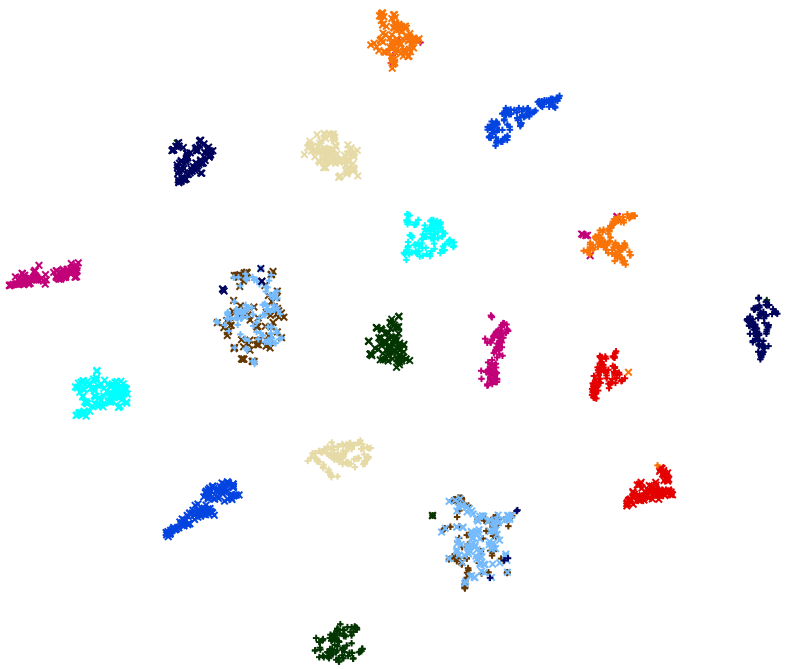
CHALMERS | UNIVERSITY OF GOTHENBURG



Representation learning for natural language

OLOF MOGREN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Representation learning for natural language

OLOF MOGREN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG

Gothenburg, Sweden 2018

Representation learning for natural language
OLOF MOGREN

ISBN 978-91-7597-675-4

Doktorsavhandlingar vid Chalmers Tekniska Högskola
Ny serie nr.
ISSN 0346-718X
Technical Report No. 4356
Department of Computer Science and Engineering

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
Pairs of morphological word-forms embedded using a character-based neural sequence model trained to perform morphological relational reasoning. Embeddings were projected from 50 dimensions using t-SNE.

Chalmers Reproservice
Gothenburg, Sweden 2018

To Stina, Tove, and Sofie.

Abstract

Artificial neural networks have obtained astonishing results in a diverse number of tasks. One of the reasons for the success is their ability to learn the whole task at once (end-to-end learning), including the representations for data. This thesis will explore representations for natural language through the study of a number of tasks ranging from automatic multi-document summarization to named entity recognition and the transformation of words into morphological forms specified by analogies. The summarization work explores the feasibility of using neural representations for words, and how to best combine these with traditional approaches to extractive multi-document summarization. The resulting solution obtains state-of-the-art results on standard benchmark datasets. The rest of the thesis studies models trained end-to-end for the specific tasks, and focus not only on the end result of the trained models, but also on the internal representations of data that they learn. Firstly, a character-based recurrent neural network model is presented that recognize medical terms in health record data. Secondly a novel recurrent neural model that transforms a query word into the morphological form demonstrated by another word. The model is trained and evaluated using word analogies and takes as input the raw character-sequence of the words with no explicit features needed. As the model learns to transform words, it learns internal representations that disentangles morphological relations that were never specified explicitly. Thirdly, a regularizer is presented that improves disentanglement in the learned representations by penalizing the correlation between activations in a layer.

Acknowledgements

My sincerest thank you goes out to my supervisor Richard Johansson who helped me and guided me through my studies with great patience and plenty of encouragement. My co-supervisor Devdatt Dubhashi, thank you for interesting discussions and inspiring ideas. Thank you Peter Damaschke for being a great teacher, and the one who initially introduced me to machine learning.

A deep thank you also goes to my office-mate and champion co-author Mikael Kågebäck; this thesis would not be much without your contributions, discussions, and feedback. Thank you Fredrik Johansson, for inspiring chats, a creative atmosphere, and a lot of fun. Many fond memories was created with the two of you during coffee breaks and ping-pong matches. Also, thank you Nina Tahmasebi for teaching me lots of important things, Jonatan Bengtsson for your nice work on sentence similarity measures, and Prasanth Kolachina for interesting discussions, and for breaking the bubble once in a while. Christos Dimitrokakis, you introduced me to the world of trail running, and pushed me to the marathon distance. Let's do that again some time. Alexander Schliep, I sincerely appreciate your enthusiasm and collaboration. Svetoslav Marinov at Seal and Staffan Truvé at Recorded Future, you helped keep the perspective on things, and allowed for some bits of the real world to enter our world and meet with our algorithms.

This thesis was made possible by funding from the Swedish Foundation for Strategic Research (SSF, grant IIS11-0089), and the Swedish Research Council (2012-2016; dnr 2012-5738).

During my time as a PhD student, I have made many new friends. Thank you Aristide, Luis, Azam, Chien-Chung, Ankani, Hannes, Gabriele and Vinay, for making this a fun place to be! Thank you students and collaborators: Simon, Christian, Johan, William, Simon, Sean, Jacob, Hampus, Johannes, Yanlin, Albin, and others. I had the pleasure to be invited by Tapani Raiko to the Deep Learning and Bayesian Modeling Research Group at Aalto University. This was a great inspiration for me.

Thank you Ulrika and Lars, my parents, you always encouraged me and believed in me, and Lisa, Karl, and Ingrid for supporting, inspiring and challenging me from day one. I would never have made it here without you. Finally, thank you Sofie, my wonderful wife, and our strong, curious, and intelligent daughters Tove and Stina. You all teach me so much new every day, you show me the world in different ways, and you keep reminding me of what's important.

List of publications

This thesis is based on the following manuscripts.

Paper I M. Kågebäck, O. Mogren, et al. (2014). "Extractive summarization using continuous vector space models". *Second Workshop of Continuous Vector Space Models and their Compositionality*

Paper II O. Mogren, M. Kågebäck, and D. Dubhashi (2015). "Extractive summarization by aggregating multiple similarities". *Proceedings of Recent Advances in Natural Language Processing*, pp. 451–457

Paper III S. Almgren, S. Pavlov, and O. Mogren (2016). "Named Entity Recognition in Swedish Health Records with Character-Based Deep Bidirectional LSTMs". *Proceedings of the Fifth Workshop on Building and Evaluating Resources for Biomedical Text Mining*

Paper IV O. Mogren and R. Johansson (2017). "Character-based recurrent neural networks for morphological relational reasoning". *Submitted draft. Early version published at the EMNLP Workshop on Subword and Character-level Models in NLP*

Paper V M. Kågebäck and O. Mogren (2017). "Disentangled activations in deep networks". *Submitted draft. Early version presented at the NIPS Workshop on Learning Disentangled Features*

The following manuscripts have been published, but are not included in this work.

Paper VI O. Mogren (2016). "C-RNN-GAN: Continuous recurrent neural networks with adversarial training". *NIPS Workshop on Constructive Machine Learning (CML)*

Paper VII J. Hagstedt P Suorra and O. Mogren (2016). "Assisting discussion forum users using deep recurrent neural networks". *ACL Workshop on representation learning for NLP, RepL4NLP*

Paper VIII A. S. Muhammad, P. Damaschke, and O. Mogren (2016). "Summarizing online user reviews using bicliques". *Proceedings of The 42nd International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM, LNCS*

Paper IX N. Tahmasebi et al. (2015). Visions and open challenges for a knowledge-based culturomics. *International Journal on Digital Libraries* **15**.2-4, 169–187

Paper X P. Damaschke and O. Mogren (2014). Editing simple graphs. *Journal of Graph Algorithms and Applications, Special Issue of selected papers from WALCOM 2014* **18**.4, 557–576. DOI: 10.7155/jgaa.00337

Contribution summary

Paper I	I implemented the submodular optimization algorithm for sentence selection and performed the experimental evaluation. I also wrote parts of the manuscript and made some illustrations.
Paper II	I designed the study, performed the experiments, and wrote the manuscript.
Paper III	I designed the study, supervised the experimental work, and wrote the manuscript.
Paper IV	I designed the study, performed the experiments, and wrote the majority of the manuscript.
Paper V	I contributed to the design of the study, performed parts of the experiments, and wrote parts of the manuscript.

Contents

Part I
Extended summary

Chapter 1

Introduction

The advances in the field of artificial intelligence (AI) have been astonishing in recent years, and algorithms that solve new tasks with unprecedented accuracies are being proposed every other week. Until recently, self-driving cars, image classification on a super-human level, and digital assistants which can take commands in natural language were only fantasies. Yet they are now about to become ubiquitous technologies. People have worked on making machines intelligent for a long time, without having a real definition of intelligence. For many of us, however, language is central. It is possibly the foremost thing setting humans apart from other animals, and it is our tool to understand each other. The Turing test defines machine intelligence as being able to communicate with humans in their own (written) language in such a way that the human part cannot tell the machine from other humans (Turing 1950). This was published long before machines were anywhere near being able to succeed at this, a milestone that may soon be realized, if it hasn't already been done.

Most of the results mentioned above have been made possible with the recent progress in machine learning. While AI is a wide field, comprising many different research directions (and lacking a formal definition), machine learning is taking on an increasingly large role, in part replacing the rule-based systems of a few years back. There are several reasons for this development: computers are faster, a development of more clever algorithms has taken place, and we have access to much more data. Artificial neural networks (ANNs) take advantage of all this progress and learn from data using gradient based optimization, and this has resulted in a number of successful solutions for different tasks. Deep ANNs are composed of several stacked layers, each gradually transforming their input vectors into representations which are more useful for solving the end task. This has recently gone under the name of deep learning.

Prior to this development, other machine learning techniques such as support vector machines (Boser, Guyon, and Vapnik 1992), logistic regression (Cox 1958), and decision trees (Quinlan 1986) were more successful, requiring less data and computation to work. However, these techniques do not scale as well with large quantities of data (and computational power), and often require manual or semi-manual preprocessing and feature engineering. Large engineering efforts have been put into systems that compute features

for different data modalities to be used with machine learning techniques. Examples of this are SIFT features for images (Lowe 1999), and features for natural language processing (NLP), such as capitalization, the presence of numerics, contextual features, and the length of words (Ratinov and Roth 2009; Manning and Schütze 1999). Krizhevsky, Sutskever, and G. E. Hinton (2012) revolutionized the Imagenet image classification competition by training a deep convolutional neural network (CNN) with the raw images as its only input, learning its own internal representations (see section 5.6). The work included developing highly optimized code, leveraging the capabilities of paralell computing in graphics processing units (GPUs). Since then, a number of iterations have further improved this result, but deep learning is still the reigning technique.

The transition from engineered to learned features is an example of a general trend. Machine learning techniques previously needed the limitation in scope, and the help from human input to solve parts of the problem. With deep learning, a larger part of the problem is being solved by the machine, allowing for quicker turnover in experimental work, and algorithms that adapt better to new applications. The learning of representations is an example, leading to a revolution in performance in image recognition and other tasks, enabling both autonomous driving and super-human performance in games such as Atari (Mnih et al. 2013) and Go (Silver et al. 2016).

This thesis will discuss representation learning and its applications in NLP. Language is an important part of intelligence, but it poses some challenges for continuous gradient based learning algorithms. Discreteness, sparsity, high dimensionality, and sequences having variable length can pose challenges to machine learning learning algorithms, in a way different from other data modalities such as images. Many traditional approaches to problems in NLP are composed of several processing steps, where machine learning may be used in some of the steps in combination with other techniques. Paper I and Paper II are examples of this kind of approach, that leverages neural word embeddings (see section 4.1) and uses a novel approach to combine multiple ways of encoding semantic similarity for multi-document summarization.

As algorithms and models have improved, together with data availability and computing power, end-to-end learning algorithms that attack the whole problem at once, have become more common. There are many examples in NLP where such approaches have proven fruitful. In Paper III, we present a character-based recurrent neural network (RNN) approach to recognize and classify mentions of medical entity terms in Swedish health records, and in Paper IV, we propose a neural network for morphological relational reasoning where analogies show the model what morphological transformations to apply to a query word. Both of these are trained end-to-end, and learns all necessary internal representations. Paper IV includes an analysis of the internal representations and it can be noticed that they disentangle the known underlying structure, without having an explicit signal enforcing this. In Paper V, we present a regularization scheme for deep neural networks that penalizes correlation between activations in a vector representation. We demonstrate how this can be applied in a number of experiments, and that it learns to minimize the number of used activations to encode the necessary factors of variation.

Thesis outline. Chapter 2 will give some background on natural language processing and representations for language. Chapter 3 introduces artificial neural networks and

representation learning, and chapter 4 demonstrates how this can be applied for language. Specifically, the work on multi-document summarization in Paper I and Paper II will be introduced. These papers make use of learned representations and study the effects of such representations on summarization. Chapter 5 discusses how to attack problems using end-to-end learning, demonstrated by Paper III and Paper IV, some of the properties one can expect from representations learned in deep learning models, and how to train them. Paper V is introduced, with a regularization technique that helps deep networks improve disentanglement in their data representations. Chapter 6 contains some concluding remarks.

Chapter 2

Natural language processing

Natural language processing (NLP) is the study of tasks related to computational processing of natural language. One of the most fundamental of these tasks is language modeling, estimating the probability distribution over language sequences. This means that given a sequence of words $w^{(0)}, w^{(1)}, ..., w^{(N)}$, the problem is to estimate the probability:

$$p(w^{(0)}, w^{(1)}, ..., w^{(N)}).$$

Another important task is automatic document summarization: the task of producing summaries that are representative yet at the same time not redundant, given an input document, or a set of documents. This will be further explored in chapter 4. Other important tasks include sentiment analysis, machine translation, part-of-speech (POS) tagging, named entity recognition (NER), and word sense disambiguation. Natural language is ambiguous and messy, and can express an unlimited number of different meanings (Manning and Schütze 1999). For instance, one message can be expressed in many different ways, and one expression can sometimes mean different things, depending on context or author. These properties can pose serious challenges to computational approaches. To be able to succeed, an algorithm needs to have some important properties, such as to be able to adapt to variations, and at the same time be robust to noise and errors such as misspellings.

Most statistical approaches to NLP problems rely on having informative representations for the data, encoding the information needed for a given task. For solutions to NLP problems that use feature engineering, special care must be taken in the feature design to make sure that the features encode everything that is informative, predictive, and useful for the end task. Traditional approaches to NLP often apply a pipeline of algorithms, each performing some step (such as word-sense disambiguation or POS tagging), that is assumed to produce a useful input to a downstream task in the pipeline, and eventually for the component solving the end task. The fact that different tasks may require information about different aspects of the data is an important part of the motivation for using learned representations for language, as such representations can be adapted to work better for a specific task using the optimization of a loss function. This will be introduced in chapter 3 and chapter 4.

In many NLP tasks, a simple n-gram baseline often performs very well. An n-gram is a subsequence of n tokens. For instance, an n-gram language model is created by collecting n-gram statistics from a corpus, and using this to model the probability of a sequence of words $w^{(0)}, w^{(1)}, ..., w^{(N)}$. The distribution is factorized, and a history of only n words is used:

$$p(w^{(0)}, w^{(1)}, ..., w^{(N)}) \approx \prod_{i=0}^{N} p(w^{(i)} | w^{(i-(n-1))}, w^{(i-(n-2))}, ..., w^{(i-1)}).$$

Many other tasks, including text categorization (Cavnar, Trenkle, et al. 1994) and document summarization (H. Lin and Bilmes 2011) have also been successfully approached using n-gram based solutions.

2.1 Count based representations for language

Representations for sentences, paragraphs, documents, and collections. Language units such as documents, paragraphs, sentences, and context windows can be represented by their bag-of-words vectors, which have the same size as the vocabulary, and with each component corresponding to the count of a specific n-gram in the given input sequence. In this thesis, this will be referred to as "bag-of-words" even if "bag-of-ngrams" or "bag-of-terms" would have been more accurate.

Using the cosine similarity to compare bag-of-words vector representations gives a normalized[1] measure of word overlaps or co-occurrence; a convenient and computationally inexpensive way to compare language units. A bag-of-words vector using a unigram vocabulary ($n = 1$) discards all information about the order of the words. Using n-grams in the representation allows for more information about the order as n is increased. In practical applications, however, $n = 2$ or $n = 3$ is often used, because larger values of n give vocabularies that can be impractically large. A tweak of n-gram models is to weigh the n-grams by their tf-idf scores. tf-idf, the "term frequency-inverse document frequency" is a measure of how specific a term is to the current document. E.g. function words, such as "and", "it", "but", etc, are not specific to any document, as they are common in most documents. Content words on the other hand, tend to be more specific and obtain higher tf-idf scores; in many applications these are also more informative. The count-based vector representations computed for language units can be used for similarity computations and/or as input to learned systems such as classifiers. Representations for sentences based on bag-of-words have been used to compute similarity scores to determine representativeness and redundancy in automatic document summarization (Mihalcea and Tarau 2004; Radev et al. 2004; H. Lin and Bilmes 2011; Kulesza and Taskar 2012). This will be further explored in chapter 4.

Representations for words. Word level represntations can be computed based on bag-of-words statistics. The statistics for a word w is collected from a corpus by counting words in the context of w every time it occurs in a corpus. A context is typically defined as being the k preceding and k succeeding tokens to w, for some feasible k. This is an

[1] As bag-of-words vectors have no negative dimensions, this gives a score in $[0, 1]$.

example of *distributional representations*, as they naturally encode distributional semantics, adhering to the *distributional hypothesis*, that words with similar context have similar meaning (Firth 1957). The resulting representations are however very high-dimensional and sparse, and contains no information about the linguistic structure or ordering of the words. Schütze (1993) introduced the term "word space", referring to the vector space model resulting from the distributional vectors. The proposed representations used singular value decomposition to reduce the dimensions of co-occurrence based vectors which resulted in representations that performed well.

Chapter 3

Representation learning

In this chapter, we discuss strategies and algorithms involving ANNs for learning representations of data. Deep ANNs can learn non-linear transformations and to compute hierarchical abstractions of the data. This helps to disentangle the underlying factors. The joint training of all parameters in a model helps to learn representations of different levels of abstractions that are useful for solving a task. Chapter 4 will describe how representations learned using neural networks can be used in a pipeline in a traditional solution to a classical problem in NLP: automatic multi-document summarization. Chapter 5 will describe how to use ANNs to model complete problems end-to-end, and meanwhile learn the needed representations internally.

3.1 Artificial neural networks

Artificial neural networks (ANNs) are advanced function approximators and flexible machine learning models built on a few simple primitives. The combination of these primitives is straight-forward, and one can easily build very powerful networks that can model and approximate complicated, non-linear functions. One central property for all primitives used is that they are continuously differentiable. This means that one can compute the derivative of the (continuously differentiable) loss function, with respect to every parameter in the network, and then update the parameters using gradient descent (see section 3.3). The basic building block is the artificial neuron (sometimes also referred to as a unit), and in most cases such units are arranged into layers with several units working in parallel. There are weak similarities between artificial neurons and biological neurons. However, in this thesis, we only consider the units as parts of a machine learning model. Each unit is able to do a simple computation: to compute a dot product between its input vector $\mathbf{x}$ and a weight vector $\mathbf{w}$, add a bias term b, and then apply a non-linear activation function f, effectively "squashing" the pre-activation z:

$$z = \mathbf{w} \cdot \mathbf{x} + b,$$
$$h = f(z).$$

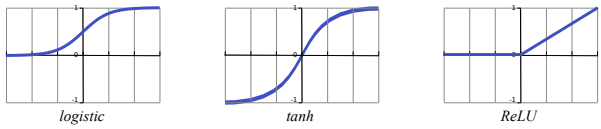


Figure 3.1.1: *Common activation functions used in artificial neural networks: logistic function (σ, left), hyperbolic tangent, (tanh, center), and rectified linear, (ReLU, right).*

Common choices for the activation function f are sigmoids (generally, the logistic function (σ): $f(z) = \frac{1}{1-e^{-z}}$ or the hyperbolic tangent (tanh): $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$), or the rectified linear function (relu): $f(z) = max(0, z)$ (see Figure 3.2.1). The logistic function and tanh are both sigmoidal, and continuously differentiable. The main difference is their range: the logistic function outputs a value in $(0, 1)$, and tanh outputs a value in $(-1, 1)$. Both the logistic and tanh activations saturate when their inputs approach positive or negative infinity, which means that their derivatives tend to 0. The relu does not saturate towards positive infinity, which is its main strength. On the other hand, once a relu output is zero, so is its derivative; this is known as the dead relu problem. When arranged in a layer, the post-activation is instead a vector $\mathbf{h}$, while the weights of the whole layer is represented by a matrix W and the bias is a vector $\mathbf{b}$:

$$\mathbf{z} = W\mathbf{x} + \mathbf{b},$$
$$\mathbf{h} = f(\mathbf{z}).$$

The activation functions are applied element-wise on the pre-activation vector $\mathbf{z}$.

There are examples of more specialized components used in ANNs, (e.g. convolutional layers, mentioned in section 5.6) but the ones mentioned here are the general building blocks. Designing a model can be done by combining layers in a feed-forward sequence, choosing the number of units in each layer (the dimensionality of the layer), choosing the activation for the output layer, and a suitable loss function to optimize (see section 3.3). Choosing the right layout (dimensions, depth, activation functions, etc) requires a good combination of intuition, experience, and experimentation. Specifically, the dimensions of the layers, and the depth of the network (number of layers) are examples of choices affecting the capacity of the network, with direct implications on the bias and the variance.

The bias-variance trade-off. This is a classical problem in machine learning. A high capacity model may tend to overfit, i.e. adapt too heavily to the training data, while performing badly on unseen test data. A model with lower capacity on the other hand, has a higher bias. If the model is biased towards a configuration that does not work well for the problem, it will fail to learn, and perform badly both on training data and on unseen test data. In both cases, the solution does not generalize well to unseen data, a crucial property for any machine learning approach.

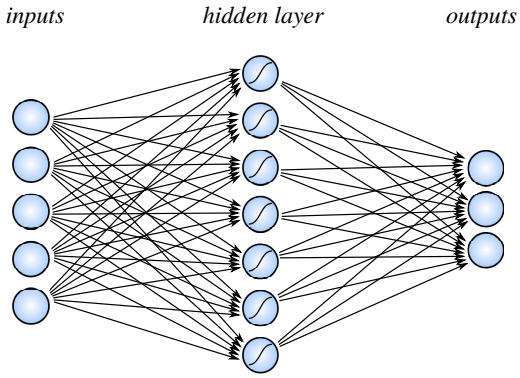


Figure 3.2.1: *An MLP with one hidden layer. Computation takes place in the units illustrated as circles, values are propagated along the arrows. Here, each arrow corresponds to one scalar value, although in this case the layers are fully connected, and thus the value propagated from one layer to the next is effectively a vector. The input layer has 5 dimensions, the hidden layer has 7, and the output layer has 3.*

3.2 Deep neural networks

Given an input vector $\mathbf{x}$, a trained ANN will produce its output by feeding the values through the network and performing the computations in the layers. With only one single layer (i.e. the output layer) in the network, it is only able to represent linear dependencies in the data (this is known as linear classification or regression; in the case of classification, it can be trained using a single-layer perceptron, support vector machine, or logistic regression[1]). An ANN can learn non-linear dependencies in the data if it has at least one hidden layer between its input layer (which is the input vector) and its output layer. This is known as a multi-layer perceptron (MLP). In fact, the universal approximation theorem (Cybenko 1989) states that a feed forward neural network (FFNN) with one hidden layer using a sigmoidal activation function, can approximate an arbitrary continuous function on a compact subset of $\mathbb{R}^n$ to an arbitrary precision, provided that it has enough units. However, the required number of units may grow exponentially with the size of the input, something that can be avoided by making the network deeper. More recent work has demonstrated that the requirement of sigmoidal activation functions is not neccessary (Hornik 1991; Pascanu, Montufar, and Bengio 2013; Montúfar and Morton 2015), but the depth is. The parity function is an example of this[2]. With one hidden layer, a number of units that is exponential in the size of the input, is required to represent the function. Yet with a network of depth $O(\log n)$, a polynomial number of units is sufficient.

A layer in an ANN is a non-linear transformation of an input vector $\mathbf{x}$ to an output vector $\mathbf{h}$. Experimental results have shown that when stacking layers together, the network

[1]Single-layer perceptrons, support vector machines, and logistic regression all produce a linear classifier, but their objective functions and training algorithms differ.

[2]The parity function given a sequence of binary inputs decides whether the number of inputs of value 1 is odd.

tends to learn representations for more simple concepts in the layers close to the input layer, while more abstract and composed concepts are represented deeper into the network (Zeiler and Fergus 2014). This means that a deep network naturally learns hierarchies in the data. For instance, in convolutional models trained for image classification, early layers have been shown to represent simple shapes such as lines with different slope, intermediate layers learn angles, curves, and color gradients, while the layers close to the output layer learn to represent complete objects such as faces, furniture, and animals. The exact activations computed in a deep neural network depend on what training data it has seen, and the choice of regularization and loss function used to train it (see section 3.3). If a network with the right data and training strategy learns to detect the underlying factors that explains the signal in the data, then the network has succeeded in disentangling the factors of variation in the data. This is something that will be discussed in section 5.7.

3.3 Training artificial neural networks

To train ANNs, predictions are performed on training data, and the quality of the predictions is measured using a loss function. When the loss has been computed, back-propagation and gradient descent are used to decide parameter updates.

Supervised learning. When training data contains datapoints $\{\mathbf{x}^{(i)}\}_{i=0}^{N}$ with targets $\{y^{(i)}\}_{i=0}^{N}$, and the task is to learn a mapping from $\mathbf{x}^{(i)}$ to $y^{(i)}$, the problem falls into the realm of *supervised learning*. If the targets are categorical, they are called labels, and the problem is called classification. If the targets are real-valued, it is a regression problem.

Unsupervised learning. In contrast, *unsupervised learning* is when you train a model without targets or labels. Since there are no targets, problems in this category do not ask for a mapping from an input to a target. Examples of this are instead problems like clustering, autoencoders (see section 3.4), and word embeddings (see section 4.1).

Loss functions. The learning objective of a model h_θ parameterized by θ is determined by a loss function measuring the quality of its predictions. θ is the set of all parameters in the model: weight matrices and bias vectors. For regression problems, where the prediction $h_\theta(\mathbf{x})$ is real valued, the loss may be the mean squared error (MSE):

$$\mathcal{L}(\theta) = \frac{\sum_i (y^{(i)} - h_\theta(\mathbf{x}^{(i)}))^2}{N}.$$

Classification problems are generally modeled using a softmax output layer, providing a normalized categorical output that sums to 1:

$$softmax(\mathbf{z}) = \frac{\exp(\mathbf{z})}{\sum_k \exp(\mathbf{z}_k)}.$$

The softmax output is interpreted as a probability distribution over the K categories, and used together with the cross-entropy loss function:

$$\mathcal{L}(\theta) = \frac{-\sum_i \mathbf{y}^{(i)} \log \mathbf{h}_\theta(\mathbf{x}^{(i)})}{N},$$

where $\mathbf{y}^{(i)}$ is the one-hot encoding of the target category for input $\mathbf{x}^{(i)}$ and the sum is over inputs in the batch.

Back-propagation. When a forward pass has been performed (feeding data into the input layer, and performing all computations specified by the layers in the model) and the loss has been computed, the back-propagation algorithm is used to compute the derivatives of the loss function with respect to the model parameters. A key to training ANNs is that every component in the model is continuously differentiable, including the loss function. A complete network models a function, and each layer application is a function composition. The derivative of a composite function is computed using the chain-rule for derivatives. Back-propagation employs a recursive strategy to compute the gradient (all partial derivatives) for a complete network (Rumelhart, G. E. Hinton, and Williams 1986), and thus lets information from the loss function flow back through the network to compute the derivatives with respect to all weights of all layers in the model.

Optimization. When the gradient of a model has been computed using back-propagation, an optimization algorithm such as stochastic gradient descent (SGD), possibly with modifications such as Momentum (Qian 1999), RMSProp, or Adam (Kingma and Ba 2015), is used to compute an update for the model weights. Gradient descent means that you iteratively take steps in the direction opposite to the gradient. In practice, stochastic gradients are often computed by using mini-batches of training data, and then the process is iterated until convergence. This leads to an approximate gradient computation, which can help the optimization procedure avoid getting stuck in local optima. The loss surface of deep ANNs is non-convex (meaning that there can be local optima and saddle points, and gradient descent may not lead to the global optimum), but in practice the optimization generally finds good local minima leading to models that perform well. One of the reasons why the non-convexity does not pose a major problem when optimizing neural network losses is that unlike low-dimensional function surfaces, neural network loss surfaces are high-dimensional, which means that saddle-points tend to be more prominent than local minima. A saddle-point is easier to escape, since SGD generally finds ways out of them, whereas local minima can be problematic. Also, for a saddle-point in a high-dimensional space, the fewer dimensions leading towards lower loss, the closer to the global minimum one tends to be (Choromanska et al. 2015).

The vanishing gradient problem. Very deep architectures can suffer from the *vanishing gradient problem*. This occurs when small numbers (less than one) are multiplied together in the gradient computation, and the resulting gradients for layers in the network that are far from the loss function will tend towards zero. (If the numbers are greater than one, there can be an exploding gradient problem instead). This makes some deep models difficult to train. A number of different solutions have been proposed to the vanishing gradient problem, such as layer-wise pretraining (see below), gated variants of RNNs (see section 5.1), and residual connections for CNNs (see section 5.6). Exploding gradients can be mitigated by gradient clipping.

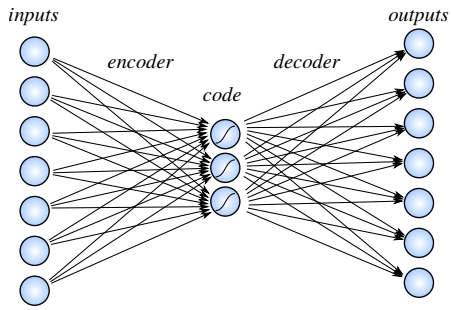


Figure 3.4.1: *A shallow autoencoder, consisting of one coding layer and one decoder layer. The output is trained to reconstruct the input.*

Pretraining. Layer-wise pretraining was presented as a solution to the difficulties of training deep neural networks. In fact, some of the work in this direction was what started much of the depth revolution (G. E. Hinton, Osindero, and Teh 2006). One layer at a time, a restricted Boltzmann machine is trained using the contrastive divergence algorithm to capture the distribution of its input. Another technique for layer-wise pretraining is the autoencoder (see section 3.4). A pretrained layer can be used as an initialization for a layer that is later fine-tuned using back-propagation (see section 3.3). In this way, a meaningful (learned) representation is available from the start of learning the end task, allowing for faster learning. While pretraining was important for the development of deep learning technology, clever random initialization (Glorot and Bengio 2010; He et al. 2015) and increasingly powerful hardware have since become more important for training deep networks on many tasks.

3.4 Autoencoders

For unsupervised representation learning, an ANN can be trained to reproduce its input. A network trained in this way is known as an autoencoder. Figure 3.4.1 shows a shallow autoencoder consisting of one coding layer and one decoder layer. A deep autoencoder has more layers before and after the coding layer. Depending on the data modality, different kinds of loss functions are suitable for this. The coding layer generally has some information compression property, e.g. having a smaller dimension than the input layer. For a model to be an autoencoder, it needs only to be trained to recreate the input. It can be composed by fully connected layers, convolutional layers (e.g. for images, see section 5.6), and recurrent layers (for sequences, see section 5.1). Autoencoders can be used for representation learning with unsupervised training. The representations computed at the coding layer can be used in a downstream task, or the encoder part of the trained network can be used as an initialization (a pre-training strategy) for a network that will later be trained for another task. Autoencoders can also be used for continuous relaxation, embedding discrete structures in contunous space, which can be

traversed or used to optimize some objective to find configurations of interest in the input space. See for instance (Gómez-Bombarelli et al. 2016). In Paper V we train convolutional autoencoders using a specific regularization scheme (see section 5.7).

3.5 Distributed representations

A distributed representation is composed of components that can be varying individually. This is a central property of the representations learned by ANNs (G. E. Hinton 1986). Non-distributed (localist) representations can only describe a number of distinct objects that is linear in the number of dimensions: examples are one-hot encodings[3] and clustering assignments, both of which can express exactly as many distinct objects as there are dimensions (categories or clusters). Examples of algorithms working with non-distributed representations are k-nearest-neighbors, clustering methods (a data point is assigned to exactly one cluster or centroid), and decision trees (similarly: a data point can only be represented by exactly one path in the tree). Distributed representations, on the other hand, can express a number of values that is exponential in the number of dimensions. E.g. a binary vector of length n can represent 2^n different values, whereas a one-hot vector can only represent n distinct values.

Distributed representations are natural for encoding independent underlying factors of variation (see section 5.7), as each dimension can encode a different factor.

[3] A one-hot encoding is a vector the size of the number of categories, which is zero at all positions except for the position representing the represented category (where it is one).

Chapter 4

Learned representations for language

While the early ground-breaking deep learning work targeted image data, ANNs have also proved to be a useful technology to model problems in NLP. As written natural language is a sparse and discrete data modality, some tricks have been developed to be able to handle it in models that are otherwise inherently dense and continuous. Section 4.1 discusses how to represent words using distributed representations learned using ANNs. Section 4.2 presents the work on multi-document summarization that leverages neural word embeddings and principles from kernel learning. Section 5.1 will go into more detail about neural sequence models such as RNNs and its variants.

Similar to how deep learning revolutionized image recognition, to a large part because of its capability to learn internal representations, solutions to problems in NLP have been improved with the use of learned representations for words (Collobert and Weston 2008; Mikolov, K. Chen, et al. 2013; Pennington, Socher, and Manning 2014), documents (Q. Le and Mikolov 2014), phrases (Cho et al. 2014), sentences (Blunsom, Grefenstette, and Kalchbrenner 2014), and even characters (X. Chen et al. 2015).

4.1 Neural word embeddings

Word embeddings are representations for words in a vector space. In contrast to the bag-of-words representations discussed in section 2.1, neural word embeddings are continuous vector space representations of much lower dimensionality (they are also denser). This class of word embeddings are *distributed representations* (see section 3.5), and use ANNs for training. They are also distributional representations, as they are trained using word contexts from large text corpora, and encode (distributionally) semantically similar words close in the vector space. Collobert and Weston (2008) trained the representations using a neural network in a multitask training scenario. In the Word2Vec skip-gram model, the vectors are obtained by training a model to predict the context words, and in the Word2Vec continuous bag-of-words (CBOW) model, the vectors are obtained by

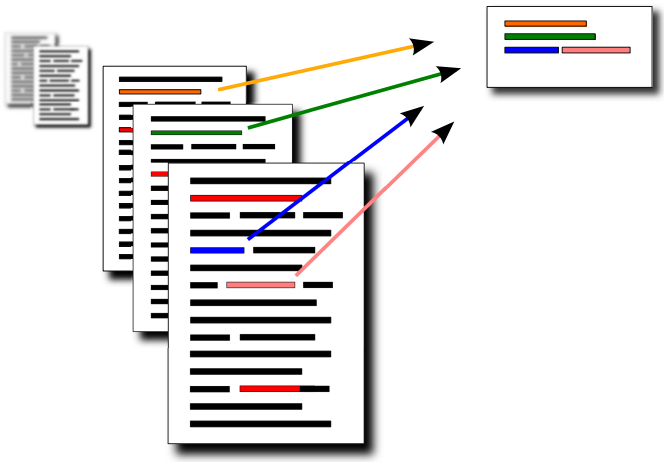


Figure 4.2.1: *Extractive summarization is the process of extracting parts from input documents into a summary that is representative but not redundant.*

averaging the context word vectors, and train a log-linear classifier to predict the word in between (Mikolov, K. Chen, et al. 2013). The GloVe model is trained using global word co-occurrence statistics (Pennington, Socher, and Manning 2014). FastText is based on the skip-gram model and learns representations that take subword information into account (Bojanowski et al. 2016). Vectors from the models mentioned above have been shown to encode many semantic aspects of words. This shows as relations between the corresponding vectors; e.g. the difference between the vectors for a country and its capital is almost constant over different countries (Mikolov, Sutskever, et al. 2013):

$$v_{Stockholm} - v_{Sweden} \approx v_{Berlin} - v_{Germany}$$

4.2 Extractive multi-document summarization

The previous sections have described how models can be trained to compute vector based representations for a number of different kinds of data. Specifically, word embeddings have proven very useful for a number of applications in NLP, including syntactic parsing (Socher, Bauer, et al. 2013), word sense induction (Kågebäck, F. Johansson, et al. 2015), word sense disambiguation (Iacobacci, Pilehvar, and Navigli 2016), and sentiment analysis (Socher, Perelygin, et al. 2013). This section introduces a novel approach that leverages the powerful expressiveness of neural word embeddings for summarizing documents, and a way of aggregating different kinds of sentence similarity measures using ideas from kernel learning.

Extractive multi-document summarization is the process of extracting parts of a set of documents D into a summary S, such that it is brief, representative, and non-redundant. (This is in contrast to abstractive summarization, where the contents of D is abstracted into some intermediate representation and then used to generate a summary, which does

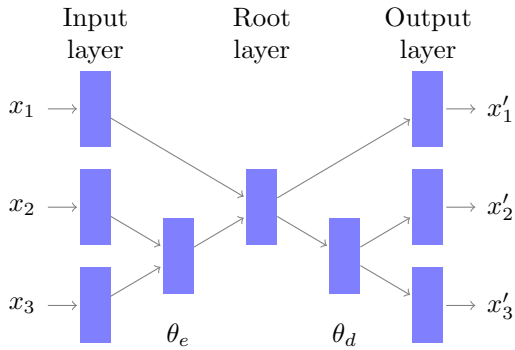


Figure 4.2.2: *An unfolding recursive auto-encoder. In this example, a three words phrase $[x_1, x_2, x_3]$ is the input. The weight matrix θ_e is used to encode the compressed representations, while θ_d is used to decode the representations and reconstruct the sentence.*

not necessarily contain parts of the original documents). A common approach is to consider sentences as candidates for the extraction.

Several factors make this a hard problem. Firstly, it is difficult to evaluate the quality of summaries automatically. Secondly, generating the output, either by selecting sentences from the source documents, or by natural language generation, is a difficult optimization problem.

Previous work (Mihalcea and Tarau 2004; Radev et al. 2004; H. Lin and Bilmes 2011; Kulesza and Taskar 2012) has attacked the extractive problem by defining a similarity scores between sentences, and then applying some form of optimization strategy to select sentences that have a high similarity to the whole input document set D (representativeness or coverage), yet a low similarity to the rest of the current summary S (diversity or non-redundancy). These two objectives are naturally conflicting, and making a good trade-off is a challenge.

H. Lin and Bilmes (2011) represented sentences using bag-of-words vectors weighted by tf-idf (with a vocabulary of unigrams and bigrams), and measured the sentence similarity using the cosine similarity:

$$M_{\mathbf{s}_i,\mathbf{s}_j} = \frac{\sum_{w\in\mathbf{s}_i} tf_{w,i}\cdot tf_{w,j}\cdot idf_w^2}{\sqrt{\sum_{w\in\mathbf{s}_i} tf_{w,\mathbf{s}_i} idf_w^2}\sqrt{\sum_{w\in\mathbf{s}_j} tf_{w,\mathbf{s}_j} idf_w^2}}.$$

They showed that a scoring function measuring sentence similarities like this is a monotonically non-decreasing submodular set-function. This allows for optimization with a greedy approximation algorithm, guaranteed to find a solution within a factor $(1-\frac{1}{e}) \approx 0.63$ of the true optimum. The resulting summaries have scored well in evaluations using the ROUGE metrics. A sentence representation similar to this was used by Radev et al. (2004). Mihalcea and Tarau (2004) also used a similar approach, without the tf-idf weighting, and with a different normalization:

$$M_{\mathbf{s}_i,\mathbf{s}_j} = \frac{|\mathbf{s}_i \cap \mathbf{s}_j|}{(\log|\mathbf{s}_i| + \log|\mathbf{s}_j|)}.$$

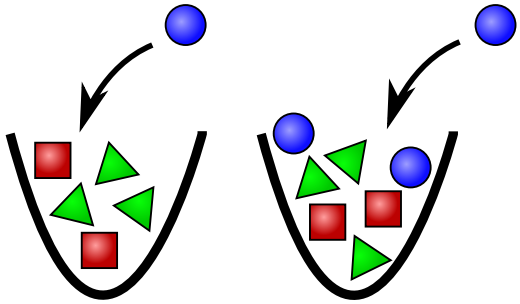


Figure 4.2.3: *The number of shapes in the bowl is a monotonically non-decreasing, submodular set function. The left bowl contains a subset (S) of the objects in the right bowl (T). The function value grows at least as much when adding the ball to S (left), as when adding it to T (right).*

In Paper I, we use word embeddings to capture more semantics in the sentence similarity score. To be able to easily compare two sentences, we first create a sentence representation of fixed dimensionality for each sentence. We evaluate two different ways of doing this: simple vector addition (Mikolov, Sutskever, et al. 2013) and unfolding recursive auto-encoders (RAEs) (see Figure 4.2.2) (Socher, E. H. Huang, et al. 2011). An RAE is an auto-encoder: a feed forward neural network that is trained to reconstruct the input at its output. In an auto-encoder, there is typically a lower-dimensional hidden layer, to obtain an amount of compression. A recursive auto-encoder uses the syntactic parse tree of a sentence to provide the layout of the auto-encoder network. In this way, intermediate representations are trained recursively for pairs of input representations. The dimensionality is the same after each pairwise input.

In the evaluation, two different word representations were used: those of Mikolov, K. Chen, et al. (2013), and of Collobert and Weston (2008). Extending the representations to more complex structures than words (such as sentences and documents) is an active area of research. The proposed summarization method was evaluated using ROUGE (version 1.5.5) (C.-Y. Lin 2004), which reports n-gram overlaps between generated summaries and the gold standard. The paper reports ROUGE-1, ROUGE-2, and ROUGE-SU4 scores, representing matches in unigrams, bigrams, and skip-bigrams, respectively. Skip-bigrams are matches of two words with up to four words in between (that do not need to match). The experimental evaluation suggests that using continuous vector space models affects the resulting summaries favorably, but surprisingly, the simple approach of summing the vectors to sentence representations outperforms the seemingly more sophisticated approach with RAEs.

Paper II builds on the approach with word representations. This paper presents a sentence similarity score based on sentiment analysis (when human authors create summaries for news paper articles, they use negative emotion words at a higher rate (Hong and Nenkova 2014)), and shows that better summaries can be created by combining the word embeddings with the sentiment score and traditional word-overlap measures, by multiplying the scores together. Using multiplication corresponds to a conjunctive aggregation, where a high result means that all involved similarity scores are high, but

if only one of them is low, the resulting score will be low. Products of experts have been shown to work well in many other applications and is a standard way of combining kernels.

We call the resulting system MULTSUM, an automatic summarizer that uses submodular optimization with multiplicative aggregation of sentence similarity scores, taking several aspects into account when selecting sentences.

$$M_{\mathbf{s}_i,\mathbf{s}_j} = \prod M^l_{\mathbf{s}_i,\mathbf{s}_j},$$

where M^l denotes the different similarity measures used.

The resulting summaries were evaluated on DUC 2004; the de-facto standard benchmark dataset for generic multi-document summarization, and obtain state-of-the-art results. Specifically, the scores where MULTSUM excels are ROUGE-2 and ROUGE-SU4, with matches in bigrams and skip-bigrams (with up to four words in between), respectively, suggesting that the resulting summaries have high fluency.

Chapter 5

End-to-end learning

The previous chapters have discussed representations that were learned separately, and then used in a downstream task such as summarization (Paper I and Paper II). This means that there are assumptions (implicit or explicit) being made when choosing the model and algorithm to learn the representations, so as to make sure that the representations will perform well for the downstream task. As mentioned in section 3.1, ANNs can learn to compute internal representations with increasing levels of abstraction, while being trained to solve the end task. This is just a result of the gradient based learning; weights are updated in the direction opposite to the gradient, at the same time modifying the computed representations, so as to minimize the loss function measuring the quality of the output. Letting the optimization of the loss function affect all parameters including the representations for the atomic language units makes sure that they become adapted to work well for the specific task.

Solutions that make use of end-to-end machine learning techniques have been proposed for all tasks mentioned in chapter 2 (Sutskever, Vinyals, and Q. V. Le 2014; Bahdanau, Cho, and Bengio 2015).

In this chapter, we will consider problems with solutions based on end-to-end learning approaches (Paper III and Paper IV), and study the representations that are learned in the models. In Paper V, we present a regularization scheme that helps to learn disentangled representations.

5.1 Neural sequence modeling

Language is often modeled using recurrent neural networks (RNNs), which can model arbitrary sequences of data, keeping a state representation as it processes the data in the sequence. The main building block in the most basic RNN variant, is the recurrent layer (see Figure 5.0.1). This layer is reused (with the same weights) at each position t in the input sequence $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}$, and at time t takes as input a concatenation of (1) an input vector $\mathbf{x}^{(t)}$ from the sequence and (2) the current state representation $\mathbf{h}^{(t-1)}$. Then

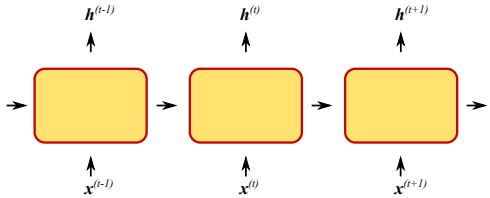


Figure 5.0.1: *A recurrent neural network, with a sequence of inputs* $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}$, *and sequence of outputs* $\mathbf{h}^{(0)}, \mathbf{h}^{(1)}, ..., \mathbf{h}^{(N)}$.

a linear transformation is performed followed by the nonlinear activation:

$$\mathbf{h}^{(t)} = \tanh(W\mathbf{x}^{(t)} + U\mathbf{h}^{(t-1)} + \mathbf{b}),$$

where W and U are trainable weight matrices, $\mathbf{b}$ is the bias vector, and tanh is the nonlinearity.

A neural language model is an RNN that has as input a textual sequence (of words, subword units, or characters). The model is trained to predict the next token at every position in the sequence. The input to the RNN is the vector representation of the token, and the prediction output is modeled using a linear layer with softmax activation which learns the probability of each token in the vocabulary given the previous tokens in the sequence. A linear input layer is used to transform one-hot encodings (see section 3.5) into real-valued vectors (this is sometimes referred to as an *embedding layer*).

In effect, the weights in the embedding layer of a neural sequence model will learn a vector representation for each token in the vocabulary, which often has properties similar to the embeddings discussed in section 4.1, but will also be adapted to work better for the end task, since the whole model (including the embeddings) is trained using back-propagation and SGD to minimize the error in the output. (For details on training, see section 3.3)

Basic RNNs struggle with learning long dependencies and suffer from the *vanishing gradient problem* from the depth in the sequence dimension (see section 3.2). This makes RNN models difficult to train (Hochreiter 1998; Bengio, Simard, and Frasconi 1994), and motivated the development of the Long short-term memory (LSTM) (Schmidhuber and Hochreiter 1997), that to some extent solves these shortcomings.

Recurrent networks with gates. An LSTM is an RNN where the cell at each step t contains an internal memory vector $\mathbf{c}^{(t)}$, and three gates controlling what parts of the internal memory will be erased from the internal memory (the forget gate $\mathbf{f}^{(t)}$), what parts of the input that will be stored (the input gate $\mathbf{i}^{(t)}$), as well as what will be included in the output (the output gate $\mathbf{o}^{(t)}$). In essence, this means that the following expressions are evaluated at each step in the sequence, to compute the new internal memory $\mathbf{c}^{(t)}$ and

the cell output $\mathbf{h}^{(t)}$. Here "$\odot$" represents element-wise multiplication:

$$\begin{aligned}
\mathbf{i}^{(t)} &= \sigma(W^{(i)}\mathbf{x}^{(t)} + U^{(i)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(i)}),\\
\mathbf{f}^{(t)} &= \sigma(W^{(f)}\mathbf{x}^{(t)} + U^{(f)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(f)}),\\
\mathbf{o}^{(t)} &= \sigma(W^{(o)}\mathbf{x}^{(t)} + U^{(o)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(o)}),\\
\mathbf{u}^{(t)} &= \tanh(W^{(u)}\mathbf{x}^{(t)} + U^{(u)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(u)}),\\
\mathbf{c}^{(t)} &= \mathbf{i}^{(t)} \odot \mathbf{u}^{(t)} + \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)},\\
\mathbf{h}^{(t)} &= \mathbf{o}^{(t)} \odot \tanh(\mathbf{c}^{(t)}).
\end{aligned} \tag{5.1.1}$$

LSTM networks has been successfully applied to many problems, including handwriting recognition (Graves et al. 2009), sequence-to-sequence learning (Sutskever, Vinyals, and Q. V. Le 2014), and language modeling (T. Wang and Cho 2016).

The gated recurrent unit (GRU) (Cho et al. 2014) is a simplification of this approach, having only two gates by replacing the input and forget gates with an update gate $\mathbf{u}^{(t)}$ that simply erases memory whenever it is updating the state with new input. Hence, the GRU has fewer parameters, and still obtains similar performance as the original LSTM in many tasks.

In Paper III and Paper IV, the LSTM and GRU variants are used, respectively, to attack problems in medical named entity recognition and morphology.

5.2 Sequence-to-sequence models

A sequence-to-sequence (or encoder-decoder) model is a conditional language model from which one can sample a target sequence given a source sequence (Sutskever, Vinyals, and Q. V. Le 2014; Cho et al. 2014). The obvious application is machine translation. Early iterations of this strategy used one RNN to encode a source sentence (as a sequence of words), the final state of the encoder was then fed into an RNN that was trained to generate the corresponding sentence in another language.

Attention mechanism. The performance of the early sequence-to-sequence models degraded quickly with longer input sentences. Sutskever, Vinyals, and Q. V. Le (2014) observed that reversing the source sentence led to better results. In that way, fewer computational steps were performed between the input of the beginning of the source sentence, and the generation of the beginning of the target sentence. Bahdanau, Cho, and Bengio (2015) further shortened the distance between the source token and the target token during translation, by introducing the *attention mechanism* (see Figure 5.2.1).

At each step t_d in the output generation, a weighted sum of the states $\mathbf{h}^{(0)}, \mathbf{h}^{(1)}, ..., \mathbf{h}^{(N_e)}$ from the encoder is computed: $\sum_{t_e} w^{(t_e)}\mathbf{h}^{(t_e)}$. Each weight $w^{(t_e)}$ in this sum, corresponding to each encoder token, is computed using an MLP with the current hidden state of the decoder $\mathbf{h}^{(t_d)}$, and the state of the encoder $\mathbf{h}^{(t_e)}$ as inputs. The weights are normalized with a softmax activation so that they sum to one. In effect, this leads to an alignment of

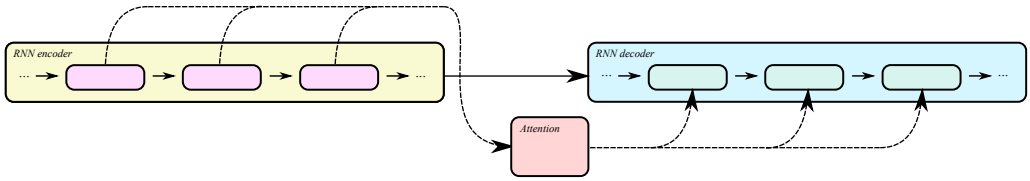


Figure 5.2.1: *Sequence-to-sequence RNN (encoder-decoder) with an attention mechanism. The attention mechanism internally consists of an MLP that computes weights for a weighted sum of encoder states.*

the two sentences, and the decoder can "focus" on a specific part of the source sentence at every step in the generation process. The introduction of attention mechanisms led to a big improvement in translation accuracy.

5.3 Subword and character-based modeling

In the models discussed above, language is modeled as sequences of words. In the context of ANNs (as well as other approaches), this is normally modeled using a fixed vocabulary of the most common words, resulting in solutions where infrequent words are out-of-vocabulary (OOV), and in the case of language generation, the inability to generate anything outside of the vocabulary. A special <UNK> token is introduced to handle the OOV words. Another issue with word-based modeling is the necessary preprocessing step of tokenization, which introduces a source of errors when applied to noisy real-world language, possibly in new domains or low-resource settings. Also, with a large vocabulary, the normalizing constant in the output layers can pose a computational challenge, and word-based models are unable to learn subword patterns.

In many NLP problems, it is possible to overcome the problems listed above by modeling the text with the raw character stream as input. This is implemented similarly to a word-based model, but instead of a word vocabulary, a character vocabulary is used. The input layer is a linear embedding layer, with embeddings for characters instead of words. In the case of a language model, the output layer is a linear transformation with softmax activation, trained to predict the next character given the sequence of preceeding characters. A character-based model can represent any word that is written using the same alphabet, can learn subword patterns, needs no tokenization as preprocessing, and requires an output layer with a size not bigger than the alphabet, reducing the computational cost of normalization in the softmax output. The downside is that a sequence of characters is about 5-10 times longer, which means that the gradients need to be propagated a long way. There have been hybrid approaches to mitigate this problem, such as using the byte-pair encoding to create a vocabulary of common subword patterns (Sennrich, Haddow, and Birch 2016), or to use word-level models with a character-based fallback RNN that is used only when an OOV is encountered (Luong and Manning 2016; Labeau and Allauzen 2017). For many problems, however, such as morphological inflection and reinflection (Kann and Schütze 2016), text classification (X. Zhang, Zhao, and LeCun 2015), and language modeling (Kim et al. 2016), using the raw character stream can be

of a great benefit.

5.4 Recognizing medical entities in electronic patient records

Character-based neural models can detect patterns on a subword level. In Paper III, we explore how this can be used to detect medical terms in electronic patient records. We frame the problem as a variation of named entity recognition (NER), where the task in general is to recognize and classify named entities such as people, organizations, and locations. In this work, the names of interest are instead 1. *disorders and findings*, 2. *pharmaceutical drugs*, 3. *body structure*. Medical documents have specific terminology including many non-standard abbreviations and multi-word expressions. The same term can often be spelled in several different ways, and abbreviations can refer to different things depending on context.

Previous work for NER in the medical domain has used conditional random fields (CRFs) (Y. Wang and Patrick 2009) and hidden Markov models (HMMs) (Zhou et al. 2004). For Swedish medical health records, the state-of-the-art results was presented by Skeppstedt et al. (2014), with a CRF model. These all require engineered features, such as lemma, POS, prefixes, suffixes, and orthographics (e.g. number, word, capitalisation). More recently, some work has been published that uses RNNs for sequence labeling tasks. Cícero Nogueira dos Santos (2015) presented a model that learns word embeddings along with character embeddings from a convolutional layer, which are used as representations for a window-based feed forward neural network. Z. Huang, W. Xu, and Yu (2015) proposed a bidirectional LSTM model, but it used traditional feature engineering, and the classification was performed using a CRF layer in the network. In contrast, the model proposed in Paper III learns all its features, and can be trained efficiently with simple back-propagation and stochastic gradient descent. Ma and Hovy (2016) presented a model that uses a convolutional network to compute representations for sub-word units. The representations are combined with character-level features and fed into a bidirectional LSTM network, and finally a CRF performs the labelling. Chiu and Nichols (2016) presented a similar model but with a softmax output instead of the CRF layer. Lample et al. (2016) presented two different architectures, one using LSTMs with CRFs, and one using a shift-reduce approach. Gillick et al. (2016) presented a character-based model with LSTM units similar to a translation model, but instead of decoding into a different language, the state from the encoder is decoded into a sequence of tags. Like our system, the models are trained end-to-end and obtains good results on standard NER evaluations, however, the model in Paper III is conceptually simpler, and learns all of its features directly from the character stream.

In Paper III, the problem was approached using a character based RNN, that simultaneously recognizes and classifies each mention of a term into one of the three classes above. The network is trained to compute a classification for each character in the sequence, removing the need for IOB encodings: either the character should be classified as one of the three categories above, or it was not part of a mention. The boundary detection follows automatically from this. Traditional named entity recognition systems

rely heavily on hand-crafted character-based features, such as capitalization, numerical characters, prefixes, and suffixes (Ratinov and Roth 2009), things that are well-suited to be learned using a character-based RNN. The proposed model outputs one classification for each character of an input document, and there is nothing limiting the model from classifying a word inconsistently. For this reason, the character classification is treated as a voting mechanism, and the majority for each word is chosen. If a space between two tokens is classified consistently with the two surrounding tokens, it is considered part of a multi-word expression. Otherwise, they are treated as two different entity mentions.

The model was evaluated using Stockholm EPR corpus (Dalianis, Hassel, Henriksson, et al. 2012), and a new dataset was developed for training and development, consisting of medical texts in Swedish from Läkartidningen (Kokkinakis and Gerdin 2010), Swedish Wikipedia, and the Swedish online health care guide, `1177.se`. The test set with real patient records contain misspellings, redundancy, and highly diverse writing style (Dalianis, Hassel, and Velupillai 2009). In the experimental evaluation, the proposed model outperformed a bag-of-words based baseline (S. Zhang and Elhadad 2013), and obtained good results for entity classification. This work demonstrates the ability of character-based neural models to learn subword patterns, and to generalize well to data that is noisy and with a difference in quality between the training data and the test data.

5.5 Morphological relational reasoning

For many tasks in NLP, words appearing in different forms have been considered a problem. The solution has sometimes been to use stemming: a way of removing common suffixes and prefixes from words to retain its stem, or lemmatization: a more involved process to predict the lemma given an inflected word form. This is often done as a preprocessing step before the problem of interest is attacked. This preprocessing may lead to better performance for some algorithms working with count based representations (see section 2.1), but it removes much of the information in the text. A character-based RNN can learn to identify and make use of the information encoded by prefixes, suffixes, infixes, and transformations such as umlaut. In Paper IV, we demonstrate how such a network learns to identify morphological forms in its internal embeddings, while solving the task of generating morphological analogies.

The paper defines *morphological relational reasoning*: given a morphological demo relation (a word in a source form and a target form), and a query word q in the source form, what is the target form of q? E.g.: *see* is to *saw* as *eat* is to what? We approach the problem using an extended sequence-to-sequence-model with an attention mechanism (see section 5.2), with the major difference being that we introduce the relation encoder. The relation encoder takes the demo relation using two separate RNNs (having shared weights), followed by a fully connected layer with tanh activations. The embeddings computed by the relation encoder are fed together with the representation computed by a separate RNN for the query word to the decoder RNN (see Figure 5.5.1). The model is trained to generate the query word in its target form using the decoder RNN, and the only training signal comes from the loss function measuring how close the decoder is to generating the correct characters.

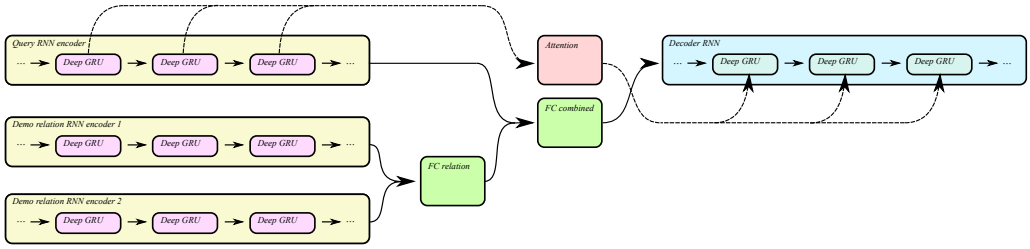


Figure 5.5.1: *The layout of the proposed model for morphological relational reasoning. The demo relation is encoded using two encoder RNNs with shared weights for the two demo words. A fully connected layer* FC relation *follows the demo relation pair. The query word is encoded separately, and its embedding is concatenated with the output from* FC relation*, and fed as the initial hidden state into the RNN decoder which generates the output while using an attention pointer to the query encoder.*

Through visualizations done using t-SNE projections (see Figure 5.5.2), one can see that the embeddings computed by the demo relation encoder RNN (for the English validation set) can capture and disentangle the information about the morphological relation being demonstrated by the word pair. Without any explicit training signal telling it about morphological tags, the relation encoder has learned to separate all morphological relations in the English data, except for two: *"singular-plural"* for nouns, and *"infinitive-3rd pers. singular present"* for verbs, both realized in English by appending the suffix *"-s"* in one direction, and removing it in the inverse direction. These exact relations share the same space in the embeddings, and can therefore be mistaken for each other by the model. Related (and somewhat simpler) problems are morphological inflection and reinflection, where a source form is given with its morphological tags, together with the tags for the target form, and the task is to generate the word in the target form. Character-based RNNs have successfully been applied to solve also these related problems (Kann and Schütze 2016). Morphological relational reasoning is a harder problem, since our model needs not only to perform the transformation from source form to target form, but also first infer the morphological forms represented by the demo relation.

One may recall (see section 4.1) that neural word embeddings are evaluated similarly, using either semantic or syntactic analogies. However, word embedding models are confined to a fixed vocabulary, whereas the proposed character-based model is not. In fact, the paper contains a comparison using word embeddings as a baseline, and the proposed character-based model outperforms the baseline by a large margin.

The proposed model was evaluated experimentally on five different languages, and beat the baseline methods consistently over all languages. Particularly, for English, the prediction accuracy is 95.60%. Furthermore, we show that the representations learned by the relation encoder disentangles the morphological forms well.

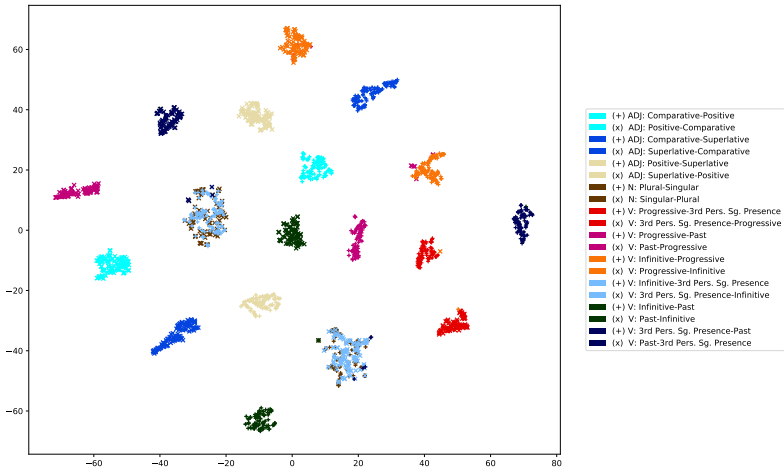


Figure 5.5.2: *Morphological relations (transformations from one morphological form to another), embedded using the internal representation in the model in Paper IV.*

5.6 Convolutional neural networks

Convolutional neural networks (CNNs) were proposed as a neural architecture for image classification (LeCun et al. 1990), with one of the most effective ways of reusing parameters. CNNs are currently the best-performing architechture for many different tasks involving images. The Imagenet image classification competition has been dominated by CNNs since 2012, when Krizhevsky, Sutskever, and G. E. Hinton trained a large CNN with the raw images as its only input, learning its own internal representations. Karen Simonyan (2015) trained a network with smaller convolution filters, but were able to train a deeper network with 19 layers. The inception architecture (Szegedy et al. 2015) allowed for training an even deeper network, improving the results even further. This network had 22 layers. He et al. (2016) was able to make the network even deeper, by incorporating *residual connections*, bypassing one or more layers and thus letting information flow more easily both in the forward-pass and in the backward-pass, limiting the effects of the vanishing gradient problem (see section 3.2). They obtained significantly better results, with a model traned with a total of 152 layers. CNNs have been used successfully as components in solutions of many related problems, such as deep reinforcement learning (Mnih et al. 2013; Silver et al. 2016), and generative adversarial networks (GANs) (Goodfellow, Pouget-Abadie, et al. 2014).

Convolutional layers. A CNN is a neural network with one or more convolutional layers. Instead of the standard linear transformation using a matrix multiplication performed in fully connected layers, convolutional layers have convolution filters, which are applied using the *convolution operator* (see Figure 5.6.1). The convolution operator takes the input data with grid structure, and one or more convolutional filters, small

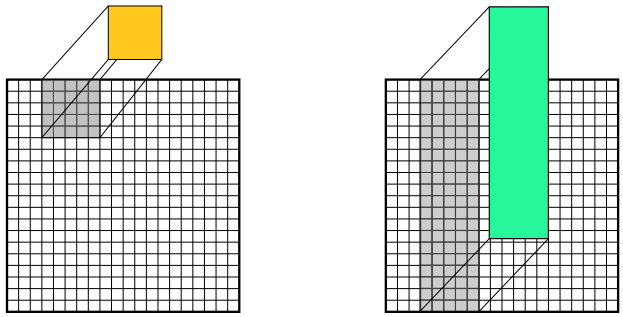


Figure 5.6.1: *Convolutional filters applied to a part of an input.* **Left***: a* 5×5 *filter (yellow) applied to a 2-dimensional input (e.g. the pixel values of a grey-scale image). The convolution operator applies the filter to every fitting position of the input (with some stride* ≥ 1*). Applying the filter means taking the dot product between the filter and the matching region (here: grey).* **Right***: a* 5×20 *filter (green) applied to a 2-dimensional (*20×20*) input. This results in a 1-dimensional convolution, something that can be used to model sequences such as language, with an input being a stacked sequence of embedding vectors.*

matrices that is trained in the normal way with back-propagation and SGD. The filters are applied at every possible position of the input data, computing a dot product between the filter and the corresponding part of the input, resulting in one (scalar) activation for each such possible position. When using a suitable amount of zero-padding around the input data (image), the output activation of a convolutional layer can be made to have the same dimensions as the input.

For instance, for two-dimensional images, the convolution filter may be a 5×5 pixel square which has been trained to detect some shape in the input image. At locations of the input image where it is similar to the learned shape, the resulting dot product will be high.

The convolution operation gives the layer an invariance to location of the features. The filters learn detectors of different structures in the input, and they are insensitive to where in the input the structures are found. The filter's weights are used many times on the same input, generating one scalar valued output at every position. The convolution operation is highly parallelizable, and have optimized implementations on GPU hardware, making CNNs fast to train and evaluate. Compared to making an equally expressive fully connected network, the corresponding CNN is much smaller.

A CNN can be used to model sequences. Filters are then convolved over the sequence dimension of the data, and can detect patterns over the filter length (see Figure 5.6.1, right). This is a 1-dimensional convolution, in contrast to 2-dimensional convolutions for image data. In this way they have been applied successfully to some natural language tasks (Kalchbrenner and Blunsom 2013; Kalchbrenner, Grefenstette, and Blunsom 2014; Kim et al. 2016).

Image captioning. The beauty and flexibility of deep learning is demonstrated when the different building blocks are put together. Using the gradient based training, a network can learn to find patterns in all data modalities, also combined in the same model. An example of this is models for image captioning (K. Xu et al. 2015). A part of the network learns representations for an input image with convolutional layers, while an RNN part learns to take those representations as input and to generate a text describing the image. An image captioning system with an attention mechanism computes the attention weights for convolutional activations corresponding to regions of the input image. The whole process is analogous to sequence-to-sequence models for translation, but the encoder RNN is replaced with a CNN: the model makes "translations" from images to text.

Convolutional autoencoders. A convolutional autoencoder uses convolutional layers to encode suitable data to a vector embedding (see section 3.4). It is trained using a decoder with the objective to recreate the inputs. In this thesis, we consider convolutional autoecoders using a decoder with transposed convolution layers[1], with the mean squared error loss function (see section 3.3).

5.7 Disentanglement

When the representations computed by a model explains the underlying variations in the data well, we say that the representations disentangle the underlying factors in the data. The final layer in an ANN can only make a linear transformation (essentially, in the softmax classification case, it performs logistic regression, possibly multi-class), and for this to be meaningful, the representations coming as inputs to this layer need to have been transformed into easily distinguishable regions. In Paper IV, we demonstrated that the morphological relations were disentangled by the relation encoder, as they were embedded in a space where they were easily distinguished and separated (see section 5.5). This is a good example of disentangling the factors that are interesting for solving the problem, something beneficial to most problems and models. Training deeper models helps with diesentanglement, as each non-linear transformation helps to uncover the underlying factors. The hierarchical representations learned by deep neural networks are well suited to find such underlying factors in real data, as they are often possible to express in terms of other, simpler factors of variation (Goodfellow, Bengio, and Courville 2016). In some situations, a model may struggle to find and disentangle the underlying causes of variation, possibly due to weak training signal, or a signal containing factors that have been mixed together. A lower-dimensional representation generally learns dimensions in the data that are less dependent; an effective way to decrease dimensionality is to remove redundant components (Goodfellow, Bengio, and Courville 2016).

In Paper V, we present a regularization scheme that helps an ANN learn uncorrelated features. The approach is a regularization term that penalizes the terms in the covariance matrix of the activations of a layer over a batch of data. This is a simple approach that (1) helps learn disentangled factors of variation, and (2) disables all superfluous

[1]The transposed convolution operation is sometimes incorrectly called "deconvolution operation", but it is not the same as the operation by the same name in signal processing.

dimensions in a regularized layer. Furthermore, the approach is simple to implement, and inexpensive to compute. The correlation penalty favors representations which have linearly independent dimensions. However, the representations have been computed using the highly non-linear transformations of a neural network. The requirement is weaker than to penalize all dependency e.g. through measuring the mutual information, such as suggested by works on non-linear independent component analysis, but our solution is simpler, computationally less expensive, and works well in practice (Lappalainen and Honkela 2000; Honkela and Valpola 2005).

The proposed covariance regularization term (L_Σ) for a layer ℓ, is computed as

$$L_\Sigma = \frac{\sum_{i,j=0}^{N} |\mathcal{C}_{ij}|}{d^2},$$

where d is the dimensionality of ℓ, and $\mathcal{C} \in \mathcal{R}^{d \times d}$ is the sample covariance of the activations in ℓ over N examples.

An experimental evaluation was performed, applying the covariance regularization to a number of models performing different tasks, and the following was concluded. (1) The trained models generally learned to use a *minimal number of neccessary units*. This can be used as an intermediate result when searching for the right dimensions of a model. (2) The activations that are used in the model are *minimally correlated*.

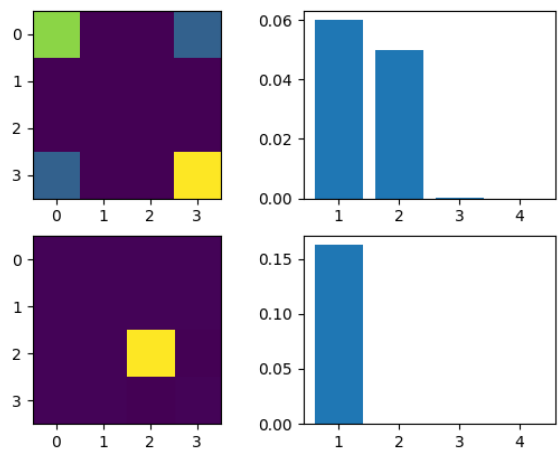


Figure 5.7.1: *Covariance matrix (left) and spectrum (right) of the hidden layers of a feed forward neural network trained with covariance regularization L_Σ to solve the XOR problem. Layer one (top) has learned to utilize unit zero and three while keeping the rest constant, and in layer two only unit two is utilized. This learned structure is the minimal solution to the XOR problem.*

When training an MLP to learn the XOR function using two hidden layers and four

units in each hidden layer (this model is overspecified, the problem can be solved with one hidden layer of 2 dimensions), the network with covariance regularization L_Σ learns to utilize only the needed dimensions in each regularized layer, and to put all variation on the two neccessary units in the first hidden layer (see Figure 5.7.1). The two dimensions used are not uncorrelated (they need to have negative correlation in order to solve this problem). A similar result was found when random vectors were projected into higher-dimensional space using a random projection, and then used to train an autoencoder with L_Σ regularization. The model discovered the number of dimensions of variation in the underlying data.

To verify the approach on real-world data, a convolutional autoencoder with two convolutional layers followed by two fully connected layers was trained on CIFAR-10 (Krizhevsky and G. Hinton 2009) with L_Σ regularization applied to the 84-dimensional coding layer. The regularization term helps the model learn uncorrelated features in the image data, and less than 36 utilized dimensions at 90% variance loss.

The experimental evaluation showed for both synthetic tasks and for the autoencoder trained on real image data, that the L_Σ regularization is a practical way to make representations uncorrelated while retaining the model's performance. This is useful for interpretability, as one can more easily identify what the representations do.

Chapter 6

Concluding remarks

Representation learning has been transforming the field of machine learning. Learned representations enable solutions that are more adaptable, and have a greater generality. Gradient based learning of hierarchical representations have performed exceptionally well for a large number of tasks. This thesis has demonstrated a number of tasks in natural language processing (NLP) where learned representations have been beneficial. In particular, this included work that makes use of neural word embeddings for extractive multi-document summarization, showing that these rich representations can be of great benefit to traditional statistical algorithms in NLP. The proposed multi-document summarizer employs a novel approach to aggregate multiple ways of measuring semantic similarity between sentences, and obtains results that beat the state-of-the-art. Furthermore, we have presented models based on recurrent neural networks (RNNs), trained end-to-end, and working on the raw character stream, that tackle some interesting problems in NLP. In Paper III, we trained a character-based RNN to detect and classify mentions of medical terms in Swedish patient health record data. In Paper IV, we proposed a novel character-based neural architecture to perform morphological relational reasoning; a syntactic analogy task similar to the tasks used to evaluate the performance of word embedding models, but where these closed-vocabulary solutions fail much because of the limited vocabulary. Within the model, the internally computed representations were evaluated and visualized, showing that the model can learn to compute representations that capture and disentangle the necessary underlying factors of variation: the class of the morphological relation that was demonstrated in the analogy. In Paper V, we proposed a regularization technique that penalizes correlation between activations in a layer. This helps a model learn more interpretable and disentangled activations. Also, it helps with the design of deep learning models by estimating the required dimensionality, while being computationally inexpensive and simple to implement.

Deep learning models are well suited to perform many tasks in NLP. The feed-forward structure bears some similarity to the traditional solutions using pipelines with different components performing subtasks. The power of training the neural models end-to-end, using one well defined optimization objective, allows for each component in the solution to be adapted specifically to the task, and for each layer in the feed-forward network to

compute an output that is increasingly useful for the task. Eventually the model can compute representations that disentangles the underlying factors of variation that explains the data, and is useful for the final output.

References

Almgren, S., S. Pavlov, and O. Mogren (2016). "Named Entity Recognition in Swedish Health Records with Character-Based Deep Bidirectional LSTMs". *Proceedings of the Fifth Workshop on Building and Evaluating Resources for Biomedical Text Mining*.

Bahdanau, D., K. Cho, and Y. Bengio (2015). Neural machine translation by jointly learning to align and translate.

Bengio, Y., P. Simard, and P. Frasconi (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on* **5**.2, 157–166.

Blunsom, P., E. Grefenstette, and N. Kalchbrenner (2014). "A convolutional neural network for modelling sentences". *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics.

Bojanowski, P. et al. (2016). Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606*.

Boser, B. E., I. M. Guyon, and V. N. Vapnik (1992). "A training algorithm for optimal margin classifiers". *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, pp. 144–152.

Cavnar, W. B., J. M. Trenkle, et al. (1994). N-gram-based text categorization. *Ann Arbor MI* **48113**.2, 161–175.

Chen, X. et al. (2015). "Joint Learning of Character and Word Embeddings." *IJCAI*, pp. 1236–1242.

Chiu, J. and E. Nichols (2016). Named Entity Recognition with Bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics* **4**, 357–370. ISSN: 2307-387X. URL: https://transacl.org/ojs/index.php/tacl/article/view/792.

Cho, K. et al. (2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. URL: http://www.aclweb.org/anthology/D14-1179.

Choromanska, A. et al. (2015). "The Loss Surfaces of Multilayer Networks". *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. Ed. by G. Lebanon and S. V. N. Vishwanathan. Vol. 38. Proceedings of Machine Learning Research. San Diego, California, USA: PMLR, pp. 192–204. URL: http://proceedings.mlr.press/v38/choromanska15.html.

Cícero Nogueira dos Santos, V. G. (2015). Boosting named entity recognition with neural character embeddings. *Proceedings of NEWS 2015 The Fifth Named Entities Workshop*.

Collobert, R. and J. Weston (2008). "A unified architecture for natural language processing: Deep neural networks with multitask learning". *Proceedings of ICML*, pp. 160–167.

Cox, D. R. (1958). The Regression Analysis of Binary Sequences. *Journal of the Royal Statistical Society. Series B (Methodological)* **20**.2, 215–242. ISSN: 00359246. URL: http://www.jstor.org/stable/2983890.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)* **2**.4, 303–314.

Dalianis, H., M. Hassel, A. Henriksson, et al. (2012). Stockholm EPR Corpus: A clinical database used to improve health care. *Swedish Language Technology Conference*, 17–18.

Dalianis, H., M. Hassel, and S. Velupillai (2009). "The Stockholm EPR Corpus—characteristics and some initial findings". *Proceedings of the 14th International Symposium on Health Information Management Research - iSHIMR*.

Damaschke, P. and O. Mogren (2014). Editing simple graphs. *Journal of Graph Algorithms and Applications, Special Issue of selected papers from WALCOM 2014* **18**.4, 557–576. DOI: 10.7155/jgaa.00337.

Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*.

Gillick, D. et al. (2016). "Multilingual Language Processing From Bytes". *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 1296–1306. URL: http://www.aclweb.org/anthology/N16-1155.

Glorot, X. and Y. Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256.

Gómez-Bombarelli, R. et al. (2016). Automatic chemical design using a data-driven continuous representation of molecules. *arXiv preprint arXiv:1610.02415*.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. MIT press.

Goodfellow, I., J. Pouget-Abadie, et al. (2014). "Generative adversarial nets". *Advances in Neural Information Processing Systems*, pp. 2672–2680.

Graves, A. et al. (2009). A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence* **31**.5, 855–868.

Hagstedt P Suorra, J. and O. Mogren (2016). "Assisting discussion forum users using deep recurrent neural networks". *ACL Workshop on representation learning for NLP, RepL4NLP*.

He, K. et al. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

— (2016). "Deep residual learning for image recognition". *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hinton, G. E. (1986). "Learning distributed representations of concepts". *Proceedings of the eighth annual conference of the cognitive science society*. Vol. 1. Amherst, MA, p. 12.

Hinton, G. E., S. Osindero, and Y.-W. Teh (2006). A fast learning algorithm for deep belief nets. *Neural computation* **18**.7, 1527–1554.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **6**.02, 107–116.

Hong, K. and A. Nenkova (2014). "Improving the estimation of word importance for news multi-document summarization". *Proceedings of EACL*.

Honkela, A. and H. Valpola (2005). "Unsupervised variational Bayesian learning of nonlinear models". *Advances in neural information processing systems*, pp. 593–600.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks* **4**.2, 251–257.

Huang, Z., W. Xu, and K. Yu (2015). Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Iacobacci, I., M. T. Pilehvar, and R. Navigli (2016). "Embeddings for Word Sense Disambiguation: An Evaluation Study". *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. Berlin, Germany: Association for Computational Linguistics, pp. 897–907. URL: `http://www.aclweb.org/anthology/P16-1085`.

Kågebäck, M., F. Johansson, et al. (2015). "Neural context embeddings for automatic discovery of word senses". *Proceedings of NAACL-HLT*, pp. 25–32.

Kågebäck, M. and O. Mogren (2017). "Disentangled activations in deep networks". *Submitted draft. Early version presented at the NIPS Workshop on Learning Disentangled Features*.

Kågebäck, M., O. Mogren, et al. (2014). "Extractive summarization using continuous vector space models". *Second Workshop of Continuous Vector Space Models and their Compositionality*.

Kalchbrenner, N. and P. Blunsom (2013). "Recurrent Continuous Translation Models." *EMNLP*. Vol. 3. 39, p. 413.

Kalchbrenner, N., E. Grefenstette, and P. Blunsom (2014). "A Convolutional Neural Network for Modelling Sentences". *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Baltimore, Maryland: Association for Computational Linguistics, pp. 655–665. URL: `http://www.aclweb.org/anthology/P14-1062`.

Kann, K. and H. Schütze (2016). "MED: The LMU System for the SIGMORPHON 2016 Shared Task on Morphological Reinflection". *Proceedings of the 14th Annual SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pp. 62–70.

Karen Simonyan, A. Z. (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition by". *International Conference on Learning Representations, ICLR*.

Kim, Y. et al. (2016). "Character-Aware Neural Language Models." *AAAI*, pp. 2741–2749.

Kingma, D. and J. Ba (2015). Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*.

Kokkinakis, D. and U. Gerdin (2010). Läkartidningens arkiv i en ny skepnad - En resurs för forskare, läkare och allmänhet. *Språkbruk*, 22–28. URL: `http://demo.spraakdata.gu.se/svedk/pbl/sprakbruk20100316.pdf`.

Krizhevsky, A. and G. Hinton (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "Imagenet classification with deep convolutional neural networks". *Advances in neural information processing systems*, pp. 1097–1105.

Kulesza, A. and B. Taskar (2012). Determinantal point processes for machine learning. *arXiv:1207.6083*.

Labeau, M. and A. Allauzen (2017). "Character and Subword-Based Word Representation for Neural Language Modeling Prediction". *Proceedings of the First Workshop on Subword and Character Level Models in NLP*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 1–13. URL: http://www.aclweb.org/anthology/W17-4101.

Lample, G. et al. (2016). "Neural Architectures for Named Entity Recognition". *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 260–270. URL: http://www.aclweb.org/anthology/N16-1030.

Lappalainen, H. and A. Honkela (2000). "Bayesian non-linear independent component analysis by multi-layer perceptrons". *Advances in independent component analysis*. Springer, pp. 93–121.

Le, Q. and T. Mikolov (2014). "Distributed representations of sentences and documents". *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1188–1196.

LeCun, Y. et al. (1990). "Handwritten digit recognition with a back-propagation network". *Advances in neural information processing systems*, pp. 396–404.

Lin, C.-Y. (2004). "Rouge: A package for automatic evaluation of summaries". *Text Summarization Branches Out: Proc. of the ACL-04 Workshop*, pp. 74–81.

Lin, H. and J. Bilmes (2011). "A Class of Submodular Functions for Document Summarization." *ACL*.

Lowe, D. G. (1999). "Object recognition from local scale-invariant features". *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee, pp. 1150–1157.

Luong, M.-T. and C. D. Manning (2016). "Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models". *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. Berlin, Germany: Association for Computational Linguistics, pp. 1054–1063. URL: http://www.aclweb.org/anthology/P16-1100.

Ma, X. and E. Hovy (2016). "End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF". *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. Berlin, Germany: Association for Computational Linguistics, pp. 1064–1074. URL: http://www.aclweb.org/anthology/P16-1101.

Manning, C. D. and H. Schütze (1999). *Foundations of statistical natural language processing*. MIT Press.

Mihalcea, R. and P. Tarau (2004). "TextRank: Bringing order into texts". *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP)*. Vol. 4.

Mikolov, T., K. Chen, et al. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781*.

Mikolov, T., I. Sutskever, et al. (2013). "Distributed representations of words and phrases and their compositionality". *Advances in Neural Information Processing Systems*, pp. 3111–3119.

Mnih, V. et al. (2013). "Playing Atari With Deep Reinforcement Learning". *NIPS Deep Learning Workshop*.

Mogren, O. (2016). "C-RNN-GAN: Continuous recurrent neural networks with adversarial training". *NIPS Workshop on Constructive Machine Learning (CML)*.

Mogren, O. and R. Johansson (2017). "Character-based recurrent neural networks for morphological relational reasoning". *Submitted draft. Early version published at the EMNLP Workshop on Subword and Character-level Models in NLP*.

Mogren, O., M. Kågebäck, and D. Dubhashi (2015). "Extractive summarization by aggregating multiple similarities". *Proceedings of Recent Advances in Natural Language Processing*, pp. 451–457.

Montúfar, G. F. and J. Morton (2015). When does a mixture of products contain a product of mixtures? *SIAM Journal on Discrete Mathematics* **29**.1, 321–347.

Muhammad, A. S., P. Damaschke, and O. Mogren (2016). "Summarizing online user reviews using bicliques". *Proceedings of The 42nd International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM, LNCS*.

Pascanu, R., G. Montufar, and Y. Bengio (2013). On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098*.

Pennington, J., R. Socher, and C. D. Manning (2014). Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP)* **12**, 1532–1543.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks* **12**.1, 145–151.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning* **1**.1, 81–106.

Radev, D. R. et al. (2004). Centroid-based summarization of multiple documents. *Information Processing & Management* **40**.6, 919–938.

Ratinov, L. and D. Roth (2009). "Design challenges and misconceptions in named entity recognition". *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, pp. 147–155.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning representations by back-propagating errors. *Nature* 323, 533–536.

Schmidhuber, J. and S. Hochreiter (1997). Long short-term memory. *Neural computation* **7**.8, 1735–1780.

Schütze, H. (1993). "Word space". *Advances in neural information processing systems*, pp. 895–902.

Sennrich, R., B. Haddow, and A. Birch (2016). "Neural Machine Translation of Rare Words with Subword Units". *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. Berlin, Germany: Association for Computational Linguistics, pp. 1715–1725. URL: `http://www.aclweb.org/anthology/P16-1162`.

Silver, D. et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature* **529**.7587, 484–489.

Skeppstedt, M. et al. (2014). Automatic recognition of disorders, findings, pharmaceuticals and body structures from clinical text: An annotation and machine learning study. *Journal of Biomedical Informatics* **49**, 148–158.

Socher, R., J. Bauer, et al. (2013). "Parsing with Compositional Vector Grammars". *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 455–465. URL: http://www.aclweb.org/anthology/P13-1045.

Socher, R., E. H. Huang, et al. (2011). "Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection." *Advances in Neural Information Processing Systems*. Vol. 24, pp. 801–809.

Socher, R., A. Perelygin, et al. (2013). "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, pp. 1631–1642. URL: http://www.aclweb.org/anthology/D13-1170.

Sutskever, I., O. Vinyals, and Q. V. Le (2014). "Sequence to sequence learning with neural networks". *Advances in neural information processing systems*, pp. 3104–3112.

Szegedy, C. et al. (2015). "Going deeper with convolutions". *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Tahmasebi, N. et al. (2015). Visions and open challenges for a knowledge-based culturomics. *International Journal on Digital Libraries* **15**.2-4, 169–187.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind* **59**.236, 433–460.

Wang, T. and K. Cho (2016). "Larger-Context Language Modelling with Recurrent Neural Network". *Proceedings of the 54th Annual Meeting of the Association for Computational Linguisticsi*. Berlin, Germany: Association for Computational Linguistics, pp. 1319–1329. URL: http://www.aclweb.org/anthology/P16-1125.

Wang, Y. and J. Patrick (2009). "Cascading classifiers for named entity recognition in clinical notes". *Proceedings of the workshop on biomedical information extraction*. Association for Computational Linguistics, pp. 42–49.

Xu, K. et al. (2015). "Show, attend and tell: Neural image caption generation with visual attention". *International Conference on Machine Learning*, pp. 2048–2057.

Zeiler, M. D. and R. Fergus (2014). "Visualizing and understanding convolutional networks". *European conference on computer vision*. Springer, pp. 818–833.

Zhang, S. and N. Elhadad (2013). Unsupervised biomedical named entity recognition: Experiments with clinical and biological texts. *Journal of biomedical informatics* **46**.6, 1088–1098.

Zhang, X., J. Zhao, and Y. LeCun (2015). "Character-level Convolutional Networks for Text Classification". *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., pp. 649–657. URL: http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf.

Zhou, G. et al. (2004). Recognizing names in biomedical texts: a machine learning approach. *Bioinformatics* **20**.7, 1178–1190.

Part II
Publications

Paper I

Extractive summarization using continuous vector space models

M. Kågebäck, O. Mogren, et al.

***Reprinted from Second Workshop of Continuous Vector Space Models and their Compositionality*, 2014**

Extractive Summarization using Continuous Vector Space Models

Mikael Kågebäck, Olof Mogren, Nina Tahmasebi, Devdatt Dubhashi
Computer Science & Engineering
Chalmers University of Technology
SE-412 96, Göteborg
{kageback, mogren, ninat, dubhashi}@chalmers.se

Abstract

Automatic summarization can help users extract the most important pieces of information from the vast amount of text digitized into electronic form everyday. Central to automatic summarization is the notion of similarity between sentences in text. In this paper we propose the use of continuous vector representations for semantically aware representations of sentences as a basis for measuring similarity. We evaluate different compositions for sentence representation on a standard dataset using the ROUGE evaluation measures. Our experiments show that the evaluated methods improve the performance of a state-of-the-art summarization framework and strongly indicate the benefits of continuous word vector representations for automatic summarization.

1 Introduction

The goal of summarization is to capture the important information contained in large volumes of text, and present it in a brief, representative, and consistent summary. A well written summary can significantly reduce the amount of work needed to digest large amounts of text on a given topic. The creation of summaries is currently a task best handled by humans. However, with the explosion of available textual data, it is no longer financially possible, or feasible, to produce all types of summaries by hand. This is especially true if the subject matter has a narrow base of interest, either due to the number of potential readers or the duration during which it is of general interest. A summary describing the events of World War II might for instance be justified to create manually, while a summary of all reviews and comments regarding a certain version of Windows might not. In such cases, automatic summarization is a way forward.

In this paper we introduce a novel application of continuous vector representations to the problem of multi-document summarization. We evaluate different compositions for producing sentence representations based on two different word embeddings on a standard dataset using the ROUGE evaluation measures. Our experiments show that the evaluated methods improve the performance of a state-of-the-art summarization framework which strongly indicate the benefits of continuous word vector representations for this tasks.

2 Summarization

There are two major types of automatic summarization techniques, extractive and abstractive. *Extractive summarization* systems create summaries using representative sentences chosen from the input while *abstractive summarization* creates new sentences and is generally considered a more difficult problem.

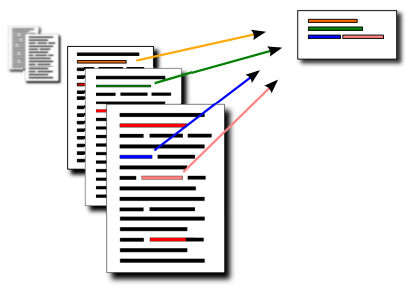


Figure 1: Illustration of Extractive Multi-Document Summarization.

For this paper we consider extractive multi-document summarization, that is, sentences are chosen for inclusion in a summary from a set of documents D. Typically, extractive summarization techniques can be divided into two components, the summarization framework and the similarity measures used to compare sentences. Next

Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC) @ EACL 2014, pages 31–39,
Gothenburg, Sweden, April 26-30 2014.

we present the algorithm used for the framework and in Sec. 2.2 we discuss a typical sentence similarity measure, later to be used as a baseline.

2.1 Submodular Optimization

Lin and Bilmes (2011) formulated the problem of extractive summarization as an optimization problem using monotone nondecreasing submodular set functions. A submodular function F on the set of sentences V satisfies the following property: for any $A \subseteq B \subseteq V \backslash \{v\}$, $F(A + \{v\}) - F(A) \geq F(B + \{v\}) - F(B)$ where $v \in V$. This is called the diminishing returns property and captures the intuition that adding a sentence to a small set of sentences (i.e., summary) makes a greater contribution than adding a sentence to a larger set. The aim is then to find a summary that maximizes diversity of the sentences and the coverage of the input text. This objective function can be formulated as follows:

$$\mathcal{F}(S) = \mathcal{L}(S) + \lambda \mathcal{R}(S)$$

where S is the summary, $\mathcal{L}(S)$ is the coverage of the input text, $\mathcal{R}(S)$ is a diversity reward function. The λ is a trade-off coefficient that allows us to define the importance of coverage versus diversity of the summary. In general, this kind of optimization problem is NP-hard, however, if the objective function is submodular there is a fast scalable algorithm that returns an approximation with a guarantee. In the work of Lin and Bilmes (2011) a simple submodular function is chosen:

$$\mathcal{L}(S) = \sum_{i \in V} \min\{\sum_{j \in S} \mathrm{Sim}(i,j), \alpha \sum_{j \in V} \mathrm{Sim}(i,j)\} \tag{1}$$

The first argument measures similarity between sentence i and the summary S, while the second argument measures similarity between sentence i and the rest of the input V. $\mathrm{Sim}(i,j)$ is the similarity between sentence i and sentence j and $0 \leq \alpha \leq 1$ is a threshold coefficient. The diversity reward function $\mathcal{R}(S)$ can be found in (Lin and Bilmes, 2011).

2.2 Traditional Similarity Measure

Central to most extractive summarization systems is the use of sentence similarity measures ($\mathrm{Sim}(i,j)$ in Eq. 1). Lin and Bilmes measure similarity between sentences by representing each sentence using *tf-idf* (Salton and McGill, 1986) vectors and measuring the cosine angle between vectors. Each sentence is represented by a word vector $\mathbf{w} = (w_1, \ldots, w_N)$ where N is the size of the vocabulary. Weights w_{ki} correspond to the *tf-idf* value of word k in the sentence i. The weights $\mathrm{Sim}(i,j)$ used in the $\mathcal{L}$ function in Eq. 1 are found using the following similarity measure.

$$\mathrm{Sim}(i,j) = \frac{\sum_{w \in i} \mathrm{tf}_{w,i} \times \mathrm{tf}_{w,j} \times \mathrm{idf}_w^2}{\sqrt{\sum_{w \in i} \mathrm{tf}_{w,i}^2 \times \mathrm{idf}_w^2} \sqrt{\sum_{w \in j} \mathrm{tf}_{w,j}^2 \times \mathrm{idf}_w^2}} \tag{2}$$

where $\mathrm{tf}_{w,i}$ and $\mathrm{tf}_{w,j}$ are the number of occurrences of w in sentence i and j, and idf_w is the inverse document frequency (*idf*) of w.

In order to have a high similarity between sentences using the above measure, two sentences must have an overlap of highly scored *tf-idf* words. The overlap must be exact to count towards the similarity, e.g, the terms *The US President* and *Barack Obama* in different sentences will not add towards the similarity of the sentences. To capture deeper similarity, in this paper we will investigate the use of continuous vector representations for measuring similarity between sentences. In the next sections we will describe the basics needed for creating continuous vector representations and methods used to create sentence representations that can be used to measure sentence similarity.

3 Background on Deep Learning

Deep learning (Hinton et al., 2006; Bengio, 2009) is a modern interpretation of artificial neural networks (ANN), with an emphasis on deep network architectures. Deep learning can be used for challenging problems like image and speech recognition (Krizhevsky et al., 2012; Graves et al., 2013), as well as language modeling (Mikolov et al., 2010), and in all cases, able to achieve state-of-the-art results.

Inspired by the brain, ANNs use a neuron-like construction as their primary computational unit. The behavior of a neuron is entirely controlled by its input weights. Hence, the weights are where the information learned by the neuron is stored. More precisely the output of a neuron is computed as the weighted sum of its inputs, and squeezed into the interval $[0, 1]$ using a sigmoid function:

$$y_i = g(\theta_i^T \mathbf{x}) \tag{3}$$

$$g(z) = \frac{1}{1 + e^{-z}} \tag{4}$$

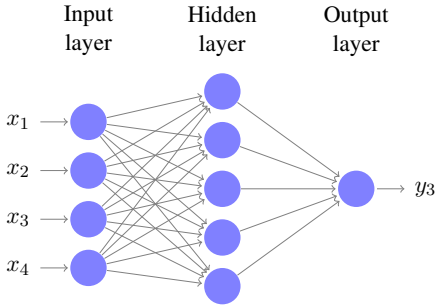


Figure 2: FFNN with four input neurons, one hidden layer, and 1 output neuron. This type of architecture is appropriate for binary classification of some data $\mathbf{x} \in \mathbb{R}^4$, however depending on the complexity of the input, the number and size of the hidden layers should be scaled accordingly.

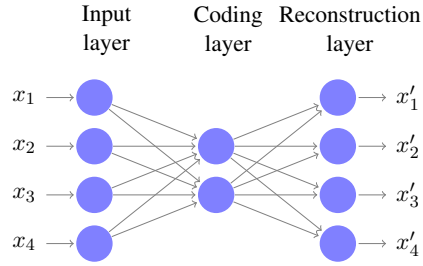


Figure 3: The figure shows an auto-encoder that compresses four dimensional data into a two dimensional code. This is achieved by using a bottleneck layer, referred to as a coding layer.

where θ_i are the weights associated with neuron i and $\mathbf{x}$ is the input. Here the sigmoid function (g) is chosen to be the logistic function, but it may also be modeled using other sigmoid shaped functions, e.g. the hyperbolic tangent function.

The neurons can be organized in many different ways. In some architectures, loops are permitted. These are referred to as *recurrent neural networks*. However, all networks considered here are non-cyclic topologies. In the rest of this section we discuss a few general architectures in more detail, which will later be employed in the evaluated models.

3.1 Feed Forward Neural Network

A feed forward neural network (FFNN) (Haykin, 2009) is a type of ANN where the neurons are structured in layers, and only connections to subsequent layers are allowed, see Fig 2. The algorithm is similar to logistic regression using non-linear terms. However, it does not rely on the user to choose the non-linear terms needed to fit the data, making it more adaptable to changing datasets. The first layer in a FFNN is called the input layer, the last layer is called the output layer, and the interim layers are called hidden layers. The hidden layers are optional but necessary to fit complex patterns.

Training is achieved by minimizing the network error (E). How E is defined differs between different network architectures, but is in general a differentiable function of the produced output and the expected output. In order to minimize this function the gradient $\frac{\partial E}{\partial \Theta}$ first needs to be calculated, where Θ is a matrix of all parameters, or weights, in the network. This is achieved using backpropagation (Rumelhart et al., 1986). Secondly, these gradients are used to minimize E using e.g. gradient descent. The result of this processes is a set of weights that enables the network to do the desired input-output mapping, as defined by the training data.

3.2 Auto-Encoder

An auto-encoder (AE) (Hinton and Salakhutdinov, 2006), see Fig. 3, is a type of FFNN with a topology designed for dimensionality reduction. The input and the output layers in an AE are identical, and there is at least one hidden bottleneck layer that is referred to as the coding layer. The network is trained to reconstruct the input data, and if it succeeds this implies that all information in the data is necessarily contained in the compressed representation of the coding layer.

A shallow AE, i.e. an AE with no extra hidden layers, will produce a similar code as principal component analysis. However, if more layers are added, before and after the coding layer, non-linear manifolds can be found. This enables the network to compress complex data, with minimal loss of information.

3.3 Recursive Neural Network

A recursive neural network (RvNN), see Fig. 4, first presented by Socher et al. (2010), is a type of feed forward neural network that can process data through an arbitrary binary tree structure, e.g. a

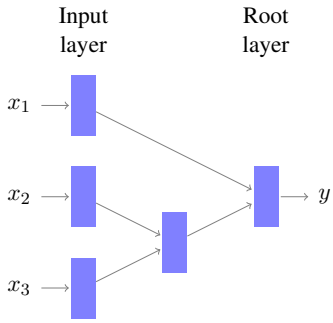


Figure 4: The recursive neural network architecture makes it possible to handle variable length input data. By using the same dimensionality for all layers, arbitrary binary tree structures can be recursively processed.

binary parse tree produced by linguistic parsing of a sentence. This is achieved by enforcing weight constraints across all nodes and restricting the output of each node to have the same dimensionality as its children.

The input data is placed in the leaf nodes of the tree, and the structure of this tree is used to guide the recursion up to the root node. A compressed representation is calculated recursively at each non-terminal node in the tree, using the same weight matrix at each node. More precisely, the following formulas can be used:

$$z_p = \theta_p^T [x_l; x_r] \tag{5a}$$

$$y_p = g(z_p) \tag{5b}$$

where y_p is the computed parent state of neuron p, and z_p the induced field for the same neuron. $[x_l; x_r]$ is the concatenation of the state belonging to the right and left sibling nodes. This process results in a fixed length representation for hierarchical data of arbitrary length. Training of the model is done using backpropagation through structure, introduced by Goller and Kuchler (1996).

4 Word Embeddings

Continuous distributed vector representation of words, also referred to as word embeddings, was first introduced by Bengio et al. (2003). A word embedding is a continuous vector representation that captures semantic and syntactic information about a word. These representations can be used to unveil dimensions of similarity between words, e.g. singular or plural.

4.1 Collobert & Weston

Collobert and Weston (2008) introduce an efficient method for computing word embeddings, in this work referred to as *CW* vectors. This is achieved firstly, by scoring a valid n-gram ($\mathbf{x}$) and a corrupted n-gram ($\bar{\mathbf{x}}$) (where the center word has been randomly chosen), and secondly, by training the network to distinguish between these two n-grams. This is done by minimizing the hinge loss

$$\max(0, 1 - s(\mathbf{x}) + s(\bar{\mathbf{x}})) \tag{6}$$

where s is the scoring function, i.e. the output of a FFNN that maps between the word embeddings of an n-gram to a real valued score. Both the parameters of the scoring function and the word embeddings are learned in parallel using backpropagation.

4.2 Continuous Skip-gram

A second method for computing word embeddings is the Continuous Skip-gram model, see Fig. 5, introduced by Mikolov et al. (2013a). This model is used in the implementation of their word embeddings tool *Word2Vec*. The model is trained to predict the context surrounding a given word. This is accomplished by maximizing the objective function

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t) \tag{7}$$

where T is the number of words in the training set, and c is the length of the training context. The probability $p(w_{t+j}|w_t)$ is approximated using the hierarchical softmax introduced by Bengio et al. (2002) and evaluated in a paper by Morin and Bengio (2005).

5 Phrase Embeddings

Word embeddings have proven useful in many natural language processing (NLP) tasks. For summarization, however, sentences need to be compared. In this section we present two different methods for deriving phrase embeddings, which in Section 5.3 will be used to compute sentence to sentence similarities.

5.1 Vector addition

The simplest way to represent a sentence is to consider it as the sum of all words without regarding word orders. This was considered by

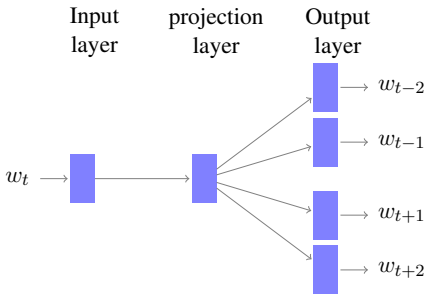


Figure 5: The continuous Skip-gram model. Using the input word (w_t) the model tries to predict which words will be in its context ($w_{t\pm c}$).

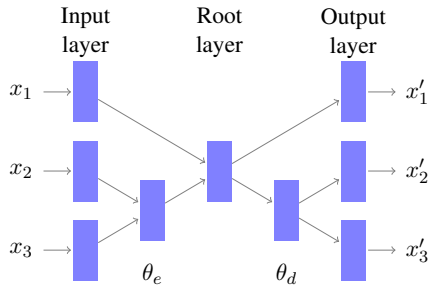


Figure 6: The structure of an unfolding RAE, on a three word phrase ($[x_1, x_2, x_3]$). The weight matrix θ_e is used to encode the compressed representations, while θ_d is used to decode the representations and reconstruct the sentence.

Mikolov et al. (2013b) for representing short phrases. The model is expressed by the following equation:

$$\mathbf{x}_p = \sum_{\mathbf{x}_w \in \{\text{sentence}\}} \mathbf{x}_w \tag{8}$$

where x_p is a phrase embedding, and x_w is a word embedding. We use this method for computing phrase embeddings as a baseline in our experiments.

5.2 Unfolding Recursive Auto-encoder

The second model is more sophisticated, taking into account also the order of the words and the grammar used. An unfolding recursive auto-encoder (RAE) is used to derive the phrase embedding on the basis of a binary parse tree. The unfolding RAE was introduced by Socher et al. (2011) and uses two RvNNs, one for encoding the compressed representations, and one for decoding them to recover the original sentence, see Figure 6. The network is subsequently trained by minimizing the reconstruction error.

Forward propagation in the network is done by recursively applying Eq. 5a and 5b for each triplet in the tree in two phases. First, starting at the center node (root of the tree) and recursively pulling the data from the input. Second, again starting at the center node, recursively pushing the data towards the output. Backpropagation is done in a similar manner using backpropagation through structure (Goller and Kuchler, 1996).

5.3 Measuring Similarity

Phrase embeddings provide semantically aware representations for sentences. For summarization, we need to measure the similarity between two representations and will make use of the following two vector similarity measures. The first similarity measure is the cosine similarity, transformed to the interval of $[0, 1]$

$$\text{Sim}(i,j) = \left(\frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_j\| \|\mathbf{x}_j\|} + 1 \right) /2 \tag{9}$$

where $\mathbf{x}$ denotes a phrase embedding The second similarity is based on the complement of the Euclidean distance and computed as:

$$\text{Sim}(i,j) = 1 - \frac{1}{\max\limits_{k,n} \sqrt{\| \mathbf{x}_k - \mathbf{x}_n \|^2}} \sqrt{\| \mathbf{x}_j - \mathbf{x}_i \|^2} \tag{10}$$

6 Experiments

In order to evaluate phrase embeddings for summarization we conduct several experiments and compare different phrase embeddings with *tf-idf* based vectors.

6.1 Experimental Settings

Seven different configuration were evaluated. The first configuration provides us with a baseline and is denoted *Original* for the Lin-Bilmes method described in Sec. 2.1. The remaining configurations comprise selected combinations of word embeddings, phrase embeddings, and similarity measures.

The first group of configurations are based on vector addition using both Word2Vec and CW vectors. These vectors are subsequently compared using both cosine similarity and Euclidean distance. The second group of configurations are built upon recursive auto-encoders using CW vectors and are also compared using cosine similarity as well as Euclidean distance.

The methods are named according to: VectorType_EmbeddingMethod$_{SimilarityMethod}$, e.g. W2V_Add$_{Cos}$ for Word2Vec vectors combined using vector addition and compared using cosine similarity.

To get an upper bound for each ROUGE score an exhaustive search were performed, where each possible pair of sentences were evaluated, and maximized w.r.t the ROUGE score.

6.2 Dataset and Evaluation

The Opinosis dataset (Ganesan et al., 2010) consists of short user reviews in 51 different topics. Each of these topics contains between 50 and 575 sentences and are a collection of user reviews made by different authors about a certain characteristic of a hotel, car or a product (e.g. *"Location of Holiday Inn, London"* and *"Fonts, Amazon Kindle"*). The dataset is well suited for multi-document summarization (each sentence is considered its own document), and includes between 4 and 5 gold-standard summaries (not sentences chosen from the documents) created by human authors for each topic.

Each summary is evaluated with ROUGE, that works by counting word overlaps between generated summaries and gold standard summaries. Our results include R-1, R-2, and R-SU4, which counts matches in unigrams, bigrams, and skip-bigrams respectively. The skip-bigrams allow four words in between (Lin, 2004).

The measures reported are recall (R), precision (P), and F-score (F), computed for each topic individually and averaged. Recall measures what fraction of a human created gold standard summary that is captured, and precision measures what fraction of the generated summary that is in the gold standard. F-score is a standard way to combine recall and precision, computed as $F = 2\frac{P*R}{P+R}$.

6.3 Implementation

All results were obtained by running an implementation of Lin-Bilmes submodular optimization summarizer, as described in Sec. 2.1. Also, we have chosen to fix the length of the summaries to two sentences because the length of the gold-standard summaries are typically around two sentences. The CW vectors used were trained by Turian et al. (2010)[1], and the Word2Vec vectors by Mikolov et al. (2013b)[2]. The unfolding RAE used is based on the implementation by Socher et al. (2011)[3], and the parse trees for guiding the recursion was generated using the Stanford Parser (Klein and Manning, 2003)[4].

6.4 Results

The results from the ROUGE evaluation are compiled in Table 1. We find for all measures (recall, precision, and F-score), that the phrase embeddings outperform the original Lin-Bilmes. For recall, we find that CW_Add$_{Cos}$ achieves the highest result, while for precision and F-score the CW_Add$_{Euc}$ perform best. These results are consistent for all versions of ROUGE scores reported (1, 2 and SU4), providing a strong indication for phrase embeddings in the context of automatic summarization.

Unfolding RAE on CW vectors and vector addition on W2V vectors gave comparable results w.r.t. each other, generally performing better than original Linn-Bilmes but not performing as well as vector addition of CW vectors.

The results denoted OPT in Table 1 describe the upper bound score, where each row represents optimal recall and F-score respectively. The best results are achieved for R-1 with a maximum recall of 57.86%. This is a consequence of hand created gold standard summaries used in the evaluation, that is, we cannot achieve full recall or F-score when the sentences in the gold standard summaries are not taken from the underlying documents and thus, they can never be fully matched using extractive summarization. R-2 and SU4 have lower maximum recall and F-score, with 22.9% and 29.5% respectively.

6.5 Discussion

The results of this paper show great potential for employing word and phrase embeddings in summarization. We believe that by using embeddings we move towards more semantically aware summarization systems. In the future, we anticipate

[1] http://metaoptimize.com/projects/wordreprs/
[2] https://code.google.com/p/word2vec/
[3] http://nlp.stanford.edu/ socherr/codeRAEVectorsNIPS2011.zip
[4] http://nlp.stanford.edu/software/lex-parser.shtml

Table 1: ROUGE scores for summaries using different similarity measures. OPT constitutes the optimal ROUGE scores on this dataset.

ROUGE-1			
	R	**P**	**F**
OPT_R	**57.86**	21.96	30.28
OPT_F	45.93	48.84	**46.57**
CW_RAE_{Cos}	27.37	19.89	22.00
CW_RAE_{Euc}	29.25	19.77	22.62
CW_Add_{Cos}	**34.72**	11.75	17.16
CW_Add_{Euc}	29.12	**22.75**	**24.88**
$W2V\_Add_{Cos}$	30.86	16.81	20.93
$W2V\_Add_{Euc}$	28.71	16.67	20.75
Original	25.82	19.58	20.57
ROUGE-2			
	R	**P**	**F**
OPT_R	**22.96**	12.31	15.33
OPT_F	20.42	19.94	**19.49**
CW_RAE_{Cos}	4.68	3.18	3.58
CW_RAE_{Euc}	4.82	3.24	3.67
CW_Add_{Cos}	**5.89**	1.81	2.71
CW_Add_{Euc}	5.12	**3.60**	**4.10**
$W2V\_Add_{Cos}$	5.71	3.08	3.82
$W2V\_Add_{Euc}$	3.86	1.95	2.54
Original	3.92	2.50	2.87
ROUGE-SU4			
	R	**P**	**F**
OPT_R	**29.50**	13.53	17.70
OPT_F	23.17	26.50	**23.70**
CW_RAE_{Cos}	9.61	6.23	6.95
CW_RAE_{Euc}	9.95	6.17	7.04
CW_Add_{Cos}	**12.38**	3.27	5.03
CW_Add_{Euc}	10.54	**7.59**	**8.35**
$W2V\_Add_{Cos}$	11.94	5.52	7.12
$W2V\_Add_{Euc}$	9.78	4.69	6.15
Original	9.15	6.74	6.73

improvements for the field of automatic summarization as the quality of the word vectors improve and we find enhanced ways of composing and comparing the vectors.

It is interesting to compare the results of different composition techniques on the CW vectors, where vector addition surprisingly outperforms the considerably more sophisticated unfolding RAE. However, since the unfolding RAE uses syntactic information, this may be a result of using a dataset consisting of low quality text.

In the interest of comparing word embeddings, results using vector addition and cosine similarity were computed based on both CW and Word2Vec vectors. Supported by the achieved results CW vectors seems better suited for sentence similarities in this setting.

An issue we encountered with using precomputed word embeddings was their limited vocabulary, in particular missing uncommon (or common incorrect) spellings. This problem is particularly pronounced on the evaluated Opinosis dataset, since the text is of low quality. Future work is to train word embeddings on a dataset used for summarization to better capture the specific semantics and vocabulary.

The optimal R-1 scores are higher than R-2 and SU4 (see Table 1) most likely because the score ignores word order and considers each sentence as a set of words. We come closest to the optimal score for R-1, where we achieve 60% of maximal recall and 49% of F-score. Future work is to investigate why we achieve a much lower recall and F-score for the other ROUGE scores.

Our results suggest that the phrase embeddings capture the kind of information that is needed for the summarization task. The embeddings are the underpinnings of the decisions on which sentences that are representative of the whole input text, and which sentences that would be redundant when combined in a summary. However, the fact that we at most achieve 60% of maximal recall suggests that the phrase embeddings are not complete w.r.t summarization and might benefit from being combined with other similarity measures that can capture complementary information, for example using multiple kernel learning.

7 Related Work

To the best of our knowledge, continuous vector space models have not previously been used in summarization tasks. Therefore, we split this section in two, handling summarization and continuous vector space models separately.

7.1 Continuous Vector Space Models

Continuous distributed vector representation of words was first introduced by Bengio et al. (2003).

They employ a FFNN, using a window of words as input, and train the model to predict the next word. This is computed using a big softmax layer that calculate the probabilities for each word in the vocabulary. This type of exhaustive estimation is necessary in some NLP applications, but makes the model heavy to train.

If the sole purpose of the model is to derive word embeddings this can be exploited by using a much lighter output layer. This was suggested by Collobert and Weston (2008), which swapped the heavy softmax against a hinge loss function. The model works by scoring a set of consecutive words, distorting one of the words, scoring the distorted set, and finally training the network to give the correct set a higher score.

Taking the lighter concept even further, Mikolov et al. (2013a) introduced a model called Continuous Skip-gram. This model is trained to predict the context surrounding a given word using a shallow neural network. The model is less aware of the order of words, than the previously mentioned models, but can be trained efficiently on considerably larger datasets.

An early attempt at merging word representations into representations for phrases and sentences is introduced by Socher et al. (2010). The authors present a recursive neural network architecture (RvNN) that is able to jointly learn parsing and phrase/sentence representation. Though not able to achieve state-of-the-art results, the method provides an interesting path forward. The model uses one neural network to derive all merged representations, applied recursively in a binary parse tree. This makes the model fast and easy to train but requires labeled data for training.

7.2 Summarization Techniques

Radev et al. (2004) pioneered the use of cluster centroids in their work with the idea to group, in the same cluster, those sentences which are highly similar to each other, thus generating a number of clusters. To measure the similarity between a pair of sentences, the authors use the cosine similarity measure where sentences are represented as weighted vectors of *tf-idf* terms. Once sentences are clustered, sentence selection is performed by selecting a subset of sentences from each cluster.

In TextRank (2004), a document is represented as a graph where each sentence is denoted by a vertex and pairwise similarities between sentences are represented by edges with a weight corresponding to the similarity between the sentences. The Google PageRank ranking algorithm is used to estimate the importance of different sentences and the most important sentences are chosen for inclusion in the summary.

Bonzanini, Martinez, Roelleke (2013) presented an algorithm that starts with the set of all sentences in the summary and then iteratively chooses sentences that are unimportant and removes them. The sentence removal algorithm obtained good results on the Opinosis dataset, in particular w.r.t F-scores.

We have chosen to compare our work with that of Lin and Bilmes (2011), described in Sec. 2.1. Future work is to make an exhaustive comparison using a larger set similarity measures and summarization frameworks.

8 Conclusions

We investigated the effects of using phrase embeddings for summarization, and showed that these can significantly improve the performance of the state-of-the-art summarization method introduced by Lin and Bilmes in (2011). Two implementations of word vectors and two different approaches for composition where evaluated. All investigated combinations improved the original Lin-Bilmes approach (using *tf-idf* representations of sentences) for at least two ROUGE scores, and top results where found using vector addition on CW vectors.

In order to further investigate the applicability of continuous vector representations for summarization, in future work we plan to try other summarization methods. In particular we will use a method based on multiple kernel learning were phrase embeddings can be combined with other similarity measures. Furthermore, we aim to use a novel method for sentence representation similar to the RAE using multiplicative connections controlled by the local context in the sentence.

Acknowledgments

The authors would like to acknowledge the project *Towards a knowledge-based culturomics* supported by a framework grant from the Swedish Research Council (2012–2016; dnr 2012-5738), and the project *Data-driven secure business intelligence* grant IIS11-0089 from the Swedish Foundation for Strategic Research (SSF).

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Yoshua Bengio. 2002. New distributed probabilistic language models. Technical Report 1215, Département d'informatique et recherche opérationnelle, Université de Montréal.

Yoshua Bengio. 2009. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127.

Marco Bonzanini, Miguel Martinez-Alvarez, and Thomas Roelleke. 2013. Extractive summarisation via sentence removal: Condensing relevant sentences into a short summary. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 893–896. ACM.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Kavita Ganesan, ChengXiang Zhai, and Jiawei Han. 2010. Opinosis: a graph-based approach to abstractive summarization of highly redundant opinions. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 340–348. ACL.

Christoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *IEEE International Conference on Neural Networks*, volume 1, pages 347–352. IEEE.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. *arXiv preprint arXiv:1303.5778*.

S.S. Haykin. 2009. *Neural Networks and Learning Machines*. Number v. 10 in Neural networks and learning machines. Prentice Hall.

Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Dan Klein and Christopher D Manning. 2003. Fast exact inference with a factored model for natural language parsing. *Advances in neural information processing systems*, pages 3–10.

Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1106–1114.

Hui Lin and Jeff Bilmes. 2011. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 510–520. ACL.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81.

Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing order into texts. In *Proceedings of EMNLP*, volume 4. Barcelona, Spain.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *ArXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *AISTATS'05*, pages 246–252.

Dragomir R Radev, Hongyan Jing, Małgorzata Styś, and Daniel Tam. 2004. Centroid-based summarization of multiple documents. *Information Processing & Management*, 40(6):919–938.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Gerard Salton and Michael J. McGill. 1986. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA.

Richard Socher, Christopher D Manning, and Andrew Y Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. 2011. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In *Advances in Neural Information Processing Systems 24*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394. ACL.

Paper II

Extractive summarization by aggregating multiple similarities

O. Mogren, M. Kågebäck, and D. Dubhashi

Reprinted from Proceedings of Recent Advances in Natural Language Processing, **2015**

Extractive Summarization by Aggregating Multiple Similarities

Olof Mogren, Mikael Kågebäck, Devdatt Dubhashi
Department of Computer Science and Engineering
Chalmers University of Technology,
412 96 Göteborg, Sweden
mogren@chalmers.se

Abstract

News reports, social media streams, blogs, digitized archives and books are part of a plethora of reading sources that people face every day. This raises the question of how to best generate automatic summaries. Many existing methods for extracting summaries rely on comparing the similarity of two sentences in some way. We present new ways of measuring this similarity, based on sentiment analysis and continuous vector space representations, and show that combining these together with similarity measures from existing methods, helps to create better summaries. The finding is demonstrated with MULTSUM, a novel summarization method that uses ideas from kernel methods to combine sentence similarity measures. Submodular optimization is then used to produce summaries that take several different similarity measures into account. Our method improves over the state-of-the-art on standard benchmark datasets; it is also fast and scale to large document collections, and the results are statistically significant.

1 Introduction

Extractive summarization, the process of selecting a subset of sentences from a set of documents, is an important component of modern information retrieval systems (Baeza-Yates et al., 1999). A good summarization system needs to balance two complementary aspects: finding a summary that captures all the important topics of the documents (*coverage*), yet does not contain too many similar sentences (*non-redundancy*). It follows that it is essential to have a good way of measuring the similarity of sentences, in no way a trivial task. Consequently, several measures for sentence similarity have been explored for extractive summarization.

In this work, two sets of novel similarity measures capturing deeper semantic features are presented, and evaluated in combination with existing methods of measuring sentence similarity. The new methods are based on sentiment analysis, and continuous vector space representations of phrases, respectively.

We show that summary quality is improved by combining multiple similarities at the same time using kernel techniques. This is demonstrated using MULTSUM, an ensemble-approach to generic extractive multi-document summarization based on the existing, state-of-the-art method of Lin and Bilmes (2011). Our method obtains state-of-the-art results that are statistically significant on the de-facto standard benchmark dataset DUC 04. The experimental evaluation also confirm that the method generalizes well to other datasets.

2 MULTSUM

MULTSUM, our approach for extractive summarization, finds representative summaries taking multiple sentence similarity measures into account. As Lin and Bilmes (2011), we formulate the problem as the optimization of monotone non-decreasing submodular set functions. This results in a fast, greedy optimization step that provides a $(1 - \frac{1}{e})$ factor approximation. In the original version, the optimization objective is a function scoring a candidate summary by coverage and diversity, expressed using cosine similarity between sentences represented as bag-of-terms vectors. We extend this method by using several sentence similarity measures M^l (as described in Section 3) at the same time, combined by multiplying them together element-wise:

$$M_{\mathbf{s}_i,\mathbf{s}_j} = \prod M^l_{\mathbf{s}_i,\mathbf{s}_j}.$$

In the literature of kernel methods, this is the standard way of combining kernels as a conjunction (Duvenaud, 2014; Schölkopf et al., 2004, Ch 1).

3 Sentence Similarity Measures

Many existing systems rely on measuring the similarity of sentences to balance the coverage with the amount of redundancy of the summary. This is also true for MULTSUM which is based on the existing submodular optimization method. Similarity measures that capture general aspects lets the summarization system pick sentences that are representative and diverse in general. Similarity measures capturing more specific aspects allow the summarization system to take these aspects into account.

We list some existing measures in Table 3 (that mainly relies on counting word overlaps) and in Sections 3.1 and 3.2, we present sentence similarity measures that capture more specific aspects of the text. MULTSUM is designed to work with all measures mentioned below; this will be evaluated in Section 4. Interested readers are referred to a survey of existing similarity measures from the litterature in (Bengtsson and Skeppstedt, 2012). All these similarity measures require sentence splitting, tokenization, part-of-speech tagging and stemming of words. The Filtered Word, and TextRank comparers are set similarity measures where each sentence is represented by the set of all their terms. The KeyWord comparer and LinTFIDF represent each sentence as a word vector and uses the vectors for measuring similarity.

DepGraph first computes the dependency parse trees of the two sentences using Maltparser (Nivre, 2003). The length of their longest common path is then used to derive the similarity score.

The similarity measure used in TextRank (Mihalcea and Tarau, 2004) will be referred to as TRComparer. The measure used in submodular optimization (Lin and Bilmes, 2011) will be referred to as LinTFIDF. All measures used in this work are normalized, $M_{\mathbf{s}_i,\mathbf{s}_j} \in [0,1]$.

Name	Formula
Filtered	$M_{\mathbf{s}_i,\mathbf{s}_j} = \lvert(\mathbf{s}_i \cap \mathbf{s}_j)\rvert / \sqrt{\lvert(\mathbf{s}_i)\rvert + \lvert(\mathbf{s}_j)\rvert}$
TRCmp.	$M_{\mathbf{s}_i,\mathbf{s}_j} = \lvert\mathbf{s}_i \cap \mathbf{s}_j\rvert / (log\lvert\mathbf{s}_i\rvert + log\lvert\mathbf{s}_j\rvert)$
LinTFIDF	$M_{\mathbf{s}_i,\mathbf{s}_j} = \frac{\sum_{w \in \mathbf{s}_i} tf_{w,i} \cdot tf_{w,j} \cdot idf_w^2}{\sqrt{\sum_{w \in \mathbf{s}_i} tf_{w,\mathbf{s}_i} idf_w^2} \sqrt{\sum_{w \in \mathbf{s}_j} tf_{w,\mathbf{s}_j} idf_w^2}}$
KeyWord	$M_{\mathbf{s}_i,\mathbf{s}_j} = \frac{\sum_{w \in \{\{\mathbf{s}_i \cap \mathbf{s}_j\} \cap K\}} tf_w \cdot idf_w}{\lvert\mathbf{s}_i\rvert + \lvert\mathbf{s}_j\rvert}$
DepGraph	*See text description.*

Table 1: Similarity measures from previous works.

3.1 Sentiment Similarity

Sentiment analysis has previously been used for document summarization, with the aim of capturing an average sentiment of the input corpus (Lerman et al., 2009), or to score emotionally charged sentences (Nishikawa et al., 2010). Other research has shown that negative emotion words appear at a relative higher rate in summaries written by humans (Hong and Nenkova, 2014). We propose a different way of making summaries sentiment aware by comparing the level of sentiment in sentences. This allows for summaries that are both representative and diverse in sentiment.

Two lists, of positive and of negative sentiment words respectively, were manually created[1] and used. Firstly, each sentence $\mathbf{s}_i$ is given two sentiment scores, $positive(\mathbf{s}_i)$ and $negative(\mathbf{s}_i)$, defined as the fraction of words in $\mathbf{s}_i$ that is found in the positive and the negative list, respectively. The similarity score for positive sentiment are computed as follows:

$$M_{\mathbf{s}_i,\mathbf{s}_j} = 1 - |positive(\mathbf{s}_i) - positive(\mathbf{s}_j)|$$

The similarity score for negative sentiment are computed as follows:

$$M_{\mathbf{s}_i,\mathbf{s}_j} = 1 - |negative(\mathbf{s}_i) - negative(\mathbf{s}_j)|$$

3.2 Continuous Vector Space Representations

Continuous vector space representations of words has a long history. Recently, the use of deep learning methods has given rise to a new class of continuous vector space models. Bengio et al. (2006) presented vector space representations for words that capture semantic and syntactic properties. These vectors can be employed not only to find similar words, but also to relate words using multiple dimensions of similarity. This means that words sharing some sense can be related using

[1] To download the sentiment word lists used, please see `http://www.mogren.one/`

translations in vector space, e.g. $v_{king} - v_{man} + v_{woman} \approx v_{queen}$.

Early work on extractive summarization using vector space models was presented in (Kågebäck et al., 2014). In this work we use a similar approach, with two different methods of deriving word embeddings. The first model (CW) was introduced by Collobert and Weston (2008). The second ($W2V$) is the skip-gram model by Mikolov et al. (2013).

The Collobert and Weston vectors were trained on the RCV1 corpus, containing one year of Reuters news wire; the skip-gram vectors were trained on 300 billion words from Google News.

The word embeddings are subsequently used as building blocks for sentence level phrase embeddings by summing the word vectors of each sentence. Finally, the sentence similarity is defined as the cosine similarity between the sentence vectors.

With MULTSUM, these similarity measures can be combined with the traditional sentence similarity measures.

4 Experiments

Our version of the submodular optimization code follows the description by Lin and Bilmes (2011), with the exception that we use multiplicative combinations of the sentence similarity scores described in Section 3. The source code of our system can be downloaded from `http://www.mogren.one/`. Where nothing else is stated, MULTSUM was evaluated with a multiplicative combination of TRComparer and FilteredWordComparer.

4.1 Datasets

In the evaluation, three different datasets were used. DUC 02 and DUC 04 are from the Document Understanding Conferences, both with the settings of task 2 (short multi-document summarization), and each consisting of around 50 document sets. Each document set is comprised of around ten news articles (between 111 and 660 sentences) and accompanied with four gold-standard summaries created by manual summarizers. The summaries are at most 665 characters long. DUC 04 is the de-facto standard benchmark dataset for generic multi-document summarization.

Experiments were also carried out on Opinosis (Ganesan et al., 2010), a collection of short user reviews in 51 different topics. Each topic consists of between 50 and 575 one-sentence user reviews by different authors about a certain characteristic of a hotel, a car, or a product. The dataset includes 4 to 5 gold-standard summaries created by human authors for each topic. The the gold-standard summaries is around 2 sentences.

4.2 Baseline Methods

Our baseline methods are Submodular optimization (Lin and Bilmes, 2011), DPP (Kulesza and Taskar, 2012), and ICSI (Gillick et al., 2008). The baseline scores are calculated on precomputed summary outputs (Hong et al., 2014).

4.3 Evaluation Method

Following standard procedure, we use ROUGE (version 1.5.5) for evaluation (Lin, 2004). ROUGE counts n-gram overlaps between generated summaries and the gold standard. We have concentrated on recall as this is the measure with highest correlation to human judgement (Lin and Hovy, 2003), on ROUGE-1, ROUGE-2, and ROUGE-SU4, representing matches in unigrams, bigrams, and skip-bigrams, respectively.

The Opinosis experiments were aligned with those of Bonzanini et al. (2013) and Ganesan et al. (2010)[2]. Summary length was 2 sentences. In the DUC experiments, summary length is 100 words[3].

5 Results

Our experimental results show significant improvements by aggregating several sentence similarity measures, and our results for ROUGE-2 and ROUGE-SU4 recall beats state–of–the–art.

5.1 Integrating Different Similarity Measures

Table 2 shows ROUGE recall on DUC 04. MULTSUM[4] obtains ROUGE scores beating state-of-the-art systems, in particular on ROUGE-2 and ROUGE-SU4, suggesting that MULTSUM produce summaries with excellent fluency. We also note, that using combined similarities, we beat original submodular optimization.

Figure 5.1 shows, for each $n \in [1..9]$, the highest ROUGE-1 recall score obtained by MULTSUM, determined by exhaustive search

[2]ROUGE options on Opinosis: -a - m -s -x -n 2 -2 4 -u.

[3]ROUGE options on DUC: -a -n 2 -m -l 100 -x -c 95 -r 1000 -f A -p 0.5 -t 0 -2 4 -u.

[4]Here, MULTSUM is using TRComparer and FilteredWordComparer in multiplicative conjunction.

	ROUGE-1	ROUGE-2	ROUGE-SU4
MULTSUM	39.35	**9.94**	**14.01**
ICSISumm	38.41	9.77	13.62
DPP	**39.83**	9.62	13.86
SUBMOD	39.18	9.35	13.75

Table 2: ROUGE recall scores on DUC 04. Our system MULTSUM obtains the best result yet for ROUGE-2 and ROUGE-SU4. DPP has a higher ROUGE-1 score, but the difference is not statistically significant (Hong et al., 2014).

1	**2**	**3**	**4**
1.0	0.00038	0.00016	0.00016

Table 3: p-values from the Mann-Whitney U-test for combinations of similarity measures of size $n \in [1..4]$, compared to using just one similarity measure. Using 2, 3, or 4 similarity measures at the same time with MULTSUM, gives a statistically significant improvement of the ROUGE-1 scores. Dataset: DUC 04.

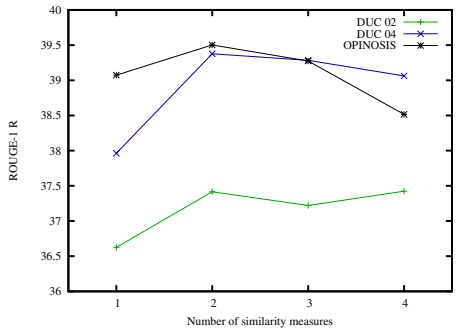


Figure 1: MULTSUM ROUGE-1 recall performance for each top-performing combination of up to four similarity measures. On all datasets, using combinations of two, three, and four similarity measures is better than using only one.

among all possible combinations of size n. The performance increases from using only one sentence similarity measure, reaching a high, stable level when $n \in [2..4]$. The behaviour is consistent over three datasets: DUC 02, DUC 04 and OPINOSIS. Based on ROUGE-1 recall, on DUC 02, a combination of four similarity measures provided the best results, while on DUC 04 and Opinosis, a combination of two similarity scores provided a slightly better score.

Table 3 shows p-values obtained using the Mann-Whitney U-test (Mann et al., 1947) on the ROUGE-1 scores when using a combination of n similarities with MULTSUM, compared to using only one measure. The Mann-Whitney U-test compares two ranked lists A and B, and decides whether they are from the same population. Here, A is the list of scores from using only one measure, and B is the top-10 ranked combinations of n combined similarity measures, $n \in [1..4]$). One can see that for each $n \in [1..4]$, using n sentence similarity measures at the same time, is significantly better than using only one.

On DUC 02, the best combination of similarity measures is using CW, LinTFIDF, NegativeSentiment, and TRComparer. Each point in Figure 5.1 represents a combination of some of these four similarity measures. Let n be the number of measures in such a combination. When $n = 1$, the "combinations" are just single similarity measures. When $n = 2$, there are 6 different ways to choose, and when $n = 3$, there are four. A line goes from each measure point through all combinations the measure is part of. One can clearly see the benefits of each of the combination steps, as n increases.

5.2 Evaluation with Single Similarity Measures

In order to understand the effect of different similarity measures, MULTSUM was first evaluated using only one similarity measure at a time. Table 4 shows the ROUGE recall scores of these experiments, using the similarity measures presented in Section 3, on DUC 04.

We note that MULTSUM provides summaries of high quality already with one similarity measure (e.g. with TRComparer), with a ROUGE-1 recall of 37.95 Using only sentiment analysis as the single similarity measure does not capture enough information to produce state-of-the-art summaries.

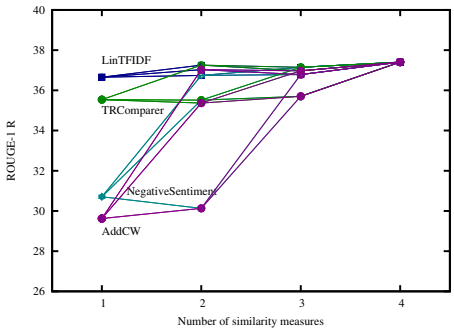


Figure 2: ROUGE-1 recall for the top-performing four-combination on DUC 2002 (CW, LinTFIDF, NegativeSentiment, and TRComparer), and all possible subsets of these four similarity measures. (When the number of similarity measures is one, only a single measure is used).

	ROUGE-1	ROUGE-2	ROUGE-SU4
$TRComparer$	**37.95**	**8.94**	**13.19**
$Filtered$	37.51	8.26	12.73
$LinTFIDF$	35.74	6.50	11.51
$KeyWord$	35.40	7.13	11.80
$DepGraph$	32.81	5.43	10.12
$NegativeSent.$	32.65	6.35	10.29
$PositiveSent.$	31.19	4.87	9.27
$W2V$	32.12	4.94	9.92
CW	31.59	4.74	9.51

Table 4: ROUGE recall of MULTSUM using different similarity measures, one at a time. Dataset: DUC 04. The traditional word-overlap measures are the best scoring when used on their own; the proposed measures with more semantical comparisons provide the best improvements when used in conjunctions.

6 Discussion

Empirical evaluation of the method proposed in this paper shows that using several sentence similarity measures at the same time produces significantly better summaries.

When using one single similarity at a time, using sentiment similarity and vector space models does not give the best summaries. However, we found that when combining several similarity measures, our proposed sentiment and continuous vector space measures often rank among the top ones, together with the TRComparer.

MULTSUM, our novel summarization method, based on submodular optimization, multiplies several sentence similarity measures, to be able to make summaries that are good with regards to several aspects at the same time. Our experimental results show significant improvements when using multiplicative combinations of several sentence similarity measures. In particular, the results of MULTSUM surpasses that of the original submodular optimization method.

In our experiments we found that using between two and four similarity measures lead to significant improvements compared to using a single measure. This verifies the validity of commonly used measures like TextRank and LinTFIDF as well as new directions like phrase embeddings and sentiment analysis.

There are several ideas worth pursuing that could further improve our methods. We will explore methods of incorporating more semantic information in our sentence similarity measures. This could come from systems for Information Extraction (Ji et al., 2013), or incorporating external sources such as WordNet, Freebase and DBpedia (Nenkova and McKeown, 2012).

7 Related Work

Ever since (Luhn, 1958), the field of automatic document summarization has attracted a lot of attention, and has been the focus of a steady flow of research. Luhn was concerned with the importance of words and their representativeness for the input text, an idea that's still central to many current approaches. The development of new techniques for document summarization has since taken many different paths. Some approaches concentrate on what words should appear in summaries, some focus on sentences in part or in whole, and some consider more abstract concepts.

In the 1990's we witnessed the dawn of the data explosion known as the world wide web, and research on multi document summarization took off. Some ten years later, the Document Understanding Conferences (DUC) started providing researchers with datasets and spurred interest with a venue for competition.

Luhn's idea of a frequency threshold measure for selecting topic words in a document has lived on. It was later superseded by $tf \times idf$, which measures the specificity of a word to a document,

The two bombers who carried out Friday's attack, which led the Israeli Cabinet to suspend deliberations on the land-for-security accord signed with the Palestinians last month, were identified as members of Islamic Holy War from West Bank villages under Israeli security control. The radical group Islamic Jihad claimed responsibility Saturday for the market bombing and vowed more attacks to try to block the new peace accord. Israel radio said the 18-member Cabinet debate on the Wye River accord would resume only after Yasser Arafat's Palestinian Authority fulfilled all of its commitments under the agreement, including arresting Islamic militants.

Table 5: Example output from MULTSUM. Input document: d30010t from DUC 04. Similarity Measures: W2V, TRComparer, and FilteredWordComparer.

something that has been used extensively in document summarization efforts. RegSum (Hong and Nenkova, 2014) trained a classifier on what kinds of words that human experts include in summaries. (Lin and Bilmes, 2011) represented sentences as a $tf \times idf$ weighted bag-of-words vector, defined a sentence graph with weights according to cosine similarity, and used submodular optimization to decide on sentences for a summary that is both representative and diverse.

Several other methods use similar sentence-based formulations but with different sentence similarities and summarization objectives (Radev et al., 2004; Mihalcea and Tarau, 2004).

(Bonzanini et al., 2013) introduced an iterative sentence removal procedure that proved good in summarizing short online user reviews. CLASSY04 (Conroy et al., 2004) was the best system in the official DUC 04 evaluation. After some linguistic preprocessing, it uses a Hidden Markov Model for sentence selection where the decision on inclusion of a sentence depends on its number of signature tokens. The following systems have also showed state–of–the–art results on the same data set. ICSI (Gillick et al., 2008) posed the summarization problem as a global integer linear program (ILP) maximizing the summary's coverage of key n-grams. OCCAMS_V (Davis et al., 2012) uses latent semantic analysis to determine the importance of words before the sentence selection. (Kulesza and Taskar, 2012) presents the use of Determinantal point processes (DPPs) for summarization, a probabilistic formulation that allows for a balance between diversity and coverage. An extensive description and comparison of these state–of–the–art systems can be found in (Hong et al., 2014), along with a repository of summary outputs on DUC 04.

Besides the aforementioned work, interested readers are referred to an extensive survey (Nenkova and McKeown, 2012). In particular, they discuss different approaches to sentence representation, scoring and summary selection and their effects on the performance of a summarization system.

8 Conclusions

We have demonstrated that extractive summarization benefits from using several sentence similarity measures at the same time. The proposed system, MULTSUM works by using standard kernel techniques to combine the similarities. Our experimental evaluation shows that the summaries produced by MULTSUM outperforms state-of-the-art systems on standard benchmark datasets. In particular, it beats the original submodular optimization approach on all three variants of ROUGE scores. It attains state-of-the-art results on both ROUGE-2 and ROUGE-SU4, showing that the resulting summaries have high fluency. The results are statistically significant and consistent over all three tested datasets: DUC 02, DUC 04, and Opinosis.

We have also seen that sentence similarity measures based on sentiment analysis and continuous vector space representations can improve the results of multi-document summarization. In our experiments, these sentence similarity measures used separately are not enough to create a good summary, but when combining them with traditional sentence similarity measures, we improve on previous methods.

Acknowledgments

This work has been done within "Data-driven secure business intelligence", grant IIS11-0089 from the Swedish Foundation for Strategic Research (SSF). We would like to thank Gabriele Capannini for discussion and help and Jonatan Bengtsson for his work on sentence similarity measures.

References

Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*, volume 463. ACM press New York.

Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. 2006. Neural probabilistic language models. In *Innovations in Machine Learning*. Springer.

Jonatan Bengtsson and Christoffer Skeppstedt. 2012. Automatic extractive single document summarization. Master's thesis, Chalmers University of Technology and University of Gothenburg.

Marco Bonzanini, Miguel Martinez-Alvarez, and Thomas Roelleke. 2013. Extractive summarisation via sentence removal: condensing relevant sentences into a short summary. In *SIGIR*, pages 893–896.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of ICML*, pages 160–167.

John M Conroy, Judith D Schlesinger, Jade Goldstein, and Dianne P O'leary. 2004. Left-brain/right-brain multi-document summarization. In *DUC 2004*.

Sashka T Davis, John M Conroy, and Judith D Schlesinger. 2012. Occams–an optimal combinatorial covering algorithm for multi-document summarization. In *ICDMW*. IEEE.

David Duvenaud. 2014. *Automatic model construction with Gaussian processes*. Ph.D. thesis, University of Cambridge.

Kavita Ganesan, ChengXiang Zhai, and Jiawei Han. 2010. Opinosis: a graph-based approach to abstractive summarization of highly redundant opinions. In *Proceedings of COLING*, pages 340–348. ACL.

Dan Gillick, Benoit Favre, and Dilek Hakkani-Tur. 2008. The icsi summarization system at tac 2008. In *Proceedings of TAC*.

Kai Hong and Ani Nenkova. 2014. Improving the estimation of word importance for news multi-document summarization. In *Proceedings of EACL*.

Kai Hong, John M Conroy, Benoit Favre, Alex Kulesza, Hui Lin, and Ani Nenkova. 2014. A repository of state of the art and competitive baseline summaries for generic news summarization. *LREC*.

Heng Ji, Benoit Favre, Wen-Pin Lin, Dan Gillick, Dilek Hakkani-Tur, and Ralph Grishman. 2013. Open-domain multi-document summarization via information extraction: Challenges and prospects. In *Multi-source, Multilingual Information Extraction and Summarization*, pages 177–201. Springer.

Mikael Kågebäck, Olof Mogren, Nina Tahmasebi, and Devdatt Dubhashi. 2014. Extractive summarization using continuous vector space models. *Proceedings of (CVSC)@ EACL*, pages 31–39.

Alex Kulesza and Ben Taskar. 2012. Determinantal point processes for machine learning. *arXiv:1207.6083*.

Kevin Lerman, Sasha Blair-Goldensohn, and Ryan McDonald. 2009. Sentiment summarization: Evaluating and learning user preferences. In *Proceedings of EACL*, pages 514–522. ACL.

Hui Lin and Jeff Bilmes. 2011. A class of submodular functions for document summarization. In *ACL*.

Chin-Yew Lin and Eduard Hovy. 2003. Automatic evaluation of summaries using n-gram co-occurrence statistics. In *Proceedings of NAACL/HLT*, pages 71–78.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proc. of the ACL-04 Workshop*, pages 74–81.

Hans Peter Luhn. 1958. The automatic creation of literature abstracts. *IBM Journal*, 2(2):159–165.

Henry B Mann, Donald R Whitney, et al. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 18(1):50–60.

Rada Mihalcea and Paul Tarau. 2004. Textrank: Bringing order into texts. In *Proceedings of EMNLP*, volume 4.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv:1301.3781*.

Ani Nenkova and Kathleen McKeown. 2012. A survey of text summarization techniques. In Charu C. Aggarwal and ChengXiang Zhai, editors, *Mining Text Data*, pages 43–76. Springer.

Hitoshi Nishikawa, Takaaki Hasegawa, Yoshihiro Matsuo, and Genichiro Kikui. 2010. Optimizing informativeness and readability for sentiment summarization. In *Proceedings of ACL*, pages 325–330. ACL.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of IWPT*. Citeseer.

Dragomir R. Radev, Timothy Allison, Sasha Blair-Goldensohn, John Blitzer, Arda Çelebi, Stanko Dimitrov, Elliott Drábek, Ali Hakim, Wai Lam, Danyu Liu, Jahna Otterbacher, Hong Qi, Horacio Saggion, Simone Teufel, Michael Topper, Adam Winkel, and Zhu Zhang. 2004. Mead - a platform for multidocument multilingual text summarization. In *LREC*.

Bernhard. Schölkopf, Koji. Tsuda, and Jean-Philippe. Vert. 2004. *Kernel methods in computational biology*. MIT Press, Cambridge, Mass.

Paper III

Named Entity Recognition in Swedish Health Records with Character-Based Deep Bidirectional LSTMs

S. Almgren, S. Pavlov, and O. Mogren

Reprinted from Proceedings of the Fifth Workshop on Building and Evaluating Resources for Biomedical Text Mining, **2016**

Named Entity Recognition in Swedish Health Records with Character-Based Deep Bidirectional LSTMs

Simon Almgren[1], **Sean Pavlov**[1], **Olof Mogren***
Chalmers University of Technology, Sweden
*olof@mogren.one

Abstract

We propose an approach for named entity recognition in medical data, using a character-based deep bidirectional recurrent neural network. Such models can learn features and patterns based on the character sequence, and are not limited to a fixed vocabulary. This makes them very well suited for the NER task in the medical domain. Our experimental evaluation shows promising results, with a 60% improvement in F_1 score over the baseline, and our system generalizes well between different datasets.

1 Introduction

Named Entity Recognition (NER) is the task of finding mentions of named entities in a text. In non-medical NER, entity classes are typically people, organizations, and locations. It is one of the fundamental Natural Language Processing (NLP) tasks and has been studied extensively.

In this paper, we approach the problem of finding medical entities such as (1) *disorders and findings*, (2) *pharmaceutical drugs*, and (3) *body structure*. Our proposed method uses deep bidirectional character-based recurrent neural networks (RNNs), trained in an end-to-end fashion to perform both boundary detection and classification at the same time.

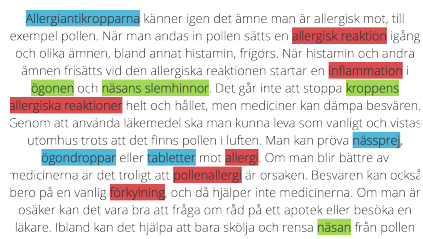


Figure 1: A Swedish medical example text with NER tags illustrated with colour.

There are a number of properties that make this problem especially challenging in biomedical text (Zhou et al., 2004). Firstly, names composed of multiple words are frequently used to describe an entity, highlighting the requirement of good boundary detection on an NER system. Secondly, one noun can be part of a mention of several entities at the same time E.g: “91 and 84 kDa proteins” consists of two entity names: “91 kDa proteins” and “84 kDa proteins”. Thirdly, it is common to write the same biomedical entity in different ways, e.g: “N-acetylcysteine”, “N-acetyl-cysteine”, “NAcetylCysteine”. Lastly, ambiguous mentions are common, including abbreviations that refer to different things in different contexts. (The examples above are from Zhou et al. (2004)).

Our proposed method has a number of benefits over previous work: Firstly, the model can simultaneously recognize and classify entity mentions. Secondly, using an end-to-end neural network approach eliminates the need for feature engineering. All features needed are learned by the model during training. Thirdly, because our model works on the raw character sequence, it does not suffer from out-of-vocabulary terms, it can learn that different character patterns represent the same thing, and it can learn the typical character-based features often used in traditional machine learning based solutions to NER.

We evaluate the model on Swedish health records in the Stockholm EPR corpus and obtain promising results. We also note that the method generalizes well between different datasets.

1. Equal contribution.

2 Background

A recurrent neural network (RNN) is a feedforward artificial neural network that can model a sequence of arbitrary length, using weight sharing between each position in the sequence. In a language setting, it is common to model sequences of words, in which case each input x_t is the vector representation of a word. In the basic RNN variant, the transition function is a linear transformation of the hidden state and the input, followed by a pointwise nonlinearity:

$$h_t = \tanh(Wx_t + Uh_{t-1} + b),$$

where W and U are trainable weight matrices, b is a bias term, and tanh is the nonlinearity.

Basic RNNs struggle with learning long dependencies and suffer from the vanishing gradient problem. This makes RNN models difficult to train (Hochreiter, 1998; Bengio et al., 1994), and provoked the development of the Long Short Term Memory (LSTM) (Schmidhuber and Hochreiter, 1997), that to some extent solves these shortcomings. An LSTM is an RNN where the cell at each step t contains an internal memory vector c_t, and three gates controlling what parts of the internal memory will be kept (the forget gate f_t), what parts of the input that will be stored in the internal memory (the input gate i_t), as well as what will be included in the output (the output gate o_t). In essence, this means that the following expressions are evaluated at each step in the sequence, to compute the new internal memory c_t and the cell output h_t. Here "$\odot$" represents element-wise multiplication.

$$\begin{aligned}
i_t &= \sigma(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}),\\
f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}),\\
o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}),\\
u_t &= \tanh(W^{(u)}x_t + U^{(u)}h_{t-1} + b^{(u)}),\\
c_t &= i_t \odot u_t + f_t \odot c_{t-1},\\
h_t &= o_t \odot \tanh(c_t).
\end{aligned} \tag{1}$$

Most RNN based models work on word level. Words are coded as a one-hot vector, and each word is associated with an internally learned embedding vector. In this work, we propose a character-level model that is able to learn features based on arbitrary parts of the character sequence.

LSTM networks have been used successfully for language modelling, sentiment analysis (Tang et al., 2015), textual entailment (Rocktäschel et al., 2016), and machine translation (Sutskever et al., 2014). In the following sections, we will see that the learned features are also suitable for recognizing and classifying mentions of medical entities in health record data.

3 Named Entity Recognition with Character-Based Deep Bidirectional LSTMs

In this paper, we propose a character based RNN model with deep bidirectional LSTM cells (BiLSTM) to do Named Entity Recognition in the medical domain (see Figure 2). The model is trained and evaluated on medical texts in Swedish. It has a softmax output layer with four outputs corresponding to each position in the input sequence, representing the three different entity labels, and a special label for all non-entity characters.

The model is trained end-to-end using backpropagation and the Adam optimizer (Diederik Kingma, 2015) to perform entity classification on a character-by-character basis. A neural network learns to internally represent data with representations that are useful for the task. This is an effect of using backpropagation, and allows us to eliminate all manual feature engineering, enabling quick deployment of our system.

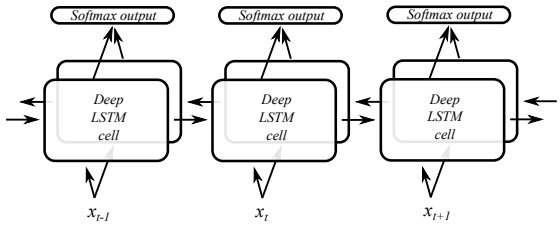


Figure 2: A deep bidirectional LSTM network. At each input x_t, the model is trained to output a prediction y_t of the correct entity class. In this paper, each block is a deep LSTM cell (see Figure 3), and the network is trained using backpropagation through time (BPTT).

3.1 Character classification

Our model works on the raw character sequence of the input document. This is an approach that has proven to work well in some other NLP applications, (see e.g. Luong and Manning (2016), **?**)).

Compared to a word-based sequence model, this means that we can use a much smaller vocabulary for the input tokens. Traditional (non-neural) entity recognition systems typically rely heavily on hand-engineered character-based features, such as capitalization, numerical characters, word prefixes and suffixes (Ratinov and Roth, 2009). Having the capacity of learning this kind of features automatically is what motivated us to use this kind of model. A character-based model does not rely on words being in its vocabulary: any word can be represented, as long as it is written with the given alphabet.

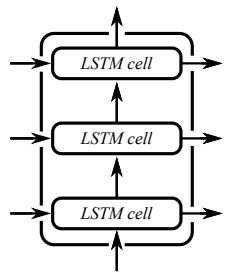


Figure 3: A deep LSTM cell, consisting of 3 internally stacked LSTM cells.

The character sequence model computes one classification output per input character. The label is one of: (1) *disorders and findings*, (2) *pharmaceutical drugs*, (3) *body structure*, (4) *non-entity term*. Using these labels (including the special "non-entity" label), we can simultaneously recognize and classify entity mentions by computing one label per character in the input text. This means that we can interpret each connected subsequence with the same classification as an entity mention.

However, there are some special cases: Firstly, to handle the situation when sporadic characters are classified inconsistently, we treat the character classifications as a voting mechanism for each word, and the majority class is chosen. Secondly, if a space between two tokens is classified consistently with the two tokens, both tokens are interpreted as belonging to the same entity mention. If the space is classified as a non-entity character, the two tokens are treated as two different entity mentions.

4 Experimental setup

This section explains the set-up of the empirical study of our model.

4.1 Model layout

We used a deep bidirectional recurrent neural network with LSTM cells. The depth of the LSTM cells was set to 3, and we used 128 hidden units in the LSTM cells. The model was implemented using Tensorflow. Learning rate: 0.002, decay rate: 0.975. Using drop-out on activations from the input embedding layers as well as on the LSTM output activations were evaluated, but was left out in the final version. See Section 4.4 for details on hyperparameters. The source code of our model is available on Github[1].

[1] `https://github.com/withtwist/medical-ner/`

4.2 Seed-term collection

Seed-terms are used both to build the datasets (see Section 4.3), and to build up the representations for the classification centroids in the BOW baseline method. Seed-terms were extracted from two taxonomies, SweMeSH[2], a taxonomy of Swedish medical terms and Snomed CT[3], consisting of Swedish medical concept terms. Using the hierarchical structure of the two taxonomies, all terms that was descendants of each of our predefined categories was extracted and considered seed-terms. The following predefined categories was used for the extraction: *disorder & finding* (*sjukdom & symtom*), *pharmaceutical drug* (*läkemedel*) and *body structure* (*kroppsdel*). The choice of these main entity classes was aligned with Skeppstedt et al. (2014).

4.3 Datasets

We use an approach similar to Mintz et al. (2009) to obtain the data needed for training and evaluation. The datasets that we prepared for training, validating and testing our model are available for download at `https://github.com/olofmogren/biomedical-ner-data-swedish/`.

The *Läkartidningen* corpus was originally presented by Kokkinakis and Gerdin (2010), and contains articles from the Swedish journal for medical professionals. This was annotated for NER as a part of this work. All occurrences of seed-terms were extracted (see Section 4.2), along with a context window of 60 characters (approximately ten words). The window is positioned so that the entity mention is located randomly within the sequence. In addition, negative training examples were extracted in order to prevent the model from learning that classified entities always occur in every sequence. All the characters in these negative training examples had the same "non-entity" label. Neural models typically benefit from large amounts of training data. To increase the amount of training data, each occurrence of seed-terms were extracted three more times, where the window was shifted by a random number of steps. The resulting data is a total of 775,000 of sequences with 60 characters each. 10% of the data is negative data, where every character has the "non-entity" label.

Another dataset was built from medical articles on the *Swedish Wikipedia*. Firstly, an initial list of medical domain articles were chosen manually and fetched. Secondly, articles were fetched that were linked from the initial articles. Finally, the seed-terms list (see Section 4.2) was used to create the labels and extract training examples of 60 character sequences, in the same way as was done with *Läkartidningen*.

1177 Vårdguiden iis a web site provided by the Swedish public health care authorities, containing information, counselling, and other health-care services. The corpus consists of 15 annotated documents downloaded during May 2016. This dataset was manually annotated with the seed-terms list as support (see Section 4.2). The resulting dataset has 2740 annotations, out of which 1574 are *disorder and finding*, 546 are *pharmaceutical drug*, and 620 are *body structure*.

The *Stockholm Electronic Patient Record (EPR) Clinical Entity Corpus* (Dalianis et al., 2012) is a dataset with health records of over two million patients at Karolinska University Hospital in Stockholm encompassing the years 2006-2014. It consists of 7946 documents containing real-world anonymized health records with annotations in 4 categories: *disorder*, *finding*, *drug* and *body structure*. Since we have a category where "*disorder*" and "*finding*" are bundled together they were considered the same.

Läkartidningen, *Swedish Wikipedia*, and *1177 Vårdguiden* are all datasets with rather high quality text, most of it even professionally edited. This is in stark contrast to the text in *Stockholm EPR* where misspellings are common, there are reduntant parts in many records, and writing style is highly diverse (Dalianis et al., 2009).

[2] `http://mesh.kib.ki.se/swemesh/swemesh_se.cfm`
[3] `http://www.socialstyrelsen.se/nationellehalsa/snomed-ct`

4.4 Hyperparameter search

A number of settings for hyperparameters were explored during development. In the variations listed below, one hyperparameter at the time is varied and evaluated, and if we saw an improvement, the change of setting was retained. A more thorough hyperparameter investigatin is left for future work. For the three first experiments, dropout was used on the activations from the embedding layer, as well as on the activations on the LSTM outputs. (See Section 4.1 for details).

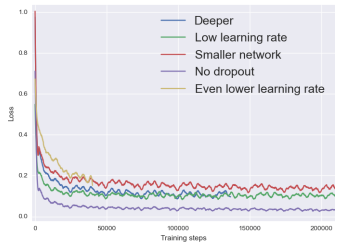


Figure 4: Validation loss.

Deeper: A model using 4 stacked LSTM cells. Learning rate: 0.05, decay rate: 0.975, drop probability: 0.5. *Low learning rate*: LSTK depth: 3, learning rate: 0.002, decay rate: 0.975. Lowering the learning rate proved useful and 0.002 became the default setting for learning rate. Drop probability: 0.5. *Smaller network*: 64 hidden units in each LSTM cell. LSTM depth: 3: learning rate: 0.002, decay rate: 0.975, drop probability: 0.5. *No dropout*: This model left all the settings as default but removed dropout entirely. 128 hidden units in each LSTM cell, LSTM depth: 3, learning rate: 0.002 and decay rate: 0.975. This setting proved to be the best, which meant that the default settings subsequently never used dropout. *Even lower learning rate*: Learning rate: 0.0002 and decay rate: 0.975. No drop-out.

See Figure 4 for the validation performance of the different settings. The resulting model used in the final experiments reported in Section 5 had 3 stacked LSTM layers with 128 hidden units in each. Learning rate: 0.002, decay rate 0.975, and no drop-out.

4.5 Baselines

Two baselines were implemented and used. The **dictionary baseline** simply consist of dictionary look-ups of the encountered words in the list of seed-terms. The **BOW (Bag-Of-Words) baseline** is based on Zhang and Elhadad (2013). The original version was developed for and evaluated on medical texts in English. The approach considers each noun-phrase as an entity candidate, and represents each candidate using a weighted bag-of-words-vector for the context. The required preprocessing is tokenization, sentence splitting, part-of-speech-tagging, and noun phrase chunking. The first steps was done using GATE (Cunningham et al., 2011) and OpenNLP[4], while noun phrase chunking was done using Swe-SPARK (Aycock, 1998). An IDF threshold is used to filter out uncommon or unspecific noun phrases. Then for each category the algorithm builds an average context vector representing the mentions in a training corpus. We used a triangular window for the context vectors, giving the central words a weight of 20, and context words the weight of $1/k$, where k is the distance from the central word. Mentions with a cosine similarity lower than 0.005 to any of the category vectors was discarded. Candidate mentions that have a difference between the top two scoring categories that is lower than 0.7 are also discarded.

Zhang and Elhadad (2013) used one bag-of-words vector for the internal words of an entity mention, and one for the context words of the mention. The two vectors were then concatenated, resulting in a vector which is twice the size of their vocabulary. Since the bag-of-words-vectors are already sparse to begin with, we added them together instead and made it possible to use a larger vocabulary size.

4.6 Training

Development and training were performed using text from *Läkartidningen* (Kokkinakis and Gerdin, 2010). Validation was done using the *Medical Wikipedia* dataset. Training was done using the Adam optimizer (Diederik Kingma, 2015).

[4] `http://opennlp.sourceforge.net/models-1.5/`

4.7 Evaluation

Evaluation of the proposed model was performed on two different datasets: *Stockholm EPR corpus* (Dalianis et al., 2012), with anonymized health record data in Swedish, and *1177 Vårdguiden*.

We report F_1 scores for total named entity recognition, as well as only entity classification (given correct boundary detection, we report scores of the entity classification performed by the system).

In the BOW baseline the entities are determined before hand while the Char-BiLSTM model recognizes and classifies them as it traverses the document.

5 Results

This section presents the results of the experimental evaluation of the proposed model. Table 1 shows the results of running the dictionary baseline model on *Stockholm EPR corpus* (Dalianis et al., 2012). The baseline model achieves a precision of over 0.70 on *disorder & Finding* and *body structure*, but is substantially lower for *pharmaceutical drug*. It has a higher precision than recall in general due to the fact that if a match is found it is probably correct. The algorithm got a precision of 0.67, a recall of 0.12 and an F_1 score of 0.20.

Category	**P**	**R**	**F_1**
Disorder & finding	0.76	0.12	0.20
Pharmaceutical drug	0.25	0.04	0.07
Body structure	0.70	0.29	0.41
Total	0.67	0.12	0.20

Table 1: Dictionary baseline performance on the *Stockholm EPR corpus*. Although total precision is reasonably good (0.67), the precision (0.12) is not.

The evaluation of the Char-BiLSTM model was performed on 733 real-world examples of health records from *Stockholm EPR corpus* (Dalianis et al., 2012). Since the data is very sensitive the evaluation was not performed by ourselves but instead the holder of the data performed the evaluations.

Char-BiLSTM overall results: The results in Table 2 shows the results of the Char-BiLSTM model. Both *disorder & finding* and *body structure* have a much higher precision than recall, while *pharmaceutical drug* is better balanced.

Category	**P**	**R**	**F_1**
Disorder & findings	0.72	0.18	0.29
Pharmaceutical drugs	0.69	0.43	0.53
Body structure	0.46	0.28	0.35
Total	0.67	0.24	0.35

Table 2: Results for Char-BiLSTM on *Stockholm EPR corpus*. The model obtains a total precision that matches the dictionary baseline (0.67), and a recall that is much higher than the baseline (0.24).

Char-BiLSTM results, classification only: Given the boundaries for the entities in *Stockholm EPR corpus*, the performance of the Char-BiLSTM model (performing only classification of the given entities) is given in Table 3. The table shows promising results for both the category *disorder & finding* and *pharmaceutical drug* which has an F_1 score of 0.81 and 0.74 respectively. *body structure* shows a weaker F_1 score of 0.47. The model got an overall F_1 score of 0.75.

We compare our system with the two baselines using the *1177 Vårdguiden corpus*. Since the BOW baseline detects boundaries using whole noun phrases, we re-ran the experiments, adjusting the evaluation data, so that the boundaries were complete noun-phrases.

Category	P	R	F_1
Disorder & findings	0.92	0.73	0.81
Pharmaceutical drugs	0.64	0.87	0.74
Body structure	0.36	0.68	0.47
Total	0.75	0.75	0.75

Table 3: Entity classification results of Char-BiLSTM on *Stockholm EPR corpus*. Given the entity boundaries, we can see that the classification of entities work very well, obtaining a total F_1 score of 0.75.

Comparing the results of the models in Table 4, we see that the BOW baseline does not perform well due to the wider boundaries that it detects. This can clearly be seen in the experiments with adjusted data, where it performs around 47% better. All models have around 0.50 in precision, except for the adjusted BOW baseline which comes in at 0.32. Recall is much lower (between 0.08 and 0.17), except for the BiLSTM model which has a recall of 0.21. This is the main reason why the BiLSTM model has the highest F_1 score with 0.29. At the second place comes the adjusted BOW baseline at 0.22, followed by the dictionary baseline model at 0.22 and lastly the BOW baseline with 0.15. Even though the dictionary baseline model and the adjusted BOW baseline have similar performance scores, we can see that their precision and recall are vastly different. The dictionary baseline model has a high precision and a low recall, while the adjusted BOW baseline is more balanced between precision and recall.

Model	P	R	F_1
Dictionary	0.54	0.14	0.22
BOW	**0.55**	0.09	0.15
BOW (adj.)	0.32	0.17	0.22
Char-BiLSTM	0.48	**0.21**	**0.29**

Table 4: Comparison of the results between each model on *1177 Vårdguiden*.

6 Related work

Supervised NER has been thoroughly explored in the past. Finkel et al. (2005) used Conditional Random Fields (CRF), a technique often used for NER. Zhou et al. (2004) used Hidden Markov Models (HMMs) along with extensive feature engineering to perform NER on medical texts. State-of-the-art in the medical domain have been achieved by Wang and Patrick (2009) with a combination of CRF, Support Vector Machines (SVM) and Maximum Entropy (ME) to recognize and classify entities.

Skeppstedt et al. (2014) currently holds the state-of-the-art in Swedish for the medical domain, based on CRF.

Recently, a number of papers have proposed using RNNs for sequence labelling tasks. Cícero Nogueira dos Santos (2015) presented a model that learns word embeddings along with character embeddings from a convolutional layer, which are used in a window-based fixed feed forward neural network. Huang et al. (2015) proposed a bidirectional LSTM model, but it used traditional feature engineering, and the classification was performed using a CRF layer in the network. In contrast, our proposed model learns all its features, and can be trained efficiently with simple backpropagation and stochastic gradient descent. Ma and Hovy (2016) presented a model that uses a convolutional network to compute representations for parts of words. Then the representations are combined with some character-level features and fed into a bidirectional LSTM network, and finally a CRF performs the labelling. Chiu and Nichols (2016) presented a similar model but with a softmax output instead of the CRF layer. Like our system, the models are trained end-to-end and obtains good results on standard NER evaluations, however our system is conceptually simpler, and learns all of its features directly from the character stream. Lam-

ple et al. (2016) presented two different architechtures, one using LSTMs and CRFs, and one using a shift-reduce approach. Gillick et al. (2016) presented a character-based model with LSTM units similar to a translation model, but instead of decoding into a different language, the state from the encoder is decoded into a sequence of tags.

Learning representations for text is important for many other tasks within natural language processing. A common way of representing sequences of words is to use some form of word embeddings (Mikolov et al., 2013; Pennington et al., 2014; Collobert and Weston, 2008), and for each word in the sequence, do an element-wise addition (Mitchell and Lapata, 2010). This approach works well for many applications, such as phrase similarity, multi-document summarization (Mogren et al., 2015), and word sense induction (Kågebäck et al., 2015), even though it disregards the order of the words. In contrast, RNNs and LSTMs (Hochreiter and Schmidhuber, 1997) learn representations for text that takes the order into account. They have been successfully been applied to sentiment analysis (Tang et al., 2015), question answering systems (Hagstedt P Suorra and Mogren, 2016), and machine translation (Sutskever et al., 2014).

Character-based neural sequence models have recently been presented to tackle the problem of out-of-vocabulary terms in neural machine translation (Luong and Manning, 2016; Chung et al., 2016) and for language modelling (Kim et al., 2016).

7 Discussion

The results of the empirical evaluation of the proposed system show some interesting points, suggesting that this approach should be researched further.

We have evaluated our model on the Stockholm EPR corpus of Swedish health records, but we did not compare our scores with other approaches that was evaluated on the same dataset. The reason is that we were unable to do a fair comparison since our model was trained on other data. We believe that our scores are competitive, and indicates that the model is promising. While systems that were trained on data from the same corpus show better performance in the evaluation on *Stockholm EPR* (Skeppstedt et al., 2014), we note that our solution can be trained on a dataset that is quite diferent from the test set. This can be explained in part with the documented robustness of character-based recurrent neural models to misspellings and out-of-vocabulary terms.

We are convinced that our solution would be able to obtain even better scores if able to train on the same data.

8 Conclusions

In this paper, we have proposed a character-based deep bidirectional LSTM model (Char-BiLSTM) to perform named entity recognition in Swedish health records. We beat two different baseline methods on the task, and show that this is a promising research direction. The proposed model obtains an F_1 score of 0.35 which is about 60% better than the BOW baseline (Zhang and Elhadad, 2013). Our model learns all the features it needs, and therefore eliminates the need for feature engineering. We have seen that a character-based neural model adapts well to this domain, and in fact that it is able to generalize from relatively well-written training data to test-data with lesser quality text.

Acknowledgments

This work has been done within the project “Data-driven secure business intelligence”, grant IIS11-0089 from the Swedish Foundation for Strategic Research (SSF).

References

John Aycock. 1998. Compiling little languages in python. In *Proceedings of the 7th International Python Conference*, pages 69–77.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.

Jason Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370.

Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. 2016. A character-level decoder without explicit segmentation for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1693–1703, Berlin, Germany, August. Association for Computational Linguistics.

Victor Guimarães Cícero Nogueira dos Santos. 2015. Boosting named entity recognition with neural character embeddings. *Proceedings of NEWS 2015 The Fifth Named Entities Workshop*.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. 2011. *Text Processing with GATE (Version 6)*.

Hercules Dalianis, Martin Hassel, and Sumithra Velupillai. 2009. The stockholm epr corpus—characteristics and some initial findings. In *Proceedings of the 14th International Symposium on Health Information Management Research - iSHIMR*.

Hercules Dalianis, Martin Hassel, Aron Henriksson, and Maria Skeppstedt. 2012. Stockholm epr corpus: A clinical database used to improve health care. *Swedish Language Technology Conference*, pages 17–18.

Jimmy Ba Diederik Kingma. 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics.

Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. 2016. Multilingual language processing from bytes. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1296–1306, San Diego, California, June. Association for Computational Linguistics.

Jacob Hagstedt P Suorra and Olof Mogren. 2016. Assisting discussion forum users using deep recurrent neural networks. *Proceedings of the 1st Workshop on Representation Learning for NLP at ACL 2016*, page 53.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Mikael Kågebäck, Fredrik Johansson, Richard Johansson, and Devdatt Dubhashi. 2015. Neural context embeddings for automatic discovery of word senses. In *Proceedings of NAACL-HLT*, pages 25–32.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Dimitrios Kokkinakis and Ulla Gerdin. 2010. Läkartidningens arkiv i en ny skepnad - en resurs för forskare, läkare och allmänhet. *Språkbruk*, pages 22–28.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California, June. Association for Computational Linguistics.

Minh-Thang Luong and Christopher D. Manning. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1054–1063, Berlin, Germany, August. Association for Computational Linguistics.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074, Berlin, Germany, August. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *ICLR*.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429.

Olof Mogren, Mikael Kågebäck, and Devdatt Dubhashi. 2015. Extractive summarization by aggregating multiple similarities. In *Recent Advances in Natural Language Processing*, page 451.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–43.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, and Phil Blunsom. 2016. Reasoning about entailment with neural attention. In *International Conference on Learning Representations*.

Jürgen Schmidhuber and Sepp Hochreiter. 1997. Long short-term memory. *Neural computation*, 7(8):1735–1780.

Maria Skeppstedt, Maria Kvist, H. Nilsson Gunnar, and Dalianis Hercules. 2014. Automatic recognition of disorders, findings, pharmaceuticals and body structures from clinical text: An annotation and machine learning study. *Journal of Biomedical Informatics*, 49:148–158.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Duyu Tang, Bing Qin, and Ting Liu. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1422–1432.

Yefeng Wang and Jon Patrick. 2009. Cascading classifiers for named entity recognition in clinical notes. In *Proceedings of the workshop on biomedical information extraction*, pages 42–49. Association for Computational Linguistics.

Shaodian Zhang and Noémie Elhadad. 2013. Unsupervised biomedical named entity recognition: Experiments with clinical and biological texts. *Journal of biomedical informatics*, 46(6):1088–1098.

Guodong Zhou, Jie Zhang, Jian Su, Dan Shen, and Chewlim Tan. 2004. Recognizing names in biomedical texts: a machine learning approach. *Bioinformatics*, 20(7):1178–1190.

Paper IV

Character-based recurrent neural networks for morphological relational reasoning

O. Mogren and R. Johansson

Reprinted from Submitted draft. Early version published at the EMNLP Workshop on Subword and Character-level Models in NLP, **2017**

Character-based recurrent neural networks for morphological relational reasoning

Olof Mogren
Chalmers University of Technology
Sweden
mogren@chalmers.se

Richard Johansson
University of Gothenburg
Sweden
richard.johansson@gu.se

Abstract

We present a model for predicting inflected word forms based on *morphological relational reasoning* with analogies. While previous work has explored tasks such as morphological inflection and reinflection, these models rely on an explicit enumeration of morphological features, which may not be available in all cases. In contrast, our model is feature-free: instead of explicitly representing morphological features, the model is given a *demo pair* that implicitly specifies a morphological relation (such as *write:writes* specifying *infinitive:present*). Given this demo relation and a *query word* (e.g. *watch*), the model then predicts the target word (e.g. *watches*). To address this task, we devise a character-based recurrent neural network architecture using three separate encoders and one decoder.

Our experimental evaluation on five different languages shows that the exact form can be predicted with high accuracy, consistently beating the baseline methods. Particularly, for English the prediction accuracy is 95.60%.

1 Introduction

Neural networks operating on sequences have become a part of the standard toolbox in modern natural langauge processing (NLP). Recently, a number of papers have been published that use character-level neural models as a way to address the inherent drawbacks of traditional models that represent words as atomic symbols. This offers a number of advantages: the vocabulary in a character-based model can be much smaller, as it only needs to represent a finite and fairly small alphabet, and as long as the characters are in the alphabet, no words will be out-of-vocabulary (OOV). Character-level models can capture distributional properties, not only of frequent words but also of words that occur rarely (Luong and Manning, 2016), and they need no tokenization, freeing the system from one source of errors. Neural models working on character- or subword-levels have been applied in several NLP tasks, ranging from relatively basic tasks such as text categorization (Zhang et al., 2015) and language modeling (Kim et al., 2016) to complex prediction tasks such as translation (Luong and Manning, 2016; Sennrich et al., 2016).

In particular, because they can recognize patterns on a subword level, character-based neural models are attractive in NLP tasks that require an awareness of *morphology*, or word inflection. Morphological analysis and prediction models using character-based recurrent neural networks have recently become popular, as evidenced by their complete dominance at the 2016 and 2017 SIGMORPHON shared tasks on morphological reinflection (Cotterell et al., 2016; Cotterell et al., 2017). However, in these models, including the top-performing system in the shared task (Kann and Schütze, 2016), an explicit feature representation of the morphological inflection needs to be provided as an input. These features represent number, gender, case, tense, aspect, etc.

In this paper, we take a new approach to predicting word forms that bypasses the need for an explicit representation of morphological features. We present a model that learns morphological *analogy relations* between words: given a *demo relation* $R_{demo} = (w_1, w_2)$, represented as a pair of words w_1 and w_2, and a *query word* q, can we apply the same relation as represented by R_{demo} to the query

word, and arrive at the correct target t? The task may be illustrated with a simple example: *see* is to *sees* as *eat* is to what?

The relation in the example above is trivial on a superficial level, as the model just needs to add an *s* to the query word. However, the analogy task is more challenging in the general case. The model needs to take into account that words belong to groups whose inflectional patterns are different – morphological *paradigms*. For instance, if we consider the past tense instead of the present in the example above, the relation is more complex: *see* is to *saw* as *eat* is to what? The model also needs to pick up general patterns that cut across paradigms, including phonological processes such as umlaut and vowel harmony, as well as orthographic quirks such as the rule in English that turns *y* into *ie* in certain contexts.

The fact that our model does not rely on explicit features makes it applicable in scenarios where features are unavailable, such as when working with under-resourced languages. However, since the model is trained using a weaker signal than in the traditional feature-based scenario, it needs to learn a latent representation from the analogies that plays the same role as the morphological features otherwise would, making the task more challenging.

2 Neural Morphological Analogies System

In this paper, we present the Neural Morphological Analogies System (NMAS), a neural approach for morphological relational reasoning. We use a deep recurrent neural network with GRU cells that take words represented by their raw character sequences as input.

2.1 Morphological relational reasoning with analogies

We define the task as follows. Given a query word q and two demo words w_1, w_2, where the demo words represent a transformation from one word form to another, and where q is another word in the same form as w_1, the task is to transform q into the form represented by w_2.

2.2 Recurrent neural networks

A recurrent neural network (RNN) is an artificial neural network that can model a sequence of arbitrary length. Gated RNNs were proposed to solve some issues of basic "vanilla" RNNs (the difficulty to capture long dependencies and vanishing gradients) (Hochreiter, 1998; Bengio et al., 1994). The Long Short Term Memory (LSTM) (Schmidhuber and Hochreiter, 1997) is one of the most famous types. At every step in the sequence, it has a cell with three learnable gates that controls what parts of the internal memory vector to keep (the forget gate f_t), what parts of the input vector to store in the internal memory (the input gate i_t), and what to include in the output vector (the output gate o_t). The Gated Recurrent Unit (GRU) (Cho et al., 2014a) is a simplification of this approach, having only two gates by replacing the input and forget gates with an update gate u_t that simply erases memory whenever it is updating the state with new input. Hence, the GRU has fewer parameters, and still obtains similar performance as the original LSTM.

An RNN can easily be trained to predict the next token in a sequence, and when applied to words this essentially becomes a language model. A sequence-to-sequence model is a neural language model conditioned on another input sequence. Such a model can be trained to translate from one sequence to another (Sutskever et al., 2014; Cho et al., 2014b). This is the major building block in modern neural machine translation systems, where they are combined with an attention mechanism to help with the alignment (Bahdanau et al., 2014).

In language settings it is common to have a linear input layer that learns embeddings for a vocabulary of words. However, these models suffer from the limitations of having fixed word vocabularies, and being unable to learn subword patterns. As an alternative, an RNN can work either using a vocabulary of subword units, or directly on the raw character stream, as is the case in this paper.

2.3 Model layout

The proposed model has three major parts, the relation encoder, the query encoder, and the decoder, all working together to generate the predicted target form given the three input words: the demo relation (w_1, w_2), and the query word q. The whole model is trained end-to-end and requires no other input than the raw character sequences of the three input words w_1, w_2, and q.

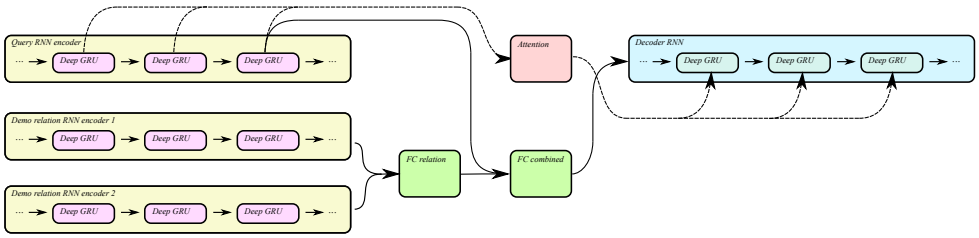


Figure 1: The layout of the proposed model. The demo relation is encoded using two encoder RNNs with shared weights for the two demo words. A fully connected layer *FC relation* follows the demo relation pair. The query word is encoded separately, and its embedding is concatenated with the output from *FC relation*, and fed as the initial hidden state into the RNN decoder which generates the output while using an attention pointer to the query encoder.

A. The relation encoder. The first part encodes the demo relation $R_{demo} = (w_1, w_2)$ using an encoder RNN for each of the two words w_1 and w_2. The relation encoder RNNs share weights but have separate internal state representations. The outputs of the relation encoders are fed into a fully connected layer with $\tanh$ activation *FC relation*.

B. The query encoder. The query word q is encoded separately using a distinct encoder RNN. The final output from the query encoder is fed together with the output from *FC relation* (A), through a second fully connected layer (with $\tanh$ activation) *FC combined*, and the result is fed as the initial hidden state into the RNN decoder.

C. The decoder. The decoder RNN employs a standard attention mechanism (Bahdanau et al., 2014), computing a weighted sum of the sequence of outputs of the query encoder at every step t_d in the generation process. For each step t_e in the query encoder, the attention weight is computed using a multi-layer perceptron taking the decoder state at t_d and the query encoder state at t_e as inputs. For each decoder step t_d, the output character is decided by computing a distribution over the alphabet using the softmax output layer, and then sampling greedily from this distribution; this is fast and has yielded good results.

The whole model is similar to a sequence-to-sequence model used for translation, with the addition of the relation encoder. Figure 1 shows the architecture of the model pictorially.

3 Experimental setup

This section explains the setup of the empirical evaluation of our model: how it is designed, trained, and evaluated.

3.1 Implementation details

The model was implemented using PyTorch[1]; all source code will be made openly available after publication. With the final hyperparameter settings (see Section 3.2), the model contains approximately 155000 parameters, and it can be trained in a few hours on a modern GPU.

3.2 Hyperparameters

The hyperparameters relevant to the proposed model are presented in Table 1. The hidden size parameter decides the dimensionality of all four RNNs in the model, as we noticed no performance gain from varying them individually.

3.3 Training

Training was done with backpropagation through time (BPTT) and minibatch learning with the Adam optimizer (Kingma and Ba, 2015). For each example in a minibatch, a relation type is selected uniformly randomly. Then two word pairs are selected randomly from that realation type; one of these will be the demo relation, and one will be the query–target pair. Training duration was decided using early stopping (Wang et al., 1994). One model was

[1] `http://pytorch.org/`

Hyperparameter	Explored	Selected
Embedding size	50-350	100
Hidden size	25-350	50
FC relation size	50-350	50
FC combined size	50-350	100
RNN depth	1-3	2
Learning rate		1×10^{-3}
L2 weight decay		5×10^{-5}
Drop probability	0.0, 0.2, ..., 0.8	0.0

Table 1: Hyperparameters in the model.

trained per language (ensembling did not improve the results).

3.4 Baselines

The task considered in this work is closely related to morphological reinflection, and systems for this generally obtain higher absolute numbers of prediction accuracy than ours, because more information is given at test time through the explicit enumeration of morphological tags. Our task is also related to the syntactic analogy task used to evaluate word embeddings (Mikolov et al., 2013c), and we also include the word embedding-based word-level analogy solution as a baseline.

Suffix copy (CP) This baseline works by copying suffixes from the demo relation: (1) Find the longest common prefix of the two words in the demo relation, consider the rest being the suffixes. (2) If the source suffix is also a suffix of the query word, then remove it, and add the target suffix to the query word. If the source suffix is not a suffix of the query word, the query word is returned as is. For instance, consider the demo relation *colder:coldest* and the query *wiser*. In this case, the longest common prefix is *colde*, the source suffix *r*, and the target suffix *st*. The target word is constructed by first removing the source suffix and then adding the target suffix, resulting in *wisest*.

Word embedding baseline This baseline uses pre-trained word embeddings using word2Vec CBOW (W2V) (Mikolov et al., 2013b) and FastText (FT) (Bojanowski et al., 2016). The FastText embeddings are designed to take subword information into account, and they performed better than Word2Vec CBOW-based vectors in our experiments. The prediction is selected by choosing the word in the vocabulary that has an embedding with the highest cosine similarity compared to

$$v(q) - v(w_1) + v(w_2),$$

where $v(w)$ is the word embedding of the word w, q is the query word, and w_1, w_2 are the two demo words in the demo relation.

Word embeddings have been used in previous work for this task (then called syntactic analogies), but the solution is limited by a fixed vocabulary, and needs retraining to incorporate new words. Although it is trained without supervision, training requires much data and comparing the resulting vector above with all words in the vocabulary is expensive.

To make the word embedding baseline stronger, we used the suffix copy baseline as a fallback whenever any of the three input words are missing in the vocabulary.

3.5 Datasets

One model was trained and evaluated on each of five different languages. Data for all languages except for English and Swedish was taken from the SIGMORPHON 2016 dataset (Cotterell et al., 2016).

English: A total of 10 relations and their corresponding inverse relations were considered:

- nouns:
  - singular–plural, e.g. *dog–dogs*
- adjectives:
  - positive–comparative, e.g. *high–higher*
  - positive–superlative, e.g. *high–highest*
  - comparative–superlative, e.g. *higher–highest*
- verbs:
  - infinitive–past, e.g. *sit–sat*
  - infinitive–present, e.g. *sit–sits*
  - infinitive–progressive, e.g. *sit–sitting*
  - past–present, e.g. *sit–sits*
  - past–progressive, e.g. *sit–sitting*
  - present–progressive, e.g. *sits–sitting*

For English, the dataset was constructed using the word list with inflected froms from the SCOWL project[2]. In the English data, 25,052 nouns, 1,433

[2]See `http://wordlist.aspell.net/`.

	Training set		Validation set		Test set		Total	
Language	**Rels**	**WPs**	**Rels**	**WPs**	**Rels**	**WPs**	**Rels**	**WPs**
English	10	74187	10	1000	10	1000	10	76187*
Finnish	1293	50269	431	1255	1092	11471	1323	62995
German	1572	70429	421	1271	1249	7768	1572	79468
Russian	1001	56119	290	1421	666	11492	1003	69031
Swedish	10	146551	10	1000	10	1000	10	148551*

Table 2: Number of relations ("Rels", after discarding size-1 relations) and word pairs ("WPs") in the data set. * English and Swedish word pairs are all used exactly twice, once in original order, and once reversed. This means that the effective number of word pairs for these two languages are double the numbers in this table.

adjectives, and 7,806 verbs were used for training. 1000 word pairs were selected randomly for validation and 1000 for testing, evenly distributed among relation types.

Swedish: Words were extracted from SALDO (Borin et al., 2013). In the Swedish data, 64,460 nouns, 12,507 adjectives, and 7,764 verbs were used for training. The division into training, validation, and test sets were based on the same proportions as in English. The same forms were used as in English, except that instead of the progressive form for verbs, the passive infinitive was used, e.g. *äta:ätas* 'eat:be eaten'.

Finnish, German, and Russian: For these languages, data from task1 and task2 in SIGMORPHON 2016 was used for training, and task2 data was used for evaluation. In this dataset, each word pair is provided along with morphological tags for the source and target words. We define a relation R as the combination of two sets of morphological tags, for which there exist words in the data. As the task described in this paper differs from the original SIGMORPHON task, with the additional requirement that every query–target word pair needs to be accompanied by a demo relation with the same forms, all relations with only one word pair were discarded. Of the SIGMORPHON datasets, we did not include Arabic, Georgian, Hungarian, Maltese, Navajo, Spanish, and Turkish, either because of the sparsity problem mentioned above,[3] or because the morphological features used in the language made it difficult to generate query–target pairs. The percentage of test set word pairs being discarded in the remaining languages: Finnish: 1.2%, German: 2.1%, and Russian: 0.5%. Details about dataset sizes can be found in Table 2.

[3]We decided on a threshold of at most 3% of the word pairs that could be discarded for the evaluation to be meaningful.

3.6 Evaluation

To evaluate the performance of the model, the datasets for English and Swedish were randomly split into training, validation, and test sets. For the SIGMORPHON languages (Finnish, German, and Russian), the provided dataset split was used, and the test was performed as specified in the dataset. For English and Swedish, each word pair was tested in both directions (switching the query word and the target word). Within one relation type, each word pair was randomly assigned another word pair as demo relation. Each word pair was used exactly once as a demo relation, and once as a query–target pair. Where nothing else is specified, reported numbers are the prediction accuracy. This is the fraction of predictions that exactly match the target words.

4 Results

This section presents the results of the experimental evaluation of the system.

4.1 Language-wise performance

The prediction accuracy results for the test set can be seen in Table 3, reaching an accuracy of 95.60% for English. While Finnish is a morphologically rich language, with 1323 distinct relations in the dataset, and with the lowest *Suffix Copy* baseline score of all evaluated languages (27.80%), NMAS is able to learn its relations rather well, with a prediction accuracy of 83.26%. For German and Swedish, the performance is 89.12%, and 90.10%, respectively. They both have more complex morphologies with

	Accuracy		AVG Levenshtein	
	NMAS	CP	NMAS	CP
English	95.60%	57.50%	0.06	0.64
Finnish	83.26%	27.80%	0.25	2.73
German	89.12%	78.27%	0.18	0.58
Russian	70.54%	46.21%	0.45	1.42
Swedish	90.10%	67.85%	0.16	0.60

Table 3: Prediction accuracy and average Levenshtein distance of the proposed model (NMAS) trained using one language. CP is the Suffix Copy baseline. Hidden size 50.

Size	Accuracy
25	94.70%
50	**95.60%**
75	94.70%
100	95.45%
150	94.30%
200	93.15%
250	90.95%

Table 4: Prediction accuracy of the proposed model on English test set using different hidden sizes.

Variant	Accuracy
No attention	89.30%
No relation input	37.30%

Table 5: Prediction accuracy of the proposed model without attention mechanism, and without relation encoder, respectively. English test set. Size: 50.

more inflectional patterns for nouns and verbs. On Russian, NMAS obtains an accuracy of 70.54%.

4.2 Model variants

Model size. Table 4 shows prediction accuracy for different sizes of the hidden state in the RNN modules. Although the difference in performance between the explored values is small, using a hidden size of 50 obtained the best score on both the validation and test sets.

Attend to relation. (Kann and Schütze, 2016) explicitly feeds the morphological tags as special tokens being part of the input sequence, and the attention mechanism learns when to focus on the forms during output generation. Inspired by this we decided to evaluate a variant of our model where the embedding of the relation encoder is appended to the query encoder output sequence, allowing the decoder to attend to the whole query as well as the relation embedding. The performance of the model changed minimally by this change (see Table 6), and there was no clear trend spanning over different languages. When also disabling *FC combined*, and thus the relation embedding input to the decoder, there was a noticeable decrease in performance: 94.75% accuracy on the English test set.

Variant	Accuracy
Attend to relation	95.50%
Attend to relation & No *FC combined*	94.75%
Reversed words	94.10%
Dropout	95.05%
Relation shortcut	94.45%
Auxiliary tags classification	93.85%

Table 6: Prediction accuracy of the proposed model trained with "attend to relation", with and without the relation embedding fed to initial hidden state (*FC combined*), all words reversed, using Dropout, feeding the relation embedding using a shortcut to each step in the decoder RNN, and using auxiliary tags classification criterion, respectively. English test set. Size: 50.

Relation shortcut. In the layout of the proposed model, the information from the relation encoder is available to the decoder only initially. To explore if it would help to have the information available at every step in the decoding process, a shortcut connection was added from *FC relation* to the final layer in the decoder. This helped the model to start learning fast (see Figure 4), but then resulted in a slight decrease in accuracy (94.45% on English test set). (See Table 6).

Dropout. Dropout (Srivastava et al., 2014) was applied to all fully connected layers and to the outputs of the recurrent layers. There was no improvement in performance for the evaluated values of drop probability (see Table 6).

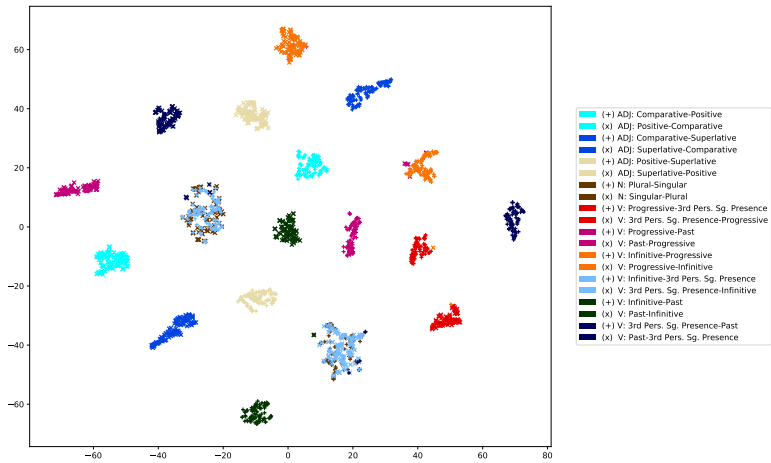


Figure 2: t-SNE visualization of all demo relation pairs from English validation set embedded using the relation encoder. Each point is colored by the relation type that it represents.

Auxiliary training criterion. Multi-task learning using a related *auxiliary task* can lead to stronger generalization and better regularized models (Bingel and Søgaard, 2017). We evaluated a model that used an auxiliary training task: the model had to predict the morphological tags as an output from the relation encoder. This addition gave a slight initial training speedup (see Figure 4), but did not give a better performing model once the model finished training. This indicates a strength in the originally proposed solution: the model can learn to differentiate the morphological forms of the words in the demo relation, even without having this explicit training signal, something that is also demonstrated by the visualized relation embeddings (see Figure 2).

Disabling model features. The relation encoder learns to represent the different morphological relations with nicely disentangled embeddings (see Figure 2). The fact that the prediction accuracy drops as far as to 37.30% when disabling the relation input (see Table 5) indicates that the setup is useful, and that the model indeed learns to utilize the information from the demo relation. Disabling the attention mechanism is a small modification of our model, but also substantially degrades performance, resulting in 89.30% accuracy on the English test set.

4.3 Mechanisms of word inflection

As English (as many other languages) forms inflections mainly by changing suffixes, an experiment was performed where every word was reversed (e.g. *requirement* $\rightarrow$ *tnemeriuqer*), to evaluate whether the model can cope with other mechanisms of word inflection. On this data, NMAS obtains a prediction accuracy that is only slightly worse than the original version. This indicates that the model can cope with different kinds of inflectional patterns (i.e. suffix and prefix changes). As can be noted in the example outputs (see Table 7), the model does handle several different kinds of inflections (including orthographic variations such as *y/ie*), and it does not require the demo relation to show the same inflectional pattern as the query word. In fact, often when the system fails, it does so by inflecting irregular words in a regular manner, suggesting that patterns with less data availability poses the major problem.

4.4 Relation embeddings

Figure 2 shows a t-SNE visualization of the embeddings from the relation encoder (*"FC relation"*) of all datapoints in the English validation set. One can

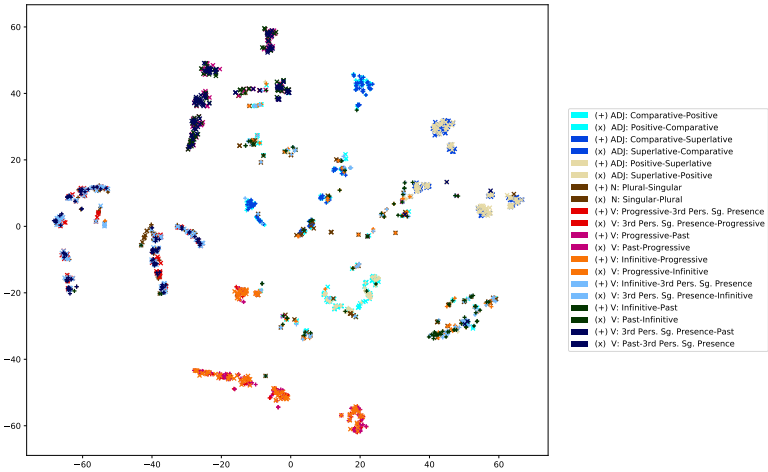


Figure 3: t-SNE visualization of all query words from English validation set embedded using the query encoder. Each point is colored by the relation type that it represents.

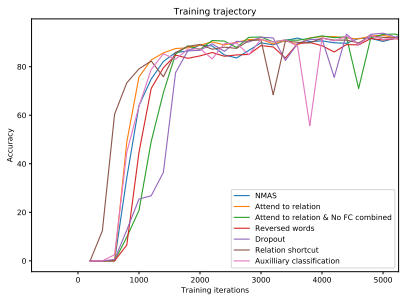


Figure 4: Prediction accuracy on the English validation set during training for some variations of the model.

see that most relations have been clearly separated into one distinct cluster each, with the exception of two clusters, both containing points from two relations each. The first such cluster contains the two relation types *"N: Singular-Plural"* and *"V: Infinitive-3 Pers. Sg. Present"*; both of these are realized in English by appending the suffix *-s* to the query word. The second cluster contains the relation types *"N: Plural-Singular"* and *"V: 3 Pers. Sg. Present-Infinitive"*; both of these are realized by the removal of the suffix *-s*. It is worth noting that no explicit training signal has been provided for this to happen. The model has learned to separate different morphological relations to help with the downstream task.

Figure 3 shows a t-SNE visualization of the embeddings from the query encoder. As we saw with the relation encoder, query embeddings seem to encode information about the morphology as similar morphological forms cluster together, albeit with more internal variation and more inter-cluster overlaps. The task for the query encoder is more complex as it needs to encode all information about the query word and provide information on how it may be transformed. To solve the task, and be able to correctly transform query words with the same relation type but with different inflection patterns, it needs to be able to deduce what subcategory of a relation a given query word belongs to.

4.5 Word embedding analogies

The suffix copy baseline proved to be the strongest baseline for all languages. For instance, for English it obtains prediction accuracy of 57.50%, compared to 41.55% for the Word2Vec baseline, and 45.40% for the FastText baseline. Without the suffix copy

Correct:

Demo word 1	Demo word 2	Query	Target	Output
misidentify	misidentifies	bottleneck	bottlenecks	bottlenecks
obliterate	obliterated	prig	prigged	prigged
ventilating	ventilates	disorganizing	disorganizes	disorganizes
crank	cranker	freckly	frecklier	frecklier
debauchery	debaucheries	bumptiousness	bumptiousnesses	bumptiousnesses

Incorrect:

Demo word 1	Demo word 2	Query	Target	Output
repackage	repackaged	outrun	outran	outrunned
misinformed	misinform	gassed	gas	gass
julep	juleps	catfish	catfish	catfishes
cedar	cedars	midlife	midlives	midlifes
affrays	affray	buzzes	buzz	buzze

Table 7: Correct (top), and incorrect (bottom) example outputs from the model. Samples from English validation set.

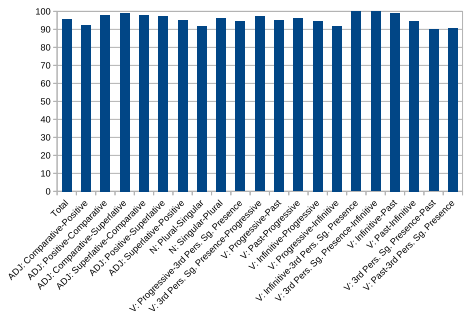


Figure 5: Results for all relations (total), and for each specific relation of the English test set.

fallback, the Word2Vec baseline scored 14.45%, and the FastText baseline scored 22.75%. For other languages, the results were even worse. The datasets in our study contains a rather large vocabulary, not only including frequent words. While the fixed vocabulary is one of the major limitations (explaining the difference between the embedding baselines and the corresponding ones without fallback), the word embedding baseline predictions were often incorrect even when the words were included in the vocabulary. This led us to use the Suffix Copy baseline in the result tables.

4.6 Relation-wise performance

Figure 5 shows the performance for each relation type, showing that our model obtains 100% test set accuracy for the transforms between *3rd pers. sg present–infinitive*. It obtains the lowest accuracy (91.76%) for *plural–singular*. From Figure 2 we have learned that these very relations are the most difficult ones for the relation encoder to distinguish between. (The inverse *singular–plural* is generally easier; one difficulty of *plural–singular* seem to be to determine how many character to remove, while the patterns for adding the *-s* suffix is generally simpler). An example demonstrating this can be seen in Table 7: *buzzes:buzz*, where the model incorrectly predicted *buzze*.

4.7 Example outputs

We have collected some examples from the English validation set where our model succeeds and where it fails (see Table 7). Examples of patterns that can be observed in the failed examples are (1) words with irregular inflections that the model incorrectly inflects using regular patterns, e.g. *outrun:outran*, where the model predicted *outrunned*; (2) words with ambiguous targets, e.g. *gassed:gas*, where the model predicted *gass*. If there had existed a verb *gass*, it could very well have been *gassed* in its past-tense form.

5 Related work

Morphological inflection and reinflection are the problems of turning a stem or a word form into a word form with another inflection. These problems have been studied using rule-based systems

(Koskenniemi, 1984; Ritchie et al., 1992), learned string transducers (Yarowsky and Wicentowski, 2000; Nicolai et al., 2015a; Ahlberg et al., 2015; Durrett and DeNero, 2013), and more recently, using neural sequence-to-sequence models. Faruqui et al. (2016) applied a character-based bidirectional LSTM model. They trained one model for each tag pair. Kann and Schütze (2016) used a similar model for inflection and reinflection, but trained one model for several tag pairs, feeding the source and target tags as part of the input sequence. The solution was the winner in the SIGMORPHON 2016 and 2017 shared tasks (Cotterell et al., 2016; Cotterell et al., 2017). While these papers showed that it is possible to learn reinflection using an input word with explicit source tags and target tags, our solution solves a more complex problem (of which one could consider the tagged version a sub-part): our model needs to infer not only the target form, but first also infer the forms of source and target words from the example relation.

Word analogies have been proposed as a way to demonstrate the utility of, and to evaluate the quality of neural word embeddings (Mikolov et al., 2013a; Mnih and Kavukcuoglu, 2013; Nicolai et al., 2015b; Pennington et al., 2014). Such embeddings show simple linear relationships in the resulting continuous embedding space that allow for finding impressive analogous relations such as

$$v(king) - v(man) + v(woman) \approx v(queen).$$

Analogies have been categorized as either semantic or syntactic. (The example with "king" and "queen" is a semantic analogy, while syntactic analogies relate different morphological forms of the same words). Google's dataset for syntactic analogies (Mikolov et al., 2013a) was proposed as a task to evaluate word embedding models on English.

6 Discussion and conclusions

In this paper, we have presented a neural model that can learn to carry out *morphological relational reasoning* on a given query word q, given a demo relation consisting of a word in the two different forms (source form and desired target form). Our approach uses a character based encoder RNN for the demo relation words, and one for the query word, and generates the output word as a character sequence. The model is able to generalize to unseen words as demonstrated by good prediction accuracy on the held-out test sets in five different languages: English, Finnish, German, Russian, and Swedish. It learns representations that separate the relations well provided only with the training signal given by the task of generating the words in correct form.

Our solution is more general than existing methods for morphological inflection and reinflection, in the sense that they require explicit enumeration of the morphological tags specifying the transformation; our solution instead learns to build its own internal representation of this information by observing an analogous word pair demonstrating the relation.

Acknowledgments

RJ was supported by the Swedish Research Council under grant 2013–4944. OM was supported by Swedish Foundation for Strategic Research (SSF) under grant IIS11-0089.

References

Malin Ahlberg, Markus Forsberg, and Mans Hulden. 2015. Paradigm classification in supervised learning of morphology. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1024–1029, Denver, Colorado, May–June. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.

Joachim Bingel and Anders Søgaard. 2017. Identifying beneficial task relations for multi-task learning in deep neural networks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 164–169, Valencia, Spain, April. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vec-

tors with subword information. *arXiv preprint arXiv:1607.04606*.

Lars Borin, Markus Forsberg, and Lennart Lönngren. 2013. SALDO: a touch of yin to WordNet's yang. *Language Resources and Evaluation*, 47(4):1191–1211.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014a. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014b. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *EMNLP*, pages 1724–1734. ACL.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, David Yarowsky, Jason Eisner, and Mans Hulden. 2016. The SIGMORPHON 2016 shared task – morphological reinflection. In *Proceedings of the 14th Annual SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 10–22.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, Géraldine Walther, Ekaterina Vylomova, Patrick Xia, Manaal Faruqui, Sandra Kübler, David Yarowsky, Jason Eisner, and Mans Hulden. 2017. CoNLL-SIGMORPHON 2017 shared task: Universal morphological reinflection in 52 languages. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 1–30, Vancouver, Canada, August. Association for Computational Linguistics.

Greg Durrett and John DeNero. 2013. Supervised learning of complete morphological paradigms. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1185–1195, Atlanta, Georgia, June. Association for Computational Linguistics.

Manaal Faruqui, Yulia Tsvetkov, Graham Neubig, and Chris Dyer. 2016. Morphological inflection generation using character sequence to sequence learning. In *Proceedings of NAACL-HLT*, pages 634–643.

Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

Katharina Kann and Hinrich Schütze. 2016. MED: The LMU system for the SIGMORPHON 2016 shared task on morphological reinflection. In *Proceedings of the 14th Annual SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 62–70.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Kimmo Koskenniemi. 1984. A general computational model for word-form recognition and production. In *Proceedings of the 10th international conference on Computational Linguistics*, pages 178–181. Association for Computational Linguistics.

Minh-Thang Luong and Christopher D. Manning. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1054–1063, Berlin, Germany, August. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. In *ICLR*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013c. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, June. Association for Computational Linguistics.

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2265–2273. Curran Associates, Inc.

Garrett Nicolai, Colin Cherry, and Grzegorz Kondrak. 2015a. Inflection generation as discriminative string transduction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 922–931, Denver, Colorado, May–June. Association for Computational Linguistics.

Garrett Nicolai, Colin Cherry, and Grzegorz Kondrak. 2015b. Morpho-syntactic regularities in continuous

word representations: A multilingual study. In *VS@ HLT-NAACL*, pages 129–134.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Graeme D Ritchie, Graham J Russell, Alan W Black, and Stephen G Pulman. 1992. Computational morphology.

Jürgen Schmidhuber and Sepp Hochreiter. 1997. Long short-term memory. *Neural computation*, 7(8):1735–1780.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

C. Wang, S. S. Venkatesh, and J. S. Judd. 1994. Optimal stopping and effective machine complexity in learning. In *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann.

David Yarowsky and Richard Wicentowski. 2000. Minimally supervised morphological analysis by multimodal alignment. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 207–216. Association for Computational Linguistics.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pages 649–657.

Paper V

Disentangled activations in deep networks

M. Kågebäck and O. Mogren

***Reprinted from Submitted draft. Early version presented at the NIPS Workshop on Learning Disentangled Features*, 2017**

DISENTANGLED ACTIVATIONS IN DEEP NETWORKS

Mikael Kågebäck & Olof Mogren *
Chalmers University of Technology
{kageback,mogren}@chalmers.se

ABSTRACT

Deep neural networks have been tremendously successful in a number of tasks. One of the main reasons for this is their capability to automatically learn representations of data in levels of abstraction, increasingly disentangling the data as the internal transformations are applied. In this paper we propose a novel regularization method that penalize covariance between dimensions of the hidden layers in a network, something that benefits the disentanglement. This makes the network learn nonlinear representations that are linearly uncorrelated, yet allows the model to obtain good results on a number of tasks, as demonstrated by our experimental evaluation. The proposed technique can be used to find the dimensionality of the underlying data, because it effectively disables dimensions that aren't needed. Our approach is simple and computationally cheap, as it can be applied as a regularizer to any gradient-based learning model.

1 INTRODUCTION

A good data representation should ultimately uncover underlying factors in the raw data while being useful for a model to solve some task. Deep neural networks learn representations that are increasingly abstract in deeper layers, disentangling the causes of variation in the underlying data (Bengio et al., 2013). Formal definitions of disentanglement are lacking, although Ver Steeg & Galstyan (2015); Achille & Soatto (2017) both use the total correlation as a measure of disentanglement. Inspired by this, we consider a simpler objective: a representation disentangles the data well when its components do not correlate, and we explore the effects of penalizing this linear dependence betweeen different dimensions in the representation. Ensuring independence in the representation space results in a distribution that is factorizable and thus easy to model (Kingma & Welling, 2014; Rezende et al., 2014).

We propose a novel regularization scheme that penalizes the cross-correlation between the dimensions of the learned representations, and helps artificial neural networks learn disentangled representations. The approach is very versatile and can be applied to any gradient-based machine learning model that learns its own distributed vector representations. A large body of literature have been published about techniques for learning non-linear independent representations (Lappalainen & Honkela, 2000; Honkela & Valpola, 2005; Dinh et al., 2015), but in comparison our approach is simpler, and does not impose restrictions on the model used. The proposed technique penalizes representations with correlated activations. It strongly encourages the model to find the dimensionality of the data, and thus to disable superfluous dimensions in the resulting representations. The experimental evaluation on synthetic data verifies this: the model is able to learn all useful dimensions in the data, and after convergence, these are the only ones that are active. This can be of great utility when pruning a network, or to decide when a network needs a larger capacity. The disabling of activations in the internal representation can be viewed as (and used for) dimensionality reduction. The proposed approach allows for interpretability of the activations computed in the model, such as isolating specific underlying factors. The solution is computationally cheap, and can be applied without modification to many gradient-based machine learning models that learns distributed representations.

Moreover, we present an extensive experimental evaluation on a range of tasks on different data modalities, which shows that the proposed approach disentangles the data well; we do get uncor-

*Equal contribution.

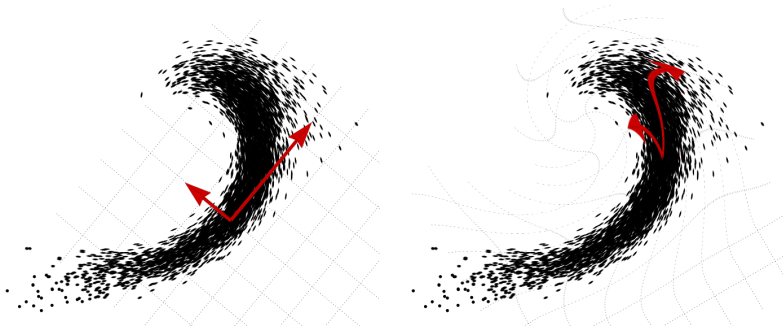


Figure 1: When data is distributed along non-linear manifolds, a linear model cannot describe the data well (left). However, with a non-linear model (right), it is possible to capture the variations of the data in a more reasonable way and unfold it into a compact orthogonal representation space.

related components in the resulting internal representations, while retaining the performance of the models on their respective task.

The main contributions of this work include: L_Σ regularization, a novel approach penalizing the covariance between dimensions in a representation (see Section 2). The regularizer encourages a model to use the minimal number of dimensions needed in the representation. The approach is computationally cheap and can be applied without any restrictions on the model. The experimental evaluation shows how different models can benefit from using L_Σ regularization. From autoencoders on synthetic data to deep convolutional autoencoders trained on CIFAR-10, we show that L_Σ helps us learn uncorrelated and disentangled representations (see Section 3).

2 Disentanglement through penalizing cross-correlations

We present a novel regularizer based on the covariance of the activations in a neural network layer over a batch of examples. The aim of the regularizer is to penalize the covariance between dimensions in the layer to decrease linear correlation.

2.1 Definition

The covariance regularization term (L_Σ) for a layer, henceforth referred to as the coding layer, is computed as

$$L_\Sigma = \frac{1}{p^2}||\mathcal{C}||_1 \tag{1}$$

where p is the dimensionality of the coding layer,

$$||\mathcal{C}||_1 = \sum_{i,j=1}^{N} |\mathcal{C}_{ij}|, \tag{2}$$

is the element wise L1 matrix norm of $\mathcal{C}$, and $\mathcal{C} \in \mathcal{R}^{p \times p}$ is the sample covariance of the activations in the coding layer over N examples

$$\mathcal{C} = \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{H} - \mathbf{1}_N\bar{\mathbf{h}})^T(\mathbf{H} - \mathbf{1}_N\bar{\mathbf{h}}).$$

Further, $\mathbf{H} = [\mathbf{h}_1; ...; \mathbf{h}_N]$ is a matrix of all activations in the batch, $\mathbf{1}_N$ is an N-dimensional column vector of ones, and $\bar{\mathbf{h}}$ is the mean activation.

2.2 USAGE

As L_Σ has the structure of a regularizer, it can be applied to most gradient based models without changing the underlying architecture. In particular, L_Σ is simply computed based on select layers and added to the error function, e.g. $Loss = Error + \lambda L_\Sigma$

3 EXPERIMENTS

This section describes the experimental evaluation performed using L_Σ regularization on different models in various settings, from simple multi-layer perceptron-based models using synthetic data (see Section 3.2 and 3.3) to convolutional autoencoders on real data (see Section 3.4). However, before describing the experiments in detail we define the metrics that will be used to quantify the results.

3.1 EVALUATION METRICS

A number of different metrics are employed in the experiments to measure different aspects of the results.

Mean Absolute Pearson Correlation (MAPC) Pearson correlation report the normalized linear correlation between variables $\in [-1, 1]$ where 0 indicates no correlation. To get the total linear correlation between all dimensions in the coding layer the absolute value of each contribution is averaged.

$$\mathbf{MAPC} = \frac{2}{(p^2 - p)} \sum_{i<j}^{p} \frac{|\mathcal{C}_{ij}|}{\sqrt{\mathcal{C}_{ii}}\sqrt{\mathcal{C}_{jj}}}$$

Covariance/Variance Ratio (CVR) Though mean absolute Pearson correlation measure the quantity we are interested in it becomes ill defined when the variance of one (or both) of the variables approaches zero. To avoid this problem we define a related measure where all variances are summed for each term. Hence, as long as some dimension has activity the measure remains well defined. More precise, the CVR score is computed as:

$$\mathbf{CVR} = \frac{1}{p^2} \frac{||\mathcal{C}||_1}{\mathbf{tr}(\mathcal{C})}$$

where $||\mathcal{C}||_1$ is defined as in Equation 2. The intuition behind CVR is simply to measure the fraction of all information that is captured in a linear uncorrelated fashion within the coding layer.

Top d-dimension Variance/total variance (TdV) TdV measure to what degree the total variance is captured inside the variance of the top d dimensions. When d is equal to the actual dimension of the underlying data this measure is bounded in [0,1].

Utilized Dimensions (UD) UD is the number of dimensions that needs to be kept to retain a set percentage, e.g. 90% in the case of $\text{UD}_{90\%}$, of the total variance. This measure has the advantage that the dimension of the underlying data does not need to be known a priori.

3.2 DIMENSIONALITY REDUCTION

The purpose of this experiment is to investigate if it is possible to disentangle independent data that has been projected to a higher dimension using a random projection, i.e. we would like to find the principal components of the original data.

The model we employ in this experiment is an auto encoder consisting of a linear $p = 10$ dimensional coding layer and a linear outputlayer. The model is trained using the proposed covariance regularization L_Σ on the coding layer.

The data is generated by sampling a $d = 4$ dimensional vector of independent features $z \sim N(0, \Sigma)$, where $\Sigma \in \mathcal{R}^{d \times d}$ is constrained to be non-degenerate and diagonal. However, before the data is fed

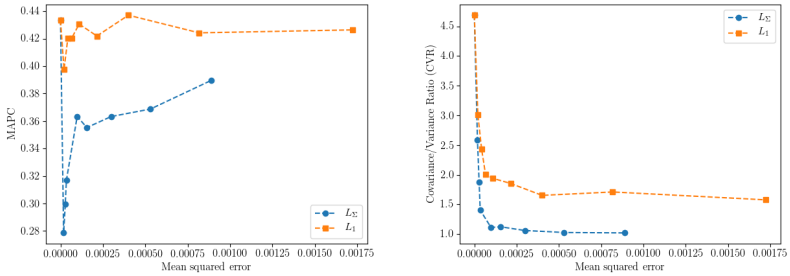


Figure 2: In this figure we compare the amount of residual linear correlation after training the model with L_Σ and L_1 regularization respectively, measured in MAPC (left) and CVR (right). The first point on each curve corresponds to $\lambda = 0$, i.e. no regularization, followed by 8 points logarithmically spaced between 0.001 and 1. All scores are averaged over 10 experiments using a different random projection (Ω).

to the autoencoder it is pushed through a random linear transformation $x = \Omega z$. The goal of the model is to reconstruct properties of z in the coding layer while only having access to x.

The model is trained on 10000 iid random samples for 10000 epochs. 9 experiments were performed with different values for the regularization constant λ. The first point on each curve (in Figure 2 and 3) is $\lambda = 0$, i.e. no regularization, followed by 8 points logarithmically spaced between 0.001 and 1. Each experiment is repeated 10 times using a different random projection Ω and the average is reported.

The result of the experiment is reported using all four metrics defined in Section 3.1. The result in terms of MAPC and CVR is reported in Figure 2. The first thing to notice is that L_Σ consistently lead to lower correlation while incurring less MSE penalty compared to L_1. Further, looking at the MAPC it is interesting to notice that it is optimal for a very small values of L_Σ. This is because higher amounts of L_Σ leads to lowering of the dimensionality of the data, see Figure 3, which in turn yields unpredictable Pearson correlation scores between these inactivated neurons. However, this effect is compensated for in CVR for which L_Σ quickly converges towards the optimal value of one, which in turn indicates no presence of linear correlation.
Turning the attention to dimensionality reduction, Figure 3 shows that L_Σ consistently outperform L_1. Further, looking closer at the TdV score, L_Σ is able to compress the data almost perfectly, i.e. TdV=1, at a very small MSE cost while L_1 struggle even when accepting a much higher MSE cost. Further, the $\text{UD}_{90\%}$ scores again show that L_Σ achieves a higher compression at lower MSE cost. In this instance the underlying data was of 4 dimensions which L_Σ quickly achieves. At higher amounts of L_Σ the dimensionality even locationally fall to 3, however, this is because the threshold is set to 90%.

3.3 Deep network of uncorrelated features

In Section 3.2 we showed that we can learn a minimal orthogonal representation of data that is generated to ensure that each dimension is independent. However, in reality it is not always possible to encode the necessary information, to solve the problem at hand, in an uncorrelated coding layer, e.g. the data illustrated in Figure 1 would first need a non linear transform before the coding layer. However, using a deep network it should be possible to learn such a nonlinear transformation that enables uncorrelated features in higher layers. To test this in practice on a problem that has this property but still is small enough to easily understand we turn to the XOR problem.

It is well known that the XOR problem can be solved by a neural network of one hidden layer consisting of a minimum of two units. However, instead of providing this minimal structure we would like the network to discover it by itself during training. Hence, the model used is intentionally over-

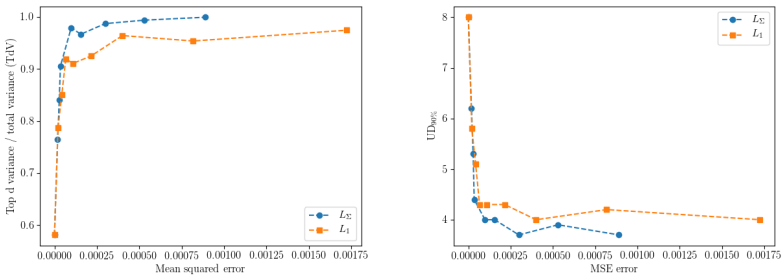


Figure 3: The resulting dimensionality the coding layer after training the model with L_Σ and L_1 regularization respectively, measured in TdV (left) and $UD_{90\%}$ (right). The first point on each curve corresponds to $\lambda = 0$, i.e. no regularization, followed by 8 points logarithmically spaced between 0.001 and 1.All scores are averaged over 10 experiments using a different random projection (Ω).

specified consisting of two hidden layers of four logistic units each followed by a one dimensional logistic output layer.

The model was trained on XOR examples, e.g. [1,0]=1, in a random order until convergence with L_Σ applied to both hidden layers and added to the cost function after scaling it with $\lambda = 0.2$.

As can be seen in Figure 4 the model was able to learn the optimal structure of exactly 2 dimensions in the first layer and one dimension in the second. Further, as expected, the first layer do encode a negative covariance between the two active units while the second layer is completely free from covariance. Note that, even though the second hidden layer is not the output of the model it does encode the result in that one active neuron. For comparison, see Figure 5 for the same model trained without L_Σ.

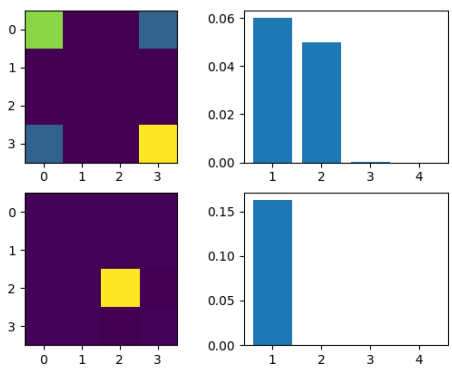


Figure 4: Covariance matrix (left) and spectrum (right) of the hidden layers of a feed forward neural network trained with L_Σ regularization to solve the XOR problem. Layer one (top) has learned to utilize unit zero and three while keeping the rest constant, and in layer two only unit two is utilized. This learned structure is the minimal solution to the XOR problem.

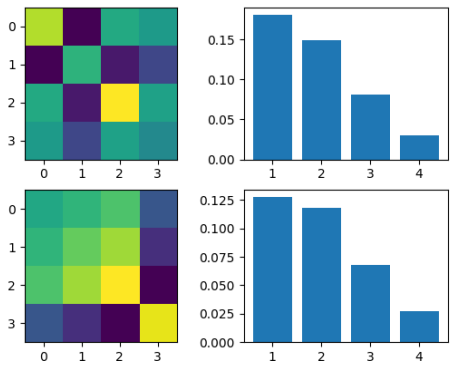


Figure 5: Covariance matrix (left) and spectrum (right) of the hidden layers of a feed forward neural network trained without regularization to solve the XOR problem.

3.4 NON-LINEAR UNCORRELATED CONVOLUTIONAL FEATURES

Convolutional autoencoders have been used to learn features for visual input and for layer-wise pretraining for image classification tasks. Here, we will see that it is possible to train a deep convolutional autoencoder on real-world data and learn representations that have low covariance, while retaining the reconstruction quality.

To keep it simple, the encoder part of the model used two convolutional layers and two fully connected layers, with a total of roughly 500.000 parameters in the whole model. The regularization was applied to the coding layer which has 84 dimensions, giving a bottleneck effect. The model was trained and evaluated on the CIFAR-10 dataset (Krizhevsky & Hinton, 2009), containing 32x32 pixel colour images tagged with 10 different classes. The model was trained on 45,000 images, while 5,000 were set aside for validation, and 10,000 make out the test set. We compare the results from using L_Σ regularization with L1 regularization and with no regularization at all.

The autoencoder was trained with a batch size of 100, using the Adam optimizer (Kingma & Ba, 2015) with an initial learning rate of 0.001. Training was run until the MSE score on the validation set stopped improving[1]. The regularization parameter λ was chosen to be 0.08, for a reasonable trade-off between performance and covariance/variance ratio. The reported scores in Table 1 and Figure 6 are averages from training the model five times with different initialization.

The results (see Table 1) show that the high-level features become more disentangled and has a lower CVR (6.56) using L_Σ regularization. Without regularization, the score is 20.00, and with L1 regularization the score is 4.03. The model with L_Σ regularization obtains a reconstruction error (MSE) of 0.0398, roughly the same as without regularization (0.0365), both of which are much better than using L1 regularization, with an MSE of 0.0569. Figure 6 shows the CVR score plotted against the MSE, illustrating that the L_Σ technique leads to more disentangled representations while retaining a better MSE score. As you increase the regularization factor both L_Σ regularization pushes down the CVR quickly, while retaining an MSE error that is almost constant. L1 regularization also pushes the model towards learning representation with lower CVR, although slower, and while worsening the MSE error. The $\text{UD}_{90\%}$ results show that L_Σ encourages representations that concentrate the variation, and the model constantly learns representations with lower $\text{UD}_{90\%}$ score than using L1. With $\lambda > 0.08$, the MSE, the CVR, and the $\text{UD}_{90\%}$ all becomes much worse when using L1 reg-

[1]The source code will be made available when the paper is deanonymized.

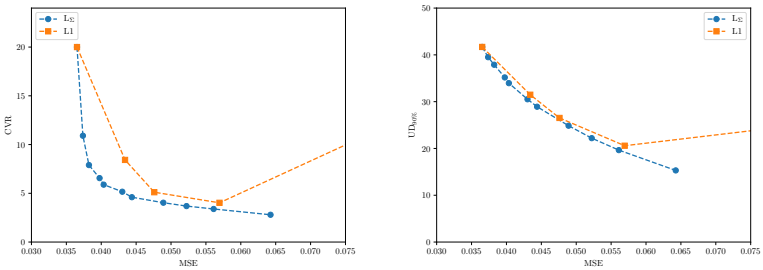


Figure 6: Results from the convolutional autoencoder experiments on CIFAR-10: **Left:** CVR plotted against MSE on the CIFAR-10 test set, using L_Σ regularization and L1 regularization, respectively. **Right:** $\text{UD}_{90\%}$ plotted against MSE on the CIFAR-10 test set, using L_Σ regularization and L1 regularization, respectively. Each point in the plots correspond to doubling the regularization parameter: $\lambda \in [0.0, 0.2, ..., 10.24]$.

ularization, while the L_Σ seems to continue smoothly to improve CVR and $\text{UD}_{90\%}$, as the MSE starts to grow.

Regularizer	**CVR**	**$\text{UD}_{90\%}$**	**MSE**
L_Σ	6.56	35.18	0.0398
L1	4.03	20.59	0.0569
No regularization	20.00	41.69	0.0365

Table 1: Results from the convolutional autoencoder experiments on CIFAR. The coding covariance is a normalized sum of covariance over the activations in the coding layer. Reproduction MSE is the mean squared error of the reconstructed images produced by the decoder.

4 Related work

Disentanglement is important in learned representations. Different notions of independence have been proposed as useful criteria to learn disentangled representations, and a large body of work has been dedicated to methods that learn such representations.

Principal component analysis (PCA; Pearson, 1901) is a technique that fits a transformation of the (possibly correlated) input into a space of lower dimensionality of linearly uncorrelated variables. Nonlinear extensions of PCA include neural autoencoder models (Kramer, 1991), using a network layout with three hidden layers and with a bottleneck in the middle coding layer, forcing the network to learn a lower-dimensional representation. Self-organizing maps (Kohonen, 1982) and kernel-based models (Schölkopf et al., 1998) have also been proposed for nonlinear PCA.

Independent component analysis (ICA; Hyvärinen et al., 2004) is a set of techniques to learn additive components of the data with a somewhat stronger requirement of statistical independence. A number of approaches have been made on non-linear independent components analysis, (Lappalainen & Honkela, 2000; Honkela & Valpola, 2005). While ICA has a somewhat stronger criterion on the resulting representations, the approaches are generally more involved. Dinh et al., (2015; 2017) proposed a method to train a neural network to transform data into a space with independent components. Using the substitution rule of differentiation as a motivation, they learn bijective transformations, letting them use the neural transformation both to compute the transformed hidden state, to sample from the distribution over the hidden variables, and get a sample in the original data space. The authors used a fixed factorial distribution as prior distribution (i.e. a distribution with independent dimensions), encouraging the model to learn independent representations. The model is demonstrated as a generative model for images, and for inpainting (sampling a part of the image,

when the rest of it is given). Achille & Soatto (2017) connected the properties of disentanglement and invariance in neural networks to information theoretic properties. They argue that having invariance to nuisance factors in a network requires that its learned representations to carry minimal information. They propose using the information bottleneck Lagrangian as a regularizer for the weights. Our approach is more flexible and portable, as it can be applied as a regularization to learn uncorrelated components in any gradient-based model that learns internal representations.

Brakel & Bengio (2017) showed that it is possible to adversarial training to make a generative network learn a factorized, independent distribution $p(\mathbf{z})$. The independence criterion (mutual information) makes use of the Kullback-Leibler divergence between the joint distribtion $p(\mathbf{z})$ (represented by the generator network) and the product of the marginals (which is not explicitly modelled). In this paper, the authors propose to resample from the joint distribution, each time picking only the value for one of the components z_i, and let that be the sample from the marginal for that component, $p(z_i)$. A discriminator (the adversary) is simultaneously trained to distinguish the joint from the product of the marginals. One loss function is applied to the output of the discriminator, and one measures the reconstruction error from a decoder reconstructing the input from the joint.

Thomas et al. (2017) considers a reinforcement learning setting where there is an environment with which one can interact during training. The authors trained one policy $\pi_i(a|\mathbf{s})$ for each dimension i of the representation, such that the policy can interact with the environment and learn how to modify the input in a way that modifies the representation only at dimension i, without changing any other dimensions. The approach is interesting because it is a setting similar to humans learning by interaction, and this may be an important learning setting for agents in the future, but it is also limited to the setting where you do have the interactive environment, and cannot be applied to other settings discussed above, whereas our approach can.

5 Conclusions

In this paper, we have presented L_Σ regularization, a novel regularization scheme based on penalizing the covariance between dimensions of the internal representation learned in a hierarchical model. The proposed regularization scheme helps models learn linearly uncorrelated variables in a non-linear space. While techniques for learning independent components follow criteria that are more strict, our solution is flexible and portable, and can be applied to any feature-learning model that is trained with gradient descent. Our method has no penalty on the performance on tasks evaluated in the experiments, while it does disentangle the data.

We saw that our approach performs well applied to a standard deep convolutional autoencoder on the CIFAR-10 dataset (Krizhevsky & Hinton, 2009); the resulting model performs comparable to the model without L_Σ regularization, while we can also see that the covariances between dimensions in the internal representation decrease drastically.

References

A. Achille and S. Soatto. Emergence of Invariance and Disentangling in Deep Representations. *ICML Workshop on Principled Approaches to Deep Learning*, 2017.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Philmon Brakel and Yoshua Bengio. Learning independent features with adversarial nets for non-linear ICA. *arXiv preprint arXiv:1710.05050*, 2017.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear independent components estimation. *ICLR*, 2015.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *ICLR*, 2017.

Antti Honkela and Harri Valpola. Unsupervised variational Bayesian learning of nonlinear models. In *Advances in neural information processing systems*, pp. 593–600, 2005.

Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *ICLR*, 2014.

Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.

Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 2009.

Harri Lappalainen and Antti Honkela. Bayesian non-linear independent component analysis by multi-layer perceptrons. In *Advances in independent component analysis*, pp. 93–121. Springer, 2000.

Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara (eds.), *Proceedings of the 31st International Conference on Machine Learning*, pp. 1278–1286, 2014.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

Valentin Thomas, Jules Pondard, Emmanuel Bengio, Marc Sarfati, Philippe Beaudoin, Marie-Jean Meurs, Joelle Pineau, Doina Precup, and Yoshua Bengio. Independently controllable features. *arXiv preprint arXiv:1708.01289*, 2017.

Greg Ver Steeg and Aram Galstyan. Maximally informative hierarchical representations of high-dimensional data. In *Artificial Intelligence and Statistics*, pp. 1004–1012, 2015.