INF721 - Deep Learning
L15: Word Embeddings

Prof. Lucas N. Ferreira
Universidade Federal de Viçosa

2024/2

1 Introduction

Word embeddings are dense vector representations of words that capture semantic relationships between words. Unlike traditional one-hot encoding, word embeddings allow models to generalize across words with similar meanings.

1.1 Motivation: Problems with One-Hot Encoding

Consider a vocabulary $V = [a, aaron, \ldots, zebra, zulu]$ with $|V| = 10,000$ words. In one-hot encoding, each word is represented as a sparse vector with a single 1 and zeros elsewhere:

$$\mathbf{o}_{man} = [0, 0, \ldots, 1, \ldots, 0] \in \mathbb{R}^{10000} \tag{1}$$

Key limitations of one-hot encoding:

- All words are equally distant from each other
- No notion of semantic similarity
- Cannot generalize across similar words
- High dimensionality and sparsity

Example: Consider a language model trained on the sentence ”I want a glass of orange juice”. When presented with ”I want a glass of apple ___”, the model cannot leverage the similarity between ”apple” and ”orange” to predict ”juice”.

2 Word Embeddings: Dense Vector Representations

Word embeddings solve these problems by learning dense vector representations where:

- Similar words have similar vectors
- Semantic relationships are captured in vector space
- Dimensionality is much lower (typically 100-300 dimensions)

2.1 The Embedding Matrix

The embedding matrix $E \in \mathbb{R}^{d \times |V|}$ maps one-hot vectors to dense embeddings:

$$\mathbf{e}_w = E\mathbf{o}_w \tag{2}$$

where:

- d is the embedding dimension (e.g., 300)
- $\mathbf{o}_w$ is the one-hot vector for word w
- $\mathbf{e}_w$ is the embedding vector for word w

3 Properties of Word Embeddings

Word embeddings exhibit interesting algebraic properties. For example, the famous analogy:

"Man is to Woman as King is to Queen" can be expressed as:

$$\mathbf{e}_{king} - \mathbf{e}_{queen} \approx \mathbf{e}_{man} - \mathbf{e}_{woman} \tag{3}$$

To solve word analogies:

$$w = \arg\max_{w} \text{sim}(\mathbf{e}_w, \mathbf{e}_{king} - \mathbf{e}_{man} + \mathbf{e}_{woman}) \tag{4}$$

3.1 Cosine Similarity

The similarity between word vectors is typically measured using cosine similarity:

$$\text{sim}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}||_2 ||\mathbf{v}||_2} \tag{5}$$

Properties:

- Range: [-1, 1]
- sim($\mathbf{u}$, $\mathbf{v}$) = 1 when vectors point in same direction
- sim($\mathbf{u}$, $\mathbf{v}$) = 0 when vectors are orthogonal
- sim($\mathbf{u}$, $\mathbf{v}$) = -1 when vectors point in opposite directions

4 Learning Word Embeddings

4.1 Neural Language Model

One of the earliest methods for learning word embeddings is the to use a neural language model. In (Bengio et al., 2003), the authors proposed to add an embedding layer to a multilayer perceptron language model as follows:

$$\begin{align}
E_s &= EO_s \tag{6}\\
A^{[1]} &= g^{[1]}(W^{[1]}E_s + \mathbf{b}^{[1]}) \tag{7}\\
A^{[2]} &= g^{[2]}(W^{[2]}A^{[1]} + \mathbf{b}^{[2]}) \tag{8}\\
&... \tag{9}\\
\hat{\mathbf{y}} &= \text{softmax}(W^{[L]}A^{[L-1]} + \mathbf{b}^{[L]}) \tag{10}
\end{align}$$

where:

- E_s is the embedding matrix for the input sentence
- O_s is the one-hot encoding of the input sentence
- E is the embedding matrix
- $W^{[l]}$ and $\mathbf{b}^{[l]}$ are the weights and biases of layer l
- $g^{[l]}$ is the activation function of layer l
- $\hat{\mathbf{y}}$ is the predicted output

4.2 Word2Vec

Word2Vec is another popular method for learning word embeddings. In general, Word2Vec is a two-layer neural network that learns word embeddings from a large corpus of text by predicting a target word from a context or the context from a target word. The former is known as the Continuous Bag of Words (CBOW) model, while the latter is known as the Skip-gram model.

4.2.1 Continuous Bag of Words (CBOW) Model

In the CBOW model, the goal is to predict a target word $w^{<t>}$ given its context $\{w^{<t-m>}, \ldots, w^{<t-1>}, w^{<t+1>}, \ldots, w^{<t+m>}\}$, where m is the context window size. For example, given the sentence "I want a glass of orange juice", the model is trained to predict "orange" given the context "a glass of juice". The model can be formulated as follows:

$$\mathbf{e}_{w^{<j>}} = E\mathbf{o}_{w^{<j>}} \tag{11}$$

$$\mathbf{h} = \frac{1}{2m} \sum_{j=t-m, j\neq t}^{t+m} \mathbf{e}_{w^{<j>}} \tag{12}$$

$$\hat{\mathbf{y}} = \text{softmax}(W\mathbf{h} + \mathbf{b}) \tag{13}$$

where:

- $\mathbf{e}_{w^{<j>}}$ is the embedding vector for word $w^{<j>}$
- $\mathbf{h}$ is the average of the context word embeddings
- W and $\mathbf{b}$ are the weights and biases of the output layer
- $\hat{\mathbf{y}}$ is the predicted output

4.2.2 Skip-gram Model

In the Skip-gram model, the goal is to predict the context words $\{w^{<t-m>}, \ldots, w^{<t-1>}, w^{<t+1>}, \ldots, w^{<t+m>}\}$ given a target word $w^{<t>}$, where m is the context window size. For example, given the sentence "I want a glass of orange juice", the model is trained to predict the words "a", "glass", "of", and "juice" given the target word "orange". The model can be formulated as follows:

$$\mathbf{e}_{w^{<t>}} = E\mathbf{o}_{w^{<t>}} \tag{14}$$
$$\hat{\mathbf{y}} = \text{softmax}(W\mathbf{e}_{w^{<t>}} + \mathbf{b}) \tag{15}$$

where:

- $\mathbf{e}_{w^{<t>}}$ is the embedding vector for the target word $w^{<t>}$
- W and $\mathbf{b}$ are the weights and biases of the output layer
- $\hat{\mathbf{y}}$ is the predicted output

Both CBOW and skip-gram models can learn word embeddings by minimizing the cross-entropy loss between the predicted output and the true output. The embeddings are then extracted from the emvedding matrix E.

4.2.3 Negative Sampling

Negative sampling is a technique used to speed up training of the Word2Vec models. Instead of training the model to predict the correct word from the context, the model is trained to distinguish between positive context-target pairs and negative context-target pairs. The positive samples are pairs of words that appear together in the corpus, while the negative samples are randomly sampled from the vocabulary. The model is trained using a binary classifier that predicts whether a given pair is positive or negative.

This approach can be applied to both the CBOW and skip-gram models. For example, we could modify the skip-gram model as follows:

$$\mathbf{e}_{w^{<t>}} = E_t\mathbf{o}_{w^{<t>}} \tag{16}$$
$$\mathbf{e}_{w^{<c>}} = E_c\mathbf{o}_{w^{<c>}} \tag{17}$$
$$\mathcal{L} = -\log(\sigma(\mathbf{e}^T_{w^{<c>}}\mathbf{e}_{w^{<t>}})) - \sum_{i=1}^{k} \log(\sigma(-\mathbf{e}^T_{w^{<n_i>}}\mathbf{e}_{w^{<t>}}) \tag{18}$$

where:

- $\mathbf{e}_{w^{<t>}}$ is the embedding vector for the target word $w^{<t>}$
- $\mathbf{e}_{w^{<c>}}$ is the embedding vector for the context word $w^{<c>}$
- $\mathcal{L}$ is the loss function

- σ is the sigmoid function
- k is the number of negative samples
- $w^{<n_i>}$ are the negative samples

Note how the model now predicts whether the context word $w^{<c>}$ is positive or negative with respect to the target word $w^{<t>}$. The model is trained to maximize the probability of the positive pairs and minimize the probability of the negative pairs.

We typically use two embedding matrices E_t and E_c for the target and context words, respectively. However, the final embeddings are extracted from the embedding matrix E_t.

5 GloVe: Global Vectors for Word Representation

GloVe is another popular method for learning word embeddings. The key idea behind GloVe is to learn word embeddings by factorizing the word-word co-occurrence matrix. The co-occurrence matrix X is a symmetric matrix where X_{ij} represents the number of times word i appears in the context of word j. For example, for the sentence "I want a glass of orange juice", the co-occurrence matrix would be:

	I	want	a	glass	of	orange	juice
I	0	1	$\frac{1}{2}$	0	0	0	0
want	1	0	1	$\frac{1}{2}$	0	0	0
a	$\frac{1}{2}$	1	0	1	$\frac{1}{2}$	0	0
glass	0	$\frac{1}{2}$	1	0	1	$\frac{1}{2}$	0
of	0	0	$\frac{1}{2}$	1	0	1	$\frac{1}{2}$
orange	0	0	0	$\frac{1}{2}$	1	0	1
juice	0	0	0	0	$\frac{1}{2}$	1	0

The GloVe model learns two weight vectors w_i and $\tilde{w}_j$ for each word w in the vocabulary. The dot product of these vectors is used to predict the co-occurrence count X_{ij} between words i and j. The model is trained by minimizing the squared error between the predicted and true co-occurrence counts. The loss function can be formulated as follows:

$$\mathcal{L} = \sum_{i,j} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log(X_{ij}))^2 \tag{19}$$

where:

- w_i and $\tilde{w}_j$ are the weight vectors for words i and j
- b_i and $\tilde{b}_j$ are the biases for words i and j
- X_{ij} is the co-occurrence count between words i and j
- $f(X_{ij})$ is a weighting function that downweights the importance of rare co-occurrences

After training, we can extract the word embeddings e_i for a given word w_i by averaging the two learned embeddings $e_i = \frac{w_i + \tilde{w}_i}{2}$.

6 Practical Considerations

Nowadays, most Natural Language Processing models use an embedding layer to learn word embeddings. In PyTorch, the `nn.Embedding` module is used to create an embedding layer. For example, the following code snippet shows how to create an embedding layer in a recurrent neural network model:

Word Embedding Layer

```
class RNNModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim):
        super().__init__()
        self.embedding = nn.Embedding(
            num_embeddings=vocab_size,
            embedding_dim=embedding_dim
        )
        # Rest of the model...
```

6.1 Pre-trained vs. Learned Embeddings

When working with Natural Language Problems, you can choose to use pre-trained word embeddings or learn the embeddings from scratch. Pre-trained embeddings are useful when you have a small dataset or want to leverage the semantic information captured in the embeddings. Popular pre-trained embeddings include Word2Vec, GloVe, and FastText. On the other hand, learning embeddings from scratch can be beneficial when you have a large dataset or want to fine-tune the embeddings for a specific task.

6.2 Conclusion

In this lecture notes, we discussed the concept of word embeddings and their importance in Natural Language Processing. We covered the properties of word embeddings, different methods for learning word embeddings, and practical considerations when working with word embeddings. Word embeddings are a powerful tool that can help improve the performance of NLP models by capturing semantic relationships between words.

Exercises

1. Given word vectors $\mathbf{e}_{\text{king}} = [2, 1]$, $\mathbf{e}_{\text{man}} = [1, 0]$, and $\mathbf{e}_{\text{woman}} = [1, 2]$, solve the analogy: "man is to woman as king is to queen" by computing $\mathbf{e}_{\text{king}} - \mathbf{e}_{\text{man}} + \mathbf{e}_{\text{woman}}$. What is the resulting vector?

   (a) $[2, 3]$
   (b) $[2, 2]$
   (c) $[3, 2]$
   (d) $[1, 3]$

2. For a CBOW model with window size m=2, what would be the context words used to predict the target word "deep" in the sentence: "I love deep learning models today"?

   (a) $["I", "love", "learning", "models"]$
   (b) $["love", "learning"]$
   (c) $["I", "learning"]$
   (d) $["love", "models"]$

3. Given vectors $\mathbf{u} = [3, 4]$ and $\mathbf{v} = [6, 8]$, calculate their cosine similarity.

   (a) 0.8
   (b) 0.9
   (c) 1.0
   (d) 0.7

4. In negative sampling with Skip-gram model, if $\mathbf{e}_{\text{target}} = [1, 1]$ and $\mathbf{e}_{\text{context}} = [2, 2]$, calculate the first term of the loss function: $-\log(\sigma(\mathbf{e}_{\text{context}}^T \mathbf{e}_{\text{target}}))$. Given $\sigma(4) = 0.982$, round to 3 decimal places.

   (a) 0.018
   (b) 0.982
   (c) 0.035
   (d) 4.000

5. Consider the co-occurrence matrix in GloVe for the sentence "the cat sat on the mat". What would be the co-occurrence count X_{ij} between "the" and "cat" with window size 1?

(a) 2

(b) 1

(c) 0

(d) 3