**1. Run a few experiments with different embedding dimensions, batch sizes, and number of epochs run. Try with and without the pre-trained GloVe embeddings. Across your experiments, did the pre-trained embeddings work better, or was it better to train your own embedding layer for this task? Did you notice any trends with certain parameters? What happens if you freeze the GloVe embeddings during training time?**  
**Answer**:

In my experiments, pre-trained GloVe embeddings generally helped the model converge faster, but their effectiveness depended on whether they were fine-tuned. When frozen, they preserved general semantic relationships but sometimes failed to adapt to domain-specific sentiment classification. Training embeddings from scratch often took longer but, given enough data, sometimes performed better as they captured task-specific word associations. Higher embedding dimensions (e.g., 200) improved expressivity but increased computational cost, while smaller dimensions (e.g., 50) led to faster training but sometimes missed subtle semantic differences.

Batch size and number of epochs also influenced performance. Smaller batches (16-32) introduced noise in updates but often helped generalization, whereas larger batches (64+) led to faster but potentially overfitting models. With pre-trained embeddings, fewer epochs (5-10) were sufficient, while training embeddings from scratch required more epochs (20+) to reach similar accuracy. Freezing GloVe embeddings sped up training but limited adaptation, whereas fine-tuning them improved results at the cost of longer training time. Ultimately, the best approach depended on the dataset size and computational constraints.

**2. Compare your own embeddings, trained on the Rotten Tomatoes task, against the more general-purpose GloVe embeddings. Choose some words you think might be interesting: for example, since this is a dataset of movie reviews, the words “good” and “bad” might be interesting, since what is considered “good” in the movie domain may differ significantly from the word “good” more generally. Find the nearest neighbors for the words for both your own embeddings, and the GloVe embeddings. What do you notice about the differences in the vector space between your own embeddings and the pre-trained embeddings?  
Answer**: When comparing my learned embeddings from the Rotten Tomatoes dataset to the general-purpose GloVe embeddings, I noticed significant differences in how words like "good" and "bad" were contextualized. My trained embeddings captured sentiment-specific associations, where "good" was closely related to words like "entertaining," "well-acted," and "engaging," while "bad" was near terms like "boring," "predictable," and "disappointing." In contrast, the GloVe embeddings, being trained on a broad corpus, associated "good" with more general terms like "great," "nice," and "better," and "bad" with "worse," "awful," and "poor." This suggests that my embeddings better reflected the language used in movie reviews, focusing on qualitative descriptors specific to film criticism.

The differences in vector space highlight the importance of domain adaptation. While GloVe embeddings provided a solid starting point, they lacked the nuanced sentiment relationships found in movie reviews. My trained embeddings captured these subtleties better, but they might not generalize well to other contexts outside film analysis. Additionally, freezing GloVe embeddings retained their broad semantic structure, whereas fine-tuning allowed them to shift closer to the sentiment-specific relationships observed in my trained model. This experiment underscores the trade-off between generalization and domain specificity when choosing between pre-trained and task-specific embeddings.

**3. See if you can make any interesting or illuminating visualizations of your own embeddings with the two plot\_embeddings functions in the a3-explore-scaffolding.py file. If none of the visualizations make any sense to you, explain why you’re surprised by what you see, and what you would’ve expected instead. Speculate whether it’s because of the idiosyncrasy of the data, or if it’s because the embeddings haven’t been trained to optimally represent language, or something else entirely. Then, compare and contrast with visualizations of the GloVe embeddings.**

Answer: When visualizing my own trained embeddings using PCA and t-SNE, I observed that while some words clustered together based on semantic similarity, the overall structure was not as clearly defined as expected. The 3D PCA visualization showed some degree of separation between different groups of words, but there were also cases where unrelated words appeared close together (doesn't and ripe), which was surprising. With t-SNE, the embeddings often appeared scattered without distinct clusters, likely due to the model's limited training data and the specialized nature of the Rotten Tomatoes dataset. I initially expected clearer sentiment-based groupings, with words related to positive or negative reviews forming more distinct regions, but the results suggested that the embeddings had not fully captured these relationships.

In contrast, the GloVe embeddings exhibited a more structured and intuitive (not as much as I thought they would be) distribution in both the PCA and t-SNE visualizations. Words with similar meanings were more consistently clustered (even though a few words I could see were nto clustered the way I expected but only a few), and sentiment-related words appeared in more well-defined regions, reinforcing the idea that these pre-trained embeddings capture broader semantic relationships learned from large-scale corpora. The difference in visualization suggests that my trained embeddings might be overfitting to the Rotten Tomatoes dataset, making them useful for domain-specific tasks but less generalizable. This highlights the trade-off between training embeddings from scratch versus using pre-trained ones—while domain adaptation is possible, achieving well-structured representations requires significantly more data and training iterations.