Hancock and Khoshgoftaar *J Big Data* (2020) 7:28
https://doi.org/10.1186/s40537-020-00305-w

Journal of Big Data

SURVEY PAPER Open Access

Survey on categorical data for neural networks

John T. Hancock* and Taghi M. Khoshgoftaar

*Correspondence: jhancoc4@fau.edu
Florida Atlantic University, 777 Glades Road, Boca Raton, FL, USA

Abstract

This survey investigates current techniques for representing qualitative data for use as input to neural networks. Techniques for using qualitative data in neural networks are well known. However, researchers continue to discover new variations or entirely new methods for working with categorical data in neural networks. Our primary contribution is to cover these representation techniques in a single work. Practitioners working with big data often have a need to encode categorical values in their datasets in order to leverage machine learning algorithms. Moreover, the size of data sets we consider as big data may cause one to reject some encoding techniques as impractical, due to their running time complexity. Neural networks take vectors of real numbers as inputs. One must use a technique to map qualitative values to numerical values before using them as input to a neural network. These techniques are known as embeddings, encodings, representations, or distributed representations. Another contribution this work makes is to provide references for the source code of various techniques, where we are able to verify the authenticity of the source code. We cover recent research in several domains where researchers use categorical data in neural networks. Some of these domains are natural language processing, fraud detection, and clinical document automation. This study provides a starting point for research in determining which techniques for preparing qualitative data for use with neural networks are best. It is our intention that the reader should use these implementations as a starting point to design experiments to evaluate various techniques for working with qualitative data in neural networks. The third contribution we make in this work is a new perspective on techniques for using categorical data in neural networks. We organize techniques for using categorical data in neural networks into three categories. We find three distinct patterns in techniques that identify a technique as determined, algorithmic, or automated. The fourth contribution we make is to identify several opportunities for future research. The form of the data that one uses as an input to a neural network is crucial for using neural networks effectively. This work is a tool for researchers to find the most effective technique for working with categorical data in neural networks, in big data settings. To the best of our knowledge this is the first in-depth look at techniques for working with categorical data in neural networks.

Keywords: Deep learning, Neural networks, Embedding, Encoding, Qualitative data, Big data

© The Author(s) 2020. This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Introduction

There is a spectrum of techniques for using categorical variables in neural networks. Finding the right technique can have a significant impact on a model's performance. At one end of the spectrum, we have determined encoding techniques. Determined techniques are simplest. We characterize determined encodings as having the quality that given the same dataset, a determined encoding will produce the same encoding every time a practitioner employs it. Also, determined techniques have low running time complexity. In the middle of the spectrum, we find algorithmic techniques, such as Latent Dirichlet Allocation. Algorithmic techniques may or may not have deterministic outcomes, but we wish to identify them in a class separate from determined techniques because they are more complex in terms of running time. At the other end of the spectrum, we have automatic techniques, where neural networks dynamically generate their own data representations as a part of their training phase. This is the key difference between automatic and algorithmic techniques. Automatic techniques encode categorical data as a side effect of a machine learning task, whereas algorithmic techniques encode categorical data *prior* to the beginning of a machine learning task. In practice, we find encoding techniques are blended as we depict in Fig. 1.

In this survey, we aim to give the reader a solid understanding of current methods for applying neural networks to qualitative data. Therefore, our aim is to describe in detail the three primary techniques. One thing all three techniques have in common is that one must have a way to represent categorical data with numbers in order to use it in a neural network. *Determined* techniques convert categorical data to vectors with a low amount of computational complexity. For example, the determined Label encoding technique simply replaces a categorical value with a distinct, scalar numerical value. *Algorithmic* techniques are more sophisticated than determined techniques and require more intensive computation. Algorithmic techniques are not directly incorporated into

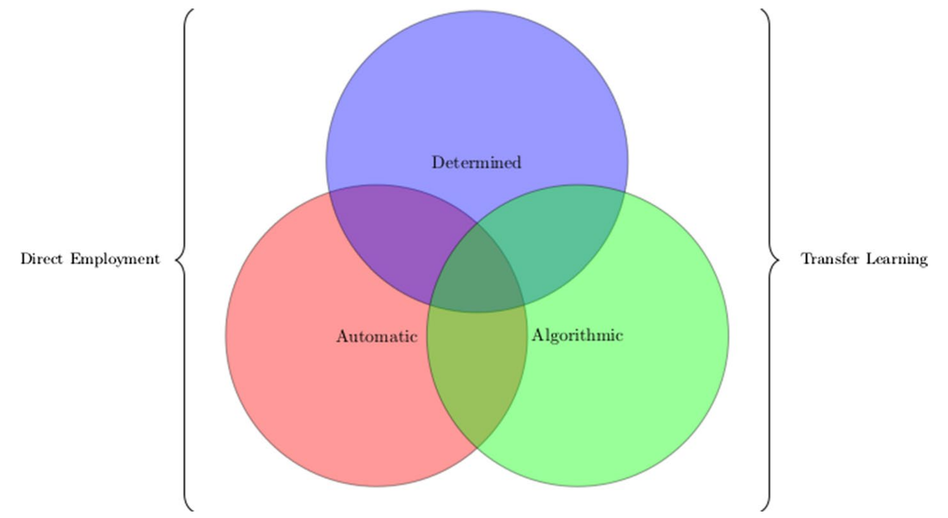

Fig. 1 Encoding techniques are determined, algorithmic or automatic. Most works we review employ some blend of encoding techniques. It is possible to use compositions of up to all three varieties. After selecting a method for representing data, practitioners can transfer it to another algorithm, or use it directly in the next stage

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

the training of a neural network, but rather take the form of more involved pre-processing of data. *Automatic* techniques employ machine learning algorithms to discover a data representation. Hence, they have an intrinsic dependence on the dataset the practitioner uses. In the literature, we find authors sometimes employ techniques separately, sometimes they compose techniques. Here we use the term "composition" in the mathematical sense of the word; i.e., to compose functions f and g is to apply f to the output of g, often denoted $f \circ g$. Determined techniques are suitable for big-data because they have low running time complexity. Algorithmic and automatic techniques require more resources to encode qualitative data for use in neural networks. However, researchers may be able to leverage transfer learning to side-step having to generate encodings for their qualitative data.

By definition, neural networks operate on vectors of real numbers [1]. Therefore, in order to apply neural networks to qualitative data, we must find a way to transform the categorical data to vectors of real numbers. One must begin with what we call a determined technique for representing categorical values with vectors of real numbers. The most common determined technique we find in this review is One-hot encoding. A determined technique is one that transforms categorical values to vectors with minimal processing. This is a defining characteristic of determined techniques. We may choose to use the result of a determined technique for transforming qualitative data as the input for a neural network. On the other hand, we may choose to use it only as the first step for an algorithmic or automatic technique. Both algorithmic and automatic techniques may exploit dataset labels in the computations they perform to convert qualitative data to numerical form. They are also more computationally complex than determined techniques. What separates algorithmic and automatic techniques is that practitioners must apply algorithmic techniques before initiating the learning process in neural networks. With automatic techniques, one need only convert data with a simple determined technique, like One-hot encoding, and the neural network itself will learn a dense encoding of the categorical data. Latent Dirichlet Analysis (LDA) [2] is an example of an algorithmic technique. Entity embedding [3] is an example of an automatic technique.

There are clear differences in techniques. However, a close look at experiments involving categorical data and neural networks reveals practitioners use these techniques in combination. We use overlapping regions in Fig. 1 to show that researchers often employ combinations of all three techniques. If one aims to use qualitative data in neural networks, one should be open to using a combination of techniques. If the reader plans on extending another researcher's work, understanding which sorts of encoding techniques are used is key to understanding over-all research results.

To help the reader understand the spectrum of techniques, in "Definitions" section, we give definitions for some terms used in this work. We then present our search methodology and related works. In addition, we provide a table of works studied that summarizes them and includes the encoding technique category (determined, automatic, or aglorithmic) for each work. After that, we proceed with an analysis of techniques. Our research for this study reveals that there is a *spectrum* of techniques. We separate works into sections by the interesting encoding technique. However, many works exhibit composition of techniques. Finally, we come to our conclusions on which encoding techniques show the most promise for future research.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Definitions

One term central to this paper is “deep learning algorithm”. Here, deep learning algorithms and feed forward networks are synonyms. We rely on Goodfellow et al.’s definition of the term “feedforward network” in [1]: “A feedforward network defines a mapping $\mathbf{y} = f(x; \theta)$ and learns the values of the parameters θ that result in the best function approximation”. Hence, the term “learning” in what we call a deep learning algorithm. A deep learning algorithm is an algorithm that employs a composition of functions where each function equates conceptually to a layer of a k-partite directed graph. For further clarification on what a deep learning algorithm is, we refer the reader to chapter 6 of [1]. We use the terms “deep learning algorithm”, “neural network”, “feedforward network”, and “deep neural network”, interchangeably.

The next important term in this work is, “categorical data”. We use Lacey’s definition of categorical data, “Categorical variables represent types of data which may be divided into groups. Examples of categorical variables are race, sex, age group, and educational level [4]”. Furthermore, we use the terms “categorical data” and “qualitative data” interchangeably.

We find a more specific definition of categorical variables in Lane [5]. Lane writes about scales of measurement. He defines nominal scales, ordinal scales, interval scales, and ratio scales of measurement. Our notion of categorical variable includes variables that we measure using nominal or ordinal scales according to Lane’s definition of nominal and ordinal scales. A nominal variable is a variable whose values we cannot order. For example, the political party a person belongs to is a nominally scaled random variable. It does not make sense to say that Democrat comes before Republican. Variables we measure with ordinal scales have order, but we cannot associate a number with them. Lane gives the example of the variable “customer satisfaction” that may take levels such as “very dissatisfied” or “somewhat dissatisfied”.

In the works we study here, we find techniques for using categorical data in deep learning algorithms. We find works using several terms to refer to these techniques. Examples of terms referring to these techniques are “entity embeddings” [3], “dense encoding” [6], “distributed representation” [7], and simply “encoding.”. In this work, the terms dense encoding, dense embedding, entity embedding, and distributed representation refer to different ways of mapping categorical data to vectors $v \in \mathbb{R}^n$ for the purpose of using those vectors as input to a deep learning algorithm.

We find that researchers have not reached a consensus on what to call what we refer to as entity embedding. We attempt to follow the succession of important works in deep learning research to find the terms the authors of these important works are using. Our conclusion is that machine learning researchers’ interest in deep learning starts with Krizhevsky, Sutskever, and Hinton’s 2012 success applying deep learning algorithms to the problem of classifying Imagenet data [8]. It is our belief that this success then inspired researchers to find ways to apply deep learning to data other than image data. The next milestone we see is Mikolov et al.’s publication, “Distributed representations of words and phrases and their compositionality” [9]. Interestingly, both the 2012 Imagenet paper, and the 2013 paper on distributed representations of words have a common author: Ilya Sutskever. This fact may lead one to believe the community should settle on, “distributed representations”, as the term for mapping categorical data to vectors

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

for deep learning. There is, however, another work we find important. This is Guo and Berkhahn's, "Entity embeddings of categorical variables" [3]. We prefer the term entity embedding because we feel current and future research will develop techniques Guo and Berkhahn outline in their work.

In this analysis, we categorize, compare and contrast various techniques, in the form of algorithms, that carry out entity embeddings. We refer to these techniques as entity embedding algorithms. The output of an entity embedding algorithm is a mapping from some set of categorical values S to a space of n-dimensional vectors in $\mathbb{R}^n$. We denote the mapping with the letter e. An entity embedding algorithm may change the behavior of e over time. One technique for updating e that we find employed in Mikolov et al. [9] is to express e as a feed forward neural network, and alter the behavior of e by updating the parameters of the neural network. We have many options for network structure and weight updating schemes to choose from, thanks to the work of previous researchers. We find many works where the authors use entity embeddings to produce input values for other deep learning algorithms. Guo and Berkhahn give a rigorous definition in [3] that we summarize here to introduce notation. Let S be the set of distinct values of some variable that is characterized as categorical data. Then an entity embedding e is a mapping

$$e : S \to \mathbb{R}^d \tag{1}$$

of the elements of S to vectors of real numbers. We define the range of an entity embedding as $\mathbb{R}^d$ to allow us the luxury of employing any theorems that hold for real numbers. In practice, the range of our embedding is a finite subset of vectors of rational numbers $\mathbb{Q}^d$ because computers are currently only capable of storing rational approximations of real numbers. We refer to this as "entity embedding", "embedding categorical data", "embedding categorical values", or simply as "embedding" when the context is clear.

The terms we define in this section are key to understanding the works we cover here. In the next section we report the search method we use to find the works we include. This search process is what leads to our formulation of categories of embedding techniques.

Search method

We employ Google Scholar [10] and the Florida Atlantic University (FAU) online library database OneSearch [11] to search for terms synonymous with embedding. These are the synonymous terms: "entity embedding", "tabular data", "dense encoding", "distributed representation", and, "encoding categorical variables".

We combine each of the five synonyms for entity embedding above with each of the following phrases: "deep learning", "neural network", "deep learning survey", and, "neural network survey".

Our research group is interested in entity embeddings as they relate to medical research with respect to the Healthcare Common Procedure Coding System (HCPCS), so we also searched the databases we listed above using phrases, "deep learning HCPCS", and, "neural network HCPCS".

Hence, we conduct a total of 40 searches in each of the two databases. We include the term "survey" in our searches to find related works. In addition, we search for the phrase

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

"encoding categorical variables" by itself in order to find literature that covers techniques that have potential to be used in neural networks. Through the course of conducting these searches, we discovered the spectrum of techniques for using qualitative data in neural networks.

Related work

We find several existing studies that involve using categorical data in neural networks. However, we do not find work that identifies the spectrum of techniques we discover here. Related works we find here cover some aspects of what we find, but none of them present techniques for using qualitative data in neural networks with our unified approach.

Interestingly, when we query the Google Scholar search engine "encoding categorical variables neural networks" we find "A comparative study of categorical variable encoding techniques for neural network classifiers" [12] is the second result. This 3-page work is from October 2017. One positive aspect of this work is that it covers seven techniques for encoding categorical variables. On the other hand, the authors evaluate each technique on the UCI Cars dataset [13], with one neural network architecture. In our search for works to cover in this review we do not find papers that employ all of these techniques. Furthermore, we find that each of the seven techniques the authors listed are all available in the Scikit-learn Category Encoders library. All the techniques covered in this paper fall into the determined category of techniques for using categorical data in neural networks. This work appears to be a report on an exercise where the authors applied various encoding techniques available in a popular machine learning library to a dataset commonly used for teaching purposes. Although the scope of the work is limited, we find it useful for discussing determined techniques for encoding categorical variables.

One work with a promising title for researchers interested in using categorical data as input to neural networks is, "An overview on data representation learning: From traditional feature learning to recent deep learning" [14], by Zhong et al. However, this work traces the history of data representation starting with principal component analysis (PCA) and linear discriminant analysis (LDA) (On a side note, the reader should be aware that the abbreviation LDA is often used for two different techniques in machine learning literature, Latent Dirichlet Allocation, or linear discriminant analysis). Zhong et al. make a great contribution in describing PCA, LDA, and its descendants, but this work is not strictly dedicated to techniques for using categorical data in neural networks in the manner that this survey is. Another major theme in [14] is learned embeddings. Zhong et al. introduce learned embeddings by covering Mikolov et al.'s distributed representations in [9]. There is some overlap in this work and [14] in that regard. However, Zhong et al. present a collection of techniques from a historical point of view. We do not take a historical perspective in this work. This could imply some techniques are passe and others are en vogue. We find that there is a spectrum of techniques, and researchers often compose techniques.

Natural language is a common form of categorical data. On the subject of natural language processing, we also have the survey, "Semantic text classification: A survey of past and recent advances" [15] by Altinel and Ganiz. Part of this work covers embeddings for natural language data. However, the focus of [15] is on text classification, so it only

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

covers techniques for using categorical data in neural networks for text classification. This work is not limited to neural networks only. For example, it also covers knowledge bases such as WordNet, non-parametric techniques for text processing such as K-Nearest Neighbors, Support Vector Machine, and so on. There is some overlap with Altinel and Ganiz's coverage of deep learning based natural language processing techniques, and the automatic class of embedding techniques we study here. Altinel and Ganiz do not write about embedding techniques from a general perspective, so they do not provide a unified description of embedding techniques.

"Graph Embedding Techniques, Applications, and Performance: A Survey" [16] is another survey of embedding techniques albeit exclusively for graph embeddings. We feel this is an interesting, emerging subject in deep learning. Moreover, one may characterize a qualitative attribute of some data as connections between data that share the attributes' value. However, in this work we focus on existing techniques for working with qualitative data in neural networks. As previously, we feel applying graph embedding techniques to categorical data is a subject for future research.

In "Deep EHR: A Survey of Recent Advances in Deep Learning Techniques for Electronic Health Record (EHR) Analysis" [17], Shickel et al. cover entity embeddings in the section entitled, "Concept representation". The introduction mentions DeepCare, and Med2Vec. However, the scope of their work is smaller than the scope of this work. They cover deep learning applications to electronic health records (EHR) data. This survey is interesting to us because it covers techniques for dealing with categorical health care data such as International Statistical Classification of Diseases (ICD) codes as input for deep learning algorithms. We aim to cover the same types of techniques in the context of qualitative data in general.

Survey of techniques

In this section, we present our findings on the works we review. We present embedding techniques in works we study in terms of three categories: determined, algorithmic, and automated. Therefore, we group works into sections by category. We devise categories of techniques based on the degree of automation in the technique, and the degree of coupling the technique has with the neural network training. There is no strict dividing line among techniques. The absence of strict dividing lines is why we prefer the phrase "spectrum of techniques".

We start with the simplest techniques first. We call these types of techniques *determined* techniques, because the encoding process is straightforward, and does not require heavy computation. Determined techniques often require one pass over the dataset to complete an encoding. The next step in these techniques is to construct some data structure such as a look-up table. Finally, to encode the categorical variable, determined techniques require one pass over the entire dataset. After the encoding is complete, practitioners using determined techniques do not need to change the values of the encodings once computed.

We find some authors use a separate technique for mapping qualitative data to vectors of real numbers in a process that is unrelated to the neural network's learning process. These techniques are computationally more intensive than determined techniques. Once the data is transformed, then the authors use the transformed data as input to a

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

neural network. Since the authors employ the embedding technique outside of the functioning of the neural network, we call this class of embedding techniques *algorithmic* embedding. Examples of algorithmic embedding techniques are Latent Dirichlet Allocation (LDA) [18], and "Generalized Feature Embedding for Supervised, Unsupervised, and Online Learning Tasks" [19]. Encoding values computed in this class of techniques may change as the encoding progresses. Once the algorithmic encoding process is complete, the encoded values are fixed, and the practitioner uses these fixed values as input to a neural network.

Automatically optimized embedding techniques rely on the neural network's weight update process. We give an example of how an automated weight update process can work. However, the reader should be aware that algorithms that learn embeddings, such as Global Vectors for Word Representation (GloVe) are more sophisticated than this simple example.

The example we work through here is similar to the functioning of a Keras embedding layer [20]. To compute the embedded value $\mathbf{e}$ of the input we compute $\mathbf{e} = W\mathbf{v}$, where, W is the edge weight matrix for nodes in the neural network's embedding layer, and $\mathbf{v}$ is the input value. For embedding categorical variables, $\mathbf{v}$ must be some encoded value of a categorical variable. Typically, $\mathbf{v}$ is a One-hot encoded value. Please see "One-hot encoding" section for a definition of One-hot encoding. Equations 2, and 3 give an example of how one may compute the embedded value of a One-hot encoded categorical variable.

$$\mathbf{e} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots \\ \vdots & \ddots & \\ w_{1,k} & \cdots & w_{n,k} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{bmatrix} \tag{2}$$

Assuming the entry equal to 1 in the column vector on the right-hand side of Eq. 2 is the encoding vector $\mathbf{v}$, and the value equal to 1 is on the *jth* row of $\mathbf{v}$, the product on the right-hand side of Eq. 2 will be

$$\mathbf{e} = \begin{bmatrix} w_{j,1} \\ \vdots \\ w_{j,k} \end{bmatrix} \tag{3}$$

An easy way to think of the embedding process we illustrate in Eqs. 2 and 3 is that W is a look-up table for One-hot encoded values. A neural network's optimizer applies some procedure such as Stochastic Gradient Descent (SGD) to update its weight values, including those of the weight matrix W. Over time the optimizer finds a value of W that minimizes the value of some loss function, J. As the neural network changes the value of W, the components of the embedded vectors $\mathbf{e}$ change as well. Since the values of $\mathbf{e}$ are updated as a result of the neural network's weight update function, we call the embedding *automatic* embedding.

The reader should bear in mind that the example we give in Eqs. 2 and 3 is not comprehensive. This example illustrates the basic principle of how to use the values of edge weights that a neural network learns as embedded values of One-hot encoded vectors.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

In the research we cover below, we find more intricate adaptations in automated techniques. Some researchers use the weights in W directly in a neural network that does some machine learning task. We call this the direct employment method that we describe more in the next section. At other times, practitioners may use the values in W in a completely different neural network. This is the transfer learning employment technique we also describe in the next section. Transfer learning is not an embedding technique. Rather, it is a way to employ a data representation.

To sum up, we find three major categories of embedding techniques used to encode categorical values for use in neural networks. The degree of complexity in the techniques is one aspect that differentiates them. The other aspect that differentiates techniques is whether the encoding process is part of the neural network's learning process. Determined techniques are the simplest. Algorithmic and automatic techniques are computationally more complex than determined techniques. Algorithmic techniques produce encodings in a manner that is not part of a neural network's weight updating process. Automatic techniques learn embedded values of categorical variables. Once a categorical value is fully encoded, we have options for how to use it. We refer to these options as "employment techniques" that we cover in the next section.

Employment techniques

Practitioners have options for employing embedding techniques. We illustrate these options in Fig. 1 with the curly braces around the Venn diagram that depicts embedding techniques. The first option is to re-use an existing embedding. This option is known as *transfer learning*. Other researchers refer to this technique as using a pre-computed embedding. A pre-computed, or transfer learning, employment technique uses an existing encoding that some other algorithm has learned. In this work we only find practitioners using transfer learning in conjunction with automatic embedding techniques. An example of a transfer learning technique is using the embedding from the GloVe algorithm. This algorithm computes a map of natural language words to vector values [21]. Francois Chollet covers this transfer learning technique in Chapter 6 of [22]. In his example Chollet creates a neural network with an embedding layer. The weights for the embedding layer are the embedded values (vectors), that the GloVe algorithm learned when Pennington, Socher, and Manning ran it using a specific corpus of text for input. The corpus they used consists of a copy of Wikipedia from 2014, and the Gigaword dataset [23]. We know Chollet is using this particular instance of GloVe vectors because we cross-reference the file name Chollet uses in his example, `glove.6B.100d.txt`, with the contents of [24]. In Chollet's example he presents a classifier for sentiment analysis of a collection of movie reviews. In the example, Chollet defines a neural network with an embedding layer. He sets the weight matrix of the embedded layer to have one row for every word in the corpus of movie reviews. He then fixes the values of the weights in the embedding layer so that his algorithm does not update them during training. Furthermore, these fixed values are the embedded equivalent GloVe vectors from `glove.6B.100d.txt`. Chollet's example is typical of how a practitioner employs transfer learning. One may simply take the appropriate part of the embedding layer that another algorithm learns, and use it as a layer in a completely different neural network. Some researchers use embedded values in machine learning algorithms other than

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

neural networks. Guo and Berkhahn do this in [3]; they use an embedded representation of categorical data for input into the K-Nearest Neighbors, Random Forest, and Gradient Boosted Trees algorithms.

The second option researchers have for employing an embedding technique is to use it directly as a part of a larger machine learning algorithm. We call this manner of employing an embedding technique *direct* employment. Often embeddings that are reused in transfer learning are the result of one scientist's direct employment of an embedding technique. In the same work where Guo and Berkhahn employ transfer learning with algorithms other than a neural network, they directly feed the representation of categorical variables the neural network learns into layers above it, and the parameters of the embedding layer are updated as part of the training phase of their algorithm. Just as researchers compose embedding techniques, they may use both direct and transfer employment techniques in the course of one paper. Hence, we encounter the documentation of both methods of employment in some works.

Table of works studied

We summarize the techniques we find in this survey in the table below where we classify them according to the categories we describe above (Table 1).

The common thread we find with research on distributed representation, dense encoding, and embedding is that all three of these subjects employ the same general idea. The idea is to define some mapping from qualitative data to vectors in $\mathbb{R}^n$ that are suitable as input to neural networks.

Usually the authors first assign One-hot encoded vectors to each possible value of some qualitative data as a starting point, and then they apply some transformation of the One-hot encoded vectors to obtain new vectors that are more suitable as input to neural networks.

Determined techniques

In this section we cover determined techniques. Determined techniques are the least computationally complex encoding techniques. One way to spot a determined technique is the encoded values will be the same every time we apply the technique. We begin with the simplest possible of determined encoding techniques, label encoding.

Label encoding

Label encoding is simply assigning an integer value to every possible value of a categorical variable. For example, if a dataset has a categorical variable with values drawn from the set {"vanilla", "chocolate", "strawberry"} then label encoding might assign the mapped values from the set $\{0, 1, 2\}$ respectively. We cannot think of a simpler way to convert categorical values to numerical values. Label encoding is a good example of why we choose to call this class of encoding techniques determined. Once we know all the possible values of a categorical variable, their encoded equivalents are determined by how we arbitrarily choose to assign integer values to them.

The Python Scikit-learn library [63] has a Category Encoders module, with an Ordinal Encoder. The documentation for the Scikit-learn Ordinal Encoder explains that it is an

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Table 1 Works Studied

Description	Reference, implementation reference
Automatic techniques	
Paper introducing TensorFlow framework, explains how Tensorflow is designed for large datasets, including distributed representations	[25, 26]
Seminal work defines automatic embedding technique for natural language processing, transform natural language words to one-hot encoded vectors, and learn lower-dimensional embedded values	[27], N/A
Use embedding layers for categorical variables to predict taxi destination	[28, 29]
Entity embedding categorical data from computing system events for intrusion detection	[30], N/A
Widely used Keras library, provides embedding layer implementation	[20, 31]
BERT leverages word piece embeddings, can be used in a feature-based approach to learn a representation of natural language, shown to work well for named entity recognition tasks	[32, 33]
Survey of techniques for graph embedding, Python package with examples for techniques covered,	[16, 34]
Use Keras embedding layer for entity embedding of categorical values, won third place in a Kaggle competition, map One-hot encodings of categorical data to lower dimensional vectors	[3, 35]
Library for automatic embeddings, part of fast.ai framework, supports PyTorch neural networks library	[36, 37]
Concatenate embeddings of categorical data to time-series data for input to neural network	[38], N/A
Comparison of several pre-computed embedding techniques employing transfer learning, cited in documentation for Pytorch embedding layer	[39], N/A
Learn patient representation from EHR data such as ICD-10 codes, using denoising autoencoder for transfer learning to use learned representation for various tasks	[40], N/A
Survey of deep learning techniques for bioinformatic data use-cases, Examples of applying graph embeddings of proteins to predict protein-protein interactions, and recurrent and convolutional neural networks for predicting expression of one-hot encoded DNA sequences, authors provide source code for techniques they cover	[41, 42]
Word2vec algorithm breakthrough automatic technique for natural language processing, transform natural language data to embedded vectors, suitable for transfer learning	[43, 44]
Apply `word2vec` to corpus of protein sequences to obtain lower dimensional representation, for transfer learning in various machine learning algorithms	[45], N/A
Natural language processing algorithm, uses word co-occurrence matrix for embedding qualitative data in lower dimensional space, suitable for transfer learning	[21, 46]
Transfer GloVe embedding for sentiment analysis to classify emotion, introduces weighting scheme for imbalanced data	[47], N/A
Use GRU's to learn representations of clinical descriptions, representations fed into dense layers that act as classifiers to produce ICD-10 codes	[48, 49]
Embed One-hot encoded vectors for categorical data, feed as input to a log-bilinear neural model for unsupervised anomaly detection in 12 datasets	[50], N/A
Use One-hot encoding of demographic and diagnostic data plus embedding of medical coding (ICD-9) with Long Short-term Memory (LSTM) to predict hospital re-admission, ICD-9 embedding transferred from previous work using Word2vec variant to embed ICD-9 codes	[51, 52]
Embedding technique that Lin et al. leverage in [51], leverage Word2vec algorithm to encode ICD-9 codes as real-valued vectors	[53, 54]
Algorithmic techniques	
Predict patient mortality, categorical inputs include ICD-9, Current Procedural Terminology (CPT), and RxNorm Codes; value counts are used to encode data, authors claim techniques are suitable for big data	[55], N/A
GEL Pre-processing technique for embedding qualitative data for convolutional neural networks, authors claim suitable for any type of categorical variable, and datasets with both numerical and categorical variables	[19, 56]
EDLT pre-processing technique leveraging Pearson correlation for converting tabular data to matrix form for convolutional neural networks	[57, 58]

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Table 1 (continued)

Description	Reference, implementation reference
Use loss function to optimize parameters of hash algorithm for mapping One-hot encoding of data to lower dimensional vectors	[59, 60]
Use Latent Dirichlet Allocation (LDA) to extract features from text of automobile insurance claims, then use features from LDA as input to neural network	[18], N/A
Determined techniques	
Experiment to compare encoding techniques, all encoding techniques mentioned are available in Python Scikit-learn Category encoders librar	[12], N/A
Use One-hot encoding for low-cardinality categorical variables to compute loan risk	[61], N/A
Use leave-one-out encoding for categorical variables as input to convolutional neural network for computer network intrusion detection system	[62], N/A

Table 2 Accuracy results reproduced from [12]

Encoding technique	Accuracy (Percentage)
One-hot coding	90
Ordinal coding	81
Sum coding	95
Helmert coding	89
Polynomial coding	91
Backward difference coding	95
Binary coding	90

implementation of Label encoding. Both Label encoding and the Ordinal Encoder assign a unique integer value to the distinct values of a categorical variable.

Potdar et al. compare seven encoders in "A comparative study of categorical variable encoding techniques for neural network classifiers". One of the encoders in the comparison is the Ordinal encoder. The authors' description of the ordinal encoder matches that of the Scikit-learn label encoder. Also, the names of the seven encoders match the names of seven encoders available in the Scikit-learn Category Encoders module. We suspect the authors used this module for the experiments in [12]. For their experiments, the authors encode data from the Cars dataset [13], pass it to a neural network, and report the accuracy. We reproduce the accuracy report in Table 2.

One may be curious about the best performing encoding techniques that Potdar et al. report in this table. However, as we noted in "Related works" section [12] is a brief work. From the authors' description of their experiments, we do not find a theoretical justification for why the best techniques do so well. On the other hand, the result of the worst performing encoding technique is in line with our expectations. The worst performing technique is Ordinal Coding (*a.k.a. Label encoding*). We expect poor performance of label encoding because it introduces artificial order among categorical variables. For example, when we directly use label encoded values in neural networks optimized with stochastic gradient descent, the larger label values will contribute more to gradients we use to update weights. This does not make sense because we assigned integer values to the categorical variables arbitrarily. Label encoding also presents a problem

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

for unsupervised learning with neural networks. The problem is that an unsupervised algorithm cannot exploit the numerical difference between encoded values because the numbers we assign those values are arbitrary. We feel this informal argument provides enough justification for why practitioners must use something more sophisticated than label encoding for input values. Moreover, it also aligns with the poor performance that Potdar et al. report.

While Label encoding may not be suitable for direct input to neural networks, one may need to use it as the very first step in an encoding technique. Some libraries for One-hot encoding require a list of integer values for input, and Label encoding is a reasonable way to obtain that list of values. Label encoding is the simplest determined technique, but it has its uses.

Code counting

In "Improving palliative care with deep learning" [55], Avati et al. document a simple determined encoding technique that is effective for the purpose of predicting patient mortality. Avati et al. derive a dataset from electronic heath records (EHR) data. They then use the derived dataset as input for a classifier that they employ to predict whether a patient will die in the next year. The EHR data Avati et al. use includes qualitative data in the form of International Classification of Diseases, 9th revision (ICD-9) code, Current Procedural Terminology (CPT) codes, and RxNorm prescription codes. To derive a record from the EHR data for a patient, the authors use the counts of types of codes as features. In order to calculate the counts, Avati et al. use the concept of windows and slices. In order to define a window for EHR data, Avati et al. select a prediction date. The prediction date is not necessarily the current date. We copy a table from [55] to give the reader a clear idea of how Avati et al. segment EHR data into windows and slices. The copy of the table is in Table 3. After the data is arranged into slices, Avati et al. use the counts of various qualitative values as features. Finally, they concatenate the lists of counts for each slice to complete the encoding technique.

Avati et al. explain that their machine learning task for predicting patient mortality is a proxy for recommending a patient for palliative care. They report that their model achieves 0.69 AUC for predicting whether the patient will die within the year after their prediction date. Avati et al. qualify this performance metric by pointing out that this model achieves a 0.93 AUC when they use it as a proxy to predict whether a patient is a candidate for palliative care. It would not be surprising to find that an embedding technique that discards less information in the EHR data would achieve better performance. For example one could employ techniques Duarte et al. use in

Table 3 Window slices in [55], PD stands for prediction date

	Start date	End date	Duration
Observation window	PD—365	PD	356
Observation slice 1	PD—30	PD	30
observation slice 2	PD—90	PD—30	60
Observation slice 3	PD—180	PD—90	90
Observation slice 4	PD—365	PD—180	185

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

[48] to encode ICD-9 codes. On the other hand, the code counting technique that Avati et al. employ is amenable to large data sets since has a low running time complexity. Assuming there are n EHR records, and the records have a patient ID, we could sort all the EHR records in $O\big(n\log(n)\big)$ time. See Section 7.2 of [64] for proof of running time complexity of sorting. Then we can count the codes for each patient ID in $O(n)$ time, which means that count encoding has $O(n)$ running time complexity. Count coding is a determined technique that might work when other techniques fail due to a huge dataset size. Indeed, in Section 2 of [55] the authors explicitly claim their technique is suitable for big data. Otherwise, we suspect using other techniques that discard less information about qualitative data will provide better performance.

One-hot encoding

One-hot encoding is a technique that requires very little work for to use, and practitioners often use it as a first step in more sophisticated techniques, such as employing a Keras embedding layer. Known colloquially as One-hot encoding [65], we find that Guo and Berkhahn [3] identify One-hot encoding as the Kroneker Delta Function. We express One-hot encoding formally as follows. Let $\mathbf{x}$ be some discrete categorical random variable with n distinct values $x_1, x_2, \ldots x_n$. Then, the One-hot encoding of a particular value x_i is a vector $\mathbf{v}$ where every component of $\mathbf{v}$ is zero except for the *ith* component, which has the value 1. For example, assume we have some random variable $\mathbf{x}$ that takes values from the set $S = \{a, b, c\}$. Let $x_1 = a$, $x_2 = b$, and $x_3 = c$. A One-hot encoding for $\mathbf{x}$ is: $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$. Since the One-hot encoding of the levels of a categorical variable only depends on the number of levels, One-hot encoding falls into the determined family of techniques for encoding categorical variables for use in neural networks.

One clear disadvantage to One-hot encoding is that the distance between One-hot encoded vectors carries just a bit of information. Cui, Xie, and Shen mention in [66] that the distance between any pair of distinct One-hot encoded vectors is the same. It is beyond the scope of this analysis to prove that a general distance metric is the same for any pair of distinct One-hot encoded vectors, but we can easily do so for the Euclidean distance

$$d_n(\mathbf{v_1}, \mathbf{v_2}) = \left(\sum_{i=1}^{n} \left(v_{1_i} - v_{2_i} \right)^2 \right)^{\frac{1}{2}} \tag{4}$$

Only one component of $\mathbf{v_1}$ and $\mathbf{v_2}$ is 1 because they are One-hot encoded, so Eq. 4 simplifies to

$$d_n(\mathbf{v_1}, \mathbf{v_2}) = 2^{\frac{1}{2}} \tag{5}$$

in the case that the vectors are different, and

$$d_n(\mathbf{v_1}, \mathbf{v_2}) = 0 \tag{6}$$

in the case the vectors are the same.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Hence, the Euclidean distance between two One-hot encoded features tells us only that two features are identical, or different. Therefore, distance between One-hot encoded features carries little information about their similarity.

A second disadvantage of One-hot encoding is that it consumes storage resources aggressively if we store One-hot encoded values directly. By storing One-hot encoded values directly, we mean storing a literal one or zero for every component of a vector. Suppose a dataset has a feature that is a categorical variable with n values. Then we might require on the order of n times the size of the feature in the dataset to store the One-hot encoded values of the feature. We can overcome this increase in the amount of storage that One-hot encoding requires with sparse representations. Sparse vector and matrix representations store data in a compressed form. Python's Scikit-learn One-hot encoder uses a "sparse" option by default [67], so unless the user specifies otherwise, the Scikit-learn One-hot encoder will store One-hot encoded values efficiently.

Nevertheless, One-hot encoding is often a first step to using existing embedding libraries such as the Keras embedding layer [20]. Indeed, one may look at other encoding techniques in the spectrum as answering the question of what might we do to improve our results after One-hot encoding our data.

However, One-hot encoding it is not without its uses. One good aspect of One-hot encoding is that is simple to implement, and has efficient running time. Referring to the Kroneker Delta Function definition of One-hot encoding above, we see that the running time to One-hot encode a categorical variable is on the order of the number of possible values of the categorical variable, plus the size of the dataset to encode. A simple look-up table will suffice to produce the One-hot encoded equivalent of a categorical variable. The running time for using a simply implemented look-up table is also on the order of the number of One-hot encoded values, and slightly more sophisticated implementations would have order log_2 running time in the number of encoded values.

We find research where One-hot encoding categorical variables for use in neural networks is sufficient to design a neural network that outperforms other machine learning algorithms. The work is entitled, "Financial system modeling using deep neural networks (DNNs) for effective risk assessment and prediction" [61], by Jing Duan. In this work Duan evaluates various classifiers' ability to classify loan applications into risk categories. The input to the classifiers includes categorical variables such as home ownership, and loan purpose (e.g. credit card debt consolidation, or home improvement). Duan uses One-hot encoding to encode these categorical values. Nevertheless, Duan reports the best results in loan risk prediction accuracy for a deep neural network classifier. In [61], Duan compares the Performance of a Deep Neural Network to Logistic Regression, Linear Discriminant Analysis, Decision Tree, Support Vector Machine, Radial Basis Functions, and Adaptive Gradient Boosted Decision Tree. While it is true that Duan's experiment shows a Deep Neural Network outperforms other machine learning algorithms with One-hot encoding, we see an opportunity for future research to employ other classes of encoding techniques, such as Golinko et al.'s algorithmic technique in [68].

One-hot encoding may be a good first choice for getting started on selecting a machine learning algorithm. For example, Duan's experiment shows that under the conditions of his tests, a Deep Neural Network is the strongest candidate. After evaluating different

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

algorithms, one may be convinced a Neural Network is the best model to use. After that, one may decide to invest resources in trying different encoding schemes for further gains in performance.

Leave-one-out encoding

We learn about the Leave-one-out pre-processed encoding technique for categorical values in "Enhanced Network Intrusion Detection using Deep Convolutional Neural Networks" [62], by Naseer and Saleem. The popular Scikit-learn Python library, that machine learning practitioners often employ, includes this encoding technique in its Category Encoders module [69]. We are unable to find a peer reviewed publication that defines leave-one-out encoding. To the best of our knowledge, Zhang discovered the leave-one-out encoding technique, and it is documented on slides 25 and 26 of [70].

Leave-one-out encoding clearly falls into the determined class of techniques for using categorical data in neural networks. To leave-one-out encode a categorical value, we represent the categorical value with the mean value of all the response variables except the response that the categorical variable currently appears with. To complete the encoding, to avoid over fitting, we add a small random noise value to the mean we compute.

Leave-one-out encoding must be implemented carefully when working with big data. First of all, to compute the mean values of target variables, one must use a technique tailored to computing the mean value of a large set of numbers safely to avoid overflow errors. Second, for large datasets we must have an efficient method to calculate encoded values. A naive approach where we take a pass over the entire dataset to compute the mean value of labels for each and every row is an $O\left(n^2\right)$ operation that is impractical for large datasets. We speculate that one should be able to compute all the possible leave-one-out encoded values of a categorical feature ahead of time, and store them in a look-up table indexed by the value of the categorical variable, and the value to be left out for efficient encoding. If there are so many values for the target variable that such a scheme is not feasible, then one may need to use an alternative encoding technique. This scenario is more likely the large datasets.

The first interesting result Nasir and Saleem give in [62] is the result of an experiment they conduct to select an encoding technique. For this experiment Nasir and Saleem compare ten encoding techniques. They pre-process data from the NSLKDD [71] dataset with each of the ten encoding techniques, and then feed the pre-processed data to Random Forest. Naseer and Saleem find that they get the highest accuracy from Random Forest when they use the leave-one-out encoding technique. We include a table from [62] that summarizes these results in Table 4.

Naseer and Saleem illustrate an interesting method for selecting an encoding technique. They use a classifier with fast training time to rank encoding techniques. One can inspect the training time column in Table 4 to see that this entire experiment has a total running time of about three minutes. However, Naseer and Saleem do not stop here. Their goal is to employ deep neural networks in a computer network intrusion detection system. Therefore, Naseer and Saleem compare the performance of five different classifiers on the leave-one-out encoded NSLKDD data. We include a copy of results from [62] in Table 5.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Table 4 Table comparing Random Forest accuracy for various encoding techniques, reproduced from [62]

Encoding scheme	Dimensionality	Training time (s)	Average training score	Score StDev
BackwardDifference	81	9.445193	0.961925	0.002291
BinaryEncoder	13	9.234833	0.962050	0.002472
HashingEncoder	8	20.524086	0.918650	0.002197
HelmertEncoder	81	9.418384	0.962100	0.002359
OnehotEncoder	84	8.884236	0.961950	0.002361
OrdinalEncoder	3	8.443738	0.961950	0.002513
SumEncoder	81	9.405340	0.961975	0.002560
PolynomialEncoder	81	9.642599	0.962000	0.002327
BaseNEncoder	13	10.734352	0.961925	0.002342
LeaveOneOutEncoder	3	8.746265	0.962150	0.002444

Table 5 Classifier performance results, reproduced from [62], classifier names are abbreviated as follows: Deep Convolutional Neural Network (DCNN), Radial Basis Function Support Vector Machine (RBF SVM), Multi-Layer Perceptron (MLP), Extreme Learning Machine (ELM), Radial Basis Function (RBF); NSLKDDTest+ and NSLKDDTest21 are partitions of the NSLKDD dataset [71] for evaluating trained models

Classifier name	AuC for NSLKDDTest+	Classifier name	AuC for NSLKDDTest21
DCNN	*0.965*	DCNN	*0.926*
Random-Forest	0.958	RBF SVM	0.867
RBF SVM	0.920	k-NN	0.825
Decision Tree	0.915	ELM Generalized	0.807
MLP	0.887	ELM RBF	0.803

We see in Table 5 that Naseer and Saleem obtain the best performance with a DCNN when they use leave-one-out encoding to pre-process its input data. One important note on how Naseer and Saleem use the NSLKDD data in the DCNN is that they reshape observations in the processed NSLKDD data into 32×32 grayscale images. We consider these steps to be elements of a pre-processing technique for using categorical data in neural networks. Naseer and Salim write that the way they convert the leave-one-out encoded data to an image is simply to triplicate each 41-element observation to obtain a list of 123 elements. Next they concatenate the first five elements of the observation to obtain a list of 128 values, and then they replicate this list 8 more times to obtain a 1024 element list that they reshape into a 32×32 array that may be interpreted as a 32×32 grayscale image.

Naseer and Saleem's work is an example of a composition of encoding techniques. First the authors use Leave-one-out encoding, then convert their leave-one-out encoded data to images for use in a DCNN. We see an opportunity for future research to employ this composition technique on other types of qualitative data.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Hashing

In our survey, we find researchers sometimes use hash functions to convert categorical values to numeric values for input to deep learning algorithms. Hashing is on the borderline of determined and algorithmic techniques, since hash algorithms can be rather involved. However, a hash algorithm is deterministic, and they do not require something like a pass over all possible inputs before they are ready to produce output values. For these reasons we include hashing in the determined category of techniques for using qualitative data in neural networks. In the context of this work, we use the definition of hash function from MacKay [72]. MacKay defines a hash function as a pseudo-random function that maps an N-bit string to an M-bit string where N is smaller than M. However, we relax the constraint that the input to the hash function be strictly longer than the output of the hash function. We find in practice that hash functions such as the built-in Python `hash` function, or the `md5sum` function available in Debian Linux distributions behave in this manner. In practice, we find that both `hash` and `md5sum` accept values as small as a single character and produce a longer output.

In "Deep learning with Python" [22], Chollet mentions hashing in Chapter 6. Chollet writes that one may choose to use hashing in lieu of One-hot encoding when the number of possible values one wishes to encode is so large that creating a mapping of the values to One-hot vectors is not practical. This is often the case when working with large datasets in a big data ecosystem. In [22] Chollet gives an example that uses Python's built in `hash` function. Python's `hash` function accepts a string as a parameter and returns an integer. We refer the reader to [73] for details on this function. Reverting to Chollet's coverage of the hashing technique in [22] we point out that we see hashing categorical values to integers as a viable alternative to Label encoding categorical values as vectors. Even when we map categorical values to vectors using some embedding technique, we first need to map the categorical values to some vectors that we can then embed in a lower dimensional space. We conjecture that mapping categorical values to integers using some hashing function is an equivalent and possibly more practical method as one does need to maintain an explicit mapping in one's own software. We give a caveat for using hashing that Chollet also mentions in [22], and that is that one must be careful to ensure that hash collisions do not frequently occur for distinct values of the categorical values. One may choose a hash algorithm with a large range of output values to lower the probability of collisions. We recommend [74] for some context on the ranges of popular hash algorithms.

In our review, we find research that takes hashing a step further. In [22] in the caption of Figure 6.2, Chollet writes, "...word representations obtained from One-hot encoding or hashing are sparse, high-dimensional, and hardcoded..." Chollet assumes a hash algorithm that does not change over time. We find research that challenges Chollet's assumption in "Hash embeddings for efficient word representations" [59]. In this work Svenstrup, Hansen, and Winther introduce hash embeddings. Hash embeddings incorporate hashing into trainable functions. That is Svenstrup, Hansen, and Winther discuss a technique where they apply several hash functions to input data, and use a weighted sum of the hash functions' output for an embedding. The authors treat the output of the hash functions as vectors. The authors then combine the vectors into a matrix that they then multiply by another matrix P whose entries are updated with a learning

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

algorithm that discovers the optimal values for the entries of *P*. In [59] the authors perform experiments that compare their hash embedding method to the hashing method Chollet presents in [22], as well as to several other embedding methods. The point of the experiments is not that one may use hash embeddings to obtain better performing deep learning algorithms, but that hash embeddings require fewer trainable parameters and achieve performance on par with competing techniques. Therefore, the conclusion that the authors wish the reader to come to is that their technique is better from a resource consumption perspective. This may resonate well with researchers working in big data. From our point of view, Svenstrup, Hansen, and Winther's is a unique composition of an automatic technique and a determined technique. We include it here because the focus of the work is employment of hashing functions.

We include this section on hash functions here because we find the idea of using hash algorithms in lieu of maintaining a look-up table to map categorical values to vectors to be an attractive technique, especially when working with big data. Hashing qualitative input values is a determined technique because the encoded value of hashed categorical data does not change with the context of the data.

Summary of determined techniques

Here, the determined techniques we cover are: Code counting, One-hot encoding, Label encoding, Leave-one-out encoding, and hash-encoding. Our choice of encoding techniques to include is based on our finding these techniques used in recent works where the authors use categorical data for input to neural networks.

Determined techniques are suitable for big data when they have a low running time complexity. Researchers may need to use sparse representations for One-hot encoded values when working with big data to avoid overtaxing storage resources. Hash encoding, and label encoding are also suitable for big data since we can defer producing the encoded value of a variable until we require it for input to a machine learning algorithm. Leave-one-out encoding may not be feasible for encoding qualitative values in big data datasets if the subset of labels has a high cardinality.

The common theme we see in works where researchers employ determined techniques is that using one of them is unavoidable. It is necessary to use a determined technique to transform qualitative data to numerical form for use in neural networks. Often researchers use a determined technique as the beginning of a more sophisticated algorithmic or automatic technique. As a best practice for selecting a determined technique we recommend avoiding leave-one-out encoding for datasets with label sets of high cardinality, and avoiding label encoding for input values to neural networks because that introduces arbitrary order and magnitude to those values.

An experiment comparing the effectiveness of Svenstrup, Hansen, and Winther's determined technique in [59], to that of Zhu and Han's work involving an algorithmic technique in [57], Zhu and Golinko's work involving the algorithmic technique in [19], and Guo and Berkhahn's work on an automatic technique in [3] is an avenue for future research. We find Svenstrup makes his hash embedding technique source code available in [60]. The researcher may want to use this source code to get started on comparing techniques.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Algorithmic techniques

Techniques for representing qualitative data that require extensive computations fall into the algorithmic class of data representation techniques. These algorithmic techniques involve applying a process to convert categorical variables to numeric values for neural networks. 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. This effort sets algorithmic techniques apart from techniques like One-hot encoding.

Latent Dirichlet Allocation

The first algorithmic data representation technique we cover is Latent Dirichlet Allocation (LDA). In "Leveraging deep learning with LDA-based text analytics to detect automobile insurance fraud" [18], Wang and Xu employ Latent Dirichlet Allocation in an algorithm to detect fraudulent auto insurance claims. The references of [18] contain a citation of "Latent Dirichlet Allocation" [2] that appears to be a seminal work on LDA. In "Latent Dirichlet Allocation" Blei, Ng, and Jordan write, "The basic idea [of LDA] is that documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words". In [18] Wang and Xu add to the description of LDA, writing, "LDA uses a joint distribution to compute the conditional distribution of the hidden variable under a given observable variable. The observable variable is a set of words, and the latent variable is the topics". From these two quotes we surmise that LDA is a method to represent a collection of documents as a distribution of topics where we infer the topic distribution using the distribution of words in documents. Both the direct encoding approach, such as One-hot encoding, and LDA involve executing a deterministic set of rules for converting categorical data to numerical form. However, algorithmic techniques, like LDA are more involved.

One distinguishing feature of algorithmic techniques is that they have hyper-parameters one must set. We see in [18] that the authors chose to set the number of topics parameter for LDA to 5, and that the authors' model associates 5 scores with each document, one for each topic. In the context of Wang and Xu's experiments, a document is the description section of an auto insurance claim. It contains a natural language description of the accident over which the insured is filing the claim. Wang and Xu write that they use One-hot encoding for other categorical variables for their models. In their experiments, Wang and Xu find that a deep neural network has the best performance when compared with support vector machine, and random forest models. Furthermore, in their conclusion they attribute the success of the deep neural network to increased sensitivity to the LDA values they compute for the claim descriptions. We reproduce a table of experimental results from [18] in Table 6.

Wang and Xu note that their DNN model performs worse when they remove the LDA values from its input. This fact supports the theory that the LDA values carry information about fraudulent and legitimate auto insurance claims. We see an opportunity for future research in applying Wang and Xu's technique to other types of insurance claim data.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Table 6 Results reproduced from [18], abbreviations are as follows: Support Vector Machine (SVM); Random Forest (RF); Deep Neural Network (DNN); true positive (TP); false positive (FP); F1 score (F1)

	TP rate	FP rate	Accuracy	Precision	F1
SVM	0.682	0.108	0.787	0.863	0.762
RF	0.823	0.198	0.812	0.806	0.814
DNN	0.910	0.082	0.914	0.917	0.913

Generalized feature embedding learning

Zhu and Golinko introduce an algorithmic technique for embedding categorical data in their paper entitled, "Generalized Feature Embedding for Supervised, Unsupervised, and Online Learning Tasks" [19]. The authors name their technique "Generalized feature embedding learning", or, GEL, for short. The first thing we notice about GEL that sets it apart from automatic techniques is that Zhu and Golinko's GEL does not employ something like gradient descent to update the embedding it provides. This is a distinguishing characteristic of algorithmic techniques. In [19], the authors present GEL as a systematic technique for transforming a dataset into an alternate representation that one may then use as input to a variety of supervised and unsupervised machine learning algorithms. GEL assumes that the instances of a dataset form a matrix X. It is important to note that the entries of X may have any data type; X is different from matrices that are familiar to us in that the elements of X may be text values, real numbers, ranges, or any kind of data that one might encode with One-hot encoding. GEL then transforms X into a matrix of binary values W. We perform One-hot encoding of the rows of X to obtain W. Therefore, W may have many more columns than X. GEL then performs a series of computations on W that expand it into a larger matrix of derived values S. GEL has a hyper-parameter k that holds the number of eigenvectors of S we use to form a matrix V. Finally, we multiply W by V^2 to obtain a matrix $\mathcal{F}$ that we use as input to a machine learning algorithm for training and evaluation. As practitioners, we are not so interested in the algebra of the calculations for GEL, but we are very interested in Golinko's R language implementation of GEL on Github.com [56] as we can directly use this code, or easily translate it into implementations in different languages for use in future experiments.

In [19], Zhu and Golinko compare the performance of various machine learning algorithms on two different types of datasets. We refer the reader to Section 5.1.4 of [19] for more detail on what Zhu and Golinko refer to as baseline methods. The baseline method is one where Zhu and Golinko apply minimal data preparation to the dataset for use with a collection of machine learning algorithms. The second kind of dataset that Zhu and Golinko employ to train the collection of machine learning algorithms is one where they first transform the dataset using GEL, and then train the machine learning algorithm with the transformed dataset. So, we conclude that we have an opportunity for future research to do a comparison of GEL to some other embedding technique or collection of techniques. In Figure 4 of [19] Zhu and Golinko show that using GEL to prepare the input for a collection of machine learning algorithms generally gives better performance than what they term the baseline method. However, one cannot use this result by itself to conclude how GEL performs in comparison with other embedding techniques.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Another interesting feature of GEL is that we can use it in an online learning scenario. For the reader who is unfamiliar with the term online learning, we recommend seeing [75], Section II, B for a precise definition. Also, we refer the reader to Section 5.2.4 of [19] for details on how the authors use GEL in an online learning context. Here we summarize that Zhu and Golinko show that preparing data with GEL, and then feeding that data to a Naive Bayes classifier, has better performance in an online learning task, than when they use the same classifier, but feed it with data from a baseline data preparation. We see an opportunity for future work in comparing GEL to other embedding methods in an online learning setting.

We give a caveat for using GEL with large datasets. In a big data setting, the step where we convert a dataset to matrix form, and then calculate the values of the eigenvectors may not be feasible. Therefore, another avenue for future research would be to determine whether or not one can split a large dataset into smaller partitions and apply GEL to each partition.

Convolutional neural networks for categorical data

In this section, we review another algorithmic technique in papers that Zhu is also an author on. We find "EDLT: Enabling Deep Learning for Generic Data Classification" [76] and "Convolutional neural network learning for generic data classification" [57]. Han, Zhu, and Li are the authors of both works. We do not find that the authors explicitly define the term EDLT but it appears to be an abbreviation of "Enabling Deep Learning for Generic Classification Tasks". Both papers cover the same topic of EDLT. EDLT is a technique for transforming rows of data that is in tabular form to matrices that we then use as input to convolutional neural networks.

Han, Zhu, and Li's EDLT technique begins with finding the matrix $\mathcal{M}$, which is the Pearson correlation matrix of all the features of some dataset $\mathbf{x}$, and a correlation vector of all the labels of $\mathbf{x}$ that we denote with $\mathcal{L}$. EDLT then uses $\mathcal{M}$ and $\mathcal{L}$ in an iterative process to generate a matrix $\mathcal{O}$ that we use to re-order and transform $\mathbf{x}$, one instance $\mathbf{x}_t$ at a time into synthetic matrices $\mathcal{F}(\mathbf{x}_t)$. After we use EDLT to obtain the $\mathcal{F}(\mathbf{x}_t)$, we use these matrices as input to convolutional neural networks. EDLT transforms instances of x, x_t, into matrices $\mathcal{F}(\mathbf{x}_t)$ such that the elements of the matrices $\mathcal{F}(\mathbf{x}_t)$ are spatially close to one and other if they are correlated. Convolutional neural networks are then suitable consumers of the $\mathcal{F}(\mathbf{x}_t)$ because convolutional neural networks excel in recognizing spatially correlated features in data such as images. In [57, 76], Han, Zhu and Li are able to show that convolutional neural networks show better performance in both supervised and unsupervised learning tasks if they use data that is treated with the EDLT technique versus what they term as baseline techniques.

According to Han, Zhu and Li in [57, 76] there are two baseline methods in the context of their work. The first baseline method is a random reordering of features. Han, Zhu, and Li name this baseline method RR. The second baseline method Han, Zhu, and Li use is a partial application of EDLT where we do not carry out the steps to optimize the reordering matrix $\mathcal{O}$ to globally maximize correlation between nearby entries of the synthetic matrix $\mathcal{F}(\mathbf{x}_t)$. As Golinko and Zhu do in, "Generalized Feature Embedding for Supervised, Unsupervised, and Online Learning Tasks" [19], Zhu and Li compare the performance of EDLT to baseline methods that are not methods we would consider to

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

be competitors to EDLT. We feel it is obviously an interesting area of future research to make the comparison.

The authors make the source code for the EDLT technique available in [58]. However, one should note that this source code is a mixture of Python and Matlab code. We see an opportunity for a contribution. The source code in [58] should be translated into a single language that one might easily employ EDLT as a library function.

We see a challenge to employ EDLT on large datasets similar to the one we find with GEL. EDLT's requirement to calculate the Person correlation of a large dataset may be prohibitively expensive for big data. An interesting topic for future research is a study on whether or not we can make EDLT amenable to big data in some way, such as partitioning the data in some manner so as to make the Pearson correlation calculation tractable.

Summary of algorithmic techniques

Algorithmic techniques may not be suitable for big data projects because they are computationally intensive. As a best practice for deciding to use an algorithmic technique, we recommend evaluating the running time of an algorithmic technique relative to a determined technique on a small sample of a dataset to gauge how much more time the algorithmic technique will require. One should also be able to compare the relative sizes of the datasets obtained with both the algorithmic and determined techniques to get a sense for how much space the algorithmic technique will require.

One common theme we notice about research on algorithmic techniques is that the authors present them as techniques to employ on a dataset directly. We did not notice any work where the authors recommend employing transfer learning when using an algorithmic technique. Indeed, it is unclear to us what, besides the encoded data itself, one would be able to reuse for separate machine learning tasks. In the long run, this may be an Achilles heel in algorithmic techniques.

We find researchers use algorithmic techniques to encode categorical variables to enhance the performance of some models. We suspect that algorithmic techniques tend to be more domain specific because they may exploit aspects of data in particular domains. For example, LDA operates on collections of documents, and EDLT operates on tabular data. GEL stands out in this regard because it can operate on categorical data in general.

Automatic techniques

Automatic techniques for using categorical data in neural networks incorporate finding a data representation into the neural network's learning process. Automatic techniques are appealing because they are more general-purpose than determined or algorithmic techniques. For example, Mikolov et al.'s Word2vec algorithm is reused in many domains. We cover some of these re-uses below. Due to their proven potential for reuse we include details on several works using automatic techniques, whether they are employed in a transfer learning manner, or directly. For big data applications, one may prefer to employ automatic techniques via transfer learning. This is because automatic techniques for encoding qualitative data often involve optimizing models with a number of parameters large enough that the encoding process becomes time-consuming.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Distributed representation learning

One body of machine learning research literature uses the term “distributed representation” to refer to the concept of mapping qualitative data to vectors of real numbers. We find that when the term distributed representation appears in the literature, that it refers to an automatic technique for using qualitative data in neural networks. Our research indicates the lineage of the term is as follows: in “A neural probabilistic language model”, Bengio et al. [27] propose a technique for fighting what the authors refer to as the “curse of dimensionality”. The curse of dimensionality is an expression that refers to how the number of observations and labels one needs to accurately model a dataset grows as the number of labels in the data set raised to the power of the number of features. To use Goodfellow, Bengio, and Courville’s terminology, if we have a model that uses d features and has v distinct output values, then in order to accurately train and evaluate the model we require $O\left(v^d\right)$ observations with their labels [1]. For example, in [27] the authors point out the set of all possible 10-word phrases one may draw from a 100,000-word vocabulary contains $10^{50} - 1$ elements. To accurately model such a dataset, we require $O\left(10^{50}\right)$ labels and observations. Current storage technology capacity is on the order of terabytes $\left(10^{12}\right)$, or perhaps petabytes $\left(10^{15}\right)$. Therefore, it is not possible to store a sample of such a large dataset, such that the sample is large enough that we can accurately evaluate the model’s performance. Hence, Bengio et al. propose a technique for mapping words that appear in text documents to vectors. This technique incorporates machine learning to do the mapping. Bengio et al. refer to the collection of vectors as a distributed representation for words. They summarize their technique for finding distributed representations for words in natural language as follows:

1. *Associate with each word in the vocabulary a distributed word feature vector (a real-valued vector in $\mathbb{R}^m$),*
2. *Express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence, and*
3. *Learn simultaneously the word feature vectors and the parameters of that probability function.*

Bengio et al. emphasize the term “probability function” because their paper makes the contribution of showing how we can use neural networks to implement a function that computes the probability of a sequence of words. Also, the third step in the technique Bengio et al. describe is a key characteristic of automatic techniques; the process for embedding the qualitative data is tightly coupled with the algorithm that finds the parameters of a probability function. It is worth noting that the probability function Bengio et al. write about is a neural network.

One work related to Bengio et al. [27] that stands out to us due to the number of times we see references to it is “Distributed representations of words and phrases and their compositionality”, by Mikolov et al. [9]. This work builds on Mikolov’s earlier, oft cited work, “Efficient Estimation of Word Representations in Vector Space” [43]. These two works are both published in 2013, with [43] preceding [9] by a few months. We use Google Scholar to check the number of references to [43], and [9]. The numbers of references are 11,298, and 13,924, respectively. Furthermore, we use our research facility database to search the term, “distributed representation”, for a sample

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

of works related to this keyword. We take the first 49 works that the search engine returns, and check the references in these works. We use no other search parameters save for the search term "distributed representation". Approximately 71%, or 35 out of 49, of the works refer to Mikolov et al.'s [9] or [43]. So not only is the work of Mikolov et al. widely cited, but also specifically in the context of work where the term, "distributed representation", is a subject. Moreover, we are interested in the techniques Mikolov et al. employ because these are techniques for using categorical data as input for deep learning algorithms. Therefore, it behooves the reader to have a good understanding of Mikolov et al.'s work because other researchers have extended it in so many ways. In addition, many of the works citing Mikolov et al. include domain specific embeddings that researchers may be able to re-use in their work.

"Efficient estimation of word representations in vector space" [43] is interesting because this is the work where Mikolov et al. introduce two techniques for word embeddings: skip-gram and continuous bag of words (CBOW). The objective function for the deep learning algorithm in CBOW is a function of how well the embedding vectors of surrounding words predict the word they surround. In [43] the authors refer to the i words that surround a given word as, "the context", and the word itself as, "the center word".

We find [9] more interesting because it has more content to it. This work focuses primarily on the skip-gram approach to word embedding. A key result reported in the work is that skip-gram produces representations of words that do better than previous representations, and the skip-gram algorithm has a faster running time than its competitors. Thankfully, there is a footnote at the end of [9] that links to source code for running both algorithms. The entry of the references section in this work [44], has the uniform resource locator (URL) for this source code. If one is curious as to precisely how skip-gram and CBOW learn vector representations for natural language words as categorical data, the file Word2vec.c in [44] contains these calculations. Another important file in [44] is distance.c, which serves as an example of how to use the mappings that CBOW and skip-gram learn, to convert categorical data to vectors in $\mathbb{R}^n$. Here n is smaller than the number of dimensions one would need to use One-hot encoding. The points that the reader should gather from [9, 43] are:

- Both skip-gram and CBOW are algorithms that employ neural networks to learn a mapping of words in natural language, i.e. categorical data, to vectors in $\mathbb{R}^n$,
- Skip-gram learns a mapping with a loss function that measures how well skip gram predicts the words that surround the current word,
- CBOW learns a mapping with a loss function that is a measure of how well CBOW predicts the current word, given the words that surround the current word,
- In practice, Mikolov et al. [9] find that when they use the vectors for categorical data that skip-gram learns, they get the best results, and skip-gram consumes fewer resources than its competitors, and
- One can do arithmetic on the vectors in $\mathbb{R}^n$ that skip-gram or CBOW learn in such a way that the vector arithmetic is compatible with human intuition.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

To expand on the last point on vector arithmetic above, we give an example from [9]. If we use vectors from the Word2vec embedding algorithm, the result of subtracting the vector for "man" from the vector that represents, "king", and adding the vector for, "woman" to the result gives a vector that is closest to the vector that skip-gram or CBOW learns for, "queen". This is one result that inspires much subsequent research. Researchers find that one may use qualitative data, other than words, as input to deep learning algorithms and obtain similar, intuitive results. In the next section, we study the progeny of Mikolov et al.'s discoveries.

There are many implementations of skip-gram and CBOW that researchers should take advantage of and re-use, if transfer learning is feasible. One can easily find implementations on http://github.com. At the time of this writing a search on http://github.com for the term, "Word2vec", garners 6320 results. Suffice it to say, Word2vec is an automated technique for converting categorical data for use in neural networks.

Optimal embedding vector length

Word2vec discovers representations for categorical, natural language data automatically. However, it is up to the user of the Word2vec algorithm to select the number of dimensions the embedded vectors will have. In our search for material for this work, we find Yin and Shen's work, "On the Dimensionality of Word Embedding" [77]. Yin and Shen present a method to find an optimal value for the dimensionality of word embeddings. Word embeddings are a special case of entity embeddings. Therefore, we see opportunities for future work to apply Yin and Shen's method to entity embeddings.

Yin and Shen's method for estimating the dimension of a word embedding is to estimate the Pairwise Inner Product (PIP) loss of embeddings as we increase the number of dimensions in the embedding. Yin and Shen show that the value of the PIP loss versus dimensions reflects a trade off between the bias and the variance of the word embedding algorithms. This trade off implies the existence of an optimal value for the number of dimensions one may select when choosing the number of dimensions in the domain of an embedding.

Yin and Shen qualify their technique as useful for embedding algorithms that are based on matrix factorization. In [77] the authors write that GloVe [21] and Word2vec [43], "... learn word embeddings by optimizing over some objective functions using stochastic gradient methods, they have both been shown to be implicitly matrix factorizations." We feel it lies in the realm of future work to check if other embedding techniques for embedding categorical data other than words might be characterized as implicit matrix factorizations. This would be a strong indicator that applying Yin and Shen's method for calculating the optimal number of dimensions in an embedding applies. We might be able to improve upon existing results that use embeddings of categorical data by setting the number of dimensions according to the value that Yin and Shen's method finds.

Our discovery of the work [77] leads us to two related works regarding the dimensionality of word embedding. First of all, we find a longer work that Yin writes on the same subject as [77], "Understand Functionality and Dimensionality of Vector Embeddings: the Distributional Hypothesis, the Pairwise Inner Product Loss and Its Bias-Variance Trade-off" [78]. This work is far longer than [77]. We see that [77] was submitted to arxiv.org December 2018, and [78] was submitted to arxiv.org March 2018, so [77]

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

summarizes the greater detail of [78]. Therefore, the interested reader may wish to consult [78] for clarification.

Researchers interested in applying Yin and Shen's method for finding the optimal dimensionality for an embedding should also know about the repository [79] that has an implementation of the techniques that Yin and Shen cover in [77, 78]. The implementation in this repository is in the Python programming language. Hence, we have source code for using Yin and Shen's technique, which makes it easy to incorporate into existing projects. We feel that the degree of generality of Yin and Shen's technique is unclear. It may apply to more than just embedding techniques for natural language data. If the technique proves to be generally applicable for entity embeddings then an implementation similar to the code in [79] should become part of any library for entity embedding.

Reusing natural language processing techniques

In "Using word embedding technique to efficiently represent protein sequences for identifying substrate specificities of transporters" [45], the authors Nguyen et al. reuse skip gram and continuous bag of words embedding techniques. They treat amino acids as letters, and the proteins the amino acids comprise as words. Nguyen et al. then apply natural language processing techniques to a database of protein data that they treat as their corpus. This illustrates an interesting possiblity with embedding techniques. If a pre-computed embedding is not available, one may find it is possible to reuse an existing algorithm from another domain to produce one suitable for the machine learning task at hand.

There are myriads of derivative works and refinements to Mikolov's work and an exposition of those works is a topic worthy of a survey in its own right. We give Nguyen et al.'s work as one example of how general Mikolov's technique is and its applicability to other domains beyond natural language processing. The success of Mikolov et al. is rooted in an automatic technique for embedding qualitative data for use in neural networks (Word2vec), and shows that automatic techniques are attractive due to their potential for reuse. This is especially true in big data environments where computing a new embedding of a large dataset may be too time-consuming to be practical.

Entity embeddings

Guo and Berkhahn employed an automatic technique to win third place in a 2015 Kaggle competition. The technique is known as entity embedding. Entity embedding is a method to embed categorical variables in real-valued vector spaces [3]. Guo and Berkhan's success gives us an indication that neural networks are perhaps capable of operating on all sorts of qualitative data, if we can map that data to vectors we can use as input to neural networks effectively. In [3] Guo and Berkhahn do not reveal a new technique. In fact, we find that in the source code for their Kaggle competition entry [35] Guo and Berkhahn use the Keras library Embedding Layer function. The Keras Embedding Layer is a convenient means to automatically find a dense encoding for qualitative data. Nevertheless, we believe the embedding technique that Guo and Berkhahn use shows promise. We believe their success serves as an endorsement of the technique.

The abstract of [3] mentions the Rossman Store Sales [80] Kaggle competition. Part of [3] is an explanation of the workings of their solution to a Kaggle competition

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

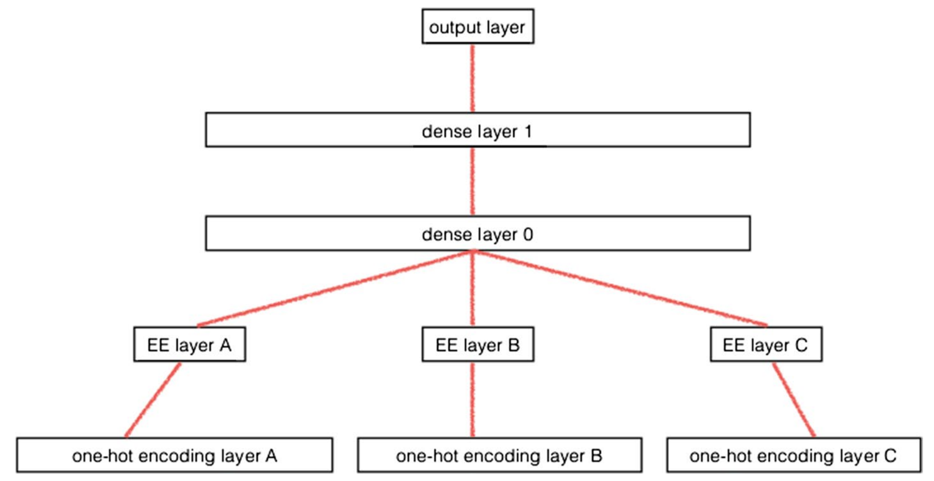

Fig. 2 Image copied from [3]. The authors perform experiments with and without embedding layers. The schematic here shows the Author's approach when they use embedding layers. Categorical variables are embedded separately, then the output of the embedded layers are concatenated and fed forward to dense layers

where the authors won third place using a model that employs ideas they expose in this paper. The goal of the Kaggle competition is to discover the best model for predicting the sales, by store, for stores in the Rossman Drugstore chain. The source code of Guo and Berkhahn's entry into the Kaggle competition, "Entity-embedding-rossman" [35], is publicly available. In this repository, we see the authors use Keras [20] embedding layers repeatedly.

We are unable to find a seminal work that explicitly documents a theoretical justification for when encoding categorical variables with embedding layers can improve a model's performance. Therefore, we see an opportunity for future work to provide this theoretical justification. We include a summary of the history and references here for those who might be interested in researching a theoretical justification for the performance enhancements Guo and Berkhahn report in their results in [3]. One can surmise that events happen in rapid succession. We can see from the Keras source code repository commit history [20] that the authors add the embedding layer to Keras starting in April 2015. We see in Arxiv.org's listing for Gal and Ghahramani's paper that their first submission is in December 2015, so the Keras embedding functionality must have existed in Github before Gal and Ghahramani submitted their work to Arxiv.org.

Therefore, we conclude that the attribution implicit in the Keras documentation authors' listing of Gal and Ghahramani's work is correct. Furthermore, we conclude that Gal and Ghahramani's is part of the succession of ideas that leads to Guo and Berkhahn's work. And Guo and Berkhan's work depends on Gal and Ghahramani's work in that they use Keras embedding layers to embed categorical variables' values for use in a neural network.

Guo and Berkhan's design is sufficiently general that we feel compelled to explain its details. We include a copy of a diagram from [3] in Fig. 2 to help us explain the architecture.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Table 7 Results copied from Guo and Berkhahn [3], MAPE is mean absolute percentage error, EE is entity embedding KNN is K-Nearest Neighbors, in all cases using entity embedding for encoding gives lower error, transfer learning employed for all models except neural network where direct learning is employed

Method	MAPE	MAPE (with EE)
KNN	0.290	0.116
Random forest	0.158	0.108
Gradient boosted trees	0.152	0.115
Neural network	0.101	0.093

Guo and Berkhahn's architecture is elegant. The architecture diagram in Fig. 2 aligns with the actual neural network Guo and Berkhan implement in the source code in reference [35]. Therefore, understanding the diagram will assist the reader in understanding the source code. The primary idea for Guo and Berkhahn's architecture is to use One-hot Encoding for categorical variables, then feed the encoded categorical values to embedding layers. The output of all the embedding layers is then concatenated and fed to dense layers. Results Guo and Berkhahn present in [3] highlight the utility of embedding layers. Furthermore, the results show that using embedding layers can improve performance when employed directly, or employed with transfer learning. The results in Table 7 for "neural network" are where Guo and Berkhahn directly employ the encoding that the embedding layers provide. The results for other machine learning algorithms are where Guo and Berkhahn employ transfer learning. Guo and Berkhahn report the best performance when using direct employment with neural networks.

Guo and Berkhahn's results may lead one to believe that entity embedding as a technique for encoding categorical variables is something one should always do. However, we raise two points one should consider when digesting their results. The first is that Guo and Berkhahn report one metric, mean absolute percentage error (MAPE). They define MAPE as

$$\left|\frac{Sales - Sales_{predict}}{Sales}\right|. \tag{7}$$

We think it would be interesting to see how much of the variance in MAPE is explained by the model used, versus the use of entity embeddings.

We see some opportunities for future work given the results Guo and Berkhahn present. Guo and Berkhahn provide the source code for their experiments in [35]. This makes extending their work easier. The first opportunity for future work that we see is computing the analysis of variance we mention above. Another opportunity for future research is to add the determined encoding technique as a level to the experiments Guo and Berkhahn perform in [3]. Guo and Berkhahn's implementation is written in Python, and the Scikit-learn Python library includes modules for seven different determined encoding techniques. Therefore, adding these different techniques is a straightforward effort.

Guo and Berkhan's work in [3] is a clear example of how to use an automatic technique to encode categorical variables, and how to employ it using transfer learning, or directly into a larger neural network. When it is possible to use a library such as Keras

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

for implementation, embedding layers require little effort to implement. Guo and Berhahn's results show that using embedding layers may improve performance. Therefore, practitioners may wish to include an embedding layer for any One-hot encoded categorical variables they are working with since it only requires adding an extra layer to an existing neural network.

Entity embedding for anomaly detection

One interesting application of entity embeddings we find when searching for work using the term, "entity embedding", is "Entity Embedding-based Anomaly Detection for Heterogeneous Categorical Events" [30] by Chen et al. Here we have an automatic technique for anomaly detection. We find the authors report that they use methods similar to Mikolov et al. However, we do not find that the authors share the source code for their experiments so it is not easy to leverage their research in other avenues. On the other hand, the work is interesting, since it describes a technique for using categorical data for input into deep learning algorithms. The authors do not give precise details on how they map categorical features that constitute events in some computing environment. Chen et al. write that in the context of their research, an event is a collection of categorical values describing something that happens in a computer system. They treat events as categorical values that they then embed into a vector space. Since the authors do not provide their source code, we speculate that they use One-hot encoding for the event data, and then use something like a Keras embedding layer to implement the embedding. After embedding the authors indicate they optimize the maximum likelihood objective

$$\underset{\theta}{\operatorname{argmax}} \sum_{e \in D} \log P_\theta(e) \tag{8}$$

where e is an event, $P(\cdot)$ is the probability of $\cdot$, and D is the set of events.

Our understanding of the results Chen et al. present is that they give enough detail to earn a place to present this work in the International Joint Conference on Artificial Intelligence, but not quite enough detail for one to be able to reproduce their results easily. However, since the work falls under the subject of this review we feel it is worth mentioning. One thing researchers should note is that the technique Chen et al. cover in [30] is a way to assign a probability to a pair of events that occur in sequence. This is interesting and reusable in future research. The technique works as follows: let v_{a_i}, and v_{a_j} be the embedding vectors for two entities. Note that we are free to think of v_{a_i}, and v_{a_j} as lists of categorical data mapped to vectors in $\mathbb{R}^n$. Furthermore, we define the event e to be an ordered collection of m entities. Then we can define a scoring function $S_\theta(e)$,

$$S_\theta(e) = \sum_{i,j:1 \le i \le j \le m} w_{ij}\left(v_{a_i} \cdot v_{a_j}\right). \tag{9}$$

where w_{ij} is a parameter that we can adjust via some machine learning algorithm. We can furthermore define a loss function for training that machine learning algorithm that determines values for w_{ij} that give the best performance. Hence, the algorithm is capable of assigning importance to pairs of events. Under proper conditions, this algorithm learns to produce significantly different outputs when pairs of events occur

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

simultaneously. One should be able to abstract this technique to apply to entities in general, and not simply events.

Since Chen et al. do not provide source code for their experiments we need to do some reverse engineering. We see an opportunity for future research to use Mikolov and Sutskever's source code from [44] and incorporate the ideas in [30] in detecting fraudulent transactions. We treat the transactions as Chen et al. treat entities in [30], and we use the time of the transaction to place an ordering on entities. It is important to note that we do not find that Chen et al. report an embedding technique, so to get started on the future research we are suggesting, we would use Keras' embedding layer [20].

Chen et al.'s work also shows how using an automatic technique for encoding qualitative data, in this case entity embedding, is reusable in different domains. Here we see the same technique used for estimating sales in drugstores, and for detecting anomalies in computing systems.

Neural probabilistic outlier detection

"A Neural Probabilistic outlier detection method for categorical data" [50], by Cheng, Wang and Ma presents an automatic technique for finding outliers in datasets with categorical data. Outlier detection is a suitable perspective to take for classifying highly imbalanced datasets. With imbalanced datasets, minority class observations may be identifiable as outliers. In this case one may be able to employ an automatic technique for representing categorical data in a neural network for the unsupervised task of outlier detection. If one finds an acceptable overlap between the set of outliers and the minority class, then one also has a classifier that can separate a dataset. Hence, Cheng, Wang and Ma's automatic way to learn a data representation for categorical variables for outlier detection is also a potential classifier for imbalanced data.

The architecture Cheng, Wang, and Ma present is a three-layer neural network. The simplicity of the architecture is attractive. Cheng, Wang, and Ma propose a loss function that guides their model to learn parameters such that the loss function will output large values for outliers, and small values for observations that are not outliers. We do not find source code for Cheng, Wang, and Ma's work, so we must rely on the theoretical discussion they give in [50]. The loss function Cheng, Wang, and Ma propose is

$$\mathcal{L} = -\frac{1}{n}\sum_{o\in\mathcal{O}} \log P\left(o^i \middle| con\left(o^i\right)\right) \tag{10}$$

We use Cheng, Wang, and Ma's terminology to define the elements of Eq. 10. The *i*th attribute value of an object is o^i. The context of o^i is $con\left(o^i\right)$. An attribute's context is the value of all the other attributes that we observe along with it in a particular instance of a dataset. Therefore, $P\left(o^i \middle| con\left(o^i\right)\right)$ is the conditional probability of the attribute value given the other attributes along with it. To compute the loss function Cheng, Wang, and Ma sum the negative logarithm of the conditional probabilities over all the objects in $\mathcal{O}$. Cheng, Wang, and Ma define $\mathcal{O}$ as a set of objects with categorical attributes. Since we are taking the negative logarithm of the conditional probabilities, this sum will be large when its components are small. When the components of the sum are small, this implies that the conditional probabilities are small. When the conditional probabilities are small, this means we have an attribute that is not likely to occur with other attributes, or if

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

we prefer, an outlier. Hence, the three-layer Neural Network that Cheng, Wang, and Ma propose is an algorithm that automatically learns to partition elements of a dataset into outliers, and non-outliers. It learns via backpropagation of the loss function defined in 10. They name this algorithm Neural Probabilistic Outlier Detection (NPOD).

In order to evaluate NPOD, Cheng, Wang, and Ma compare NPOD's ability to detect outliers to eight algorithms for detecting outliers in categorical data. They use 12 publicly available datasets as input to the nine algorithms and report that their technique achieves the best Area under the curve (AUC) score for seven of the twelve datasets.

The simplicity and elegance of Cheng, Wang, and Ma's approach makes it an attractive choice for working with categorical data in neural networks, but it is limited in scope. On the one hand, it is a technique that is capable of finding outliers in data that employs unsupervised machine learning. However, we find it difficult to conceive of how to employ it as a general purpose technique for transforming categorical data for use in neural networks. We choose to include it in this study because it involves neural networks, and feeding categorical data into them. We are aware of important machine learning tasks, such as medicare fraud detection Johnson et al. cover in [6], that can be solved from an outlier detection perspective.

Hierarchical gated recurrent units

"Deep neural models for ICD-10 coding of death certificates and autopsy reports in free-text" [48], by Duarte et al., presents an intricate model for automated labeling of medical documents. We consider the author's technique for encoding qualitative data an automated technique because at a high level it relies on entity embedding via Keras Embedding layers. However, the details in their paper, and source code reveal a blending of techniques. Their approach involves nested components. The source code in [49] that accompanies the paper [48] is most helpful in guiding one to understand Duarte et al.'s approach. The authors use a Keras Embedding layer to encode the words in the fields of the clinical forms that constitute the input to their model. They then connect the embedding layer to a Bi-directional Gated Recurrent Unit (BiGRU) that is capable of detecting patterns in the word embedding layer. Duarte et al. then connect the first BiGRU to an attention layer that is able to recognize patterns in the sequences of word embeddings in the BiGRU layer to form a representation of a sentence. In addition, Duarte et al. use a duplicate word embedding layer that they combine with the output of the attention layer to feed into an output layer. The output layer has three components that give a representation of various ICD-10 codes. We provide the reader a copy of a diagram of Duarte et al.'s in Fig. 3. Duarte et al.'s design is intricate, and specialized for the purpose of generating ICD-10 codes from autopsy reports and death certificates. Their use of a Keras embedding layer implies that their model automatically learns a representation of its input.

Duarte et al. compare six variations of the model we show in Fig. 3 with two variations of a Support Vector Machine (SVM) model for evaluation. They report their model outperforms SVM, but that their model requires a trade off between accuracy, precision, and recall. The trade-off is in the choice of initialization technique for the output layers of their model. Duarte et al. get the highest values for accuracy when

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

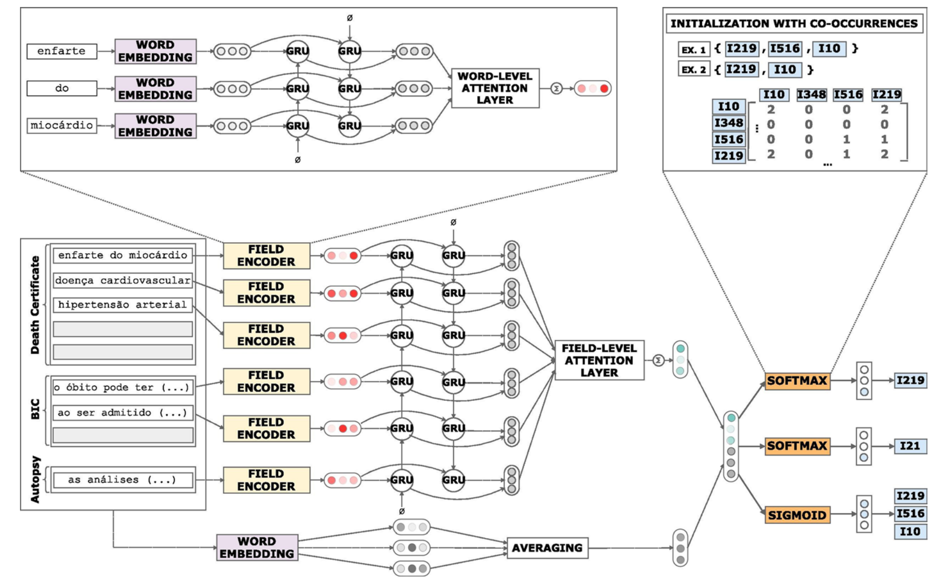

Fig. 3 Architecture diagram copied from Duarte et al. [48]

they initialize the output layer of their model with weights they derive from the frequencies of co-occurrences of ICD-10 codes using the Apriori algorithm. On the other hand, Duarte et al. report the highest precision and recall scores for the configuration of their model where they initialize the weights of its output layers with Non-Negative Matrix Factorization (NMF). The interested reader should consult the source code in [49] to see precisely how Duarte et al. employ these initialization techniques.

Duarte et al. present an automated technique for encoding qualitative natural language clinical data for use in a classifier that categorizes the data into ICD-10 codes. Since they provide the source code [49] to accompany [48], we have easy opportunities to extend their work. One possibility is replacing the embedding layer they use for representing text with another encoding technique such as Latent Dirichlet Allocation that Wang and Xu use in [18]. Duarte et al.'s work is a clear example for researchers interested in employing representation learning to use qualitative data in neural networks.

Transfer learning

Instead of reusing an algorithm to learn data representation, we can transfer an existing data representation for use in another algorithm. This technique saves training time, since we do not have to learn a new representation for input data. Bengio, Courville, and Vincent define transfer learning as, "... the ability of a learning algorithm to exploit commonalities between different learning tasks in order to share statistical strength, and transfer knowledge across tasks" [81]. Practically, in the context of neural networks, transfer learning amounts to using part of a neural network that we train using one algorithm, for use in another. This is the meaning of the term transfer learning in the context of this study. In this section we describe one work that uses transfer learning, as a way to use categorical data as input for a neural network.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Before we cover the example of transfer learning, we must clarify that our definition of "transfer learning" differs from another definition present in current literature. In works such as [82], "transfer learning" refers to training a supervised machine learning algorithm with a labeled dataset, and then feeding the trained algorithm input values from a different dataset. This is not the technique we refer to as transfer learning here.

The sent2affect algorithm

In "Deep learning for affective computing: Text-based emotion recognition in decision support" [47], Kratzwald et al. use transfer learning for classifying the emotional content of various samples of texts from different datasets. Kratzwald et al. reuse the text embedding that the GloVe algorithm learns for the machine learning task of recognizing the emotional nature of a body of text.

The samples of text that Kratzwald et al. use in their experiments are labeled according to the emotional content of the text. Krazwald et al. write that they use a factorial experiment design to evaluate the performance of different machine learning algorithms' performance on various datasets. Krazwald et al. use five datasets, and six different machine learning algorithms. The datasets are samples of text from literary tales, twitter messages related to an election, self-reported experiences, headlines, and a general collection of twitter messages. The classifiers Kratzwald et al. use are Random Forest, Support Vector Machine (SVM), Long Short-term Memory (LSTM), and Bi-directional Long Short-term Memory (BiLSTM). Furthermore, Kratzwald et al. explore two architectural variations on LSTM and BiLSTM. In one configuration they use a randomly initialized embedding layer. In this configuration the authors do not apply transfer learning. This is necessary because they are controlling for the effect of using transfer learning to assess the impact of it in their experiments. In the second configuration of their LSTM and BiLSTM models, Kratzwald et al. use a pre-trained word embedding from a GloVe model to convert the text they classify into vectors. Most notably, Kratzwald et al. report that the experiments where they record the highest F1-scores for classifying the emotional content of text are the trials involving BiLSTM with the pre-trained GloVe word embedding. In fact, this configuration is the one that achieves the highest F1 score on every dataset.

Another important fact about the work Kratzwald et al. report in [47] is that the corpus they apply the GloVe algorithm to, to obtain their pre-trained word embedding is a dataset from a Kaggle Twitter Sentiment Analysis contest in 2017 [83]. Their approach is to train GloVe on the labeled dataset from the Kaggle Twitter Sentiment Analysis contest where Twitter messages (tweets) are labeled as positive or negative. They then replace the output layer above an LSTM or BiLSTM with a layer that has more output values, and train the new model again with one of the different datasets we mention above. It is important to note that Kratzwald et al. do not freeze the parameters of their LSTM or BiLSTM once they are finished training it with the Kaggle Twitter Sentiment Analysis contest dataset.

Krazwald et al.'s paper is a good example of why we use the term spectrum to write about techniques for using categorical data in neural networks. They start with reusing an existing word embedding algorithm for sentiment analysis which we would consider

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

an automatic technique. Ultimately, they apply transfer learning to reuse a data representation to achieve the best performance in their experiments.

Autoencoders

Another interesting application of embeddings is with time series data, and autoencoders. In [38] Kieu et al. use an embedding algorithm similar to Guo and Berkhahn's [3]. Kieu et al. show that their model improves with an embedding layer. They encode categorical values using an embedding to capture contextual information to enhance time-series data. They show that using the embedded contextual information improves the accuracy of their deep neural network algorithm for outlier detection. Autoencoders provide another form of automatic technique for using categorical data in neural networks.

Lei et al. [40] use an autoencoder to embed medical records in a vector space. Goodfellow, Bengio, and Courville cover autoencoders in Chapter 14 of [1]. Autoencoders are pairs of neural networks that together learn the identity function of their input. We do not find modeling the identity function interesting. However, the process of training the autoencoder to compute the identity function causes the hidden layers of the autoencoder to develop a representation of the input domain. In order to accommodate time-series data, Lei et al. use recurrent neural networks in their autoencoder. Lei et al. treat medical records as time series data. However, their approach is different from that of Kieu et al. because Lei et al. use an autoencoder to do their embedding, whereas Kieu et al. first embed their input data using an embedding layer, and use the output of the embedding layer as input to an autoencoder. The reader should note that according to the way Lei et al. describe their approach, they use a sequence of time series data to produce one feature vector that is the output of the autoencoder. That is interesting because the model produces an embedding that is sensitive to the way the data that represents a patient changes over time, and thus the embedding technique is capable of producing similar representations of patients when their records change in the same way over time.

One area for further research is to compare the effectiveness of Kieu et al.'s and Lei et al.'s techniques. Since Kieu et al. found that using the embedding layer enhances the performance of their algorithm, it could be the case that we could improve the algorithm that Lei et al. devise if we add an embedding layer that would then provide the input for an autoencoder. On the other hand, it may be the case that we could feed an autoencoder the data Kieu et al. use, without an embedding layer, and obtain a result with performance comparable to what Lei et al. report.

Dealing with categorical values in time series data as inputs for deep learning algorithms presents its own unique challenges. However, we find researchers use different architectures to accommodate time series data. Which approach is more advantageous is an avenue for future research. Another opportunity for future research is to compare automatic techniques with a controlled set of machine learning algorithms. One could design a set of experiments to feed a fixed group of algorithms data from autoencoders, or entity embedding to determine which technique is better.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

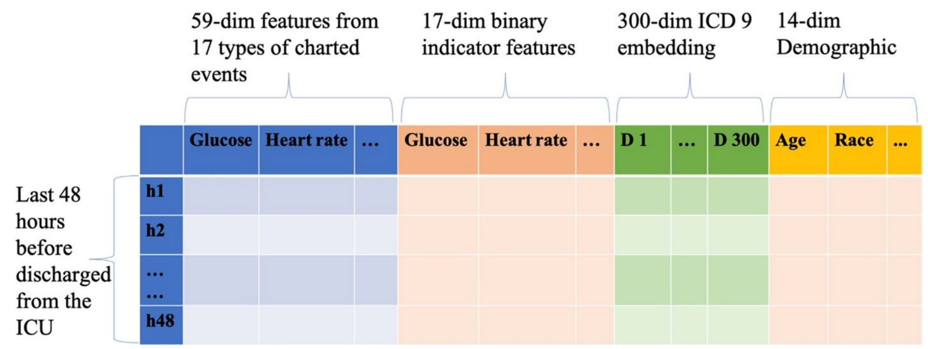
59-dim features from 17 types of charted events
17-dim binary indicator features
300-dim ICD 9 embedding
14-dim Demographic
Glucose
Heart rate
...
Glucose
Heart rate
...
D 1
...
D 300
Age
Race
...
Last 48 hours before discharged from the ICU
h1
h2
...
...
h48

Fig. 4 Image copied from [51], structure of time series dataset shows mix of quantitative and qualitative data, including embedding of ICD-9 codes

Convolutional neural network for time series data

"Analysis and prediction of unplanned intensive care unit readmission using recurrent neural networks with long short-term memory" [51], by Lin et al. is another work that involves using a convolutional neural network that uses qualitative data for input. The central theme of the work is using electronic health records (EHR's) to predict when a patient will be readmitted to a hospital's intensive care unit (ICU). One may readily leverage this work because the authors provide the source code [52], and the data [84] for their experiments. We include this work in the automatic family of techniques for using categorical data in neural networks because of the manner in which Lin et al. prepare their data for input into a neural network. However, we view [51] as a borderline case because the authors report their best metrics with a model that uses both transfer learning and One-hot encoding for its qualitative inputs. It is borderline cases, such as [51] that inspires the term spectrum when we discuss techniques for using categorical data in neural networks.

Figure 4 shows the reader Lin et al.'s approach to using qualitative data for input to neural networks. It also indicates how the authors can use a block of 48 h of EHR data for one patient as one input to a convolutional neural network for experiments where they use a convolutional neural network. For the purpose of this work we are interested in how Lin et al. deal with the categorical values they use as input to their neural networks. We see from Fig. 4 that the authors convert ICD-9 codes to embedded values. Lin et al. write that they use Choi, Chill, and Sontag's technique from "Learning low-dimensional representations of medical concepts" [53] to embed the ICD-9 codes. Lin, Chill, and Sontag's technique is based on Mikolov et al.'s Word2vec algorithm. This is why we chose to place this work in the category of automatic techniques. The reader may be interested to know that the source code for Lin, Chill, and Sontag's embedding technique is available in [54]. For categorical features in the charted events and demographics areas indicated in Fig. 4 Lin et al. use One-hot encoding. This use of One-hot encoding immediately raises a prospect for future research. We see their Glasgow coma scale categorical value has 13 dimensions. In [3], Guo and Berkhahn embed categorical variables with a similar number of One-hot encoding dimensions, and they achieve a lower mean approximate percentage error when they use embedded categorical variables.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Lin et al. evaluate the performance of Convolutional Neural Networks (CNN's), Long Short-term Memory (LSTM), and combinations of LSTM and CNN's that use the input data pictured in Fig. 4. In this evaluation they also include a comparison to Logistic Regression, Naive Bayes, Random Forest, and Support Vector Machine classifiers, but they must treat their input data differently for use as input to these classifiers. Out of all the models they include in their experiments, Lin et al. find that the best performing configuration is an LSTM that takes the data illustrated in Fig. 4, and the output of the LSTM goes to a CNN that produces the final prediction of whether the patient is likely to return to the ICU.

The model Lin et al. present is a complex example that reflects the authors' expertise in the domain of the input data they use. When such expertise is employed to construct a dataset with categorical data, we encounter a blending of techniques. Still, one can analyze the techniques the authors use and identify determined and automatic techniques. This work is an example of how to employ multiple techniques to encode categorical data for use in a complex neural network.

Summary of automatic techniques

Researchers have achieved state-of-the-art results using data encoded with automatic techniques. See [32, 43] for reports of such superior results. However, the reader should bear in mind that when starting from scratch, automatic techniques are resource intensive. Especially when working with big data, a new automatic technique could require a lot of space and time to compute an embedding of qualitative values. For example, in [43] Mikolov et al. report a training times on the order of one day on datasets of billions of words.

Fortunately many automatic techniques lend themselves to transfer learning, so often it is possible to reuse parts of models previously trained to avoid duplicating the effort. On one hand, we notice there is a common theme of current best-in-class algorithms employing automatic techniques. On the other we recommend trying to use transfer learning first to avoid the resource intensive process of employing an automatic technique for working with qualitative data in neural networks.

Automatic techniques are compelling to study because they are reusable across a wide variety of domains. Furthermore, the symmetry between operations on vectors learned by Word2vec, and semantic relationships between the encoded words is exciting. It is exciting because this pattern emerges from an automated technique that does not encode prior knowledge of its input domain. In this section, we cover distributed representations, entity embeddings, autoencoders, and how transfer learning plays a role in sophisticated models. We also find that while an automatic technique may be the most important aspect of the way in which researchers deal with qualitative data in neural networks, in practice one must employ a composition of techniques.

Conclusion

In this survey, we provide an overview of techniques for using categorical data as inputs to deep learning algorithms. To the best of our knowledge, this is the first work to present a unified description of techniques for encoding categorical values for use in neural

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

networks. We present a new perspective on encoding techniques. The new perspective is a view of encoding techniques in three categories: determined, algorithmic, and automatic. In practice, one must use a blend of techniques for working with categorical data in neural networks, unless one is satisfied with using only determined techniques. Automatic and algorithmic techniques rely on some determined technique as a first step. Determined and automatic techniques are most suitable for big data projects. We find algorithmic techniques to be less attractive because they have lower potential for re-use, and when working with big data, may have running times that are prohibitive. We see an opportunity for future research in a systematic study to compare combinations of embedding techniques on various machine learning tasks. We provide the reader with references to the source code of multiple implementations of models that use categorical encoding techniques to facilitate opportunities for future research. For the long term, it appears that automatic techniques for using categorical data in neural networks may be the most important. We assume industry will be more interested in funding development of techniques that are easy to employ, general-purpose, and reusable. Algorithmic techniques fit this description. However, for the short term, determined techniques are more suitable for encoding quantitative data in large datasets because they are less computationally expensive than automatic or algorithmic techniques. Despite longstanding research on embedding techniques for neural networks, opportunities for further research still exist. For specific tasks, it could be the case that more sophisticated techniques, such as EDLT [76] and GEL [19], are better. However, to the best of our knowledge we do not have a study that compares GEL or EDLT to automatic techniques. This is an opportunity for future research. In this study, we have found a way to organize techniques for working with categorical data in neural networks. Researchers should be aware of the spectrum of techniques available, as well as the methods of employing them.

Abbreviations

AUC: Area under the curve; BERT: Bidirectional Encoder Representations from Transformers; BiGRU: Bi-directional Gated Recurrent Unit; BiLSTM: Bi-directional Long Short-term Memory; CBOW: Continuous bag of words; CNN: Convolutional neural network; CPT: Current Procedural Terminology; DCNN: Deep Convolutional Neural Network; DNA: Deoxyribonucleic acid; DNN: Deep neural network; EDLT: Enabling Deep Learning for Generic Classification Tasks; EE: Entity embedding; EHR: Electronic health records; ELM: Extreme learning machine; FP: False positive; GEL: Generalized feature embedding learning; RR: Random reordering; SGD: Stochastic gradient descent; SVM: Support Vector Machine; GloVe: Global Vectors for Word Representation; ICD: International Statistical Classification of Diseases; ICD-10: International Statistical Classification of Diseases Tenth Revision; ICD-9: International Statistical Classification of Diseases Ninth Revision; ICU: Intensive care unit; KNN: K-Nearest Neighbors; LDA: Latent Dirichlet Analysis; LSTM: Long Short-term Memory; MAPE: Mean absolute percentage error; NMF: Non-Negative Matrix Factorization; NPOD: Neural probabilistic outlier detection; PCA: Principal component analysis; PIP: Pairwise Inner Product; RBF: Radial Basis Function; RBF SVM: Radial Basis Function Support Vector Machine; TP: True positive; URL: Uniform resource locator.

Acknowledgements

The authors would like to thank the anonymous reviewers for their constructive evaluation of this paper, and the various members of the Data Mining and Machine Learning Laboratory, Florida Atlantic University, for assistance with the reviews.

Authors' contributions

JTH performed the literature review and drafted the manuscript. TMK worked with JTH to develop the article's framework and focus. TMK introduced this topic to JTH. Both authors read and approved the final manuscript.

Funding

Not applicable.

Availability of data and materials

Not applicable.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 31 January 2020 Accepted: 30 March 2020
Published online: 10 April 2020

References

1. Goodfellow I, Bengio Y, Courville A. Deep learning. Cambridge: MIT Press; 2016.
2. Blei DM, Ng AY, Jordan MI. Latent dirichlet allocation. J Mach Learn Res. 2003;3(Jan):993–1022.
3. Cheng G, Berkhahn F. Entity embeddings of categorical variables. CoRR. 2016. arXiv:1604.06737.
4. Lacey M. Categorical data. 2019. http://www.stat.yale.edu/Courses/1997-98/101/catdat.htm. Accessed 23 Sept 2019.
5. Lane DM. Online statistics education: an interactive multimedia course of study. 2019. http://onlinestatbook.com/2/index.html. Accessed 15 Dec 2019.
6. Johnson JM, Khoshgoftaar TM. Medicare fraud detection using neural networks. J Big Data. 2019;6:1–35.
7. Hinton GE, et al. Learning distributed representations of concepts. In: Proceedings of the eighth annual conference of the cognitive science society, vol. 1. Amherst, MA. 1986. p. 12.
8. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ, editors. Advances in neural information processing systems 25. Red Hook: Curran Associates, Inc.; 2012. pp. 1097–105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. Accessed 12 Sept 2019.
9. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J. Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. 2013. pp. 3111–9.
10. Google.com: Google Scholar. scholar.google.com.
11. University FA. OneSearch. 2019. https://library.fau.edu/. Accessed 15 Sept 2019.
12. Potdar K, Pardawala TS, Pai CD. A comparative study of categorical variable encoding techniques for neural network classifiers. Int J Comput Appl. 2017;175(4):7–9.
13. Dua D, Graff C. UCI machine learning repository. 2017. http://archive.ics.uci.edu/ml. Accessed 24 Aug 2019.
14. Zhong G, Wang L-N, Ling X, Dong J. An overview on data representation learning: From traditional feature learning to recent deep learning. J Finance Data Sci. 2016;2(4):265–78. https://doi.org/10.1016/j.jfds.2017.05.001.
15. Altınel B, Ganiz MC. Semantic text classification: a survey of past and recent advances. Inf Process Manage. 2018;54(6):1129–53.
16. Goyal P, Ferrara E. Graph embedding techniques, applications, and performance: a survey. Knowl Based Syst. 2018;151:78–94.
17. Shickel B, Tighe PJ, Bihorac A, Rashidi P. Deep EHR: a survey of recent advances in deep learning techniques for electronic health record (EHR) analysis. IEEE J Biomed Health Inform. 2018;22(5):1589–604. https://doi.org/10.1109/JBHI.2017.2767063.
18. Wang Y, Xu W. Leveraging deep learning with lda-based text analytics to detect automobile insurance fraud. Decis Support Syst. 2018;105:87–95.
19. Golinko E, Zhu X. Generalized feature embedding for supervised, unsupervised, and online learning tasks. Inf Syst Front. 2018;21:125–42.
20. Chollet F, et al. Embedding. 2019. https://keras.io/layers/embeddings/. Accessed 29 Nov 2019.
21. Pennington J, Socher R, Manning C. Glove: global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014. pp. 1532–43.
22. Chollet F. Deep learning with python. Shelter Island: Manning Publications Company; 2017.
23. Linguistic Data Consortium TTotUoP. English Gigaword 5th edition–Linguistic Data Consortium. 2019. https://catalog.ldc.upenn.edu/LDC2011T07. Accessed 11 Jul 2019.
24. Pennington J, Socher R, Manning C. GloVe: global vectors for word representation. 2015. https://nlp.stanford.edu/projects/glove/. Accessed 25 Nov 2019.
25. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, et al. Tensorflow: a system for large-scale machine learning. In: 12th USENIX symposium on operating systems design and implementation (OSDI 16). 2016. pp. 265–83 . https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf. Accessed 12 Sept 2019.
26. Contributors T. tensorflow/tensorflow: an open source machine learning framework for everyone. 2019. https://www.kaggle.com/c/gendered-pronoun-resolution/overview/evaluation. Accessed 14 Dec 2019.
27. Bengio Y, Ducharme R, Vincent P, Jauvin C. A neural probabilistic language model. J Mach Learn Res. 2003;3(6):1137–55.
28. De Brébisson A, Simon É, Auvolat A, Vincent P, Bengio Y. Artificial neural networks applied to taxi destination prediction. 2015. arXiv:1508.00021.
29. De Brébisson A, Simon É, Auvolat A, Vincent P, Bengio Y. Adbrebs/taxi: winning entry to the kaggle taxi competition. 2015. https://github.com/adbrebs/taxi. Accessed 7 Dec 2019.
30. Chen T, Tang L-A, Sun Y, Chen Z, Zhang K. Entity embedding-based anomaly detection for heterogeneous categorical events. 2016. arXiv:1608.07502.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

31. Chollet F et al. Keras/embeddings.py at master—keras-team/keras. 2019. https://github.com/keras-team/keras/blob/master/keras/layers/embeddings.py. Accessed 26 Dec 2019.
32. Devlin J, Chang M-W, Lee K, Toutanova K. BERT: pre-training of deep bidirectional transformers for language understanding 2018. arXiv:1810.04805.
33. Devlin J, Chang M-W, Lee K, Toutanova K et al. google-research/bert: TensorFlow code and pre-trained models for BERT. 2019. https://github.com/google-research/bert. Accessed 17 Nov 2019.
34. Goyal P, Ferrara E. Gem: a python package for graph embedding methods. J Open Sour Softw. 2018;3(29):876.
35. Guo C. Entity-embedding-rossmann. 2015. https://github.com/entron/entity-embedding-rossmann/blob/kaggle/models.py. Accessed 19 Dec 2019.
36. Howard J et al. layers.fast.ai 2019.
37. Howard J, et al. fastai. GitHub 2019.
38. Kieu T, Yang B, Jensen CS. Outlier detection for multidimensional time series using deep neural networks. In: 2018 19th IEEE international conference on mobile data management (MDM), IEEE. 2018. pp. 125–34.
39. Kocmi T, Bojar O. An exploration of word embedding initialization in deep-learning tasks. 2017. arXiv:1711.09160.
40. Lei L, Zhou Y, Zhai J, Zhang L, Fang Z, He P, Gao J. An effective patient representation learning for time-series prediction tasks based on EHRs. In: 2018 IEEE international conference on bioinformatics and biomedicine (BIBM), IEEE. 2018. pp. 885–92.
41. Li Y, Huang C, Ding L, Li Z, Pan Y, Gao X. Deep learning in bioinformatics: introduction, application, and perspective in the big data era. Methods. 2019;. https://doi.org/10.1016/j.ymeth.2019.04.008.
42. Li Y, Huang C, Ding L, Li Z, Pan Y, Gao X. Lykaust15/deep_learning_examples: examples of using deep learning in bioinformatics. 2019. https://github.com/lykaust15/Deep_learning_examples. Accessed 22 Aug 2019.
43. Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. 2013. arXiv preprint arXiv:1301.3781.
44. Mikolov T, Sutskever I. word2vec. 2015. https://code.google.com/archive/p/word2vec/. Accessed 29 Dec 2019.
45. Ho Q-T, Phan D-V, Ou Y-Y, et al. Using word embedding technique to efficiently represent protein sequences for identifying substrate specificities of transporters. Anal Biochem. 2019;577:73–81. https://doi.org/10.1016/j.ab.2019.04.011.
46. Pennington J, Socher R, Manning C et al. stanfordnlp/GloVe: GloVe model for distributed word representation. 2018. https://github.com/stanfordnlp/GloVe/. Accessed 23 Aug 2019.
47. Kratzwald B, Ilić S, Kraus M, Feuerriegel S, Prendinger H. Deep learning for affective computing: text-based emotion recognition in decision support. Decis Support Syst. 2018;115:24–35.
48. Duarte F, Martins B, Pinto CS, Silva MJ. Deep neural models for ICD-10 coding of death certificates and autopsy reports in free-text. J Biomed Inform. 2018;80:64–77.
49. Contributors T. ciscorduarte/mortality_coding_dnn. 2018. https://github.com/ciscorduarte/mortality_coding_dnn. Accessed 4 Dec 2019.
50. Cheng L, Wang Y, Ma X. A neural probabilistic outlier detection method for categorical data. Neurocomputing. 2019;365:325–35.
51. Lin Y-W, Zhou Y, Faghri F, Shaw MJ, Campbell RH. Analysis and prediction of unplanned intensive care unit readmission using recurrent neural networks with long short-term memory. PLoS ONE. 2019;. https://doi.org/10.1371/journal.pone.0218942.
52. Lin Y-W, Zhou Y, Faghri F, Shaw MJ, Campbell RH. Jeffreylin0925/MIMIC-III_ICU_Readmission_Analysis: This is the source code for the paper 'Analysis and Prediction of Unplanned Intensive Care Unit Readmission'. 2018. https://github.com/Jeffreylin0925/MIMIC-III_ICU_Readmission_Analysis.
53. Choi Y, Chiu CY-I, Sontag D. Learning low-dimensional representations of medical concepts. AMIA Summits Transl Sci Proc. 2016;16:41.
54. Choi Y, Chiu CY-I, Sontag D. clinicalml/embeddings: Code for AMIA CRI 2016 paper "Learning low-dimensional representations of medical concepts". 2016. https://github.com/clinicalml/embeddings. Accessed 23 Dec 2019.
55. Avati A, Jung K, Harman S, Downing L, Ng A, Shah NH. Improving palliative care with deep learning. In: 2017 IEEE international conference on bioinformatics and biomedicine (BIBM). 2017.
56. Golinko E. egolinko/GEL. 2019. https://github.com/egolinko/GEL. Accessed 13 Oct 2019.
57. Han H, Li Y, Zhu X. Convolutional neural network learning for generic data classification. Inf Sci. 2019;477:448–65. https://doi.org/10.1016/j.ins.2018.10.053.
58. Han H, Li Y, Zhu X. ELDT. 2019. https://github.com/hhmzwc/EDLT. Accessed 12 Jul 2019.
59. Svenstrup DT, Hansen J, Winther O. Hash embeddings for efficient word representations. In: Advances in neural information processing systems. 2017. pp. 4928–36.
60. Svenstrup DT, Hansen J, Winther O. Hashembedding. 2017. https://github.com/dsv77/hashembedding. Accessed 8 Aug 2019.
61. Duan J. Financial system modeling using deep neural networks (DNNs) for effective risk assessment and prediction. J Frankl Inst. 2019;356(8):4716–31.
62. Naseer S, Saleem Y. Enhanced network intrusion detection using deep convolutional neural networks. KSII Trans Internet Inf Syst. 2018;12(10):5159.
63. Buitinck L, Louppe G, Blondel M, Pedregosa F, Mueller A, Grisel O, Niculae V, Prettenhofer P, Gramfort A, Grobler J, et al. API design for machine learning software: experiences from the scikit-learn project. In: ECML PKDD workshop: languages for data mining and machine learning. 2013. pp. 108–22.
64. Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. Cambridge: MIT Press; 2009.
65. Hackeling G. Mastering machine learning with Scikit-learn. Birmingham: Packt Publishing Ltd; 2017.
66. Cui L, Xie X, Shen Z. Prediction task guided representation learning of medical codes in EHR. J Biomed Inform. 2018;84:1–10.
67. Developers S-l. sklearn.preprocessing.OneHotEncoder –scikit-learn 0.21.3 documentation. 2019. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html. Accessed 29 Jul 2019.

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

68. Golinko E, Sonderman T, Zhu X. Learning convolutional neural networks from ordered features of generic data. In: 2018 17th IEEE international conference on machine learning and applications (ICMLA), IEEE. 2018. pp. 897–900.
69. McGinnis W. Leave one out—category encoders latest documentation. 2016. https://contrib.scikit-learn.org/categorical-encoding/leaveoneout.html. Accessed 16 Aug 2019.
70. Zhang O. Tips for data science competitions. 2016. https://www.slideshare.net/OwenZhang2/tips-for-data-science-competitions. Accessed 5 Aug 2019.
71. Tavallaee M, Bagheri E, Lu W, Ghorbani AA. A detailed analysis of the kdd cup 99 data set. In: 2009 IEEE symposium on computational intelligence for security and defense applications. 2009. pp. 1–6. https://doi.org/10.1109/CISDA.2009.5356528.
72. MacKay DJ. Information theory, inference and learning algorithms. Cambridge: Cambridge University Press; 2003.
73. Foundation PS. Built-in functions. 2019. https://docs.python.org/3/library/functions.html. Accessed 29 Oct 2019.
74. Why haven't any SHA-256 collisions been found yet? 2017. https://crypto.stackexchange.com/a/47810. Accessed 17 Nov 2019.
75. Kivinen J, Smola AJ, Williamson RC. Online learning with kernels. IEEE Trans Signal Process. 2004;52(8):2165–76.
76. Han H, Zhu X, Li Y. EDLT: enabling deep learning for generic data classification. In: 2018 IEEE international conference on data mining (ICDM). Washington, DC: IEEE; 2018. https://doi.org/10.1109/icdm.2018.00030.
77. Yin Z, Shen Y. On the dimensionality of word embedding. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, editors. Advances in neural information processing systems 31. Red Hook: Curran Associates, Inc.; 2018. pp. 887–98. http://papers.nips.cc/paper/7368-on-the-dimensionality-of-word-embedding.pdf. Accessed 16 Sept 2019.
78. Yin Z. Understand functionality and dimensionality of vector embeddings: the distributional hypothesis, the pairwise inner product loss and its bias-variance trade-off. 2018. arXiv preprint arXiv:1803.00502.
79. Yin Z. Word embedding dimensionality selection. 2019. https://github.com/ziyin-dl/word-embedding-dimensionality-selection. Accessed 10 Dec 2019.
80. Inc K. Rossmann store sales. 2015. https://www.kaggle.com/c/rossmann-store-sales. Accessed 11 Dec 2019.
81. Bengio Y, Courville A, Vincent P. Representation learning: a review and new perspectives. IEEE Trans Pattern Anal Mach Intell. 2013;35(8):1798.
82. Day O, Khoshgoftaar TM. A survey on heterogeneous transfer learning. J Big Data. 2017;4(1):1–42.
83. Kaggle I. Twitter sentiment analysis. 2017. https://www.kaggle.com/c/twitter-sentiment-analysis2. Accessed 10 Dec 2019.
84. Johnson AE, Pollard TJ, Shen L, Li-wei HL, Feng M, Ghassemi M, Moody B, Szolovits P, Celi LA, Mark RG. Mimic-iii, a freely accessible critical care database. Sci Data. 2016;3:160035.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

- ▶ Convenient online submission
- ▶ Rigorous peer review
- ▶ Open access: articles freely available online
- ▶ High visibility within the field
- ▶ Retaining the copyright to your article

Submit your next manuscript at ▶ springeropen.com

Content courtesy of Springer Nature, terms of use apply. Rights reserved.

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH ("Springer Nature").
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users ("Users"), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use ("Terms"). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com