Static Word Embeddings and RNN

Razeen Wasif (u7283652)

August 25th, 2025

1 Question 1 Training Static Word Embeddings

Use the implementation of the method called skip-gram with negative sampling (SGNS) from Gensim's Word2Vec class to train static word embeddings (word2vec models).

(A) Train the model (M1) with a window size of 2, seed of 1, no stopwords removal. Calculate the cosine similarities of 'baseball' and 'basketball' and the cosine similarity between 'baseball' and 'computer'. Report the two cosine similarities, state whether the cosine similarities are what you have expected, and briefly explain why there is a clear difference between the two cosine similarities.
Answer:
Similarity between 'baseball' and 'basketball': **0.3946**
Similarity between 'baseball' and 'computer': **0.1672**
The cosine similarities results are within expectation. Baseball and basketball have a higher cosine similarity than baseball and computer because the Word2Vec model places words with similar context closer together in the vector space.
In the case of "basketball" and "baseball", both are nouns that appear frequently alongside words like: sport, game, team, etc. so the model assigns them highly similar vectors, resulting in a high cosine similarity.
With "baseball" and "computer", "baseball" appears with sporting terms, while "computer" appears with words like: software, code, program. The sets of words that surround them in typical text are almost entirely disjoint, therefore their vectors are oriented in very different directions in the vector space, resulting in a low cosine similarity.

(B) Train another word2vec model M2 with the same configuration as in (A) except that random seed is now set to be 2. Obtain the vectors for 'baseball' and 'basketball' from M2 and calculate their cosine similarity. Compare it with the cosine similarity between 'baseball' and 'basketball' from (A). Are the two cosine similarities the same? If not, are they close?
Answer:
Similarity between 'baseball' and 'basketball': **0.3522**
The result is ever so slightly different this time after changing the seed value. The reason being before any training happens, each word is assigned a vector of random numbers. The seed parameter just makes that initial random assignment predictable and repeatable. So, when the seed was changed from 1 to 2, the model got a different set of random starting vectors resulting in the weights being updated differently.

(C) Calculate the cosine similarity between M1's vector for 'baseball' and M2's vector for 'basketball'. Report the cosine similarity in your answers PDF and briefly explain why the similarity is not high.
Answer:
The cosine sim of m1's baseball and m2's basketball is **0.07976650446653366**.
The reason its so low is because the coordinate system for M1's vector space is different and arbitrarily oriented compared to the coordinate system for M2. So, while the relationships between words inside the same model are meaningful, comparing a raw vector from M1 directly to a raw vector from M2 is like comparing an address from one city map to an address from a completely different, unaligned map. The comparison doesn't mean anything, which is why the similarity score is low and effectively random.

(D) Using either M1 or M2, find the top-20 most similar words to 'would' and the top-20 most similar words to 'greece'. Report these words in your answers PDF and briefly explain why these words are close to 'would' and 'greece', respectively, in the learned embedding space.
Answer:
Using M1, these were the top-20 most similar words to 'would' (most were **modal verbs** like would):
[('might', 0.7345126271247864), ('will', 0.7170650362968445), ('should', 0.7164881825447083),
('could', 0.6890514492988586), ('wouldnt', 0.6764636039733887), ('must', 0.6379968523979187),

('id', 0.6334534287452698), ('may', 0.6271624565124512), ('can', 0.6166791915893555),
('wont', 0.6144319772720337), ('youd', 0.6134152412414551), ('shouldnt', 0.5986385941505432),
('theyd', 0.5531565546989441), ('theyll', 0.5434325933456421), ('hed', 0.5398749113082886),
('itll', 0.5320702195167542), ('itd', 0.5300721526145935), ('couldnt', 0.5166771411895752),
('seems', 0.5134642720222473), ('didnt', 0.5128338932991028)]

and top-20 words to 'greece' (most are name of other **countries** with some being **cities** and **regions**):
[('serbia', 0.5105131268501282), ('turkey', 0.4738893508911133), ('iran', 0.47203320264816284),
('uae', 0.46764472126960754), ('greek', 0.46178752183914185), ('macedonia', 0.4574611783027649),
('russia', 0.45088034868240356), ('yugoslavia', 0.4430218040943146), ('kuwait', 0.4427393674850464),
('azerbaijan', 0.4399421811103821), ('armenia', 0.43797361850738525), ('erzurum', 0.43644872307777405),
('cyprus', 0.43432700634002686), ('thrace', 0.42204856872558594),
('bh', 0.42170870304107666), ('croatia', 0.42065131664276123), ('bulgaria', 0.4205595850944519),
('thessaly', 0.4145038425922394), ('moscow', 0.4087381064891815), ('athens', 0.40216490626335144)]

(E) Train a third word2vec model M3 using a window size of 5. Do not remove stop words. Use any random seed. Get the top-20 similar words to 'would' and 'greece' by M3. Report these words in your answers PDF and explain in what way they are different from those you have found in (D) above. Explain why you think there is a difference

Answer:

top20 for would:
[('will', 0.6942469477653503), ('might', 0.6909372210502625), ('could', 0.6849875450134277),
('should', 0.6603264212608337), ('id', 0.6522570252418518), ('wouldnt', 0.6521463990211487),
('youd', 0.6169575452804565), ('must', 0.5853491425514221), ('can', 0.574256956577301),
('may', 0.5618827939033508), ('theyd', 0.5459462404251099), ('wont', 0.5396793484687805),
('shouldnt', 0.5369870066642761), ('theyll', 0.5081683397293091), ('ill', 0.49680885672569275),
('itd', 0.48317503929138184), ('hed', 0.48192623257637024), ('needs', 0.4747238755226135),
('seems', 0.4721110165119171), ('ought', 0.4684658646583557)]

top20 for greece:
[('greek', 0.5265691876411438), ('cyprus', 0.5090834498405457), ('macedonia', 0.49300387501716614),
('thrace', 0.4861134886741638), ('turkey', 0.4819062352180481), ('bulgaria', 0.4789105951786041),
('macedonians', 0.4775065779685974), ('croatia', 0.46201109886169434), ('athens', 0.44056499004364014),
('eastern', 0.4371405839920044), ('uae', 0.43693017959594727), ('treaty', 0.426980584859848),
('armenia', 0.4268345832824707), ('serbia', 0.42331406474113464), ('albania', 0.421310156583786),
('azerbaijan', 0.4182000756263733), ('ottoman', 0.41031578183174133), ('junta', 0.4098655581474304),
('yugoslavia', 0.4080924689769745), ('morea', 0.40005943179130554)]

Using a smaller context window, the model mostly learns from a word's immediate neighbors which is good for learning **grammatical similarity**. That is, the words frequently appear in the same immediate contexts and can often be substituted for one another in a sentence structure.
But increasing the context window size allows the model to learn about the broader topics associated with a word, by looking further away in the sentence. This is known as **Thematic Similarity**. In the case of Greece, the vectors group words that relate to the general theme of **Greece's history, politics, and geography.**

(F) Train a fourth word2vec model M4 using a window size of 5. Remove stop words this time when pre-processing the data. Use any random seed. Get the top-20 similar words to 'would' and 'greece' by M4. Report these words in your answers PDF and explain in what way they are different from those you have found in (E) above. Explain why you think there is the difference.

Answer:

would:
[('id', 0.5851573944091797), ('youd', 0.514158308506012), ('theyd', 0.5109085440635681),
('wouldnt', 0.498060405254364), ('might', 0.4517602324485779), ('could', 0.4256553053855896),
('theyll', 0.40042203664779663), ('shouldnt', 0.3941325843334198), ('wont', 0.38515347242355347),
('seemed', 0.382360577583313), ('didnt', 0.3823085129261017), ('hed', 0.3807334005832672),
('letting', 0.37630873918533325), ('situations', 0.3755369782447815), ('desire', 0.37117356061935425),
('meant', 0.3560308814048767), ('willingness', 0.3524215519428253), ('fuel', 0.3514575660228729),
('sort', 0.3493422269821167), ('expense', 0.34305673837661743)]
greece:
[('cyprus', 0.6027013063430786), ('greek', 0.59275883436203), ('macedonia', 0.5827649235725403),

('bulgaria', 0.5719061493873596), ('macedonians', 0.5573636293411255), ('turkey', 0.5463683009147644), ('aegean', 0.5131086707115173), ('islands', 0.5059866309165955), ('turkiye', 0.49718645215034485), ('corfu', 0.4966506361961365), ('junta', 0.49536943435668945), ('cypriot', 0.4823779761791229), ('ar', 0.4795033931732178), ('turkish', 0.4768601059913635), ('thrace', 0.4685835540294647), ('island', 0.46390479803085327), ('colonial', 0.46111294627189636), ('uae', 0.4590393900871277), ('falkland', 0.45633387565612793), ('morea', 0.4555879831314087)]

Based on the results, it can be concluded that removing stop words can hurt the embeddings for function words like **"would"**. It can be seen that after the first few contractions, the list becomes very noisy with words like **situations**, **desire** and **willingness**. This is because for function words, it's meaning is tied to the grammatical stopwords around it. However, removing stopwords can create a more topically dense context for content words like **'greece'** leading to more focused and thematically relevant similarities.

2 Question 2 Static Word Embeddings for Text Classification

With static word embeddings, we can represent documents using a dense vector derived from the static word embeddings of the words inside the document and use these dense vector representations of documents for text classification. Use the average (i.e. mean) of the static word embeddings from a document to represent that document. Specifically, let D represent the set of words inside a document $\mathcal{D}$ after tokenisation, and let e_w represent the static word embedding of word w. Then we will use the following vector representation e_w to represent d:

$$e_d = \frac{1}{|\mathcal{D}|} \sum_{w \in \mathcal{D}} e_w \tag{1}$$

(A) Without running any experiments, which word2vec model among M1, M3 and M4 from Question 1 do you think is likely to work better for this task? Provide your justification to support your hypothesis.
Answer:
A larger window size captures more topical or semantic relationships. Words that appear in the same general context, even if not immediately adjacent, will influence each other's vectors. This is ideal for topic classification, as you want to group words like "rocket," "planet," and "mission" together, even if they are separated by a few other words. Finally, for topic classification, stopwords are just noise since we care more about the thematic similarity than functional/grammatical similarity. so training the model without stopwords will allow for a more topically dense context vector. Therefore, for this task M4 would be the best model as it combines a large context window (for topic) with the removal of stopwords (noise reduction).

(B) Note that when training M1 and M3 in Question 1, stop words were not removed in the training corpus. When you use either M1 or M3 to construct your document embeddings for text classification in Question 2, how do you think stop words in the documents to be classified should be handled when tokenising the documents? Choose one of the options below and justify your answer without running any experiments. (1) The stop words should not be removed from the documents. (2) It does not make any difference whether stop words are removed or not. (3) It is better to remove the stop words from the documents.
Answer:
It is better to remove the stopwords. Including the stopword vectors in the average "dilutes" the "signal". It pulls the document vector away from its specific topic region and towards a generic, central point in the vector space. This makes the document vector less distinct and, therefore, harder for the classifier to categorize correctly. By removing the stopwords before calculating the document vector, we are averaging only the vectors of the meaningful, content-bearing words. This creates a "purer," more topic-specific document vector that is easier to classify.

(C) On the other hand, when training M4 in Question 1, stop words were removed. When using M4 for the classification task in Question 2, how do you think stop words in the documents to be classified should be handled when tokenising the documents?
Answer:
The stopwords should be removed. The model has no information about them. Including them is not just adding noise; it's attempting to use data that doesn't exist in the model so they cannot contribute to the document vector and must be discarded.

(D) run experiments to verify your hypotheses for Part (A), Part (B) and Part (C) above. Please list the different settings you have tried and use a table to present the results you have obtained. Discuss whether your empirical results are aligned with your hypotheses in the earlier parts. If your earlier hypotheses earlier contradict the experiment results, explain what you think are possible reasons for the empirical results you have obtained.

Answer:
The provided results are presented in Table 1: Comparison of Word2Vec Models for Text Classification.
Hypothesis A:
The best model for text classification is M4 which was trained with a large window size (5) and stop word removal. The justification was that a large window size captures topical/semantic relationships , and removing stop words during training creates a more topically dense context vector.
Empirical Result for M4: M4 achieved the highest accuracy of **0.8944**.
The empirical results align with Hypothesis (A). M4's performance is superior to all other tested configurations (M1, M2.1, and M3), confirming that the combination of a large context window and the removal of stop words during training is most effective for learning document-level, topical representations.

Hypothesis B:
When using models trained with stop words (like M1 or M3), it is better to remove the stop words from the documents before calculating the average document vector. The justification was that including stop word vectors "dilutes" the topical "signal".
Empirical Results: Comparing a model trained with a window size of 2:

- M1 (Stopwords **NOT** removed for document vector): **0.8743**.
- M2.1 (Stopwords **REMOVED** for document vector): **0.8880**.

The empirical results align with Hypothesis (B). Removing stop words during the document vector calculation (M2.1) significantly improved the accuracy from **0.8743** to **0.8880** compared to keeping them (M1). This supports the idea that the removal of these frequent, non-content-bearing words creates a "purer," more topic-specific document vector that is easier to classify.

Hypothesis C:
When using a model trained without stop words (like M4), the stop words in the documents to be classified should be removed. The justification was that the model has no information about these stop words, and attempting to use them would involve data that doesn't exist in the model's vocabulary.
The choice to use M4 with stop words removed for vector averaging aligns with Hypothesis (C), as any alternative approach would be technically flawed (no vector for the stop word) or introduce noise from a default vector, thus validating the claim that they must be removed.

(E) List two possible ways to train better word2vec embeddings for a topic-based classification task (assuming that you have unlimited storage space, computation power, and access to data) and briefly explain why you think they can potentially improve the quality of word2vec embeddings.
Answer:
Increasing the window size hyper-parameter is one of the most effective ways to improve topic-based embeddings as seen from running the various experiments.
A larger window allows the model to learn about broader topics associated with a word by considering words further away in a sentence or document. For a topic classifier, you want words that are conceptually related to have similar vectors, even if they aren't immediate neighbors. A larger window captures these long-range dependencies, making the resulting document vector (when averaged) more representative of the overall topic. For this, unlimited or very high computation power is required because a larger window drastically increases the number of context-word pairs that must be processed and used to update the word vectors.
The quality of Word2Vec embeddings is fundamentally limited by the data they are trained on. With unlimited resources, the best data-side improvement is to increase the volume and variety of text. Training on a vastly larger and more diverse corpus (e.g., all of Common Crawl, Wikipedia, and various topic-specific documents) allows the model to encounter every content word in a much wider array of contexts. This means rare words will receive sufficient updates, and even common words will have their vectors fine-tuned.

3 Question 3: RNNs for Sequence Modeling

develop an autoregressive RNN model which can generate people's names. The RNN will generate each character of a person's name given all previous characters. Your model should look like the following when training:

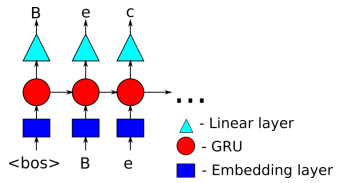

Figure 1: RNN architecture

Note that the input is shown here as a sequence of characters but in practice the input will be a sequence of character ids. There is also a softmax non-linearity after the linear layer but this is not shown in the diagram. The output (after the softmax) is a categorical probability distribution over the vocabulary, what is shown as the output here is the ground truth label. Notice that the input to the model is just the expected output shifted to the right one step with the ¡bos¿ (beginning of sentence token) prepended.

The three dots to the right of the diagram indicate that the RNN is to be rolled out to some maximum length. When generating sequences, rather than training, the model should look like the following:

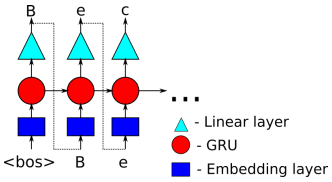

Figure 2: Enter Caption

Specifically, we choose a character from the probability distribution output by the network and feed it as input to the next step. Choosing a character can be done by sampling from the probability distribution or by choosing the most likely character (otherwise known as argmax decoding). The character vocabulary consists of the following:

- “” The null token padding string.
- `<bos>` The beginning of sequence token.
- . The end of sequence token.
- a-z All lowercase characters.
- A-Z All uppercase characters.
- 0-9 All digits.
- “ ” The space character.

3.1 Part I

(A) After training, what is the most likely name your model generates using argmax decoding.
Answer:
Argmax: Jon Barter

(B) After training, generate 10 different names using sampling and include them in your answers PDF. (For example, one of the names sampled from the model solution was Dasbie Miohmazie.)
Answer:
Random: Kerryz Loberton, Jomin Errick, Ston Hustova, John, Stin Bobne, Aly Jickan, Ramas Hiechay, Gay Veyngen, Alin Peelles, Joan Cary

3.2 Part II

If the names used for training have been grouped by their origins (e.g., German names, Chinese names, Vietnamese names, Italian names, etc.), how would you train your name generator differently in order to generate more realistic names?
Answer:
I could improve the realism of the generated names by training a single model that incorporates the origin/category as an additional input feature. This will allow the model to learn the unique phonological patterns specific to each culture.
One way this could be done is by prepending the category token as a special starting character in the sequence:

$$Sequence =< Category >< BOS > C1\ C2\ \ldots\ Cn \tag{2}$$

The category ID would force the RNN to initialize its hidden state with information about the name’s origin.

4 Question 4: Sentiment Classification

4.1 Part I: Training a Transformer-based Sentiment Classifier

Your task in this part is to train a transformer-based classifier to predict the sentiment label from the review text. Specifically, you will train a transformer encoder using PyTorch. An input sequence is first tokenised and a special [CLS] token prepended to the list of tokens. Similar to BERT1 , the final hidden state (from the transformer encoder) corresponding to the [CLS] token is used as a representation of the input sequence in this sentiment classification task.

Answer:

Accuracy (test): 0.8871

4.2 Part II: LLM for Sentiment Classification

Prompt used: "You will perform sentiment classification on a movie review. Your classification options are POSITIVE or NEGATIVE. Analyze the sentiment of the following movie review:
[Movie Review Text]
Output only one word: POSITIVE or NEGATIVE."
The accuracy was 100%.
The strong performance of the large language model i used for this task (Gemini) stems from three main factors that differ from the classifier I trained.
First, the sheer scale of training data is vastly different; Gemini was pre-trained on a massive and diverse corpus of text from the internet, not just the movie reviews. This allows Gemini to develop a much deeper and more nuanced understanding of language, including context, subtlety, and sarcasm. Second, its model architecture is significantly larger, containing many more parameters, which provides a greater capacity to learn and store complex linguistic patterns. This combination of massive data and model scale enables effective transfer learning.
In contrast, my model learns everything it knows only from the provided reviews, limiting its knowledge base and ability to generalize.

5 Appendix

Table 1: Comparison of Word2Vec Models for Text Classification

Model	Window Size	Seed	Stopwords Removed	Accuracy
m1.model	2	1	No	0.8743
m2.1.model	2	2	Yes	0.8880
m3.model	5	1	No	0.8820
m4.model	5	1	Yes	0.8944