Embeddings

Let's learn what embeddings are and how do we generate them using machine learning techniques.

We'll cover the following

Embeddings

Embeddings enable the encoding of entities (e.g., words, docs, images, person, ad, etc.) in a low dimensional vector space such that it captures their semantic information. Capturing semantic information helps to identify related entities that occur close to each other in the vector space.

This representation of entities in a lower-dimensional vector space has been of massive help in various ML-based systems. The use of embeddings has seen a major increase because of the recent surge in the use of neural networks and transfer learning.

Usually, they are generated using neural networks. A **neural network** architectures can be set up easily to learn a dense representation of entities. We will go over a few of such architectures later in this lesson.

Transfer learning (https://www.educative.io/collection/page/10370001/6237869033127936/5927951246819328) refers to transferring information from one ML task to another. Embeddings easily enable us to do that for common entities among different tasks. For example, Twitter can build an embedding for their users based on their organic feed interactions and then use the embeddings for ads serving. Organic interactions are generally much greater in volume compared to ads interactions. This allows Twitter to learn user interests by organic feed interaction, capture it as embedding, and use it to serve more relevant ads.

Another simple example is training word embeddings (like Word2vec) from Wiki data and using them as spam-filtering models.

In this lesson, we will go through some general ways of training neural networks to learn embeddings, using real-world example scenarios of their usage.

Text embeddings

We will go over two popular text term embeddings generation models and examples of their utilization in different ML systems.

Word2vec

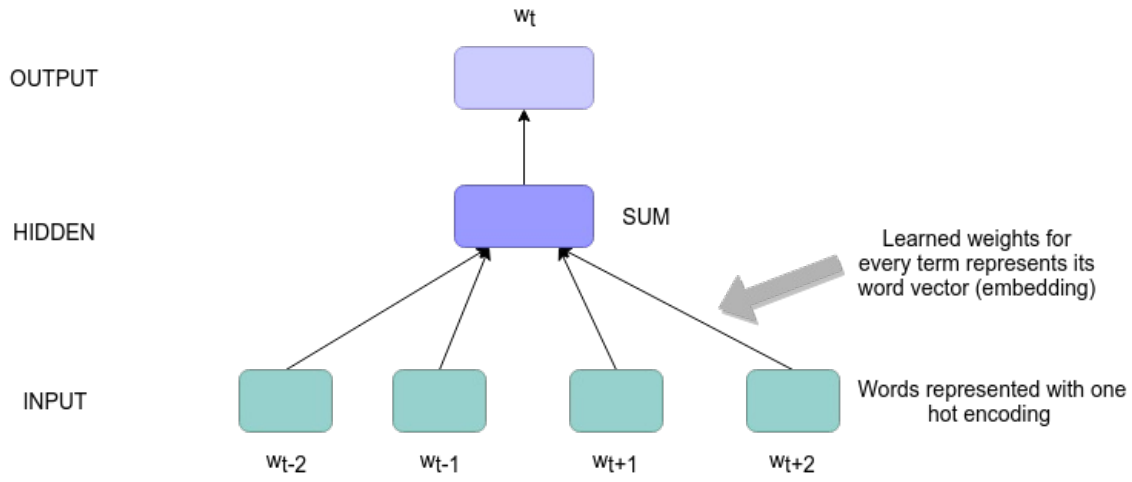
Word2vec produces word embeddings by using shallow neural networks (having a single hidden layer) and self-supervised learning from a large corpus of text data. Word2vec is self-supervised as it trains a model by predicting words from other words that appear in the sentence(context). So, it can utilize tons of text data available in books, Wikipedia, blogs, etc. to learn term representation.

Representing words with a dense vector is critical for the majority of Natural language processing (NLP) tasks. Word2vec uses a simple but powerful idea to use neighboring words to predict the current word and in the process, generates word embeddings. Two networks to generate these embeddings are:

1. **CBOW:** Continuous bag of words (CBOW) tries to predict the current word from its surrounding words by optimizing for following loss function:

$$Loss = -log\ (\text{p}(w_t \,|\, w_{t-n},...,w_{t-1}, w_{t+1}, w_{t+n}))$$

where n is the size of our window to look for the corresponding word. It uses the entire contextual information as one observation while training the network. Utilizing the overall context information to predict one term helps generate embeddings with the smaller training dataset. The architecture would look like the following:

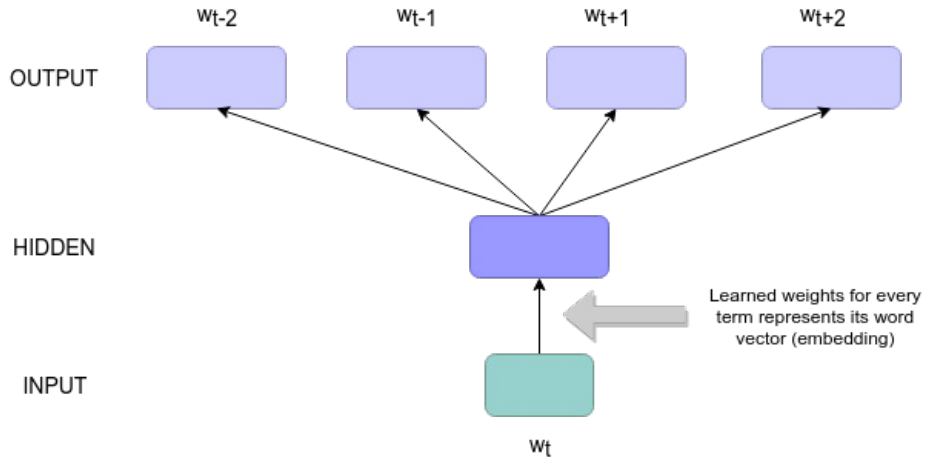


Word2vec using CBOW of window size two, i.e., predicting current word using surrounding words

2. **Skipgram:** In this architecture, we try to predict surrounding words from the current word. The loss function will now look like:

$$Loss = -log\ (\text{p}(w_{t-n},...,w_{t-1}, w_{t+1}, w_{t+n} \,|\, w_t))$$

where n is now the size of surrounding words that we are trying to predict using the current word. In training this network, each context pair will result in a different training example that the model tries to predict. This architecture is most helpful when we have a large training set. The model architecture now looks like:



Word2vec using Skip-gram to predict two surrounding words to current word

Example

Any machine learning task that wants to utilize text terms can benefit from this dense embedding vector, which captures word semantic meanings.

Let's assume that we want to predict whether a user is interested in a particular document given the documents that they have previously read. One simple way of doing this is to represent the user by taking the mean of the Word2vec embeddings of document titles that they haved engaged with. Similarly, we can represent the document by the mean of its title term embeddings. We can simply take the dot product of these two vectors and use that in our ML model.

Another way to accomplish this task is to simply pass the user and the document embedding vector to a neural network to help with the learning task.

Context-based embeddings

Once trained, Word2vec embeddings have a fixed vector for every term. So, a Word2vec embedding doesn't consider the context in which the word appears to generate its embedding. However, words in a different context can have very different meanings. For example, consider these two sentences:

1. I'd like to eat an apple.
2. Apple makes great products.

Word2vec will give us the same embedding for the term "apple" although it points to completely different objects in the above two sentences.

So, contextualized information can result in different meanings of the same word, and context-based embeddings look at neighboring terms at embedding generation time. This means that we have to provide contextual information (neighboring terms) to fetch embeddings for a term. In a Word2vec case, we don't need any context information at the embeddings fetch time as embedding for each term was fixed.

Two popular architectures used to generate word context-based embedding are:

1. Embeddings from Language Models (ELMo)
2. Bidirectional Encoder Representations from Transformers (BERT)

The idea behind **ELMO** is to use the bi-directional LSTM model to capture the words that appear before and after the current word.

BERT uses an attention mechanism and is able to see all the words in the context, utilizing only the ones (i.e., pay more attention) which help with the prediction.

We will have an in-depth explanation of these models and architectures in the entity linking chapter (https://www.educative.io/collection/page/10370001/6237869033127936/5575618289729536). Entity recognition and linking are also a great examples of how contextual embeddings can be very effective; we will discuss this in detail in that chapter.

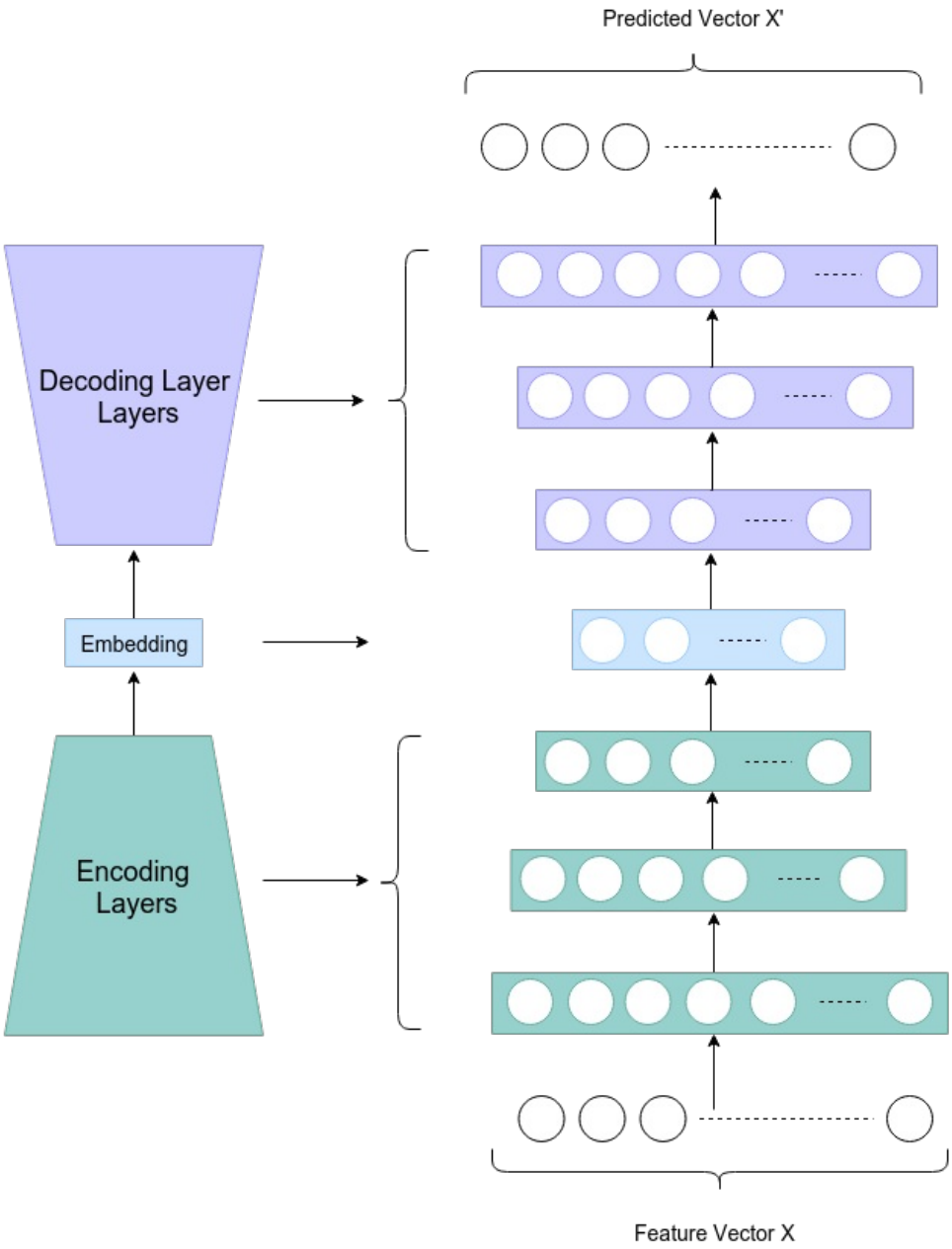
Visual embedding

Let's discuss a couple of interesting ways to generate image embedding.

Auto-encoders

Auto-encoders use neural networks consisting of both an encoder and a decoder. They first learn to compress the raw image pixel data to a small dimension via an encoder model and then try to de-compress it via a decoder to re-generate the same input image. The last layer of encoder determines the dimension of the embedding, which should be sufficiently large to capture enough information about the image so that the decoder can decode it.

The combined encoder and decoder tries to minimize the difference between original and generated pixels, using backpropagation to train the network. The network will look like the following:



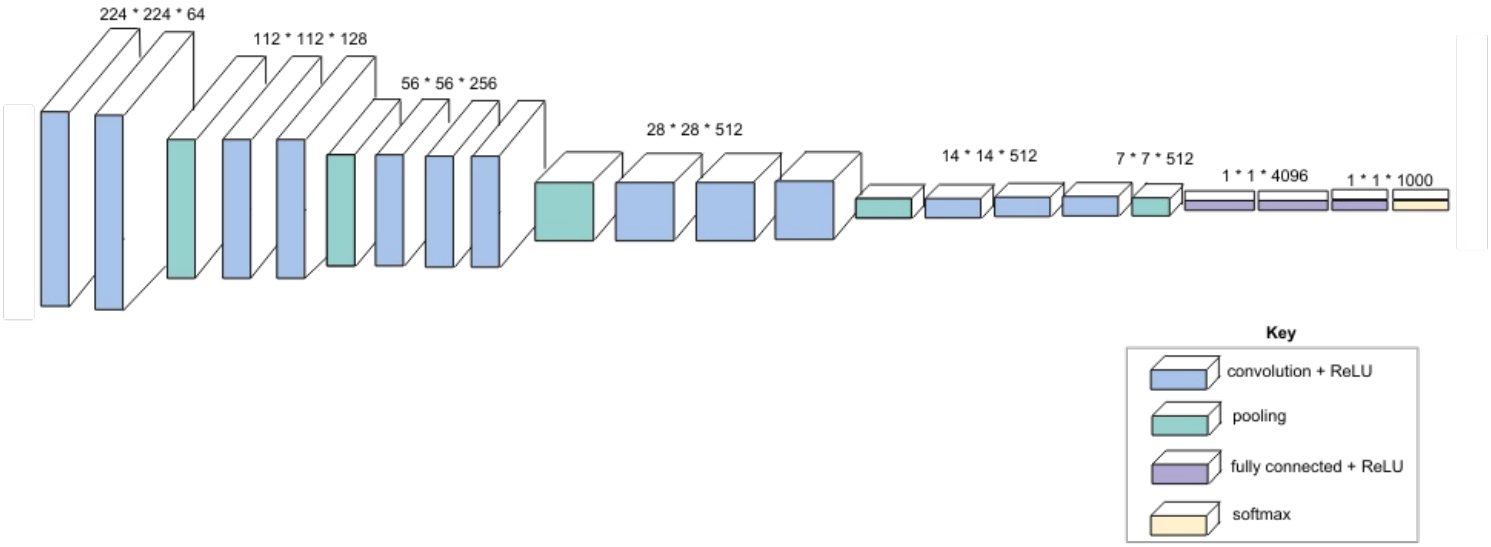
Auto-encoder model architecture for generating visual embeddings

Once we have trained the model, we only use the encoder (first N network layers) to generate embeddings for images.

Auto-encoders are also an example of self-supervised learning, like Word2vec, as we can use an image data set without any label to train the model and generate image embeddings.

Visual supervised learning tasks

Visual supervised learning tasks such as image classification or object detection, are generally set up as convolution layers, pooling layers, and fully connected network layers, followed by final classification(softmax) layers. Let's consider the example of the *ImageNet VGG16 model* that is shown in the figure below. The input passes through a set of convolution, pooling, and fully connected layers to the last softmax layer for the final classification task. The penultimate layer before softmax captures all image information in a vector such that it can be used to classify the image correctly. So, we can use the penultimate layer value of a pre-trained model as our image embedding.



VGG16 architecture

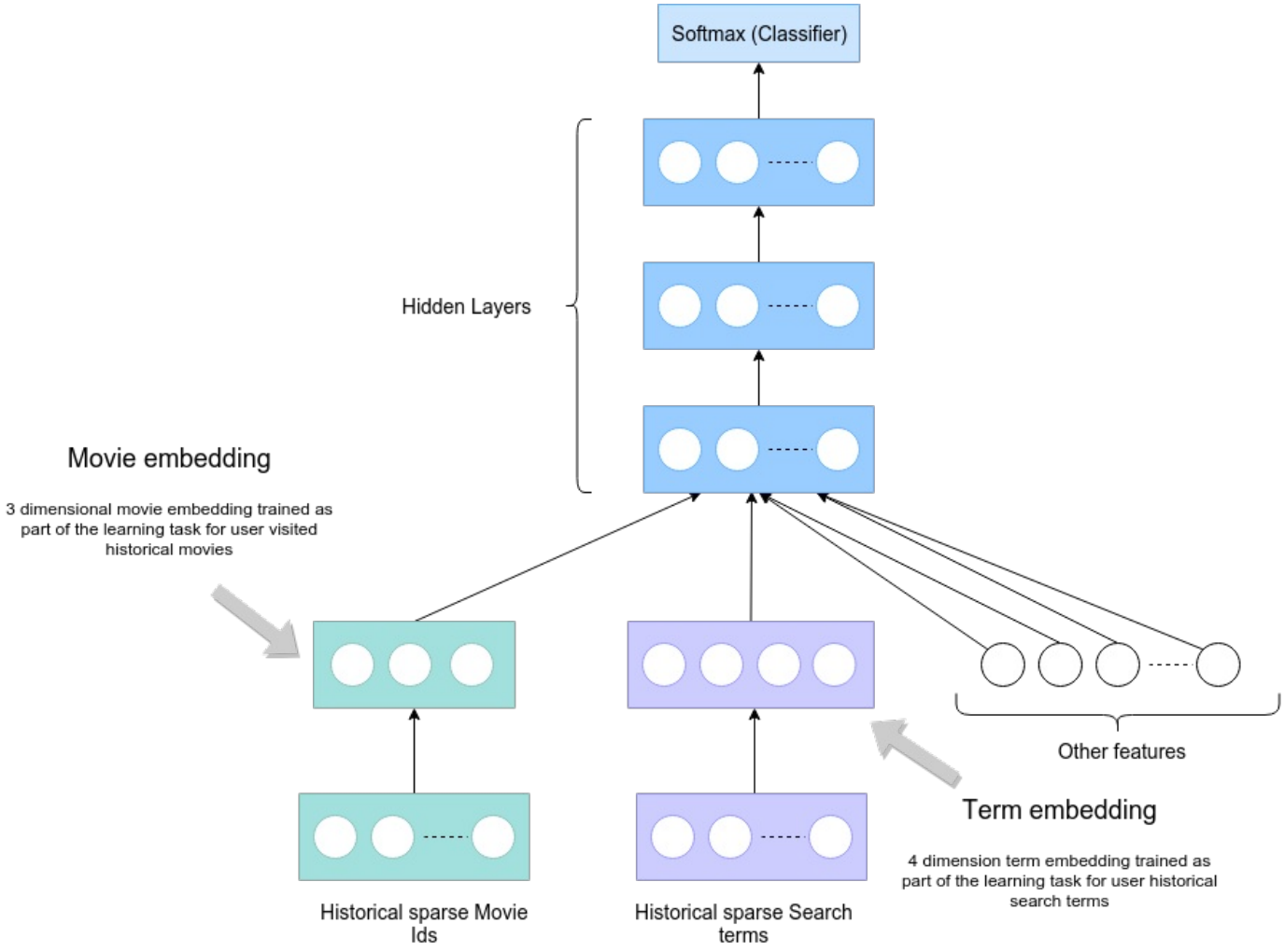
An **example** of image embedding usage could be to find images similar to a given image.

Another example is an image search problem where we want to find the best images for given text terms, e.g. query “cat images”. In this case, image embedding along with query term embedding can help refine search relevance models.

Learning embeddings for a particular learning task

Most of our discussion so far has been about training a general entity embedding that can be used for any learning task. However, we can also embed an entity as part of our learning task. The advantage of this embedding is a specialized one for the given prediction task. One important assumption here is that we have enough training data to be able to learn such representation during model training. Another consideration is that training time for learning the embedding as part of the task will be much higher compared to utilizing a pre-trained embedding.

Let’s consider an example where we are trying to predict whether a user will watch a particular movie based on their historical interactions. Here, utilizing movies that the user has previously watched as well as their prior search terms can be very beneficial to the learning task. We can do this by embedding sparse vector of movies and terms in the network itself, as shown in the image below. We will also discuss this specialized embedding usage in detail in the recommendation system chapter (https://www.educative.io/collection/page/10370001/6237869033127936/5559503958310912).



Network/Relationship-based embedding

Most of the systems have multiple entities, and these entities interact with each other. For example, Pinterest has users that interact with Pins, YouTube has users that interact with videos, Twitter has users that interact with tweets, and Google search has both queries and users that interact with web results.

We can think of these interactions as relationships in a graph or resulting in interaction pairs. For the above example, these pairs would look like:

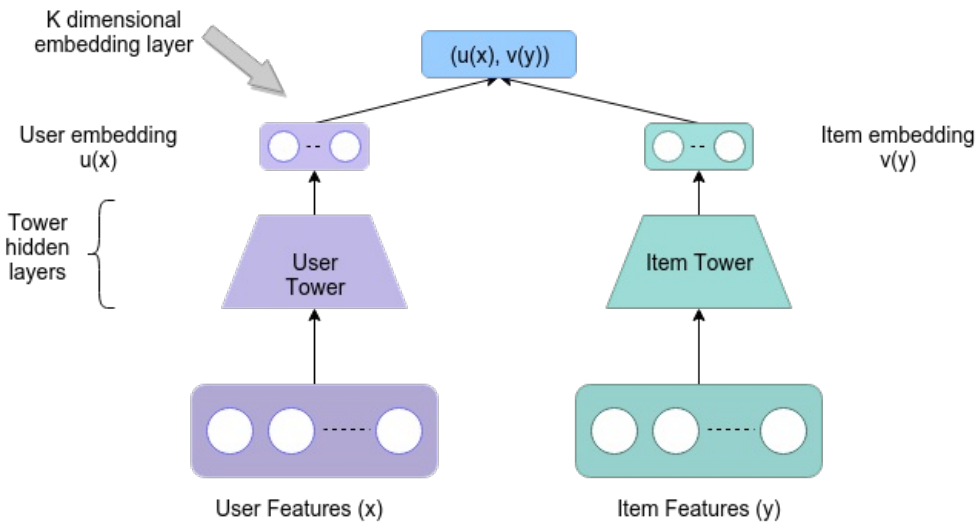
1. (User, Pin) for Pinterest
2. (User, Video) for YouTube
3. (User, Tweet) for Twitter
4. (Query, Webpage) for Search
5. (Searcher, Webpage) for Search

In all the above scenarios, the retrieval and ranking of results for a particular user (or query) are mostly about predicting how close they are. Therefore, having an embedding model that projects these documents in the same embedding space can vastly help in the retrieval and ranking tasks of recommendation, search, feed-based, and many other ML systems.

We can generate embeddings for both the above-discussed pairs of entities in the same space by creating a two-tower neural network model that tries to encode each item using their raw features. The model optimizes the inner product loss such that positive pairs from entity interactions have a higher score and random pairs have a

lower score. Let's say the selected pairs of entities (from a graph or based on interactions) belong to set A. We then select random pairs for negative examples. The loss function will look like

$$Loss = max(\sum_{(u,v)\in A} dot(u,v) - \sum_{(u,v)\notin A} dot(u,v))$$



Two tower model to optimize inner product loss for user and item embeddings

---

Back

Online Experimentation

Next

Transfer Learning

Mark as
Completed
