Gautam Ethiraj Follow
Feb 12, 2022 · 3 min read · Listen

What is nn.Embedding really?

$$\underbrace{\begin{bmatrix}0 & 0 & 0 & 1 & 0\end{bmatrix}}_{\text{One-hot vector}} \times \underbrace{\begin{bmatrix}8 & 2 & 1 & 9\\ 6 & 5 & 4 & 0\\ 7 & 1 & 6 & 2\\ 1 & 3 & 5 & 8\\ 0 & 4 & 9 & 1\end{bmatrix}}_{\text{Embedding Weight Matrix}} = \underbrace{\begin{bmatrix}1 & 3 & 5 & 8\end{bmatrix}}_{\text{Hidden layer output}}$$

In this brief article I will show how an embedding layer is equivalent to a linear layer (without the bias term) through a simple example in PyTorch. This might be helpful getting to grips with the nitty-grittys of implementing it in your models (even if you might have already *conceptually* known about this equivalence).

Embedding vs Linear definition-wise?

An embedding is basically the same thing as a linear layer but works differently in that it does a lookup instead of a matrix-vector multiplication.

Why use an embedding when we have a linear layer?

An embedding is an efficient alternative to a single linear layer when one has a *large number of input features*. This may happen in natural language processing (NLP) when one is working with text data or in some (language-like) tabular data that is treated as a bag-of-words (BoW). In such cases its also quite common to have the input data available as a sparse matrix (typically a result of an output from sklearn's CountVectorizer of TfidfVectorizer as a *sparse.scipy.csr_matrix*) and it is memory-inefficent to convert that in to a dense matrix but really easy to access its non-zero elements and their positions directly instead (using the *data* and *indices* attributes).

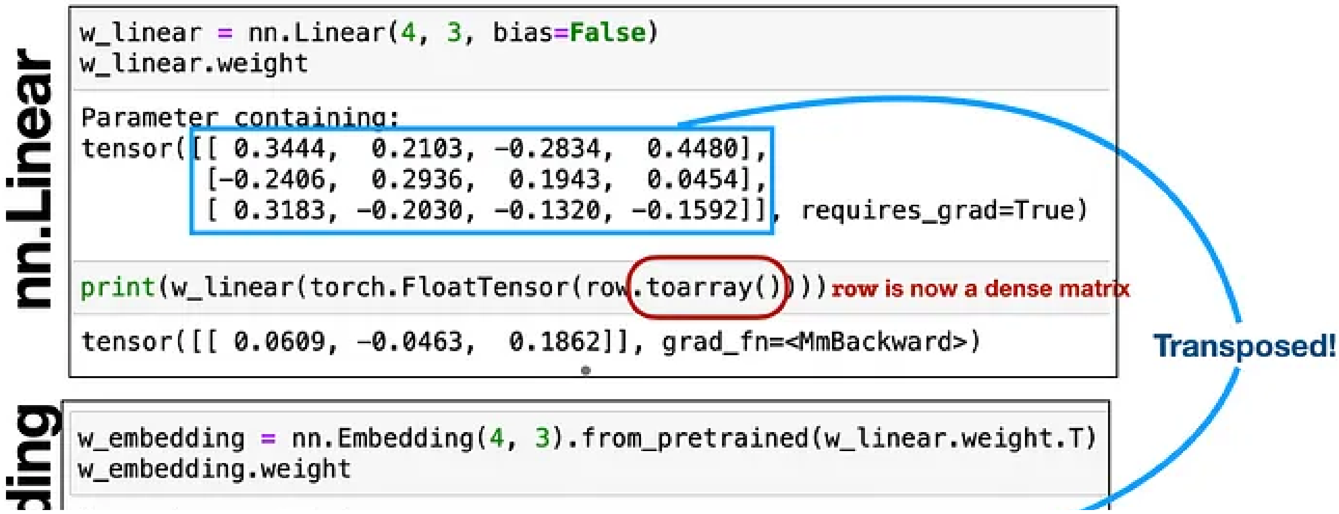
Lets look at the example

Assume *X_train* is the input data of the training set in a sparse matrix format. For this simple example I am creating a small sparse matrix so that one can easily follow along. I also extract one *row* of this matrix which is like taking one sample from the dataset that would go through the forward method of your model. The *.getrow()* method helps here and that's what you will need in your PyTorch *Dataset/Dataloaders.*

```
# Initialize a sparse matrix: This could be your training set
X_train = csr_matrix(np.array([[1, 0, 1, 0],
                               [0, 0, 1, 1],
                               [1, 1, 1, 0]]))

# Get one row: One sample in the training set
row = X_train.getrow(0)
```

Now let's pass the training example *row* through the linear layer and the embedding so that we get the same result in each case.



Same final result with an embedding layer as with a linear layer!

The outputs are the same. Yay! A couple of observations to keep in mind when you're using this in your own *nn.Module*:

1. The embedding weights and the linear layers weights are transposed to each other.
2. The linear layer *w_linear* does the actual matrix vector multiplication and therefore needs the row to be converted to dense format. In contrast, *w_embedding* just needed the indices of *row* to do a lookup. Not only is this faster, but it's also quite convenient with the *scipy.sparse.indices* attribute that is available for the sparse matrix!
3. The embedding requires the *sum(0).* Don't forget it!

Thanks Jeremy Howard for exposing me to this idea several years ago!

Pytorch Embedding Linear Logistic Regression Optimization