Lecture 7: Distributed Representations

Roger Grosse

1 Introduction

We'll take a break from derivatives and optimization, and look at a particular example of a neural net that we can train using backprop: the neural probabilistic language model. Here, the goal is to model the distribution of English sentences (a task known as language modeling), and we do this by reducing it to a sequential prediction task. I.e., we learn to predict the distribution of the next word in a sentence given the previous words. This lecture will also serve as an example of one of the most important concepts about neural nets, that of a distributed representation. We can understand this in contrast with a localized representation, where a particular piece of information is stored in only one place. In a distributed representation, information is spread throughout the representation. This turns out to be really useful, since it lets us share information between related entities — in the case of language modeling, between related words.

2 Motivation: Language Modeling

Language modeling is the problem of modeling the probability distribution of natural language text. I.e., we would like to be able to determine how likely a given sentence is to be uttered. This is an instance of the more general problem of **distribution modeling**, i.e. learning a model which tries to approximate the distribution which some dataset is drawn from. Why would we want to fit such a model? One of the most important use cases is Bayesian inference.

Suppose we are building a speech recognition system. I.e., given an acoustic signal $\mathbf{a}$, we'd like to infer the sentence $\mathbf{s}$ (or a set of candidate sentences) that was probably spoken. One way to do this is to build a **generative model**. In this case, such a model consists of two probability distributions:

- The **observation model**, represented as $p(\mathbf{a} \mid \mathbf{s})$, which tells us how likely a sentence is to lead to a given acoustic signal. You might, for instance, build a model of the human vocal system. A lot of work has gone into this, but we're not going to talk about it here.

The notation $p(\cdot \mid \cdot)$ denotes the conditional distribution.

- The **prior**, represented as $p(\mathbf{s})$, which tells us how likely a given sentence is to be spoken, before we've seen $\mathbf{a}$. This is the thing we're trying to estimate when we do language modeling.

Given these two distributions, we can combine them using **Bayes' Rule** to infer the **posterior distribution** over sentences, i.e. the probability

distribution over sentences taking into account the observations. Recall that Bayes' Rule is as follows:

$$p(\mathbf{s} \mid \mathbf{a}) = \frac{p(\mathbf{s})\, p(\mathbf{a} \mid \mathbf{s})}{\sum_{\mathbf{s}'} p(\mathbf{s}')\, p(\mathbf{a} \mid \mathbf{s}')}. \tag{1}$$

The denominator is simply a normalization term, and we rarely ever have to compute it or deal with it explicitly. So we can leave the normalization implicit, using the notation $\propto$ to denote proportionality:

$$p(\mathbf{s} \mid \mathbf{a}) \propto p(\mathbf{s})\, p(\mathbf{a} \mid \mathbf{s}). \tag{2}$$

Hence, Bayes' Rule lets us combine our prior beliefs with an observation model in a principled and elegant way.

Having a good prior distribution $p(\mathbf{s})$ is very useful, since speech signals are inherently ambiguous. E.g., "recognize speech" sounds very similar to "wreck a nice beach", but the former is much more likely to be spoken. This is the sort of thing we'd like our language models to capture.

2.1 Autoregressive Models

Now we're going to recast the distribution modeling task as a sequential prediction task. Suppose we're given a corpus of sentences $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(N)}$. We'll make the simplifying assumption that the sentences are independent. This means that their probabilities multiply:

$$p(\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(N)}) = \prod_{i=1}^{N} p(\mathbf{s}^{(i)}). \tag{3}$$

Hence, we can talk instead about modeling the distribution over *sentences*.

We'll try to fit a model which represents a distribution $p_{\boldsymbol{\theta}}(\mathbf{s})$, parameterized by $\boldsymbol{\theta}$. The **maximum likelihood** criterion says we'd like to choose the $\boldsymbol{\theta}$ which maximizes the **likelihood**, or the probability of the observed data:

$$\max_{\boldsymbol{\theta}} \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(\mathbf{s}^{(i)}). \tag{4}$$

At this point, you might be concerned that the probability of any particular sentence will be vanishingly small. This is true, but we can fix that problem by working with log probabilities. Then the probability of the corpus conveniently decomposes as a sum:

$$\log \prod_{i=1}^{N} p(\mathbf{s}^{(i)}) = \sum_{i=1}^{N} \log p(\mathbf{s}^{(i)}). \tag{5}$$

Since it's easier to work with positive numbers, and log probabilities are negative, we often rephrase maximum likelihood as minimizing negative log probabilities.

The *log* probability of monkeys typing the entire works of Shakespeare is on a scale we can reasonably work with. And if slightly better trained monkeys are slightly more likely to type *Hamlet*, it will give us a smooth training criterion we can optimize with gradient descent.

What is this probability, under the assumption that they type all keys uniformly at random?

A sentence is a sequence of words A sentence is a sequence of words $w_1, w_2, \ldots, w_T$. The **chain rule of conditional probability** implies that

$p(\mathbf{s})$ factorizes as the products of conditional probabilities of individual words:

$$p(\mathbf{s}) = p(w_1, \ldots, w_T) = p(w_1)p(w_2 \mid w_1) \cdots p(w_T \mid w_1, \ldots, w_{T-1}). \quad (6)$$

Note that the Chain Rule applies to any distribution, i.e. we're not making any assumptions here.

Hence, the language modeling problem is equivalent to being able to predict the next word!

We typically make a **Markov assumption**, i.e. that the distribution over the next word only depends on the preceding few words. I.e., if we use a context of length 3, this means

$$p(w_t \mid w_1, \ldots, w_{t-1}) = p(w_t \mid w_{t-3}, w_{t-2}, w_{t-1}). \quad (7)$$

Such a model is called **memoryless**, since it has no memory of what occurred earlier in the sentence. When we decompose the distribution modeling problem into a sequential prediction task with limited context lengths, we call that an **autoregressive model**. "Regressive" because it's a prediction problem, and "auto" because the sequences are used as both the inputs and the targets.

Statisticians use "regression" to refer to general supervised prediction problems, not just least squares.

2.2 n-Gram Language Models

The simplest sort of Markov model is a **conditional probability table (CPT)**, where we explicitly represent the distribution over the next word given the context words. This is a table with a row for every possible context word senence, and a column for every word, and the entry gives the conditional probability. Since each row represents a probability distribution, the entries must be nonnegative, and the entries in each row must sum to 1. Otherwise, the numbers can be anything.

The simplest way to estimate a CPT is using the **empirical counts**, i.e. the number of times a sequence of words occurs in the training corpus. For instance,

3-gram

$$p(w_3 = \text{cat} \mid w_1 = \text{the}, w_2 = \text{fat}) = \frac{\text{count(the fat cat)}}{\text{count(the fat)}} \quad (8)$$

We'll show later in the course that the formula corresponds to the maximum likelihood estimate of the CPT.

This requires counting the number of occurrences of all sequences of length 2 and 3. Sequences of length n are called n-grams, and a model based on counting such sequences is called an n**-gram model**. For $n = 1, 2, 3$, these are called unigram, bigram, and trigram models. See here[1] for some examples of language models. Notice that unigram models are totally incoherent (since they sample all the words independently from the marginal distribution over words), but trigram models capture a fair amount of syntactic structure.

Gotcha: this example is a 3-gram model, even though it uses a context of length 2.

Observe that the number of possible contexts grows exponentially in n. This means that except for very small n, you're unlikely to see all possible n-grams in the training corpus, and many or most of the counts will be 0. This problem is referred to as **data sparsity**. The model described above is somewhat of a straw man, and natural language processing researchers came

#context = vocab_size ^ context_len

[1] https://lagunita.stanford.edu/c4x/Engineering/CS-224N/asset/slp4.pdf#page=10

up with a variety of clever ways for dealing with data sparsity, including adding imaginary counts of all the words, and combining the predictions of different context lengths.

But there's one problem fundamental to the n-gram approach: it's hard to share information between related words. If we see the sentence "The cat got squashed in the garden on Friday", we should estimate a higher probability o seeing the sentence "The dog got flattened in the yard on Monday", even though these two sentences have few words in common. Distributed representations give a great way of doing this.

2.3 Distributed Representations

Conditional probability tables are a kind of **localist representation**, which means a given piece of information (e.g. the probability of seeing "cat" after "the fat") is stored in just one place. If we'd like to share information between related words, we might want to use a **distributed representation**, where the same piece of information would be distributed throughout the whole representation. E.g., suppose we build a table of attributes of words:

	academic	**politics**	**plural**	**person**	**building**
students	1	0	1	1	0
colleges	1	0	1	0	1
legislators	0	1	1	1	0
schoolhouse	1	0	0	0	1

as well as the effect (+ or −) of those attributes on the probabilities of seeing possible next words:

	bill	**is**	**are**	**papers**	**built**	**standing**
academic	−			+		
politics	+			−		
plural		−	+			
person						+
building					+	+

grammatical

Information about the distribution over the next word is distributed throughout the representation. E.g., the fact that "students" is likely to be followed by "are" comes from the fact that "students" is plural, combined with the fact that plural nouns are likely to be followed by "are". Since "colleges" is also plural, this information is shared between "students" and "colleges".

3 Neural Probabilistic Language Model

Now let's talk about a network that learns distributed representations of language, called the **neural probabilistic language model**, or just **neural language model**. This network is basically a multilayer perceptron. It's an autoregressive model, so we have a prediction task where the input is the sequence of context words, and the output is the distribution over the next word. We associate each word in the dictionary with a unique and arbitrary integer index.

If we write out the negative log-likelihood for a sentence, it decomposes as the sum of cross-entropies for predicting each word:

minimize

$$-\log p(\mathbf{s}) = -\log \prod_{t=1}^{T} p(w_t \mid w_1, \ldots, w_{t-1}) \tag{9}$$

$$= -\sum_{t=1}^{T} \log p(w_t \mid w_1, \ldots, w_{t-1}) \tag{10}$$

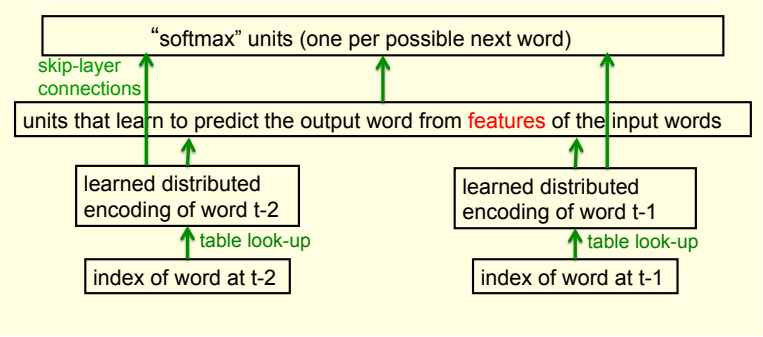
$$= -\sum_{t=1}^{T} \log y_{tv} \tag{11}$$

T is sentence_length; V is vocab_size

$$= -\sum_{t=1}^{T} \sum_{v=1}^{V} t_{tv} \log y_{tv}, \tag{12}$$

where $y_{tv} = p(w_t \mid w_1, \ldots, w_{t-1})$ is the predicted probability of the next word, and t_{tv} is the one-hot encoding of the target word. So this justifies using cross-entropy loss, just as we did in multiway classification.

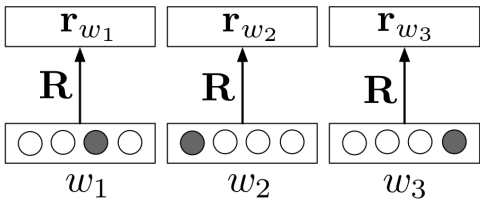
The neural language model uses the following architecture:



The only new concept here is the table look-up in the first layer. The network learns a **representation** of every word in the dictionary as a vector, and keeps these in a lookup table. This can be seen as a matrix $\mathbf{R}$, where each column gives the vector representation of one word. The network does one table lookup for each of the context words, and the activation vector for the **embedding layer** is the concatenation of the representations of all the context words.

There's another way to think of the embedding layer: suppose the context words are represented with one-hot encodings. Then we can think of the embedding layer as basically a linear layer whose weights are shared between all the context words. Recall that a linear layer just computes a matrix-vector product. In this case, we're multiplying the representation matrix $\mathbf{R}$ by the one-hot vectors, which corresponds to pulling out the corresponding column of $\mathbf{R}$.

You should convince yourself that this is the case.



After the embedding layer, there's a hidden layer, followed by a softmax output layer, which is what we'd expect if we're using cross-entropy loss. This architecture also includes a skip connection from the embedding layer to the output layer; we'll talk about skip connections later in the course, but roughly speaking, they help information travel faster through the network. This whole network can be trained using backpropagation, exactly as we've discussed in the previous lecture. You'll implement this for your first homework assignment.

There are various synonyms for word representation:

- **Embedding**, to emphasize that it's a location in a high-dimensional space. As we'll see, semantically related words should be close together.
- **Feature vector**, to emphasize that it picks out semantically relevant features that might be useful for downstream tasks. This is analogous to the polynomial feature mappings for polynomial regression, or the oriented edge filters in our MNIST classifier.
- **Encoding**, to emphasize that it's a sort of code, and that we can go back and forth between the words and their encodings.

Observe that unlike n-gram models, the neural language model is very compact, even for long context lengths. While the size of the CPTs grows exponentially in the context length, the size of the network (number of weights, or number of units) grows linearly in the context length. This means that we can efficiently account for much longer context lengths, such as 10.

The number of weights is linear only assuming the number of hidden units stays fixed. But in practice, we might need more hidden units to represent longer contexts.

If all goes well, the learned representations will reflect the semantic relationships between words. Here are two common ways to measure this:

- If two words are similar, the dot product of their representations, $\mathbf{r}_1^\top \mathbf{r}_2$, should be large.
- If two words are dissimilar, the Euclidean distance between their representations, $\|\mathbf{r}_1 - \mathbf{r}_2\|$, should be large.

These two criteria aren't equivalent in general, but they are equivalent in the case where $\mathbf{r}_1$ and $\mathbf{r}_2$ are both unit vectors:

$$\|\mathbf{r}_1 - \mathbf{r}_2\|^2 = (\mathbf{r}_1 - \mathbf{r}_2)^\top (\mathbf{r}_1 - \mathbf{r}_2) \tag{13}$$

$$= \mathbf{r}_1^\top \mathbf{r}_1 - 2\mathbf{r}_1^\top \mathbf{r}_2 + \mathbf{r}_2^\top \mathbf{r}_2 \tag{14}$$

$$= 2 - 2\mathbf{r}_1^\top \mathbf{r}_2 \tag{15}$$

If the representations are unit vectors, $\mathbf{r}_1^\top \mathbf{r}_2$ is also referred to as **cosine similarity**, since it is the cosine of the angle between the representations.

2 vector with same orientation has similarity of 1,

To visualize the learned word vectors, we need to somehow map them down to two dimensions. There's an algorithm called tSNE that does just that. Roughly speaking, it tries to assign locations to all the words in two dimensions to match the high-dimensional distances as closely as possible. This is impossible to do exactly (e.g. you can't map the vertices of a cube to 2 dimensions while preserving all the distances), and the low-dimensional representation introduces distortions. E.g., words that are far away in high

dimensions might be put close together in 2-D. But it is still a pretty instructive visualization. Here[2] is an example of a tSNE visualization of word representations learned by a different model, but one based on similar principles. Notice that semantically similar words get grouped together.

[2]http://www.cs.toronto.edu/ hinton/turian.png