Journal of Global Economic Analysis, Volume 11 (2026), No. 01, pp. 49-85.

Surrogate modelling of a recursive-dynamic single country computable general equilibrium model

BY WOLFGANG BRITZ[a] AND HUGO STORM[b]

We design, detail and test an approach which allows optimizing policy instruments to maximize welfare over results emerging from a Computable General Equilibrium (CGE) model. To technically enable policy optimization over a CGE model, we train a deep learning-based surrogate model that approximates the behavior of the CGE model. The final policy optimizing is then subject to the surrogate model. To show case our approach, we optimize the timing of emission abatement and a carbon tax recycling strategy for Tanzania from 2018 to 2050 with a detailed recursive-dynamic single-country CGE model. Our proposed approach and the provided code to train surrogate models for CGE models is quite generic, allowing its application to differently structured CGE models and associated policy instruments. Even though generation of the observation sample is computing time intensive, the surrogate model enables policy optimization not possible by using the CGE model directly. The trained neural network replicates the simulation behavior of the CGE model quite accurately with an average R^2 of 99.99% over the outputs. Besides optimization, such a surrogate model representing the key input-output relations of a CGE model could also be easily integrated into other modelling frameworks.

JEL codes: C68, C88

Keywords: Surrogate Modeling; Neural Network; Policy Optimization; Deep Learning; Computable General Equilibrium

[a] Institute for Food and Resource Economics, University of Bonn, Nussallee 21, D-53115 Bonn, Germany (e-mail: Wolfgang.britz@ilr.uni-bonn.de).
[b] Institute for Food and Resource Economics, University of Bonn, Nussallee 21, D-53115 Bonn, Germany (e-mail: Hugo.Storm @ilr.uni-bonn.de).

1. Introduction

CGE models are a widely used tool to assess policies related to trade, climate change mitigation, and energy (Böhringer and Löschel, 2006). These models are benchmarked to replicate a snapshot of an economy, assuming that the observed transactions of the different aggregate agents are optimal ones given the prices, taxes, income flows, etc., they face. For a policy impact assessment, the analysis introduces changes in exogenous policy instruments to the model, such as emission taxes, potentially along with other drivers, such as technological progress or preference shifts. These changes disturb the benchmark equilibrium and let the model simulate a new one. It is characterized by market clearing for all commodity and primary factor markets and certain macro-economic balancing conditions. As behavioral equations are designed to be regular over the input space, agents continue to take optimal decisions. The new equilibrium comes about by adjustments in the quantities produced and demanded, driven by price changes, which jointly also imply changes in revenue flows. Assumed policy changes, can stem from planned legislation or literature analysis, see, for instance, Brown and Thérivel (2000).

This approach to policy assessment quantifies impacts of considered exogenous policy and other changes in endogenous variables of the CGE model and indicators derived from them, such as on prices, quantities, Gross Domestic Product (GDP), emissions, and welfare metrics. One might run a range of scenarios with different policy choices and select the best one using some selection criteria. But there is no guarantee that the considered scenarios comprise optimal policy changes. Therefore, previous authors have typically optimized welfare metrics over a limited number of tax rate changes, taking the equations of the CGE model as constraints. Such an optimization approach can inform "optimal" changes in tax rates and other policy instruments given the model structure and the optimized welfare metric. To date, this approach has been applied, to our knowledge, to single country CGE models, only, by André and Cardenete (2009) in a comparative-static setting and by Sánchez and Cicowiez (2022) in a fully-dynamic model over a five-year time horizon. Optimization over a long-term simulation horizon of multiple decades is likely to be numerically quite challenging and can additionally require larger changes to the model codes which are typically set-up to solve for a single year. For instance, updates to parameters in between solution points can depend on endogenous lagged variables.

We therefore propose, develop, and assess in this paper a different approach. To overcome the computational burden of optimizing over the full simulation horizon, we aim to leverage the flexibility, predictive power and functional approximation capabilities of modern machine learning approaches (Storm et al. 2019). Specifically, we first train a deep neural network (NN) as a surrogate model of the CGE model's behavior over multiple decades. Existing agricultural

economics literature has already showcased the usefulness of deep learning surrogate modeling for single farm level models (Shang et al., 2024). In our case of building a surrogate model for a CGE model the training set comprises model results under systematic changes in policy instruments. The resulting NN is far smaller compared to the CGE model it replicates, which eases policy optimization. To showcase the proposed approach we apply it to a case study in which we optimize simultaneously the timing of emission abatement and the carbon tax recycling strategy for Tanzania from 2018 to 2050. The example is interesting from a technical perspective due to the highly non-linear relations between larger sets of the inputs and outputs. This application of surrogate model training, and its use in policy optimization, is quite generic. We provide software code for our approach to facilitate the transfer of this approach to other applications. NN based surrogate models are not only useful tools for policy optimization but can also serve, for instance, as a compact but highly accurate reduced form representation of core input-output relations of a CGE model to be integrated into another simulation model.

The paper is organized as follows. We will (1) present the set-up of the single-country recursive-dynamic CGE model and the underlying data; (2) explain how the training set is constructed and (3) detail the process of estimating the NN. Afterwards, we present the results of the different steps, before we discuss and summarize.

2. Methods

2.1 CGE model set-up

We employ the MANAGE-WB model (Beyene et al., 2025) of the World Bank[1], implemented in General Algebraic Modelling System (GAMS).[2] The Social-Accounting Matrix for the chosen example of Tanzania is derived from the GTAP Version 11 power database (Aguiar et al., 2022), maintaining the full detail of 76 sectors. Multi-layer Constant- Elasticity-of-Substitution (CES) functions depict in detail the substitution possibilities between different inputs in the production functions. Household demand is modelled by a top-level Constant-Difference-in-Elasticities (CDE) demand system which depicts the allocation of available income to broader commodity groups. These are disaggregated into individual product demands via a multi-layer CES system. The parameters in the CDE demand systems are updated over time based on empirically estimated relationships to better capture dynamic evolution of the income elasticities of demand. In order to

[1] An open-source version is available from https://github.com/worldbank/MANAGE-WB.

[2] GAMS (see www.gams.com) is a widely used high level Algebraic Modelling Language for economic simulation models.

address the impact of carbon taxation on production and demand choices, both firms and final demands can substitute across different energy carriers, and firms can substitute capital for energy consumption to depict investments into energy saving technologies. Power generation is depicted by multiple electricity producing sectors (gas, coal, oil, hydro, solar, other) where the mix can be adjusted to some degree. Process-emission coefficients depend on carbon taxes based on Marginal Abatement Cost Curves (MACC); related costs are depicted by intermediate demand for a MACC commodity produced by a dedicated production activity. The set-up uses the default closures of the model.[3] Specifically, investments are savings driven.

The baseline takes projections of population and real GDP as given; they stem from the UN's population division and the World Bank's MFMod model. Real GDP is recovered by endogenous technical progress, with rates for primary, secondary, and tertiary sectors differentiated depending on the GDP growth rate. The baseline incorporates projections of global energy prices and some exogenous assumptions about changes in emission intensity of the livestock and water/waste sectors. The share of government consumption devoted to GDP is adjusted slightly over the simulation horizon to avoid exploding debt. Foreign savings relative to GDP depend on expected capital returns.

2.2 Policy application

The proposed combination of sensitivity experiments with an economic model that changes exogenous policy instruments such as tax rates, followed by an estimation of a surrogate model from the results of these experiments and the subsequent optimization of the policy instruments using the surrogate model is quite generic. We have therefore selected a showcase that is interesting from a surrogate modelling perspective. Specifically, it optimizes policy instruments for Tanzania related to climate change mitigation, adjusting the timing of mitigation efforts based on endogenous carbon taxes under yearly emission caps. This is combined with decisions about how to recycle the resulting emissions tax revenues. The complex interactions between future emissions caps, carbon tax levels and revenues, as well as the impacts on macro-economic variables, render this case interesting from a surrogate modelling perspective. The well-known limitations of using a recursive-dynamic model for intertemporal optimization are addressed in the discussion section.

The motivation to optimize the timing of mitigation efforts in our application is that cumulative emissions drive climate change. Fixing emissions for a final year, only, in an international agreement can imply different climate impacts, depending on the timing of abatement. Climate mitigation agreements should instead target cumulative emissions, letting participants decide on the timing of

[3] See page 53 in Beyene et al. (2025).

mitigation efforts. This perspective is fundamentally different from global studies such as Nordhaus (1992) that optimize jointly annual emissions and saving rates such to maximize economic growth, implicitly equilibrating the cost of marginal climate change damages and of emission abatement.

The so-called double dividend hypothesis (Goulder, 1995) states that environmental taxation can come along with economic benefits if the generated tax revenues allow for economic reforms that might otherwise not be feasible, such as reductions of distortionary taxes (tax recycling). These revenues can also be used to dampen or offset adverse distributional impacts of the carbon taxes. It is therefore inviting to jointly optimize the tax recycling strategy and the timing of emissions. Specifically, we consider reducing, at varying rates, factor, sales, and direct taxes, with the option to use part of the additional tax revenue to reduce the government deficit such that not all tax revenue is recycled. This adds further complexity to the estimation of the surrogate model.

The experiments define annual emission caps, with the size of the yearly caps varying across experiments, depicting different timing of abatement but always meeting a predefined limit of cumulative emissions. This limit is defined by a *base* case where emissions are reduced by 1.5% per annum compared to a Business-as-Usual (*BaU*) baseline. Accordingly, in the *base* case the emission cap for 2030 is defined by $0.985^{13} = 82.2\%$ of the emissions in the *BaU* in 2030, dropping to around 60% in 2050. The yearly caps drive endogenous year-specific carbon taxes. Taxed emissions include all carbon dioxide (CO_2) and non-CO_2 emissions measured in CO_2 equivalents but exclude carbon stock changes from land cover change. Due to population and economic growth, the *BaU* emissions increase strongly, so that the capped emissions in 2050 under the *base* case are still above the 2017 benchmark level.

In the experiments, we draw yearly reduction rates randomly first for four anchor years (2018, 2030, 2040, 2050). The reduction rates for the years in between these four anchor points are linearly interpolated. For instance, with randomly drawn reduction rates of 2% in 2018 and 3% in 2030, the additional reduction in 2026 compared to 2025 will be 2.5%. The interpolation ensures a continuous reduction in emissions over time compared to the *BaU*. Instead, randomly drawing reductions independent for all years would result in jumpy time series of caps which is an unrealistic policy. The cumulated annual reduction rates for each year applied to the *BaU* emissions of this year define the time series of emissions caps in each experiment. As the reduction rates for the four anchor years are randomly drawn, their sum over time will differ from the targeted cumulative emissions defined by the *base* case. To address this problem, we scale the yearly caps in a specific way to avoid the case where caps for certain years exceed the *BaU* emissions. Otherwise, this would lead to negative carbon taxes (i.e., carbon subsidies). The scaling approach implies that there will be periods where the caps

are below and above the *base* caps in each experiment. For details, refer to Appendix A.

2.3 Conducting the experiments

The experiments with different timing of abatement and recycling rates for taxes generate the observations on which the neural network is trained. The outputs or dependent variables are the time series of the sum of government and household consumption, entering the objective function, and the level of the investment in the last year, entering as a constraint. The inputs are the policy instruments over which we optimize, the time series of emission caps and the three recycling rates for sales, factor and direct taxes. The three recycling rates remain constant across the simulation period, hence with a simulation period of 33 years from 2018-2050 this gives 34 outputs and 36 inputs.

Figure 1 gives a technical overview. The mother process (*sensitivity.gms*), implemented in GAMS, generates the stochastic draws for the reduction rates in the four anchor years and the three tax recycling rates via a LHS (Latin-Hypercube Sampling) utility (*lhs.exe*, van der Mensbrugghe, 2023).[4] The LHS is designed to cover the relevant value ranges of these variables. As a specific stratified random sampling process, LHS can cover efficiently a multi-dimensional distribution with relatively small sample sizes (Iman and Conover, 1980). The GAMS mother process dispatches parallel GAMS child processes (*run.gms*) which each conduct their own counterfactual. Each child process interpolates first its specific random draws of the reduction rates for the four anchor years and scales the resulting yearly caps to generate cumulative emissions equal to the *base* case as described above. It then conducts a recursive-dynamic counterfactual model simulation against the *BaU* using the resulting time series of emission caps and the three tax recycling rates. Each child stores solely these 36 inputs and the results for the 34 outputs. The experiments are run in packages of 100 child processes.

[4] The sensitivity analysis is part of the MANAGE-WB code and steered by its Graphical User Interface, see Beyene et. al. (2025), pages 180-120.

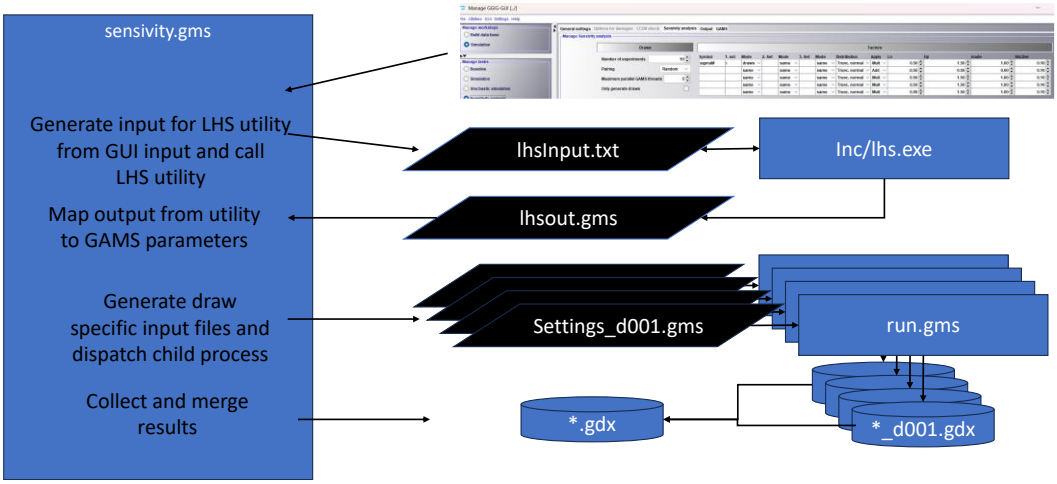


Figure 1: Schema of LHS based design of experiments in MANAGE-WB.

Source: Beyene et al. (2025), Figure 6.1

The mother process combines the inputs and the outputs of all experiments into one data frame and stores them to disk. It takes about 12 minutes on the chosen computer, a workstation with 24 cores based on Intel's i9 chips, to generate results for 100 draws. We run this process repeatedly to generate a large set of observations for training. It requires about 1Mb to store results for 1,000 draws. As new packages arrive, one can retrain the surrogate model and monitor the model's fit. This avoids generating more observations than required.

2.4 Estimating the surrogate model

We train a feed-forward deep NN with the observations on the 36 inputs and 34 outputs from the experiments. While the NN solely predicts the outputs from given level of inputs, any other variable of interest can then be determined by running a simulation with the CGE model using the optimized policy choices.

As the trained surrogate model needs to be coded in GAMS for the policy optimization, sigmoid activation functions are chosen in the hidden layers of the NN. These sigmoid functions are available in GAMS and have continuous derivatives, which eases the application of a gradient-based solver in the policy optimization. The number of neurons in the so-called input and output layers are given by the data set, but there is freedom in designing the rest of the NN. We opt for four hidden layers, one as usually fed by the inputs and three additional hidden ones which transform the output from the first over the second, third and fourth one. We later present results for different sizes for these layers. This allows us to quantify the trade-off between higher complexity of the Multilayer Perceptron (MLP) and related computational load and fit.

All inputs and outputs are normalized based on their minimal and maximal values found in the generated observations, such that all variables fall in the zero to unity range. Normalization increases numerical stability during training of the NN but also helps to define a useful overall loss metric, here chosen as the minimization of the mean squared error in sum over the outputs. To train the NN on the relevant ranges of inputs and outputs we have added several selection criteria:

- We exclude observations where the welfare metric to optimize is more than 1% below the base case.
- Additionally, reflecting a constraint during optimization, observations are excluded where investment in 2050 drops by more than 1% below the base case.
- Finally, we also remove observations where an output to be optimized is more than two times the distance between the second and first percentile below the first percentile, to avoid that minimum outcomes for the metrics to optimize have a significant impact on the results. For instance, if the first percentile for a specific year is at 1% and the second at 1.5%, twice their difference of 0.5%, i.e. 1% is subtracted from the first percentile to yield a cut-off limit of (1% - 2 * 0.5%) = 0%. Observations with negative changes for the sum of private and public consumption for this year would be excluded in this example.

Applying these criteria to the 10,000 observations required about 30 hours to be generated, and 5,700 observations remained for training purposes.

As we are maximizing an objective function that is strictly increasing in the outputs, removing observations with very low outputs should not exclude relevant information. This cleansing acts also as an outlier control, as unfortunate draws of the policy instruments can lead to large drops of public and private consumption or investment. Outliers have a disproportionate impact on the chosen loss metric and could result in a less precise approximation of the input/output relations for the relevant input ranges.

Details on the training process of the NN can be found in Appendix F where the Python code is discussed. The whole training takes about 15 minutes using around 5,700 observations (each obtained from one experiment in the CGE model). We determined the size of the training set by monitoring if additional observations further improve model fit. For the given size of the NN that was the case at around 5,700 observations.

3. Policy optimization

3.1 Implementing the Neural Network in GAMS

We recode the NN in GAMS, introducing the estimated NN parameters. The code is relatively simple given the structure of the NN (see first part of *nn_mng.gms* in the supplementary material). Let $d_{n,l}$ denote the drivers for neuron n in layer l, and $x_{i,l}$ the inputs feeding into it.[5] For the input layer, $x_{i,l}$ are the random variables from the experiments, for follow-up layers, these are the outcomes of the neurons of the preceding layer:

$$d_{n,l} = \sum_i w_{n,i,l}\, x_{i,l} + b_{n,l} \tag{1}$$

The weights w and biases b shown above are the parameters to be trained. The values $v_{n,l}$ for neuron n in layer l are a sigmoid function of its driver $d_{n,l}$, and $v_{n,l} = sigm(d_{n,l})$ are the outcomes of layer l which become the inputs in layer $l+1$. No sigmoid function is used for the output layer. The simple equations above with trained weights and biases and their transformation with a sigmoid function for the input and hidden layer are recoded in GAMS. The implementation in GAMS is quite generic for the case of a NN even with an arbitrary number of hidden layers.

In the Python code, the inputs and outputs are vectors, one for each observation, while in GAMS, they carry indices as labels, for instance to indicate the year. It is therefore useful to transform these output and input vectors in additional GAMS equations back with mapping sets into multi-dimensional variables, including a denormalization. This allows us to define the welfare function and additional constraints relating to policy variables (= inputs), indicators (= outputs) in the definition, and format used in the CGE model.

3.2 Adding the objective function and topical constraints

While the realization of the NN in GAMS is generic and can be used for different applications, code for the specific optimization must be added. In our application, the objective function maximizes changes in the sum of real household consumption *xc* and government consumption *xg*, normalized by the results of the baseline *BaU*, summed over all time points t = 2018, .. , 2050: [6]

$$\max \sum_t \frac{xc_{ts}^{opt} + xg_{ts}^{opt}}{xc_{bau}^{opt} + xg_{bau}^{opt}} \tag{2}$$

[5] We use a sigmoid activation function. For this case, drivers are sometime called logits.

[6] The details on the demand systems and the definition of the price indices used to define real consumption can be found in Beyene et al. (2025), page 30 and 31.

Due to the relatively high GDP growth in the underlying *BaU* without greenhouse gas (GHG) mitigation, the use of relative changes implies that absolute changes in later years receive much lower weights than early ones. Taking population growth into account, the implicit discount rate is around 3.5%/year. It reflects the fact that future generations benefit from capital accumulation and productivity gains. We are not converting the consumption quantities into utility as found in other studies (for instance, Nordhaus, 1992) as this requires choosing a functional form and parameters, which subsequently interact with the discount rate during optimization. One interpretation of our choice of using relative changes against the *BaU* is that this is equivalent to applying a logarithmic utility function without a discount rate.

We add a constraint to ensure that the sum of the *GHG* emissions over time is equal to the *base* case with its reduction of 1.5% per annum relative to the *BaU*:

$$\sum_t ghgs_t^{opt} = \sum_t ghgs_t^{bau}\ 0.985^t \tag{3}$$

The optimization over a finite horizon runs the risk of reducing investment, especially towards the end of the simulation horizon, to maximize current consumption, to the detriment of later generations. We therefore require that real investment in the final year 2050 cannot fall below the minimum of the levels of the *base* and *BaU* case:

$$xi_{2050}^{opt} \geq \min(xi_{2050}^{base}, xi_{2050}^{bau}) \tag{4}$$

Bounds on the policy variables restrict them to the minima and maxima in the experimental sample to avoid the surrogate model from being used outside the range it is trained for. Equally, constraints restrict the changes in GHG emissions to the maximum positive and negative changes found for each year in the experiments. Following the set-up of the NN, the optimization considers 36 endogenous policy instruments: emission caps for each year between 2018 and 2050, and the three tax recycling rates.

3.3 Interfacing

The different parts of the application are realized in their specific software environments such as GAMS, Python, or stand-alone programs. We opt for soft-linking between separate processes (generation and execution of experiments, collection of samples, NN training, policy optimization) where interfacing is based on files. The different processes are started manually rather than building an overarching application frame steering all processes. Interfaces between the processes are designed to be generic, i.e., independent of the inputs and outputs in the sampling and later policy optimization. The sensitivity analysis is part of the MANAGE-WB set of utilities and steered via its Graphical User Interface (GUI) realized in GGIG (Britz, 2014), either in interactive or batch mode. The random

variables (policy variables, behavioral or other model parameters) are inputted via a table in the GUI which defines, for instance, their lower and upper limits or the distribution to use (see also upper right part of Figure 1 above).

Once the set-up of the experiments is defined via the GUI, we switch to batch mode to automatically generate the packages of 100 model runs. As new packages arrive, the NN can be trained again to check if the fit improves. The LHS utility is directly called from the GAMS code which generates for each of the model runs its specific input file. This mother process deploys child processes in parallel and collects their results into one multi-dimensional parameters saved to disk, see Figure 2.

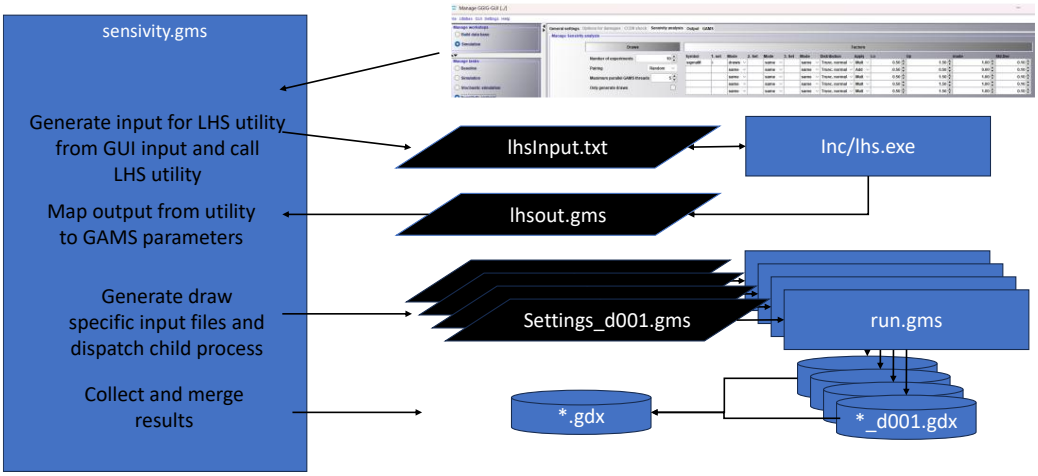


Figure 2: Schema of LHS based design of experiments in MANAGE-WB.

A second small GAMS script combines the packages into one data set (*collectScens_mng.gms,* see supplementary material), splits it into an input and output data set and stores these as zipped CSV files, after vectorizing the multi-dimensional parameters delivered by the CGE model (see Figure 3). This small script has no GUI as its sole parameter is the number of samples to combine. This GAMS program is repeatedly called while the sampling process in batch mode is running, to generate training sets of increasing sizes for the NN training. The zipped Comma-Seperated Values (CSV) files could be easily ported to another computer, for instance, to one with multiple Graphics Processing Units (GPUs) to speed up the training of the NN.

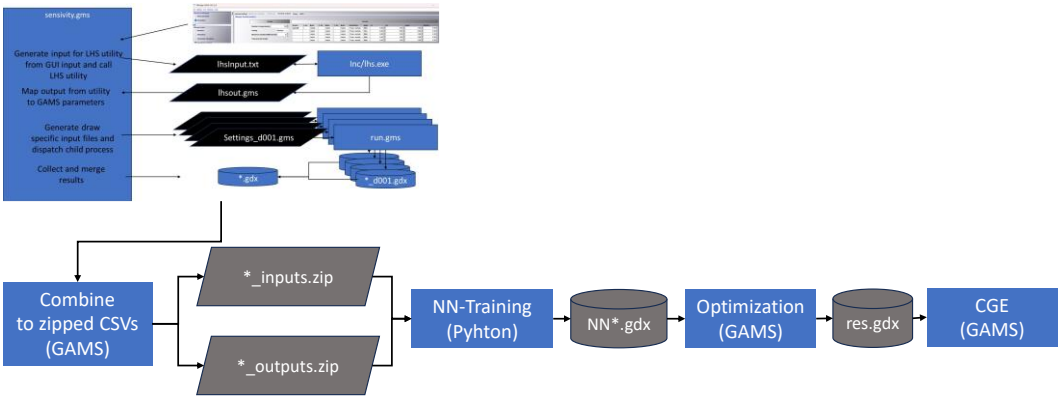


Figure 3: Processing chain of sampled experiments.

The Python code (*nn_torch_mp.py*, see supplementary material) trains the NN and retrieves the core information of the surrogate model to build from the data in the CSV files. The Python code defines the number of inputs and outputs from the data in the CSV files as key parameters of the NN's input and output layer. Due to the high-level Python packages such as *pytorch* used to train and evaluate the NN, the Python code is rather compact. As it stores and loads the trained parameters from earlier runs if available, adding new observations for training is efficient. The Python script generates one GDX container with the necessary input for the last step, the optimization in GAMS (*nn.gms_mng*, see supplementary material). This GDX container comprises the minimum and maximal values to de-normalize the inputs and outputs, along with the weights and biases of the layers. It also comprises list of inputs and outputs as GAMS sets. This allows us to generate the NN in GAMS in a generic fashion; clearly, context specific code must be added for an application which optimizes a specific metric. Command line parameters to the Python script permit changes in the number of epochs, the number of neurons in the hidden layer and other details of the process such that different layouts and training approaches can be tested without recoding. For details, refer to Appendix E.

The GAMS script (*nn_mng.gms,* see supplementary material) optimizes policy over the surrogate model and stores its results in a GDX container which is read in by a final run of the CGE model. This follow-up simulation with the CGE model serves two main purposes. First, it allows you to check if the simulated output from the surrogate model at the optimal point fits the simulation response of the CGE. Second, it provides all other outputs of interest to the analyst.

While the above production chain is to a large degree automated, a first-time installation requires the user to install Python with the necessary packages (see Appendix C). Users of a CGE model will need to have the necessary software installed for model runs and optimization, such as GAMS or GEMPACK.

We trained different NNs with four hidden layers of equal size to find an optimal layout for the NN, with 10 to 140 neurons in steps of ten. Each training uses a sequence of steps with different solver settings. Due to stochastics built into the solver, the first step is started multiple times for a limited number of iterations from which the best one serves as the starting point for further processing. Additionally, each step uses multiple processes in parallel. Details are discussed in Appendix E.

4. Results

4.1 Surrogate model fit

Before moving to the results of the policy optimization, we first present results for the surrogate model's fit. Despite our efforts to reduce the impact of stochastics on the training process, we found sizeable differences between repeated runs. We therefore repeated the training for each NN five times. Figure 4 reports the minimum, maximum, and average fit over these five trials for the different sized NNs. Already, the simplest layout with 10 neurons in each hidden layer achieved an average R^2 of 99.8% and a minimal R^2 of 96.6% over the five trials. The minimal R^2 increased to 99.91% for a network with 140 neurons in the hidden layers. For the best case of the five trials with 140 neurons, a minimum R^2 of 99.95% was achieved. The average fit for this best case was 99.9924%. Figure 4 below shows a relatively steep increase in the minimal R^2 up to 60 neurons, while the average R^2 approaches unity. Further progress when adding neurons is quite limited and only visible from an improvement in the minimal R^2. In some cases, such as with 90 or 120 neurons, the worst of the five trials delivered even lower fits than the worst trials of simpler networks.

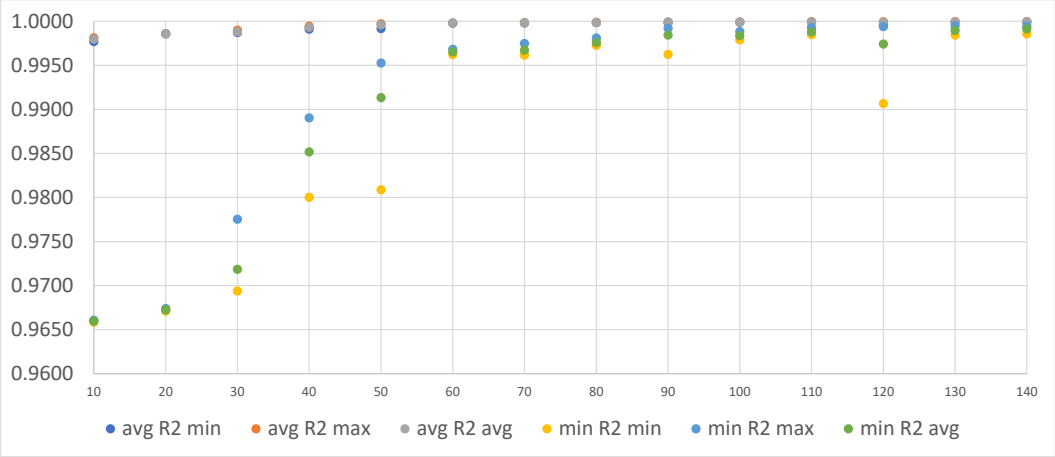


Figure 4: Min, max and average over five trainings for differently sized MLPs, of the average R^2 over all outputs and of the output with the lowest R^2.

4.2 Impacts of carbon taxation and optimized policy choices

This section presents results of the policy optimization. The first order negative impact on the consumption quantities from carbon taxation stems from increases in the price indices for private and government consumption. With unchanged revenues, this implies lower consumption quantities. In contrast to, for instance, Britz and Roson (2019), the saving rate of the household in MANAGE-WB does not depend on changes in demography and income levels, but is updated as a function of the consumer price index and a price index composed of expected returns to savings and yields on government bonds. For details, see Beyene et al. (2025, p. 20 and 25). The increase in the consumer price index from carbon taxes thus reduces the saving rate. Government consumption is fixed relative to real GDP in our application. If the price index for government consumption is increasing, budgetary outlays for consumption must increase. At unchanged tax revenues, this implies higher borrowing needs. Both the decreased saving rate and the higher borrowing needs reduce the amount of saving available to finance investment, implying reduced capital accumulation. This negative impact is amplified by price increases for investment goods from carbon taxation.

In this setup, tax recycling can directly dampen the impact on consumption by lowering sales taxes to offset the impact of carbon taxation on the price indices for household and governments. As the reduction encompasses sales taxes for intermediate consumption, pass-through effects of carbon taxes along supply chains are also dampened, reducing per unit production costs compared to a case without tax recycling. Since the saving rate reacts to changes in the consumer price index, its dampening due to lower sales taxes stabilizes the saving rate. These

impacts explain why it is less efficient to lower direct or factor taxes as they would not directly affect the saving rate.

Figure 5 highlights how the optimal choices for the timing of abatement interact with the tax recycling strategy. If solely tax rates are optimized while the timing of abatement is equal to the *base* case, shown in middle bar, the taxes are fully recycled. About 12% are used in this case to reduce factor taxes, the remainder reduce sales taxes. In the combined case where the timing of abatement and the tax rates are jointly optimized, shown in the last bar, about 6% of the tax revenue covers borrowing needs of the government and the remainder is almost entirely used to reduce sales taxes.

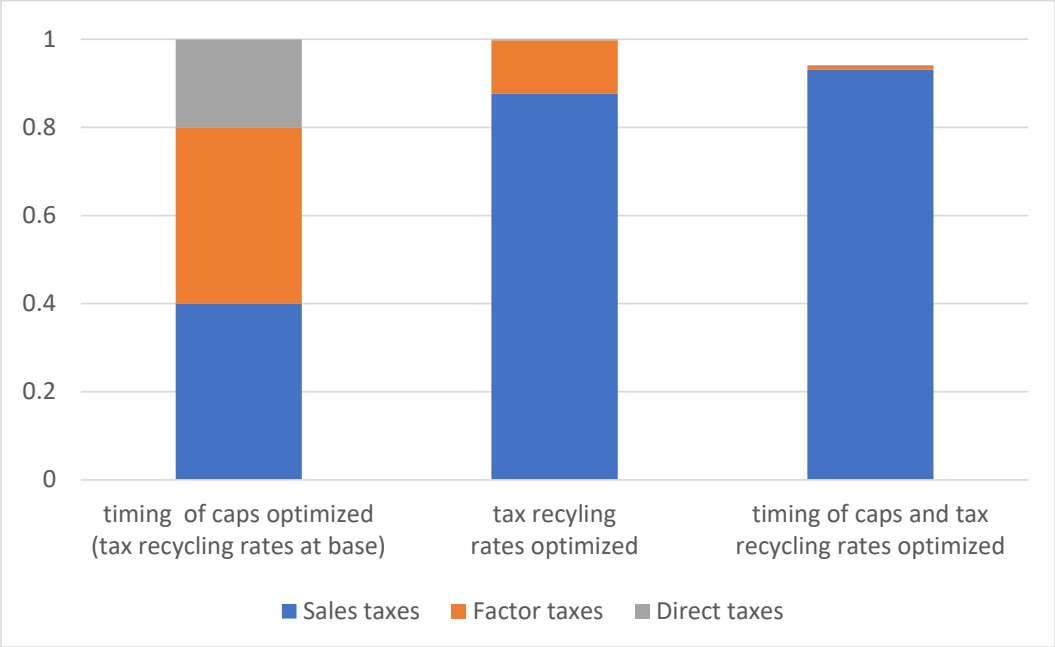


Figure 5: Shares of carbon tax revenue recycled to lower different sales taxes, factor taxes and direct taxes.

Figure 6 depicts the resulting changes in the metric being optimized. It highlights that the tax recycling strategy has a larger impact on increasing consumption in this example. But already the tax recycling rates chosen for the base case allow for some small welfare gains. Attributing these gains to the introduction of carbon taxes highlights again the importance of the assumptions underlying the double dividend hypothesis.

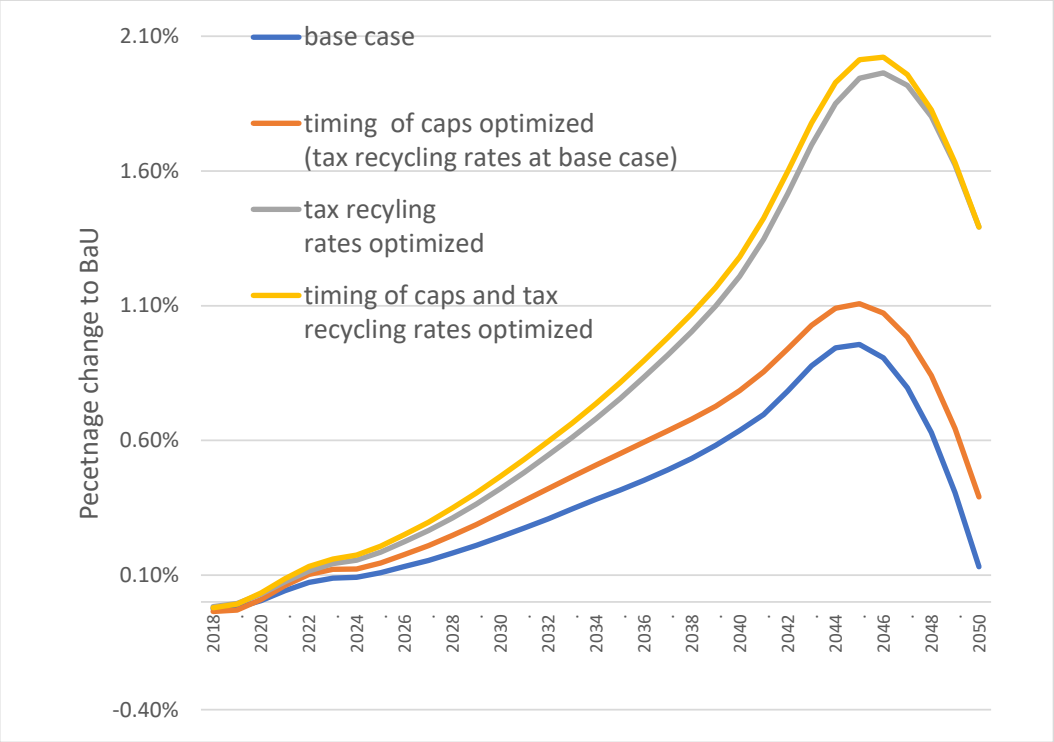


Figure 6: Sum of real private and public consumption compared to BaU [%].

Figure 7 shows the paths of emission reductions for the base case and the optimized ones. Because the share of hard to abate agricultural process emissions decreases over time while related per unit emission factors drop, abating less at the beginning is beneficial. This effect is stronger if the tax recycling strategy is not optimized.

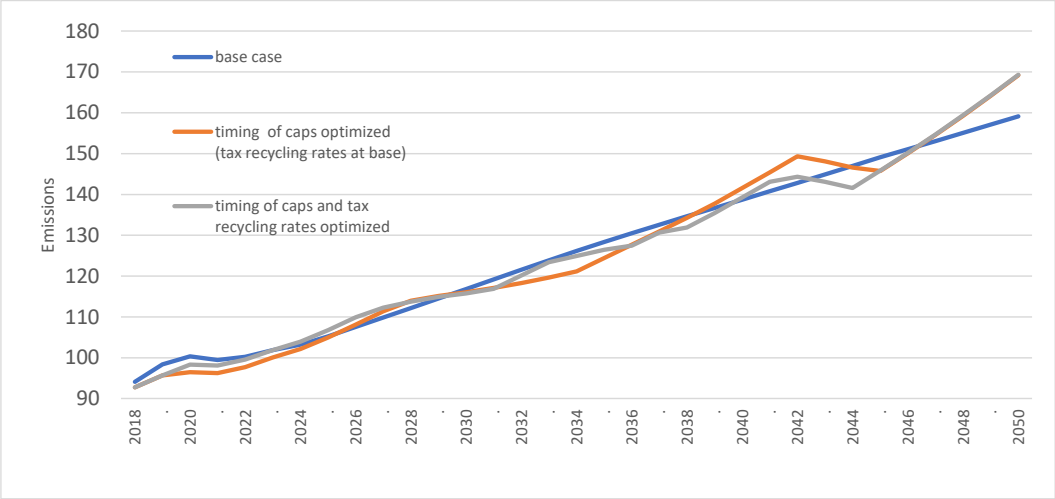


Figure 7: Emissions [Mil tons of CO_2 equivalents] in base case (bas), if solely the abatement timing is optimized (timing) and if both timing and tax recycling are optimized (both).

5. Discussion

5.1 Surrogate modelling

Surrogate models have been used widely in different disciplines, including the emulation of yield response to climate change from crop simulation models (Blanc, 2017); for a broader discussion see cf. Hussain et al. (2002) or Reis dos Santos and Reis dos Santos (2008). They aim to replace a complex simulation model with simpler mathematical functions to speed up computation (Kleijnen and Sargent, 2000). This can be clearly observed for our case: the NN can be easily implemented in GAMS or another software language and solves several magnitudes faster even compared to a single recursive-dynamic run of the CGE model. Similar to our application, surrogate models have been used for policy optimization; Carnevale et al. (2012), for instance, use a surrogate model to optimize air pollution policies. Their use in CGE modelling is also not new, for instance, Britz and Hertel (2011) summarize the behavior of a detailed partial agri-food equilibrium model in a compact revenue function which is integrated in a global CGE model to improve its representation of agricultural supply. Lengers et al. (2014), Kuhn et al. (2019) and Pahmeyer et al. (2020) build surrogate models of detailed single-farm models based on Mixed Linear Programming based on regressions. Employing a comparable approach, Seidel and Britz (2019) estimate a dual value function as a surrogate model of the same single farm model for later integration in an Agent-Based Model. Like our simplest NN, they find a relatively good fit already with a function with a limited number of parameters to estimate, even for a model with

non-continuous derivatives. However, as some variables in this paper show larger error terms, Seidel et al. (2023) and Shang et al. (2024) replace the classical econometric techniques by neural networks, using observations generated by the same bio-economic model. As in our case, they achieve a very good fit. Surrogate models are also useful to quantify environmental impacts, such as with climate (Smith et al., 2024) or carbon emissions (Britz and Leip, 2009) which can then be incorporated into an economic model directly or used for post-model reporting.

In cases where the surrogate model serves as a means to find and analyze key relations in the underlying model, such as in Lengers et al., (2014), questions of consistency across multiple simulated outputs are of minor importance. The use of consistent systems of FOC conditions such as in Britz and Hertel (2011) and Seidel and Britz (2019) ensures that the surrogate model stays in line with key micro-economic assumptions. Such consistency might be required for integration in an overall modelling approach, for instance, to ensure that exhaustion conditions hold. An NN approach cannot guarantee this consistency. Therefore, additional adjustments to simulated values might be necessary for model integration. This issue is, however, not of relevance for our optimization approach.

Our set-up requires about 10,000 samples, equivalent to 30 hours of computing time, to arrive at a sample of 5,700 usable observations to train a surrogate model with a sufficiently accurate fit. However, training the NN is fast, requiring only 15-20 minutes. The required sample size for an appropriate fit for a CGE model will likely depend on the specific set-up, such as sectoral detail and the model configuration, for instance, if highly non-linear elements such as the MACC mechanism in our example are present. It is therefore recommended to follow our approach to retrain the NN on an expanded set of observations to check for a stable fit. In our tests, expanding the observation samples was at least as important, in achieving a very low loss, as increasing the number of trainable parameters. Our tests seem to indicate that a set-up where the hidden layers are about as large as the maximum of the number of inputs and outputs might be a good starting point to test different set-ups of the NN. If the fit of this first setup is deemed convincing, one might try to reduce the size and vice versa. For our model application, where the NN is used for three policy optimizations, only, the size of the NN is of limited relevance. In other applications, finding the lowest size of the NN which still guarantees an appropriate fit might be important for computational reasons. One could clearly automate the search for this, for example by expanding our Python script using a grid search over the number of neurons under a minimum fit requirement.

The training of a surrogate model of a CGE or other mechanistic simulation models has different properties than many other reported applications of NNs or classical econometric estimations. A correctly set-up CGE model that serves as the data generation process in our analysis will deliver deterministic results in response to a shock. There are no inbuilt random (unobserved) disturbances

beyond the feasibility tolerance of the solver. The results are also observed without any error. This also holds if changes in model parameters are included in the inputs. In classical econometrics as well as machine learning applications with real-world data, the key aim is to identify the signal along with moments of the noise. In our setting, there is no noise and overfitting in the classical sense is hardly possible. Indeed, adding observations systematically decreased the loss towards zero, which is not possible in the presence of noisy observations. The statistical properties of the packages drawn by LHS are quite similar, as this sampling strategy efficiently covers the multi-dimensional input space in each package (see also van der Mensbrugghe, 2023).

Hornik (1991) shows that an NN with an appropriate number of neurons is able to replicate any non-linear differentiable function to a desired accuracy. As such, it is solely a question of the necessary number of observations and the complexity of the NN to find an appropriate surrogate model. Our example opens a rather optimistic perspective for CGE modelling as we found that, even with a limited number of simulations, we were able to train an NN of high accuracy. Obviously, any change in the model set-up and parameters requires a re-estimation of the surrogate model. As this process is largely automated, this is mostly a question of computing time to run the experiments.

Besides optimization, there might be other cases where a surrogate model is useful, such as to include key relations between inputs and outputs of a CGE model in a larger coupled modelling system. Solving CGE models is mostly done in Algebraic Modelling Languages (AML) such as GAMS or GEMPACK whereas other modelling systems might use less specialized and more generic computer languages. In Earth System Modelling, for instance, FORTRAN is still relatively common. GAMS offers APIs for Java, C and Python. Even if the modelling system hosting a CGE Model is written in any of these languages, it might still be challenging to integrate it in an overarching application, for instance, due to licensing issues or non-memory based I/O. Once an NN as a surrogate model is trained, it can be recoded easily in any the above programming languages and many others. Its forward feeding character does not require a solver for prediction.

The alternative to policy optimization using a surrogate model is the direct optimization over the CGE model itself. This has been done in a limited number of studies for single-country CGE models (André and Cardenete, 2009; Sánchez and Cicowiez, 2022), considering changes in a few policy tax rates, only, and using models of far lower complexity compared to our example. When using GEMPACK or an MCP solver in GAMS, or with parameter updates outside the model between solution points, switching from solving the equation system representing the CGE model to policy optimization will also require quite fundamental changes to the model's code.

5.2 Application case: optimized mitigation path

While the focus of this paper is on surrogate modelling and not the application *per se*, some critical points of the example application merit discussion.

Studies have optimized the timing and level of future global GHG emissions, balancing the economic costs of abatement with the benefits of reduced climate change damages. The famous Nordhaus (1992) paper uses a highly stylized global one-sector growth model called DICE, in which a Cobb-Douglas production function employs a labor stock following population growth and a capital stock driven by capital accumulation. Output not used as investment translates into per capita consumption which generates utility in each year. The model uses empirically estimated relations to quantify emission concentrations in the atmosphere from economic output and to calculate from there climate change damages, defined as a relative reduction in global output. Finally, there is a function that defines how much it costs, measured relative to global output, to reduce GHG emissions. This set-up allows the user to optimize simultaneously the saving rate and level of GHG emissions by maximizing the discounted per-capita utility.

There are obvious differences between the Nordhaus model and our application case. First, we are not aiming at finding an optimal balance between the cost of climate change abatement and the impact of climate change damages. A single small country such as Tanzania cannot change global climate change and related damages. Rather, its commitment to reduce emissions in the context of an international agreement is likely motivated by fostering global efforts to reduce damages and by potentially gaining access to funds to support its adaptation to climate change. We assume here that the country's cumulative reduction efforts are predetermined by the agreement and therefore not subject to optimization.

A second difference is that, in DICE, the global saving rate is endogenously optimized. In MANAGE-WB, the saving rate at the benchmark for each agent is defined by the data. In our application, the aggregate household has a positive saving rate in absence of an enterprise account and government surpluses. The household endogenously adjusts its saving rate to changes in the consumer price index and the expected returns to saving instead of the inter-temporal optimization in DICE.

One might see a logical inconsistency between using the CGE model outcomes for inter-temporal optimization and its recursive-dynamic nature where aggregate agents in each period behave myopically.[7] In our view, assuming bounded rationality with regard to saving's behavior is not unreasonable, especially when compared to assuming both perfect foresight for all relevant prices and revenue

[7] MANAGE-WB comprises a still experimental extension with limited forward-looking behavior, see pages 58-59 in Beyene et al. (2025). Due to numerical challenges, this extension was not used in our large-scale sensitivity experiments.

flows over decades and fully rational inter-temporal decisions as required to optimize saving decisions as in DICE. One might also argue that investment choices can depend on expectations of future carbon tax levels which we don't consider. However, in our experiments, the carbon tax is not fixed *ex-ante* but instead the emissions are capped. Intertemporal optimal investment choices would require that investors have perfect foresight on carbon taxes emerging from future caps and how carbon taxes and tax recycling influence the variables of their investment decisions, such as output and input prices and future demands. Still, as investors know that carbon taxes will increase in the future under stricter caps, they might opt for technologies abating more emission than optimal under current carbon taxes. This is not reflected in our set-up.

Moreover, we follow the assumptions of the double-dividend hypothesis that reductions in in taxes and government deficits are limited by the emission tax revenue, as otherwise, other taxes besides carbon taxes would need to increase. Clearly, if increases of other taxes were considered, one could optimize the tax system independently of introducing or raising carbon prices. In direct contrast to DICE, where an aggregate agent sets an optimal saving rate, policy makers in our application can change the economy-wide saving rate via two channels, but only to a limited degree. First, they can directly influence the government's (negative) savings by not fully recycling carbon taxes and second, they can influence the saving behavior of other agents by changing tax rates as far as carbon tax recycling allows. Our optimization assumes that solely the policy maker is forward looking when deciding on the timing of the emissions and the tax recycling strategy, thereby having perfect foresight. Given these limitations with regard to intertemporal optimization, results are rather an example of what optimization with a surrogate model can deliver.

The layout and parameterization of MANAGE-WB simulate small positive outcomes on public and private consumption already in the base case where a default tax recycling strategy is used. This underlines the importance of the assumption of the double-dividend hypothesis in such studies. The work of the World Bank to develop strategies towards net-zero emissions for a larger set of low and middle-income countries, published in the so-called Climate Change Development Reports, hints at limited GDP losses in the single digit range if all abatement options, including carbon capture under very large emission reductions, are taken into account.[8] As GDP losses tend to increase more than proportionally with the emission reduction, slight welfare gains under GHG abatement of up to 60% of BaU emissions, as found in this paper, might be realistic, at least if they are gradually introduced over time and the carbon taxes are wisely recycled.

[8] https://www.worldbank.org/en/publication/country-climate-development-reports

One reason for the limited welfare impacts in our study is the assumption of comprehensive carbon taxation covering all relevant gases and emitting processes. Existing taxation schemes mostly target fossil fuel use and do not cover all sectors. These schemes are therefore likely to be more costly than in our study as they require both higher carbon prices to reach the same emission reduction and they concentrate the abatement in a few sectors. Drawing on the EU Emissions Trading System (ETS) as an example, the EU has recently decided to extend the coverage of the existing ETS and to introduce an additional one for transport and buildings.[9] This new scheme will work at the point of sale, different from the existing ETS for industrial emissions which works at the point of emission. For the new ETS, sellers of fossil fuels (gas stations etc.) will need to buy emission rights. This extension of market-based instruments to abate emissions fits our idea of a full coverage ETS. But clearly, comprehensive abatement in a low-income economy, encompassing sectors such as subsistence agriculture will be quite challenging.

6. Summary and conclusion

Surrogate modelling of an evolved economic simulation model such as the MANAGE recursive-dynamic single-country CGE model used here opens the door to new applications, such as integrating the core responses of a CGE model into an overarching modelling system or to optimize a social welfare function implicitly over the economy as depicted by the CGE model. We present here a generic framework implemented in GAMS and Python which performs the three major steps for such an optimization: (1) generating the training observations based on stratified random sampling of experiments which are simulated by the CGE model, (2) training a Multilayer Perceptron (MLP) from these data and (3) operationalizing the MLP in a software allowing for constrained optimization. As in other studies, we find that MLPs are able to deliver surrogate models with an extremely good fit. Their implementation, for instance in an algebraic modeling language such as GAMS, is rather straightforward and allows for very fast solution times. In the Appendix we provide code, which is quite generic, for the estimation of a MLP using given model outcomes, along with the implementation of the MLP in GAMS that should be easy to use for other applications. A good place to start would be to replicate our proof of concept example which allows for the optimization of the timing of emission abatement along with carbon tax recycling options, to limit welfare losses from emissions abatement.

[9] https://climate.ec.europa.eu/eu-action/eu-emissions-trading-system-eu-ets/what-eu-ets_en

Acknowledgements

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under project SFB 1502/1–2022 – project number 450058266 (DETECT) and by the DFG under Germany's Excellence Strategy - EXC 2070–390732324 (PhenoRob).

References

Aguiar, A., M. Chepeliev, E. Corong, and D. van der Mensbrugghe. 2022. "The global trade analysis project (GTAP) data base: Version 11". *Journal of Global Economic Analysis*, 7(2). https://doi.org/10.21642/JGEA.070201AF.

André, F. J., and M.A. Cardenete. 2009. "Designing efficient subsidy policies in a regional economy: a multicriteria decision-making (MCDM)-computable general equilibrium (CGE) approach". *Regional Studies*, 43(8): 1035-104. https://doi.org/10.1080/00343400802049062.

Beyene, L.M., W. Britz, M. Christensen, H. Dudu, and R. Galindev. 2025. "The MANAGE-WB Applied General Equilibrium Model of the World Bank: Model Documentation and User Guide (English)". *Policy Research Working Paper*; Washington, D.C. : World Bank Group. http://documents.worldbank.org/curated/en/099543511132534267.

Blanc, É. 2017. "Statistical emulators of maize, rice, soybean and wheat yields from global gridded crop models". *Agricultural and Forest Meteorology*, *236:* 145-161. https://doi.org/10.1016/j.agrformet.2016.12.022.

Böhringer, C., and A. Löschel. 2006. "Computable general equilibrium models for sustainability impact assessment: Status quo and prospects". *Ecological economics*, 60(1): 49-64. doi: 10.1016/j.ecolecon.2006.03.006.

Britz, W. 2014. "A new graphical user interface generator for economic models and its comparison to existing approaches". *German Journal of Agricultural Economics*, 63(4): 71-285. https://doi.org/10.52825/gjae.v63i4.1964.

Britz, W., and T.W. Hertel. 2011. "Impacts of EU biofuels directives on global markets and EU environmental quality: An integrated PE, global CGE analysis". *Agriculture, Ecosystems & Environment*, 142(1-2): 102-109. https://doi.org/10.1016/j.agee.2009.11.003.

Britz, W., and A. Leip. 2009. "Development of marginal emission factors for N losses from agricultural soils with the DNDC–CAPRI meta-model". *Agriculture, ecosystems & environment*, 133(3-4): 267-279. https://doi.org/10.1016/j.agee.2009.04.026.

Britz, W., and R. Roson. 2019. "G-RDEM: A GTAP-based recursive dynamic CGE model for long-term baseline generation and analysis". *Journal of Global Economic Analysis*, 4(1): 50-96. https://doi.org/10.21642/JGEA.040103AF.

Brown, A. L., and R. Thérivel. 2000. "Principles to guide the development of strategic environmental assessment methodology". *Impact Assessment and project appraisal*, 18(3): 183-189. https://doi.org/10.3152/147154600781767385.

Carnevale, C., G. Finzi, G. Guariso, E. Pisoni, and M. Volta. 2012. "Surrogate models to compute optimal air quality planning policies at a regional scale". *Environmental Modelling and Software*, 34: 44–50. https://doi.org/10.1016/j.envsoft.2011.04.007.

Goulder, L. H. 1995. "Environmental taxation and the double dividend: a reader's guide". *International tax and public finance*, 2(2): 157-183. https://doi.org/10.1007/BF00877495.

Hornik, K. 1991. "Approximation capabilities of multilayer feedforward networks". *Neural networks*, 4(2): 251-257. doi: 10.1016/0893-6080(91)90009-T.

Hornik, K., M. Stinchcombe, and H. White. 1989. "Multilayer feedforward networks are universal approximators." *Neural Networks*, 2: 359–366. https://doi.org/10.1016/0893-6080(89)90020-8.

Hussain, M.F., R.R. Barton, and S.B. Joshi. (2002). "Metamodeling: radial basis functions, versus polynomials". *European Journal of Operational Research*, 138(1): 142-154. https://doi.org/10.1016/S0377-2217(01)00076-5.

Iman, R. L. and W.J, Conover. 1980. "Small sample sensitivity analysis techniques for computer models with an application to risk assessment". *Communications in statistics-theory and methods*, 9(17): 1749-1842. https://doi.org/10.1080/03610928008827996.

Kleijnen, J.P.C., and R.G. Sargent. (2000). "A methodology for fitting and validating meta-models in simulation". *European Journal of Operational Research*, 120(1): 14–29. https://doi.org/10.1016/S0377-2217(98)00392-0.

Kuhn, T., D, Schäfer, K. Holm-Müller, and W. Britz. 2019. "On-farm compliance costs with the EU-Nitrates Directive: A modelling approach for specialized livestock production in northwest Germany". *Agricultural Systems*, 173: 233-243. https://doi.org/10.1016/j.agsy.2019.02.017.

Lengers, B., W. Britz, and K. Holm-Müller. 2014. "What drives marginal abatement costs of greenhouse gases on dairy farms? A meta-modelling approach." *Journal of Agricultural Economics*, 65(3): 579-599. https://doi.org/10.1111/1477-9552.12057.

Nordhaus, W.D. 1992. "An optimal transition path for controlling greenhouse gases." *Science*, 258(5086), 1315-1319. https://doi.org/10.1126/science.258.5086.1315.

Pahmeyer, C. and W. Britz. 2020. "Economic opportunities of using crossbreeding and sexing in Holstein dairy herds". *Journal of dairy science*, 103(9): 8218-8230. https://doi.org/10.3168/jds.2019-17354.

Reis dos Santos, M.I., and P.M. Reis dos Santos. 2008. "Sequential experimental designs for nonlinear regression metamodels in simulation". *Simulation*

Modelling Practice and Theory, 16(9): 1365-1378. https://doi.org/10.1016/j.simpat.2008.07.001.

Sánchez, M.V., and M. Cicowiez. 2022. "Optimising policies to achieve agricultural transformation objectives: an application for Ethiopia". *Journal of Applied Economics*, 25(1), 765-783. https://doi.org/10.1080/15140326.2022.2056407.

Seidel, C., and W. Britz. 2019. "Estimating a dual value function as a meta-model of a detailed dynamic mathematical programming model". *Bio-Based and Applied Economics Journal*, 8(1), 75-99. https://doi.org/10.13128/bae-8147.

Seidel, C., L. Shang, and W. Britz. 2023. "A critical assessment of neural networks as meta-model of a farm optimization model", Discussion Paper 2023:1, Institute for Food and Resource Economics, University of Bonn. DOI: 10.22004/ag.econ.338200.

Shang, L., J. Wang, D. Schäfer, T. Heckelei, J. Gall, F. Appel, and H. Storm. 2024. "Surrogate modelling of a detailed farm-level model using deep learning". *Journal of Agricultural Economics*, 75(1): 235-260. https://doi.org/10.1111/1477-9552.12543.

Smith, C., D.P. Cummins, H.B. Fredriksen, Z. Nicholls, M. Meinshausen, M. Allen, S. Jenkins, N. Leach, C. Mathison, and A-I. Partanen. 2024. "fair-calibrate v1. 4.1: calibration, constraining, and validation of the FaIR simple climate model for reliable future climate projections". *Geoscientific Model Development*, 17(23), 8569-8592. https://doi.org/10.5194/gmd-17-8569-2024.

Storm, H., K. Baylis, and T. Heckelei. 2019. "Machine Learning in Agricultural and Applied Economics." *European Review of Agricultural Economics*, 47(3): 849–92. https://doi.org/10.1093/erae/jbz033.

van der Mensbrugghe, D. (2023). A Latin Hypercube Sampling Utility: with an application to an Integrated Assessment Model. *Journal of Global Economic Analysis*, 8(1). https://doi.org/10.21642/JGEA.080102AF.

Appendix

Appendix A: Definition of emission caps in experiments

Table A.1. Hypothetical example showing how emissions caps are defined in base case and for experiments.

	BaU	Base			Experiment							Not used
	Emissions BaU	Reduction relative to BaU	Caps Base	Differences cap to BaU emissions	Drawn reduction factors	Interpolated reduction factor	Reduction relative to BaU	Caps before correction	Differences cap to BaU emissions	Corrected difference cap to BaU emissions	Caps after correction	Naive correction of caps
2018	100.00	0.99	98.50	-1.50	1.00	1.00	1.00	99.90	-0.10	-0.07	99.93	116.47
2019	102.00	0.97	98.96	-3.04		1.00	1.00	101.55	-0.45	-0.31	101.69	118.40
2020	104.04	0.96	99.43	-4.61		0.99	0.99	102.98	-1.06	-0.73	103.31	120.06
2021	106.12	0.94	99.90	-6.23		0.99	0.98	104.17	-1.95	-1.34	104.78	121.45
2022	108.24	0.93	100.36	-7.88		0.99	0.97	105.12	-3.12	-2.15	106.09	122.56
2023	110.41	0.91	100.84	-9.57		0.99	0.96	105.82	-4.59	-3.16	107.25	123.37
2024	112.62	0.90	101.31	-11.31		0.98	0.94	106.26	-6.35	-4.38	108.24	123.89
2025	114.87	0.89	101.79	-13.08		0.98	0.93	106.45	-8.42	-5.80	109.07	124.10
2026	117.17	0.87	102.27	-14.90		0.98	0.91	106.37	-10.80	-7.44	109.73	124.01
2027	119.51	0.86	102.75	-16.76		0.98	0.89	106.03	-13.48	-9.29	110.22	123.62
2028	121.90	0.85	103.23	-18.67		0.97	0.86	105.43	-16.47	-11.35	110.55	122.91
2029	124.34	0.83	103.71	-20.62		0.97	0.84	104.57	-19.77	-13.62	110.72	121.91
2030	126.82	0.82	104.20	-22.62	0.97	0.97	0.82	103.46	-23.37	-16.09	110.73	120.62
2031	129.36	0.81	104.69	-24.67		0.97	0.79	102.15	-27.21	-18.74	110.62	119.10
2032	131.95	0.80	105.18	-26.76		0.97	0.76	100.65	-31.30	-21.55	110.39	117.35
2033	134.59	0.79	105.68	-28.91		0.96	0.74	98.97	-35.62	-24.53	110.06	115.39
2034	137.28	0.77	106.17	-31.10		0.96	0.71	97.11	-40.17	-27.66	109.62	113.22
2035	140.02	0.76	106.67	-33.35		0.96	0.68	95.09	-44.93	-30.94	109.08	110.87
2036	142.82	0.75	107.17	-35.65		0.96	0.65	92.92	-49.90	-34.37	108.46	108.33
2037	145.68	0.74	107.68	-38.00		0.96	0.62	90.61	-55.07	-37.93	107.75	105.64
2038	148.59	0.73	108.18	-40.41		0.95	0.59	88.17	-60.43	-41.61	106.98	102.80
2039	151.57	0.72	108.69	-42.87		0.95	0.56	85.62	-65.95	-45.42	106.15	99.82
2040	154.60	0.71	109.20	-45.39	0.95	0.95	0.54	82.96	-71.64	-49.33	105.26	96.72
2041	157.69	0.70	109.72	-47.97		0.95	0.51	80.47	-77.22	-53.18	104.51	93.82
2042	160.84	0.69	110.23	-50.61		0.95	0.49	78.14	-82.70	-56.95	103.89	91.11
2043	164.06	0.68	110.75	-53.31		0.95	0.46	75.96	-88.10	-60.67	103.39	88.56
2044	167.34	0.66	111.27	-56.07		0.95	0.44	73.92	-93.43	-64.34	103.00	86.18
2045	170.69	0.65	111.79	-58.89		0.96	0.42	72.00	-98.69	-67.96	102.72	83.95
2046	174.10	0.65	112.32	-61.78		0.96	0.40	70.21	-103.89	-71.55	102.55	81.86
2047	177.58	0.64	112.85	-64.74		0.96	0.39	68.53	-109.05	-75.10	102.48	79.90
2048	181.14	0.63	113.38	-67.76		0.96	0.37	66.97	-114.17	-78.62	102.51	78.08
2049	184.76	0.62	113.91	-70.85		0.96	0.35	65.51	-119.25	-82.13	102.63	76.38
2050	188.45	0.61	114.45	-74.01	0.96	0.96	0.34	64.15	-124.31	-85.61	102.85	74.79
Cummulative	4611.16		3507.24	-1103.92				3008.20	-1602.95	-1103.92	3507.24	3507.24

The table above illustrates the scaling approach based on a hypothetical example. The first column, "Emissions BaU" assumes a 2% increase in emissions per annum, starting from 100 units in 2018, shown in the dark blue line in Figure A.1. below. The second column defines the *base* case, using annual emission reduction factors of 0.985^t. Multiplying these factors by the *BaU* emission defines the emission caps for the *base* case, which are shown in the fourth column "Caps base" and the dark orange line in Figure A.1. For the latter scaling, we define for each year the difference to the *BaU* emissions in the column "Differences cap to BaU emissions", i.e. the abated yearly emissions.

The next block of columns refers to a constructed experiment. The first column "Drawn reduction factors" shows the factors for the four anchor years (2018, 2030, 2040, 2050), only. They would be randomly drawn in each experiment based on LHS. By linear interpolation, we arrive at a time series of reduction factors ("Interpolated relative to BaU"), their cumulative products up to each year are shown in the column "Reduction relative to Bau". Multiplying them by the *BaU*

emission results in the time series of emission caps shown in "Caps before correction", the grey line in Figure A.1. Their sum is with around 3,008 units of emissions below the *base* case of around 3507 units, i.e. the cumulative abatement is too high.

A usual way to correct values such that their sum is equal to a given a target is the use of a multiplicative correction factor. The last column shows this "naïve" correction where the "caps before correction" are multiplied with a scaling factor defined from the relation of the cumulative emissions (3,507/3,008). While the resulting time series of caps leads as required to the targeted cumulative emissions of the base case, the light blue line in Figure A.1. shows that the resulting caps until 2028 would exceed the emissions in the *BaU*, implying negative carbon taxes.

This motivates us to use a more evolved scaling approach. It defines corrections of the yearly caps based on the sum of abatement emissions, i.e., the differences between the annual caps and the *BaU* emissions. For the targeted *base* case, this sum of abated emissions amounts to around 1,104 emission units versus 1,602 units for the experiment before correction. From these two sums, a correction factor is derived (1,104/1,602) which is applied to the abated emissions in each year ("Differences cap to BaU emissions" for the experiment). The resulting required absolute difference between the cap and *BaU* emission is shown in "Corrected difference cap to BaU emissions". Added to the "BaU emissions", it defines the time series of caps used in the simulation with the CGE model ("Caps after correction") for this experiment, shown in light orange in Figure A.1.

As the cumulative emission caps for any experiment are scaled to be equal to the *base* case, they can never be for all years above or below the base case. This implies that they have to intersect the base case one or multiple times, as shown also for our hypothetical example by the light orange line below. As the drawn reduction factors for the first two anchor years are smaller than the ones for the later ones, the resulting caps are above the *base* case, as seen from the light orange line in the beginning, here up to year 2037, requiring lower caps compared to the *base* case for the remaining years until 2050.

The graph also suggests that maximal differences in the timing of abatement across experiments are relatively muted as cumulative emissions are fixed and caps above the BaU are excluded.

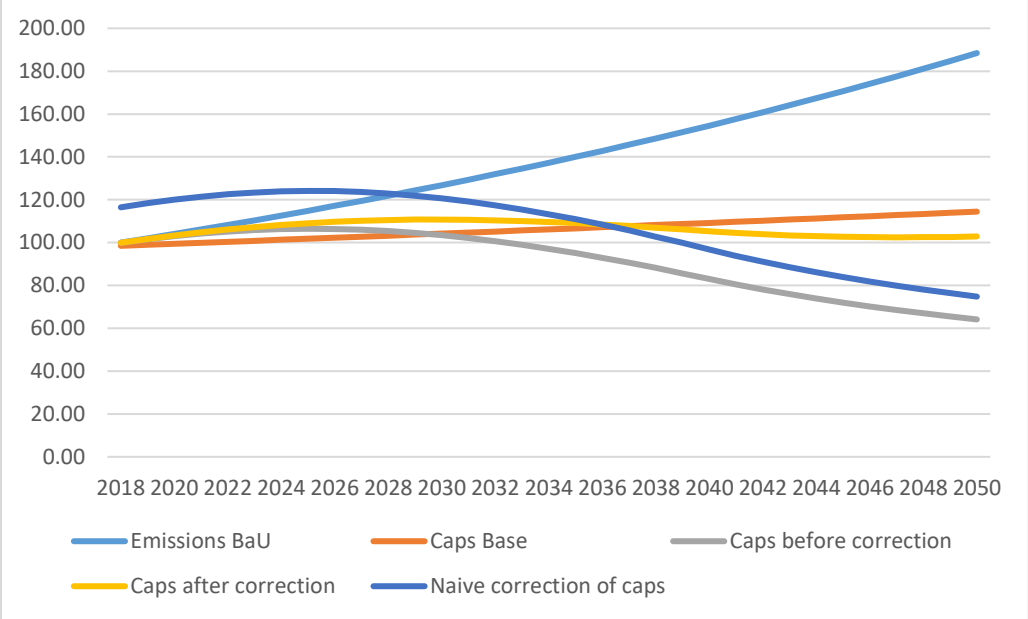


Figure A.1. Time series of emissions respectively emission caps for table above.

Appendix B: Folder structure

The supplementary material comprises a zipped file that comprises the Python code for optimization, the GAMS code for the optimization and the GAMS code used to combine the experiments from different packages of draws generated by the sensitivity analysis. The latter is there for information only. Table B.1 informs of the content of the different sub-folders.

Table B.1. Files and sub-folders used by the training and optimization.

Files	
nn_torch_mp.py	Python script for MLP training
nn_mgns.gms	Code for optimization over MLP
conopt4.opt	Option files for CONOPT4 solver used in optimization
collectScens.gms	Code to collect packages of sensitivity experiments, map the resulting observation samples in the vectorized format and stored them as zipped CSV files as input for the training. Provided for transparency.
runMLPTrain.txt	Batch files for use with MANAGE-WB to generate the DB, the BaU baseline, the baseline case and to run sensitivity analysis
Sub-folders	
obs	Location of the observations. Here, zipped CSV files with inputs and outputs should be placed, with different variables in the columns. The first column in both files must be called "draws". The python code will place there a file called "obsSample.h5" that comprise the observation in an internal data structure. Equally, the code will generate two Excel-sheets with core statistics (mean, standard deviation, min, max, percentiles) on the observations called "stats_raw.xsls" (as provided) and "stats_norm.xlsx" (after normalization). This can help to find, for instance, outliers in the data. The folder also comprises a GDX container containing (1) the mapping sets from the vectorized names of the inputs and outputs (TZA_GHGS_2030) to the three-dimensional structure used in the GAMS code for optimization ("TZA","GHGS","2030"). (2) the BaU required in the objective function, (3) the observations as samples in a 4-dimension array.
gdx	Folder with gdx files generated that describe the structure (number and length of input, output of hidden layers, etc.) and parameterization (biases and weights) of the trained NN. The GDX files also comprise the minima and maxima of the inputs and outputs required to map the normalized results back.
pt	Folder with NNs in internal structure. Used to restart from given trained network
png	png-graphics showing the development of the losses for the training and test set over the fitting steps.
tmp	Internal folder for temporary files
batch	Batch file to start the relevant steps in MANGE-WB. Not required for the training or the optimization. Thought as a documentation of the process or for conducting similar analysis for users familiar with MANAGE-WB.

Appendix C: Python installation

In order to train the network, a specific Python installation with a collection of packages is needed. Please follow the steps below to set-up the relevant Python environment on your machine:

- Download Anaconda (https://anaconda.org/)
- Generate a folder on your computer (such as NN_opt), open a Windows command prompt there and generate a new environment with Anacoda:
  conda create -n torch python=3.12
- Activate this environment
  conda activate torch
- Install the required core package for the NN training:
  pip3 install torch==2.9.1 torchvision==0.24.1
- Install the GAMS-Transfer API
  pip install gamsapi[transfer]==52.4.0
- Install other pakages used by the Python code (mainly for reporting):
  pip install matplotlib==3.10.7
  pip install torchsummary==1.5.1
  pip install scikit-learn==1.7.2
  pip install tabulate==0.9.0
  pip install openpyxl==3.1.5
  pip install tables=3.10.2

Appendix D: Generation of the training set

The supplementary material does not comprise the code of the CGE model MANAGE-WB, its Graphical User Interface and the GTAP Data base required to repeat the generation of the observations by large-scale sensitivity analysis. Instead, the results of the sensitivity analysis are made available (*see obs/mng_wb_os_ccdr.gdx*). Should a reader want to generate its own training sets with this or another model realized in GAMS, the code "*CollectScens_mng.gms*" might give useful hints.

Appendix E: Training the network

The results from the sensitivity analysis with MANAGE-WB stored in the two zipped CSV files "ccdr_inputs.zip" and "ccdr_outputs.zip" in the "obs" folder. The name "*ccdr*" is currently the default used by the Python code. You can place in these folder other observations to train your own network. To do so, generate two zipped CSV files with your inputs and outputs and replace the ccdr by your own name. This name is then passed to the python code with an argument, see argument scen in table.

In order to train the network, call:

```
python nn_torch_mp.py
```

Output to the Windows prompt informs the user about the training progress.

The network is trained in the Python package *pytorch* which supports the use of a GPU. GPUs can outperform the CPUs for specific matrix operations used during the training of a NN. Extremely large NNs are therefore mostly trained on GPU computing clusters. Tests by us on a computer with a single GPU speeded up training to some degree. The main effect was that it reduced the load on the CPU which allows to train and sample in parallel, if needed.[10] The gradient-based ADAM solver was applied; it uses stochastic perturbations. This solver is used with varying learning rates but otherwise default settings. After experimenting, we removed the dropout layers from the NN as there was no sign of overfitting observed while applying dropout decreases the training speed. Once a reasonable number of experiments (>1000) has been sampled, we trained the NN the first time.

The training uses by default 16 parallel threads that train each its own set of parameters. The threads are allocated to two groups. During iterations (called epochs), gradients are calculated on "batches" of observations. They are trained one after each other, using the optimized parameters from the previous one. The larger the batches, the more accurate but also time consuming is the estimation for each batch. For very large training sets, it might be impossible to load all observations in one single batch. As parameters are perturbed in each epoch and for each batch, having many small batches increases the chance for an improvement coming from the perturbations. The so-called "learning rate" determines the size of the stochastic perturbation. If it is chosen as too high, the solver might not be able to predict successfully gradients for improvements. If it is too low, the solver might more accurately estimate gradients, but given the high non-linearity and dimension of the solution space, the high precision might not come along with a faster reduction of the loss. These points motivate why the training process is here conducted in steps where these exogenous parameters are subsequently changed. In the beginning, when still far from an optimum, precise gradients are not so important and the process allows for higher perturbations combined with smaller batches, increasing the chance to randomly find a better

[10] Only specific types of GPUs are supported by the used Python packages; using the GPU(s) requires to first install additional third-party software. We therefore opted to use a computer without a GPU for training to ensure that our Python code can be easily used on standard Windows machines. We noted that the solver behavior was quite different when training on a GPU or CPU. The code therefore uses solver parameters in these two cases.

set of parameters. Once the loss has dropped, precise gradients become more important to make success such that batch sizes are increased and learning rates are decreased. This motivates a set-up of the training process in steps as documented in Table E.1.where the early steps use small batch size and higher learning rates compared to later ones. Starting with higher perturbation rates in the early step allows to quickly improve the fit as randomly different set-ups are explored. However, once a region is found which allows for a certain fit, reducing the learning rate can stabilize the directional search of the gradient-based loss minimizing algorithm. The documented settings were developed by experimenting with different choices and can be partially overwritten by the user by passing arguments to the Python script. It is important to note that the code is designed for surrogate model estimation where no stochastic errors are present and was so far tested only with outputs from non-linear constrained systems.

The number of iterations required to achieve a certain fit depends also on the size of the observation sample. Increasing the number of observations at unchanged number of epochs will explore more different parameter combinations, likely improving the loss compared to a smaller observation. This holds if the batch size is smaller than the number of observations.

If the process is started and the code finds a set of parameters for a MLP with a matching structure in the *pt* directory, i.e. the same number of layers and neurons in the layers, only the last training step is restarted. This allows to improve the fit without requiring to restart the whole process again with increased number of epochs. If there was still considerable process observed in the last step (>1% drop in fit), it might be worth to restart with a learning rate >0.0001 as used by default for restarts.

Table E.1. Parameters used in the different training steps

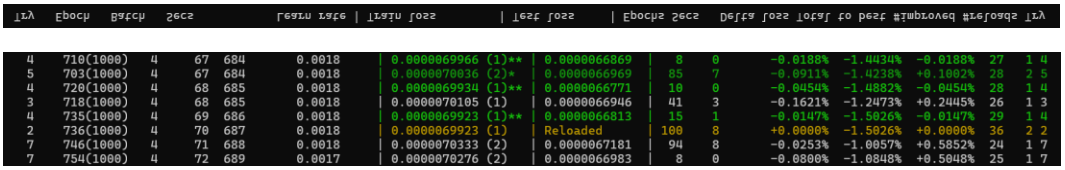
Step	Batch_size	Start learning rate[1]		End learning rate[1]		Epochs
		GPU	CPU	GPU	CPU	
0	64	0.04000	0.00400	0.01500	0.00200	1000[4]
1	128	0.01500	0.00200	0.01000	0.00100	1000[4]
2	256	0.01000	0.00100	0.00500	0.00050	1000[4]
3	512	0.00500	0.00050	0.00250	0.00025	1000[4]
4	1024	0.00250	0.00025	0.00150	0.00015	1000[4]
5	2048[2]	0.00150[3]	0.00010	0.00100[3]	0.00015	2000[5]

Notes: [1] All learning rates can be changed by the lr_mult argument. [2] Can be changed by the batch_size argument, [3] Can be increased by the learning_rate argument, [4] Can be changed by the epochs argument, [5] Can be changed by the epochs_last argument.

Table E.2. shows a screen shot for the output during the training process. If a thread improves the current best loss of its group, it stores its parameters to a group specific file. Such cases are shown in green. The number in brackets, such as (1), indicates the group for which an improvement was achieved. Light green cases, as also indicated by **, mean that a new overall optimum was found. Cases

in white show improvements for a single thread without improvement of the current best fit for its group. If a thread could not decrease the loss after the number of iterations, by default 100, is reloads the current best case for its group. Such cases are shown in yellow.

Table E.2. Screen shot of output during training iterations



Trd	Epoch	Batch	Secs	Learn rate	Train loss	Test loss	Epochs Secs	Delta loss Total to best #Improved #Reloads Trd
4	710(1000)	4	67 684	0.0018	0.0000069966 (1)**	0.0000066869	8 0	-0.0188% -1.4434% -0.0188% 27 1 4
5	703(1000)	4	67 684	0.0018	0.0000070036 (2)*	0.0000066969	85 7	-0.8911% -1.4238% +0.1002% 28 2 5
4	720(1000)	4	68 685	0.0018	0.0000069934 (1)**	0.0000066771	10 0	-0.8454% -1.4882% -0.0454% 28 1 4
3	718(1000)	4	68 685	0.0018	0.0000070105 (1)	0.0000066946	41 3	-0.1621% -1.2473% +0.2445% 26 1 3
4	735(1000)	4	69 686	0.0018	0.0000069923 (1)**	0.0000066813	15 1	-0.0147% -1.5026% -0.0147% 29 1 4
2	736(1000)	4	70 687	0.0018	0.0000069923 (1)	Reloaded	100 8	+0.0000% -1.5026% +0.0000% 36 2 2
7	746(1000)	4	71 688	0.0018	0.0000070333 (2)	0.0000067181	94 8	-0.0253% -1.0057% +0.5852% 24 1 7
7	754(1000)	4	72 689	0.0017	0.0000070276 (2)	0.0000066983	8 0	-0.0800% -1.0848% +0.5048% 25 1 7

The remaining columns inform about aspects of the training process. The first column lists the thread number for which an improvement or reload is reported. The second column shows the current iteration (epoch) and in brackets the total epochs in the step. The third column shows the current number of batches (i.e., sub-samples of the total observations) used, followed by the seconds spent in the current steps (column 4) and in total (column 5). The so-called learn rate (i.e. the random perturbation applied to the current parameters) is shown in the next column (6). Columns 7 & 8 report on the current loss for the training and tests sets. The number in brackets indicate the group the thread belongs to. The next columns show the number of epochs and seconds since the last improvement, followed by the delta in loss compared to the last successful iteration. The following column "to best" shows the difference to the current best candidate test of parameters. The last four columns show the number of epochs with improvements for this thread since the step was started, the number of reloads, and again the thread number. Observing these statistics can help to fine tune the training process by changing parameters relevant for the training.

After each training step, a table is outputted. It shows for each output the R^2, following by the label of the observation with the highest positive deviation and this deviation, followed by the label of the observation with the highest negative deviations and this deviation. The last three rows report the highest, average and lowest R^2 over all outputs.

Table E.3. Arguments for the Python code.

Argument	Short cut	Default	Explanation
anew	anew		Use to reload to build the training and test set. If no such sets are present, they will be built automatically from the two zipped CSV containers
alt_solvers	as		Switch to use ADAM, ADAMW, adaGrad and SGD solver each on 1/4 of threads, default is ADAM for all threads
batch_size	bs	2048	Batch size in last fitting setting.
cpu	cpu		Use CPU even if GPU with necessary software is found
dropout_rate	do	0	Switch to use ADAM,ADAMW,DadGrad and SGD on 1/4 of threads, default is ADAM
epochs	ep	1000	Number of epochs (iterations) used in each of the first five fitting steps.
epochs_last	epl	2000	Number of epochs (iterations) used in the last fitting steps. If a trained network is found, solely this last step is performed.
epochs_restart	epr	50	Number of epochs for restarts in first step
help	H		Show brief explanation of arguments
hidden1	h1	40	Neurons in first hidden layer. Size of input and output layers is defined from the data.
hidden2	h2	40	Neurons in second hidden layer. A zero removes the layer.
hidden3	h3	40	Neurons in third hidden layer. A zero removes the layer.
hidden4	h4	0	Neurons in fourth hidden layer. A zero removes the layer.
learning_rate	lr	0.0001	Learning in last fitting step.
lr_mult	lrm	1	Multiplier to learning rates in different steps.
nRestart	nr	10	Number of restarts to define starting point for first step
nProc	nt	8	Number of parallel threads conducting the training. Each thread maintains its own trained parameters.

(Continued . . .)

Table E.3. Arguments for the Python code. (. . . Continued)

Argument	Short cut	Default	Explanation
postfix	pf		Postfix appended on name of output files. The rest of the name is derived from the number of neurons in the hidden layers
reload	r		Take an existing MLP with a name without a prefix as a start point if existing. Allows to continue training from a given MLP without overwriting the existing results.
scen	s	ccdr	Prefix of zipped CSV container. The two files %scen%_inputs.zip and %scen%_outputs.zip have to be placed into the sub-directory "obs".
stall_iter	sit	100	If a thread has not reduced the loss after the number of iterations defined by stall_iter, it will restart the training with the current best set of parameters.
test_size	ts	0.001	Share of observations used for testing. Remaining observations are used for training.

Note: The defaults of the arguments are equal to the ones used in the training in the paper such that no arguments have to be passed to repeat the training process discussed here.

The arguments documented in Table E.3. allow the user to experiment, for instance, with differently sized NN. The code should be easily applicable to train any other surrogate model by storing the input data sets in two zip files in the "obs" directory. It was also used to train a NN on a multi-regional CGE with the GTAP standard model as its core, a related working paper is available at https://ideas.repec.org/p/ags/ubfred/355478.html.

The script stores the results in the sub-directory GDX, the GDX containers are named according to the structure of the neural network (number of neurons in the hidden layer, such as 80x80x80.gdx). To experiment with different settings of the training process but equally structured networks, the post fix argument can be used to avoid that previous results are overwritten.

Appendix F: Welfare optimization

The GAMS code for optimization *nn_mng.gms* reads the layout and parameters for the NN from the GDX container generated by the training step. To use a differently structured NN, change the name of the GDX file in line 20 *($setglobal NN*). The results are also outputted to a GDX file (*gdx/p_optres.gdx*) from which they could be retrieved, for instance, in a shock file. In order to run the optimization, call:

```
GAMS nn_mng
```