Estimating individual treatment effect: generalization bounds and algorithms - Reproduction project

Bente Gielen [5182794], Ella Fokkinga [4555759], Toke Camps [4544145], and Tom Lim [4442520]

{b.e.j.gielen, e.p.fokkinga, t.m.camps, t.y.lim-1}@student.tudelft.nl

This blog post describes our attempt to reproduce the paper "Estimating individual treatment effect: generalization bounds and algorithms" [1]. More specifically, we will try to recreate table 1 from [1] by implementing our own version of the Counterfactual Regression network in Python using the PyTorch library.

Estimating individual treatment effect, more generally (individual-level) causal inference, is applicable in different fields such as healthcare, economics and education. An example could be a clinician who decides which treatment would be more beneficial for a patient. In this paper, these decisions are based on observational data. Observational data takes into account all of the various explanatory factors that account for the observed correlation between the treatment and response variables [2]. The observed results of for instance a treatment often depends on more factors than only on the variables observed in the data. Because of this, *confounding* can occur. You can see an example of confounding in Fig. 1. Ideally you would want to work under conditions in which all factors that have influenced the results are observed; this is defined as the *no-hidden confounding* assumption. This assumption leads to the *strong ignorability condition*: $(Y_1, Y_0) \perp t|x$ and $0 < p(t = 1|x)$ for all x. This expression does not mean that Y is not dependent on t, but it means that everyone, every x, is equally likely to receive treatment. In this case, we are able to figure out what Y_0 would have been for those who did get treatment, by looking at the effect for those who didn't get treatment! This is a rather unique feature of the model, since we can only observe one of two outcomes, never both. We can now estimate a treatment effect, by comparing those who got treatment to those who did not. To be able to meet this ignorability condition, we need to get a way to make sure that those who get treated are not fundamentally different from those who don't get treated.

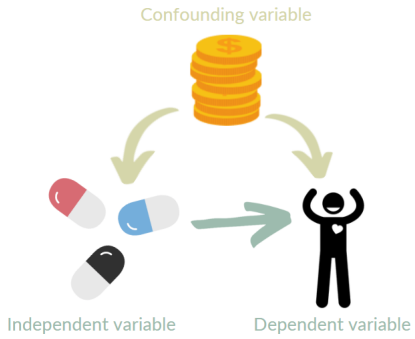

Fig. 1: The principle of confounding: rich people might often get treated earlier with a certain type of expensive drugs. Because you don't measure the *confounding variable* (how rich people are) in your observational data set, you will not see how this influences the dependent variable. In this way, you might overestimate the effect of the *independent variable* (the drugs) on the *dependent variable* (the outcome on people's health), because you don't know how much of the effect is caused by the independent variable and how much by the confounding variable.

To do this, two different types of loss are used in the model, the factual ϵ_F and the counterfactual ϵ_{CF} loss. The factual loss is in this case the loss as we know it, while the counterfactual loss is the interesting part. This ϵ_{CF} describes how well our prediction would do in a reversed world, where the patients are the same; but the doctors are inclined to prescribe exactly the opposite treatment than the one the real-world doctors would prescribe. In reality, we cannot estimate ϵ_{CF}. Therefore, in this paper, it is bounded by using an *Integral Probability Metric (IPM)*: the IPM functions as a measure of distance between the control distribution $p(x|t = 0)$ and the treated distribution $p(x|t = 1)$. By using this metric for the model, the algorithm learns a balanced representation in which the induced treated and control distributions look similar. Two specific IPMs are used: the Maximum Mean Discrepancy (MMD) and the Wasserstein distance.

Model

In this section, we will describe the model architecture and how we implemented our own version in Python. In order to get as similar results as possible, we will recreate the network using the hyper-parameters defined in the paper [1]. Next to recreating the network, we will also evaluate how certain changes in the network affect the results.

Architecture: A basic overview of the architecture is shown in Fig. 2. The first part of the network are the representation layers. These layers learn a function ϕ to map the input X to a certain R (representation). This is done in order to learn a certain representation of the data so that the hypothesis will yield better results.

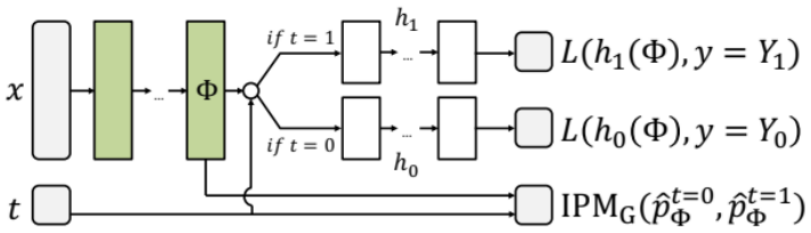

Fig. 2: Neural network architecture for ITE estimation. L is a loss function, IPM_G is an integral probability metric. Note that only one of h0 and h1 is updated for each sample during training.

The representation part of the network consists of three fully-connected linear exponential layers. After the representation layers the data is split into two groups: the treatment (t=1)and the control (t=0) group. Splitting the data in two allows for the use of two hypothesis, one tries to predict the effect of a patient have a treatment and the other one tries to predict the opposite. Both hypothesis are made using three fully-connected linear exponential layers. Together, the representation and hypothesis layers, form the Counterfactual Regression Network (CFRnet) used for Individual Treatment Effect (ITE) estimation. As ϕ is sensitive to scaling, ϕ is linearly normalized.

Loss: In order to optimize the CFRnet, the paper [1] suggests to use a loss function that combines the prediction loss with a IPM penalty term (see Fig.**??**). Besides these two terms, a standard model complexity term is added as well in order to prevent weights to grow too much. This paper uses two IPM terms: the Wasserstein distance and the Maximum Mean Discrepancy (MMD). These two IPM terms calculate the distance between the probability

distribution of the treatment and control group. Ignoring the IPM term could for example lead to rich patients always getting treatment and poor patients not (see Fig. 1). Including the IPM term in the loss function will therefore lead to a more reliable estimation for both groups.

$$\min_{\substack{h,\Phi \\ \|\Phi\|=1}} \frac{1}{n}\sum_{i=1}^{n} w_i \cdot L\left(h(\Phi(x_i), t_i), y_i\right) + \lambda \cdot \mathfrak{R}(h)$$
$$+ \alpha \cdot \text{IPM}_{\text{G}}\left(\{\Phi(x_i)\}_{i:t_i=0}, \{\Phi(x_i)\}_{i:t_i=1}\right),$$
$$\text{with } w_i = \frac{t_i}{2u} + \frac{1-t_i}{2(1-u)}, \text{ where } u = \frac{1}{n}\sum_{i=1}^{n} t_i,$$
and $\mathfrak{R}$ is a model complexity term.

Fig. 3: Function that seeks a representation ϕ and hypothesis h that minimizes a trade-off between predictive accuracy and imbalance in the representation space.

In order to use this loss function to optimize our CFRnet we need a technique to determine in which direction the weights of the CFRnet need to be adjusted. The paper [1] describes the use of Adam [3] as the optimizer function, as we attempt to get similar results we will use Adam as well. In short Adam finds in which direction the loss function is minimizing which allows us to adjust the weights of the network slightly and in doing so, converting to a optimal solution. Adam is a very popular choice when it comes to optimizers as it combines momentum with RMSprop, resulting in a very fast and accurate optimizer.

Experimental set-up

In this section we will describe the setup we used in our attempt to recreate table 1 from [1]. First we will briefly discuss the relevant hyperparameters of our network. After this, we will talk about the dataset used for training and which metrics were used for evaluation. Also, we propose some alterations in our network and which parameters we could tune to investigate whether we can improve the results.

Hyperparameters: Neural networks can have millions of parameters, luckily many of those are learned parameters meaning that you don't have to specify them yourself. However, in order to configure a network for a specific case some hyperparameter do have to be specified. Some of these are for example the amount of layers the network is built with, these parameters define the general architecture of the network and are already defined in section . Besides these general hyperparameters there are some parameters that are tuned during the training process. Fortunately, since we are trying to reproduce the results obtained in [1], we can just use the parameters used in the paper. Therefore we don't see much use in just copying these basic hyper-parameters, like the learning rate, from the paper into this report. However, there is one parameter worth mentioning since it has a large impact on the type of experiment that is performed. That hyperparameter is alpha (α in the equation in Fig. 3). This parameter defines the impact of the earlier described IPM penalty. Using a value of $\alpha > 0$ is referred to as Counterfactual Regression (CFR) and using a value of $\alpha = 0$ is referred to as Treatment-Agnostic Representation (TAR). We experiment with both scenarios. For reproduction of the results in table 1 from [1] we use $\alpha = 0.3$ which was also done in the paper.

Datasets: This paper uses two datasets, one is a simulated dataset called the Infant Health and Development Program (IHDP) The IHDP dataset consists of 747 units (139

treated, 608 control) and 25 covariates measuring aspects of children and their mothers. Using this dataset one can predict the effect of a treatment for a patient based on 25 features. The results in the paper are the averaged results over 1000 realizations with 63/27/10 train/validation/test splits. Since we don't have as much computing power we will average over less realizations.

Evaluation measures The evaluation metrics that have been used in the paper for the IHDP dataset are ϵ_{PEHE} and ϵ_{ATE}. The Precision in Estimation of Heterogenous Effect (PEHE) is the most important metric. PEHE measures the ability of a model to estimate the difference in effect between treatment t_0 and t_1 [4]. ϵ_{PEHE} is calculated by Eq. 1, this measures the mean squared error between the true difference in effect and the predicted difference in effect indexed by n over N samples.

$$\epsilon_{PEHE} = \int_{\chi} (\hat{\tau_f}(x) - \tau(x))^2 p(x) dx \tag{1}$$

The absolute error in Average Treatment Effect (ATE) measures the average difference in effect across the whole population and the formula for calculating ϵ_{ATE} is given in Eq. 2 with $m_1 = E[Y_1|x]$ and $m_0 = E[Y_0|x]$ are is a useful measure to check whether your ITE estimator performs well at comparing two treatments across the entire population [4].

$$\epsilon_{ATE} = | \frac{1}{n} \sum_{i=1}^{n} (f(x_i, 1) - f(x_i, 0)) - \frac{1}{n} \sum_{i=1}^{n} (m_1(x_i) - m_0(x_i)) | \tag{2}$$

These outcome measures are estimated for two different tasks, *within-sample* and *out-of sample*. For within-sample, the ITE is computed for all units, for which we know the factual outcome of the treatment is observed. This is the same as selecting a cohort once for a specific treatment, and not changing this choice. The within-sample is computed over the training and validation sets combined. On the other hand, the out-of-sample corresponds with estimating the ITE for a unit with no observed outcomes; you don't know the effect of any treatments. This is the same as selecting the best treatment for a new patient. It is computed over the test set.

Altering of parameters: We choose to look into the relative effects of altering our network. The relative effects will be investigated on CFR_{wass} as using Wasserstein showed most promising results in the paper, so we were curious to see if this could be improved even more. First, we will look into the effect of increasing α to 0.5 and 0.7. This means the IPM penalty will have a bigger influence on the calculated loss. As was described in Architecture, ϕ is linearly normalized. We also look into the performance of the network when ϕ is not normalized. This is interesting to see as we didn't totally understand their explanation on why to normalize ϕ. From how we interpreted the network, we assumed that the representation layers as well as the regression layers were updated by the IPM_G term. However, if we look at the Fig. 2, it looks like only the regression layers are updated. As this was ambiguous in the paper we also tried to see what the effect was if we only updated the regression layers.

Results

In Table 1 you can find the $\sqrt{\epsilon_{PEHE}}$ ϵ_{ATE} for OLS/LR_1, OLS/LR_2, TARNet and the CFR using MMD and Wasserstein. The table shows the within-sample and out-of-sample results. The within-sample result is calculated over the training and validation set and the out-of-sample result is calculated over the test set. Every experiment has been trained with the training-set of the training data. In total 600 epochs per experiment were used and

per epoch random mini-batches of 100 samples were used. Per 20 epochs the predictions have been computed over the complete training data (including validation set) and the complete test data. The final $\sqrt{\epsilon_{PEHE}}$ and ϵ_{ATE} are averaged over all 20 experiments that were used. The results are visualised over all experiments for $CFR_{wasserstein}$ (Fig. 4), CFR_{MMD} (Fig. 5) and TARNet (Fig. 6. Next to that we also visualed on how the $\sqrt{\epsilon_{PEHE}}$ and ϵ_{ATE} progressed within one experiment in Fig. 7, this is visualised for the first experiment of

In Table 2, the results for tuning the parameters are shown. For the evaluation of the hyperparameters we will look at the relative changes to the network, therefore we first ran the CFR using Wasserstein with our default values using 5 experiments and 600 epochs. Next the α was adjusted to 0.5 and 0.7 to investigate the effect of the IPM terms. Then we looked at the effect of deleting the normalization from ϕ (CFR_{nonorm}). Last, the effect of only updating the regression layers was investigated ($CFR_{updatereg}$).

Table 1: This table shows the results of the $\sqrt{\epsilon_{PEHE}}$ ϵ_{ATE} for OLS/LR_1, OLS/LR_2, TARNet and the CFR using MMD and Wasserstein. The values are calculated within-sample and out-of-sample.

	Within-sample		**Out-of-sample**	
	$\sqrt{\epsilon_{PEHE}}$	ϵ_{ATE}	$\sqrt{\epsilon_{PEHE}}$	ϵ_{ATE}
OLS/LR_1	7.90±8.97	4.89±2.27	7.25±7.59	4.87±2.93
OLS/LR_2	7.91±8.97	4.90±2.26	7.24±7.60	4.86±2.94
TARNet	5.43±7.74	1.99 ± 1.91	5.00±6.26	2.12±2.21
CFR_{MMD}	5.82± 7.87	2.40 ± 3.42	5.59 ± 6.68	2.62 ± 3.81
CFR_{WASS}	5.13 ± 7.44	0.674 ± 1.69	5.00 ± 6.24	0.991 ± 2.22

Table 2: This table shows the results of the $\sqrt{\epsilon_{PEHE}}$ ϵ_{ATE} for tuning the parameters. The values are calculated within-sample and out-of-sample.

	Within-sample		**Out-of-sample**	
	$\sqrt{\epsilon_{PEHE}}$	ϵ_{ATE}	$\sqrt{\epsilon_{PEHE}}$	ϵ_{ATE}
Default CFR	0.831 ± 0.157	0.223 ± 0.0870	0.948± 0.254	0.219± 0.089
CFR $\alpha = 0.5$	1.17 ± 0.241	0.208 ± 0.0892	1.28 ± 0.320	0.225 ± 0.0728
CFR $\alpha = 0.7$	1.28 ± 0.199	0.325±0.283	1.37±0.237	0.399±0.253
CFR_{nonorm}	5.88 ± 5.68	4.30 ± 4.70	5.62 ± 5.17	3.98 ± 4.15
$CFR_{updatereg}$	1.39 ± 0.213	0.302 ± 0.0247	1.52±0.265	0.297±0.108

Discussion and Conclusion

Discussion

We did not manage to reproduce similar results. The main problem that arises from our results is in the progression of the ATE and PEHE of the model. You would expect the error to decrease while training the model. Even though our model sometimes performs very well

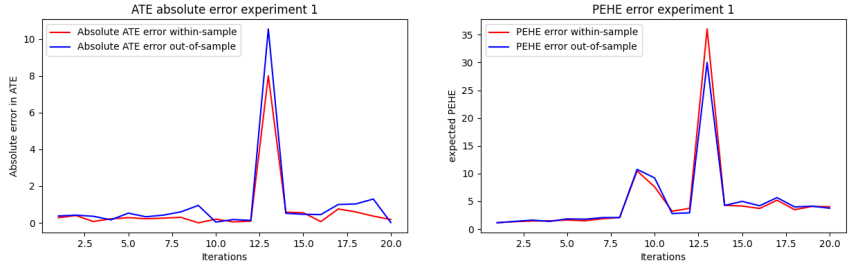

Fig. 4: Wasserstein Progress of $\sqrt{\epsilon_{PEHE}}$ and ϵ_{ATE} over 20 experiments using Wasserstein.

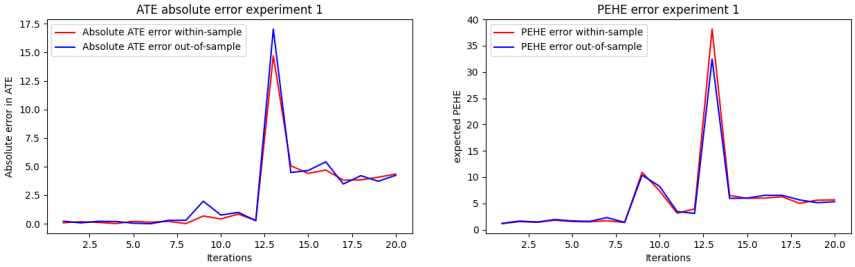

Fig. 5: MMD Progress of $\sqrt{\epsilon_{PEHE}}$ and ϵ_{ATE} over 20 experiments using MMD.

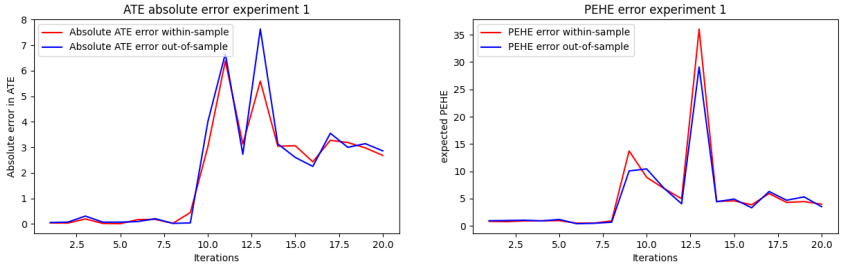

Fig. 6: TARnet Progress of $\sqrt{\epsilon_{PEHE}}$ and ϵ_{ATE} over 20 experiments using no IPM term (TARnet).

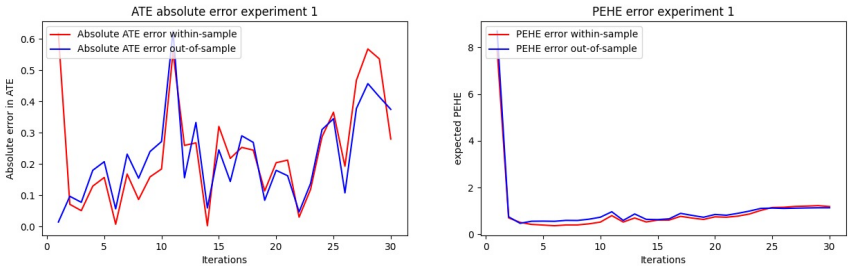

Fig. 7: MMD Progress of $\sqrt{\epsilon_{PEHE}}$ and ϵ_{ATE} within experiment 1 using $CFR_{wasserstein}$.

(low ATE and PEHE), very large peaks in the error also show up. In addition, no decrease is visible in the figures. As can be seen in Table 2, when running less experiments the results are better and are more similar to the results from Table 1 of [1]. We did not find the cause of this error and it would be very interesting to hear if somebody can find an explanation for this. Two possible explanations could be a mistake in our network architecture or due to the fact that our computers cannot run 1000 experiments. Next to that, there are also some variations on how to calculate $\sqrt{\epsilon_{PEHE}}$ and ϵ_{ATE}, it is not totally clear to us which values they used in the paper. You can take the last value in every experiment after 600 epochs but you can also take the average value over one experiment. We decided to take the last value in every experiment after 600 epochs. As can be visualized in Fig. 7 the $\sqrt{\epsilon_{PEHE}}$ decreases within one experiment but the ϵ_{ATE} keeps on fluctuating during 600 epochs.

As already stated, we had some doubts about how to interpret the network architecture from the paper. From Fig. , it appears as if the IPM term does not affect the weights of the representation layers. However, from the algorithm description it appears to also update these. When we noticed this, we decided to incorporate this in our experiment about altering the parameters. From the results of Table 2 it looks like updating the representation layers as well at the regression layers obtains better results. Next tot that, increasing α doesn't improve the results and neither does setting the α to zero in TARNet so it appears that that $\alpha = 0.3$ is the optimal value that has been used in the paper. As we didn't fully understand why it was necessary to normalize the representation layer ϕ, it was interesting to see its effect to the network. From Table 2, we found that it is necessary to normalize to obtain better results.

Conclusion

We didn't manage to reproduce the values of the paper. Some possible explanations on why we were unable to do this could be a mistake in our network. Next to that, it could also be due to some ambiguous parts in the paper which made it hard to fully understand what the authors had done.

References

1. U. Shalit, F. D. Johansson, and D. Sontag, "Estimating individual treatment effect: generalization bounds and algorithms," in *International Conference on Machine Learning*, pp. 3076–3085, PMLR, 2017.
2. P. R. Hahn, C. M. Carvalho, D. Puelz, and J. He, "Regularization and confounding in linear regression for treatment effect estimation," *Bayesian Analysis*, vol. 13, no. 1, pp. 163–182, 2018.
3. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
4. P. Schwab, L. Linhardt, and W. Karlen, "Perfect match: A simple method for learning representations for counterfactual inference with neural networks," *arXiv preprint arXiv:1810.00656*, 2018.

Who did what?

Bente: Implementation of Wasserstein, find out which parameters might be useful to adjust and what to alter in the network. Writing of results and discussion.

Ella: Implementation of mmd, and the evaluation. Writing of the introduction for the blog-post and Ella made our poster for the presentation.

Toke: General structure of the CFR, we could call Toke our project manager.
Tom: Tom made OLS_{LR1} and OLS_{LR2} and wrote the model and experimental set-up for the blog post.