Embeddings: Obtaining embeddings

This section covers several means of obtaining embeddings, as well as how to transform static embeddings into contextual embeddings.

Dimensionality reduction techniques

There are many mathematical techniques that capture the important structures of a high-dimensional space in a low-dimensional space. In theory, any of these techniques can be used to create an embedding for a machine learning system.

For example, principal component analysis (PCA) has been used to create word embeddings. Given a set of instances like **bag of words** vectors, PCA tries to find highly correlated dimensions that can be collapsed into a single dimension.

Training an embedding as part of a neural network

You can create an embedding while training a **neural network** for your target task. This approach gets you an embedding well customized for your particular system, but may take longer than training the embedding separately.

In general, you can create a hidden layer of size *d* in your neural network that is designated as the **embedding layer**, where *d* represents both the number of nodes in the hidden layer and the number of dimensions in the embedding space. This embedding layer can be combined with any other features and hidden layers. As in any deep neural network, the parameters will be optimized during training to minimize loss on the nodes in the network's output layer.

Returning to our food recommendation example, our goal is to predict new meals a user will like based on their current favorite meals. First, we can compile additional data on our users' top five favorite foods. Then, we can model this task as a supervised learning problem. We set four of these top five foods to be feature data, and then randomly set aside the fifth food as the positive label that our model aims to predict, optimizing the model's predictions using a **softmax** loss.

During training, the neural network model will learn the optimal weights for the nodes in the first hidden layer, which serves as the embedding layer. For example, if the model contains three nodes in the first hidden layer, it might determine that the three most relevant dimensions of food items are sandwichness, dessertness, and liquidness. Figure 12 shows the one-hot encoded input value for "hot dog" transformed into a three-dimensional vector.

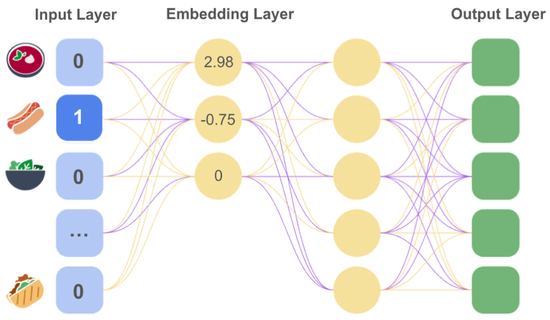

Figure 12. *A one-hot encoding of* `hot dog` *provided as input to a deep neural network. An embedding layer translates the one-hot encoding into the three-dimensional embedding vector* `[2.98, -0.75, 0]`.

In the course of training, the weights of the embedding layer will be optimized so that the **embedding vectors** for similar examples are closer to each other. As previously mentioned, the dimensions that an actual model chooses for its embeddings are unlikely to be as intuitive or understandable as in this example.

Contextual embeddings

One limitation of `word2vec` static embedding vectors is that words can mean different things in different contexts. "Yeah" means one thing on its own, but the opposite in the phrase "Yeah, right." "Post" can mean "mail," "to put in the mail," "earring backing," "marker at the end of a horse race," "postproduction," "pillar," "to put up a notice," "to station a guard or soldier," or "after," among other possibilities.

However, with static embeddings, each word is represented by a single point in vector space, even though it may have a variety of meanings. In the last exercise, you discovered the limitations of static embeddings for the word *orange*, which can signify either a color or a type of fruit. With only one static embedding, *orange* will always be closer to other colors than to *juice* when trained on the `word2vec` dataset.

Contextual embeddings were developed to address this limitation. Contextual embeddings allow a word to be represented by multiple embeddings that incorporate information about the surrounding words as well as the word itself. *Orange* would have a different embedding for every unique sentence containing the word in the dataset.

Some methods for creating contextual embeddings, like ELMo, take the static embedding of an example, such as the `word2vec` vector for a word in a sentence, and transform it by a function that incorporates information about the words around it. This produces a contextual embedding.

Click here for details on contextual embeddings

- For ELMo models specifically, the static embedding is aggregated with embeddings taken from other layers, which encode front-to-back and back-to-front readings of the sentence.
- BERT models mask part of the sequence that the model takes as input.
- Transformer models use a self-attention layer to weight the relevance of the other words in a sequence to each individual word. They also add the relevant column from a **positional embedding matrix** (see positional encoding) to each previously learned token embedding, element by element, to produce the input embedding that is fed into the rest of the model for inference. This **input embedding**, unique to each distinct textual sequence, is a contextual embedding.

NOTE: See the LLM module for more details on transformers and encoder-decoder architecture.

While the models described above are language models, contextual embeddings are useful in other generative tasks, like images. An embedding of the pixel RGB values in a photo of a horse provides more information to the model when combined with a positional matrix representing each pixel and some encoding of the neighboring pixels, creating contextual embeddings, than the original static embeddings of the RGB values alone.

Key terms:

- Bag of words
- BERT
- Embedding layer
- Embedding vector
- Neural network
- Positional encoding
- Softmax