Predictive Medicine Using Interpretable Recurrent Neural Networks

André Cristóvão Neves Ferreira
andre.c.n.ferreira@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2020

Abstract

Deep learning has been revolutionizing multiple aspects of our daily lives, thanks to its state-of-the-art results. However, the complexity of its models and its associated difficulty to interpret its results has prevented it from being widely adopted in healthcare systems. This represents a missed opportunity, specially considering the growing volumes of Electronic Health Record (EHR) data, as hospitals and clinics increasingly collect information in digital databases. While there are studies addressing artificial neural networks applied to this type of data, the interpretability component tends to be approached lightly or even disregarded. Here we demonstrate the superior capability of recurrent neural network based models, outperforming multiple baselines with an average of 0.94 test AUC, when predicting the use of non-invasive ventilation by Amyotrophic Lateral Sclerosis (ALS) patients, while also presenting a comprehensive explainability solution. In order to interpret these complex, recurrent algorithms, the robust SHAP package was adapted, as well as a new instance importance score was defined, to highlight the effect of feature values and time series samples in the output, respectively. These concepts were then combined in a dashboard, which serves as a proof of concept in terms of a AI-enhanced detailed analysis tool for medical staff.
Keywords: Deep learning, interpretability, recurrent neural network, electronic health records, disease progression, data visualization

1. Introduction

Through deep learning models, academia and industry alike have disrupted a wide variety of areas. However, compared to previous machine learning models, these high-performing yet more complex deep learning models are less intuitive, in terms of interpreting their outputs. This observation started a performance and interpretability tradeoff, as while in some cases one might desire accuracy above all, in other, more critical scenarios, it is also very important to validate and understand how the model gets to each result.

One case where interpretability matters particulary is in healthcare. When a decision can define recovery or deteorating health, life or death, any error can result in serious consequences. So, each decision must be carefully thought of, neatly planned out and made with thorough understanding of the situation. Medics cannot afford to just blindly trust an algorithm, no matter how good it claims to be. This difficult interpretability of deep learning models is likely why AI has not yet been massively integrated into healthcare systems worldwide. Traditional machine learning models, with relatively low accuracy, do not represent enough value, while deep learning cannot be trusted.

Towards reaching an AI approach that can be considered for integration in a healthcare system, several steps need to be taken. This work hopes to push in that direction, by contributing in key areas and showcasing the potential of it. To do so, the main objectives have been defined as:

- Training machine learning models that excel in the prediction of disease progression, based on Electronic Health Records (EHR) data.
- Find or develop an adequate interpretability technique to allow for model validation and output explainability.
- Create a prototype of a platform that can allow for intuitive interaction with the trained models, gathering insights from it and interpreting its outcomes.

2. Related Work

Over the last few years, there has been an increasing number of papers published on deep learning applied to EHR data [4, 3, 7, 13, 16, 1, 14]. These

tend to focus mostly on these models' main advantage, the gain in performance. After an initial study [4] proved the abilities of Recurrent Neural Networks (RNN) in medical time series contexts, a slew of other papers started following the same steps, with some variations to try to achieve state-of-the-art. However, despite the critical case of healthcare, these studies do not usually give due priority to interpretability of the models. For instance, in the DeepCare paper [13], they present an interesting custom model, that is based on Long Short Term Memory (LSTM) with incorporation of elapsed time between samples, as well as embedding layers and an intriguing targeting of intervention data. Yet they do not offer any explanation of the model's outputs. Later, in 2018, a team at Google AI developed a model ensemble, composed of RNNs, Time Aware Neural Networks and boosted decision trees, which achieved state-of-the-art results on the predictions of mortality, readmission, long stay and discharge diagnosis. While they did not ignore the interpretability factor, the authors only interpreted part of the model with its attention mechanism, which only allows to see what was highlighted by the model, not more specifically how each data point impacted the outcomes. A paper that put a more serious focus on output explanations was the RETAIN [3] paper that managed to make a model where feature contribution scores, i.e. how each feature impacts the output, could be calculated analytically. Furthermore, in its follow up paper RetainVis [7], the authors presented a dashboard that showcased how medical staff could interact with these analysis, see relevant instances in the patients' time series, and gather insights from the model. However, both of these papers' interpretability approach relies on a very specific and complex model architecture, which would not work for other, potentially better performing models.

3. Background

3.1. Recurrent Neural Networks (RNN)

Recurrent Neural Networks [11] are the backbone of sequential or temporal classification and prediction problems and, as seen in the literature review, EHR problems are no exception. The way it works is that, by receiving a sequential input, each vector in the sequence is fed into a block of this neural network, which is usually referred to as a cell. Then, this cell computes a vector h_t, based on the current input x_t and the previous cell's h_{t-1}, and sends it to the next cell, the one that gets the next vector in the input sequence. These h values are called hidden memories, as they accumulate information from previous inputs. From this hidden memories, an output can be calculated, using for instance a Softmax activation function to obtain a classification probability, as seen in the following equation:

$$\begin{cases} h_t = \sigma(Ux_t + Wh_{t-1}) \\ y_t = Softmax(Vh_t) \end{cases} \tag{1}$$

It is also worth noting that multiple layers of RNN can be stacked on top of each other, with the possibility of them having opposite directions, in terms of the recurrent connection that traverses the input sequence, which creates a bidirectional RNN.

When applying backpropagation in the vanilla RNN, there is a problem that stops it from working correctly in long sequences. As the error backpropagates to previous cells through the recurrent connection, it is repeatedly multiplied by the same weights. This causes two well known problems called vanishing gradient and exploding gradient. If $||W|| < 1$, the gradients of weights tends to 0, stopping it from updating and contributing to increasing the model's performance. On the other hand, if $||W|| > 1$, the gradients of weights tends to ∞, which renders the model unfeasible.

3.2. Long Short-Term Memory (LSTM)

In order to fix the vanilla RNN's problem with vanishing and exploding gradients, research has been made on possible modifications of this recurrent architecture. One of the most famous variations is the Long Short-Term Memory[5]. It solves the gradient issues by creating what is sometimes referred to as a "gradient highway", a recurrent connection that avoids repeated multiplications by the same weight matrix over and over again. Instead, in the LSTM, the hidden memories are calculated through pointwise multiplications and additions, which change along the sequence according to the inputs, as seen on Figure 1 and in the following equations:

$$\begin{cases} i_t = \sigma(W_i h_{t-1} + U_i x_t) \\ f_t = \sigma(W_f h_{t-1} + U_f x_t) \\ o_t = \sigma(W_o h_{t-1} + U_o x_t) \\ \widetilde{C}_t = tanh(W_g h_{t-1} + U_g x_t) \\ C_t = \sigma(f_g \odot C_{t-1} + i_t \odot \widetilde{C}_t) \\ h_t = tanh(C_t) \odot o_t \end{cases} \tag{2}$$

3.3. LSTM with Varying Timestamps

We now know about RNN and LSTM models, which have the ability to hold memory of previous instances, with longer memory in LSTM models, granting them an advantage over other models when handling sequential data. However, these models do not have any built-in procedure to tackle varying time differences between samples. Often is the case that patients get examined in irregular time intervals, with the possibility of them being in very different scales. In this case, time variation between samples can be particularly important, as a

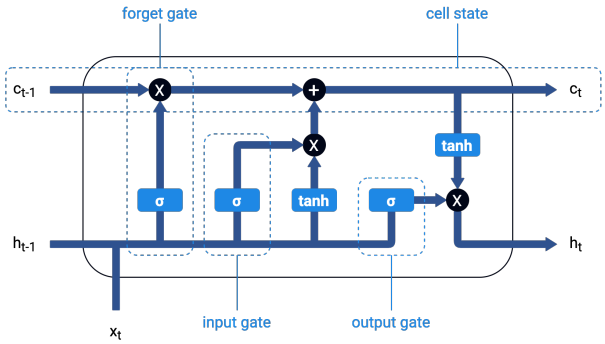

Figure 1: Overall structure of a LSTM cell. Blue rectangles represent activation functions, rounded dark circles represent pointwise operations.

sample very far away from the last one should take past information less into account, several frequent samples can indicate a more severe status or the presence in a certain type of medical unit, among others. So, we need to find a way to include information of time variation between samples, from here on out also refered to as Δt, in these recurrent models. To the models that include this information somehow, we can classify as "Time Aware Models", as Figure 2 shows.

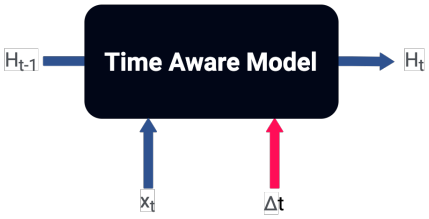

Figure 2: Overview of a time aware model. H can represent either a single hidden memory variable h or with the addition of the cell memory c. The inclusion of time variation information can either be through using it directly as a feature or through the model's intrinsic architecture.

A straightforward solution to this problem is to simply add a Δt feature to the model. This technique requires minimal intervention in the models' overall structure, with just the addition of more weights regarding the Δt feature. Hopefully, in a well optimized model and with enough data, it will make the model learn how to handle the time differences in the current prediction or regression task.

Meanwhile, there has been research [13, 2] to modify the LSTM architecture, aiming to have a better integration of time variation between the samples of a sequence. One of these modified versions is referred to in the paper of Inci M. Baytas et. al [2] as MF1-LSTM, which stands for Modified Forget Gate LSTM. Indeed, the main change brought into the typical LSTM format is in the forget gate. After the usual calculations, it multiplies the forget gate's output by $g(\Delta t)$ such as $f_t = g(\Delta t) * f_t$. This $g(\Delta t)$ corresponds to a function that transforms the time variation value into a better scaling factor. In this paper, we use $g(\Delta t) = \max\left(1, \frac{1}{\Delta t}\right)$, with normalized Δt. The intention here was as to avoid attributing too much importance and possibly even saturation-prone high values to samples that are very close in time.

With a similar logic, there is the MF2-LSTM model type. Instead of scaling the forget gate's output, it directly alters its formula by introducing parametric time weight such as $f = \sigma\left(W_f x_t + U_f h_{t-1} + Q_f q_{\Delta t} + b_f\right)$. Similarly to the original paper, we use $q_{\Delta t} = \left(\frac{\Delta t}{15}, \left(\frac{\Delta t}{90}\right)^2, \left(\frac{\Delta t}{180}\right)^3\right)$, with unnormalized Δt, with a notion of short, medium and long time differences.

Then there is T-LSTM, a model proposed in [2]. It preserves the original LSTM structure, but adds more steps when defining the previous cell state c_t. So now, it replaces LSTM's direct connection of c_{t-1} with c^*_{t-1}, according to the following equations:

$$\begin{cases} c^S_{t-1} = \tanh\left(W_d c_{t-1} + b_d\right) \\ \hat{c}^S_{t-1} = c^S_{t-1} * g(\Delta t) \\ c^T_{t-1} = c_{t-1} - c^S_{t-1} \\ c^*_{t-1} = c^T_{t-1} + \hat{c}^S_{t-1} \end{cases} \tag{3}$$

3.4. SHAP Values

In 2017, Scott Lundberg and Su-In Lee published the paper "A Unified Approach to Interpreting Model Predictions" [10]. As the title suggests, they proposed a new method to interpret machine learning models that unifies previous ones. They found out that many interpretability methods follow the same core logic: learn a simpler explanation model from the original one, through a local linear model. Because of this, the authors call them additive feature attribution methods.

Essentially, for each sample x that we want to interpret, using model f's output, we train a linear model g, which locally approximates f on sample x. However, the linear model g doesn't directly use x as input data. Rather, it converts it to x', which represents which features are activated (for instance, $x'_i = 1$ means that we're using feature i, while $x'_i = 0$ means that we're "removing" feature i). As such, and considering that we have M features and $M + 1$ model coefficients (named ϕ), we get the following equation for the interpreter model:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i. \tag{4}$$

And, having the mapping function h_x that transforms x' into x, the interpreter model should locally

approximate model f by obeying to the following rule, whenever we get close to x' (i.e. $z' \approx x'$):

$$g(z') \approx f(h_x(z')), \text{ if } z' \approx x'. \quad (5)$$

Knowing that the sample x that we want to interpret naturally has all features available (in other words, x' is a vector of all ones), this local approximation dictates that the sum of all ϕ should equal the model's output for sample x:

$$\sum_{i=0}^{M} \phi_i = f(x). \quad (6)$$

Each coefficient ϕ, being this a linear model, relates to each feature's importance on the model. For instance, the bigger the absolute value of ϕ_i is, the bigger the importance of feature i is on the model. Naturally, the sign of ϕ is also relevant, as a positive ϕ corresponds to a positive impact on the model's output (the output value increases) and the opposite occurs for a negative ϕ. An exception here is ϕ_0. There is no feature 0, so it is not associated with any feature in particular. In fact, if we have an all zeroes vector z' as an input, the output of the interpreter model will be $g(0) = \phi_0$. In theory, it should correspond to the output of the model when no feature is present. Practically, what is done in SHAP is that ϕ_0 assumes the average model output on all the data, so that it represents a form of starting point for the model before adding the impact of each feature. Because of this, we can see each of the remaining coefficients as each feature's push on the base output value (ϕ_0) onto a bigger or smaller output (depending on the coefficient's sign), with the sum of all of these feature related coefficients resulting in the difference between the output on the current sample and the average output value.

In summary, SHAP has similarities with Shapley values [15], mantaining its foundational properties and even fulfilling three additional ones that were not contempleted before. The main advantage of SHAP over traditional Shapley values is its largely improved efficiency, due to its sampling and regularization approaches.

SHAP has multiple explainers, i.e. algorithms to estimate SHAP values. The single most important SHAP explainer for this work, as it is currently the only one suited for interpreting recurrent neural networks, is the Kernel Explainer. It relies in a linear model that locally approximates the original model as an interpreter, and uses a simplified input x', where values of 1 correspond to the feature's original value being used and values of 0 represent the feature being missing. Furthermore, it does not assume any specific model component or characteristics, which makes it completely model-agnostic. The linear model's loss function has a specific weighting $\pi_x(z')$ that is applied on the training samples, which fulfills Shapley properties and gives more value to combinations of small and large numbers of active feature values.

Another particularly relevant estimator, considering its usability in one of this work's baseline models, is the Tree Explainer [9, 8]. Following the same basic structure of SHAP values, it optimizes part of the calculation for tree-based models such as decision trees, random forests and gradient boosted trees. Without going into more details, considering its optimizations, the Tree Explainer can do a complete calculation of SHAP values in a considerably faster way than the model-agnostic Kernel Explainer.

4. Methodology

4.1. Data

The main dataset which this paper addresses is the Portuguese ALS dataset, a collection of data from Amyotrophic Lateral Sclerosis (ALS) patients, collected between 1995 and 2018 in the Translational Clinic Physiology Unit, Hospital de Santa Maria, IMM, Lisbon. The dataset contained 1110 patients and a variety of feature types, both static and temporal, categorical and numerical, and from multiple topics, ranging from demographics and family history to genetic and respiratory data. A particularly relevant column is the date of Non-Invasive Ventilation (NIV), as it is the source from which we extract the label. The goal is to predict the use of NIV anytime over the next 90 days.

4.2. Data Utils Package

Along the work done for this paper, a toolbox was developed, which encompassed and standardized the core data science pipelines that were needed: the Data Utils package. It is divided on multiple modules, each one addressing a different type of tasks. These can be seen as separate core parts of a data science pipeline, ranging from the usual data preprocessing to training neural networks. While it mostly relies on well known best practises, such as normalization, model versioning and hyperparameter tuning, there are also some less common intuitions builtin. One of them is the embedding in a multivariate time series context.

Deep learning models cannot handle categorical features directly. So, we need to find a way to convert them into a numerical representation, be it binary, integer or float. A straightforward solution is to apply one hot encoding, where each category is converted into its own binary feature, which indicates if that category is present in the sample or not. However, using embedding layers instead can lead to better results. As such, a solution was developed inside Data Utils. The method relies on a pre-existing PyTorch function, which is the embedding bag. It is essentially an embedding layer but with

an averaging operation on top, in case we have multiple categories to embed. It is however optimized for unidimensional sequences. So, we first one hot encode the categorical features then, for those that we want to embed, the embedding pipeline multiplies each one hot encoded column with its index, counting only the columns that originated from the same categorical feature. This way, we can feed sequences of keys to the embedding bag, which it can encode and return the average of embeddings, row by row. Inside Data Utils, the code then handles all the intermediate steps required to integrate these lists of embeddings into the data and remove the former one hot encoded columns.

4.3. Interpretability

The Kernel Explainer, a SHAP values estimator addressed in Section 3.4, is referred to as being model-agnostic, being able to interpret any machine learning model. However, this very advantage of being agnostic, with virtually no assumptions regarding the model that it will explain, can be a disadvantage. Some specific machine learning models, including RNNs, do need a special treatment, which is not builtin. When applied to RNNs, the resulting sum of the SHAP coefficients does not match with the model output, which breaks Eq.(6). This is due to the fact that the Kernel Explainer always tries to explain each sample individually, separate from all the others. This would still be fine if we used a simpler model, one that only uses a single instance to predict the label. However, as we are using a RNN, which accumulates memory from previous instances in its hidden state, it will consider the samples as being separate sequences of one single instance, eliminating the use of the model's memory. In order to fix this issue, SHAP's code had to be changed, so that it includes the option to interpret recurrent models on multivariate sequences. There were more subtle changes needed, but the core changes can be summarized by changing KernelSHAP's iteration through data so as to preserve the model's hidden memory throughout the sequences. By going sequence by sequence, instead of sample by sample, we can mantain the flow of the RNN-type model's hidden state. We can now recover the local accuracy property (Eq.(6)), where the sum of SHAP values equals the machine learning model's output.

The typical KernelSHAP approach can be unpractical because of its slowness. Knowing this, every chance we have to make it faster should be considered. Naturally, a computationally heavy part of the process is the iteration through multiple combinations of samples from the background data, when training the interpreter model. So, if we could reduce the number of samples used, we would be able to get a speedup. Now the question is if we could represent the missing features by just a single reference value. And a hint for the answer lies on of SHAP's formulas, of the marginal expectation $f_x(h_x(z')) \approx f([z_S, E[z_{\bar{S}}]])$. If we are integrating over samples to get the expected values of the missing features, it seems reasonable to directly use the average values of those features as a reference value. And this is even easier to do if, in the preprocessing phase, we normalized the data into z-scores:

$$z = \frac{x - \mu}{\sigma}. \tag{7}$$

This way, we just need to use an all zeroes vector as the sole background sample, as zero represents each feature's average value.

Similarly to the approach discussed in Section 4.2, the interpretability pipelines were wrapped in methods inside a package, called Model Interpreter, so as to facilitate and standardize its use. However, it does not only include the custom SHAP approach mentioned before, but also adds a new process. In a multivariate time series context, SHAP values only lead to feature importance, without interpreting the relevance of instances, i.e. individual samples in a time series, as a whole. An instance importance score was missing, so we needed to define one. An initial approach comes rather naturally: just remove the instance of which we want to get an importance score and see how it affects the final output. To do this, we subtract the original output by the output of the sequence without the respective instance, so that we get a score whose sign indicates in which direction that instance pushes the output (if the new output has a lower value, that means the instance has a positive effect; vice-versa for the higher value). We could call this an occlusion score, as we are blocking an instance in the sequence. So, considering N as the index of the last instance of the sequence, i as the index of the instance that we are analysing and S as the set of instances in the sequence, we get the following formula:

$$occlusion\_score = output_S^N - output_{S\setminus i}^N. \tag{8}$$

While it makes sense, it is likely not enough. When keeping track of something along time, such as a patient's disease progression, there tend to be certain moments where something new happens that can have repercussions or be repeated in the following events. For instance, if we were predicting the probability of worsening symptoms, we could have a patient that starts very ill but, after successful treatment, gets completely cured, with a low probability of getting sick again. If we were to only apply the previous method of instance occlusion, all instances of the patient after the treatment could receive similar scores, although it is clear that the

moment that he received the treatment is, in fact, the crucial one. In order to address this, we can take into account the variation in the output brought in by the instance. That is, we compare the output at the instance being analyzed (i) with the one immediately before it ($i-1$), like if we were calculating a derivative:

$$outvar\_score = output_S^i - output_S^{i-1}. \quad (9)$$

Of course, the occlusion score might still be relevant in many scenarios, so the ideal solution is to combine both scores in a weighted sum. Considering the more straightforward approach of occlusion, and some empirical analysis, the weights were picked to be 0.7 for occlusion and 0.3 for output variation. And since these changes in the output can be somewhat small, usually not exceeding a change of 25 percentage points in the output, we should also apply a nonlinear function on the result, so as to amplify high scores. This is particularly relevant as, considering that the plot to visualize these scores uses colors, we must reinforce the higher impact samples with more distinguishable colors. For that, I have chosen the *tanh* function, as it keeps everything in the range of -1 to 1, and added a multiplier of 4 inside, so that a change of 25 percentage points in the output gets very close to the maximum score of 1. In summary, we can define the instance importance score as:

$$inst\_score = tanh(4 \times [w \times occlusion\_score + (1-w) \times outvar\_score]). \quad (10)$$

4.4. Dashboard

A dashboard was developed, composed of four major components: performance; dataset overview; feature importance; detailed analysis. As Figure 3 shows, its homepage begins with two dropdown boxes, where the user is prompted to select a dataset and a model to analyse. After both are selected, each component's preview card loads a subset of information, related to that part's intent. At first glance, we can already see in the homepage the current model's accuracy, AUC and F1 metrics on the test set of the chosen dataset, some demographics and label balance information of the dataset, the model's three most influential features and an overview of four patients' time series and the importance of each of their samples. We can then click on any preview card's "expand" button to go into its associated page or use the dashboard's banner to do so.

Most of the pages serve somewhat self-contained information. In "Performance" we can compare models by their architecture and metrics; "Dataset overview" dives into the chosen dataset, showing information such as data size and demographics; "Feature importance" displays aggregate SHAP values, which paints a picture of how important each feature usually is for the model's output. Then, the "Detailed analysis" page connects all the ideas into a deeper visualization experience. It contains several distinct yet interconnected cards, so let us go one by one. Looking at Figure 4, in the first, biggest card, we can see an instance importance plot. It is inspired by the RetainVis paper [7], which has its own dashboard with a "patients list" segment, in which we can see each patient's time series and their clinical visits, colored according to their impact on the final output (red indicates an positive impact in the output and blue indicates a negative impact). The main difference here is that while in the "RetainVis" paper the instance importance scores are calculated based on sums of attention values, which are incorporated on their specific model architecture, in here we use the formula that was described in Section 4.3, which can be applied to any sequential machine learning model. Through this graph, we can see all patients disease progression, identify interesting samples and select one for further examination. As one sample is hovered or click on in the instance importance plot, all of the remaining page updates according to it. On the right, we can see an indication of whether or not that sample's patient used NIV in the end of his or her time series, as well as the model's final output predicting exactly this NIV usage. On the left side of the page, we have a card that lists the patient's most salient feature values, according to their SHAP values. This can give a fast reveal of the patient's main characteristics and symptoms along its medical history. Meanwhile, in the middle of the page, we have a feature importance visualization, which indicates how each feature value in the selected sample pushed the output from the average, expected value to the sample's output value. Once again, a red color means that the feature value

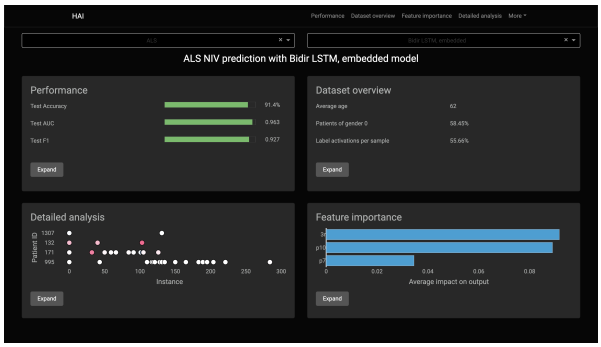

Figure 3: The dashboard homepage, when selecting "ALS" as the dataset and "LSTM, embedded" as the model.

was responsible for an increase in the output, while blue refers to a decrease. The size of the horizontal bar is equivalent to the magnitude of that output change. On the bottom right corner of the page, we have also a "Edit sample" button. By clicking on it, a new card emerges from bellow, with a table that shows the selected sample's values. With this table, we can edit the sample as much as we want and, when we click on "Stop editing", the data and the whole page is updated.

Figure 4: HAI dashboard's detailed analysis page, where every patient's time series can be inspected. In this screenshot, the option to edit the selected sample is activated.

4.5. Reproducibility

The models were trained in Google Cloud's AI Platform, using n1-highmem-4 instances, with 4 vCPUs, 26 GB RAM and 1 NVIDIA Tesla T4 GPU.

In the list bellow you can find the links to the repositories and packages that were developed during the study:

- Preprocessing and training on the Portuguese ALS dataset: `https://github.com/AndreCNF/FCUL_ALS_Disease_Progression`
- Preprocessing and training on the eICU dataset: `https://github.com/AndreCNF/eICU-mortality-prediction`
- Data Utils package: `https://github.com/AndreCNF/data-utils`
- Model Interpreter package: `https://github.com/AndreCNF/model-interpreter`
- Custom SHAP package: `https://github.com/AndreCNF/shap`
- HAI dashboard: `https://github.com/AndreCNF/hai-dash`

5. Results

5.1. Model Performance

Before diving into the results, it is important to notice that all models were trained using common hyperparameters. This setting, shown in Table 1, was determined according to the best performing RNN model from an hyperparameter tuning procedure. From the same model, which had an embedding layer that learned its weights alongside it, the embedding layer was attached to all the models that relied on embeddings, with this layer frozen to avoid further changes to it. By fixing the main hyperparameters like this, we guarantee a fairer comparison between the models, without interference from different parameters, which we do not intend to analyse in detail. Additionally, all the deep learning based models have a fully connected layer which transforms a sample's hidden state, from its recurrent cell, into the output score.

Hyperparameter	Value
n_hidden	653
n_layers	2
p_dropout	0.4250806721766345
embedding_dim[1]	7

Table 1: Common model hyperparameters.

Another relevant detail to consider is that, as the models' performance is dependant on their initial random state, each model was trained three times, on different random seeds, with the results presented on Table 2, showing the average and standard deviation across these experiments. With this approach, we can reduce the "luck" element out of the randomness of model initialization, getting a fairer comparison between the models based on their core characteristics, instead of on stochastic processes, which are less relevant for the study.

Test AUC was the chosen metric to highlight here, as it can give us a sense of how well the model detects positive samples without labelling too many false positives, doing so across several classification thresholds. Meanwhile, other metrics such as accuracy, loss, recall, precision and F1 score can be viewed in the ALS and dashboard repositories linked in Section 4.5, including also in the training and validation sets.

The first observation that we might get from the results of Table 2, where the models' average test AUC are shown in a descending order, is how not only are the intrinsically time-aware models MF1-LSTM, MF2-LSTM and T-LSTM not in the top of the table, but they are in fact at the bottom, with some of the worse performance. In the case of

[1] Only used when the model has an embedding layer.

MF2-LSTM and T-LSTM, one of the reasons might be because of added parameters, which for a small dataset such as the Portuguese ALS one, it might be too much to allow for proper learning. But still, it does not seem to be the whole reason why the metrics are so low, as they do not add a very significant amount of parameters, compared for instance to the conversion of a recurrent model to bidirectional. Unfortunately, as I had to develop these models almost from scratch in PyTorch, the issues may originate from an implementation flaw, which I was not able to detect.

Another criticism that we can do is that the baselines have similar performance to the artificial neural networks. Although they do not reach the top of the ranking, logistic regression and specially XGBoost have results comparable to unidirectional recurrent models, which can be surprising as they only analyse sample by sample, without handling the sequential nature of the data. This can be because the problem might be somewhat easy to solve, perhaps due to a couple of features that have a strong connection with the label, the NIV treatment (as we will see in the interpretation results of Section 5.2).

Model	Test AUC	
	Avg.	Std.
Bidir.[2] LSTM, Δt^3	**0.937**	0.026
Bidir.[2] LSTM, embed	0.927	0.026
Bidir.[2] LSTM	0.916	0.016
Bidir.[2] LSTM, embed, Δt^3	0.915	0.021
Bidir.[2] RNN, embed, Δt^3	0.897	0.022
XGBoost	0.833	0.036
LSTM, embed, Δt^3	0.822	0.035
RNN	0.797	0.015
LSTM	0.793	0.023
Logistic Regression	0.782	0.003
MF1-LSTM	0.675	0.028
MF2-LSTM	0.669	0.024
T-LSTM	0.649	0.023

Table 2: Model performance results, indicated through test AUC. The presented values correspond to the average and standard deviation over three different trained models for each type.

We can then see that the best performing models are all bidirectional. It is more predictable, contrary to the previous observations, not only because of other cases where bidirectionality was a key factor but also due to the clear advantage in its logic. As the hidden memory travels in both directions, forward and backwards in the sequences, it also updates past samples with information from the future, in a time series context. Although this might seem like an undesirable trick, it has upsides such as being able to extract more insights from the added direction, correcting previous outputs to give a better analysis of a sequence, and, in a more subjective way, we can view it as more similar to human behaviour, where we sometimes check a sequence of events from the most recent one to the oldest.

As we have been discussing, and as Table 2 makes it clear, each artificial neural network can have several add-ons, as if pieces which can be added or removed according to the desired behaviour. Comparing all the models' performances, between those that are equal except with the variation of having or not a given component, we can see the average impact of each component on Figure 5. This plot really emphasizes the considerable benefit of making a model be bidirectional, adding an average of more than 0.1 to the test AUC. LSTMs also show an improvement over traditional RNNs, although not on the scale that one would expect, considering the better handling of gradients as explained in Section 3.2. Besides the previously mentioned hypothesis of dominant features, the sequences are very short, averaging on just six samples per patient. This way, there is less risk of vanishing or exploding gradients and so RNNs could be good enough, without running into many memory issues. Meanwhile, the time awareness (i.e. adding time variation between samples as a feature) and embedding have an even smaller scale of effect, with the latter even having a negative contribution to the performance, on average. The time difference between instances adds more parameters to the model and, as the data has short sequences and we suspect of features that have a strong correlation with the label, it seems reasonable that it does not have a big payoff in the current use case. Meanwhile, the embedding layer could also not be of particular relevance as there is only one categorical feature and, once again, there is a subset of other features which tend to outshine all the others.

Having all the flaws and surprises in consideration, it is still noteworthy to see the impressive performance of the top models, sustained mainly by bidirectional LSTMs, which reach almost the maximum possible value of test AUC (which would be 1). There is no clear, specific winning combination, considering the standard deviation of the performances and the comparably small difference between the average metrics, but it becomes clear the advantage of recurrent neural networks, in particular those of type LSTM, and of these being bidirec-

[2]Bidirectional recurrent model.

[3]Time aware model which uses elapsed time information as a feature.

tional.

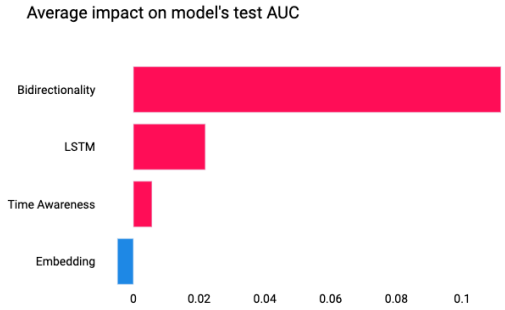

Figure 5: The average performance impact of each model component, indicated through test AUC. These only apply to comparisons between LSTM and RNN models. As in other plots, the red color indicates an increase, in this case of the test AUC metric, while the blue color corresponds to a decrease.

5.2. Model Interpretation

Observing the feature importance plots, such as that of Figures 6 and 7, it becomes clear right away that, as suspected, there is a small subset of features that have a large impact on the models' output. For the LSTM and RNN models, features $3r$ and $p10$ have a contribution that, on average, is over two times larger than the third most influential feature, with an increasing distance to the remaining ones. On the XGBoost models, even just the $3r$ feature seems to be enough to decide most predictions, with an average SHAP value that is several times larger than that of all the other features (note however that the XGBoost model has worse metrics than the exemplified LSTM model). This might be the main reason why the top performing models are so good and even most of the baselines have considerable performance, since just one or two features are usually so decisive. It is also interesting to observe that, being both $3r$ and $p10$ features related to respiratory symptoms, it makes sense that they have a high correlation with the use of NIV, the label of this machine learning task, since it constitutes respiratory support. Other features, such as the also respiratory symptoms related r and $2r$, motor capabilities related $p7$ and $p5$, and the patient's age are also top impactful features that are common to multiple models. These are not only subjectively seen as a proxy to ALS disease progression, but also can be seen directly from the data as clearly having different patterns depending on whether the patient is on NIV or not.

The similarities between the feature importance summary of different models, including with the XGBoost which uses a more optimized version of SHAP, helps to validate the robustness of the implemented feature importance approach. In particular, it supports the custom SHAP version that was developed in this thesis, which made some adaptations to make it compatible with RNN-based models.

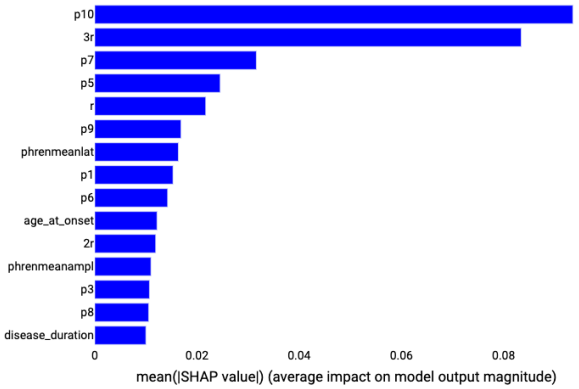

Figure 6: Feature importance of model bidirectional LSTM, time aware.

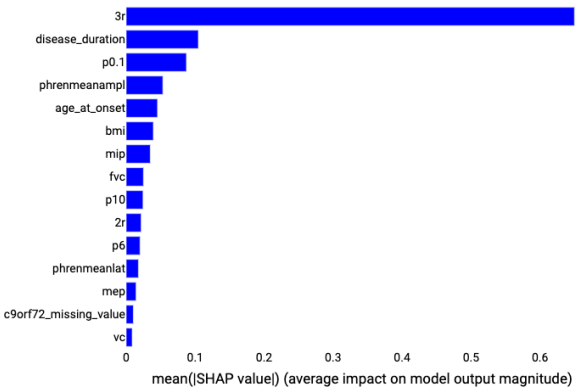

Figure 7: Feature importance of model XGBoost.

While the features that we mentioned before are, on average, the most influential, when filtering data to smaller aggregations or going into a sample by sample analysis, we can see further patterns and scenarios. Looking at the example from Figure 4, where a sample was selected based on its high instance importance score (i.e. it has an intense red color on the graph), we see that gender has one of the five most impactful feature values in that sample and the feature *el_escorial_reviewed_criteria_pma* assumes an even larger importance than $3r$. As such, these more detailed analysis of interpretability, with a combination of feature and instance importance, give us the ability to discover more insights and understand how the models adapt to different contexts.

6. Conclusions

This study's accomplishments can be summarized in three parts:

- **Performance:** Deep learning models were trained to predict NIV in ALS patients, with the top performing models surpassing the baseline by a significant margin, i.e. over 0.1 increase in the average test AUC. This performance boost was demonstrated to be mainly caused by the bidirectionality and use of LSTM cells in the artificial neural networks.

- **Interpretability:** Kernel SHAP, a robust technique for the explanation of model outputs, was adapted to RNN-based models, including the majority of its possible variations, such as bidirectionality. This represents an important step to truly make Kernel SHAP a model-agnostic method and towards the interpretation of recurrent models. There was also an introduction to a new instance importance score, which allows us to interpret how each sample influenced a sequence's final output, based on its impact on the model's outputs.

- **Usability:** A dashboard was developed, in a prototype of what a real healthcare solution could look like, integrating all the concepts that this paper addresses.

This paper was focused on a relatively small and disease specific dataset. As future work, given enough time and resources, it could be scaled up to larger, more complex scenarios. With larger datasets, we could have a better test to embedding and time variation concepts, as well as get a more interesting case to compare deep learning sequential models to traditional ones. It would also be more interesting to address such datasets that contain more treatment data. With this kind of data, model interpretation, as it was performed in this thesis, could be more useful in the sense of optimizing the patients' treatment.

It could also make sense to analyse alternative interpretability methods. There is active research [6, 12] debating SHAP's reliability, discussing in particular its unrealistic assumption of feature independence.

Above all, without dismissing the progress so far and the usefulness of current approaches, we should remain sceptic and continue to improve the interpretation of models, in a time when machine learning keeps on entering our lives and we are asked to trust the models.

References

[1] A. Avati, K. Jung, S. Harman, L. Downing, A. Ng, and N. H. Shah. Improving Palliative Care with Deep Learning. 11 2017.

[2] I. M. Baytas, C. Xiao, X. Zhang, F. Wang, A. K. Jain, and J. Zhou. Patient Subtyping via Time-Aware LSTM Networks. 10, 2017.

[3] E. Choi, M. T. Bahadori, J. A. Kulas, A. Schuetz, W. F. Stewart, and J. Sun. RETAIN: An Interpretable Predictive Model for Healthcare using Reverse Time Attention Mechanism. 8 2016.

[4] E. Choi, M. T. Bahadori, A. Schuetz, W. F. Stewart, and J. Sun. Doctor AI: Predicting Clinical Events via Recurrent Neural Networks. 11 2015.

[5] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[6] G. Hooker and L. Mentch. Please Stop Permuting Features: An Explanation and Alternatives, 2019.

[7] B. C. Kwon, M.-J. Choi, J. T. Kim, E. Choi, Y. B. Kim, S. Kwon, J. Sun, and J. Choo. RetainVis: Visual Analytics with Interpretable and Interactive Recurrent Neural Networks on Electronic Medical Records. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2018.

[8] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2(1):2522–5839, 2020.

[9] S. M. Lundberg, G. G. Erion, and S.-I. Lee. Consistent Individualized Feature Attribution for Tree Ensembles, 2018.

[10] S. M. Lundberg and S.-I. Lee. A Unified Approach to Interpreting Model Predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[11] Michael I. Jordan. Attractor Dynamics and Parallelism in a Connectionist Sequential Machine. In Joachim Diederich, editor, *Artificial Neural Networks*, pages 112–127. IEEE Press, 1990.

[12] C. Molnar, G. König, J. Herbinger, T. Freiesleben, S. Dandl, C. A. Scholbeck, G. Casalicchio, M. Grosse-Wentrup, and B. Bischl. Pitfalls to Avoid when Interpreting Machine Learning Models. 7 2020.

[13] T. Pham, T. Tran, D. Phung, and S. Venkatesh. Predicting healthcare trajectories from medical records: A deep learning approach. *Journal of Biomedical Informatics*, 69:218–229, 5 2017.

[14] A. Rajkomar, E. Oren, K. Chen, A. M. Dai, N. Hajaj, P. J. Liu, X. Liu, M. Sun, P. Sundberg, H. Yee, K. Zhang, G. E. Duggan, G. Flores, M. Hardt, J. Irvine, Q. Le, K. Litsch, J. Marcus, A. Mossin, J. Tansuwan, D. Wang, J. Wexler, J. Wilson, D. Ludwig, S. L. Volchenboum, K. Chou, M. Pearson, S. Madabushi, N. H. Shah, A. J. Butte, M. Howell, C. Cui, G. Corrado, and J. Dean. Scalable and accurate deep learning for electronic health records. 1 2018.

[15] L. S. Shapley. A Value for n-Person Games. 1952.

[16] H. Suresh, N. Hunt, A. Johnson, L. A. Celi, P. Szolovits, and M. Ghassemi. Clinical Intervention Prediction and Understanding using Deep Networks. 5 2017.