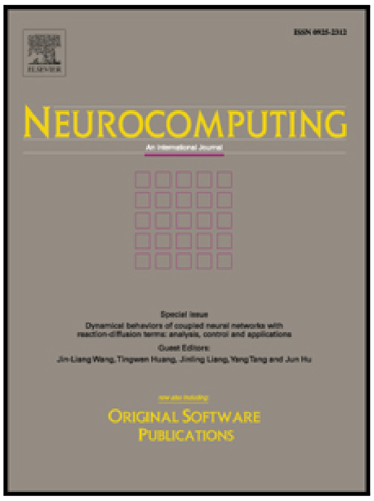
Journal Pre-proof



A Non-parametric Softmax for Improving Neural Attention in Time-Series Forecasting

Simone Totaro, Amir Hussain, Simone Scardapane

PII: S0925-2312(19)31526-7
DOI: https://doi.org/10.1016/j.neucom.2019.10.084
Reference: NEUCOM 21479

To appear in: *Neurocomputing*

Received date: 20 March 2019
Revised date: 8 October 2019
Accepted date: 22 October 2019

Please cite this article as: Simone Totaro, Amir Hussain, Simone Scardapane, A Non-parametric Softmax for Improving Neural Attention in Time-Series Forecasting, *Neurocomputing* (2019), doi: https://doi.org/10.1016/j.neucom.2019.10.084

This is a PDF file of an article that has undergone enhancements after acceptance, such as the addition of a cover page and metadata, and formatting for readability, but it is not yet the definitive version of record. This version will undergo additional copyediting, typesetting and review before it is published in its final form, but we are providing this version to give early visibility of the article. Please note that, during the production process, errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

© 2019 Published by Elsevier B.V.

Journal Pre-proof

A Non-parametric Softmax for Improving Neural Attention in Time-Series Forecasting

Simone Totaro[c], Amir Hussain[b], Simone Scardapane[a,*]

[a]*Department of Information Engineering, Electronics and Telecommunications (DIET), Sapienza University of Rome, Via Eudossiana 18, 00184 Rome, Italy*
[b]*School of Computing, Edinburgh Napier University, 10 Bankhead Terrace, Edinburgh EH11 4DY, UK*
[c]*Department of Information and Communication Technologies (DICT), Universitat Pompeu Fabra, Carrer de Tànger, 122-140, 08018 Barcelona, Spain*

Journal Pre-proof

Abstract

Neural attention has become a key component in many deep learning applications, ranging from machine translation to time series forecasting. While many variations of attention have been developed over recent years, all share a common component in the application of a softmax function to normalize the attention weights, in order to transform them into valid mixing coefficients. In this paper, we aim to improve the modeling flexibility of a generic attention module by innovatively replacing this softmax operation with a *learnable* softmax, in which the normalizing functions are also adapted from the data. Specifically, our generalized softmax builds upon recent work in learning activation functions for deep networks, in particular the kernel activation function and its extensions. We describe the application of the proposed technique for the case of time series forecasting with the dual-stage attention-based recurrent neural network (DA-RNN), a state-of-the-art

*Corresponding author. Phone: +39 06 44585495, Fax: +39 06 4873300.
Email address: simone.scardapane@uniroma1.it (Simone Scardapane)

Journal Pre-proof

model for predicting time series that employs two different attention modules for handling exogenous factors and long-term dependencies. A series of real-world benchmarks are used to show that simply plugging-in our generalized attention model can improve results on all datasets, even when keeping the number of trainable parameters in the model constant. To further evaluate the algorithm, we collect a novel dataset for the prediction of the Bitcoin closing exchange rate, a problem of high practical significance lately. Finally, to foster research in the topic, we also release both the dataset and our model as an open source extensible library. Over a baseline DA-RNN, our proposed model delivers an improvement of MAR ranging from 6% to 15% using our newly-released dataset.

Keywords: Attention, Activation function, Softmax, Time series forecasting

Journal Pre-proof

1. Introduction

Many of the advancements in deep learning over the last few years have revolved around the concept of *neural attention*, first introduced in [4]. Attention was originally developed to alleviate one of the major drawbacks of encoder-decoder architectures for sequence to sequence problems [31, 7, 41]. In these models, the input sequence is processed by an encoding recurrent neural network (RNN), while a second RNN produces the desired output sequence (e.g., the translation of the input sequence into a different language). However, in the basic setup the decoder has only access to the last state of the input RNN, lacking more fine-grained control on the information to be accessed during decoding. Attention solves this problem by allowing the

Journal Pre-proof

decoder to focus at every time-step on different parts of the input sequence, by computing dynamically a set of attention weights that determine the importance of each of the parts [14, 27, 8].

Later, it was understood that this concept could be extended to a much larger set of problems, and attention modules have been applied in fields as diverse as sentence summarization [27], graph neural networks [33], time series forecasting [26], generative models [13, 39], combinatorial problems [34], collaborative filtering [11], image processing [36], neural processes [18], and many more [20]. State-of-the-art architectures for neural machine translation are now composed almost exclusively of attention layers [32], and these models are also being extended to more generic sequence modeling tasks [9].

Roughly speaking, an attention layer is composed of three steps, explained more in detail later on in Section 3: (a) computation of a set of weight vectors to decide how much attention to provide to each element in the input sequence; (b) normalization of these weights (typically via a softmax nonlinearity); and (c) construction of an aggregated vector using these normalized weights. A lot of literature has been devoted to improving point (a), by devising different methods of computing these coefficients, most notably the multi-head model from [32].

In this paper, we propose to investigate whether the performance of neural attention mechanisms can be further improved by focusing on point (b), the normalization of the coefficients. In particular, we take inspiration from previous work which considered a similar setup in the context of gates inside recurrent neural networks [29], extending work on learning activation functions [30]. The basic idea is that we can improve learning in neural net-

Journal Pre-proof

Journal Pre-proof

works by replacing fixed nonlinearities (such as the softmax) with flexible ones, at the same time simplifying optimization and possibly increasing the accuracy of resulting models. Note that there is growing literature showing that learning standard activation functions (e.g., the ReLU) can be beneficial [21, 42, 1, 25, 30]. However, learning more structured nonlinearities, such as the softmax, has been rarely explored as of today.

To this end, in this paper, each component of the softmax is replaced with a *generalized softmax* function, whose shape can be learnt from the data during the optimization phase. This flexibility is achieved with the addition of a very small number of additional parameters, in the form of an efficient kernel expansion at every component of the nonlinearity. To provide a practical demonstration, we show that simply plugging-in our module in a state-of-the-art model for short-term time series forecasting allows it to learn faster and more accurately on multiple datasets.

Contributions of the paper

The main contribution of the paper is a novel, more general formulation of neural attention, wherein the softmax function is replaced with a trainable softmax. This trainable softmax leverages the kernel activation function (KAF) described in [30], a non-parametric activation function where each scalar function is modeled as a one-dimensional kernel expansion. To extend the KAF for our problem, we modify it in an appropriate fashion in order to preserve key properties of the softmax, most notably the positivity and normalization of its output values.

In order to evaluate the proposal, it is applied to a use case of time series forecasting [40]. The dual-attention RNN (DA-RNN) [26] is a state-of-

Journal Pre-proof

Journal Pre-proof

the-art network for short-term time series forecasting that incorporates two attention modules to, respectively, decide how much weight is required to provide a single input and how much weight to provide a past time instant. In our implementation, we replicate the DA-RNN by substituting two softmax operations with the trainable version. The model is evaluated using two datasets from [26], which show that the model can converge faster to a higher level of accuracy. We also compare its performance on a third dataset that we collect specifically for this paper, on the short-term forecasting of Bitcoin closing prices, a task of significant practical interest lately, e.g., [17, 2, 3]. Finally, we release our code and Bitcoin dataset as an open-source Python library.

Organization of the paper

Section 2 briefly describes key related works from the last few years. Section 3 introduces in detail neural attention mechanisms and their variants. Next, in Section 4 we describe our main contribution, i.e., the generalized softmax mechanism. Section 5 details our use case, specifically short-term time series forecasting, together with the DA-RNN architecture. We perform extensive experimental evaluation in Section 6, before concluding in Section 7.

Notation

Throughout the paper, bold lowercase letters are used for vectors, e.g., $\mathbf{x}$, and bold uppercase letters for matrices, e.g., $\mathbf{A}$. v_i denotes the ith (scalar) entry of vector $\mathbf{v}$. When considering a temporal dependency (e.g., for RNNs),

Journal Pre-proof

Journal Pre-proof

$\mathbf{x}(t)$ is used to denote the vector or scalar at time-instant t. All symbols and parameters are explained when introduced for the first time.

2. Related works

Early works in time series forecasting using machine learning models can be found in, e.g., [19, 38]. More recently, the explosion of interest in deep learning techniques fostered the development of deep networks for the problem, either based on recurrent (autoregressive) models, like in our case [10], or using convolutive networks or some variants [5]. The DA-RNN model we consider in this paper [26] combines a recurrent component with multiple so-called attention modules [4] to scale the model to a large set of external (exogenous) time series, and to alleviate the problem of long-term dependencies. However, their employed attention mechanism is relatively simple, and the architecture sometimes failed to improve over a baseline neural network. To overcome this, in this paper we improve on this attention mechanism by increasing the flexibility of the softmax function, resulting in a novel attention-based module which can be further extended to other architectures (as discussed in Section 7).

The KAF, which we use as starting point to build our novel softmax formulation, was introduced in [30]. Each activation function in a KAF network is replaced with a cheap kernel expansion, allowing for a small number of trainable parameters and a much wider flexibility. Adapting activation functions is a topic that has received considerable interest since the introduction of simple adaptable versions of the ReLU [15]. Differently from these, the KAF is a fully non-parametric model, where the flexibility of each acti-

Journal Pre-proof

Journal Pre-proof

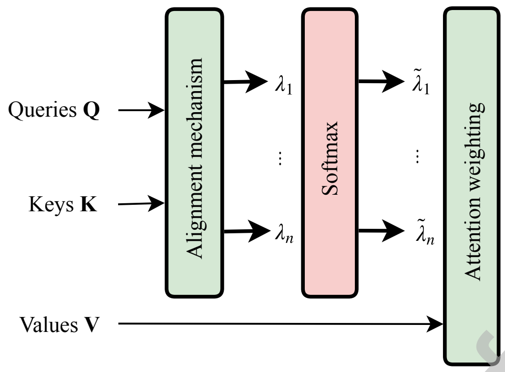


Figure 1: Schematic depiction of a neural attention module. The softmax operation is highlighted in light red.

vation can be controlled with a hyper-parameter by the user. Related works in this field include the adaptive piecewise linear (APL) unit [1], where each activation function is replaced by a sum of ReLU functions, or the max-out [42], where each activation takes the maximum across multiple independent branches of the network (see [30] for a full comparison of all these approaches).

3. Neural attention mechanism

We start by describing a generic mechanism for neural attention, first introduced in [4] for neural machine translation and later extended in a number of papers, including [23] (dot-product attention), [32] (multi-head attention), and others. The mechanism is also schematically summarized in Fig. 1.

Roughly speaking, attention is a way of selecting (or *attending*, in the original terminology) information from a set of key/values pairs, based on the content of one or more *queries*, in a differentiable fashion, where keys, values, and queries are all represented as vectors of reals numbers. More

Journal Pre-proof

Journal Pre-proof

in detail, suppose we have n key vectors $\mathbf{K} = \left[\mathbf{k}_1^T, \ldots, \mathbf{k}_n^T\right]^T \in \mathbb{R}^{n \times K}$, each one represented as a vector of dimensionality K, and corresponding values $\mathbf{V} = \left[\mathbf{v}_1^T, \ldots, \mathbf{v}_n^T\right]^T \in \mathbb{R}^{n \times V}$, each of dimension V. We consider the simplified case where we have a single query vector $\mathbf{q} \in \mathbb{R}^K$, but everything extends immediately to having multiple queries stacked into a matrix. The idea of attention is to select the most important values among all n, based on the similarity between their corresponding keys and our query, all in a differentiable form so that back-propagation is possible.

As shown in Fig. 1, we start with an *alignment* mechanism that evaluates the aforementioned similarity, providing us with a set of unnormalized alignment coefficients λ_i, $i = 1, \ldots, n$. This is where most of the attention mechanisms proposed in the literature differ. The original formulation in [4] used what is known as *additive attention*, i.e., a single-hidden layer feedforward network with adaptable coefficients:

$$\lambda_i = \operatorname{align}\left(\mathbf{k}_i, \mathbf{q}\right) = \mathbf{w}^T \tanh\left(\mathbf{W}\left[\mathbf{k}_i; \mathbf{q}\right]\right) , \tag{1}$$

where we use $[\ldots]$ to denote vector concatenation, and $\mathbf{w}$, $\mathbf{W}$ are parameters of the model. A simplified *dot-product attention* was later proposed in [23], where the alignment is computed as the dot-product between the key and the query:

$$\lambda_i = \operatorname{align}\left(\mathbf{k}_i, \mathbf{q}\right) = \mathbf{k}_i^T \mathbf{q} \, . \tag{2}$$

This form of attention is simpler to implement, as all coefficients $\boldsymbol{\Lambda} = \left[\lambda_1, \ldots, \lambda_n\right]^T \in \mathbb{R}^n$ can be obtained as $\boldsymbol{\Lambda} = \mathbf{K}\mathbf{q}$, but may loose expressiveness due to the lack of adaptable coefficients.

Journal Pre-proof

Journal Pre-proof

Irrespective of how the alignment coefficients are obtained, one needs to normalize these in order to use them to appropriately select information from the original set. In the majority of attention implementations, this is obtained via the application of a softmax nonlinearity, highlighted in light red in Fig. 1:

$$\tilde{\lambda}_i = \text{softmax}(\lambda_i) = \frac{\exp(\lambda_i)}{\sum_{j=1}^{n} \exp(\lambda_j)} . \tag{3}$$

Due to the properties of the softmax operation, the attention weights $\tilde{\lambda}_i$ lie in the standard simplex and can be used to represent how much weight should be assigned to each original value vector $\mathbf{v}_i$, $i = 1, \ldots, n$. For example, one can now combine all original vectors into a fixed-size representation of dimensionality V by weighted averaging:

$$\bar{\mathbf{v}} = \sum_{i=1}^{n} \tilde{\lambda}_i \mathbf{v}_i \, . \tag{4}$$

Depending on the origin of the keys, values, and queries, attention layers can be used for a variety of tasks. For example, *self-attention* layers, where $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are all linear projections of the same input matrix, are the basic building block of transformer architectures [32].

Journal Pre-proof

4. Proposed attention mechanism with generalized softmax

As stated in the introduction, our aim in this paper is to improve the generic attention layer described in the previous section to incorporate a more flexible softmax mechanism, to provide the model with more degrees-of-freedom during training and simplify optimization. In this section, we first

Journal Pre-proof

introduce the KAF, as originally described in [30], before moving on to our KAF-based generalized softmax formulation.

4.1. Kernel activation function: basic formulation

The KAF is designed as a trainable activation function, whose parameters are adapted per-neuron during the optimization process. More specifically, a KAF is described by a one-dimensional kernel expansion:

$$g(s) = \text{KAF}(s) = \sum_{i=1}^{D} \alpha_i \kappa\left(s, d_i\right) \,, \tag{5}$$

where s is an activation value and κ is a user-specified kernel function. To simplify the implementation, the points d_i over which the kernel is evaluated (the dictionary) are fixed before training by sampling D values uniformly around 0 with a predetermined interval, while the α_i coefficients are adapted separately for every neuron. Differently from other proposals for adapting the activation function that only introduce a *fixed* number of adaptable parameters (e.g. [15]), D in (5) is a hyper-parameter controlling the complexity (and, consequently, the flexibility) of every KAF. Since the number of adaptable parameters is also controlled by D, we call the approach non-parametric.

One has a large variety in the choice of the kernel function, leading to a wide range of different properties for the actual implementation. For example, [30] uses one-dimensional Gaussian kernels given by:

$$\kappa(s, d_i) = \exp\left\{-\gamma\left(s - d_i\right)^2\right\} \,, \tag{6}$$

where γ is another hyper-parameter whose choice is analyzed in detail in [30]. In this case, each KAF is a mixture-of-Gaussians where every mixing

Journal Pre-proof

Journal Pre-proof

coefficient α_i has a limited receptive field determined by γ. Alternatively, one can use ReLU-like kernels, leading to a scheme similar to the one proposed independently in [1, 25]:

$$\kappa(s, d_i) = \max\left(0, s - d_i\right) \,, \tag{7}$$

More generally, it is possible to use linear combinations of base kernels whose weights are in turn adapted independently from the data [28].

4.2. Learnable softmax functions with KAFs

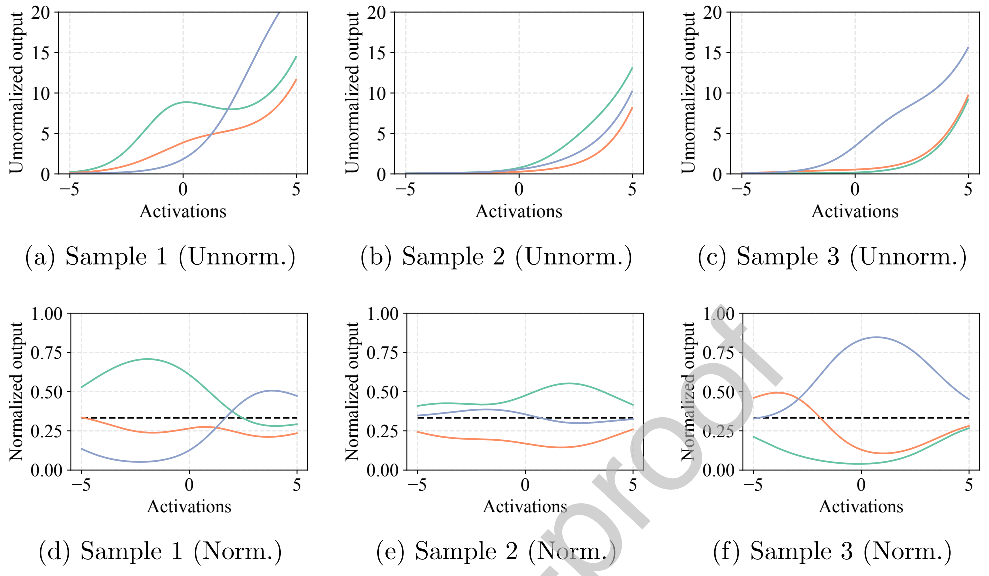
The basic idea of our proposal is to replace each component of the softmax operation in (3) with a set of *trainable* scalar functions based on KAFs. These provide the model with more flexibility in its weighting scheme. However, note that directly replacing the softmax operation in (3) with KAFs is not feasible, since we loose two key properties of the softmax, namely the positivity and normalization of the outputs. In addition, when using a Gaussian kernel, the output of the KAF will quickly decrease to zero outside the range of the dictionary. Similar problems were encountered in [29] when tackling learnable gates in a recurrent architecture.

While both problems could be solved by introducing appropriate constraints on the α_i coefficients, a simpler solution is to innovatively combine KAFs and the softmax as follows:

$$\text{KAF}_{\text{softmax}}(\lambda_i) = \frac{\exp\left(\frac{1}{2}\text{KAF}(\lambda_i) + \frac{1}{2}\lambda_i\right)}{\sum_{j=1}^{n} \exp\left(\frac{1}{2}\text{KAF}(\lambda_j) + \frac{1}{2}\lambda_j\right)} \tag{8}$$

Journal Pre-proof

Journal Pre-proof



(a) Sample 1 (Unnorm.) (b) Sample 2 (Unnorm.) (c) Sample 3 (Unnorm.)

(d) Sample 1 (Norm.) (e) Sample 2 (Norm.) (f) Sample 3 (Norm.)

Figure 2: Three random samples of our proposed generalized softmax function, with $n = 3$ and $D = 10$ values of the dictionary sampled uniformly between -5 and $+5$. In all plots we evaluate the softmax with all three inputs set to the value on the abscissa and shown with different colors. (a-c) Unnormalized values. (d-f) Normalized values (with the output of a standard softmax shown with a dashed black line).

where λ_i are the coefficients in (3). Like in a standard softmax, exponentiation guarantees the positivity of the outputs, while the denominator ensures that the weights sum to one. Inspired by [29], we also add a set of residual connections inside the exponential so that, when the output of the KAF is zero, the function will converge to a classical softmax function.

We show three random samples of the proposed generalized softmax in Fig. 2. For each column, we fix the dictionary by sampling 10 values uniformly around -5 and $+5$, and we sample randomly the mixing coefficients from a normal distribution. All plots are shown by applying the function to

Journal Pre-proof

Journal Pre-proof

a three-dimensional vector with all components equal to the value in the x-axis. The first row of plots (Fig. 2 (a)-(c)) shows the unnormalized outputs, before applying the normalization in the denominator, while the second row (Fig. 2 (d)-(f)) shows the corresponding output after normalization. Note that a standard softmax would simply transform the inputs to $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$ irrespective of their value (shown with a dashed black line in the second row), while our proposed function can learn a range of potentially more expressive shapes. These go from simply giving more weight to one feature over the others, e.g., Fig. 2c, or giving weight in a non-uniform fashion depending on their specific value, e.g., Fig. 2a.

5. Time series forecasting with dual-stage attention

To provide a concrete use case, we apply our proposed attention layer to the state-of-the-art DA-RNN framework for time series forecasting [26]. We first introduce the problem formulation and a simplified RNN architecture in Section 5.1, before moving on to the DA-RNN and our proposed model in Section 5.2.

5.1. Forecasting with encoder/decoder networks

Suppose that at a certain time-step T we have observed the last $T-1$ values $y(1), \ldots, y(T-1)$ of a target time series. In addition, we have also observed the last T values (up to the current time) $\mathbf{x}(1), \ldots, \mathbf{x}(T)$ of d exogenous inputs that we may assume correlated with the target time series, i.e., $\mathbf{x}(t) \in \mathbb{R}^d$ represent auxiliary information at time t for the forecasting method (e.g., weather conditions). The task is to learn a mapping between all the observed values and the next time-step of the target time series:

Journal Pre-proof

Journal Pre-proof

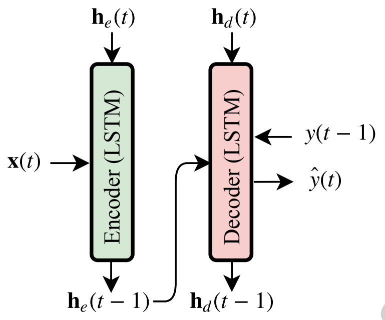


Figure 3: Encoder/decoder formulation for forecasting of time series (with no attention).

$$\hat{y}(T) = F\left(\mathbf{x}(1), \ldots, \mathbf{x}(T), y(1), \ldots, y(T-1)\right) \approx y(T)\,. \tag{9}$$

While there is a vast literature on the problem of learning $F(\cdot)$ [35], we focus here on a recent class of state-of-the-art methods that are based on an encoder/decoder architecture with RNNs [10], which have been inspired by previous work on sequence to sequence modeling with deep networks mentioned in the introduction [31]. Apart from their flexibility and accuracy, this type of models can easily handle several heterogeneous driving inputs and, with a small modification of their training procedure, multi-step-ahead prediction and generative applications [10].

The architecture is described in Fig. 3 and is composed of two sequential RNN models, one for processing the driving time series, and the second for predicting the target one. First, an encoder RNN (shown in green in Fig. 3) is applied iteratively at every time-step t on the current observations $\mathbf{x}(t)$:

$$\mathbf{h}_e(t) = \text{RNN}_{\text{enc}}\left(\mathbf{x}(t), \mathbf{h}_e(t-1)\right)\,, \tag{10}$$

Journal Pre-proof

Journal Pre-proof

where $\mathbf{h}_e(t-1)$ is a vector containing the internal state of the RNN at the previous time step. In practice, we implement (10) as an LSTM network [16] following [26], but other choices are possible. One could also substitute the recurrent encoder with an autoregressive convolutional network as in [5], although this is out of the scope of this paper. Roughly speaking, the vector $\mathbf{h}_e(t)$ can be interpreted as a fixed-size embedding of the past observations of the driving inputs.

This embedding is fed as input to another RNN model for *decoding*, together with the past value $y(t-1)$ of the target time series:

$$\mathbf{h}_d(t) = \text{RNN}_{\text{dec}}\left(y(t-1), \mathbf{h}_e(t), \mathbf{h}_d(t-1)\right) , \tag{11}$$

where $\mathbf{h}_d(t-1)$ is the internal state of the decoder at the previous time step. The decoder is implemented with a separate LSTM network with independent parameters. Finally, one can obtain a prediction for the target time series by feeding the new decoder state at the last time-step to a feedforward model:

$$\hat{y}(T) = \mathbf{v}^T \tanh\left(\mathbf{W}\mathbf{h}_d(T) + \mathbf{b}\right) , \tag{12}$$

where $\mathbf{v}$, $\mathbf{W}$ and $\mathbf{b}$ are also learned. The overall architecture can be trained end-to-end by minimizing the mean-squared error or mean-absolute error over a set of known time series [26].

5.2. Dual-stage attention

Despite its good empirical performance, the standard encoder/decoder formulation suffers from two drawbacks. Firstly, increasing the number of driving inputs might damage the forecasting process, especially when the

Journal Pre-proof

Journal Pre-proof

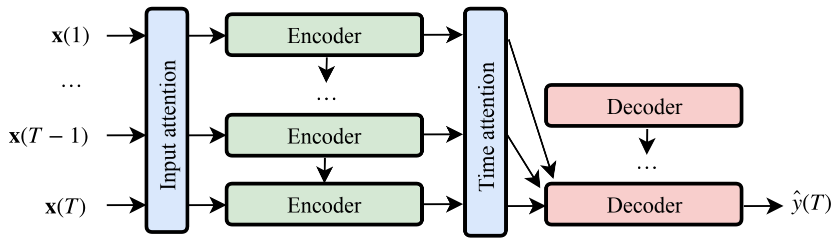


Figure 4: DA-RNN model. Compared to Fig. 3, DA-RNN adds two attention layers, denoted as *input attention* and *time attention* in the diagram. The decoder is shown operating on time-step T.

information contained in them is redundant or noisy. Secondly, at time step t only the encoder's hidden state at the current step is used, while the other steps are discarded (this is a known problem in sequence to sequence models [4]).

The DA-RNN [26] solves both problems by including two different attention layers, respectively after the input and between the encoder and the decoder, to modulate in a selective way the information flowing in the system. This is shown schematically in Fig. 4.

The first attention layer, denoted as *input attention* in Fig. 4, enables selective weighting of the input components at every time-step. This is done by computing a set of input attention coefficients with a variation of the additive alignment model in Eq. (1):

$$\lambda_k(t) = \mathbf{v}_e^T \tanh\left(\mathbf{W}_e \mathbf{h}_e(t-1) + \mathbf{U}_e \mathbf{x}_k\right) \,, \tag{13}$$

where $\mathbf{h}_e(t-1)$ is the previous state of the encoder, $\mathbf{x}_k \in \mathbb{R}^T$ is a vector

Journal Pre-proof

Journal Pre-proof

containing all T values of the kth driving input[1], and $\mathbf{v}_e$, $\mathbf{W}_e$, and $\mathbf{U}_e$ are trainable parameters.

In the original formulation, these input attention coefficients are processed with a softmax function, as described in Section 3, to obtain a set of normalized coefficients $\widetilde{\boldsymbol{\lambda}}(t) = \left[\tilde{\lambda}_1(t), \ldots, \tilde{\lambda}_d(t)\right]^T$. In our proposed implementation in this paper, instead, we replace this softmax operation with a trainable one, as described in Section 4.

Finally, this vector (regardless of how it is computed) is used to selectively amplify a few driving inputs:

$$\widetilde{\mathbf{x}}(t) = \widetilde{\boldsymbol{\lambda}}(t) \odot \mathrm{x}(t)\,, \tag{14}$$

where $\odot$ represents the elementwise product. $\widetilde{\mathbf{x}}(t)$ instead of $\mathbf{x}(t)$ is now used as input to the encoder RNN.

For the decoder part, the DA-RNN uses a similar mechanism (denoted as *time attention* in Fig. 4) to compute a set of T attention coefficients corresponding to all the encoder states:

$$\sigma_k(t) = \mathbf{v}_d^T \tanh\left(\mathbf{W}_d \mathbf{h}_d(t-1) + \mathbf{U}_d \mathbf{h}_e(k)\right)\,. \tag{15}$$

Like before, we can process these coefficients either with a standard softmax function as in the original implementation, or with a trainable one, as in the proposed implementation. Anyway, the resulting normalized coefficients

[1]We follow here the original derivation of the DARNN in [26]. In order to use this model in an autoregressive fashion at every time-step, this vector can be replaced with a sliding window of fixed length.

Journal Pre-proof

Journal Pre-proof

$\tilde{\sigma}_1(t), \ldots, \tilde{\sigma}_T(t)$ are used to build a context vector $\mathbf{c}(t)$ as a linear combination of all encoder's hidden states, similar to (4):

$$\mathbf{c}(t) = \sum_{i=1}^{T} \sigma_i(t)\mathbf{h}_e(i)\,. \tag{16}$$

Another projection provides an updated value for feeding the decoder:

$$\widetilde{y}(t-1) = \mathbf{w}^T\left[y(t-1); \mathbf{c}(t)\right] + \widetilde{b}\,. \tag{17}$$

$\widetilde{y}(t-1)$ is now used in place of $y(t-1)$ to feed the decoder in (11). The readout is also modified to read:

$$\hat{y}(T) = \mathbf{v}^T \tanh\left(\mathbf{W}\left[\mathbf{h}_d(T); \mathbf{c}(T)\right]\right)\,. \tag{18}$$

To summarize, our proposed modification of the DA-RNN considers the same overall structure as the original paper described here, but the two softmax operations in the input and time attention layers are replaced with our generalized softmax described in Section 4.

6. Experimental evaluation

Here we consider three challenging real-world time series datasets, two of which were also considered in the original DA-RNN paper [26], while the third one is collected especially for this experimental evaluation. The next subsections describe the benchmark datasets, the architectures employed, and comparative results. For the purpose of repeatability, we also release all our source code in the form of a time series analysis open-source library in

Journal Pre-proof

Journal Pre-proof

Python.[2]

6.1. Datasets

SML 2010 [37] is a dataset containing 4137 measurements of a set of 24 sensors in a house equipped with domotic capabilities. Measurements were collected every minute and later smoothed using a 15-minute running mean. Following [26] we use the initial 3000 points in the dataset for training, another 400 for validation, and the rest for testing. The task is to perform short-term forecasting of the room indoor temperature, following measurements of the additional 23 driving time series. Some of these time series were found to have constant values over the entire dataset and were thus excluded, resulting in 16 inputs to the encoder RNN.

The second dataset, NASDAQ 100,[3] was collected in [26] by storing minute-by-minute stock prices of 81 corporations included in the NASDAQ 100 index, and the index itself as the target time series. In total, 391 measurements are collected from July 26, 2016, to December 22, 2016, except for November 25 and December 22. Also following [26] we use the initial 35100 points for training, 2730 for validation, and the rest for testing.

For the third dataset, we evaluate the model on a realistic task of predicting the closing Bitcoin/USD price exchange, based on the closing values of 7 crypto exchange rates with Bitcoin: Ethereum, Litecoin, Monero, Stellar, NeosCoin, and Dash. We collected data on the Poloniex website,[4] with a frequency of 30 minutes, starting from 8th August 2015 to 31st July 2018.

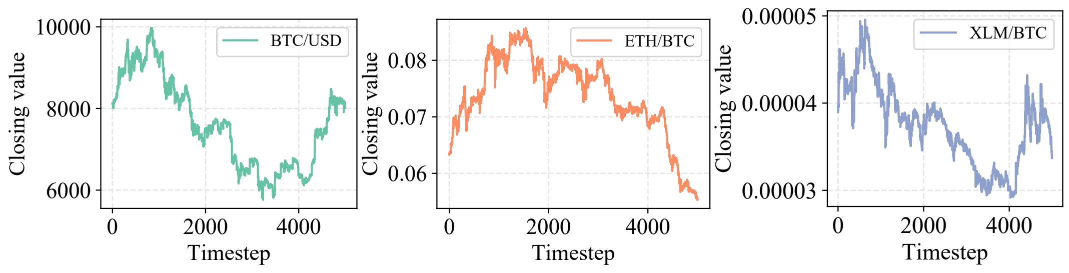
Journal Pre-proof

[2]`https://github.com/d3sm0/ntsa`

[3]`http://cseweb.ucsd.edu/~yaq007/NASDAQ100_stock_data.html`

[4]`https://poloniex.com/`

Journal Pre-proof



(a) Bitcoin/USD (b) Ethereum/Bitcoin (c) Stellar/Bitcoin

Figure 5: Sample of our Bitcoin dataset: closing exchange prices for the last 5000 time steps of (a) Bitcoin/USD, (b) Ethereum/Bitcoin, and (c) Stellar/Bitcoin.

Name	Driving time series	Training size	Validation size	Test size
SML 2010	16	3200	400	537
NASDAQ 100	81	35100	2730	2730
BITCOIN	7	41788	NA	10447

Table 1: Description of the datasets.

The overall dataset is composed of 52235 observations split in training and test set with a constant proportion of 0.8. A sample of the last 500 values of the test part for the target time series and two corresponding exogenous time series are shown in Fig. 5. Instructions to download the dataset are provided in our open-source library.

In all cases, features are first normalized with an affine transformation to lie in the $[0, 1]$ range. A summary of the datasets is provided in Table 1.

Journal Pre-proof

Journal Pre-proof

6.2. Architectures and training

From the datasets described in the last section, all possible training sequences of length $T = 10$ are extracted. Our main strong baseline is the original DA-RNN from [26], which was already shown to outperform several state-of-the-art short-term forecasting models, including ARIMA [24] and an autoregressive version of the RNN [6]. We use the same final setup from [26], where hyper-parameters were already heavily fine-tuned to maximize performance. Both the encoder and the decoder are built with LSTM networks having inner dimensionality of 68 and 128 respectively. Overall, our aim is to show that our trainable softmax can improve performance even when used as a simple drop-in replacement of the standard softmax, in a heavily fine-tuned model.

To this end, for the proposed version of the DA-RNN we simply replace both softmax operations with the novel, flexible formulation described in Section 4. For the KAFs, we use 20 dictionary elements equispaced from -4.0 to 4.0, and initialize their mixing coefficients from a normal distribution, similar to prior works. For completeness, we also consider another baseline using the encoder-decoder architecture from Section 5.1 without attention mechanisms, similar to the model described in [31].

For training, the Adam optimizer is used with mini-batches of 128 elements, reducing the learning rate of a factor 1/10 from 10^{-3} every 10000 iterations. We optimize the mean-absolute error over the training set, repeating every experiment 5 times. Since all hyper-parameter tuning was already done in [26], for our experiments we combine training and validation sets into a single set.

Journal Pre-proof

Journal Pre-proof

6.3. Results and discussion

To evaluate the algorithms in a comprehensive way, we consider five different error measures. Assuming we have M desired forecasts $y_1, \ldots, y_M$ and M predictions $\hat{y}_1, \ldots, \hat{y}_M$, we compute:

- **Mean-squared error** (RMSE):

$$\text{RMSE} = \frac{1}{M}\sum_{i=1}^{M} (y_i - \hat{y}_i)^2 \,. \tag{19}$$

- **Mean absolute error** (MAE), which is the metric we optimize for during training:

$$\text{MAE} = \frac{1}{M}\sum_{i=1}^{M} |y_i - \hat{y}_i| \,. \tag{20}$$

- **Mean absolute percentage error** (MAPE):

$$\text{MAE} = \frac{1}{M}\sum_{i=1}^{M} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \,. \tag{21}$$

- **Symmetric mean absolute percentage error**, which is similar to MAE but can be interpreted as a percentage:

$$\text{MAPE} = \frac{2}{M}\sum_{i=1}^{M} \frac{|y_i - \hat{y}_i|}{|y_i| + |\hat{y}_i|} \,. \tag{22}$$

- **R-squared coefficient** (R^2), which can be interpreted as the proportion of variance explained by the predictive method:

$$\text{R}^2 = 1 - \frac{\sum_{i=1}^{M} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{M} (y_i - \bar{y})^2} \,, \tag{23}$$

  where $\bar{y}$ is the empirical mean of the desired values.

Journal Pre-proof

Journal Pre-proof

Dataset	**Algorithm**	**MSE**	**MAE**	**SMAPE**	**MAPE**	$\mathrm{R}^{\mathbf{2}}$
	Encoder-Decoder	0.091 ± 0.032	0.096 ± 0.020	0.140 ± 0.028	1.101 ± 0.510	0.907 ± 0.067
SML	DA-RNN	0.061 ± 0.024	$\mathbf{0.077 \pm 0.018}$	0.124 ± 0.030	0.888 ± 0.431	$\mathbf{0.940 \pm 0.049}$
	Proposed DA-RNN	$\mathbf{0.060 \pm 0.022}$	$\mathbf{0.077 \pm 0.016}$	$\mathbf{0.113 \pm 0.028}$	$\mathbf{0.862 \pm 0.433}$	$\mathbf{0.940 \pm 0.050}$
	Encoder-Decoder	0.092 ± 0.009	0.186 ± 0.007	0.281 ± 0.011	0.218 ± 0.052	0.921 ± 0.044
NASDAQ	DA-RNN	0.077 ± 0.008	$0.164 \pm 0.0.006$	0.204 ± 0.008	0.211 ± 0.047	0.926 ± 0.047
	Proposed DA-RNN	$\mathbf{0.065 \pm 0.009}$	$\mathbf{0.135 \pm 0.006}$	$\mathbf{0.169 \pm 0.008}$	$\mathbf{0.178 \pm 0.050}$	$\mathbf{0.934 \pm 0.049}$
	Encoder-Decoder	0.103 ± 0.011	0.181 ± 0.006	0.266 ± 0.008	5.580 ± 0.877	0.863 ± 0.049
BITCOIN	DA-RNN	$\mathbf{0.056 \pm 0.011}$	0.096 ± 0.005	0.154 ± 0.008	$\mathbf{2.617 \pm 0.927}$	0.932 ± 0.048
	Proposed DA-RNN	0.057 ± 0.011	$\mathbf{0.091 \pm 0.005}$	$\mathbf{0.146 \pm 0.008}$	2.645 ± 0.943	$\mathbf{0.941 \pm 0.050}$

Table 2: Results of the three models on datasets. Details of error measures are given in the text. For each combination, mean and standard deviation are shown over different runs. The best performing algorithm in each column is highlighted in bold.

Journal Pre-proof

Results in term of these error measures (both mean and standard deviation over the runs) are given in Table 2. It can be clearly seen that the proposed variant of DA-RNN provides better results in practically all cases. This is particularly pronounced for the NASDAQ-100 dataset, where we obtain a 15%, 18%, 17% ans 16% improvement, respectively, in MSE, MAE, SMAPE and MAPE. Similarly, we obtain a 6% improvement in MAE and SMAPE for the BITCOIN dataset. On SML, we obtain a similar MAE as the DA-RNN, but a 9% improvement in SMAPE. We note how, in all cases, DA-RNN is significantly better than a standard encoder/decoder architecture, highlighting again the importance of incorporating the attention modules.

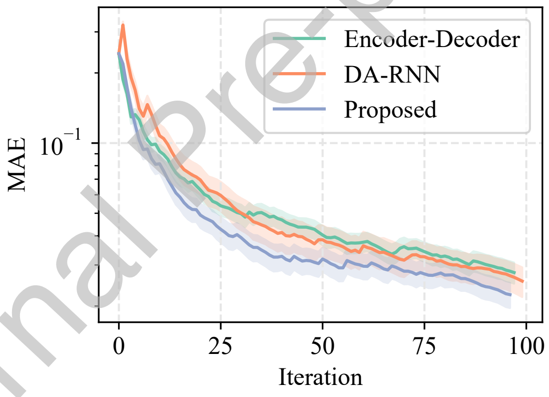


Figure 6: Sample evolution of the loss on BITCOIN dataset for the three models (zoomed over the first 100 iterations). The loss is smoothed with an exponential running mean, and standard deviation is shown with a lighter color.

Interestingly, this gain in accuracy can be obtained while simultaneously making the optimization process faster. We show in Fig. 6 a sample evolution of the loss for the three models on the BITCOIN dataset, zoomed over the first 100 iterations. We can see that the loss decreases faster for

Journal Pre-proof

Journal Pre-proof

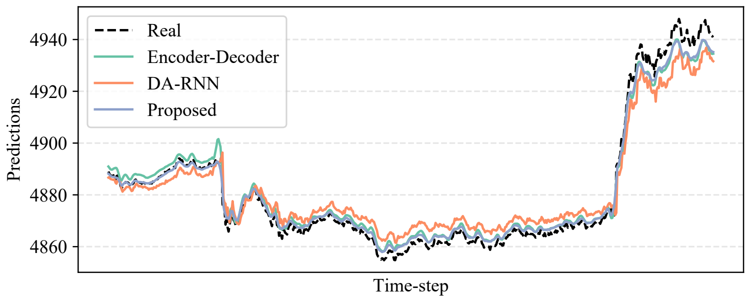


Figure 7: Predictions of the trained models on 500 time-steps from the test portion of the BITCOIN dataset.

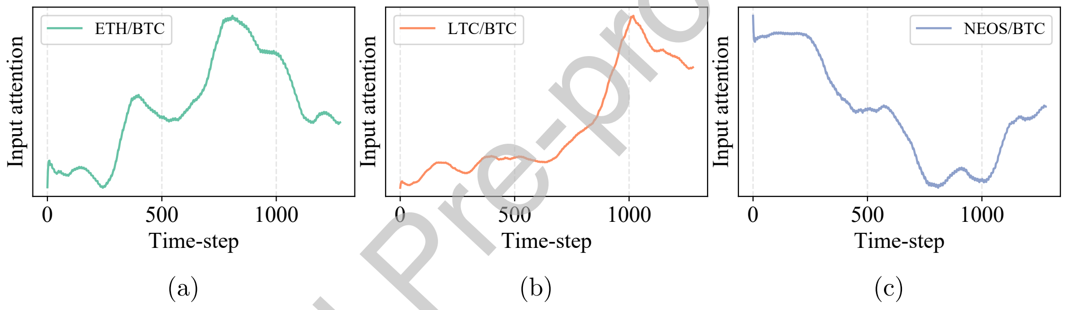


Figure 8: Samples of the input attention weights on the BITCOIN dataset for three driving time series.

the proposed model, with a per-iteration improvement consistent across the entire optimization process. This behavior is in line with our previous works [29, 30].

We also plot a sample of the predictions from the three models using the BITCOIN dataset in Fig. 7, where the improved performance of the proposed DA-RNN is clearly visible. For completeness, we show a sample of the input attention weights on the same dataset in Fig. 8. Note that the model chooses to give more importance to different driving time series in different time-steps, further motivating the use of the attention module.

Journal Pre-proof

Journal Pre-proof

7. Conclusions and future work

Time series forecasting with deep networks is an active area of research. State-of-the-art solutions have explored the inclusion of attention-based modules for addressing the problem, however, less research has been devoted to develop more flexible, advanced attention components.

In this paper we introduce a novel variant of a classical neural attention mechanism, where the fixed softmax operation is replaced with a trainable one. To achieve this, we leverage the kernel activation function (KAF) from [30], appropriately modifying it, in order to preserve a number of necessary properties. We evaluate our proposal using a model for short-term time series forecasting, where simply plugging in our new formulation results in an increased accuracy over three different real-world tasks. In particular, our proposed model is shown to produce an improvement of MAE ranging from 6% to 15% on the three real-world datasets, compared to baseline networks.

For future work, we plan to evaluate the proposed model in other contexts where attention has been found useful, most notably the transformer architecture [32]. More generally, mechanisms inspired to attention have been used in many other neural components, such as differentiable memories [12] and algorithms for architecture search [22]. Future research will investigate whether these problems can also benefit from improved optimization and accuracy delivered by our trainable softmax function.

Journal Pre-proof

Journal Pre-proof

References

References

[1] Agostinelli, F., Hoffman, M., Sadowski, P., Baldi, P., 2014. Learning activation functions to improve deep neural networks. arXiv preprint arXiv:1412.6830.

[2] Alessandretti, L., ElBahrawy, A., Aiello, L. M., Baronchelli, A., 2018. Anticipating cryptocurrency prices using machine learning. Complexity 2018.

[3] Amjad, U., Jilani, T. A., Tariq, H., Hussain, A., 2018. A quantum based evolutionary algorithm for stock index and bitcoin price forecasting. International Journal of Advanced Computer Science and Applications 9 (9), 123–132.

[4] Bahdanau, D., Cho, K., Bengio, Y., 2015. Neural machine translation by jointly learning to align and translate. In: Proc. 2015 International Conference on Learning Representations (ICLR).

[5] Borovykh, A., Bohte, S., Oosterlee, C. W., 2017. Conditional time series forecasting with convolutional neural networks. arXiv preprint arXiv:1703.04691.

[6] Chen, S., Wang, X., Harris, C. J., 2008. Narx-based nonlinear system identification using orthogonal least squares basis hunting. IEEE Transactions on Control Systems Technology 16 (1), 78–84.

Journal Pre-proof

Journal Pre-proof

[7] Cho, K., van Merrienboer, B., Bahdanau, D., Bengio, Y., 2014. On the properties of neural machine translation: Encoder–decoder approaches. In: Proc. Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST). pp. 103–111.

[8] Choi, H., Cho, K., Bengio, Y., 2018. Fine-grained attention mechanism for neural machine translation. Neurocomputing 284, 171–176.

[9] Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., Kaiser, Ł., 2018. Universal transformers. arXiv preprint arXiv:1807.03819.

[10] Flunkert, V., Salinas, D., Gasthaus, J., 2017. Deepar: Probabilistic forecasting with autoregressive recurrent networks. arXiv preprint arXiv:1704.04110.

[11] Fu, M., Qu, H., Moges, D., Lu, L., 2018. Attention based collaborative filtering. Neurocomputing 311, 88–98.

[12] Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., et al., 2016. Hybrid computing using a neural network with dynamic external memory. Nature 538 (7626), 471.

[13] Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., Wierstra, D., 2015. Draw: a recurrent neural network for image generation. In: Proc. 32nd International Conference on International Conference on Machine Learning (ICML). JMLR. org, pp. 1462–1471.

[14] Guo, M., Zhao, Y., Zhang, C., Chen, Z., 2014. Fast object detection based on selective visual attention. Neurocomputing 144, 184–197.

Journal Pre-proof

Journal Pre-proof

[15] He, K., Zhang, X., Ren, S., Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proc. IEEE International Conference on Computer Vision (ICCV). pp. 1026–1034.

[16] Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural computation 9 (8), 1735–1780.

[17] Jang, H., Lee, J., 2018. An empirical study on modeling and prediction of bitcoin prices with bayesian neural networks based on blockchain information. IEEE Access 6, 5427–5437.

[18] Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., Teh, Y. W., 2019. Attentive neural processes. arXiv preprint arXiv:1901.05761.

[19] Kim, K.-j., 2003. Financial time series forecasting using support vector machines. Neurocomputing 55 (1-2), 307–319.

[20] Li, Y., Yang, L., Xu, B., Wang, J., Lin, H., 2019. Improving user attribute classification with text and social network attention. Cognitive Computation, 1–10.

[21] Lin, M., Chen, Q., Yan, S., 2013. Network in network. arXiv preprint arXiv:1312.4400.

[22] Liu, H., Simonyan, K., Yang, Y., 2018. Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055.

Journal Pre-proof

Journal Pre-proof

[23] Luong, M.-T., Pham, H., Manning, C. D., 2015. Effective approaches to attention-based neural machine translation. In: Proc. 2015 Empirical Methods in Natural Language Processing (EMNLP).

[24] Makridakis, S., Hibon, M., 1997. Arma models and the box–jenkins methodology. Journal of Forecasting 16 (3), 147–163.

[25] Marra, G., Zanca, D., Betti, A., Gori, M., 2018. Learning neuron non-linearities with kernel-based deep neural networks. arXiv preprint arXiv:1807.06302.

[26] Qin, Y., Song, D., Cheng, H., Cheng, W., Jiang, G., Cottrell, G. W., 2017. A dual-stage attention-based recurrent neural network for time series prediction. In: Proc. 26th International Joint Conference on Artificial Intelligence (IJCAI). AAAI Press, pp. 2627–2633.

[27] Rush, A. M., Chopra, S., Weston, J., 2015. A neural attention model for abstractive sentence summarization. In: Proc.2015 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 379–389.

[28] Scardapane, S., Nieddu, E., Firmani, D., Merialdo, P., 2019. Multikernel activation functions: formulation and a case study. arXiv preprint arXiv:1901.10232.

[29] Scardapane, S., Van Vaerenbergh, S., Comminiello, D., Totaro, S., Uncini, A., 2018. Recurrent neural networks with flexible gates using kernel activation functions. In: Proc. 2018 IEEE 28th International

Journal Pre-proof

Journal Pre-proof

Workshop on Machine Learning for Signal Processing (MLSP). IEEE, pp. 1–6.

[30] Scardapane, S., Van Vaerenbergh, S., Totaro, S., Uncini, A., 2019. Kafnets: kernel-based non-parametric activation functions for neural networks. Neural Networks 110, 19–32.

[31] Sutskever, I., Vinyals, O., Le, Q. V., 2014. Sequence to sequence learning with neural networks. In: Advances in neural information processing systems. pp. 3104–3112.

[32] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. In: Advances in Neural Information Processing Systems. pp. 5998–6008.

[33] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y., 2017. Graph attention networks. arXiv preprint arXiv:1710.10903.

[34] Vinyals, O., Fortunato, M., Jaitly, N., 2015. Pointer networks. In: Advances in Neural Information Processing Systems. pp. 2692–2700.

[35] Weigend, A. S., 2018. Time series prediction: forecasting the future and understanding the past. Routledge.

[36] Ye, Z., Lyu, F., Li, L., Sun, Y., Fu, Q., Hu, F., 2019. Unsupervised object transfiguration with attention. Cognitive Computation, 1–10.

[37] Zamora-Martinez, F., Romeu, P., Botella-Rocamora, P., Pardo, J., 2014. On-line learning of indoor temperature forecasting models towards energy efficiency. Energy and Buildings 83, 162–172.

Journal Pre-proof

Journal Pre-proof

[38] Zhang, G. P., 2003. Time series forecasting using a hybrid arima and neural network model. Neurocomputing 50, 159–175.

[39] Zhang, H., Goodfellow, I., Metaxas, D., Odena, A., 2018. Self-attention generative adversarial networks. arXiv preprint arXiv:1805.08318.

[40] Zhang, H.-G., Wu, L., Song, Y., Su, C.-W., Wang, Q., Su, F., 2018. An online sequential learning non-parametric value-at-risk model for high-dimensional time series. Cognitive Computation 10 (2), 187–200.

[41] Zhang, L., Liu, Z., Zhang, S., Yang, X., Qiao, H., Huang, K., Hussain, A., 2019. Cross-modality interactive attention network for multispectral pedestrian detection. Information Fusion 50, 20–29.

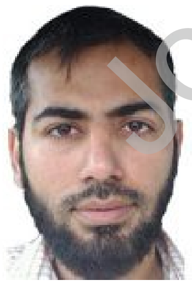
[42] Zhang, X., Trmal, J., Povey, D., Khudanpur, S., 2014. Improving deep neural network acoustic models using generalized maxout networks. In: Proc. 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp. 215–219.

Journal Pre-proof

Journal Pre-proof

Simone Totaro is a Master student in statistics at Sapienza University of Rome, and a Research Assistant at UPF working on representation leaning and reinforcement learning. In the past, he has been a data analyst intern at Unicredit and involved in multiple startup projects.

Amir Hussain obtained his B.Eng. (with the highest 1st Class Honors) and Ph.D. (in novel neural network architectures and algorithms) from the University of Strathclyde in Glasgow, Scotland, UK, in 1992 and 1997 respectively. Following postdocoral and academic positions at the University of West of Scotland (1996-98), University of Dundee (1998-2000), and University of Stirling (2000-2018) respectively, he joined Edinburgh Napier University, in Scotland, UK, in 2018, as Professor of Computing Science, and founding Director of the Cognitive Big Data and Cybersecurity (CogBiD) Research Laboratory. His research interests are cross-disciplinary and industry focused, and include secure and context-aware 5G-IoT driven AI, and multi-modal cognitive and sentic computing techniques and applications. He has published more than 400 papers, including over a dozen books and around 150 journal papers. He has led major national, European and international projects and supervised more than 30 PhD students, He is founding Editor-in-Chief of two leading journals: Cognitive Computation (Springer Nature), and BMC Big Data Analytics (BioMed Central); and Chief-Editor of the Springer Book Series on Socio-Affective Computing, and Cognitive Computation Trends. He has been appointed invited Associate Editor of several prestigious journals, including the IEEE Transactions on Neural Networks and Learning Systems, the IEEE Transactions on Emerging Topics in Computational Intelligence, and (Elsevier) Information Fusion. He is Vice-Chair of the Emergent Technologies Technical Committee of the IEEE Computational Intelligence Society (CIS), and Chapter Chair of the IEEE UK and RI Industry Applications Society.



Simone Scardapane received his B.Sc. in Computer Engineering at Roma Tre university in 2009, and a M.Sc. in Artificial Intelligence and Robotics in "Sapienza" University two years later. After working one year as a software/web developer, he obtained a Ph.D. in the same university in 2016, researching mainly in the fields of distributed machine learning and adaptive audio processing. Currently, he is an assistant professor at "Sapienza" University (UK).

Journal Pre-proof

Journal Pre-proof

Journal Pre-proof

Journal Pre-proof

Declaration of interests

[✓ ☐The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.]

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Rome, 7/10/2019

[illegible]

Journal Pre-proof