**2 Preliminaries**

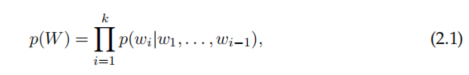
This work exploits the interaction between language models, vector-based representations and semantic representations. In this Chapter we provide the essential background to such topics. The Chapter is organized as follows: first we introduce language models (Section 2.1), where we report on both N-grams (Section 2.1.1)

and neural language models together with the main traits of the neural architectures actually employed to acquire such models (Section 2.1.2); the second part of the Chapter is focused on lexical resources (Section 2.2) illustrating distributional resources (Section 2.2.1) and semantic networks (Section 2.2.2).

**2.1 Language Models**

Language models are statistical inference tools that allows estimating the probability of a word sequence (Goldberg, 2017; C. Manning & Schutze, 1999). For example, a language model is able to assign the probability to the sentence *I shot an elephant in my pajamas*. More frequently, language models are employed to compute the probability of seeing a given word following a sequence of words —usually called context—. For example, what is the probability of seeing the word *pajamas* after the sequence *I shot an elephant in my*? The probabilities as signed by language models are the result of a learning process, i.e. the training phase, in which the model is exposed to a particular kind of textual data —i.e. the training corpus—. The goal of the training process is to teach the model to predict sentences that closely resemble the sentences seen during learning.

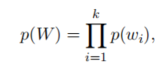
Formally, the Language Modeling task is defined as the assignment of probability to any possible sequence of words *W* = *{w*1 *. . . wk}*, so as to compute *p*(*W*) (Goldberg, 2017). Such probability can be computed as



where the probability of each word is conditioned on the preceding context. Depending on the adopted language model as well as the assumptions on the conditioning factor the probability of the sentence may be framed differently.

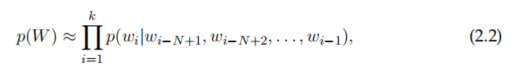
**2.1.1 N-grams**

The simplest idea is to consider words individually, that is, each word is a single unit, this is what is called the unigram model. A unigram model considers no preceding context, given the sequence of words *W* = *{w*1 *. . . wk}*, the probability of the sequence is defined as:



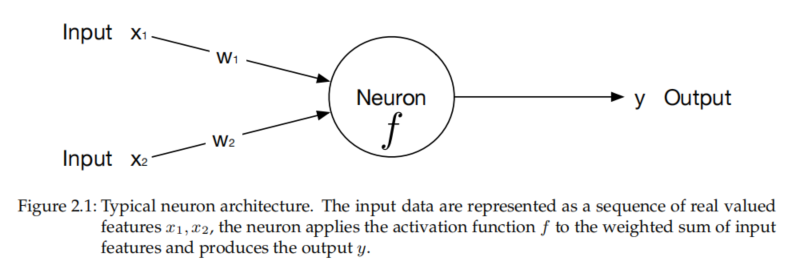
where the probability of the *i*-th word *p*(*wi*) can be estimated by exploiting the frequency of the word *wi* in the training corpus. The natural extensions of unigram models, are the *N*-gram models, where *N* is an integer indicating the size of the context, i.e. the preceding *N −* 1 words are exploited to estimate the probability of

the *N*-th term of the sequence (Jurafsky & Martin, 2014). Formally, in the *N*-gram setting, the probability of the sequence *W* is defined as:



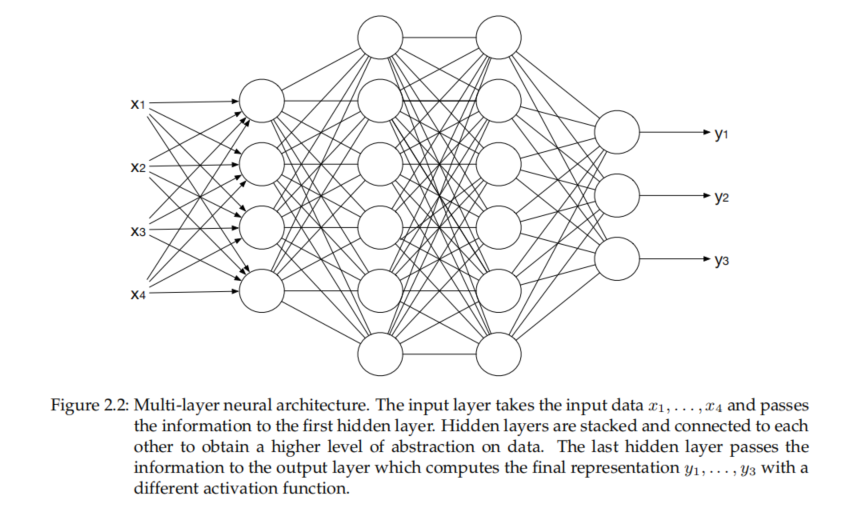
where the probability of each word is conditioned on the preceding context. In this case only blocks of few (exactly *N*) words are considered to predict the whole *W*: we can thus predict the word sequence based on N-grams, that are blocks of two, three or four preceding elements (bi-grams, tri-grams, four-grams, respectively). Since the *N*-gram models are statistical models, their knowledge strictly depends

on the training corpus, that is, the probability distribution reflects the language property of the training corpus itself. The natural consequence of estimating the probability of the upcoming word, relying on the preceding *N −*1 words, is that *N*-gram models obtain better performances at representing the training corpus as *N* increases, with the drawback of making the estimation of *p*(*wk|w*1 *. . . wk−*1) harder

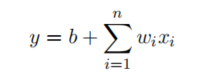


Increasing the context size, also, involves dealing with data sparsity drawback: the larger the context the less likely it is to find more than one sequence with the same length in the training corpus. That is, the probability for all the *N*-grams that does not occur in the training corpus has to be estimated, this gives rise to plenty of sequences with the same low probability and few sequences with high probability. In order to deal with *N*-grams not occurring in the training corpus, called outof-vocabulary *N*-grams, language models have been provided with an additional step of regularization, to allow a non-zero probability to unseen *N*-grams (Gale & Church, 1994; Kneser & Ney, 1995).

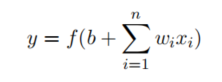
**2.1.2 Neural Networks and Neural Language Models**

Neural networks are computational systems inspired by the human brain. The history of neural networks started with the McCulloch-Pitt neuron —or unit—, that is, a computational model of human neurons, entirely described with propositional logic (McCulloch & Pitts, 1943). Modern neural networks organize neurons into layers, each unit is connected to each unit of the subsequent layer through *synapses* or *edges*. Each layer of the network accepts as input the output of the preceding layer, performs some transformation on the received data, and produces an output according to the layer architecture. Different layers apply different transformations; edges, in turn, are usually provided with a weight, an integer expressing the strength of the connection among two neurons, which is usually exploited to alter data coming from the preceding layer. A graphical representation of a simple neuron is depicted in Figure 2.1. A single unit receives as input real valued features,

*x*1, *x*2, *. . .* , *xn*, which represent the input data, then, it combines the input features through a weighted sum:



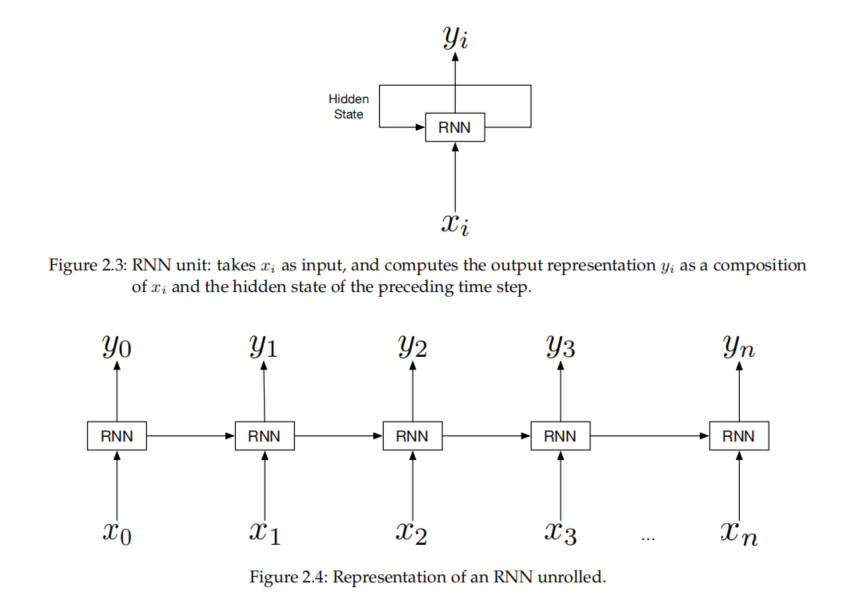
where *wi* are the weight of the edges connecting data with the input layer and *b* is a bias term which is an additional term that allows shifting the data transformation by adding a constant. Additionally, each neural unit is then provided with an activation function, that is, a non-linear function applied to the weighted sum of input features and bias term. So the final output value *y* of a neural unit is:



where *f* is the activation function of the single neuron. Different activation functions correspond to different transformation, and several functions may be employed depending on the addressed task.1 Neurons are organized into layers, and each layer may play a different role in the architecture, also according to the activation function. An example of a multi-layer neural network is depicted in Figure 2.2. In particular, the layers can be partitioned in three macro-categories according to their position: (i) the input layer, which is dedicated to take input data, represented as numerical features, and to pass information to the hidden layer; (ii) hidden layer, made of non exposed units, is the heart of neural computation, all the major transformations happen here, and all neural units of the layer share the same activation function; (iii) output layer, is the final layer of the architecture, usually apply a different function, with respect to hidden layers, so to build the final representation.

The role played by hidden layers is fundamental in computing the final representation. Each hidden layer provides additional abstraction to the neural model: in fact, hidden layers can be stacked one on each other to obtain a higher abstraction level. Architectures provided with multiple hidden layers are usually called deep neural networks. Since neural networks deal with real valued representations of data, it is essential to extract features from data, which are texts in our case, and map them to a numerical vector representation —usually called embedding— able to fully grasp the key features from the input data. The role of vector representations is central to neural models: modern neural networks are provided with an embedding layer, which is responsible for the creation of a fixed-length vector for each element of the input sequence. It is worth noting that these vector representations mitigate the data sparsity problem by building a continuous space, each word has its corresponding vector in the network space. Since language models deal with textual data, we report on the most popular neural networks for Natural Language Processing (NLP).

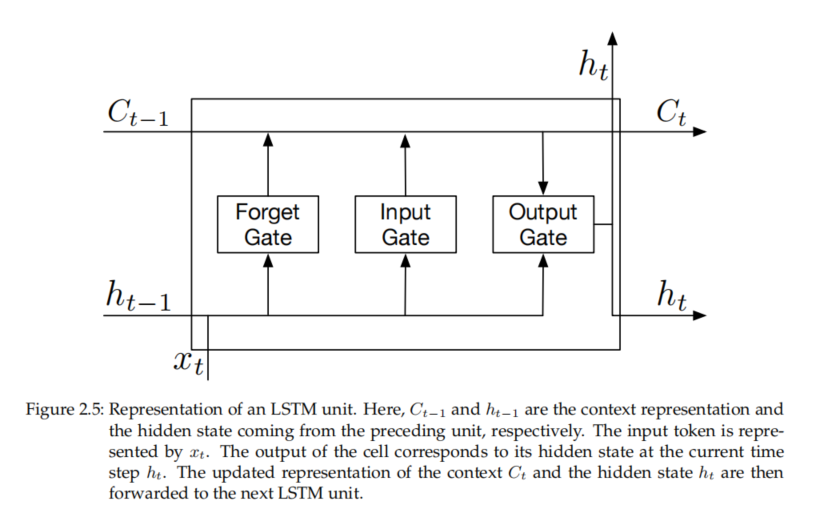
In general, dealing with sentences involves dealing with ordered sequences of words: in order to fully seize the meaning of a sentence, a model should be able to account for word ordering information. Given the relevance of modelling the word order in sentences, one of the most employed architectures is the Recurrent Neural Network (RNN). RNNs have been largely employed in text processing due their ability of processing input data as sequences: that is, RNNs are able to grasp and model word order (Elman, 1990). RNNs are particularly suited to process sequence data thanks to the internal loop they are provided with, that is, the input of each unit is conditioned by the output of its own output at the preceding iteration. Figure 2.3 shows a graphical illustration of a single RNN unit. The hidden state encodes a memory for the context, it provides all the additional information that is exploited to compute the output of later time steps.



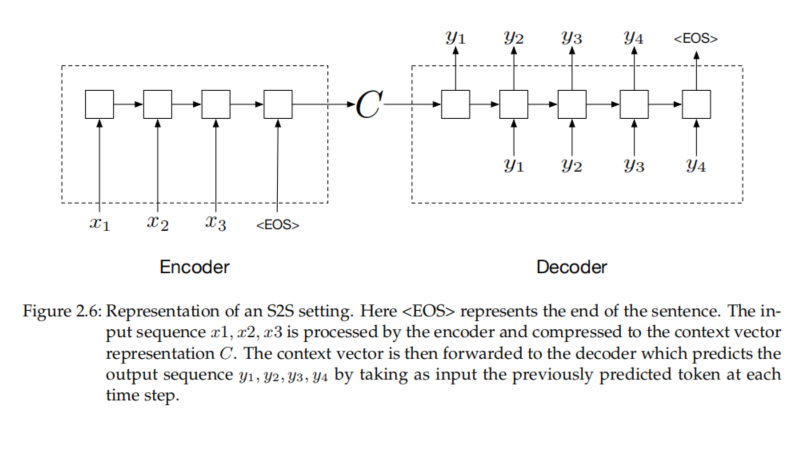
An RNN model does not fix a limit for the length of the input sequence, however, for a finite input sequence, an RNN model can be represented as unrolled, a graphical illustration is shown in Figure 2.4. In the unrolled setting we can see that the last hidden state depends on the entire input sequence, in that the prediction of the next word is conditioned on the previous words in the sentence. The ability of conditioning the prediction of the next word to the preceding context, that is, dealing with sequences, is the mostt appealing feature of the RNN architectures, nevertheless, these models struggle in modeling the context when facing long range dependencies. More precisely, some tasks can be addressed by accounting for the recent information only, for example the five words before the next one. Conversely, to tackle other tasks, we need the information from the whole sequence, for example, let us consider the sentence *I was born in Italy and then, to follow my work, I moved to Germany. However, my family still lives in Italy.*, to predict the last word *Italy* we should exploit the information at the beginning of the sentence, that is, the distance between the prediction and the useful context is quite large. Unfortunately, however, RNNs are progressively less suited to model dependencies as the intervening distance between dependents grows (Bengio, Simard, & Frasconi, 1994).

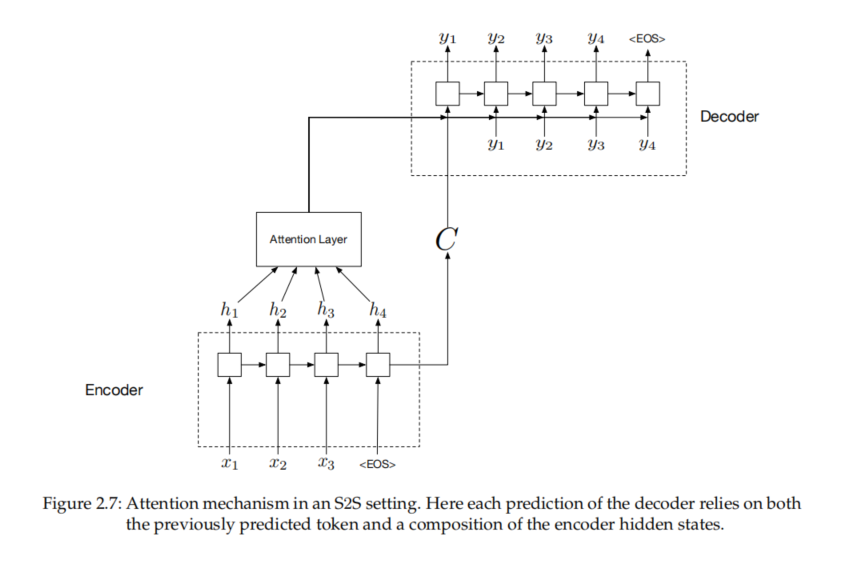
Given the difficulties in seizing long range dependencies, RNNs were quickly replaced by the Long Short-Term Memory networks (LSTM) (Hochreiter & Schmidhuber, 1997). LSTMs are RNNs specifically devised to learn long range dependencies; this is obtained by providing units with an explicit context memory that conveys the information about the preceding context through time. The context representation is performed through two main operations: (i) forgetting information no longer needed from the context and (ii) adding new information probably needed for next word prediction. Both sub-tasks are addressed through specialized neural units called gates, that manage the information flow through the memory state and the output of the LSTM cell. These gates follow the same design pattern: a neural layer followed by a sigmoid activation; the output is then combined through multiplication with the information to be filtered. The sigmoid function, combined with point-wise multiplication, allows the gate to decide whether to retain information (that is by setting the output to 1) or to forget the information, setting the output to 0. More precisely, an LSTM unit is composed by three gates: the forget gate, which is responsible for deleting information no longer needed from the context representation; the input gate, concerned with adding new information to the context representation; the output gate, that is aimed at computing the output representation, which coincides with the unit hidden state, by accounting for the preceding hidden state and the new context representation. A graphical illustration of an LSTM unit is provided in Figure 2.5.

LSTMs are particularly suited to deal with sequences and long range dependencies. Despite their abilities, simple LSTM models can only work with fixed length input sequences, but some tasks such as machine translation or speech recognition are likely better expressed by dealing with sequences whose length is not fixed. The Sequence-to-Sequence (S2S) model has been proposed in 2014 (Sutskever, Vinyals, & Le, 2014) to overcome such limitations. The S2S model relies on LSTMs to map a sequence *x*1, *. . .* , *xn* of an arbitrary length to another sequence *y*1, *. . .* , *yk* where *k* may be different from *n*. In this setting, the input sequence is processed by an encoder, which compresses the sequence to a fixed length vector representation *C*. The decoder is then initialized on the *C* vector, and predicts the output token by token, accounting for the previously predicted token at each time step. A graphical representation of the S2S architecture is depicted in Figure 2.6.



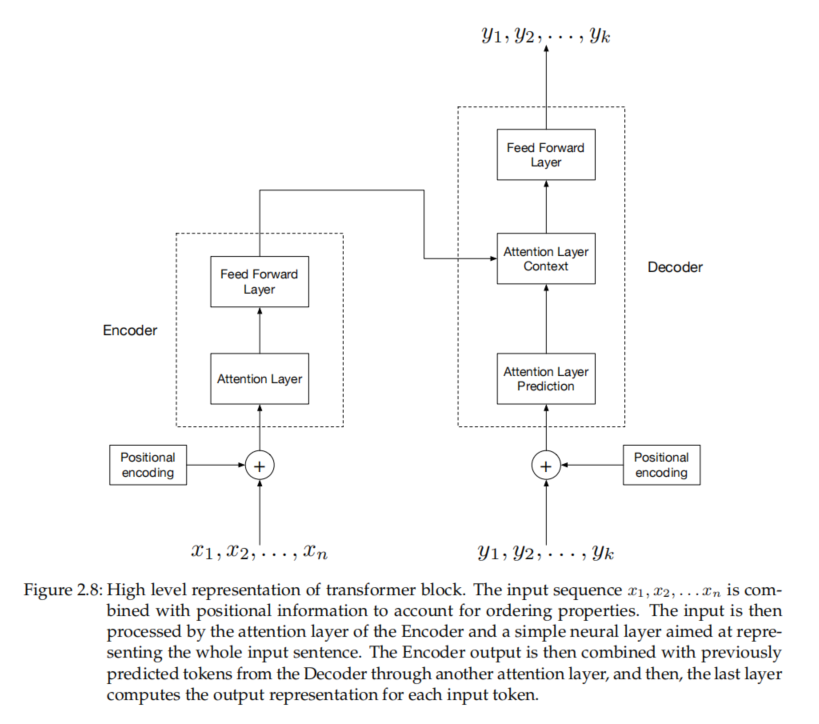
Despite the ability of LSTM architectures to deal with long range dependencies, these models still struggle in representing larger pieces of text and suffer from high training time due to the recurrent connections which build these units. Additionally, the S2S architecture suffers from the loss of informative load in compressing the whole input sequence into a single fixed length vector representation. Transformers (Vaswani et al., 2017), together with the attention mechanism (Bahdanau, Cho, & Bengio, 2014) alleviate these problems by both increasing the amount of exploited information from the context, and getting rid of the recurrent connections. The attention mechanism has been designed to alleviate the difficulties in S2S models; this is done by allowing the decoder to directly exploit the encoder’s hidden states rather than just using the final context representation provided by the encoder itself. Adopting an attention mechanism allows the model to selectively focus on parts of the input that are likely the most useful. The attention mechanism is particularly suited to address tasks which need to take decisions relying on parts of the input data. An illustration of the attention mechanism fitted in the S2S setting is depicted in Figure 2.7. Attention plays a key role in the Transformer architecture, in particular, the model illustrated in Figure 2.7 follows the S2S design pattern





where the encoder processes the input sequence, the output is then forwarded to the decoder which curates the output predictions. In this setting, we will refer to the encoder-decoder model as the Transformer block. Since Transformers get rid of recurrent connections, which allow models to deal with sequences, the encoder represents the input through a combination of word representations and information about the position of words in the input sentence. In so doing, the model is able to account for ordering information. After this first operation, the encoder block is made of an attention layer followed by a simple neural layer which is deputed to compute the context representation. As for encoder, the decoder block combines the previously predicted word representations with the positional information to keep track of the order of the words, and forwards these vectors through an attention layer that selects the most useful information among the predictions. After these first steps the decoder combines the information from previously predicted tokens with the context representation, from the encoder, through another attention layer, and finally a simple neural layer is concerned with computing the output representation. Most popular models consist of several Transformer blocks stacked one on each other, this allows the model to increase its abstraction capabilities, the more the number of stacked blocks the more abstract the representation that can be calculated. A graphical illustration of a transformer block is provided in Figure 2.8.

Neural language models (NLMs) are language models based on neural networks. Such models improve the language modeling capabilities of *n*-grams by exploiting the ability of neural networks to deal with longer histories. Additionally, neural models do not need regularization steps for unseen *n*-grams and address the data sparsity curse of *n*-grams by dealing with distributed representation. The predictive power of neural language models is higher than *n*-gram language models given the same training set. Despite the great improvement of neural language models on NLP tasks, these models are affected by higher training time rather than *n*-gram language models. Since the introduction of Transformers, such models have been widely adopted and improved to address diverse Natural Language Understanding benchmarks such as GLUE (Wang et al., 2018) and SuperGLUE (Wang et al., 2019). With the introduction of highly-scalable Transformer architectures two kinds of very deep NLMs emerged: causal (or left-to-right) models, primarily represented by the Generative Pre-trained Transformer (Radford et al., 2019) where the objective is to predict the next word given a past sequence of words; and masked models, where the objective is to predict a masked (i.e., hidden) word given its surrounding words, of which the most prominent example is the Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018). The difference in training objectives results in these two varieties of NLMs specializing at different tasks, with causal models excelling at language generation and masked models at language understanding. Models such as BERT (Devlin et al., 2018) and GPT-2 (Radford et al., 2019) started a whole host of experiments and triggered an intense research activity to improve such models (Brown et al., 2020; Lan et al., 2019; Y. Liu et al., 2019; Raffel et al., 2019; Yang et al., 2019). Given the relevance of the two mentioned models, we briefly report on both BERT and GPT-2 as precursors to more recent lines of NLP research as well as responsible for main advances in addressing NLP benchmarks.

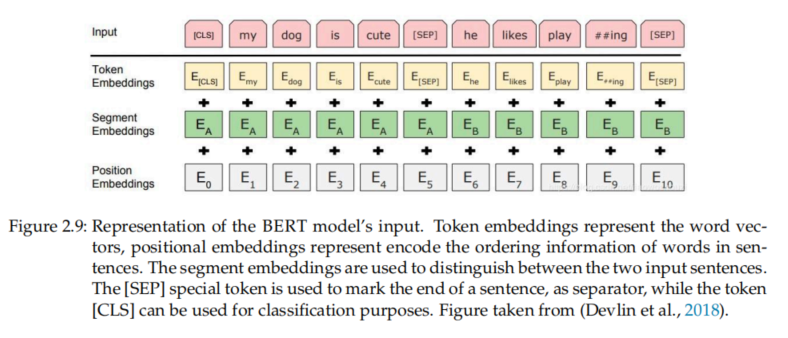


**BERT**

BERT is a large language model based on Transformers, but it differs in the training objective: the key innovation is applying the bidirectional training of Transformers to language modeling. Using BERT involves to deal with two different rather important phases: *pre-training*, where the model is exposed to unlabeled textual data so as to learn the main features from the type of employed texts; and *fine-tuning*, where the pre-trained model is fine-tuned to address a specific task by exploiting labeled data. During the pre-training phase BERT is designed to address two unsupervised predictive tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). The MLM is what gives BERT the bidirectional attribute: in this setting, a small percentage of the input tokens —15% in the released models— is masked and the training objective is to predict those masked tokens. For example, the sentence *I shot an elephant in my pajamas* may be rewritten into *I shot an [MASK] in my pajamas* and the BERT target is to predict the word *elephant* istead of the *[MASK]* token. Unlike causal language models pre-training, the MLM objective allows representing both left and right context thus making the training bidirectional at the cost of misaligning the pre-training with the fine-tuning: the *[MASK]* token does not appear during fine-tuning. To mitigate the misalignment issue, the pre-training process does not always replace the masked word with the *[MASK]* token; in some cases (20%), the word is replaced with another word randomly taken from the dictionary or just left unaltered the word itself. The loss function is computed by accounting for the masked tokens only, thus ignoring the prediction on non-masked words. The NSP training objective has been devised to allow BERT to learn the relationship between sentences. The NSP involves selecting pairs of sentences A and B: in half of the cases the sentence B directly follows the sentence A, while in the remaining half the sentence B is randomly selected within a textual corpus. The purpose of the training objective is to learn whether sentence B directly follows A or not. Since many NLP tasks involve learning the relationship between sentences, such as, for example, Recognizing Textual Entailment (RTE) or Question Answering (QA), the NSP objective is to strengthen the model by precisely learning the relationship between pairs of sentences.

The BERT architecture follows the Transformer’s design pattern reported in Figure 2.8; the difference is that BERT is made of stacked Transformers blocks, one on another, to increase the abstraction level. Stacking Transformers blocks not only allows dealing with more and more abstract representations, but also to reproduce the NLP pipeline, that is, different BERT layers deal with different linguistic levels (Tenney, Das, & Pavlick, 2019; Tenney, Xia, et al., 2019). In Figure 2.9 we report a graphical illustration of the BERT model’s input as provided by the authors (Devlin et al., 2018). Given that BERT has to deal with pair of sentences as well as masked tokens, the authors, provided the model with two special tokens, the *[CLS]* token is placed at the beginning of the first sentence and is used from following classification layers placed on top of BERT, while the token *[SEP]* is used as separator, to mark the end of a sentence. The sentence embeddings reported in the figure are used to distinguish between the sentence A and B for the NSP task, while the positional information is accounted through the positional embeddings.

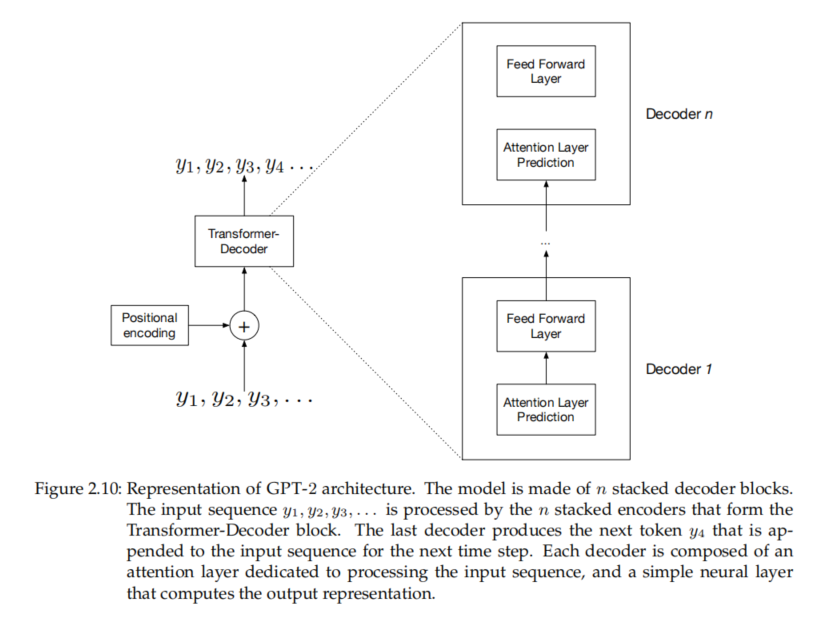
Once the pre-training of the model has been completed, the model may be fine-tuned to address a specific NLP task. For each downstream NLP task a fine-tuning process is needed, in particular, to specialize a pre-trained model is sufficient to plug a classification layer on top of BERT relying on the encoded representation for words as well as for the [CLS] token. For example, a sentence pair classification task such as QA or RTE, a simple classification layer may be plugged on top of the Transformer’s output relying on the [CLS] token. We refer the readers to Devlin et al. (2018) for an exhaustive list of design pattern for different kind of task.



**GPT-2**

GPT-2 is a large language model based on Transformers and trained to predict the next word given the preceding context (Radford et al., 2019). GPT-2, like traditional language models, predicts one token at a time, and the new prediction is appended to the input sequence for next time step. Inspired by P. J. Liu et al. (2018), which proposed the Transformer-Decoder architecture, GPT-2 is made of stacked decoder blocks only. More precisely, the Transformer-Decoder block is very similar to the decoder of the Transformer architecture; it simply gets rid of the encoder block as the contextual attention layer of the decoder. The GPT-2 architecture is portrayed in Figure 2.10.

The GPT-2 model has been trained 40GB of Internet text carefully selected for quality, that is a selection of documents curated or approved by humans. One main trait featuring the training data selection is that many different domains have been exploited as data sources; this allows the neural network to model language properties from each such domain, avoiding a strong polarization for a single one. Additionally, it is worth noting that the number of stacked decoder blocks impacts on performances, as the number of levels increases the language modeling capabilities improve.



**2.2 Lexical Resources**

Since the purpose of this work is to build lexical resources on top of language models, we will briefly report on two families of lexical resources. In particular we are interested in lexical resources aimed at representing word senses according to two different principles: distributional resources, that represent word senses through dense vectorial representations, and semantic networks that represent word senses

as vertices of a labeled graph.

**2.2.1 Distributional resources**

Pioneering works in the vector semantics area postulated that the meaning of a word always related to the context in which it occurs, depending on its usage in a language (Firth, 1935). This kind of meaning representation is known as the Distributional Hypothesis (Harris, 1954). The distributional hypothesis states that words that occur in similar contexts tend to convey similar meanings, for example if the word *wi* and the word *wk* often occur in the same context, then they probably have close meanings; if they are interchangeable in the same contexts of occurrence, then they are synonyms. For example, in the sentences ‘We used the *board* to shut down the power plant’ and ‘We used the *panel* to shut down the power plant’, the words *board* and *panel* are intended with the same meaning.

Several techniques have been devised to acquire the distributional profiles of terms, usually in the form of dense unit vectors of real numbers over a continuous, high-dimensional Euclidean space. In this setting each word can be described through a vector, usually called *word embedding*, and each such vector can be mapped onto a multidimensional space where distance (such as, e.g., the Euclidean distance between vectors) acts like a proxy for similarity, and similarity can be interpreted as a metric. As a result, words with similar semantic content are expected to be closer than words semantically dissimilar. Early works in this family started by generating vectors from co-occurrence matrices (Harman, 1993; Schütze & Pedersen, 1997), optionally treated with latent semantic indexing (Landauer, Foltz, & Laham, 1998), or point-wise mutual information (Hindle, 1990). Historically, such early distributional representations provided *explicit* (that is, directly meaningful and human interpretable) information: the features of such vectors were composed by, e.g., binary values, or probabilistic measures (Navigli & Martelli, 2019). The number of dimensions of such vectors was determined by the size of the vocabulary.

On the other side, in *implicit* or latent representations, features were used resulting from Latent Semantic Analysis (LSA). LSA is a multidimensional associative model based on the distributional hypothesis: word meaning is encoded as a multi-dimensional (usually 300 or 400 dimensions) vector obtained by elaborating large *corpora* to estimate the co-occurrence frequencies for each word. In order to assess the quality of vectors built through LSA, such representations have been assessed on synonymy tests in (Landauer & Dumais, 1997), finding that these embeddings performed comparably to school-aged children, when measuring similarity between word pairs as the cosine similarity between their corresponding embeddings. Given the interesting features of word embeddings, further energies have been invested in building vectorial representation through neural networks, in particular it has been shown how NLMs implicitly develop word embeddings when training for the word prediction task (Bengio et al., 2003). The research has rapidly progressed demonstrating that word embeddings could be incorporated into neural architectures for various NLP tasks (Collobert & Weston, 2007, 2008). In particular, the use of pre-trained word vectors for initializing the embedding layer of a task-specific network is an instance of multi-task learning, with language modeling as a supporting task (Goldberg, 2017, p. 243).

Despite the impact of word embeddings on the NLP area, the design of such representation implicitly includes a major limitation: it ignores the fact that words can have multiple meanings and conflates all these meanings into a single representation (Camacho-Collados & Pilehvar, 2018). Such limitation, also referred as Meaning Conflation Deficiency, may affect the semantic understanding of an NLP system that uses word embeddings at its core: word embeddings seem to be unable to grasp different meanings of a word, even if those meanings occur in the training corpus (Schütze, 1998; Yaghoobzadeh & Schütze, 2016). Additionally, the meaning conflation may affect the semantic modeling of word senses, for example two semantically unrelated words similar to different word senses of the same word may be pulled together (Neelakantan, Shankar, Passos, & McCallum, 2014a; Pilehvar & Collier, 2016). In order to alleviate the impact of the meaning conflation deficiency several directions have been taken, one of these research directions is building word embeddings as representations sensitive to the context, what are called contextualized word embeddings. In contrast to word embeddings which represents words with a single static vector, contextualized word embeddings dynamically change depending on the context in which they appear. We will report on the milestones of both kinds of representation in Chapter 3.

**2.2.2 Semantic Networks**

Another popular approach to characterize word senses is the adoption of semantic networks. Semantic networks are knowledge bases in which the core unit, the *synset*, represents a uniquely identified sense, which mostly reflects cognitively grounded uses of a given term. Synsets are usually represented as vertices of a labeled graph, where edges represent semantic relationships intervening between each two senses. In contrast to the conflated word embedding representations, semantic networks contain unique entries for different word senses, thus constituting the reference word sense dictionary, i.e. *sense inventory*, for many diverse NLP applications.

**WordNet** WordNet constitutes the most popular English word sense inventory, manually curated by experts (G. A. Miller, 1995). Word senses are represented through synsets, that are sets of synonyms expressing distinct word senses. In WordNet, each lemma (word or multi-word expression), belongs to one or more synsets, and word senses are represented as combination of word form, i.e. the lexicalization, and synset (usually referred to as sense-key). These nodes are provided with a unique identifier, called synset id, and endowed with a gloss and various usage examples. For example, given the word *bank*, we might intend the word as the river bank or the financial institution, depending on the context of usage. The river bank word sense entry is defined as *sloping land (especially the slope beside a body of water)*, the synset is made of the term *bank* only and is identified by the synset identifier wn09213565n. Together with the definition two examples are reported: *they pulled the canoe up on the bank* and *he sat on the bank of the river and watched the currents*. The financial institution word sense is represented by the synset *depository financial institution, banking concern, banking company, bank* identified by the id wn08420278n and defined as *a financial institution that accepts deposits and channels the money into lending activities*. The reported usage examples are *he cashed a check at the bank* and *that bank holds the mortgage on my home*. 2

WordNet is actually partitioned into four categories, modeled upon the four open-class parts of speech: nouns, verbs, adjectives and adverbs. Each portion of WordNet has its own relations connecting entities herein. Nouns are organized in a lexical memory as hierarchies, verbs are organized by a variety of entailment relations, while adjectives and adverbs are organized as N-dimensional hyperspaces: each of these lexical structures reflects a different way of categorizing experience. The WordNet version 3.0 contains 117, 659 synsets, 206, 949 senses (sensekeys) and 147, 306 different lemmas. Following WordNet several energies have been invested in developing semantic networks or translating WordNet in different languages (Bond & Paik, 2012; J. P. McCrae, Rademaker, Rudnicka, & Bond, 2020; J. P. McCrae, Wood, & Hicks, 2017; Pianta, Bentivogli, & Girardi, 2002; Rudnicka, Witkowski, & Kali ´nski, 2015).

**BabelNet**

BabelNet is a multilingual lexicalized semantic network, containing about about 20 million entries and distributed in 500 different languages (Navigli & Ponzetto, 2010).3 The architecture of BabelNet is borrowed from WordNet: the network was built by automatically linking Wikipedia pages to WordNet synsets, thus exploiting the multilingual features of Wikipedia: each BabelNet’s node contains multilingual lexicalizations for the same word sense, collected from Wikipedia.

More precisely, BabelNet’s generation apporach may be partitioned into three steps: synsets mapping, multilingual expansions and synsets linking. In the first step, WordNet and Wikipedia are combined by automatically acquiring a mapping between synsets and Wikipedia pages: the conditional probability of a WordNet synset given a Wikipedia page is computed through disambiguation contexts obtained from the two resources. The precision of the first step is fundamental to avoid duplicate word senses as well as building solid foundations to the multilingual expansion. The second step is aimed at extending English synsets to multiple languages through both Wikipedia and machine translation. In this setting, each synset, identified through a BabelNet synset id, is enriched with lexicalizations in multiple languages, thus representing each word sense as a collection of lemmas in many different languages. The last step is aimed at building the net

work between synsets. Relationships are inherited from WordNet and further expanded by considering the degree of correlation between the two Wikipedia pages connected to both nodes. The final resource consists in a semantic network in which nodes (BabelNet synsets) offer multilingual lexicalizations and are linked by all the WordNet relationships plus an underspecified relatedness relation inherited by the Wikipedia page links. Further works have been focused on injecting in BabelNet other information extracted from other resources such as WordNet 2020 (J. P. McCrae et al., 2020), Omegawiki 4 , Wiktionary5 , Wikidata (Vrandeˇci´c & Krötzsch, 2014), GeoNames 6 , ImageNet (Deng et al., 2009), Open Multilingual WordNet (Bond & Paik, 2012), BabelPic (Calabrese, Bevilacqua, & Navigli, 2020) and VerbAtlas (Di Fabio, Conia, & Navigli, 2019).

**3 Related Work**

In this Chapter we introduce the state-of-the-art approaches to word and sense embeddings. The Chapter is organized as follows: in Section 3.1 the main contributions to word embeddings are illustrated; both static word embeddings (Section 3.1.1) and context sensitive word embeddings (Section 3.1.2) are surveyed. The

second part of the chapter (Section 3.2) elaborates on the main contributions to sense embeddings : in Section 3.2.1 we review static sense embeddings, whilst in Section 3.2.2 context sensitive sense embeddings are introduced.

**3.1 Word Embeddings**

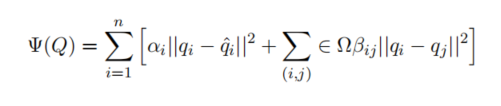
Given the relevance of the embeddings in lexical resources as well as for neural networks, several energies have been invested in finding an efficient approach to embed words into vectorial representations. As mentioned, according to their underlying constructive rationale, word embeddings might be partitioned into *static* word embeddings and *contextual* word embeddings. Since both families of word embeddings are relevant to our work, in this section we briefly report on both design paradigms.

**3.1.1 Static Word Embeddings**

One of the main contributions in word embedding techniques is provided by Word2Vec, the innovation lies in making the learning of word embedding efficient, enabling training of word embeddings on large-scale corpora (Mikolov, Chen, et al., 2013). Word2Vec has been published in two different fashions: Skip-gram and Continuous Bag-Of-Words (CBOW). The difference among the two strategies lies in the training objective: in the CBOW setting, given the context surrounding a word *w*1, *wi−*1 and *wi*+1, *wn*, we have to predict the word at position *wi* ; conversely, in Skip-gram setting, given the word *wi* we have to predict the surrounding context *w*1, *wi−*1 and *wi*+1, *wn*. Both CBOW and Skip-gram training objectives are graphically illustrated in Figure 3.1. Acquiring word embeddings from large text corpora allows them to incorporate relation among words, such as the relation among country and the relative capital, or the gender of words (Mikolov, Chen, et al., 2013; Mikolov, Sutskever, Chen, Corrado, & Dean, 2013; Mikolov, Yih, & Zweig, 2013).

Another word embedding architecture that deserves to be mentioned is GloVe (Pennington, Socher, & Manning, 2014). Such architecture belongs to a different line of research: while Word2Vec model is acknowledged to be a predictive model, GloVe belongs to the count-based family of models: that is, representations are learned by applying dimensionality reduction techniques to the co-occurrence counts matrix. In particular GloVe embeddings have been acquired through a training on 840 billion words from the Common Crawl dataset 1 . One of the latest contributions in prediction-based static word embeddings field is fastText (Bojanowski, Grave, Joulin, & Mikolov, 2017). Such model improves over skip-gram architecture by learning N-gram embeddings rather than word embeddings. The intuition underlying this decision is that language relies heavily on morphology and compositional word-building encodes information also in subwords, so this may be generalized to unseen words (Joulin, Grave, Bojanowski, & Mikolov, 2016).

A different line of research goes in the direction of improving pre-trained word embeddings by exploiting knowledge bases (Faruqui & Dyer, 2014). Such technique, called retrofitting, improves vectors quality in a post-processing step that updates word representations by running a belief-propagation algorithm on a graph constructed from lexicon-derived relational information. More precisely, given a set of pre-trained word vectors *Q*ˆ = *{q*ˆ1, *. . .* , ˆ*qn}* such technique is aimed at refining such descriptions by accounting for the information provided by a knowledge base. In particular, given a vocabulary *V* = *{w*1, *. . .* , *wn}*, an ontology Ω that encodes the semantic relations between words in *V* and the set of pre-trained word vectors *Q*ˆ for words in *V* , the objective of the retrofitting is to learn the word vector representations *Q* = *{q*1, *. . .* , *qn}* such that the representation *qi* for the word *wi* is close to its counterpart *q*ˆ*i ∈ Q*ˆ and adjacent to the neighbours of *wi* in the ontology Ω. The refined word vectors are obtained through a learning process, since the objective is to achieve a word representation *qi* close to its pre-trained vector *q*ˆ*i* as well as its neighbours *qj ∀j* such that the words *wi* and *wj* are connected by a semantic relation (*i*, *j*) *∈* Ω, the objective to be minimize is:



where *α* and *β* tune the balance between the pre-trained vector and the ontology association factors. It is worth noting that the selection of the vector distance metric is not bounded, in this setting the Euclidean Distance has been adopted. The retrofitting technique shown improvements in pre-trained word vectors quality, in particular, the authors proved its beneficial effect on word similarity, syntactic relation extraction, synonym selection and sentiment analysis. The retrofitting is also at the core of ConceptNet Numberbatch (CNN) (Speer & Chin, 2016). In particular, CNN is built exploiting the “expanded retrofitting”, which adjusts the values of existing word embeddings based on a new objective function that also takes a knowledge graph into account: the authors applied the retrofitting separately to multiple sources of embeddings, i.e. GloVe, Word2Vec and fastText adopting ConceptNet as knowledge base (Speer et al., 2017): the results are then aligned on a unified semantic space. More precisely, the authors exploited ConceptNet as Ω knowledge base, where only non negative relations have been retained —i.e., negative relations such as *NotUsedFor* or *Antonym* have been removed—. Additionally, the vocabulary *V* was constructed based on the different vector resources: since multiple pre-trained embedding resources were adopted, the authors took all terms appearing in the first 500, 000 rows of each such resource, and retained all the words occurring in the first 200, 000 rows of at least one of them. Such terms are then combined with the set of words contained in the Ω knowledge base. Subsequently, each vectorial resource was refined though the expanded retrofitting technique so as to obtain the unified embedding matrix *M*1. More precisely, for each source of embeddings *Q*ˆ (Word2Vec, GloVe) the authors obtained its inferred version *Q* by applying retrofitting, then such representations were concatenated to the unified embedding matrix *M*1. Given that multiple word representations were concatenated in the unified matrix, each word is represented through a vector *qi ∈* R *k* , that is, each embedding is represented with *k* dimensions corresponding to features that may be redundant. Therefore, the representations from *M*1 were subject to dimensionality reduction: such process was designed to learn a projection from *k* dimensions to *k ′* = 300 to remove the redundancy stemming from the concatenation. Additionally, to deal with multiple languages, the authors calculated their own multilingual distributional embeddings through fastText for words occurring in the OpenSubtitles2016 parallel corpus (Lison & Tiedemann, 2016) and employed such vectors as input together with Word2Vec and GloVe. The latest version of CNN 2 covers 78 different languages.

**3.1.2 Contextualized Word Embeddings**

As mentioned, despite their impact, word embeddings suffer from the meaning conflation deficiency. In order to address such issue two parallel lines of research have emerged: modeling sense representations and building context sensitive word representations. In this section we focus on the latter family.

One of the pioneering works employing contextualized representation is the sequence tagger from W. Li and McCallum (2005). Such model derives context sensitive representations for each word based on word clustering, then integrates them as additional features to a sequence tagger. The attention to context sensitive representations remained latent until the lights were turned on the limitations of word embeddings. One of the earliest attempts to address the meaning conflation issue is Context2Vec (Melamud, Goldberger, & Dagan, 2016). Such model represents the context of a target word by extracting the output embedding of a multi-layer perceptron built on top of a bi-directional LSTM language model. Context2Vec started the research line in which contextualized embeddings are pre-trained on large unlabeled data. At the test time word contextualized embeddings are then combined to their static embedding and fed to the main model. Following such direction, the prominent ELMo (Embeddings from Language Models) technique (Peters et al., 2018) built context sensitive word embeddings through the pre-training of a bi-directional language model on large text corpus. The difference with respect to preceding works is that ELMo jointly maximizes the log likelihood of the forward and backward directions, thus sharing and combining weights from both directions. Similarly to ELMo, the Context Vectors (CoVe) model computes contextualized representations using a two-layer bidirectional LSTM network, in the machine translation setting: CoVe vectors are pre-trained exploiting an LSTM encoder from an attentional sequence-to-sequence machine translation model (McCann, Bradbury, Xiong, & Socher, 2017). The introduction of Transformers architecture (Vaswani et al., 2017) started two varieties of NLMs: predictive and bidirectional language models (more on this in Chapter 2). In particular, bidirectional NLMs such as BERT, exploit the encoder of Transformers to build context sensitive word embeddings. Such models are pre-trained on more and more larger text corpus, thus obtaining representation as much general as possible. Pre-trained language models can subsequently be exploited to compute representations for words in sentences: the embeddings produced by such models are then used to address many diverse NLP tasks in many different languages (Lee et al., 2020; Qu et al., 2019; Raffel et al., 2019; Souza, Nogueira, & Lotufo, 2019).

**3.2 Sense Embeddings**

In order to alleviate the meaning conflation deficiency of word embeddings, a parallel direction of research has emerged over the past years, which tries to directly model individual meanings of words. In this section we focus on sense representations relying on word embeddings. Since we partitioned word embeddings in *static* and *contextualized* representations, we therefore partitioned sense representations according to the underlying word embeddings design paradigm. Additionally, attempting to address the meaning conflation limitation, mainly two lines of sense vector representations have emerged: in the first line, unsupervised, senses are learned directly from text corpus or knowledge-based (E. H. Huang, Socher, Manning, & Ng, 2012; Reisinger & Mooney, 2010; Vu & Parker, 2016), while in the second approach senses are linked to pre-defined sense inventories. In this work we focus on the latter type of representation.

**3.2.1 Static Sense Embeddings**

Provided that the evolution of word representations flows from static word embeddings to contextualized word embeddings, the first vector representations for word senses have been introduced relying on static word embeddings. One of the main contributions between the sense vectorial representations is NASARI (Camacho-Collados, Pilehvar, & Navigli, 2015b; Pilehvar & Navigli, 2015). In the same spirit of BabelNet, NASARI puts together two sorts of knowledge: one coming from WordNet (originally handcrafted by a team of lexicographers), based on synsets and on the intervening semantic relations, and one available in Wikipedia, which is conversely the outcome of a large collaborative effort. Pages in Wikipedia are considered as concepts. In NASARI embeddings each item (concept or named entity) is defined through a dense vector over a 300-dimensions space. NASARI vectors have been acquired by starting from the vectors trained on the Google News dataset, provided along with the Word2vec toolkit. All NASARI vectors share the same semantic space also with Word2vec, so that their representations can be used to compute semantic distances between any two such vectors. Thanks to the structure provided by the BabelNet resource, the resulting 2.9*M* embeddings are part of a huge semantic network. NASARI includes sense descriptions for nouns, but not for other grammatical categories.

Directly following NASARI, but with totally different building rationale, SENSEEMBED has been introduced, containing representations for the four main parts of speech (nouns, verbs, adjectives and adverbs) (Iacobacci, Pilehvar, & Navigli, 2015). The approach proposed by SENSEEMBED is aimed at obtaining continuous representations of individual senses. In order to build sense representations, the authors exploited Babelfy (Moro, Raganato, & Navigli, 2014) to disambiguate the September-2014 dump of the English Wikipedia.3 Subsequently, the Word2vec toolkit has been employed to build vectors for 2.5 millions of unique word senses.

The obtained resource contains the representation for both terms —e.g., the embedding for the term Bank— and word senses —e.g., the embedding representing the meaning of bank intended as *financial institution*, endowed with the identifier Bank-bn:00008364n—. Given the negative impact of the meaning conflation deficiency implicitly coded in word embeddings, DECONF has been introduced with particular attention to the mentioned limitation (Pilehvar & Collier, 2016). DECONF is a sense representation technique that starts from a semantic network and a set of pre-trained word embeddings. The proposed approach computes a list of “sense biasing words” for a given word sense. The whole process is characterised by two phases: *(i)* the extraction of the most representative words that express the semantics of a synset, and *(ii)* the sense representations learning. In the extraction phase, the control strategy starts from a target synset *yt* , leverages the structure of the semantic network of WordNet and produces as output an ordered list *Bt*

of semantically related terms that provide a cue for the sense usage. The latter phase is aimed at learning the representation of the word sense *st* (the sense for term *t*): to these ends the procedure deconflates the representation of all the lexicalizations of the sense *st* , and biases them towards the list *Bt* . In order to generate the DECONF resource, the authors chose WordNet 3.0 as semantic network and the 300*-d* Word2Vec word embeddings trained on the Google News dataset. The final resource contains about 207 thousand vectors for WordNet word senses, each such sense representation lives in the same space which is also shared by the word embeddings. Different from previous resources, SW2V (so named after ‘Senses and Words to Vectors’) is a neural model devised to represent both term and sense vector representations (Mancini, Camacho-Collados, Iacobacci, & Navigli, 2017). The proposed approach jointly learns both representations by exploiting text corpora and semantic networks. Due to the temporal complexity of the state-of-the-art disambiguation systems, the authors devised an unsupervised shallow word sense connectivity algorithm. Such algorithm exploits the connections of a semantic network and associates a term with its top candidate senses according to the number of sense connections and word context. Once the corpus of sense tagged words has been generated, an extension of the Word2Vec’s CBOW model is employed. The extension of the CBOW model in order to deal with word senses follows the assumption that since a word is a lexicalization of an underlying sense, an update of the word embedding should entail a similar update of the sense representation, and vice versa. The authors chose BabelNet as reference semantic network and its underlying sense inventory; the pre-trained version of SW2V contains over 6 million vectors representing both words and word senses, in the same spirit of SENSEEMBED. Following SW2V, LSTMEMBED is a recently proposed model based on bidirectional LSTM for learning embeddings of words and senses in the same semantic space (Iacobacci & Navigli, 2019). The model starts from a sense-tagged text, which is processed with a bidirectional LSTM analyzing both the preceding and the posterior context of a token *si* , where *si* is either a word or a sense tag. The output computed by the LSTM on both directions is concatenated and linearly weighted with a dense layer. Subsequently the model compares the output with the pre-trained embedding vector of the target token *si* . The training phase maximizes the similarity among the output of the network and the pre-trained embeddings: the loss is computed in terms of cosine distance.4 LSTMEMBED pre-trained embeddings contain about 2 millions vectors. The obtained resource is eatured by three sorts of representation: the word-sense representation —e.g., the vector for the sense bn:00008363n, which refers to Bank, intended as “*Sloping land, especially the slope beside a body of water*”—; the representation for a given lexicalization associated to a given sense —e.g., for the pair Bank-bn:00008363n—; and the word embedding —e.g., the vector for the term Bank—, possibly conflating all senses underlying the given term

**3.2.2 Contextualized Sense Embeddings**

The contextualized sense representations line of research follows directly from the introduction of contextualized language models, and strongly relies on such representations. Despite such language models provide context sensitive representations, they still lack semantic grounding to sense inventories. The first attempt at demonstrating that contextual embeddings from pre-trained language models can be enriched by exploiting sense inventories is Language Modelling Makes Sense (LMMS) (Loureiro & Jorge, 2019a). LMMS is an approach for generating sense embeddings relying on pre-trained contextualized language models that covers the entire WordNet 3.0 sense inventory. The proposed approach computes a list of sense embeddings starting from annotations, i.e. a sense tagged corpus. In particular, the sense vector is computed as the average of all the contextual representation for words tagged with the word sense: given *n* contextual embeddings *ci* for a word sense *s*, the vector *vs* is computed as *vs* = *n* 1 P*n i*=1 *ci* .

Since the sense tagged corpus covers only a small percentage of the WordNet vocabulary, the authors improve sense inventory coverage exploiting WordNet structure: in order to build embeddings for higher-level abstractions, the average of the embeddings of all lower-level constituents is employed. That is, the embedding of an unseen word sense corresponds to the average of all its children representation.

LMMS pre-trained embeddings cover the entire WordNet vocabulary, thus containing embeddings for 117, 659 synsets corresponding to 206, 949 unique senses. Since LMMS is grounded to the WordNet sense inventory, the resource represent vectors for English words only. Following LMMS, SENSEMBERT has been introduced relying on the pre-trained version of BERT large (Scarlini, Pasini, & Navigli, 2020a). SENSEMBERT is a knowledgebased approach to produce latent semantic representations of word meanings in multiple languages. The construction of SENSEMBERT relies on Babelnet, Wikipdia and NASARI sense embeddings together with the pre-trained BERT large model. The proposed approach starts by collecting from Wikipedia all the sentences that are suitable for characterizing a given word synset: this is done by exploiting the link between BabelNet and Wikipedia. Once contextual information has been collected, the authors compute the contextualized word embedding of each relevant word for the target synset: relevant words for each synset are identified exploiting NASARI lexical vectors, then contextualized representation for such words are obtained through BERT large language model. Eventually, the synset embedding is built by exploiting word representations together with their rank in the NASARI lexical vector. In the same spirit of LMMS, synset representation quality is improved by exploiting the semantic network structure. Since the linking between BabelNet and Wikipedia involves nouns only, the proposed approach build representations for nouns only. Thanks to the multilingual nature of BabelNet, the authors exploited also the multilingual version of BERT to build sense embeddings for multiple languages. SENSEMBERT pre-trained embeddings contain vectors for 146, 313 senses. ARES, so dubbed after context-AwaRe Embeddings of Senses, has been introduced few months later, as the extension of SENSEMBERT (Scarlini et al., 2020b). ARES is a semi-supervised approach to producing sense embeddings for the lexical meanings within a lexical knowledge base that lie in a space that is comparable to that of contextualized word vectors. The construction of ARES relies on several resources: WordNet, SyntagNet (Maru, Scozzafava, Martelli, & Navigli, 2019), UKB (E. Agirre, de Lacalle, & Soroa, 2014) and BERT. The proposed approach starts collecting contexts for WordNet’s synsets exploiting BERT: given a sense *s* and one of its lexicalizations *l*, the authors collected all occurrences of *l* in a corpus and computed their contextualized representation and clustered through *k-means*. To such groups the UKB algorithm is exploited so as to label each cluster with one of the senses for *l*. Each such cluster is then refined exploiting the collocations from SyntagNet. Contextual information retrieved is then exploited so as to build embeddings for WordNet’s synsets as a combination of embeddings computed though BERT for sentences and collocations. ARES pre-trained embeddings contain vectors for 206, 950 senses, covering 65% of WordNet’s vocabulary (77, 195 out of 117, 659).

LMMS Reloaded (LMMS-R) is the most recent resource belonging to the contextualized sense embeddings family, it has been introduced as extension of LMMS (Loureiro, Jorge, & Camacho-Collados, 2022). LMMS-R is a principled approach for sense representation based on contextual NLMs trained exclusively with self-supervision. Following LMMS the synset embbedding for *s* is built by averaging the contextualized representations for the lexicalization *l* in a sense-tagged corpus. Such vectors are then refined exploiting the WordNet structure. LMMS-R allows for a different characterization of multiple layers NLMs according to the task for which the embeddings are designed. LMMS-R pre-trained embeddings for Word Sense Disambiguation (WSD) cover the entire WordNet vocabulary, thus containing embeddings for 117, 659 synsets corresponding to 206, 949 unique senses. Since LMMS-R is grounded to the WordNet sense inventory, the resource represents vectors for English words only.