Faiza Abdullah

Lab 05

ITAI 2376 Deep Learning in Artificial Intelligence

Professor: Patricia Mcmanus

From Code to Wisdom: A Reflective Journey Through NLP in Deep Learning

INTRODUCTION

This reflective journal chronicles my exploration of NLP through four hands-on labs in Module 2 of the AWS course. Spanning foundational text preprocessing, vectorization techniques, semantic embeddings, and advanced sequence modeling, these labs offered a progressive dive into transforming raw text into actionable insights using machine learning. As I navigated code, outputs, and concepts, I transitioned from a novice to a more confident analyst ready to tackle complex NLP challenges.

REFLECTIVE JOURNAL: MODULE 2, LAB 1 - PROCESSING TEXT

INTRODUCTION

Through hands-on exercises, during this lab I explored how to preprocess and analyze text data through NLP techniques, including word clouds, stemming, lemmatization, part-of-speech (POS) tagging, and named entity recognition (NER) using spaCy and other Python libraries.

Learning Insights

- Word Clouds: Visualizing word frequency (e.g., wordcloud.generate(preProcessText(text))) highlighted key terms like “NLP” and “language” from the AWS NLP page text.
- POS Tagging: Assigning grammatical roles to words (e.g., Out[9]: “Natural” as ADJ, “language” as NOUN) showed how context informs meaning.

- Stemming and Lemmatization: Reducing words to roots (e.g., Out[15]: "cleaned" to "clean" vs. Out[17]: "cleaned" to "clean") revealed trade-offs in speed vs. accuracy.
- NER: Identifying entities like "Amazon" as ORG or "2040" as DATE (Out[29]) demonstrated extracting structured insights from unstructured text.

These techniques are building blocks for feature extraction, enabling text to be transformed into a format machines can process, much like normalizing numerical data. A real-life example is social media sentiment analysis: stemming "running" and "ran" to "run" groups related sentiments, while NER identifies brands or people for targeted insights. This lab bridges to Lab 2's BoW by showing how cleaned text feeds into vectorization, forming a cohesive NLP workflow.

Most Impactful Learning Moments: The NER visualization (Out[30]) was a standout moment—seeing "Amazon ORG" and "The Climate Pledge WORK_OF_ART" highlighted in context made entity extraction tangible. Another impactful insight was comparing stemming (Out[15]: "messag" for "message") and lemmatization (Out[17]: "message" intact)—lemmatization's precision impressed me. These moments clarified how preprocessing choices shape downstream model performance.

Challenges and Struggles

Installing spaCy and its model was a technical hurdle; the lengthy dependency list felt overwhelming. Conceptually, I struggled to grasp why stemming produced errors like "messag" until the lab explained its rule-based nature (PAGE14). Differentiating POS tags (e.g., ADV vs. ADP) also confused me initially due to overlapping definitions.

Problem Solving Strategies: I ran the code multiple times with different inputs (e.g., "running" vs. "jumping") to see patterns in suffix removal, confirming its limitations. I tackled POS confusion by reading a bit online. I adopted a "divide and conquer" approach—isolating one technique (e.g., NER) and mastering it before moving to the next.

Personal Growth

My understanding evolved from seeing text as chaotic to recognizing it as structured data after preprocessing. I now appreciate NLP as a blend of linguistics and math, not just coding. I was surprised by stemming's imperfections—expecting "message" but getting "messag" showed me that rule-based systems sacrifice accuracy for speed. The visual power of word clouds also caught me off guard; I didn't expect such a simple tool to reveal insights so quickly, like spotting "customer" as a key theme.

These skills will support my academic goal of analyzing social media trends—NER could tag influencers, while word clouds highlight trending topics. Professionally, I could preprocess customer reviews for a business, using POS tagging to filter adjectives for sentiment analysis, enhancing decision-making.

Critical Reflection

I would experiment with larger, messier texts (e.g., tweets with emojis) to test preprocessing robustness, rather than sticking to the lab's clean examples. I'd also explore spaCy's dependency parsing (displacy.render(doc, style="dep")) more deeply to understand sentence structure, which the lab only hinted at.

Research Time: How do these techniques handle multilingual text or typos? Could combining stemming and NER improve entity detection in noisy data? I'm eager to explore tokenization further, as it's briefly mentioned but critical for BoW in Lab 2. These gaps point to practical challenges in real-world NLP.

CONCLUSION

Lab 1 clarified text processing, like Out[29] (NER tags) and Out[15] (stemming results) grounded abstract concepts in practice. Real-life parallels—sorting customer feedback or tagging news entities—made it relatable.

REFLECTIVE JOURNAL: MODULE 2, LAB 2 - USING THE BOW METHOD

INTRODUCTION

This lab focused on the Bag-of-Words (BoW) method to convert text data into numerical representations, by introducing techniques like binary classification, word counts, term frequency (TF), and term frequency-inverse document frequency (TF-IDF) using Python's sklearn library. Through hands-on exercises, I explored how text can be transformed into a format suitable for predictive models, bridging theoretical concepts with practical implementation.

Learning Insights

The BoW method represents text as numerical vectors based on word occurrences, enabling machine learning algorithms to process unstructured data, which introduced me to the foundational process of text vectorization. I learned several scoring methods:

Binary Classification: Simply checks if a word exists in a sentence (e.g., output from x.toarray(): [[0, 1, 1, 1, 0, 0, 1, 0, 1]] for "This document is the first document").

Word Counts: Tracks how many times a word appears (e.g., xc.toarray(): [[0, 2, 1, 1, 0, 0, 1, 0, 1]] showing "document" appears twice).

TF-IDF: Weighs word importance by frequency and rarity across documents (e.g., xf.toarray(): [[0., 0.73, 0.48, 0.28, 0., 0., 0.28, 0., 0.28]]).

These methods revealed how raw text can be quantified, a prerequisite for tasks like sentiment analysis or document classification. The lab also showcased sklearn's efficiency in handling sparse matrices (e.g., using toarray() to view compressed vectors), highlighting computational optimization in machine learning.

The BoW method is a stepping stone to supervised learning, where these vectors become inputs for classification or regression tasks. A real-life example is email spam filtering: BoW could count words like “free” or “win” to flag spam, while TF-IDF might downplay common words like “the” to focus on distinctive terms. This lab laid the groundwork for understanding more advanced embeddings, like those in Lab 3 (GloVe).

Most Impactful Learning Moments: The most impactful moment was seeing the TF-IDF output (Out[22]) and its accompanying table (Out[26]), which showed how “is” and “the” had low IDF values (1.0) due to high frequency (TF=3), while rarer words like “first” had higher IDF (1.69). This clarified how TF-IDF balances word frequency with uniqueness, a practical insight into avoiding bias toward common words.

Challenges and Struggles

One technical challenge was grasping sparse matrix storage and the need for toarray(). Initially, I didn’t understand why x wasn’t a standard array until I saw the lab note: “Sklearn stores these vectors in a compressed form.” This sparsity concept—most elements being zero—was abstract until I visualized it with the output tables (e.g., Out[13]). Conceptually, distinguishing between BoW variants (binary vs. counts vs. TF-IDF) was tricky; I confused their purposes until I saw their outputs side-by-side.

Problem Solving Strategies:

To overcome the sparse matrix confusion, I revisited the lab’s explanation and ran x.toarray() repeatedly, comparing it to the vocabulary (Out[10]). This hands-on iteration helped me whereas for the conceptual mix-up, I utilized web sources and took it as an analogy: binary BoW is like a yes/no checklist, word counts are a tally sheet, and TF-IDF is a weighted scorecard. Testing these with the lab's sentences ("This document is the first document") and new inputs solidified my understanding. I breake down complex outputs into smaller parts—e.g., analyzing one row of x.toarray() against get_feature_names_out()— and also started sketching tables manually (like the one in Page 3) to connect code to intuition.

Personal Growth

Lab illustrated how machines "read" text, evolving my perception of NLP from magic to math. I was surprised by how simple BoW is yet how effective it can be—recording word presence (binary) or counts captures meaning enough for basic tasks. The efficiency of sparse storage also shocked me.

These skills pave the way to understand sentiment analysis projects. Professionally, I could use BoW to build a customer feedback classifier for a retail company, turning reviews into actionable insights. The lab's focus on sklearn also prepares me for data science roles requiring rapid prototyping.

Critical Reflection

I would like to repeat it with custom sentences beyond "This is the new sentence" to test BoW's limits—e.g., using misspellings to see how vocabulary handles noise. BoW is a baseline NLP technique in contrast to GloVe or BERT, but it gives a good starting point for understanding feature extraction. In the broader landscape, it's a historical cornerstone—used in early spam filters and search engines—showing how foundational methods evolve into advanced deep learning approaches.

Research Time: How does BoW handle synonyms or context (e.g., "bank" as river vs. finance)? This limitation hints at why embeddings (Lab 3) are needed. I am curious to explore how BoW scales with larger datasets—would sparsity still hold up? I would also like to compare BoW with n-grams or stemming from Lab 1 to see trade-offs in complexity vs. accuracy.

CONCLUSION

Lab 2 was a pivotal step as transforming text from an abstract concept into a tangible input for machine learning was a great learning. The hands-on exploration of BoW variants, backed by outputs like Out[12] (word frequencies) and Out[24] (TF-IDF tables), provided concrete evidence of its utility. Real-life parallels, like sorting emails or reviews, made the concepts relatable.

REFLECTIVE JOURNAL: MODULE 2, LAB 3 - GLOVE WORD VECTORS

INTRODUCTION

This lab explored GloVe word embeddings to represent words as vectors capturing semantic meaning by introducing pre-trained GloVe vectors, generating embeddings, and measuring word similarity using cosine similarity, offering a leap from Lab 2's BoW to contextual representations.

Learning Insights

Lab 3 introduced word embeddings, a leap beyond BoW's frequency-based approach:

- GloVe Vectors: Pre-trained embeddings (e.g., cat -> tensor([0.4528, -0.5011, ...]), Out[12]) encode word meaning in 50 dimensions.
- Cosine Similarity: Measures vector closeness (e.g., simCompare("cat", "dog", "sea") output: "'cat' is closer to 'dog' than 'sea'"), revealing semantic relationships.

These concepts showed how embeddings capture context—unlike BoW, which ignores word order or meaning—making them ideal for tasks like translation or sentiment analysis. GloVe's pre-training mirrors transfer learning, leveraging vast corpora (e.g., Wikipedia) to generalize across tasks. A real-life example is a chatbot: BoW might count "happy" and "sad" separately, but GloVe recognizes their emotional proximity, improving response nuance.

Most Impactful Learning Moments: Running simCompare("actor", "pen", "film") and seeing "'actor' is closer to 'film' than 'pen'" (Out[18]) was eye-opening as it was the human-like quality of intuitively grasping domain relationships. Printing the "cat" vector (Out[12]) and imagining its 50 dimensions as a "word fingerprint" also worked like magic.

Challenges and Struggles

The 50-dimensional vectors (Out[12]) were conceptually daunting—how do abstract numbers represent "cat"? Loading GloVe (In[11]) also felt opaque; I didn't fully grasp its pre-training process. Technically, the challenge exercise (PAGE4) stumped me initially—I wasn't sure how to extract embeddings until I saw embeddings_index (Out[14]).

Problem Solving Strategies: I tackled the vector abstraction by comparing "cat" and "dog" outputs (Out[14] for "computer" and "human"), noting similar patterns for related words. For the loading process, I skimmed the GloVe website to ease my discomfort. The challenge clicked after tracing the lab's helper function (load_glove_embeddings), reinforcing step-by-step debugging. I started visualizing vectors as "arrows" in space, with cosine similarity as the angle between them—simplifying math into intuition. Pairing outputs (e.g., Out[18]) with real-world analogies (e.g., "cat" and "dog" as pets) also grounded abstract code.

Personal Growth

My understanding shifted from viewing words as isolated units (Lab 2) to seeing them as interconnected concepts in a semantic space. GloVe revealed NLP's depth, blending linguistics with geometry, and boosted my confidence in handling pre-trained models. I was stunned that "car" was closer to "truck" than "bike" (Out[22])—embeddings captured subtle vehicular hierarchies I hadn't considered. The ease of loading pre-trained GloVe (a massive model) with one line (glove = GloVe(...)) also amazed me, highlighting the power of transfer learning.

Academically, GloVe can be used for a thesis on semantic text clustering, grouping related concepts efficiently. Professionally, it's perfect for enhancing search engines—querying "dog" could return "puppy" results based on vector proximity, improving user experience.

Critical Reflection

I would play with higher dimensions (e.g., 100d vs. 50d) to see if similarity scores change, testing GloVe's sensitivity. Lab 3 sits in the feature representation stage, advancing Lab 2's BoW into contextual embeddings critical for deep learning models like RNNs. It's a middle ground between basic NLP and modern transformers, reflecting NLP's shift toward meaning over frequency—pivotal in applications like Google Translate.

Research Time: How do embeddings handle rare words or slang not in the training corpus? Could I fine-tune GloVe for a specific domain (e.g., legal texts)?

CONCLUSION

Lab 3 elevated my NLP journey, showing how GloVe embeddings encode meaning beyond counts. Outputs like Out[18] (similarity comparisons) and Out[14] (vector prints) made abstract vectors concrete, while real-life parallels—chatbots or search engines—brought relevance.

REFLECTIVE JOURNAL: MODULE 2, LAB 4 - RECURRENT NEURAL NETWORKS

INTRODUCTION

This session reflects my exploration of RNNs, where Amazon product review sentiments were classified using two notebook runs. The initial run trained an RNN with GloVe embeddings for 35 epochs, while the revised run extended to 45 epochs to assess validation improvements, as prompted by the lab's challenge. Across both, I transformed text, integrated pre-trained embeddings, built an RNN model, and trained it, building on prior labs' NLP foundations.

Learning Insights

Lab 4 introduced RNNs for sequence modeling:

- **Text Transformation:** Tokenizing and padding (In[15], both docs) formatted reviews into 50-length tensors (e.g., Out[15]: "Its just great…").
- **GloVe Embeddings:** 300-dimensional vectors (In[18], first doc) enriched inputs, fixed in training (In[22]).
- **RNN Setup:** Two stacked layers with 128 hidden units (In[21]) captured word dependencies.
- **Training Dynamics:**
    - ❖ 35 Epochs: Loss dropped from 0.6838 to 0.4240, validation accuracy rose from 0.6254 to 0.7906.
    - ❖ 45 Epochs: Loss fell from 0.6737 to 0.4154, validation accuracy climbed from 0.6334 to 0.7879, peaking at 0.7881.

These runs showcased RNNs' ability to learn sequential patterns, with extended epochs refining performance. In real life, this may apply to review trend analysis—RNNs detect sentiment shifts like "was great, now bad," unlike static methods.

Most Impactful Learning Moments: The RNN diagram (Out[20], first doc) clarified sequence prediction—e.g., "I like machine…" unfolding step-by-step. Training outputs were striking:

- ❖ 35 Epochs: val_acc 0.7906 and loss 0.4240 showed steady gains.
- ❖ 45 Epochs: Peak val_acc 0.7881 (epoch 44) and loss 0.4154 revealed nuanced progress, though initial accuracy was lower (0.5711 vs. 0.6241, Out[23]).

Challenges and Struggles

Conceptually, RNNs' recurrence was vague—how does it "remember"? The leap from GloVe to dynamic processing took time to grasp. Technically:

- 35 Epochs: Padding to 50 (Out[15]) puzzled me with varying review lengths.
- 45 Epochs: Longer runtime (18m1s vs. 15m51s, PAGE18) and slight accuracy dip (0.7906 to 0.7879) raised overfitting concerns.

Problem Solving Strategies: I traced data flow from tokens (In[9]) to RNN outputs (In[21]), visualizing recurrence as a rolling memory. For padding, I tested pad_features outputs (Out[15]) to confirm consistency. Comparing epoch 35 metrics (0.7906 vs. 0.7852) and train/validation gaps suggested overfitting, mitigated by studying PyTorch RNN docs. I refined a "metric-tracking" approach—comparing val_acc and loss across runs to spot trends.

Personal Growth

I progressed from static (Labs 1-3) to sequential modeling, appreciating RNNs' temporal depth. My ability to tune training duration grew, balancing accuracy and generalization. The initial low accuracy (0.6241, 35 epochs; 0.5711, 45 epochs, Out[23]) surprised me—GloVe needed training to shine. The 45-epoch dip (0.7906 to 0.7879, PAGE18) was unexpected, teaching me about overfitting risks.

Academically, RNNs can be used for time-series sentiment, like tweet trends. Professionally, they suit customer feedback tracking, skills I will enhance with optimization.

Critical Reflection

I would test intermediate epochs (e.g., 40) to find an accuracy sweet spot and add dropout (In[21]) to curb overfitting seen in 45 epochs. Exploring the summary field could boost context.

Research Time: How do GRU/LSTM compare here? Could early stopping optimize val_acc?

Effect of changing num_epochs

- **Before (35 Epochs):** val_loss 0.4240, val_acc 0.7906 —strong, nearing plateau.

- **After (45 Epochs):** val_loss 0.4154, val_acc 0.7879, peaking at 0.7881—loss improved by 0.0086, but accuracy dipped 0.0027 from 0.7906. Train accuracy rose (0.7982 to 0.8020), suggesting overfitting as validation stagnated (0.7852-0.7881, epochs 35-45). The slight loss reduction didn't translate to better generalization, indicating 35 epochs were near-optimal for this setup.

CONCLUSION

Lab 4's dual RNN runs revealed sequence modeling's power and pitfalls, with outputs showing val_acc peaking at 0.7906 (35 epochs) and dipping to 0.7879 (45 epochs). Real-life ties—like review trend detection—made it tangible. Extended epochs refined loss but risked overfitting, sharpening my tuning skills for NLP's evolving challenges.

REFERENCES

https://eagleonline.hccs.edu/courses/282423/files/72350570?module_item_id=19474367

https://www.kaggle.com/code/theainerd/beginners-s-guide-to-nlp-using-spacy

https://www.analyticsvidhya.com/blog/2021/06/text-analysis-with-spacy-to-master-nlp-techniques/

https://www.geeksforgeeks.org/python-pos-tagging-and-lemmatization-using-spacy/

https://spacy.io/usage/linguistic-features

https://medium.com/biased-algorithms/a-gentle-introduction-to-the-bag-of-words-model-323be25abb09

https://www.geeksforgeeks.org/word-embeddings-in-nlp/

https://www.geeksforgeeks.org/bag-of-word-and-frequency-count-in-text-using-sklearn/

https://nlp.stanford.edu/projects/glove/

https://muneebsa.medium.com/mastering-nlp-with-glove-embeddings-word-similarity-sentiment-analysis-and-more-27f731988c48

https://www.geeksforgeeks.org/pre-trained-word-embedding-using-glove-in-nlp-models/

https://www.geeksforgeeks.org/pre-trained-word-embedding-using-glove-in-nlp-models/

https://orenshapira.medium.com/using-neural-networks-and-nlp-word-embedding-to-predict-amazon-user-review-sentiment-28156f69e1e1

https://www.geeksforgeeks.org/amazon-product-review-sentiment-analysis-using-rnn/