Explainable Career Path Prediction using Neural Models

Roan Schellingerhout BSc
Randstad Groep Netherlands
University of Amsterdam

ABSTRACT

Career path prediction aims to determine a potential employee's next job, based on the jobs they have had until now. While good performance on this task has been achieved in recent years, the models making career predictions often function as black boxes. By integrating components of explainable AI (xAI), this paper aims to make these predictions explainable and understandable. To study the effects of explainability on performance, three non-explainable baselines will be compared to three similar, but explainable, alternatives.

1 INTRODUCTION

With the rise of the modern gig economy, it has become more difficult for job seekers to find stable positions of employment [17]. In addition, due average education level of the workforce having increased considerably in recent years, potential employees are faced with more opportunities than ever before [9]. This has made it significantly more difficult for job seekers, and employment agencies alike, to find positions that fit their needs. To assist in this complex and challenging task, many employment agencies have started to make use of machine learning (ML) techniques to find suitable positions for individuals, and capable employees for companies.

Previous research on automated career path prediction has been done, but this research tends to share a common flaw: a lack of explainability [12, 13, 15]. While deep learning and reinforcement learning tend to deliver good performance, these models often function as a black box of sorts. Although having good results that are difficult to interpret is acceptable in many use cases, choosing a new career is such an impactful event in a person's life that it is unrealistic to expect users to blindly trust the model's decision. Even when the prediction done by the model is perfectly accurate, it will be difficult to accept without proper explanation of which variables lead to the ultimate decision. This is why explainability and interpretability are such crucial requirements for career path prediction models. Through the use of explainable artificial intelligence (xAI) [8], individuals with little knowledge of deep learning (e.g. the employees of an employment agency, or even job seekers themselves) are able to interpret to what extent, and in what ways exactly, each variable contributed to the final outcome of the model. By being able to concretely determine *why* a given position is ideal for a person, the recommendation becomes significantly more transparent, understandable, and thus more trustworthy.

This leads to the following research question: *To what degree can career path predictions done by deep learning models be made explainable?* However, explainability by itself is not the end-all be-all for career path prediction; if that was the case, decision trees would be a perfect model for the task. What is important, is the trade-off between explainability and performance. Therefore, this research will focus predominantly on the balancing act of those two factors. This will be done by means of the following sub-questions:

- **RQ1:** How well do non-deep learning models perform on career path predictions?
- **RQ2:** How well do regular (non-explainable) deep learning model perform on career path predictions?
- **RQ3:** How do different ways of making model predictions more explainable impact performance?
- **RQ4:** What degree of explainability gives the ideal balance between explainability and performance?

2 THEORETICAL FRAMEWORK

Career path predictions

The goal of career path prediction is to determine what position of employment is a logical next step given a job seeker's career [13]. Due to the enormous amount of factors that come into play for such a decision, this prediction has become incredibly difficult to do by hand. For one, a holistic approach needs to be taken to determine what type of job is fitting for a job seeker: what education have they enjoyed, what certificates have they obtained, what previous job experience do they have, where do their interests lie, etc. Additionally, these factors need to be matched to an employer that is looking for an employee with matching features. Considering the number of different career opportunities, this task can be compared to finding a needle in a haystack. As a result, employment agencies have started making use of machine learning, specifically deep learning, in order to assist in, or even automate, this process.

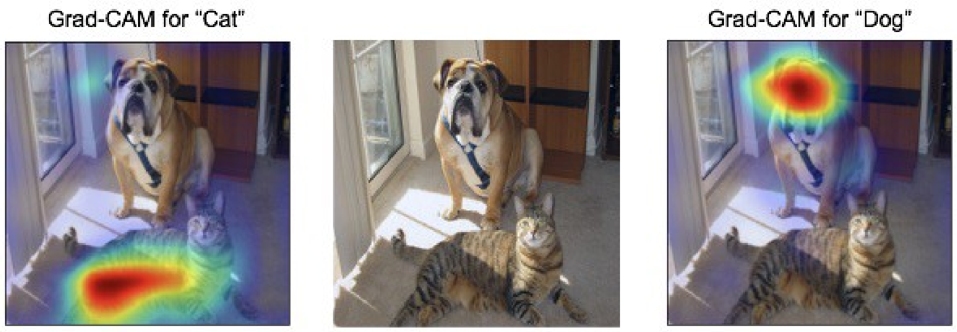


Figure 1: An example of a saliency map generated by Grad-CAM for predicting a cat and a dog. Red colours indicate higher prediction importance.[1]

Deep learning for time series predictions

An important aspect of career path predictions, however, is that they are not as straight-forward as some other classification tasks. For example, someone with just a high school diploma who is at the start of their career is unlikely to be hired for a high-paying position. However, someone with the same academic qualification, but with 30 years of high-level experience under their belt, will probably be able to find a lucrative position more easily. Thus, not just a person's general features are important, but also their past as a whole. This means that career path predictions are a form of time series predictions; past trends influence the next step [4].

Time series predictions were originally done by recurrent neural networks (RNNs); networks in which a node's output is fed back into the node itself, allowing past data to influence future data. However, while this architecture is functional, it tends to be unusable for larger time series, due to, in part, the vanishing/exploding gradient problem [11]. An improvement over classical RNNs are the more optimized long short-term memory (LSTM) networks, which consist of units composed of a cell, an input gate, an output gate, and a forget gate. The inclusion of this forget gate alters the model's derivative in such a way that the vanishing gradient problem no longer occurs [22].

Explainability in deep learning

Explainability and performance are often considered inverses of each other in the field of AI. A simple, easy to explain model is likely to perform mediocre at best, while a complex, difficult to explain model is more likely to perform well [8]. A common example of this inverse relationship can be seen in the difference between decision trees and random forests: random forests are based on decision trees, but with a higher degree of complexity, which strongly increases performance at the cost of explainability.

However, with the increasing interest in explainable AI, more and more solutions have been brought up that can make even the most complex deep learning models explainable to a degree (although often in exchange for a bit of performance) [3]. Most commonly, this explainability takes shape of visualizations of the networks' behaviour. Saliency maps and attention maps allow for the importance of different variables to be visualized, usually through some type of colour scheme indicating higher or lower importance (Figure 1). Interpreters can then determine which specific variables had a large impact on the model's final prediction, which makes it possible to better understand why the model gave its output.

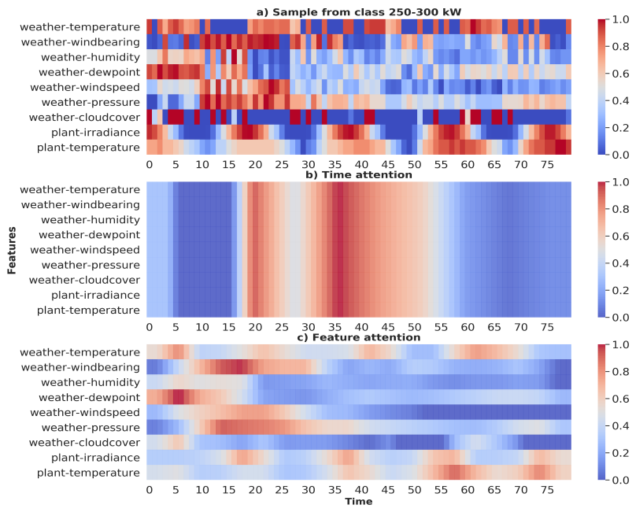


Figure 2: An example of an attention map used for time series predictions, based on Assaf and Schumann [1]. a) gives a sample of feature values, where red colors indicate higher values. b) shows the general attention given by the model to each time step. c) shows the relative importance of each feature at each time step .

[1]https://medium.com/@mohamedchetoui/grad-cam-gradient-weighted-class-activation-mapping-ffd72742243a

Explainability in time series predictions

Time series bring an additional factor into the mix: the temporal dimension. Simply visualizing which features garner the most attention thus becomes insufficient in this scenario. While a given variable might be highly important to the network initially, it could become less relevant as time progresses. Thus, in order to make explainable time series predictions, not only should there be an explanation of which variables contributed the most to the final prediction, but also at what moment their values were most decisive [20]. This makes the usage of saliency maps more difficult, as a unique saliency map would have to be used for each time step. By making use of attention mechanisms, it becomes possible to both determine to what time steps the model pays most attention, as well as to what features get the most attention at each time step. By combining these two visualizations, it becomes possible to interpret time series predictions to a high degree (Figure 2). In the case of career path predictions, this would allow job seekers to see what previous experience had impact on the recommendation for their next job.

3 DESCRIPTION OF THE DATA

The data on which the models will be trained and tested, will be provided by Randstad Groep Nederland (RGN). Due to the nature of RGN's operations, they have an exhaustive data lake consisting of temporal employee-related data (e.g. previous work experience, education, acquired skills, etc.).

Overview of the datasets

Randstad has an enormous dataset of over two million job experiences of over 500 thousand individuals (mean job count: 3.99 per person). For each of these job experiences, the dataset includes a unique ID of the candidate who had the job, the date at which the individual started their job, the date at which the individual stopped working that job (if applicable), a unique ID of the type of job (function), the level of the job (based on the international standard classification of occupations, ISCO[2]), the ISCO classification of the job type, the name of the function, the company for which the person worked, and whether the data was gathered through Randstad finding a position for the person, or through the person disclosing the data with Randstad themselves.

Additionally, Randstad also stores data on the education that their clients enjoyed, as well as their skill sets - both of which are interesting when trying to predict an individuals next job. For education, Randstad stores the level of the education (high school up to doctorate), the start and end (if applicable) date of the education, and whether or not the user has completed their education (yet).

[2]https://www.ilo.org/public/english/bureau/stat/isco/isco08/

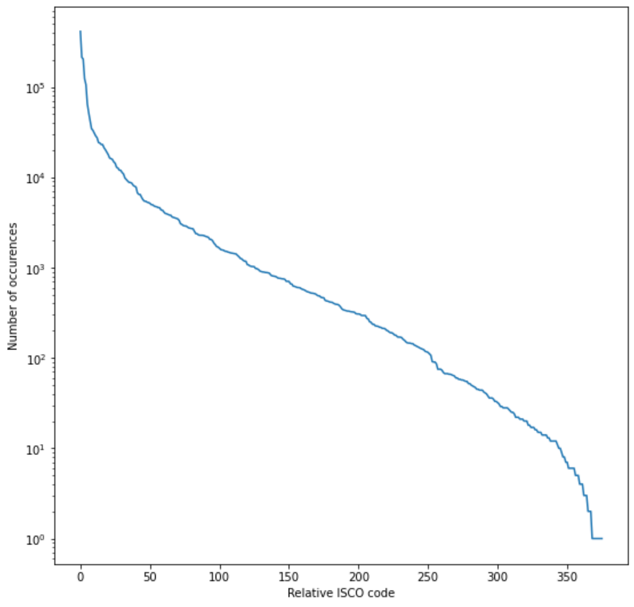


Figure 3: Distribution of ISCO codes

For the skills that the user possesses, Randstad makes use of two separate datasets: one including candidate IDs, skill IDs, and the date at which the user obtained the skill, and a second that merely maps skill IDs to actual skill names.

Data imbalance

The data in the datasets is strongly imbalanced; function IDs, ISCO codes, and education levels are all heavily skewed towards a few common values. For the function IDs, this skew is the most apparent. This makes sense, since this variable is the most granular; while both the function IDs and ISCO codes refer to job positions, ISCO codes tend to be more 'clustered', while function IDs are highly specific. This is clearly visible when looking at the number of unique values for both variables: the dataset includes a mere 376 distinct ISCO codes, but well over 3000 function IDs. Both variables, however, struggle with a similar level of imbalance, as can be seen in Figures 3 and 4.

This same imbalance is also present in the highest education obtained by candidates; the overwhelming majority has education level 0, while only a fraction of the candidates has obtained the highest level of education. This imbalance is less impactful, as the education level of candidates is merely an predictor, unlike the ISCO code and function ID, both of which could be used as the actual labels to be predicted.

To account for this strong imbalance, use will be made of class weights during training. This will prevent the classifier from always predicting the majority class, since, without

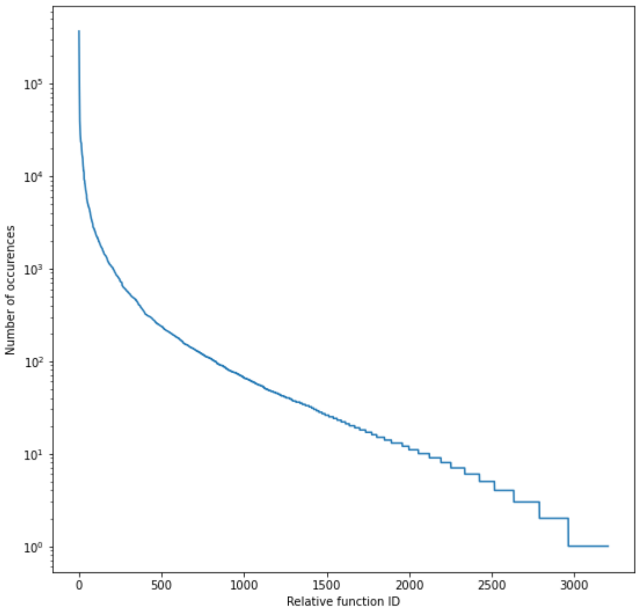


Figure 4: Distribution of function IDs

class weights, that would be an easy way to get rather high performance.

Another major imbalance takes shape in the form of sequence lengths - i.e. the number of positions candidates have had. Here, it also becomes apparent that the dataset does contain some outliers. When looking at the sequence lengths,

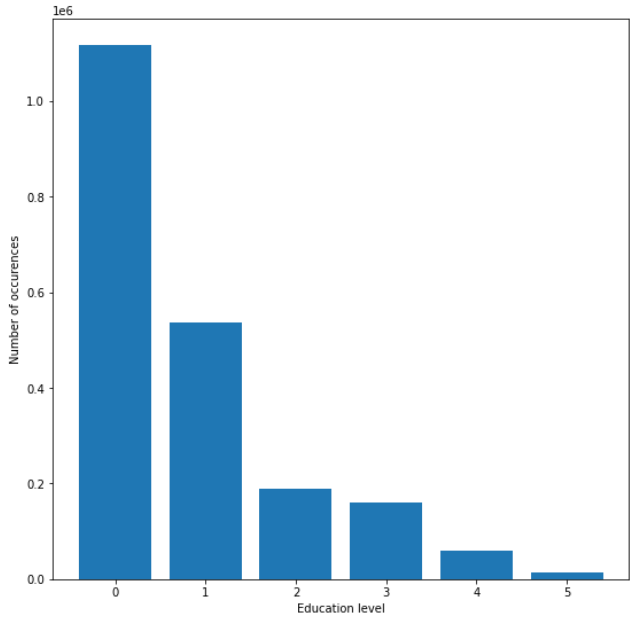


Figure 5: Distribution of highest education level obtained

it becomes clear that there are candidates in the dataset that, supposedly, have had hundreds of jobs (Figure 6). While theoretically possible, this most likely nonsensical data. As a result, only the 25 most-recent jobs of each candidate will be considered.

Undersampling

Lastly, another, less obvious imbalance is also present within the dataset. The majority of career steps tend to be within a single position, i.e., someone changing *jobs*, but not *functions*. For example, people often keep the same function, but start performing this function for a different company. Considering over 57% of all career steps consisted of the same function as in the time step before it, data points that adhered to this rule were dropped in order to prevent the models from always predicting a candidate's previous job.

4 METHODOLOGY

In order to make career path predictions, candidates' profiles were turned into time series which could be fed into different (deep-learning) models. This section outlines how candidates' careers were converted into time series, as well as how those time series were fed into different models. Lastly, an overview of the models used is given. The used models can be split into three separate categories: non-neural baselines, non-explainable neural models, and explainable neural models. The neural models were created in PyTorch and trained on a titan K80 GPU [18]. 80% of the data was used as a training set, 10% of the data was used as a validation set, on which the optimal hyperparameters were determined, and the last 10% of the data was used as a test set to evaluate model performance on unseen data.

Data prepocessing

Due to the availability of temporal data, candidates' career paths were turned into time series. For these time series, each job held by a candidate was considered to be one time step. The order of the time steps was determined by the date at which the candidate started the position. As a result, every career was turned into a time series, in which each time step was a candidate's current job, combined with the skills, certificates, languages, and education they had at achieved at the time of starting the position. To also include candidates' curriculum vitaes (CVs) at each time step, the most recent CV uploaded by a candidate at each time step was converted to numerical features using Word2Vec [16] and combined with the other features.

With over 250 thousand careers, each spanning 25 time steps, and over 500 features per time step (embedding values for skills, certificates, previous jobs, previous companies, addresses, and spoken languages, as well as 300 w2v dimensions per CV), feeding the data into deep learning models

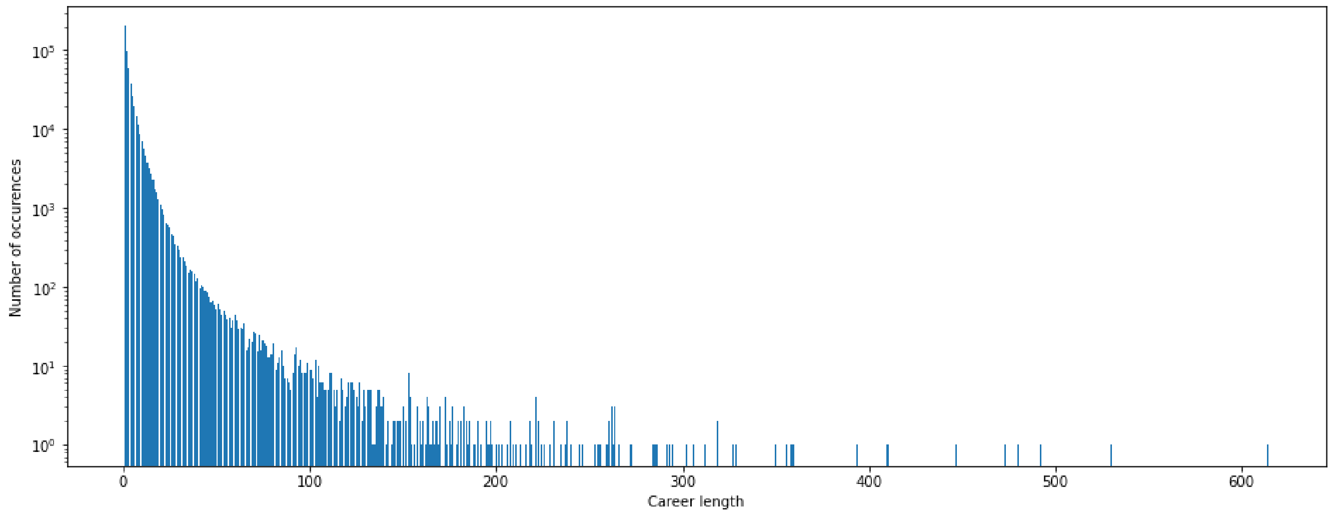


Figure 6: Distribution of job sequence lengths

as-is, turned out to be infeasible. Making use of sparse vectors to lower memory usage also was impossible, due to the incompatibility between CUDA and sparse vectors/matrices [5]. However, due to the large amount of duplicate data (i.e., a candidate's skills/certificates/CVs do not change at every time step, and can therefore often be repeated), use was made of indices in order to lower memory usage, at the cost of a slight time complexity increase. For each candidate, a location within each index was created that contained their unique attributes, and the time steps from which those attributes became the most recent ones. By then retrieving the relevant attributes for each candidate in a batch during training, the required memory usage was lowered drastically.

Baselines and Models

In order to create a comprehensive, exhaustive overview of the different ways to make career path predictions, three different types of models were implemented. For each type of model, three unique architectures were used to allow for comparison between different solutions.

RQ1 - Non-deep learning performance. Considering the fact that careers do not necessarily follow a logical trend, they can be rather difficult to model properly. For example, a person might work a job for a while not because they want to, but because they are forced to do so in order to support themselves. A person going from a position as a software engineer to a store clerk does not constitute a logical progression, but can obviously occur in the real world whenever someone gets laid off and needs to work a temporary job while they search for new alternatives. This makes career path prediction a notoriously difficult problem for deep learning models [15]. To evaluate the added value of using such models, baselines were set with non-deep learning (but coincidentally highly explainable) models as a comparison. The following models were used:

Majority class : This baseline always predicts the most common class in the dataset. Due to the strong class skew in the dataset, this will lead to performance significantly higher than a random predictor.

Majority switch : This baseline always predicts the most common switch from a given job. Due to the lack of same-function career steps in the undersampled data, this prediction is always different from the current job.

Dynamic Time Warping with k-nearest neighbors : The most sophisticated baseline will use a combination of Dynamic Time Warping (DTW) [2] and k-nearest neighbors (knn) to match candidates to the most similar candidate in the dataset. Through DTW, candidates can be matched to candidates with a similar career, even if the order or duration of each time step does not match exactly. By then finding the distance of the current candidate to all other time warped candidates, their closest neighbors can be found. Using the k-nearest neighbors algorithm on the neighbors' most recent job can then be used to make a prediction.

RQ2 - Non-explainable deep learning performance. In order to determine the impact of explainability mechanisms on model performance, it is important to compare the performance of explainable deep learning models to that of non-explainable ones. As a result, three non-explainable alternatives were implemented to allow for direct comparison between the explainable and non-explainable models. Therefore, a long short-term memory (LSTM) model, a convolutional neural network (CNN), and a CNN-LSTM, which combines the two, were implemented as follows:

LSTM : The LSTM used in this paper is based on the *hierarchical career path-aware neural network* HCPNN by Meng et al. [15]. While the original HCPNN combines candidate-specific data with company-specific data, its modular architecture allows for the removal of some of the model's components. As a result, the HCPNN was implemented using only candidate-specific features. This results in a model that takes embedded position features, feeds them into an LSTM, runs the LSTM's output through an attention layer, and combines that output with a candidate's embedded static features, after which it makes a prediction.

CNN : The CNN used in this paper was designed by He et al. [10]. It feeds the data into a 1D convolutional layer, followed by a 1D Max-Pooling layer. The output is then flattened and ran through a drop-out layer. Lastly, a fully-connected layer is used to do the final prediction.

CNN-LSTM : The CNN-LSTM used in this paper was created by Livieris et al. [14]. It uses two sequential convolutional layers, followed by a max-pooling layer as a feature extractor. The max-pooled features then get fed into an LSTM, after which a fully-connected layer does the final predictions of the model.

RQ3 - Explainable deep learning performance. In order to draw fair comparisons, the explainable models used were all as similar to their non-explainable counterparts as possible.

Explainable LSTM : The explainable LSTM used in this paper is based on the spatio-temporal attention LSTM (STA-LSTM) by Ding et al. [6]. This architecture starts off by determining spatial attention: it runs each individual time step through a linear layer. The Hadamard product between the linear layer's output and the features per time step is taken in order to determine the importance of each feature at each time step. The output hereof is then fed into an LSTM, after which the temporal attention is calculated. This is done by flattening the output of the LSTM and running it through another linear layer. This calculates a normalized importance of each time step, based on that step's hidden values. The dot product between the linear layer's output and the LSTM's hidden output is then calculated, which is fed into a last linear layer in order to make the final predictions.

Explainable CNN : The explainable CNN used in this paper is based on the explainable convolutional neural network for multivariate time series classification (XCM) by Fauvel et al. [7]. It makes use of two stages which run in parallel. The first stage uses a 2D convolutional layer with kernel size ($1 \times window\_size$) that generated $F1$ feature maps. A 1×1 2D convolutional layer is then used to summarize those $F1$ feature maps into a single feature map. The other stage, running independently, uses a 1D convolutional layer with kernel size ($N\_features \times window\_size$) and also generates $F1$ feature maps, which are summarized by a 1×1 1D convolutional layer. The two feature maps generated by the two stages are then concatenated in the feature-dimension, after which a 1D convolutional layer with kernel size ($N\_features+1 \times window\_size$) generates $F2$ feature maps. These feature maps are then ran through a 1D GlobalAveragepooling layer, which is also responsible for the predictions.

Explainable CNN-LSTM : The explainable CNN-LSTM used in this paper is based on that of Schockaert et al. [21]. This model uses a 1D convolutional layer for each time series and uses its output to augment the original time series. This combined output gets fed into an LSTM that generates one hidden state for each time step. Here, the architecture splits into two parts; one for the attention mechanism, and one for the actual prediction. The attention mechanism consists of a dense layer that takes input from the LSTM's hidden states (except the very last one), after which it takes the dot product of those. Then, an activation function is used to allow for non-linear patterns to emerge. This activation function outputs the temporal attention of the model, as well as a context vector (which includes the attention given to individual features). This context vector is then combined with the second part of the split: the last hidden state of the LSTM, and fed into another dense layer, which makes the final prediction.

RQ4 - Balancing explainability and performance. To determine what model strikes the best balance between predictive performance and explainability, each model was evaluated using metrics for both criteria. In order to evaluate performance, the following classification metrics were used: accuracy at k ($k \in \{1, 5, 10\}$), which shows how often the correct answer was within the top k predictions given by the model; micro/macro precision, which shows what fraction of positive predictions was correct for each class; micro/macro recall, which shows what part of truly positive samples were correctly marked as such for each class; and the F1-score, which is the harmonic mean of the precision and recall [19].

Addutionnaly, user-testing was done in order to evaluate the explanations generated by the models. Potential users of the models (i.e. Randstad's recruiters), were tasked to determine which variables were most relevant for a prediction made by the system. Nine recruiters were split into three groups based on their recruiting expertise (banking, catering industry, health care), and shown three separate predictions within that industry (one per model). For each predictions, they were tasked to distribute 100 'relevance points' over all of the features used by the models (previous jobs, education, skills, etc.), after which their distribution was compared to that of the system. In order to determine the sensibility of the models' explanations, the Pearson's correlation and root mean squared error (RMSE) of each models' distributions compared to the recruiters' distributions were calculated.

5 RESULTS

RQ1 - Non-deep learning performance

Other than the value of k in k-nearest neighbors, the non-deep learning baselines do not make use of any hyperparameters. As a result, the performance of these baselines is static, and cannot be improved without changing the baselines themselves. An overview of the baselines' performance can be seen in Table 1.

For DTW-KNN, optimal performance was found with $k = 15$. While both DTW-KNN and majority class predictions performed poorly, the majority switch baseline set a high bar for the deep learning models to surpass.

RQ2 - Non-explainable deep learning performance

To allow for direct comparisons, the architectures used for the non-explainable and explainable models were made as similar as possible. However, different hyperparameter configurations lead to different performance for different architectures. The results shown in Table 2 only indicate the performance given by the best hyperparameter configuration found for each model. For a full overview of hyperparameter configurations and their related performance see appendix ??.

(Extensive hyperparameter tuning has not actually been done yet, so these results are simply the best ones I found so far)

	Acc@1	Acc@5	Acc@10
DTW-KNN	0.087 ± 0.004	0.246 ± 0.006	0.273 ± 0.006
Majority class	0.105 ± 0.004	0.368 ± 0.006	0.491 ± 0.007
Majority switch	**0.191 ± 0.005**	**0.466 ± 0.006**	**0.613 ± 0.006**

Table 1: Performance of each of the non-deep learning baselines at different values of k. Different values of k indicate how often the correct answer was within the top k predictions given by the model.

	Accuracy @ 1	Accuracy @ 5	Accuracy @ 10
LSTM	0.209 ± 0.005	0.498 ± 0.006	**0.635 ± 0.006**
CNN-LSTM	**0.234 ± 0.006**	**0.505 ± 0.006**	0.634 ± 0.006
CNN	0.169 ± 0.005	0.439 ± 0.006	0.578 ± 0.006

Table 2: Performance of each of the non-explainable neural networks at different values of k. Different values of k indicate how often the correct answer was within the top k predictions given by the model.

The hyperparameters used for the results in Table 2 were the following:

LSTM : the LSTM was trained using the ADAM solver with cross-entropy loss, a batch size of 512, a learning rate of 0.001, and a single LSTM layer with hidden size 1000.

CNN-LSTM : the CNN-LSTM was also trained using the ADAM solver with cross-entropy loss, a batch size of 512, and a learning rate of 0.001. Additionally, the first 2D convolutional layer made use of a (3×3) kernel, with a (1×1) stride and padding, and generated 32 feature maps. The second 2D convolutional layer made use of the same kernel size, stride, and padding, but generated 64 feature maps. These convolutional layers were followed by a 3D max-pooling layer with a $(64 \times 2 \times 2)$ kernel and a $(1 \times 2 \times 2)$ stride. Then, use was made of a single LSTM layer with hidden size 1000.

CNN : the CNN was also trained using the ADAM solver with cross-entropy loss, a batch size of 512, and a learning rate of 0.001. Additionally, the 2D convolutional layer consisted of a (3×3) kernel, (1×1) padding and stride, and generated 48 feature maps. Following the use of a ReLU activation layer, use was made of 2D max-pooling with a (2×2) kernel and stride, and no padding. The pooling layer's output was then flattened and ran through a dropout layer with a probability of 0.4.

RQ3 - Explainable deep learning performance

The performance of the explainable counterpart of each model can be seen in Table 3.

All of the explainable models used the ADAM solver and cross-entropy loss. The other optimal hyperparameters found for the explainable models were the following:

eLSTM : The explainable LSTM made use of a learning rate of 0.001, a batch size of 512, and a single LSTM with hidden size 1000 and a dropout probability of 0.3.

	Accuracy @ 1	Accuracy @ 5	Accuracy @ 10
eLSTM	0.205 ± 0.005	0.469 ± 0.006	0.604 ± 0.006
eCNN-LSTM	**0.224 ± 0.005**	**0.511 ± 0.011**	**0.644 ± 0.009**
eCNN	0.171 ± 0.005	0.452 ± 0.006	0.569 ± 0.006

Table 3: Performance of each of the non-explainable neural networks at different values of k. Different values of k indicate how often the correct answer was within the top k predictions given by the model. 95% confidence-interval

	Corr.	RMSE
eLSTM	X.XX	X.XX
eCNN-LSTM	Y.YY	Y.YY
eCNN	Z.ZZ	Z.ZZ

Table 4: The pearson correlation and root mean squared error (RMSE) of each model compared to the scores given by the recruiters. For each time step/feature combination, both the models and the recruiters gave a score; the scores are calculated based on the difference between those scores. For correlation, higher is better, for RMSE, lower is better.

eCNN-LSTM : The explainable CNN-LSTM used a learning rate of 0.001, a batch size of 2048, a 2D convolutional layer with kernel size ($sequence\_length \times 1$) and half padding, followed by a single LSTM with hidden size 1000, using a dropout probability of 0.25.

eCNN : The first part of the explainable CNN used a 2D convolutional layer with a (15×1) kernel, a (1×1) stride and "same"-padding, that generated 8 feature maps. The following (1×1) layer did not require any hyperparameters. For the second part, the 1D convolutional layer used a ($15 \times N\_features$) kernel, with (1×1) stride, and "same"-padding, generating 8 feature maps. Another (1×1) layer was then used again. The final 1D convolutional layer used a kernel size of ($15 \times N\_features + 1$), a (1×1) stride, and "same"-padding, and generated 256 feature maps. The AveragePooling2D layer again did not require any hyperparameters.

RQ4 - Balancing explainability and performance

Each explainable model is able to generate three separate explanations for a prediction: the weight of each feature, the weight of each time step, and a time step/feature interaction map. The way in which these explanations are generated differs per model, but the final visualizations are the same, regardless of the method used to generate them (Figure 7a, 7b, and 7c).

This part has yet to be done, so the text is more of a template. In order to verify the integrity of these explanations, user research has been done with RGN's recruiters. The samples used for this user research are the same as those shown in Figures 7a, 7b, and 7c. After providing the recruiters with the predictions made by the model, they were asked to estimate which variables, at which time steps, were most important. The averaged estimates made by the recruiters can be seen in Figures **??**, **??**, and **??** (**TBD**). The results indicate that the models' explanations were strongly/slightly/barely correlated with those made by the recruiters. However, recruiters put more/less emphasis on certain features/time steps, such as **XYZ**.

To measure the sensibility of each models' explanations, two separate metrics were calculated for each of them: RMSE and pearson's correlation. This was done by calculating the difference between the average score that recruiters gave to each feature/time step combination and the attention put towards that data point by the models (RMSE), as well as the correlation between the models' values and the recruiters' values (pearson's correlation). The results can be seen in Table 4.

By weighing the performance of the models against their explainability score, I'll be able to show which model is the most balanced/excels at what. If there is a model that is both the best/competitive in terms of performance and explainability, that will obviously be the best. However, if explainability comes at a cost, the best model will not be set in stone. Therefore, in that case, I will mention how the best model depends on the use case, but that for the current use case, a model with good explainability is preferable.

REFERENCES

[1] Roy Assaf and Anika Schumann. 2019. Explainable Deep Neural Networks for Multivariate Time Series Predictions.. In *IJCAI*. 6488–6490.

[2] Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series.. In *KDD workshop*, Vol. 10. Seattle, WA, USA:, 359–370.

[3] Jaegul Choo and Shixia Liu. 2018. Visual analytics for explainable deep learning. *IEEE computer graphics and applications* 38, 4 (2018), 84–92.

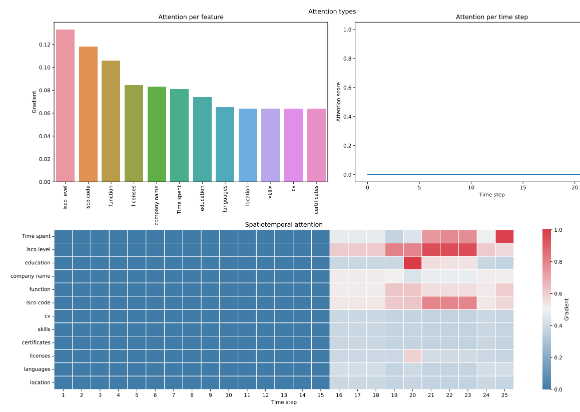
[4] Jerome T Connor, R Douglas Martin, and Les E Atlas. 1994. Recurrent neural networks and robust time series prediction. *IEEE transactions on neural networks* 5, 2 (1994), 240–254.

[5] Torch Contributors. [n.d.]. Torch.sparse¶. https://pytorch.org/docs/stable/sparse.html

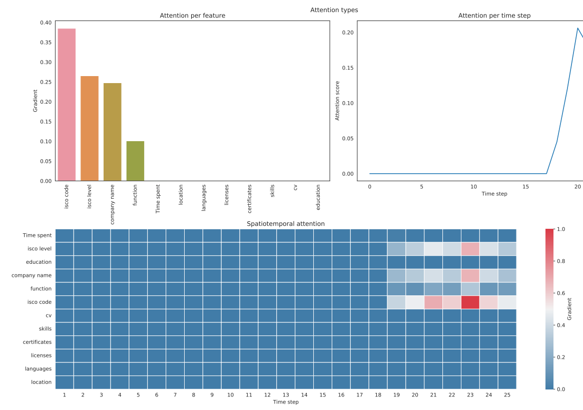
[6] Yukai Ding, Yuelong Zhu, Jun Feng, Pengcheng Zhang, and Zirun Cheng. 2020. Interpretable spatio-temporal attention LSTM model for flood forecasting. *Neurocomputing* 403 (2020), 348–359.

[7] Kevin Fauvel, Tao Lin, Véronique Masson, Élisa Fromont, and Alexandre Termier. 2021. XCM: An Explainable Convolutional Neural Network for Multivariate Time Series Classification. *Mathematics* 9, 23 (2021), 3137.

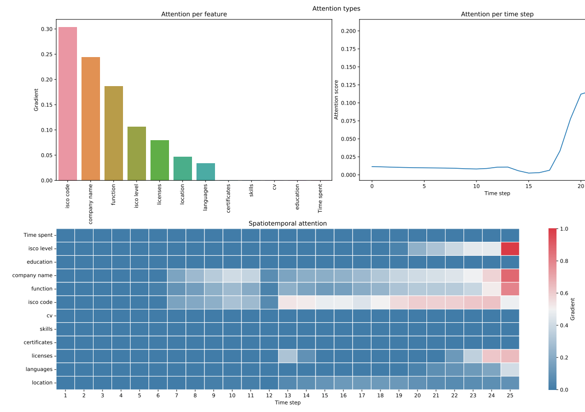
[8] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang. 2019. XAI—Explainable artificial



(a) Explanations provided by the explainable LSTM. Top left: attention per feature. Top right: attention per time step. Bottom: Feature/time step interaction (spatio-temporal attention).



(b) Explanations provided by the explainable CNN. Top left: gradient weight per feature. Top right: gradient weight per time step. Bottom: Feature/time step interaction (grad-CAM)



(c) Explanations provided by the explainable CNN-LSTM. Top left: gradient weight per feature. Top right: attention per time step. Bottom: Feature/time step interaction (guided backpropagation)

intelligence. *Science Robotics* 4, 37 (2019), eaay7120.

[9] Melanie Hanson and Fact Checked. 2021. Educational attainment statistics [2022]: Levels by demographic. https://educationdata.org/education-attainment-statistics

[10] Miao He, Dayong Shen, Yuanyuan Zhu, Renjie He, Tao Wang, and Zhongshan Zhang. 2019. Career trajectory prediction based on cnn. In *2019 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*. IEEE, 22–26.