NEURAL NETWORKS

Neural networks, also known as artificial neural networks (ANNs). Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, possibly one or more hidden layers, and an output layer. The input variables are related to the output variable(s) through a network of interconnected nodes, with associated weight and threshold. If, and only if, the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. During the learning process of the NN these weights adjusts. The optimal values for these weights are estimated when a Neural Network is fitted, such that a predetermined loss function is minimized At each layer, the sum of the weights plus an intercept,

referred to as the bias, is taken and transformed by a function which is called the activation function. To advance from one layer to another, the ANN uses so-called activation functions. These connect

each node of a certain layer with every node of the subsequent layer

The input layer of the NN consists of n nodes, where n is equal to the number of explanatory variables. Since we model the Hb concentration, we only have one node in the output layer. For the hidden layers and output function we need activation functions. For regression purposes, an ANN only has one node in the output layer. In our setting, the output node represents the predicted Hb concentration.

According to the Universal Approximation Theorem, a Neural Network with one hidden

layer can approximate any Borel measurable function, given that the hidden layer contains enough nodes. However, in practice, a network with multiple hidden layers can be more efficient.

All networks are estimated with a squared error

loss function, and with regularization, bagging and early stopping to prevent overfitting.

One of the risks of Neural Networks is that it tends to overfit on the training data. To

prevent this, we implement three strategies. First, we use early stopping; we set aside ten

percent of the training data as validation data and we stop our training if we do not improve the

performance of our validation set by 0.0001 in ten consecutive epochs. As a second strategy, we add a Ridge penalty, consisting of a regularization parameter, multiplied with the L2 norm of

all δ parameters (Hastie et al., 2009). The magnitude of the penalty can be tuned by adjusting

the regularization parameter, which we explain further on. Finally, we apply a bagging regressor

We compare four commonly used

activation functions: the identity function (h(x) = x), the logistic sigmoid function (h(x) =

1

1+e−x ), the hyperbolic tangent function (h(x) = tanh(x) = ex−e−x

ex+e−x ) and the rectified linear

unit (relu) function (h(x) = max(0, x)). Finally, we optimize the Ridge regularization term. We

compare values of 0.0001, the standard parameter in the sklearn.neural network package, and

increased values of 0.001, 0.01 and 0.1, which we multiply with the L2 norm of all parameters.

The architecture of ANN models has been developed by Lippmann (1987). We use the implementation of

the ANN by Cruset Pla (2021) for the quarterly estimation of GDP and extend the model to a monthly

estimation, as discussed in Section 4.1.

The Multi Layer Perceptron (MLP) is a class of feedforward ANNs. The general MLP consists of an

input layer, one or more hidden layers and an output layer. Each layer consists of a collection of nodes,

which are called neurons in the hidden layers. The vector of input variables transmits a signal to each

neuron, where each signal has its own weight.

The ANN needs to be trained using a training data set and a learning algorithm.

To clarify an ANN with two hidden layers, the structure of this model is presented in Figure 4. The

input layer of this model is represented X = (X1, . . . ,Xn), indexed by i, and the neurons in the first and

second hidden layers are z(1) = (z

(1)

1 , . . . , z

(1)

M1

) and z(2) = (z

(2)

1 , . . . , z

(2)

M2

) respectively, indexed by j and

k. Here, n represent the number of input values, and M1, and M2 the number of nodes in the first and

second hidden layer. The output layer consists of a single node y. The ANN progresses from the input

layer through the hidden layers to the output layer. This is first done for a training data set to estimate

the model parameters. Hereafter, the test data set is used to produce an estimate of the output layer.

Figure 4: Structure of our Artificial Neural Network model. Source: Ognjanovski (2019).

. To go from the input layer to the

first hidden layer, the ANN uses

a

(1)

j =

nΣ

i=1

β

(1

For our ANN we use a forward pass and start training the algorithm from 2005Q1 onward on quarterly

data. This is because the GT data in 2004 consists of zeros only, and can thus not be used to train

our model. Because of this, the estimation will start from 2006Q1, as we wish to have a few training

observations for our first nowcast. After each estimation iteration, the training data gets expanded, which

ensures that the most recent information is available. For our hyperparameters, we use the ones that

perform best according to Cruset Pla (2021), since tuning is not desirable due to a high running time for

the ANN.

An ANN consists of various layers: an input layer, an output layer and (multiple) hidden layers.

The input layer contains our explanatory variables, each as a single node in the layer, and the

output layer contains (the prediction of) the dependent variable, which is a single node. The

hidden layers contain the neurons that connect the input layer with the output layer.

15

Assume we have a single hidden layer. If we have p number of explanatory variables (x1,

..., xp) and m neurons in the hidden layer (z1, ..., zm), the ANN becomes the following set of

equations (Hastie et al., 2009):

aj =

Xp

i=1

δ(1)

ji xi + δ(1)

j0 ,

zj = h(aj), j = 1, ..., m,

a =

Xm

j=1

δ(2)

kj zj ,

ˆy = h(a),

(6)

where h is the activation functions and δ(1) and δ(2) are the set of parameters in the ANN. There

are various activation functions, which we decide on based on a tuning procedure, explained

further on. The set of explanatory variables xi consists of the quarterly Google Trends variables,

the SPF quarterly GDP, inflation and unemployment forecasts, the quarterly EUROSTOXX 50

and MSCI Europe Small Cap index, country-specific constant terms and the monthly variables of

the European Business cycle indicator survey, all shown in Appendix C. The monthly variables

are aggregated to a quarterly frequency by specific weights, explained in Section 4.1.4.

Equation 6 shows the parameters δ that we need to optimize. As this cannot be solved analytically,

we use a solver for this. For the implementation of our ANN, we use the sklearn.neural

network package, where we stop after 7000 iterations if convergence is not achieved.

One of the risks of Neural Networks is that it tends to overfit on the training data. To

prevent this, we implement three strategies. First, we use early stopping; we set aside ten

percent of the training data as validation data and we stop our training if we do not improve the

performance of our validation set by 0.0001 in ten consecutive epochs. As a second strategy, we add a Ridge penalty, consisting of a regularization parameter, multiplied with the L2 norm of

all δ parameters (Hastie et al., 2009). The magnitude of the penalty can be tuned by adjusting

the regularization parameter, which we explain further on. Finally, we apply a bagging regressor

to fit the data on the quarter that we want to predict (Breiman, 1996): we use our ANN with

given hyperparameters and we fit this model on five random subsets of our data with the same

size as the original data to obtain five individual nowcasts for our dependent variable, where we

bootstrap the subsets. We then average these nowcasts to obtain a final nowcast.

hidden layers and nodes, the solver of the weights, the activation function and the regularization

term parameter. First, we optimize the structure, where we fix the three other hyperparameters;

Adam as solver, the hyperbolic tangent as activation function and a regularization

parameter of 0.0001. After this, we optimize the solver, where we use the optimized structure,

16

and fix the two other hyperparameters. We repeat this procedure for the activation function

and finally the regularization term parameter.

We start by optimizing the structure of the hidden layers; the number of hidden layers and

the number of nodes for each hidden layer. For the number of layers, Heaton (2008) indicate that

we can only represent linear functions with zero hidden layers, whereas we can approximate any

function with a continuous mapping with finite spaces with one hidden layer. Furthermore, the

author indicates that two hidden layers can represent and approximate every smooth mapping

to any desired accuracy. Thus, there is no need to use more than two hidden layers, and we

compare the performance for zero, one and two hidden layers. Woloszko (2020) shows that

we require enough neurons to model all the country specific interactions between the Google

Trends and the (components of) GDP. The author refers to at least 100 nodes in their paper.

However, we have a smaller set of observations, but a larger number of input nodes. Therefore,

we examine the performances of an ANN with both a smaller and a larger number of nodes for

the first layer; we consider 50, 100, 200 and 300 nodes. Furthermore, Woloszko (2020) uses a

second hidden layer of size 10. Therefore, as we are also interested in higher number of nodes

in the second hidden layer, we examine the following number of nodes for the second hidden

layer: 10, 50, 100, 200 and 300. However, we require the second layer to have at most the same

number of nodes as the nodes in the first layer, as done by Woloszko (2020). This results in

twenty options for the structure of the ANN for which we compare the nowcasting performance.

After the structure of the ANN, we compare the performance for the solver we use to optimize

the weights of the ANN.We compare two solvers: Stochastic Gradient Descent (SGD) and Adam

(Kingma and Ba, 2014). Both solvers require a learning rate, which we set at 0.001. However,

Adam allows for a time-varying learning rate, which decreases over time.

After the solver, we optimize the activation function. We compare four commonly used

activation functions: the identity function (h(x) = x), the logistic sigmoid function (h(x) =

1

1+e−x ), the hyperbolic tangent function (h(x) = tanh(x) = ex−e−x

ex+e−x ) and the rectified linear

unit (relu) function (h(x) = max(0, x)). Finally, we optimize the Ridge regularization term. We

compare values of 0.0001, the standard parameter in the sklearn.neural network package, and

increased values of 0.001, 0.01 and 0.1, which we multiply with the L2 norm of all parameters.

SVR

SVM has the advantage that this classifier performs generally well for high dimensional input spaces (Trustorff et al., 2011).

SVM works by separating binary classified data using a hyperplane as a decision boundary such that the margin between the classes is maximised (Cortes 10 & Vapnik, 1995). In other words, the input attributes xi are transformed into a m-dimensional feature space by using a non-linear mapping after which the SVM algorithm searches for the best separating hyperplane related to a set of support vectors (Moro et al., 2014). In practice, data are often not perfectly separable by a single hyperplane (Bellotti & Crook, 2009). Therefore, a so-called soft margin is used where some slack is allowed for observations to be on the wrong side of the margin.

SVM with RBF kernel is preferred over the linear and polynomial kernel since the linear kernel is a special case of the RBF kernel, and the polynomial kernel could be numerically difficult (Keerthi & Lin, 2003).

The classification performance is affected by hyperparameters C and σ. The penalty parameter C decides on the trade-off between the complexity of the model and the size of the margin error (Cortes & Vapnik, 1995). If C is too large, the SVM algorithm tries to reduce the margin error drastically. To achieve this, a complex hyperplane is formed to separate classes, which could result in overfitting. If C is too low, more observations are used as support vectors and the size of the margin error increases, which could lead to underfitting.

Essentially what SVM tries to do, is to find a seperating hyperplane f(x) that optimally distinguishes between two classes (Du and Swamy, 2013

However Maldonado and Weber (2009) show that when using SVM in an imbalanced setting, a nonlinear SVM with the Gaussian (i.e., the Radial basis function) kernel obtains the best performance in most cases. Therefore we will base our analysis the Radial basis function (Rbf) kernel.

The objective of a support vector machine algorithm is to find a hyperplane in an n-dimensional space that distinctly classifies the data points. The data points on either side of the hyperplane that are closest to the hyperplane are called SupportVectors. These influence the position and orientation of the hyperplane and thus help build the SVM.

Hyperparameters: [**https://towardsdatascience.com/unlocking-the-true-power-of-support-vector-regression-847fd123a4a0**](https://towardsdatascience.com/unlocking-the-true-power-of-support-vector-regression-847fd123a4a0)

Support Vector Regression is a supervised learning algorithm that is used to predict discrete values. Support Vector Regression uses the same principle as the SVMs. The basic idea behind SVR is to find the best fit line.

Unlike other Regression models that try to minimize the error between the real and predicted value, the SVR tries to fit the best line within a threshold value. The threshold value is the distance between the hyperplane and boundary line. The fit time complexity of SVR is more than quadratic with the number of samples which makes it hard to scale to datasets with more than a couple of 10000 samples.

For large datasets, Linear SVR or SGD Regressor is used. Linear SVR provides a faster implementation than SVR but only considers the linear kernel. The model produced by Support Vector Regression depends only on a subset of the training data, because the cost function ignores samples whose prediction is close to their target.

SVMs solve binary classification problems by formulating them as convex optimization problems (Vapnik 1998). The optimization problem entails finding the maximum margin separating the hyperplane, while correctly classifying as many training points as possible. SVMs represent this optimal hyperplane with support vectors. The sparse solution and good generalization of the SVM lend themselves to adaptation to regression problems. SVM generalization to SVR is accomplished by introducing an *ε*-insensitive region around the function, called the *ε*-tube. This tube reformulates the optimization problem to find the tube that best approximates the continuous-valued function, while balancing model complexity and prediction error. More specifically, SVR is formulated as an optimization problem by first defining a convex *ε*-insensitive loss function to be minimized and finding the flattest tube that contains most of the training instances.

Some important terms that are synonymous with the working of SVR are :

**Kernel:**

The function for converting a lower-dimensional data set to a higher-dimensional data set. A kernel aids in the search for a hyperplane in higher-dimensional space while reducing the computing cost.

When the size of the data grows larger, the computing cost usually rises. When we are unable to identify a separating hyperplane in a particular dimension and must shift to a higher dimension, this increase in dimension is necessary.

**Hyper Plane:**

This is the separating line between the data classes in SVM. Although, in SVR, we will describe it as a line that will assist us in predicting a continuous value or goal value.

**Boundary line:**

Other than Hyper Plane, there are two lines in SVM that produce a margin. The support vectors might be within or outside the boundary lines. The two classes are separated by this line.

The premise is the same in SVR. A decision boundary line can be conceived of as a demarcation line (for simplicity), with positive examples on one side and negative examples on the other.

The instances on this line can be characterised as either good or negative. The same SVM approach will be used in Support Vector Regression as well.

**Support vectors:**

The data points closest to the border are listed here. The distance between the locations is little or negligible. Support vectors are locations that are outside the -tube in SVR. The smaller the value of, the more points outside the tube there are, and hence the more support vectors there are.

(Also read: [Different types of learning in ML](https://www.analyticssteps.com/blogs/what-are-different-types-learning-machine-learning))

**Conclusion**

SVR really proves to be better than deep learning methods in cases of limited datasets and also require much less time than its counterpart. In comparison with other regression algorithms, SVR uses much less computation and has high accuracy and credibility.