# 8

# Selected Applications in Language Modeling and Natural Language Processing

Research in language, document, and text processing has seen increasing popularity recently in the signal processing community, and has been designated as one of the main focus areas by the IEEE Signal Processing Society's Speech and Language Processing Technical Committee. Applications of deep learning to this area started with language modeling (LM), where the goal is to provide a probability to any arbitrary sequence of words or other linguistic symbols (e.g., letters, characters, phones, etc.). Natural language processing (NLP) or computational linguistics also deals with sequences of words or other linguistic symbols, but the tasks are much more diverse (e.g., translation, parsing, text classification, etc.), not focusing on providing probabilities for linguistic symbols. The connection is that LM is often an important and very useful component of NLP systems. Applications to NLP is currently one of the most active areas in deep learning research, and deep learning is also considered as one promising direction by the NLP research community. However, the intersection between the deep learning and NLP researchers is so far not nearly as large as that for the application areas of speech or vision. This is partly because the hard evidence for the superiority of deep learning over the current state of the art NLP methods has not been as strong as speech or visual object recognition.

## 8.1 Language modeling

Language models (LMs) are crucial part of many successful applications, such as speech recognition, text information retrieval, statistical machine translation and other tasks of NLP. Traditional techniques for estimating the parameters in LMs are based on N-gram counts. Despite known weaknesses of $N$-grams and huge efforts of research communities across many fields, $N$-grams remained the state-of-the-art until neural network and deep learning based methods were shown to significantly lower the perplexity of LMs, one common (but not ultimate) measure of the LM quality, over several standard benchmark tasks [245, 247, 248].

Before we discuss neural network based LMs, we note the use of hierarchical Bayesian priors in building up deep and recursive structure for LMs [174]. Specifically, Pitman-Yor process is exploited as the Bayesian prior, from which a deep (four layers) probabilistic generative model is built. It offers a principled approach to LM smoothing by incorporating the power-law distribution for natural language. As discussed in Section 3, this type of prior knowledge embedding is more readily achievable in the generative probabilistic modeling setup than in the discriminative neural network based setup. The reported results on LM perplexity reduction are not nearly as strong as that achieved by the neural network based LMs, which we discuss next.

There has been a long history [19, 26, 27, 433] of using (shallow) feed-forward neural networks in LMs, called the NNLM. The use of DNNs in the same way for LMs appeared more recently in [8]. An LM is a function that captures the salient statistical characteristics of the distribution of sequences of words in natural language. It allows one to make probabilistic predictions of the next word given preceding ones. An NNLM is one that exploits the neural network's ability to learn distributed representations in order to reduce the impact of the curse of dimensionality. The original NNLM, with a feed-forward neural network structure works as follows: the input of the N-gram NNLM is formed by using a fixed length history of $N-1$ words. Each of the previous $N-1$ words is encoded using the very sparse 1-of-V coding, where V is the size of the vocabulary. Then, this 1-of-V orthogonal representation of words is projected linearly to a lower dimensional space, using the projection matrix shared among words at different positions in the history. This type of continuous-space, distributed representation of words is called "word embedding," very different from the common symbolic or localist presentation [26, 27]. After the projection layer, a hidden layer with nonlinear activation function, which is either a hyperbolic tangent or a logistic sigmoid, is used. An output layer of the neural network then follows the hidden layer, with the number of output units equal to the size of the full vocabulary. After the network is trained, the output layer activations represent the "$N$-gram" LM's probability distribution.

The main advantage of NNLMs over the traditional counting-based $N$-gram LMs is that history is no longer seen as exact sequence of $N-1$ words, but rather as a projection of the entire history into some lower dimensional space. This leads to a reduction of the total number of parameters in the model that have to be trained, resulting in automatic clustering of similar histories. Compared with the class-based $N$-gram LMs, the NNLMs are different in that they project all words into the same low dimensional space, in which there can be many degrees of similarity between words. On the other hand, NNLMs have much larger computational complexity than $N$-gram LMs.

Let's look at the strengths of the NNLMs again from the viewpoint of distributed representations. A distributed representation of a symbol is a vector of features which characterize the meaning of the symbol. Each element in the vector participates in representing the meaning. With an NNLM, one relies on the learning algorithm to discover meaningful, continuous-valued features. The basic idea is to learn to associate each word in the dictionary with a continuous-valued vector representation, which in the literature is called a word embedding, where each word corresponds to a point in a feature space. One can imagine that each dimension of that space corresponds to a semantic or grammatical characteristic of words. The hope is that functionally similar words get to be closer to each other in that space, at least along some directions. A sequence of words can thus be transformed into a sequence of these learned feature vectors. The neural network learns to map that sequence of feature vectors to the probability distribution over the next word in the sequence. The distributed representation approach to LMs has the advantage that it allows the model to generalize well to sequences that are not in the set of training word sequences, but that are similar in terms of their features, i.e., their distributed representation. Because neural networks tend to map nearby inputs to nearby outputs, the predictions corresponding to word sequences with similar features are mapped to similar predictions.

The above ideas of NNLMs have been implemented in various studies, some involving deep architectures. The idea of structuring hierarchically the output of an NNLM in order to handle large vocabularies was introduced in [18, 262]. In [252], the temporally factored RBM was used for language modeling. Unlike the traditional $N$-gram model, the factored RBM uses distributed representations not only for context words but also for the words being predicted. This approach is generalized to deeper structures as reported in [253]. Subsequent work on NNLM with "deep" architectures can be found in [205, 207, 208, 245, 247, 248]. As an example, Le et al. [207] describes an NNLM with structured output layer (SOUL–NNLM) where the processing depth in the LM is focused in the neural network's output representation. Figure 8.1 illustrates the SOUL-NNLM architecture with hierarchical structure in the output layers of the neural network, which shares the same architecture with the conventional NNLM up to the hidden layer. The hierarchical structure for the network's output vocabulary is in the form of a clustering tree, shown to the right of Figure 8.1, where each word belongs to only one class and ends in a single leaf node of the tree. As a result of the hierarchical structure, the SOUL–NNLM enables the training of the NNLM with a full, very large vocabulary. This gives advantages over the traditional NNLM which requires shortlists of words in order to carry out the efficient computation in training.

As another example neural-network-based LMs, the work described in [247, 248] and [245] makes use of RNNs to build large scale language

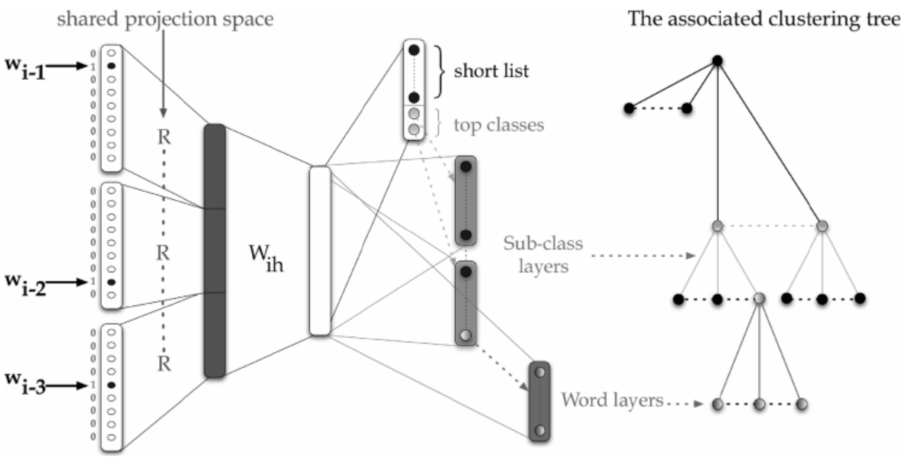

**Figure 8.1:** The SOUL–NNLM architecture with hierarchical structure in the output layers of the neural network [after [207], @IEEE].

models, called RNNLMs. The main difference between the feed-forward and the recurrent architecture for LMs is different ways of representing the word history. For feed-forward NNLM, the history is still just previous several words. But for the RNNLM, an effective representation of history is learned from the data during training. The hidden layer of RNN represents all previous history and not just $N-1$ previous words, thus the model can theoretically represent long context patterns. A further important advantage of the RNNLM over the feed-forward counterpart is the possibility to represent more advanced patterns in the word sequence. For example, patterns that rely on words that could have occurred at variable positions in the history can be encoded much more efficiently with the recurrent architecture. That is, the RNNLM can simply remember some specific word in the state of the hidden layer, while the feed-forward NNLM would need to use parameters for each specific position of the word in the history.

The RNNLM is trained using the algorithm of back-propagation through time; see details in [245], which provided Figure 8.2 to show during training how the RNN unfolds as a deep feed-forward network (with three time steps back in time).

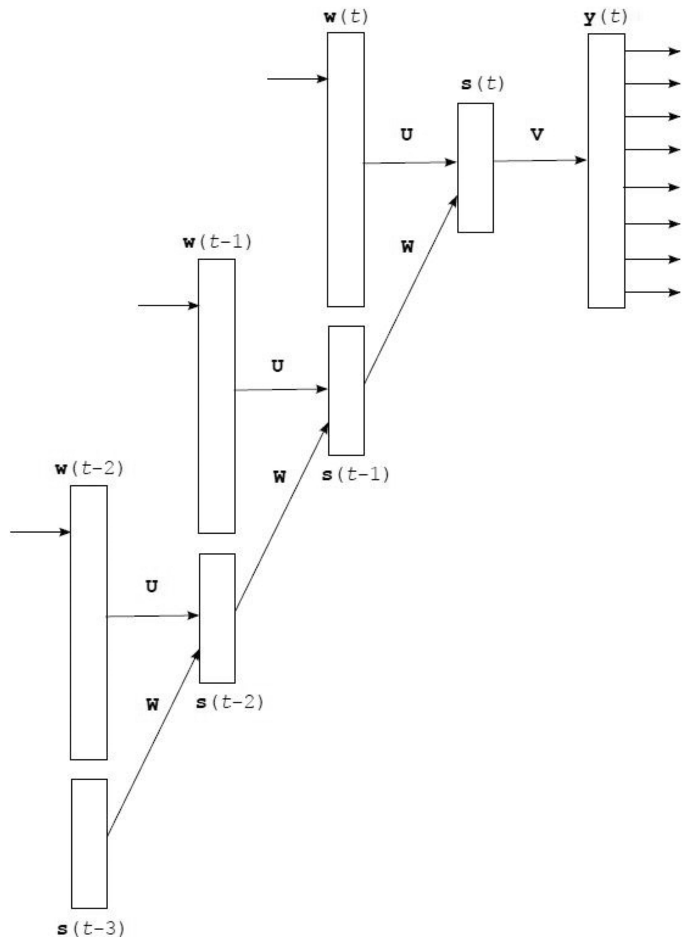

**Figure 8.2:** During the training of RNNLMs, the RNN unfolds into a deep feed-forward network; based on Figure 3.2 of [245].

The training of the RNNLM achieves stability and fast convergence, helped by capping the growing gradient in training RNNs. Adaptation schemes for the RNNLM are also developed by sorting the training data with respect to their relevance and by training the model during processing of the test data. Empirical comparisons with other state-of-the-art counting-based $N$-gram LMs show much better performance of RNNLM in the perplexity measure, as reported in [247, 248] and [245].

A separate work on applying RNN to an LM with the unit of characters instead of words can be found in [153, 357]. Many interesting properties such as predicting long-term dependencies (e.g., making open and closing quotes in a paragraph) are demonstrated. However, the usefulness of characters instead of words as units in practical applications is not clear because the word is such a powerful representation for natural language. Changing words to characters in LMs may limit most practical application scenarios and the training become more difficult. Word-level models currently remain superior.

In the most recent work, Mnih and Teh [255] and Mnih and Kavukcuoglu [254] have developed a fast and simple training algorithm for NNLMs. Despite their superior performance, NNLMs have been used less widely than standard $N$-gram LMs due to the much longer training time. The reported algorithm makes use of a method called noise-contrastive estimation or NCE [139] to achieve much faster training for NNLMs, with time complexity independent of the vocabulary size; hence a flat instead of tree-structured output layer in the NNLM is used. The idea behind NCE is to perform nonlinear logistic regression to discriminate between the observed data and some artificially generated noise. That is, to estimate parameters in a density model of observed data, we can learn to discriminate between samples from the data distribution and samples from a known noise distribution. As an important special case, NCE is particularly attractive for unnormalized distributions (i.e., free from partition functions in the denominator). In order to apply NCE to train NNLMs efficiently, Mnih and Teh [255] and Mnih and Kavukcuoglu [254] first formulate the learning problem as one which takes the objective function as the distribution of the word in terms of a scoring function. The NNLM then can be viewed as a way to quantify the compatibility between the word history and a candidate next word using the scoring function. The objective function for training the NNLM thus becomes exponentiation of the scoring function, normalized by the same constant over all possible words. Removing the costly normalization factor, NCE is shown to speed up the NNLM training over an order of magnitude.

A similar concept to NCE is used in the recent work of [250], which is called negative sampling. This is applied to a simplified version of an NNLM, for the purpose of constructing word embedding instead of computing probabilities of word sequences. Word embedding is an important concept for NLP applications, which we discuss next.

## 8.2 Natural language processing

Machine learning has been a dominant tool in NLP for many years. However, the use of machine learning in NLP has been mostly limited to numerical optimization of weights for human designed representations and features from the text data. The goal of deep or representation learning is to automatically develop features or representations from the raw text material appropriate for a wide range of NLP tasks.

Recently, neural network based deep learning methods have been shown to perform well on various NLP tasks such as language modeling, machine translation, part-of-speech tagging, named entity recognition, sentiment analysis, and paraphrase detection. The most attractive aspect of deep learning methods is their ability to perform these tasks without external hand-designed resources or time-intensive feature engineering. To this end, deep learning develops and makes use an important concept called "embedding," which refers to the representation of symbolic information in natural language text at word-level, phrase-level, and even sentence-level in terms of continuous-valued vectors.

The early work highlighting the importance of word embedding came from [62], [367], and [63], although the original form came from [26] as a side product of language modeling. Raw symbolic word representations are transformed from the sparse vectors via 1-of-V coding with a very high dimension (i.e., the vocabulary size V or its square or even its cubic) into low-dimensional, real-valued vectors via a neural network and then used for processing by subsequent neural network layers. The key advantage of using the continuous space to represent words (or phrases) is its distributed nature, which enables sharing or grouping the representations of words with a similar meaning. Such sharing is not possible in the original symbolic space, constructed by 1-of-V coding with a very high dimension, for representing words. Unsupervised learning is used where "context" of the word is used as the learning signal in neural networks. Excellent tutorials were recently given by Socher et al. [338, 340] to explain how the neural network is trained to perform word embedding. More recent work proposes new ways of learning word embeddings that better capture the semantics of words by incorporating both local and global document contexts and better account for homonymy and polysemy by learning multiple embeddings per word [169]. Also, there is strong evidence that the use of RNNs can also provide empirically good performance in learning word embeddings [245]. While the use of NNLMs, whose aim is to predict the future words in context, also induces word embeddings as its by-product, much simpler ways of achieving the embeddings are possible without the need to do word prediction. As shown by Collobert and Weston [62], the neural networks used for creating word embeddings need much smaller output units than the huge size typically required for NNLMs.

In the same early paper on word embedding, Collobert and Weston [62] developed and employed a convolutional network as the common model to simultaneously solve a number of classic problems including part-of-speech tagging, chunking, named entity tagging, semantic role identification, and similar word identification. More recent work reported in [61] further developed a fast, purely discriminative approach for parsing based on the deep recurrent convolutional architecture. Collobert et al. [63] provide a comprehensive review on ways of applying unified neural network architectures and related deep learning algorithms to solve NLP problems from "scratch," meaning that no traditional NLP methods are used to extract features. The theme of this line of work is to avoid task-specific, "man-made" feature engineering while providing versatility and unified features constructed automatically from deep learning applicable to all natural language processing tasks. The systems described in [63] automatically learn internal representations or word embedding from vast amounts of mostly unlabeled training data while performing a wide range of NLP tasks.

The recent work by Mikolov et al. [246] derives word embeddings by simplifying the NNLM described in Section 8.1. It is found that the NNLM can be successfully trained in two steps. First, continuous word vectors are learned using a simple model which eliminates the

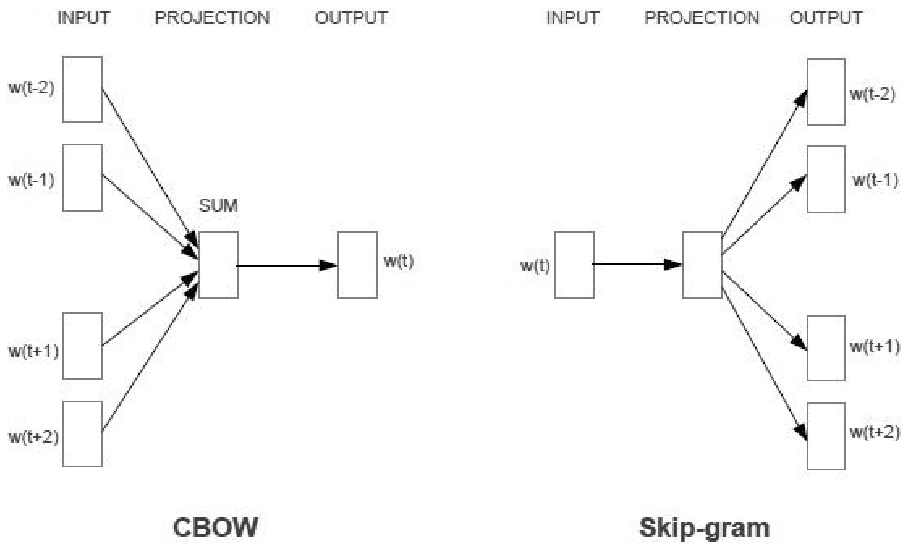

**Figure 8.3:** The CBOW architecture (a) on the left, and the Skip-gram architecture (b) on the right. [after [246], @ICLR].

nonlinearity in the upper neural network layer and share the projection layer for all words. And second, the $N$-gram NNLM is trained on top of the word vectors. So, after removing the second step in the NNLM, the simple model is used to learn word embeddings, where the simplicity allows the use of very large amount of data. This gives rise to a word embedding model called Continuous Bag-of-Words Model (CBOW), as shown in Figure 8.3a. Further, since the goal is no longer computing probabilities of word sequences as in LMs, the word embedding system here is made more effective by not only to predict the current word based on the context but also to perform inverse prediction known as “Skip-gram” model, as shown in Figure 8.3b. In the follow-up work [250] by the same authors, this word embedding system including the Skip-gram model is extended by a much faster learning method called negative sampling, similar to NCE discussed in Section 8.1.

In parallel with the above development, Mnih and Kavukcuoglu [254] demonstrate that NCE training of lightweight word embedding models is a highly efficient way of learning high-quality word representations, much like the somewhat earlier lightweight LMs developed by Mnih and Teh [255] described in Section 8.1. Consequently, results that used to require very considerable hardware and software infrastructure can now be obtained on a single desktop with minimal programming effort and using less time and data. This most recent work also shows that for representation learning, only five noise samples in NCE can be sufficient for obtaining strong results for word embedding, much fewer than that required for LMs. The authors also used an "inversed language model" for computing word embeddings, similar to the way in which the Skip-gram model is used in [250].

Huang et al. [169] recognized the limitation of the earlier work on word embeddings in that these models were built with only local context and one representation per word. They extended the local context models to one that can incorporate global context from full sentences or the entire document. This extended models accounts for homonymy and polysemy by learning multiple embeddings for each word. An illustration of this model is shown in Figure 8.4. In the earlier work by the same research group [344], a recursive neural network with local context was developed to build a deep architecture. The network, despite missing global context, was already shown to be capable of successful

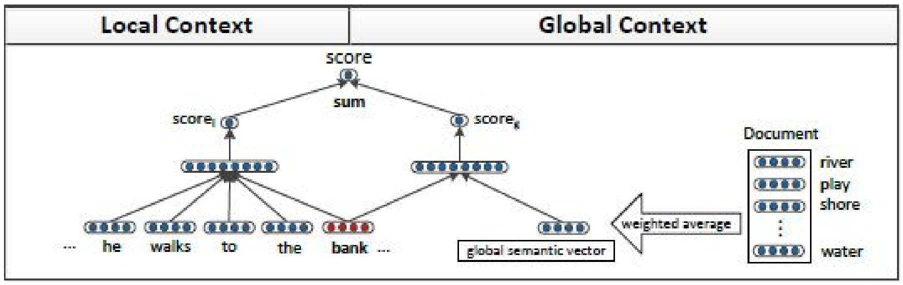

**Figure 8.4:** The extended word-embedding model using a recursive neural network that takes into account not only local context but also global context. The global context is extracted from the document and put in the form of a global semantic vector, as part of the input into the original word-embedding model with local context. Taken from Figure 1 of [169]. [after [169], @ACL].

merging of natural language words based on the learned semantic transformations of their original features. This deep learning approach provided an excellent performance on natural language parsing. The same approach was also demonstrated to be reasonably successful in parsing natural scene images. In related studies, a similar recursive deep architecture is used for paraphrase detection [346], and for predicting sentiment distributions from text [345].

We now turn to selected applications of deep learning methods including the use of neural network architectures and word embeddings to practically useful NLP tasks. Machine translation is one of such tasks, pursued by NLP researchers for many years based typically on shallow statistical models. The work described in [320] are perhaps the first comprehensive report on the successful application of neural-network-based language models with word embeddings, trained on a GPU, for large machine translation tasks. They address the problem of high computation complexity, and provide a solution that allows training 500 million words with 20 hours. Strong results are reported, with perplexity down from 71 to 60 in LMs and the corresponding BLEU score gained by 1.8 points using the neural-network-based language models with word embeddings compared with the best back-off LM.

A more recent study on applying deep learning methods to machine translation appears in [121, 123], where the phrase-translation component, rather than the LM component in the machine translation system is replaced by the neural network models with semantic word embeddings. As shown in Figure 8.5 for the architecture of this approach, a pair of source (denoted by $f$) and target (denoted by $e$) phrases are projected into continuous-valued vector representations in a low-dimensional latent semantic space (denoted by the two $y$ vectors).Then their translation score is computed by the distance between the pair in this new space. The projection is performed by two deep neural networks (not shown here) whose weights are learned on parallel training data. The learning is aimed to directly optimize the quality of end-to-end machine translation results. Experimental evaluation has been performed on two standard Europarl translation tasks used by the NLP community, English–French and German–English. The results show that the new semantic-based phrase translation model significantly

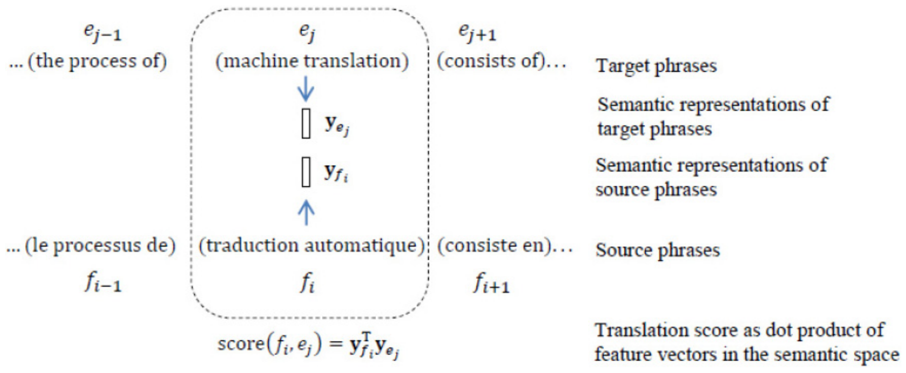
$e_{j-1}$
... (the process of)
$e_j$
(machine translation)
$e_{j+1}$
(consists of)...
Target phrases
$\mathbf{y}_{e_j}$
Semantic representations of target phrases
$\mathbf{y}_{f_i}$
Semantic representations of source phrases
... (le processus de)
(traduction automatique)
(consiste en)...
Source phrases
$f_{i-1}$
$f_i$
$f_{i+1}$
$\text{score}(f_i, e_j) = \mathbf{y}_{f_i}^{\mathrm{T}} \mathbf{y}_{e_j}$
Translation score as dot product of feature vectors in the semantic space

**Figure 8.5:** Illustration of the basic approach reported in [122] for machine translation. Parallel pairs of source (denoted by f) and target (denoted by e) phrases are projected into continuous-valued vector representations (denoted by the two $y$ vectors), and their translation score is computed by the distance between the pair in this continuous space. The projection is performed by deep neural networks (denoted by the two arrows) whose weights are learned on parallel training data. [after [121], @NIPS].

improves the performance of a state-of-the-art phrase-based statistical machine translation system, leading to a gain close to 1.0 BLEU point.

A related approach to machine translation was developed by Schwenk [320]. The estimation of the translation model probabilities of a phrase-based machine translation system is carried out using neural networks. The translation probability of phrase pairs is learned using continuous-space representations induced by neural networks. A simplification is made that decomposes the translation probability of a phrase or a sentence to a product of n-gram probabilities as in a standard $n$-gram language model. No joint representations of a phrase in the source language and the translated version in the target language are exploited as in the approach reported by Gao et al. [122, 123].

Yet another deep learning approach to machine translation appeared in [249]. As in other approaches, a corpus of words in one language are compared with the same corpus of words translated into another, and words and phrases in such bilingual data that share similar statistical properties are considered equivalent. A new technique is proposed that automatically generates dictionaries and phrase tables that convert one language into another. It does not rely on versions of the same document in different languages. Instead, it uses data mining techniques to model the structure of a source language and then compares it to the structure of the target language. The technique is shown to translate missing word and phrase entries by learning language structures based on large monolingual data and mapping between languages from small bilingual data. It is based on vector-valued word embeddings as discussed earlier in this chapter and it learns a linear mapping between vector spaces of source and target languages.

An earlier study on applying deep learning techniques with DBNs was provided in [111] to attack a machine transliteration problem, a much easier task than machine translation. This type of deep architectures and learning may be generalized to the more difficult machine translation problem but no follow-up work has been reported. As another early NLP application, Sarikaya et al. [318] applied DNNs (called DBNs in the paper) to perform a natural language call–routing task. The DNNs use unsupervised learning to discover multiple layers of features that are then used to optimize discrimination. Unsupervised feature discovery is found to make DBNs far less prone to overfitting than the neural networks initialized with random weights. Unsupervised learning also makes it easier to train neural networks with many hidden layers. DBNs are found to produce better classification results than several other widely used learning techniques, e.g., maximum entropy and boosting based classifiers.

One most interesting NLP task recently tackled by deep learning methods is that of knowledge base (ontology) completion, which is instrumental in question-answering and many other NLP applications. An early work in this space came from [37], where a process is introduced to automatically learn structured distributed embeddings of knowledge bases. The proposed representations in the continuous-valued vector space are compact and can be efficiently learned from large-scale data of entities and relations. A specialized neural network architecture, a generalization of “Siamese” network, is used. In the follow-up work that focuses on multi-relational data [36], the semantic matching energy model is proposed to learn vector representations for both entities and relations. More recent work [340] adopts an alternative approach, based on the use of neural tensor networks, to attack the problem of reasoning over a large joint knowledge graph for relation classification. The knowledge graph is represented as triples of a relation between two entities, and the authors aim to develop a neural network model suitable for inference over such relationships. The model they presented is a neural tensor network, with one layer only. The network is used to represent entities in a fixed-dimensional vectors, which are created separately by averaging pre-trained word embedding vectors. It then learn the tensor with the newly added relationship element that describes the interactions among all the latent components in each of the relationships. The neural tensor network can be visualized in Figure 8.6, where each dashed box denotes one of the two slices of the tensor. Experimentally, the paper [340] shows that this tensor model can effectively classify unseen relationships in WordNet and FreeBase.

As the final example of deep learning applied successfully to NLP, we discuss here sentiment analysis applications based on recursive deep

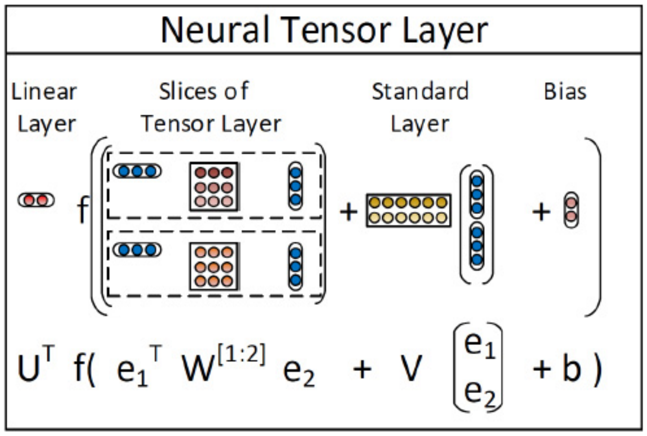

**Figure 8.6:** Illustration of the neural tensor network described in [340], with two relationships shown as two slices in the tensor. The tensor is denoted by $\boldsymbol{W}^{[1:2]}$. The network contains a bilinear tensor layer that directly relates the two entity vectors (shown as $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$) across three dimensions. Each dashed box denotes one of the two slices of the tensor. [after [340], @NIPS].

models published recently by Socher et al. [347]. Sentiment analysis is a task that is aimed to estimate the positive or negative opinion by an algorithm based on input text information. As we discussed earlier in this chapter, word embeddings in the semantic space achieved by neural network models have been very useful but it is difficult for them to express the meaning of longer phrases in a principled way. For sentiment analysis with the input data from typically many words and phrases, the embedding model requires the compositionality properties. To this end, Socher et al. [347] developed the recursive neural tensor network, where each layer is constructed similarly to that of the neural tensor network described in [340] with an illustration shown in Figure 8.6. The recursive construction of the full network exhibiting properties of compositionality follows that of [344] for the regular, non-tensor network. When trained on a carefully constructed sentiment analysis database, the recursive neural tensor network is shown to outperform all previous methods on several metrics. The new model pushes the state of the art in single sentence positive/negative classification accuracy from 80% up to 85.4%. The accuracy of predicting fine-grained sentiment labels for all phrases reaches 80.7%, an improvement of 9.7% over bag-of-features baselines.