Chapter 8

Generation

The traditional generation task involved mapping some kind of semantic representation to text. As we’ve seen, defining a semantic representation is difficult, prompting the question, “Generation from what?!“ (Wilks). In this chapter, we’re going to sidestep this question by focusing exclusively on generation from vector representations.

8.1 What to generate?

First I’d like to take a step back to think about what generation is supposed to do. Suppose we have some model $\hat{P}(y \mid x)$ that is an estimate of some true distribution $P(y \mid x)$. For now, you can assume that $x$ is some user prompt (like a question) and $y$ is the machine’s response.

What is this true distribution supposed to be? All human beings? A particular (real or imaginary) human being? An omniscient person?

Supposing that we can estimate $\hat{P}(y \mid x)$ perfectly, how do we generate responses from it, given $x$? Do we sample randomly from $\hat{P}(y \mid x)$, or do we choose $\arg \max_y \hat{P}(y \mid x)$, or something else?

To see why these are hard questions, consider how we would want a computer to respond to:

- What is your name?
- Please tell me a joke.
- What is the 123,456,789th digit of $\pi$?
- Please write me a sonnet on the subject of the Forth Bridge.
- Will a baseball dropped on the moon fall down?\(^1\)

I don’t have a good answer to the question of the true distribution; here, we just assume that we are given data drawn from somewhere. As for the question

\(^1\)In a survey of 305 respondents, mostly studying to become elementary school teachers, only 32.8% correctly answered “yes” (Stein et al., “A study of common beliefs and misconceptions in physical science”, *Journal of Elementary Science Education* 20:2, 2008, pages 1–11).
of what \( y \) we should choose, it seems that there are some tasks (machine translation, factual question answering) where we want the argmax, and some (creative writing) where we want to sample, and some tasks where we might want something in between. In the next section, we’ll see some additional technical reasons that contribute to this question.

8.2 Generation from language models

We’ve already seen an example of a model for generation, as part of machine translation. We used an encoder–decoder for machine translation, but here we’re going to consider language models based on decoder-only transformers.

8.2.1 Model

The input to the model is a prefix \( \text{BOS} \cdot w_1 \cdots w_{t-1} \), and the output is a distribution over the next word, \( P(w_t) \).

\[
V_0, \ldots, V_{t-1} = \text{Embedding}(\text{BOS} \cdot w_1 \cdots w_{t-1})
\]

\[
H_0, \ldots, H_{t-1} = \text{Transformer}(V_0, \ldots, V_{t-1})
\]

\[
y(t) = \log(\text{softmax}(\text{LinearLayer}(H_{t-1}))).
\]

The transformer uses masked self-attention. Then \( y(t) \) is the vector of log-probabilities \( \log P(w_t) \).

Now suppose you want to ask the model a question \( x = x_1 \cdots x_m \). The model doesn’t immediately give a distribution over answers \( y \); it only gives distributions over next words. The mathematically most correct way to get a distribution over answers would be

\[
P(y | x) = P(y_1 | \text{BOS} \cdot x_1 \cdots x_m) \cdot P(y_2 | \text{BOS} \cdot x_1 \cdots x_m \cdot y_1) \cdots P(y_n | \text{BOS} \cdot x_1 \cdots x_m \cdot y_1 \cdots y_{n-1}) \cdot P(\text{EOS} | \text{BOS} \cdot x_1 \cdots x_m \cdot y_1 \cdots y_n)
\]

where \( n = |y| \). (Perhaps there could be some separator symbol between \( x_m \) and \( y_1 \).)

8.2.2 Ancestral sampling

Then there are two natural choices for an algorithm to choose an answer. First, we could randomly choose one. This algorithm is called **ancestral sampling**:

- \( w \leftarrow \text{BOS} \cdot x \)
- while \( w \) does not end in EOS
– sample $a$ from $P(a | w)$
– $w \leftarrow w \cdot a$

But, in addition to the problems alluded to above (when there is a correct answer, the user usually wants the correct answer and not a random answer), the randomly-generated answer will be fairly incoherent. The following example is from Holtzman et al. (2020) using GPT-2:

• Prompt: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

• Response: They were cattle called Bolivian Cavalleros; they live in a remote desert uninterrupted by town, and they speak huge, beautiful, paradisiacal Bolivian linguistic thing. They say, 'Lunch, marge.' They don’t tell what the lunch is,” director Professor Chuperas Omwell told Sky News. "They’ve only been talking to scientists, like we’re being interviewed by TV reporters. We don’t even stick around to be interviewed by TV reporters. Maybe that’s how they figured out that they’re cosplaying as the Bolivian Cavalleros.”

Recall that to avoid overfitting, and in particular to avoid assigning a zero probability to any word, we applied smoothing to $n$-gram language models, adding some probability to words that have never been seen in a given context. Neural language models don’t require smoothing, but there are various factors that achieve a similar effect:

• The next-word distribution is a softmax, which cannot output zero
• The model works with vectors whose size is smaller than the vocabulary size
• The training of the model is stopped early, when perplexity on a validation set is minimized

The result is that the model assigns a too-high probability to weird choices. That makes ancestral sampling too willing to actually make a weird choice (e.g., cattle), and the more weird words get added to the context, the weirder future choices will become.

### 8.2.3 Exact search and beam search

How about the alternative, choosing the most probable $y$? First of all, this is in general an NP-hard problem (Higuera and Oncina, 2013). It’s possible to use heuristics to solve it slowly (Stahlberg and Byrne, 2019), but in practice we always use an approximate search, beam search.

The idea of beam search is to generate a string from left-to-right, and at each time step, we keep only the best $k$ strings so far:
Chapter 8. Generation

- \( b^{(i)} \leftarrow \text{BOS} \cdot x \) for \( i = 1, \ldots, k \)
- while any \( b^{(i)} \) does not end in EOS
  - \( B \leftarrow \emptyset \)
  - for \( i = 1, \ldots, k \)
    * if \( b_i \) ends in EOS, \( B \leftarrow B \cup \{b^{(i)}\} \)
    * else, \( B \leftarrow B \cup \{b^{(i)} \cdot a \mid a \in \Sigma\} \)
  - \( b^{(1)}, \ldots, b^{(k)} \leftarrow \) the best \( k \) members of \( B \)
- Output the best of \( \{b^{(i)} \mid i = 1, \ldots, k\} \)

This is what’s typically used in machine translation (with typical \( k \) being quite small, like \( k = 4 \)). But this suffers from problems of its own. The best output may be empty, or highly repetitive, or a copy of \( x \).

Why empty? Since the probability of a string is the product of the probability of the words, and multiplying probabilities always makes them smaller, it’s easy for shorter strings to have higher probability than longer strings. Even if the model knows that \( P(\text{EOS} \mid \text{BOS}) \) is low, it needs to be lower than the probability of the whole correct string, and that’s not easy to do (Murray and Chiang, 2018).

Why a copy? Even if the data has just a few examples of copying in it, the fact that there’s only one way to copy a sentence but many ways to actually respond to it, the probability of a copy may be higher than the probability of any legitimate response (Ott et al., 2018).

As an example of repetition, beam search for the above unicorn prompt gives (Holtzman et al., 2020):

- Response: “The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the Universidad Nacional Autónoma de México (UNAM) and the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México…”

8.2.4 Compromises

In machine translation, the usual fix is to do something simple like divide the log-probability of a translation by its length (Koehn and Knowles, 2017). Additionally, the fact that beam search is approximate turns out to be helpful, so keeping \( k \) low is good. The special case \( k = 1 \) is called greedy search, which you will be asked to implement:

- \( w \leftarrow \text{BOS} \cdot x \)
- while \( w \) does not end in EOS
  - \( a \leftarrow \text{arg max}_a P(a \mid w) \)
  - \( w \leftarrow w \cdot a \)
• output \( w \)

But in other settings, the common practice is to use some kind of compromise between ancestral sampling and greedy search. All of the following affect only the line that chooses \( a \):

• Sample \( a \) from \( P(a \mid w) \), but modify the model \textit{during generation only} so that

\[
o^{(t)} = \log(\text{softmax}((\text{LinearLayer}(\mathbf{H}_{t-1})/T)))
\]

where \( T \) is called the \textit{temperature} (and in fact corresponds to the temperature in a Boltzmann distribution). At \( T = 1 \), this is ancestral sampling, and as \( T \) approaches 0, this becomes greedy search. (Warning: sometimes people incorrectly call \( 1/T \) the temperature.)

• Top-\( k \) sampling: Assume a fixed \( k > 0 \). Let \( a_1, \ldots, a_k \) be the symbols that have the top \( k \) values of \( P(a \mid w) \), then sample \( a \) from the distribution

\[
\text{Top}(a) = \frac{P(a \mid w)}{\sum_{i=1}^{k} P(a_i \mid w)} \quad (a \in \{a_1, \ldots, a_k\}).
\]

• Nucleus or top-\( p \) sampling (Holtzman et al., 2020): Assume a fixed \( p > 0 \). Let \( a_1, \ldots, a_k \) be the smallest subset of \( \Sigma \) such that \( \sum_{i=1}^{k} P(a_i \mid w) \geq p \). (That is, sort the alphabet in decreasing order according to \( P(a \mid w) \), and go down the list until the total probability is \( p \) or more.) Then sample \( a \) from the distribution \( \text{Top}(a) \) as above.

All of these methods work reasonably well in practice, but a truly principled account of how to sample from language models is still a topic of research.