Chapter 6

Semantics

When we began our study of syntax, I had to spend some time convincing you that syntax exists (that is, that there is some kind of representation of syntactic structure in your mind when you use language), but the basic shape of those representations has not been all that controversial.

By contrast, as we turn our attention to *semantics*, or meaning, I probably don't need to convince you at all that meaning exists, but in theories of semantics and computational approaches to semantics, representations of meaning come in all kinds of shapes and sizes. As I see it, there are three main approaches to representing meaning in NLP:

1. Vectors: whose components are features, either designed by humans or automatically learned.
2. Graphs: in which nodes are usually entities and edges are various kinds of relationships among them.
3. Logic: formulas of logics of various kinds, or SQL queries, or even computer programs.

We should also distinguish between meanings of words, known as *lexical semantics*, and the meanings of sentences, which is usually just called *semantics*.

6.1 Vectors

Possibly the simplest way to represent the "meaning" of a text is to assume that each word in the text contributes a little bit of meaning, which we can represent as a one-hot vector, and that we can combine the "meanings" of words simply by adding up their vectors.

$$\text{the cat sat on the mat} \rightarrow \{\text{cat} : 1, \text{mat} : 1, \text{on} : 1, \text{sat} : 1, \text{the} : 2\}$$

In information retrieval, this is called a *term vector*, but in NLP it's more commonly known as a *bag of words*. This representation completely ignores any semantic relationships between words (like synonymy) and any kind of structure of the text (like word order or syntax). Yet, in many situations, it can be very effective.

6.1.1 Classification

To get started, it's convenient to focus on the problem of text classification. Suppose we have a collection of documents in different classes, and we want to learn to classify new documents. For example:

- We have a collection of e-mails that are labeled either as "spam" or "ham" and we want to learn to classify new e-mails.
- We have some texts whose author is known (e.g., Alexander Hamilton, James Madison, and John Jay) and want to classify anonymous texts (the Federalist Papers).
- We have a database of product reviews together with star ratings, and we want to be able to predict star ratings based on reviews alone.
- We have samples of (transcribed) speech from children diagnosed as autistic or not autistic, and we want to automatically diagnose other children using their speech.

Question 14. What are some other possible applications?

We'll continue working with the example of spam filtering, but just remember that this is only one of many possible applications. This is a fairly easy problem, but occasionally difficult even for humans. For example:

```
From: fas@nsf.gov
Subject: Payment
To: chiang@isi.edu

DFM has approved your requested payment and has asked the U.S. Treasury to
issue a payment to you within the next 4 working days. This payment is
being sent directly to the Bank or Financial Institution identified by
you for this purpose. If you are an NSF employee, this payment is being
sent directly to the bank/financial institution where your bi-weekly pay
is being deposited.

This payment for 560.00 is 1099 reportable if it totals $600 or more
for the year (i.e. You will receive an IRS 1099-Misc form from
NSF)P131318.

Head, Accounts Payable Section.
-------------------------------
```

More formally, we are given documents $d_1, \ldots, d_n$ together with their correct classes $k_1, \ldots, k_n$. We want to learn a model $P(k \mid d)$, where k is a class and d is a document, and given a new document d, we want to be able to find, with high accuracy,

$$k^* = \arg\max_k P(k \mid d). \tag{6.1}$$

Naïve Bayes

Instead of thinking about how to classify a document, let's think about how the document came to be. First, someone decided to write an e-mail; he was either a

spammer or a "hammer." Then, this person authored a document:

$$\arg\max_k P(k \mid d) = \arg\max_k P(d)P(k \mid d) \tag{6.2}$$
$$= \arg\max_k P(k, d) \tag{6.3}$$
$$= \arg\max_k P(k)P(d \mid k). \tag{6.4}$$

As we've seen in other situations, the advantage of thinking about it backwards like this is that it is much easier to write down a model for $P(d \mid k)$ than for $P(k \mid d)$.

The simplest way to define $P(k)$ and $P(d \mid k)$ is using categorical distribution and a unigram language model, respectively. The result is called a *naïve Bayes* classifier:

$$P(k, d) = p(k) \prod_{w \in d} p(w \mid k), \tag{6.5}$$

where the $p(k)$ and $p(w \mid k)$ are the parameters of the model. (I'm using uppercase P for all kinds of probabilities, but lowercase p specifically for model parameters that are to be learned.)

Naïve Bayes is called naïve because it naïvely assumes that all the words in a document are independent of each other. All it captures is that spammers are more likely to use certain words and hammers are more likely to use certain words.

Training the model, or estimating the parameters $p(k)$ and $p(w \mid k)$, is easy. It's just:

$$\begin{aligned} p(k) &= \frac{c(k)}{\sum_k c(k)} \\ p(w \mid k) &= \frac{c(k, w)}{\sum_{w'} c(k, w')}. \end{aligned} \tag{6.6}$$

Once we've trained the model, finding the most probable class of a new document is also easy:

$$k^* = \arg\max_k P(k \mid d) \tag{6.7}$$
$$= \arg\max_k P(k, d) \tag{6.8}$$
$$= \arg\max_k p(k) \prod_{w \in d} p(w \mid k). \tag{6.9}$$

A classic dataset for text classification is Pang and Lee's dataset of movie reviews.[1] Here are some example sentences from positive reviews:

```
the rock is destined to be the 21st century's new " conan " and that
he's going to make a splash even greater than arnold schwarzenegger ,
jean-claud van damme or steven segal .

the gorgeously elaborate continuation of " the lord of the rings "
trilogy is so huge that a column of words cannot adequately describe
```

[1] http://www.cs.cornell.edu/people/pabo/movie-review-data/

```
co-writer/director peter jackson's expanded vision of j . r . r
. tolkien's middle-earth .

effective but too-tepid biopic
```

And some example sentences from negative reviews:

```
simplistic , silly and tedious .

it's so laddish and juvenile , only teenage boys could possibly find
it funny .

exploitative and largely devoid of the depth or sophistication that
would make watching such a graphic treatment of the crimes bearable .
```

When we run the naive Bayes classifier on this data, we get an accuracy of 75.7%. Some of the model probabilities are shown in Figure 6.1. Some of them are pretty obvious ("bad" is usually a pretty good indicator of a bad movie), but some are more interesting, like "cinema" being associated with good movies.

Logistic regression

Another approach is to directly model $P(k \mid d)$. Given that we've developed notation for neural networks already, the simplest way to write this is as a little neural network:

$$\mathbf{x}^{(i)} \in \mathbb{R}^{|\Sigma|} \qquad \text{one-hot vectors } (i = 1, \ldots, n) \tag{6.10}$$

$$\mathbf{d} = \sum_i \mathbf{x}^{(i)} \qquad \text{document vector} \tag{6.11}$$

$$\mathbf{y} = \log \text{softmax}(\mathbf{W}\mathbf{d} + \mathbf{b}) \qquad \mathbf{W} \in \mathbb{R}^{2 \times |\Sigma|}, \mathbf{b} \in \mathbb{R}^2 \tag{6.12}$$

$$\log P(k \mid d) = \mathbf{y}_k \tag{6.13}$$

However, this formulation doesn't make very clear how closely related to naive Bayes this is. The parameters $\mathbf{W}_{k,w}$ correspond to $p(w \mid k)$ except that they are not constrained to be probabilities, and the parameters $\mathbf{b}_k$ correspond to $p(k)$.

On the movie-reviews dataset, this gets a test accuracy of 77.9%, and some of the feature values (parameters of the SoftmaxLayer) are shown in Figure 6.2.

Neural text classification

The above formulation does make it very easy to imagine how to extend the model to a deeper neural network. We can add a word-embedding layer (using D for the number of dimensions, since d stands for documents):

$$\mathbf{x}^{(i)} \in \mathbb{R}^{|\Sigma|} \qquad \text{one-hot vectors } (i = 1, \ldots, n) \tag{6.14}$$

$$\mathbf{e}^{(i)} = \mathbf{V}\mathbf{x}^{(i)} \qquad \mathbf{V} \in \mathbb{R}^{D \times |\Sigma|} \tag{6.15}$$

$$\mathbf{d} = \sum_i \mathbf{e}^{(i)} \qquad \text{document vector} \tag{6.16}$$

$$\mathbf{y} = \log \text{softmax}(\mathbf{W}\mathbf{d} + \mathbf{b}) \qquad \mathbf{W} \in \mathbb{R}^{2 \times D}, \mathbf{b} \in \mathbb{R}^2 \tag{6.17}$$

$$\log P(k \mid d) = \mathbf{y}_k \tag{6.18}$$

$$
\begin{aligned}
p(+) &= 0.5 \\
p(-) &= 0.5 \\
p(\text{moving} \mid +) &= 0.000582 \\
p(\text{portrait} \mid +) &= 0.000463 \\
p(\text{touching} \mid +) &= 0.000404 \\
p(\text{powerful} \mid +) &= 0.000394 \\
p(\text{engrossing} \mid +) &= 0.000296 \\
p(\text{cinema} \mid +) &= 0.000611 \\
p(\text{beautiful} \mid +) &= 0.000404 \\
p(\text{culture} \mid +) &= 0.000374 \\
p(\text{enjoyable} \mid +) &= 0.000453 \\
p(\text{wonderful} \mid +) &= 0.000286 \\
p(\text{bad} \mid -) &= 0.001862 \\
p(\text{dull} \mid -) &= 0.000657 \\
p(\text{boring} \mid -) &= 0.000488 \\
p(\text{too} \mid -) &= 0.003285 \\
p(\text{flat} \mid -) &= 0.000328 \\
p(\text{tv} \mid -) &= 0.000388 \\
p(\text{worst} \mid -) &= 0.000458 \\
p(\text{unfunny} \mid -) &= 0.000258 \\
p(\text{jokes} \mid -) &= 0.000378 \\
p(? \mid -) &= 0.001473
\end{aligned}
$$

Figure 6.1: Some of the probabilities of the naive Bayes classifer learned from the movie review dataset. The words shown above are *not* the most frequent words; those are rather boring. Instead, we selected the words with the highest $p(w \mid k)/(p(w) + 10)$.

$$
\begin{aligned}
\mathbf{b}_{+} &= -0.0582295576575 \\
\mathbf{b}_{-} &= 0.0582871175659 \\
\\
\mathbf{W}_{+,\text{engrossing}} &= 0.489182061478 \\
\mathbf{W}_{+,\text{wonderful}} &= 0.476123720915 \\
\mathbf{W}_{+,\text{enjoyable}} &= 0.471651953627 \\
\mathbf{W}_{+,\text{cinema}} &= 0.459620146853 \\
\mathbf{W}_{+,\text{unexpected}} &= 0.453405446842 \\
\mathbf{W}_{+,\text{powerful}} &= 0.449412825023 \\
\mathbf{W}_{+,\text{provides}} &= 0.445648025943 \\
\mathbf{W}_{+,\text{solid}} &= 0.424138376044 \\
\mathbf{W}_{+,\text{rare}} &= 0.422520925496 \\
\mathbf{W}_{+,\text{refreshing}} &= 0.410572454316 \\
\\
\mathbf{W}_{-,\text{dull}} &= 0.621595244267 \\
\mathbf{W}_{-,\text{worst}} &= 0.556222428753 \\
\mathbf{W}_{-,\text{boring}} &= 0.526661381249 \\
\mathbf{W}_{-,\text{fails}} &= 0.494008518961 \\
\mathbf{W}_{-,\text{unfunny}} &= 0.487500395879 \\
\mathbf{W}_{-,\text{flat}} &= 0.467475895425 \\
\mathbf{W}_{-,\text{mess}} &= 0.455630761843 \\
\mathbf{W}_{-,\text{neither}} &= 0.442770935893 \\
\mathbf{W}_{-,\text{lack}} &= 0.43677543437 \\
\mathbf{W}_{-,\text{mediocre}} &= 0.427000091059
\end{aligned}
$$

Figure 6.2: Some of the feature weights learned by logistic regression.

The document vector continues to be the sum of the word vectors, which is a simple and common scheme.

We can also add more layers between the embedding and the summation, like RNNs or self-attention. With an RNN, an alternative to summing all the word encodings would be simply to take the last hidden state of the RNN. With self-attention, an alternative to summing would be to create a fake word `CLS` and using the encoding of `CLS`.

6.1.2 Topic modeling

In the models of the previous section, we had vector representations of words, and we formed vector representations of documents by summing the word vectors. We also could have thought of the $\mathbf{W}_{k,:}$ as vector representations of classes.

In this section, we return to the naive Bayes classifier and modify it to obtain a different way to represent documents as vectors. These vectors are *very* widely used in text analysis in the humanities and social sciences.

Naïve Bayes with topics

Recall that we modified the logistic regression model by introducing a word embedding layer. We can do the same thing to the naive Bayes classifier by introducing a variable t (for *topic*):

$$P(k, d) = p(k) \prod_{w \in d} \sum_{t} p(t \mid k) p(w \mid t). \tag{6.19}$$

The topic t is drawn from a finite set of predetermined size D. We are given training data consisting of documents d_i and classes k_i, but we are not given the topics of the words in the d_i.

The class distribution $p(k)$ is estimated by counting and dividing:

$$p(k) = \frac{c(k)}{\sum_k c(k)}. \tag{6.20}$$

The rest of the model is not as simple because of the hidden variables. We want to maximize the log-likelihood,

$$\log L = \sum_i \left(\log p(k_i) + \sum_{w \in d_i} \log \sum_t p(t \mid k_i) p(w \mid t) \right).$$

One way to do this is using stochastic gradient ascent, which we can do by using the (by now) usual trick of expressing the probability distributions as softmaxes of logit parameters.

$$p(t \mid k) = \operatorname{softmax} \lambda(t' \mid k) = \frac{\exp s(k, t)}{\sum_{t'} \exp s(k, t')}$$
$$p(w \mid t) = \operatorname{softmax} \lambda(w' \mid t) = \frac{\exp s(t, w)}{\sum_{w'} \exp s(t, w')}$$

We ran this model on the movie dataset, using 2 topics, 25 iterations, and add-10^{-6} smoothing, and got an accuracy of 77.4%, which is a small improvement over the original model (75.7%).

With 20 topics, here's what the model learns:

topic	words
0	stupid boring dull flat routine bad mess unfunny fails neither
1	funny both drama us look performances romantic entertaining world those
2	too lack bad silly title worst already ? mess tedious
3	compelling ride fascinating both entertaining works surprisingly heart family experience
4	made up through ' all never out comes audience than
5	worst too tries bad gross-out things ? title premise tv
6	engrossing beautiful refreshingly terrific portrait quiet riveting thoughtful moving enjoyable
7	poorly dull loud bad flat boring jokes generic numbers badly
8	long just isn't where when be doesn't plot movie or
9	de drama film love with your and their you life
10	who often story their picture cast and funny movies you
11	that's up quite as lot for little which it's out
12	remarkable quiet our relationships study moving fascinating culture performances powerful
13	compelling smart performances personal family portrait world intelligent different our
14	far can one but that in little , itself feel
15	unfunny flat boring bad mediocre dull plodding tv product badly
16	many not never it into has time (from but
17	vividly lively provides touching warm wonderful disturbing powerful engrossing riveting
18	film gives picture makes without you with at first and
19	too bad title jokes fails feels problem nothing silly tv

Many of the topics are clearly dominated by positive or negative words. There are also seem to be some topics with neutral words (4, 8, 9, 10, 11, 14, 16, 18). It's hard to guess what might further differentiate the topics (if anything).

Probabilistic LSA

The topics above are colored by the choice of classes (positive versus negative reviews); if we want the topics to be generic, we can put *each document in its own class*. Intuitively, that's like building a classifier that, given a test document, will choose the training document that it's the most similar to. But the point here is not classification; what we're interested in is the probabilities $p(t \mid k_i)$, which can be thought of as a vector representation of document d_i, and the probabilities $p(t \mid w)$, which can be thought of as a vector representation of word w. This is called *probabilistic latent semantic analysis* (PLSA) or *probabilistic latent semantic indexing* (PLSI) (Hofmann, 1999).

We ran PLSA using the same settings as before, but for more iterations. This time, the topics we got were:

topic	words
0	made last since fans once passion again perfect cinema animated
1	? don't man video stuff satisfying problem what's how why
2	far journey summer getting job surprises surprise water setting crush
3	works worth genre idea seeing cliches already appeal add female
4	documentary heart mind fascinating do difficult satire strong light filmmakers
5	her romantic truly formula girl start delightful major puts play
6	there's those lot women enjoy need beautiful close something real
7	your you'll flat deep throughout murder forgettable coming-of-age seriously [a]
8	" other ! comes year silly ending horror less five
9	if fun because watch mr you're beyond hilarious sad witty
10	too quite moving place amusing visually human somewhat important appealing
11	performance keep sometimes romance melodrama compelling wrong wonderful strange straight
12	entertaining family comic mostly surprisingly touching ride leaves brilliant quiet
13	de short hard psychological que falls y la ideas o
14	bad study sort rarely gives entire modern fire ugly awful
15	piece part fine beautifully humor pleasure storytelling did usual solid
16	doesn't care lacks power else yet work they rich impossible
17	moments my thing dramatic neither nor mess there funny career
18	i like feels me pretty still looks left long worst
19	direction seem everyone opera help tedious soap nature alone predictable

Note how different these topics look from before. None are strongly positive or negative, because the model is no longer receiving information about positivity or negativity. But there doesn't seem to be that much coherence in general in these topics.

To get more interesting topics, I ran the same algorithm on 1000 posts from Reddit's r/notredame:

topic	words
0	ted insurance gsu liams representatives hesburgh coronavirus aetna abroad pride
1	band grades fail p minecraft lizzy browns nfl pass alma
2	updated colors recovered arms r_0 * :* fatality parameters shields
3	engineering meet advice clubs friends club summer lonely freshman join
4	returning william aug. practices services july programs travel fall >
5	| he hr him trump manager pro barrett leader court
6	winter links ✓ whistle screwing pic break rise wimbush stress
7	racism shirt expelled race color racist minority scene hijab privilege
8	lsu jpw tcu table helmets meme shoulder ap turf row
10	vaccination water dose vaccinated vaccine conference receive hypocritical jenkins april
11	individuals · antigen cole • wellness diagnosed tracing surveillance august
12	cough johnson gpa legacy pope usccb allergies fear blame depression
13	cable tv computer tonelli phones flanner labs /u cell phone
14	irish michigan congrats ♣ win fans season game championship teams
15	renovations jerry parking halls residential salad hoax car manti mike
16	tests total wifi yards numbers data fewer companies gipp sample
17	negative tested ucc therapy test senate tracer facts quarantine rapid
18	diversity rich diverse religion religious uva catholic commencement racial pretentious
19	cafe mishawaka expensive food –will pizza pho places sushi breakfast

Although there are some mixed topics, it's pretty easy to see what most topics are about.

Latent Dirichlet Allocation

No one uses PLSA anymore. *Latent Dirichlet Allocation* (Blei, A. Y. Ng, and Jordan, 2003) is a further development of PLSA that is by far the most widely known and used topic model. LDA is more math-heavy and we don't give a full description here, but just a little bit of the basic idea.

Let's forget about language for a little while and talk about coin flips. Suppose I flip a coin and show it to you: it lands heads. Then I ask you to bet on whether the second flip will be heads or tails.

$$P(C_2 \mid C_1 = \text{heads}) = ? \tag{6.21}$$

There are many possible answers; for example:

- Based on what I know about how coins work, $P(C_2 = \text{heads}) = \frac{1}{2}$.
- Based on my observation, $P(C_2 = \text{heads}) = \frac{c(\text{heads})}{c(\text{heads})+c(\text{tails})} = 1$.

If we were doing maximum-likelihood estimation, as we have been doing all semester, then the latter would indeed be the right answer. But this is obviously not a rational conclusion. To make sense out of this, we have to introduce the idea of a *prior* distribution. Suppose for some reason (the real reason is I want to avoid doing calculus) that we have just three hypotheses about the coin:

(HT) It's a fair coin.

(HH) Both sides are heads.

(TT) Both sides are tails.

So we can write:

$$P(\text{heads} \mid \text{HT}) = \tfrac{1}{2} \qquad P(\text{tails} \mid \text{HT}) = \tfrac{1}{2} \tag{6.22}$$

$$P(\text{heads} \mid \text{HH}) = 1 \qquad P(\text{tails} \mid \text{HH}) = 0 \tag{6.23}$$

$$P(\text{heads} \mid \text{TT}) = 0 \qquad P(\text{tails} \mid \text{TT}) = 1 \tag{6.24}$$

Before observing any coin flips, you thought that these three hypotheses were equally likely. This is your prior probability.

$$P(\text{HT}) = P(\text{HH}) = P(\text{TT}) = \tfrac{1}{3}. \tag{6.25}$$

Now, after observing a coin flip, you think:

$$P(C_2 = \text{heads} \mid C_1 = \text{heads}) = \frac{P(C_1 = \text{heads}, C_2 = \text{heads})}{P(C_1 = \text{heads})} \tag{6.26}$$

$$= \frac{\sum_{\theta=\text{HT,HH,TT}} P(C_1 = \text{heads}, C_2 = \text{heads}, \theta)}{\sum_{\theta=\text{HT,HH,TT}} P(C_1 = \text{heads}, \theta)} \tag{6.27}$$

$$= \frac{\sum_{\theta=\text{HT,HH,TT}} P(\theta)\, P(C_1 = \text{heads}, C_2 = \text{heads} \mid \theta)}{\sum_{\theta=\text{HT,HH,TT}} P(\theta)\, P(C_1 = \text{heads} \mid \theta)} \tag{6.28}$$

$$= \frac{\frac{1}{3} \cdot \frac{1}{4} + \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 0}{\frac{1}{3} \cdot \frac{1}{2} + \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 0} \tag{6.29}$$

$$= \tfrac{5}{6}. \tag{6.30}$$

Now, imagine that we don't have three discrete hypotheses, but infinitely many hypotheses, $P(\text{heads} \mid \theta) = \theta$ for any $\theta \in [0, 1]$. The sums over θ become integrals, but the idea remains the same. Interestingly, if we continue with a uniform prior distribution over θ's, we end up with add-one smoothing.

Back to PLSA, we have a model of classes, topics, and documents, and if we're only interested in the topics, and not θ, then we shouldn't try to estimate θ. Instead, we should integrate over all possible values of θ:

$$P(\text{topics} \mid \text{data}) = \frac{P(\text{topics}, \text{data})}{P(\text{data})} \tag{6.31}$$

$$= \frac{\int P(\text{topics}, \text{data}, \theta)\, \mathrm{d}\theta}{\int P(\text{data}, \theta)\, \mathrm{d}\theta} \tag{6.32}$$

That looks bad, and in all but the simplest cases, it is bad, but that doesn't stop people from trying to do it anyway. There are two main ways of approximating these integrals: one way tries to approximate a difficult-to-integrate function with a simpler one (*variational* methods), and the other tries to use random sampling (*Monte Carlo* methods). The latter generally work better. For further information, see the references listed below.

6.2 Graphs

The next broad category of semantic representation I want to talk about is semantic *graphs*. I'm including under this heading a bunch of tasks like

- Named entity recognition: Which noun phrases are names of people, places, etc., and what entity do they refer to?
- Coreference resolution: Which noun phrases (including pronouns) refer to the same thing?
- Word sense disambiguation: If a word has more than one sense, which one is being used in this context?
- Semantic role labeling: For each action in the sentence, who/what is the agent (the one doing the action), the patient (the one upon whom the action is done), etc.?
- Relation extraction: Sometimes we're interested in higher-level relationships between entities, like "x is the president of y."

Ultimately, the entities and relationships that these tasks find can be assembled into a graph – either a graph that represents the meaning of a sentence, or a big graph that represents the knowledge contained in a whole collection of text.

6.2.1 Named entity recognition

Among these more focused tasks, I'd like to focus on one, named entity recognition (NER), both because it's especially useful in a lot of applications, and because the standard method for NER is one that we haven't seen before.

Problem

The NER problem is, given a string, which strings are named entities and what kind of named entity they are. Typical types are:

- Person
- Organization
- Location (also called GPE or geopolitical entity)
- Miscellaneous

(These types are from the CoNLL 2003 shared task, the most widely used (and probably overused) dataset for NER.)

A natural next step would be *entity linking*, which is to determine what person, organization, location, in the real world a given substring refers to. (Commonly, the URL of an entity's Wikipedia entry is used as a proxy for "the real world.")

BIO tagging

The usual first step in NER is to reduce it to a *sequence labeling* problem, where we have to assign a label to every word in a sequence. Other examples of sequence labeling are part-of-speech tagging and slot-filling (e.g., in ATIS, identifying the departure/arrival airport/date/time).

To reduce NER (and slot-filling) to a sequence-labeling problem, we use *BIO tagging*:

B-ORG	I-ORG	O	B-PER	I-PER	O	O	B-LOC	O
United	Nations	official	Rolf	Ekeus	heads	for	Baghdad	.

B stands for "begin" and is used for the first word in each entity; I stands for "inside" and is used for the second and subsequence words in each entity. O stands for "outside" and is used for any word that does not belong to an entity. Other schemes exist, like BILOU (L for the last word an entity, U 'unit' for the only word in an entity), but this is the simplest and most common.

RNN-CRFs

An RNN-CRF (more often referred to in the literature as a biLSTM-CRF, where a biLSTM is the specific kind of RNN used) is very similar to the neural parsing model we saw in the last chapter. The neural parsing model was an RNN encoder with a weighted CFG on top; this model is an RNN encoder with a weighted finite automaton on top.

The input is a string $\mathbf{w} = w_1 \cdots w_n$. Let T be the set of possible tags. We start off with an RNN encoder, then a linear layer to get a score $\mathbf{S}_{i,t}$ for each position

$i \in [1, n]$ and each possible tag t:

$$\mathbf{V} \in \mathbb{R}^{n \times d}$$
$$\mathbf{V}_i = \text{Embedding}^{\boxed{1}}(w_i) \qquad i = 1, \ldots, n \tag{6.33}$$
$$\mathbf{H} \in \mathbb{R}^{n \times d}$$
$$\mathbf{H} = \text{RNN}^{\boxed{2}}(\mathbf{V}) \tag{6.34}$$
$$\mathbf{S} \in \mathbb{R}^{n \times |T|}$$
$$\mathbf{S} = \text{LinearLayer}^{\boxed{3}}(\mathbf{H}). \tag{6.35}$$

In a language model, you'd expect something like a softmax to predict the next word; in the neural parser, we had a "super-softmax" over trees. Here, we want a "super-softmax" over all sequences of tags, $\mathbf{t}$:

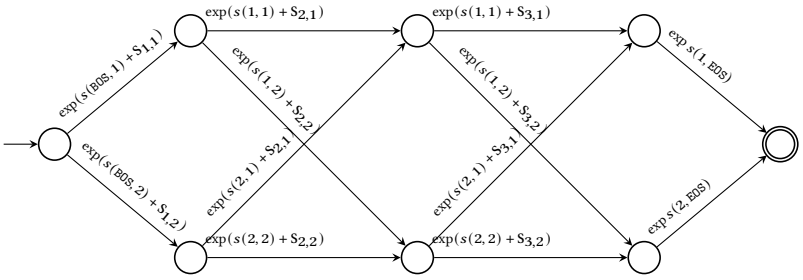
$$P(\mathbf{t} \mid \mathbf{w}) = \frac{\exp s(\mathbf{t})}{\sum_{\mathbf{t}'} \exp s(\mathbf{t}')} \tag{6.36}$$

where the summation over $\mathbf{t}'$ sums over all tag sequences of length n, and $s(\cdot)$ is the score of a tag sequence:

$$s(\mathbf{t}) = \sum_{i=1}^{n} \left(s(t_{i-1}, t_i) + \mathbf{S}_{i,t_i} \right) + s(t_n, \texttt{EOS}). \tag{6.37}$$

where $t_0 = \texttt{BOS}$. This $s(t_{i-1}, t_i)$ is reminiscent of a bigram (log-)probability, except that it can be any positive or negative number.

As before, we're now left with two challenges: (1) During training, we need to compute the sum $\sum_{\mathbf{t}'} \exp s(\mathbf{t}')$, and (2) during testing, we need to compute $\arg\max_{\mathbf{t}} P(\mathbf{t} \mid \mathbf{w})$. We solve both challenges with this finite automaton (shown for $n = 3$, $|T| = 2$):



Each path through this automaton corresponds with a possible tag sequence $\mathbf{t}$. The weight of this path is $\exp s(\mathbf{T})$. We saw a *very* long time ago how to compute the total weight of all paths (eqs. 2.13–2.14); here it is in terms of the current notation:

$$\alpha_{1,t'} = \exp(s(\texttt{BOS}, t') + \mathbf{S}_{1,t'}) \tag{6.38}$$
$$\alpha_{i,t'} = \sum_{t} \alpha_{i,t} \exp(s(t, t') + \mathbf{S}_{i,t'}) \qquad i = 2, \ldots, n \tag{6.39}$$
$$\alpha_{n+1,\texttt{EOS}} = \sum_{t} \alpha_{n,t} \exp s(t, \texttt{EOS}). \tag{6.40}$$

Then

$$\sum_{\mathbf{t}} \exp s(\mathbf{t}) = \alpha_{n+1,\texttt{EOS}}. \tag{6.41}$$

To find the best tag sequence $\mathbf{t}$ given a word sequence $\mathbf{w}$, we do the same computation, but everywhere there's a sum, we do a max.

6.2.2 Semantic graphs

The first part of this section is mostly from an earlier paper (Chiang et al., 2018).

Above we mentioned various semantics-related tasks like semantic role labeling (Gildea and Jurafsky, 2000), word sense disambiguation (Brown et al., 1991), coreference resolution (Soon, H. T. Ng, and Lim, 2001), and so on. Resources like OntoNotes (Hovy et al., 2006) provided separate resources for each of these tasks.

Some more recent work in semantic processing tries to consolidate these tasks into one. For example, the Abstract Meaning Representation (AMR) Bank (Banarescu et al., 2013) began as an effort to unify the various annotation layers of OntoNotes. Others include: the Prague Dependency Treebank (Böhmová et al., 2003), DeepBank (Oepen and Lønning, 2006), and Universal Conceptual Cognitive Annotation (Abend and Rappoport, 2013). By and large, these resources are based on, or equivalent to, *graphs*, in which vertices stand for entities and edges stand for semantic relations among them.

Abstract Meaning Representations

Here, I'll focus on AMRs, just because they're the representation I'm most familiar with. AMRs can be written in a serialized form or as directed graphs. Examples of these two representations, from the AMR Bank (LDC2014T12), are reported in Figure 6.3 and Figure 6.4. Nodes are labeled, in order to convey lexical information. Edges are labeled to convey information about semantic roles. Labels at the edges need not be unique, meaning that edges impinging on the same node might have the same label. Furthermore, our DAGs are not ordered, meaning that there is no order relation for the edges impinging at a given node, as is usually the case in standard graph structures. A node can appear in more than one place (for example, in Figure 6.3, node `s2` appears six times).

The numbers (e.g., ask-01) require some explanation. These are from PropBank (Palmer, Gildea, and Kingsbury, 2005), which catalogues and numbers, for each verb, the different senses of the verb and ways it can be used. For example,

- ask-01 is for asking questions
- ask-02 is for asking favors
- ask-03 is for asking a price
- ask_out-04 is for asking someone on a date.

Each of these senses comes with a numbered list of arguments. For example, for ask-01,

```
(a / and
      :op1 (a2 / ask-01
            :ARG0 (i / i)
            :ARG1 (t / thing
                  :ARG1-of (t2 / think-01
                        :ARG0 (s2 / she)
                        :ARG2 (l / location
                              :location-of (w / we))))
            :ARG2 s2)
      :op2 (s / say-01
            :ARG0 s2
            :ARG1 (a3 / and
                  :op1 (w2 / want-01 :polarity -
                        :ARG0 s2
                        :ARG1 (t3 / think-01
                              :ARG0 s2
                              :ARG1 l))
                  :op2 (r / recommend-01
                        :ARG0 s2
                        :ARG1 (c / content-01
                              :ARG1 i
                              :ARG2 (e / experience-01
                                    :ARG0 w))
                        :ARG2 i))
            :ARG2 i)
      :op3 c)
```

Figure 6.3: Example AMR in its standard format, number `DF-200-192403-625_0111.7` from the AMR Bank. The sentence is: "I asked her what she thought about where we'd be and she said she doesn't want to think about that, and that I should be happy about the experiences we've had (which I am)."

- arg0 is the asker
- arg1 is the question
- arg2 is the hearer.

AMR parsing

Semantic parsing is the task of taking a natural language sentence and mapping it to a representation of its meaning. If the semantic representation is AMR, we call this AMR parsing.

These days, it's easy to build a barebones AMR parser – just run a neural machine translation system on parallel text consisting of English sentences and their AMRs in textual format (Figure 6.3). Research on AMR parsing has gotten

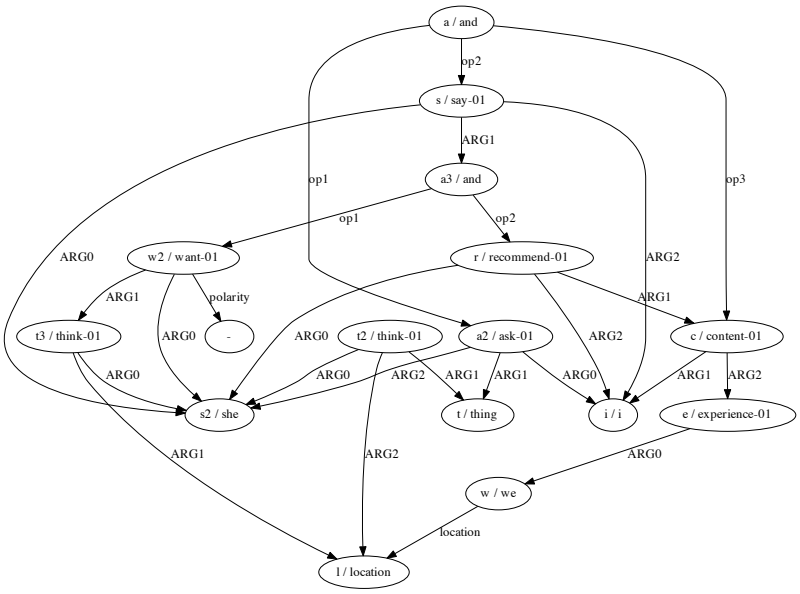


Figure 6.4: The AMR of Figure 6.3, presented as a directed graph.

plenty more sophisticated than that, but here I want to focus on one improvement to the basic NMT system (which you will implement in HW4).

Unlike language-to-language translation, it's typical for the AMR to have words in common with the source text. The NMT system can do a good job learning to copy these words if they're frequent, but for rare words, it may do so less reliably, and for unknown words, it will be unable to. So we want to add a *copy* mechanism to the model.

The basic idea is to introduce a fake target word, `COPY`, which instructs the system to copy a word from the source sentence. Which word? We use the source-to-target attention, which is a distribution over source positions, to choose one source word.

We're only interested in the last two steps of the model, which were the same in our presentation of both the RNN and Transformer models. These are for the context vector $\mathbf{c}^{(i)}$, and the output word distribution:

$$\begin{aligned}
\mathbf{g}^{(i)} &\in \mathbb{R}^d \\
\mathbf{H} &\in \mathbb{R}^{n \times d} \\
\mathbf{o}^{(i)} &\in \mathbb{R}^d \\
\mathbf{c}^{(i)} &\in \mathbb{R}^d \\
\mathbf{c}^{(i)} &= \text{Attention}(\mathbf{g}^{(i)}, \mathbf{H}, \mathbf{H}) \qquad (6.42) \\
P(e_{i+1}) &= \text{SoftmaxLayer}^{\boxed{6}}(\mathbf{o}^{(i)}). \qquad (6.43)
\end{aligned}$$

The equation for $\mathbf{c}^{(i)}$ computes both the attention and the weighted average,

so we're going to "break it open" to get at the attention inside:

$$\alpha^{(i)} \in \mathbb{R}^n \tag{6.44}$$

$$\alpha^{(i)} = \text{softmax}\,\mathbf{H}\mathbf{g}^{(i)} \tag{6.45}$$

The output word distribution now includes COPY. We modify this to:

$$\mathbf{p}^{(i)} \in \mathbb{R}^n \tag{6.46}$$

$$\mathbf{p}^{(i)} = \text{SoftmaxLayer}^{\boxed{6}}(\mathbf{o}^{(i)}) \tag{6.47}$$

$$P(e) = \mathbf{p}_e^{(i)} + \mathbf{p}_{\texttt{COPY}}^{(i)} \sum_{\substack{j=1,\ldots,n \\ f_j=e}} \alpha_j^{(i)}. \tag{6.48}$$

This means that there are one or more ways of choosing word e: first, we could choose it directly from the output distribution $\mathbf{o}^{(i)}$, or, for each source word f_j that is equal to e, we could copy word f_j with probability $\mathbf{p}_{\texttt{COPY}}^{(i)}\alpha_j$.

Note that

- $\alpha^{(i)}$ and $\mathbf{p}^{(i)}$ are both vectors of probabilities, not log-probabilities.
- The test $f_j = e$ compares the words *as words*, not as numbers.

6.3 Logic

The last category of semantic representations is that of logical formulas, understood broadly to include not only logics like first-order logic, but languages like SQL or even programming languages.

In these notes, I'd like to focus on a traditional approach to semantics called *Montague grammar*.

6.3.1 Logical forms

We start with a very simple example:

(6.49) a. John sees Mary.

b. $see(\mathit{John}, \mathit{Mary})$.

Entities are represented by constants or variables, and events are represented by predicates.

A variation (called neo-Davidsonian semantics) represents events by variables, too:

(6.50) a. John sees Mary.

b. $\exists e.see(e) \wedge agent(e, \mathit{John}) \wedge theme(e, \mathit{Mary})$.

This is quite similar to AMR. But let's stick with events as predicates.

A key way that logical semantics differs from graph representations like AMR is in handling of quantifiers.

(6.51) a. John sees a girl.

b. $\exists g.girl(g) \wedge see(\mathit{John}, g)$.

(6.52) a. A boy sees Mary.

b. $\exists b.boy(b) \wedge see(b, \mathit{Mary})$.

6.3.2 Compositionality

How do we compute these representations? We want to follow the principle of *compositionality*, that the meaning of any expression is computed from the meaning of its subexpressions. In other words, we want to write a recursive function that processes a syntax tree bottom-up, something like

```
function SEMANTICS(root)
    if root = S and root.children = (NP, VP) then
        s1 ← SEMANTICS(root.children[1])
        s2 ← SEMANTICS(root.children[2])
        build s from s1 and s2
        return s
    else...
    end if
end function
```

So we want to associate with each context-free grammar rule (e.g., S $\rightarrow$ NP VP) a little function that build the semantics of S from the semantics of NP and VP. To do that, it will be convenient to have some new notation for writing little functions.

6.3.3 Lambda calculus

A λ-expression (lambda-expression) is a self-contained way of writing a function. Many programming languages now have them:

λ-calculus	$\lambda x.x \cdot x$
Scheme/Lisp	`(lambda (x) (* x x))`
Python	`lambda x: x * x`
C++	`[](float x) { return x * x; }`

In λ-calculus, the application of a function f to an expression e is simply written as fe. So

$$(\lambda x.x \cdot x)10 \longrightarrow 10 \cdot 10 = 100.$$

Lambda expressions can do a lot of things you might not expect them to be able to do at first; here, I want to mention just one. Traditionally, λ-expressions take exactly one argument. But you can effectively write a function of two argu-

ments as a function that returns another function, like this:

$$f = \lambda x.\lambda y.\sqrt{x^2 + y^2} \tag{6.53}$$

$$f\ 3\ 4 = (\lambda x.\lambda y.\sqrt{x^2 + y^2})\ 3\ 4 \tag{6.54}$$

$$\longrightarrow (\lambda y.\sqrt{3^2 + y^2})\ 4 \tag{6.55}$$

$$\longrightarrow \sqrt{3^2 + 4^2} \tag{6.56}$$

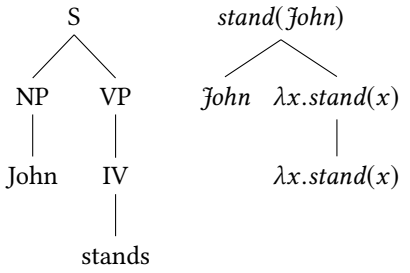
$$= 5. \tag{6.57}$$

This is called *currying* after Haskell Curry, who had nothing to do with it.

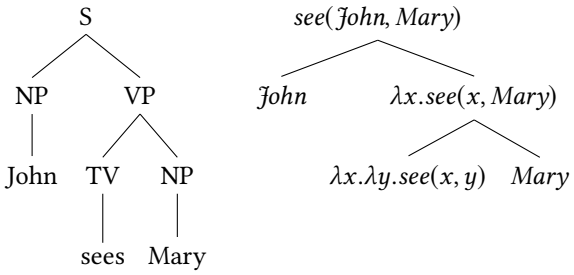
6.3.4 Examples

Here's a very simple CFG, each of whose rules is associated with a function that computes the semantics of the left-hand side (that is, the parent) in terms of the semantics of the right-hand side (that is, the children):

S → NP VP	$\lambda x.\lambda P.Px$
NP → John	John
NP → Mary	Mary
VP → IV	$\lambda P.P$
IV → stands	$\lambda x.\mathit{stand}(x)$

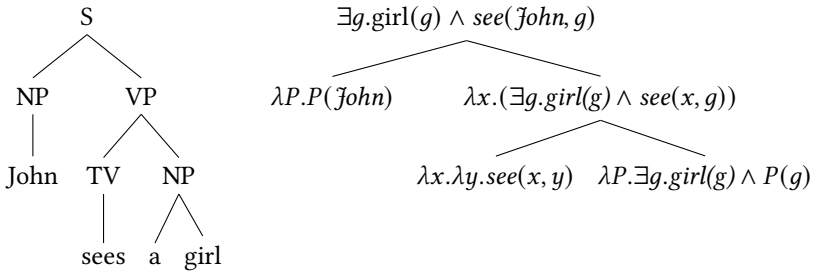


S → NP VP	$\lambda x.\lambda P.Px$
NP → John	John
NP → Mary	Mary
VP → IV	$\lambda P.P$
VP → TV NP	$\lambda P.\lambda y.\lambda x.P(x, y)$
IV → stands	$\lambda x.\mathit{stand}(x)$
TV → sees	$\lambda x.\lambda y.\mathit{see}(x, y)$



When we try to get examples like (6.51–6.52), however, problems arise. The meaning of "a" should be $\exists$, but how do we get this quantifier to appear on the very outside of the formula? The solution is to flip everything around so that the semantics for "a boy" should not just be a formula representing a boy; it should be a function that takes a predicate P about boys and returns the formula $\exists b.boy(b) \wedge P(b)$.

S $\rightarrow$ NP VP	$\lambda f.\lambda P.fP$
NP $\rightarrow$ John	$\lambda P.P(John)$
NP $\rightarrow$ Mary	$\lambda P.P(Mary)$
NP $\rightarrow$ a boy	$\lambda P.(\exists b.boy(b) \wedge Pb)$
NP $\rightarrow$ a girl	$\lambda P.(\exists g.girl(g) \wedge Pg)$
VP $\rightarrow$ TV NP	$\lambda P.\lambda f.\lambda x.f(\lambda y.Pxy)$
TV $\rightarrow$ sees	$\lambda x.\lambda y.see(x, y)$



The computation of VP is particularly complicated, so we write it out step by

step:

$$\underline{(\lambda P.\lambda f.\lambda x.f(\lambda y.Pxy))(\lambda x.\lambda y.see(x,y))}(\lambda P.(\exists g.girl(g) \wedge Pg))$$
$$\longrightarrow (\lambda f.\lambda x.f(\lambda y.\underline{(\lambda x.\lambda y.see(x,y))xy}))(\lambda P.(\exists g.girl(g) \wedge Pg))$$
$$\longrightarrow (\lambda f.\lambda x.f(\lambda y.\underline{(\lambda y.see(x,y))y}))(\lambda P.(\exists g.girl(g) \wedge Pg))$$
$$\longrightarrow \underline{(\lambda f.\lambda x.f(\lambda y.see(x,y)))(\lambda P.(\exists g.girl(g) \wedge Pg))}$$
$$\longrightarrow \lambda x.\underline{(\lambda P.(\exists g.girl(g) \wedge Pg))(\lambda y.see(x,y))}$$
$$\longrightarrow \lambda x.(\exists g.girl(g) \wedge \underline{(\lambda y.see(x,y))g})$$
$$\longrightarrow \lambda x.(\exists g.girl(g) \wedge see(x,g)).$$

Finally, we refine our grammar so that "a" has its own semantics.

S → NP VP	$\lambda f.\lambda P.fP$
NP → John	$\lambda P.P(John)$
NP → Mary	$\lambda P.P(Mary)$
NP → Det N	$\lambda d.\lambda f.df$
Det → a	$\lambda N.\lambda P.(\exists x.Nx \wedge Px)$
Det → every	$\lambda N.\lambda P.(\forall x.Nx \wedge Px)$
N → boy	$\lambda b.boy(b)$
N → girl	$\lambda g.girl(g)$
VP → TV NP	$\lambda P.\lambda f.\lambda x.f(\lambda y.Pxy)$
TV → sees	$\lambda x.\lambda y.see(x,y)$

References

Abend, Omri and Ari Rappoport (2013). "Universal Conceptual Cognitive Annotation (UCCA)". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Sofia, Bulgaria, pp. 228–238. URL: http://www.aclweb.org/anthology/P13-1023.

Asuncion, Arthur et al. (2009). "On Smoothing and Inference for Topic Models". In: *Proc. UAI*, pp. 27–34.

Banarescu, Laura et al. (2013). "Abstract Meaning Representation for Sembanking". In: *Proceedings of the Linguistic Annotation Workshop*. Sofia, Bulgaria, pp. 178–186.

Blei, David M., Andrew Y. Ng, and Michael I. Jordan (2003). "Latent Dirichlet Allocation". In: *J. Machine Learning Research* 3, pp. 993–1022.

Böhmová, Alena et al. (2003). "The Prague Dependency Treebank: A Three-Level Annotation Scenario". In: *Treebanks: Building and Using Parsed Corpora*. Ed. by A. Abeillé. Kluwer, pp. 103–127.

Brown, Peter F. et al. (1991). "Word-sense disambiguation using statistical methods". In: *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL-91)*. Berkeley, CA, pp. 264–270.

Chiang, David et al. (2018). "Weighted DAG automata for semantic graphs". In: *Computational Linguistics* 44.1, pp. 119–186.

Gildea, Daniel and Daniel Jurafsky (2000). "Automatic Labeling of Semantic Roles". In: *Proceedings of the 38th Annual Conference of the Association for Computational Linguistics (ACL-00)*. Hong Kong, pp. 512–520.

Hofmann, Thomas (1999). "Probabilistic Latent Semantic Analysis". In: *Uncertainty in Artificial Intelligence (UAI)*, pp. 289–296.

Hovy, Eduard et al. (2006). "OntoNotes: The 90% Solution". In: *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*. New York City, USA, pp. 57–60. URL: http://www.aclweb.org/anthology/N/N06/N06-2015.

Oepen, Stephan and Jan Tore Lønning (2006). "Discriminant-Based MRS Banking". In: *International Conference on Language Resources and Evaluation (LREC)*. Genoa, pp. 1250–1255.

Palmer, Martha, Daniel Gildea, and Paul Kingsbury (2005). "The Proposition Bank: An Annotated Corpus of Semantic Roles". In: *Computational Linguistics* 31.1, pp. 71–106. DOI: 10.1162/0891201053630264. URL: https://www.aclweb.org/anthology/J05-1004.

Soon, Wee Meng, Hwee Tou Ng, and Daniel Chung Long Lim (2001). "A Machine Learning Approach to Coreference Resolution of Noun Phrases". In: *Computational Linguistics* 27.4, pp. 521–544.