Code Instructions*

These are the instructions for the Python code for a Monte Carlo simulation used to support findings of Farrell, M.H., Liang, T. and Misra, S., 2019: "Deep neural networks for estimation and inference", arXiv preprint arXiv:1809.09953. The code was tested on a 64-bit Windows using Python 3.6.8.
This code first creates an artificial dataset according to a Data Generating Process (DGP). Next, it generates a neural network with a prescribed architecture that estimates treatment effect coefficients and, in the case when treatment is not randomized, it also creates a neural network that estimates propensity scores. These neural networks are then trained and their outputs are fed to influence functions which estimates are used to build 95% confidence interval for average treatment effect. Next, the code checks whether the true average treatment effect of the artificial dataset is contained within the estimated confidence interval. Finally, each run of the model is appended to a summary .csv file that contains information about all the relevant statistics recorded during the run (see Summary file section below for more information).

DGP

The artificial dataset is a set of consumer characteristics (covariates), $\boldsymbol{x}_i$, treatments, t_i, and outcomes, y_i, where $i \in \{1, ..., N\}$ labels different consumers. Covariates are vectors of length $d+1$, where first elelment is set to 1 and the rest are drawn from a uniform probability distribution, $\mathcal{U}(0,1)$. For each consumer, we assign $t_i = 1$ with a probability $p(\boldsymbol{x}_i)$ and $t_i = 0$ with a probability $1-p(\boldsymbol{x}_i)$. The probability $p(\boldsymbol{x}_i)$ is chosen in one of two ways: 1) it is equal to 0.5 and independent of $\boldsymbol{x}$, or 2) it depends on $\boldsymbol{x}_i$ as

$$p(\boldsymbol{x}_i) = \frac{1}{1+e^{-\boldsymbol{\alpha}_p' \boldsymbol{x}_i}} \ , \tag{1}$$

where $\boldsymbol{\alpha}_p$ is a vector of length $d+1$ and the number of non-zero elements $\left\|\alpha_p\right\|_0 = 21$. The first entry of this vector, $\alpha_{p,1}$, is bias term and is equal to 0.09, while the rest are drawn from a uniform probability distribution $\mathcal{U}(-0.55; 0.55)$. Once chosen, this vector is kept the same for all the runs of the Monte Carlo simulation. Once treatments and covariates are chosen, the outcomes, y_i, are calculated as:

$$\begin{aligned} y_i &= \mu_0(\boldsymbol{x}_i) + \tau(\boldsymbol{x}_i) t_i + \varepsilon_i \ , \\ \mu_0(\boldsymbol{x}_i) &= \boldsymbol{\alpha}_\mu' \boldsymbol{x}_i + \boldsymbol{\beta}_\mu' \varphi(\boldsymbol{x}_i) \ , \\ \tau(\boldsymbol{x}_i) &= \boldsymbol{\alpha}_\tau' \boldsymbol{x}_i + \boldsymbol{\beta}_\tau' \varphi(\boldsymbol{x}_i) \ , \end{aligned} \tag{2}$$

where "errors," ε_i, are drawn independently from a normal distribution $\mathcal{N}(0,1)$. The function $\varphi(\boldsymbol{x}_i)$ is a second degree polynomial including pairwise interactions. First entry of the constant vector $\boldsymbol{\alpha}_\mu$, $\alpha_{\mu,1}$, is equal to 0.09 while the rest are drawn from a normal distribution, $\mathcal{N}(0.3, 0.7)$. First entry of the constant vector $\boldsymbol{\alpha}_\tau$, $\alpha_{\tau,1}$, is equal to -0.05 while the rest are drawn from a uniform distribution, $\mathcal{U}(0.1, 0.22)$. The model can be prescribed to be linear in the code in which case, $\boldsymbol{\beta}_\mu = \boldsymbol{\beta}_\tau = 0$. Otherwise $\boldsymbol{\beta}_\mu$ is drawn from a normal distribution, $\mathcal{N}(0.01, 0.3)$, while $\boldsymbol{\beta}_\tau$ is drawn from a uniform distribution, $\mathcal{U}(-0.05, 0.06)$. Once $\boldsymbol{\alpha}_\mu$, $\boldsymbol{\alpha}_\tau$, $\boldsymbol{\beta}_\mu$, and $\boldsymbol{\beta}_\tau$ are chosen, they are kept the same for all the runs of the Monte Carlo simulation.

*The code and this instructions were created by Milica Popovic under supervision of Sanjog Misra.

Model parameters

Below is a list of parameters that specify the model. Parameters of the type `FLAGS.parameter` are flagged and can be defined by the user from a command prompt, while other parameters are hard-coded and need to be specified directly in the code.
You can change a default value of the `FLAGS.parameter` from command prompt by typing the following command:

```
$python MonteCarlo_simulation_for_average_treatment_effects_using_NNs.py
    --nconsumer_characteristics=100
```

To change multiple flagged parameters simply add `-- parameter value` as many times as you would like. For example:

```
$python MonteCarlo_simulation_for_average_treatment_effects_using_NNs.py
    --nconsumer_characteristics=100 --update=True --model=simple
```

The following parameters are flagged:

1. `FLAGS.update`: If True, the simulation results will be saved in the summary .csv file. (Default value: False).
2. `FLAGS.plot_true`: Turn plotting on or off. If many simulations are run, set to False since the plots are saved in memory until the program finishes all the runs which can exhaust memory. (Default value: False).
3. `FLAGS.verbose`: Turn printing detailed messages on or off. If more than a few simulations are run, than maybe it's better to set the value for this parameter to False. (Default value: True).
4. `FLAGS.nsimulations`: Number of simulations to run. (Default value: 1).
5. `FLAGS.nconsumer_characteristics`: Number of consumer characteristics in the artificial dataset. In our simulations, we used 20 and 100 consumer characteristics. (Default value: 100).
6. `FLAGS.treatment`: Method for applying treatment in the artificial dataset. Options are `'random'` and `'not_random'`. If `'random'`, consumers are being treated at random with probability of 0.5. Otherwise, probability of being treated is a function of consumer characteristics. (Default value: `'not_random'`).
7. `FLAGS.model`: Choose the dependence of coefficients μ_0 and τ on consumer characteristics. Options are `'simple'` and `'quadratic'`. If `'simple'`, coefficients μ_0 and τ in the artificial dataset depend linearly on consumer characteristics. Otherwise, the dependence is quadratic. (Default value: `'quadratic'`).
8. `FLAGS.architecture`: Choose the architecture of the neural net. Options are `'architecture_1_'`, `'architecture_2_'`, ... , `'architecture_9_'`. We consider a variety of network architectures in estimation phase of our Monte-Carlo exercise. These architectures are variants of the ones used in the empirical application (which were customized for the application). All networks we investigate are ReLU-based MLPs, but they vary in their depth and width. We varied the depth from 3 to 6 layers and the width from 5 to 80. The exact specifications are in Table below. All hidden layers have ReLU activation function. No activation function was used on output layer. (Default value: `'architecture_1_'`).
9. `FLAGS.data_seed`: Which seed number to use for creation of fake dataset. If it is set to None, a random seed is used. Default None.

Model parameters continued on next page...

Table 1: Monte Carlo Architectures Explored

Architecture	Structure
1	{20, 10, 5}
2	{60, 30, 20}
3	{80, 80, 80}
4	{20, 15, 10, 5}
5	{60, 30, 20, 10}
6	{80, 80, 80, 80}
7	{20, 15, 15, 10, 10, 5}
8	{60, 30, 20, 20, 10, 5}
9	{80, 80, 80, 80, 80, 80}

Below is a list of hard-coded parameters.

1. `train_proportion`: A proportion of the dataset to be used for training. Has to be between 0 and 1. If it is set to 1, than `early_stopping` is automatically set to False and both neural networks (NNs) will be trained on the whole dataset for the `max_nepochs`. Otherwise, training will be stopped when there is no improvement of the loss on the validation set for `max_epochs_without_change` or when `max_nepochs` is reached. The parameters will be retrieved at the epoch where best validation loss is recorded. (Default: 0.9)

2. `max_nepochs`: Maximum number of epochs to run before stopping the training of the NN. (Default: 5000)

3. `max_epochs_without_change`: How many epochs with no improvement to wait before stopping the training. (Default: 30)

4. `hidden_layer_sizes`: Hidden layers for the first NN that estimates the treatment effect coefficients. Should be supplied as a list of integers. Length of the list defines the number of hidden layers. Entries of the list define the number of hidden units in each layer. (Default depends on `FLAGS.architecture`)

5. `activation_functions`: Activation function for each layer of the first NN that estimates the treatment effect coefficients. Should be supplied as a list with entries that can be `'relu'`, `'prelu'`, `'srelu'`, `'plu'`, `'elu'`, or `'none'`. Length of this list has to be `len(hidden_layer_sizes)+1`. The last element in the list should be `'none'`, as it corresponds to the output layer. (Default: all `'relu'` except `'none'` on the last layer).

6. `dropout_rates_train`: Dropout rate on input and each hidden layer of the first NN that estimates the treatment effect coefficients. Should be supplied as a list with entries between 0 and 1. Length of this list has to be `len(hidden_layer_sizes)+1`. (Default depends on `FLAGS.architecture`)

7. `hidden_layer_sizes_treatment`: Hidden layers for the second NN that estimates the propensity scores. Should be supplied in the same way as `hidden_layer_sizes`. (Default: $[50, 30]$)

8. `activation_functions_treatment`: Activation function for each layer of the second NN that estimates the propensity scores. Should be supplied in the same way as `activation_functions`. (Default: `['relu', 'relu', 'none']`)

9. `dropout_rates_train_treatment`: Dropout rate on each hidden layer of the second NN that estimates the propensity scores. Should be supplied in the same way as `dropout_rates_train`. (Default: $[0, 0, 0]$)

Model parameters continued on next page...

10. `Optimizer`: Which optimizer to use. Options are `'RMSProp'`, `'GradientDescent'`, or `'Adam'`. (Default: `'Adam'`)
11. `learning_rate`: Learning rate. (Default: 0.009)
12. `batch_size`: Batch size. Should be smaller than the length of the training set. To train on the whole training set rather than on mini-batches, set to None. (Default: 128)
13. `alpha`: Regularization strength parameter. (Default: 0.)
14. `r`: Mixing ratio between Ridge and Lasso regression. If it's equal to 0., than the regularization is equal to Ridge regression. If it is equal to 1., it is equal to Lasso regression.. (Default: 0.2)
15. `nconsumers`: Number of consumers. (Default: 10000)

Summary file

After each model run, the summary is appended to a .csv file. This file contains a performance summary of the neural networks on that particular dataset. Below is a short explanation of each column in the summary file.

1. `Model number`: Run number.
2. `seed`: Seed number.
3. `best_epoch`: Epoch at which best validation loss was recorded for the first NN if `train_proportion` less than 1. Otherwise, equal to `max_nepochs` for which the NN is trained.
4. `best_epoch_t`: Epoch at which best validation loss was recorded for the second NN if `train_proportion` less than 1. Otherwise, equal to `max_nepochs` for which the NN is trained.
5. `total_nparameters`: Total number of parameters (weights and biases) in the first NN.
6. `total_nparameters_t`: Total number of parameters (weights and biases) in the second NN.
7. `Loss best`: Minimum value of loss for the first NN achieved on the validation set if `train_proportion` less than 1. Otherwise, loss achieved on the whole dataset during the last epoch.
8. `Loss best_treatment`: Minimum value of loss for the second NN achieved on the validation set if `train_proportion` less than 1. Otherwise, loss achieved on the whole dataset during the last epoch.
9. `Mean mu0_pred`: Mean $\mu_0(\boldsymbol{x})$ predicted by the first NN across the sampled population.
10. `Std mu0_pred`: Standard deviation of predicted $\mu_0(\boldsymbol{x})$ across the sampled population.
11. `Mean mu0_real`: Actual mean $\mu_0(\boldsymbol{x})$ across the sampled population.
12. `Std mu0_real`: Standard deviation of actual $\mu_0(\boldsymbol{x})$ across the sampled population.
13. `Mean tau_pred`: Mean $\tau(\boldsymbol{x})$ predicted by the first NN across the sampled population.
14. `Std tau_pred`: Standard deviation of predicted $\tau(\boldsymbol{x})$ across the sampled population.
15. `Mean_tau_real`: Actual mean $\tau(\boldsymbol{x})$ across the sampled population.

Summary file continued on next page...

16. `Std tau_real`: Standard deviation of $\tau(\boldsymbol{x})$ across the sampled population.
17. `tau_true_mean`: Mean τ of the entire population.
18. `Mean_prob_t_pred`: Mean propensity score, $p(\boldsymbol{x})$, predicted by the second NN across the sampled population.
19. `Mean prob_of_t_real`: Actual mean propensity score, $p(\boldsymbol{x})$, across the sampled population.
20. `mean_T_real`: Actual mean treatment, t, across the sampled population, $\mathbb{E}(t) = \sum_{i=1}^{N} t_i/N$.
21. `Y_real_mean`: Actual mean outcome, y, across the sampled population.
22. `Y_pred_mean`: Predicted mean outcome, y, across the sampled population.
23. `CI_lower_bound`: Lower bound for the 95% confidence interval of the average treatment effect.
24. `CI_upper_bound`: Upper bound for the 95% confidence interval of the average treatment effect.
25. `psi_0_mean`: Mean influence function estimate, across the sampled population, of outcome, y, when not treated, $t = 0$.
26. `psi_1_mean`: Mean influence function estimate, across the sampled population, of outcome, y, when treated, $t = 1$.
27. `mean_diff_psi_1_psi_0`: Influence function estimate for average treatment effect.
28. `std_diff_psi_1_psi_0`: Standard deviation of influence function estimate for average treatment effect.
29. `in_interval`: Is `tau_true_mean` within the 95% confidence interval or not.
30. `model_parameters_dict`: Dictionary that stores all model parameters.

Software requirements

Below we specify a version of python and versions of libraries we used to run the code. If the code is run using other versions, slight modifications in the script may be necessary.

1. Python 3.6.8
2. numpy 1.16.4
3. scipy 1.2.0
4. tensorflow 1.15.0
5. matplotlib 3.0.3
6. pandas 0.24.2