on some variables. We think that our proposed LEAP net is efficient only when gradient is backpropagated in its LEAP block only for selected samples, allowing for a specialization.

285 A system with **additive perturbations** is defined as a system  $S(\mathbf{x}, \boldsymbol{\tau})$   
 286 that satisfies

$$\begin{cases} S(\mathbf{x}, \boldsymbol{\tau}^0) = F(\mathbf{x}) \\ S(\mathbf{x}, \boldsymbol{\tau}^i) = F(\mathbf{x}) + \epsilon_i(\mathbf{x}), \quad i = 1, \dots, c \end{cases} \quad (4)$$

287 and

$$S(\mathbf{x}, \boldsymbol{\tau}^{\mathcal{I}}) = F(\mathbf{x}) + \sum_{i \in \mathcal{I}} \epsilon_i(\mathbf{x}), \quad |\mathcal{I}| \geq 2 \quad (5)$$

288 for some (unknown) deterministic functions  $F(\mathbf{x}), \epsilon_1(\mathbf{x}), \dots, \epsilon_c(\mathbf{x})$ .

289 We begin with a theorem showing the super-generalization ability of  
 290 LEAP nets in the case where a trained model makes perfect predictions  
 291 on data coming from the same distribution as the training data. Then we  
 292 generalize this result to the noisy case with imperfect predictions. All proofs  
 293 are in appendix.

294 **Theorem 1. (Super-generalization)** *Let  $S(\mathbf{x}, \boldsymbol{\tau})$  be a system satisfying*  
 295 *Equations 4 and 5; and let  $NN(\mathbf{x}, \boldsymbol{\tau})$  be a LEAP net with linear decoders*  
 296  *$\mathbf{d}$  and  $\mathbf{D}$ . If  $NN(\mathbf{x}, \boldsymbol{\tau})$  is trained to make perfect predictions on **simple***  
 297 ***perturbations** (data triplets  $(\mathbf{x}, \boldsymbol{\tau}, \mathbf{y})$  coming from distribution defined by*  
 298 *Equation 4), then it will make perfect predictions on **combinations of per-***  
 299 ***turbations** (data coming from test distribution defined by Equation 5).*

The above theorem shows that, under some conditions, LEAP nets are indeed capable of performing *super-generalization*: making good predictions on complex structural parameters  $\boldsymbol{\tau}$  while it was only trained with unary cases  $\boldsymbol{\tau}^i$ . The fact that  $\mathbf{d}$  and  $\mathbf{D}$  are linear is essential for capturing the additivity of perturbations. For other types of perturbations such as the multiplicative case defined as follows

$$S(\mathbf{x}, \boldsymbol{\tau}^{\mathcal{I}}) = F(\mathbf{x}) \prod_{i \in \mathcal{I}} (1 + \epsilon_i(\mathbf{x})),$$

300 LEAP nets can achieve similar results if  $\mathbf{D}$  has, for example, an exponential-  
 301 like behavior (*i.e.* it transforms additions into multiplications).

302 Despite its limitations, the fact that Theorem 1 works for any unknown  
 303 functions  $\epsilon_i(\mathbf{x})$  makes the results very general. We can consider for example  
 304 linear perturbations with  $\epsilon_i(\mathbf{x}) = \mathbf{W}_i \mathbf{x}$ , constant additive perturbations with  
 305  $\epsilon_i(\mathbf{x}) = \boldsymbol{\alpha}_i$  or heteroskedastic perturbations with  $\epsilon_i(\mathbf{x}) = \boldsymbol{\alpha}_i G(\mathbf{z})$ , etc. And  
 306 all these types of perturbation are special cases of the above theorem.

307 Theorem 1 gives a strong theoretical guarantee, but one may wonder if  
 308 the condition of making *perfect* predictions can be too restrictive. So we also  
 309 investigated the slightly more general case of *imperfect* predictions, when the  
 310 LEAP net is trained only on unary changes.

311 To do this, we first need to introduce a notion of distance between func-  
 312 tions. As we used Mean Square Error (MSE) in our paper for the regression  
 313 problem, the distance we use will be defined in the same flavor. Let  $\mathcal{X} = \mathbb{R}^p$   
 314 be the support of  $\mathbf{x}$ ,  $\mathcal{Y} = \mathbb{R}^l$  be the support of  $\mathbf{y}$ . Let  $\mu$  be a probability  
 315 measure on  $\mathcal{X}$ , and  $f, g$  two functions from  $\mathcal{X}$  to  $\mathcal{Y}$ . We define the distance  
 316  $d_\mu(f, g)$  between  $f$  and  $g$  to be :

$$d_\mu^2(f, g) = \int_{\mathcal{X}} \|f(\mathbf{x}) - g(\mathbf{x})\|^2 d\mu(\mathbf{x}) \quad (6)$$

317 Here  $\|\cdot\|$  represents the  $\ell_2$ -norm on  $\mathbb{R}^l$ . Notice that this distance depends  
 318 on the probability measure  $\mu$ . If we put  $\mu = D(\mathcal{X})$  (the ground truth dis-  
 319 tribution of  $\mathbf{x}$ ) and write  $\mathbf{y} = g(\mathbf{x})$ , the right hand side becomes nothing  
 320 but the generalization error<sup>9</sup> of  $f$ . If we instead select  $\mu = \frac{1}{m} \sum_{i=1}^m \delta_{\mathbf{x}_i}$  (the  
 321 empirical distribution), the right hand side becomes the MSE test error (or  
 322 training error). We are now ready to formulate our theorem on the imperfect  
 323 prediction case.

324 **Theorem 2.** *Let  $S(\mathbf{x}, \boldsymbol{\tau})$  be a system satisfying Equations 4 and 5. Let*  
 325  *$NN(\mathbf{x}, \boldsymbol{\tau})$  be a LEAP net with linear submodules  $\mathbf{d}$  and  $\mathbf{D}$  such that*

$$\begin{aligned} d_\mu (NN(\cdot, \boldsymbol{\tau}^0), S(\cdot, \boldsymbol{\tau}^0)) &\leq d_0 \\ d_\mu (NN(\cdot, \boldsymbol{\tau}^i), S(\cdot, \boldsymbol{\tau}^i)) &\leq d_i, \quad i = 1, \dots, c. \end{aligned} \quad (7)$$

326 *for some constant  $d_0, d_1, \dots, d_c \in \mathbb{R}$ . Then, for any  $\mathcal{I} \subset \{1, \dots, c\}$ , we have*

$$d_\mu (NN(\cdot, \boldsymbol{\tau}^{\mathcal{I}}), S(\cdot, \boldsymbol{\tau}^{\mathcal{I}})) \leq (|\mathcal{I}| + 1)d_0 + \sum_{i \in \mathcal{I}} d_i. \quad (8)$$

327 Theorem 2 shows that as long as a LEAP neural network approximates  
 328 well the ground truth functions 4 by fitting data from a given distribution of

---

<sup>9</sup>We remind that in our problem setting, the system  $S(\cdot, \boldsymbol{\tau})$  we want to identify is deterministic (solutions of differential equations), the labels  $\mathbf{y}$  are thus deterministic given  $\mathbf{x}$  and  $\boldsymbol{\tau}$ . So we don't need to consider the joint distribution on the pair  $(\mathbf{x}, \mathbf{y})$ .



329  $(\mathbf{x}, \boldsymbol{\tau})$ , its error on any points where  $\boldsymbol{\tau}$  is altered and results in combination of  
 330 the vector observed in the training set can be bounded too. This can indeed  
 331 be considered as a super-generalization property. Notice that the weight of  $d_0$   
 332 is larger than that of the others, which suggests we should make more efforts  
 333 on improving the accuracy of the predictions on the reference topology case  
 334  $\boldsymbol{\tau} = \boldsymbol{\tau}^\theta$ . This was the case in all our experiments where more data came  
 335 from this distribution.

336 Obviously, Theorem 1 is a special case of Theorem 2 with  $d_0 = d_1 = \dots =$   
 337  $d_c = 0$ .

338 At last, we emphasize that the distance  $d_\mu$  is a very flexible notion as it  
 339 can be defined as the generalization error or more importantly in practice,  
 340 as the test error.

## 341 4. Empirical evaluation

342 In this section, we experimentally validate our proposed LEAP net archi-  
 343 tecture against baseline methods, using a toy problem and standard bench-  
 344 marks.

### 345 4.1. Experimental setting, simulations, and DC approximation

346 In all experiments, the electricity transportation system is defined by a  
 347 power grid of interconnected nodes (called buses) joined by power transmis-  
 348 sion lines. Most buses, in addition to being interconnected, are connected to  
 349 productions (sources of electrical power, such as power plants) and consump-  
 350 tions (loads consuming power, such as home appliances), altogether referred  
 351 to as “injections”. We recall that we group in a vector  $\mathbf{x}$  all injections and  
 352 that we denote by vector  $\mathbf{y}$  the resulting flows of electricity on all lines, which  
 353 can be calculated with physical equations. We use Hades2<sup>10</sup> to compute the  
 354 flows  $\mathbf{y}$  in our experiments, the official simulator of the French power trans-  
 355 mission operator RTE. Such calculations are called “load flow” in the jargon  
 356 of power systems; they are performed with a Newton-Rhapson solver. Even  
 357 in real data, flows are actually computed with Hades2 because RTE possesses  
 358 very few sensors on power lines measuring actual flows.

359 It is worth pointing out that “load flow” equations are **non linear**, and  
 360 losses caused by Joule’s effect are included. Injected real and reactive power

---

<sup>10</sup>Freeware available at <http://www.rte.itesla-pst.org/>.

361 at a bus are functions of voltage magnitudes and voltage angles. The general  
 362 equations governing a load flow are termed the “AC power flow equations”.  
 363 To simplify analysis, one often considers only real power via the “DC ap-  
 364 proximation,” which allows one to write the vector of power injections as  
 365 a linear function of the vector of voltage angles. Because the DC approxi-  
 366 mation yields somewhat faster calculations than solving the full AC power  
 367 flow, we use the DC approximation as a baseline method against which we  
 368 compare the LEAP nets.

369 One drawback of the DC approximation however, is that it requires the  
 370 exact knowledge of the grid topology (line interconnections). In contrast, our  
 371 LEAP net approach only requires topological changes to be known through  
 372 and abstract numbering representation, which we encode with a binary vector  
 373  $\boldsymbol{\tau}$  representing one-hot encoded unary changes to a reference topology  $\boldsymbol{\tau}^\theta$ .  
 374 The LEAP net task is to learn to emulate the Newton Raphson solver  $S$  that  
 375 computes power flows  $\mathbf{y} = S(\mathbf{x}; \boldsymbol{\tau})$ , without requiring the specific knowledge  
 376 of the graph of the transmission grid. This feature is important, as we will  
 377 see in real data experiments.

#### 378 *4.2. Illustrative experiments on a toy power grid*

379 The objective of these first experiments is to conduct a series of compar-  
 380 isons of LEAP net against other neural network architectures (Fully Con-  
 381 nected network and ResNet). We use the toy power grid illustrated in Figure  
 382 9.

383 In terms of system identification objective, our goal is to train a neural  
 384 network emulator of a load flow simulator, which is capable of making good  
 385 predictions for all possible disconnections of a single line (so-called (N-1)  
 386 situation) and for all possible disconnections of two lines (so-called (N-2)  
 387 situation). In the jargon of power systems, a ”N-1” test (with capital N)  
 388 means evaluating ALL possible single line disconnections, while a ”N-2” test  
 389 would mean evaluating all possible disconnections of pairs of lines.

390 In our experimental setting, for the distribution of  $\mathbf{x}$ , we simulate data  
 391 using semi-realistic injections, inspired by real production and consumption  
 392 time series. For the grid topology, in training data, we vary the topology  
 393 around a reference topology  $\boldsymbol{\tau} = \boldsymbol{\tau}^\theta$  by making unary changes. We give a  
 394 high probability of selecting  $\boldsymbol{\tau}^\theta$  and a small (identical) probability  $p$  to all  
 395 unary changes (single line disconnections). Even though, in practice, higher  
 396 order changes (multiple line disconnections) occur with a small probability,  
 397 we do not include any higher order change in our training data, to make

398 the situation more extreme. In other words, the training distribution is very  
 399 imbalanced with respect to  $\tau$  and by varying  $p$  we monitor the effect of this  
 400 imbalance on system identification.

401 The training was performed on mini batches of 100 samples during 100  
 402 epochs. The input is a vector of 4 elements (2 productions, 2 consumptions),  
 403 and the output is a vector of size 8. The encoder  $\mathbf{E}$  has 4 hidden layers, with  
 404 a latent dimension of 15 and ReLU activations.  $\mathbf{e}$ ,  $\mathbf{d}$  are single layer linear  
 405 neural networks with respective output sizes of 8 and 15. The decoder  $\mathbf{D}$  has  
 406 2 hidden layers, with a latent dimension of 15 and leaky ReLU activations.  
 407 The training of one model takes approximately 5 minutes on a single Nvidia  
 408 Titan Xp GPU.

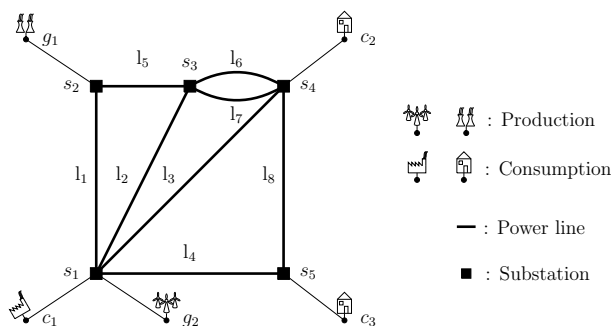


Figure 9: **Small power grid experiment:** Reference topology  $\tau^0$ .

409 Our evaluation method is to use the  $l_2$ -loss on test data sampled in a way  
 410 that topologies corresponding to line disconnections are re-balanced. Two  
 411 distinct test sets are used, namely the “N-1” and the “N-2” test sets.

412 In the “N-1” test single line, disconnections are uniformly distributed.  
 413 Figure 10 shows the results on the “N-1” test after training on the imbal-  
 414 anced training set parameterized by  $p$ . For each value of  $p$ , we have repeated  
 415 experiments for 11 instances of **Fully Connected** neural nets, **ResNets**, and  
 416 **LEAP nets**. We report the  $l_2$ -loss on the test set at the end of the learning  
 417 process. One can observe that for a fixed value of  $p$  LEAP nets performs  
 418 significantly better than Fully Connected nets and ResNets. The two base-  
 419 lines show similar results. In particular, the LEAP net fares better for small  
 420 values of  $p$ , which highlights that it deals better with training distribution  
 421 imbalance.

422 In the “N-2” test, we measure the accuracy on a set of situations in which  
 423 both lines 1 and 6 are disconnected, which has never been observed in the

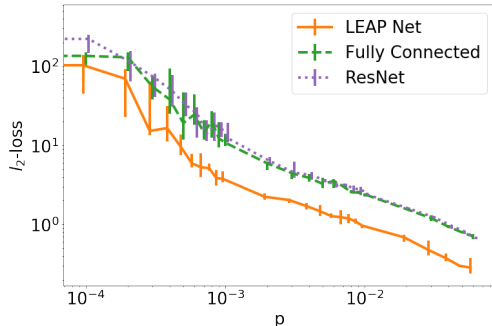


Figure 10: **Small power grid experiment “N-1” test:**  $p$  is the probability of every line to be disconnected in the training set. The “N-1” test set is composed of datapoints that were scarcely observed in the training set. This figure should not be mistaken for a learning curve.

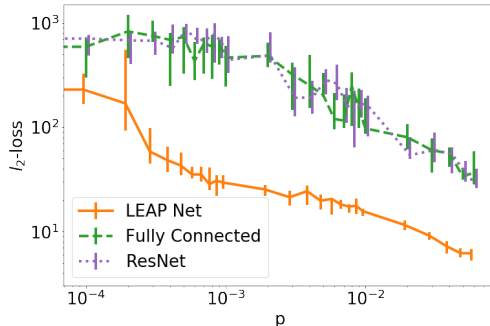


Figure 11: **Small power grid experiment “N-2” test:**  $p$  is still the probability of every line to be disconnected in the training set. The “N-2” test set is now only composed of datapoints where lines 1 and 6 are disconnected. This situation has never occurred in training data.

424 training set. We note that we are not doing a “full” “N-2” test in these  
 425 experiments but consider a single pair. Figure 11 shows the results on the  
 426 “N-2” test after being trained on the training set. For each value of  $p$ , we have  
 427 fully trained multiple instances of a **Fully Connected** neural net, a **ResNet**  
 428 and a **LEAP net**, and we plot here the resulting  $l_2$ -loss on the test set at the  
 429 end of the learning process. The performances are lower than in the “N-1”  
 430 test, which is normal since the situation in this test is not observed during  
 431 training, but the gap between LEAP net and its contenders is larger.

#### 432 4.3. Case 118 synthetic data experiment

433 To further assess the performance of our method, we conducted controlled  
 434 experiments on a standard medium-size benchmark from “Matpower” [25],  
 435 a library commonly used to test power system algorithms [1]: case118, a  
 436 simplified version of the Californian power grid ( $\dim \mathbf{x} = 153$  injections and  
 437  $\dim \mathbf{y} = 186$  power lines). Power grid topology changes consist in recon-  
 438 figuring line connection patterns in one or more substations (see Figure 1).  
 439 Such changes are more complex than simple line disconnections considered  
 440 in the previous section and in [7]. There are 11 558 possible unary actions  
 441 (corresponding to single node splitting or merging, compared to the reference  
 442 topology).

443 From this medium size power grid, we derived 3 distinct datasets:

- 444 • The *training set* is made of 150 000 independent data points: (a)

445 50 000 of them are sampled with injections  $\mathbf{x}$  and the reference topology  
 446  $\boldsymbol{\tau} = \boldsymbol{\tau}^\theta$  (all substations in the reference topology), (b) then 100 unary  
 447 actions are sampled randomly among the 11 558 unary topologies, as  
 448 well as 1 000 injections  $\mathbf{x}$ , which generates the other 100 000 samples.  
 449 Compared to the smallest experiments presented in subsection 4.2, this  
 450 is equivalent to having  $p = 6.7 \times 10^{-3}$ .

- 451 • The *simple generalization test set* consists in datapoints sampled from  
 452 the same distribution as the training set. There are as many samples  
 453 as in the training set.
- 454 • The *super generalization test set* is made of points drawn from a dif-  
 455 ferent distribution than the training one: we sampled 1 500 topologies  
 456 among the 4 950 possible double actions  $\boldsymbol{\tau}^{ij}$ , (knowing that both  $\boldsymbol{\tau}^i$   
 457 and  $\boldsymbol{\tau}^j$  have been observed at training time). Then, for each of these  
 458 1 500  $\boldsymbol{\tau}^{ij}$ , we sampled 100 inputs  $\mathbf{x}$  (with the same distribution as  
 459 the one used for the training and simple generalization test set). We  
 460 used the same physical simulator to compute the  $\mathbf{y}$  from the  $\mathbf{x}$  and the  
 461  $\boldsymbol{\tau}$ . The super-generalization set counts then 150 000 different triplets  
 462  $(\mathbf{x}, \boldsymbol{\tau}^y, \mathbf{y})$ .

463 We compare the proposed **LEAP** net with two benchmarks: the DC ap-  
 464 proximation, a standard baseline in power systems, which is a linearization  
 465 of the AC (Alternative Current) non-linear power flow equations, and the  
 466 **ResNet** neural network architecture (Figure 2), the architecture that resem-  
 467 bles the most to the LEAP net in terms of layer organization. Indeed, in  
 468 this architecture the topology vector  $\boldsymbol{\tau}$  is simply an input and the leap net is  
 469 replaced with a residual layer with the same parameters. The mean-square  
 470 error was optimized using the Tensorflow Adam optimizer with a learning  
 471 rate of  $1 \times 10^{-5}$ . To make the comparison least favorable to LEAP net, all  
 472 hyper-parameters (learning rates, number of units, number of layers, etc.)  
 473 were optimized by cross-validation (using a separate validation set drawn iid  
 474 with the same distribution as the training set) for the baseline ResNet ar-  
 475 chitecture. This could explain why the LEAP net, despite learning faster at  
 476 first, stops learning after some epoch while the ResNet baseline architecture  
 477 keeps improving.

478 The training was performed on mini batches of 100 samples during 250  
 479 epochs. The input is a vector of 153 elements (54 productions, 99 consump-  
 480 tions), and the output is a vector of size 186. The encoder  $\mathbf{E}$  has 6 hidden

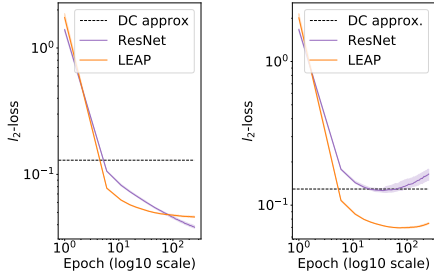


Figure 12: **Case 118 synthetic data experiment** - Neural nets trained with 15000 injections, for  $\tau^\theta$  and unary changes  $\tau^i$ . (left) **Simple generalization**. Test injections for **unary changes**  $\tau^i$ . (right) **Super generalization**. Test injections for **double changes**  $\tau^{i,j}$ . Error bars are 20% – 80% intervals, computed over 30 repeat experiments.

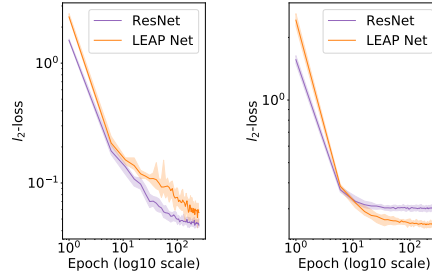


Figure 13: **Real French ultra-high voltage power grid data experiment** - The neural net in both cases is **trained from data until May 2017**. (left) **Simple generalization**. Test set made of randomly sampled data in same time period as training data. (right) **Super generalization**. Test set made of the months of June and July 2017.

481 layers, with a latent dimension of 600 and ReLU activations.  $\mathbf{e}$ ,  $\mathbf{d}$  and  $\mathbf{D}$   
 482 are all single layer linear neural networks with respective output sizes of 200,  
 483 600 and 186. The training of one model takes approximately 2 hours on a  
 484 single Nvidia Titan Xp GPU.

485 Figure 12 indicates that the **LEAP net (orange curves)** performs better  
 486 than the DC approximation (black line) both for simple and super gener-  
 487 alization. Figure 12b shows that the **ResNet architecture (purple curve)** is  
 488 not viable: not only does it perform worse than the DC approximation, but  
 489 its variance is quite high. While it is improving in regular generalization  
 490 with the number of training epochs, its super-generalization performances  
 491 get worse.

#### 492 4.4. Real French ultra-high voltage power grid data experiment

493 We now present results on a part of the French ultra-high voltage power  
 494 grid: the “Toulouse” area with 246 consumption units, 122 production units,  
 495 387 lines and 192 substations often split in a variable number of electrical  
 496 nodes. The inputs  $\mathbf{x}$  representing injections (production and consumption)  
 497 are of dim  $\mathbf{x} = 368(= 246 + 122)$  and the outputs  $\mathbf{y}$  (flows) of dim  $\mathbf{y} = 387$ .  
 498 In this study,  $\mathbf{x}$  and  $\mathbf{y}$  come from real historical data from the company

499 RTE<sup>11</sup>. One important difference when using played-back data, compared to  
500 simulation, is that we cannot intervene (this is strictly observational data).

501 As for the previous experiment we used three distinct data sets in this  
502 setting:

- 503 • The *training set* is composed of data from January 2012 to May 2017  
504 uniformly distributed between these two dates.
- 505 • The *simple generalization test set* is made of data coming from the  
506 same distribution as the one used for training.
- 507 • The *super generalization test set* is made of data coming from the same  
508 area of the French power grid but sampled between June and July  
509 2017. The distribution of the data is different in this set compared  
510 to the training data. This phenomenon is in particular due to the  
511 maintenance operations: power lines are periodically disconnected from  
512 the grid for maintenance purposes. As this type of event happens only  
513 once or twice every 5-10 years for each line<sup>12</sup> there are some power  
514 lines out for maintenance operation in this test set that have never  
515 been disconnected in the training set.

516 The mean-square error was optimized using the Tensorflow Adam opti-  
517 mizer with a learning rate of  $5 \times 10^{-5}$ . Similarly to the previous experiment,  
518 all hyper-parameters (learning rates, number of units, number of layers, etc.)  
519 were optimized by cross-validation (using a separate validation set drawn  
520 iid with the same distribution as the training set) for the baseline ResNet  
521 architecture.

522 The training was performed on mini batches of 100 samples during 250  
523 epochs. The input is a vector of 368 elements (122 productions, 246 con-  
524 sumptions), and the output is a vector of size 387. The encoder  $\mathbf{E}$  has 16  
525 hidden layers, with a latent dimension of 500 and ReLU activations.  $\mathbf{e}$ ,  $\mathbf{d}$   
526 and  $\mathbf{D}$  are all single layer linear neural networks with respective output sizes  
527 of 387, 500 and 387. The training of one model takes approximately 2 hours  
528 on a single Nvidia Titan Xp GPU.

---

<sup>11</sup>Even in real records, flows are estimated, not measured.

<sup>12</sup>The period of time for which a powerline is inspected can vary depending on its age, the past problems it has suffered, the corrosivity of the ambient atmosphere. In general it happens with a few years interval.

529 Another difference between this experiment and the previous one detailed  
530 in section 4.3, is the “action space”. In real data, *actual* power grid topolo-  
531 gies (i.e. specifying line interconnections) are not precisely recorded. Only  
532 information on *line outages* is available to us as surrogate information on  
533 topology. This makes the neural net task harder: it must learn the effects  
534 of hidden topological changes. This unfortunate loss of information on ex-  
535 act grid topology interventions makes it impossible for us to compare our  
536 method to the DC approximation: computing this approximation requires  
537 a full description of the topology. The results of Figure 13 yield the same  
538 conclusions as in the previous section: the **LEAP** model generalizes not only  
539 to data drawn from a similar distribution it was trained on (Figure 13a) but  
540 also to unseen grid states (Fig. 13b), better than the reference architecture,  
541 which is a critical property for our application.

## 542 5. Discussion and further work

543 In this section, we highlight differences between the approach taken in  
544 this paper (LEAP net) and another approach called “Graph Neural Solver”,  
545 which we have introduced elsewhere[8]. We present various research direc-  
546 tions, which could extend the capabilities of the LEAP net. We also make  
547 connections between our approach and related frameworks of transfer learn-  
548 ing, causal modeling, and reinforcement learning.

### 549 5.1. Differences between LEAP net and Graph Neural Solver

550 Alternative approaches to the LEAP architecture to solve the power  
551 flow problem with neural networks include Graph Convolutional Networks  
552 (GCN) [14, 4]. GCNs generalize Convolutional Networks applicable to spatio-  
553 temporal data to any type of structured data whose structure can be repre-  
554 sented by a graph. In [8], we introduced “Graph Neural Solvers” (GNS), a  
555 family of GCNs exploiting the graph structure of power grids. GNSs emulate  
556 Newton-Raphson physical solvers, assuming a **complete knowledge of the**  
557 **power grid structure and its injections**. The key difference between this  
558 approach and the present paper is that LEAP nets do NOT require an ex-  
559 plicit description of the grid topology, the topology is coded in an arbitrary  
560 way via variable  $\tau$ . Lack of explicit knowledge of the graph is a practical  
561 problem facing power grid operators [7]. This situation is exemplified in  
562 the real world experiments of the present paper. From the point of view of  
563 making empirical comparisons, the lack of a full description of the topology



564 makes it impossible to compare Graph Neural Solvers and LEAP nets in this  
565 setting. In that sense, LEAP nets are more versatile.

### 566 5.2. Extensions of LEAP net

567 In this paper, we have made several restrictive assumptions, which were  
568 guided by our application setting. However those could be easily lifted, al-  
569 though we do not have experiments to report:

- 570 • **Structural variable  $\tau$** : In our application setting, structural (action-  
571 able) variables  $\tau$  take only discrete values, but continuous values can  
572 also be considered. They would act as smooth gating factors modu-  
573 lating neurons’ influence rather than disconnecting them. Our choice  
574 of encoding could also be changed from “one hot” encoding to *e.g.* a  
575 redundant or more compact encoding.
- 576 • **Stacked LEAP modules**: Similarly to deep ResNet networks [11],  
577 LEAP modules can be stacked with possible performance gains.

### 578 5.3. Connection with Transfer Learning

579 In this paper we have treated our problem from the angle of system iden-  
580 tification and cast it as a machine learning problem from iid data, lumping  
581 distributional changes in our objective function, which re-balances the im-  
582 portance of various values of  $\tau$ . Another angle would be to treat our problem  
583 as a true change in distribution. The issue of having to deal with changes  
584 in distributions has been extensively studied in the framework of Transfer  
585 Learning (TL) [19, 9]. To make the analogy with our problem setting, we  
586 can think of our reference case  $\tau^0$  as a **source domain**, and other  $\tau$  val-  
587 ues as **target domains**.<sup>13</sup> Future work includes using LEAP nets to model  
588 changes in distributions in which  $\tau$  is a latent variable rather than an ac-  
589 tionable variable.

---

<sup>13</sup>Alternatively, the reference case and unary changes can be lumped in the source domain, then combinations of changes would make the target domain. In this context, the LEAP net architecture, as exemplified in our experimental section, can be interpreted as an architecture capable of “zero shot learning”, i.e. learning combinations of perturbations from zero examples. Structural variable  $\tau$  should be interpreted as a latent variable (rather than an actionable variable) and may take continuous values.

590 *5.4. Link to Causal Modeling*

591 For researchers in causal modeling, we highlight a few connections, be-  
592 cause we have borrowed from vocabulary familiar to this community: **Ac-**  
593 **tions** (or actionable parameter), **Interventions**, **Manipulated distribu-**  
594 **tion** [20]. The problem we addressed is not to recover the causal structure  
595 of our system under study (power grid). Such structure is assumed to be  
596 known, at least to the extent that the inputs to our system (*injections*  $\mathbf{x}$   
597 and *topologies*  $\boldsymbol{\tau}$ ) are considered “causes” and the outputs (power line flows  
598  $\mathbf{y}$ ) are considered “consequences”. We identify function  $\mathbf{y} = S(\mathbf{x}, \boldsymbol{\tau})$  from  
599 observational data, then **intervene** on  $\boldsymbol{\tau}$  and make good predictions. By in-  
600 tervening, we change the input distribution from the **observational distri-**  
601 **bution** to a **manipulated distribution**. Future research directions include  
602 using LEAP nets to identify causal direction, in the spirit of cause-effect pair  
603 studies [10], by comparing alternative models:  $\mathbf{x} = S(\mathbf{y}, \boldsymbol{\tau})$  or  $\mathbf{y} = S(\mathbf{x}, \boldsymbol{\tau})$ .

604 *5.5. Control problems and Reinforcement Learning*

605 Our ultimate goal is to address identification and control problems. One  
606 of the advantages of using neural networks as system emulators is that gradi-  
607 ent policy algorithms may be used to train a “policy network”, *i.e.* a neural  
608 network controller (see *e.g.* [24, 2, 22]). This is one popular approach in  
609 reinforcement learning [23], an area of machine learning, which addresses  
610 the problem of designing “agents” (controllers). Such agents learn from ex-  
611 amples which actions to perform in order to optimize some objective (in  
612 our case minimizing the losses while ensuring that the grid is operated in  
613 security). To that end, we have developed an open-source simulation envi-  
614 ronment to simulate small power grids<sup>14</sup>, and we have started organizing a  
615 series of challenges in reinforcement learning, the first edition of which just  
616 ended ([12rpn.challearn.org](http://12rpn.challearn.org)). The initial analysis of the results is promis-  
617 ing, demonstrating that machine learning matches or exceeds conventional  
618 approaches used by professional dispatchers [15].

619 **6. Conclusion**

620 The LEAP net architecture has been evaluated on a number of real and  
621 artificial test cases, as well as on a synthetic example. Training was performed

---

<sup>14</sup><https://pypownet.readthedocs.io/en/latest/>

622 on data triplets  $(\mathbf{x}, \boldsymbol{\tau}, \mathbf{y})$ , for which  $\boldsymbol{\tau}$  is an actionable variable. The LEAP  
623 net generalizes not only by approximating well  $\mathbf{y}$  for new values of  $\mathbf{x}$  when  $\boldsymbol{\tau}$   
624 follows the observational distribution on which it was trained, but also when  
625  $\boldsymbol{\tau}$  is re-sampled uniformly on a domain included in the support of the original  
626 distribution. This property, which we called super-generalization, allows us  
627 to use LEAP nets for system identification in application, in particular, to  
628 power systems. We analyzed theoretically the properties of LEAP nets and  
629 demonstrated in some restricted case that super-generalization properties  
630 could be explained and guaranteed. We discussed the relationship between  
631 super-generalization and transfer learning and showed that LEAP nets could  
632 be used either for Domain Adaptation or for Multi-Task learning. In our  
633 experiments, we achieved a speed-up of  $\approx 300$  times using the LEAP net,  
634 compared to running the physical simulator, on the synthetic dataset (power  
635 grid of 118 nodes). With data stored in computer memory, our experiments  
636 on the Toulouse area attain a speed of  $\simeq 2000$  times compared to running the  
637 physical simulator. These computational evaluations were carried out using  
638 a single high-end Graphical Processing Unit (GPU) Nvidia Titan X. Further  
639 work includes scaling up our method computationally to the entire French  
640 extra high voltage power grid. We also need to improve prediction accuracy  
641 before our system could be deployed to production. However, the fact that  
642 the regular generalization performance is already within an acceptable ac-  
643 curacy range shows great promises. We anticipate several developments of  
644 LEAP nets in other application domains involving various types of transfer  
645 learning or bias correction, and the incorporation of LEAP nets in control  
646 applications.

## 647 References

- 648 [1] O. Alsac and B. Stott. Optimal load flow with steady-state security.  
649 *IEEE transactions on power apparatus and systems*, PAS-93(3):745–751,  
650 1974.
- 651 [2] J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation.  
652 *Journal of Artificial Intelligence Research*, 15:319–350, 2001.
- 653 [3] Y. Bengio. Deep learning of representations: Looking forward. In *In-*  
654 *ternational Conference on Statistical Language and Speech Processing*,  
655 pages 1–37. Springer, 2013.

- 656 [4] K. Cheolmin, K. Kibaek, B. Prasanna, and A. Mihai. Graph convolutional neural networks for optimal load shedding under line contingency. In *IEEE Power Engineering Society General Meeting (to appear)*, 2019.  
657  
658
- 659 [5] B. Donnot, B. Donon, I. Guyon, Z. Liu, A. Marot, P. Panciatici, and M. Schoenauer. LEAP nets for power grid perturbations. In *ESANN*, Bruges, Belgium, Apr. 2019.  
660  
661
- 662 [6] B. Donnot, I. Guyon, M. Schoenauer, A. Marot, and P. Panciatici. Anticipating contingencies in power grids using fast neural net screening. In *IEEE WCCI 2018*, Rio de Janeiro, Brazil, July 2018.  
663  
664
- 665 [7] B. Donnot, I. Guyon, M. Schoenauer, A. Marot, and P. Panciatici. Fast Power system security analysis with Guided Dropout. In *26th European Symposium on Artificial Neural Networks*, Electronic Proceedings ESANN 2018, Bruges, Belgium, Apr. 2018.  
666  
667  
668
- 669 [8] B. Donon, B. Donnot, I. Guyon, and A. Marot. Graph Neural Solver for Power Systems. In *International Joint Conference on Neural Networks*, Budapest, Hungary, July 2019.  
670  
671
- 672 [9] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.  
673
- 674 [10] I. Guyon, A. Statnikov, and B. B. Batu, editors. *Cause Effect Pairs in Machine Learning*. The Springer Series on Challenges in Machine Learning. Springer International Publishing, 2019.  
675  
676
- 677 [11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.  
678
- 679 [12] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, pages 630–645. Springer, 2016.  
680
- 681 [13] 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.  
682  
683  
684
- 685 [14] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.  
686

- 687 [15] A. Marot, B. Donon, I. Guyon, and B. Donnot. Learning To Run a  
688 Power Network Competition. In *CiML Workshop, NeurIPS*, Montréal,  
689 Canada, Dec. 2018.
- 690 [16] K. S. Narendra and K. Parthasarathy. Identification and control of  
691 dynamical systems using neural networks. *IEEE Transactions on neural  
692 networks*, 1(1):4–27, 1990.
- 693 [17] R. M. Neal. Annealed importance sampling. *Statistics and computing*,  
694 11(2):125–139, 2001.
- 695 [18] T. Nguyen. Neural network load-flow. *IEE Proceedings - Generation,  
696 Transmission and Distribution*, 142:51–58(7), January 1995.
- 697 [19] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions  
698 on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010.
- 699 [20] J. Pearl. *Causality: models, reasoning and inference*, volume 29.  
700 Springer, 2000.
- 701 [21] J. Pearl. *Causality*. Cambridge university press, 2009.
- 702 [22] J. Peters and S. Schaal. Reinforcement learning of motor skills with  
703 policy gradients. *Neural networks*, 21(4):682–697, 2008.
- 704 [23] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduc-  
705 tion*. Adaptive computation and machine learning series. The MIT Press,  
706 Cambridge, Massachusetts, second edition edition, 2018.
- 707 [24] R. J. Williams. Simple statistical gradient-following algorithms for con-  
708 nectionist reinforcement learning. *Machine learning*, 8(3-4):229–256,  
709 1992.
- 710 [25] R. D. Zimmerman and et al. Matpower. *IEEE Trans. on Power Systems*,  
711 pages 12–19, 2011.

## 712 **Appendix A. Proof of Theorem 1**

713 Theorem 1 (Section 3.3) proves that a LEAP net trained on a system  
714 with linear perturbations exhibits super-generalization properties, if it makes  
715 perfect predictions on data triplets coming from the reference configuration.

*Proof.* We recall the LEAP Net architecture  $NN(\mathbf{x}, \boldsymbol{\tau})$  which gives predictions

$$\hat{\mathbf{y}} = NN(\mathbf{x}, \boldsymbol{\tau}) = \mathbf{D}(\mathbf{E}(\mathbf{x}) + \mathbf{d}(\mathbf{e}(\mathbf{E}(\mathbf{x})) \odot \boldsymbol{\tau})) \in \mathbb{R}.$$

Since we assumed  $l = 1$ ,  $\mathbf{D}$  is actually a scalar linear function. By writing  $\boldsymbol{\tau} = (\tau_1, \dots, \tau_c) = \sum_{i=1}^c \tau_i \boldsymbol{\tau}^i$ , we can use the linearity of  $\mathbf{d}$  and  $\mathbf{D}$  to write the output of LEAP Net as

$$\begin{aligned} \hat{\mathbf{y}} &= \mathbf{D} \left( \mathbf{E}(\mathbf{x}) + \mathbf{d} \left( \mathbf{e}(\mathbf{E}(\mathbf{x})) \odot \sum_{i=1}^c \tau_i \boldsymbol{\tau}^i \right) \right) \\ &= \mathbf{D} \left( \mathbf{E}(\mathbf{x}) + \sum_{i=1}^c \tau_i \mathbf{d}(\mathbf{e}(\mathbf{E}(\mathbf{x})) \odot \boldsymbol{\tau}^i) \right) \\ &= f_0(\mathbf{x}) + \sum_{i=1}^c \tau_i f_i(\mathbf{x}) \end{aligned}$$

where

$$\begin{aligned} f_0(\mathbf{x}) &= \mathbf{D}\mathbf{E}(\mathbf{x}) \\ f_i(\mathbf{x}) &= \mathbf{D}\mathbf{d}(\mathbf{e}(\mathbf{E}(\mathbf{x})) \odot \boldsymbol{\tau}^i), \quad i = 1, \dots, c. \end{aligned}$$

As  $NN(\mathbf{x}, \boldsymbol{\tau})$  makes perfect predictions for any data point  $(\mathbf{x}, \boldsymbol{\tau}, \mathbf{y})$  coming from training domains we have:

$$S(\mathbf{x}; \boldsymbol{\tau}) = N(\mathbf{x}, \boldsymbol{\tau})$$

which means that the following equalities hold

$$\begin{aligned} F(\mathbf{x}) &= f_0(\mathbf{x}) \\ F(\mathbf{x}) + \epsilon_i(\mathbf{x}) &= f_0(\mathbf{x}) + f_i(\mathbf{x}), \quad \forall i = 1, \dots, c. \end{aligned}$$

So we must have

$$\begin{aligned} F(\mathbf{x}) &= f_0(\mathbf{x}) \\ \epsilon_i(\mathbf{x}) &= f_i(\mathbf{x}), \quad \forall i = 1, \dots, c. \end{aligned}$$

716 for all  $\mathbf{x}$ . As in our case the distribution  $D(\mathcal{X})$  of  $\mathbf{x}$  doesn't depend on  $\boldsymbol{\tau}$ , the  
717 above equality holds for all  $\mathbf{x}$ .

So when we use the trained  $NN(\mathbf{x}, \boldsymbol{\tau})$  to make predictions for different  $\boldsymbol{\tau}$ ,

$$NN(\mathbf{x}, \boldsymbol{\tau}^{\mathcal{I}}) = f_0(\mathbf{x}) + \sum_{i=1}^c \tau_i f_i(\mathbf{x}) = f_0(\mathbf{x}) + \sum_{i \in \mathcal{I}} f_i(\mathbf{x}) = F(\mathbf{x}) + \sum_{i \in \mathcal{I}} \epsilon_i(\mathbf{x}) = S(\mathbf{x}, \boldsymbol{\tau}^{\mathcal{I}})$$

718 holds for any  $\boldsymbol{\tau}^{\mathcal{I}}$ , which concludes the proof.  $\square$

719 **Appendix B. Proof of Theorem 2**

720 Theorem 2 (Section 3.3) generalizes Theorem 1 to the case of imperfect  
 721 predictions made by LEAP net on data triplets coming from the reference  
 722 configuration.

*Proof.* According to (4) and (5) in Theorem 1, we can write  $S(\cdot, \boldsymbol{\tau}^\theta) = F$  and  $S(\cdot, \boldsymbol{\tau}^i) = F + \epsilon_i$ . And since  $\mathbf{d}$  and  $\mathbf{D}$  are linear, we can write  $NN(\cdot, \boldsymbol{\tau}^\theta) = f_0$  and  $NN(\cdot, \boldsymbol{\tau}^i) = f_0 + f_i$  according to the same argument in the proof of Theorem 1. Then we can rewrite (7) as

$$\begin{aligned} d_\mu(f_0, F) &\leq d_0 \\ d_\mu(f_0 + f_i, F + \epsilon_i) &\leq d_i, \quad i = 1, \dots, c. \end{aligned}$$

Because the distance  $d_\mu$  defined above satisfies triangle inequality (and is translation invariant), we have

$$\begin{aligned} d_\mu(NN(\cdot, \boldsymbol{\tau}^{\mathcal{I}}), S(\cdot, \boldsymbol{\tau}^{\mathcal{I}})) &= d_\mu\left(f_0 + \sum_{i \in \mathcal{I}} f_i, F + \sum_{i \in \mathcal{I}} \epsilon_i\right) \\ &= d_\mu\left(f_0 + \sum_{i \in \mathcal{I}} [(f_0 + f_i) - f_0], F + \sum_{i \in \mathcal{I}} [(F + \epsilon_i) - F]\right) \\ &\leq d_\mu(f_0, F) + \sum_{i \in \mathcal{I}} d_\mu(f_0 + f_i, F + \epsilon_i) + \sum_{i \in \mathcal{I}} d_\mu(f_0, F) \\ &\leq d_0 + \sum_{i \in \mathcal{I}} d_i + \sum_{i \in \mathcal{I}} d_0 \\ &= (|\mathcal{I}| + 1)d_0 + \sum_{i \in \mathcal{I}} d_i, \end{aligned}$$

723 which concludes the proof. □