Tokens and Word Vectors: Advanced Topics

Lecture Notes Consolidated with Claude AI — CS6140 Machine Learning

---

*Optional. Assumes the **TOKENS** note (BPE, skip-gram, negative sampling). Nothing here is needed for the required HW7 problems; §4 backs optional Problem 12.*

1 The Full-Softmax Gradient: Observed Minus Expected

For one (center c, context o) pair the full skip-gram model has

$$\log P(o \mid c) = u_o^\top v_c - \log \sum_{w \in \mathcal{V}} \exp(u_w^\top v_c).$$

Differentiating with respect to the center vector,

$$\begin{aligned} \frac{\partial \log P(o \mid c)}{\partial v_c} &= u_o - \sum_{w \in \mathcal{V}} P(w \mid c)\, u_w \\ &= \underbrace{u_o}_{\text{observed}} - \underbrace{\mathbb{E}_{w \sim P(\cdot \mid c)}[u_w]}_{\text{expected}}. \end{aligned}$$

In words: move v_c toward the context vector that actually occurred and away from the average context vector the model currently predicts. This is the same "target minus prediction" shape as the softmax/cross-entropy gradient of HW2 and HW5 (there: $y - p$ times the input); here the "input" and the "weights" are both learned vectors.

Worked example: a 3-word vocabulary $v_c = (1, 0)$, context vectors $u_1 = (2, 0)$, $u_2 = (0, 1)$, $u_3 = (-1, 0)$, observed context $o = 1$. Scores $(2, 0, -1)$, probabilities $(0.8438, 0.1142, 0.0420)$, expected context vector $(1.6456, 0.1142)$, gradient $u_1 - \mathbb{E}[u] = (0.3544, -0.1142)$: a small push toward u_1, since the model already gives it 84%.

The expected vector needs every u_w: one pass over the whole vocabulary per pair, $O(|\mathcal{V}|\, d)$ work. Negative sampling does not estimate this gradient; it replaces the whole full-softmax objective by a different, binary one (real neighbour vs K random noise words, a sigmoid per word), which costs $O(K\, d)$ and never computes $P(o \mid c)$ at all. A useful way to see what it learns: Levy and Goldberg showed that SGNS implicitly factorizes a matrix of word–context pointwise mutual information values, shifted by $\log K$ — a bridge to the count-based methods of §4.

2 CBOW: the Other word2vec

Skip-gram predicts each context word from the center. The **continuous bag of words** (CBOW) model reverses the conditional probability (the board's "REVERSE COND PROB", Lecture 8/13, pp. 12–14): predict the center from the *average* of its context vectors,

$$P(c \mid o_1, \ldots, o_{2m}) \;=\; \frac{\exp\left(u_c^\top \bar{v}\right)}{\sum_w \exp\left(u_w^\top \bar{v}\right)}, \qquad \bar{v} = \frac{1}{2m} \sum_{j=1}^{2m} v_{o_j}.$$

There is no independence product over the window ("evaluate the score on all context at once"). CBOW is faster (one prediction per window instead of $2m$) and smooths over the context; skip-gram makes one update per (center, context) pair, which helps rare words. The usual folklore — CBOW for speed, skip-gram for rare words and small corpora — comes from the original word2vec experiments; both are trained with negative sampling in practice.

3 Hierarchical Softmax: the Other Way Around the Normalizer

Instead of sampling noise words, put the vocabulary at the leaves of a binary tree and write $P(o \mid c)$ as the product of left/right decisions on the path from the root to o, each a sigmoid of $u_{\text{node}}^{\top} v_c$. The probabilities of all leaves sum to 1 automatically, and one pair costs $\log_2 |\mathcal{V}|$ sigmoids: about 13 for $|\mathcal{V}| = 5{,}274$, 17 for $|\mathcal{V}| = 100{,}000$. word2vec builds the tree by Huffman coding, so frequent words have short paths. Negative sampling became the default because it is simpler and works at least as well for the vectors; hierarchical softmax keeps a proper probability model.

4 GloVe: Vectors From Counts

word2vec reads the text window by window. **GloVe** (Pennington, Socher and Manning) first counts once: x_{ij} = how often word j appears in a window around word i in the whole corpus ("MULTICOUNT" on the board, Lecture 8/13, pp. 18–20). It then fits vectors so that their dot product predicts the log count, by weighted least squares:

$$J \;=\; \sum_{i,j:\, x_{ij}>0} h(x_{ij}) \left(v_i^{\top} u_j + b_i + c_j - \log x_{ij} \right)^2, \qquad h(x) = \min\!\left(1, (x/x_{\max})^{3/4}\right), \;\; x_{\max} = 100.$$

In words: a regression of log(co-occurrence) on a dot product plus two biases. The weight h is the board's "monotonic, controls scale and sensitivity": rare pairs, whose counts are noisy, get little weight ($h(1) = 0.03$, $h(10) = 0.18$), and very frequent pairs are capped at 1 so that `the` does not dominate (Figure 1).

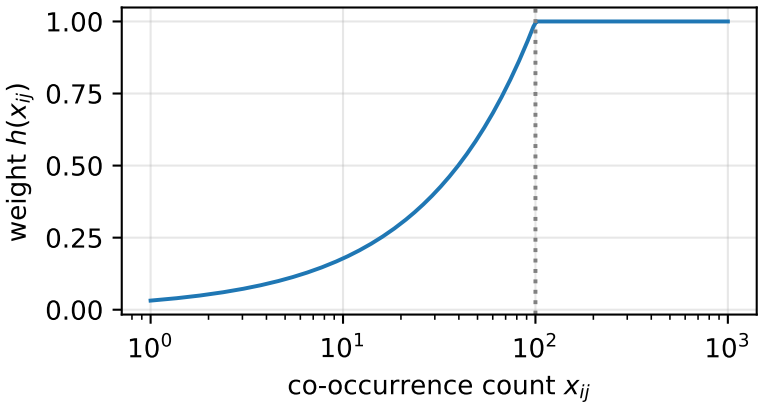


Figure 1: GloVe's weight $h(x)$: zero weight for pairs that never co-occur, a slow rise, capped at $x_{\max} = 100$.

Why it matters in practice: GloVe vectors were trained once on very large corpora and are downloaded in several sizes (the board: "general word vectors, pretrained on a large corpus; tune for

your specific corpus"). The course data folder has `glove.6B` (6 billion tokens, 400k words; 50, 100, 200 and 300 dimensions). Optional Problem 12 compares them with the TinyStories vectors: a large general corpus knows many more words and their adult meanings; a small specific corpus knows the words of its domain and how *it* uses them.

5 fastText: a Word Is the Sum of Its Pieces

fastText (Bojanowski et al.) keeps skip-gram with negative sampling but changes what a center vector is: the word is wrapped in boundary marks and split into all character n-grams, $3 \le n \le 6$; for $n = 3$, `<where>` gives `<wh, whe, her, ere, re>`, plus the whole word `<where>` as its own unit. The center vector is the **sum** of the vectors of its n-grams (the board: "embedding $= \sum$ embed(grams)", Lecture 8/13, p. 16).

- *friend*, *friends*, *friendship* share most of their n-grams, so they share most of their parameters, the board's "stemming / lexicographics" for free.
- A word never seen in training still gets a vector, from its n-grams — the same promise BPE makes for tokens.
- The cost: many more vectors (hashed into a fixed number of buckets).

fastText and BPE both answer "words are too coarse". fastText uses *all* n-grams of a word inside one vector; BPE *chooses* a few frequent pieces and makes each one a separate token.

6 Other Subword Tokenizers

WordPiece (BERT; Schuster and Nakajima). The same merge loop as BPE, with a different choice: merge the pair with the largest

$$\text{score}(a, b) \;=\; \frac{\text{count}(ab)}{\text{count}(a)\,\text{count}(b)},$$

i.e. the merge that most increases the likelihood of the training text under a unigram model of the pieces. A pair wins when its parts rarely appear *apart* (`q`+`u` in English), not merely when it is frequent. BERT marks word-internal pieces with `##` (`play ##ing`).

Unigram language model (Kudo; T5, many SentencePiece models). Work in the opposite direction: start from a large set of candidate pieces, fit a unigram probability to each, and repeatedly remove the pieces whose removal hurts the likelihood of the corpus least, until the target size is reached. A word can then be split in several ways; the tokenizer picks the most probable split (and can sample other splits during training, "subword regularization").

Byte-level BPE (GPT-2 and most GPT-style models). BPE whose alphabet is the 256 byte values of the UTF-8 text, so the vocabulary always covers every input. GPT-2 maps each byte to a printable character for its merge table; that is why a leading space shows up as `Ġ` and an accented *ó* as the two bytes `Ã³`. Its 50,257 entries are the 256 bytes, 50,000 merges and one end-of-text token.

SentencePiece (Kudo and Richardson). A library, not an algorithm (it implements BPE and Unigram). Its key idea: no pre-tokenization. It reads raw text, spaces included, writing the space as the visible symbol _. Nothing assumes that words are separated by spaces, which matters for Chinese, Japanese or Thai, and decoding is exact (join the pieces, turn _ back into spaces).

7 What a Tokenizer Does to Languages and Numbers

Languages pay different prices. A tokenizer is learned from its training text, and GPT-2's was mostly English web text. On 5,000 Tatoeba sentence pairs, GPT-2 uses **1.22 pieces per English word but 2.18 per Spanish word** (build script): *El gato se sentó en la alfombra* becomes 13 pieces (`El Ġg ato Ġse Ġsent Ã³ Ġen Ġla Ġal f omb ra .`) where *The cat sat on the mat* is 7. The same sentence costs a Spanish speaker almost twice the context window and compute. Modern tokenizers use larger, more multilingual vocabularies partly for this reason (the tokenizer table of the TOKENS note, §3).

Numbers are split arbitrarily. `12345 + 678 = 13023` becomes `123 45 Ġ+ Ġ6 78 Ġ= Ġ130 23`: the pieces follow frequencies in the training text, not place value, so "carry the one" has to be learned across inconsistent splits. Some later tokenizers split every digit, or groups of three digits from the right, to make arithmetic regular.

Spaces and case create different tokens. `Ġcat` (after a space) and `cat` (at the start of a text) are different IDs; so are `Cat` and `cat`. The model must learn that they are related.

Vocabulary size is a trade-off. A bigger vocabulary means shorter sequences (less compute per sentence, more text in the context window) but a bigger embedding table, $|\mathcal{V}| \times d$ numbers, and rarer pieces with fewer training examples each:

Model	$\lvert\mathcal{V}\rvert$	d	**embedding table**
HW6/HW7 word models	5,274	256	1.4M
BERT-base	30,522	768	23.4M
GPT-2 small	50,257	768	38.6M
LLaMA 2 7B	32,000	4096	131.1M
LLaMA 3 8B	128,256	4096	525.3M

Table 1: Embedding-table size = $|\mathcal{V}| \cdot d$ numbers (one table; untied models have a second one of the same size for the output layer).

8 Static vs Contextual Vectors; Analogies

word2vec, GloVe and fastText give one vector per word. Two consequences:

- **Senses are averaged.** *bank* (river) and *bank* (money) share one vector, a mix weighted by how often each sense occurs. A Transformer's hidden states are *contextual*: the same token gets a different vector in each sentence. Static vectors survive as the *first layer* of every such model.

- **Analogies, with caveats.** The famous $v_{\text{king}} - v_{\text{man}} + v_{\text{woman}} \approx v_{\text{queen}}$ holds for some relations (capitals, plurals, gender) in vectors trained on large corpora, but the usual evaluation excludes the three input words from the answers (otherwise *king* itself is often nearest), and many relations fail. Treat analogies as an illustration that directions in the space can carry meaning, not as a property every embedding has. The vectors also absorb the associations of their training text, including social biases.

9 Further Reading

- d2l, Chapter 15 (15.1–15.7): word2vec, approximate training, GloVe, subword embedding, similarity and analogy.
- T. Mikolov, K. Chen, G. Corrado, J. Dean, *Efficient Estimation of Word Representations in Vector Space*, ICLR Workshop 2013 (arXiv:1301.3781). Skip-gram and CBOW.
- T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, *Distributed Representations of Words and Phrases and their Compositionality*, NeurIPS 2013 (arXiv:1310.4546). Negative sampling, subsampling.
- J. Pennington, R. Socher, C. D. Manning, *GloVe: Global Vectors for Word Representation*, EMNLP 2014.
- O. Levy, Y. Goldberg, *Neural Word Embedding as Implicit Matrix Factorization*, NeurIPS 2014.
- P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, *Enriching Word Vectors with Subword Information*, TACL 2017 (arXiv:1607.04606). fastText.
- R. Sennrich, B. Haddow, A. Birch, *Neural Machine Translation of Rare Words with Subword Units*, ACL 2016 (arXiv:1508.07909). BPE for neural models.
- M. Schuster, K. Nakajima, *Japanese and Korean Voice Search*, ICASSP 2012. WordPiece.
- T. Kudo, *Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates*, ACL 2018 (arXiv:1804.10959). The Unigram tokenizer.
- T. Kudo, J. Richardson, *SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing*, EMNLP 2018 demo (arXiv:1808.06226).
- A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *Language Models are Unsupervised Multitask Learners*, OpenAI 2019. GPT-2 and byte-level BPE.
- C. McCormick, *Word2Vec Tutorial* (course reading): the "fake task" view of the TOKENS note's §6.