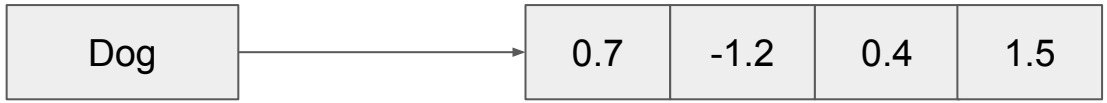
Embedding

Deep Learning

What are Embeddings?

Embeddings are a type of representation that converts sparse, categorical data into a dense, lower-dimensional, and continuous vector space.



Each category or token in the original data is mapped to a vector in this space. These vectors capture some semantic meanings of the original tokens, meaning that similar tokens will have similar vectors in the embedding space.

Example 1

```
import torch
import torch.nn as nn

# Define the size of the vocabulary and the dimension of the embedding space
vocab_size = 10  # suppose we have 10 unique tokens
embedding_dim = 3  # each token is represented by a 3D vector

# Create an embedding object
embed = nn.Embedding(vocab_size, embedding_dim)

# Sample input: indices for words in the vocabulary
input_indices = torch.LongTensor([1, 2, 3, 4])

# Get the embeddings for the input indices
embeddings = embed(input_indices)
print(embeddings)
```

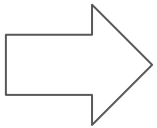
Results

```python
import torch
import torch.nn as nn

# Define the size of the vocabulary and the dimension of the embedding space
vocab_size = 10  # suppose we have 10 unique tokens
embedding_dim = 3  # each token is represented by a 3D vector

# Create an embedding object
embed = nn.Embedding(vocab_size, embedding_dim)

# Sample input: indices for words in the vocabulary
input_indices = torch.LongTensor([1, 2, 3, 4])

# Get the embeddings for the input indices
embeddings = embed(input_indices)
print(embeddings)
```

```
tensor([[ 0.6301, -1.8918,  1.1685],
        [-0.1520,  0.9526, -0.2101],
        [-0.0721, -1.2851, -1.6476],
        [ 0.9777,  0.3207,  1.6224]], grad_fn=<EmbeddingBackward0>)
```

Results

1
2
3
4



0.63	-1.89	1.17
-0.15	0.95	-0.21
-0.07	-1.29	-1.65
0.98	0.32	1.62

Example 2

```
import torch
import torch.nn as nn

# voca size = 10, embedding size = 2
embedding = nn.Embedding(num_embeddings=10, embedding_dim=2)

# test
input = torch.LongTensor([3, 1, 7, 4, 0])
output = embedding(input)

print(output)
```

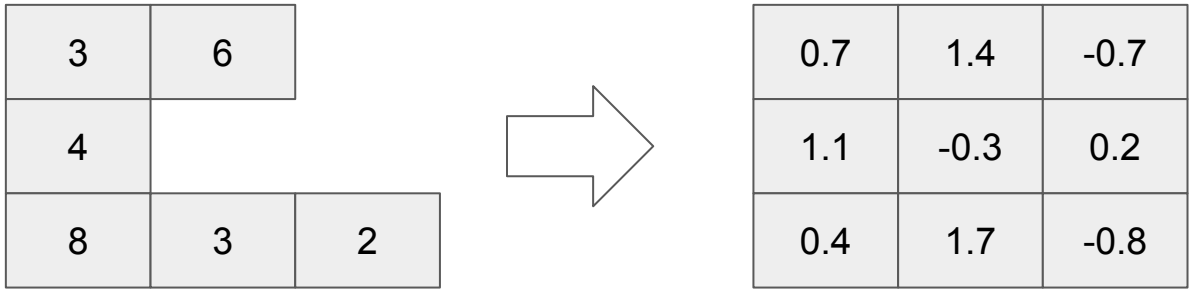
```
tensor([[ 0.6458,  0.3422],
        [-1.1166, -1.2370],
        [ 0.3500,  1.2212],
        [-2.0306,  0.3001],
        [ 1.1726,  0.7080]], grad_fn=<EmbeddingBackward0>)
```

EmbeddingBag

What is an Embedding Bag?

An embedding bag takes multiple indices as input, aggregates their embeddings directly within the lookup table, and outputs a single vector.

This aggregation can be done by taking the sum, average, or maximum of the embeddings. The EmbeddingBag does not require explicitly passing padding indices, making it particularly efficient for processing sentences or documents of different lengths.



Advantage and Application

Efficiency:

It avoids the explicit expansion of the embedding vectors in memory when it is only going to be reduced immediately by operations like sum or mean. This can lead to more memory-efficient and faster computations, especially in batched operations.

No Need for Padding:

Since EmbeddingBag automatically handles varying input lengths internally and outputs just one vector per batch item, you don't need to worry about padding your input sequences to the same length, which is a common requirement in other sequence processing operations.

Applications:

It is widely used in tasks like document classification, sentiment analysis, or any other form of text classification where the entire text needs to be represented as a single fixed-size vector.

Example 1

```
import torch
import torch.nn as nn

# Define the size of the vocabulary and the dimension of the embedding space
vocab_size = 100
embedding_dim = 5

# Create an EmbeddingBag object
embedding_bag = nn.EmbeddingBag(vocab_size, embedding_dim, mode='mean')

# Indices of words in the vocabulary
input_indices = torch.LongTensor([1, 2, 4, 5, 3, 2, 9])

# Offsets indicating the start of each sequence in the input_indices
offsets = torch.LongTensor([0, 3])

# Get the averaged embeddings for the input sequences
embeddings = embedding_bag(input_indices, offsets)
print(embeddings)
```

```
tensor([[-0.0971, -0.4928, -0.3263,  0.5550, -0.0586],
        [-0.5649, -1.3519,  0.2386,  0.1616,  0.5038]],
       grad_fn=<EmbeddingBagBackward0>)
```

Example 2

```
import torch
import torch.nn as nn

# voca size = 10, embedding size = 3
embeddingbag = nn.EmbeddingBag(num_embeddings=10, embedding_dim=3)

# test
input = torch.LongTensor([3, 1, 7, 4, 0])
output = embeddingbag(input, torch.tensor([0, 2]))

print(output)
```

```
tensor([[ 0.8344, -1.1330,  0.1764],
        [-0.0624,  1.7668,  0.0934]], grad_fn=<EmbeddingBagBackward0>)
```

2D Input (Sequence, Feature) for RNN

```python
import torch
import torch.nn as nn

# 2D input (seq_len, features)
seq_len, features = 10, 16
example_input = torch.randn(seq_len, features)  # 2D input

# LSTM model
input_size = features
hidden_size = 20  # hidden feature
lstm = nn.LSTM(input_size, hidden_size)

# 2D to 3D (batch, seq_len, features)
example_input_batched = example_input.unsqueeze(0)

print(example_input.shape)
print(example_input_batched.shape)

# both works
output, (hn, cn) = lstm(example_input)
output, (hn, cn) = lstm(example_input_batched)
```

```
torch.Size([10, 16])
torch.Size([1, 10, 16])
```