UCLouvain

École polytechnique de Louvain

Feature embedding techniques for enhancing deep learning models on tabular data

Author: **Patrick Tchoupe Djatcheu**
Supervisor: **John Lee**
Readers: **Edouard Couplet**, **Christophe De Vleeschouwer**, **Pierre Lambert**
Academic year 2023–2024
Master [120] in Data Science: Information technology

Abstract

Many datasets are stored in tabular form, where rows correspond to samples and columns to attributes or features. While deep learning models excel in other data modalities such as images or text, they often underperform when applied to tabular data compared to traditional models like gradient-boosted tree ensembles (e.g., XGBoost). This is partly due to the heterogeneous nature of tabular data features. Each feature may represent a different variable, either categorical or numerical, complicating the learning process for neural network-based models.

An important step in any deep learning pipeline for tabular data is to encode categorical features in a way that is usable by the network and to project all features into a more homogeneous space that is more conducive to learning. A common approach is to use one-hot encoding for categorical features and allow the first few layers of a neural network to learn suitable feature representations in a supervised manner during the training phase.

Self-supervised learning has led to considerable performance gains in the natural language processing world: a decade ago with static embedding techniques like Word2Vec, and more recently with contextual embedding techniques that are key elements in the success of large language models like GPT. The goal of this thesis is to explore such self-supervised feature embedding techniques for tabular data and evaluate their impact on the performance of simple deep learning architectures (e.g., MLP) and more advanced architectures like Transformer-like models for various downstream tasks. The primary focus is on understanding how these feature embedding techniques impact the learning process and how they can be leveraged to improve performance. We have developed a self-supervised embedding technique inspired by Word2Vec, and through our research, we have obtained insights that can aid in the development of better deep learning models for tabular data.

Acknowledgements

First and foremost, I thank the lord for his presence in my life.

I would like to thank my supervisor John LEE for his advice and guidance during the completion of this thesis.

I would like to thank Edouard COUPLET, Christophe DE VLEESCHOUWER and Pierre LAMBERT for agreeing to serve as readers on the jury for this thesis.

Special thanks to Edouard COUPLET, who made himself available for our many meetings and sometimes at late hours to give me his opinion on my work.

I would like to thank my mother Sandrine TCHOUPE for her emotional and financial support throughout my university studies.

I would like to thank my brother Franck CHADJOU for passing on his passion for computing and for his support throughout my studies.

I would like to thank Myrra NGOLLE for her support and attention throughout this year of study.

Finally, I would like to thank all my family and friends who helped me to perfect this work through their proofreading and feedback.

Disclaimer

I acknowledge that I have used artificial intelligence tools to assist me in writing this thesis manuscript, mainly for text translation and syntax correction. Any parts of the code that come from other sources are clearly indicated in the code comments.

Contents

1 Introduction

1.1 Context

In the era of big data, vast amounts of information are stored in tabular format, where each row represents a sample and each column denotes attributes or features. While deep learning models have demonstrated remarkable performance in domains like image and text processing, they often struggle when applied to tabular data compared to traditional models like XGBoost. This performance gap can be attributed, in part, to the heterogeneous nature of tabular data features.

1.2 Problem

Because tabular data is heterogeneous in nature, with each feature potentially representing a different variable, whether categorical or numerical, the learning process for neural networks becomes complex and difficult. Deep learning relies on spatial correlation to learn a new representation of features. The challenge is to encode features efficiently and to project them into a more homogeneous space conducive to learning.

1.3 Motivation

Enhancing the performance of deep learning models via feature embeddings is fundamental, particularly in a context of multi-modal exploitation. The aim of embeddings is to provide an efficient representation of different types of data (text, images, audio, etc.) in a continuous vector space, making it easier to integrate and process different types of data simultaneously. With the correct representation of features, models become more robust and accurate, significantly improving tasks such as emotion recognition, medical diagnosis and recommendation systems, where understanding and exploiting the interactions between different data sources is essential.

1.4 Objectives

The primary objective of this thesis is to explore self-supervised feature embedding techniques tailored specifically for tabular data. By taking advantage of these techniques, we aim to enhance the performance of sim-

ple deep learning architectures, such as multilayer perceptrons (MLPs) and transformer-like architectures across a range of downstream tasks. The focus is on understanding how these features embeddings techniques impact the learning process and how they can be exploited to improve performances.

1.5 Approach

Our approach is to examine the difficulties faced by deep learning models when attempting to represent tabular data, focus on self-supervised embedding techniques. We will implement various embedding techniques and compare the performance of models using these techniques, at the end of which we will draw conclusions from this comparison.

1.6 Contributions

This thesis aims to make both theoretical and practical contributions to the field of deep learning for tabular data. Theoretically, by evaluating the impact of self-supervised feature embedding techniques, we aim to uncover insights that can inform the development of more robust and efficient deep learning models for tabular data. As a practical contribution, we introduce a new embeddings technique for tabular data inspired by a technique for embeddings words in the domain of natural language processing, Word2Vec.

1.7 Roadmap

The remainder of this thesis is structured as follows: In the next three chapters, we provide the necessary background material and report on related work regarding tabular data, deep learning, and embeddings. Chapter 5 enumerates the different embedding techniques implemented to evaluate their impact on deep learning models. In Chapter 6, we detail the setup established for this evaluation, including the datasets used, the deep learning models employed, the tasks on which the models will be evaluated, and the performance evaluation procedure. Chapter 7 reports the results of our evaluations, provides an in-depth analysis of these results, identifies factors that may have influenced the outcomes, suggests improvements for our techniques, and offers insights that can inform the development of more robust and efficient deep learning models for tabular data. Finally, in Chapter 8,

we conclude the work by summarizing the key elements of our study and providing suggestions for future research directions in the field.

2 Tabular data

Tabular data is one of the most common formats for organizing information in various domains such as finance, healthcare, marketing, and social sciences. This data structure is characterized by rows and columns, where each row represents an individual sample, and each column corresponds to a specific attribute or **feature**. The simplicity and versatility of tabular data make it a foundational element in data analysis and machine learning.

For example, in a healthcare dataset, rows might represent patients, and columns might include features such as age, gender, blood pressure, and diagnosis.

Such a tabular structure allows to store and process a significant volume of data, along with a diverse range of data types. This flexibility means that tabular data can possess numerous attributes, depending on the specific information we intend to store and process.

2.1 Types of features

The features of tabular data can be classified into two types:

- Numerical Features: These are quantitative and represent measurable quantities. They can be further divided into:
  - Continuous Features: Such as height, weight, or temperature, which can take any value within a range.
  - Discrete Features: Such as the number of hospital visits or the count of certain events, which take integer values.
- Categorical Features: These are qualitative and represent categories or groups. They can be further divided into:
  - Nominal Features: Such as gender, nationality, or blood type, which have no inherent order.
  - Ordinal Features: Such as education level, satisfaction rating, or socioeconomic status, which have a meaningful order but no fixed interval between categories.

One of the defining characteristics of tabular data is the heterogeneity of its features. Unlike image or text data, where the input features are typically homogeneous (pixels in images or words in text), tabular data can have a diverse set of features with varying data types and scales.

2.2 A clear example

PatientID	Gender	Age	Zip code	Test
55998	M	19	15723	Negative
88557	F	35	15674	Positive
55868	F	35	15674	Positive
44551	M	45	15623	Negative
58524	M	45	15623	Negative
25584	F	61	15633	Negative
58744	F	61	15643	Positive
87524	M	19	15762	Positive
87384	M	19	15762	Negative
17583	F	19	15762	Positive

M: male; F: female

Figure 1: Example of a tabular dataset, showcasing a mix of categorical (PatientID, Gender, Zip code, Test) and numerical (Age) features; Source: https://www.researchgate.net

The provided dataset image reflects the features and heterogeneity of tabular data discussed earlier. It includes both categorical features (Gender, Zip code, Test) and numerical features (Age), highlighting the diverse nature of attributes that deep learning models must handle effectively.

Although there is a PatientID column in the dataset, it is not considered as a feature in the context of machine learning. PatientID serves as a unique identifier for each patient, ensuring that each row represents a distinct individual. However, it does not provide any meaningful information about the relationship between the input features and the target variable, Test.

In machine learning, a common objective is to develop models that can predict a target variable or class, based on a set of input features. In the

context of the provided dataset, the goal is to predict the test result (Positive or Negative) using the other variables (Gender, Age, and Zip code) as predictors. This task falls under the category of supervised learning, specifically binary classification, where the model learns to differentiate between two possible outcomes based on patterns in the data.

Each of the predictive features provides unique information that the model uses to make accurate predictions. An important step in using a deep learning model to perform the task of predicting the Test class will be to give a numerical value to the categorical features. This numerical value must be able to best preserve the basic information contained in the category, which is the objective sought by the embeddings.

2.3 Challenges posed by heterogeneity

The heterogeneity of tabular data features can complicate the learning process for several reasons:

- Different scales and units: Numerical features can have vastly different ranges and units. For instance, age might range from 0 to 100, while income could range from thousands to millions. This disparity can affect the performance of machine learning algorithms.
- Mixed data types: The presence of both numerical and categorical features requires different preprocessing steps. Numerical features might need normalization or standardization, while categorical features might require encoding techniques to be used by an machine learning model.
- Irregular distributions: Features can have different distributions, such as normal, skewed, or multimodal distributions. This irregular distribution can lead to biased models.
- Missing values: Sometimes, not all the data is available in a dataset. The value of an attribute may be missing or unavailable when the data is encoded. This is a common situation with tabular data.
- Absence of inductive bias: Unlike data with a homogeneous structure such as images, where pixels have a clear and predictable spatial relationship (the value of a pixel is correlated with adjacent pixels). There is no inherent structure to tabular data that may be utilized as an inductive bias by a dedicated architecture.

Embeddings aim to overcome these challenges by projecting all the features into a more homogenous space that is more conducive to learning.

To sum up, in this chapter we have presented what tabular data is, how it is structured, the nature of the data it contains and, finally, we have listed the challenges that can arise when using this type of data. In the following chapter, we will discuss the relationship between tabular data and deep learning.

3 Deep Learning with tabular data

In this chapter, we discuss the relationship between tabular data and deep learning. For this purpose, we begin with a brief theoretical reminder of what deep learning is and its concepts. Then, we present the deep learning architectures generally used for tabular data. Finally, we discuss the difficulties of applying deep learning to tabular data and review the literature and research in this area.

3.1 What is Deep Learning ?

Deep learning (DL), a subset of machine learning (as shown in figure 2), involves algorithms inspired by the structure and function of the brain, called artificial neural networks (ANN). Deep learning models are capable of learning from data in a way that mimics human cognition, allowing them to uncover complex patterns and make predictions. Deep learning has revolutionized fields such as computer vision, natural language processing, and speech recognition due to its ability to handle large amounts of data and perform feature extraction automatically.

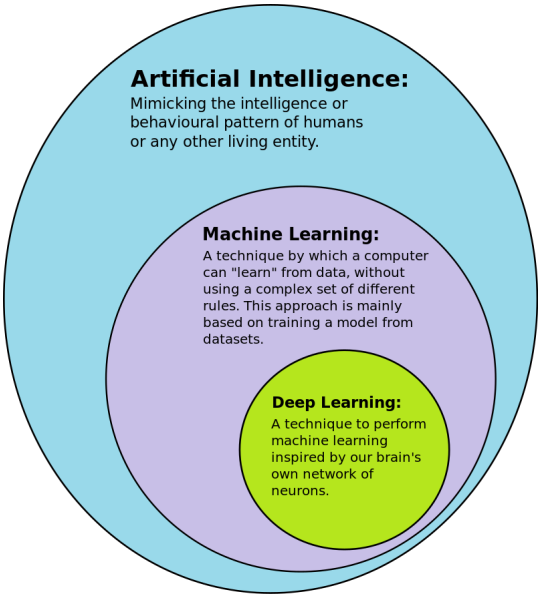


Figure 2: What is deep learning ? Source : Wikipedia

Several terms make up the DL lexical field and we will define those we consider the most important :

- Neurons and Layers: The basic unit of a neural network is a neuron, which receives input, processes it, and produces output. Neurons are organized into layers: input layer, hidden layer, and output layer. The input layer receives the initial data, the hidden layer transforms the data through learned weights, and the output layer produces the final prediction. Figure 3 illustrates the similarities between the structure of a human neuron and an artificial neuron.

- Activation Functions: They determine the output of a neuron given an input or set of inputs. Common activation functions include the sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). Figure 4 shows the formulas for these activation functions and the shape of their curves. These functions introduce non-linearity, allowing neural networks to model complex relationships.

- Loss Functions: They measure the difference between the predicted output and the actual output. The goal is generally to minimise the loss function.

- Optimization Algorithms: These algorithms adjust the weights of the network to minimize the loss function. Gradient descent and its variants (e.g., stochastic gradient descent, Adam) are widely used optimization techniques in deep learning.

- Backpropagation: It is an important algorithm for training neural networks. The gradient of the loss function with respect to each weight is calculated using the chain rule so that the weights can be updated to reduce the loss.

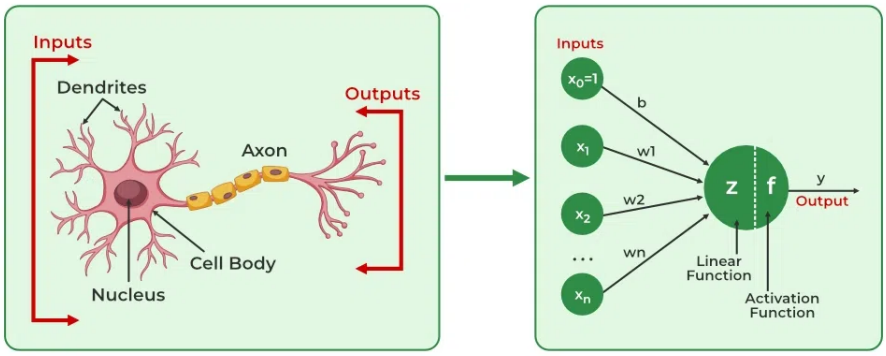


Figure 3: Illustration of the structure of a biological neuron (left) compared to an artificial neuron in a neural network (right). Source: GeeksforGeeks

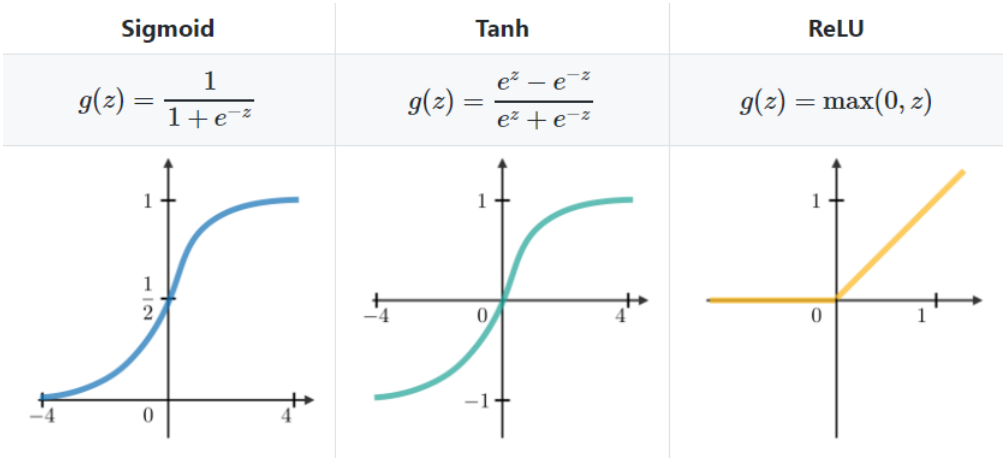


Figure 4: Common activation functions used in neural networks: Sigmoid, Tanh, and ReLU. Source: Medium

Machine learning models that use the terms defined above belong to the family of deep learning models. They are distinguished by their composition, their learning technique and the form of the data they receive. Some of the most common deep learning models are :

- Feedforward Neural Networks (FNNs): The simplest type of artificial neural network, where connections between the nodes do not form cycles. Data flows in one direction, from input to output.
- Convolutional Neural Networks (CNNs): Primarily used in image processing, CNNs are designed to automatically and adaptively learn spatial hierarchies of features from input images.
- Recurrent Neural Networks (RNNs): These networks are designed for sequence data, such as time series or natural language. RNNs have connections that form directed cycles, allowing information to persist.
- Generative Adversarial Networks (GANs): GANs are network architecture composed of two distinct neural networks, a generator and a discriminator, which compete against each other. The generator creates data samples, while the discriminator evaluates them.
- Transformers: Originally designed for natural language processing tasks, transformers use a mechanism called self-attention to process input data, mechanism introduced by Vaswani et al. [22]. They have since been applied to various domains, demonstrating state-of-the-art performance in many tasks.

3.2 Deep Learning Models for Tabular Data

In the domain of tabular data analysis with deep learning, two main types of deep learning architectures are often studied: Multi-Layer Perceptrons (MLPs) and Transformer-based models. Both of these architectures have unique strengths and factors to consider when using them for tabular data.

Multi-Layer Perceptrons (MLPs)

MLPs, also known as feedforward neural networks (FNNs), are among the simplest and most widely used deep learning architectures. They consist of an input layer, multiple hidden layers, and an output layer. Each layer is fully connected to the next, and the model learns by adjusting the weights of these connections through backpropagation.

Among the difficulties of MLPs for tabular data we have: Difficulty in handling categorical features; Sensitivity to feature scaling and normalization; Potential overfitting due to high dimensionality and sparse features.

Tabular data can be composed of several features, which do not always provide information that can help the DL model to carry out a task. These features are called "uninformative features" and it has been shown in [9] that they considerably reduce the learning capabilities of MLP models.

To address these challenges, various preprocessing techniques are employed. Despite these efforts, MLPs often struggle to outperform traditional models like gradient-boosted trees on tabular data.

Transformer-based Models

Transformers, after their excellent performance in the domain of natural language processing, have gained attention for their ability to handle sequential and structured data through a mechanism called self-attention. This mechanism allows transformers to weigh the importance of different parts of the input data dynamically, making them highly effective for complex pattern recognition.

Transformers are composed of two main components: encoders and decoders.

The encoder is responsible for processing the input data and capturing its essential features through multiple layers of self-attention and feedforward neural networks. Each layer in the encoder applies self-attention mechanisms to weigh the importance of different input features, followed by a feedforward network that processes these weighted features.

The decoder, on the other hand, is designed to generate output sequences based on the encoded representations from the encoder. In the context of sequence-to-sequence tasks, such as translation or text generation, the decoder uses the encoder's output along with its own self-attention layers to produce the final output. In the context of tabular data, the decoder can be adapted to generate predictions or classifications based on the encoded feature representations, leveraging the same self-attention mechanisms to ensure accurate and informed outputs.

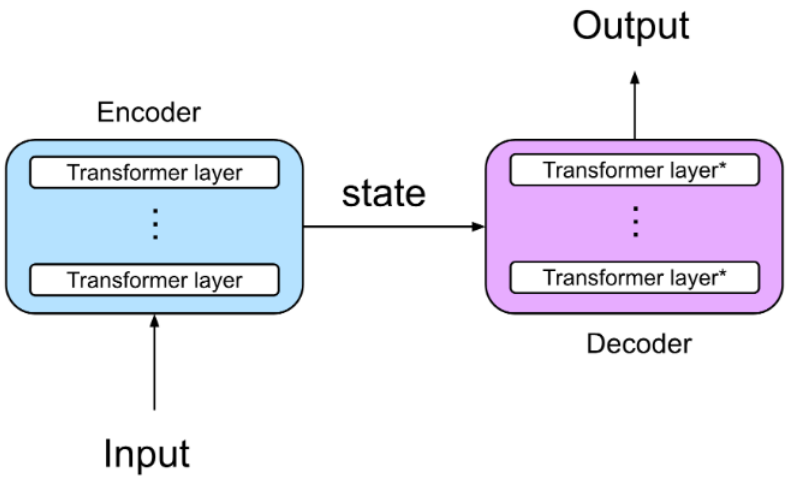


Figure 5: Simplified schematic representation of a transformer model showing the encoder and decoder components. Source: Jithin S L.

Models based on the transformer architecture have been designed for processing tabular data. They are called tabular-specific transformers architectures and examples include SAINT from [18], TabNet from [2], TabTransformer from [13].

Challenges with Transformers: High computational cost and resource requirements; Need for extensive training data to achieve optimal performance;

3.3 The unconquered castle

Deep learning has shown remarkable success in fields such as computer vision and natural language processing. However, when it comes to tabular data, deep learning models often struggle to outperform traditional machine learning methods such as gradient-boosted trees. Kadra et al. refer to tabular data as the last "**unconquered castle**" for DL in [15].

The questions that then arise are:

1. Why don't DL models perform as well on tabular data as they do on other types of data?
2. What are the characteristics of tabular data that favour decision tree models and disfavour deep learning models?

3. Can't we simply use tree-based models to process tabular data?
4. How can we adapt deep learning models to perform better on tabular data?
5. How can tabular data be modified so that deep learning models perform better?

Over the years, a number of researchers have attempted to answer these questions.

One of the most notable papers that addressed questions 1 and 2 is [9]. In this paper, the authors attempt to answer these questions by investigating the learning processes of tree-based models and neural networks (NN).

Following their experiments, the authors proposed three key findings: First, neural networks are biased towards overly smooth solutions. Due to their differentiable nature, neural networks tend to smooth out irregularities, making it more difficult for them to learn these functions effectively without significant architectural adjustments. Second, tabular data often consist of numerous features, some of which are uninformative. These uninformative features affect deep learning models much more in their learning process compared to tree-based models. Third, MLPs are rotationally invariant, meaning they do not change their learning behavior when features are rotated, which makes it difficult to eliminate uninformative features. The fact that very different types of embedding appear to improve performance suggests that the presence of an embedding that breaks this invariance is a key element in these improvements. In the comprehensive survey [3], the authors provide an overview of the current state of deep learning models applied to tabular data. They discuss the inherent challenges, such as the heterogeneous nature of features, and review various architectural adaptations and preprocessing techniques designed to address these challenges.

For question 3, tree-based models use decision rules to separate data based on features. This decision-making process is inherently non-differentiable, meaning that we cannot compute derivatives with respect to the model's parameters. In addition to being non-differentiable, decision trees do not produce representations of the input data, which complicates their integration into joint optimization processes or multimodal pipelines.

For question 4, many architectures have been implemented for this purpose: The paper [8] introduces TabNet, a deep learning architecture specifi-

cally designed for tabular data. TabNet utilizes sequential attention to process features in a stepwise manner, focusing on the most relevant features at each step. AutoInt, introduced in the paper [19], leverages self-attention mechanisms to automatically learn feature interactions in tabular data. The creation of so-called hybrid models which benefit from the flexibility of NN while retaining the inductive biases of other algorithms such as tree-based models ([16], [17], [1], [21], [18]).

For question 5, some techniques for data encoding to make tabular data better suited for deep learning have emerged. The paper [7] investigates how numerical features can be effectively embedded to improve the performance of deep learning models on tabular data. Other techniques were presented in [11] and [23].

From all these researches, it emerges that conquering the castle of tabular data with deep learning can be approached in two ways: adapt DL models to be more suitable for the heterogeneous nature of tabular data, or adapt the representation of tabular data features to allow DL models to better learn and capture the relationships within them.

Of all the models published to provide an answer to question 4, none appears clearly to be the best to use and the aspects of the deep learning models to be adapted remain ambiguous for the moment. In this thesis, we will attempt to answer question 5 : "How can tabular data be modified so that deep learning models perform better?" and look at self-supervised embedding techniques.

In the next chapter, we will discuss embeddings, we will list the different types of embeddings techniques, talk about their utility, and provide a detailed examination of an embedding algorithm that has significantly enhanced the performance of models in the field of Natural Language Processing (NLP): Word2Vec.

4 Embeddings

In this chapter, we explore the concept of embedding and we list the types of embeddings. In addition, we will take a detailed look at the Word2Vec algorithm.

In machine learning, embedding is one of the main technique for representing categorical or discrete data in a continuous vector space. Embedding aims to capture intrinsic relationships and semantics between different categories, allowing the model to learn more effectively.

The core of embedding is a mapping from a discrete space to a continuous vector space, where each category or entity is represented by a dense or sparse vector of real numbers. This transformation allows the model to learn meaningful representations of categorical variables, which can then be used for various tasks such as classification, regression, or recommendation systems.

In the context of tabular data features embeddings, a distinction is made between supervised and unsupervised or self-supervised embedding processes. In a dataset for a y label prediction task, supervised processes generate embeddings based on the labels of each sample. This thesis explores techniques that do not use y labels to generate embeddings, self-supervised process.

There are many embeddings techniques in machine learning, each with its own strengths and weaknesses. Hancock et al. in their article "A Survey of Neural Networks for Categorical Data" [11] propose to define the boundaries between different embeddings techniques based on several characteristics, including computational complexity. The article only discusses embedding techniques for categorical data, but we found that embeddings techniques for numerical data can also be divided into these groups of techniques. Let us present the group of embeddings techniques.

4.1 Type of embeddings techniques

4.1.1 Determined techniques

Hancock et al. [11] define determined techniques as embedding techniques for which the encoding values will always be the same at each execution. "Given the same dataset, a determined encoding will produce the same encoding ev-

ery time a practitioner employs it." They are also the least computationally complex and generally form the basis of many other more complex techniques.

The most commonly used encoding determined techniques include:

- One-hot encoding

  This is the most widely used encoding technique in machine learning, requiring very little set-up to use and working only on categorical features. It creates vectors the size of all the values of a categorical variable and assigns 0s to all positions except the one where the current value is located, which takes a 1.

- Quantile Transformer

  This technique transforms numerical values so that they follow a specified distribution (uniform or normal). Each value is mapped to a quantile following the target distribution.

Determined techniques have the advantage of being easy to compute but, due to their simplicity, the representations provided do not generally carry much information that can help deep learning models learn from the data. And sometimes, with an encoding like One-Hot Encoding, the dataset can be significantly expanded, which can introduce noise into the model learning process.

4.1.2 Algorithmic techniques

These techniques require extensive computations. Hancock et al. [11] stated that "When researchers put a significant amount of work in designing a process that converts qualitative data to numerical form, we call this algorithmic data representation." We suggest applying the same definition to techniques that convert quantitative data to another numerical form.

Algorithmic techniques can be considered deterministic, but they differ from determined techniques in that they are more complex in terms of execution time. They have hyperparameters whose values can modify the embedding result.

Examples of algorithmic techniques are Latent Dirichlet Allocation (LDA), a generative statistical model that categorizes text into different topics. Golinko

et al. introduced Generalized Feature Embedding Learning (GEL) in their study "Generalized Feature Embedding for Supervised, Unsupervised, and Online Learning Tasks" [6], which systematically converts datasets into alternative representations suitable for diverse machine learning algorithms. Additionally, Han et al., in their paper "EDLT: Enabling Deep Learning for Generic Data Classification" [10], describe EDLT encoding, which transforms tabular data into matrices for use in convolutional neural networks. These examples highlight the adaptability and computational demands of algorithmic embeddings techniques.

Algorithmic techniques are able to identify complex relationships and hidden structures in the data that deterministic techniques cannot. However, these methods often require significant computing resources, making them quite demanding. According to Hancock et al. algorithmic techniques also tend to be more domain-specific, as they can exploit the unique characteristics of data in specific domains.

4.1.3 Automatic techniques

Automatic techniques are those that produce data embeddings during the learning process of a neural network. These techniques are interesting because of their versatility, as they are more general than determined or algorithmic methods. They usually use a determined technique as the first representation of the data, especially if it contains numerical and categorical variables as with tabular data, since neural networks only work with numbers.

Automatic techniques allow neural networks to use the embeddings computed by another neural network to improve their learning process. Optimising neural network parameters makes the embedding process time-consuming.

The general aspect of automatic techniques means that they can be adapted to different types of data. A striking example is Word2Vec in natural language processing. In the study 'Using word embedding technique to efficiently represent protein sequences for identifying substrate specificities of transporters' [12], Nguyen et al. applied the principles of Word2Vec to protein sequences. They treated amino acids as letters and proteins as words, using their protein database as a corpus. This illustrates an interesting potential of automatic embeddings techniques. It may be possible to adapt an existing automatic algorithm in a different domain to create an embedding tailored to the machine learning task in question.

Below we provide a table summarising all that has just been said about the types of techniques above.

Types of embedding technique	**Characteristics**
Determined embeddings	• Always produce the same encoding values • Require minimal computational resources • One-hot, Label encoding, Binary encoding ...
Algorithmic embeddings	• Encoding values vary with different hyperparameter settings • More complex in terms of running time than determined techniques • LDA, GEL, EDLT.
Automatic embeddings	• Find embeddings during neural network training • Complex computations and time-consuming • Word2Vec

Table 1: Table summarising the types of embedding techniques and their characteristics.

In the next section, we will explain the implementation details of the Word2Vec algorithm as we will draw inspiration from it to define our automatic technique.

4.2 Details on Word2Vec

Word2Vec is a widely used technique in natural language processing (NLP) for representing words as dense vectors in a continuous vector space. Developed by Tomas Mikolov and his team at Google in 2013, Word2Vec has significantly impacted NLP tasks such as word similarity, document classification, and machine translation.

4.2.1 Concepts

Word2Vec uses the distributed representation hypothesis, which suggests that words with similar meanings tend to appear in similar contexts. By analyzing the contexts in which words occur in a large corpus of text, Word2Vec generates vector embeddings that capture meaningful information about word semantics.

Word2Vec has two main architectures: Continuous Bag of Words (CBOW) and Skip-gram.

The Continuous Bag of Words (CBOW) model predicts a target word from its surrounding context within a fixed window size. Unlike traditional models that focus on predicting the next word, CBOW creates efficient word representations based on the surrounding context. It trains a neural network to minimize the cross-entropy loss between predicted and actual target words. The input layer uses one-hot encoded vectors for context words, which are then projected onto a hidden layer. This hidden layer combines the context vectors to generate a fixed-size embedding, which is passed through a softmax layer to produce a probability distribution. Through iterative backpropagation, CBOW learns embeddings that capture semantic similarities based on how words are used in context.

The Skip-gram model, known for its simplicity and efficiency, takes a different approach from CBOW to learn word embeddings. Instead of pre-

dicting a target word from its context, Skip-gram predicts context words given a target word, aiming to capture syntactic and semantic relationships based on their proximity. Using a neural network, Skip-gram minimizes the negative log-likelihood loss between predicted and actual context words. It uses one-hot encoded vectors for target words at the input layer, which are projected onto a hidden layer to create embeddings. Through iterative stochastic gradient descent, Skip-gram effectively learns embeddings that encode rich semantic information in a continuous vector space.

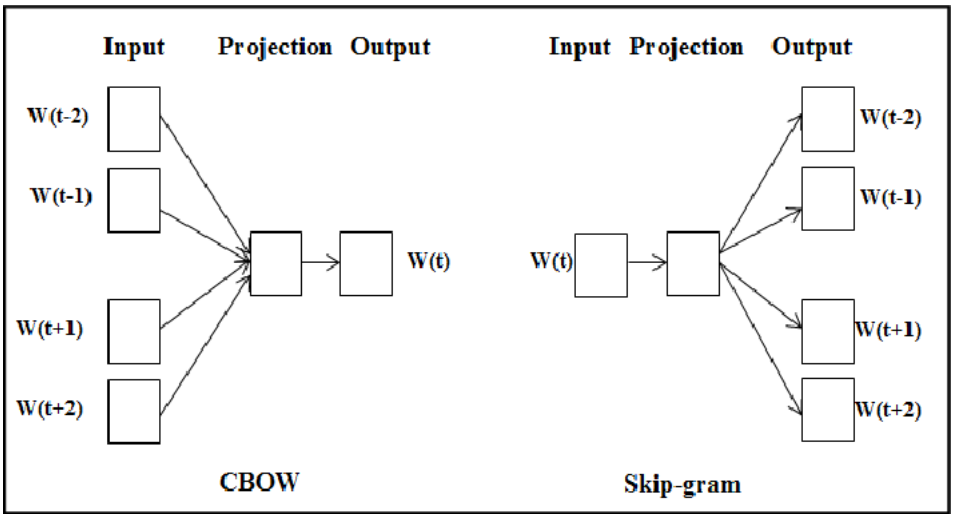


Figure 6: The architecture of CBOW and Skip-gram algorithms. Source: ResearchGate.

4.2.2 Calculations

In this section, we will explain a method for computing embeddings known as skip-gram with negative sampling (SGNS) following the great book from [14].

Word2Vec embeddings produced static embeddings because the method learns a single fixed embedding for each word in the vocabulary. The intuition of Word2Vec is to train a classifier on a binary prediction task:
"**Is word w_1 likely to show up near w_2?**"

As this is an automatic embedding technique, the aim is to use the weights of the classifier after training as word embeddings. The input text serves as implicitly supervised training data and allows the classifier to answer the prediction question mentioned above.

The skip-gram approach operates on the principle of using a target word and its neighboring context words as positive examples, while randomly selecting other words from the vocabulary as negative samples. A classifier is then trained using logistic regression to distinguish between positive and negative examples, with the resulting learned weights serving as the word embeddings.

Given a tuple (w, c) of a target word w and a candidate context word c, the classifier returns the probability that c is a real context word: $P(+|w, c)$.

The probability that word c is not a real context word for w is just equal to $P(-|w,c) = 1 - P(+|w,c)$

The probability $P(+|w, c)$ is calculated based on the similarity of the embeddings: a word is likely to appear near the target if its embedding vector is similar to the target embedding. Similarity is then evaluated using a dot product of the vectors representing the words.

To convert the dot product into a probability, we use the sigmoid function $\sigma(x)$, as the word vectors can contain negative numbers and the dot product can range from $-\infty$ to $+\infty$. The probability is calculated as follows:

$$P(+|\mathrm{w,c}) = \sigma(\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1+\exp(-\mathbf{c}.\mathbf{w})}$$

Skip-gram simplifies the model by assuming that all context words are independent of each other. This assumption allows us to calculate the overall probability by multiplying the individual probabilities of each context word:

$$P(+|w,c_{1:L}) = \prod_{i=1}^{L} \sigma(\mathbf{c}_i.\mathbf{w})$$
$$\log P(+|w,c_{1:L}) = \sum_{i=1}^{L} log\sigma(\mathbf{c}_i.\mathbf{w})$$

With L being the size of the context window.

Skip-gram maintains two embeddings for each word: one for when the word is used as a target and another for when it is used as a context word. Therefore, the parameters to be learned are two matrices, $\mathbf{W}$ and $\mathbf{C}$, each containing embeddings for all $|V|$ words in the vocabulary V.

Next, we focus on learning these embeddings, which is the primary objective of training this classifier.

4.2.3 The learning of the embeddings

The skip-gram embeddings learning algorithm requires a text corpus and a specified vocabulary size N as input. Initially, it assigns a random embedding vector to each of the N vocabulary words. Then, it iteratively adjusts the embedding of each word w to make it more similar to the embeddings of nearby words in the text and less similar to the embeddings of words that are not nearby.

In order to train a binary classifier, negative examples are also necessary. The skip-gram with negative sampling (SGNS) approach utilizes a higher number of negative examples compared to positive examples. The ratio between them is determined by a parameter k. For every training instance (w, c_{pos}), k negative samples are generated. Each negative sample consists of the target word w and a 'noise word' c_{neg}'.

Given the set of positive and negative training instances, along with an initial set of embeddings, the learning algorithm aims to adjust these embeddings to maximize the similarity between target word and context word pairs (w, c_{pos}) from the positive examples, while minimizing the similarity between target word and noise word pairs (w, c_{neg}) from the negative examples.

Considering a word/context pair (w, c_{pos}) along with its k noise words $c_{neg_1}, .., c_{neg_k}$, we can formulate these objectives using the following loss function L, which needs to be minimized (indicated by the negative sign). The first term ensures that the classifier assigns a high probability to the real context word c_{pos} being a neighbor of w, while the second term ensures that each noise word c_{neg_i} is assigned a high probability of not being a neighbor. These terms are multiplied due to the assumption of independence:

$$\begin{aligned} \mathrm{L}_{CE} &= \text{-log } [P(+ \mid w, c_{pos}) \prod_{i=1}^{k} P(- \mid w, c_{neg_i})] \\ &= -\left[\log \sigma (c_{\text{pos}} \cdot w) + \sum_{i=1}^{k} \log \sigma (-c_{neg_i} \cdot w) \right] \end{aligned} \tag{1}$$

The objective is to maximize the dot product between a word and its actual context words, and to minimize the dot products between the word with the k negatively sampled non-neighbor words.

We minimize this loss function using stochastic gradient descent. To obtain the gradient, we need to compute the derivative of the loss function

with respect to the various embeddings. The derivatives are as follows:

$$\begin{aligned}
\frac{\partial L_{CE}}{\partial c_{pos}} &= \left[\sigma\left(\mathbf{c}_{pos} \cdot \mathbf{w}\right) - 1\right] \mathbf{w} \\
\frac{\partial L_{CE}}{\partial c_{neg}} &= \left[\sigma\left(\mathbf{c}_{neg} \cdot \mathbf{w}\right)\right] \mathbf{w} \\
\frac{\partial L_{CE}}{\partial w} &= \left[\sigma\left(\mathbf{c}_{pos} \cdot \mathbf{w}\right) - 1\right] \mathbf{c}_{pos} + \sum_{i=1}^{k} \left[\sigma\left(\mathbf{c}_{neg_i} \cdot \mathbf{w}\right)\right] \mathbf{c}_{neg_i}
\end{aligned} \tag{2}$$

The update equations at each iteration step in stochastic gradient descent are:

$$\begin{aligned}
\mathbf{c}_{pos}^{t+1} &= \mathbf{c}_{\text{pos}}^{t} - \eta \left[\boldsymbol{\sigma}\left(\mathbf{c}_{\text{pos}}^{t} \cdot \mathbf{w}^{t}\right) - 1\right] \mathbf{w}^{t} \\
\mathbf{c}_{\text{neg}}^{t+1} &= \mathbf{c}_{\text{neg}}^{t} - \eta \left[\sigma\left(\mathbf{c}_{\text{neg}}^{t} \cdot \mathbf{w}^{t}\right)\right] \mathbf{w}^{t} \\
\mathbf{w}^{t+1} &= \mathbf{w}^{t} - \eta \left[\left[\sigma\left(\mathbf{c}_{\text{pos}}^{t} \cdot \mathbf{w}^{t}\right) - 1\right] \mathbf{c}_{pos}^{t} + \sum_{i=1}^{k} \left[\sigma\left(\mathbf{c}_{\text{neg}\ i}^{t} \cdot \mathbf{w}^{t}\right)\right] \mathbf{c}_{neg_i}^{t}\right]
\end{aligned} \tag{3}$$

The learning algorithm begins with randomly initialized $\mathbf{W}$ and $\mathbf{C}$ matrices. It then iterates through the training corpus, using equation in (3) to adjust $\mathbf{W}$ and $\mathbf{C}$ in order to minimize the loss.

Each word i is associated with two separate embeddings: the target embedding w_i, stored in the target matrix $\mathbf{W}$, and the context embedding c_i, stored in the context matrix $\mathbf{C}$.

The final common representation of each word i is the vector w_i. We can also combine embeddings by adding them together, representing the word i by the vector $w_i + c_i$.

Now that we have explained all the theoretical concepts that make up this thesis, we will present the embedding techniques implemented to evaluate their impact on the performance of deep learning models.

5 Embeddings techniques on tabular data

In this chapter, we present the three embedding techniques used to evaluate their impact on deep learning models when applied to tabular data. It was suggested in section 3.3 that effective embedding techniques can transform the heterogeneous features of tabular data into continuous vector representations, thereby enhancing the learning capabilities and generalization potential of deep learning models.

Our three used techniques are : a baseline embeddings, numerical embeddings PLE, and Feature2Vec. The baseline embeddings serves as a point of reference, while numerical embeddings aim to represent numerical features in a dense vector space. Feature2Vec, inspired by Word2Vec, seeks to generate meaningful embeddings for both numerical and categorical features by using contextual relationships within the data.

This chapter focuses on presenting the formalization of these techniques. The study to highlight the strengths and limitations of each technique will be conducted further, after we report the results of our experiments and analyses.

5.1 Baseline embeddings

This technique will serve as the baseline for evaluating the performance of our models, it employs a systematic approach to encoding categorical features and normalizing numerical features. It can be defined as determined techniques.

Encoding Categorical Features

The baseline technique applies one-hot encoding to categorical features. One-hot encoding transforms each categorical value into a binary vector where each element corresponds to a category. The dimensionality of the transformed space for each feature is equal to the number of unique categories in the feature.

Encoding Numerical Features

For numerical features, the baseline method employs normalization via the Quantile Transformer of scikit-learn. This transformation adjusts the feature values to follow a normal distribution, which is a common requirement for many deep learning models. By transforming numerical values to a

common scale, the Quantile Transformer enables a more stable and effective training process.

5.2 Numerical embeddings

This technique is exactly the same as the one introduced by [7] in their paper. The technique explores how changing the representation of numerical features can improve the learning capabilities of deep learning models for tabular data.

The inspiration comes from classical machine learning techniques, in particular the one-hot encoding method widely used to represent categorical features in tabular data and tokens in natural language processing (NLP). While this method is effective for discrete entities, it is not suitable for numerical features due to its sparse nature and lack of expressiveness. Therefore, a continuous alternative to one-hot encoding for numerical features was devised.

Let $X = \{(\mathbf{x^{(j)}}, y^{(j)})\}_{j=1}^{n}$ be the dataset for a given supervised learning problem on tabular data, with $y^j \in \mathbb{Y}$ representing the object's label and $\mathbf{x}^j = \left(x^{j(num)}, x^{j(cat)}\right) \in \mathbb{X}$ representing the object's features (numerical and categorical).

$x_i^{j(num)}$ denotes the i-th numerical feature of the j-th object. We obtain $z_i = f_i(x_i^{(num)}) \in \mathbb{R}^{d_i}$ where $f_i(x)$ is the embedding function for the i-th numerical feature, z_i is the embedding of the i-th numerical feature and d_i is the dimensionality of the embedding.

Learnings of the embeddings

The range of values of the i-th numerical feature is divided into a disjoint set of T^i intervals $B_1^i, ..., B_T^i$, called bins: $B_t^i = [b_{t-1}^i, b_t^i)$.

To construct the bins, the ranges of values are divided according to uniformly chosen empirical quantiles of the distributions of the corresponding numerical feature.
Specifically, for the ith feature: $b_t = Q_{\frac{t}{T}}(x_i^{j(num)})$, where Q is the empirical quantile function.

Once the bins have been determined, the embeddings are formed using Piecewise linear encoding (PLE). PLE produces embeddings of the size of

the number of intervals T and for the numerical value x, it sets 1s in the intervals preceding the interval where the value x lies, and 0s for the intervals following the interval where the value x lies. For the interval where x lies, the value set is the ratio of the difference between x and the lower limit of this interval to the difference between the two limits of the interval. The encoding scheme is defined as follows:

$$\mathbf{PLE}(x) = [e_1, \ldots, e_T] \in \mathbb{R}^T$$

$$e_t = \begin{cases} 0, & \text{if } x < b_{t-1} \text{ and } t > 1 \\ 1, & \text{if } x \geq b_t \text{ and } t < T \\ \frac{x - b_{t-1}}{b_t - b_{t-1}}, & \text{otherwise} \end{cases} \tag{4}$$

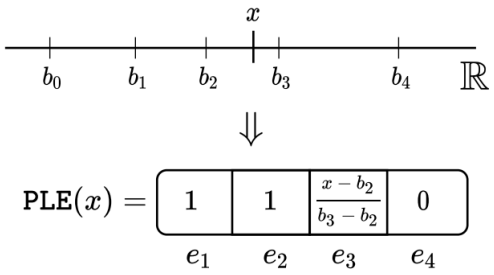


Figure 7: The piecewise linear encoding (PLE) in action for T = 4. Source : Gorishniy et al.

This technique only applies to numerical characteristics. Given the complexity of calculating embeddings and the variation in the final embeddings as a function of the value T, the number of intervals, PLE can be categorised as a algorithmic technique. For the encoding of categorical features, we used the determined One-hot encoding technique.

5.3 Feature2vec

The Feature2Vec embedding technique is an automatic technique for transforming tabular data features into meaningful vector representations that can be effectively used by deep learning models. This technique draws inspiration from the Word2Vec model and adapts it for use with tabular data.

To apply the principles of Word2Vec to tabular data, several transformations were necessary. Because Word2Vec was originally designed for words within a text corpus, Feature2Vec required adaptation to handle the structure and nature of tabular data. We needed to conceptualize and represent tabular data in a way that mimicked the corpus used in natural language processing.

The first step was to transform the tabular data features into word forms. In our case, this involved representing the entire dataset (corpus) by categories (words).

As the categorical variables were already in the form of categories, this first step did not apply to them. For the numerical variables, a discretization was carried out. In the end, the dataset is represented by a set of n_{cat} categories. We then use a one-hot encoding for these categories. These transformations ensure that all features, regardless of type, are represented uniformly.

Let X_i be a numerical variable, after applying discretization for k bins, X_i now has k categories. Let X_j be a categorical variable with l categories. After encoding categories, the variable X_i is made up of k one-hot vectors β_1,...,β_k and the variable X_j is made up of l one-hot vectors δ_1,...,δ_l. The dimension of the vectors is equal to the total number of categories n_{cat} across all attributes.

Word2Vec provides static embeddings for all the words in the vocabulary, which means that a word that appears several times in different places in the text will have the same embeddings. In our case of tabular data, let's consider two numerical variables X_1 and X_2, a numerical value x_i in X_1 cannot have the same embedding as the same numerical value x_i in X_2 because the feature described by X_1 is not the same as that described by X_2. So, the numerical values may not have the same meaning and the two must therefore be considered as two distinct vocabulary values. We wanted to reflect this difference in our representations.

Let D_0 be the state of the dataset after data categorisation, we build a table $D = D_0^T D_0$. D contains all the contingency tables of the categories taken in pairs. D can be divided in two blocks: The block $D_{i,i}$ contains on its diagonal the frequencies of apparition of the category in the original table D_0. The block $D_{i,j}$ gives the conditional frequencies of the categories of feature X_i, given the categories of feature X_j.

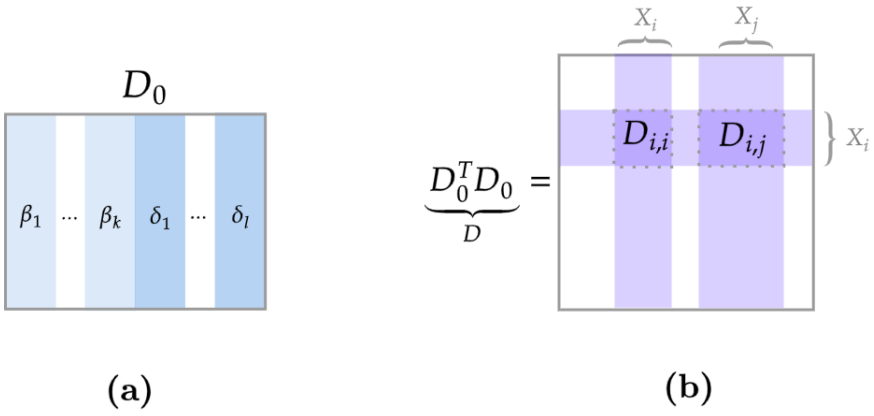


Figure 8: (a) Representation of the state of the dataset after categorisation and (b) Representation of the D contingency table. Source : [4]

For the Word2vec architecture, we utilized the skip-gram model with negative sampling loss (SGNS). We need training pairs (t, c) composed of a target category and one of its context category.

The procedure to sample the training pairs is:

1. Randomly select a target attribute from all available attributes with equal probability. Let's assume we have selected X_i.
2. Create a probability mass function (PMF) for the categories of X_i, where each category's probability mass is the logarithm of its frequency. The frequencies are provided on the diagonal $D_{i,i}$.
3. Randomly select a target category based on the constructed PMF.
4. Randomly select a context attribute from all available attributes with equal probability.

5. Create a conditional PMF for the other categories, where each category's probability mass is the logarithm of its frequency given the selected target category. The conditional frequencies are provided by $D_{i,j}$.

6. Randomly select a positive context category based on the conditional PMF. This is a category for which the probability mass in the conditional PMF is different of 0.

7. Randomly select a negative context category. This is a category for which the probability mass in the conditional PMF is 0. We still have the k factor for the number of negative pairs to generate.

In summary, the process of generating target-context pairs for training the Feature2Vec classifier involves several key steps. First, the sampler randomly selects a feature, analogous to a word in Word2Vec, and identifies its category indices to represent the "target category" ($target_{cat}$). The context for this target category is formed by the other categories present in the same sample, mimicking the idea of surrounding words in a sentence in NLP, and these are identified as the "context categories" ($context_{cat}$).

To effectively train the classifier, negative samples are also required. These negative samples are categories that do not co-occur with the target category within the same sample. We randomly selects a fixed number of negative categories ($negative_{cat}$) for each target-category pair, similar to the negative sampling technique used in Word2Vec.

For each iteration, we form a batch consisting of target categories, context categories, and negative categories, which are then used to train the embedding model. The model is trained to maximize the similarity between the target and its context categories while minimizing the similarity between the target and the negative categories, thereby distinguishing relevant contexts from irrelevant ones.

The procedure remains similar to Word2vec for updating the target $\mathbf{W}$ and context matrix $\mathbf{C}$ weights.

The embeddings retained for the features at the end of our Feature2Vec are those from the target matrix $\mathbf{W}$.

6 Experimental setup

6.1 Datasets

We selected 8 datasets to conduct our studies. The datasets are taken from the work carried out in [9] by Grinsztajn et al. In their work Grinsztajn et al. used 45 datasets, but we have chosen 8 datasets from these 45 which correctly represent the different forms of datasets that can be encountered when working with tabular data.

Thus, we have datasets with a mixture of numerical and categorical features, others consisting solely of numerical features; datasets for regression and classification tasks; large and medium-sized datasets. These datasets have no missing values. ↑ refers to a classification task; ↓ refers to a regression task.

To easily reference datasets in the next sections, we define aliases for each dataset.

Name	**Alias**	**Task**	**Size**	**# Features**	**# Cat**
Bank-marketing	BM	↑	10578	7	0
Bike_Sharing_Demand	BSD	↓	17379	11	5
California	CAL	↓	20640	8	0
Electricity	ELEC	↑	38474	8	1
House_sales	HS	↓	21613	17	2
KDDCup09_upselling	KDD	↑	5032	45	11
Compass	COM	↑	16644	17	9
Wine_quality	WQ	↓	6497	11	0

Table 2: Datasets informations

6.2 Implementation

Here, we present the implementation carried out within the scope of this thesis, including the DL models and parameters used, as well as the specifics of the embedding techniques.

6.2.1 General formalisation

This formalisation follows, apart from a few elements, what was discussed in the article [4].
Consider a dataset $X = \{(\mathbf{x^{(i)}}, y^{(i)})\}_{i=1}^{n}$ where $\mathbf{x^{(i)}} = (x_1^{(i)}, x_2^{(i)}, ..., x_m^{(i)}) \in \mathbb{X}$ represents the features of the ith sample and $y^{(i)} \in \mathbb{Y}$ denotes the sample label. We consider two types of tasks: binary classification when $\mathbb{Y} = \{$ 0,1 $\}$ and regression when $\mathbb{Y} = \mathbb{R}$.

The task of label prediction is formulated as an encoding decoding problem and we propose the following general model for our experiment:

$$\mathbf{y} = \rho(\text{pool}(\phi(f(\mathbf{x}))))$$

where $f(\cdot)$ represents the pre-encoder, $\phi(\cdot)$ is a neural network that represents the encoder and $\rho(\cdot)$ is a neural network that represents the decoder.

Pre-encoder : It is the embedding technique applied to the data. The different techniques used were presented in Chapter 5.

Depending on the deep learning model used, which we will present in the following sections, we define two levels of feature representations:

- Sample-level representation : all embeddings $f_j(x_j)$ are typically concatenated into a single sample representation.

  We define $f : \mathbb{X} \to \mathbb{R}^d$ as $f(\mathbf{x}) = \text{concat}(f_1(x_1), f_2(x_2), ..., f_m(x_m))$.

- Feature-level representation : The embedding of each feature x_j is maintained separately in the common space $\mathbb{R}^d$.

  We define $f : \mathbb{X} \to \mathbb{R}^{d \times m}$ as $f(\mathbf{x}) = (f_1(x_1), f_2(x_2), ..., f_m(x_m))$.

  With d being the final size of the embeddings and m the number of features.

6.2.2 Models

Section 3.2 presented the two types of model mainly used in deep learning for tabular data. For this thesis, we will implement these two models in a similar way, respecting the formalism defined in section 6.2.1. The specifics of each model lie primarily in the architecture of the encoder and decoder.

- MLP model : Which is used when we are in sample-level representation.

  Encoder: The encoder ϕ is composed of two hidden layers: ϕ_0 and ϕ_1. The first layer ϕ_0 takes as input $\mathbf{e} \in \mathbb{R}^d$ and outputs $\mathbf{h}_0 = \phi_0(\mathbf{e}) \in \mathbb{R}^{d'}$. The second layer ϕ_1 takes as input $\mathbf{h}_0 \in \mathbb{R}^{d'}$ and outputs $\mathbf{h}_1 = \phi_1(\mathbf{h}_0) \in \mathbb{R}^{d'}$.The equations for the two layers are:

$$\begin{aligned} \mathbf{h}_0 &= \phi_0(\mathbf{e}) = \mathrm{ReLU}(\mathbf{W}_0\mathbf{e} + b_0) \\ \mathbf{h}_1 &= \phi_1(\mathbf{h}_0) = \mathrm{ReLU}(\mathbf{W}_1\mathbf{h}_0 + b_1) \end{aligned} \tag{5}$$

  There is no pooling operator in this model.

  Decoder: The decoder takes $\mathbf{h}_1$ as input and produces a label prediction $\hat{y}$. It comprises a single linear layer: $\hat{y} = \mathbf{W}_2\mathbf{h}_1$.

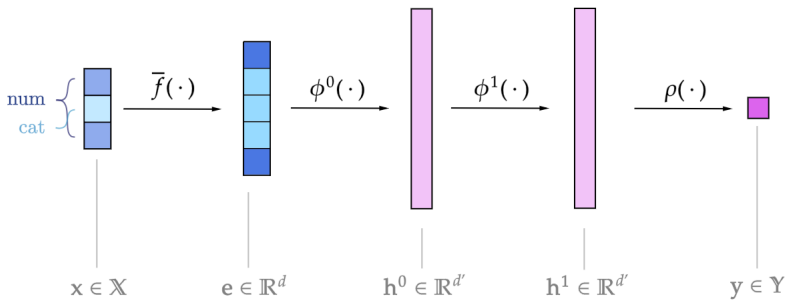


Figure 9: MLP architecture for tabular data. Source : Edouard Couplet et al. [4]

- A Transformer-like model : It is used when we are in feature-level representations.

  Encoder: The encoder ϕ is also composed of two hidden layers: ϕ_0 and ϕ_1. The first layer ϕ_0 takes as input $\mathbf{e} \in \mathbb{R}^{d \times m}$ and outputs

$\mathbf{h}_0 = \phi_0(\mathbf{e}) \in \mathbb{R}^{d' \times m'}$. The second layer ϕ_1 takes as input $\mathbf{h}_0 \in \mathbb{R}^{d' \times m'}$ and outputs $\mathbf{h}_1 = \phi_1(\mathbf{h}_0) \in \mathbb{R}^{d' \times m'}$.

The equations for the two layers are:

$$\begin{aligned} \mathbf{h}_0 &= \phi_0(\mathbf{e}) = \text{ReLU}(\mathbf{W}_0\tilde{\mathbf{e}} + b_0) \text{ with } \tilde{\mathbf{e}} = \mathbf{e}A_0 \\ \mathbf{h}_1 &= \phi_1(\mathbf{h}_0) = \text{ReLU}(\mathbf{W}_1\tilde{\mathbf{h}}_0 + b_1) \text{ with } \tilde{\mathbf{h}}_0 = \mathbf{h}_0 A_1 \end{aligned} \tag{6}$$

The pooling operator in this model is: $\mathbf{z} = \sum_{j=1}^{m'} \mathbf{h}_{1j}$ with $\mathbf{z} \in \mathbb{R}^{d'}$.

Decoder: The decoder takes $\mathbf{z}$ as input and produces a label prediction $\hat{y}$. It comprises a single linear layer: $\hat{y} = \mathbf{W}_2\mathbf{z}$.

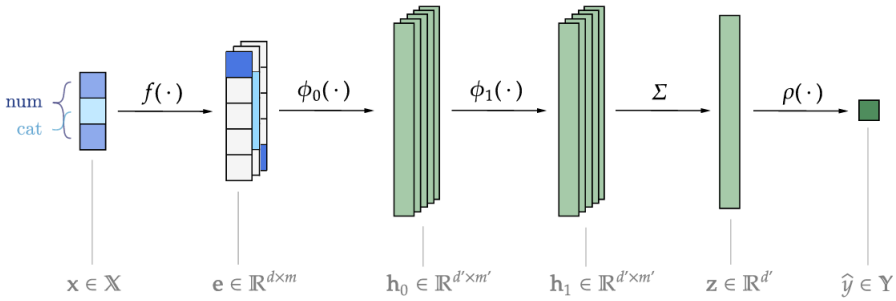


Figure 10: Transformer-like architecture for tabular data. Source : Edouard Couplet et al. [4]

$\mathbf{W}_k$ and $\mathbf{b}_k$ are trainable parameters. $\mathbf{A}_k$ are matrices of size $m \times m$ representing the attention matrices for the simplified transformer, also a trainable parameters.

For regression tasks, our aim is to minimise the mean square error (MSE), which measures the average of the squared errors between predicted and actual values. For the classification tasks, we integrate a sigmoid activation function into the model to convert the outputs into probabilities, and then minimise the binary cross-entropy, which measures the divergence between the predicted and actual probability distributions.

6.3 Scope of experiment

In our work, we evaluate the impact of feature embeddings techniques on the performance of two different DL architectures: a standard MLP and a simplified transformer-type architecture, as described in the previous section. We do not dwell on other practices aimed at improving models, such as additional loss functions, data augmentation. Although they could bring improvements to the models, we are more interested in the impact of our three techniques, baseline, PLE and Feature2Vec on the models in basic configurations.

6.4 Evaluation procedure

For classification tasks, the metric used is accuracy. For regression, we employ RMSE (Root Mean Squared Error).

Each dataset is loaded and split into training, validation, and test sets, with the proportion 70%, 20%, 10% respectively and a specific random seed to ensure reproducibility. A model is then trained on the training set and validated. After training, predictions are made on the test set.

The metric values we use to evaluate the model's performance are the averages of the metric over 5 random seeds. The standard deviations are detailed in the Appendix. A result is considered better than another if its average score is higher.

6.5 Hyperparameters and optimization details

In the course of conducting our experiments, we encountered limitations in the computational capabilities of our computer. Consequently, we selected hyperparameter values that were feasible to run on our available hardware. This subsection provides a detailed overview of these fixed hyperparameters and the optimization algorithms employed. By carefully choosing these parameters, we ensured that the experiments could be executed efficiently while still maintaining the integrity and relevance of the results.

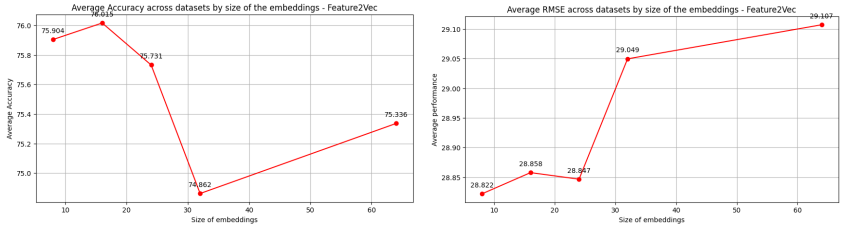
For the embeddings techniques:

- Numerical embeddings PLE: Embedding size = 8
- Feature2vec:

- Numbers of bins for numerical discretization = 24
- Learning rate = 0.1
- Number of k negatives sampling = 10
- Epochs = 300

For the Feature2Vec embedding technique, we tuned the size of the embedding produced after training the neural network. We have limited ourselves to an embedding size that is executable on our machine. Thus, the search space for the embedding size hyperparameter is [8,16,24,32,64].

Figure 13 illustrates the performance progression of the MLP model across all datasets in relation to the embedding size. It is composed of two graphs, one for the datasets for the classification task and one for the datasets of the regression task. The value of the metric represented is the average value over all the datasets.



(a) Classification task (b) Regression task

Figure 11: Average metrics (a) Accuracy (b) RMSE across datasets by size of the embeddings - Feature2Vec

For the classification task, the best performance on average is obtained with an embedding size of 16. For the regression task, the best performance on average is obtained with an embedding size of 8. We decided to choose d = 8 as the final embedding size. This size provides the best performance for regression tasks and the second best for classification tasks. The same could be said for size 16, but given that the difference between the performance of the two sizes is not enormous, we prefer the one that will require fewer resources.

For the models training, the optimizer used for parameter updates is an Adam optimizer with a fixed learning rate of 0.001.

7 Results and analysis

In this chapter, we present the results of our experiments and provide a comparative analysis of the different embedding techniques on models. The primary objective is to evaluate the performance of the proposed methods on various tabular datasets and to identify the strengths and limitations of each approach.

We start by presenting the experimental outcomes, including performance metrics, visualizations, and an in-depth discussion of the results.

7.1 Experimental outcomes

In this section, we present the results of our experiments, focusing on the comparative performance of our three different embedding techniques: the Baseline technique (BASE), Piecewise Linear Encoding (PLE) and Feature2vec (F2V). When applied to a Transformer-like model, we note BASE-T, PLE-T, F2V-T. The results are reported in 2 differents tables :

- Classification tasks using MLP and Transformer-like models.
- Regression tasks using MLP and Transformer-like models.

The best score among the techniques is highlighted in bold for each dataset per column.

	BM	ELEC	KDD	COM
BASE	**78.563**	83.941	75.308	**75.974**
PLE	78.366	**84.783**	**76.143**	74.856
F2V	75.917	77.687	75.268	72.392
BASE-T	**78.412**	85.209	76.103	76.526
PLE-T	76.087	**85.511**	**77.137**	**76.887**
F2V-T	76.530	77.323	74.916	70.526

Table 3: Performance comparison of different embedding techniques applied on both models MLP and Transformer-like across various datasets for a classification task. Bold values indicate the highest accuracy within each dataset.

	WQ	CAL	BSD	HS
BASE	**0.677**	0.158	87.612	**0.204**
PLE	0.683	**0.157**	**62.583**	0.209
F2V	0.731	0.247	114.26	0.297
BASE-T	**0.668**	0.164	50.908	0.213
PLE-T	0.680	**0.161**	**45.184**	**0.204**
F2V-T	0.730	0.249	114.650	0.287

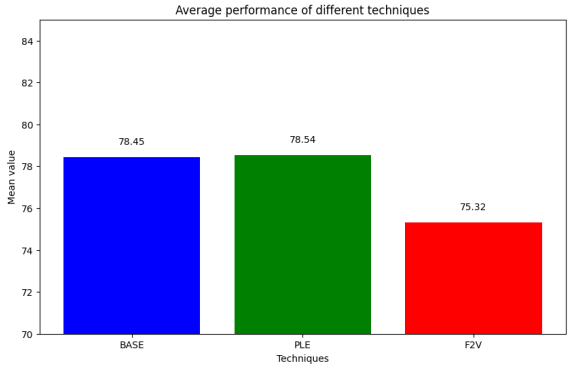
Table 4: Performance comparison of different embedding techniques applied on both models MLP and Transformer-like across various datasets for a regression task. Bold values indicate the lowest RMSE within each dataset.

7.2 Analysis

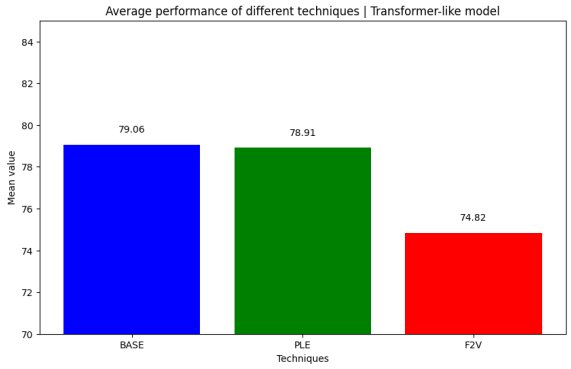
The experimental results presented in tables 5 and 6 provide a comprehensive comparison of the performance of different embedding techniques (BASE, PLE, F2V) and their transformer variants (BASE-T, PLE-T, F2V-T) across the 8 datasets. These datasets cover both classification and regression tasks, allowing for an evaluation of the methods.

- Classification task:

  The results for the classification tasks show a general trend: PLE is better than the other techniques. Specifically, the performance of the MLP model was improved with PLE on 2 out of 4 datasets. For the Transformer-like model, baseline is also slightly better than PLE, but PLE improves the model's performance 3 times out of 4. However, the difference between PLE and Baseline remains minimal across all datasets. For both models, using our Feature2Vec technique did not bring any improvement over the Baseline. Figure 12 illustrates this analysis by providing the average performance of the techniques for our two models.



(a) Multi-layer perceptron

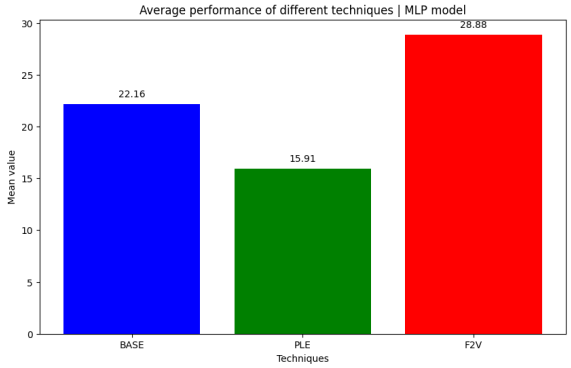


(b) Transformer-like model

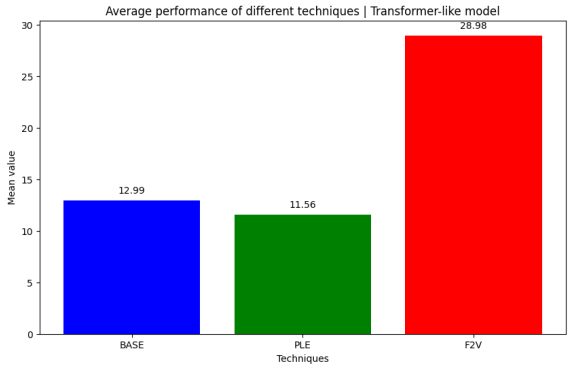
Figure 12: Average accuracy across the 4 classification datasets for each embedding technique

- Regression task:

  The results for the regression tasks are similar to those for classification in terms of performance improvements. Once again, the performances of Baseline and PLE are close. The difference in the average RMSE between Baseline and PLE is slightly greater than for the classification task. This is due to the Bank-sharing demand (BSD) dataset, which has a high RMSE for the three techniques. Additionally, for this task also, our Feature2Vec technique does not seem to bring any significant improvements to the model performances. Figure 12 illustrates this analysis by providing the average performance of the techniques for our two models.



(a) Multi-layer perceptron



(b) Transformer-like model

Figure 13: Average RMSE across the 4 regression datasets for each embedding technique

At the beginning of our work, and given how they were presented, we expected to see an improvement in model performance as we transitioned from a determined technique to an algorithmic or automatic technique. This expectation was partially met with the algorithmic technique and much less so with our automatic technique. We will attempt to understand why.

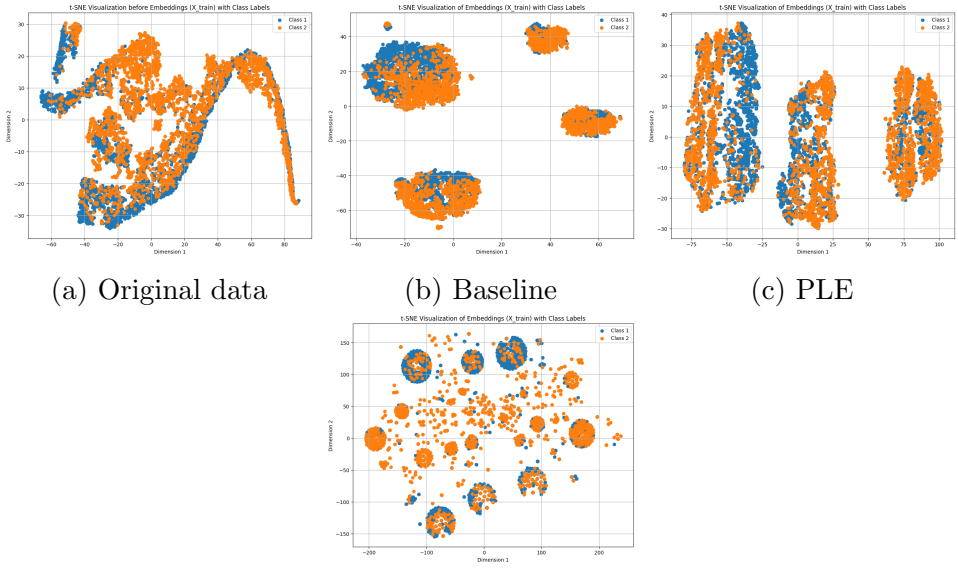
To draw conclusions about the performance improvements achieved through different techniques, we will begin by examining the impact of embeddings on the spatial representation of our data.

To do this, we will study the representation of data before and after embeddings for the Bank-Marketing (BM) dataset. We choose this dataset in particular because it was one of those for which the baseline technique provided the best performance across all models.

We represent below the state of BM data before and after applying the 3 embedding techniques to see if a visual difference between classes can be made after using the concerned embedding technique.

Since the data is multi-dimensional, we need to employ a dimensionality reduction technique to display the plots in 2D. We have chosen t-SNE. Student-t distributed Stochastic Neighbor Embedding (t-SNE) is a dimensionality reduction technique used to represent multi-dimensional data in two or three dimensions. It is particularly effective for visualizing complex structures and clusters in the data. t-SNE preserves local relationships within the data, making similar groups of points easily identifiable in the reduced space.

t-SNE only works with numerical variables, which is convenient because BM does not have any categorical variables.



(a) Original data (b) Baseline (c) PLE

(d) Feature2vec

Figure 14: t-SNE representations of BM data before (a) and after embeddings (b,c,d). Perplexity=90, iterations=3000.

Figure 14 allows us to make the following observations:

- (a) The data in the BM dataset are similar because the initial spatial distribution does not allow for a clear distinction between the classes. When there are clear differences between the data of different classes,

these differences are visible via a t-SNE plot, which is not the case for our data. The data from both classes overlap for the major part on the plot.

- (b) The baseline technique produces well-visible clusters among our data, but these clusters are not specific to any particular class. We still observe an overlap of classes within these clusters.
- (c) The PLE technique also produces clusters, but fewer than those produced by the baseline technique. However, with this technique, we still observe an overlap of classes within the clusters.
- (d) The Feature2Vec technique, on the other hand, does not produce easily visible clusters. However, we do have some small groups of data points formed on the plot. Between these small groups, other data points are scattered. This is logical given that in the algorithm we are discretising the space of numerical variables and BM is made up of 7 numerical variables.

The above observations lead us to draw the following conclusions:

1. The Baseline and PLE techniques produce slightly similar embeddings, which is why they have very close performance regardless of the model and the task.
2. The Feature2Vec technique produces embeddings that do not allow for a clear representation of the data in their new space. Its ability to capture complex relationships and semantic similarities in high-dimensional data is inferior to that of the other two techniques.

With these conclusions drawn, we will now try to examine the causes. This is what we will do in the next section.

7.2.1 Threats to validity

In this section, we will talk about the factors that prevented us from achieving our objectives, at least not completely.

First conclusion: baseline and PLE embeddings are similar

Remember that the performance of models using the PLE technique is on average slightly better than that of models using the baseline technique.

If we look closely, we can see that this can be explained by the formalisation of the two techniques. The baseline technique, which is a simple technique, applies a determined technique to features depending on their nature : One-hot encoding for categorical features and normalization for numerical features.

The PLE technique, on the other hand, requires a little more calculation for its encoding and it is classified in the algorithmic techniques. The essence of PLE is to modify the representation of numeric features only, but for categorical features it uses the determined One-hot encoding technique.

Overall, it is understandable that the embeddings of the baseline and PLE techniques are similar because both use One-hot encoding to encode categorical features. The difference in performance comes from the encoding of the numerical values: while baseline performs a simple normalisation, PLE transforms the features into vectors of size T, where T is the number of intervals in which the values of the numerical variable are divided.

Second conclusion: Feature2Vec's embeddings are lacking in information

The idea of applying the Word2Vec principle to tabular data was an interesting avenue to explore and we hoped to improve the performance of the models using our technique.

An explanation could come from the process of adapting the tabular data so that it mimics the data on which Word2Vec was designed to work. Word2Vec works with words, phrases, corpus and vocabulary. Tabular data, as presented in section 2.1, is made up of rows and columns. 1 row corresponds to an entry in the table and the value of a column corresponds to an attribute of this entry. The idea was to treat a row as a sentence and the value of a column as a word; the whole dataset as a corpus and the vocabulary will be made up of all the unique values in this corpus.

However, tabular data is made up of categorial and numerical features. Therefore, work needed to be done on the numerical data.

Numerical variables can be of two types: continuous and discrete. Discrete numerical variables can already be seen as "categories" even though they are in $\mathbb{N}$, while continuous numerical variables pose a problem for cate-

gorisation. A discretisation step was carried out on the continuous numerical variables. This transformation was not done without loss of information: the information contained in the basic numerical value is reduced so that it can be made into one of k categories, k being the number of bins to be produced by the discretisation process. Once we had categories for all the variables, a simple one-hot encoding was used to obtain an initial vector representation. A major part of the loss of information in the embedding process in our Feature2Vec occurs during the categorisation of the numerical variables, this could justify the poor representation of the relationships between the data.

Another factor that may explain the results obtained is our computing capacity. As our hardware architecture was limited during our work, we did not have the opportunity to explore large embedding sizes or large numbers of bins. We tried to run our work on cloud computing platforms such as Google colab using a GPU runtime, but there too, we had errors indicating a lack of RAM. For Feature2Vec, for example, it took 1.5 hours to calculate the embeddings for a size of 24 with the 5 random seeds.

Now that we have identified the reasons why the objectives were not achieved, we are going to suggest a few areas for improvement.

7.3 Possible improvements and future work

Having recognised the problems that have prevented us from achieving our objectives, we are now going to propose some improvements to our techniques.

- Baseline: This technique does not seem to require any change, as it provides a good basis for simple embeddings that are less expensive to compute.
- PLE: Although the results obtained so far are encouraging, there is still room for improvement over the baseline. To achieve better performance, we should focus on a categorical variable encoding technique other than One-hot encoding that preserves as much category information as possible. In their paper 'A comparative study of categorical variable encoding techniques for neural network classifiers', Potdar et al. compared seven categorical data encoding techniques. Their comparison indicates that Backward Difference Coding and Sum coding perform best on the Cars dataset from [5]. However, they caution that

these findings cannot be generalised as the study was conducted on a single dataset. Therefore, the use of a better categorical encoding technique, combined with the excellent performance of ELP, could significantly improve deep learning models.

- Feature2Vec: For this technique, the avenues for improvement are more research work to be explored. Finding a better way of adapting the Word2Vec principle to tabular data, bearing in mind that the information contained in the numerical variables is very useful for the learning process of deep learning models. We could imagine a solution using PLE on numerical variables, but we'd have to think about how to define the co-occurrence matrix and obtain the context-target pairs.

The availability of equipment enabling large-scale calculations to be carried out within a reasonable timeframe will play an important role in improving our techniques. This will enable us to try out different sizes of embedding and properly study their impact on model performance.

In this thesis, we studied the impact of embeddings in vector form for 2 main reasons. Firstly it is the common form of embeddings and secondly it is the form in which the MLP and Transformer-like models used receive the data.

In recent years, there has been significant work focused on transforming tabular data into a format more suitable for deep learning models, which are reputed to perform best with homogeneous data. For instance, Zhu et al. [24] propose converting tabular data into images and using convolutional models like CNNs. In their method, each feature corresponds to a pixel, and an image is generated for each data sample, where the pixel intensity reflects the value of the corresponding feature. However, their work was limited to gene expression profiles and molecular descriptors of drugs, leaving the broader applicability of this uncertain approach. Another avenue, SuperTML (Super Tabular data Machine Learning) by Sun et al. [20], also seeks to leverage the performance of CNNs by adapting tabular data into images. In this method, tabular features are projected onto a two-dimensional embedding, which is then processed by fine-tuned two-dimensional CNN models for classification. An image illustrating this projection is shown in Figure 15. The effectiveness of this technique is also uncertain, as the experiments were conducted on small datasets.

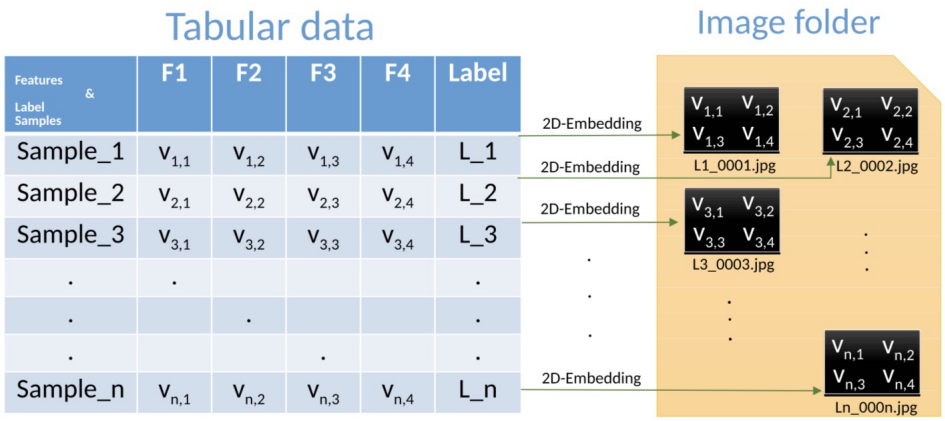


Figure 15: Converting tabular data into images with 2D-dimensional embeddings. Source Sun et al. [20]

That said, the exploration of tensor or matrix embeddings for tabular data is an avenue for future work. Even if these embeddings can only be exploited by a subset of deep learning models but this will enable tabular data to be transformed into a form of homogeneous data where deep learning models perform better.

The exploration of supervised embedding techniques could also produce conclusive results for deep learning models. In particular, in the original PLE paper [7], the authors propose an alternative way of calculating the bins of numerical variables based on the values of the y labels.

8 Conclusion

The aim of this thesis was to evaluate the impact of feature embedding techniques on the improvement of deep learning models on tabular data. The interest in the improvement research stems from the fact that deep learning models do not perform as well on tabular data as they do on other types of data such as images or text, and with a view to setting up a multimodal data pipeline, it would be wise to use the learning models that perform best on tabular data. The tree-based models that perform best on tabular data cannot be included in this multimodal pipeline because their learning process is not differentiable and therefore does not allow joint optimisation of the models block.

We began by defining the framework of our work by providing theoretical elements on tabular data, deep learning and embeddings. Then, we pointed out that our focus would be on self-supervised embedding techniques and distinguished between three types of embedding techniques: determined, algorithmic and automatic.

For our work, we have used an embedding technique for each type, namely a baseline technique for the determined type, PLE for the algorithmic type and Feature2Vec for the automatic type. The big innovation here is the Feature2Vec technique, which is inspired by the automatic embedding technique used in NLP, Word2Vec. By adapting tabular data so that the Word2Vec algorithm can be applied to it, we hope to achieve performance similar to that provided by Word2Vec in the text domain.

At the end of our experiments, we concluded that deep learning models used to process tabular data would gain in performance with embeddings of numerical variables. The biggest challenge for neural network models lies in the transformation of categorical data. Feature2Vec aimed to overcome this challenge by providing contextual embeddings of features, but we realized that applying the Word2Vec principle to tabular data represented yet another challenge in terms of preserving information during transformations. Models using embeddings produced by Feature2Vec performed less well than models using the other two embedding techniques.

As an answer to the question: **How can we represent tabular data for better performance of deep learning models?**, the use of embeddings that preserve data information will help to enhance the models. For numerical data, an algorithmic technique such as PLE, which is inspired by

One-hot encoding but generates embeddings that maintain the order of numerical values, proves beneficial. For categorical data, embeddings that preserve informations such as similarity, order among categories and frequency information will improve the performance of deep learning models.

We have evaluated the impact of self-supervised techniques throughout this thesis and suggest as future work the exploration of supervised embeddings for categorical and numerical data. We have also concluded that the use of label values can help to create more domain-specific embeddings for each dataset, paying particular attention to the occurrence of overfitting problems. In addition, it would be useful to investigate techniques that transform tabular data into a format on which deep learning models are known to perform best.

Appendix

A Useful Links

The code of our work : Github
Link to detailed description of the datasets used : Details about datasets

B Plot of median performance for Feature2vec

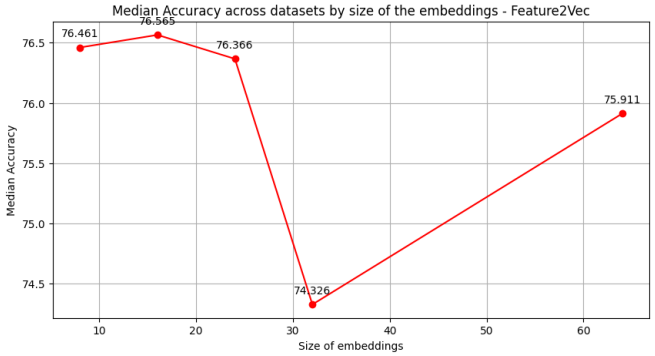


Figure 16: Median Accuracy across datasets by size of the embeddings - Classification task - Feature2Vec

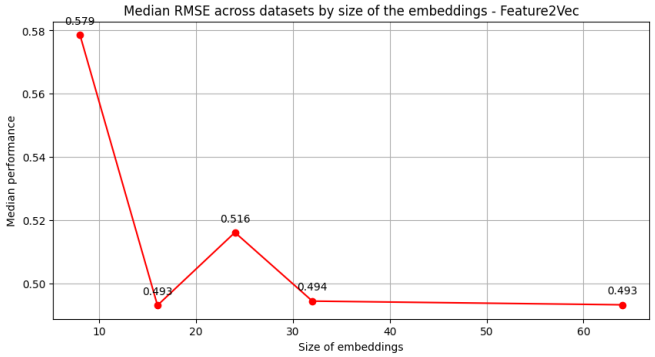


Figure 17: Median RMSE across datasets by size of the embeddings - Regression task - Feature2Vec

The median accuracy follows the trend of the average, and embeddings of size 8 perform close to embeddings of size 16. The median RMSE, on the other hand, is not: there is a big difference between the values of 8 and 16.

C Standard deviation of embedding techniques

	BM	ELEC	KDD	COM
BASE	1.038	0.507	2.025	0.952
PLE	1.223	0.624	0.815	0.720
F2V	0.963	0.235	1.938	1.059
BASE-T	1.389	0.770	1.447	1.082
PLE-T	1.039	0.623	1.625	0.594
F2V-T	0.436	0.076	0.860	1.22

Table 5: Standard deviation - classification

	WQ	CAL	BSD	HS
BASE	0.013	0.006	2.328	0.007
PLE	0.016	0.004	9.318	0.024
F2V	0.010	0.004	1.418	0.027
BASE-T	0.011	0.007	1.677	0.030
PLE-T	0.008	0.004	1.269	0.020
F2V-T	0.013	0.002	2.382	0.012

Table 6: Standard deviation - regression

There is a large variation in values for the BM and KDD datasets for classification and the BSD dataset for regression.

References

[1] Ami Abutbul, Gal Elidan, Liran Katzir, and Ran El-Yaniv. Dnf-net: A neural architecture for tabular data. *arXiv preprint arXiv:2006.06465*, 2020.

[2] Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6679–6687, 2021.

[3] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[4] Edouard Couplet, Pierre Lambert, Michel Verleysen, John A. Lee, and Cyril de Bodt. Investigating latent representations and generalization in deep neural networks for tabular data. *Neurocomputing*, 2024. Manuscript Number: NEUCOM-S-24-01450.

[5] Dheeru Dua and Casey Graf. Uci machine learning repository, 2017. Accessed: 2024-05-24.

[6] Eric Golinko and Xingquan Zhu. Generalized feature embedding for supervised, unsupervised, and online learning tasks. *Information Systems Frontiers*, 21:125–142, 2019.

[7] Yury Gorishniy, Ivan Rubachev, and Artem Babenko. On embeddings for numerical features in tabular deep learning. *Advances in Neural Information Processing Systems*, 35:24991–25004, 2022.

[8] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943, 2021.

[9] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520, 2022.

[10] Huimei Han, Xingquan Zhu, and Ying Li. Edlt: enabling deep learning for generic data classification. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 147–156. IEEE, 2018.

[11] John T Hancock and Taghi M Khoshgoftaar. Survey on categorical data for neural networks. *Journal of big data*, 7(1):28, 2020.

[12] Quang-Thai Ho, Dinh-Van Phan, Yu-Yen Ou, et al. Using word embedding technique to efficiently represent protein sequences for identifying substrate specificities of transporters. *Analytical biochemistry*, 577:73–81, 2019.

[13] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

[14] Daniel Jurafsky and James H. Martin. Speech and language processing (3rd ed. draft), 2023. Accessed: 2024-05-23.

[15] Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. *Advances in neural information processing systems*, 34:23928–23941, 2021.

[16] Nathan Lay, Adam P Harrison, Sharon Schreiber, Gitesh Dawer, and Adrian Barbu. Random hinge forest for differentiable learning. *arXiv preprint arXiv:1802.03882*, 2018.

[17] Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*, 2019.

[18] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

[19] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1161–1170, 2019.

[20] Baohua Sun, Lin Yang, Wenhan Zhang, Michael Lin, Patrick Dong, Charles Young, and Jason Dong. Supertml: Two-dimensional word embedding for the precognition on structured tabular data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 0–0, 2019.

[21] Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In *International Conference on Machine Learning*, pages 6166–6175. PMLR, 2019.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[23] Jinsung Yoon, Yao Zhang, James Jordon, and Mihaela Van der Schaar. Vime: Extending the success of self-and semi-supervised learning to tabular domain. *Advances in Neural Information Processing Systems*, 33:11033–11043, 2020.

[24] Yitan Zhu, Thomas Brettin, Fangfang Xia, Alexander Partin, Maulik Shukla, Hyunseung Yoo, Yvonne A Evrard, James H Doroshow, and Rick L Stevens. Converting tabular data into images for deep learning with convolutional neural networks. *Scientific reports*, 11(1):11325, 2021.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**