Work in progress: Why does the (sample-estimated) word2vec loss increase at first during stochastic gradient descent?

Note for Stanford CS 224n: Natural Language Processing with Deep Learning

John Hewitt

January 2020

Abstract

We've noticed an unintuitive behavior during word2vec training: the loss goes up for a bunch of gradient steps before finally decreasing and converging. In this note, we explain this behavior by first recognizing that in the first itrations of gradient descent, context words that have been sampled, $c \in C$, are assigned more mass under the conditional probability distribution $p_{\text{w2v}}(c|m)$ of context words given middle words; all other words $\bar{C} = \Sigma \backslash C$ in the vocabulary are consequentially given less probability mass. But at first, C is small (we've only sampled a few words), and so when take new samples of $p(c|m)$, we're *very* likely, at first, to draw context words c that we've just removed probability mass from! So, our sample estimate of the loss goes up. We explain this behavior occur under a simplified model of language.

1 word2vec

In this section we recall the word2vec model of word-context probabilities, changing notation a bit from the class notes to help point out some details.

Let Σ be our vocabulary, a finite set $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_n\}$ of n elements. Let $M \in \Sigma$ be a random variable denoting a *middle* word, and $C \in \Sigma$ be a random variable denoting a *context* word.[1] A word2vec model defines a conditinoal probability distribution $p(C|M)$ of a context word given a middle word.[2] In particular, we have:

$$p_{\text{w2v}}(C = c|M = m) = \frac{\exp(v_c^\top u_m)}{\sum_{\sigma \in \Sigma} \exp(v_c^\top u_\sigma)}, \tag{1}$$

where $v_\sigma, u_\sigma \in \mathbb{R}^d$ are parameters of the model. We'll denote this more concisely as $p_{\text{w2v}}(c|m)$.

Note that our model $p_{\text{w2v}}(c|m)$ is attempting to approximate some true distribution of context words given middle words, $p(c|m)$. We can also reason about true unconditional distributions of middle words $p(m)$ and of context words, $p(c)$.[3]

[1] Note the change from the lecture notes; we use C to denote *context* words, not "center" words.

[2] We'll be sloppy and denote $p(C|M)$ as $p(c|m)$; similarly for $p(C, M)$ as $p(c, m)$, etc.

[3] Note that these might not be the same distributions, since for example words that tend to occur at the very beginning or end of sentences may show up fewer times as *context* words than words that show up in the middle of sentences.

1.1 Word2vec loss

Recall from Assignment 2 the loss of the model on a single middle-context word pair (m, c):

$$\mathcal{L}(m, c, u_{\sigma\in\Sigma}, v_{\sigma\in\Sigma}) = -\log p_{\text{w2v}}(c|m). \tag{2}$$

While we compute this in practice for each word-context pair in a corpus, we can write the expected loss in theory based on true probabilities $p(m)$ and $p(c|m)$.[4] In particular, we want to minimize the expectation of the middle-context word pair loss over the distribution over middle-context word pairs, $p(m, c)$.

$$\mathcal{L}(u_{\sigma\in\Sigma}, v_{\sigma\in\Sigma}) = \mathbb{E}_{(m,c)\sim p(m,c)}\big[-\log p_{\text{w2v}}(c|m)\big]. \tag{3}$$

2 A very simple language distribution $p(m, c)$

When training our word2vec model, we sample middle words sequentially and make a gradient step on each: $m_1, m_2, \ldots$. We'll make the simplifying assumption that sequences of middle words are independently sampled from the distribution $p(m)$.[5] For any two independent samples $m_1, m_2 \sim p(M)$ from our distribution over middle words, we can compute the probability that they're the same word, $m_1 = m_2$:

$$\begin{align} p(m_1 = m_2) &= \sum_{\sigma\in\Sigma} p(\sigma)p(m_1 = \sigma)p(m_2 = \sigma|m_1 = \sigma) \tag{4}\\ &= \sum_{\sigma\in\Sigma} p(\sigma)p(m_1 = \sigma)p(m_2 = \sigma) \tag{5}\\ &= \mathbb{E}_{p(m)}[p(m)^2]. \tag{6}\end{align}$$

Consider if $p(m)$ is the uniform distribution over Σ, that is $p(M = \sigma) = \frac{1}{n}$. Then $p(m_1 = m_2) = \frac{1}{n^2}$ This means that as our vocabulary grows large, we're unlikely to sample the same middle word twice in quick succession.

Now let's do a similar analysis for $p(c|m)$. A premise of training word2vec (and a premise of the distributional hypothesis more broadly) is that $p(c|m) \neq p(c)$, that is, in general the middle word is informative about the distribution over context words. So, let's not assume that $p(C = \sigma|m)$ is uniform over Σ; instead, we'll make a simple but intuitive assumption: we'll assume that the distribution over context words given middle words is uniform over some strict subset of Σ defined with respect to the middle word:

$$p(c|m) = \frac{1}{\Sigma_m} \tag{7}$$

where $\Sigma_m \subset \Sigma$. We'll further make the simplifying assumption that all Σ_m are the same size, namely k. Intuitively, Σ_m is the subset of words *permitted* to occur in the context of m, and any

[4]Throughout this note, we'll use the convention that p refers to a true probability distribution, and p_{w2v} refers to our model's distribution.

[5]In pracice, for a sequence of middle words $m_1, m_2, \ldots$ from a real document, we'll have $p(m_2|m_1) \neq p(m_2)$ because seeing m_1 gives information, e.g., about the topic of the document. However, we could imagine training word2vec by first compiling all (m, c) pairs and then shuffling them, which would make the $p(m_2|m_1) = p(m_2)$ assumption hold.

$\sigma \in \Sigma_m$ is equally likely to occur given m.[6] This allows us to reason about the probability that two context word samples $c_1, c_2 \sim p(c|m)$ conditioned on the same middle word are the same, for a single fixed m:[7]

$$\begin{aligned}
p(c_1 = c_2|M = m) &= \mathbb{E}_{p(c|M=m)}\left[p(C_1 = c|m)p(C_2 = c|m)\right] && (8)\\
&= \mathbb{E}_{p(c|M=m)}\left[p(c|m)^2\right] && (9)\\
&= \frac{1}{|\Sigma_m|^2} && (10)\\
&= \frac{1}{k^2}. && (11)
\end{aligned}$$

Intuitively, this means that as the size of the sets of words permitted in the context of words m grows, the probability that the same word occurs twice in quick succession in the context of m becomes very small.

3 Word2vec at random initialization

Now that we have our simple model of language, we'll look at the behavior of word2vec. To understand the behavior of word2vec early in its training, it's necessary to understand the properties of $p_{\text{w2v}}(c|m)$ when the model is (randomly) initialized. Intuitively, **w2v intitializes each $p_{\mathbf{w2v}}(c|m)$ close to the uniform distribution over Σ.**[8] Let's see why that might be. Assume we have the following initialization scheme:

$$\begin{aligned}
v_\sigma &\sim \mathcal{N}[0,1]^d && (12)\\
u_\sigma &\sim \mathcal{N}[0,1]^d && (13)
\end{aligned}$$

Independent of d, we have that $\mathbb{E}_{p(m,c)}[v_m^\top u_c] = 0$ (why?). Note that this does *not* mean that $\mathbb{E}_{p(m,c)}[\exp(v_m^\top u_c)] = e^0$ (why?), but it does equal some $x \in \mathbb{R}_{++}$ (positive real number).[9]

Now consider the following derivation where we attempt to show that $p_{\text{w2v}}(c|m)$ is uniform at initialization:

$$\begin{aligned}
\mathbb{E}_{p(m,c)} \frac{\exp(v_m^\top u_c)}{\sum_{\sigma\in\Sigma}\exp(v_m^\top u_\sigma)} &= \frac{\mathbb{E}_{p(m,c)}\exp(v_m^\top u_c)}{\mathbb{E}_{p(m,c)}\sum_{\sigma\in\Sigma}\exp(v_m^\top u_\sigma)} && (14)\\
&= \frac{\mathbb{E}_{p(m,c)}\exp(v_m^\top u_c)}{\sum_{\sigma\in\Sigma}\mathbb{E}_{p(m,c)}\exp(v_m^\top u_\sigma)} && (15)\\
&= \frac{x}{\sum_{\sigma\in\Sigma} x} && (16)\\
&= \frac{1}{|\Sigma|}, && (17)
\end{aligned}$$

[6]Note that for $m \neq m'$, we may have overlap between Σ_m and $\Sigma_{m'}$

[7]Assuming that $p(c_2|c_1, m) = p(c_2|m)$, which isn't strictly true in practice, but is true in our model, and could be done by shuffling all (m, c) pairs.

[8]Note that this behavior is not due to the assumptions we made about $p(c|m)$ in the last section; this initialization behavior (approximately) holds independent of the true $p(c|m)$ distributions.

[9]I think x depends on d since the variance of $u_m^\top u_c$ depends on d, and exp isn't a linear function.

which is the uniform distribution, as required. However, we snuck in a new assumption in this derivation. In equating the left-hand side with the right-hand side in Equation 14, we assumed that random variable representing the numerator is independent of the random variable representing the denominator. This is, in fact, false, given that the numerator shows up as a term in the summation in the denominator. But if $|\Sigma|$ is large enough, the two are approximately independent (since there are so many terms of similar magnitude in the sum.)

4 Walking through the first gradient steps of word2vec

Now that we understand the behavior of word2vec *before* we've started to train on samples from $p(m, c)$, we can now attempt to understand the behavior in the first few gradient steps.

In the first gradient step of word2vec, we observe a handful of (m, c) samples, say 4:

$$(m, c_1), (m, c_2), (m, c_3), (m, c_4), \tag{18}$$

where m is a single sample from $p(m)$ and all c_i are independently sampled according to $p(c|m)$. Let $p^{(0)}_{\text{w2v}}$ be the word2vec model at random initialization, $p^{(1)}_{\text{w2v}}$ after one gradient step, and $p^{(j)}_{\text{w2v}}$ after j gradient steps.

In expectation, the probability mass assigned to each sample by the randomly initialized word2vec model is (from our previous section), $\mathbb{E}[p(c|m)] = \frac{1}{|\Sigma|}$. The expected loss per term is thus $-\log \frac{1}{|\Sigma|}$ (which is large if $|\Sigma|$ is large!) Abstracting away details of the gradients of the loss w.r.t. the model parameters, we know for sure that the gradient update makes $p^{(1)}_{\text{w2v}}(c_i|m) > p^{(0)}_{\text{w2v}}(c_i|m)$; that is, it increases the probability of the observed samples under the model.

In a discrete probability distribution, the sum of probabilities over the support of the distribution must be 1. So, we come to the following consequence when we fix m:

$$p^{(1)}_{\text{w2v}}(c_i|m) > p^{(0)}_{\text{w2v}}(c_i|m) \implies \exists \sigma : p^{(1)}_{\text{w2v}}(\sigma|m) < p^{(0)}_{\text{w2v}}(\sigma|m), \tag{19}$$

that is, we had to take the probability mass away from at least one symbol, σ. We'll go one step further. Note how *all* u_σ vectors show up in the denominator of $p^{(0)}_{\text{w2v}}(c_i|m)$. Intuitively, for each u_σ that showed up in the denominator but not the numerator, the gradient of the loss of this sample with respect to the u_σ will be telling the model to make $p^{(1)}_{\text{w2v}}(\sigma|m)$ less likely.

Consider that the expected sum of probabilities assigned to one of the four examples c_1, c_2, c_3, c_4 at initialization is $\sum_{i=1}^{4} \mathbb{E}[p^{(0)}_{\text{w2v}}(c_i|m)] = \frac{4}{|\Sigma|}$, and after the first gradient step, we added some mass to each, let's say δ in expectation, so the total sum for $p^{(1)}_{\text{w2v}}$ is $\frac{4}{|\Sigma|} + 4\delta$. We make the simplifying assumption that at the beginning of training, the model subtracts mass uniformly from the σ_i that weren't one of the four sampled. So after the first gradient step we have:

$$\mathbb{E}[p^{(1)}_{\text{w2v}}(\sigma|m, \sigma \notin \{c_1, c_2, c_3, c_4\})] = \frac{1}{|\Sigma|} - \frac{4\delta}{|\Sigma| - 4} \tag{20}$$

Does this make sense? I'd argue yes. Recall that or our middle word m, there's some set of words $\Sigma_m \subset \Sigma$ that occur uniformly as context words for m; the rest of Σ never occurs as a context word for m. Given our first sample, we know that c_1, c_2, c_3, c_4 are in Σ_m, and every other word becomes less likely to be in that set.

> Under our simplified model of language, if we were to compute the true expected loss over $p(m, c)$, and we'd used a small enough learning rate, we would find that $p^{(1)}_{\text{w2v}}$ is a better estimate of $p(c|m)$ than is $p^{(0)}_{\text{w2v}}$.

Consider the next gradient step at which we see the same middle word, m.[10] We take samples $(m, c_5), (m, c_6), (m, c_7), (m, c_8)$. What is the loss of $p^{(1)}_{\text{w2v}}$ on these samples? There are two cases for each sample. One is that $c_i \in \{c_1, c_2, c_3, c_4\}$, that is, it's one of the words that we saw in the first gradient step. The other is that c_i is a new word, not seen during the first gradient step. So we can compute the expected probability mass assigned to c_i by these cases:

$$\mathbb{E}_{p(m,c)}\Big[p^{(1)}_{\text{w2v}}(c_i|m, i \in \{5,6,7,8\})\Big] = \begin{cases} \frac{1}{|\Sigma|} + \delta & c_i \in \{c_1, c_2, c_3, c_4\} \\ \frac{1}{|\Sigma|} - \frac{4\delta}{|\Sigma|-4} & \text{otherwise} \end{cases} \tag{21}$$

In the second gradient step, if we see words c_i we've seen before, then $p^{(1)}_{\text{w2v}}$ performs better (assigns higher probability) to the words than $p^{(0)}_{\text{w2v}}$ did (Case 1 of Equation 21) – just what we asked it to. But if we see *new* words, $p^{(1)}_{\text{w2v}}$ actually does **worse** than $p^{(0)}_{\text{w2v}}$ (case 2). Now, what are the probabilities of these cases? Let's start with the probability that any one word in the new set (c_5, c_6, c_7, c_8) was a member of the old set (c_1, c_2, c_3, c_4):

$$p(c_i \in \{c_1, c_2, c_3, c_4\}|i \in \{5,6,7,8\}) = \frac{4}{|\Sigma_m|}. \tag{22}$$

Consider a modest vocabulary size of $|\Sigma| = 20{,}000$, and let's say Σ_m is $\frac{1}{4}$ the size, so $5{,}000$ – this means the probability each one new word showed up in our first sample is $\frac{1}{5000}$. The probability that *none* of the new words were seen before is $\frac{4999}{5000}^4 \approx .999$. So, with high probability, *all* our new samples are drawn from words that $p^{(1)}_{\text{w2v}}$ assigns less probability mass to than does $p^{(0)}_{\text{w2v}}$. So, we have:

$$\begin{aligned}
\mathbb{E}[\mathcal{L}(p^{(1)}_{\text{w2v}}, c_5, c_6, c_7, c_8)] &= \sum_{i=5}^{8} -\log\Bigg(\quad p(c_i \notin \{c_1, c_2, c_3, c_4\})p^{(1)}_{\text{w2v}}(c_i|m, c_i \notin \{c_1, c_2, c_3, c_4\}) && (23)\\
&+ p(c_i \in \{c_1, c_2, c_3, c_4\})p^{(1)}_{\text{w2v}}(c_i|m, c_i \in \{c_1, c_2, c_3, c_4\})\Bigg) && (24)\\
&\approx \sum_{i=5}^{8} -\log 1 * p^{(1)}_{\text{w2v}}(c_i|m, c_i \notin \{c_1, c_2, c_3, c_4\}) && (25)\\
&= \sum_{i=5}^{8} -\log(\frac{1}{|\Sigma|} - \frac{4\delta}{|\Sigma|-4}) && (26)\\
&> \sum_{i=5}^{8} -\log(\frac{1}{|\Sigma|}) && (27)\\
&= \mathbb{E}[\mathcal{L}(p^{(0)}_{\text{w2v}}, c_5, c_6, c_7, c_8)]. && (28)
\end{aligned}$$

[10]Perhaps the biggest hole in this explanation is that the next center word m isn't likely to be the first, so why do we already see an increase in loss?

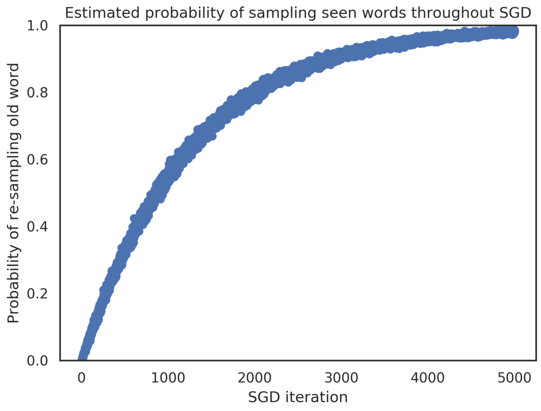

Figure 1: Estimated probability of seeing a word in $C_m^{(i)}$ if $|\Sigma_m| = 5,000$, and $p(\sigma|m) = \frac{1}{|\Sigma_m|}$, as a function of i.

Where the approximate equality in (25) is due to the fact that, with high probability, our samples $\{c_5, ..., c_8\}$ are not the samples from $\{c_1, ..., c_4\}$. Thus, with some handwaving and assumptions about our data distribution, we've shown that we actually *should* see higher (sample-estimated) loss after the first gradient step, $p_{\text{w2v}}^{(1)}$ than when we initialized, $p_{\text{w2v}}^{(0)}$!

4.1 After the first gradient step

After the first gradient step, everything gets messier, since our assumptions about how word2vec behaves at random initialization cease to hold. But we'll still attempt to reason about the behavior intuitively.

Recall our sequence of word2vec distributions $\{p_{\text{w2v}}^{(i)} \mid i = 0, 1, \dots\}$, where i indicates the number of gradient steps we've taken. We now define $C_m^{(i)}$ to be the set of context words seen around middle word m in any sample up through gradient step i. Intuitively, this is the set of words that we've *ever* assigned more mass to under the model. All other words, $\bar{C}_m^{(i)} = \Sigma \backslash C_m^{(i)}$, have only ever (explicitly) had mass removed from them.[11]

When we estimate our loss after gradient step i, we're taking samples from $p(c|m)$, meaning $c \in \Sigma_m$, and with high probability until we've gone through many iterations, $c \in \bar{C}_m^{(i)}$. Consider if we were to sample c uniformly from $5,000$ possible context words in Σ_m as our model states. The probability at gradient step i that each c_i word came from the set of words seen already, $C_m^{(i)}$, grows slowly, as in Figure 1. Words in natural language aren't uniformly distributed, but the principle is the same as that shown in Figure 1: for the first (many!) iterations i, it's much more likely that the words you're using to compute your estimate of the loss come from $\bar{C}_m^{(i)}$, the words you haven't seen yet. So for a while, your sample estimate of the loss goes up!

[11]This actually makes an additional assumption; because the dimensionality of our vector space is lower than the size of our vocabulary, increasing the probability of some context word c_i can increase the probability of other context words, even if they weren't seen. But approximately, we can think of $\bar{C}_m^{(i)}$ as the set of words we've removed mass from.

5 Next steps

This is just a start to understanding this phenomenon in practice! For example, language is *not* uniformly distributed. How would these claims hold if we assumed conditional Zipfian distributions? In particular, this would mean that our estimate of many words' probabilities would be vastly overestimated at start (since we start at a uniform distribution.) Does this actually explain the phenomenon in total? This simplified theoretical analysis predicts that if we were to estimate our loss on the whole training dataset instead of with a sample, we really would see the loss go down immediately. But SGD esimtates of the gradient are noisy! Maybe this isn't the case in practice! We could see whether this prediction holds in practice, and if not, make fewer assumptions in our theory and see if we can explain more real behavior. Also, as noted in a footnote, our analysis doesn't consider how gradient steps on different middle words m affect each other's $p_{\text{w2v}}(c|m)$.

Further, can we use these intuitions to help word2vec **train faster in practice**? One insight from this theory is that we spend a lot of time working with word frequency problems, since we start at effectively the uniform distribution. Could a word frequency-aware initialization help word2vec converge much faster? This would be very impactful! If you've gotten here and are interested, consider talking to John about final project ideas!