HINDSIGHT: An R-Based Framework Towards Long Short Term Memory (LSTM) Optimization

Konstantinos Kousias
Simula Research Laboratory
Norway
kostas@simula.no

Michael Riegler
Simula Metropolitan Center of Digital Engineering
Norway
michael@simula.no

Özgü Alay
Simula Metropolitan Center of Digital Engineering
Norway
ozgu@simula.no

Antonios Argyriou
University of Thessaly
Greece
anargyr@uth.gr

ABSTRACT

Hyperparameter optimization is an important but often ignored part of successfully training Neural Networks (NN) since it is time consuming and rather complex. In this paper, we present HINDSIGHT, an open-source framework for designing and implementing NN that supports hyperparameter optimization. HINDSIGHT is built entirely in R and the current version focuses on Long Short Term Memory (LSTM) networks, a special kind of Recurrent Neural Networks (RNN). HINDSIGHT is designed in a way that it can easily be expanded to other types of Deep Learning (DL) algorithms such as Convolutional Neural Networks (CNN) or feed-forward Deep Neural Networks (DNN). The main goal of HINDSIGHT is to provide a simple and quick interface to get started with LSTM networks and hyperparameter optimization.

KEYWORDS

Deep Learning (DL), Long Short Term Memory (LSTM) networks, hyperparameter optimization, Manual Search (MS), Random Search (RS)

ACM Reference Format:
Konstantinos Kousias, Michael Riegler, Özgü Alay, and Antonios Argyriou. 2018. HINDSIGHT: An R-Based Framework Towards Long Short Term Memory (LSTM) Optimization. In *Proceedings of ACM (MMSys).* ACM, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

The need for understanding and capturing temporal effects in time-series data has led to the growth of Recurrent Neural Networks (RNN) popularity. Common time-series forecasting challenges include the AMS 2013-2014 Solar Energy Prediction Contest[1], where the problem was to predict the solar energy consumption based on meteorological forecasts, and the Global Energy Forecasting Competition 2012[2], a power generation problem given wind predictions. Unlike traditional feed-forward architectures, RNN introduce feedback loops that allow information to persist or vanish from one network to another. The connection between the present and the past is crucial for recognizing patterns and motifs. The innovative and unique design of RNN has tremendously increased their popularity in fields like image captioning, speech and text recognition [9, 11, 17].

Despite of all the advantages mentioned above, authors in [3] highlight a major drawback. Using theoretical but also empirical evidence, they show that RNN lack the ability to learn tasks that involve long-term dependencies. This is where Long Short Term Memory (LSTM) networks come into play [12]. LSTM networks are a special type of RNN that can handle such dependencies and carry on information for long intervals. Throughout the last decade, LSTM networks have been proved a handy tool in the Deep Learning (DL) community and are currently used for Artificial Intelligence (AI) problems that require long range memory. Such examples include reinforcement learning, handwriting recognition and tasks requiring precise timing that traditional RNN architectures are unable to solve [2, 7, 8, 10].

A regular LSTM architecture is comprised of a series of connected cells named memory blocks. An overview architecture of a memory block can be seen in Figure 1. Each of these cells has three main mechanisms responsible for exploiting network's memory, known as gates. A forget gate multiplies the input x_t at the current time step with cell's previous state h_{t-1}. That way, it removes information from the cell that is redundant and optimizes the overall performance of the network. On the contrary, an input gate is responsible for adding information that is valuable for the memory block. Involving a `sigmoid` and a `tanh` function, it uses multiplication and addition operations to ensure that no redundant information has been stored in the cell. Last part of the architecture is the output gate which, by using similar functionality as the forget gate, decides a cell's final state. As its name implies, it is the link between two consecutive cells.

Hyperparameter optimization in DL is critical since the performance of every NN architecture is highly connected on the selection of the hyperparameters. Several studies have been conducted to determine efficient algorithms that optimize the set of hyperparameter values [6, 15]. Among the most popular approaches include Manual Search (MS), Grid Search (GS), Random Search (RS) and the probability-based Bayesian Optimization (BOA) algorithm [4, 22]. Nevertheless, hyperparameter optimization is rarely applied as it is time-consuming and increases significantly the computational complexity of the models.

[1] www.kaggle.com/c/ams-2014-solar-energy-prediction-contest
[2] www.kaggle.com/c/GEF2012-wind-forecasting

MMSys, June 12-15, 2018, Amsterdam
2018. ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

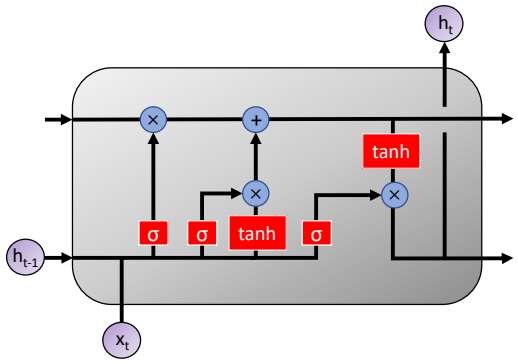

Figure 1: An illustration of a cell or memory block. Circles serve as point wise operations while boxes represent Neural Networks (NN) layers.

Since LSTM networks have shown such great promise, in this paper, we introduce HINDSIGHT, a comprehensive tool that we develop and offer to the community for experimentation with LSTM networks. The purpose of HINDSIGHT is first, to minimize the complexity of using LSTM networks, and second, to optimize the selection of the LSTM hyperparameters in different application domains. Below, a list of the main contributions of this paper is outlined:

- HINDSIGHT is an open-source framework written exclusively in R.
- It allows for easy and quick experimentation with LSTM networks in a wide variety of fields.
- HINDSIGHT supports hyperparameter optimization, thus allowing users to easily explore the best choice, depending on the application.

In the remainder of the paper, the structure of HINDSIGHT is described and an overview of the hyperparameter optimization algorithms is given. Prior to the conclusion and future work, a use case from the networking domain is presented.

2 HINDSIGHT

The code of HINDSIGHT is written in R and can be found in the Bitbucket repository[3]. The source files are organized as follows: A list of five functions that compose HINDSIGHT, a proof of concept test script, a *.RData* image of two example datasets (training and testing) and a *README.md* file. In the latter, one can find details on how to install and use the framework. HINDSIGHT version 1.0 supports only LSTM networks but is designed in a versatile manner so it can be expanded to other DL architectures.

HINDSIGHT is based on the services of two packages and libraries in R. The backbone of HINDSIGHT is the CRAN `Keras` package[4]. `Keras` is a high level Neural Networks (NN) API which allows for dynamic experimentation with DL approaches such as feed-forward Deep Neural Networks (DNN), Convolutional Neural Networks (CNN), RNN and so forth. It supports multiple backend environments including `Theano` and `Tensorflow`. HINDSIGHT current version is configured to run on `Tensorflow` by using the `install_keras()` function. It can be operated with or without Graphical Processing Unit (GPU) support. To install the GPU version, the 'gpu' keyword must be included as:

```
install_keras(tensorflow = "gpu")
```

Prior to the installation, all the Cuda and CUDNN libraries have to be installed and a GPU with compute capability higher than 3.0 is required. HINDSIGHT is a wrapper to `Keras` and does not introduce any novel functions but rather using the existing libraries to provide easy experimentation with LSTM networks.

HINDSIGHT is designed in a way so that it is compatible with both univariate and multivariate time series prediction problems. A widespread use case that illustrates the former is forecasting in stock price market. There exists a recent trend towards DL approaches in fields such as economics, climatology and medicine. The concept of exploiting information from the past to predict the future seems to improve the performance of NN. Framing univariate data for LSTM is relatively simple since a single variable is involved [12]. Multivariate data consisting of multiple variables are more complex in terms of required preprocessing. HINDSIGHT discloses all preliminary steps required to bring data in the appropriate format and allows for quick and easy experimentation with LSTM networks.

The `hindsight` function forms the nucleus of HINDSIGHT. A list of the function's input parameters, a short description and the default values are summarized in Table 1. The fields missing default values are compulsory and have to be set during the call of `hindsight`. In the case of multivariate data, the formatting requirements for both `training` and `testing` sets is as follows. The dependent variable **must** always reside in the first column of the dataframe followed by the set of regressors. Ordering of the latter is of no significant importance.

Data Preprocessing

After both training and testing datasets are in the required format, the `time_series` function is called. The input parameters set is comprised of the `data`, `nfeatures` and `nlags`. `nfeatures` is the number of features to be used as exogenous variables and if this value is smaller than the available data features, the first features, starting from the left are selected. The `nlags` parameter defines the number of steps to 'look back' in time. The selected value depends on the nature of the problem and fully determines the shape of the data. For example, in text recognition where learning long-term dependencies is crucial, a `nlags` value bigger than one is needed. A pseudo-code version of `time_series` is presented in Listing 1. In short, for `nlags` equal to one, the endogenous variable will be shifted one position down and concatenated as a new column in the dataset. For larger values, the `shift` function will be applied to the complete set of features that is defined by `nfeatures`. All resulting Not Available (NA) values are eliminated and removed permanently from the data. The number of features after `time_series` is called equals to **`nlags*nfeatures`** + 1. Finally, each dataset is split into two separate arrays (one consisting of the dependent variable and one of the regressors) as required by the `Keras` API.

Designing Neural Network Architectures

In the following, the LSTM design phase is described in depth and a short overview of the hyperparameters is presented. The function

[3] https://bitbucket.org/konstantinoskousias/hindsight
[4] https://cran.r-project.org/web/packages/keras/keras.pdf

Table 1: HINDSIGHT input parameters. Parameters in bold are available for optimization.

ID	Parameter	Short Description	DEF
1	training	Training dataset	-
2	testing	Testing dataset	-
3	nfeatures	No. features	-
4	nlags	No. time steps	-
5	**units**	**No. neurons for input layer**	**10**
6	**units1**	**No. neurons for hidden layer *1***	**10**
7	**units2**	**No. neurons for hidden layer *2***	**10**
8	**units3**	**No. neurons for hidden layer *3***	**10**
9	**lr**	**Learning rate**	**0.01**
10	**nepochs**	**No. epochs**	**20**
11	**bs**	**Batch size**	**32**
12	**nlayers**	**No. layers**	**1**
13	**opt**	**List of optimization algorithms**	**1**
14	activation	Activation function	*relu*
15	valsplit	Validation split	0.1
16	patience	No. epochs before early stopping	5
17	rs	Selection between MS and RS	FALSE
18	niter	No. iterations (only for RS)	20
19	units_l	units <upper limit>	256
20	units1_l	units1 <upper limit>	256
21	units2_l	units2 <upper limit>	256
22	units3_l	units3 <upper limit>	256
23	lr_l	lr <upper limit>	0.1
24	nepochs_l	nepochs <upper limit>	200
25	bs_l	bs <upper limit>	128
26	nlayers_l	nlayers <upper limit>	4
27	opt_l	opt <upper limit>	4

Algorithm 1: time_series: Preprocessing data for LSTM.

```
function time_series (data, nlags, nfeatures)
for i = 1; i < nlags do
    for j = 1; j < nfeatures do
        if i == nlags then
            data['feat_j(t)'] <- Shift data[j] i positions
              down break
        end
        else
            data['feat_j(t-(nlags − i))'] <- Shift data[j] i
              positions down
        end
    end
end
for k = 1; k < nfeatures do
    column_names(data)[k] = 'feat_k(t-nlags)'
end
Remove last N = nlags rows from data comprising of NA
  values
```

responsible for the NN part in HINDSIGHT appears under the name of `lstm`. First, an LSTM sequential model is created via:

```
model <- keras_model_sequential()
```

Next, by using the pipe operator `%>%`, new layers are added in the LSTM network. The first layer is called *input layer*. The majority of times, the number of neurons comprising the input layer equals the number of features in the data. Since the above statement is not a principle, experimentation with a diverse number of neurons is allowed by tweaking the `units` hyperparameter (see Table 1). The `input_shape` argument defines the shape of the input layer and equals to *c*(`nlags`, `nlags*nfeatures`). In a similar fashion, the hidden layers are added next. HINDSIGHT version 1.0 supports LSTM architectures up to three hidden layers which is sufficient to solve most learning problems. Hyperparameters `units1`, `units2` and `units3` define the number of neurons for each of the hidden layers respectively. Hyperparameter `nlayers` defines the number of layers with a range between 1 (only one input layer) and 4 (one input layer and three hidden layers). Last, the output layer is added by typing the below command:

```
layer_dense(units = 1, activation = "relu")
```

As for the activation function, *relu* is selected since it converges to a solution faster [14]. Other popular approaches include the *sigmoid* and *softmax* function, with the latter one used mostly in binary classification problems. `activation` is used for selecting between the activation functions.

During the following compilation phase, the loss function, the metrics and the optimization algorithm are defined.

Error Metrics. The Mean Absolute Error (MAE) is used as the loss function of the model. It is described as the sum of residuals divided by the number of samples in the data. The mathematical expression can be seen in Equation 1.

$$MAE = \sum_{i=1}^{n} \frac{|e_i|}{n} \tag{1}$$

As for the metrics, the Mean Absolute Percentage Error (MAPE) is selected, which measures the size of error in a percentage fashion. The mathematical formula is defined in Equation 2.

$$MAPE = 100\frac{1}{n}\sum_{i=1}^{n} |\frac{e_i}{X_i}| \tag{2}$$

Gradient Descent (GD) is one of the most prominent algorithms used for NN optimization. Various different approaches to optimize GD have been proposed, targeting on exploiting algorithm's performance to the full potential. Evaluating and selecting between them though is not always an easy task, as most of the times they are used as black boxes. Below, a list of the GD optimization algorithms that HINDSIGHT supports is presented.

Optimization algorithms. Stochastic Gradient Descent (SGD) optimizer has recently gained popularity with scientists finding application to large-scale and sparse Machine Learning (ML) problems in fields as text classification and natural language recognition. SGD adopts an iterative method for minimizing the objective function and is primarily known for its efficiency and ease of implementation. It is used as the default optimizer for HINDSIGHT. **Adam**, comes from adaptive moment estimation, is an extension of SGD which has recently found application to DL fields like computer vision [13]. Adam requires minimum computational memory during optimization, its hyperparameters are straightforward to tune and

it is known for converging to a solution relatively fast. **Adagrad**, stands for adaptive gradient, optimizes GD by updating the learning rate with respect to the appearing frequency of the parameters. It is well suited for dealing with DL problems that involve sparse data. In [5], authors found that Adagrad improved the robustness of SGD and used it for training large scale NN at Google. Adagrad was also used in text classification with great success [19]. Last, **RmsProp** is a still unpublished learning rate optimizer [23] that was developed to resolve Adagrad's radically diminishing learning rates. `lr` hyperparameter can be used to adjust the learning rate for each of the optimization algorithms.

Training and Validation

The LSTM model is trained by using the `fit` function available in CRAN. The hyperparameters to be tuned during this phase are `nepochs` and `bs`. Validation split defines the percentage of data that is used for validating the training data. Note that, the data is never shuffled before training and the percentage always reflects the last portion of samples. For example, a validation split equal to 0.1 means that the last 10% of the data will be used as the validation set. Validation split can be adjusted with the `valsplit` parameter.

During the training-validation phase, the `early stopping` callback is exploited to avoid overfitting and decrease the total running time. With early stopping, the training of a NN model stops when the monitored error metric has stopped improving. A critical parameter to be specified when using early stopping is the number of epochs where no improvement is observed, called patience. Patience is basically a way of controlling the error sensitivity during training. For example, a patience of value zero means that training will stop if the observed error increases from one epoch to the next. In [21], the authors discuss about the trade-off of using slow stopping criteria (large value for patience) and the total training time. They conclude that slower stopping criteria decrease the generalization error but lead to increased training time and possibly overfitting. Therefore, in HINDSIGHT current version 1.0, `patience` parameter is set to five as default but can be adjusted as needed.

Last, the `evaluate` function is called on the testing data and returns the MAE. A snippet of the code can be seen below:

```
loss = model %>% evaluate(test_2, test_1)
```

In case where RS has been selected, a table (`gridtable`) comprised of each LSTM configuration and the testing loss is returned to user's R workspace. A pseudo-code version of `lstm` is presented in a similar fashion as the `time_series` function in Listing 2.

Saving and Loading LSTM Models

The underlying `Keras` library provides built-in functions to save and load NN models on demand. The corresponding R command can be seen in the snippet below:

```
save_model_hdf5(model, 'lstm.h5')
```

This command allows for saving the architecture, the weights of the model, the training configuration and the state of the optimizer in a single `HDF5` file. Using saved files, the model can be loaded and instantiated by:

```
model <- load_model_hdf5('lstm.h5')
```

Algorithm 2: `lstm`: Building LSTM Networks.

1 function `lstm` (`train_1`, `train_2`, `test_1`, `test_2`, `nfeatures`, `nlags`, `units`, `units1`, `units2`, `units3`, `lr`, `nepochs`, `bs`, `nlayers`, `opt`, `activation`, `valsplit`, `patience`)
2 Create a sequential model:
3 > **model <- keras_model_sequential ()**
4 Add the input layer:
5 > **layer_lstm (units, return_sequences = F, input_shape = c (nlags, nlags*nfeatures))**
6 Add **n = nlayers - 1** hidden layers and the output layer.
7 Compile the model:
8 > **model %>% compile (loss = 'MAE', metrics = 'MAPE', optimizer = opt)**
9 Fit the model:
10 > **model %>% fit (train_2, train_1, nepochs, bs, valsplit, callback = early_stopping (patience))**
11 Evaluate the model:
12 > **loss = model %>% evaluate (test_2, test_1)**

Saving and loading DL models are useful assets when comparing different architectures. Users are highly recommended to exploit the stated commands when using HINDSIGHT. A save command is placed by default at the end of the code for each LSTM model. The path folder in which the model will be stored coincides with the R working directory but can be easily modified using the *setwd()* built-in command. Furthermore, an image of the R user workspace is saved after each iteration as:

```
save.image("HINDSIGHT.RData")
```

Moreover, timers have been placed at the beginning and at the end of the `lstm` function. Timing a session can be proved critical during experimentation of deep architectures with a varied range of hyperparameters. For example, in most DL models, time complexity increases along with the number of layers in the architecture. Other cases where timing can be beneficial is when the RS hyperparameter optimization algorithm has been selected or when timing requirements between various GPUs need to be calculated.

3 HYPERPARAMETER OPTIMIZATION

LSTM models involve a list of *hyperparameters* that need to be carefully specified during the design phase. A hyperparameter is a variable that cannot be learned from the model but need to be set exclusively before optimizing the actual model's parameters. The number of hidden layers, the number of neurons per layer and the learning rate are a few examples of hyperparameters in DL. Setting hyperparameters manually is not always intuitive neither straightforward. As humans, we have hard time to visualize and handle multi-dimensional spaces. Hyperparameter optimization is the process of selecting an appropriate combination of hyperparameters towards improving the performance of the model. Some of the approaches that have been proposed include MS, GS, RS and BOA.

Manual Search (MS). As its name implies, MS is the manual tuning of hyperparameters. It can be a trial an error process but can

also be used in the case where prior knowledge about the learning problem exists. MS is the least efficient optimization algorithm from the list since acquiring such knowledge a priori is rare and laborious. On the positive side, it is a trivial approach and requires minimum coding.

Grid Search (GS). In GS, every available combination of hyperparameters in the search space is used to evaluate the performance of the model. A search space can be defined as a multidimensional region consisting of the set of all possible solutions. A range of values for each hyperparameter needs to be specified respectively. GS always leads to the global optimal solution. Its biggest drawback though, is the inability to scale with the increasing number or range of the available hyperparameters. The computational complexity can increase significantly, that even with a powerful computer, optimization can take days or even weeks. GS should only be used for low complexity problems.

Random Search (RS). A significantly faster but slightly less efficient method compared to GS is RS [4]. Instead of trying all different combinations of hyperparameters, random combinations in the search space are tried for a given number of iterations. The more the iterations, the more likely it is to approach the optimal solution. RS does not lead to the global optimal combination of hyperparameters but approximates it in significantly less time. In DL, where the space of hyperparameter combinations is huge, RS can be proven valuable.

Bayesian Optimization (BOA). BOA is a probabilistic approach that incorporates learning to perform hyperparameter optimization [18]. It is a well-known method that has found application to a wide range of problems including robotics, interactive animation and DL [22]. BOA is based on the bayes theorem where conditional probability is used for making new predictions.

In HINDSIGHT, `rs` is used as a *flag* variable for selecting between MS and RS. By default, `rs` is set to FALSE (MS). The parameter `niter` (defaults to 20) specifies the number of iterations for RS. The list of LSTM hyperparameters that are available for optimization is summarized in Table 1 (IDs 5-13 in bold). In addition, the search space for each hyperparameter can be implicitly specified. Parameters with IDs 19-27 define the search space upper limit for each hyperparameter.

BOA is not included in the current version of HINDSIGHT (1.0) but is in a testing phase and will become available in a future update.

4 MONROE-NETTEST: A USE CASE EXAMPLE

To show the functionality of HINDSIGHT, a demonstration based on a real use case from the domain of networking is presented. In the last years, mobile 'speed', quantified most commonly by data rate, gained popularity as the widely accepted metric to describe Mobile Broadband (MBB) networks performance since it is straightforward for any customer to understand that when it comes to mobile connectivity, higher speeds are better. Brand-new tools and applications appear in the market daily, promising to deliver more accurate and reliable mobile speed results. From the end user point of view, such tools (e.g Speedtest, OpenSignal, Mobiperf, RTR-Nettest, etc) help them assess the performance of their Internet connection while operators often use the statistics (e.g Download/Upload data rate, Delay, etc) for advertising campaigns.

Table 2: MONROE-Nettest features and a short description.

ID	Feature	Short Description
1	`download_kbit`	Download data rate ($Mbps$)
2	`upload_kbit`	Upload data rate ($Mbps$)
3	`ping_ms`	Latency (ms)
4	`lte_rsrp`	Signal strength (dBm)
5	`lte_rsrq`	Signal quality (dB)

Mobile speed measurements are very dynamic in nature since MBB networks exhibits daily and weekly patterns [16]. Furthermore, the mobile network operators are continuously improving their infrastructure, resulting in long term trends in the results. Such properties makes the mobile speed measurements a perfect fit for showcasing HINDSIGHT.

Within the scope of this use case, we use MONROE-Nettest [16] to collect mobile speed measurements in operational MBB networks. MONROE-Nettest is built as an Experiment as a Service (EaaS) on top of the MONROE platform [1], [20], Europe's first and only open dedicated platform for experimentation in operational MBB networks. MONROE makes it possible to conduct a wide range of repeatable measurements in the same location, using the same devices/modems, for multiple operators at the same time, and from an end-user perspective. Therefore, MONROE-Nettest enables the empirical analysis of mobile speed measurements through large-scale experimentation in operational MBB networks.

Dataset Description. The MONROE-Nettest dataset consists of 164 samples taken between July 2017 and January 2018. `training` set (July 2017 - December 2017) is composed of 139 samples while a `testing` set (January 2018) of 25 samples is used during the evaluation phase. All corrupted or false measurements that can affect the reliability of the model are removed (e.g samples with negative values for download/upload data rate, latency etc.). On the contrary, measurements consisting of extreme but valid values are not considered outliers, thus, are not removed from the data.

Features Description. The list of MONROE-Nettest features is as follows. `download_kbit` and `upload_kbit` (converted both to Mbps) represent the download and upload data rate, respectively. Latency is given by `ping_ms` in milliseconds. For the above features, the natural logarithm is used to reduce skewness and kurtosis of the distributions. Reference Signal Received Power (RSRP) (`lte_rsrp`) represents the power of a reference received signal at the user. Range of RSRP is between −140dBm and −44dBm. Similarly, Reference Signal Received Quality (RSRQ) (`lte_rsrq`) is a metric of the wireless channel quality, that is the ratio of RSRP versus the total received signal power, including interference and noise. RSRQ ranges from −19.5dB to −3dB. A complete list of the features along with their short descriptions is shown in Table 2.

Performance Evaluation

In this section, a proof of concept example using the MONROE-Nettest dataset is demonstrated. `nlags` and `nfeatures` are set to one and five respectively. A list of the hyperparameter values is shown in Table 3. For presentation purposes, the selection of this list

Table 3: MONROE-Nettest hyperparameters values. `units3` is not provided since `nlayers` equals to three.

ID	Hyperparameter	Value
1	units	225
2	units1	188
3	units2	100
4	units3	-
5	lr	0.06
6	nepochs	22
7	bs	39
8	nlayers	3
9	opt	1

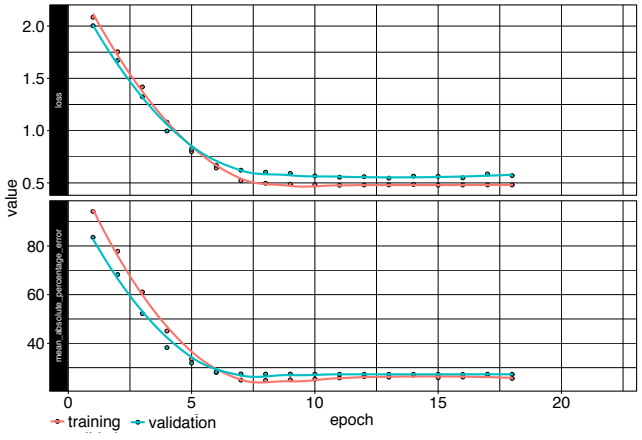

Figure 2: MAE and MAPE along the number of epochs. Training stops before the 22nd epoch since early stopping is called.

is performed using MS. The upper part of Figure 2 depicts the training and validation MAE along the number of epochs. Smoothing lines are fitted to ease readability of the plots. In the same fashion, the training and validation MAPE is illustrated in the lower part. One can see how the loss decreases in the first epochs, which is an indicator that the model learns. In addition, the MAPE decreases accordingly. Training stops during the 18th epoch where early stopping is called. Users can validate the results in HINDSIGHT since the dataset is also available in the GitHub repository[5].

5 CONCLUSIONS AND FUTURE WORK

In this paper, we presented HINDSIGHT, an R-based open-source framework that allows for easy and quick experimentation with Long Short Term Memory (LSTM) networks. Towards increasing the performance of NN, we introduced optimization approaches for selecting efficient combinations of hyperparameters. A use case from the networking domain was used as proof of concept for the framework. We envision that HINDSIGHT will be used in a wide range of fields outside the computer networking area including multimedia, economics, medicine and many more. Future work includes expanding HINDSIGHT to account for LSTM architectures with more than three hidden layers as well as including BOA to the hyperparameter optimization algorithms list.

[5]https://github.com/acmmmsys/2018-HINDSIGHT

6 ACKNOWLEDGMENTS

This work is funded by the Norwegian Research Council project No. 250679 (MEMBRANE).

REFERENCES

[1] Özgü Alay, Andra Lutu, Rafael García, Miguel Peón Quirós, Vincenzo Mancuso, Thomas Hirsch, Kristian Evensen, Audun Hansen, Stefan Alfredsson, Jonas Karlsson, Anna Brunstrom, Ali Safari Khatouni, Marco Mellia, and Marco Ajmone Marsan. 2017. Experience: An Open Platform for Experimentation with Commercial Mobile Broadband Networks. *MobiCom '17* (2017).

[2] Bram Bakker. 2002. Reinforcement learning with long short-term memory. In *Advances in neural information processing systems*. 1475–1482.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5, 2 (1994), 157–166.

[4] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.

[5] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems*. 1223–1231.

[6] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. 2015. Speeding Up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves.. In *IJCAI*, Vol. 15. 3460–8.

[7] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. 2002. Learning precise timing with LSTM recurrent networks. *Journal of machine learning research* 3, Aug (2002), 115–143.

[8] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. 2009. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence* 31, 5 (2009), 855–868.

[9] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 6645–6649.

[10] Alex Graves and Jürgen Schmidhuber. 2009. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*. 545–552.

[11] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. 2015. DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623* (2015).

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[13] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[15] Dougal Maclaurin, David Duvenaud, and Ryan Adams. 2015. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*. 2113–2122.

[16] Cise Midoglu, Leonhard Wimmer, Andra Lutu, Özgü Alay, and Carsten Griwodz. 2018. MONROE-Nettest: A Configurable Tool for Dissecting Speed Measurements in Mobile Broadband. *Infocom CNERT* (2018).

[17] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

[18] Martin Pelikan, David E Goldberg, and Erick Cantú-Paz. 1999. BOA: The Bayesian optimization algorithm. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*. Morgan Kaufmann Publishers Inc., 525–532.

[19] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[20] Miguel Peon-Quiros, Vincenzo Mancuso, V. Comite, Andra Lutu, Özgü Alay, Stefan Alfredsson, Jonas Karlsson, Anna Brunstrom, Marco Mellia, Ali Safari Khatouni, and Thomas Hirsch. 2017. Results from Running an Experiment as a Service Platform for Mobile Networks. *MobiCom WiNTECH* (2017).

[21] Lutz Prechelt. 2012. Early stopping?but when? In *Neural networks: tricks of the trade*. Springer, 53–67.

[22] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.

[23] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4, 2 (2012), 26–31.