KU LEUVEN

Semi-supervised autoencoders for long text classification

Quentin Meeus

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Engineering and Computer Science

Thesis supervisor:
Prof. Marie-Francine Moens

Assessor:
Prof. Jesse Davis, Elias Moons

Mentor:
Sumam Francis

Academic year 2018 – 2019

Semi-supervised autoencoders for long text classification

Quentin Meeus

August 14, 2019
Version: Final Version

Katholiek Universiteit Leuven

Departement of Computer Science
Master of Artificial Intelligence

Master Thesis

Semi-supervised autoencoders for long text classification

Quentin Meeus

Promotor
Prof. Marie-Francine Moens

Supervisor
Sumam Francis

Reviewer
Prof. Jesse Davis

Reviewer
Elias Moons

August 14, 2019

Quentin Meeus
Semi-supervised autoencoders for long text classification
Master Thesis, August 14, 2019
Promotor: Prof. Marie-Francine Moens
Supervisor: Sumam Francis
Reviewers: Prof. Jesse Davis and Elias Moons

Katholiek Universiteit Leuven
Master of Artificial Intelligence
Departement of Computer Science
Celestijnenlaan 200 A
B-3001 and Leuven

Abstract

One major difficulty of text classification learning algorithms is the need of large labelled datasets to be able to reach good performances. In this work, we explore how supervised classifiers trained in conjunction with unsupervised autoencoders can create robust models that do not require much labelled examples to perform well. In particular, we focus on the processing of large text documents, on which traditional models such as LSTMs perform poorly. We propose four semi-supervised autoencoders that impose different constraints on the encodings and compare their performance on the 20Newsgroups dataset. We evaluate the predictions of the classifier as well as the quality of the representations learned by the autoencoder. We show that the joint optimisation of a supervised and unsupervised objective improves both the classification performances and the quality of the representations. Indeed, the semi-supervised autoencoders perform better than a supervised classifier with a matching architecture trained with the same amount of labelled data. Although the advantages provided by the unsupervised objective fade away as more labelled examples are available, the results of our models compare to state-of-the-art classifiers performing the same task. Further, we show that the representations produced by the autoencoders can be used for classification by a Linear SVM or for other related learning tasks such as clustering. Finally, we demonstrate that even when trained on only one label per class, the autoencoder manages to identify words that belong to the same topic, although this finding must be put into perspective, as some topics were ill-defined and some words appeared in multiple unrelated topics.

Acknowledgement

Many people have supported me during the period that lead to this thesis. First and foremost, if anyone is looking for a good advisor, let me recommend my promotor, Sien Moens. Both as a professor and as a supervisor, she has always been there to ask the right questions when help was needed. She is passionate about her work and succeed in instilling this passion in others.

I give my warmest thanks to Sumam Francis for her encouragements and for the helpful discussions that we shared. I owe her numerous insights and advices and this thesis would not have been possible without her. I wish her and her soon-to-be-born baby a world of joy and happiness.

Finally, I express my gratitude to all the people, friends and family who have helped me during this journey either by encouraging me or by giving me advices and suggestions.

Contents

Introduction 1

1 Background 5
- 1.1 Machine Learning . . . 5
  - 1.1.1 Machine Learning Paradigms . . . 5
  - 1.1.2 Generalisation and Overfitting . . . 6
  - 1.1.3 Model Selection and Cross Validation . . . 8
  - 1.1.4 Machine Learning Models . . . 9
  - 1.1.5 Gradient-Based Optimisation . . . 13
- 1.2 Artificial Neural Networks and Deep Learning . . . 15
  - 1.2.1 Multilayer Perceptron . . . 16
  - 1.2.2 Regularisation for Deep Learning . . . 19
  - 1.2.3 Convolutional Neural Networks . . . 20
  - 1.2.4 Recurrent Neural Networks . . . 21
  - 1.2.5 Attention Mechanisms . . . 22
  - 1.2.6 Autoencoders . . . 23
- 1.3 Natural Language Processing . . . 28
  - 1.3.1 A Brief History of NLP Approaches . . . 29
  - 1.3.2 Representation of Text Data . . . 30
  - 1.3.3 Representation Learning . . . 34
  - 1.3.4 Document Classification . . . 36

2 Approach 39
- 2.1 Proposed Models . . . 39
  - 2.1.1 Semi-supervised Autoencoders . . . 40
  - 2.1.2 Loss Functions . . . 42
  - 2.1.3 Baseline Models . . . 43
- 2.2 Methodology and Implementation Details . . . 43
  - 2.2.1 Dataset . . . 43
  - 2.2.2 Training Details and Hyperparameters Tuning . . . 45
- 2.3 Performance Evaluation . . . 46
  - 2.3.1 Effect of unlabelled data on classification . . . 47
  - 2.3.2 Effect of labelled data on representations . . . 47
- 2.4 Exploration of the Parameters . . . 48

3 Results and Discussion **51**
3.1 Semi-supervised Classification Scores 51
3.1.1 Comparison with supervised models 51
3.1.2 Analysis of predictions 52
3.2 Semi-supervised Representation Learning 54
3.2.1 Support Vector Classifier 54
3.2.2 k-means 55
3.3 Exploration of the Network Parameters 55
3.3.1 Topic compositions 56
3.3.2 Representation similarities 56

Conclusion **59**

Bibliography **61**

Introduction

The field of deep learning has seen tremendous growth in the past decade both in terms of commercial application and academic researches. Many tasks that were always considered difficult, even impossible, for computer systems 60 years ago are now achieved successfully with artificial neural networks. The recent progress with self-driving cars, gaming, speech recognition, text generation, question answering and resources planning have shown that artificial intelligence now competes with and often overtakes human abilities. Although, deep learning systems are still unable to perform some tasks that a child masters in no time. In his critical appraisal of deep learning, Gary Marcus predicts that unsupervised learning will take off in the next few years [47]. Other renown scientists in the field share this vision. Marcus arguments that unsupervised learning is by nature less restricted than its supervised counterpart in many ways and provides a stepping stone to building intelligent systems able to do reasoning and problem-solving at a more abstract level. The present work explores how unsupervised data can improve traditional supervised methods with a framework that benefits from the advantages of both sides. In particular, we focus on the possibility to use semi-supervised learning to learn dense representations of long textual documents and build classifiers with a limited amount of labelled data.

Motivation and Problem Statement

Still nowadays, machine learning strongly relies on large amounts of labelled data to make strong models, able to generalise to unseen data. However, if modern datasets are typically composed of millions of examples, labels are often missing or scarce. Automatic label collection sometimes offer a solution for large companies such as Google and Facebook, which can harvest their large user base to perform crowd labelling. Smaller businesses, however, do not benefit from the same resources and must rely on other techniques which often involve to pay some other company or hire human operators to label their datasets manually. A second problem arises from the intrinsic subjectivity of some kinds of labels which can lead to biased datasets. One

common practice to solve this is to use a voting process where multiple operators assign labels to an example and the most assigned label wins. This increases ever further overhead costs and the time needed to build a decent dataset. These issues are even more relevant in text mining where tremendous amounts of unstructured data exist but labels need to be produced manually. In this work, we explore how we can use unlabelled data to create performing machine learning models. Using unlabelled data in addition to labelled examples have sometimes allowed to improve the performance over fully-supervised models. Following this idea, we make the following hypotheses: (1) unlabelled data can be used to improve the performance on supervised tasks by providing additional information on the underlying data distribution; and (2) introducing supervision in an unsupervised task like abstract feature learning helps in creating more meaningful representations that can be used to perform other tasks. In particular, we explore semi-supervised autoencoders trained on long textual documents and compare their performance with unsupervised autoencoders and fully-supervised text classifiers.

Results

We propose four model architectures that impose different constraints on the representations. We use a benchmark dataset, the 20Newsgroups email collection [50], to compare their ability to generate dense representations and predict categories after being trained on a semi-supervised training set with different threshold of labelled data. Our results show that semi-supervised autoencoders make strong models with classification scores that compare to the state-of-the art models. Additionally, we show that the unsupervised objective helps the classification by identifying patterns in the unlabelled dataset that are not captured by a fully supervised model, thus creating models that generalise better to unseen data. However, this advantage decrease as the propotion of labelled data in the dataset approaches 100%. The use of a supervised objective in conjunction with an unsupervised one also produces better representations of the input documents that can be used to solve related but different tasks such as clustering or classification, with better performances than when the models are entirely unsupervised. We find that the generated representations show a stronger similarity when two documents belong to the same category and inversely when the documents belong to different categories. Finally, the models manage to identify the words that belong to the same semantic family and map them to the same topics even when very few labels are used during training.

Thesis Structure

Chapter 1: Background

The first chapter gives an broad overview of the current state of research in the fields of machine learning, deep learning and natural language processing. We give the necessary information to fully understand the methods proposed in this work. The chapter starts with an introduction to machine learning, including the main paradigms of learning, the challenges of generalisation and the practical methodology framework to train machine learning algorithms. Then we present a selection of traditional machine learning algorithms that are mentioned in the thesis. Finally, we quickly go over the principles of gradient-based optimisation and present the gradient descent and backpropagation algorithms. The second section reviews deep learning theory and is widely inspired by the book of Goodfellow et al. [25]. We cover the theoretical basis behind neural networks with the multilayer perceptron and discuss some considerations and techniques for training deep learning models, finishing with a presentation of popular models and architectures. We conclude this chapter with a presentation of the natural language processing and text mining. After a short review of the history of the field and a discussion about the techniques to handle textual data, we discuss the modern methods and state-of-the-art models specific to natural language processing.

Chapter 2: Approach

The second chapter discusses the details of the approach driving this work. We start by presenting the proposed model architectures and the related optimisation techniques and objective functions. Next we detail the steps necessary to reproduce our results starting with a description of the selected dataset and the methodology followed for the data preprocessing, selection of hyperparameters and the training of the models. We list all the hyperparameters used for the selection and the ones that were selected for the final models. Finally, we propose various qualitative and quantitative experiments to evaluate the performance of the proposed algorithms and validate our hypotheses.

Chapter 3: Results and Discussion

The final chapter presents our results in depth. We relate to the experiments detailed in Chapter 2 to measure and compare the performances of our models to each other and to those reported in published works. Finally, we discuss the limitations of this work and the potential improvements, as well as some interesting directions for future researches.

1 Background

1.1 Machine Learning

In very simple words, a machine learning algorithm is "*an algorithm that is able to learn from data*" [25]. Learning in this context is defined as follows: "*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E*" [49]. A machine learning approach typically uses a **training set** to tune the parameters of an adaptive model in the learning phase [5]. The model is trained to perform a specific task, such as approximating the distribution that generated the dataset, and its performance is measured on a **test set**. The test set is a subset of data that was either collected separately or removed from the dataset before the training phase so that we can assess the capacity of the model to generalise to unseen data. Working with both a training and a test set in paramount in machine learning algorithm and in particular in supervised learning, as we will see in the later sections.

1.1.1 Machine Learning Paradigms

Machine learning algorithms can be categorised as supervised or unsupervised depending on the type of dataset on which they are trained [25]. In **supervised learning**, the dataset is composed of a set of features x to which is associated as set of labels or targets, y. The objective of the model is to learn a mapping from an input variable x to a target variable y. The task is called *regression* when y is continuous and *classification* when y can take one or more of K discrete values. For examples, identifying spams from other e-mails is a classification task and predicting the price of a house based on features such as the number of rooms and the square footage is a regression task. Both examples are supervised. Supervised algorithms are divided into two families: Generative models try to learn the joint probability distribution $p(x, y) = p(x|y)p(y)$ and discriminative models estimate the posterior distribution of a label y after observing an example x, $p(y|x)$. Discriminative models are often considered to be more efficient because their goal is more directly aligned with the goal of supervised learning [7]. **Unsupervised models** experience datasets that only contain features and no labels. These algorithms typically try to learn the

entire probability distribution that generated the data, $p(x)$ either explicitly (density estimation) or implicitly (synthesis or denoising) [25]. Clustering, which is the task of dividing a dataset into groups, or clusters of similar observations is another example of unsupervised learning algorithm. Note that there is no clear line between both terms and a supervised task can easily be turned into an unsupervised one and vice versa. For example, as pointed out in Goodfellow et al. [25], we can solve the supervised problem of learning $p(y|x)$ by learning the joint distribution $p(x, y)$ with an unsupervised algorithm and inferring

$$p(y|x) = \frac{p(x, y)}{\sum_i p(x, y_i)}.$$

Compared to supervised learning, where the task is well-defined and a model can be evaluated through its predictive performance on test examples, unsupervised tasks are more delicate because the target distribution $p(x)$ is typically unknown [7]. Finding the right evaluation method is therefore a major issue with unsupervised learning tasks. Finally, somewhere between the supervised and unsupervised learning tasks are the **semi-supervised learning algorithms**, which deal with datasets where some examples have labels and other don't.
In many cases, building supervised datasets is expensive when the target variable y cannot be automatically collected and must be provided by a human. Labelling datasets is a cumbersome task and is not always feasible if the size of the dataset is very large. This is particularly true in natural language processing tasks where a profusion of unstructured data exists but labels often need to be produced manually [75]. In these cases, semi-supervised algorithms may provide the advantage of learning a good approximation of the underlying data distribution from a partially labelled dataset without requiring too much labels, that is, provided of course that the distribution of the unlabelled examples is relevant for the classification problem. This provision is a necessary condition for semi-supervised learning to work [7]. Further, it has been showed that working with semi-supervised data can sometimes improve the model performance on small or noisy datasets [39]. For example, in the case of fraud detection, the goal is to build a model that is able to detect schemes that have never been observed before. A supervised algorithm fails at this task because it relies on past examples to predict future observations whereas a semi-supervised model can be used to efficiently identify patterns that have been categorised as fraud in the past as well as detecting abnormal transactions that can potentially be fraudulent.

1.1.2 Generalisation and Overfitting

The main focus of machine learning is to build an algorithm that performs well on previously unseen inputs. This is called **generalisation**. In the typical machine

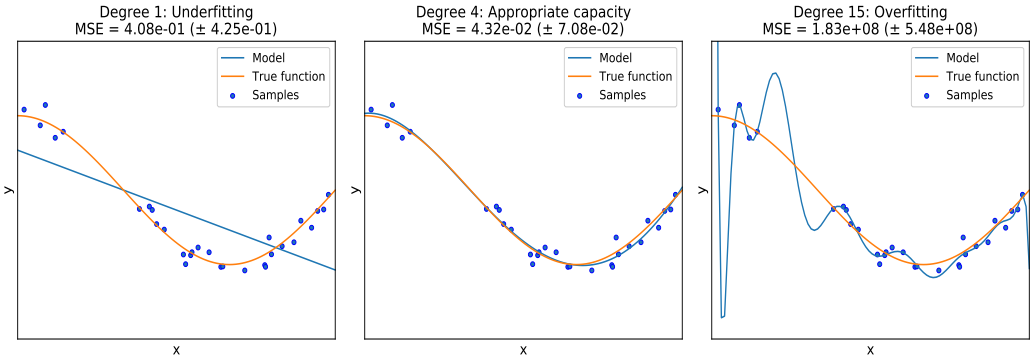


Fig. 1.1: Model capacity and fitness. Source: Pedregosa et al. [54]

learning approach, the developer works with a training set and a test set. The training set is used to estimate the parameters of the model through a process that minimises the **training error** during the learning phase. The capacity of the model to generalise to new data is measured by calculating the **generalisation error** or the test error on the test set. The optimisation process is successful when both the training and the test error are low. When the training error is large, the model is **underfitting**. This means that we were not able to learn a good approximation of the training data. When the training error is low but the generalisation error is large, the model is **overfitting** on the training set and even though the model is able to classify training examples, it fails at predicting new, unobserved points. The capacity of a model is "*its ability to fit a wide variety of functions*"[25]. Underfitting occurs when the model's capacity is too low to fit the training set. Overfitting arises when the model fits perfectly the training set but is unable to generalise to unseen data. In Figure 1.1 we try to approximate a cosine function with polynomial models. The generalisation error is the mean squared error between the predictions $\hat{y}$ and the targets y:

$$MSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

On the left, we try to fit a polynomial function of degree 1 (a line) to the data. The capacity of the model is too low to capture the correct structure of the data and the model is underfitting. The graph on the right shows that a polynomial function of degree 15 is able to fit to the training data but is unable to generalise to unseen data. This is revealed by the large test error. The graph on the centre shows a good fitness to the data and an ability to generalise to test data: in this case, a polynomial model of degree 4 is the right choice for the dataset. This demonstrate a major trade-off in machine learning between bias and variance. The model on the left has low variance and high bias and the model on the right has high variance and low bias.

In the example above, the model was kept simple for the sake of illustration. In practice, overfitting is often a more delicate process and finding the right set of hyperparameters like we did with the degree of the polynomial function is not

always enough. More complex models are not only affected by the number of functions that they can represent but also by the specific identity of those functions [25]. One popular practice to control the capacity of a model is to add a penalty term to the objective function that we are trying to optimise. This is referred to as **regularisation** and will be discussed in details in Section 1.2.2. Another usual method to prevent overfitting is to monitor the generalisation error of the model and stop the training process when we observe the signs of overfitting. This is called **early stopping** and it must be conducted on an independent subset of the training data, called a validation set (see Section 1.1.3), that is neither used for training nor for the final performance evaluation of the model. Typically, one will estimate the generalisation error at each number of iterations of the training algorithm and stop the optimisation when the performance has not improved after a defined number of steps. We control when training should stop through two parameters: the patience, or the number of iterations during which the performance is allowed not to improve, and the minimum change in performance that qualify as an improvement. When the patience is too low, the algorithm might stop before converging and when the patience is too high, chances of overfitting to the training data increase. The minimum performance criterion provides a trade-off between training time and final performance.

1.1.3 Model Selection and Cross Validation

Most machine learning models have hyperparameters that can be adapted to control the algorithm's behaviour. For example, we have already shown in Section 1.1.2 that we can increase or decrease the capacity of a polynomial regression model by changing the degree of the polynomial features. Some families of algorithms such as neural networks have a very large number of hyperparameters and finding the right set of values for a specific dataset is far from easy. If these hyperparameters could theoretically be learned as part of the training process, it should technically be avoided because this can quickly lead to overfitting. Instead, a good practice is to train models using different values for the hyperparameters and compare their performance on a held-out test set. It should be noted that we must not make any decision about the model on the test set that is used to measure its final performance. That is, we do not want to choose a model because it performs well on our test set. Rather, we want to choose a good model, then measure its performance on a test set. For this reason, we use a **validation** or **development set**, *i.e.* a subset of the training set that is held-out and that will be used to make choices about the model by monitoring the **validation error**. Typically, 80% of the training set will be used for training the parameters and 20% will be held out for validation purposes such as hyperparameters selection and early stopping.

Gridsearch and Hyperparameter Selection

When we can limit the number of hyperparameters to reasonable levels, we can perform an exhaustive grid search. This consists in training multiple models with different sets of hyperparameters and comparing their estimated generalisation performance on a development set. When the set of potential hyperparameters is too large, one can perform a randomised grid search where each set has an equal probability of being selected. Finally, advanced search algorithms exist that perform a directed search by investigating further the hyperparameters that show a good results and abandoning the values that consistently lead to bad performance scores. In practice, these methods are very computationally expensive and not always adequate when the training of the models takes a long time.

k-fold Cross Validation

In order to address the bias of using a validation set that is drawn from the training set, one can use a technique called **k-fold cross validation**, where the dataset is randomly divided into k subsets of equal size. One model is trained k times with $k-1$ sets assigned to the training set and the remaining set is used for validation. The scores resulting from each individual training are averaged to get the final cross-validation score. Again, this method is not always feasible because it involves retraining multiple times the same model. However, it has the advantage to provide an average performance of the model that is less biased than when using one unique validation set.

1.1.4 Machine Learning Models

Many machine learning algorithms exist and this is not the scope of this work to review them all. In the present section, we present briefly the models that are relevant to this work and the aspects that are necessary to understand the next chapters. Many books have been written on the topic and are left to the discretion of the reader. For an extensive analysis of the mathematical foundations of machine learning, we direct the reader to Bishop [5]. Barber [4] gives an exhaustive explanation of probabilistic graphical models and Goodfellow et al. [25] is a must-read for every deep learning practitioner.
In the following paragraphs, we give a brief introduction of linear support vector machines, probabilistic graphical models, principal component analysis and the k-means clustering algorithm.

Support Vector Machines

A Support Vector Machine (SVM) is a very powerful machine learning model, capable of performing linear or non-linear classification, regression and even outlier detection [21]. Let us consider a dataset of feature vectors $X = (x^{(1)}, ..., x^{(n)})$ where $x^{(i)} \in \mathbb{R}^m$ is a training example. Let $Y = (y^{(1)}, ..., y^{(n)})$ be a set of binary targets where $y^{(i)} \in \{-1, 1\}$ is the label of the example $x^{(i)}$. A binary SVM will find the hyperplane that separates the data by the largest margin (see Figure 1.2). The equation of the hyperplane is $w^T x + b = 0$ where $w = (w_1, ..., w_n)$ is a vector of coefficients and b is a bias term. In our binary case, an observation will be classified as a positive example when $w^T x + b > 0$ and as a negative example when $w^T x + b < 0$. In other words, the classifier can be written as $\hat{y} = sign(w^T x + b)$.

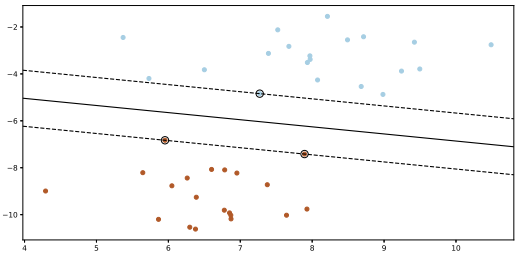


Fig. 1.2: A SVM learns the hyperplane that maximises the margin between two sets of linearly separable examples. Source: Pedregosa et al. [54]

Multiclass classification can be realised through a one-vs-one strategy where we train a separate classifier for each different pair of labels. In total, an ensemble of $\frac{N(N-1)}{2}$ classifiers are trained together and each individual classifier specialises in discriminating between two classes. Predictions are made by presenting the example to each model. The predicted label is the one that get the most votes.

The model described above works for linearly separable data but is easily adapted to non-linear data with the so-called *kernel trick*. Put simply, the original data is projected into another dimensional space by applying a non-linear function to it. Popular kernels include polynomial kernels, radial basis functions and sigmoid kernels. In the present work, we will focus on linear kernels because we are interested in learning representations that are linearly separable.

Probabilistic Models

Probabibilistic Graphical Models (PGMs) use probability distributions to describe complex environments represented as graphical models. The nodes of the graph represent the model variables and the edges correspond to conditional dependencies between the variables [4]. The assumptions on the conditional dependencies allow to realise important savings in terms of computations but it also involves a prior knowledge of the model structure. For example, Naive Bayes (NB) (top-left network in Figure 1.3) assumes that all features are conditionally independent given the

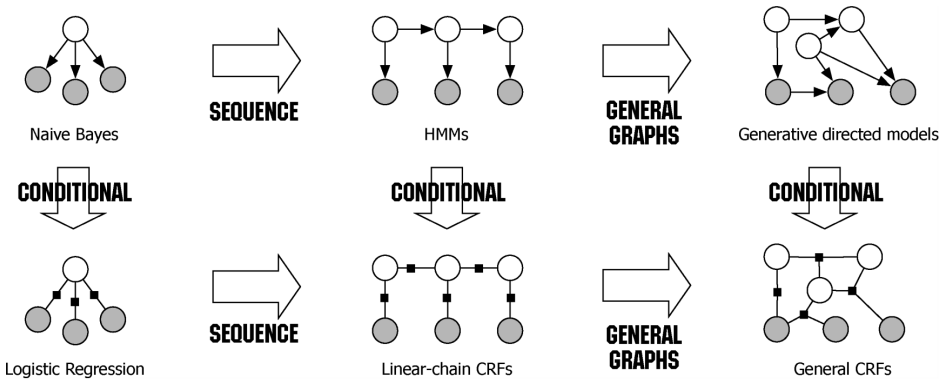


Fig. 1.3: Probabilistic Graphical Models. Source: Sudderth [64]

classes. The parameters of the model consist of the conditional probability distributions of the variables states. Learning is often achieved with an iterative algorithm such as Expectation Maximisation (EM) to find the set of parameters that maximises the likelihood of the data (Maximum Likelihood Estimate), $\theta^* = \max_\theta P(D|\theta)$; or that maximises the posterior given a prior distribution over the parameters (Maximum A Posteriori), $\theta^* = \max_\theta P(\theta|D) = \max_\theta P(D|\theta) \cdot P(\theta)$. Logistic regression, and Conditional Random Fields (CRFs) are examples of discriminative PGMs and Naive Bayes and Hidden Markov Models (HMMs) of generative PGMs (Figure 1.3). Such graphical representation generally induce a strong factorisation that simplifies greatly the otherwise intractable joint distribution. In the example of Naive Bayes, the predicted class of an example $X = (x_1 ... x_n)$ is given by:

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, P(x, y) = \underset{y}{\operatorname{argmax}} \prod_{i=1}^{n} P(x_i|y) \cdot P(y)$$

The number of parameters that fully define such a model grows linearly with the number of features, whereas the full joint probability without the assumptions of independence requires to compute an exponential number of parameters. Of course, such strong assumptions rarely holds in practice and the probabilities predicted by this model are often not reliable. That being said, Naive Bayes has been successfully applied in many real-world situation including spam filtering and document classification.

Another model that has performed well on text classification is the logistic regression algorithm. It performs linear regression with the difference that the prediction is transformed into a probability with the logistic function, $\sigma(x) = \frac{1}{1+e^{-x}}$. This model can be seen as a probabilistic model as well, where the goal is to predict the conditional probability of a target variable y given an observation x: $P(y|x)$. In HMMs, we introduce the notion of time and add hidden variables, h_t, also called latent variables, as opposed to the visible (emitted) variables, v_t. The model is fully defined with the initial probability distributions of the hidden variables, $P(h_0)$ and the emissions and transitions probabilities $P(h_{t+1}|h_t)$ and $P(v_t|h_t)$. The HMM in

Figure 1.3 is of first order, *i.e.* it makes the assumption that the future is only function of the present and not of the past (local Markov property). In other words, $h_{t+1:T}$ and $v_{t:T}$ are conditionally independent of $h_{0:t-1}$ and $v_{0:t-1}$ given h_t. Therefore, the joint probability distribution is given by

$$P(h_{0:T}, v_{0:T}) = P(h_0) \cdot \prod_{t=0}^{T-1} P(h_{t+1}|h_t) \cdot \prod_{t=0}^{T} P(v_t|h_t)$$

CRFs are used to predict structured data such as sequences or lattices by modelling the conditional distribution (rather than the joint distribution) over the variables. The discriminative nature of such models makes weaker assumptions about features independences which in turns simplify inferences. For given positive potential functions $\phi_k(y, x)$, the probability of the latent variables y conditioned by the visible variables x is given by

$$P(y|x) = \frac{1}{Z(x)} \prod_k \phi_k(y, x)$$

where $Z(x)$ is the partition function, which ensures that the probabilities sum to one.

In a nutshell, probabilistic models are extremely useful for many machine learning tasks involving statistical (in)dependence relationships between random variables including natural language processing tasks. They can make very complex probabilities distributions computationally tractable and can be combined with other models such as neural networks, which makes them very popular for a large number of tasks.

Principal Component Analysis

Dimensionality reduction is an unsupervised machine learning task that seeks to decrease the number of features in a dataset while conserving as much relevant information as possible. Principal component analysis (PCA) projects a dataset of n features in a k-dimensional space represented by vectors, called principal components, that are orthogonal to each other. The components are chosen to be the k largest eigenvalues of the covariance matrix of the dataset X. In practice, each new component captures a little more variance in the dataset and it is important to find the right balance between the amount of variance explained by the principal components and the complexity of the dataset. In Figure 1.4, the first component (largest arrow) captures the underlying structure of the data and

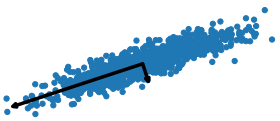


Fig. 1.4: The two principal components of a dataset with two dimensions

the second component captures the remaining noise and is orthogonal to the first component. Finally, we can apply the same kernel trick as for SVMs to find non-linear representations of the dataset.

k-means Clustering

The k-means clustering algorithm [19, 44] is an unsupervised learning technique that divides the training set into k different clusters of similar examples. It works as follows:

1. Initialise the cluster centroids $\{\mu^{(1)}, ..., \mu^{(k)}\}$ randomly
2. Until convergence:
   a) Assign each training example to the cluster with the nearest centroid
   b) Update each centroid $\mu^{(i)}$ to the mean of all training examples assigned to cluster i

Convergence depends on a quality function Q, generally chosen as the sum of squared distances of the samples to the closest centroid and the distance function can be any metric that respects the triangle inequality (often the Euclidean distance):

$$Q(C) = \sum_{C_i \in C} \sum_{x_j \in C_i} d(x_j, \mu_i)^2$$

k-means poses a few problems, some of which have already been hinted at when we discussed unsupervised learning. Firstly, no single criterion exists that measures how well a clustering of the data represents reality. This is typical from unsupervised learning algorithms for which there is no known "ground truth". This means that evaluating such model is not as straightforward as it is with classification. Of course, when we perform clustering on a supervised dataset, the labels can be used to measure the performance of the model. Secondly, the final state of the algorithm depends largely on the initialisation of the centroids. Consequently, running the algorithm multiple times with different initialisations is generally a good practice.

1.1.5 Gradient-Based Optimisation

Any machine learning algorithm involves maximising or minimising some objective function or criterion. In this work, whenever we use the term optimisation, we refer to minimising a **cost function** or a **loss function**. In calculus, we use the derivative of the function $f'(x) = \frac{dy}{dx}$, also denoted as the slope of $f(x)$ at the point x. Gradient descent [6] is a technique used to minimise $f(x)$ by moving x in small steps in the

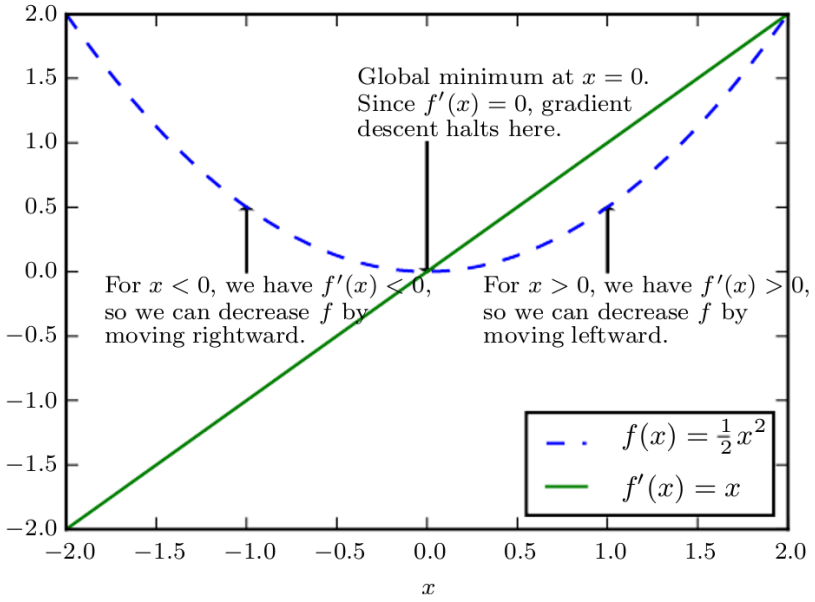


Fig. 1.5: An illustration of how the gradient descent algorithm finds the minimum of a function by taking steps in the direction of the descent.
Source: Goodfellow et al. [25]

direction of the sign of the derivative $f'(x)$ (Figure 1.5). This method suffer from a major drawback because of what is called **local optima**. The ultimate goal of any optimisation technique is to find the point x where the function $f(x)$ is the lowest, called the **global minimum**, where the derivative evaluates to zero. However, not all points where $f'(x) = 0$ is a global minimum. In fact, such critical points can be divided into three categories: local minima, global minima and saddle points. Only the lowest of the local minima is the global minimum. The problem arises from the fact that in a high dimensional space, critical points are very common and there is a large risk that the optimisation algorithm ends up in a local minimum. This is especially the case when many critical points exist surrounded by very flat regions. However, it is sometimes enough to settle for finding a value that is low but not necessary minimal in a formal sense [25].
When the function to optimise takes multiple inputs, we use the gradient, a vector of the partial derivatives with respect to each of the individual variables:

$$\nabla_x f(x_1, ..., x_n) = (\frac{\partial}{\partial x_1} f, ..., \frac{\partial}{\partial x_n} f)$$

We minimise f by updating x by a small step proportional to the learning rate, ϵ, in the direction where the descent is the steepest:

$$x' = x - \epsilon \nabla_x f(x)$$

When both the inputs and the outputs of a function are vectors, like in the case for neural networks, minimising the function is done by finding the set of points for

which the **Jacobian matrix**, the matrix of first partial derivatives, is null. Often, we can improve the optimisation process by considering the second derivatives of the function as well, or the **Hessian matrix**. The second derivative gives an indication of the curvature of the function at a definite point. In other words, it measures how much we can expect the gradient to change if we update x along the negative gradient: if the slope is almost flat, learning will be slow, and we might need to take a larger step. The Hessian matrix is also useful to determine whether a critical point is a local maximum, a local minimum or a saddle point.

Stochastic gradient descent (SGD) extends gradient descent to large datasets, for which calculating the exact gradient is often computationally too expensive to work in practice. SGD approximates the gradient by decomposing the dataset into minibatches and performing updates of the parameters on the estimated gradient calculated over one subset of the dataset [25]. Most optimisation algorithms used to train machine learning and deep learning models are variations of SGD. One example is the Adam algorithm [36], which updates the learning rate at which the parameters are updated based on first and second order momentum estimate of the gradients. This method is generally chosen to train neural networks because it is computationally efficient and works very well in practice.

1.2 Artificial Neural Networks and Deep Learning

Deep learning tries to solve tasks by learning a hierarchy of concepts where more complicated are built out of simpler ones [25]. A deep learning model is a graph with multiple layers stacked on top of each other. The more layers, the deeper the model. Although artificial neural networks were originally inspired by their biological counterpart, the comparison stops there and both their structure and the learning process have evolved to be quite different from what we know of the brain workings. With the increased availability of computational power, this field of study has grown tremendously in the past decade and problems that were previously considered intractable for computer systems are now solved with deep learning. In the field of computer vision, convolutional neural networks show performances that were long though impossible to reach and OpenAI has recently created a language generation model that beats the odds in terms of text quality. Professional gamers got beaten by Google's Deepmind with strategies that were never encountered before. These are only examples from a long list of victories of A.I. and much more is yet to come.

In the next following sections, we will review the most common architectures for building deep learning models. We also discuss some considerations for training such models.

1.2.1 Multilayer Perceptron

The **perceptron** was introduced in 1958 by Frank Rosenblatt [59]. It is the simplest form of artificial neural network. Given a set of inputs $x \in \mathbb{R}^m$, m real-valued weights $(w_1, ..., w_m)$ and a bias term b, it calculates the linear combination $\sum_{i=1}^{m}(w_i x_i) + b$ and applies a non-linear **activation function** $\varphi(.)$ to the result, as shown in Figure 1.6. A **multilayer perceptron (MLP)** is essentially a stacked architecture of layers with multiple units where each unit is an individual perceptron. As each neuron, or unit is connected to all the units of the previous layer and all the units of the next layer, MLP are often called fully-connected networks. The term deep learning comes from the structure of the network where many layers are typically stacked on top of each other. The number of layers in a neural network is the depth of the network and the number of units in each layer is its width.

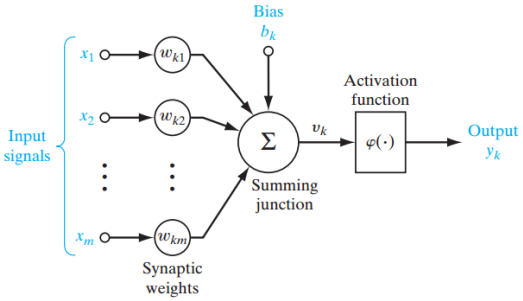


Fig. 1.6: The perceptron
Source: `http://analyticsvidhya.com`

Information Flow and Back-propagation

The objective of a neural network is to learn a mapping $y = f(x)$ between an input x and an output y. Inference, or **forward propagation**, is the process of passing an input signal from layer to layer all the way to the output of the network, $\hat{y}$. In a feedforward network like the MLP, the information flows from the input to the output and there is no feedback connection that feeds the output back to the network. This is not the case for all neural networks, as we will see in Section 1.2.4. Once the output $\hat{y}$ has been calculated, a scalar cost $J(\theta)$ penalises the deviation between the predicted value $\hat{y}$ and the target value y. This is generally denoted as the loss or objective function to be minimised during training. Depending on the task at hand, some functions might be more appropriate than others and choosing the right loss requires careful considerations. Generally, training a model is characterised by a series of iterative steps where the error is calculated for a number of inputs in the forward pass. Then the derivative of the error with respect to the weights is calculated recursively and the parameters are updated to minimise the error, generally with a gradient-based optimisation algorithm (see Section 1.1.5). The method for derivating the error function in neural networks, called **back-propagation**, or simply backprop, was popularised in 1986 by Rumelhart et al. [60]. It applies the chain rule of calculus recursively to compute the derivatives at each layer by multiplying the Jacobian matrix $\frac{\partial y}{\partial x}$ by a gradient $\nabla_y z$. Back-propagating the gradient through many layers

can lead to some complications because we are multiplying small or large values repeatedly together. We will see later in this section how we can address the problem of vanishing gradients by setting the type of hidden units and initialising the weights. Exploding gradients can be avoided by clipping the values of the derivatives to arbitrary values.

Depth and Abstract Representations

Although theoretically the universal approximation theorem [29] states that an MLP with one hidden layer can represent any function given that it has enough units, it has been shown that a greater depth often results in better generalisation performances (see Goodfellow et al. [25] for references and examples). This is because a neural network learns abstract features in its hidden layers that combine together to create more complicated functions. In other words, more depth means that the network can model more complex representations of the data that one hidden layer would not be able to represent [25]. For example, convolutional networks are able to process images by extracting features such as lines and curves that are combined together to creates recognisable forms. As we get deeper in the layers, the representations become more abstract and it is often complicated to understand the final prediction of a network. For this reason, neural networks are often considered as black box models. Nonetheless, we can try to understand the decisions of simple models based on the connections between their layers and depending on what neuron is activated for a specific input.

Other Considerations

Aside from the width and depth of the network, one important aspect of deep neural network is the type of hidden units. The function that one unit is performing is the reason why neural networks are able to learn complex non-linear functions. Many researches have been devoted to this topic. The most common are depicted in Figure 1.7. Sigmoid functions such as the logistic sigmoid and the hyperbolic tangent (respectively in

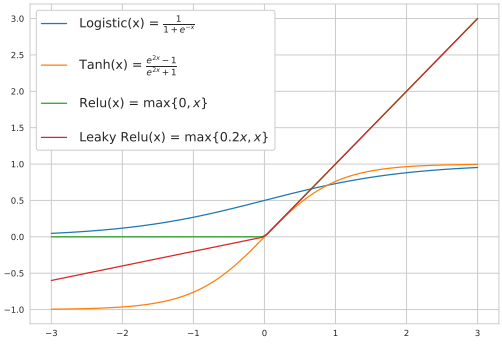


Fig. 1.7: Common activation functions

blue and orange on the graph) were long considered the best choice for feedforward networks because they are smooth and easily derivable. However, they saturate on the most part of their domain (they have a flat slope for large absolute values) which makes it difficult for gradient-based optimisation to know in which direction the parameters should be updated. For this reason, their use in feedforward networks is nowadays discouraged in favour of the rectified linear units (ReLU, the green and red lines on the graph) [25]. Although functions from the ReLU family are non-differentiable in zero, they proved to be extremely good choices for activations. When designing a neural network, another important consideration is the initialisation of the model parameters. A good initialisation strategy is critical because it affects strongly the convergence of the model. If the initial values of the weights are unstable, the optimisation process can encounter numerical difficulties and the training will fail. Further, the initialisation affects how quickly an algorithm will converge. For examples, if all the parameters are assigned the same initial value, the units will end up computing the same function because the optimisation algorithm will update the weights in the same way. Consequently, the initial parameters must be set to different values in order to *break symmetry* [25]. Generally, the biases are initialised to a constant that is known to provide good results (e.g. zeros) and the weights are initialised randomly by using a parametrised probability distribution such as the uniform or the normal dstributions. Glorot and Bengio [22] researched this problem and provided a heuristic to set the weights of the network by drawing them randomly from a Gaussian distribution of mean 0 and variance that depends on the number of connection to and from each unit. This strategy, known as the Xavier initialisation from the name of the main author, Xavier Glorot are particularly efficient for ReLU activations.

Objective Functions

Depending on the task that we want a model to learn, one objective function might be more appropriate than another. For example, a regression task where the output is a real-valued number, the mean squared error is generally a good choice. For classification tasks, where the output can be one of K classes, the cross-entropy loss is generally preferred:

$$CE(x_i, y_i) = -\sum_{c=1}^{K} y_{i,c} \log(p(y_i = c|x_i))$$

where $p(y_i = c|x_i)$ is the probability calculated by the model that input x_i belong to the class y_i. This loss function will have the effect of penalising more strongly the network when the probability of the right class is far from 100%. Finally, when

the output should be a probability distribution, one can use the Kullback-Leibler divergence:

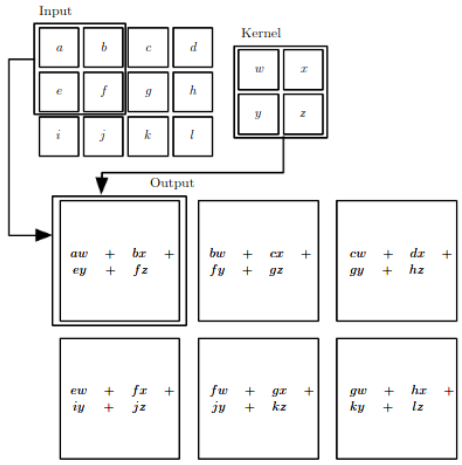
$$KL(P||Q) = -\sum_{x \in X} P(x) \log \frac{Q(x)}{P(x)}.$$

where P is the target distribution and Q is the distribution predicted by the network.

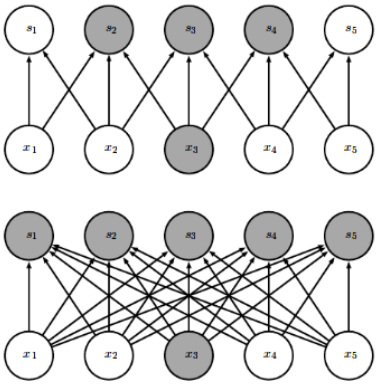
1.2.2 Regularisation for Deep Learning

Neural networks are prone to overfitting, perhaps even more so than any other machine learning models. For this reason, it is important to set up strategies to detect and avoid it. The Occam's razor principle states that between two models that perform equally well, the simpler one is the most likely. Regularisation offers a way of controlling the complexity of the model. This can be done by explicitly imposing constraints on the model parameters or activations, by randomly dropping out hidden units during training or by introducing noise in the training process. Overfitting detection can be achieved by monitoring an approximation of the generalisation error, typically the validation loss. Then, one can use early stopping to end the training if the performance starts to worsen as discussed in Section 1.1.2. Another popular technique is to impose constraints on the weights to enforce some desirable properties. For example, weight decay, or *l2*-regularisation turn the unconstrained optimisation problem $\min J(\theta)$ to a constrained optimisation problem $\min \mathcal{L}(\theta) = \min J(\theta) + \lambda \sum_i \theta_i^2$, where $J(\theta)$ is the prediction error. This additional term penalises large values of the weights. The hyperparameter λ trade a model that fits the data well with one that is simple. It should be chosen as to maximise the performance on a validation set. *l1*-regularisation uses a constraint on the absolute value of the weights, $\sum_i |\theta_i| = 0$, which makes sure that the parameters are sparse. Sparsity is desirable in representation learning and thus often used in models such as autoencoders which will be discussed in Section 1.2.6. More advanced constraints exist that favour specific properties of the parameters and will be discussed in time in Section 1.2.6. Dropout is another popular technique to decrease the risk of overfitting in neural networks. During training, hidden units are randomly dropped with a fixed probability p. This result in a reduced network and forces the model to learn more robust features. Dropout is attractive because it is simple, computationally efficient and leads to good performances. However, it has been shown to be less effective when only a few labelled examples are available [63].

1.2.3 Convolutional Neural Networks



(a) Example of 2D convolution with a 2x2 kernel with a stride of 1

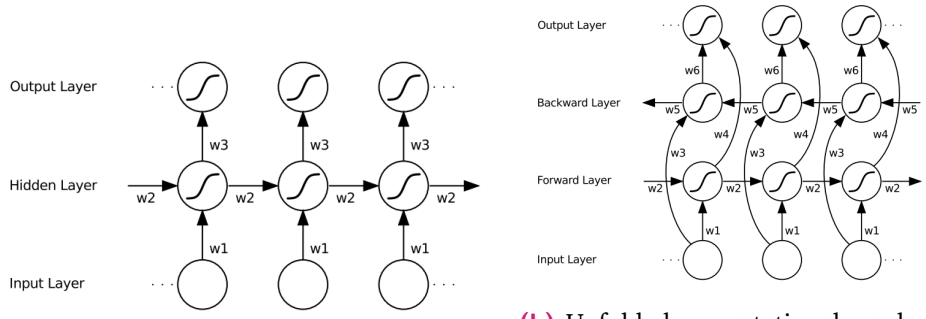


(b) Sparse connectivity of a CNN with a kernel of size 3 (Top) and fully-connected layer formed by matrix multiplication (Bottom)

Fig. 1.8: Convolutional Neural Networks use sparse connectivity and are parametrised by a fixed size kernel. Source: Goodfellow et al. [25]

Convolutional networks (CNN) [41] are widely used in computer vision tasks. They provide considerable computational advantages over older techniques through shared parameters and local connections (Figure 1.8b). They also naturally support invariance to translation which makes them particularly well suited for spatial data such as images, where features can appear in different locations of the input data. CNNs are a special kind of feedforward networks that use convolution operations to extract abstract features from the input data such as lines and colour blobs. These are combined in deeper layers into recognisable shapes. The convolution operation is best explained by imagining a window, or kernel, that slides by one or more pixels at each step until the whole image has been scanned. Local matrix multiplication between the weights of the kernel and the value of each pixel below the window are performed at each position (Figure 1.8a). Generally, multiple convolutional filters are applied in one layer, each with a different kernel. As the window traverses a picture, the network identifies relevant features or aspects of the image. These features are combined together in the next layers and build more complex concepts. The first layers of a CNN typically learn to identify edges at different orientations and scales. Temporal or 1D convolutions apply the same principle to sequential data such as text and time series and have shown good performances on text classification tasks and time series predictions. However, CNN generally require a lot of training data which poses a problem when only few labels are available.

1.2.4 Recurrent Neural Networks



(a) Unfolded computational graph of a recurrent neural network

(b) Unfolded computational graph of a bidirectional recurrent neural network

Fig. 1.9: Computational Graph of Recurrent Neural Networks. Source: Graves [26]

Feedforward neural networks work well on many tasks, but they fall short on sequence modelling, where it is important to capture time dependencies. Although CNNs have been successfully used to extract time information, they are shallow: we have seen that the output of a convolution is a function of a small number of neighbouring members of the input. Recurrent neural networks (RNN) are specialised neural networks for the processing of sequences $x_1, ..., x_T$. RNNs share parameters across time steps by reusing the output of one time step to compute the output of the next time step, or as Goodfellow et al. [25] put it, "*each member of the output is produced using the same update rule applied to the previous outputs*". Figure 1.9a shows the computational graph of a recurrent neural network that uses the result of the previous time step to compute the output of the current time step. Each node represents a layer of network units at a single time step and the network is parametrised by three sets of weights shared across time steps: $w1$, $w2$ and $w3$ and biases (not represented). Compared to PGMs, which require an exponential number of connections to model a conditional distribution, RNNs have a more efficient parametrisation. It should be noted that the elements in the input layer do not need to explicitly arise from the passage of time and can simply denote of elements at different positions in the sequence, such as words in a text or nucleotides in a DNA sequence. This means that the connections can go backward in time, provided that we observe the entire sequence before it is processed by the network. A technique that is popular in natural language processing is thus to process a sequence both forward and backward and use the result calculated in both directions to output a prediction for a time step that depends on the whole input sequence, as shown in Figure 1.9b. The input nodes are connected to a forward and a backward layers which include connections between time steps. The output of both recurrent layers are summed together in the output layer and a non-linear function is applied. If the sequence has many steps, the computational graph is unfolded to a very deep neural

network, which makes it particularly sensitive to the problem of vanishing gradients [26].

LSTM [28] and other gated recurrent units (GRUs) [9] came as a solution to this problem by introducing gated recurrent units that allow the network to store information over a long duration and the ability to learn when it should be kept in memory and when to release it. Figure 1.10 shows a LSTM cell. The **input gate** controls whether the input value should be accumulated into the state unit, the **forget gate** controls when the network needs to store the information or release it and the output gate decides whether to use the value in the state cell. LSTMs learn long-term dependencies more easily than simple RNN architectures, but they do not completely solve the problem of vanishing gradients and thus still fall short when modelling sequences with more than a thousand steps. Consequently, very long sequences are still considered a challenge and LSTMs generally perform better on limited-size sequences such as short documents and sentence classification [42, 53]. Another issues with RNNs and LSTMs is that they are not hardware friendly and training them might take a long time.

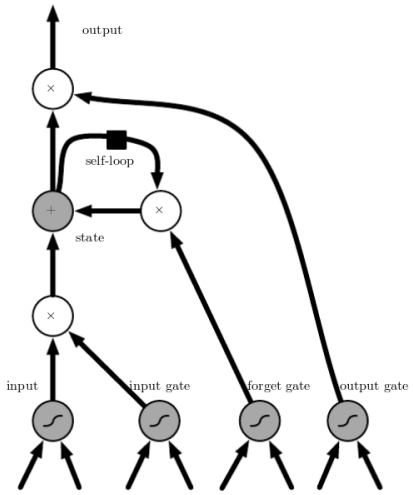
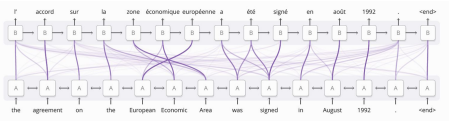


Fig. 1.10: Long Short-Term Memory. Source: Goodfellow et al. [25]

1.2.5 Attention Mechanisms

Since 2015, the concept of attention mechanisms has been the source of many publications in the deep learning community. This technique was largely adopted by the deep learning community and gave rise to many research papers, effectively replacing recurrent neural networks as the state of the art for many natural language processing tasks. Indeed, many models based on the Transformer, introduced in Vaswani et al. [67], have successively surpassed each other in the years 2018 and 2019 only [17, 18, 15, 72], which lead us to assume that this is only a beginning. How does it work? An attention layer equips



(a) In neural machine translation, attention is used to learn sentence alignments [51, 2]

(b) Visualisation of the attention weights applied to an image captioning task [71]

Fig. 1.11: Illustration of attention mechanisms

a neural network with the ability to learn which part of the input vector is relevant for predicting the output. During training, the network learns a probabilistic mask that is used to scale the input vector. The attention weight for one input feature is proportional to the expected relevance of the variable for making a prediction. For example, if the task is to translate a text from French to English, the model learns to identify for each fragment of the sequence in French the portions of the English sentence that contain the relevant information (Figure 1.11a). In a computer vision task where the objective is to generate a caption in English, attention is used to highlight the portion of the image relevant for making a prediction (Figure 1.11b). Aside from the major performance boost that attention can mean for predictive model, it also provides major advantages in terms of result explanability, an area that has been highly problematic when it comes to deep learning.

1.2.6 Autoencoders

Autoencoders are unsupervised models that are trained with the same techniques used in supervised learning. They have been widely used for dimensionality reduction, representation learning or image denoising. Autoencoders are composed of an **encoder** and a **decoder** that are chained together. The encoder learns a function $h = f(x)$ that maps an input x to a fixed-size representation h, named **code** or **encoding**. The decoder learns to reconstruct the original input from the code, $\tilde{x} = g(h) = g(f(x))$. Since the ground truth, or expected output of the model is known, this model can be trained with the same optimisation techniques as supervised neural networks. Typically, we learn the parameters by minimising a reconstruction error $L(x, g(f(x)))$ that is large when the input and the reconstruction are dissimilar. Obviously, we do not want the network to overfit, which in the case of autoencoders translate into trivially learning the identity function $g(f(x)) = x$ by storing the whole dataset in its hidden layers. In general, if the model is allowed to have too much capacity it will not learn anything useful. To prevent this, the model is often constrained in some ways to force the encoding to demonstrate some useful properties like sparsity of the representation or invariance to small variations of the input. Autoencoders have been around a long time and when deeper architectures were necessary, it was common to train the autoencoders layer by layer. Stacked autoencoder are composed of multiple layers in the encoder and the decoder where the first layer is trained as a shallow autoencoder. Then, the weights are frozen and the next layer is added to the model. This method is quite cumbersome and with the increased computing power, autoencoders nowadays are generally trained all layers at once. In the next paragraphs, we will review some of the most popular architectures of autoencoders.

Undercomplete Autoencoders

The simplest constraint that we can impose to the autoencoder is to limit the dimension of the encoding to be smaller than the input x. In that way, the network is forced to learn a compressed representation of the input x by identifying the most important features in the training data. If the autoencoder uses only linear activation and the cost function is the mean squared error, then it can be shown that the model is performing principal component analysis (PCA) [21]. By using non-linear activations, the autoencoder can learn more powerful low-dimensional representations than PCA [25]. Autoencoders are symmetrical models where the decoder match the encoder architecture (Figure 1.12). In such architecture, we often tie the weights of the decoder to the weights of the encoder. Then, the weights of the decoder are equal to the transpose of the weights of the encoder: $W_d = W_e^T$. By doing so, the training process is faster and the risk of overfitting is reduced because the number of weights in the model is divided by a factor of almost 2. It should be noted that the biases must remain independent.

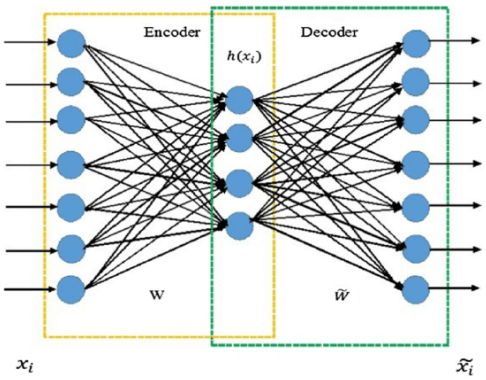


Fig. 1.12: Two layers undercomplete autoencoder with one hidden layer. Source: `http://cican17.com`

Denoising Autoencoders

Aside from limiting the dimension of the encoding, one obvious technique to make sure that the autoencoder does not learn to copy the input data in its weights is by adding small random noise to x. A **denoising autoencoder (DAE)** learns to reconstruct an input given a corrupted version of it. Consequently, the DAE learns a function $p(x|\tilde{x})$ where $\tilde{x}$ is a corrupted version of the original input x. The amount of corruption to the input is controlled with a parameter that can be tuned to control overfitting. Aside from representation learning which is common to all autoencoders, DAE have interesting applications in cleaning noisy images and optical character recognition (OCR) to translate handwritten texts into digital documents [69].

Sparse Autoencoders

Regularisation provides a way to use autoencoders with a large capacity and still learn interesting patterns in the data distribution while enforcing some properties in the encoding. For example, a **sparse autoencoder (SAE)** learns encodings where most of the units are small. These representations are typically used as input to perform another task. It has been shown that forcing the representation to be sparse leads to better results classification scores [45]. Different methods exist to enforce sparsity. One of them is to add a $l1$-constraint on the activation of the encoder. This forces the encoder to learn representation that are mostly zero:

$$J_{sparse}(\theta) = J(\theta) + \lambda \sum_i |h_i|$$

where $J(\theta)$ is the reconstruction error, h_j is the jth activation in the hidden layer and λ is a hyperparameter that controls the sparsity. Glorot et al. [23] showed that it is possible to achieve actual zeros in the code by using rectified linear units (ReLU) as activation function of the encoder in conjunction with $l1$-regularisation. Another way to enforce sparsity is to use the definition of the Kullback-Leibler divergence. The KL divergence measures how similar or dissimilar two probability distributions $P(x)$ and $Q(x)$ are. If both distributions are discrete and defined on the same space X, it is defined as:

$$KL(P||Q) = -\sum_{x \in X} P(x) \log \frac{Q(x)}{P(x)}.$$

Let $a^{(j)}$ be the activation of the jth hidden unit in the encoding layer and $\hat{\rho}_j$ be the activations of this hidden unit averaged over the training set. Formally, we want $\hat{\rho}_j = \rho$ where ρ is a sparsity parameter whose value is close to zero. By adding a penalty term to the objective function that encourages $\hat{\rho}_j$ to be close to ρ, we force the activations to be mostly close to zero [52]:

$$J_{sparse}(\theta) = J(\theta) + \lambda \sum_{j=1}^{|D|} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$

where $|D|$ is the dimension of the encoding. The penalty term described above is in fact the Kullback-Leibler divergence between a Bernouilli random variable of mean ρ and a Bernouilli random variable of mean $\hat{\rho}_j$ [52].

Contractive Autoencoders

Contractive autoencoders (CAE) [58] use a loss function that balances the reconstruction error with a regularisation term that penalises large derivatives in the

encoder. In other words, it encourages two similar inputs to have similar encodings and therefore produces representation that are robust to infinitesimal perturbations of the inputs. The penalty is proportional to the squared Frobenius norm of the Jacobian matrix:

$$\Omega(h) = \lambda \left\| \frac{\partial f(x)}{\partial x} \right\|_F^2 .$$

Minimising the reconstruction error alone encourages the autoencoder to learn the identity function while the contractive penalty alone would lead to features that are constant with respect to x. Balancing these two quantities gives an autoencoder whose derivatives are mostly tiny [25]. Rifai et al. [58] show that tying the weights of the encoder and the decoder helps not to learn the identity function.

k-autoencoders

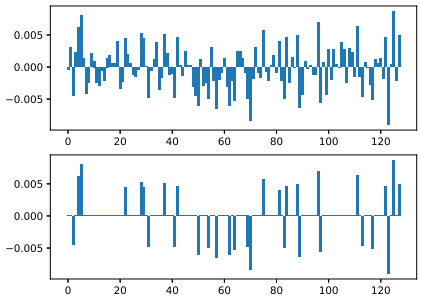


Fig. 1.13: The k-sparse autoencoder selects the top and bottom activities and resets the others to zero.

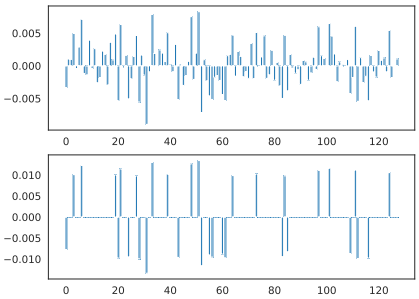


Fig. 1.14: The k-competitive autoencoder resets low-activity units and redistributes the lost energy among the "winners".

Finally, Makhzani and Frey [45] proposed a method for learning sparse encodings, called **k-sparse autoencoder (KSAE)**. During the training phase, this model only keep the k highest activities and reset the other to zero. Because this algorithm specifically use linear activations, the only non-linearity comes from the selection of the k largest activities. Figure 1.13 shows an example of a vector before and after filtering the relevant activities. The authors show that the learned representation can be efficiently used in image classification tasks.

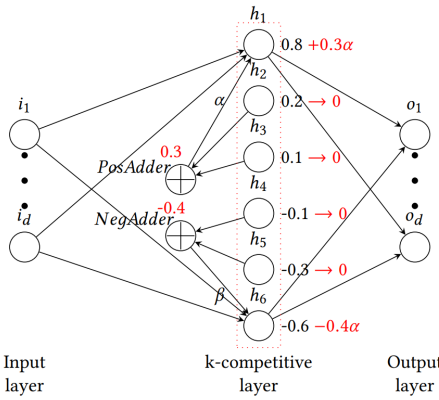


Fig. 1.15: Competition among neurons.
Source: Chen and Zaki [8]

Building on this, Chen and Zaki [8] proposed the **k-competitive autoencoder for text (KATE)**. Aside from selecting the top-k activities, it introduces competition in

the encoding layer by reallocating among the remaining units the energy lost after dropping low activity neurons. The energy reallocation is proportional to the sum of the activations that were set to zero. It encourages the hidden units to compete for the right to respond to a given set of input patterns. A hyperbolic tangent activation is used in the hidden layer. Figure 1.14 shows an example of a vector before and after reallocating the energy and Figure 1.15 demonstrates the energy reallocation in the encoder layer as it was proposed in the original unsupervised model.

Multi-task Learning

By nature, autoencoders are unsupervised. They learn to reconstruct their input without the need for labelled data. However, they can easily be adapted to be able to perform two related but different tasks. The idea behind combining two different objectives is that the joint training process might lead to a model that performs better both tasks than if multiple models had been trained separately to solve each of the individual tasks by combining the advantages of both approaches [78]. By training a model to not only minimise the reconstruction error but also reduce a classification objective, we can hope to achieve a better classification score by harvesting the intrinsic distribution of the data learned by the autoencoder while achieving better representations with the help of the classifier. This means that a potentially small number of labelled data might be enough to reach performances that would have been impossible without the unsupervised objective. In practice, semi-supervised autoencoders look just like unsupervised ones with the difference that a classifier layer is connected to the encoding layer (Figure 1.16). The reconstruction and classification losses are generally combined together and a parameter controls the relative importance of each function in the global objective function [62]. The network is trained by minimising the total loss function with gradient descent or a similar algorithm. These architectures have shown good performances on computer vision tasks and short sentences modelling but have yet to be used for long text documents modelling.

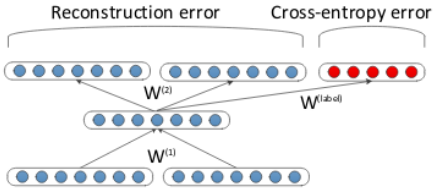
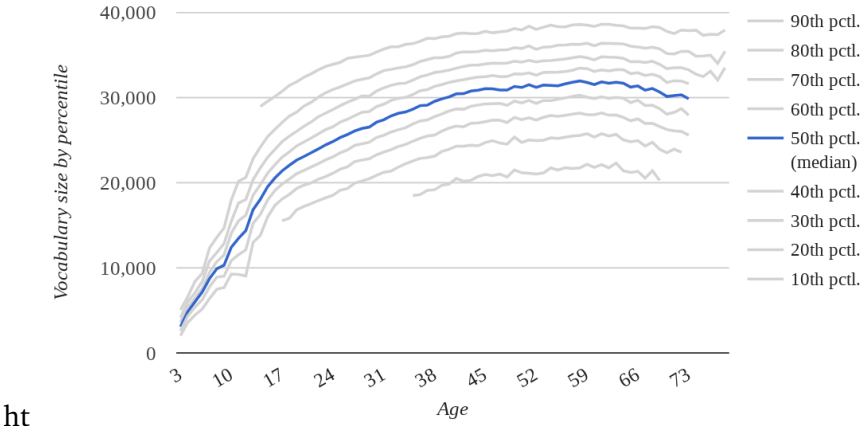


Fig. 1.16: A semi-supervised autoencoder combines an unsupervised autoencoder and a supervised classifier [62]

Other Architectures

Many other variants exists for autoencoders, although the same principles described in the previous sections apply. For example, when the dataset is composed of im-



ht

Fig. 1.17: English native speaker vocabulary by age and survey percentile (486,496 responses) Source: Baldwin [3]

ages, convolutional autoencoders are a natural choice. For short to medium size sequences, RNN based architectures have become popular and LSTM autoencoders have demonstrated good performances in various natural language processing tasks [14]. However, RNN autoencoders suffer from the same problems as RNNs which makes them not well suited for the processing of long documents. Autoencoders can also be trained in an adversarial manner, following the principle popularised in Goodfellow et al. [24]. In a nutshell, a discriminator network is connected to the encoding layer of the autoencoder and is trained to predict whether a specific encoding was generated by the encoder or was sampled from a given distribution. The autoencoder is trained to fool the discriminator while minimising the reconstruction error. The gradient of the error propagated from the discriminator to the encoder acts as a regulariser that forces the autoencoder to generate encoding that match a prior distribution $p(z)$. This specific architecture has been used in the field of computer vision in all three learning settings [46].

1.3 Natural Language Processing

Natural language processing (NLP) is a subfield of artificial intelligence that focusses on creating computer systems that are able to perform useful tasks involving human language [33]. Popular tasks include language generation (producing realistic texts), named entity recognition (extract places, people, companies, etc. in the text), topic extraction (recognise the main topics in a discourse), opinion mining (identify subjective information), text categorisation (assign a label to a text such as spam or not spam), machine translation (automatic translation of a document from a source language to a target language), etc. The focus of the present work is on semi-supervised classification of long text documents. Although the first attempts at making such software date back to the 1950s, it is only recently, with the increased

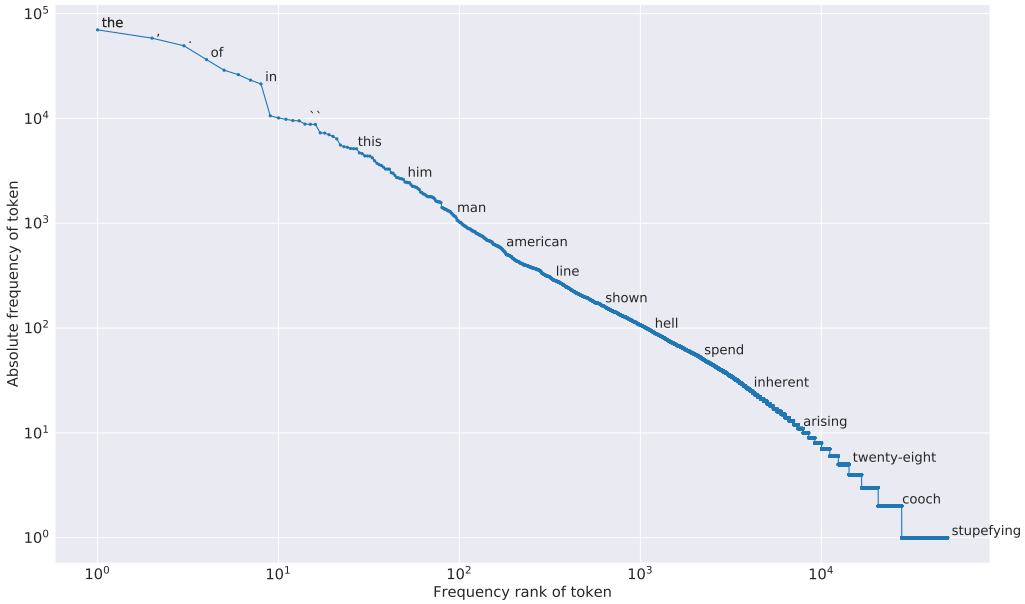


Fig. 1.18: Zipf plot for Brown corpus [20] tokens

processing power and the discoveries in the field of deep learning that such systems started to match and even exceed human performances. Indeed, although children master natural languages very quick, building computer systems able to do the same thing is no easy task. Human languages are ambiguous and incomplete and understanding them often require a good amount of common sense that cannot be codified by a set of static rules [51]. Additionally, these systems must be able to deal with enormous amounts of data. As Figure 1.17 shows, the median English native speaker knows over 20,000 words at 18 and over 30,000 at 40. Furthermore, word occurrences in documents follow a power-law distribution, often denoted as $x(k) = x_M/k$ where the frequency of any word $x(k)$ is inversely proportional to its rank k in the frequency table [13]. The most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc. [57] (Figure 1.18). It also means that the vocabulary size for a text grows significantly faster than the square root of its length in words [33]. Consequently, natural language processing systems must be able to handle highly dimensional and sparse data.

1.3.1 A Brief History of NLP Approaches

In the early days of natural language processing, much of the effort to build intelligent systems leveraged human knowledge of natural languages. These systems relied on hard-coded sets of rules and well-defined grammar to extract information from a text. However, these methods were not scalable as they were not robust to small variations of the language and relied on large resources that are difficult

to create and maintain. In the 1970s, statistical methods that did much more computations proved to be more efficient and came to dominate the field in the following decades [65]. Decision trees were very popular in the early stages as they produce sets of rules similar to the previous rule-based approaches. In the mean time, statistical inference with probabilistic graphical models (See Section 1.1.4) gained in popularity after achieving much greater success, for examples in part-of-speech tagging (assign a label to a word denoting the nature or the word (part-of-speech)) with Hidden Markov Models (HMM) [33] and Naive Bayes and Logistic Regression models have proved to be very efficient in document classification. Another famous use of statistical models was brought on by IBM researchers to solve machine translation. Later on, deep learning models like LSTMs and CNNs became the new state-of-the-art by significantly improving the performances over older techniques. Nowadays, neural networks equipped with attention mechanisms show the best performances in various NLP tasks.

1.3.2 Representation of Text Data

Two main approaches are usually preferred to handle text: process the text sequentially as a human would read it or summarise it by creating so-called bag-of-words, or bag-of-features, where a document is represented as a vector of its components. In the first case, the order in which each token arises is conserved whereas in the latter case, the order in which the components appear is discarded and the features that are considered relevant to the engineer are compiled together. As machine learning models generally require numerical inputs, machine learning practitioners can spend a large amount of their time selecting and engineering features that capture the most information for a particular task. Nowadays, as the trend has shifted towards deep learning, artificial neural networks often learn these representations automatically in their hidden layers. For example, a convolutional network trained to classify images will learn abstract features such as lines and shapes in their first layers and combine these into more abstract features in the subsequent layers. Similarly, a neural network trained on text might learn to identify the words that belong to the same lexical registry.
In the next paragraphs, we discuss various methods of transforming and processing texts for machine learning tasks.

Categorical Feature Representation and Preprocessing

From a natural language perspective, a text can be decomposed in hierarchical components, namely paragraphs, sentences, phrases, words and characters. The first step when building an NLP system is to decide the level at which the analysis

should occur. Above the word level, one should recourse to advanced abstract representations that often combine features of smaller components. In this work, we focus on words as the main components of text documents.
The easiest way to think about how to represent categorical features is to create so-called one-hot vectors. Feature i will thus be encoded as a vector $x = (x_1, ..., x_n)$ where $x_j = 1$ if $j = i$ and $x_j = 0$ otherwise. We keep track of an index, or **vocabulary**, that maps index i with feature w_i. Obviously, this gives very sparse matrices where only a handful of values are not null. In NLP, the features are often words or **n-grams**. N-grams are groups of n terms that, when taken together, carry a different meaning than each of the individual word. For example, 'White House' carries a different meaning than the words 'white' and 'house'. Using n-grams has the effect of increasing exponentially already large vocabulary sizes and there is a trade-off between the amount of information that should be kept and the dimensionality of the data.
Using one-hot vectors results in very high dimensional datasets. Although, the size of the vocabulary can be reduced to some extent by applying some preprocessing rules such as removing infrequent words or words that appear in all documents, regardless of their class. For example, in the case of topic modelling, a lot of words in a text document do not give relevant information to determine the topic. For instance, stop words are words that occur all the time and that are not specific to a particular topic, such as 'and' and 'the'. Obviously, for other tasks such as language generation, these words are necessary to produce meaningful texts and should not be discarded but in other cases, dropping them significantly reduces the number of words to be dealt with. Finally, one can reduce the vocabulary size by applying lemmatisation and stemming. These techniques truncate words to a general form. For example, the words 'approximation' and 'approximating' might both become 'approxim'.

Bag-of-words

One easy way to represent documents is to encode them as bags-of-words (BOW). A document is then represented as a 1d-vector of length n where n is the number of tokens or expressions in the vocabulary. This provides the advantage of creating fixed-length feature vectors required by many machine learning algorithm while keeping the dimensionality within bounds. The resulting vectors can be binary (for example denoting the presence or absence of a term in the document), integer-valued (such as term counts) or real-valued (such as term frequencies). A commonly used feature is the term-frequency inverse-document-frequency (tf-idf) [31] which is inversely proportional to the number of documents in the corpus where a term occurs

and directly proportional to its frequency in the document. Often, the smoothed inverse-document-frequency is used:

$$\text{idf}(t) = \log \frac{1+n}{1+\text{df}(t)} + 1$$

where n is the total number of documents in the document set, and $\text{df}(t)$ is the number of documents in the document set that contain term t. The resulting vector is normalised using the Euclidean norm or the sum of absolute values:

$$v_{norm} = \frac{v}{\|v\|^2} = \frac{v}{\sqrt{v_1^2 + v_2^2 + ... + v_n^2}}$$

The tf-idf score of the index term i in document j, w_{ij}, is then given by $w_{ij} = tf_{ij} \cdot idf_j$ where tf_{ij} is the frequency of the term i occurs in document j and idf_{ij} is the inverse document frequency of term i in the corpus. In some cases, for example when the term frequency is misleading such as in documents with different lengths, it can be useful to perform length normalisation: $\widetilde{\text{tf}_i} = \frac{\text{tf}_i}{\max_k tf_k}$. It is easy to use idf scores to identify and filter out stop words or uncommon words in the corpus.

Word embeddings

Handling large vocabulary sizes can be impractical for various reasons. BOW representations provide a way to keep the dimensionality within bounds but present two major weaknesses: they lose the ordering of the words and ignore semantic information. Mikolov et al. [48] introduced a technique to represent words as dense vectors, called Word2Vec. The main idea is that words that share a similar meaning often appear in similar contexts. Figure 1.19 shows two models to learn such representations, or **word embeddings**: the continuous bag of words (CBOW) and the skip-gram model. In the first case, the goal is to predict a word given its context, $P(w_t|w_{t-k:t-1}, w_{t+1:t+k})$ for a given context size k. In the second case, the model predicts the context given a certain word: $P(w_{t+i}|w_t)$, where $i \in \{1, ..., k\} \cup \{-1, ..., -k\}$. In both cases, the hidden layer is the embedded representation of the word and the weight matrix of the first layer is used to project the words as one-hot vectors in the embedding dimension.

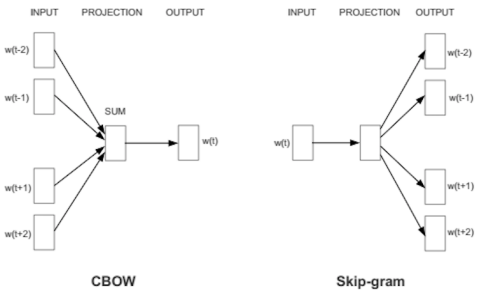


Fig. 1.19: Word2Vec models Source: `https://skymind.ai/wiki/word2vec`

Word embeddings have proved to be particularly efficient for various natural language processing tasks and many deep learning models include an embedding layer

as their first layer. Learning embeddings can be done as an unsupervised pretraining task, and/or fine-tuned as part of the main task. Further, multiple general purpose embedding matrices trained on very large corpora are available online (see for example GloVE [55]) and can be used to initialise the embedding matrix. However, even with pretrained embeddings, further tuning is often necessary to get embeddings that are specialised to a given corpus. These word vectors have interesting properties. First, traditional similarity and distance measures can be used to compare different words. Words with similar meanings, such as synonyms, or belonging to the same semantic registry have a strong similarity, while words that never occur together are distant from each other. Additionally, arithmetic can be used on the vector representations. For example, the difference between the word vectors for 'Paris' and 'France' added to the word vector for 'Germany' gives a word vector close to the word representation for 'Berlin'. They also provide the advantage to considerably reduce the dimensionality of the features from very sparse one-hot representations of the size of the vocabulary to typically 50 to 300 dimensions. Obviously, such dimensionality reduction comes at a price. For example, sub-word level representation is not possible, like capturing the meaning of suffixes such as -less, denoting 'lack of'. Neither one-hot representations nor word embedding are able to manage out-of-vocabulary words, where a word is encountered in the evaluation that was not encountered in the training phase.

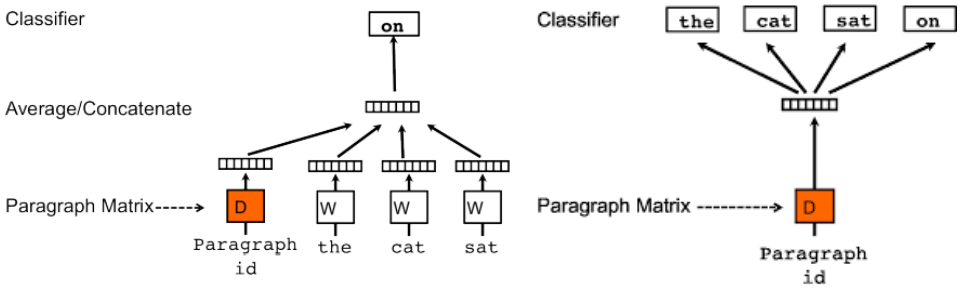
Text as sequences of words

The most natural way to process a text is to read it, one word at a time with a model able to handle sequences such as HMMs, CRFs, RNNs or temporal CNNs. This is the preferred method for tasks where the relation between the words is important such as part-of-speech tagging, neural machine translation and sentiment analysis. However, as the length of the texts increases, these models tend to fall short, whether because of the increased computational complexity or the inability to model long-term dependencies or variable size inputs. Liu et al. [42] gives the example of a mid-sized document of around 10,000 words, where the words are embedded as vectors of size 300: such a representation will require several gigabytes, more than most GPUs on the market. Aside from the large computational and memory requirements involved, such a representation can be impractical if a small subset of the features turns out to be enough to perform some tasks well enough. For example, topic modelling and text classification of long documents do not require to understand all the intricacies hidden in the text and a high level understanding of the main topics is often enough to get good performances. Consequently, for long documents and for tasks where the order of the words is not essential, other representations are often preferred.

Text corpora as undirected graphs

On a final note, it should be said that other methods for representing documents exist such as undirected graphs. This is the case, for example, of Yao et al. [74] where a collection of documents is encoded as a large graph with documents and words as nodes where edges represent global word co-occurrences in the corpus (with an adequate metric such as frequency of occurrence or pointwise mutual information) or word occurrences in documents (term frequencies or tf-idf scores). A CNN (named Text GCN) learns topics and performs classification based on the information encoded in the graph.

1.3.3 Representation Learning



(a) Distributed Memory model of Paragraph Vectors (PV-DM)

(b) Distributed Bag of Words model of Paragraph Vectors (PV-DBOW)

Fig. 1.20: Distributed representations of sentences and documents [40]

Sometimes, it is best to learn features automatically as part of a learning task rather than using manually engineered features. The reason is that identifying which kind of features is relevant and how to best extract them from the data is not a trivial task. Instead, letting the model decide how to build the right features for a specific task can often improve the generalisation capability of the algorithm. This is something that researchers in computer vision have realised with CNNs and is probably one of the main reasons of the large adoption of deep learning in the AI community. Similar to a model learning by itself to detect edges and shapes in a CNN, an NLP model learns to identify patterns and configuration in an input sequence that is relevant to predict its class. Word embeddings are an example of such automatically learned representations. In this work, we are interested in semi-supervised long document classification. Having a framework that allows us to summarise documents as dense vectors where documents that cover the same topics yield vectors that are close to each other would be beneficial. Building on the idea of word embeddings, Le and Mikolov [40] proposed to learn paragraph vectors to efficiently create high level representation of larger chunks of texts. Finding clever representations of the input data has considerable advantages. Aside from being able to represent documents

with tensors of limited dimensions, which has obvious interest in the storing of documents, it allows encoding documents with meaningful features that can be used in information retrieval or other NLP related tasks such as categorisation and topic modelling. Another major motivation is the ability to learn these representations in an unsupervised or weakly supervised setting. Labelled data is therefore not always required. Nonetheless, it has been shown that having some labels available can sometimes help the optimisation algorithm to converge faster and to build representation with interesting properties [30].

When the length of the texts in the corpus is small (sentences or short texts), RNN-based encoder are the go-to model because they process an input sequentially and in doing so, are able to model the order in which each feature occurs. In this regard, LSTM networks are the most popular models. They use word embeddings to represent each word component and learn a time-sensitive dense representation of a sentence by processing it token by token from left to right, and sometimes right to left as well (bidirectional LSTMs). However, they fail on large chunks of texts and are thus not well adapted to our problem of working on long documents. In computer vision, CNN are used to generate abstract representations. Further, it has been shown that a deeper network allows learning hierarchical abstract representations that greatly simplify the learning task [25]. Conneau et al. [12] showed that the same observation can be made for documents although creating shortcuts or wormholes are necessary to help the gradients flow through the network [27]. It is worth noting that an increase depth often comes at the extent of explanability and it is therefore difficult to interpret the representation that the network learns. Further, CNNs are known to be data greedy, which makes this approach valid only when many examples are available. Finally, Johnson and Zhang [30] show that it is possible to learn multiple region embeddings by using parallel CNN layers with kernels of different sizes. This approach can be seen as extracting information from n-grams, where n is the size of the kernel. These models are by nature supervised and thus require labelled data. When labels are scarce, we have to turn to unsupervised learning. The following paragraph discusses different unsupervised methods for learning representations of text.

In the distributed memory model of paragraph vectors (PV-DM) [40], the model is trained to predict the next word given the previous words (the context) and a paragraph token. During training, the model learns a paragraph embedding matrix where the rows correspond to the paragraph vectors for each index. The distributed bag of words version of the paragraph vector (PV-DBOW) [40] is trained to predict the words in a small window given a paragraph token, similar to the skip-gram model in Word2Vec. However, in both presented methods, the notion of context is rather limited and not well adapted to long documents. Another method, fastText [32], takes another approach to build document embeddings by averaging the vector representation of its components into one dense vector. Simple word embedding-based models (SWEM) [61] perform pooling operations on word embeddings to extract

the most important features of a word vector and combine them with each other into a single representation and have managed to build representations that achieved outstanding performance in long document classification. It is also possible to combine representations from different objects into one, for examples in cross-modal information retrieval, it is common to combine image and text representations into one low-dimensional vector. Label embedding attentive model (LEAM) [68] uses label descriptors to create label embeddings that are combined with word vectors into the same latent space. Then, the representation of a sequence is generated by combining the different dimensions in each of the components depending on probabilistic weights learned in an attention mask. Finally, word embeddings can be harvested to create a similarity graph on which CNNs can be applied to create dense representations much like in image processing [16].
We have already discussed the ability of autoencoder to create abstract representations (Section 1.2.6). The same principles apply to text data. We have seen that autoencoders can have many architectures. For short text and sentence modelling, RNN-based autoencoders perform very well to learn language-specific features. One-dimensional convolutional autoencoders are able to analyse language and learn parsing trees of sentences [34]. For large documents, though, learning the distribution of the word occurrences in a text is often enough. Besides, the distribution follows some specific rules and therefore might be easier to model than the full length document, aside from the advantage of dealing with fixed-length vectors with a limited number of dimensions. When the dimensionality is not too large, it is possible to use autoencoders with fully-connected layers and impose specific constraints on the weights of the encoder and on the encodings (see for example the models presented in Section 1.2.6). Variational autoencoders (VAE) [37] and adversarial autoencoders [46] go one step further and directly impose a prior on the distribution of the code.

1.3.4 Document Classification

Learning to categorise documents can be achieved in different ways, for sequential models, HMMs, CNNs [11, 35, 70, 77], RNNs and attention models [67, 76, 73] and for feature-based models such as BOW features, SVMs, logistic regression, naive bayes and MLPs. Following what we have discussed in the previous section and assuming that we have a representation framework g such as an encoder, able to map an example document x_i to a fixed-size representation ξ_i, building a classifier is straightforward: it comes to learning a mapping f between the representation and the target classes, $y_i = f(\xi_i) = f(g(x_i))$ where y_i is a K-dimensional vector and K the number of classes. The class distribution is given by a softmax activation that

transforms the logits produced by the network into a probability distribution. For example, the probability that document x_i belongs to class k is

$$P(c_i = k|x_i, f, g) = \text{softmax}(y_i^{(k)}) = \frac{e^{y_i^{(k)}}}{\sum_{j=1}^{K} e^{y_i^{(j)}}}$$

In this work, we are mostly interested in using this approach of explicitly learning adequate representations jointly with the classification task in a semi-supervised way. However, fully-supervised models exists that implicitly learn representations as part of the learning process, with the sole difference that there is no explicit definition of the representation framework. Learning thus require a labelled dataset.

2 Approach

The objective of the present work is to explore semi-supervised techniques to automatically classify long text documents when only few labels are available. In other words, a model must successfully identify patterns in the unsupervised distribution $p(x)$ that can be used to learn the conditional distribution $p(y|x)$. To do this, we train the models on a dataset with partially labelled data (training set) and we measure their performance on previously unobserved data (test set). Only the training set is semi-supervised so that the classification performance can be measured on a full dataset. We compare our results with scores obtained on published works performing similar tasks. The present chapter first discuss the different aspects of the proposed models. Then, we describe our methodology and the details necessary to reproduce the results, namely the steps followed in terms of data preparation, model optimisation, hyperparameters selection and performance evaluation.

2.1 Proposed Models

Autoencoder-like architectures turn out to be well adapted for semi-supervised learning. Their two main advantages are their simplicity and their versatility. They are by nature unsupervised models because they do not require labelled data for learning. Yet, they are trained in a supervised way to reconstruct their input, in the meantime learning a non-linear mapping to a lower dimensional space. This means that the models can be trained with the traditional supervised learning tools. Second, the model architecture offers a large number of design choices, for example to learn encodings that demonstrate desirable characteristics or to train a model to perform multiple tasks. When the objectives are complementary, a joint objective can potentially improve both the training time and the final performance over a model trained for one task only. In this work, we want to verify this hypothesis. We do so by training semi-supervised autoencoders and comparing their performance with supervised or unsupervised models with a similar architecture. We have discussed in the previous chapter the difficulties encountered when trying to model a long sequence such as a text document. We overcome this problem by using an approach similar to bag-of-words where a document is summarised by the distribution of its components and the order of the words is discarded entirely. We make a small modification to the traditional BOW models however. Rather than

trying to reconstruct the exact number of words in the document or some tf-idf variant, we summarise a document to a probability distribution of the words. This allows us to use the Kullback-Leibler divergence for the reconstruction loss (see Section 2.1.2). An example document is therefore encoded as a word distribution $p = (p_1, p_2, ..., p_V)$ where V is the size of the vocabulary, $p_i = \frac{c_i+\epsilon}{\sum_j c_j+\epsilon}$ is the smoothed term frequency, c_i is the number of time the word at index i occur in the document and ϵ is a constant set to 10^{-7}. The denominator in the probability ensures that the total probability sums to one and the smoothing constant avoids zero-division when a document after preprocessing has no words left.
In this section, we first describe the semi-supervised architectures that are used in this work. Then, we go over the networks that will serve as baseline to compare the results of our models.

2.1.1 Semi-supervised Autoencoders

All the models presented in this section are semi-supervised autoencoders. They extend previously published unsupervised models that were addressed in Chapter 1 by adding small modifications and introducing a semi-supervised objective function, that is, the models learn to minimise jointly a classification and a reconstruction error. The resulting models have thus three components: the encoder f produces a low dimensional representation ξ of the input X and is parametrised by a set of weights $W_e \in \mathbb{R}^{V \times D}$, biases $b_e \in \mathbb{R}^{D \times 1}$ and activation function $\sigma(.)$; the decoder g produces the reconstruction of the input $\widetilde{X}$ and is parametrised by the weights $W_d = W'_e$, tied to those of the encoder, and the biases $b_d \in \mathbb{R}^{V \times 1}$; and the classifier h predicts the class distribution p_X with weights $W_c \in \mathbb{R}^{D \times K}$ and biases $b_c \in \mathbb{R}^{K \times 1}$, where V, D and K are respectively the size of the vocabulary, the dimension of the code and the number of classes. Thus,

$$\xi = f(X) = \sigma(W'_e \cdot X + b_e)$$

$$\widetilde{X} = g(\xi) = g(f(X)) = \text{softmax}(W'_d \cdot \xi + b_d) = \text{softmax}(W_e \cdot \xi + b_d)$$

$$p_X = h(\xi) = h(f(X)) = \text{softmax}(W'_c \cdot \xi + b_c)$$

The predicted class corresponds to the class with the largest probability: $c^* = \text{argmax}(p_1, ..., p_K)$. All models have only one hidden layer. Aside from an easier learning process, this will come useful when we try to understand various aspects of the trained model and in particular which words are mapped to the same topic. Finally, during the training phase, we randomly drop input features with a given probability that is cross-validated. Although the proposed neural networks share a common architecture, they differ in the various regularisation techniques applied (see Section 1.2.6) and their activation functions.

The semi-supervised autoencoder (SSAE) is a tied autoencoder with and $l2$-regularisation constraints on the weights of the encoder and $l1$-regularisation on the encodings. The penalty on the weights forces them to remain small, while the penalty on the activations, coupled with a ReLU activation enforces sparsity. Weight tying is a common practice with autoencoders where the kernel of the decoder is tied to the encoder's weights. This simplifies the optimisation process by reducing the number of parameters by a factor of almost 2, reducing the risk of overfitting in the meantime. Each regularisation term is proportional to one hyperparameter chosen after the cross-validation.

The semi-supervised contractive autoencoder (CAE), whose unsupervised version was initially presented in Rifai et al. [58], is a tied autoencoder with an explicit regulariser on the encoding proportional to the squared Frobenius norm of the Jacobian matrix of the encoder (see Section 2.1.2). This penalty encourages the derivative of the hidden layer to be as small as possible. In other words, it encourages two similar inputs to have similar encodings and produces representation that are robust to small perturbations of its inputs.

The semi-supervised k-sparse autoencoder (KSAE) extend the algorithm proposed by Makhzani and Frey [45]. Although the model was originally developed to work with images, it can easily be adapted to text. Compared to the original model, we add a semi-supervised objective and we explore a symmetric k-sparse autoencoder, where only the $k/2$ top and bottom activities are kept and the other activities are reset to zero. The rationale behind this decision is that features can be both positively and negatively correlated with a topic.

The semi-supervised k-competitive autoencoder (KATE) builds on the k-sparse autoencoder and introduces competition in the encoding layer by reallocating among the remaining units the energy lost after dropping low activity neurons. The energy reallocation is proportional to the sum of the activations that were set to zero. It encourages the hidden units to compete for the right to respond to a given set of input patterns. A hyperbolic tangent activation is used in the hidden layer [8]. Like for the other models, we propose a semi-supervised version of this model where the classification layer is connected to the encoder output before the resetting and reallocation of energy.

It is important to note that for both the k-sparse autoencoder and the k-competitive autoencoder, the classifier layer is not subject to the operations applied on the code. The reason motivating this decision is that the modifications that these models apply to the encoding layer disturb the learning process of the classification layer. Indeed, the models learns to discard extinct units during the training which are not present during testing, as the sparsity constraint does not apply during the prediction phase. Experimentation proved us right in this regard.

2.1.2 Loss Functions

During the forward pass of a semi-supervised autoencoder, the models estimates the class probabilities and the word distribution in the original document. The loss function is a weighted sum of the classification error (supervised objective) and the reconstruction error (unsupervised objective), similar to the approach followed by Socher et al. [62]:

$$J(\theta) = (1 - \alpha) \times \mathcal{L}_U + \alpha \times \mathcal{L}_S$$

where α is a constant factor that balances the supervised and the unsupervised components of the loss. The supervised loss, $\mathcal{L}_S$, is defined as the cross-entropy between the class distribution predicted by the network $p(y_c|x)$, which is the probability that example x belongs to the class y_c, and the true class y_c:

$$\mathcal{L}_S = -\sum_{c=1}^{C} y_c \log p(y_c|x)$$

It should be noted that for datasets with a limited number of supervised examples, the supervised loss is calculated only for labelled examples. In practice, we multiply the supervised loss with a binary mask denoting of the presence or absence of a label.

The reconstruction error of the autoencoder measures how well the model is able to reconstruct the original word distribution in the document. We use the Kullback-Leibler divergence between the distribution predicted by the decoder and the input distribution:

$$D_{KL}(P \parallel Q) = -\sum_{x \in \chi} P(x) \log \frac{Q(x)}{P(x)}$$

where $Q(x)$ is the prediction of the decoder and $P(x)$ is the true distribution in the document. In all models except for k-sparse and KATE, we obtain the unsupervised loss, $\mathcal{L}_U$ by adding a penalty $\pi(W, h)$ to the reconstruction error. This penalty is a regularisation term on the weights of the encoder W or on its activation h:

$$\mathcal{L}_U = D_{KL}(P \parallel Q) + \lambda \pi(W, h)$$

where λ is a constant factor that balances the importance given to the regularisation term. The tied autoencoder uses $l1$ and $l2$ penalties calculated on the encoder's encodings and weights respectively. The contractive autoencoder penalises the second partial derivatives of the weight matrix to enforce representations that are robust to small variations in the input. For a sigmoid activation, the contractive loss as it was introduced in Rifai et al. [58] is defined as:

$$\|\mathcal{J}_f(x)\| = \sum_{i=1}^{D} (h(1-h))^2 \sum_{j=1}^{|V|} W_{ij}^2$$

where h is the activation of the encoding layer, D is the dimension of the embedding and W is the weight matrix. The k-sparse autoencoder and KATE do not add a penalty to the unsupervised loss because only the most relevant activations are kept during the training phase, which acts as a regulariser and already enforces sparsity in the encoding layer.

2.1.3 Baseline Models

We use one supervised and four unsupervised models as baselines to put into perspective the results of the semi-supervised classifiers. The same preprocessing steps as for the semi-supervised models are performed. The model were designed to match closely the structure of our semi-supervised autoencoders so that we can compare their respective performances and the gains that we get from using both unsupervised and supervised objectives rather than only one. We tweak the loss to simulate supervised or unsupervised networks by setting the parameter α to 1 or 0 respectively. When $\alpha = 1$, the total loss of the model is given by $J(\theta) = \mathcal{L}_S$ and when $\alpha = 0$, the total loss is given by $J(\theta) = \mathcal{L}_U$. In other words, the models are exactly the same except for the supervised (resp. unsupervised) component, which is set to zero. We also compare the results obtained on the dataset with all the labels with those from published models performing the same task. These models are described in Chapter 1. To our knowledge, only one paper published results from semi-supervised learning with different thresholds of labelled data on the same dataset as ours. It will serve as a baseline to compare our semi-supervised results.

2.2 Methodology and Implementation Details

We follow a similar methodology as described in Section 1.1: after cleaning the data, we separate the dataset into three non-overlapping sets to train, monitor and evaluate the models. For each of the proposed model, we define a set of potential hyperparameters and search the resulting space by performing a random search (see Section 1.1.3). The resulting models are finally retrained, and we evaluate their performance on the testing set.

2.2.1 Dataset

The data used for the experiments is the 20newsgroups dataset sorted by dates [50]. It consists of 18,846 messages taken from 20 Usenet newsgroups and is often used as a benchmark dataset in the literature. The newsgroups discuss different subjects such as religion, politics or science. The categories sometimes share some degrees

	Raw data			Clean data		
	train	**validation**	**test**	**train**	**validation**	**test**
count	9,051	2,263	7,532	9,051	2,263	7,532
mean	279	253	259	238	216	220
std	577	510	448	545	488	435
min	13	12	16	0	0	1
25%	105	98	103	71	68	70
50%	166	160	164	128	126	127
75%	277	256	272	234	219	230
max	13,199	10,250	9,168	12,692	10,256	8,878

Tab. 2.1: Number of documents and words per document statistics in the different subsets before and after preprocessing

of similarities, such as “soc.religion.christian” and “talk.religion.misc”, where the topics discussed might have some overlap. The dataset is divided chronologically into 60% training and 40% testing sets for benchmark comparison. We follow the same strategy to split the provided training set in the final training set (80%) and validation set (20%). The resulting training, validation and testing sets are composed of respectively 9,051, 2,263 and 7,532 examples. The training set is used to train the models both during the model selection and evaluation; the validation set is used to monitor the training, perform early stopping and compare the model for the hyperparameter selection; and the testing set is discarded in the first phases and used for the final evaluation after selecting and retraining the final models. All preprocessing operations that require calculations are computed on the training set, then applied to the other subsets.

We are interested in measuring the ability of the models to classify the documents when trained on a small number of labels. Therefore, we define different thresholds of labelled data, namely the number of labels per class and remove the surplus. This operation is only applied on the training set so that all labels are available to evaluate the performance. Next, the following preprocessing steps are performed: removal of email addresses, digits, special characters and email headers. The motivation for removing emails and headers is that these pieces of information do not consist of information that is relevant to the topic and it can create an unintended leakage of information between the training set and the test set if the models learns that a particular person, domain name or organisation is particularly present in one class rather than another. Finally, we remove the stop words and the terms that appear in less than 5 documents and stem the words to a general form using Porter’s algorithm [56]. This results in a total vocabulary of 14,376 tokens. We shuffle the dataset before each epoch during the training phase. Some statistics about the number of words per document are displayed in Table 2.1.

2.2.2 Training Details and Hyperparameters Tuning

For the hyperparameter selection, all models are trained three times for 30 epochs. During the training phase, if the validation performance does not increase for a number of iterations, the learning rate is reduced by a fixed factor. If the performance does not increase for 2 epochs after that, the training stops (early stopping). The final results on the validation set are averaged into one score that is used to select the best parameters for each model. The final models are retrained for 100 epochs with early stopping and learning rate reduction on plateau for the performance evaluation phase.

The best hyperparameters are selected using a random search in a predefined parameter grid. Each model is optimised individually and the search is performed on a specific grid for a defined number of labels. The winning set of hyperparameters is the one that yields the best average performance on the validation set. This means that we end up with one set of hyperparameters per label threshold and per model. The reason for optimising the hyperparameters for a fixed number of labels is that the optimal model structure may vary depending on the number of labels available and it would be unfair to compare the performance of models optimised for a certain number of labels as we increase the threshold. We measure the performance of the models on the validation set so that we remove the bias originating from the model selection. This avoids the vicious effect of selecting a model whose set of hyperparameters happen to give good results on the test set rather than actually measuring the ability of the model to generalise to unseen data.

The hyperparameters specific to the selected models are chosen among a range of heuristically good values. The loss balancing weight, α is chosen to be one of 0.2, 0.5 and 0.8, or set to 1 or 0 for the supervised and unsupervised models. Input units are dropped with a probability of 0.0, 0.2 or 0.5. The encoding dimension is either 128 or 256. For k-sparse and KATE models, 32 units are kept when the encoding dimension is 128 and 64 units when the encoding size is 256. The weight of the $l1$- and $l2$-penalties for the semi-supervised autoencoder are either 0.001, 0.01 or 0 and the weight of the contractive penalty for the CAE is 0.0001, 0.001 or 0.01. Including the supervised and unsupervised models, the total size of the search space is a set of 1,464 hyperparameters. For each model and for each supervised threshold, we randomly draw 50 sets to train the models, thus ending up with 800 models to train. The final selection is the set of hyperparameters that yielded the best validation score averaged over 3 runs. The final sets are displayed in Table 2.2.

The semi-supervised and supervised models trained on less than 100 labels per class use mini-batches of 2 examples and all the other models use a batch size of 32. All models are trained for maximum 30 epochs to determine the best hyperparameters and the best models are retrained for maximum 100 epochs. We decrease the learning rate if the optimisation process reaches a plateau, *i.e.* the performance does

model	labels/class	α	top k	contractive	encoding dim	l1	l2	f1 score (val) mean	std
cae	1	0.8		1e-04	256			17.3%	6e-02
	10	0.8		1e-04	128			53.1%	4e-02
	100	0.5		1e-04	256			81.7%	4e-02
	1000	0.8		1e-02	256			87.7%	5e-02
kate	1	0.8	32		128			23.1%	1e-02
	10	0.8	64		128			67.5%	8e-03
	100	0.2	32		128			85.8%	5e-03
	1000	0.2	64		256			92.4%	4e-04
ksparse	1	0.2	64		256			28.5%	
	10	0.2	32		256			62.6%	2e-03
	100	0.5	32		256			86.0%	6e-03
	1000	0.8	64		256			92.6%	
ssae	1	0.2			256		1e-02	30.9%	6e-04
	10	0.2			256	1e-02	1e-03	60.8%	5e-03
	100	0.5			256	1e-01		85.3%	2e-03
	1000	0.8			256		1e-03	92.1%	8e-04

Tab. 2.2: Best hyperparameters from the grid search (calculated over 30 epochs)

not improve during 3 successive epochs. We stop the training if the performance on the validation set does not increase after 5 epochs. All models are trained by using the Adam optimisation algorithm [36] and an initial learning rate of 0.001. The models are implemented with Tensorflow version 1.13 [1] and Keras [10] and the code is publicly available on GitLab[1]. We are training the models on a computer with an Nvidia GeForce 860M with 4GB memory and 16 GB of RAM. Since this is not a powerful GPU and the memory is limited, we have to keep the number of parameters of the models to reasonable levels.

2.3 Performance Evaluation

We perform three types of experiments to evaluate the models proposed in Section 2.1. First, we analyse the benefits of semi-supervised over fully-supervised models by comparing the predictions made by the models trained in both learning settings. Second, we analyse how the supervised data can help to learn good representations by comparing the encodings generated in a semi-supervised versus unsupervised way. Finally, we study the encodings in depth by delving into the layers' weights, and we try to understand how features, encoding units and predicted classes fit together. In the remaining of the chapter, we will discuss each experiment in detail and how we evaluate the models.

[1] `https://gitlab.com/qmeeus/semi-supervised-text-classifier`

2.3.1 Effect of unlabelled data on classification

Artificial neural networks traditionally require large amounts of data to converge. Therefore, fully-supervised models trained on a small number of data are more prone to overfitting, meaning that they store the training data in their parameters and never learn the underlying distribution of the input. Consequently, an overfitted model is not able to generalise to unseen data. In contrast, we make the hypothesis that semi-supervised models use the unlabelled examples to regularise the learning process by identifying patterns in the unstructured data that help the model to generalise and to avoid overfitting on a small number of labels.

In order to evaluate how the models benefit from the unsupervised data, we compare the predictions of fully-supervised models with our semi-supervised models. We are particularly interested in observing how the scores change as we decrease the number of labels in the dataset. We compare the final models on multiple classification metrics measured on a testing set. During the optimisation, the supervised models only use the available labelled data, whereas the semi-supervised models have also access to the unsupervised dataset. We expect that this gives an advantage to the semi-supervised models and that classification scores would consequently show an improvement over the models that did not have access to the unlabelled data.

The predictions of the different models will be compared over a range of traditional classification metrics. Accuracy is often considered misleading. Therefore, we will primarily focus on the f1-score, which takes into account both the precision ($\frac{\text{TP}}{\text{TP}+\text{FP}}$) and the recall ($\frac{\text{TP}}{\text{TP}+\text{FN}}$) where TP is the true positive rate, FP is the false positive rate, FN is the false negative rate and TN is the true negative rate. The f1-score is given by:

$$\text{f1-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

These metrics are calculated for each class and averaged using macro-average. Additionally, we evaluate the results visually by plotting the confusion matrix, which maps the predictions and true labels in a squared matrix. This will give us an idea of the kind of errors that are made and which classes are often confused with each other. Finally, we also plot the f1-score as a function of the number of labelled examples to get a grip of how the models compare at different threshold of supervised data. We also compare the accuracy of our models to published results on the same dataset, including state-of-the-art methods.

2.3.2 Effect of labelled data on representations

Unsupervised autoencoders learn to reconstruct their input and build dense representations in the process. It is important to make sure that the network did not trivially learn the identity function and end up storing the whole information contained in

the training set in the weights and learns to summarise it instead. In other words, we want the network to learn interesting patterns of the training data that are used to summarise the input data into a dense format, useful for other tasks such as storage, document retrieval, or in our case, classification. We make the assumption that introducing a supervised objective in the optimisation process guides the model to learn encodings (1) that are easier to interpret, and (2) that capture interesting properties of the data distribution.

To validate this hypothesis, we will compare the encodings learned from a fully unsupervised autoencoder with those learned in a semi-supervised setting. Hopefully, we will show that the latter representations perform better when they are used as input to other learning algorithms. In particular, we will use encodings trained with different thresholds of supervised data, including encodings generated by unsupervised autoencoders, as inputs to a support vector classifier and a k-means clustering algorithm. Linear SVMs learn the hyperplane that best separate the different classes in a linear space. If the autoencoders learn encodings that are separable, a linear SVM will be quick to learn how to classify them. However, if this is not the case, the SVMs will yield poor classification performances. The performance of the SVM will be measured with the f1-score. In a second time, we use k-means to learn clusters of data in an unsupervised way. k-means works by grouping together the examples that are similar and assigns dissimilar examples to different clusters. Consequently, if the autoencoders manage to minimise intra-class distance while maximising inter-class distances, the clustering algorithm will be able to create clusters that can relate to classes more easily. We will assess the quality of the clusters with the assignment accuracy. Namely, we use the Hungarian algorithm [38] for assigning each class to a cluster, and we measure the accuracy of the assignment, *i.e.* the average number of time that an example was predicted to belong to a cluster whose assigned class corresponds to its target class.

2.4 Exploration of the Parameters

One aspect of deep learning that is often highlighted as an issue is the difficulty to interpret the model outputs. It is even more the case in representation learning where the output of the network is an abstraction of the reality. In this last experiment, we will try to understand how each step of the prediction process relates to the others. Namely, we explore how input features are mapped to the different units of the encoding layer and how these topics are then combined together into document categories (topic representations). In the first case, we perform a qualitative analysis of the topic composition by looking which token is most relevant in one topic. In a second time, we explore how the inputs from each class are encoded in the latent space and in particular, the similarity of the examples inside and between

classes. For this purpose, we propose to use the cosine similarity, which measures the cosine of the angle between two vectors. It ranges between -1 (when the vectors point in opposite direction) and 1 (when the vectors are in the same direction). Orthogonal vectors have a cosine similarity of zero. This means that this is a measure of orientation and not magnitude. For two vectors A and B, it is defined as follow:

$$sim(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

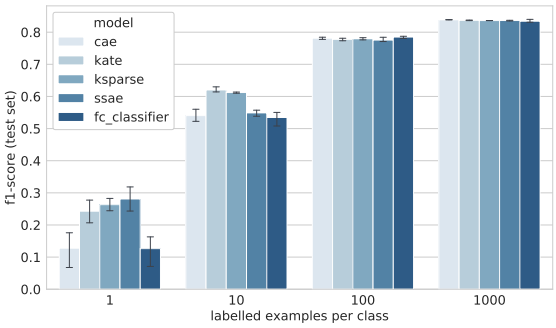
In practice, we calculate the similarity between the encoded representation of pairs of documents from the testing set in order to assess whether documents from the same category share a similar representation, while documents belonging to different classes have a low similarity.

3 Results and Discussion

3.1 Semi-supervised Classification Scores

3.1.1 Comparison with supervised models

Compared to the fully supervised models, the semi-supervised classifiers perform better when the number of labels are small. In Figure 3.1a, we see that the difference between the test scores gets smaller as more labels are included in the training data. When trained on 20 labels, the semi-supervised autoencoder comes out first and when 200 labels are available, KATE yields better scores. When the threshold of 100 labels per class is reached, all semi-supervised models level-off with the supervised classifier. This suggests that the advantage provided by the unsupervised objective becomes negligible when the classifier is trained on more data (1,000 labels per class corresponds to all available labels). The same observation was reached by Chapelle et al. [7] that when we have a lot of labelled data, the unlabelled data does not help nearly as much. Yet, Chapelle et al. [7] displays scores slightly better with the EM algorithm for small amounts of labelled data (about 35% accuracy for 1 la-



(a) Comparison of the f1 scores obtained on the best models on the test set (averaged over 3 runs)

	Average	StDev	Rank
TF-IDF + LR	0.8319	0.0000	8
CNN-rand	0.7693	0.0061	14
CNN-non-static	0.8215	0.0052	9
LSTM	0.7543	0.0172	16
Bi-LSTM	0.7318	0.0185	18
PV-DBOW	0.7436	0.0018	17
PTE	0.7674	0.0029	15
fastText	0.7938	0.0030	13
fastText (bigrams)	0.7967	0.0029	12
SWEM	0.8516	0.0029	2
LEAM	0.8191	0.0024	10
Graph-CNN-C	0.8142	0.0032	11
Text GCN	0.8634	0.0009	1
CAE (semi-sup.)	0.8457	0.0004	3
KATE (semi-sup.)	0.8436	0.0009	4
K-sparse (semi-sup.)	0.8428	0.0003	5
SSAE (semi-sup.)	0.8428	0.0012	6
FC classifier	0.8405	0.0044	7

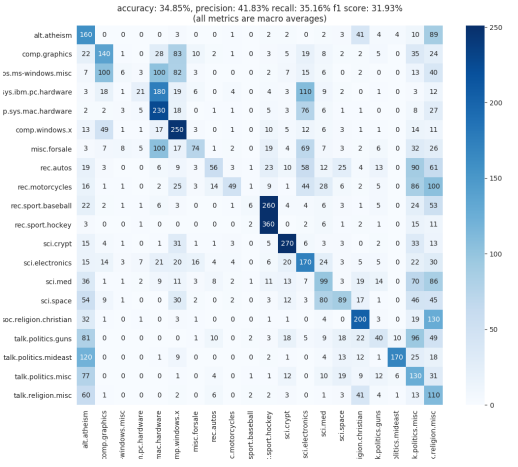
(b) Test accuracy on 20Newsgroup dataset of various models including ours

Fig. 3.1: Classification performances

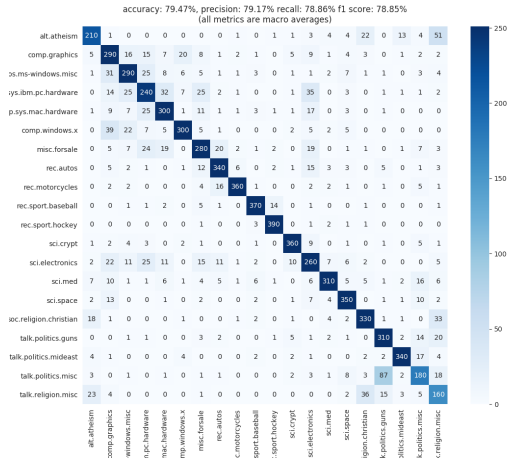
bel per class versus 31% accuracy on average for our best model, with a f1-score of 28%). We reach similar results as Chapelle et al. [7] for superior thresholds. Table 3.1b shows our results for the models trained with all the labels (the last 5 rows) compared to previously published models (taken from Yao et al. [74]). TFIDF-LR refers to a logistic regression model with tf-idf features and performs particularly well on long documents. CNN-rand and CNN-non-static [35] refer to convolutional neural network with word embeddings initialised randomly or with GloVE word vectors respectively. The LSTM models [43] fail to model large sequence, as discussed in previous sections. PV-DBOW [40] is the distributed bag-of-words version of the paragraph vector model and PTE [66] refers to predictive text embedding, all two are presented in Section 1.3.3. fastText and fastText (bigrams) [32] use averaged word embeddings as document embeddings. SWEM [61] performs pooling operations on word embeddings and LEAM [68] embed words and labels in a single vector space. Graph-CNN-C [16] is a model that performs convolutions over word embedding similarity graph. Finally, Text-GCN [74] uses convolutional networks on graphs which contains word nodes and document nodes, allowing them to model word co-occurrences in an efficient way. Although we do not reach the best scores, we range 3rd in this ranking, after Text-GCN and SWEM. Given that our models are much less complex and faster to train, these results are still impressive.

3.1.2 Analysis of predictions

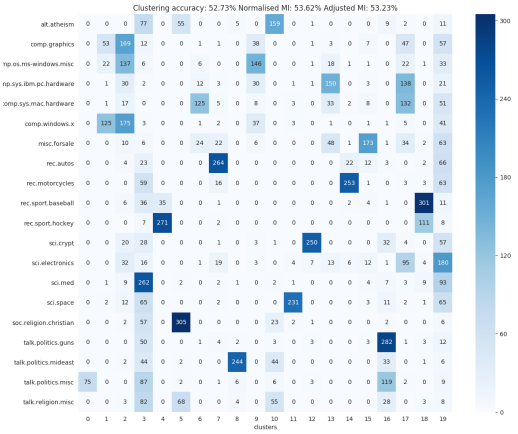
Figures 3.2a and 3.2b show the predictions of the best semi-supervised autoencoders (SSAE and KATE) after being trained on 1 and 100 labels. Correct classifications are visible on the diagonal of the matrix and misclassifications are depicted outside of the diagonal. Naturally, as the amount of available supervised data increases, the model makes fewer mistakes. When one unique example per class is used during training, only 34.9% of the test examples are classified correctly. However, a quick analysis of the frequent errors shows that the classes that are confused with each other often share some overlap. For example, the classes about technical subjects (comp.graphics, comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, comp.windows.x, sci.crypt and sci.electronics) form one group where predictions are often attributed to other classes from the same group. Similarly, the topics about religion or atheism are often confused as well and hockey and baseball all end up classified in the class hockey. Even though the amount of errors decreases considerably when more labelled examples are available from each class, the few remaining misclassifications still occur between the same classes. This is due to the fact that the word families that often appear in one class are also frequently used in similar classes and since we force the model to summarise the information it receives, the differences between classes may become less clear.



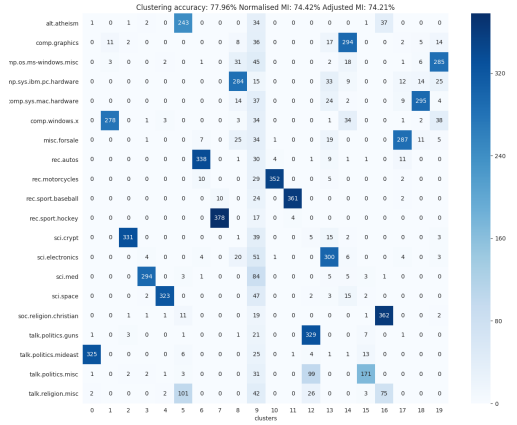
(a) Predictions of the best SSAE trained on 1 label/class



(b) Predictions of the best KATE trained on 100 labels/class



(c) Cluster assignment of best k-sparse trained on 10 labels/class



(d) Cluster assignment of best CAE trained on 1000 labels/class

Fig. 3.2: Confusion matrices and cluster assignments of the different models trained on various thresholds of labelled data

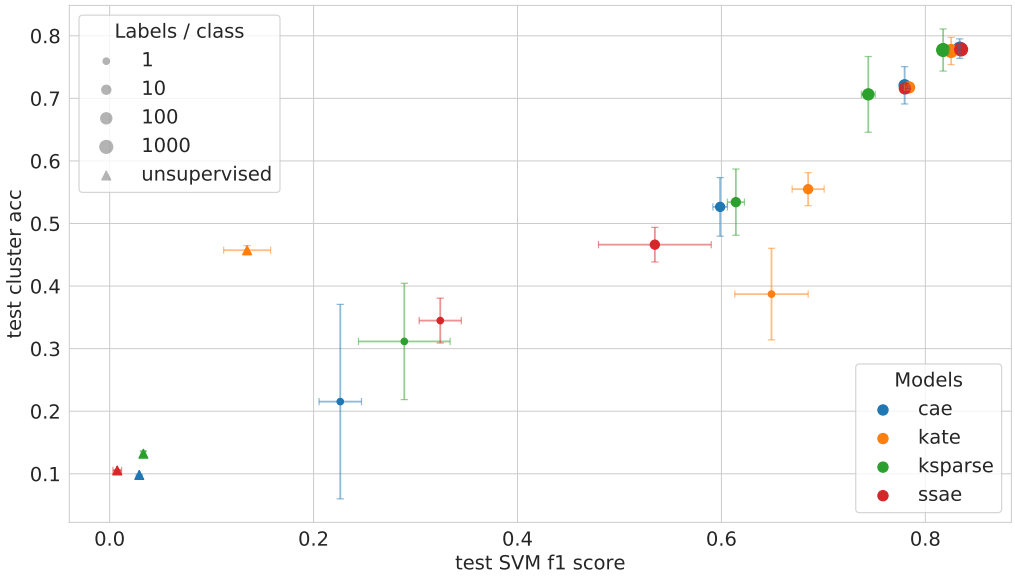


Fig. 3.3: Comparison of the encodings learned by unsupervised and semi-supervised models with various thresholds on classification (SVM) and clustering tasks (k-means): the points represent the mean score obtained by one model trained with a specific number of labelled examples and the error bars show the standard deviation around the mean. The classification power of the encoding is measured by calculating the f1-score of a linear SVM trained with 5-fold CV. The clustering accuracy is the proportion of examples that were assigned to a correct cluster, that is, a cluster assigned to the correct target after running a linear assignment algorithm (Hungarian algorithm).

3.2 Semi-supervised Representation Learning

In this section, we compare the results obtained on the models trained in an unsupervised setting with the autoencoders that incorporate a supervised objective. All four models can be trained in an unsupervised way if we do not connect a classifier layer to the output of the encoder (or if we set the parameter α to zero). To compare the quality of the encodings, we use the learned representations as input to other models and compare their performances. In a first step, we train a support vector classifier to predict the class labels on the test set. Next, we train a k-means algorithm to cluster the input data and compare the resulting clusters with the class labels.

3.2.1 Support Vector Classifier

On Figure 3.3, we observe that none of the unsupervised models manage to create representations that allow the SVM to linearly separate the examples. Although the unsupervised K-competitive autoencoder demonstrates slightly better results, it does not come close to the semi-supervised models, even when they are trained with only one label per class (20 labels in total). This is in line with the results of Chen and Zaki [8], with the exception of their model and the k-sparse autoencoders.

They were able to show that a fully connected classifier trained on their model's representations performed considerably better, achieving a 76% accuracy on the test set. However, we were not able to reproduce these results. Among the semi-supervised autoencoders, KATE clearly stands out as it achieves 64.91% f1-score (averaged over 3 runs) even though it was trained with only 20 labels. As the amount of available labelled examples increases, the difference between the scores gets smaller and disappears completely when all labels are included in the training set. Again, the CAE shows the worst performances, both when trained in an unsupervised and semi-supervised way, whereas the SSAE proves to be more efficient when labels are scarce.

3.2.2 k-means

We observe the cluster assignment accuracy of the unsupervised and semi-supervised models on the y-axis of Figure 3.3. Again, except for the unsupervised K-competitive autoencoder, the unsupervised models do not learn representations that can successfully be clustered by k-means algorithm. Unsupervised KATE proves to be a valid competitor as it shows better results than the representations learned in a semi-supervised setting when trained on limited number of labels. As more labels are included, the models show similar performances, with a slight advantage for KATE until the threshold of 100 labels per class is reached. Figures 3.2c and 3.2d show the cluster assignments of the test examples by the k-sparse autoencoder and CAE after being trained on respectively 10 and all labelled examples. We see that even after being trained with all available labels, one cluster (#9) is ill-defined and some confusion exists between similar classes. Indeed, the class "talk.politics.misc" is either assign to cluster 15 or to cluster 12, together with the class "talk.politics.guns". A similar observation can be made regarding topics about religion or technology. By contrast, model trained on only 200 labels such as shown in Figure 3.2c is only partially able to create encodings that can easily be separated into clusters. Nonetheless, we see that even with a small amount of labels, the model is able to generate encoding that bring similar examples together while distancing examples that belong far apart.

3.3 Exploration of the Network Parameters

In this section, we focus on KATE. In particular, we explore the parameters that the network has learned when trained with the smallest amount of labelled data possible (one labelled example per class or 20 labelled examples). As a first step, we explore how the encodings are created by looking at the word composition in the encoding units. Next, we see how the encoded inputs relate to the classes and if comparable

documents are represented in a similar way. Inversely, we also verify that documents covering different topics have distinct representations.

3.3.1 Topic compositions

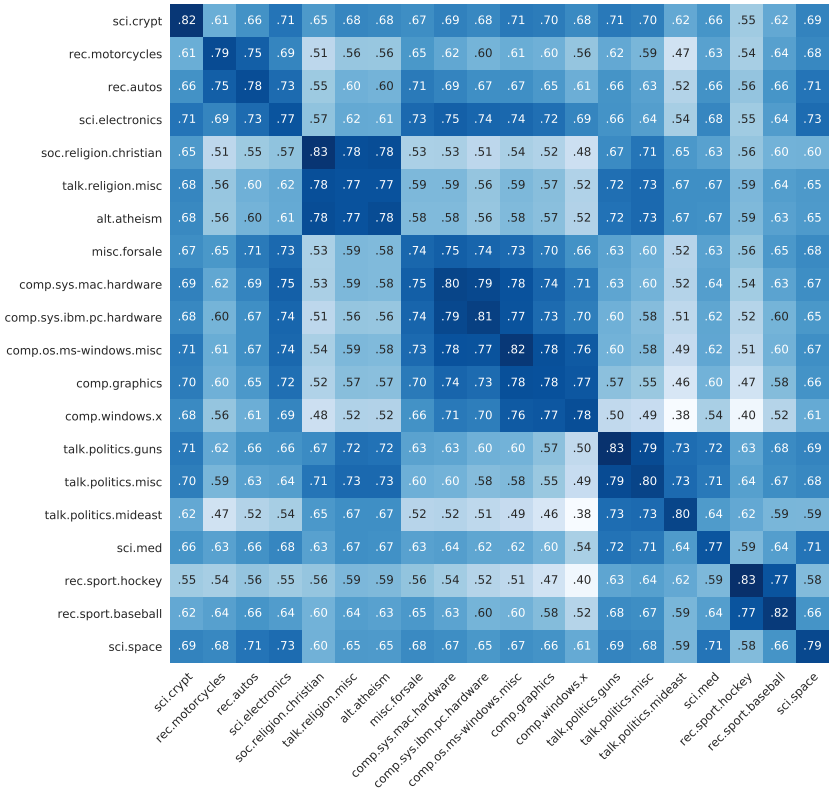
Table 3.1 shows for each target class the topic that is most predictive for this class and the 20 words that are most strongly linked to this topic, *i.e.* the word composition of the topic. Although we directly see that some topics make perfect sense, other topics are composed of words from multiple semantic families that should not end up together. Further, one word can appear in multiple topics and it is common to observe the same word strongly related to multiple topics. For example, the word "armenian" appears in different topics that do not relate to each other (topics 4, 10, 103 and 120). Some topics show a strong similarity, which raises some doubts about the capacity of the model to learn different topics for each of the hidden units.

3.3.2 Representation similarities

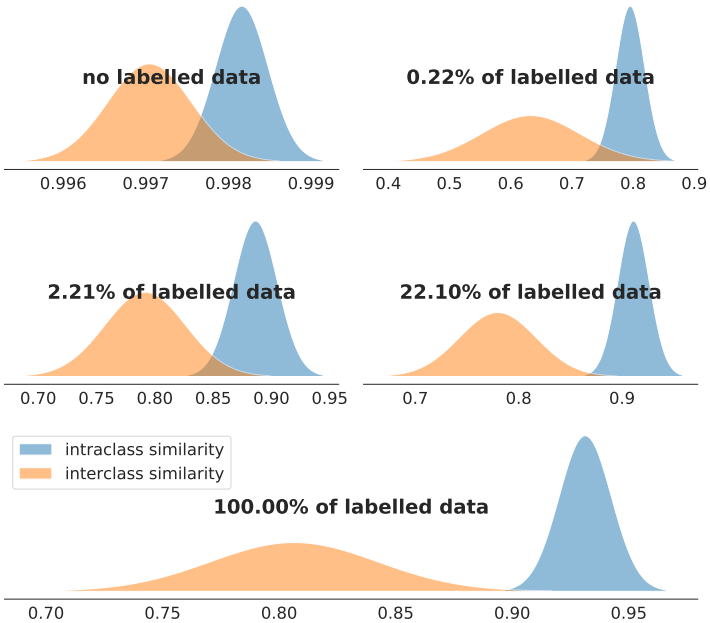
In order to verify that the model produce similar encodings for documents that share similar topics, we have encoded the test examples using KATE's encoder, after being trained on a dataset with 20 labelled examples. Next, we compute the cosine similarity between each pair of encoded documents, and we average the score for each pair of target class. The result is displayed in Figure 3.4a. This time again, we see that the documents that cover analogous topics yield comparable encodings and the average similarity is almost systematically larger inside classes. Figure 3.4 shows the overlap between the intra- and inter-class similarities. The interclass similarity (orange curve) should ideally cover a large portion of the x-axis and have a small height, which denote of large differences in between the representations from different classes. The inter-class similarity should cover a small portion of the x-axis and be centered on a value close to 1. For the unsupervised model, all similarities range between 0.99 and 1, and the two distributions overlap, which means that some examples from different classes are more similar than examples from the same class, and all encodings are very similar to one another. As the number of labels increases, the overall similarity moves away from 1, but the relative standard deviation increases for the inter-class similarity and decreases for the intra-class similarity. When there is no overlap between the two curves, this means that the examples from the same class are consistently more similar than examples from different classes.

Topic	Most likely target class	Top words
0	comp.sys.mac.hardware	lc pc modem simm mb disk port ram mhz screen dresden driver vga serial font meg hardwar mous centri cach
1	rec.sport.baseball	player team hockey season pitch basebal helmet leagu net hitter big tu expo case hand shoei nhl cub rm buy
4	talk.religion.misc	tu de christ armenian atho dresden biblic medic sin stratu valu assum graphic pixmap open religion beck faith religi appressian
10	sci.electronics	de armenian chip signal clipper circuit output design radar orbit light henri ford speed input mi small hp widget mile
22	sci.crypt	crypto clipper escrow sunset sternlight sunris nsa eisa isa bubblejet ax chip deskjet ide vlb metzger pmetzger keyseach drexel bj
23	rec.autos	bike ride tu hp bmw honda mous cool dog nec motorcycl informatik dresden lc wheel bnr rear ground printer mph
24	comp.os.ms-windows.misc	window file printer pc screen simm mous font port mb driver vga directori lc bit meg motif interfac comp disk
47	talk.politics.misc	repli nasa gov problem orbit space drive scienc david bike email window ibm compani launch hole ysu flight fax de
49	sci.space	state gatech orbit support de launch prism tax pitch hitter exist thoma armenian cmu made ac brave season henri helmet
79	comp.graphics	widget tu de dresden xterm motif font pixel pixmap lc graphic pov inf button patch gif sunris beck sunset output
81	alt.atheism	christ faith gld sin lord religion sgi vb vice ico dwyer wpd atho beauchain rushdi agnost risc solntz timmon gregg
84	sci.med	de widget gov diseas tu patient imag stratu nasa water orbit medic medicin graphic book mous nec lc treatment scientif
85	rec.motorcycles	bike zx dresden lafibm behanna uv inf biker beck vb pettefar mcmaster patch quartz maidenhead yfn lafayett stankowitz prism jeep
103	misc.forsale	armenian institut gatech fax soviet screen prism tape ohanu extermin appressian intern sahak dept memori sell melkonian imag sera upenn
113	talk.politics.mideast	mike bike chip clipper ride mail hp colleg team sell season contact dept king driver radio cost blue robert size
114	talk.politics.guns	gun pitch gatech american bill batf brave firearm atf tax atlanta columbia weapon hitter stratu udel cub je staff prism
119	comp.windows.x	tu dresden sunris pixmap question directori app sunset de window font au file pov inf read motif beck andr drexel
120	soc.religion.christian	moral armenian christ rutger fact atho homosexu human children sin religion biblic faith religi book cramer frank polit keith belief
121	comp.sys.ibm.pc.hardware	pc window disk screen drive modem chip mb ms port ram format simm vga printer hardwar memori scsi widget lc
127	rec.sport.hockey	season team pitch philli hitter leagu hockey louisvil dl cub upenn espn sell sunris toyota mike nhl player sunset vb

Tab. 3.1: Top words for the topics with the highest predictive scores with regard to the target class (KATE trained on 20 labelled examples)



(a) Average pairwise cosine similarity of the encoded test examples between and inside the different target classes. The encodings were produced by by KATE trained with 20 (0.22%) labelled examples



(b) Overlap between intra- and interclass similarities of the encodings generated by KATE at different label thresholds

Fig. 3.4: Cosine similarities between examples from the test set

Conclusion

The present work was an attempt to explore semi-supervised learning could be used in the context of long textual documents, especially when few labels are available. This is a problematic that is very relevant for many companies that have large volumes of unstructured data at their disposal but do not have the resources nor the time to label them. We have showed that under certain conditions, semi-supervised autoencoders can be used to improve the classification scores over similar models trained in a supervised way. Indeed, three out of the four semi-supervised models presented performed significantly better than the classifier trained without unlabelled data when few labels were available. However, we found that this advantage tend to fade away as more labels are available, which confirmed the conclusions of other works. We have also seen that labelled data can help an unsupervised model to learn useful representations that can be used for other related tasks such as clustering and classification. The generated representations share some degree of similarity if they are from documents that discuss related topics. Inversely, the representations learned by the unsupervised models were often very hardly distinguishable and we were not able to separate them linearly nor cluster them. Finally, the models were able to identify words relating to the same topics although this finding was not always consistent and some words were mapped to multiple or unexpected topics.

Limitations

The models discussed in this thesis have only been tested on the 20Newsgroups dataset, and we did not explore other collections of documents. Although nothing suggests that the techniques employed here would not work on other documents, this would further validate the results. Yet, the dataset employed here is rather small, when compared to the amount of available resources online and it would be interesting to train the models developed here on more data. Next, although we observed better performances from the semi-supervised models, the gains were not huge and a large number of documents might be misclassified. Therefore, should one need to label a dataset, it might be best to perform this operation iteratively, by

first labelling the documents with the classes predicted at a high probability, and retraining the model with the new labels.
Finally, many possible architectures that could potentially give better results were put aside because of limited computational resources. Provided more powerful computers, one would be able to experiment further, for example by performing a more advanced search through the hyperparameters or using more complex architectures and consequently build models that would likely perform much better than what was presented here.

Future Work

At the time of the writing, major breakthrough are made in many natural language processing tasks, mostly driven by attention mechanisms that give the ability to a neural network to learn which part of an input is most relevant to answer a specific question. In this work, we did not explore this technique and it is probable that in this use case as well, attention would be able to improve the results. For example, the autoencoders considered here do not give more importance *a priori* to some parts of the document distribution. However, if we are able to learn which region of the Zipf curve is most relevant and contains the discriminant features, this would have the effect of identifying stop words and statistically irrelevant regions that do not bring any more information that there already is in the relevant regions. Additionally, word embeddings are known to be very efficient artefacts of NLP models and would probably be helpful as well.

Bibliography

[1] Martin Abadi, Ashish Agarwal, Paul Barham, et al. “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems”. In: *CoRR* (2016). arXiv: 1603.04467 (cit. on p. 46).

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *CoRR* abs/1409.0473 (2014) (cit. on p. 22).

[3] Michael Baldwin. *Test your vocab Survey*. 2013 (cit. on p. 28).

[4] David Barber. *Bayesian Reasoning and Machine Learning*. New York, NY, USA: Cambridge University Press, 2012 (cit. on pp. 9, 10).

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006 (cit. on pp. 5, 9).

[6] Louis Augustin Cauchy. “Méthode générale pour la résolution des systèmes d’équations simultanées”. In: *Comptes Rendus de l’Académie des Sciences* 25 (1847), pp. 536–538 (cit. on p. 13).

[7] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*. 1st. The MIT Press, 2010 (cit. on pp. 5, 6, 51, 52).

[8] Yu Chen and Mohammed J. Zaki. “KATE: K-Competitive Autoencoder for Text”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’17. Halifax, NS, Canada: ACM, 2017, pp. 85–94 (cit. on pp. 26, 41, 54).

[9] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. “On the Properties of Neural Machine Translation: Encoder-Decoder Approaches”. In: *CoRR* (2014). arXiv: 1409.1259 (cit. on p. 22).

[10] François Chollet et al. *Keras*. https://keras.io. 2015 (cit. on p. 46).

[11] Ronan Collobert, Jason Weston, Léon Bottou, et al. “Natural Language Processing (Almost) from Scratch”. In: *J. Mach. Learn. Res.* 12 (Nov. 2011), pp. 2493–2537 (cit. on p. 36).

[12] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. “Very Deep Convolutional Networks for Natural Language Processing”. In: *KI - Künstliche Intelligenz.* 26 (June 2016) (cit. on p. 35).

[13] Matthieu Cristelli, Michael Batty, and Luciano Pietronero. “There is More than a Power Law in Zipf”. In: *Scientific reports* 2 (Nov. 2012), p. 812 (cit. on p. 29).

[14] Andrew M Dai and Quoc V Le. “Semi-supervised Sequence Learning”. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., 2015, pp. 3079–3087 (cit. on p. 28).

[15] Zihang Dai, Zhilin Yang, Yiming Yang, et al. “Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context”. In: *CoRR* (2019). arXiv: 1901.02860 (cit. on p. 22).

[16] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering”. In: *CoRR* (2016). arXiv: 1606.09375 (cit. on pp. 36, 52).

[17] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. “Universal Transformers”. In: *CoRR* (2018). arXiv: 1807.03819 (cit. on p. 22).

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *CoRR* (2018). arXiv: 1810.04805 (cit. on p. 22).

[19] Charles Elkan. “Using the Triangle Inequality to Accelerate k-Means”. In: *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*. 2003, pp. 147–153 (cit. on p. 13).

[20] W. N. Francis and H. Kučera. *A Standard Corpus of Present-Day Edited American English, for use with Digital Computers (Brown)*. 1964, 1971, 1979 (cit. on p. 29).

[21] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O’Reilly Media, 2017 (cit. on pp. 10, 24).

[22] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS’10). Society for Artificial Intelligence and Statistics*. 2010 (cit. on p. 18).

[23] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, pp. 315–323 (cit. on p. 25).

[24] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, et al. “Generative Adversarial Nets”. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’14. Montreal, Canada: MIT Press, 2014, pp. 2672–2680 (cit. on p. 28).

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on pp. 3, 5–9, 14, 15, 17, 18, 20–22, 24, 26, 35).

[26] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Studies in Computational Intelligence. Berlin: Springer, 2012 (cit. on pp. 21, 22).

[27] Çaglar Gülçehre, Sarath Chandar, and Yoshua Bengio. “Memory Augmented Neural Networks with Wormhole Connections”. In: *CoRR* (2017). arXiv: 1701.08718 (cit. on p. 35).

[28] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780 (cit. on p. 22).

[29] K. Hornik, M. Stinchcombe, and H. White. “Multilayer Feedforward Networks Are Universal Approximators”. In: *Neural Netw.* 2.5 (July 1989), pp. 359–366 (cit. on p. 17).

[30] Rie Johnson and Tong Zhang. “Semi-supervised Convolutional Neural Networks for Text Categorization via Region Embedding”. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’15. Montreal, Canada: MIT Press, 2015, pp. 919–927 (cit. on p. 35).

[31] Karen Spärck Jones. “A statistical interpretation of term specificity and its application in retrieval”. In: *Journal of Documentation* 28 (1972), pp. 11–21 (cit. on p. 31).

[32] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. “Bag of Tricks for Efficient Text Classification”. In: *CoRR* (2016). arXiv: 1607.01759 (cit. on pp. 35, 52).

[33] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (3rd Edition draft)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2018 (cit. on pp. 28–30).

[34] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. “A Convolutional Neural Network for Modelling Sentences”. In: *CoRR* (2014). arXiv: 1404.2188 (cit. on p. 36).

[35] Yoon Kim. “Convolutional Neural Networks for Sentence Classification”. In: *CoRR* (2014). arXiv: 1408.5882 (cit. on pp. 36, 52).

[36] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015 (cit. on pp. 15, 46).

[37] Diederik P Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: (2013). arXiv: 1312.6114 (cit. on p. 36).

[38] Harold W. Kuhn. “The Hungarian Method for the assignment problem”. In: *Naval Research Logistics Quarterly* 2 (1955), pp. 83–97 (cit. on p. 48).

[39] Ludwig Lausser, Florian Schmid, Matthias Schmid, and Hans A. Kestler. “Unlabeling Data Can Improve Classification Accuracy”. In: *Pattern Recogn. Lett.* 37 (Feb. 2014), pp. 15–23 (cit. on p. 6).

[40] Quoc V. Le and Tomas Mikolov. “Distributed Representations of Sentences and Documents”. In: *CoRR* (2014). arXiv: 1405.4053 (cit. on pp. 34, 35, 52).

[41] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE*. 1998, pp. 2278–2324 (cit. on p. 20).

[42] Liu Liu, Kaile Liu, Zhenghai Cong, et al. “Long Length Document Classification by Local Convolutional Feature Aggregation”. In: *Algorithms* 11.8 (2018) (cit. on pp. 22, 33).

[43] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. “Recurrent Neural Network for Text Classification with Multi-Task Learning”. In: *CoRR* (2016). arXiv: 1605.05101 (cit. on p. 52).

[44] S. Lloyd. “Least squares quantization in PCM”. In: *IEEE Transactions on Information Theory* 28.2 (Mar. 1982), pp. 129–137 (cit. on p. 13).

[45] Alireza Makhzani and Brendan J. Frey. “k-Sparse Autoencoders”. In: *CoRR* (2013). arXiv: 1312.5663 (cit. on pp. 25, 26, 41).

[46] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. “Adversarial Autoencoders”. In: *International Conference on Learning Representations*. 2016 (cit. on pp. 28, 36).

[47] Gary Marcus. “Deep Learning: A Critical Appraisal”. In: *CoRR* (2018). arXiv: 1801.00631 (cit. on p. 1).

[48] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient Estimation of Word Representations in Vector Space”. In: *CoRR* (2013). arXiv: 1301.3781 (cit. on p. 32).

[49] Thomas M. Mitchell. *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997 (cit. on p. 5).

[50] Tom Mitchell. *20 Newsgroups Dataset*. 1997 (cit. on pp. 2, 43).

[51] Marie-Francine Moens. *Natural Language Processing (Slides)*. KU Leuven. 2018 (cit. on pp. 22, 29).

[52] Andrew Ng. *Deep Learning and Unsupervised Feature Learning (Lecture Notes)*. Stanford University. 2011 (cit. on p. 25).

[53] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “Understanding the exploding gradient problem”. In: *CoRR* (2012). arXiv: 1211.5063 (cit. on p. 22).

[54] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 7, 10).

[55] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543 (cit. on p. 33).

[56] Martin F Porter. “An algorithm for suffix stripping”. In: *Program* 14.3 (1980), pp. 130–137 (cit. on p. 44).

[57] David M. W. Powers. “Applications and Explanations of Zipf’s Law”. In: *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*. NeMLaP3/CoNLL ’98. Sydney, Australia: Association for Computational Linguistics, 1998, pp. 151–160 (cit. on p. 29).

[58] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. “Contracting auto-encoders: Explicit invariance during feature extraction”. In: *Proceedings of the Twenty-eight International Conference on Machine Learning (ICML’11*. 2011 (cit. on pp. 25, 26, 41, 42).

[59] Frank Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain”. In: *Psychological Review* (1958), pp. 65–386 (cit. on p. 16).

[60] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (Oct. 1986) (cit. on p. 16).

[61] Dinghan Shen, Guoyin Wang, Wenlin Wang, et al. “Baseline Needs More Love: On Simple Word-Embedding-Based Models and Associated Pooling Mechanisms”. In: *CoRR* (2018). arXiv: 1805.09843 (cit. on pp. 35, 52).

[62] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. “Semi-supervised Recursive Autoencoders for Predicting Sentiment Distributions”. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. EMNLP ’11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 151–161 (cit. on pp. 27, 42).

[63] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958 (cit. on p. 19).

[64] Erik Sudderth. *Probabilistic Graphical Models 24: Conditional Random Fields, MAP Estimation & Max-Product BP (Slides)*. Brown University. 2013 (cit. on p. 11).

[65] Richard S. Sutton. *The Bitter Lesson*. 2019. URL: http://www.incompleteideas.net/IncIdeas/BitterLesson.html (cit. on p. 30).

[66] Jian Tang, Meng Qu, and Qiaozhu Mei. “PTE: Predictive Text Embedding through Large-scale Heterogeneous Text Networks”. In: *CoRR* (2015). arXiv: 1508.00200 (cit. on p. 52).

[67] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. “Attention Is All You Need”. In: *CoRR* (2017). arXiv: 1706.03762 (cit. on pp. 22, 36).

[68] Guoyin Wang, Chunyuan Li, Wenlin Wang, et al. “Joint Embedding of Words and Labels for Text Classification”. In: *CoRR* (2018). arXiv: 1805.04174 (cit. on pp. 36, 52).

[69] C. Wiraatmaja, K. Gunadi, and I. N. Sandjaja. “The Application of Deep Convolutional Denoising Autoencoder for Optical Character Recognition Preprocessing”. In: *2017 International Conference on Soft Computing, Intelligent System and Information Technology (ICSIIT)*. Sept. 2017, pp. 72–77 (cit. on p. 24).

[70] Jiaming Xu, Bo Xu, Peng Wang, et al. “Self-Taught Convolutional Neural Networks for Short Text Clustering”. In: *CoRR* (2017). arXiv: 1701.00185 (cit. on p. 36).

[71] Kelvin Xu, Jimmy Ba, Ryan Kiros, et al. “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”. In: *CoRR* (2015). arXiv: 1502.03044 (cit. on p. 22).

[72] Zhilin Yang, Zihang Dai, Yiming Yang, et al. “XLNet: Generalized Autoregressive Pretraining for Language Understanding”. In: *CoRR* (2019). arXiv: 1906.08237 (cit. on p. 22).

[73] Zichao Yang, Diyi Yang, Chris Dyer, et al. “Hierarchical Attention Networks for Document Classification”. In: *HLT-NAACL*. 2016 (cit. on p. 36).

[74] Liang Yao, Chengsheng Mao, and Yuan Luo. “Graph Convolutional Networks for Text Classification”. In: *CoRR* (2018). arXiv: 1809.05679 (cit. on pp. 34, 52).

[75] David Yarowsky. “Unsupervised Word Sense Disambiguation Rivaling Supervised Methods”. In: *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*. ACL ’95. Cambridge, Massachusetts: Association for Computational Linguistics, 1995, pp. 189–196 (cit. on p. 6).

[76] Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. “ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs”. In: *CoRR* (2015). arXiv: 1512.05193 (cit. on p. 36).

[77] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. “Character-level Convolutional Networks for Text Classification”. In: *CoRR* (2015). arXiv: 1509.01626 (cit. on p. 36).

[78] F. Zhuang, D. Luo, X. Jin, et al. “Representation Learning via Semi-Supervised Autoencoder for Multi-task Learning”. In: *2015 IEEE International Conference on Data Mining*. Nov. 2015, pp. 1141–1146 (cit. on p. 27).

List of Figures

1.1 Model capacity and fitness. Source: Pedregosa et al. [54] 7
1.2 A SVM learns the hyperplane that maximises the margin between two sets of linearly separable examples. Source: Pedregosa et al. [54] 10
1.3 Probabilistic Graphical Models. Source: Sudderth [64] 11
1.4 The two principal components of a dataset with two dimensions 12
1.5 An illustration of how the gradient descent algorithm finds the minimum of a function by taking steps in the direction of the descent. Source: Goodfellow et al. [25] 14
1.6 The perceptron Source: http://analyticsvidhya.com 16
1.7 Common activation functions 17
1.8 Convolutional Neural Networks use sparse connectivity and are parametrised by a fixed size kernel. Source: Goodfellow et al. [25] 20
1.9 Computational Graph of Recurrent Neural Networks. Source: Graves [26] 21
1.10 Long Short-Term Memory. Source: Goodfellow et al. [25] 22
1.11 Illustration of attention mechanisms 22
1.12 Two layers undercomplete autoencoder with one hidden layer. Source: http://cican17.com 24
1.13 The k-sparse autoencoder selects the top and bottom activities and resets the others to zero. 26
1.14 The k-competitive autoencoder resets low-activity units and redistributes the lost energy among the “winners”. 26
1.15 Competition among neurons. Source: Chen and Zaki [8] 26
1.16 A semi-supervised autoencoder combines an unsupervised autoencoder and a supervised classifier [62] 27
1.17 English native speaker vocabulary by age and survey percentile (486,496 responses) Source: Baldwin [3] 28
1.18 Zipf plot for Brown corpus [20] tokens 29
1.19 Word2Vec models Source: https://skymind.ai/wiki/word2vec 32
1.20 Distributed representations of sentences and documents [40] 34

3.1 Classification performances 51
3.2 Confusion matrices and cluster assignments of the different models trained on various thresholds of labelled data 53

3.3 Comparison of the encodings learned by unsupervised and semi-supervised models with various thresholds on classification (SVM) and clustering tasks (k-means): the points represent the mean score obtained by one model trained with a specific number of labelled examples and the error bars show the standard deviation around the mean. The classification power of the encoding is measured by calculating the f1-score of a linear SVM trained with 5-fold CV. The clustering accuracy is the proportion of examples that were assigned to a correct cluster, that is, a cluster assigned to the correct target after running a linear assignment algorithm (Hungarian algorithm). 54
3.4 Cosine similarities between examples from the test set 58

List of Tables

2.1 Number of documents and words per document statistics in the different subsets before and after preprocessing 44
2.2 Best hyperparameters from the grid search (calculated over 30 epochs) 46

3.1 Top words for the topics with the highest predictive scores with regard to the target class (KATE trained on 20 labelled examples) 57

Master's thesis filing card

Student: Quentin Meeus

Title: Semi-supervised autoencoders for long text classification

UDC: 681.3*I20

Abstract:
One major difficulty of text classification learning algorithms is the need of large labelled datasets to be able to reach good performances. In this work, we explore how supervised classifiers trained in conjunction with unsupervised autoencoders can create robust models that do not require much labelled examples to perform well. In particular, we focus on the processing of large text documents, on which traditional models such as LSTMs perform poorly. We propose four semi-supervised autoencoders that impose different constraints on the encodings and compare their performance on the 20Newsgroups dataset. We evaluate the predictions of the classifier as well as the quality of the representations learned by the autoencoder. We show that the joint optimisation of a supervised and unsupervised objective improves both the classification performances and the quality of the representations. Indeed, the semi-supervised autoencoders perform better than a supervised classifier with a matching architecture trained with the same amount of labelled data. Although the advantages provided by the unsupervised objective fade away as more labelled examples are available, the results of our models compare to state-of-the-art classifiers performing the same task. Further, we show that the representations produced by the autoencoders can be used for classification by a Linear SVM or for other related learning tasks such as clustering. Finally, we demonstrate that even when trained on only one label per class, the autoencoder manages to identify words that belong to the same topic, although this finding must be put into perspective, as some topics were ill-defined and some words appeared in multiple unrelated topics.

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Engineering and Computer Science

Thesis supervisor: Prof. Marie-Francine Moens

Assessor: Prof. Jesse Davis, Elias Moons

Mentor: Sumam Francis