Word Embeddings

Lenin G. Falconí

2024-10-20

Outline

Word Embeddings

- Son representaciones vectoriales densas de las palabras en un espacio vectorial continuo
- Objetivo: capturar relaciones semánticas y sintácticas entre palabras
- Permiten que las máquinas comprendan y procesen el lenguaje natural

Neural Word Embeddings

- Utilizan redes neuronales artificiales
- La arquitectura predice la palabra siguiente dado un conjunto de palabras vecinas en una secuencia
- Los pesos aprendidos en las capas ocultas sirven como representaciones numéricas vectoriales [4]

Modelado de Lenguaje

- Un modelo del lenguaje asigna probabilidades a una secuencia de palabras
- Se asume que la probabilidad de cada palabra es independiente
- La probabilidad es el producto de las probabilidades condicionales de cada subsecuencia

$$P(x_1, \ldots, x_t) = P(x_1)P(x_2|x_1)P(x_3|x_2, x_1)\ldots P(x_t|x_{t-1}, \ldots, x_1) \quad (1)$$

$$P(x_1, \ldots, x_t) = \prod_{i=1}^{t} P(x_i|x_{i-1}, \ldots, x_1) \quad (2)$$

Word2Vec

- Propuesto por Mikolov en 2013 [2]
- El modelo está formado por dos arquitecturas principales:
  1. Continuos Bag-of-Words (CBOW): predice una palabra en función del contexto[1]
  2. Skip-gram: Dada una palabra predice las palabras de contexto
- CBOW y Skip-gram aprenden *word-embeddings* mediante el entrenamiento de una *shallow neural network* en un gran corpus de texto
- Los *word vectors* aprendidos capturan relaciones semánticas y sintácticas
- Palabras de significado similar tienen representaciones vectoriales similares
- Operaciones algebraicas lineales sobre *word vectors* revelan analogías interesantes

[1] Palabras que rodean a la palabra objetivo

Continuos Bag of Words (CBOW)

- Las palabras de contexto se tratan como un *bag*[2] sin considerar su orden
- El modelo calcula el promedio de los *context word vectors*

$$p(w_t|w_c) = softmax(w_c \cdot w_t) \tag{3}$$

donde:

- w_t es el *target word vector*
- w_c es el *average context word vector*

[2] Bolsa

Skip-gram

- El modelo aprende a identificar las palabras que aparecen frecuentemente cerca de la *target word*

$$p(w_c|w_t) = \prod_{w_i \in context} p(w_i|w_t) \tag{4}$$

Donde:

- w_c: representa las *context words*
- w_t: es el *target word vector*

GloVe I

- GloVe: Global Vectors for Word Representation
- Desarrollado por Pennington, Socher y Manning en 2014 [3]
- Combina las fortalezas de métodos de Global Matrix Factorization[3] y métodos locales de contexto de ventana[4]
- Usa *global-word-co-ocurrence* para aprender representaciones de las palabras
- El modelo predice la probabilidad de co-ocurrencia de dos palabras basado en sus representaciones vectoriales
- GloVe aprende embeddings de palabras que codifican relaciones semánticas al minimizar la diferencia entre las probabilidades de co-ocurrencia predichas y reales.

$$w_i^T w_j + b_i + b_j = log(X_{ij}) \tag{5}$$

donde

- w_i and w_j son los vectores de las palabras i y j

GloVe II

- b_i and b_j: son los bias
- X_{ij}: cuenta de co-ocurrencia de las palabras i y jx
- GloVe tiene un rendimiento *state of the art* en cálculo de similitud, analogía con palabras y *named entity recognition*

[3] Latent Semantic Analysis(LSA)
[4] Skip-gram

BERT EMBEDDINGS I

- Propuesto en 2018 por Devlin, Chang, Lee y Toutanova [1]
- BERT: Bidirectional Encoder Representations from Transformers
- Modelo de Deep Learning basado en la arquitectura de Transformers
- Entrenado en un dataset masivo de texto no etiquetado
- Utilzó dos técnicas no supervisadas su diseño:
  1. Masked Language Modeling(MLM)
  2. Next Sentence Prediction
- El diseño bi-direccional permite capturar contexto de una palabra en ambos sentidos (derecha $\rightarrow$ izquierda e izq $\rightarrow$ der)
- El modelo pre-entrenado de BERT puede ser *fine-tuned* para tareas específicas de NLP (e.g. clasificación)
- BERT ha alcanzado un desempeño state of the art en resolver 11 tareas de Lenguaje Natural (e.g. question-answering, named entity recognition, language inference, etc.)

BERT EMBEDDINGS II

- Se usa una sola arquitectura para resolver diferentes tareas.
- Se utilizaron dos etapas para conformar BERT: pre-training y fine-tuning

Métrica	Score
GLUE[5]	80.5 %
MultiNLI Acc[6]	86.7 %
SQuAD[7]	93.2 %

[5] GLUE: General Language Understanding Evaluation
[6] MultiNLI:Multi-Genre Natural Language Inference (MultiNLI)
[7] SQuAD:Stanford Question Answering Dataset

BERT EMBEDDINGS

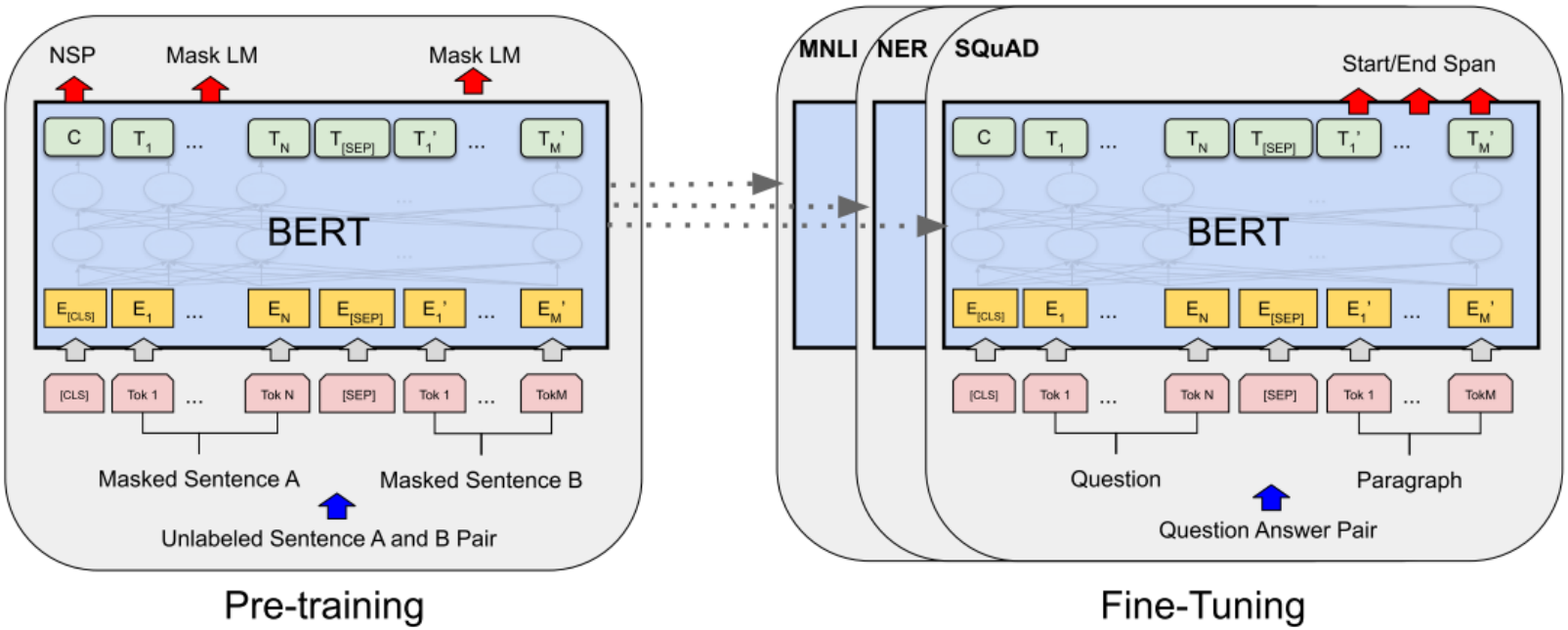

Figura: Procedimientos de pre-training y fine-tuning

Masked Language Modeling (MLM)

- Consiste en enmascarar aleatoriamente algunos de los tokens de entrada y predecir el id del vocabulario original de la palabra enmascarada
- El enmascaramiento es necesario para evitar que el modelo haga predicciones triviales
- El porcentaje de enmascaramiento utilizado es del 15 %.
- Los vectores del *hidden state* final correspondientes a los tokens de máscara se pasan por una *softmax* sobre el vocabulario
- No consiste en hacer una reconstrucción (denoising auto-encoder)
- Una limitante de este procedimiento es que se genera una discrepancia entre el pre-training y el fine-tuning ya que el token enmascarado no aparecería en el fine-tuning

Next Sentence Prediction (NSP)

- Varias tareas de Lenguaje Natural como Question Answering (QA), Natural Language Inference (NLI) dependen de comprender la relación entre 2 oraciones.
- Consiste en la predicción binaria de la siguiente oración. Es decir dadas dos oraciones A y B, el 50 % del tiempo B es la siguiente oración[8] y 50 % del tiempo es una oración aleatoria cualquiera[9]
- Permite mejorar la capacidad de entender el contexto en y la coherencia en textos largos

[8] IsNext
[9] NotNext

Arquitectura I

- Transformer encoder bidireccional multi capa basado en el diseño de Vaswani [5]
- Sea L el número de bloques de Transformer (capas), H el *hidden state* y A el número de *self attention heads*, entonces:

Modelo	Arquitectura	Parámetros
$\text{BERT}_{\text{BASE}}$	$L = 12$, $H = 768$, $A = 12$	110M
$\text{BERT}_{\text{BASE}}$	$L = 24$, $H = 1024$, $A = 16$	340M

- La entrada del modelo permite representar tanto oraciones solas como pares de oraciones (e.g. Pregunta, Respuesta) en una sola secuencia de tokens.
- BERT utiliza *WordPiece embeddings* con un vocabulario de 30000 tokens.

Arquitectura II

- WordPiece embeddings permite manejar palabras desconocidas o fuera del vocabulario al dividir las palabras en subunidades (e.g. playing $\rightarrow$ play y ##ing)
- La entrada combina: *token embeddings*, *segment embeddings* y *position embeddings*
- Token Embeddings: dispone de un vocabulario de 30000 tokens
- Segment Embeddings: Distingue entre pares de oraciones.
- Position Embeddings: Codifican (*Encode*) la posición de cada token en la secuencia

Arquitectura III

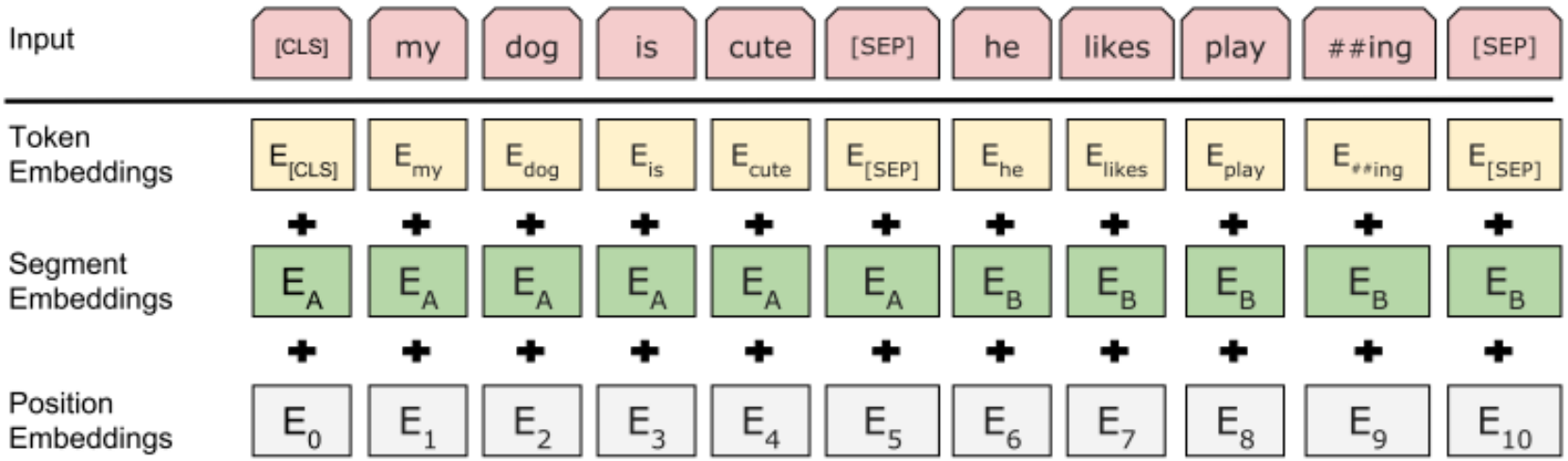

Figura: Input Embeddings

Cálculo de BERT Embeddings I

Para obtener los BERT embeddings se utiliza la librería de HuggingFace

```
from transformers import BertTokenizer, BertModel
import torch
print(torch.__version__)
```

```
2.4.1
```

Se procede a cargar el tokenizador pre-entrenado y el modelo

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
```

Se declara un texto y se lo codifica

```
text = "42 is the answer to the ultimate question of life, the Universe and Everything"
encoded_input = tokenizer(text, return_tensors='pt')
token_ids = encoded_input['input_ids']
attention_mask = encoded_input['attention_mask']
print(f"Token ID: {token_ids}")
print(f"Attention mask: {attention_mask}")
```

```
Token ID: tensor([[ 101, 4413, 2003, 1996, 3437, 2000, 1996, 7209, 3160, 1997, 2166, 1010,
         1996, 5304, 1998, 2673,  102]])
Attention mask: tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

Cálculo de BERT Embeddings II

Obtención de los Embeddings

```
with torch.no_grad():
   outputs = model(**encoded_input)
   embeddings = outputs.last_hidden_state
print(f"Word Embeddings Shape: {embeddings.shape}")
```

```
Word Embeddings Shape: torch.Size([1, 17, 768])
```

Decodificando los tokens

```
decodedText = tokenizer.decode(token_ids[0], skip_special_tokens=False)
print(f"Decoded Text: {decodedText}")
tokenizedText = tokenizer.tokenize(decodedText)
print(f"Tokenized Text: {tokenizedText}")
encodedText = tokenizer.encode(text, return_tensors='pt')
print(f"Encoded Text: {encodedText}")
```

```
Decoded Text: [CLS] 42 is the answer to the ultimate question of life, the universe and everything [SEP]
Tokenized Text: ['[CLS]', '42', 'is', 'the', 'answer', 'to', 'the', 'ultimate', 'question', 'of', 'life', ',',
Encoded Text: tensor([[ 101, 4413, 2003, 1996, 3437, 2000, 1996, 7209, 3160, 1997, 2166, 1010,
         1996, 5304, 1998, 2673,  102]])
```

Referencias Bibliográficas

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee y Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". En: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[2] Tomas Mikolov. "Efficient estimation of word representations in vector space". En: *arXiv preprint arXiv:1301.3781* (2013).

[3] Jeffrey Pennington, Richard Socher y Christopher D Manning. "Glove: Global vectors for word representation". En: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, págs. 1532-1543.

[4] Erhan Sezerer y Selma Tekir. "A Survey On Neural Word Embeddings". En: *CoRR* abs/2110.01804 (2021). arXiv: 2110.01804. URL: https://arxiv.org/abs/2110.01804.

[5] A Vaswani. "Attention is all you need". En: *Advances in Neural Information Processing Systems* (2017).