DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

Deep learning models as decision support in venture capital investments

Temporal representations in employee growth forecasting of startup companies

SONJA HORN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Deep learning models as decision support in venture capital investments

Temporal representations in employee growth forecasting of startup companies

SONJA HORN

Master's Programme, Machine Learning, 120 credits
Date: June 15, 2021

Supervisor: Pawel Herman
Examiner: Erik Fransén
School of Electrical Engineering and Computer Science
Host company: EQT Ventures
Swedish title: Djupa artificiella neuronnät som beslutsstöd vid riskkapitalinvesteringar
Swedish subtitle: Representation av tid vid prediktering av anställningstillväxt för nystartade bolag

Abstract

Venture capital investors are constantly exposed to high levels of risk in investment scenarios. To lower that risks, various decision support tools can be exploited, such as machine learning models aimed at predicting startup success. In the related work, we observe a lack of deep learning models and solutions that cater to the time-dependent features that are naturally present in the data. This thesis compares two state-of-the-art deep learning models, and inherently their different temporal representations, in their ability to capture long-term dependencies in both time-dependent and static data. We consider the sequential Long Short-Term Memory (LSTM) and the attention-based Transformer in the comparison, motivated by their popularity and contrasting approaches to temporal representation. The method solves a binary classification problem: given time-dependent and static features describing a company, predict whether this company will have intense employee growth in the coming period. The thesis raises questions regarding the suitability of the Transformer under certain data conditions, and establishes that the LSTM is successful in representing long-term dependencies in the data, both with and without the influence of static features.

Sammanfattning

Riskkapitalinvesterare är konstant utsatta för hög risk i investeringsscenarion. För att minska risken kan olika beslutsstödsverktyg utnyttjas, såsom maskininlärningsmodeller avsedda för att prediktera framgången hos nystartade bolag. I relaterade studier observerar vi en brist på djupinlärningsmodeller och lösningar som möjliggör användning av tidsberoende variabler som är naturligt närvarande i datan. Uppsatsen jämför två av de främsta djupinlärningsmodellerna och deras inneboende tidsrepresentationer i deras förmåga att fånga långsiktiga beroenden i både statisk och tidsberoende data. I jämförelsen ingår den sekventiella LSTM-modellen samt den 'attention'-baserade Transformern, vilket motiveras av deras popularitet och mycket skilda sätt att representera tid. Metoden löser ett binärt klassificeringsproblem: givet tidsberoende och statisk data som beskriver ett företag, förutse om företaget kommer att växa intensivt i antalet anställda den kommande perioden. Uppsatsen väcker frågor kring Transformerns lämplighet under specifika förhållanden, och fastställer att LSTM-modellen framgångsrikt kan fånga långsiktiga beroenden, både med och utan påverkan av statisk data.

Contents

Chapter 1

Introduction

In the field of venture capital (VC), decision makers are constantly dealing with high levels of uncertainty and risk in investment scenarios. As a result, investors are always interested in expanding their toolkit of decision support in order to lower that risk and make safer bets. In this setting, machine learning has proven to be a valuable tool in predicting startup success [1–9]. Although previous works vary slightly in their use of features as well as model choice, most use at least general company information and funding data to make binary (and in one case multi-class) success predictions based on events such as IPO or acquisition [1–4, 8]. While these events are objective measures of success, they are not the only relevant proof of success, nor are they common outcomes. The essential skill of high performing VC funds is the ability to predict the valuation of a startup company in order to maximize their return on investment. Thus, a high valuation is a good approximation of success. Indeed, events such as IPO and acquisition indicate a high valuation and hence a successful outcome, however, they only apply to a very restricted amount of companies.

A more general valuation indicator is the rate at which the company team grows, the positive correlation being evidenced by several studies [10–12]. Startup companies commonly undergo extensive hiring processes in order to scale up their business and meet demands. The up-scaling is highly related to increased revenue generation, in turn being the most obvious approximation of valuation. However, revenue data (much like valuation data) on private companies is difficult to come by and thus not suitable as a prediction target. In addition to team growth indicating revenue growth, it also indicates increased spending in terms of salaries and other expenses. One could argue

that such companies are more likely to seek funding to balance their expenses, which suggests that a team growth prediction could also serve as a valuable deal sourcing tool. Lastly, a team growth prediction could be valuable when assessing portfolio companies as it provides insight into the expected growth in the coming period, enabling companies to take precautionary action if needed.

Apart from a common definition of success, another trait among related studies is the use of shallow machine learning models, where Logistic Regression and Decision Tree variants dominate [1, 3–5, 8, 13]. The authors have often chosen approaches and models that do not cater to the many time-dependent features that are widely considered as important metrics in VC, such as the number of domain visits, app ranks and employee counts that change over time. Similarly, many studies represent funding round data in a static manner, e.g. by aggregating the historical funding events into a 'Total Funds' feature [1–8, 13]. In a realistic setting, one would consider *when* the funding took place in relation to other fundings, the startup founding date, the current date and so on. Indeed, one would comparably consider the evolution of employee count or domain visits over time, as opposed to statically at the most recent moment. In modern deep learning, the most suitable representation of time is not obvious when working with time-dependent features. The rise of attention-based models such as the Transformer has mitigated the reliance on sequential learners such as Long Short-Term Memory (LSTM) models in some tasks, especially in natural language processing (NLP), where Transformers tend to outperform traditional LSTMs while lending themselves to parallelization [14–16]. However, some comparative studies show that the Transformer is more unstable during training [17, 18], and that LSTMs could be preferred in some situations where, for example, the dataset is smaller (the record setting Transformers are trained on millions of samples, for a very long time) [19]. Additionally, when the time-dependent input is scalar rather than samples from a finite set of possible values (as they are in NLP or image processing), the result of using Transformers is unclear. This degree project is concerned with two problems: VC success prediction and suitable temporal data representations in deep learning models.

1.1 Problem statement

The relevant information to predicting team growth of startup companies include both time-dependent features such as funding round data, employee count and domain visits, and static features such as the company location.

From these varying signals, investors are interested in predicting the team growth of the startup several months into the future in order to make profitable investments. Preferably, the prediction period p describing the time interval between the last observed time-dependent input and the prediction date should be relatively long, so that a value increase in a realistic time window can be modelled. Furthermore, the suitable temporal representation for the time-dependent features is unknown, taking the form of either a sequential feed of feature values or, alternatively, some positional encoding, i.e. a product of some embedded low-dimensional structures. In the first case, the inputs are fed into the model sequentially, thereby inherently capturing the temporal position of the inputs in relation to each other. A widely used and state-of-the-art deep learning model that realizes this approach is the LSTM [20]. In the second case, the input is fed as a fixed length vector, which is complemented with another vector containing the positional encoding of each input. The attention-based Transformer implements this idea [16], and has in some predictive tasks challenged and even outperformed the LSTM.

In summary, this project evaluates different temporal representations and their ability to model long-term dependencies when combined with the representation of static features. The conditions and engineering issues described above lead us to the following research question:

When predicting employee growth for startup companies, how does the temporal data representations of the LSTM and the Transformer compare in their ability to capture long-term dependencies with and without the influence of static input features?

1.2 Scope and limitations

This project considers two different approaches to temporal representation: a sequential approach and an approach including positional embeddings and fixed length inputs. The models chosen to exploit and evaluate these approaches are LSTM and Transformer models, respectively, motivated by their state-of-the-art performances and frequency of use in recent studies. Other possible models that fit the requirements of the two approaches are not considered in this project. The approaches are compared only in the context of a very specific problem, and not across multiple problem domains. Furthermore, this project does not focus on the interpretability of the decision made by the deep learning models in this project, but instead focuses on their

objective ability to model long-term dependencies with and without the input of static features. Lastly, feature selection also falls beyond the scope of this project, where we instead rely on field expertise to select the features.

A comparison of methods and models in this application area is rather complicated due to the lack of a reference dataset. Indeed, the authors of the studies in section 2.2 use their own datasets in their experiments, and consequently to reach conclusions about model performance. Thus, their findings can not be appropriately compared or transferred to other application areas or even datasets. These shortcomings are inherited in this project, for which the comparisons are also based on a dataset unique to the company and problem definition. In other application areas such as NLP, several benchmark datasets are available to compare models and subsequently draw general conclusions of model suitability and performance for a specific application area. As mentioned, this project and the application area it targets suffer from the lack of a reference dataset and thus the conclusions of this project are limited to similar datasets and/or prediction tasks. Moreover, this project makes some assumptions about the model architectures (such as the considered hyperparameters). Other assumptions may or may not affect the outcome, thus limiting the results and conclusions of this project to the specific assumptions in this project.

1.3 Outline

In chapter 2, the reader is first introduced to the most essential VC investment practices, followed by an introduction to the deep learning models used in this project. The related work on startup success prediction is presented and summarized. Chapter 3 describes the methods used to answer the research question, including data preprocessing, a definition of the prediction task, the considered model architectures and the evaluation strategy. The results are presented in chapter 4. The results are discussed and analyzed in chapter 5 and, finally, the most important findings and suggestions for future work are presented in chapter 6.

Chapter 2

Background

2.1 VC investment strategy

VC as a business is mainly motivated by the opportunity to generate financial returns for the fund's investors, a so called return on investment (ROI). When measuring the success of VCs, a common measure and indicator of ROI is the multiple on invested capital (MOIC) which compares the capital distributed over a portfolio of investments (or a single deal) to the realized and unrealized amount(s) returned upon exit(s). A desirable exit for a VC is that a portfolio company is either acquired by another company or goes public at a much higher valuation than at the time of the initial investment. The exit then generates a realized return. The MOIC can be expressed by equation as

$$MOIC = \frac{Realized\,value + Unrealized\,value}{Total\,amount\,invested} \tag{2.1}$$

Naturally, the aim is to source investment with great growth potential that result in higher valuations in the future which in turn increases the MOIC. The ability to foresee growth in the valuation of a company is thereby of the essence for all VC funds. If a company has not yet exited or gone public, another way to measure the success of the investment is through the post money valuation of consecutive rounds, i.e. the value of the company taking into account the newly acquired capital. If a VC fund invests in an early round in return for company shares and the next round results in a much higher post money valuation, the shares from the first investment are consequently worth more.

In recent years, a high valuation has not necessarily been accompanied by a net profit. New business models have emerged where the company has presented red figures since their inception while simultaneously growing their valuation exponentially, Spotify being just one example. Other companies have received high valuations before exhibiting any revenue based on factors like market potential, impressive technology, the founding team etc. At the very least, using financial figures as approximations of a valuation is risky. Additionally, private companies' financial data is highly exclusive (i.e. not available in large scale), which outrules such variables as candidates to approximate valuation in a predictive machine learning model.

2.2 Related work

2.2.1 Startup success indicators

Regarding success indicators of startups, previous studies show different but not necessarily contradicting evidence of key indicators. Kaplan et al. [21] concluded in a 2009 study that the startup's core business is more important than the founding team when trying to foresee the future success of the company. Another study of Russian and Latvian companies suggested that founders with experience in building companies and managing a professional team are more likely to attract funding than other founders [22].

A survey by Da Rin et al. [23] concluded that one strong success indicator is indeed VC backing, and that startups that receive this sort of funding are more likely to go public that non-VC backed startups. Yet another study [24] suggests that VC-backed startups operating in hot markets often end up bankrupt, however if they do make it to an initial public offering (IPO), the valuation is often higher than that of startups in non-hot markets.

Furthermore, Pagano et al. [11] have shown that apart from the market-to-book ratio, the company size is the most important determinant of an IPO, i.e. that larger companies are more likely to go public than smaller companies. This suggests that growing companies are indeed good candidate investments for VCs, indicating successful exits (IPO). Related to this are the findings of [12] by Stanescu et al. that proved a direct link and statistically significant correlation between the number of employees and both turnover and total income (which are both company value drivers).

The non-data driven approaches to predicting startup success are not consistent nor contradicting in their findings. There may be several important startup success indicators that all affect the outcome without excluding each other.

2.2.2 Machine learning and data driven startup success predictions

Various approaches have been taken to predict startup success using machine learning, most of which are based on a success versus failure classification task where the definition of success and failure differ slightly.

Xiang et al. [25] used a Bayesian Network to predict the success of startups, considering both factual and topic features. A startup was deemed successful if it got acquired and unsuccessful otherwise, even if it went public. On the contrary, Bento [3] labeled a startup as successful in the event of an acquisition or an IPO, using Random Forests to make the prediction based on factual features. Also Ünal [4] used acquisition or IPO as an indicator of startup success, also implementing Random Forests along with Ensemble Methods and Extreme Gradient Boosting (the latter outperforming the other models). Krishna et al. [8] used funding data for a success versus failure prediction, and found that Logistic Regression, Random Forests and Alternating Decision Tree performed better than other alternatives including a Bayesian Network and Naive Bayes classifier.

Other studies like those of Ghassemi et al. [5], Böhm et al. [26] and Lussier at al. [13] define success in terms of survival, where [5] only expected a startup to solely exist after a 2 year period to be deemed successful. [26] also required the revenue growth to exceed some threshold, while [13] labeled a startup as successful if it had made at least industry-average profits for the previous three years. [5] used a mix of team information and ventures properties based on free-text descriptions to make the predictions and obtained best results using Logistic Regression over the next best Support Vector Machine, Decision Trees, Discriminant Analysis, k-Nearest Neighbors, Ensemble Methods and Neural Networks. On the other hand, [13] only implemented Logistic Regression and used mostly managerial and team data and initial conditions (e.g. start capital) as features for the model. [26] only tried a Support Vector Machine classifier following an initial k-means clustering of companies, based on business model patterns developed by

Gassmann et al. [27].

Unlike the previously mentioned approaches that try to model a binary classification task (success/failure), Arroyo et al. [9] utilize multi-class prediction labels that describe the next event in the startup life cycle. The events were "acquired", "funding round" (another round), "IPO", "closed" (shut down) or "no event". Also here, several models were tested and Gradient Tree Boosting outperformed Support Vector Machines, Decision Trees (worst performance), Random Forests and Extremely Randomized Trees. Another study that tried to predict a future event was that of Sharchilev et al. [7], where the authors used a Neural Network based model to predict if a startup (given initial angel or seed funding) would attract follow up funding in the future given general, investor and team/staff features. An interesting find from this study was that team features were the weakest predictors, agreeing with the findings of [21] and thereby, in some sense, disagreeing with the results of [22].

Deep learning models are underrepresented in this application area, where Gastaud et al. [6] are among the few who used such a model (a Graph Neural Network) to identify the most predictive success factors for early stage and later stage (Series B) companies. Notably, they found that they vary between these two company stages, and that a low intensity of competition increases the probability for early stage companies to succeed, while this feature had little effect on the outcome of later stage companies. However, they report that the company network features had big predictive power for later stage companies.

In general, Logistic Regression and variations of Random Forests (e.g. extreme gradient boosting) have shown to perform well in startup success predictions. Varying feature sets have been used for the task, where funding and factual features are recurring. Notably, there is a clear gap in the related work regarding Deep Learning with all the above mentioned approaches being shallow, basic model architectures that do not cater to time-dependent features. This report covers two state-of-the-art deep learning models for startup success prediction that are able to take both static feature inputs much like the models presented above, but also time-dependent features that change over a time.

2.3 Sequential temporal representation: LSTM

One suitable way to account for the temporal aspect when dealing with time-dependent features is to use sequential learners that process the input according to their temporal location. Such models process the feature value at one timestep together with the information obtained by processing the input at the previous timestep, thereby being fully dependent on the outcomes of previous steps to continue learning. The consequence of this setup is that the models are temporally dynamic, but also that the learning can not be parallelized.

2.3.1 Vanilla RNN

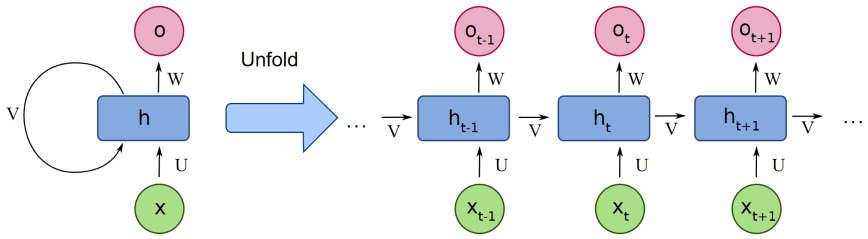


Figure 2.1: Vanilla RNN

The simplest version of an artificial neural network (ANN) of this sort is the basic recurrent neural network (RNN). At each timestep, the RNN unit produces an output and a hidden state using the input at the current timestep and hidden state from the previous timestep. Figure 2.1 shows the simple RNN unfolded through time, where the model inputs and outputs are defined as:

$$h_t = \sigma(Ux_t + Vh_{t-1} + b_h) \tag{2.2}$$

$$o(t) = \sigma(Wh_t + b_y) \tag{2.3}$$

where x_t is the input at time t, h_t the hidden state and o_t is the output. σ represents an activation function and the weights U, W, V and biases b_h and b_y are learned parameters. RNNs are trained with backpropagation through time (BPTT), and suffer from the same issues as very deep regular neural networks - exploding and vanishing gradients when backpropagating the error.

2.3.2 LSTM

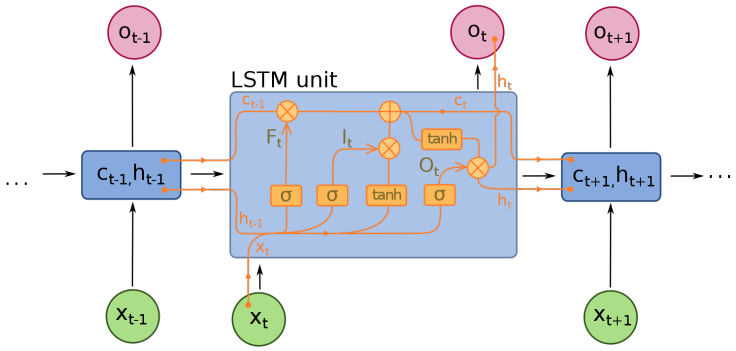


Figure 2.2: LSTM unit

In 1997, Sepp Hochreiter and Jurgen Schmidhuber published the paper *"Long short-term memory"* [20], which proposed a new kind of RNN using their LSTM units that allowed the model to remember important information through long time lags and simultaneously forget unnecessary information along the way . The suggested architecture solved the problem of unstable gradients in the learning process, and has become widely used in sequence learning problems.

The LSTM *cell state* can be regarded as the long-term memory of the model. It is updated each recursion by gated operations that control what information the cell state should contain through addition and forgetting. First, the *forget gate* determines what information should be forgotten from the cell state given the previous cell state. Next, the *input gate* determines what information should be added to the cell state considering the previous hidden state and the observation at time t, leaving some information forgotten. Finally, the *output gate* controls what information should be passed on to the next recursion in the form of a hidden state at time t, and what information should be outputted from the model at time t.

By equation, the LSTM unit that is displayed in Figure 2.2 can be expressed as:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{2.4}$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{2.5}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{2.6}$$
$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{2.7}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \tag{2.8}$$
$$h_t = o_t \circ \sigma_h(c_t) \tag{2.9}$$

where, again, x_t is the input at time t, h_t the hidden state and o_t is the output. σ represents an activation function and the weights W and U and biases b are learned parameters that decide how the input, output and forget gates operate, and thereby how the long-term memory is created.

Apart from the obvious advantage of stable gradients in the learning process compared to a RNN, the LSTM has also proven to outperform other models in time series forecasting, especially for large datasets containing non-linearities and long-term dependencies. Compared to the popular Autoregressive Integrated Moving Average (ARIMA) model, the LSTM model has proven to significantly improve the predictive performance [28–32]. Furthermore, Yan and Ouyang [33] reported that the LSTM model outperformed Multi-Layer Perceptron, Support Vector Machine and k-Nearest Neighbour models in the task of forecasting financial time series. The LSTM model has become a staple in sequence learning tasks and has reached state-of-the-art performance across many applications.

2.3.3 Deep LSTM

The concept of depth is more straight forward in traditional feed-forward networks than it is for RNNs. RNNs are inherently deep in time, but there are representation benefits to increasing the spatial depth as well. In traditional feed-forward networks, the motivation for adding layers is that they allow the model to learn representations at multiple levels of abstraction, hence learning more complex patterns than shallow architectures [34, 35].

Recurrent neural networks can increase their spatial depth by stacking multiple layers much like a traditional feed forward network. The output at each timestep of one layer is then used as input to the next layer's unit, an approach that *"allows the hidden state at each level to operate at different*

timescale" [36]. This type of architecture has reached state-of-the-art performance in sequence learning tasks such as language modelling, music prediction and handwriting recognition [36–38].

As for temporal depth, there is no definite rule for the best number of nodes in each LSTM layer. The best architecture is often found through a trial and error process like k-fold cross validation.

2.4 Alternative temporal representation: Transformer

Unlike the LSTM which has a sequential approach to time, an alternative solution is to represent time as a product of some embedded structures that can be defined in a lower dimension. The idea is to complement the input at time t with a temporal embedding describing the temporal position of the input. Unlike the LSTM architecture which sequentially takes the input, models of this sort can process all inputs simultaneously much like a traditional feed-forward network. A recent and state-of-the-art example of this type is the Transformer model, especially replacing LSTMs in natural language tasks where performance exceeded the previous records on benchmark datasets [16]. Apart from performing well, the Transformer architecture also lends itself to parallelization which is a huge advantage in terms of training (and inference) efficiency.

2.4.1 Transformer

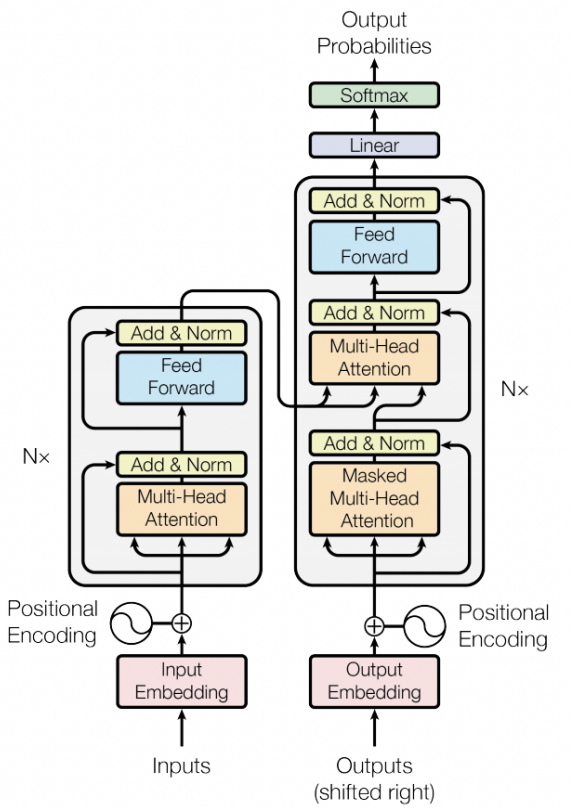


Figure 2.3: Transformer [16]

The Transformer was first introduced in the 2017 paper *"Attention is all you need"* [16] as a sequence-to-sequence model made up of two main components, an encoder and a decoder as illustrated in Figure 2.3. The encoder consist of N identical layers, each layer with a multi-head attention mechanism and a fully-connected feed-forward layer, as well as residual connections and layer normalization. The decoder too consists of N identical layers with the same sub-layers as the encoder, with the addition of another multi-head attention layer which targets the right-shifted decoder output.

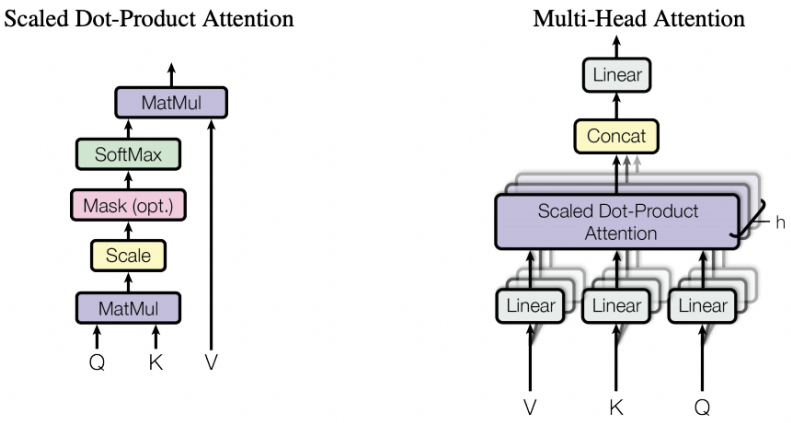


Figure 2.4: Scaled Dot-Product Attention (left) and the Multi-Head Attention Mechanism (right) [16]

The multi-head attention mechanism can be divided into two steps. The first is a *scaled dot-product attention*, defined by the equation:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.10}$$

where Q is the query, K are the keys and V are the values. d_k is the dimension of a query which is the same as the dimension of the keys. The scaled dot-product attention essentially multiplies V with with a weight a which is defined by how each input in the sequence (Q) is influenced by all other inputs in the sequence (K). The softmax function transforms these weights into a distribution between 0 and 1 that are then multiplied with V.

The right part of Figure 4 illustrates a linear projection of Q, K and V onto d_k, d_k and d_v dimensions respectively, conducted h times using h different learned linear projections (d_v is the dimensionality of V). The variable h is often referred to as the number of *heads*. After the projections are fed into the attention function (equation 2.10), the h outputs are d_v-dimensional. All outputs are concatenated and projected again resulting in the final values. This multi-head attention mechanism can be expressed by equation:

$$MultiHead(Q, K, V) = concat(head_1, ..., head_h)W^O \tag{2.11}$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.12}$$

where W_i^Q is the projection of Q onto a d_k-dimensional plane, W_i^K is the projection of Q onto a d_k-dimensional plane and W_i^V is the projection of V onto a d_v-dimensional plane. W^O is the projection of the output of the attention mechanism and is $\in R^{hd_v \times d_{model}}$.

Q, K and V are different depending on where in the model they are used, allowing the model to attend to different parts of the encoder and decoder input sequences. For example, the decoder uses *masking* in the multi-headed attention mechanism which attends to the right-shifted decoder output to ensure that the model does not have access to "future words" of the output.

The Transformer takes a fixed length input containing the input sequence as well as a positional encoding for each input to account for the temporal ordering of the sequence inputs. In the original paper, the positional encodings were obtained through applying a sine or cosine function to the scaled position index depending on if the position is even or uneven respectively. These positional encodings were fixed, thus not learnable, and summed with the original inputs before being fed to the model. While the original implementation was purposed for language translation tasks where each word has a unique embedding and indeed benefited from the summation of inputs and positional encodings, a model which takes scalar values (such as the one considered in this project) requires another approach to positional encoding.

2.4.2 Time2Vec

Kazemi et al. [39] developed an alternative representation of time than that used in [16]. It is invariant to time rescaling, simple, and able to capture both periodic and non-periodic patterns. It can be expressed by equation as:

$$t2v(\tau)[i] = \begin{cases} \omega_i \tau + \varphi_i, & \text{if } i = 0 \\ \mathcal{F}(\omega_i \tau + \varphi_i), & \text{if } 1 \leq i \leq k \end{cases} \tag{2.13}$$

where τ is the time series, $\mathcal{F}$ is a periodic activation function like the

sine function, k is the $t2v(\tau)$ dimension and ω, φ are learnable parameters. $t2v(\tau)[i]$ is then the i^{th} element of $t2v(\tau)$. The advantages of this approach compared to the original one is that is can capture periodic (the second case in equation 2.13) as well as non-periodic (the first case in equation 2.13) patterns. It is also flexible given the learnable parameters, meaning it does not constrict the frequencies and time-shifts to pre-defined values when modelling the timeseries. Finally, $t2v(\tau)$ is fed as an input to the model, and is not added to the input at time t as the positional encoding of [16], thus preserving the integrity of scalar input sequences while successfully representing their position in relation to inputs at other time steps.

2.5 LSTM and Transformer: compared

2.5.1 Performance and stability

Most comparative studies on LSTM and Transformer models are based on NLP tasks, and the findings are contradicting. Merkx et al. [14] and Karita et al. [15] both found the Transformer to outperform RNN-like architectures on their respective reference datasets, and so did [16] in the original paper. However, Zeyer et al. [40] found that the LSTM and Transformer models reached similar performance, although they reported that the Transformer was more prone to overfitting, required less training time and was more stable in the training. On the other hand, multiple studies [17, 18] suggest the opposite, namely that the Transformer in general exhibits unstable training patterns as a result of gradient vanishing due to an undesirable interaction between the residual connections and layer normalization operations. Finally, one study [19] highlights that the comparative performance of LSTM and Transformer architectures depend heavily on the size and nature of the dataset. For small datasets however, they found that the LSTM model reaches significantly higher results than the Transformer.

2.5.2 Modelling long-term dependencies

Although LSTMs have a longer memory than vanilla RNNs, the memory is still restricted. At each time step, the memory is warped to some degree (even though it is controlled by the gates) meaning the dependencies between the input and the target are affected by the temporal distance between them. The context vector is created from the last hidden state of the LSTM, and can not access the original information once the recursions are completed. Hochreiter

et al. [41] argue that the distance between the input and target (the length of the *path*) substantially affects the ability of the network to learn long-term dependencies. In that sense, they suggest that dependencies that grow longer and longer get increasingly harder to learn for LSTMs since the length of the path increases in parallel.

In contrast to the LSTM, the context vector created by the Transformer encoder has access to the entire input sequence through residual connections and the multi-headed attention mechanism, meaning it has (in theory) unlimited memory and bidirectional properties. The distance between the input and target is consequently shortened. Essentially, this means that the Transformer encoder is not restricted by the distance between the input and target. [16] claims that the shortened path, and thus unlimited memory, allows the Transformer to better represent long-term dependencies compared to recurrent neural networks.

Assume that a dataset of input sequences indeed contain long-term dependencies, i.e. that the future behaviour of the sequences depend on behaviours in the distant past. Further assume that the prediction period (the temporal distance between the cutoff and prediction date) increases. The conditions described above suggest that the Transformer would (given all other conditions equal) be better suited for longer prediction periods, since it can better learn long-term dependencies that are supposedly present in the data.

Chapter 3

Method

The goal of this project is to compare two distinct and different approaches to representing time-dependent features in order to predict employee growth of startup companies. We examine how these approaches are able to capture long-term dependencies using only time-dependent features as well as a combination of time-dependent and static features. The considered approaches are directly connected to the LSTM and Transformer models, which have distinct mechanisms to account for time and temporal embeddings. The task for both approaches is to predict whether the employee count of a startup will grow significantly in the next few months. The employee count variable is used as an approximation of a future valuation, motivated by its availability and strong correlation with increased value.

Both model approaches should thus be able to take both time-dependent features (such as number of domain visits in a time period) and static features (such as the country of origin) as input and predict if the company will grow significantly within a longer prediction period. An increased length of the prediction period tests the models ability to model long-term dependencies by restricting the influence of nearby observations and relying on observations far back in the sequence to make the prediction.

The time-dependent features are collected by the company from different data sources, and as a result they differ in measurement practice in terms of measurement frequency and time interval. Therefore, section 3.1 will focus on feature engineering and preprocessing of data. Section 3.2 defines and explains the model input, label and evaluation strategy, while section 3.3 and 3.4 describes the model architectures that are considered in this project.

3.1 Data description & preprocessing

The data is sourced by the company (EQT Ventures), who in turn sources the data from different providers. For example, App Store ranking data and domain visits are both examples of time-dependent features used in this project that are sourced from different providers. Similarly, funding data and basic organisational data have their unique sources.

The unprocessed dataset consists of data on startup companies that operate within the regions that EQT Ventures does. The companies have recorded venture capital related investments such as angel investments, seed rounds, series A rounds and so on, which qualifies them as potential investment opportunities. The number of data samples prior to any preprocessing is 105369. Note that each company contributes to only a single sample at this point. After the preprocessing described in sections 3.1.1 to 3.1.5, the number of samples in the final dataset reaches 154666. In the process, companies have been removed, but several samples have possibly been created from a single company according to 3.1.5, resulting in an expansion of the dataset.

The feature selection is based on expertise within the company and consists of:

- Five static features, including the country of origin, the company description, the date of founding, the total number of funding rounds and when in time the sample is taken from in relation to the current date.
- Two funding features that are handled as time-dependent features, including the funding amount and cumulative total funding at various time-steps.
- Six time-dependent features, including three that describe the number of visits on the company web domain (all from various sources, one of them counting only mobile visits and another counting only desktop visits), the employee count and two global rankings from different sources.

The first data cleaning step consists of excluding any companies that do not have recorded employee count data for at least 18 consecutive months between January 2014 and January 2020 to ensure there is enough training data given a longer prediction period. All data after January 2020 is discarded due to the

possibility of it being affected by the COVID-19 pandemic (while the goal is to model a "natural" employee growth behaviour). All data prior to January 2014 is deemed too outdated and thereby is also discarded.

3.1.1 Static features

To handle categorical static features (country of origin), One-Hot Encoding is enforced. The k classes are strings and thus first mapped to a unique integer value, which is subsequently used to map each class to the k-dimensional sparse one-hot vector.

The company descriptions are encoded using a pre-trained token-based text embedding model from TensorFlowHub, trained on the English Google News 7B corpus [42]. The model maps a text input to a 50-dimensional embedding vector, which can then be used as input to a neural network.

Numerical static features are scaled in the same way as the numerical time-dependent features (see the section 3.1.4).

3.1.2 Time-dependent features

The time-dependent features for a single company may be of different length, i.e. they could start, end and be measured at different time-steps since measurement practices and frequency differ among the sources and thus time series. The goal of the preprocessing steps is that all series should be of equal length for every company (though the time interval may vary between companies). This ensures that the models receive a fixed length input (sequence or vector) for each sample. Although this is not a necessity for recurrent networks (which is one of the approaches tested in this project), it is indeed a necessity for the Transformer network (the other approach tested). Since the same data will be used for both models in order to fairly compare them, this preprocessing step is necessary.

All time-dependent features of a company are bounded by a start date, t_0, which is defined by the first recording of employee count and a stop date, t_s defined by the last recording of employee count. By definition, there will be employee count data for the 18 months leading up to t_s, but there might be missing values in the middle of the series. For other features, there could be missing values in any capacity that need to be estimated in order for the

time series to be of equal length, i.e. between t_0 and the first recording, in the middle of the series of between the last recording and t_s.

Data imputation between t_0 and the first recording of any time series is handled by zero padding. If missing values appear between the last recording and t_s, such values are estimated using the Last-Observation-Carried-Forward (LOCF) method, which imputes the last recorded value of the series at each succeeding timestep. In regards to missing values in the middle of a time series, such are estimated by means of linear interpolation according to the equation below:

$$y = y_1 + \frac{x - x_1}{x_2 - x_1}(y_2 - y_1) \tag{3.1}$$

$$\tag{3.2}$$

where x is the known value on the x-axis, y is the missing value, x_1 and y_1 are the lower bounding coordinates and x_2 and y_2 are the upper bounding coordinates.

Furthermore, all data associated with a date succeeding t_s is discarded. Similarly, data prior to t_0 is discarded.

The result of this data imputation and cleaning is that all series are of equal length for every company, while the actual time interval may vary between companies. Although we may suffer from information loss for time series other than employee count when discarding data prior to t_0 and succeeding t_s, we simultaneously guarantee that each company has valid and non-estimated employee count data for the entire interval. Later on (described in section 3.1.5) when each company may generate multiple samples from its timeseries interval, it is crucial that there is employee count data for the entire range and not only for some of those generated samples, as it would otherwise interfere with the creation of labels (see section 3.2.2).

3.1.3 Funding features

Unlike time-dependent features that are measured consistently, funding information is only recorded on the relevant funding date. The chosen method to handle the funding information is therefore to spread it along the time interval of the time-dependent features (that are now of equal length). Prior to

that, possible duplicates are removed. Duplicates are common in the funding data, and are characterized by their closeness in recorded funding date (in reality, funding events are not very close in time). Between two duplicates describing the same funding event, the recording associated with the most information is kept (one of them might include the funding amount, while the other does not).

Similarly to the time-dependent features, any funding data prior to t_0 or succeeding t_s is discarded, with the exception of the numerical static feature which describes the total amount of funding in the company history. This information is essential to keep intact, since it provides an overview of the funding behaviour of a company unlike features that describe *when* funding events take place.

The funding data imputation resembles that of the regular time-dependent features in that zero padding is used to estimate missing values in the lower part of the interval if the first funding event within the interval occurs after t_0. The difference is that LOCF is used to estimate values for all timesteps between two funding events and succeeding the last funding event, instead of linear interpolation. The funding data will hence evolve to have a step function appearance over the time interval, where each step indicates a funding event and the size of the step is determined by the amount raised.

3.1.4 Feature scaling

The time-dependent numerical features and static numerical features are scaled with Min-Max Scaling, where values are mapped to a range [0, 1] but remain equal in their difference related to each other. This can be interpreted as squeezing the values into a smaller range.

3.1.5 Rolling window

Each company can contribute to several samples in the final dataset by sliding a fixed size window over the available timeseries and thereby sampling from the original data. The window size that is used is 48 months, and a new sample is taken every 12 months to avoid samples that are very similar (too close in time). The result of this operation is an expansion of the dataset while ensuring that all samples are of equal size.

3.2 Binary classification using static and time-dependent features

Predicting intense employee growth of startup companies can be interpreted as a binary classification problem where one class indicates that such growth has indeed taken place, while the other class consists of all other outcomes. The threshold for a sample belonging to the growth class will differ between the considered prediction periods, and is defined in section 3.2.2. The label thus depends on the prediction period, which in turn varies in the experiments in order to evaluate the models' ability to represent long-term dependencies. Furthermore, the input to the models varies between experiments, as some include only time-dependent features and others both time-dependent and static features.

The full model architectures are split between an encoding pipeline (the two different models) and the unchanged static feature encoding which is concatenated with the encoder output. The encoder models considered are stacked LSTMs and an adapted Transformer encoder, further described in section 3.3 and 3.4.

The optimal hyperparameter setting of both encoders are set prior to varying the prediction period and adding static features, using only time-dependent features and the shortest considered prediction period in this project. After the encoder architectures are set, the models are evaluated on their ability to represent possible long-term dependencies with and without the input of additional static features.

The following sections describes the model input, label, dataset sampling, evaluation and considered model architectures.

3.2.1 Model input

For each sample $n \in (0, N)$ there is an input X_n carrying the time-dependent features, defined by:

$$X_n = \begin{pmatrix} x_n^1(t) \\ x_n^2(t) \\ \vdots \\ x_n^M(t) \end{pmatrix} \tag{3.3}$$

where M is the number of time-dependent features and $x_n^m(t)$ is the m-th time-dependent feature, i.e. a time series depending on time t.

Furthermore, there is a separate input that contains information about the temporal position of the sample in relation to other samples. This input, T_n, is a scalar value based on the number of days between the cutoff date of sample n and the current date. T_n is an important feature as it allows the model to learn different behaviours that depend on where in time the sample originates from. It is not unlikely that the employee growth behaviour in recent years differs from the behaviour historically. T_n is concatenated with the output of the LSTM or Transformer encoder according to figure 3.1.

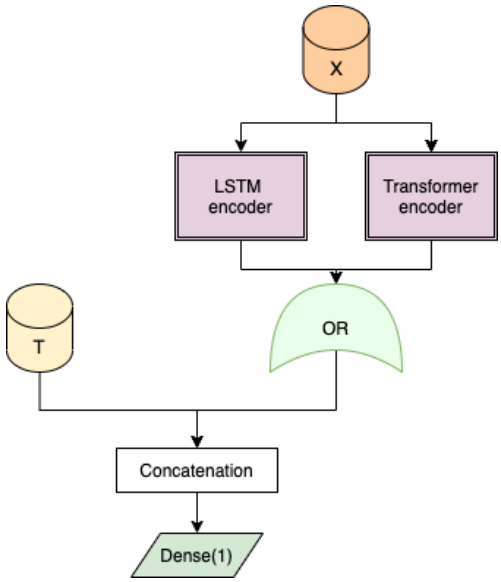


Figure 3.1: After X is encoded using either the LSTM or the Transformer encoder modules, the encoder output is concatenated with T in a concatenation layer, which is then passed through an additional dense layer generating the final model output.

X and T are used as inputs in all experiments and constitute the

preliminary features.

The *secondary features* are made up of input A. For each sample $n \in (0, N)$, A_n is:

$$A_n = \begin{pmatrix} a_n^1 \\ a_n^2 \\ \vdots \\ a_n^P \end{pmatrix} \tag{3.4}$$

where P is the number of static features and a_n^p is the p-th static feature for the sample. The secondary features are not used during hyperparameter selection or the initial experiments. However, when used, the secondary features will be concatenated with the encoder output and T after possible initial encoding according to figure 3.2. Note that A^4 (One-hot encoded country feature) is passed though an initial dense layer to reduce dimensionality, while A^3 (company description) is passed though the pre-trained token-based text embedding model, while the scalar valued A^{1-2} are immediately concatenated.

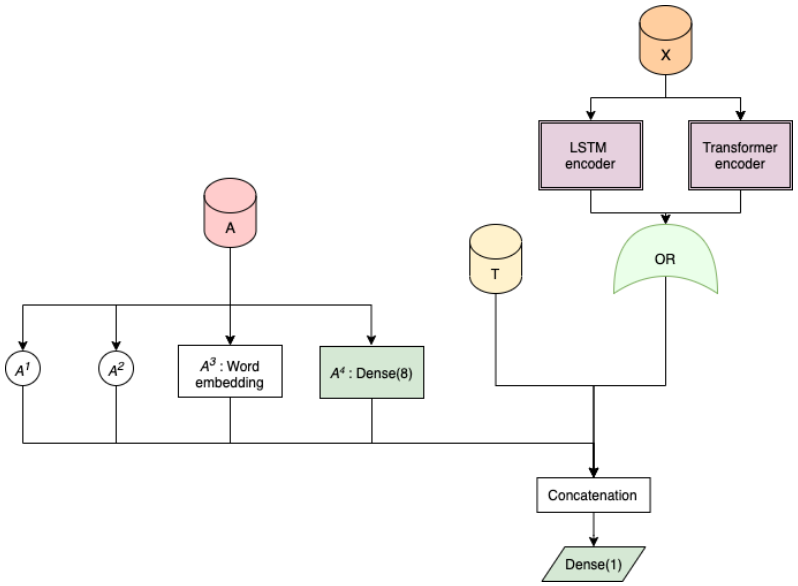


Figure 3.2: If secondary features A are used, they may be encoded separately before being concatenated with T and the encoder output. The concatenated data is then passed through a dense layer generating the final model output.

3.2.2 Data labeling

For each sample $n \in (0, N)$ the target label y_n is a binary target, i.e. either 0 or 1 according to:

$$y_n = \begin{cases} 1, & \text{if } growth\ case \\ 0, & \text{otherwise} \end{cases} \tag{3.5}$$

where a *growth case* is defined as a company that grows at least $g\%$ in the prediction period p while having at least 10 employees in the beginning of the period. The limit of 10 employees prior to the prediction period is implemented to avoid that large relative growths that are actually small growths in the number of employees (e.g. from 2 to 5 employees) are considered as growth cases. The growth g required to classify as a growth case varies depending on the prediction period p, as the percentual growth g is steeper (and thus less common) for smaller p.

For each p, g is obtained by sorting the dataset on growth and letting the threshold be defined by the smallest growth in the top 7%. This labeling strategy ensures that the minority class is of equal size for all p, and the resulting thresholds are $g_6 = 23.5\%$, $g_{12} = 41.2\%$ and $g_{18} = 55.1\%$. Consequently, Class 1 constitutes a minority class and the predictive task is considered an anomaly detection where the goal is to distinguish the few growth cases from the general population.

These values are validated through an analysis of startup companies that have increased their valuation intensely between two consecutive funding rounds (*golden companies*), measured through the post money valuation. Ideally, a VC fund would like to be able to predict the employee growth p months into the future with the starting point being prior to the first funding round in order to make a good investment.

First, the employee count is measured three months prior to the first round, at t_f. It is again measured p months after that and the growth is calculated in percent. For $p = 6, 12$, and 18 months, over 74% of the golden companies met the growth requirements of the thresholds g_6, g_{12} and g_{18}. The percentage of companies that met the requirements grew as p grew. The analyses, visualized in figure 3.3, shows that g_6, g_{12} and g_{18} are common outcomes in the golden set but anomalies in the general population, validating that the thresholds suggest

an increase in valuation and that successful companies grow in this distinct way in terms of employee count. The label is thus consistent with the overall investment strategy of VCs.

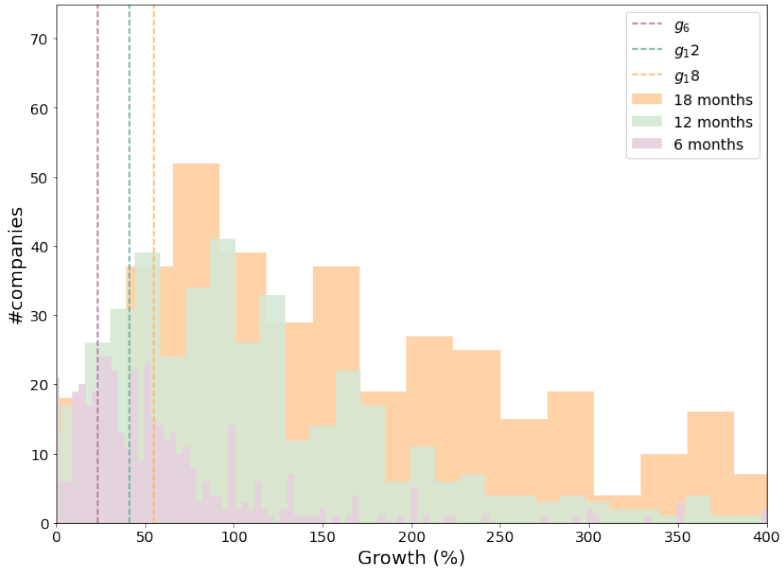


Figure 3.3: Analyses of *golden companies* and their growth behaviour for 6, 12 and 18 months in a value increasing period, validating the choice of label thresholds g_6, g_{12} and g_{18}.

3.2.3 Data handling - upsampling the minority class, validation and evaluation split

The dataset is first split into a train/test split of 80/20. The test set remains unseen, while the train set is iteratively split into a train and validation set following a 5-fold cross validation training scheme to determine the hyperparameter settings. After the hyperparameters are fixed and the model architectures are final, the models are evaluated on the unseen test set according to section 3.2.4. The original train set (80%) is used for statistical testing. We generate 10 evaluation measures per model by 10 times randomly selecting 20% of the data used for testing and training on the rest (using 20% for validation).

Because the dataset is significantly unbalanced, a machine learning model would reach high classification accuracy (93%) by classifying all samples as negative, and would probably converge to that behaviour. One

solution is to drastically upsample the minority class so that the classes are of equal size, thus forcing the model to learn the representation of both classes. An upsampled set is in this case considered the sum of an upsampled minority class and a majority class, where the minority class is upsampled to the size of the majority class. The upsampling is random, and simply makes copies of samples in the minority class until the desired size is obtained.

Upsampling is only performed on the relevant training and validation set for a specific run, leaving the test set in its original condition. The performance on the validation set is used to enforce early stopping and an unbalanced validation set would thus favor a classifier which classifies all samples to the majority class. Therefore, the validation set is also upsampled. The upsampling procedure is performed after the dataset splits are complete to guarantee that there is no data leakage between sets, i.e. that two copies of the same sample end up in different sets. Regarding data leakage, each split of the data is performed based on company ID, such that two samples stemming from the same company time series (see section 3.1.5) with possible, although not severe, overlap do not end up in different sets.

The full data handling pipeline is presented in figure 3.4

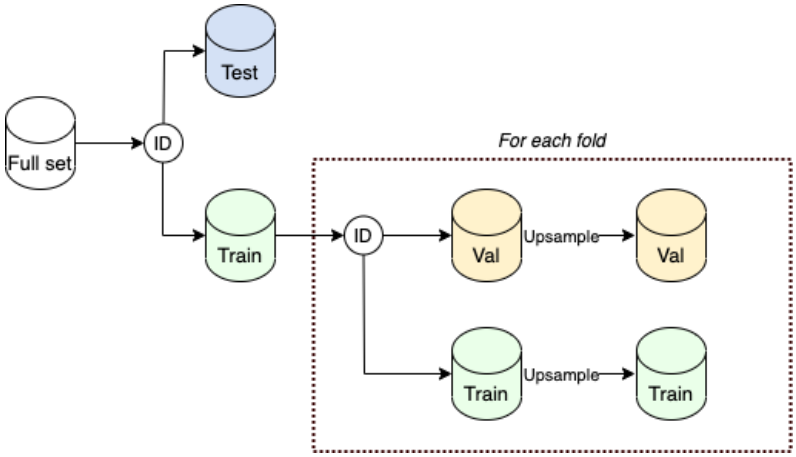


Figure 3.4: Data handling pipeline: full dataset is split into train/test on company ID. Subsequently, the train set is split into train/validation on company ID. The resulting train and validation sets are then upsampled.

3.2.4 Evaluation

As described in section 3.2.3, the evaluation is performed in several steps. First, hyperparameter settings are evaluated using validation loss and accuracy over 5-fold cross validation as the criterion. Then, the optimal model architectures are evaluated on the holdout test set according to the section below.

Precision and recall are important measures that will be used in the evaluation, expressed by equation as

$$Precision = \frac{True\ positives}{True\ positives + False\ positives} \tag{3.6}$$

$$Recall = \frac{True\ positives}{True\ positives + False\ negatives} \tag{3.7}$$

Precision and recall do not consider true negative predictions, thereby focusing on the ability of the model to represent the positive class (the minority class). This is particularly useful when dealing with an unbalanced dataset since we are indeed more interested in the accurate prediction of the anomalies, i.e. the minority class. In this project, the minority class are the companies that grow intensely in the prediction period p. Therefore, one could argue that, as decision support, the model should prioritize to have the intense growth predictions be accurate over catching all growth cases. The reason is that VC investors are short on time and want the intense growth predictions to be accurate rather than many. From a precision-recall trade-off perspective, this suggests that precision is prioritized over recall.

The evaluation is also done using the area under curve of the receiver operating characteristic curve (ROC-AUC) which is standard in binary classification problems. The higher the AUC, the better the classifier is at distinguishing between the classes. The ROC curve is a measure of the classification performance at varying thresholds, and plots the true positive rate (TPR) on the y-axis and false positive rate (FPR) on the x-axis, defined as:

$$TPR = \frac{True\ positives}{True\ positives + False\ negatives} \tag{3.8}$$

$$FPR = \frac{False\ positives}{False\ positives + True\ negatives} \tag{3.9}$$

3.3 LSTM model architecture

3.3.1 Architecture

The LSTM model architecture is based on stacked LSTM layers. When the output of the LSTM encoder is concatenated with T, a fully-connected dense layer is added to the end of the model to ensure that the desired dimensionality of the output is obtained, given the output dimensionality of the LSTM encoder concatenated with the additional feature T.

3.3.2 Regularization, optimization, loss and activation

Deep neural networks have a tendency to overfit to the training data, resulting in poor generalization capabilities. It is thus customary to accompany each LSTM layer with a dropout layer [43], reducing the model sensitivity to individual neuron weights. This method has been applied for all considered variations of the LSTM model. The optimal dropout rate is found through an iterative search for the best model hyperparameters (see section 4.1.1).

Regarding optimization, the adaptive moment estimation (ADAM) [44] is commonly used to compute the adaptive learning rate for each parameter in the LSTM network during the learning process. The algorithm has proven to perform better than similar optimizers, especially on high dimensional problems, motivating the use of ADAM as an optimizer for all LSTM model variations. The ADAM parameters used are $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-9}$. The capped learning rate for the ADAM optimizer is kept constant at $lr = 0.001$ through the entire training.

Furthermore, binary crossentropy loss is used as loss function during the training. Finally, a hyperbolic tangent activation function is used between the

output of a previous LSTM layer and the input of the next, while a sigmoid activation function is used in the recursion (as is the convention).

3.3.3 Hyperparameter search

As mentioned in section 3.2.4, the hyperparameters are found through 5-fold cross validation using a fixed 80% of the data (see section 3.2.3), and evaluated on the average validation loss and accuracy.

Because of computational constraints, an exhaustive grid search is replaced by a more basic approach where each hyperparameter is tuned in isolation while others are kept constant. Only a fixed set of values for each hyperparameter will be tested. When the optimal value for a hyperparameter is found, its new constant value for the succeeding searches (for optimal values of other hyperparameters) is the found optimal value for that hyperparameter.

3.4 Transformer model architecture

3.4.1 Architecture

Unlike the sequence-to-sequence architecture presented in figure 2.4, this project requires a sequence classification adaption. The architecture keeps the *attention blocks* intact (i.e. the N identical layers of the encoder) resulting in the Transformer encoder but swaps the decoder for an average pooling layer and a dense layer. The average pooling layer reduces the high dimensional encoder output (as a result of the number of heads in the multi-head attention mechanism) to a 2-dimensional vector which can then be passed through a dense layer and subsequently concatenated with T. The full model flow is the following: (1) The input X is first used to generate the Time2Vec vector representations of time defined in equation 13, i.e. the positional encodings; (2) T2V and X are concatenated and passed through the Transformer encoder; (3) the encoder output is passed through an average pooling layer followed by a dense layer with an output dimensionality of 8; (4) the output of the dense layer is concatenated with T and passed though a final dense layer resulting in the final output dimensionality. The adapted Transformer model used in this project is visualized in figure 3.5 (not including the concatenation step with T).

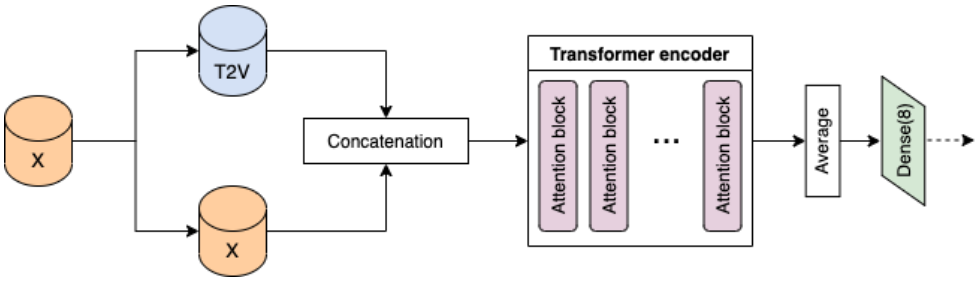


Figure 3.5: The adapted Transformer architecture first creates positional embeddings from X, T2V, which are then concatenated with X creating the encoder input. The encoder input is passed through the Transformer encoder (N identical attention blocks) followed by an average pooling layer and a dense layer of output dimensionality 8.

3.4.2 Regularization, optimization and loss

Much like the LSTM model, the Transformer is prone to overfitting on the training data. First, dropout [43] is applied after the attention mechanism as well as after the feed forward layer in each attention block. Additionally, dropout is applied between the average pooling layer and the dense layer prior to the concatenation step. The feed-forward layers in the attention body use the ReLu activation function. The choice of optimizer is ADAM [44], since it is widely used as optimizer in Transformer models. The capped learning rate for the ADAM optimizer is kept constant at $lr = 0.001$ through the entire training.

3.4.3 Hyperparameter search

The hyperparameter search method for the Transformer encoder is identical to that of the LSTM described in section 3.3.3, i.e. each hyperparameter is tuned in isolation while others are kept constant due to computational constraints.

Chapter 4

Results

The objective of this report is to compare the LSTM and Transformer encoders, and inherently their different ways of representing temporal input, in application to predicting intense employee growth in startup companies. Additionally, we compare their ability to do so using only time-dependent features as well as a combination of static and time-dependent features concatenated after the encoding step. Finally, we compare their general differences in training behaviour and performance.

A prerequisite for the comparative experiments is that the optimal model architectures for both encoders have been found prior. Thus, the results first present the considered model hyperparameters and the found optimal ones followed by a first comparison of the model training behaviours and performances. Then, we investigate their ability to model long-term dependencies with and without the influence of static features. The effect of varying p and the encoder type (LSTM or Transformer) on the model performance is statistically tested by means of two-way ANOVA tests, described in more detail in section 3.2.4.

4.1 Optimal model architectures

All experiments in this subsection were conducted using a prediction period $p = 6$ months and inputs X and T.

	r	Layers	u_1	u_2	u_3	Avg val acc	SD val acc	Avg val loss
L1	0.2	1	50			76.3%	2.2%	0.55
L2	0.2	2	50	50		80.7%	0.8%	0.40
L3	0.2	3	50	50	50	81.4%	0.6%	0.39
L4*	0.2	2	75	50	25	81.7%	0.4%	0.38
L5	0.2	3	25	50	75	80.6%	0.5%	0.40
L6	0.1	3	25	50	75	81.5%	0.3%	0.39
L7	0.3	3	25	50	75	81.5%	0.3%	0.38

Table 4.1: The average validation accuracy and loss and standard deviation of validation accuracies from a 5-fold cross validation, for varying hyperparameter settings of the LSTM encoder. We vary the dropout rate (r), the number of layers and the number of units in each layer (u_1, u_2 and u_3). The best model is denoted with '*'.

4.1.1 LSTM

Initially, the dropout rate r is kept constant, as is the number of units in each layer (u_1, u_2 and u_3). The aim is to obtain the optimal number of stacked LSTM layers by comparing the average validation accuracy and loss over a 5-fold cross validation run, each fold consisting of 30 epochs where the best model weights from each fold is used to compute the average. The resulting optimal number of layers is used to obtain the optimal values for u_1, u_2 and u_3. Finally, the dropout rate is varied using the best model obtained from the previous experiments. Table 4.1 shows that L4 is the optimal model architecture, visualized in 4.1. Notice how an increase in the number of stacked LSTM layers increased the validation accuracy while decreasing the standard deviation between folds, which could be interpreted as increasing that stability during training. We also observe that the undercomplete encoding structure of L4 resulted in the highest average validation accuracy, compared to the overcomplete encoding structure of L5.

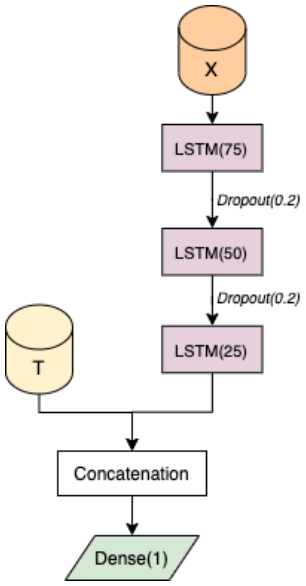


Figure 4.1: Model L4: X is passed through 3 stacked LSTM layers with 75, 50 and 25 units per layer respectively with dropout (with rate 0.2) between layers. The output of the final layer is concatenated with T and passed through a dense layer to generate the model output.

4.1.2 Transformer

Initially, the dropout rate r, the number of heads (Heads) and the feed-forward dimension (ff_dim) are kept constant. The aim is to obtain the optimal number of stacked attention blocks by comparing the average validation accuracy and loss over a 5-fold cross validation run, each fold consisting of 30 epochs where the best model weights from each fold is used to compute the average. The resulting optimal number of attention blocks is used to obtain the optimal number of heads in each blocks, and so on. Table 4.2 shows that T5 is the optimal model architecture, visualized in figure 4.2. Comparing the validation accuracy and loss for T1, T2 and T3, it is clear that a simple model is favoured, and that several attention blocks lead to very little to no learning. This is further confirmed by the fact that a lower feed-forward dimensionality is preferred (T5 compared to T1). We note that the standard deviation over folds is high in comparison to the LSTM models, and that for T5, some folds converge to validation accuracies around 78% while others barely learn.

	r	Blocks	Heads	ff_dim	Avg val acc	SD val acc	Avg val loss
T1	0.2	1	2	8	60.6%	11.2%	0.62
T2	0.2	2	2	8	59.9%	8.3%	0.64
T3	0.2	3	2	8	53.2%	4.1%	0.68
T4	0.2	1	4	8	57.6%	4.3%	0.67
T5*	0.2	1	2	4	63.2%	11.9%	0.59
T6	0.2	1	2	12	54%	4.3%	0.68
T7	0.1	1	2	4	58.9%	7.5%	0.65
T8	0.3	1	2	4	57.3%	5.0%	0.67

Table 4.2: The average validation accuracy and loss and standard deviation of validation accuracies from a 5-fold cross validation, for varying hyperparameter settings of the Transformer encoder. We vary the dropout rate (r), the number of attention blocks, the number of heads and the feed-forward dimension (ff_dim). The best model is denoted with '*'.

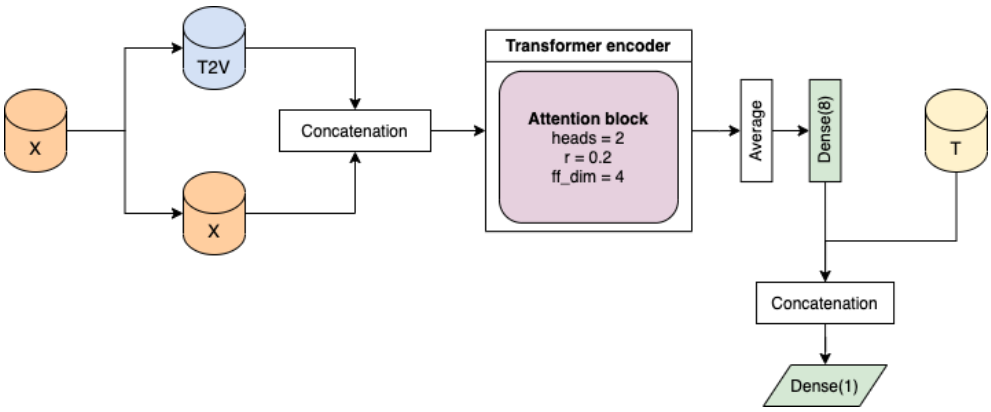


Figure 4.2: Model T5: X generates positional encodings T2V which are concatenated with X before being passed though the attention body consisting of a single attention block with 2 heads, a dropout rate (r) of 0.2 and a feed-forward dimension (ff_dim) of 4. The output of the Transformer encoder is passed through an average pooling layer followed by a dense layer with output dimensionality 8. Finally, the output of the dense layer is concatenated with T and passed through another dense layer to generate the model output.

4.1.3 Initial comparison of performance and stability

The first notable result is the general difference in performance between the LSTM and Transformer models, where we observe a substantially higher average performance level for the LSTM model compared to the Transformer

model (see tables 4.1 and 4.2). For example, the best Transformer model, T5, exhibited an average validation accuracy of around 63% while the worst LSTM model had an average validation accuracy of around 76%.

If we shift our focus from the average validation accuracy to the standard deviation of these we notice yet another considerable difference between the models. From table 4.2, column 'SD val acc', we can see that the Transformer exhibits high standard deviation of runs, indicating a clear instability in its training as some folds result in very low performances (almost no learning) while others converge to relatively high performing models (around 78% validation accuracy). The average standard deviation across the considered Transformer models (T1-T8) is 7.1. On the contrary, the LSTM models (L1-L7) have very low standard deviations as seen in table 4.1, column 'SD val acc', the average landing at 0.7. Essentially, this means that every L4 fold exhibits a very similar training pattern while T5 is rather unpredictable in the evolution of the training. Note that measures were taken to decrease the instability of the Transformer, such as the use of specific learning rate schedules aimed at ramping up and decaying the learning rate in varying fashions (the Adam optimizer adjusts each parameter's learning rate individually, but the maximum possible learning rate is still adjusted by a schedule). However, the stability did not improve.

4.2 Modelling long-term dependencies with time-dependent & static features

In the following experiments, we evaluate the ability of the LSTM and Transformer encoder to represent long-term dependencies using only time-dependent features as well as a combination of both time-dependent and static features. We do so by varying the prediction period p, i.e. the time period between the cutoff and prediction date. p takes the values $p = 6$, $p = 12$ and $p = 18$ months.

The results in this section consist of two distinct parts. For each subsection (4.2.1 and 4.2.2), we want to examine whether (1) p affects AUC, (2) encoder type has effect on the AUC and (3) the effect of p and encoder type on the AUC depends on the combination of these factors, i.e. there are interaction effects. To test the corresponding null hypotheses (1-3), a statistical test is performed in the form of a two-way ANOVA test, using previously seen data

(the 80% of the data which is initially used for hyperparameter tuning). The samples used for the analysis are obtained according to section 3.2.3. The statistical results are complemented by an evaluation of the models on the unseen test set. These results are obtained by training on the entire seen data set, using 20% for validation to monitor the training and enforce early stopping. The evaluation metrics used in the statistical tests as well as in the evaluation on the unseen test set are defined in section 3.2.4.

Models L4 and T5 pictured in figures 4.1 and 4.2 are representatives for their encoder types in the following experiments, having been evaluated as the strongest models in section 4.1. The LSTM models trained on labels corresponding to prediction periods $p = 6$, $p = 12$ and $p = 18$ using only time-dependent features are denoted as L4_6, L4_12 and L4_18. Similarly, the Transformer models trained only on time-dependent features are denoted T5_6, T5_12 and T5_18. All models that include static features are denoted with an additional 'S'. For example, L4S_6 refers to the LSTM models trained on $p = 6$ labels and both static and time-dependent features.

4.2.1 Time-dependent features

The inputs are X and T. In these experiments, we comparatively evaluate the ability of the Transformer and LSTM encoders to model long-term dependencies using solely time-dependent features. The experiment is conducted by varying the encoder type and the prediction period p between 6 and 18 months (thereby varying the prediction target), while keeping the input and rest of the model constant. The two-way ANOVA test presented in table 4.3 shows that the choice of encoder type significantly affects the performance in terms of AUC, i.e. that the two group means are statistically significantly different from each other ($p < 0.0001$). Hence, null hypothesis (2) can be rejected. Since there are only two categories of encoders, no post hoc analysis such as an Honest Significant Difference (HSD) test is necessary for this factor. It is sufficient to compare the mean AUC across all p for each encoder type - the estimated mean AUC of all L4 models was 0.89 and for T5 models - 0.56. Additionally, the test shows no interaction effects between p and the encoder type ($p > 0.8$) and, similarly, we have no statistical evidence to believe that p has any significant effect on the AUC performance ($p > 0.5$). The null hypotheses (1) and (3) cannot therefore be rejected. The distribution of the data used in the ANOVA test is plotted in figure 4.3 by p and encoder type.

	sum_sq	df	F	PR(>F)
p	4.5e-3	2	6.2e-1	5.4e-1
Encoder	1.6	1	435.2	1.6e-27
p:Encoder	1.3e-3	2	1.7e-1	8.4e-1
Error	2.0e-1	54		

Table 4.3: Result of two-way ANOVA test for models L4 and T5, investigating the effect of p and encoder type (LSTM or Transformer) on AUC, and the interaction effects between p and the encoder type. The sum of squares (sum_sq), i.e. the variance between the group means created by the levels (categories) of p and the overall mean, the F value (F), the degrees of freedom (df) and the p-value (PR(>F))are presented for p, the encoder type, their interaction (p:Encoder) and the error (the variation in AUC that can not be explained by either p or encoder type).

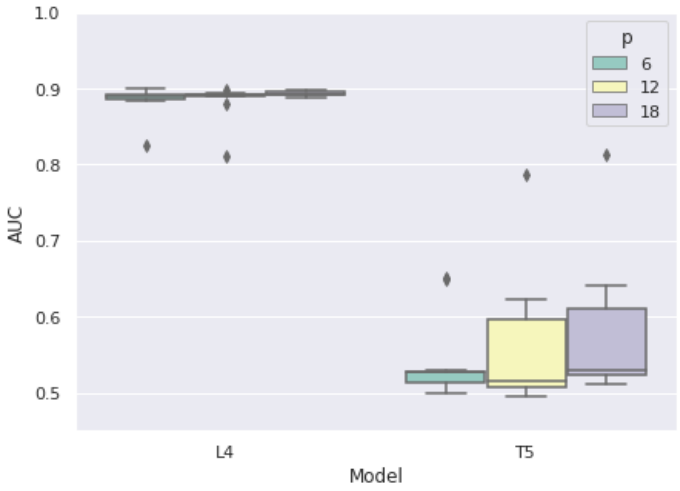


Figure 4.3: Distribution of data used for two-way ANOVA test, by p and encoder type (LSTM or Transformer) for models using only time-dependent features.

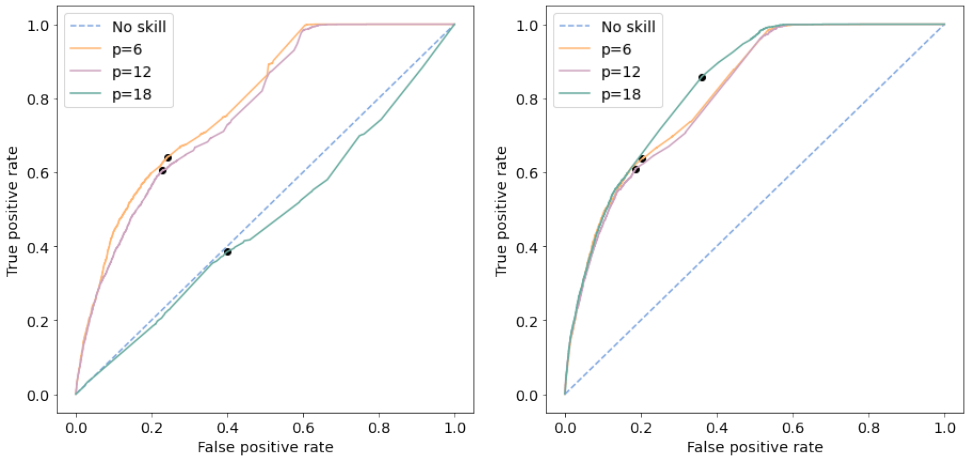
Considering the complementary results evaluated on the unseen test set, figure 4.4 shows the ROC curve for T5 (a) and L4 (b) for $p = 6$, $p = 12$ and $p = 18$, while figure 4.5 shows the precision-recall (PR) curve for the same models. In accordance with the evaluation strategy presented in section 3.2.4, the low-recall region is plotted in figure 4.6. The low-recall region (and thus the high-precision region) represents how the model would be used in a realistic setting, i.e. as VC decision support. All curves plot the no-skill

classifier for benchmark purposes.

Much like the results in the initial comparison in section 4.1.3, we find differences in training stability between the models both in the statistical tests and in the evaluation on the unseen test set. The variation among the T5 models across all p is substantial compared to the absence of variation among L4 models, seen in figure 4.3. In figure 4.4a we can see that T5_18 model does not learn and performs worse than the no-skill classifier, which confirms the unstable training. However, T5_6 and T5_12 achieve performances close to the L4 models. The promising ROC curves of T5_6 and T5_12 resulted in AUC values of around 0.76, thus not in line with the distribution of AUC for the T5 models in figure 4.3, i.e. the statistical data, and should therefore be regarded cautiously and not as fair representations of the general distribution. The cause of these unusual results could be a random sample of the validation set.

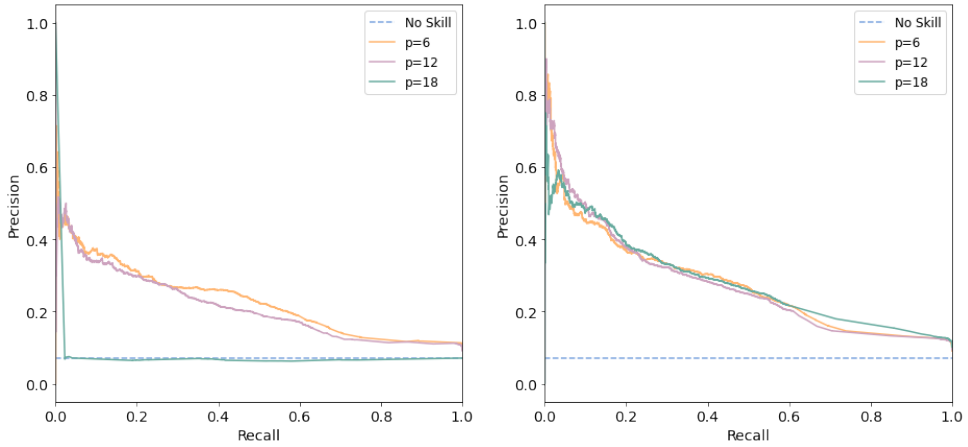
Furthermore, the minimal variations in AUC for the L4 models across all p in figure 4.3 are validated by the ROC curves in figure 4.4b, that also display very little effect of p on the ROC curve and consequently on the AUC.

The statistical evidence established that the LSTM models outperformed the Transformer models. These findings are confirmed by the evaluation on the unseen test set. If we compare the PR curves for L4 and T5 models on the full range (figure 4.5) we observe superior performances for the LSTM based models and once more notice that T5_18 mimics the no-skill curve. The low-recall regions visualized in figure 4.6 also confirm what the statistical tests determined, i.e. that the LSTM models outperform the corresponding Transformer models.



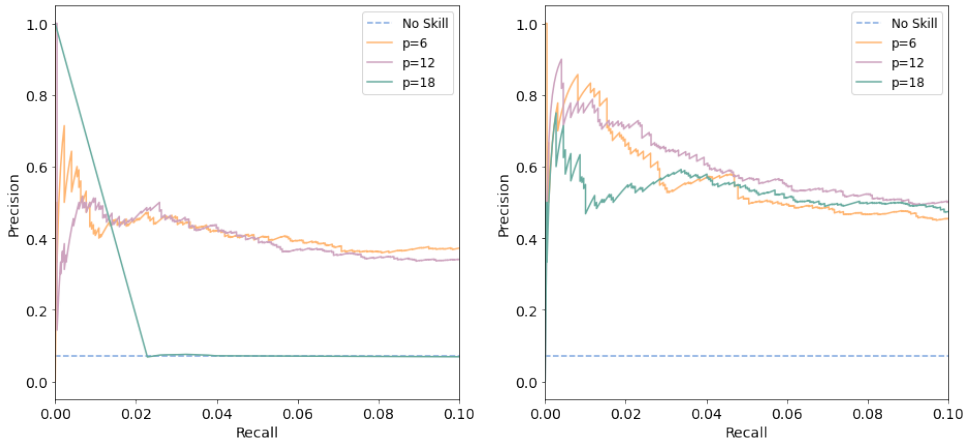
(a) T5 (b) L4

Figure 4.4: ROC curve for a) T5 and b) L4 for different p. The optimal threshold for each p is obtained by maximizing the Geometric Mean, and is marked with a black dot.



(a) T5 (b) L4

Figure 4.5: Precision recall curve for a) T5 and b) L4 for different p.



(a) T5 (b) L4

Figure 4.6: Precision recall curve for a) T5 and b) L4 for different p, in the low-recall region.

4.2.2 Time-dependent & static features

The inputs are X, T and A. In these experiments, we evaluate the ability of the Transformer and LSTM encoders to model long-term dependencies in time-dependent features when the sequence encodings are concatenated with the encodings of static features. To make a comparative analysis of the encoders and study the effects of static features, the static features are added as input while performing otherwise identical experiments as in section 4.2.1. The larger model used in these experiments is visualized in figure 3.2.

The two-way ANOVA test presented in table 4.4 shows that the encoder type significantly affects AUC ($p < 0.0001$) and hence that null hypothesis (2) can be rejected. Much like in section 4.2.1, we can determine which encoder results in the better performance by investigating the mean AUC across all p for each encoder type. The L4S models had an average AUC of 0.88 while the T5S models had an average AUC of 0.76, meaning we can determine that the LSTM encoder performs better than the Transformer encoder. The test shows no interaction effects between p and encoder type ($p > 0.2$) and that there is no statistical evidence to believe that p has any significant effect on AUC ($p > 0.1$), meaning null hypotheses (1) and (3) cannot be rejected as a result. Figure 4.7 shows the distribution of the data used in the ANOVA test, by p and encoder type.

	sum_sq	df	F	PR(>F)
p	2.1e-3	2	1.7	1.9e-1
Encoder	2.4e-1	1	383.0	3.5e-26
p:Encoder	1.7e-3	2	1.4	2.5e-1
Error	3.3e-2	54		

Table 4.4: Result of two-way ANOVA test for models L4S and T5S, investigating the effect of p and encoder type (LSTM or Transformer) on AUC, and the interaction effects between p and the encoder type. The sum of squares (sum_sq), i.e. the variance between the group means created by the levels (categories) of p and the overall mean, the F value (F), the degrees of freedom (df) and the p-value (PR(>F))are presented for p, the encoder type, their interaction (p:Encoder) and the error (the variation in AUC that can not be explained by either p or encoder type).

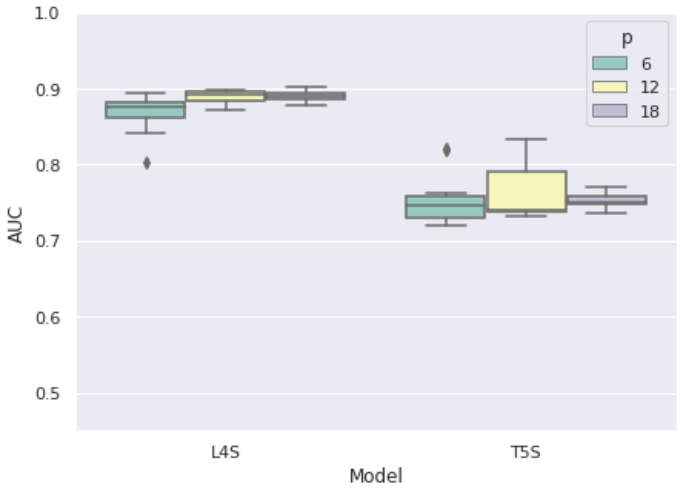


Figure 4.7: Distribution of data used for two-way ANOVA test, by p and encoder type (LSTM or Transformer) for models using time-dependent and static features.

Looking at the distribution of the data used in this two-way ANOVA test in figure 4.7, we can make two main observations: the variation of AUC for all L4S models seems to have increased compared to the L4 models in figure 4.3 and, more importantly, all T5S models have drastically increased their performance (from around 0.5-0.6 to around 0.75 AUC), while generally decreasing their variance. The increased stability among the Transformer models is confirmed by the evaluation on the unseen test set, where we notice that T5S_18, unlike T5_18, actually learned. While the ROC curve for T5_18

(figure 4.4 a)) mimicked the no-skill classifier, the ROC curve for T5S_18 (figure 4.8 a)) instead looks similar to the other T5S models. To determine if there is statistical evidence to believe that T5 is different from T5S and L4 is different from L4S, we perform another one-way ANOVA test which is presented in table 4.5. From the table we can conclude that there is a significant difference amongst the models ($p < 0.0001$), which is expected since we have already proved that L4 performs better than T5 and L4S performs better than T5S. With a post hoc analysis in the form of a Tukey HSD, we can evaluate where the difference lies, where the intention is to test if T5 is significantly different from T5S and L4 is significantly different from L4S. The results of the Tukey HSD are presented in table 4.6. The test shows that T5 and T5S are indeed different, and by investigating their group means (0.56 and 0.76 respectively), we can determine that T5S performs better than T5. Additionally, the post hoc test shows that there is no statistical evidence to believe that L4S and L4 are different in terms of performance.

	sum_sq	**df**	**F**	**PR(>F)**
Model	2.1	3	338.1	3.7e-57
Error	0.2	116		

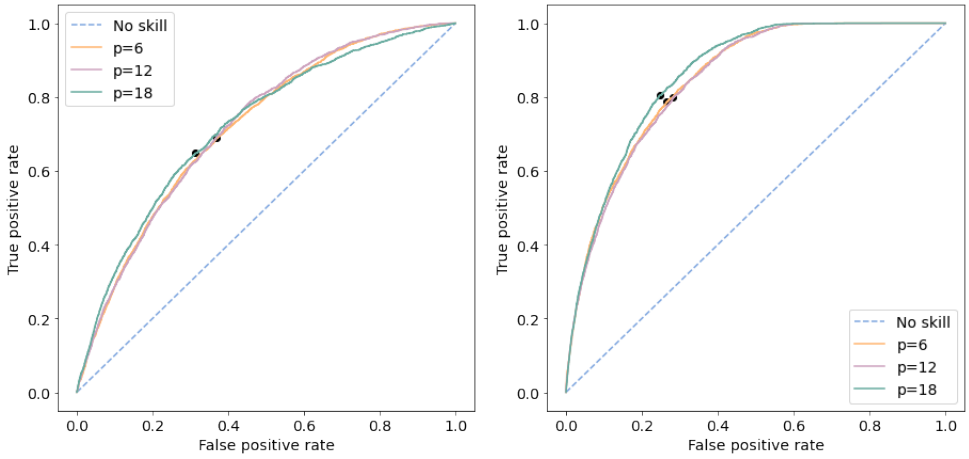
Table 4.5: Result of one-way ANOVA test for models L4, T5, L4S and T5S, investigating the effect of the model on AUC. The sum of squares (sum_sq), i.e. the variance between the group means and the overall mean, the F value (F), the degrees of freedom (df) and the p-value (PR(>F)) are presented for the 'Model' factor. The error is the variation in AUC that can not be explained by the model).

group_1	**group_2**	**mean_diff**	**p-adj**	**lower**	**upper**	**reject**
L4	L4S	-4.8e-3	0.9	-3.6e-2	-2.6e-2	False
T5	T5S	2.0e-1	1.0e-3	1.7e-1	2.3e-1	True

Table 4.6: Post hoc analysis (Tukey HSD) of the ANOVA test in table 4.5 to evaluate where the difference between models lie. The mean_diff is the difference between the group means and p-adj is the corrected p-value. The lower and upper bands of the confidence interval are presented as well as a conclusion whether to reject the null hypothesis that there is no statistically significant difference between the groups.

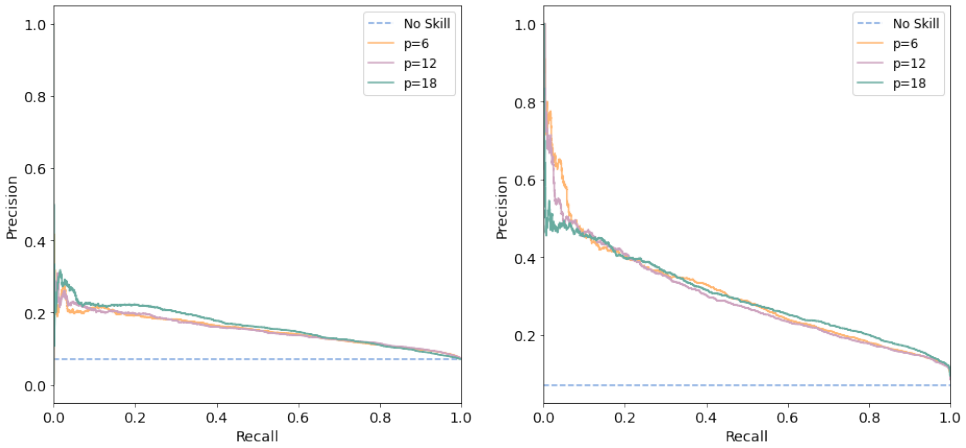
Considering the increased performance of the Transformer models and the unchanged performance of the LSTM models that table 4.6 shows, one would

expect a decreased gap in performance between the L4S and T5S models compared to the gap between L4 and T5 models. However, the evaluation on the unseen test set contradicts this expectation, where we see that the gap in performance has increased notably across all p. For example, the ROC curves in figure 4.8 show a flattening of the curves for T5S models, while curves for L4S models are stretching further toward the top left corner than observed for L4 models. The increased gap is also apparent in the PR curves in figures 4.9 and 4.10, where T5S displays lower precision values in the full recall range compared to both T5 but also L4S. However, as discussed in section 4.2.1, it is clear that this unexpected increased gap is a result of a very rare performance outcome for T5 that does not represent the distribution seen in figure 4.3 accurately. As previously mentioned, the promising results of T5_6 and T5_12 when evaluated on the unseen test set can be a result of a random sample of the validation set.



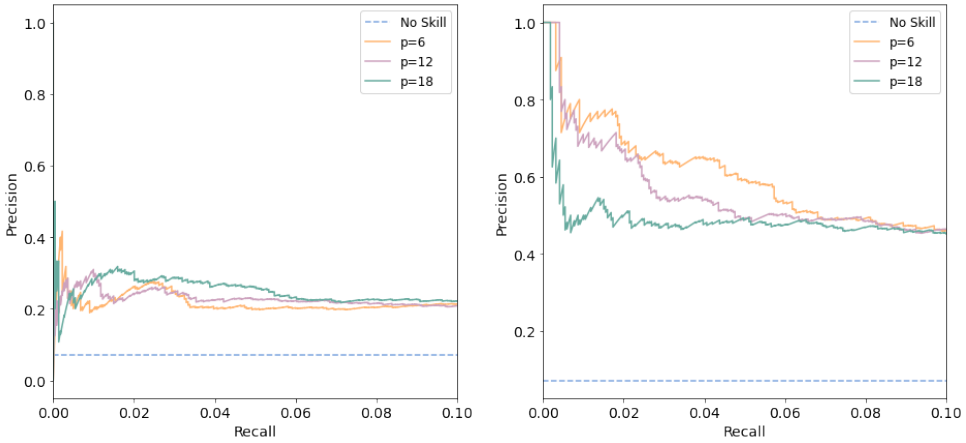
(a) T5S (b) L4S

Figure 4.8: ROC curve for a) T5S and b) L4S for different p. The optimal threshold (used to calculate AUC) for each p is obtained by maximizing the Geometric Mean, and is marked with a black dot.



(a) T5S (b) L4S

Figure 4.9: Precision recall curve for a) T5S and b) L4S for different p.



(a) T5S (b) L4S

Figure 4.10: Precision recall curve for a) T5S and b) L4S for different p, in the low-recall region.

Chapter 5

Discussion

5.1 Implications

In regards to using machine learning for startup success prediction, the notable gaps in the literature included both a lack of deep learning models [1,3–5,8,13] and the absence of models that account for the many time-dependent features relevant to assessing companies in VC [1–8, 13]. Thus, one implication of this project is the promising results obtained when filling these gaps. The statistical tests showed that the LSTM models reached AUC values of around 0.9 (see figure 4.3 and 4.7), which is comparable with, and sometimes better than, the results of previous studies using shallow models [3, 4, 7, 8, 25]. Considering the restricted time period of this project and therefore the narrow selection of model architectures, hyperparameter settings and feature sets that could be evaluated, it is likely that the results could be further improved. In short, this project provides evidence that deep learning models are successful in this application area, and that enabling the use of time-dependent features through models that cater to them is indeed profitable.

Another important outcome of this project is the demonstrated ability of the LSTM encoder and inherently its temporal representation to model long-term dependencies. The literature suggested that the Transformer would in theory be better suited for this task [16, 41]. Although a fair comparison could not be made due to the unstable nature of the Transformer, we can at least determine that the LSTM was successful in representing long-term dependencies (assuming that the dataset has them and that they are increasingly important for the forecast as the prediction period is extended). Indeed, the statistical tests showed that p did not have a statistically significant

effect on AUC. Considering the restricted length of the input sequence for each sample (42 time-steps), the LSTM could predict as far as 18 time-steps into the future without falling below an AUC of around 0.9.

Furthermore, the expected impact of increasing p was a decrease in performance and, as discussed, this was not the case for AUC. However, the classification threshold for each p is usually obtained by maximizing the Geometric Mean, which does not favour precision over recall as we do in the low-recall regions. The low-recall region was deemed interesting as it represents how an employee growth prediction would be used in a realistic venture capital setting. Other methods for obtaining the classification threshold that favor precision might therefore have been interesting to investigate from the perspective of the application area.

Moreover, the results showed that the use of static features did not affect the LSTM models in terms of performance. We therefore speculate that the static feature set can be further optimized, and that a more optimal set could even increase the performance when added to the model.

5.2 Instability of the Transformer

With regards to the lacking training stability that was observed for the Transformer models in sections 4.1 and 4.2, there are many possible explanations to this behaviour. There is, to the best of my knowledge, no reference dataset for which this adapted Transformer's performance is known, meaning we can not test the accuracy of the implementation fully. However, we can most likely rule out the possibility of an implementation error, since the attention block and T2V components are well documented online and this project follows the implementation convention of these. As for the more probable causes, we acknowledge that the Transformer was initially intended for NLP tasks [16], and that much of its success has come from tasks of similar nature, i.e. where the inputs are samples from a finite set of values as they are in NLP, image processing and so on [14–16, 18, 19, 40]. It is possible that the multi-head attention mechanism, residual connections and other structural conditions of the Transformer are worse suited for scalar inputs, and that this factor had a substantial impact on training stability. Furthermore, some literature shows that smaller datasets can have undesirable effects on the performance on the Transformer [19]. Since the dataset used in this project

is substantially smaller than the one used in e.g. [16], we speculate that the restricted amount of data in this project could also have affected training stability to some extent. We also recognize that the data is both noisy and has estimated values for some time-steps, which is never ideal but could be even more detrimental for the Transformer than for the LSTM models, especially in combination with the inputs being scalar. Considering the above, we raise the following questions: for noisy scalar inputs, is the unlimited memory and bidirectional properties of the Transformer no longer its best attributes, but its biggest downfall? Given that the context vector indeed has access to the entire input sequence, and the fact that scalar input are essentially more similar to each other than e.g. word embeddings in NLP tasks, can the Transformer really distinguish noise from a real signal in this setting? How can we then ensure that noise in the input signal does not dominate or overwhelm the hidden representation? Perhaps the recursive nature of the LSTM encoder (and the constant warping of the memory) allows the representation to suppress the noise, while the short path between input and target in the Transformer encoder makes it impossible to ignore.

The results of section 4.2.2 showed a significant increase in performance and stability for the Transformer models when using both time-dependent and static features, while the LSTM model was affected negatively to a small extent. One possible explanation is that the Transformer encoder is better suited to be incorporated into a larger model than the LSTM encoder due to its higher dimensional context vector, which in actuality gives the Transformer encoder output relatively more space in the concatenating layer compared to the LSTM encoder output. However, the unstable nature of the Transformer presented in section 4.2.1 diminishes the likelihood of this being the cause of increased performance. A more likely explanation is that the static features are 'carrying' the prediction and that the Transformer encoder does not contribute to the performance, or does so very little. Many related works used solely static features for their success predictions [1–8, 13], so their predictive power does not come as a surprise.

5.3 Evaluation

The evaluation strategy of this project is multifaceted. ROC curves, AUC and the precision recall curves are widely used evaluation metrics in binary classification task and especially in the presence of an unbalanced dataset. Simultaneously, the low-recall region evaluates the model performance in a

realistic setting, taking into account the application area and its specific needs. However, the evaluation is flawed by the lack of a reference dataset in the field, as discussed in section 1.2. In other fields such as image processing and NLP, many reference datasets are available and as a result the related works can be properly compared and the best methods can more easily be established. In that sense, it is difficult to conclude if the models used in this project are superior to the ones used in previous works, and thus if deep learning models are better than the shallow machine learning models in the related work [1, 3–5, 8, 13].

5.4 Ethics and sustainability

VC funds have, like many other organizations, long suffered from investment bias. For example, many studies have shown that women are generally less likely to receive funding and likely to receive less funding than men with equal backgrounds [45–47]. The investment bias is not only affecting the entrepreneurs negatively when they are denied funding, but could also have long term effects on the success of the fund since it can prohibit them from considering profitable investments. A data driven decision support tool like the one in this project generally inherits the human bias by modelling the real world, and therefore acts under the same prejudice. To limit the bias to some extent, this project has intentionally excluded features describing the founders. Thus, factors such as gender, age and ethnicity of the founders have no predictive power in these models. However, features such as those containing historical funding round information are likely to be affected by investment bias anyway, since those investment decisions were probably influenced by the bias of the relevant investors. In short, it is difficult to escape investment bias if the data used in the prediction is in itself skewed. Fortunately, the company (EQT Ventures) has methods in place allowing them to counterbalance that bias: an algorithmic recommendation system that helps investors screen more companies with founders from underrepresented groups. The recommendations frequently lead to successful investments and, in turn, these investments will decrease the bias inherent in the data.

Chapter 6

Conclusions and future work

This thesis has examined how the temporal data representations of the LSTM and the Transformer compare in their ability to capture long-term dependencies with and without the influence of static features when predicting employee growth for startup companies. We found that the LSTM exhibits more stable training patterns, performs better and is very successful in representing long-term dependencies (if such are present in the data), both with and without the influence of static features. In fact, the LSTM reached remarkably high performances (AUC 0.9) for prediction periods almost half the size of the input sequence. The Transformer suffered from instability, and the question was raised whether its unlimited memory and bidirectional properties that makes it so successful in, e.g., NLP tasks are rather disadvantages when dealing with noisy scalar input sequences. When it comes to comparing the LSTM and Transformer and inherently their different temporal representations, it is clear that their performance is highly affected by the type of input data and its condition, and that more research has to be made regarding their response to varying conditions, why and how these responses are triggered and new methods to account for undesirable effects.

6.1 Future work

6.1.1 Optimization and interpretability

From a practical point of view, there are many aspects of the models that could be further optimized to reach better performance. For example, one could alter the feature engineering process, undergo a more extensive hyperparameter search or iterate on the feature set. These alterations would expand the

study and provide insights into how the models are affected by the different conditions. Varying the feature set could also increase interpretability by understanding what features have predictive power.

6.1.2 Positional encoding

In terms of the positional encoding used in the Transformer models, only one method was considered in this project. Since these encoding are essential in the models ability to represent time, it is possible that the choice of encoding affects the model performance, and that the type of input data considerably affects the optimal temporal encoding. Future work investigating various positional encodings for different input data types would therefore contribute to the research and best practices regarding the use of Transformers across various tasks.

6.1.3 Understanding the Transformer

The discussion raised several questions regarding the properties of the Transformer and how they possibly contribute to its sensitivity to data quality and data type. Although the Transformer is very successful in some tasks (like NLP), research regarding its weaknesses and fragilities in other situations would not only increase its chances of being successful in other domains, but could also result in generally improved alterations.

References

[1] J. Arroyo, F. Corea, G. Jimenez-Diaz, and J. A. Recio-Garcia, "Assessment of machine learning performance for decision support in venture capital investments," *Ieee Access*, vol. 7, pp. 124 233–124 243, 2019.

[2] G. Xiang, Z. Zheng, M. Wen, J. I. Hong, C. P. Rosé, and C. Liu, "A supervised approach to predict company acquisition with factual and topic features using profiles and news articles on techcrunch." in *ICWSM*, 2012.

[3] F. R. d. S. R. Bento, "Predicting start-up success with machine learning," Ph.D. dissertation, 2018.

[4] C. Ünal and I. Ceasu, "A machine learning approach towards startup success prediction," IRTG 1792 Discussion Paper, Tech. Rep., 2019.

[5] M. Ghassemi, C. Song, and T. Alhanai, "The automated venture capitalist: Data and methods to predict the fate of startup ventures," in *AAAI KDF Workshop*, 2020.

[6] C. Gastaud, T. Carniel, and J.-M. Dalle, "The varying importance of extrinsic factors in the success of startup fundraising: competition at early-stage and networks at growth-stage," *arXiv preprint arXiv:1906.03210*, 2019.

[7] B. Sharchilev, M. Roizner, A. Rumyantsev, D. Ozornin, P. Serdyukov, and M. de Rijke, "Web-based startup success prediction," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, pp. 2283–2291.

[8] A. Krishna, A. Agrawal, and A. Choudhary, "Predicting the outcome of startups: less failure, more success," in *2016 IEEE 16th International*

Conference on Data Mining Workshops (ICDMW). IEEE, 2016, pp. 798–805.

[9] J. Arroyo, F. Corea, G. Jimenez-Diaz, and J. A. Recio-Garcia, "Assessment of machine learning performance for decision support in venture capital investments," *Ieee Access*, vol. 7, pp. 124 233–124 243, 2019.

[10] A. Davila, G. Foster, and M. Gupta, "Venture capital financing and the growth of startup firms," *Journal of business venturing*, vol. 18, no. 6, pp. 689–708, 2003.

[11] M. Pagano, F. Panetta, and L. Zingales, "Why do companies go public? - an empirical analysis," 04 1999.

[12] S.-G. Stanescu, A. Danila, and M.-G. Horga, "Econometric model necessary for analysis of existing correlations between human resources and the financial performance of the enterprises," *Journal of Science and Arts*, vol. 18, no. 1, pp. 159–166, 2018.

[13] R. N. Lussier and S. Pfeifer, "A crossnational prediction model for business success," *Journal of small business management*, vol. 39, no. 3, pp. 228–239, 2001.

[14] D. Merkx and S. L. Frank, "Comparing transformers and rnns on predicting human sentence processing data," *arXiv preprint arXiv:2005.09471*, 2020.

[15] S. Karita, N. Chen, T. Hayashi, T. Hori, H. Inaguma, Z. Jiang, M. Someki, N. E. Y. Soplin, R. Yamamoto, X. Wang *et al.*, "A comparative study on transformer vs rnn in speech applications," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019, pp. 449–456.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[17] L. Liu, X. Liu, J. Gao, W. Chen, and J. Han, "Understanding the difficulty of training transformers," *arXiv preprint arXiv:2004.08249*, 2020.

[18] B. Zhang, I. Titov, and R. Sennrich, "Improving deep transformer with depth-scaled initialization and merged attention," *arXiv preprint arXiv:1908.11365*, 2019.

[19] A. Ezen-Can, "A comparison of lstm and bert for small corpus," *arXiv preprint arXiv:2009.05451*, 2020.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[21] P. Stromberg, S. Kaplan, B. Sensoy, and P. Strömberg, "Should investors bet on the jockey or the horse? evidence from the evolution of firms from early business plans to public companies," *Journal of Finance*, vol. 64, pp. 75–115, 02 2009. doi: 10.2139/ssrn.657721

[22] A. Prohorovs, J. Bistrova, and D. Ten, "Startup success factors in the capital attraction stage: Founders' perspective," *Journal of East-West Business*, vol. 25, no. 1, pp. 26–51, 2019.

[23] M. Da Rin, T. Hellmann, and M. Puri, "A survey of venture capital research," in *Handbook of the Economics of Finance*. Elsevier, 2013, vol. 2, pp. 573–648.

[24] R. Nanda and M. Rhodes-Kropf, "Investment cycles and startup innovation," *Journal of Financial Economics*, vol. 110, no. 2, pp. 403–418, 2013.

[25] G. Xiang, Z. Zheng, M. Wen, J. Hong, C. Rose, and C. Liu, "A supervised approach to predict company acquisition with factual and topic features using profiles and news articles on techcrunch," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 6, no. 1, 2012.

[26] M. Böhm, J. Weking, F. Fortunat, S. Müller, I. Welpe, and H. Krcmar, "The business model dna: Towards an approach for predicting business model success," 2017.

[27] O. Gassmann, K. Frankenberger, and M. Csik, *The business model navigator: 55 models that will revolutionise your business*. Pearson UK, 2014.

[28] S. Siami-Namini, N. Tavakoli, and A. S. Namin, "A comparison of arima and lstm in forecasting time series," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 1394–1401.

[29] B. Fernandes, F. Silva, H. Alaiz-Moretón, P. Novais, C. Analide, and J. Neves, "Traffic flow forecasting on data-scarce environments using arima and lstm networks," in *World Conference on Information Systems and Technologies*. Springer, 2019, pp. 273–282.

[30] S. Siami-Namini, N. Tavakoli, and A. S. Namin, "A comparative analysis of forecasting financial time series using arima, lstm, and bilstm," *arXiv preprint arXiv:1911.09512*, 2019.

[31] D. Janardhanan and E. Barrett, "Cpu workload forecasting of machines in data centers using lstm recurrent neural networks and arima models," in *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE, 2017, pp. 55–60.

[32] A. Azari, P. Papapetrou, S. Denic, and G. Peters, "Cellular traffic prediction and classification: A comparative evaluation of lstm and arima," in *International Conference on Discovery Science*. Springer, 2019, pp. 129–144.

[33] H. Yan and H. Ouyang, "Financial time series prediction based on deep learning," *Wireless Personal Communications*, vol. 102, no. 2, pp. 683–700, 2018.

[34] Y. Bengio, *Learning deep architectures for AI*. Now Publishers Inc, 2009.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[36] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," *arXiv preprint arXiv:1312.6026*, 2013.

[37] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," *Advances in neural information processing systems*, vol. 26, pp. 190–198, 2013.

[38] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. Ieee, 2013, pp. 6645–6649.

[39] S. M. Kazemi, R. Goel, S. Eghbali, J. Ramanan, J. Sahota, S. Thakur, S. Wu, C. Smyth, P. Poupart, and M. Brubaker, "Time2vec: Learning a vector representation of time," *arXiv preprint arXiv:1907.05321*, 2019.

[40] A. Zeyer, P. Bahar, K. Irie, R. Schlüter, and H. Ney, "A comparison of transformer and lstm encoder decoder models for asr," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019, pp. 8–15.

[41] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[42] "tf2-preview/nnlm-en-dim50," https://tfhub.dev/google/tf2-preview/nnlm-en-dim50/1, accessed: 2020-03-15.

[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[45] M. Malmström, A. Voitkane, J. Johansson, and J. Wincent, "What do they think and what do they say? gender bias, entrepreneurial attitude in writing and venture capitalists' funding decisions," *Journal of Business Venturing Insights*, vol. 13, p. e00154, 2020.

[46] J. E. Tinkler, K. B. Whittington, M. C. Ku, and A. R. Davies, "Gender and venture capital decision-making: The effects of technical background and social capital on entrepreneurial evaluations," *Social Science Research*, vol. 51, pp. 1–16, 2015.

[47] M. Malmström, J. Johansson, and J. Wincent, "Gender stereotypes and venture support decisions: how governmental venture capitalists socially construct entrepreneurs' potential," *Entrepreneurship Theory and Practice*, vol. 41, no. 5, pp. 833–860, 2017.

TRITA -EECS-EX-2021:300