1 Conclusion

The key to artificial intelligence has always been the representation. You and I are streaming data engines.

— Jeff Hawkins, 2012

Some final reminders
This section has some final reminders, of points made earlier, and some final remarks not made elsewhere.

In this book we have introduced methods for finding representations of natural language. A special focus of this book is on neural networks and related technologies. Neural networks elegantly produce useful representations as by-products, when applied to NLP tasks. We have introduced recent advances in neural networks, in particular, various forms of recurrent neural networks, and have covered techniques for working with words, word senses, and larger structures such as phrases, sentences and documents.

Reminder: Gradient Checks
Back-propagation is a very easy algorithm to mess up, and it is hard to tell when you do mess it up, because it will often still work reasonably well. Gradient checks, either with finite-differencing, or with more sophisticated techniques (e.g. dual-numbers), is a quick and easy way to check the correctness of your implementations.

?? introduced the idea of machine learning, which allows us to train a system using examples of the desired outputs for given inputs. The system can learn to determine the outputs for inputs that were never seen during training. A key feature of modern machine learning is the decreased reliance on hand-engineered features to represent their inputs. The learnt embeddings and vector representations which are the main topic of this book, are just such automatically derived features.

Hyperparameters matter
Neural networks have many hyper parameters, from hidden layer-size, to the regularisation penalty, to the number of training iterations. Choosing the right ones can significantly impact performance.

?? showed how recurrent neural networks allowed working with inputs of varying lengths – such as natural language senses. It discussed the various types of recurrent networks as characterised by their recurrent units, including both GRU and LSTM. As well as the common RNN structures, including: the encoder for producing a fixed-size output (such as a classification) from a varying sized input; the decoder for generating a varying sized output from a fixed size input; and the encoder-decoder for when the input and output both vary in size. These models are very practical for natural language processing applications.

Preprocessing Matters
There is a large number of preprocessing tricks that can be employed when processing text. Options include removing stop-words, rare words, and punctuation, as well as removing or replacing numbers and dates. Different options for tokenization exist with regard to splitting up contractions and other factors. Often, it is good to convert all the text to lower-case. One can even lemmatize or stem every word occurrence. What is useful depends on the task.

?? discussed how we can find and use representations of words. This is one of the most important ideas in machine learning for NLP. It begins with the core idea of how a word can be input into a neural network: the input embedding via a look-up table. The input embeddings, and the complementarity output embeddings obtained from the weights of the softmax layer, capture a representation of the words. The meaning of a word is determined by its usage, which is largely characterised by what words it co-occurs with. Word representation models include the well known skip-gram model which use these principles to derive high quality representations that can be reused as features in many tasks. The skip-gram model is based around predicting which words will be co-located. It is closely related to an iterative approximation to factorising the matrix of co-occurrence counts. This is similar to the approaches commonly used in older methods such as LSI and LDA. Word embedding models commonly use hierarchical softmax or negative sampling to speed-up the training and the evaluation.

Reminder: There is more to clustering than K-means
K-means is the most well-known clustering algorithm. But it is by no means the only one. K-means in particular is very vulnerable to getting stuck in local optima, compared to many other clustering algorithms. If one does use k-means, then make sure to run it multiple times and take the best result.

?? extends the idea of representing words to representing senses. Most words have many meanings. It is thus impossible for a single representation to characterise the correct meaning in all contexts. Word senses can either be externally defined using a lexical resource, or discovered (induced) from the data. If the senses are externally defined, determining a good representation for them boils down to disambiguating a corpus to find the senses used, then creating a representation for them using the single sense word embedding methods. Inducing the senses from the corpus is more in-line with the general goal of not needing hand-engineered features, and there are many more methods for this.

The majority of word sense induction methods are either context-clustering-based, or co-location-prediction based. In both cases, the core idea is still that (like for word-embeddings) the surrounding words characterise what a particular word use means. In the *context clustering* methods, for each word the different contexts in which it occurs are represented and then clustered. Each cluster represents a word sense. In the *co-location prediction* approaches the word sense is treated as a hidden (latent) variable, which influencing the observable variables, i.e. the context words which it is co-occurring with. Classical probability methods can then be used, together with neural network methods, in order to uncover that latent variable that is the sense. These word sense induction methods give embeddings that can be used in much the same way as word embeddings.

?? takes the idea of representing words to larger structures. There is a great diversity of methods to represent phrases, sentences, and documents. In three broad categories, we can consider weakly ordered models, sequential models, and structural models. The *weakly ordered models* are the most diverse category, being a catch-all term for a variety of methods that do not directly consider word order, including the neural embedding extensions of bag of words and bag of n-grams. The *sequential methods* are largely the application of the RNNs from **??**, but also include some methods specifically for obtaining a representation. Finally, the *structural methods* allow the inputs to change the structure of the network, allowing it to process parse trees in a natural way.

All things considered, there are a great number of techniques for using natural language with machine learning. They all revolve around the core idea of a representation. Beyond that they range from simple to complex.

The methods used need to be sufficiently powerful to accomplish the task, but beyond this excess capacity is both a waste of training time, and a waste of developer effort. Spending more time to implement one of the more complicated models may not result in a final system that even works as well as a simple baseline. The ideal system is suitably simple. However, the identification of what it means for an implemen-

Machine learning beyond neural networks
Though we have barely mentioned them in this book, there are plenty of other machine learning algorithms beyond just neural networks. Many of these have great advantages over neural networks, in terms of their performance with smaller amounts of data.
Some examples of some of the diversity of options (contrasted with neural networks) can be found at `https://white.ucc.asn.au/2017/12/18/7-Binary-Classifier-Libraries-in-Julia.html`.
It is often ideal to take word-embeddings (or other neural network derived representations) and use them as features, in a classifier such as a support vector machine (Cortes and Vapnik 1995), or a gradient boosting tree ensemble (Chen and Guestrin 2016).

tation to be suitably simple takes surprising amounts of research. In part this can perhaps be attributed to our over-expectation of complexity in language. While language can be complex, a lot of it is surprisingly simple.

The systems we have discussed have been small and self-contained. They are suitable for use as a component in a larger system. If one looks at the workings of a digital assistant (such as the ones found on a smart phone), one will find many machine learning based subsystems: for speech-recognition, for intent detection, and for accomplishing subtasks within those. A complicated system such as Zara (Siddique et al. 2017) contains over a dozen separate machine learning based subsystems.

Siddique et al. (2017), "Zara Returns: Improved Personality Induction and Adaptation by an Empathetic Virtual Agent"

Zara, and defending herself from bad influences
One of the most interesting subsystems in Zara (Siddique et al. 2017), is the use of a module to detect abusive language, racism and sexism. It is a well-known problem that when allowing a learning system to learn from the open-web, such systems can pick-up bad habits (`https://www.theverge.com/2016/3/24/11297050`). The use of such an abuse detection module allows a system to be protected from such things.
This can be related to our earlier comments: with regards to the issues with WordNet's sexist definitions, and the issues that word embeddings can learn the biases representations from biases texts discussed by Bolukbasi et al. (2016) and Caliskan, Bryson, and Narayanan (2017)

There is another approach, processing huge corpora of texts, with end-to-end deep learning systems. For example using decades of user support logs to train question answering systems. Even this is based on many of the same principles we have discussed in this book. Though, at this scale other options open up, like char-RNNs which function on a character scale rather than the word-scale that we have considered here. With enough data, all the linguistic modelling concerns can be learnt into the neural network. The data and computational requirements place such things out of reach of many developers.

Most tasks do not have this scale of data. They may be small data, a few hundred examples. They may be largish data, with a few million examples. The more data you have, the more complex a model you can train, and thus the more difficult a problem you can solve. But good representations can help us when we do not have enough data. By creating word embeddings (**??**), word sense embeddings (**??**), and larger embeddings (**??**), we can leverage other larger data sources. The embeddings created using systems that are trained on other data, to perform other tasks, capture information to accomplish those tasks. We can then take those information dense embeddings, and apply them to our own tasks.

In conclusion, this book has covered popular deep neural networks for language processing. The networks have been comprehensively reviewed and explained from a practical perspective. Language modelling, vector representations, and challenging tasks such as WSD, and sentence embeddings were investigated to illustrate the use of these networks. This book shares a lot of valuable advice obtained from first-hand experience in implementing these networks. We would treat this book as a solid introduction to one of the most exciting new areas of natural language processing and computational linguistics.

Bibliography

Bolukbasi, Tolga, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam T Kalai (2016). “Man is to computer programmer as woman is to homemaker? Debiasing word embeddings”. In: *Advances in Neural Information Processing Systems*, pp. 4349–4357.

Caliskan, Aylin, Joanna J. Bryson, and Arvind Narayanan (2017). “Semantics derived automatically from language corpora contain human-like biases”. In: *Science* 356.6334, pp. 183–186. ISSN: 0036-8075. DOI: `10.1126/science.aal4230`. eprint: `http://science.sciencemag.org/content/356/6334/183.full.pdf`.

Chen, Tianqi and Carlos Guestrin (2016). “Xgboost: A scalable tree boosting system”. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, pp. 785–794.

Cortes, Corinna and Vladimir Vapnik (Sept. 1995). “Support-vector networks”. In: *Machine Learning* 20.3, pp. 273–297. ISSN: 1573-0565. DOI: `10.1007/BF00994018`.

Siddique, Farhad Bin, Onno Kampman, Yang Yang, Anik Dey, and Pascale Fung (2017). “Zara Returns: Improved Personality Induction and Adaptation by an Empathetic Virtual Agent”. In: *Proceedings of ACL 2017, System Demonstrations*, pp. 121–126.