CHAPTER 9

Language Modeling

9.1 THE LANGUAGE MODELING TASK

Language modeling is the task of assigning a probability to sentences in a language ("what is the probability of seeing the sentence *the lazy dog barked loudly*?"). Besides assigning a probability to each sequence of words, the language models also assigns a probability for the likelihood of a given word (or a sequence of words) to follow a sequence of words ("what is the probability of seeing the word *barked* after the seeing sequence *the lazy dog*?").[1]

Perfect performance at the language modeling task, namely predicting the next word in a sequence with a number of guesses that is the same as or lower than the number of guesses required by a human participant, is an indication of human level intelligence,[2] and is unlikely to be achieved in the near future. Even without achieving human-level performance, language modeling is a crucial components in real-world applications such as machine-translation and automatic speech recognition, where the system produces several translation or transcription hypotheses, which are then scored by a language model. For these reasons, language modeling plays a central role in natural-language processing, AI, and machine-learning research.

Formally, the task of language modeling is to assign a probability to any sequence of words $w_{1:n}$, i.e., to estimate $P(w_{1:n})$. Using the chain-rule of probability, this can be rewritten as:

$$P(w_{1:n}) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_{1:2})P(w_4 \mid w_{1:3}) \dots P(w_n \mid w_{1:n-1}). \tag{9.1}$$

That is, a sequence of word-prediction tasks where each word is predicted conditioned on the preceding words. While the task of modeling a single word based on its left context seem more manageable than assigning a probability score to an entire sentence, the last term in the equation

[1]Note that the ability to assign a probability for a word following a sequence of words $p(w_{i_1}|w_1, w_2, \dots, w_{i-1})$ and the ability to assign a probabilities to arbitrary sequences of words $p(w_1, w_2, \dots, w_k)$ are equivalent, as one can be derived from the other. Assume we can model probabilities of sequences. Then the conditional probability of a word can be expressed as a fraction of two sequences:

$$p(w_i|w_1, w_2, \dots, w_{i-1}) = \frac{p(w_1, w_2, \dots, w_{i-1}, w_i)}{p(w_1, w_2, \dots, w_{i-1})}.$$

Alternatively, if we could model the conditional probability of a word following a sequence of words, we could use the chain rule to express the probability of sequences as a product of conditional probabilities:

$$p(w_1, \dots, w_k) = p(w_1|\text{<s>}) \times p(w_2|\text{<s>}, w_1) \times p(w_3|\text{<s>}, w_1, w_2) \times \dots \times p(w_k|\text{<s>}, w_1, \dots, w_{k-1}),$$

where <s> is a special start-of-sequence symbol.

[2]Indeed, any question can be posed as a next-word-guessing task, e.g., *the answer to question X is* ___. Even without such pathological cases, predicting the next word in the text requires knowledge of syntactic and semantic rules of the language, as well as vast amounts of world knowledge.

still requires conditioning on $n-1$ words, which is as hard as modeling an entire sentence. For this reason, language models make use of the *markov-assumption*, stating that the future is independent of the past given the present. More formally, a kth order markov-assumption assumes that the next word in a sequence depends only on the last k words:

$$P(w_{i+1} \mid w_{1:i}) \approx P(w_{i+1} \mid w_{i-k:i}).$$

Estimating the probability of the sentence then becomes

$$P(w_{1:n}) \approx \prod_{i=1}^{n} P(w_i \mid w_{i-k:i-1}), \tag{9.2}$$

where $w_{-k+1}, \ldots, w_0$ are defined to be special padding symbols.

Our task is then to accurately estimate $P(w_{i+1} \mid w_{i-k:i})$ given large amounts of text.

While the kth order markov assumption is clearly wrong for any k (sentences can have arbitrarily long dependencies, as a simple example consider the strong dependence between the first word of the sentence being *what* and the last one being *?*), it still produces strong language modeling results for relatively small values of k, and was the dominant approach for language modeling for decades. This chapter discusses kth order language models. In Chapter 14 we discuss language modeling techniques that do not make the markov assumption.

9.2 EVALUATING LANGUAGE MODELS: PERPLEXITY

There are several metrics for evaluating language modeling. The application-centric ones evaluate them in the context of performing a higher-level task, for example by measuring the improvement in translation quality when switching the language-modeling component in a translation system from model A to model B.

A more intrinsic evaluation of language models is using *perplexity* over unseen sentences. Perplexity is an information theoretic measurement of how well a probability model predicts a sample. Low perplexity values indicate a better fit. Given a text corpus of n words $w_1, \ldots, w_n$ (n can be in the millions) and a language model function LM assigning a probability to a word based on its history, the perplexity of LM with respect to the corpus is:

$$2^{-\frac{1}{n}\sum_{i=1}^{n} \log_2 \mathrm{LM}(w_i \mid w_{1:i-1})}.$$

Good language models (i.e., reflective of real language usage) will assign high probabilities to the events in the corpus, resulting in lower perplexity values.

The perplexity measure is a good indicator of the quality of a language model.[3] Perplexities are corpus specific—perplexities of two language models are only comparable with respect to the same evaluation corpus.

[3] It is important to note, however, that in many cases improvement in perplexity scores do not transfer to improvement in extrinsic, task-quality scores. In that sense, the perplexity measure is good for comparing different language models in terms of their ability to pick-up regularities in sequences, but is not a good measure for assessing progress in language understanding or language-processing tasks.

9.3 TRADITIONAL APPROACHES TO LANGUAGE MODELING

The traditional approach to language models assumes a k-order markov property, and model $P(w_{i+1} = m|w_{1:i}) \approx P(w_{i+1} = m|w_{i-k:i})$. The role of the language model then is to provide good estimates $\hat{p}(w_{i+1} = m|w_{i-k:i})$. The estimates are usually derived from corpus counts. Let $\#(w_{i:j})$ be the count of the sequence of words $w_{i:j}$ in a corpus. The maximum likelihood estimate (MLE) of $\hat{p}(w_{i+1} = m|w_{i-k:i})$ is then:

$$\hat{p}_{\text{MLE}}(w_{i+1} = m|w_{i-k:i}) = \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})}.$$

While effective, this baseline approach has a big shortcoming: if the event $w_{i-k:i+1}$ was never observed in the corpus ($\#(w_{i-k:i+1}) = 0$), the probability assigned to it will be 0, and this in turn will result in a 0-probability assignment to the entire corpus because of the multiplicative nature of the sentence probability calculation. A zero probability translates to an infinite perplexity—which is very bad. Zero events are quite common: consider a trigram language model, which only conditions on 2 words, and a vocabulary of 10,000 words (which is rather small). There are $10{,}000^3 = 10^{12}$ possible word triplets—it is clear that many of them won't be observed in training corpora of, say, 10^{10} words. While many of these events don't occur because they do not make sense, many others just did not occur in the corpus.

One way of avoiding zero-probability events is using *smoothing techniques*, ensuring an allocation of a (possibly small) probability mass to every possible event. The simplest example is probably *additive smoothing*, also called *add-α smoothing* [Chen and Goodman, 1999, Goodman, 2001, Lidstone, 1920]. It assumes each event occurred at least α times in addition to its observations in the corpus. The estimate then becomes

$$\hat{p}_{\text{add-}\alpha}(w_{i+1} = m|w_{i-k:i}) = \frac{\#(w_{i-k:i+1}) + \alpha}{\#(w_{i-k:i}) + \alpha|V|},$$

where $|V|$ is the vocabulary size and $0 < \alpha \leq 1$. Many more elaborate smoothing techniques exist.

Another popular family of approaches is using *back-off*: if the $k$gram was not observed, compute an estimate based on a $(k-1)$gram. A representative sample of this family is *Jelinek Mercer interpolated smoothing* [Chen and Goodman, 1999, Jelinek and Mercer, 1980]:

$$\hat{p}_{\text{int}}(w_{i+1} = m|w_{i-k:i}) = \lambda_{w_{i-k:i}} \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})} + (1 - \lambda_{w_{i-k:i}})\hat{p}_{\text{int}}(w_{i+1} = m|w_{i-(k-1):i}).$$

For optimal performance, the values $\lambda_{w_{i-k:i}}$ should depend on the content of the conditioning context $w_{i-k:i}$: rare contexts should receive different treatments than frequent ones.

The current state-of-the-art non-neural language modeling technique uses *modified Kneser Ney smoothing* [Chen and Goodman, 1996], which is a variant of the technique proposed by Kneser and Ney [1995]. For details, see Chen and Goodman [1996] and Goodman [2001].

9.3.1 FURTHER READING

Language modeling is a very vast topic, with decades of research. A good, formal overview of the task, as well as motivations behind the perplexity measure can be found in the class notes by Michael Collins.[4] A good overview and empirical evaluation of smoothing techniques can be found in the works of Chen and Goodman [1999] and Goodman [2001]. Another review of traditional language modeling techniques can be found in the background chapters of the Ph.D. thesis of Mikolov [2012]. For a recent advance in non-neural language modeling, see Pelemans et al. [2016].

9.3.2 LIMITATIONS OF TRADITIONAL LANGUAGE MODELS

Language modeling approaches based on smoothed MLE estimates ("traditional") are easy to train, scale to large corpora, and work well in practice. They do, however, have several important shortcomings.

The smoothing techniques are intricate and based on back off to lower-order events. This assumes a fixed backing-up order, that needs to be designed by hand, and makes it hard to add more "creative" conditioning contexts (i.e., if one wants to condition on the k previous words *and* on the genre of the text, should the backoff first discard of the kth previous word, or of the genre variable?). The sequential nature of the backoff also makes it hard to scale toward larger ngrams in order to capture long-range dependencies: in order to capture a dependency between the next word and the word 10 positions in the past, one needs to see a relevant 11-gram in the text. In practice, this very rarely happens, and the model backs off from the long history. It could be that a better option would be to back off from the intervening words, i.e., allow for ngrams with "holes" in them. However, these are tricky to define while retaining a proper generative probabilistic framework.[5]

Scaling to larger ngrams is an inherent problem for MLE-based language models. The nature of natural language and the large number of words in the vocabulary means that statistics for larger ngrams will be sparse. Moreover, scaling to larger ngrams is very expensive in terms of memory requirements. The number of possible ngrams over a vocabulary V is $|V|^n$: increasing the order by one will result in a $|V|$-fold increase to that number. While not all of the theoretical ngrams are valid or will occur in the text, the number of observed events does grow at least multiplicatively when increasing the ngram size by 1. This makes it very taxing to work larger conditioning contexts.

Finally, MLE-based language models suffer from lack of generalization across contexts. Having observed *black car* and *blue car* does not influence our estimates of the event *red car* if we haven't see it before.[6]

[4] http://www.cs.columbia.edu/~mcollins/lm-spring2013.pdf

[5] Although see lines of work on factored language models (i.e., A. Bilmes and Kirchhoff [2003]) and on maximum-entropy (log-linear) language models, starting with Rosenfeld [1996], as well as recent work by Pelemans et al. [2016].

[6] Class-based language models [Brown et al., 1992] try to tackle this by clustering the words using distributional algorithms, and conditioning on the induced word-classes instead of or in addition to the words.

9.4 NEURAL LANGUAGE MODELS

Nonlinear neural network models solve some of the shortcomings of traditional language models: they allow conditioning on increasingly large context sizes with only a linear increase in the number of parameters, they alleviate the need for manually designing backoff orders, and they support generalization across different contexts.

A model of the form presented in this chapter was popularized by Bengio et al. [2003].[7]

The input to the neural network is a $k$gram of words $w_{1:k}$, and the output is a probability distribution over the next word. The k context words $w_{1:k}$ are treated as a word window: each word w is associated with an embedding vector $v(w) \in \mathbb{R}^{d_w}$, and the input vector $\boldsymbol{x}$ a concatenation of the k words:

$$\boldsymbol{x} = [v(w_1); v(w_2); \ldots; v(w_k)].$$

The input $\boldsymbol{x}$ is then fed to an MLP with one or more hidden layers:

$$\begin{aligned} \hat{\boldsymbol{y}} = P(w_i|w_{1:k}) = LM(w_{1:k}) &= \text{softmax}(\boldsymbol{h}\boldsymbol{W}^{\mathbf{2}} + \boldsymbol{b}^{\mathbf{2}}) \\ \boldsymbol{h} &= g(\boldsymbol{x}\boldsymbol{W}^{\mathbf{1}} + \boldsymbol{b}^{\mathbf{1}}) \\ \boldsymbol{x} &= [v(w_1); v(w_2); \ldots; v(w_k)] \\ v(w) &= \boldsymbol{E}_{[w]} \end{aligned} \tag{9.3}$$

$$w_i \in V \quad \boldsymbol{E} \in \mathbb{R}^{|V| \times d_w} \quad \boldsymbol{W}^{\mathbf{1}} \in \mathbb{R}^{k \cdot d_w \times d_{\text{hid}}} \quad \boldsymbol{b}^{\mathbf{1}} \in \mathbb{R}^{d_{\text{hid}}} \quad \boldsymbol{W}^{\mathbf{2}} \in \mathbb{R}^{d_{\text{hid}} \times |V|} \quad \boldsymbol{b}^{\mathbf{2}} \in \mathbb{R}^{|V|}.$$

V is a finite vocabulary, including the unique symbols UNK for unknown words, <s> for sentence initial padding, and </s> for end-of-sequence marking. The vocabulary size, $|V|$, ranges between 10,000–1,000,000 words, with the common sizes revolving around 70,000.

Training The training examples are simply word $k$grams from the corpus, where the identities of the first $k-1$ words are used as features, and the last word is used as the target label for the classification. Conceptually, the model is trained using cross-entropy loss. Working with cross entropy loss works very well, but requires the use of a costly softmax operation which can be prohibitive for very large vocabularies, prompting the use of alternative losses and/or approximations (see below).

Memory and Computational Efficiency Each of the k input words contributes d_{w} dimensions to $\boldsymbol{x}$, and moving from k to $k+1$ words will increase the dimensions of the weight matrix $\boldsymbol{W}^{\mathbf{1}}$ from $k \cdot d_{\text{w}} \times d_{\text{hid}}$ to $(k+1) \cdot d_{\text{w}} \times d_{\text{hid}}$, a small linear increase in the number of parameters, in contrast to a polynomial increase in the case of the traditional, count-based language models. This is possible because the feature combinations are computed in the hidden layer. Increasing the order k will likely require enlarging the dimension of d_{hid} as well, but this is still a very

[7] A similar model was presented as early as 1988 by Nakamura and Shikano [1988] in their work on word class prediction with neural networks.

modest increase in the number of parameters compared to the traditional modeling case. Adding additional nonlinear layers to capture more complex interactions is also relatively cheap.

Each of the vocabulary words is associated with one d_{w} dimensional vector (a row in $\boldsymbol{E}$) and one d_{hid} dimensional vector (a column in $\boldsymbol{W^2}$). Thus, a new vocabulary items will result in a linear increase in the number of parameters, again much better than the traditional case. However, while the input vocabulary (the matrix $\boldsymbol{E}$) requires only lookup operations and can grow without affecting the computation speed, the size of the output vocabulary greatly affects the computation time: the softmax at the output layer requires an expensive matrix-vector multiplication with the matrix $\boldsymbol{W^2} \in \mathbb{R}^{d_{\text{hid}} \times |V|}$, followed by $|V|$ exponentiations. This computation dominates the runtime, and makes language modeling with large vocabulary sizes prohibitive.

Large output spaces Working with neural probabilistic language models with large output spaces (i.e., efficiently computing the softmax over the vocabulary) can be prohibitive both at training time and at test time. Dealing with large output spaces efficiently is an active research question. Some of the existing solutions are as follows.

Hierarchical softmax [Morin and Bengio, 2005] allows to compute the probability of a single word in $O(\log |V|)$ time rather than $O(|V|)$. This is achieved by structuring the softmax computation as tree traversal, and the probability of each word as the product of branch selection decisions. Assuming one is interested in the probability of a single word (rather than getting the distribution over all words) this approach provides clear benefits in both training and testing time.

Self-normalizing aproaches, such as noise-contrastive estimation (NCE) [Mnih and Teh, 2012, Vaswani et al., 2013] or adding normalizing term to the training objective [Devlin et al., 2014]. The NCE approach improves training time performance by replacing the cross-entropy objective with a collection of binary classification problems, requiring the evaluation of the assigned scores for k random words rather than the entire vocabulary. It also improves test-time prediction by pushing the model toward producing "approximately normalized" exponentiated scores, making the model score for a word a good substitute for its probability. The normalization term approach of Devlin et al. [2014] similarly improves test time efficiency by adding a term to the training objective that encourages the exponentiated model scores to sum to one, making the explicit summation at test time unnecessary (the approach does not improve training time efficiency).

Sampling Approaches approximate the training-time softmax over a smaller subset of the vocabulary [Jean et al., 2015].

A good review and comparison of these and other techniques for dealing with large output vocabularies is available in Chen et al. [2016].

An orthogonal line of work is attempting to sidestep the problem by working at the characters level rather than words level.

Desirable Properties Putting aside the prohibitive cost of using the large output vocabulary, the model has very appealing properties. It achieves better perplexities than state-of-the-art traditional language models such as Kneser-Ney smoothed models, and can scale to much larger orders

than is possible with the traditional models. This is achievable because parameters are associated only with individual words, and not with $k$grams. Moreover, the words in different positions share parameters, making them share statistical strength. The hidden layers of the models are in charge of finding informative word combinations, and can, at least in theory, learn that for some words only sub-parts of the $k$gram are informative: it can learn to back-up to smaller $k$grams if needed, like a traditional language model, and do it in a context-dependent way. It can also learn skip-grams, i.e., that for some combinations it should look at words 1, 2, and 5, skipping words 3 and 4.[8]

Another appealing property of the model, besides the flexibility of the $k$gram orders, is the ability to generalize across contexts. For example, by observing that the words *blue*, *green*, *red*, *black*, etc. appear in similar contexts, the model will be able to assign a reasonable score to the event *green car* even though it never observed this combination in training, because it did observe *blue car* and *red car*.

The combination of these properties—the flexibility of considering only parts of the conditioning context and the ability to generalize to unseen contexts—together with the only linear dependence on the conditioning context size in terms of both memory and computation, make it very easy to increase the size of the conditioning context without suffering much from data sparsity and computation efficiency.

The ease in which the neural language model can incorporate large and flexible conditioning contexts allow for creative definitions of contexts. For example, Devlin et al. [2014] propose a machine translation model in which the probability of the next word is conditioned on the previous k words in the generated translation, as well as on m words in the source language that the given position in the translation is based on. This allows the model to be sensitive to topic-specific jargon and multi-word expressions in the source language, and indeed results in much improved translation scores.

Limitations Neural language models of the form presented here do have some limitations: predicting the probability of a word in context is much more expensive than using a traditional language model, and working with large vocabulary sizes and training corpora can become prohibitive. However, they do make better use of the data, and can get very competitive perplexities even with relatively small training set sizes.

When applied in the context of a machine-translation system, neural language models do not always improve the translation quality over traditional Kneser-Ney smoothed language models. However, translation quality does improve when the probabilities from a traditional language model and a neural one are interpolated. It seems that the models complement each other: the neural language models generalize better to unseen events, but sometimes this generalization can hurt the performance, and the rigidity of the traditional models is preferred. As an example, consider the opposite of the colors example above: a model is asked to assign a probability to the sentence *red horse*. A traditional model will assign it a very low score, as it likely observed such

[8]Such skip-grams were explored also for non-neural language models; see Pelemans et al. [2016] and the references therein.

an event only a few times, if at all. On the other hand, a neural language model may have seen *brown horse*, *black horse*, and *white horse*, and also learned independently that *black*, *white*, *brown*, and *red* can appear in similar contexts. Such a model will assign a much higher probability to the event *red horse*, which is undesired.

9.5 USING LANGUAGE MODELS FOR GENERATION

Language models can also be used for generating sentences. After training a language model on a given collection of text, one can generate ("sample") random sentences from the model according to their probability using the following process: predict a probability distribution over the first word conditioned on the start symbol, and draw a random word according to the predicted distribution. Then, predict a probability distribution over the second word conditioned on the first, and so on, until predicting the end-of-sequence </s> symbol. Already with $k = 3$ this produces very passable text, and the quality improves with higher orders.

When decoding (generating a sentence) from a trained language-model in this way, one can either choose the highest scoring prediction (word) at each step, or sample a random word according to the predicted distribution. Another option is to use *beam search* in order to find a sequence with a globally high probability (following the highest-prediction at each step may result in sub-optimal overall probability, as the process may "trap itself into a corner," leading to prefixes that are followed by low-probability events. This is called the *label-bias* problem, discussed in depth by Andor et al. [2016] and Lafferty et al. [2001].

Sampling from a multinomial distribution A multinomial distribution over $|V|$ elements associates a probability value $p_i \geq 0$ for each item $0 < i \leq |V|$, such that $\sum_{i=1}^{|V|} p_i = 1$. In order to sample a random item from a multinomial distribution according to its probability, the following algorithm can be used:

1: $i \leftarrow 0$
2: $s \sim U[0, 1]$ ▷ a uniform random number between 0 and 1
3: **while** $s \geq 0$ **do**
4: $\quad i \leftarrow i + 1$
5: $\quad s \leftarrow s - p_i$
6: **return** i

This is a naive algorithm, with a computational complexity linear in the vocabulary size $O(|V|)$. This can be prohibitively slow using large vocabularies. For peaked distributions where the values are sorted by decreasing probability, the average time would be much faster. The *alias method* [Kronmal and Peterson, Jr., 1979] is an efficient algorithm for sampling from arbitrary multinomial distributions with large vocabularies, allowing sampling in $O(1)$ after linear time pre-processing.

9.6 BYPRODUCT: WORD REPRESENTATIONS

Language models can be trained on *raw text*: for training a k-order language model one just needs to extract $(k+1)$grams from running text, and treat the $(k+1)$th word as the supervision signal. Thus, we can generate practically unlimited training data for them.

Consider the matrix $\boldsymbol{W}^2$ appearing just before the final softmax. Each column in this matrix is a d_{hid} dimensional vector that is associated with a vocabulary item. During the final score computation, each column in $\boldsymbol{W}^2$ is multiplied by the context representation $\boldsymbol{h}$, and this produces the score of the corresponding vocabulary item. Intuitively, this should cause words that are likely to appear in similar contexts to have similar vectors. Following the distributional hypothesis according to which words that appear in similar contexts have similar meanings, words with similar meanings will have similar vectors. A similar argument can be made about the rows of the embedding matrix $\boldsymbol{E}$. As a byproduct of the language modeling process, we also learn useful word representations in the rows and columns of the matrices $\boldsymbol{E}$ and $\boldsymbol{W}^2$.

In the next chapter we further explore the topic of learning useful word representations from raw text.