Learning independent causal mechanisms

Anonymous authors
Paper under double-blind review

Abstract

Independent causal mechanisms are a central concept in the study of causality with implications for machine learning tasks. In this work we develop an algorithm to recover a set of (inverse) independent mechanisms relating a distribution transformed by the mechanisms to a reference distribution. The approach is fully unsupervised and based on a set of experts that compete for data to specialize and extract the mechanisms. We test and analyze the proposed method on a series of experiments based on image transformations. Each expert successfully maps a subset of the transformed data to the original domain, and the learned mechanisms generalize to other domains. We discuss implications for domain transfer and links to recent trends in generative modeling.

1 Introduction

Humans excel at recognizing objects such as handwritten digits from modified or perturbed inputs. Even if presented with digits which are translated, corrupted, or inverted, we can usually correctly label them without the need of re-learning them from scratch. This may be due to the fact that human intelligence utilizes *mechanisms* (such as translation) that are generic and *generalize* across object classes. These mechanisms are *modular, re-usable and broadly applicable*, and the problem of learning such mechanisms from data is a fundamental question for the study of transfer.

In the study of causality, the concept of independent mechanisms plays a central role both on the conceptual level and, more recently, in applications to inference. The *independent mechanism (IM)* assumption states that the causal generative process of a system's variables is composed of autonomous modules that do not inform or influence each other (Schölkopf et al., 2012; Peters et al., 2017).

If a joint density is Markovian with respect to a directed graph $\mathcal{G}$, we can write it as

$$p(\mathbf{x}) = p(x_1, \ldots, x_d) = \prod_{j=1}^{d} p(x_j | \mathrm{pa}_{\mathcal{G}}^j), \tag{1}$$

where $\mathrm{pa}_{\mathcal{G}}^j$ denotes the parents of variable x_j in the graph.

For a given joint density, there are usually many decompositions of the form (1), with respect to different graphs. If we use the *causal* graph, i.e., if the edges of $\mathcal{G}$ denote direct causation (Pearl, 2000), then the conditional $p(x_j|\mathrm{pa}_{\mathcal{G}}{}^j)$ can be thought of as physical mechanism generating x_j from its parents (sometimes referred to as a *causal conditional*). In this case, we can think of (1) as a *generative* model where the term "generative" truly refers to a physical generative process. In the alternative view of causal models as structural equation models, each of the causal conditionals corresponds to a functional mapping and a noise variable (Pearl, 2000).

By the IM assumption, the causal conditionals are autonomous modules that do not influence or inform each other. This has multiple consequences. First, knowledge of one mechanism does not contain information about another one. Second, if one mechanism changes (e.g., due to distribution shift), there is no reason that other mechanisms should also change, i.e., they tend to remain *invariant*. As a special case, it is (in principle) possible to locally

intervene on one mechanism (for instance, by setting it to a constant) without affecting any of the other modules.

The IM assumption can be exploited when performing causal structure inference (Peters et al., 2017). However, it also has implications for machine learning more broadly, in particular for the feasibility of semi-supervised learning, depending on the underlying causal structure (Schölkopf et al., 2012). A model which is expressed in terms of causal conditionals (rather than conditionals with respect to some other factorization) is likely to have components that better transfer or generalize to other settings (Schölkopf et al., 2012), and its modules are better suited for building complex models from simpler ones. Independent modules as sub-components can be trained independently, from multiple domains, are more likely to be re-usable. They can also be easier to interpret since they correspond to physical mechanisms. Animate intelligence cannot afford to learn new models from scratch for every new task. Rather, it is likely to rely on robust local components that can flexibly be re-used and re-purposed. It also requires local mechanisms for adapting and training modules rather than re-training the whole brain every time a new task is learned.

In the present paper, we focus on a class of such modules, and on algorithms to learn them from data. We describe an architecture using competing experts specializing on different transformations. The resulting model permits a form of lifelong learning, with the possibility of easily adding, removing, retraining, and exporting its components independently.

In line with the intuition given above, we illustrate our approach on MNIST digits which have undergone different transformations such as contrast inversion, noise addition and translation. Information about the nature and number of such transformations need not be known at the beginning of training. Our goal is to identify the independent mechanisms linking a reference distribution to a distribution of modified digits, and learn to invert them without supervision. Ultimately, such inverse mechanisms could be used to transform modified digits and classify them using a standard MNIST classifier, thus exhibiting a form of robustness that animate intelligence excels at.

2 Related work

Our work draws from mixtures of experts, domain adaptation, and causality.

Early works on mixture of experts date back to the early nineties (Jacobs et al. (1991), Jordan & Jacobs (1994)), and since then the topic has been subject of extensive research. Recent work include Shazeer et al. (2017), where the authors train a mixture of 1000 experts using a gating mechanism that selects only a very small number of experts for each example, and propose several technical solutions to deal with model and data parallelism. Aljundi et al. (2016) train a network of experts on multiple tasks, with a focus on lifelong learning; autoencoders are trained for each task and used as gating mechanisms.

Another research direction that is relevant to our work is unsupervised domain adaptation (Bousmalis et al., 2016). These methods often use some supervision from labeled data and/or match the two distributions in a learned feature space.[1] With little or no supervision, Bousmalis et al. (2016) learn a function that transforms examples from one domain to another. Fully unsupervised multi-domain adaptation is a more recent and less explored topic; for instance, Yang & Eisenstein (2015) approach the problem for natural language processing tasks using feature embeddings.

The novelty of our work lies in the following aspects: (1) we automatically identify as set of independent (inverse) causal mechanisms; (2) we aim at simultaneously learning to identify and invert these mechanisms, using only data from an original distribution and from the mixture of transformed data, without labels on additional tasks; (3) the architecture is modular, can be easily expanded, and its trained modules can be reused; and (4) the method relies on competition of experts.

Ideas from the field of causal inference inspire the present work. Understanding the data generating mechanisms plays a key role in causal inference, and goes beyond the statistical

[1] See for example Tzeng et al. (2017) for an example of both.

assumptions usually exploited in machine learning. Causality provides a framework for understanding how a system responds to *interventions*, and Structural Equation Models (SEM) are a standard way of describing causal systems (Hoyer et al., 2009; Peters et al., 2014). The IM assumption discussed in the introduction can be used for identification of causal models (Zhang et al., 2015), but causality has also proven to be a useful tool for discussing and understanding machine learning in the non i.i.d. regime. Recent applications include semi-supervised learning (Schölkopf et al., 2012) and transfer learning (Rojas-Carulla et al., 2015), but the authors focus only on regression and linear models. We seek to extend applications of causal inference to more complex settings and aim to learn causal mechanisms from a SEM without any supervision.

3 Learning independent mechanisms as independent modules

The aim of this section is twofold. First, we motivate and formalize the generative process of our data. We start with a canonical distribution P and an a priori *unknown* number of independent mechanisms which act on P. At training, a sample from the canonical distribution is available, as well as a dataset obtained by applying these mechanisms to (unseen) examples drawn from P. Second, we propose an algorithm which recovers and learns to invert the mechanisms in a *completely unsupervised fashion* from these data. Importantly, the learned inverse mechanisms generalize to unseen data points. In the introduction, we illustrate some of the concepts in an example using MNIST.

3.1 Formal setting

Consider a canonical distribution P on $\mathbb{R}^d$, e.g., the empirical distribution defined by MNIST digits on pixel space. We further consider N measurable functions $M_1, \ldots, M_N : \mathbb{R}^d \to \mathbb{R}^d$, called *mechanisms*. Importantly, these are assumed to be independent in nature, and their number is a priori unknown. These mechanisms give rise to N distributions $Q_1, \ldots, Q_N$ where $Q_j = M_j(P)$.[2] In the MNIST example, we consider translations or adding noise as mechanisms, i.e., the corresponding Q distributions are translated and noisy MNIST digits.

At training time, we receive a dataset $\mathcal{D}_Q = (x_i)_{i=1}^n$ drawn i.i.d. from a mixture of $Q_1, \ldots, Q_N$, and a sample $\mathcal{D}_P$ from the canonical distribution P. Our goal is to identify the underlying mechanisms $M_1, \ldots, M_N$ and learn approximate inverse mappings. In particular, we want to map the examples from $\mathcal{D}_Q$ back to their counterpart drawn from P.

If we were given distinct datasets $\mathcal{D}_{Q_j}$ each drawn from Q_j, we could individually learn each mechanism, resulting in independent approximations regardless of the properties of the training procedure. Note that the obtained mappings are not required to approximate the true M_j, i.e., they might not generalize to a different canonical distribution. In our setting, $\mathcal{D}_Q$ is constructed by first taking the union of the sets D_{Q_j}, and then applying a random permutation to the overall set. This corresponds to a dataset where each element has been generated by one of the (independent) mechanisms, but we don't know by which one. Clearly, it should be harder to identify and learn independent mechanisms from such a dataset. This is the setting we address below, and the crucial idea will be that of *competition*.

3.2 Competitive learning of independent mechanisms

In this section, we introduce our training protocol to address the problem defined above.

The training machine is composed of N' parametric functions $E_1, \ldots, E_{N'}$ with distinct trainable parameters $\theta_1, \ldots, \theta_{N'}$. We refer to these functions as the *experts*. Note that we do not require $N' = N$, since the real number of mechanisms is unknown a priori. The objective is to maximize a fitness function $c : \mathbb{R}^d \to \mathbb{R}$ with the key property that c takes high values on the support of the canonical distribution P, and low values outside. Note that it is possible for c to be a parametric function, and for these parameters to be jointly optimized with the experts during training. Below, we specify the details of this rather general definition.

[2] Each distribution Q_j is defined as the pushforward measure induced by P via M_j.

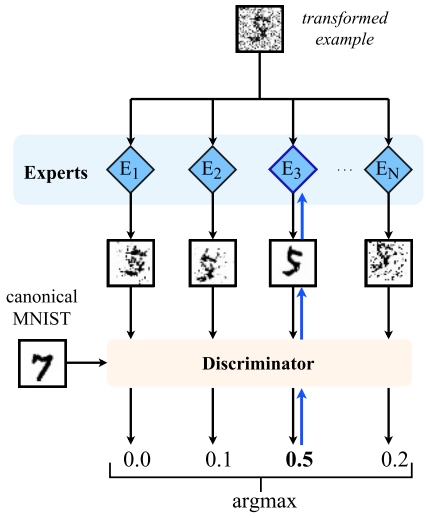


Figure 1: We show how a transformed example, here a noisy digit, is processed by a competition of experts. Only Expert 3 is specializing on denoising, it wins the example and gets updated, whereas the others perform translations and are not updated.

During training, the experts compete for the data points. Each example x' from $\mathcal{D}_Q$ is fed to all experts independently and in parallel. Depending on the fitness for the output of each expert $c_j = c(E_j(x'))$, we select the winning expert E_{j^*}, where $j^* = \arg\max_j(c_j)$. E_{j^*} *wins* the example x', and its parameters θ_{j^*} are updated as to maximize $c(E_{j^*}(x'))$, while the other experts remain unchanged. The motivation behind competitively updating only the winning expert is to enforce specialization; the best performing expert becomes even better at mapping x' back to the corresponding sample from the canonical distribution. Figure 1 depicts this procedure. Overall, our optimization problem reads

$$\theta_1^*, \ldots, \theta_{N'}^* = \arg\max_{\theta_1,\ldots,\theta_{N'}} \mathbb{E}_{x' \sim Q} \left(\max_{j \in \{1,\ldots,N'\}} c(E_{\theta_j}(x')) \right). \tag{2}$$

The training described above raises a number of questions, which we address next.

1) **Convergence criterion.** Since the problem is fully unsupervised, there is no straightforward way of measuring convergence, which raises the question of how to choose a stopping time for the competitive procedure. As an example, one may act according to one of the following: *a*) fix a maximum number of iterations or *b*) stop if each example is assigned to the same experts for a pre-defined number of iterations (i.e., each expert consistently wins the same data points).

2) **Selecting the appropriate number of experts.** Generally, the number of mechanisms N which generated the dataset $\mathcal{D}_Q$ is not available a priori. Therefore, it is fundamental to develop an adaptive procedure for setting up the number of experts N'. This is a common problem shared with most clustering techniques. Given the modular behavior of the procedure, experts may be added or removed during or after training, making the framework very flexible. Assuming however that the number of experts is fixed, we speculate that the following behaviors are likely.

   If $N' > N$ (too many experts): a) some of the experts do not specialize and do not win any example in the dataset; or b) some tasks are divided between experts (for instance, each expert can specialize in a mode of the distribution of the same task). In a), the inactive experts can be removed, and in b) experts sharing the same task can be merged into a wider expert.[3].

[3]However, note that in order to do this, it is necessary to first acknowledge that the two experts have learned part of the same task, which would require extra information or visual inspection

If $N' < N$ (too few experts): a) some of the experts specialize in multiple tasks or b) some of the tasks are not learned by the experts, so that data points from such tasks lead to a poor fitness score across all experts.

While these questions are fundamental, we do not develop them in detail and leave them for further research. However, experiments substantiating these claims can be found in Appendix A.1.

3) **Time and space complexity.** Each example has to be evaluated by all experts in order to assign it to the winning expert. While this results in a computational cost that depends linearly on the number of experts, these evaluations can be done in parallel and therefore the time complexity of a single iteration can be bounded by the complexity to compute the output of a single expert. Moreover, as each expert will in principle have a smaller architecture than a single large network, the committee of experts will typically be faster to execute.

Concrete protocol for neural networks. One possible model class for the experts are deep neural networks. Training using backpropagation is particularly well suited for the online nature of the training proposed: after an expert wins a data point x', its parameters are updated by backpropagation, while other experts remain untouched. Moreover, recent advances in generative modeling give rise to natural choices for the loss function c. For instance, given an autoencoder $A_{\mathcal{D}_P}(x') = D(E(x'))$ trained on the canonical distribution P (where D and E are the corresponding encoder and decoder), one may define $c(x') = -(A_{\mathcal{D}_P}(x') - x')^2$, i.e., the opposite of the reconstruction loss of the autoencoder. The assumption is that the autoencoder will reconstruct x' well if it is drawn from P. Another possibility is to use adversarial training ((Goodfellow et al., 2014)), and use as a fitness function the output of a discriminator network trained on the canonical sample $\mathcal{D}_P$ and against the outputs of the experts. In the next section we introduce a formal description of a training procedure based on adversarial training in Algorithm 1, and present experimental evidence of its good performance.

4 Experiments

In this set of experiments we test the method presented in Section 3 on the MNIST dataset transformed with the set of mechanisms described in detail in the Appendix C, i.e. eight directions of translations by 4 pixels (up, down, left, right, and the four diagonals), contrast inversion, addition of noise, for a total of 10 transformations. We split the training partition of MNIST in half, and transform all and only the examples in the first half: this ensures that there is no matching ground truth for the experts to learn the mechanisms, and that learning is fully unsupervised. As a preprocessing step, the digits are zero-padded so that they have size 32×32 pixels, and the pixel intensities are scaled between 0 and 1. This is done even before any mechanism is applied. We use deep neural networks for both the experts and the selection mechanism, and use an adversarial training scheme.

Each expert E_i can be seen as a generator from a GAN, that is conditioned on an input image instead of (or in addition to) a noise vector. A discriminator D provides gradients for training the experts and acts also as a selection mechanism c: only the expert whose output obtains the higher score from D *wins* the example, and is trained on it to maximize the output of D. We describe the exact algorithm used to train the networks in these experiments in Algorithm 1. The discriminator is trained to maximize the following cross-entropy loss:

$$\max_{\theta_D} \left(\mathbb{E}_{x \sim P} \log(D_{\theta_D}(x)) + \frac{1}{N'} \sum_{j=1}^{N'} \mathbb{E}_{x' \sim Q} \left(\log(1 - D_{\theta_D}(E_{\theta_j}(x'))) \right) \right) \tag{3}$$

For the sake of clarity, we move the experiments where the number of experts N' and the number of true mechanisms N does not match to Appendix A.1. Therefore, we assume for the rest of this section that $N = N'$.

Algorithm 1 Learning independent mechanisms using competition of experts and adversarial training

Precondition: X: data sampled from P; X': data sampled from $\mathcal{D}_Q$; D discriminator; N': number of experts; T: maximum number of iterations;
(p) highlights that the steps in the instruction can be executed in parallel

$\{E_i \leftarrow \text{TrainNewAutoencoderOn}(X')\}_{j=1}^{N'}$ ▷ Init set of experts as approx identity **(p)**
for $t \leftarrow 1$ to T **do**
  $x, x' \leftarrow \text{Sample}(X), \text{Sample}(X')$ ▷ Sample minibatches
  $\{c_j \leftarrow D(E_j(x'))\}_{j=1}^{N'}$ ▷ Scores from D for all outputs from the experts **(p)**
  $\theta_D^{t+1} \leftarrow \text{Adam}(\theta_D^t, \nabla \log D(x) + \nabla(1/N' \sum_{j=1}^{N'} \log(1 - c_j)))$ ▷ Update D **(p)**
  $\{\theta_{E_j}^t \leftarrow \text{Adam}(\theta_{E_j}^t, \nabla \max_{j \in 1,\ldots,N'} \log(c_j))\}_{j=1}^{N'}$ ▷ Update experts **(p)**

Neural nets details The networks and hyperparameters used in the experiments are rather standard. Each expert is a CNN with five convolutional layers, 32 filters per layer of size 3×3, ELU (Clevert et al. (2015)) as activation function, batch normalization (Ioffe & Szegedy (2015)), and zero padding. The discriminator is also a CNN, with average pooling every two convolutional layers, growing number of filters, and a fully connected layer with 1024 neurons as last hidden layer. Both networks are trained using Adam as optimizer (Kingma & Ba (2014)), with the default hyper-parameters.[4]

Unless specified otherwise, after a random weight initialization we first train the experts approximately as the identity, by pretraining them for up to 200 iterations on predicting identical input-output pairs randomly selected in the transformed dataset. This makes the experts start from very similar grounds, and we found that this improved the speed and robustness of convergence. We will refer to this as *approximate identity initialization* for the rest of the paper.

A minibatch of 32 transformed MNIST digits, each transformed by a randomly chosen mechanism, is fed to all experts E_i. The outputs are fed to the discriminator D, which computes a score for each of them. For each example the cross entropy loss in Equation (3) and the resulting gradients are computed only for the output of the highest scoring expert, and they are used to update both the discriminator (when 0 is the target in the cross entropy) and the winning expert (when using 1 as the target). In order to encourage the expert to specialize, the discriminator is also explicitly trained against the outputs of the losing experts. Then, a minibatch of canonical MNIST digit is used in order to update the discriminator with 'real' data.

We ran the experiments 10 times with different random seeds for the initializations. Each experiment is run for 2000 iterations.

5 Results

The experts correctly specialized on inverting exactly one mechanism each in 7 out of the 10 runs; in the remaining 3 runs the results were only slightly suboptimal: one expert specialized on two tasks, one expert did not specialize on any, and the remaining experts still specialized on one task each, thus still covering all the existing tasks. In Figure 2 we show a randomly selected batch of inputs and corresponding outputs from the model. Each independent mechanism was inverted by a different expert.

There are several aspects of the results that we need to analyze.

1) **The experts specialize w.r.t. c .** In Figure 3 we plot the scores assigned by the discriminator for each expert on each task in a typical successful run. Each expert is represented with the same color and linestyle across all tasks. The figure shows that after

[4]For the exact experimental parameters and architectures see the Appendix B or the Pytorch implementation that will be released online.

Figure 2: The top row contains 16 random inputs to the networks, and the bottom row the corresponding outputs from the highest scoring experts against the discriminator after 1000 iterations.

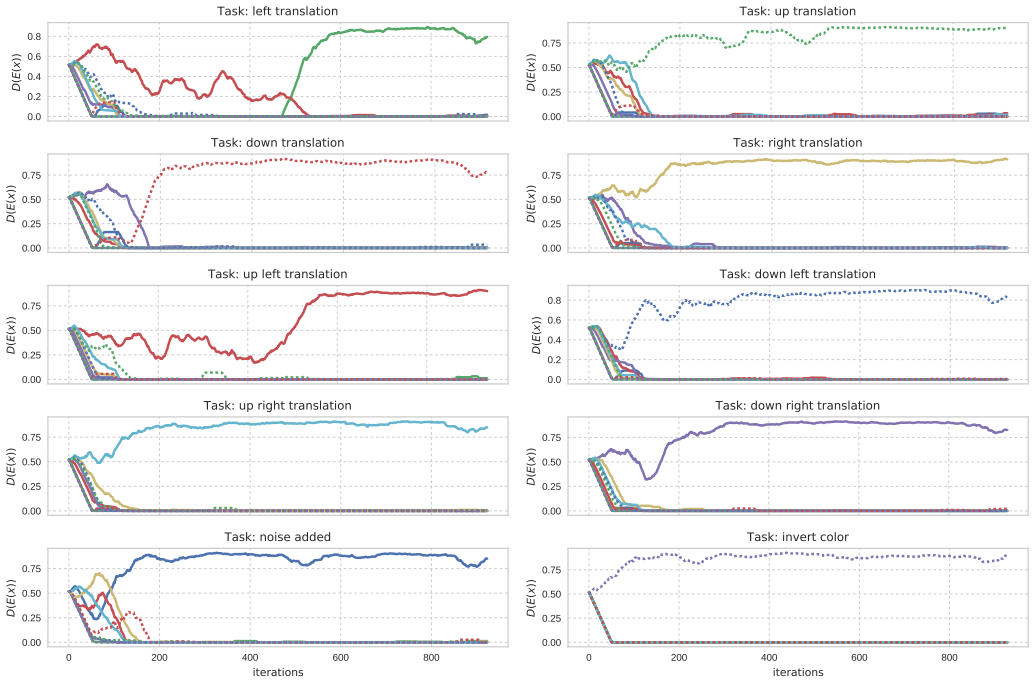
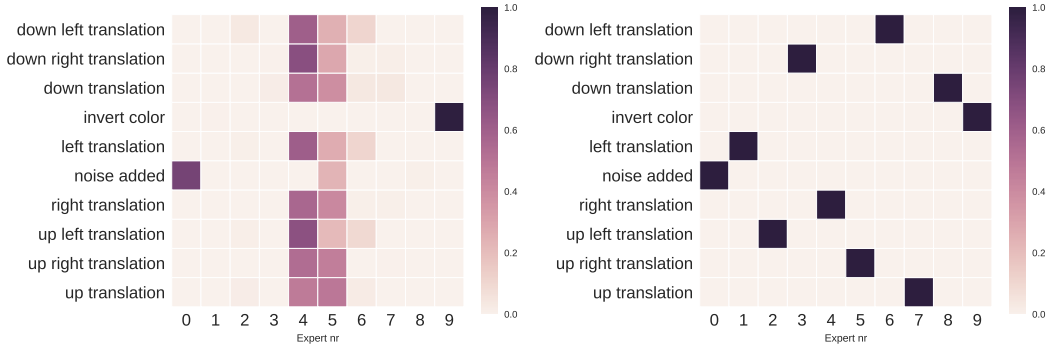


Figure 3: Each line style is associated to the score that an expert obtains on the discriminator when being fed transformed digits using one of the mechanisms. Each expert learns to specialize on a different mechanism. Each curve is smoothed with an average of the last 50 iterations for ease of visualization.

an initial chaotic phase of heavy competition, the experts exhibit the desired behavior and obtain a high score on D on one mechanism each. Note how the red expert tries to learn two similar tasks until iteration 500 (i.e. left and up-left translation), when the green expert takes over one of the task and they can then both quickly specialize. Figure 4 provides further evidence, by visualising that the clusters induced by c are meaningful. We report the proportion of examples from each task assigned to each expert at the beginning and at the end of training. At first, a couple of experts win most examples from all tasks. By the end of the successful training run, each expert wins almost all examples coming from one transformation, and no other.

2) **The transformed outputs generalize to unseen task.** In order to test if the committee of experts can overall recover a good approximation of the original digits, we test the output of our experts against a pretrained MNIST classifier. For this, we use the test partition of the data. We compute the accuracy for three inputs: a) the transformed test digits, b) the transformed digits after being processed by the highest scoring experts, c) the original test digits. The latter can be seen as an upper bound to the accuracy that can be achieved.



(a) At the beginning of training (b) After 1000 iterations

Figure 4: The proportion of data won by each expert for each transformation on the digits from the test set.

As shown by the two dashed horizontal lines in Figure 5, the transformed test digits achieve a 40% accuracy when tested directly on the classifier, while the untransformed digits would achieve $\approx 99\%$ accuracy. The accuracy for the output digits also starts at 40% — due to the identity initialization — and quickly matches the performance of the original digits as it is trained. Note also that after about 600 iterations — i.e. as the networks have seen overall about one third of the whole dataset, and once only — the accuracy is already almost at the upper bound.

3) **The experts learned independent mechanisms.** Finally, we test the networks on inputs that were not transformed with the mechanisms that each of them has learned to invert[5]. As shown in Fig. 6, each network consistently applies the same transformation also on inputs outside of its training distribution, and therefore the experts not only recovered the correct digits for the domain they have specialized on, but **actually learned the independent mechanisms**. Since the experts are fully convolutional networks in this experiment, they could be even be ported to other domains with images of different sizes.

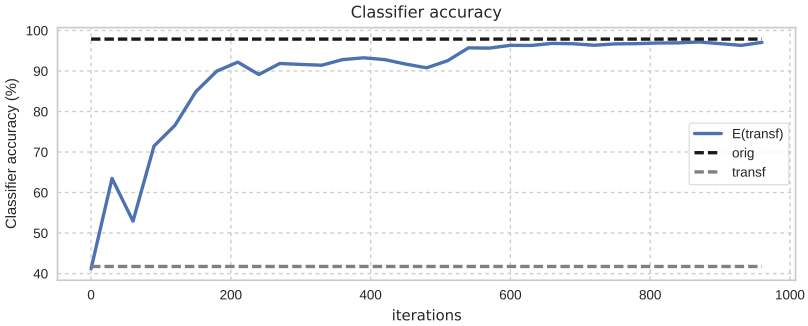


Figure 5: The accuracy of a standard CNN pretrained MNIST classifier against a baseline of transformed test digits $\mathcal{D}_Q$, the same digits after going through our model and the original digits before transformation $\mathcal{D}_P$ (here $\mathcal{D}_P$ corresponds to the ground truth pre-image of the digits in $\mathcal{D}_Q$ for the true applied mechanisms).

Effect of the approximate identity initialization When running the same experiments without the approximate identity initialization, we found that often several experts

[5]As an example, the color inverting network has only been trained on digits that had inverted colors, and never on noisy digits.

fail to specialize. Out of 10 new runs with random initialization, only one experiment had arguably good results, with eight experts specializing on one task each, one expert on two tasks, and the last expert on none. The performance was worse in the remaining runs. We tested whether the problem was that the algorithm takes longer to converge following a random initialization, and ran one additional experiment for $10,000$ iterations. The results did not improve.

A simple single-net baseline Training a single network instead of a committee of experts makes the problem more difficult to solve. Using otherwise the very same training settings, we trained a single network once with 32, once with 64, and once with 128 filters per layer, and none of them managed to correctly learn more than one inverse mechanism.[6] While we do not exclude that with careful hyperparameter tuning it would be possible to correctly learn more mechanisms, it seems at least not to be straightforward.

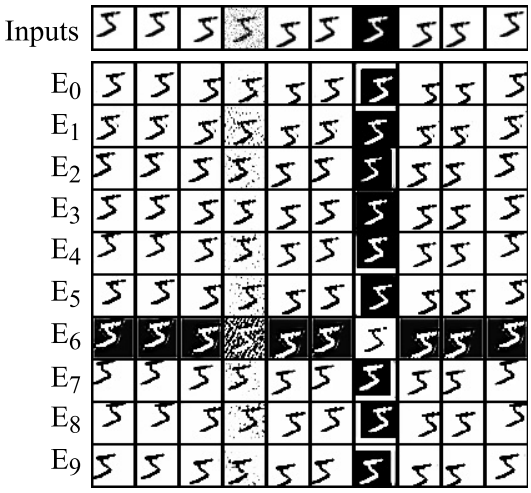


Figure 6: Each column shows how each expert transforms the input presented on top. We arrange the tasks such that on the diagonal there is the highest scoring expert for the input given at the top of the column. It is evident that the experts have learned the mechanisms, as they consistently apply it to digits outside of their training domain.

6 Conclusions

We have developed a method to identify and learn a set of independent causal mechanisms. We reported promising results in an experiment based on image transformations; future work could study more complex settings and diverse domains.

Another natural extension to our work is to consider independent mechanisms that *simultaneously* affect the data (e.g. lighting and subject in a portrait), and to allow multiple passes through our committee of experts to identify local mechanisms (akin to Lie derivatives) from complex data — for example using a recurrent neural network as a fitness function c.

Finally, a note about computational complexity. For a large number of experts, the computational cost might become unnecessarily large. This could be mitigated by hybrid approaches incorporating gated mixture of experts — which typically exhibit lower computational complexity — or a hierarchical selection of competing experts.

We believe our work constitutes an interesting connection between causal modelling and deep learning, and hope it will inspire further approaches connecting these areas.

References

Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. *arXiv preprint arXiv:1611.06194*, 2016.

Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. *arXiv preprint arXiv:1612.05424*, 2016.

[6] A single network with 128 filters per layer has slightly more parameters overall than the committee of 10 experts with 32 filters per layer each.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Patrik O Hoyer, Dominik Janzing, Joris M Mooij, Jonas Peters, and Bernhard Schölkopf. Nonlinear causal discovery with additive noise models. In *Advances in neural information processing systems*, pp. 689–696, 2009.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Judea. Pearl. *Causality*. Cambridge University Press, 2000.

Jonas Peters, Joris M Mooij, Dominik Janzing, and Bernhard Schölkopf. Causal discovery with continuous additive noise models. 2014.

Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of Causal Inference*. The MIT Press, 2017.

Mateo Rojas-Carulla, Bernhard Schölkopf, Richard Turner, and Jonas Peters. Causal transfer in machine learning. *arXiv preprint arXiv:1507.05333*, 2015.

B. Schölkopf, D. Janzing, J. Peters, E. Sgouritsa, K. Zhang, and J. M. Mooij. On causal and anticausal learning. In J Langford and J Pineau (eds.), *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pp. 1255–1262, New York, NY, USA, 2012. Omnipress.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. *arXiv preprint arXiv:1702.05464*, 2017.

Yi Yang and Jacob Eisenstein. Unsupervised multi-domain adaptation with feature embeddings. In *HLT-NAACL*, pp. 672–682, 2015.

Kun Zhang, Jiji Zhang, and Bernhard Schölkopf. Distinguishing cause from effect based on exogeneity. *arXiv preprint arXiv:1504.05651*, 2015.

A Extra results.

A.1 Too many or too few experts.

Too many experts When there are too many experts, for most tasks only one wins all the examples, as shown in Figure 7 where the model has 16 experts for 10 tasks. The remaining experts either do not specialize at all — and therefore can be removed from the architecture — or specialize on the same task, and could therefore be combined if after inspection they are considered to perform the same task. Since the accuracy on the transformed data tested on the pretrained classifier reaches again the upperbound of the untransformed data, and since the progress is very similar to that illustrated in Fig 5, we omit this plot.

Too few experts For a committee of 6 experts, the networks do not reconstruct properly most of the digits, which is reflected by an overall low fitness score on the data. Also, the score against the classifier that does not exceed 72%. A few experts are inevitably assigned to multiple tasks, and by looking at Fig 7 it is interesting to see that the clustering result is still meaningful (e.g. expert 5 is assigned to left, down-left, and up-left translation).

B Details of neural networks

In Table 1 we report the configuration of the neural networks used in these experiments.

C Transformations

In our experiments we use the following transformations

- Translations: the image is shifted by 4 pixels in one of the eight directions up, down, left, right and the four diagonals.
- Color inversion: (or contrast inversion) the value of each pixel — originally in the range $[0, 1]$ — is recomputed as $1-$ the original value.
- Noise addition: random Gaussian noise with zero mean and variance 0.25 is added to the original image, which is then clamped again to the $[0, 1]$ interval.

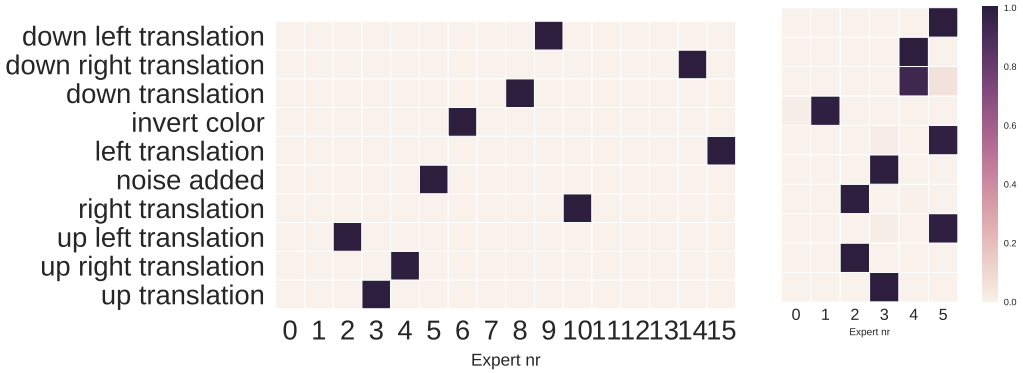


Figure 7: The proportion of data won by each expert for each transformation on the digits from the test set, for the case of 10 mechanisms and more experts (16 on left) or too few (6 on the right).

Table 1: Architectures of the neural networks used in the experiment section. BN stands for Batch normalization, FC for fully connected. All convolutions are preceded by a 1 pixel zero padding.

Expert
Layers
$3 \times 3, 32$, BN, ELU
$3 \times 3, 32$, BN, ELU
$3 \times 3, 32$, BN, ELU
$3 \times 3, 32$, BN, ELU
$3 \times 3, 1$, sigmoid

Discriminator
Layers
$3 \times 3, 16$, ELU
$3 \times 3, 16$, ELU
$3 \times 3, 16$, ELU
2×2, avg pooling
$3 \times 3, 32$, ELU
$3 \times 3, 32$, ELU
2×2, avg pooling
$3 \times 3, 64$, ELU
$3 \times 3, 64$, ELU
2×2, avg pooling
1024, FC, ELU
1, FC, sigmoid