

Paris School of Economics

Michał Miktus, Mateusz Szmidt

Machine learning approach to trade flows estimation

Final project for Trade Policy
in
Analysis and Policy in Economics

March 2019

Contents

Introduction	5
1. Literature review	6
2. Data exploration	7
3. Neural network approach	10
4. Results	13
5. Concluding remarks	14
Bibliography	15
Appendix A	16
Appendix B	22

List of Figures

3.1. Activation functions 11

List of Tables

2.1. Your caption here	9
4.1. Results of neural network	13

Introduction

Since the pioneer work of [Tinbergen \(1962\)](#), the gravity equations has been widely implemented in the estimation of bilateral trade flows. The fundamental insight that the volume of trade between two countries is proportional to the product of an index of their economic sizes diminished by the measures of “trade resistance” between them has shaped the empirical specifications mainly due to the surprisingly good fit to the majority of data sets of both regional, as well as international trade flows. Over time the [Tinbergen \(1962\)](#) approach has been modified and enhanced, not to mention the supplementary theoretical underpinnings such as additional measures of trade resistance in spite of the classical ones (geographic distance, a dummy for common borders or dummies for Commonwealth memberships) or better estimation methods, allowing for the inclusion of zero-trade flows in the framework.

The following paper aims to implement the modern machine learning algorithms in the framework of gravity modeling in order to predict the bilateral trade flows. Machine learning can be viewed as an application of artificial intelligence (AI) which provides systems the ability to automatically learn and improve from experience without being explicitly programmed. In other words, machine learning focuses on the development of computer programs that can access data and use it learn for themselves, without human intervention or assistance, and adjust actions accordingly. The latest advancements in machine learning allowed to effortlessly identify patterns in data and use them to automatically make predictions or decisions. To the authors’ best knowledge, the following paper is the first try in implementing the above-mentioned framework to the trade policy analysis.

In addition, due to the familiarity of both authors to the Polish trade environment, the Poland trade relations has been chosen as a workhorse illustration. Obtained results prove that a neural network approach can be viewed as a grievous challenger to the classical estimation methods, such as Poisson Pseudo-Maximum Likelihood models or ordinary fixed panel data estimators.

The paper is organized as follows: the first chapter consists of the brief literature review, including the common gravity models and the estimation techniques, followed by the data characterization. Next sections provide a detailed description of the neural network approach enhanced by the hyper-parameters tuning and outline the main results. The paper is completed with the concluding remarks with potential extensions, references and appendices with codes in R and Python.

Chapter 1

Literature review

The traditional gravity model was developed in the 1960s to explain factory-to-consumer trade ([Tinbergen \(1962\)](#)). The above-mentioned concept was at the heart of the first clear microfoundations of the gravity equation – the seminal [Anderson \(1979\)](#), proposing a theoretical explanation of the gravity equation based on constant elasticity of substitution preferences of nations producing a single differentiated product. In parallel, the monopolistic competition versions were introduced ([Krugman \(1980\)](#), [Bergstrand \(1985\)](#)), followed by the work of [Anderson and Van Wincoop \(2003\)](#), expanding appropriate econometric techniques and introducing the microeconomic framework to the previously promoted monopolistic competition. Subsequent theoretical refinements have further focused on showing that the gravity equation can be derived from trade models with heterogeneous firms ([Helpman et al. \(2008\)](#)).

Simultaneously, the estimation techniques were progressing, starting from the basic least square estimator and its correspondent panel data version, meaning the fixed effect estimator. The endogeneity issues guided to the establishment of instrumental variables and two step least squares methodologies in the gravity models framework. Nevertheless, all the above-mentioned techniques led to the potentially biased results due to the requirement of elimination of the zero trade flows. Therefore, the Poisson Pseudo-Maximum Likelihood model, as well as zero-inflated negative binomial models were proposed and over the years became the flagship framework for the bilateral trade flows estimation.

HERE GOES THE POISSON SHORT DESCRIPTION AND ONE SENTENCE ABOUT FE

Chapter 2

Data exploration

For the first part of the data, namely the set of explanatory variables, the CEPII statistics were used, resulting in annual data of 60 variables at the cross country level. Then, using 3 digit ISO codes the dataset was joined with the trade flows information. Nevertheless, in contrary to the first, fully available online dataset, in order to obtain data on flows from Comtrade database, a data scrapper needed to be created. The authors expanded and modified the scrapping function delivered by Comtrade which in the end allowed to bypass all the limitations build into basic API and optimize the time of data scrapping. The exact code can be found in [Appendix A](#).

The variables used in the calculations, along with their descriptions, are presented in the following table.

The summary statistics:

Variable	Description
yr	Year
rt3ISO	Standard ISO code for reporting country (three letters)
pt3ISO	Standard ISO code for partner country (three letters)
contig	Dummy for contiguity
heg_d	Dummy if parter country is current or former hegemon of origin
col_fr	Dummy for reporting and partner countries colonial relationship post 1945
colony	Dummy for reporting and partner countries ever in colonial relationship
sibling	Dummy for reporting and partner countries ever in sibling relationship i.e. two colonies of the same empire
comleg_pretrans	Dummy if reporting and partner countries share common legal origins before transition
comleg_posttrans	Dummy if reporting and partner countries share common legal origins after transition
transition_legalchange	Dummy if common legal origin changed since transition
legold_d	Legal system of partner country before transition. This variable takes the values: fr for French, ge for German, sc for Scandinavian, so for Socialist and uk for British legal origin
legnew_d	Legal system of partner country after transition. This variable takes the values: fr for French, ge for German, sc for Scandinavian, so for Socialist and uk for British legal origin
gatt_d	Dummy if partner country is GATT/WTO member
fta_wto	Dummy for Regional Trade Agreement
eu_to_acp	Dummy for ACP country exporting to EC/EU member
gsp_o_d	Dummy if origin is donator in Generalized System of Preferences (GSP)
flaggsp_o_d	Report changes in Roses data on <gsp_o_d>. No gsp recorded in Rose; Data directly from Rose; Changes in data from Rose; Assumption that gsp continues after 1999
eu_o	Dummy if reporting country a member of the European Union
eu_d	Dummy if partner country a member of the European Union
Trade_value_total	Total value of trade between reporting and partner countries
distw	Weighted bilateral distance between reporting and partner countries in kilometer (population weighted)
pop_o	Population of reporting country total in million
pop_d	Population of partner country total in million
gdp_o	GDP of reporting country (current US\$)
gdp_d	GDP of partner country (current US\$)
gdpcap_o	GDP per capita of reporting country (current US\$)
gdpcap_d	GDP per capita of partner country (current US\$)
area_d	Area of partner country in sq. kilometers
tdiff	Time difference between reporting and partner countries in number of hours. For countries which stretch over more than one time zone the respective time zone is generated via the mean of all its time zones (for instance: Russia, Canada, USA)
comrelig	Religious proximity (Disdier and Mayer (2007)) is an index calculated by adding the products of the shares of Catholics, Protestants and Muslims in the exporting and importing countries. It is bounded between 0 and 1 and is maximum if the country pair has a religion which (1) comprises a vast majority of the population and (2) is the same in both countries.

Table 2.1: Your caption here

Variable	count	unique	top	freq	mean	std	min	25%	50%	75%	max	cova
gatt_o	4060.00	NaN	NaN	NaN	1.00	0.00	1.00	1.00	1.00	1.00	1.00	0.00
area_o	4060.00	NaN	NaN	NaN	312685.00	0.00	312685.00	312685.00	312685.00	312685.00	312685.00	0.00
yr	4060.00	NaN	NaN	NaN	2004.49	6.31	1994.00	1995.00	2005.00	2010.00	2015.00	0.00
pop_o	4060.00	NaN	NaN	NaN	38.33	0.22	38.00	38.15	38.23	38.54	38.66	0.01
gdp_o	4060.00	NaN	NaN	NaN	3.19e+11	1.55e+11	1.09e+11*	1.72e+11*	3.04e+11	4.77e+11	5.45e+11	0.48
gdpcap_o	4060.00	NaN	NaN	NaN	8339.74	4052.41	2819.70	4483.24	7976.12	12554.55	14341.86	0.49
gatt_d	4060.00	NaN	NaN	NaN	0.75	0.43	0.00	1.00	1.00	1.00	1.00	0.57
distw	4060.00	NaN	NaN	NaN	6140.89	3899.61	387.07	2603.11	5845.77	8583.18	17653.91	0.64
eu_o	4060.00	NaN	NaN	NaN	0.55	0.50	0.00	0.00	1.00	1.00	1.00	0.91
tdiff	4060.00	NaN	NaN	NaN	3.38	3.12	0.00	1.00	2.00	6.00	12.00	0.92
comrelig	4060.00	NaN	NaN	NaN	0.25	0.28	0.00	0.01	0.11	0.45	0.79	1.15
gdpcap_d	4060.00	NaN	NaN	NaN	10410.33	16066.97	64.81	864.06	3223.29	13299.54	116612.88	1.54
gsp_o_d	4060.00	NaN	NaN	NaN	0.24	0.43	0.00	0.00	0.00	0.00	1.00	1.80
fta_wto	4060.00	NaN	NaN	NaN	0.23	0.42	0.00	0.00	0.00	0.00	1.00	1.84
eu_to_acp	4060.00	NaN	NaN	NaN	0.21	0.40	0.00	0.00	0.00	0.00	1.00	1.96
comleg_pretrans	4060.00	NaN	NaN	NaN	0.17	0.38	0.00	0.00	0.00	0.00	1.00	2.20
transition_legalchange	4060.00	NaN	NaN	NaN	0.13	0.34	0.00	0.00	0.00	0.00	1.00	2.55
area_d	4060.00	NaN	NaN	NaN	719321.80	1956801.49	25.00	25713.00	119902.00	547244.00	17075400.00	2.72
eu_d	4060.00	NaN	NaN	NaN	0.11	0.32	0.00	0.00	0.00	0.00	1.00	2.81
comleg_posttrans	4060.00	NaN	NaN	NaN	0.10	0.30	0.00	0.00	0.00	0.00	1.00	2.96
pop_d	4060.00	NaN	NaN	NaN	34.23	129.66	0.01	1.69	6.66	21.70	1371.22	3.79
gdp_d	4060.00	NaN	NaN	NaN	2.64e+11	1.13e+12	1.09e+07	3.16e+09	1.46e+10	1.04e+11	1.80e+13	4.30
Trade_value_total	4060.00	NaN	NaN	NaN	1.12e+09	5.30e+09	0.00	1.52e+06	2.34e+07	2.35e+08	1.03e+11	4.74
contig	4060.00	NaN	NaN	NaN	0.04	0.19	0.00	0.00	0.00	0.00	1.00	5.04
sibling	4060.00	NaN	NaN	NaN	0.03	0.16	0.00	0.00	0.00	0.00	1.00	6.02
colony	4060.00	NaN	NaN	NaN	0.01	0.07	0.00	0.00	0.00	0.00	1.00	13.55
col_fr	4060.00	NaN	NaN	NaN	0.01	0.07	0.00	0.00	0.00	0.00	1.00	13.55
heg_d	4060.00	NaN	NaN	NaN	0.01	0.07	0.00	0.00	0.00	0.00	1.00	13.55
rt3ISO	4060.00	1	POL	4060	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
pt3ISO	4060.00	190	ARE	22	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
iso2_o	4060.00	1	PL	4060	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
iso2_d	4060.00	190	BZ	22	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
comlang_off	4060.00	NaN	NaN	NaN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN
comlang_ethno	4060.00	NaN	NaN	NaN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN
comcol	4060.00	NaN	NaN	NaN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN
col45	4060.00	NaN	NaN	NaN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN
heg_o	4060.00	NaN	NaN	NaN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN
col_to	4060.00	NaN	NaN	NaN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN
curcol	4060.00	NaN	NaN	NaN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN
cursub	4060.00	NaN	NaN	NaN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN
comcur	4060.00	NaN	NaN	NaN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN
legold_o	4060.00	1	so	4060	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
legold_d	4060.00	5	fr	1759	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
legnew_o	4060.00	1	ge	4060	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
legnew_d	4060.00	5	fr	2153	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
acp_to_eu	4060.00	NaN	NaN	NaN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN
gsp_d_d	4060.00	NaN	NaN	NaN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	NaN
flagssp_o_d	4060.00	3	no gsp recorded	3099	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
flagssp_d_d	4060.00	1	in Rose no gsp recorded in Rose	4060	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Chapter 3

Neural network approach

The neural networks approach is a statistical framework allowing to find complex patterns of relations in the data. The intuition behind the above-mentioned concept is often compared to the way of how human nerve system functions. In a nutshell, it can be characterized as follows - in the first phase the external signal is received by receptors and transferred to the set of neurons. Then, during further stages, it is iteratively processed and passed to next set of neurons until the signal is finally decoded. The structure of the neural network model similarly compounds of 3 elements: the input layer of independent variables, set of "hidden layers" and finally the output layer with calculated results of a model. Given the structure, in each phase besides the last one, the values of nodes from former layer are affinely transformed and then nonlinear function is performed in order to obtain the values for each node of a new layer. The calculations are repeated until the last phase when the final value is accessed through a nonlinear function of affine product of nodes from previous layers. The aforementioned process, starting from an input data and aiming to compute the output, is called the *forward propagation* and can be seen as a function of coefficients coined within every single affine transformation taking place between all neighbouring layers.

As a result, the estimated trade flows from the neural network approach rely on finding the appropriate values of parameters under arbitrary selected structure of a model. Thus in the first stage, the values of the coefficients are randomly assigned and then the forward propagation is performed. Next, based on model's output and true values of the observable dependent variable, the arbitrary chosen loss function is calculated. It has to be underlined that due to the fact that the generated output is a result of forward propagation, the loss function can be also defined as a function of the same parameters. It allows to compute a derivative with respect to them and in the end, to recalibrate their values – such a process is called *backward propagation* and it is iteratively repeated together with forward propagation to minimize the loss function, optimizing the values of parameters.

Although the intuition and general process behind the estimation of neural network model were presented above, a plethora of aspects referred to depends on arbitrary chosen structure or so called *architecture of a model*. Therefore, some choices implemented in

the final, best suited to the data architecture of the model need to be elaborated.

Firstly, a number of hidden layers intuitively allows to approximate any continuous function more carefully, nevertheless adding any next layer is computationally costly. The charge born is strictly related to another element of a model's structure, namely the number of neurons in each layer. It has to be emphasised that the above-mentioned amount can be different depending on a layer but again bigger number directly translates into higher cost. Consequently, to take advantage of computer architecture and to optimize processing time, a power of 2 neurons in each layer were implemented, as suggested in the literature.

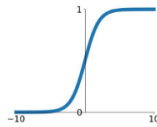
It has been already mentioned that each node is in fact defined as a function of the values of neurons from the former layer. It is thus beneficial to remark that it can be enforced that a node from hidden layer is a function of only a subset of nodes from a former one. Depending on the problem such an idea might be intuitive, not to mention the picture recognition, but it does not seem to be relevant in trade flows case. What is more, during the learning process such an exclusion of particular nodes may appear anyway, when the weights in affine transformations are relatively close to zero. Thus, the network with nodes being functions of all previous ones will be considered.

Moreover, the nonlinear transformation of a product of former nodes has to be defined. In the neural network framework, it is often called *an activation function*, aiming to activate the particular neuron on a hidden layer and assign to it some positive value when the particular pattern within a former nodes is observed. In a neural network literature, a particular set of functions can be observed, which by construction allows the model to be trained faster due to computational advantage while deriving derivatives and which satisfy the basic intuition behind activation. The most common ones are presented below.

Activation Functions

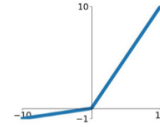
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



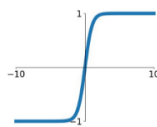
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

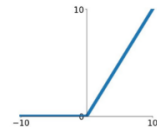


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

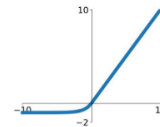


Figure 3.1: Activation functions¹

¹Source: <https://bit.ly/2uh7NyV>

The actually implemented in the end are sigmoid and relu. The first one was especially popular in the past, while the second one gained the popularity recently, outperforming the former with respect to the computational time.

At this stage, the part of hyper-parameters of models' structure directly connected to the forward propagation was covered. As far as the backward propagation choices are concerned, a loss function given the generated output has to be chosen. In the paper, the mean squared error was selected to validate the output. Moreover, in order to prevent the problem of overfitting, the regularization was implemented. The role of the aforementioned concept is simply to penalize the actual loss function of the model so that increase of the coefficients to some extent negatively affects the loss function. The value of a hyper-parameter of a penalty function identifies the size of marginal increase in a loss function alone to be compensated by the penalty.

Another regularization approach which can be implemented simultaneously is dropout. It serves to omit a fraction of randomly chosen nodes (along with their connections) on each layer while performing both forward and backward propagation. Thus, it enforces iterative deactivation of different neurons which diminish the pace of convergence but also stanches from overfitting the model. Nevertheless, it may simultaneously negatively influence the convergence, therefore a maximum number of iterations, in neural network framework called *epochs*, has to be specified. Choice of its value always brings a trade-off between a computational time and optimality of solution. One way (actually implemented in the paper) of meeting halfway is to set a threshold for marginal increase in loss function, ending the learning process sooner if the condition is fulfilled.

In fact, it is a learning process itself which determines the final performance of neural network framework. To fully define it, few more elements of a model's structure have to be recalled. As mentioned before, the estimation is based on calculating numerous derivatives with respect to all the parameters, according to the chain-rule, which in the end determines for a given set of parameters' values a point-gradient. However, as the neural networks tend to be defined over an enormous parameter space, the straightforward calculation of a gradient might be a complex task itself. Therefore, different optimization algorithms were implemented. The first one is Stochastic Gradient Decent (SGD), which calculates new iteration of parameters according to the specified learning rate, which is in turn another hyper-parameter of the model, defining the convergence speed. However, in standard SGD the learning rate is not scalable and it poorly handles updating the parameters of high variance. To deal with it, the second method is proposed, namely the Adam optimizer, which becomes gradually common recently. It allows to adjust the specified learning rate for each parameter and is often more efficient. Lastly, to speed up the whole process the hyper-parameter called batch size can be defined. The model chooses a subsample in a size of batch and performs an iteration using only selected observations. Therefore, it allows to train the model each time on different observation set and reduces the complexity of the whole process.

At this stage all the elements of model architecture are defined, allowing to implement the neural network on the presented grid of hyper-parameters and proceed with an estimation, with exact codes included in [Appendix B](#).

Chapter 4

Results

As far as the main outcomes from the trade flows prediction through a neural network approach are concerned, the best performing ten models are presented in the following **Results** Table.

Table 4.1: Results of neural network

N	N_iter	Val_loss	Val_MSE	Loss	MSE	LR	L1	L2
1	250	2,865	0,077	2,889	0,079	0,5	0,1	20,05
2	202	1,528	0,078	1,542	0,079	0,5	10075	10075
3	37	1,500	0,074	1,518	0,079	0,5	10075	10075
4	75	0,099	0,078	0,100	0,080	0,5	0,1	10075
5	201	1,567	0,078	1,598	0,080	0,5	10075	20,05
6	41	0,466	0,079	0,468	0,080	3125	0,1	0,1
7	174	0,107	0,079	0,108	0,080	0,5	0,1	30025
8	250	1,026	0,079	1,032	0,081	0,5	0,1	10075
9	129	0,677	0,080	0,679	0,081	0,5	0,1	0,1
10	65	0,379	0,080	0,381	0,081	1375	0,1	0,1

N	First	Hidden	Batch	Epochs	Dropout	Opt	Losses	Activation
1	4	1	64	250	0,000	Adam	MSE	relu
2	8	1	32	250	0,000	Adam	MSE	relu
3	4	1	32	250	0,000	Adam	MSE	relu
4	16	1	64	250	0,000	Adam	MSE	relu
5	4	1	32	250	0,000	Adam	MSE	relu
6	4	2	64	250	0,000	Adam	MSE	relu
7	4	2	32	250	0,000	Adam	MSE	relu
8	4	1	64	250	0,000	Adam	MSE	relu
9	8	2	32	250	0,000	Adam	MSE	relu
10	4	2	64	250	0,000	Adam	MSE	relu

Where columns denote respectively: *Upper*: position in ranking, number of iterations to converge, loss for validation set, MSE for validation set, loss for test set, MSE for test set, learning rate, L1 penalty, L2 penalty; *Lower*: position in ranking, first layer size, number of hidden layers, batch size, maximum number of epochs, dropout, optimizer, losses, activation function;

Chapter 5

Concluding remarks

Here goes the conclusion.

Bibliography

Cepii dataset. "<http://www.cepii.fr>".

Comtrade dataset. "<https://comtrade.un.org>".

James E Anderson. A theoretical foundation for the gravity equation. *The American Economic Review*, 69(1):106–116, 1979.

James E Anderson and Eric Van Wincoop. Gravity with gravitas: a solution to the border puzzle. *The American Economic Review*, 93(1):170–192, 2003.

Jeffrey H Bergstrand. The gravity equation in international trade: some microeconomic foundations and empirical evidence. *The review of economics and statistics*, pages 474–481, 1985.

Anne-Célia Disdier and Thierry Mayer. Je t’aime, moi non plus: Bilateral opinions and international trade. *European Journal of Political Economy*, 23(4):1140–1159, 2007.

Elhanan Helpman, Marc Melitz, and Yona Rubinstein. Estimating trade flows: Trading partners and trading volumes. *The Quarterly Journal of Economics*, 123(2):441–487, 2008.

Paul Krugman. Scale economies, product differentiation, and the pattern of trade. *The American Economic Review*, 70(5):950–959, 1980.

Jan Tinbergen. An analysis of world trade flows. *Shaping the world economy*, 3:1–117, 1962.

Appendix A

```
1 # Data Scrapper for comtrade.un.org
2 # Authors Michal Miktus & Mateusz Szmidt
3 # February 2019
4
5 # Environment setup
6
7 closeAllConnections()
8 library(rjson)
9 library(data.table)
10
11 # Defining all functions necessary to scrap the data
12
13
14 # Support function closing all connections (urls) opened during a scrapping
15 # process to avoid errors
16 #
17 # It uses a vector of connections defined at the beginning of each process
18 # and closes the opened ones when process ends
19
20 connections_dropper <- function(vector){
21   new_connections <- getAllConnections()
22   if(length(vector)<length(new_connections)){
23     connections_to_kill <- setdiff(new_connections, vector)
24     for(i in 1:length(connections_to_kill)){
25       con <- getConnection(i)
26       close(con)
27     }
28   }
29 }
30
31 # Support function Splitting numeric or string vector into vector of n-
32 # elements batches with "," separator
33 # It allows to lower the number of queries
34
35 vector_processing <- function(vector, n){
36   # We consider a case when the set size of a batch is greater than the
37   # length of vector
38   if(length(vector)> n){
```



```

39   list <- split(vector, cut(seq_along(vector), ceiling(length(vector)/n) ,
40     labels = F))
41   j = 1
42   vector <- c()
43   for(i in list){
44     subsample <- NULL
45     for(ii in i){
46       if(is.null(subsample)){
47         subsample <- paste(subsample, ii, sep="")
48       }
49       else subsample <- paste(subsample, ii, sep=",")
50     }
51
52     # Self check if the split was performed correctly
53     if(length(i) > n){
54       print("Something went wrong!")
55     }
56     vector[j] <- subsample
57     j = j + 1
58   }
59 }
60 else{
61   vector_ <- vector
62   vector <- c()
63   subsample <- NULL
64   for(i in 1:length(vector_)){
65     if(is.null(subsample)){
66       subsample <- paste(subsample, vector_[i], sep="")
67     }
68     else subsample <- paste(subsample, vector_[i], sep=",")
69   }
70   vector[1] <- subsample
71 }
72 return(vector)
73 }
74
75
76
77 # Basic data scrapper for a single query
78 # Default values of parameters adjusted to download annuall data on trade
79 # flows
80 # Source: https://comtrade.un.org/data/Doc/api/ex/r
81 get.Comtrade <- function(url="http://comtrade.un.org/api/get?"
82   ,maxrec=50000
83   ,type="C"
84   ,freq="A"
85   ,px="HS"
86   ,ps="now"
87   ,r
88   ,p
89   ,rg="all"

```

```

90         ,cc="TOTAL"
91         ,fmt="json"
92     )
93     {
94         string<- paste(url
95             , "max=", maxrec, "&" #maximum no. of records returned
96             , "type=", type, "&" #type of trade (c=commodities)
97             , "freq=", freq, "&" #frequency
98             , "px=", px, "&" #classification
99             , "ps=", ps, "&" #time period
100            , "r=", r, "&" #reporting area
101            , "p=", p, "&" #partner country
102            , "rg=", rg, "&" #trade flow
103            , "cc=", cc, "&" #classification code
104            , "fmt=", fmt #Format
105            , sep = " "
106        )
107
108        if (fmt == "csv") {
109            raw.data<- read.csv(string, header=TRUE)
110            return(list(validation=NULL, data=raw.data))
111        } else {
112            if (fmt == "json") {
113                raw.data<- fromJSON(file=string)
114                data<- raw.data$dataset
115                validation<- unlist(raw.data$validation, recursive=TRUE)
116                ndata<- NULL
117                if (length(data)> 0) {
118                    var.names<- names(data[[1]])
119                    data<- as.data.frame(t( sapply(data, rbind)))
120                    ndata<- NULL
121                    for (i in 1:ncol(data)){
122                        data[sapply(data[,i], is.null),i]<- NA
123                        ndata<- cbind(ndata, unlist(data[,i]))
124                    }
125                    ndata<- as.data.frame(ndata)
126                    colnames(ndata)<- var.names
127                }
128                return(list(validation=validation, data =ndata))
129            }
130        }
131    }
132
133
134    # Definining an object for an output of basic_scrapper function
135    output <- setRefClass("scrapper_output", fields = list(data="ANY", checked
136        = "ANY", hits = "ANY"))
137
138    # Function scrapping the data on all possible connections
139    # between countries defined in the input <vector> and all the partners
140    # available
141    # for the years defined as <year> .

```

```

141 #
142 # To control for the number of queries we use the parameter <hits>.
143 # It allows to stop the process after 100 hits to not exceed an hourly
    limit of 100 queries
144
145 basic_scrapper <- function(vector, year, hits){
146
147   # Console output and definition of an output object
148   print(paste("Trying for vector of", length(vector), "length."))
149   current_connections <- getAllConnections()
150   data <- NULL
151   checked <- NULL
152
153   # Looping over all batches of countries in a tryCatch block to avoid a
    failure of a process
154   for(i in 1:length(vector)){
155     tryCatch({
156       print(i)
157       out <- NULL
158       unit <- get.Comtrade(r=vector[i], p="all", ps=toString(year), freq="A
        ")
159
160       if(is.null(unit$data)){
161         checked <- rbind(checked, vector[i])
162         print(paste("No data available for year", year, "for", vector[i]))
163       }
164       else{
165         checked <- rbind(checked, vector[i])
166         out <- unit$data
167       }
168     },
169     error = function(e){
170       print(paste("Error for", i))
171     }
172   )
173
174   # Stopping the process for 1 hour after 100 hits
175   hits = hits + 1
176
177   if(hits >= 100){
178     Sys.sleep(3600)
179     hits = 0
180   }
181
182   # Output generation
183   data <- rbind(data, out)
184
185   # Dropping all connections opened during a process
186   connections_dropper(current_connections)
187 }
188
189 out <- output(data = data, checked = checked, hits = hits)
190 return(out)

```

```

191 }
192
193
194 # Main scrapping process using basic_scrapper function
195 # It splits the year range into batches of length 5 to optimize the number
    of queries.
196 # It also splits the list of countries into batches with initial length of
    5,
197 # the batches where the error occurred are joined and split again into
    batches of smaller size (up to 1).
198
199 main_scrapper <- function(main_vector, from, to){
200
201     # Definition of an output object and years range splitting into batches
        of 5
202     main_data <- NULL
203     years <- vector_processing(seq(from, to), 5)
204     hits = 0
205
206     # Looping over the years
207     for(i in 1:length(years)){
208         vector <- main_vector
209         cond <- TRUE
210         data <- NULL
211         try <- NULL
212         split <- 5
213
214         # Scrapping the data for all connections between the countries for a
            given batch of years
215         # It is continued until for none of a countries an error is reported
216         while(cond){
217             print(paste("Scrapping for years:", years[i]))
218             print(paste("The number of countries checked in one hit is", split))
219
220             unit <- basic_scrapper(vector, years[i], hits)
221             data <- rbind(data, unit$data)
222             hits <- unit$hits
223             try <- unique(rbind(unlist(try), unique(unlist(unit$checked))))
224
225             # Vector of countries for which error is reported and so the queries
                will be repeated
226             vector <- setdiff(unlist(strsplit(main_vector, "\\,")), unlist(
                strsplit(try, "\\,")))
227
228             # Checking if data for all countries is scrapped,
229             # then if not splitting the vector of countries into batches of
                smaller size.
230             if (length(vector) < 1){
231                 cond = FALSE
232             }
233             else{
234                 split <- max(split - 1, 1)
235                 vector <- vector_processing(vector, split)

```

```

236     }
237   }
238
239   # Overriding the state of the scrapping after each finished batch of
240   years
241   main_data <- rbind(main_data, data)
242   write.csv(file = "trade_data.csv", main_data)
243 }
244 return(main_data)
245 }
246 # Scrapping the data
247
248
249 # Scrapping the list of the countries listed in the comtrade database
250
251 download_reporters <- TRUE
252 if (download_reporters){
253   string <- "http://comtrade.un.org/data/cache/partnerAreas.json"
254   reporters <- fromJSON(file=string)
255   reporters <- as.data.frame(t(apply(reporters$results, rbind)))
256 }
257
258 # Adjusting the list of reporters for which the process works (removing "
259   world" and "all")
260 vector <- vector_processing(unlist(as.numeric(reporters$V1[3:length(
261   reporters$V1)])), 5)
262
263 # Data scrapping for the range of dates available in comtrade database
264 data <- main_scrapper(vector, 1962, 2018)
265 fwrite(file = "trade_data.csv", data)

```

Scraper.R

Appendix B

```
1 # Neural net created for the gravity model prediction for the Trade Policy
   class at PSE
2 # Author: Michal Miktus at michal.miktus@gmail.com
3 # Date: 21.02.2019
4
5 # Import libraries
6
7 import plotly.io as pio
8 import plotly.graph_objs as go
9 import plotly.plotly as py
10 from plotly.offline import init_notebook_mode, iplot
11 from matplotlib import pyplot as plt
12 from scipy.stats import mstats
13 from statsmodels.distributions.empirical_distribution import ECDF
14 from sklearn.preprocessing import MinMaxScaler, StandardScaler
15
16 import os
17 import numpy as np
18 import pandas as pd
19 #import torch
20 import seaborn as sns
21 import keras
22 import tensorflow as tf
23 import talos as ta
24 from keras.optimizers import Adam, Nadam, SGD
25 from keras.activations import relu, elu, sigmoid, tanh
26 from keras.losses import mse
27 from talos.model.normalizers import lr_normalizer
28 from talos.model.layers import hidden_layers
29 from talos.model.early_stopper import early_stopper
30 %matplotlib inline
31
32 # from plotly import tools
33
34
35 # Set seed
36
37 random_state = 123
38 np.random.seed(random_state)
39 tf.set_random_seed(random_state)
40 #torch.manual_seed(random_state)
```

```

41
42 # Supress scientific notation for pandas
43
44 pd.options.display.float_format = '{:.5f}'.format
45
46 # Templates for graphs
47
48 pio.templates.default = 'plotly_dark+presentation'
49 sns.set(style="ticks", context="talk")
50 plt.style.use("dark_background")
51 init_notebook_mode(connected=True)
52
53
54 # Path specification
55
56 #path = "/Users/miktus/Documents/PSE/Trade policy/Model/"
57 path = "C:/Repo/Trade/Trade-policy/"
58
59 # Import data
60
61 data = pd.read_csv(path + "/Data/final_data_trade.csv")
62
63 # Data exploration only for Poland
64
65 data = data.loc[data['rt3ISO'] == "POL"]
66
67 data.columns
68
69 # Number of trade partners
70
71 data["pt3ISO"].unique().shape
72
73 data.info()
74
75 # Dropping the duplicates from the dataset
76
77 data = data.drop_duplicates(keep='first')
78
79 # Handling missing data
80
81 data.isnull().sum()
82
83 data.dropna(thresh=data.shape[0] * 0.7, how='all', axis=1, inplace=True)
84
85 data.dropna(axis=0, inplace=True)
86 # data.fillna(data.mean(), inplace=True) # Or replace by the column mean
87
88 # Describe data
89
90 description = data.describe(include='all')
91 coef_variation = description.loc["std"] / description.loc["mean"]
92 description.loc["cova"] = coef_variation
93 (description.sort_values(by="cova", axis=1)).T

```

```

94
95
96 # Number of unique entries
97
98 print(data.nunique())
99
100 # Names of binary data (unstandardized)
101
102 binary = []
103 for columns in data:
104     if (data.loc[:, columns].min() == 0) & (data.loc[:, columns].max() ==
105         1):
106         binary.append(columns)
107
108 for columns in data.loc[:, binary]:
109     print(data.loc[:, binary][columns].unique())
110
111 # Remove iso_2o, iso_2d and family
112 data.drop(columns=['iso2_d', 'iso2_o'], inplace=True)
113
114 # Numeric variables
115
116 data_numeric = data._get_numeric_data()
117 data_numeric.drop(columns="yr", inplace=True)
118 data_numeric.drop(columns=binary, inplace=True)
119
120 # Visualisations
121
122 # Numerical data distribution
123 data_numeric.hist(figsize=(10, 10), bins=50, xlabelsize=8, ylabelsize=8)
124
125 for i, col in enumerate(data_numeric.columns):
126     plt.figure(i)
127     sns.distplot(data_numeric[col], color="y")
128
129 sns.distplot(data_numeric["tdiff"], color="y")
130
131 sns.pairplot(data_numeric);
132 sns.pairplot(data_numeric, vars=["pop_o", "tdiff"]) # kind="reg"/kind="kde"
133
134
135
136 # Flows
137 flows = data[['yr', 'rt3ISO', 'pt3ISO', 'Trade_value_total']]
138 data_loc = pd.read_csv(path + "/Data/CountryLatLong.csv")
139 data_loc.drop(columns=['Country'], inplace=True)
140 data_loc.columns = ["CODE", "rt_Lat", "rt_Long"]
141
142 flows = pd.merge(flows, data_loc, left_on="rt3ISO", right_on="CODE").drop('
143 CODE', axis=1)
144 data_loc.columns = ["CODE", "pt_Lat", "pt_Long"]

```



```

144 flows = pd.merge(flows, data_loc, left_on="pt3ISO", right_on="CODE").drop('
CODE', axis=1)
145
146 flow_directions = []
147 for i in range( len( flows ) ):
148     flow_directions.append(
149         dict(
150             type = 'scattergeo',
151             locationmode = 'ISO-3',
152             lon = [ flows['rt_Long'][i], flows['pt_Long'][i]],
153             lat = [ flows['rt_Lat'][i], flows['pt_Lat'][i]],
154             text = flows['pt3ISO'][i],
155             mode = 'lines',
156             line = dict(
157                 width = flows['Trade_value_total'][i]*10,
158                 color = 'red',
159             ),
160             opacity = 200 * np.power(float(flows['yr'][i]) - float(flows['
yr'].min()), 2)/float(np.power(float(flows['yr'].max()), 2)),
161         )
162     )
163
164 layout = dict(
165     title = 'Trade flows between Poland and its trading partners.',
166     showlegend = False,
167     geo = dict(
168         scope='world',
169         projection=dict( type='robinson' ),
170         showland = True,
171         landcolor = 'rgb(243, 243, 243)',
172         countrycolor = 'rgb(204, 204, 204)',
173     )
174 )
175
176 fig = dict( data=flow_directions, layout=layout )
177 iplot( fig, filename='Flows map' )
178
179 print(data['Trade_value_total'].describe())
180
181 flows_winsorized = mstats.winsorize(data['Trade_value_total'], limits
=[0.05, 0.05])
182 layout = go.Layout(
183     title="Basic histogram of flows (winsorized)")
184
185 data_hist = [go.Histogram(x=flows_winsorized)]
186 fig = go.Figure(data=data_hist, layout=layout)
187
188 iplot(fig, filename='Basic histogram of flows')
189 sns.distplot(data['Trade_value_total'], axlabel= "Basic histogram of flows"
, color="y")
190
191
192 # Corr - to correct

```

```

193
194 corr = ecdf_normalized_df.corr()
195
196 sns.heatmap(corr[(corr >= 0.5) | (corr <= -0.5)],
197             cmap='viridis', vmax=1.0, vmin=-1.0, linewidths=0.1,
198             annot=True, annot_kws={"size": 8}, square=True)
199
200 # Coef of variation
201
202 layout = go.Layout(
203     title="Coefficient of variation")
204
205 data_cova = [go.Histogram(x=description.loc["cova"])]
206 fig = go.Figure(data=data_cova, layout=layout)
207 iplot(fig,
208       filename="Coefficient of variation")
209
210 high_cova = description.loc["cova"].where(lambda x: x > 0.30).dropna().sort
211     _values(ascending=False)
212 high_cova
213
214
215 # Normalization
216
217 minmax_normalized_df = pd.DataFrame(MinMaxScaler().fit_transform(data_
218     numeric),
219                                     columns=data_numeric.columns, index=
220     data_numeric.index)
221
222 standardized_df = pd.DataFrame(StandardScaler().fit_transform(data_numeric)
223     , columns=data_numeric.columns,
224     index=data_numeric.index)
225
226 ecdf_normalized_df = data_numeric.apply(
227     lambda c: pd.Series(ECDF(c)(c), index=c.index))
228
229 # Replace data by its standardized values
230
231 data[list(ecdf_normalized_df.columns.values)] = ecdf_normalized_df
232
233 # Select only POL as rt3ISO
234
235 data_PL = data.query("rt3ISO == 'POL'")
236
237 data_PL.drop('rt3ISO', axis=1, inplace=True)
238
239 data_PL.info()
240
241 # One hot encoding
242 data_PL = pd.get_dummies(

```

```

241 data_PL, columns=["pt3ISO", "legold_o", "legold_d", "legnew_o", "legnew
    _d", "flaggsp_o_d", "flaggsp_d_d"],
242 prefix=["pt3ISO", "legold_o", "legold_d", "legnew_o", "legnew_d", "
    flaggsp_o_d", "flaggsp_d_d"])
243
244 # Splitting the data
245
246 # train_size = 0.9
247 # train_cnt = math.floor(data_PL.shape[0] * train_size)
248
249 splitting_yr = 2010
250
251 x_train = data_PL.drop('Trade_value_total', axis=1).loc[data_PL['yr'] <=
    splitting_yr].values
252 y_train = data_PL.loc[:, 'Trade_value_total'].loc[data_PL['yr'] <=
    splitting_yr].values
253 x_test = data_PL.drop('Trade_value_total', axis=1).loc[data_PL['yr'] >
    splitting_yr].values
254 y_test = data_PL.loc[:, 'Trade_value_total'].loc[data_PL['yr'] > splitting_
    yr].values
255
256 # Build NN class in PyTorch
257
258 # A fully-connected ReLU network with one hidden layer, trained to predict
    y from x
259 # by minimizing squared Euclidean distance.
260
261
262 class ThreeLayerNet(torch.nn.Module):
263     def __init__(self, D_in, H_in, H_out, D_out):
264         """
265         In the constructor we instantiate two nn.Linear modules and assign
            them as
266         member variables.
267         """
268         super(ThreeLayerNet, self).__init__()
269         self.linear1 = torch.nn.Linear(D_in, H_in)
270         self.linear2 = torch.nn.Linear(H_in, H_out)
271         self.linear3 = torch.nn.Linear(H_out, D_out)
272
273     def forward(self, x):
274         """
275         In the forward function we accept a Tensor of input data and we
            must return
276         a Tensor of output data. We can use Modules defined in the
            constructor as
277         well as arbitrary operators on Tensors.
278         """
279         h_relu_1 = self.linear1(x).clamp(min=0)
280         h_relu_2 = self.linear2(h_relu_1).clamp(min=0)
281         y_pred = self.linear3(h_relu_2)
282         return y_pred
283

```

```

284
285 #x = torch.tensor(x_train).float()
286 #y = torch.tensor(y_train).float()
287
288 # N is batch size; D_in is input dimension;
289 # H is hidden dimension; D_out is output dimension.
290 N, D_in, H_in, H_out, D_out = int(data_PL.shape[0]), int((data_PL.shape[1]
    - 1)), 50, 50, 1
291
292 # Construct our model by instantiating the class defined above
293 #model = ThreeLayerNet(D_in, H_in, H_out, D_out)
294
295 # Construct our loss function and an Optimizer. The call to model.
    parameters()
296 # in the SGD constructor will contain the learnable parameters of the three
297 # nn.Linear modules which are members of the model.
298 #criterion = torch.nn.MSELoss(reduction='sum')
299 #optimizer = torch.optim.SGD(model.parameters(), lr=1e-4)
300 #for t in range(5):
301 #     # Forward pass: Compute predicted y by passing x to the model
302 #     y_pred = model(x)
303 #
304 #     # Compute and print loss
305 #     loss = criterion(y_pred, y)
306 #     print(t, loss.item())
307 #
308 #     # Zero gradients, perform a backward pass, and update the weights.
309 #     optimizer.zero_grad()
310 #     loss.backward()
311 #     optimizer.step()
312
313 # Build NN class in Keras
314
315
316 def build_model(x_train, y_train, x_val, y_val, params):
317
318     model = keras.Sequential()
319     model.add(keras.layers.Dense(10, activation=params['activation'],
320                                input_dim=x_train.shape[1],
321                                use_bias=True,
322                                kernel_initializer='glorot_uniform',
323                                bias_initializer='zeros',
324                                kernel_regularizer=keras.regularizers.l1_
                                    l2(l1=params['l1'], l2=params['l2']),
325                                bias_regularizer=None))
326
327     model.add(keras.layers.Dropout(params['dropout']))
328
329     # If we want to also test for number of layers and shapes, that's
        possible
330     hidden_layers(model, params, 1)
331
332     # Then we finish again with completely standard Keras way

```

```

333     model.add(keras.layers.Dense(1, activation=params['activation'], use_
334         bias=True,
335         kernel_initializer='glorot_uniform',
336         bias_initializer='zeros',
337         kernel_regularizer=keras.regularizers.l1_
338             l2(l1=params['l1'], l2=params['l2']),
339         bias_regularizer=None))
340
341     model.compile(optimizer=params['optimizer'](lr=lr_normalizer(params['lr
342         '], params['optimizer'])),
343         loss=params['losses'],
344         metrics=['mse'])
345
346     history = model.fit(x_train, y_train,
347         validation_data=[x_val, y_val],
348         batch_size=params['batch_size'],
349         epochs=params['epochs'],
350         callbacks=[early_stopper(epochs=params['epochs'],
351             mode='strict')],
352         verbose=0)
353
354     # Finally we have to make sure that history object and model are
355     returned
356     return history, model
357
358 # Then we can go ahead and set the parameters space
359
360
361 params = {'lr': (0.5, 4, 4),
362     'l1': (0.1, 40, 4),
363     'l2': (0.1, 40, 4),
364     'first_neuron': [4, 8, 16],
365     'hidden_layers': [0, 1, 2],
366     'batch_size': [32, 64],
367     'epochs': [250],
368     'dropout': (0, 0.5, 5),
369     'optimizer': [Adam, SGD],
370     'losses': [mse],
371     'activation': [relu, sigmoid]}
372
373 # Alternatively small parameters space
374
375
376 params_small = {'lr': (0.5, 5, 2),
377     'l1': (0.1, 50, 2),
378     'l2': (0.1, 50, 2),
379     'first_neuron': [4],
380     'hidden_layers': [0],
381     'batch_size': [32], # [32, 64, 128, 256],
382     'epochs': [100],
383     'dropout': (0, 0.5, 2),
384     'optimizer': [Adam],
385     'losses': [mse],

```

```
381         'activation': [relu]}
382
383 # Run the experiment
384
385 os.chdir(path + "/Data/")
386
387 t = ta.Scan(x=x_train,
388            y=y_train,
389            model=build_model,
390            grid_downsample=1,
391            val_split=0.3,
392            params=params,
393            dataset_name='POL',
394            experiment_no='2')
```

Neural_net.py