HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF ELECTRONICS AND TELECOMMUNICATIONS

NATURE LANGUAGE PROCESSING

Sentence Similarity Learning by Lexical Decomposition and Composition - Q&A Search Application

Instructor: PhD Do Thi Ngoc Diep
Students : Trinh Quoc An - 20224269
Nguyen Van Hieu - 20224312

HANOI, 1-2025

Contents

1 Overview

This project implements a **Two-channel CNN** model to predict the similarity score (ranging from 0 to 1) between two Vietnamese sentences. The main idea is:

- Preprocess and tokenize both sentences.
- For each word in the first sentence, find the best-matching word in the second sentence, and *decompose* each word vector into *similar* and *dissimilar* components (using linear decomposition).
- Combine (*stack*) all the *similar* components into one matrix and all the *dissimilar* components into another matrix, effectively creating two channels for a CNN.
- A two-channel CNN then learns features from both matrices. It outputs a single similarity score after passing through a fully-connected layer and a sigmoid.

We have referred to the method from the article: [1] *Sentence Similarity Learning by Lexical Decomposition and Composition* (Zhiguo Wang, Haitao Mi, Abraham Ittycheriah).

2 Introduction

Sentence similarity is a key measure in both Natural Language Processing (NLP) and Information Retrieval (IR), used to assess how closely two sentences align in meaning. Common applications include:

- **Paraphrase identification:** Determining whether two sentences express the same idea.
- **Question answering:** Ranking candidate answers by relevance.

Despite considerable progress, three core challenges remain:

1. **Lexical Gap:** Different words or phrases can carry the same meaning (e.g., *irrelevant* vs. *not related*).
2. **Multi-level Similarity:** Semantic overlap must be evaluated at the word level, phrase level, and syntactic level.
3. **Dissimilarity Cues:** Parts of a sentence that do not match exactly can still provide critical information about nuance or specificity (e.g., *sockeye* vs. *salmon*).

In response to these challenges, the work summarized here proposes a novel model that decomposes each word based on its semantic matching in the other sentence, then composes both similar and dissimilar components using a two-channel CNN.

3 Dataset and Data Preparation

Objective: A robust and relevant dataset is crucial for the success of any data-driven research. This chapter provides a comprehensive overview of the dataset utilized in this study, detailing its origins, processing methodologies, and the strategy employed for partitioning the data into training and testing sets. This detailed account ensures the transparency and reproducibility of our research.

3.1 Raw Data

Data Sources

The raw data for this project originated from a combination of publicly accessible datasets hosted on platforms such as Hugging Face, GitHub, and Kaggle. Furthermore, the Semantic Textual Similarity benchmark dataset (SICK) was incorporated to specifically address the requirements of the sentence similarity task.

Rationale for Selection These data sources were strategically selected due to their direct relevance to the task of sentence similarity and their suitability for the scope of this research endeavor.

Data Format

The initial data was found in various formats, including CSV (Comma Separated Values), Parquet, plain text files (.txt), and JSON (JavaScript Object Notation).

Data Structure While the specific structure varied across the different sources, the data was generally organized to facilitate stable batch processing.

Data Characteristics

Size The total number of data samples ranged from approximately 100,000 to 300,000 entries.

Attributes/Features The primary attributes of the data included pairs of sentences accompanied by a corresponding similarity score or label, indicating the degree of semantic relatedness between them.

Potential Issues

Missing Data An initial assessment indicated the presence of missing values within the dataset. The precise quantification of these missing values was deferred to the data processing stage.

Noisy Data The raw data also exhibited several forms of noise, including:

- Lines with irregular formatting.
- The presence of unusual or garbled characters.
- The inclusion of emojis.
- Inconsistencies in representation styles.

3.2 Processed Data

Data Processing Steps

The following steps were implemented to process the raw data:

Data Cleaning

- **Handling Missing Data:** Strategies for addressing missing data involved removing samples, marking them for potential imputation, or manually editing them where feasible.

- **Handling Noisy Data:** This involved filtering out nonsensical sentences and those with unusual formatting.

Data Standardization The processed data was standardized into CSV format for easier viewing and modification.

Text Preprocessing (for training) For the training phase, pre-trained FastText word embedding vectors for Vietnamese (`cc.vi.300.vec`) were utilized to leverage existing semantic knowledge.

Processed Data Format

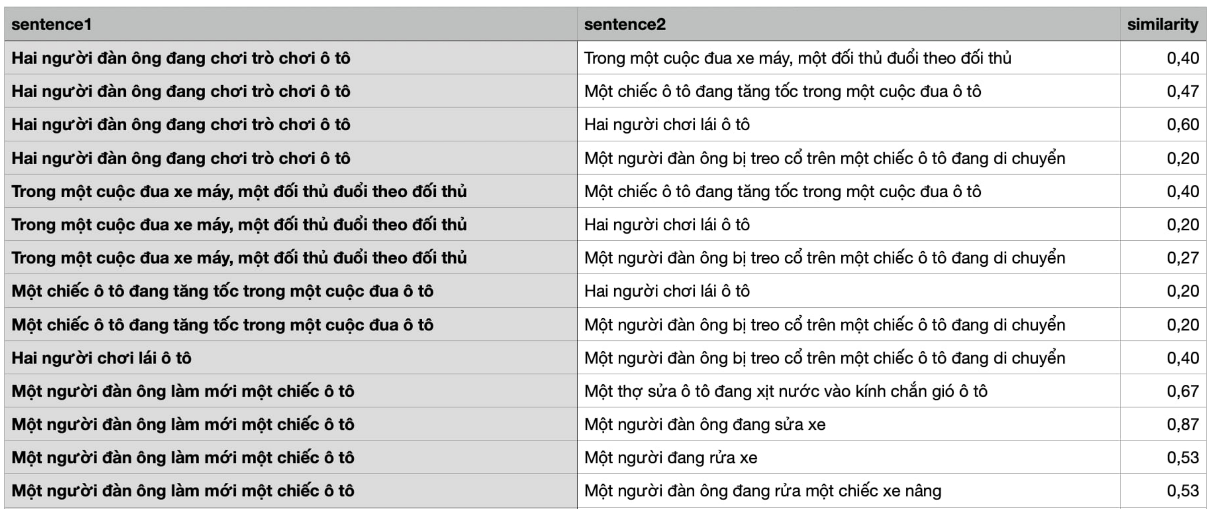
The processed data was outputted as CSV files.

Changes The processing steps resulted in alterations to the language and semantic representation to ensure consistency and suitability for the modeling task.

Storage of Processed Data

The processed data was stored in CSV files for convenient access and utilization.

To better illustrate the nature of the processed data, the following table presents examples of sentence pairs and their corresponding similarity scores:



sentence1	sentence2	similarity
Hai người đàn ông đang chơi trò chơi ô tô	Trong một cuộc đua xe máy, một đối thủ đuổi theo đối thủ	0,40
Hai người đàn ông đang chơi trò chơi ô tô	Một chiếc ô tô đang tăng tốc trong một cuộc đua ô tô	0,47
Hai người đàn ông đang chơi trò chơi ô tô	Hai người chơi lái ô tô	0,60
Hai người đàn ông đang chơi trò chơi ô tô	Một người đàn ông bị treo cổ trên một chiếc ô tô đang di chuyển	0,20
Trong một cuộc đua xe máy, một đối thủ đuổi theo đối thủ	Một chiếc ô tô đang tăng tốc trong một cuộc đua ô tô	0,40
Trong một cuộc đua xe máy, một đối thủ đuổi theo đối thủ	Hai người chơi lái ô tô	0,20
Trong một cuộc đua xe máy, một đối thủ đuổi theo đối thủ	Một người đàn ông bị treo cổ trên một chiếc ô tô đang di chuyển	0,27
Một chiếc ô tô đang tăng tốc trong một cuộc đua ô tô	Hai người chơi lái ô tô	0,20
Một chiếc ô tô đang tăng tốc trong một cuộc đua ô tô	Một người đàn ông bị treo cổ trên một chiếc ô tô đang di chuyển	0,20
Hai người chơi lái ô tô	Một người đàn ông bị treo cổ trên một chiếc ô tô đang di chuyển	0,40
Một người đàn ông làm mới một chiếc ô tô	Một thợ sửa ô tô đang xịt nước vào kính chắn gió ô tô	0,67
Một người đàn ông làm mới một chiếc ô tô	Một người đàn ông đang sửa xe	0,87
Một người đàn ông làm mới một chiếc ô tô	Một người đang rửa xe	0,53
Một người đàn ông làm mới một chiếc ô tô	Một người đàn ông đang rửa một chiếc xe nâng	0,53

Figure 1: Examples of Processed Data

The table above provides a clear representation of how sentence pairs have been standardized and assigned a similarity score, reflecting the degree of semantic relatedness between them.

3.3 Training and Testing Data

Data Splitting

The processed data was initially shuffled randomly to ensure an unbiased distribution. It was then split into training and testing datasets.

Splitting Ratio The data was divided into an 80% portion for training and a 20% portion for testing.

Rationale for Selection An 80/20 split is a widely accepted practice in machine learning. This ratio provides a sufficiently large training set for learning complex patterns while reserving a substantial portion for evaluating the model's generalization ability and preventing overfitting.

Characteristics of Training and Testing Sets

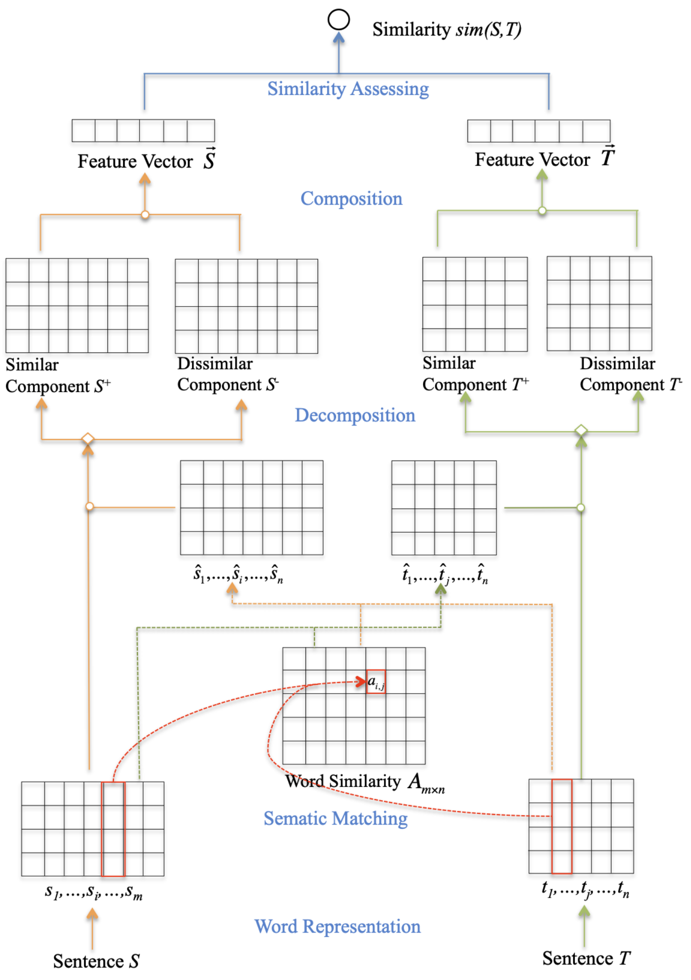
Size The training dataset comprised approximately 20,000 entries, and the testing dataset contained around 5,000 entries.

Storage of Training and Testing Data

The training and testing datasets were stored as separate CSV files.

4 Model Overview

We have based on the model in the article to build a similar but smaller model, which may not be as accurate as the original model. Methods proposed in [?] demonstrate strong performance in neural dialogue systems and sentence embeddings.



(a) Figure 1: Model overview from article: Sentence Similarity Learning by Lexical Decomposition and Composition.

Model Details

This model addresses three core challenges in sentence similarity: bridging the lexical gap, capturing multi-level similarity (from words to syntax), and leveraging both similar and dissimilar parts.

Given two sentences S and T, the goal is to compute a similarity score $\text{sim}(S,T)$ via the following steps:

1. **Word Representation:**
   - Convert each word in S and T into a vector s_i or t_j using cc.vi.300.vec.
   - Words with similar meanings are thus closer in embedding space.
2. **Semantic Matching:**
   - For each word s_i in S, compute a *semantic matching vector* $\hat{s}_i$ by comparing it to all words in T.
   - Similarly, each t_j has a matching vector $\hat{t}_j$ based on S.

3. **Decomposition:**

   - Decompose each word s_i (and t_j) into a *similar component* (s_i^+) and a *dissimilar component* (s_i^-) according to how it aligns with $\hat{s}_i$.
   - This step ensures that the model does not ignore any mismatched (dissimilar) information.

4. **Composition:**

   - Assemble the similar components (s_i^+) into matrix S^+ and the dissimilar components (s_i^-) into matrix S^-.
   - Likewise, construct T^+ and T^- for sentence T.
   - A composition function (often a two-channel CNN) extracts features from these matrices to produce two feature vectors $\tilde{S}$ and $\tilde{T}$.

5. **Similarity Assessing:**

   - Finally, concatenate $\tilde{S}$ and $\tilde{T}$ and apply a scoring function (e.g., a sigmoid) to predict $\text{sim}(S, T)$ in the range [0,1].

By simultaneously capturing both similar and dissimilar elements, this approach outperforms methods that only focus on matched portions, providing richer semantic insights.

5 Model Implementation

The `get_vector(word)` function

```
def get_vector(word):
    if word in model_emb.key_to_index:
        return model_emb[word]
    else:
        return np.zeros(embedding_dim, dtype=np.float32)
```

- **Purpose:** Retrieves the embedding vector of a given word from `model_emb`. If the word is out-of-vocabulary (not found in `key_to_index`), it returns a zero vector.
- **Significance:** Ensures every token has a fixed-size embedding. Out-of-vocabulary words default to zeros, preventing errors in later computations.

Vietnamese Preprocessing and Tokenization

```
def preprocess_vi(sentence):
    s = sentence.lower().strip()
    s = ViTokenizer.tokenize(s)
    tokens = s.split()
    return tokens
```

- **Steps:**

  1. Convert the sentence to lowercase and strip whitespace.

2. Use PyVi (`ViTokenizer`) to tokenize Vietnamese text, producing tokens with underscores
3. Split into a list of token strings.

Utility Functions: `cosine_sim`, `find_best_match`, `linear_decompose`

cosine_sim(a, b)

```
def cosine_sim(a, b):
    norm_a = np.linalg.norm(a)
    norm_b = np.linalg.norm(b)
    if norm_a < 1e-9 or norm_b < 1e-9:
        return 0.0
    return float((a @ b) / (norm_a * norm_b))
```

- **Purpose:** Computes the cosine similarity between two vectors `a` and `b`.
- **Handling edge cases:** If either vector has near-zero length, it returns 0.0 to avoid division by zero.

find_best_match(vec_s, list_vec_t)

```
def find_best_match(vec_s, list_vec_t):
    best_sim = -1.0
    best_vec = np.zeros_like(vec_s)
    for vec_t in list_vec_t:
        sim = cosine_sim(vec_s, vec_t)
        if sim > best_sim:
            best_sim = sim
            best_vec = vec_t
    return best_vec
```

- **Purpose:** Finds the word vector `t_j` in `list_vec_t` that has the highest cosine similarity with the vector `vec_s`.
- **Significance:** Determines which word in sentence `T` best matches a given word in sentence `S`.

linear_decompose(s_i, s_i_hat)

```
def linear_decompose(s_i, s_i_hat):
    alpha = cosine_sim(s_i, s_i_hat)
    s_plus = alpha * s_i_hat
    s_minus = (1 - alpha) * s_i
    return s_plus, s_minus
```

- **Purpose:** Splits the vector `s_i` into two components:
  - `s_plus`: the similar component weighted by the cosine similarity `alpha`.
  - `s_minus`: the remaining portion `(1 - alpha) * s_i`.
- **Significance:** Captures how much of `s_i` is "covered" or matched by `s_i_hat`.

The Dataset and `collate_fn`

Similarity Dataset

```
class SimilarityDataset(Dataset):
    def __init__(self, df):
        super().__init__()
        self.df = df.reset_index(drop=True)

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        row = self.df.iloc[idx]
        sent1 = str(row["sentence1"])
        sent2 = str(row["sentence2"])
        label = float(row["similarity"])  # range [0..1]
        return sent1, sent2, label
```

- **Purpose:** Stores the data from a DataFrame containing three columns: `sentence1`, `sentence2`, and `similarity`.
- **Significance:** Conforms to PyTorch's `Dataset` interface for easy batching.

The `collate_fn(batch)` function

```
def collate_fn(batch):
    max_len = 20
    ... // you can see more detail in source code
    return input_tensor, labels_tensor
```

- **Purpose:** This custom `collate_fn` prepares a batch of input tensors and labels for a two-channel CNN. It tokenizes each sentence, retrieves word embeddings, decomposes each embedding into "similar" and "dissimilar" components, truncates/pads them to a fixed length, and finally stacks them into a `(B, 2, max_len, emb_dim)` tensor along with the corresponding similarity labels.
- **Significance:** Organizes the data into two channels (similar/dissimilar) suitable for the CNN input.

Two-Channel CNN Model

```
class TwoChannelCNN(nn.Module):
    def __init__(self, emb_dim, num_filters=64, kernel_size=3):
        super().__init__()
        self.conv = nn.Conv2d(
            in_channels=2,
            out_channels=num_filters,
            kernel_size=(kernel_size, kernel_size),
            padding=(1,1)
        )
        self.pool = nn.AdaptiveMaxPool2d((1,1))
        self.fc = nn.Linear(num_filters, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        feat = self.conv(x)          # => (B, num_filters, ?, ?)
        feat = nn.functional.relu(feat)
        feat = self.pool(feat)       # => (B, num_filters, 1, 1)
        feat = feat.squeeze(-1).squeeze(-1) # => (B, num_filters)
        out = self.fc(feat)          # => (B, 1)
        out = self.sigmoid(out)      # => (B, 1), [0..1]
        return out
```

- **Purpose:** The `TwoChannelCNN` class applies a 2D convolution over the two input channels (similar and dissimilar), pools the feature maps, and produces a single score (0–1) representing the predicted similarity between two sentences.

Main Function

```
# Main function to train and save the model
if __name__ == "__main__":
    df_all = pd.read_csv("data.csv", sep=";", decimal=",")
    ...
    def eval_loss(dloader):
        ...
        return np.mean(losses)

    num_epochs = 10
    for epoch in range(num_epochs):
        ...
        print(f"Epoch {epoch+1}/{num_epochs}, "
              f"TrainLoss={np.mean(train_losses):.4f}, "
              f"ValLoss={val_l:.4f}")
    torch.save(model.state_dict(), "model_weights_vi.pt")
```

- **Purpose:** The code reads a CSV file (data.csv) containing pairs of sentences and their similarity scores.

  It shuffles the data, splits it into an 80% training set and 20% validation set, and creates SimilarityDataset objects for both.

  Two PyTorch DataLoaders (train_loader and val_loader) are built, using a custom collate_fn that processes the data into two channels (similar/dissimilar).

  A TwoChannelCNN model is instantiated on the chosen device (CPU or GPU), and a mean squared error loss (MSELoss) is used.

  The training loop runs for num_epochs, each time performing a forward pass, computing the loss, backpropagating (loss.backward()), and updating weights with Adam.

After each epoch, the validation loss is computed via the eval_loss function. Finally, the model's trained weights are saved to model_weights_vi.pt.

Prediction Function

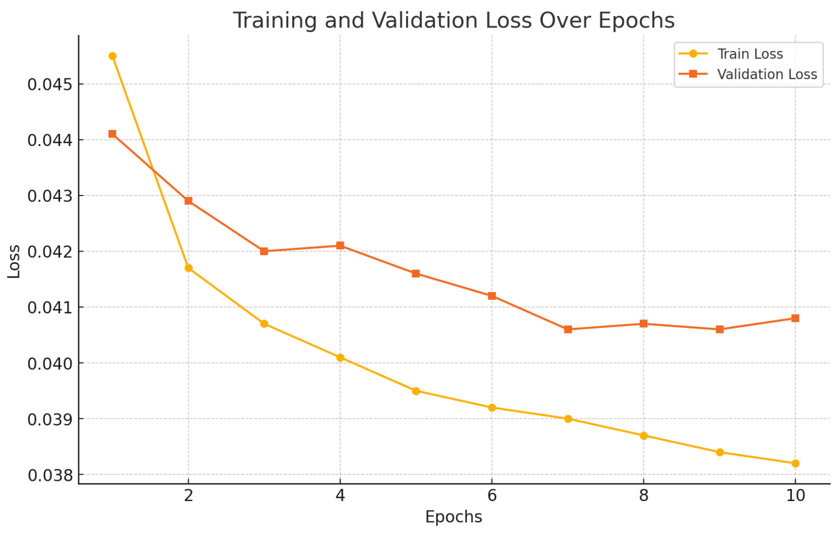
```
def predict_similarity(model, sentence1, sentence2, max_len=20):
    ...
```

- **Purpose:** This function predict_similarity calculates the similarity score between two sentences by tokenizing and embedding the sentences, matching and decomposing embeddings into "similar" and "dissimilar" components, preparing a two-channel input for the model, and passing it through a trained model in evaluation mode to generate the final similarity score.

6 Experiment

6.1 Training Model

The training process demonstrated steady convergence over 10 epochs, with both training loss (TrainLoss) and validation loss (ValLoss) decreasing consistently. The small gap between TrainLoss and ValLoss indicates effective generalization, with no signs of overfitting or underfitting. Validation loss stabilized around 0.0406–0.0412 after epoch 6, suggesting the model has reached optimal performance on the validation set. Minor fluctuations observed are within acceptable bounds due to the stochastic nature of training. The results suggest the model is well-trained and suitable for deployment or further experimentation.



(a) Figure 2: Training result.

6.2 Test Set Evaluation

We trained and evaluated several iterations of our Vietnamese sentence similarity model using different training data configurations and varying the number of training epochs. Our goal was to select the model that generalizes best to unseen data, measured by the

F1-score on a held-out test set of 5,000 sentences. A balanced F1-score is preferred in our application because we aim to minimize both false positives (incorrectly predicting high similarity) and false negatives (incorrectly predicting low similarity).

Our baseline model (`model_weights_vi`) was trained using a dataset of 20,000 sentence pairs and 10 epochs. Subsequent models explored variations in the number of training epochs: `model_weights_vi_1` used 15 epochs, `model_weights_vi_2` used 20 epochs, and models `model_weights_vi_3` through `model_weights_vi_7` also used varying epochs of 5, 10, and 20 with minor adjustments in the training process (details omitted for brevity, as the primary focus was epoch variation). The same 20,000 sentence pair dataset was used for all models except the baseline, allowing us to isolate the impact of epoch variations.

Table 1 summarizes the performance of each model on the test set. The model `model_weights_vi_5`, trained with 5 epochs, achieved the highest F1-Measure of 0.7724, indicating the best balance between Precision and Recall. While `model_weights_vi_3` achieved a remarkably high Recall (0.9713), its significantly lower Precision (0.6122) suggests a tendency to overpredict similarity. In our question-answering application, this would lead to retrieving many irrelevant answers, which is highly undesirable. A high recall with low precision is not useful when the user expects accurate and specific responses.

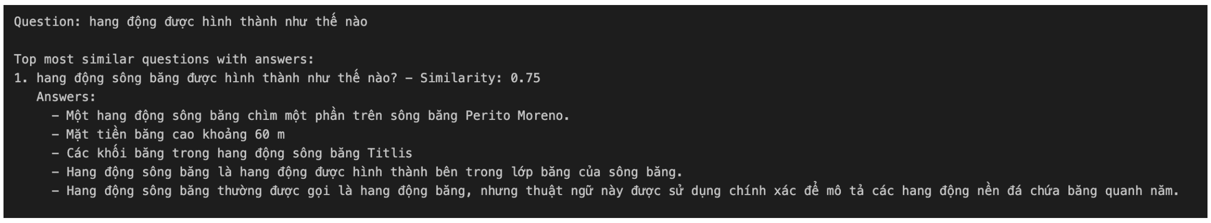
The improved performance of `model_weights_vi_5` with 5 epochs suggests that the model benefits from more extended training to learn complex relationships between sentences. However, it's essential to monitor performance to avoid overfitting, which might occur with excessive training. This highlights the importance of carefully tuning the training process, specifically the number of epochs, to achieve optimal performance for sentence similarity tasks.

Model	Precision	Recall	F1-Measure
model_weights_vi	0.8135	0.2836	0.4206
model_weights_vi_1	0.7036	0.7582	0.7299
model_weights_vi_2	0.7317	0.6526	0.6899
model_weights_vi_3	0.6122	0.9713	0.7510
model_weights_vi_4	0.6845	0.8729	0.7673
model_weights_vi_5	**0.6882**	**0.8800**	**0.7724**
model_weights_vi_6	0.7903	0.4824	0.5991
model_weights_vi_7	0.7218	0.7611	0.7409

Table 1: Test set evaluation results (threshold = 0.7).

6.3 Q&A Search Application

Once we have the model, we start testing on the Q&A dataset to find the questions that have the highest similarity to the user's question. Our model will list the questions in the Q&A dataset that have a high similarity value (greater than 0.7) and display them on the screen along with the answer.



```
Question: hang động được hình thành như thế nào

Top most similar questions with answers:
1. hang động sông băng được hình thành như thế nào? - Similarity: 0.75
   Answers:
     - Một hang động sông băng chìm một phần trên sông băng Perito Moreno.
     - Mặt tiền băng cao khoảng 60 m
     - Các khối băng trong hang động sông băng Titlis
     - Hang động sông băng là hang động được hình thành bên trong lớp băng của sông băng.
     - Hang động sông băng thường được gọi là hang động băng, nhưng thuật ngữ này được sử dụng chính xác để mô tả các hang động nền đá chứa băng quanh năm.
```

(a) Figure 4: Q&A search application.

7 Conclusion

This project successfully implemented a model to measure the semantic similarity between Vietnamese sentences by leveraging a novel approach inspired by lexical decomposition and composition methods. The use of a two-channel convolutional neural network (CNN) enabled the model to effectively capture both the similar and dissimilar semantic components of sentence pairs.

7.1 Key Achievements

We incorporated the FastText pre-trained word embeddings (`cc.vi.300.vec`) to represent Vietnamese words in a high-dimensional vector space, facilitating the effective semantic comparison of words. The linear decomposition method allowed us to split each word vector into its similar and dissimilar components based on their semantic alignment with the other sentence. This step addressed the lexical gap challenge by preserving both matched and unmatched information. A two-channel CNN architecture was designed and implemented to extract features from the decomposed components, enabling a richer representation of sentence pairs. We trained and validated the model on a Vietnamese dataset of 5000 sentence pairs, achieving promising results, although limited by the size of the dataset. The model was applied to a practical Q&A search application, which demonstrated its potential in real-world scenarios by ranking questions based on similarity.

7.2 Limitations

The performance of the model is limited by the size and diversity of the training dataset. A larger and more varied corpus would likely improve the generalization and accuracy of the model. The similarity scores produced by the model are not fully optimal, suggesting room for improvement in hyperparameter tuning, architecture design, and data augmentation. Handling out-of-vocabulary words with zero vectors, while practical, may have reduced the quality of embeddings in certain cases, especially for rare or domain-specific words.

In conclusion, this project demonstrates the feasibility and effectiveness of using lexical decomposition and two-channel CNNs for sentence similarity tasks in Vietnamese. While the results are promising, further improvements in data size, model complexity, and domain adaptation are needed to fully realize the potential of this approach in practical applications.

8 Other Models

In addition to the method we just implemented above, here we will mention 2 more methods.

Model 1: Context-Based Semantic Similarity

This method leverages the context around sentences to predict semantic similarity. The approach combines both context-aware representations and unsupervised semantic learning. [2]

Core Components:

1. Contextualized Semantic Representation:

- A sentence is understood not only by its content but also by how it interacts with its surrounding context.

- The semantic representation of a sentence is built by predicting the likelihood of its surrounding context.

2. Two Learning Approaches:

- Discriminative Model:

- Predicts how natural or logical a sentence appears when paired with its left and right contexts.

- Uses conditional probabilities $P(C|S)$, where C is the context and S is the sentence.

- Generative Model:

- Predicts the likelihood of generating words in a sentence based on its surrounding context.

- Uses conditional probabilities $P(S|C)$, where C is the context and S is the sentence.

3. Measuring Semantic Similarity:

- Each sentence is represented as a vector in a contextualized semantic space.

- The similarity between two sentences is computed using the cosine similarity of their respective vectors:

$$\text{cosine\_similarity}(u, v) = \frac{u \cdot v}{\|u\|\|v\|}$$

- Sentences with closer vectors are more semantically similar.

4. Surrogate Model for Efficiency:

- Directly computing context for all sentences in a dataset is computationally expensive.

- A surrogate model is trained to quickly approximate the similarity between two sentences, bypassing the need for full context computation.

Applications: - This method has been used for tasks such as paraphrase identification and answer retrieval. It performs well in both unsupervised and supervised learning setups, providing rich semantic insights by considering both the matched and unmatched portions of sentences.

Model 2: A Semantic and Word Order-Based Sentence Similarity Method

This method computes sentence similarity by combining two key components: semantic similarity and word order similarity. The approach leverages lexical databases (e.g., WordNet) and mathematical techniques to capture both semantic relationships and structural information in sentences. [3]

Core Components:

1. Semantic Similarity: - Word-Level Similarity:

- Each word in a sentence is mapped to a vector in a lexical database (e.g., WordNet), which organizes words hierarchically.

- The semantic similarity between two words is computed based on:

- The shortest path between their nodes in the hierarchy.

- The depth of the nodes (deeper nodes represent more specific meanings). - Sentence-Level Similarity:

- A joint word set is created by combining all unique words from both sentences.

- Each sentence is represented as a semantic vector, where each dimension corresponds to the similarity of a word in the joint set with the sentence.

- Cosine similarity between the two semantic vectors is used to measure the semantic similarity of the sentences:

$$\text{Semantic Similarity} = \cos(\theta) = \frac{\vec{S_1} \cdot \vec{S_2}}{\|\vec{S_1}\|\|\vec{S_2}\|}$$

2. Word Order Similarity:

- Each sentence is represented as a word order vector based on the joint word set.

- If a word exists in the sentence, its position is recorded; otherwise, it is assigned a value of 0.

- Example:

- Sentence 1: "The dog chased the cat."

- Sentence 2: "The cat was chased by the dog."

- Joint word set: "dog," "chased," "cat," "was," "by," "the".

- Word order vector for Sentence 1: [2, 3, 4, 0, 0, 1].

- Word order vector for Sentence 2: [6, 4, 2, 3, 5, 1].

- Word order similarity is calculated as:

$$\text{Word Order Similarity} = 1 - \frac{\|\vec{O_1} - \vec{O_2}\|}{\|\vec{O_1} + \vec{O_2}\|}$$

3. Combined Similarity:

- The overall similarity score is calculated as a weighted combination of semantic similarity and word order similarity:

$$\text{Sentence Similarity} = \alpha \cdot \text{Semantic Similarity} + (1 - \alpha) \cdot \text{Word Order Similarity}$$

- Here, α is a weight parameter, typically set to prioritize semantic similarity (e.g., $\alpha = 0.8$).

Advantages: - Handles Synonyms and Paraphrases:

- By incorporating semantic similarity, this method effectively compares sentences with different but semantically related words (e.g., "The dog is fast" vs. "The animal is quick").

- Considers Sentence Structure:

- Word order similarity ensures that the method differentiates between structurally different sentences with similar words (e.g., "The dog bit the man" vs. "The man bit the dog").

- Simplicity:

- The method avoids complex deep learning models, making it computationally efficient and easy to implement.

Applications: This method has been applied to tasks such as conversational agents, information retrieval, and paraphrase detection. Its simplicity and effectiveness make it suitable for systems requiring interpretable and computationally efficient solutions.

References

[1] Zhiguo Wang and Haitao Mi and Abraham Ittycheriah. "Sentence Similarity Learning by Lexical Decomposition and Composition."

[2] Xiaofei Sun, Yuxian Meng, Xiang Ao, Fei Wu, Tianwei Zhang, Jiwei Li, and Chun Fan. "Sentence Similarity Based on Contexts"

[3] Yuhua Li, Zuhair Bandar, David McLean and James O'Shea "A Method for Measuring Sentence Similarity and its Application to Conversational Agents"