Deep Echo State Network-Based Information Extractor for Financial Documents

Thiago Bell Felix de Oliveira

A thesis submitted in partial fulfillment for the
degree of Master of Science

in the

Faculty of Computer Science
Rheinische Friedrich-Wilhelms-Universität Bonn

Advisor:
Dr. Rafet Sifa
Supervisors:
Prof. Dr. Christian Bauckhage and
Prof. Dr. Stefan Wrobel

May 2021

Contents

Chapter 1

Introduction

Companies collect large volumes of data every day in the form of text documents. Documents created by different departments on topics such as market research, business disclosures, and contracts contain potentially meaningful information for data-driven decision-making. However, this knowledge is diffuse, and an archive with large volumes of documents is of limited use if the information contained in them is not easily accessible. When it is, it may feed internal search engines, databases, and business intelligence systems that allow organizations to better understand their business and identify opportunities. Therefore, solutions that can extract structured information from unstructured documents are needed to support such systems.

In natural language processing (NLP), information extraction concerns itself with retrieving information in a structured manner from raw text. Among its main formulations is the named entity recognition (NER) problem. It defines the concept of an entity which is a sequence of words corresponding to a semantic object in the text. Furthermore, each entity has an associated class such as name or address. The NER task concerns itself with detecting and classifying entities of different classes from the input text.

Traditional NER systems would use custom rule-based methods. Their rules are created by experts and employed to solve a task from a particular domain. Although these systems can effectively solve the NER task for diverse applications, they require expensive expert intervention in the conception of the rules and are domain exclusive [1]. Machine learning-based systems are also used to solve the NER task but feature engineering is a challenge. Features for words can be, for example, derived from morphological analysis (e.g. if a prefix is present) or if the word is present on a look-up table. These features may be sufficient for solving tasks successfully. For example, on a biomedical dataset the suffix "-in" may be a useful feature for detecting mentions of proteins (e.g. hemoglobin and keratin). However, as with rule-based systems, feature selection requires manual involvement from experts, and solutions may not be easily portable to other domains.

In deep learning, neural networks are used to generate features automatically from inputs. A prominent example of that is the advances in computer vision with convolutional neural networks. In these approaches, images are processed directly by the network using their individual pixel values. Moreover, the network learns the mapping of these pixel values to more abstract image features that are combined and used to generate additional features of increasing complexity. Language models follow a similar approach for the problem of processing text with neural networks.

Instead of using manual features, these networks learn to represent text in numerical vectors autonomously. Deep learning models require large amounts of training data due to their capacity. However, the employment of transfer learning helps to placate these concerns, and language models may be trained with a different larger dataset from the one used in the downstream task.

Supervised machine learning approaches require the selection of one or more models to be trained for the task. Echo state networks (ESN) models are of a type of recurrent neural network (RNN) that is easier to train when compared with their traditional counterparts. They contain a reservoir and a readout linear layer. The reservoir is a collection of neurons with a randomized topology and weights. It has a state which is constituted by the activations of the neurons it contains. As the sequence of inputs is provided to the reservoir, the states of the neurons change. The hyperparameters of the network govern the dynamics of the reservoir allowing for tuning of the reservoir to different datasets. The states of a network are used by the readout layer, which is a linear predictor, to produce the outputs of the network. The readout layer is trained by solving a set of linear equations with methods such as least squares. Therefore, the cost of training is considerably lower than traditional recurrent neural networks using gradient-based methods. Furthermore, a characteristic of RNNs is their memory. A reservoir is capable of representing a summary of previous inputs when certain conditions are met. This is called the echo state property. When it is fulfilled, these reservoirs can be used to generate contextual representations of the inputs for tasks that rely on temporal information.

Deep echo state networks are extensions to the ESNs that add depth to the reservoirs. Instead of a single reservoir, the network contains multiple. They remain recurrent but are stacked to form a layered architecture analogous to traditional feed-forward neural networks. In this manner, the outputs of each reservoir serve as inputs to the next layer. Finally, the output of the *deep* reservoir is formed by the concatenation of the states from each reservoir layer. Furthermore, these networks can represent information at different time scales. Then, the network may learn to distinguish components of the input signal occurring at different frequencies. This is relevant for tasks such as NLP since the meaning of the text may easily be affected by a previous word as well as a paragraph.

Language models can be used to provide automatic feature generation for machine learning-based architectures for the NER task. Furthermore, they provide a reusable solution for features generation which is also domain independent. This contrasts with the custom feature selection of other approaches. ESNs also have the potential of being used for sequence learning tasks, a category in which NER is included. ESNs have been successfully applied to the NER task [2] in combination with language models. In that work, the authors used textual embeddings as inputs to an ESN trained on the CoNLL03 [3] dataset.

In this work, we seek to extend the authors' contributions and examine the NER task applied to the extraction of key pieces of information from financial documents. We investigate if (deep) echo state networks may be used in a different domain with *limited* training data. Furthermore, we are also interested in analyzing the effect of depth in ESNs reservoirs. With this work, we seek to contribute to the search for novel approaches to NER and information extraction tasks that are both adaptable and have low cost, particularly towards enterprise solutions.

Chapter 2

Technical Background

In this chapter, we explain the most important methods and technologies used in our work. We start by providing a brief overview of feed-forward neural networks in Section 2.1. We discuss recurrent neural networks in general in Section 2.2 followed by a description of (deep) echo state networks in Section 2.3. We also define language models and the advantages of their use for automatic feature generation in Section 2.4 and shortly introduce the named entity recognition problem in Section 2.5. We conclude this chapter by briefly defining conditional random field models in Section 2.6.

2.1 Neural Networks

(Artificial) neural networks are machine learning models inspired by the brain. They are composed of neurons which are computational units capable of processing inputs and outputting a result. Although neurons are individually simple when in greater numbers they are capable of executing complex tasks. Neural networks have been successfully applied to a myriad of problems such as fraud detection and handwritten text recognition. Furthermore, advances in computing power in the last decades have enabled neural network research in more difficult tasks. This progress was accompanied by neural networks and datasets of increasing size and complexity.

2.1.1 Neurons and Activation Functions

A neuron is the simplest computational unit in a neural network. It takes an input vector $\boldsymbol{i}$ and applies a linear combination to it, effectively implementing a linear function. As in these functions, inputs have coefficients modeled by a weight vector $\boldsymbol{w}$. Furthermore, linear functions may have constants that are implemented by a bias input value of 1 and a corresponding real weight. For conciseness both are included in the $\boldsymbol{i}$ and $\boldsymbol{w}$ vectors. In this manner, we can represent any linear function with the dot product $\boldsymbol{i} \cdot \boldsymbol{w}$.

Representing linear functions is useful but not enough to solve more complex problems. Furthermore, combining neurons with linear functions is equivalent to having a single neuron with a linear function. For this reason, neurons may have activation functions that add non-linearity to their outputs. This means that neurons are capable of implementing non-linear functions and

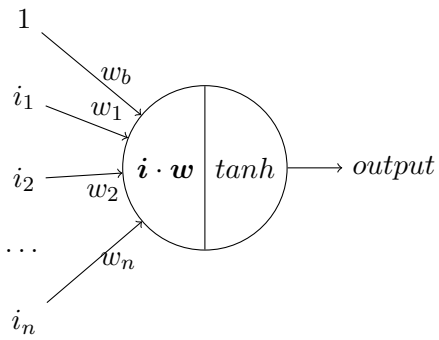
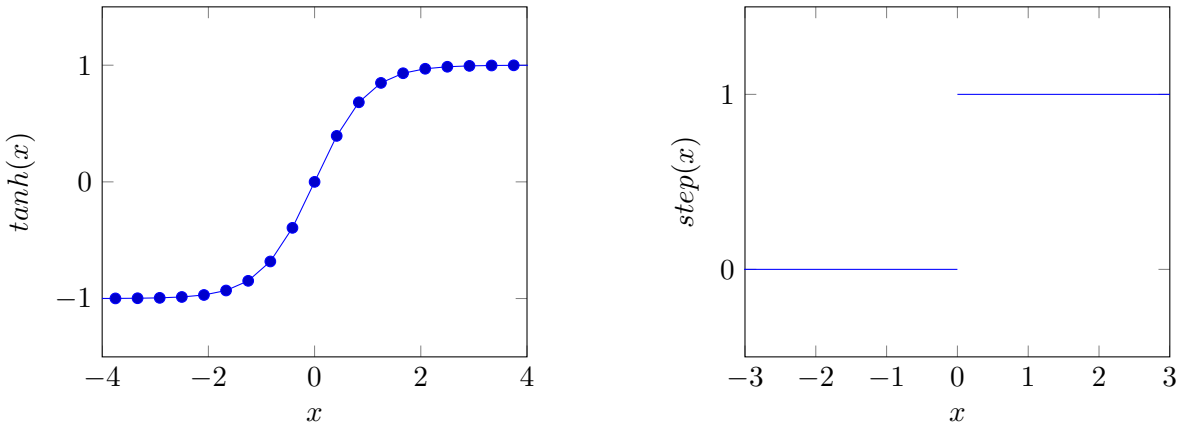


Figure 2.1: An illustrated neuron. i_1 to i_n are its inputs, and w_1 to w_n their corresponding weights.



(a) tanh

(b) step

Figure 2.2: Examples of activation functions.

solve more complex problems. In this work, we use the hyperbolic tangent ($tanh$) as defined in Equation 2.1 and shown on Figure 2.2a.

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.1}$$

A historical example of an activation function is the step function shown in Figure 2.2b. This allows a neuron to implement a binary predictor. If the result of the linear combination is positive it outputs 1, otherwise, 0. An example of such a neuron with two-dimensional inputs is illustrated in Figure 2.3.

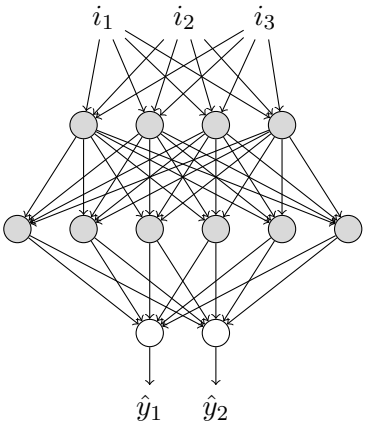


Figure 2.4: A neural network with two hidden layers. Neurons are represented as nodes, connections as edges. Layers with gray neurons are hidden. i_* are the inputs, and $\hat{y}*$ the outputs

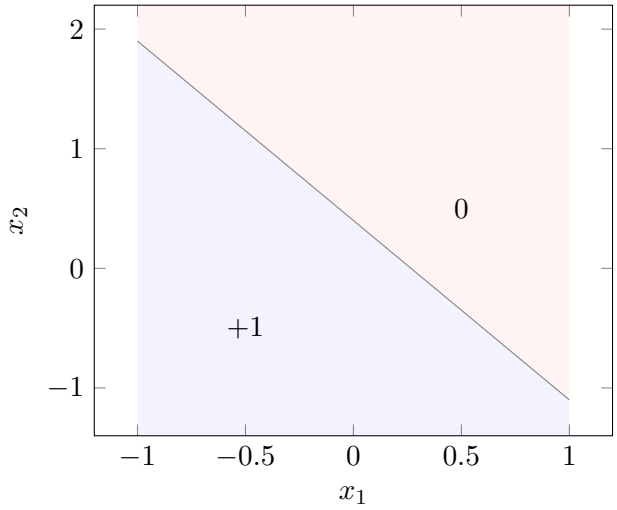


Figure 2.3: A linear separator implemented by a neuron with the parameters of: $-1.5x_1 - x_2 + 0.4$. The blue area corresponds to the region of input space that would cause the neuron to predict 1, and the red one 0. The black line is the linear function implemented by the neuron.

2.1.2 Networks of Neurons

Individual neurons can be combined into networks to perform more complex tasks. They are usually organized in layers, and networks contain one or more of them. We call networks that contain more than one of these layers a multi-layer perceptron (MLP). In Figure 2.4, we show an example of an MLP with three layers.

Each layer is a collection of neurons reading the output values of the previous layer and outputting values to the next. An advantage of organizing neurons in this manner is that they may specialize: neurons in a layer may share the same inputs but they can learn to solve different parts of a task by combining them in different ways. Furthermore, a special type of layer is the output layer. It performs the final mapping between the outputs of the previous layer into predictions. Following the example of neurons performing different tasks in a layer, a network's output layer would then consolidate the results coming from those different neurons into a prediction.

Layers that are neither outputs nor inputs are called hidden layers. The universal approximation theorem [4] states that a sufficiently large MLP with a single hidden layer can approximate a wide class of functions with arbitrary precision. However, it may not be feasible nor efficient to train such a network. A set of parameters may exist for a network architecture that solves a task, but optimization methods used may not be able to compute them. Furthermore, MLPs with additional hidden layers may be better suited for learning these same functions with a smaller number of neurons [5]. Deep learning models use this insight to solve more complex tasks with deeper models that would not have been as practically feasible to solve with shallower ones.

2.1.3 Training Neural Networks

The parameters $\boldsymbol{w}$ of each neuron have to be optimized so that the network learns to perform a task. This is called training the neural network. A dataset for supervised learning is made of samples with inputs $\boldsymbol{x}$ and target values $\boldsymbol{y}$. Furthermore, a single-layered network with linear activation functions may easily be trained by solving a system of linear equations with a method such as least squares. However, this is not the case for deeper networks implementing non-linear activation functions. These are usually trained using gradient-based methods and the backpropagation algorithm.

Gradient descent methods require gradients to be computed for the parameters (weights) of the network. Furthermore, these gradients need to be computed from a function that maps network parameters to the error in its predictions. An error function is used to calculate a loss value for the network by comparing its predictions $\hat{\boldsymbol{y}}$ with the target values $\boldsymbol{y}$ from the dataset. A common loss function is the Euclidean distance $||\hat{\boldsymbol{y}} - \boldsymbol{y}||$. Loss functions produce an *error signal* which indicates the direction of the error in the predictions. One of the challenges in training these networks is establishing the responsibility of each weight in the network for this error. That is, computing the gradient for every parameter in every layer. This is made difficult by the fact that the layers are stacked, and parameters affect the next layers. The backpropagation algorithm carries the error signal back through the network computing the gradients for each of the parameters in the process. A gradient-descent-based optimization algorithm is then used to update each weight based on its gradient.

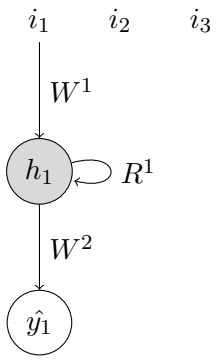
2.2 Recurrent Neural Networks (RNN)

Traditional neural networks make it difficult to apply them to problems with variable-sized inputs such as text. Furthermore, these sequential inputs usually have dependencies between inputs: e.g. an adverb modifies a verb. Here, we refer to an input (element) as a general value associated with a discrete time step that may not be a strictly scalar. As an example, an electroencephalogram time series would be composed of a series of input vectors.

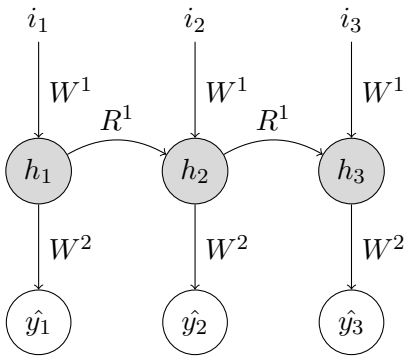
One could build feed-forward networks for tasks involving time series. The length of the sequences can be capped and all sequence values provided as inputs to the network. Then, it could learn to consider the relationships between inputs over this added dimension of inputs. This strategy would allow the network to take a sentence as an input, for example. In this scenario, different sets of weights would be responsible for each word in this sequence. However, the input

elements are similar in nature, i.e. they are all words from a language. Therefore, each word is treated separately, and the network would need to learn multiple times the same domain. Once for each position in the input series and its corresponding sets of parameters. This makes the use of traditional neural networks unwieldy and inefficient for time series.

Recurrent neural networks are extensions of feed-forward networks that allow recurrent connections in their network graph. This allows the state of the network in the present to influence it in the future. This differentiates it from feed-forward networks whose outputs (and states) are solely defined by their current inputs. These networks can process input sequences of variable lengths while still preserving the context between each element in the sequence.



(a) a network with a recurrent connection in its hidden neuron. W^1 and W^2 are its 'feed-forward' parameters and R^1 its recurrent one.

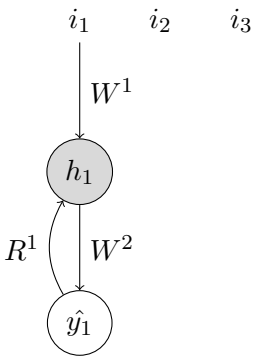


(b) The unrolled network from the left. The weights remain the same while the hidden neuron passes its activation to itself in the next time step.

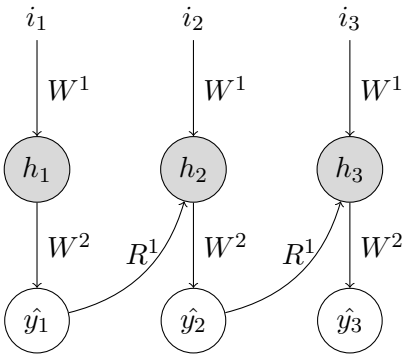
Figure 2.5: An example of a recurrent neural network and its unrolled equivalent. Variables inside nodes represent the output of the corresponding neuron. The neuron in gray is a hidden neuron. The i_* values form a time series which is inputted to the network.

Recurrent networks process inputs as sequences, with one input at a time, allowing for signals to cross the time dimension and influence the behavior of the network in the future. Since their operation is discrete in time, the recurrence is also stepped. That is, the output of a neuron at time step t can be sent as an input to a neuron in a future time step. Due to this discrete operation, these recurrent connections can be unrolled and the network transformed into a feed-forward network for a given sequence length. In figure 2.5a we show an example of a network with a recurrent connection in its hidden neuron and in 2.5b an unrolled version over three-time steps . Although recurrent networks can be unrolled for visualization into feed-forward ones, the weights in each time step remain the same. This means that the same parameters are used for all inputs over time. This provides statistical efficiency. In the previous feed-forward example we stated that multiple sets of parameters would have to be learned to perform the same task with a subset of the input for each. The sharing of parameters by recurrent networks means that only one set of parameters needs to be learned and the entire dataset may be used for that. This improves the efficiency in the use of the dataset much like in convolutional neural networks.

Recurrent networks may be configured in different ways to be employed for different tasks. They may be used to generate individual predictions for each input element (e.g. NER), a single prediction to an input variable in length (e.g. sentiment analysis), or a prediction with a variable length that is different from the one of the input (e.g language translation). Furthermore, these



(a) a network with a recurrent connection from its output to its hidden neuron. W^1 and W^2 are its 'feed-forward' parameters and R^1 its recurrent one.



(b) The unraveled network from the left over three time steps. The weights remain them same while the hidden neuron reads the previous output.

Figure 2.6: An example of a recurrent neural network and its unraveled equivalent. Variables inside nodes represent the output of the corresponding neuron. The neuron in gray is a hidden neuron. The i_* values form a time series which is inputted to the network.

different architectures are constructed with two main types of recurrent connections:

- **output-to-hidden**: The output of the network from a previous time step is used as an input to a hidden layer. See Figure 2.6.

- **hidden-to-hidden**: An output of a neuron from a hidden layer at a previous time step is provided as input to another hidden neuron. See Figure 2.5.

These types of connections can be used simultaneously in an RNN, and work in the same way. However, there is an important distinction: output-to-hidden connections require that whatever information needed for predictions in future time steps can be extracted from a previous output value of the network. This may not always be the case. Hidden-to-hidden connections share the hidden state of the network which may contain more information on the history of inputs and may be better capable of representing long-term dependencies. These recurrent connections can also vary in their length in time. A value from time step $t-4$ may be used at time step t. This setting is used to facilitate learning of longer-term relationships.

The recurrent connections in RNNs make their training less straightforward. Parameters influence the error in predictions not only in the current time step but all future ones, with differing levels of importance. Therefore, when adjusting these weights the training algorithm needs to take into account these side effects. As shown in Figure 2.5, a recurrent network may be unrolled into a feed-forward network for a given input sequence length. An extension of the back-propagation [6] algorithm uses these unrolled networks to train the network using gradient descent, allowing the error signals to propagate back in time. The added depth in RNNs, which is evident in their unrolled form, is a challenge to training with issues such as vanishing gradients. The aspects of training recurrent neural networks are outside the scope of this work since we do not employ these methods.

2.2.1 Long short-term memory (LSTMs)

Recurrent neural networks allow information to be retained over a series of inputs. However, there is no fine control mechanism for this memory. For example, information in some time steps that are not relevant may be incorporated into the hidden state. Long short term memory (LSTM) [7] is a recurrent architecture that allows models to adjust the recurrence dynamics by themselves. This network is composed of layers of cells, each of which contains a hidden state. Furthermore, specific neurons called gates control how this state is updated.

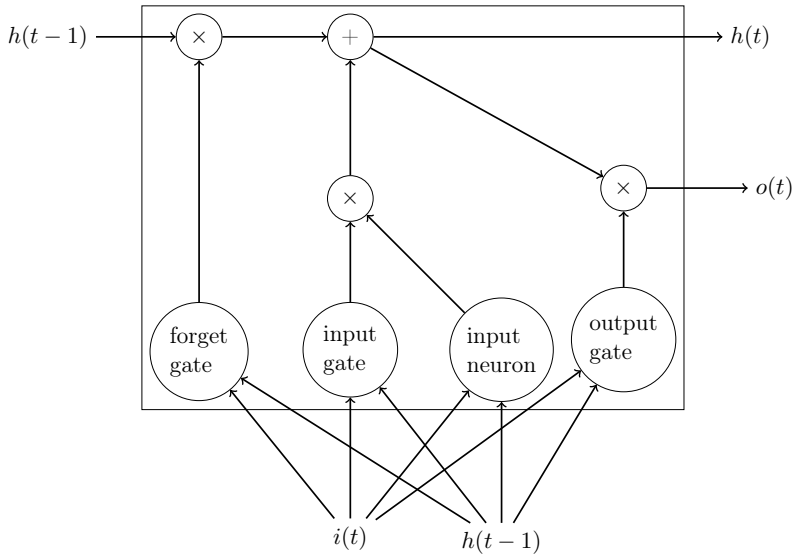


Figure 2.7: An illustration of an LSTM cell. $i(t)$ is the input to the cell at time step t, $h(t)$ the hidden state and at t and, $o(t)$ the output at the same time step. Nodes with $\times$ or $+$ represent the corresponding arithmetical operations that are applied to its inputs.

The input vector of the cell is processed by a neuron that produces an input to the cell. The input gate determines the rate at which this new information is incorporated into the hidden state. This neuron takes the inputs of the LSTM layer at the current time step and its outputs in its previous time step and produces a weight value. This value is multiplied with the input of the cell and the result is added to the state. Similarly, the cell's output is produced by multiplying the hidden state with another factor produced by an output gate.

A recurrent loop in the hidden state of the cell is also present, allowing information to be passed to the next time steps. Originally, the weight of this recurrence was determined by a learned parameter of the network. That is, after training it was fixed. This may lead to the accumulation of values into the hidden state. Therefore, information may lose relevance but persist in the network due to its recurrent connections. [8] addresses this issue by introducing a forget gate to adjust the weight of this recurrence. This gate reacts differently depending on the state the network is in and its inputs. In this way, it allows the cell to self-adjust its memory. Shortening or extending it according to its needs.

LSTM gates allow the dynamics of the recurrence to be adjusted on-demand. They use inputs and previous hidden states as features to parametrize the recurrent connections. The parameters of gates are optimized during the training stage, but they are capable of adjusting the recurrence

during the inference stage. Therefore, this is an improvement over general RNN architectures. Once trained, they will follow a fixed course in their recurrence and do not allow for adaptations.

2.3 Echo State Networks (ESN)

Echo state networks (ESN) come from the idea that the state of RNNs may be interpreted as a summary of all the input values it has received in the past, at least under certain conditions. A recurrent network capable of producing a summary of the provided inputs, regardless of its original state, is said to fulfill the echo state property [9]. This concept of states containing *echos* of previous inputs names this network.

Instead of a trained recurrent network like the ones mentioned before, ESNs contain a *reservoir* and an *output function*. The former is a recurrent neural network that is sampled from a distribution determined by hyperparameters, and, critically, its connections' weights are also sampled. The latter computes the outputs of the network $\boldsymbol{o}(t)$ at time step t using the state of the neurons in the reservoir, and is usually composed of a linear layer. This simple output function can then use the summary provided by the reservoir's states, the echo state property holding, to predict an output taking into account the history of inputs.

An illustration of the ESN is provided in Figure 2.8. The connectivity of the inputs of the network $\boldsymbol{i}(t)$ to the reservoir's neurons is determined by a sampled weight matrix $\mathbf{I}$. The input weight for each input dimension j and neuron k is determined by $\mathbf{I}_{(k,j)}$. If the value is 0, there is no connection between the two. The connections between neurons in the reservoir are also modeled by an adjacency weight matrix $\mathbf{R}$ where $\mathbf{R}_{(l,k)}$ is the weight from neurons k to l. The state of the reservoir at time step t, as shown on Equation 2.2, depends on its previous value at $t-1$ and the sum of its weighted inputs. Furthermore, each neuron has a leaking rate α which limits the speed of change in its state values.

$$\boldsymbol{o}_{(t)} = \alpha(tanh(\mathbf{I}\boldsymbol{i}_{(t)} + \mathbf{R}\boldsymbol{o}_{(t-1)})) + (1-\alpha)\boldsymbol{o}_{(t-1)} \tag{2.2}$$

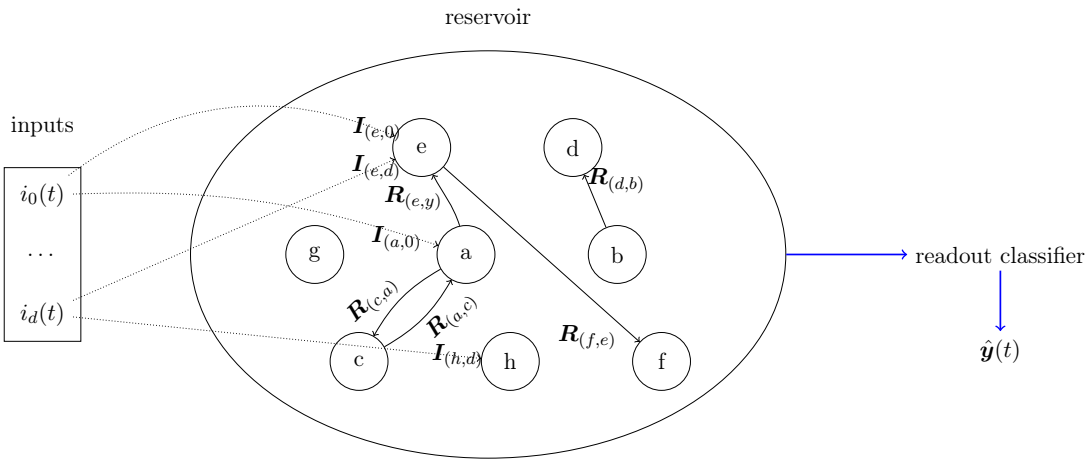


Figure 2.8: A representation of an echo state network with an 8-dimensional reservoir and n-dimensional inputs. The embedding vector generated by the reservoir is fed into a classifier which produces the predictions $\hat{\boldsymbol{y}}(t)$

A common problem of recurrent networks is their stability. Due to their recurrent connections, the same weight matrix R is applied at all time steps. This matrix performs a linear transformation that stretches components of the state along the eigenvectors of R. Given enough time, the values of the states may explode or vanish much like what may happen with gradients while training recurrent networks with gradient descent. A way to understand how this works is by assuming we can decompose R into its eigenvectors and a diagonal matrix of eigenvalues $\boldsymbol{\lambda}$:

$$\mathbf{R} = Q\ \boldsymbol{\lambda}\ Q^{-1} \tag{2.3}$$

Applying the $\mathbf{R}$ x times to the state $\boldsymbol{o}(t)$ is equivalent to having the eigenvalues of $\mathbf{R}$ to the x^{th} power.:

$$\mathbf{R}^x \boldsymbol{o}(t) = Q\ \boldsymbol{\lambda}^x\ Q^{-1} \boldsymbol{o}(t) \tag{2.4}$$

Hence, the larger the value of x, the greater the main eigenvalue will be compared with the others.

If the magnitude of the eigenvector is too great, some components in the reservoir state will be amplified until they completely drown out the rest. This compromises the memory capacity of the reservoir and its adherence to the echo state property by preventing the adequate representation of the input history. The spectral radius hyperparameter is used to rescale the largest eigenvector in the sampled R matrix. Ensuring the largest eigenvalue is smaller than 1 is usually sufficient to satisfy the echo state property [10].

The capability of ESNs to summarize past inputs is relevant for problems involving series of interrelated inputs such as time series or text which have to be encoded into a fixed-sized vector. This encoded information can then be used to make predictions. Furthermore, training the weights of the linear layer computing its outputs is simply the case of solving a set of linear equations which can be done using gradient descent or the Moore-Penrose pseudoinverse. Naturally, the sampled nature of the reservoirs means that it can be outperformed by traditional, trained, RNNs but that comes at the cost of training those recurrent weights.

Hyperparameters of Echo State Networks

The hyperparameters will affect how the reservoir perpetuates or forgets past input values and they can be tuned for improved performance. A comprehensive guide on ESNs, their hyperparameters, and how to use them is provided by [10]. Among the more important are:

- **reservoir size**: the number of nodes in the reservoir. The larger it is, the greater its capacity and stochastic stability are.
- **leaking rate**: controls the rate of change in the states of the reservoir. See Eq. 2.2.
- **reservoir connectivity rate**: governs the rate of connections within the reservoir. e.g. a rate of 10% means the probability of any directed edge between nodes in the reservoir existing.
- **input connectivity rate**: analogous to the reservoir connectivity rate but between inputs and reservoir nodes.

- **input scaling**: the interval from which the inputs weights are sampled. Determines the importance of the inputs compared with the recurrent values in the reservoir's neurons.
- **reservoir scaling**: the interval from which the reservoir weights are sampled.
- **spectral radius**: value used to adjust the largest eigenvalue of the reservoir weight matrix $\mathbf{R}$. It affects the stability of the network and spectral radius values smaller than 1 are more likely to fulfill the echo state property.

Deep Echo State Networks

Traditional echo state reservoirs have a completely unconstrained topology, governed by the distribution generated by the reservoir connectivity rate. Like for MLPs, it has been proposed to add explicit depth to the reservoirs in ESN [11].

Depth can be introduced to ESNs by stacking multiple layers of reservoirs, the output of each serving as input to the next (see Fig. 2.9). Then, each layer n has its own set of input $\mathbf{I}^{(i)}$ and recurrent $\mathbf{R}^{(i)}$ weights. At each level, the corresponding reservoir produces intermediate outputs $\boldsymbol{o}^n_{(t)}$. These are concatenated onto a single output vector for the entire deep reservoir, which is used by a classifier in the same way as the general case.

In the same way as deep MLPs, there is evidence that Deep ESNs may be able to outperform their traditional counterparts [12]. Furthermore, the increased depth in reservoirs may facilitate the representation of the input at different time scales in its different layers [11]. This may prove useful in problems with long and short dependencies like text (e.g. a pronoun refers to a noun defined previously and is the subject in the current sentence)

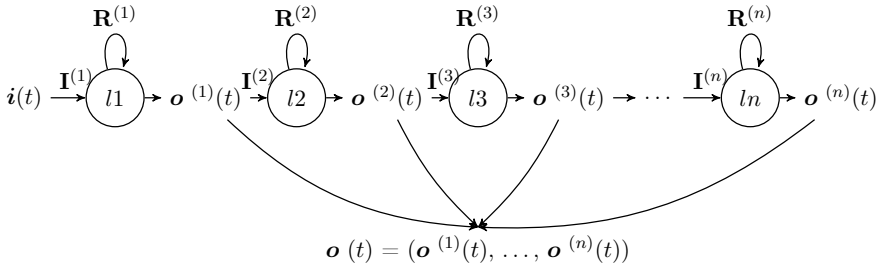


Figure 2.9: The detailed layout of a deep echo state reservoir. The different layers of reservoirs l_i produce outputs $\vec{o^i}$ that, concatenated, form the state of the deep reservoir $\vec{o}$.

2.4 Textual Embeddings

Traditionally, machine learning NLP models used to rely on handpicked features such as whether a word is capitalized or its part-of-speech (e.g. noun and verb). Selecting them can be quite tedious and may overlook important features in the text. Furthermore, they may be domain-dependent. This problem has lead to textual embeddings which are learned representations of text (e.g. characters, words, sentences) as vectors in a high-dimensional space. These (lossy) encodings of text into easy to manipulate numerical vectors can be used by machine learning models as features for NLP tasks.

Glove embeddings [13] map words from a set vocabulary considering how frequently every pair of words co-occurs within the same context in the training data. It does not take into account the context of a word being embedded after training. However, the context in which a word is included is very important to its interpretation. For example, a verb is associated with a subject but by embedding each word individually, we ignore the subject while embedding the word and vice versa. Therefore, information on how the circumstances this word is being used is lost.

It is also possible to generate such embeddings using neural architectures. In [14] a large unlabeled dataset is used to train a character-based network in an unsupervised manner. This network is implemented with the Long Short-Term Memory (LSTM) recurrent architecture. The outputs of these networks, as mentioned before, result not only from the inputs at the current time step (or character) but also from previous inputs. The hidden states in the LSTM cells of this network contain this summary information and can be used as features for other downstream tasks. These *flair embeddings* are different from Glove in the sense that they take into account the context of the sentence being embedded and not only the word itself. This architecture is also used in a bidirectional manner where one model is trained by predicting the next character in the forward direction and another in the backward direction, i.e. from the last character to the first. This allows the combined embeddings of a word to include the context from the text before and after it.

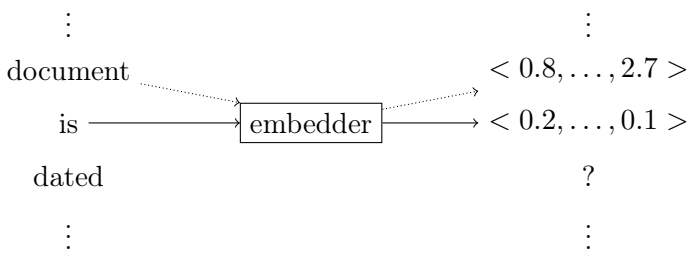


Figure 2.10: A high level example of an embedder generating vectors for each word in its input.

Another advantage of language models is the opportunity for transfer learning. Learning the representations in high dimensions require a very large dataset (the English corpus used by Flair contains a billion words). However, datasets for most tasks are a fraction of the size which is not enough for learning these representations. Fortunately, the embedders can be trained with a sufficiently large unlabeled dataset, and the trained network simply reused for other tasks with other datasets. In this case, the embeddings layers are kept fixed, while the ones solving the downstream task are trained with the application-specific dataset. Therefore, the use of pre-trained embeddings can be quite useful for solving NLP tasks.

2.5 Named Entity Recognition (NER)

Named entity recognition (NER) is a natural language processing task concerned with extracting information from text in the form of named entities. In the NER task types of information are modeled using classes such as person, address, location, or company name. An entity is a span of one or more words that belong to a NER class. The classes in NER can be customized, allowing

the application of NER methods to multiple domains. The task of NER is then to identify the span of words that define the borders of an entity and classify them into one of the defined classes. The dataset of a NER task is most commonly composed of a series of tokens (words) with assigned labels. Each label corresponds to a NER class when the token belongs to an entity and is a null or outside value when it is not part of one. The labels from the words are used to construct entities by joining consecutive words of the same class. However, when an entity follows another of the same class it is not possible to distinguish them. For this reason, a prefix is added to the name of the class in some datasets. They indicate where the token is in an entity, such as *BEGIN-PERSON, IN-PERSON, END-PERSON*. In this way, the borders of adjacent entities are easily distinguishable.

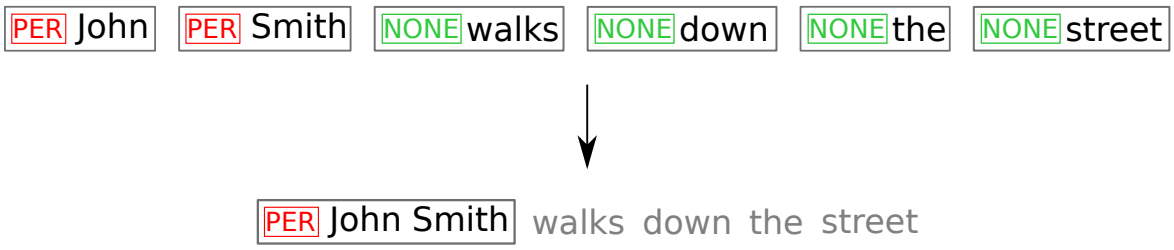


Figure 2.11: The formation of entities. John Smith are two adjacent tokens with the PER tags and they form a single entity.

In Figure 2.11 we show an example of a sentence labeled with NER tags. In this example, we have used no prefixes and assume that adjacent tokens of the same class are part of the same entity. Therefore, *John* and *Smith* form the entity *John Smith*. The goal of a named entity recognition system is to find and recover the entities of interest in the text. Solutions normally involve predicting the individual classes for tokens and then extracting entities from them.

NER is a form of information retrieval from text. It can be applied on a small scale to single sentences, but also large documents. In the latter case, it may be used to extract relevant key information from documents for tasks such as indexing. In Section 3.2, we describe some of the approaches and applications of the NER problem.

2.6 Conditional Random Fields (CRF)

In some tasks, predictions are not independent but are part of a context. In the case of NER, the class of a word may influence that of its neighbors. For example, in *baker street* the word *street* hints that the word *baker* refers to a location. Meanwhile, in the sentence *the baker is on their shop*, the word *baker* has a completely different meaning. Conditional random fields (CRF) models learn the assignment between inputs and predictions and the relationship between neighboring predictions. Therefore, predictions are based on inputs and the predictions computed for neighboring inputs. These models may be applied in tasks where there is a definition of a neighborhood between inputs such as the pixels in image segmentation, phonemes in speech recognition, and words in natural language processing.

Chapter 3

Related Work

Natural language processing research is extensive, and named entity recognition has been studied for many years. In this work, we combine language models and echo state networks to provide a neural network-based approach to the NER problem in financial documents. Our work is on the intersection of NLP and neural network research. Therefore, we organize this chapter based on the areas of interest. In the first section, we discuss language models and some of the relevant developments that enabled this work. This is followed by a discussion on the named entity recognition problem and we finish by discussing information extraction from unstructured documents.

3.1 Textual Embeddings

Supervised machine learning models rely on features that meaningfully describe samples for successfully learning tasks. Finding good features for natural languages is challenging due to their complexity and ambiguity. How can someone convey the multilayered meaning of a sentence with a numerical feature for a model? How does one account for the many meanings a word has, slang, irony, or regional differences? Specific datasets may contain particularities that make it easy to create such meaningful features. However, these approaches, when possible, are limited in applicability due to the cost of manually building them.

There are approaches for generating features automatically in a simple way. In a bag-of-words model, a vocabulary is formed from the words in the training data. This vocabulary is used to create a vector definition where each position corresponds to one of its words. A document is represented by counting the occurrences of each of its words and filling a version of the vector defined earlier with those numbers. This approach is limited by the fact that it does not consider the order the words appear in, ignores new words, and does not take into account similarities between them. Other approaches involve the use of character n-gram representations, where the text is decomposed into sequences of n characters. A sliding window scans the text and generates all n-character substrings it contains. Features may be generated from these sub-word representations and used in machine learning models.

Automatically generating features is valuable and can overcome the domain-specific limitations of manual feature construction [15]. One way to represent a document is by describing the words it contains. This has motivated research in using deep learning models and other techniques to

quantify the meaning of languages in a manner that they can be used as features for machine learning models. These representations are called textual embeddings and are projections of text onto a high-dimensional space. Moreover, embeddings that may be reused for multiple tasks and domains are of special interest.

Embeddings are learned representations of textual data which leverage large unlabelled datasets to learn the intricacies of natural languages and creates vector representation for text. In [13], the authors propose Glove embeddings which use unsupervised learning to project words onto a high-dimensional space where each word is assigned a vector representation. In this space, similar words are placed closer to each other than dissimilar ones. To achieve this, they calculate the co-occurrence probabilities of pair of words within a context to create a mapping of words to the embedding space. These text embeddings can then be used in other natural language processing models for which the meaning of individual words is relevant.

In [16], the authors propose two word embeddings extracted from the hidden layers of two models. One is called CBOW and it is trained by having the model predict words based on its neighbors. Another is the skip-gram model which is trained to predict the neighbors of a word.

The approaches at text embedding mentioned until now have focused on word-level embeddings. However, these models may have an approach that is too granular. Words may have repeating characteristics that could be exploited (e.g. suffixes) but word embeddings treat each word as indivisible. To address this problem, [17] generates n-grams from the training corpus and computes embeddings at the n-gram level. Word embeddings are calculated by summing the embedding vectors of the n-grams they contain. In [18], fasttext embeddings are created by extending the CBOW model to take into account the relative positions of words, their distance, and n-gram embeddings.

Deep neural networks have also been used to automatically generate features for text. In [14], authors build a recurrent neural network-based language model that given a sequence of characters, learns to predict the next character. This approach treats languages as a character distribution and has the model learn to represent them from their building blocks. All the concepts in a language such as words and syntax are also learned from characters. This circumvents the issues encountered with word-based embeddings. The model is then capable of generating word embeddings that take into account the spatial relationship between words in a sentence and not just the co-occurrence rates in a corpus. Due to this ability to take into account the context of words when embedding, the authors state that the embeddings produced are *contextual*. Furthermore, since it uses characters as inputs, it does not rely on a vocabulary and words not seen previously can be embedded as easily.

3.2 Named Entity Recognition

Named entity recognition is an established problem in natural language processing and is concerned with the extraction of named entities from text such as an address or a given name. Initially, rule-based systems were often used, which require domain experts to craft them. Furthermore, the reliance on rules or databases makes solutions more domain-dependent [15]. However, with the availability of labeled datasets, machine learning systems became more common [19]. These systems use engineered features (e.g. word prefix, capitalization, and part-of-speech) which require

human intervention for their selection as well as language databases. More recently, work in deep learning has allowed the use of textual embeddings as features for machine-learning models for the NER problem.

Although rule-based approaches have been used for a long time, they are still relevant. In [20] authors build a rule-based system to extract dietary recommendations from the scientific literature. An advantage of rule-based models is the explainability of the model. Developers know exactly how the model takes decisions because the rules are explicit. [21] argues that rule-based can outperform some learning-based systems. The paper investigates ways of extending a generic set of rules, specializing them towards particular domains. In this way, the effort of rule creation can be reduced since some rules may be reused and only those contradicting the requirements of the domain changed. In [22], authors compare the performance of different approaches in datasets of medical reports from imaging exams. A rule-based system is compared with two machine learning models. One is a deep learning model with LSTMs that learns character and word embeddings from the training data. The other is a semantic search system using transfer learning. The authors conclude that rule-based systems perform better but are more costly to build.

Machine learning models are also used in combination with engineered features. Such is the case in [23]. In this work, the authors use multiple features which include the words in the training data (if a word is present in the dataset, its corresponding feature is true), orthographical characteristics, and semantic features. The latter is derived from domain knowledge. A conditional random field model is used to compute the probabilities of entities and their classes. Keywords that were found to be associated with a particular NER class with a high confidence level, are used for disambiguation in cases where the model is unsure about its prediction.

The use of deep neural networks for solving the NER problem has been studied extensively, usually using RNNs [15]. The use of text embedders makes it possible to use off-the-shelf automatic feature generators in combination with custom models which contrasts with the manual techniques used previously. [14] demonstrates this by employing their contextual embeddings to the NER problem using an architecture of bidirectional LSTMs and a conditional random field output layer. [24] proposes a deep learning model for character-based NER. Their dataset is labeled at the character level, and the model uses a 5-layered bidirectional LSTM architecture to predict NER classes for each character. Class probabilities are predicted for each character and used by a post-processing step that produces the most likely class for each word.

The application of the combination of echo state networks and text embeddings to the NER problem has been approached by [2]. In that work, the authors use embedded documents as inputs to echo state network reservoirs. Although f1 scores were not as good as an LSTM-based benchmark, training times were significantly shorter indicating that in use cases with constraints on training cost, ESNs can be a good alternative.

3.3 Information Extraction (of Financial Documents)

Automatic information extraction enables institutions to leverage the amount of information they have frozen in archived documents. [25] approaches the problem of acquiring key basic

information on countries and businesses to feed into business intelligence systems. The authors use ontologies to model the types of information to be extracted from several document sources.

Automatic information extraction from financial documents is of great importance due to the number of publicly traded companies, the volume of reports they generate, and the lack of structured formats. [26] presents a system that crawls the internet for newly published reports, parses and extracts key performance indicators (metrics that concisely convey the important aspects of a company and their business) from the text, and stores them on a database. A graphical interface allows the user to query such information from the database on demand.

Anonymization can also be framed as a form of information extraction. In [27], the authors address the problem of redacting sensitive information from documents (e.g. personal data, information on business partners, or locations) so that they can be included in NLP datasets without the data security restrictions. Sensitive information is extracted through NER and then replaced with their respective entity names preserving the context.

Information extraction may be combined with other tasks. Financial reports of public companies have to be filed following standards intended to improve transparency and protect investors. Before they are published, however, the compliance of these documents must be verified by independent auditors. [28] presents a recommender tool that assists an auditor by processing financial documents and extracting the passages of text that are relevant for determining if a requirement is met. These are displayed to the user along with the matching requirement. In this task, the model must first find where a requirement is met in the text (e.g. the name of the company reporting data) and then suggest it to the user.

Similarly, in [29], the authors explore sustainability reports published by companies in the context of ESG (environmental, social, and governance) guidelines. Some ESG indicators can be easily measured such as carbon emissions but others, e.g. company policy, are not. This work uses Glove word embeddings to embed sentences in the document and the text descriptions of indicators. It then computes similarity metrics between indicators and sentences using the cosine distance in the embedding space. Those sentences considered relevant for an ESG indicator are selected, and if the indicator is quantifiable the numerical values are extracted from the text. [30] addresses the problem of extracting key information from loan agreements for assisting in credit risk assessments. The information is extracted using NER entities, and a conditional random field model with hand-picked features is used

In [31] the authors use embeddings generated with the BERT language models to build character-level deep learning models for NER tasks. Furthermore, they train models in datasets of financial or biomedical documents and explore transfer learning techniques to fine-tune models with little effort for use in another NER domain where training data was significantly limited. The ability of models to perform well with small training datasets is relevant for custom industrial applications of NER where annotated data is often scarce and expensive to acquire.

3.4 Why our Work is Relevant

Institutions have access to large volumes of unstructured documents containing potentially valuable information. However, without a solution for automatically retrieving it at scale, it becomes an unusable asset. As we have shown in this chapter, information extraction systems

are often a topic of study and have the potential for industrial applications. Moreover, the great variety in documents with their particularities, such as sustainability reports, contracts, or court decisions, presents a challenge requiring customization for each domain. Furthermore, training data may not be available in significant amounts due to the cost involved in producing annotations. Thus, for solutions to this problem to be viable to adopt, they must be adaptable and low cost. If they are too expensive to train, unflexible, or require large labeled datasets they will not be cost-effective for the niche use cases created by the variety in document data.

Traditional learning models involving hand-picked features may require manual intervention in their selection for every new domain. The same limitation occurs with rule-based ones. Language models hold promise in providing general feature generation for NLP models. This is also the case with smaller datasets, as long as pre-trained models are used. As we present in the next chapter, ESNs combined with language models may be a solution to the information extraction problem when modeled as named entity recognition. They implement cheap recurrent architectures that are also easily adaptable to different domains. As well as can perform reasonably well considering their low cost.

Chapter 4

Methodology

We frame the problem of information extraction from prospectuses as a named entity recognition (NER) task and propose a network combining contextual embedders and [deep] Echo State Networks (ESNs). Using a dataset labeled at the entity level by domain experts, we train our model to predict NER entities. Document-level information may then be extracted from the NER predictions in a post-processing step. A particular challenge we face is the scarcity in training data, with only a small fraction of tokens in the dataset assigned to an entity label.

Our model contains three main components: a bidirectional text embedder, a deep echo state *reservoir*, and a *readout layer* which is either a single linear layer or a multi-layer perceptron (MLP). We use Flair to generate contextual embeddings for each word in the text which serve as inputs to a deep echo state network. A reservoir with two sets of states is used: one for the forward and another for the backward direction. Each of these is responsible for processing the corresponding Flair embeddings. The deep echo state network with its recurrent non-linear connections generates another set of (also bidirectional) states which are used by an *output network* to predict the NER classes. In Figure 4.1, we show an example of our architecture. The details of which we describe in this chapter.

4.1 Data Preparation

The dataset is composed of *prospectuses*, financial documents issued by companies preceding public offering of securities on regulated markets. They are meant to inform potential investors of details on the company's business, for example, liabilities that may affect returns. These documents follow standards, but there is no unified structured format. The documents are stored in PDF format and are converted into raw text using the computer vision solution described in [32]. This parsing tool uses a neural network that segments each page into its different components. e.g. paragraphs, tables, headers, and figures. Each segment is then individually parsed by directly reading the text data from the PDF file or performing optical character recognition. Figures are either discarded or have the text contained in them parsed. The parsed text from each component is retrieved and used in the following steps. We also retain the structure of the document by keeping the writing from different segments separate. We call each of these segments a *paragraph* regardless if they come from a header, a table, or an actual paragraph in the document.

The PDF documents in the dataset contain annotations made by domain experts with custom NER categories. The NER labels are represented using text highlighting and notes supported by the PDF standard. After the parsing of a document, the annotations are extracted from the original PDF file and matched to their corresponding paragraphs and spans of text in the dataset.

The relationship between consecutive paragraphs in a document is relevant for information extraction. Keywords sometimes precede a NER entity, and those can be in a previous paragraph. For example, a header on the cover page announces the underwriting bank which is followed by the name of the bank itself. Recognizing its name is not enough to determine what is its relationship with the document. Is it an underwriter or the company issuing the security? However, the parser may not include this header in the same paragraph as the NER entity. This makes previous paragraphs very important for understanding the current one. To preserve their context, paragraphs are grouped by their source document and sorted according to their (approximate) position in the file.

4.2 Generating Embeddings with Flair

Since the documents used can be quite large (some 500 pages long), embedding them with Flair all at once is impractical. Consequently, we embed one paragraph at a time in the order they appear in the document, i.e., embed paragraph p before $p+1$. However, this means that each paragraph is treated as its own document and we lose the inter-paragraph contextual information we require. We address this limitation in the next section. The embeddings for each word, however, remain contextual embeddings within each paragraph.

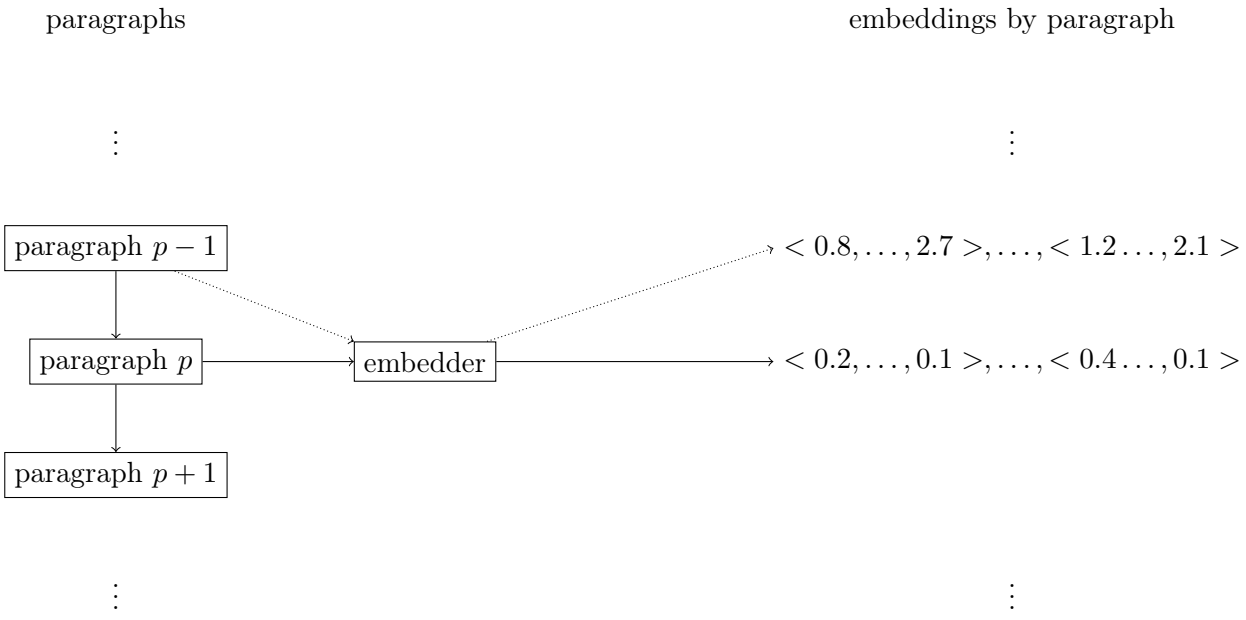


Figure 4.2: The process of embedding paragraph p with Flair.

Flair's *news-forward* and *news-backward* embedders were used to generate 4096-dimensional bidirectional embeddings for each word in the text. The backward embeddings are computed by

reading the text of each paragraph from the final character to the first. We show an illustration of the embedding process for paragraphs in Figure 4.2.

4.3 ESN Reservoir

Once the Flair word-level embeddings are computed for each paragraph, they are fed to the ESN reservoir. The ESN is used in a bidirectional manner and two ESN states are kept, one for each direction. In the same way as the previous step, the forward and backward ESN states are calculated sequentially, a paragraph at a time. We expect the echo state property of reservoirs to allow information to persist between paragraphs, albeit only in the forward direction. This should help mitigate the issue present in the way Flair embeddings are generated at a paragraph level. The backward embeddings are calculated in the same manner as the ones from Flair. They do not start from the last word in the document to the first, but only within each paragraph. That is the backward embeddings of paragraph p are computed after paragraph $p-1$. This breaks the definition of backward embeddings, but scanning paragraphs in the other direction would increase complexity due to additional I/O operations over a large dataset. Moreover, the forward relationship between paragraphs seemed to be more relevant. This breaks the definition of backward embeddings, but scanning paragraphs in the other direction would increase complexity due to additional I/O operations over a large dataset. Moreover, the forward relationship between paragraphs seemed to be more relevant. The forward and backward states outputted after each word are concatenated. The resulting vector forms the bidirectional states of the reservoir. In Figure 4.1 we illustrate how the ESN states are computed a paragraph at a time.

The hyperparameter selection was based on previous experiments with a similar setup using the CoNLL-03 dataset [3]. *leaking rate* and *spectral radius* were selected for further inspection using our dataset. Hyperparameters are kept constant between layers in deep reservoirs to constrain the size of grid searches.

4.4 Training the Readout Layer

Finally, we employ the ESN embeddings as inputs to the *readout layer*. Besides the usual linear layer, we also experimented with MLPs for this role to allow for non-linear classifiers. These had *tanh* activation functions and one or two 1000-neuron hidden layers. The readout layer predicts the log probabilities of the various NER tags from the ESN embeddings. Its weights are optimized using a Softmax loss function and Adadelta. After the Flair embedder, and later, the ESN *reservoir* has embedded an entire document, the readout layer uses the state generated for each word and predicts the log probabilities of the NER classes. We select the class with the highest probability as the model's prediction for that word.

Our goal with this design is to leverage Flair's powerful pre-trained embedders along with the recurrent capabilities of deep echo state networks to approach the NER problem for prospectuses in a low-cost manner. The output network is the only component that requires training, which uses conventional gradient descent methods. Furthermore, the reusability of components is an advantage of our approach. Changes to the text embedder, ESN reservoir, or the NER classes only require retraining the readout layer.

Another aspect we consider is that neighboring words on NER problems have an interdependency, with one influencing the probability distribution over classes of the others. We explore these relationships by introducing a recurrent connection from the output to the input of the *readout layer*. The network uses the one-hot encoded prediction from the last time step $\hat{\boldsymbol{y}}(t-1)$ as one of its inputs. This way, the model can learn to consider the previous predicted NER class when computing the class probabilities for the next word. Moreover, using an output-to-input connection does not require the expensive unraveling of recurrences. Instead, it is possible to use teacher forcing during training and provide the target $\boldsymbol{y}(t-1)$ instead of the model's prediction $\hat{\boldsymbol{y}}(t-1)$ from the previous timestep. We compare the output networks with and without these recurrent connections in the results section.

In our experiments, we first generate Flair embeddings for all the documents in the corpus and use them with several ESN configurations. Furthermore, each ESN reservoir setting is trained with every configuration of readout networks in a hierarchical grid search. The ESN reservoir states are computed and stored on disk for posterior use in training. This makes training a bit simpler: we have to generate the embeddings sequentially to preserve the context of the text, but we need to present them to the readout network in a randomized order to train it with gradient descent. Moreover, this stepped approach facilitates the reuse of the embeddings: multiple ESN reservoirs use the same Flair embeddings, and readout networks with different hyperparameters use their reservoir states.

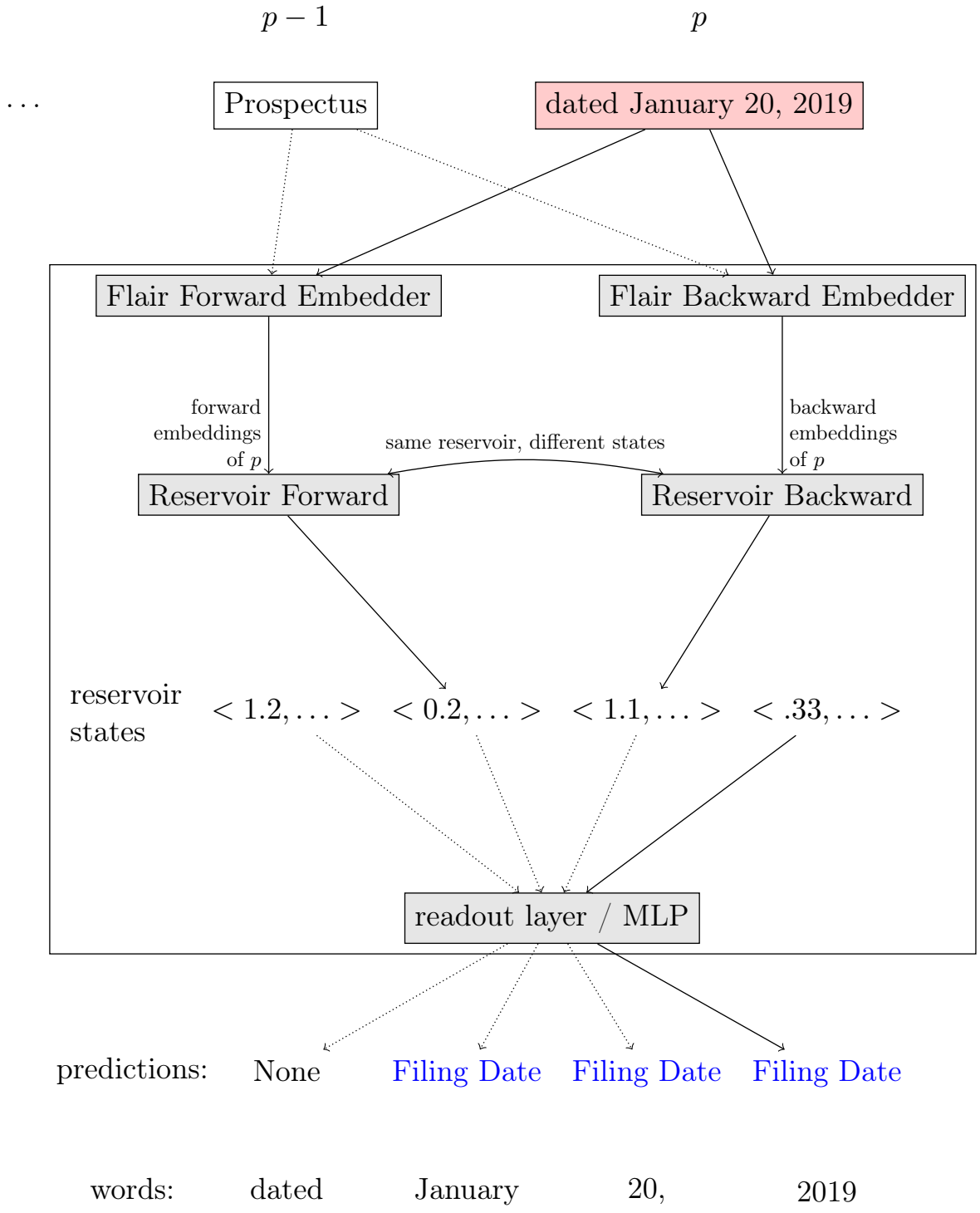


Figure 4.1: An illustration of our architecture while processing paragraph p. Rectangles on the top represent the ordered sequence of paragraphs from a document. The paragraph currently being processed is shown in red. The large box encapsulates our network. Dotted arrows between paragraph $p-1$ and the Flair embedders demonstrate that the former was previously embedded; The ones between the reservoir states and the readout layer show the word reservoir states that were provided to the predictor. The output of our network is a series of NER class predictions for each token of each paragraph.

Chapter 5

Results

In this chapter, we describe the experimental setup of our work, present results, and draw conclusions from them. In Section 5.1 we introduce our dataset and present some high-level information about its class distribution and size. We explain the steps taken to optimize the models including hyperparameter search on Section 5.2. These models were evaluated with different methods which we define in Section 5.3. Furthermore, we discuss our results in Section 5.4. We compare our models with a secondary benchmark obtained from an LSTM-based model in Section 5.5. Finally we present some examples of predictions from our best model in Section 5.6 and discuss our results in general and propose future work in 5.7.

5.1 The Dataset

We used a corpus of 59 English-language prospectuses published in the scope of public offerings of securities between the years of 2007 and 2009. All of these documents were manually annotated by domain experts with six custom NER classes or tags:

- **Accounting Standards**: the standard used for reporting accounting figures
- **Competitive Strength**: one or more sentences describing competitive advantages of the company such as *"The company's unique product range far outpaces that of its competitors"*
- **Issuer Name**: the name of the company issuing the securities
- **Initial Price Range**: The price range at which a security may be sold for. *"10 to 13 "*.
- **Investment Bank**: The bank overseeing the public offering. *e.g. Deutsche Bank*
- **Listing Venue**: The exchange where the security will be traded. *e.g. New York Stock Exchange*

The annotations represent the types of information we wish to extract from the documents. They were made at an entity level and then broken down into their constituent words (tokens). Hence, each word in the text was assigned one of the labels above or a *None* class. Although the corpus is large, only a few tokens are assigned a non-*None* class. Out of 11261178 words in the dataset, only 3515 have one of the NER tags above (see Table 5.1). Therefore, the dataset is

highly unbalanced. Moreover, the NER tags are sparse with only a few samples for each class. The number of samples may not be large enough for some NLP models. To mitigate the unbalance in targets, we rebalance the dataset by increasing the rate at which those samples are presented to the model during training. Furthermore, we did not use prefixes that subdivide the NER classes based on the token's position in their entity. *e.g., BEGIN-IssuerName, IN-IssuerName, and END-IssuerName.* This would have pulverized the few samples we have into many more. Thus, we restrict the number of classes the readout layer has to learn to predict and better use the limited number of samples for training.

This dataset is composed of individual annotated entities belonging to the 6 classes described above. Moreover, each document in the dataset may contain multiple entities of the same class or even none. The entities of the same class in a single document may also be similar to each other, e.g. multiple annotations for issuer name. Therefore, these entities should not be placed in different dataset splits to avoid skewing the results. For this reason, the splitting of the dataset was done at the document and not at the entity level. This leads to some unbalancing in the dataset splits because some documents may contain more annotated entities. We attempt to mitigate this issue with the entity evaluation method described in Subsection 5.3.2.

Table 5.1: NER class distribution of the dataset by tokens.

NER label	train	dev	test	total
Accounting Standards	282	36	12	330
Competitive Strength	1300	445	226	1971
Initial Price Range	176	65	45	286
Investment Bank	192	53	28	273
Issuer Name	298	73	42	413
Listing Venue	183	41	18	242
None	7805113	2485951	970114	11261178

5.2 The Model Setup

This corpus was used to train our model, a benchmark model without the ESN component (the readout layer attached directly to the Flair embedders), and Flair's sequence tagger. For Flair embedders, we used the *news-forward* and *news-backward* language models which combined project text to an embedding space of 4096 dimensions. To explore the effects of depth in echo state reservoirs, we experimented with one, two, and four-layer reservoirs. They contained respectively 4096, 2048, 1024 neurons per layer, maintaining the total number of 4096 neurons per deep reservoir for a fair comparison. This reservoir dimension is twice as large as the one used in [2] but the cost of increasing the reservoir size by this amount is small. In addition, three readouts were used: a linear layer, an MLP with one hidden layer, and another with two. The MLPs contained 1000 neurons per hidden layer. All readout architectures predicted log probabilities for each class.

We based the hyperparameters for the ESN reservoirs and readout layers on the ones used in [2] and also previous similar experiments. For this dataset, we initially experimented with *leaking rate* and *spectral radius* (0.7, 0.9 for both). We then fixed the *spectral radius* at 0.9 and

kept the two values for *leaking rate*. Since there are many hyperparameters, an exhaustive search was inconvenient. In total, we evaluated 24 combinations of ESN reservoir and readout layers. In Table 5.2 we show an overview of the hyperparameters used in the ESN reservoirs and the readout layers. The output networks were trained for 100 *epochs* with the Adadelta algorithm using a learning rate of 0.1 and weight decay of 10^{-5}. This algorithm and learning rate were also chosen based on preliminary experiments.

Table 5.2: Overview of hyperparameters used for the (deep) ESN reservoir and the readout layer. Hyperparameters involving the scaling of weights expect uniform distributions which we represent in brackets. The other comma-separated values represent the values used for the gridsearch.

ESN reservoir	values
reservoir size	4096
leaking rate	0.7 and 0.9
spectral radius	0.9
input scaling	[-0.25, 0.25]
input connectivity rate	0.5
reservoir scaling	[-0.5, 0.5]
reservoir connectivity rate	0.5
readout architecture	values
hidden layers	0,1 or 2
hidden layer size	1000

5.3 Evaluation Methods

5.3.1 Token Level Evaluation

The simplest evaluation method that we used is to consider the predictions from each word independently and evaluate them against their targets. We call this *token level* evaluation. In this method, we disregard entities altogether and are just concerned about how many token NER classes we correctly predicted. We then evaluate those tokens as a multi-class prediction problem.

Although this method is simple and is equivalent to the token loss-based optimization of the *readout* layer, it does not consider whether or not those predicted tokens form the desired entities. This may be too lax to serve as a good metric for information extraction.

5.3.2 Entity Level Evaluation

Our goal is to extract document-level information from prospectuses. Therefore, we used an evaluation method that better fits the application. We considered that each of the documents would have one or more entities for each type that needed to be predicted. We also wanted to account for variations in the number of target entities in different documents. If one document had 10 different target entities for a NER class while the rest had only 1, we did not want the success in predicting the targets for the former to excessively skew the results. Therefore, we adopted a layered approach. First, we compute metrics for each class in each document, then we do a simple average across all documents.

Before we can evaluate entities, we have to obtain them from the classified tokens. Since we do not predict tokens with positional prefixes, we assume adjacent words with the same NER class belong to the same entity as illustrated in Fig 5.1. For each NER class and document in the dataset split, we compute a list of the top 5 predictions for that class and document. The predicted entities are ranked by the average predicted probability of their constituent tokens. If the target entity is present in the top 5 list, we consider it a match and assign to that particular prediction a recall of 1. Here we employ a strict matching method: an entity matches another only if all their tokens match. Partial matches are not considered. The rate of predicted entities that match the target is the precision, and the F1 score is computed from these two values. If the document contains more than one target entity, we average the metrics obtained for each of the targets. We then average the document-level metrics again by NER class across all documents containing at least one entity of that class. This method is referred to as *Entity level* evaluation.

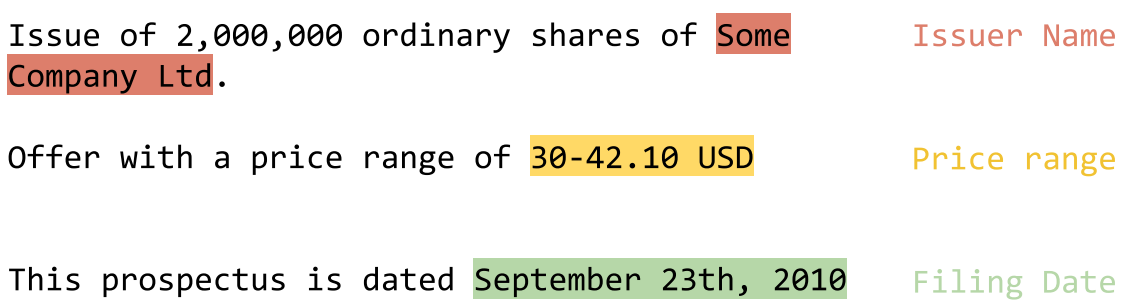


Figure 5.1: Example of entities from the domain of prospectuses. Adjacent tokens with the same class are assumed to be part of the same entity.

5.4 Experimental Results

For each model type (a combination of ESN reservoir and output network), we selected the best performing model based on the token level macro f1-score in the dev split of the dataset. In this section, we present the evaluation results of these models for the test split. To illustrate the contribution of the ESN network, we included metrics for the case where it is removed, i.e. 0 ESN layers, on the same tables. Moreover, we include data on different depths of reservoirs. We present consolidated token level metrics on Tables 5.3, 5.4 and 5.5, and Top 5 Entity metrics on 5.6, 5.7 and 5.8.

5.4.1 General Results

We present in Table 5.3 the main set of results with token level f1-scores. This type of metric is the same as the one used for model selection in the dev set.

The metrics as measured with the *token* level evaluation method are significantly better than the ones with the *Top 5 Entity* level evaluation. This is due to the strictness of entity evaluation: if we correctly predict the NER class of half of every entity's tokens, the token level recall will be 50 while that of the Top 5 entity would be 0. Nevertheless, we seek to extract entities and not tokens and the Top 5 evaluation metric is aligned with our use case.

Table 5.3: **Token** Level macro average **f1 score** for different networks on the prospectus corpus.

Model	Non-Recurrent Output Net.			Recurrent Output Net.		
	Linear	MLP1	MLP2	Linear	MLP1	MLP2
Flair-only	24.29	25.99	**30.67**	21.70	23.43	22.61
1L-ESN-4096	38.00	**41.88**	41.03	39.49	28.87	28.87
2L-ESN-2048	**39.18**	38.78	38.15	37.73	35.91	34.34
4L-ESN-1024	39.78	**43.63**	40.70	40.43	36.23	35.92

Table 5.4: **Token** Level macro average **recall** for different networks on the prospectus corpus.

Model	Non-Recurrent Output Net.			Recurrent Output Net.		
	Linear	MLP1	MLP2	Linear	MLP1	MLP2
Flair-only	56.13	52.15	49.27	26.65	26.16	24.50
1L-ESN-4096	51.56	44.75	45.35	37.80	25.45	25.34
2L-ESN-2048	47.83	40.00	37.77	36.77	32.91	31.90
4L-ESN-1024	47.77	41.75	40.62	40.90	33.61	33.20

Table 5.5: **Token** Level macro average **precision** for different networks on the prospectus corpus.

Model	Non-Recurrent Output Net.			Recurrent Output Net.		
	Linear	MLP1	MLP2	Linear	MLP1	MLP2
Flair-only	20.55	22.42	26.68	21.24	24.01	23.65
1L-ESN-4096	33.71	43.69	41.35	50.33	44.83	43.80
2L-ESN-2048	37.07	43.21	44.55	46.82	50.82	48.28
4L-ESN-1024	36.72	50.37	46.29	45.40	49.77	49.26

As seen in the previous work of [2], adding ESNs to the networks significantly increases the f1 score of the models at our task. This was observed regardless of the *readout* architecture chosen, or the depth of the ESN network. We also saw improvements in increasing the depth of the reservoir, however, they were not as consistent. This indicates that, although depth can be beneficial, it is not sufficient for better performance.

We see a curious development when comparing the recall values. Unexpectedly, recall in the Flair model is higher than in ESN-based ones when the readout layer has no recurrent connections. This happens with all ESN depths and readout layers when using *token level* evaluation. At the entity level, this pattern is broken with some combinations of ESN reservoirs and readout layers outperforming the Flair-only models.

In the precision metrics, however, ESN-based models are better which explains our f1 score results. This could be interpreted as the ESN-based models being more able to learn to distinguish the annotated tokens from the rest. The models without reservoirs seem to be more sensitive, but with lower quality predictions as shown by the precision metrics.

Our attempts at exploiting the relationships between neighboring tokens with a recurrent connection on the readout layer were not successful. In most circumstances, the performance for models with recurrent outputs was lower than feed-forward ones. This may be due to the small

number of tokens with a non-None NER class in the dataset. If they are scarce, the network may not be able to learn these relationships and generalize well. Naturally, this constraint also applies to the whole network, with or without recurrent outputs.

With regards to our experiments with increasing the depths of *readout layer*, we obtained mixed results. Using MLPs and increasing their depth did not lead to consistent improvements in the different *token* level evaluation metrics. Furthermore, when measured with the Top 5 Entity level metrics, the linear *readout layer* significantly outperformed MLPs. This could be explained by entity-level evaluation methods being more sensitive to failures in generalization. A model that fails to generalize to a stock price value would cause the entire *initial price range* entity to be rejected. Meanwhile, in the token-level evaluation the tokens the model predicted correctly would be counted. Given the limited dataset, the possibility of overfitting the MLPs should not be overlooked.

Table 5.6: Top 5 **Entity** Level macro average **f1 score** for different networks on the prospectus corpus.

Model	Non-Recurrent Output Net.			Recurrent Output Net.		
	Linear	MLP1	MLP2	Linear	MLP1	MLP2
Flair-only	17.01	13.97	13.14	7.59	6.81	6.81
1L-ESN-4096	19.33	13.35	13.42	13.97	9.10	8.81
2L-ESN-2048	18.54	15.00	14.62	16.65	14.15	10.84
4L-ESN-1024	21.90	14.49	16.98	18.96	16.16	16.28

Table 5.7: Top 5 **Entity** level macro average **recall** for different networks on the prospectus corpus.

Model	Non-Recurrent Output Net.			Recurrent Output Net.		
	Linear	MLP1	MLP2	Linear	MLP1	MLP2
Flair-only	35.87	30.00	27.22	8.06	8.06	8.06
1L-ESN-4096	39.72	26.11	26.11	20.83	11.39	11.39
2L-ESN-2048	33.06	27.50	27.50	28.61	21.94	15.28
4L-ESN-1024	41.94	24.72	31.25	26.39	20.83	20.83

Table 5.8: Top 5 **Entity** level macro average **precision** for different networks on the prospectus corpus.

Model	Non-Recurrent Output Net.			Recurrent Output Net.		
	Linear	MLP1	MLP2	Linear	MLP1	MLP2
Flair-only	12.06	9.65	9.12	7.36	6.06	6.03
1L-ESN-4096	13.56	9.91	9.96	11.54	8.03	7.78
2L-ESN-2048	13.65	11.29	10.84	13.33	11.51	9.18
4L-ESN-1024	15.82	11.10	12.24	16.12	14.25	14.83

5.4.2 Results by Class

We use six different NER classes in our experiments, which have their particularities. Some of the classes have short entities made up of a couple of words while others of whole sentences or paragraphs (e.g. Competitive Strengths). For this reason, it is interesting to consider them separately and how our models perform with each of them.

We display class-level metrics for the Flair-only model, our main baseline, and two ESN-based models. One of the ESN models is the best performing architecture with 4-layered reservoirs and a single hidden layer MLP as its output network. The other is the best forming one-layered ESN model with the same type of output network as the previous. The baseline is the best model using only Flair embeddings with its two-layered MLP as its output predictor. We present data on the token level evaluation on Tables 5.9, 5.10 and 5.11, and at the entity level on Tables 5.12, 5.13 and 5.14.

One of the entity classes of interest is the **issuer name**. They are quite short, a few words long, and usually have a predictable, standalone placement in documents. An example of this is the prospectus [1] published by Covestro AG on September 15, 2015:

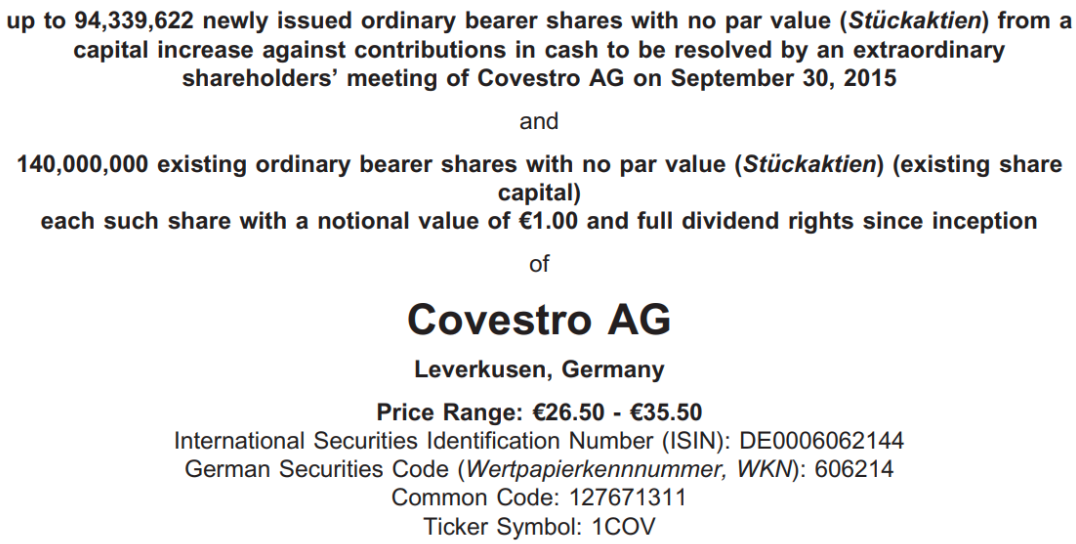


Figure 5.2: Excerpt from prospectus published by Covestro AG

The name of the issuer, **Covestro AG**, is displayed in a manner that highlights it and is separate from the rest of the text. This demonstrates the relationship between paragraphs as mentioned previously. The PDF parser used places the name of the company in a separate paragraph from the rest of the text indicating that it is, in fact, the issuer. The advantage of using an ESN reservoir becomes clear when we compare the token level metrics for *issuer name* on tables 5.10 and 5.11. The F1 score for the issuer name on the former is three times greater than on the latter. This indicates that the ESN reservoir was able to properly summarize the previous text in the document and contextualize that proper noun, allowing the correct prediction of the issuer name. Naturally, the way that the paragraphs were embedded with Flair hinders the

[1] https://www.covestro.com/-/media/covestro/country-sites/global/documents/investrors/covestro/ipo/en/20150918_covestro_prospectus.pdf?la=en

performance of the Flair-only model since it does not take into account the previous paragraphs which is aggravated when the entity is the only text in a paragraph. Therefore, this should be interpreted as an indicator of the capacity of an ESN reservoir to summarize information over long periods.

Also of note is the **Competitive Strength** NER class. This type of information is usually very subjective, and the annotations covered wide swaths of text. At the entity level, our models didn't work at all while performance was better at the token level. This is probably due to the strictness of our evaluation. If all tokens inside an annotated entity were not predicted, we did not consider it a match. Nonetheless, the reservoir-based models also performed better with more than double the f1 score.

Overall, reservoir-based models performed better than their benchmarks in most NER classes when compared by token-level f1 scores. One exception, however, is the initial price range class where performance is practically the same. With regards to the depth of the reservoir, we see large increases in some NER classes between the models with one and four layers.

Table 5.9: **Token** level evaluation metrics for the **1-layered Deep ESN** with an 1-layered MLP non-recurrent readout layer. Support in number of tokens.

NER label	precision	recall	f1 score	support
Accounting Standards	9.09	41.67	14.93	12
Competitive Strength	21.43	15.92	18.27	226
Initial Price Range	63.33	42.22	50.66	45
Investment Bank	45.16	50	47.46	28
Issuer Name	51.72	35.71	42.25	42
Listing Venue	15.15	27.77	19.60	18
macro avg	43.69	44.75	41.87	
micro avg	99.95	99.94	99.94	

Table 5.10: **Token** level evaluation metrics for the **4-layered Deep ESN** with an 1-layered MLP non-recurrent readout layer. Support in number of tokens.

NER label	precision	recall	f1 score	support
Accounting Standards	12.5	33.33	18.18	12
Competitive Strength	34.04	14.16	30	226
Initial Price Range	60	40	48	45
Investment Bank	58.82	35.71	44.44	28
Issuer Name	60	35.71	44.77	42
Listing Venue	27.27	33.33	30	18
macro avg	50.37	41.74	43.62	
micro avg	99.94	99.95	99.95	

Table 5.11: **Token** level evaluation metrics for for the model using only **Flair embeddings** and no ESN reservoir, along with a 2L MLP as a non recurrent readout layer. Support in number of tokens.

NER label	precision	recall	f1 score	support
Accounting Standards	3.4	50	6.38	12
Competitive Strength	8.12	24.77	12.24	226
Initial Price Range	45.46	55.56	50	45
Investment Bank	08.25	32.14	13.13	28
Issuer Name	10.22	21.42	13.84	42
Listing Venue	11.34	61.11	19.13	18
macro avg	26.68	49.27	30.66	
micro avg	99.94	99.86	99.89	

Table 5.12: Top 5 **entity** level evaluation metrics for the **1-layered Deep ESN** with an 1-layered MLP non-recurrent readout layer. Support in number of documents containing one or more target entities

NER label	precision	recall	f1 score	support
Accounting Standards	33.33	60	39.77	5
Competitive Strength	0.0	0.0	0.0	4
Initial Price Range	4.00	20	6.67	5
Investment Bank	12.78	50	20.03	3
Issuer Name	9.33	26.67	13.65	5
Listing Venue	0.0	0.0	0.0	5
macro avg	9.91	26.11	13.36	
micro avg	10.06	25.30	13.35	

Table 5.13: Top 5 **entity** level evaluation metrics for the **4-layered Deep ESN** with an 1-layered MLP non-recurrent readout layer. Support in number of documents containing one or more target entities

NER label	precision	recall	f1 score	support
Accounting Standards	34	50	38.92	5
Competitive Strength	0	0	0	4
Initial Price Range	0	0	0	5
Investment Bank	12.5	25	16.67	3
Issuer Name	8.44	33.33	13.33	5
Listing Venue	11.67	40	18	5
macro avg	11.10	24.72	14.48	
micro avg	11.40	25.61	14.86	

Table 5.14: Top 5 **entity** level evaluation metrics for the model using only **Flair embeddings** and no ESN reservoir, along with a 2L MLP as a non recurrent readout layer. Support in number of documents containing one or more target entities

NER label	precision	recall	f1 score	support
Accounting Standards	26.5	60	35.14	5
Competitive Strength	0	0	0	4
Initial Price Range	7.3	26.66	11.26	5
Investment Bank	8.88	16.66	11.42	3
Issuer Name	0	0	0	5
Listing Venue	12	60	20	5
macro avg	9.12	27.22	13.14	
micro avg	9.47	29.01	13.75	

5.5 Flair Sequence Tagger Benchmark

The Flair paper [14] uses a NER model which they call the Sequence Tagger as an example of application of the Flair embeddings. It is a character-based model, which uses flair embeddings and an LSTM for performing the NER task. Furthermore, it uses a conditional random field (CRF) to model the relationships between the different NER classes. As a benchmark, we trained it using default hyperparameters and stopping criteria, and the same flair embeddings as with our models. It is designed to work on relatively small sequences of text which are independent of each other. That contrasts with our application, where we have large documents and its many paragraphs. Moreover, it does not embed the sentences provided in any specific order which, as discussed previously, is important for our use case. Treating documents as single sequences would preserve context but require more RAM than what was available since the documents are so large (500 pages). As a compromise, we split the documents into their constituent pages and treated them as *sentences* for the Sequence Tagger. This way, even if the pages are presented to the network at a random order, we expect there to be sufficient text to provide a meaningful context. Our dataset contains very sparse labels. Therefore, we also tried rebalancing it by adjusting the learning rates of each class proportionally to their inverse frequency. In that case, however, the CRF model could not be used. The main results for the sequence tagger are delineated in Table 5.15 and 5.16

Table 5.15: Macro average F1-Score for Flair's Sequence Tagger using default settings and compensating for the unbalanced dataset by adjusting the learning rates

	Token Level F1-Score	Entity Level F1-Score
default settings	33.75	7.24
without CRF	17.78	5.07

Table 5.16: Macro average recall for Flair's Sequence Tagger using default settings and compensating for the unbalanced dataset by adjusting the learning rates

	Token Level recall	Entity Level recall
default settings	33.75	7.24
without CRF	17.78	5.07

When compared directly with the ESN models, the sequence tagger performed poorly. However, there are a few caveats that need to be made clear. This model is based on an RNN trained on a dataset for which it was not designed. The dataset contains very sparse labels, and a model with significant capacity such as the sequence tagger most likely requires many more to be able to learn without overfitting. Therefore, a direct comparison is not fair. It does serve to illustrate, though, how distinct models can perform significantly differently in the same datasets.

5.6 Examples of Predictions

We present a few examples of predictions obtained from the best performing model as measured by macro average f1-score on the token level.

5.6.1 Example 1

Figure 5.3: Entities in a prospectus of TNT Express from 2011. Entities are highlighted. Words with a colored background were predicted by the model while the gray ones were not. In the case of Listing Venue, the words highlighted in green were not part of any target entity but partially matched the value of the target for that class shown in gray.

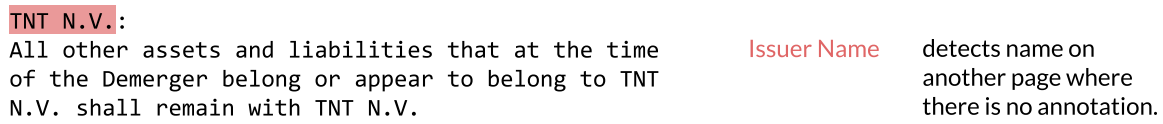


Figure 5.4: On another page the model predicted correctly an abbreviated name of the company. However, this span was not annotated leading to false positives in the token evaluation. If there was an annotation somewhere in the document for "TNT N.V", this would be considered a match.

5.6.2 Example 2

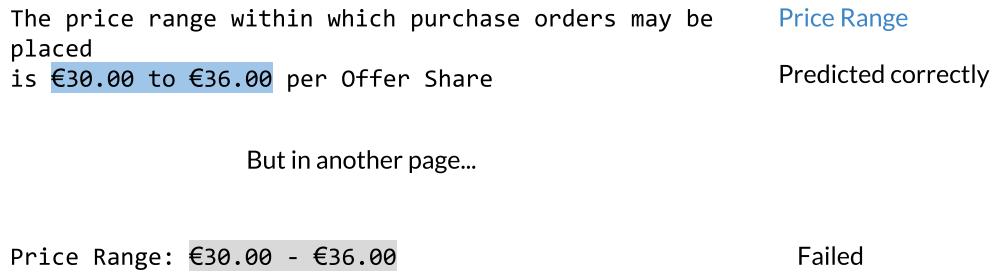


Figure 5.5: Entities in a prospectus of DWS Group from 2018. Entities are highlighted. Multiple *price range* entities are included in the document and we show two of them. In one instance, we correctly predict the entity, but the predictor failed in another target with a different format.

5.6.3 Example 3

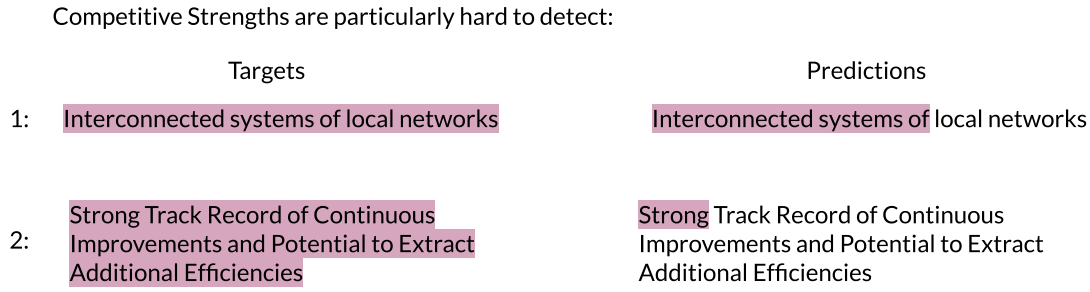


Figure 5.6: Entities from a 2018 prospectus from DWT group (1), and a 2016 one from Innogy (2). These examples show how loosely defined is the competitive strengths class which makes predictions hard. In both examples, we could not correctly predict the entire target entity.

5.7 Result Overview and Future Work

Our experiments brought to light some insights into the use of (deep) echo state networks for the task of extracting information from unstructured financial documents. This approach, as far as we are aware, is a novel strategy for this problem. Of crucial interest to us is the performance of these models with scant and noisy annotations which, although far from ideal, is often the case.

The results obtained show a clear performance improvement when compared with our baseline, though with some caveats. This performance, however, may not be at an acceptable level for every application. That is why we see two further options paths of inquiry. The first would be to further investigate this approach with richer datasets. Although labeling is expensive there may be scenarios where it still possible to obtain moderately larger datasets that would still be too small for traditional neural approaches with RNNs. The second path would be to use these models as online learning agents in a system with user interactions. The system would generate predictions that would be evaluated by the user during the use of a product. This feedback would be used to augment the dataset. One could also extend this idea by adopting reinforcement learning in the readout layer using the user's feedback as the system's reward.

Chapter 6

Conclusion

In this work, we study the problem of extracting information from unstructured financial documents with the requirements of low-cost, adaptability to different domains, and resiliency to smaller datasets. The task was formulated with Named Entity Recognition (NER) and the pieces of information labeled with custom NER classes. Our dataset of 56 prospectuses documents was annotated by domain experts in several NER entity types.

We present an approach using Flair contextual embeddings and deep echo state networks (ESN). The low cost of echo state networks combined with their capabilities of summarizing temporal data were deciding factors in their selection. Futhermore, pre-trained Flair embedders are useful tools for generating vector representations of text that are capable of also taking into account the context of every character. These embeddings are used as inputs to ESN reservoirs which, through the echo state property, contextualize the text over long lengths. Finally, *readout layers* (MLPs or simple linear layers) are used to generate predictions from the ESN states to the NER classes. We also experiment with output-to-input connections at the readout layer to try and explore any inter-label relationships.

In our experiments we compare a baseline model where Flair embeddings were used directly with a readout layer against (deep) ESN architectures with one, two ,and four reservoir layers. For evaluation we used two methods. One evaluating the predictions for each individual token and another at the entity level. The latter is more strict since it requires the entire target entity to be matched correctly.

By F1-Score, our ESN models consistently outperformed the baseline. Unfortunately, our attempt to exploit the inter-label relationships was not successful. Furthermore, the individual class-based analysis gives insight on what types of information our method works better and on which performance is worse. It seems to work better with shorter entities where the information is more concise (such as the issuer name), and worse when the entities are long and subjective.

We also compared our models with the sequence tagger used in an example by Flair's authors [14]. Our models outperformed this benchmark by a wide margin but this does not mean that they are better at the NER task in general. This model is an RNN based one, and it was trained with default parameters and our very sparse dataset. However, it could mean that for cases with limited training data, ESN-based models may perform better.

The sparsity of annotations in our dataset is a challenge, but it also reflects the scenario at which we seek to apply some of these models. Annotations are expensive to make and sometimes

not worth the cost when the productivity gains a model will bring are only moderate. In these circumstances, ESN-based models may be a good solution. One could also use these architectures to generate initial predictions to bootstrap a semi-automated annotations process, maybe even with the final users themselves.

Bibliography

[1] Archana Goyal, Vishal Gupta, and Manish Kumar. Recent named entity recognition and classification techniques: A systematic review. Computer Science Review, 29:21–43, 2018.

[2] Rajkumar Ramamurthy, Robin Stenzel, Rafet Sifa, Anna Ladi, and Christian Bauckhage. Echo state networks for named entity recognition. In Igor V. Tetko, Věra Kůrková, Pavel Karpov, and Fabian Theis, editors, Artificial Neural Networks and Machine Learning – ICANN 2019: Workshop and Special Sessions, pages 110–120, Cham, 2019. Springer International Publishing.

[3] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In Walter Daelemans and Miles Osborne, editors, Proceedings of CoNLL-2003, pages 142–147. Edmonton, Canada, 2003.

[4] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. Neural Networks, 2(5):359–366, 1989.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[6] Michael Mozer. A focused backpropagation algorithm for temporal pattern recognition. Complex Systems, 3, 01 1995.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. Neural Computation, 9(8):1735–1780, 11 1997.

[8] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[9] Herbert Jaeger. The" echo state" approach to analysing and training recurrent neural networks-with an erratum note. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, 148, 01 2001.

[10] Mantas Lukosevicius. A Practical Guide to Applying Echo State Networks, pages 659–686. Springer Berlin Heidelberg, 2012.

[11] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. Deep reservoir computing: A critical experimental analysis. Neurocomputing, 268:87–99, 2017. Advances in artificial neural networks, machine learning and computational intelligence.

[12] Claudio Gallicchio and Alessio Micheli. Deep echo state network (deepesn): A brief survey, 2020.

[13] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, 2014.

[14] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In COLING 2018, 27th International Conference on Computational Linguistics, pages 1638–1649, 2018.

[15] Vikas Yadav and Steven Bethard. A survey on recent advances in named entity recognition from deep learning models. In Proceedings of the 27th International Conference on Computational Linguistics, pages 2145–2158, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics.

[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[17] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information, 2017.

[18] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018), 2018.

[19] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. Lingvisticae Investigationes, 30(1):3–26, 2007.

[20] Tome Eftimov, Barbara Koroušić Seljak, and Peter Korošec. A rule-based named-entity recognition method for knowledge extraction of evidence-based dietary recommendations. PLOS ONE, 12(6):1–32, 06 2017.

[21] Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Frederick Reiss, and Shivakumar Vaithyanathan. Domain adaptation of rule-based annotators for named-entity recognition tasks. In Proceedings of the 2010 conference on empirical methods in natural language processing, pages 1002–1012, 2010.

[22] Philip John Gorinski, Honghan Wu, Claire Grover, Richard Tobin, Conn Talbot, Heather Whalley, Cathie Sudlow, William Whiteley, and Beatrice Alex. Named entity recognition for electronic health records: A comparison of rule-based and machine learning approaches, 2019.

[23] Burr Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. In Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA/BioNLP), pages 107–110, Geneva, Switzerland, August 28th and 29th 2004. COLING.

[24] Onur Kuru, Ozan Arkan Can, and Deniz Yuret. CharNER: Character-level named entity recognition. In Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 911–921, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee.

[25] Horacio Saggion, Adam Funk, Diana Maynard, and Kalina Bontcheva. Ontology-based information extraction for business intelligence. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, The Semantic Web, pages 843–856, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[26] Eduardo Brito, Rafet Sifa, Christian Bauckhage, Rüdiger Loitz, Uwe Lohmeier, and Christin Pünt. A hybrid AI tool to extract key performance indicators from financial reports for benchmarking. In Proceedings of the ACM Symposium on Document Engineering 2019, DocEng '19, pages 1–4. Association for Computing Machinery, 2019.

[27] David Biesner, Rajkumar Ramamurthy, Max Lübbering, Benedikt Fürst, Hisham Ismail, Lars Hillebrand, Anna Ladi, Maren Pielka, Robin Stenzel, Tim Khameneh, Vanessa Krapp, Ilgar Huseynov, Jennifer Schlums, Uwe Stoll, Ulrich Warning, Bernd Kliem, Christian Bauckhage, and Rafet Sifa. Leveraging contextual text representations for anonymizing german financial documents. In AAAI-20 KDF-The AAAI-20 Workshop on Knowledge Discovery from Unstructured Data in Financial Services, 02 2020.

[28] Rafet Sifa, Anna Ladi, Maren Pielka, Rajkumar Ramamurthy, Lars Hillebrand, Birgit Kirsch, David Biesner, Robin Stenzel, Thiago Bell, Max Lübbering, Ulrich Nütten, Christian Bauckhage, Ulrich Warning, Benedikt Fürst, Tim Dilmaghani Khameneh, Daniel Thom, Ilgar Huseynov, Roland Kahlert, Jennifer Schlums, Hisham Ismail, Bernd Kliem, and Rüdiger Loitz. Towards automated auditing with machine learning. In Proceedings of the ACM Symposium on Document Engineering 2019, DocEng '19, pages 1–4. Association for Computing Machinery, 2019.

[29] Tushar Goel, Palak Jain, Ishan Verma, Lipika Dey, and Shubham Paliwal. Mining company sustainability reports to aid financial decision-making. In AAAI-20 KDF-The AAAI-20 Workshop on Knowledge Discovery from Unstructured Data in Financial Services, 2020.

[30] Julio Cesar Salinas Alvarado, Karin Verspoor, and Timothy Baldwin. Domain adaption of named entity recognition to support credit risk assessment. In Proceedings of the Australasian Language Technology Association Workshop 2015, pages 84–90, Parramatta, Australia, December 2015.

[31] Sumam Francis, Jordy Van Landeghem, and Marie-Francine Moens. Transfer learning for named entity recognition in financial and biomedical documents. Information, 10(8), 2019.

[32] Rhys Agombar, Max Luebbering, and Rafet Sifa. A clustering backed deep learning approach for document layout analysis. In Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar Weippl, editors, Machine Learning and Knowledge Extraction, pages 423–430, Cham, 2020. Springer International Publishing.