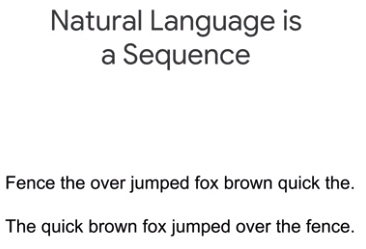
Text Classification:



A screenshot of a cell phone

Description automatically generated

ML models work with mathematical operations on numbers, so we can’t just feed in text data. We need a numerical representation of the text data.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Words that are not in our vocabulary are assigned 0.

A screenshot of a cell phone

Description automatically generated

You may be thinking, why do we need to have a constant length input? Aren't RNNs able to handle variable length inputs? Well, they are. But in practice, we don't just feed one example into an RNN at a time, we feed a whole batch; and each example in the batch needs to have the same length, otherwise it wouldn't be a legal tensor and we couldn't do efficient matrix operations on it.

A screenshot of a cell phone

Description automatically generated

At this point, we have a simple numeric representation of our sentence, but it's not the numeric representation we want. Why is that? It's because this is a categorical representation. The numbers don't have meaningful magnitude, they just serve as numeric IDs. So, in order to use them, we'd have to one-hot encode each number. If our vocabulary size is 20,000, that means a vector with 1,999 zeros and a single one.

Why use word embedding?

The advantage of using an embedding over a one-hot encoding is that it avoids this sparsity which neural networks struggle with. Also, the embeddings can learn which words are similar to each other by assigning them similar numbers.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

SELECTING A MODEL:

A close up of a logo

Description automatically generated

Here, we will use CNNs to analyze groups of adjacent words.

A screenshot of a cell phone

Description automatically generated

In image classification, each pixel was represented by three features: the red channel, the blue channel, and the green channel. In text classification, each word will be represented by the number of dimensions in our embedding space which itself is a hyperparameter.

A screenshot of a cell phone

Description automatically generated

A picture containing animal, plant

Description automatically generated

While Keras is a great way to do rapid prototyping, it doesn't support distributed training. Luckily, there's a simple way to convert a Keras model into an estimator so you can have the best of both worlds: user-friendly APIs and distributed training. Simply use the keras.estimator.model\_to\_estimator function which takes in a compiled Keras model and returns an estimator.

A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Word Embeddings:

Earlier we constructed the embeddings using the supervised learning and labels. But in reality we don’t have enough information of the labels and it is costly.

So, we can use transfer learning to suffice to this problem.

A screenshot of a cell phone

Description automatically generated

You'll learn about how researchers in other disciplines have historically constructed embeddings of words without training on a supervised task, recent techniques called GloVe and Word2Vec that are inspired by those techniques, how you can easily make use of pre-trained Word2vec embeddings using TensorFlow Hub, and how your task and the amount of data you have determine how you should use Wrd2vec and GloVe in your model.

A close up of a logo

Description automatically generated

A close up of text on a white background

Description automatically generated

But human subjects are expensive and this process was hard to scale. Beginning in the late one 1980's, researchers began exploring methods of creating numerical representations of word meaning that didn't require any human labeling. At the core of their approaches was an idea called the distributional hypothesis, which stated that the meanings of words can be found in their usage. It's an idea that is very intuitive.

A screenshot of a social media post

Description automatically generated

One of the first to these approaches came from researchers who were trying to solve the problem of ranking documents relative to a query. Their approach was called Latent Semantic Analysis and involved two steps. The first step was to compile a term-document matrix.

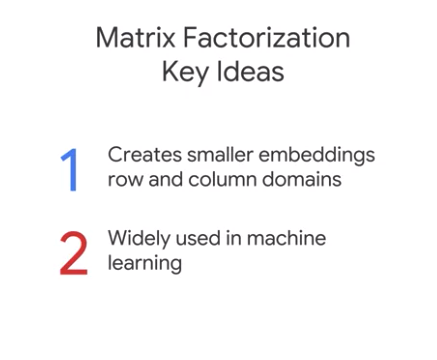
A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Think about the similarity between two terms that don't co-occur in the sample. They will look completely dissimilar. However, those two terms might co-occur in a document outside the sample. Additionally, think about the size of these vectors. They grow with the number of documents in the sample. Consequently, researchers wanted a lower-dimensional, higher-quality set of vectors. So what they did was to use a technique from linear algebra called **matrix factorization**



irstly, matrix factorization takes a matrix like the term-document matrix and creates two matrices called factors that can be treated as lower dimensional representations of the two domains, which in this case are terms and documents.

Multiplying these two smaller matrices results in an approximation of the original matrix. Secondly, matrix factorization is useful in a variety of machine learning scenarios. We'll use matrix factorization again to build our recommendation systems in the next course.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Term to Term matrix

Later, researchers would change this approach so that instead of a term-document matrix, they created a term-term matrix, where every value corresponding to the number of times the two words co-occured. They constructed these matrices by sliding a window over a corpus and treating words in that window at the same time as co-occurring. This was consistent with the idea that the context necessary for understanding a word was located in its immediate surroundings more so than in the document in which it occurs.

In the last section, we introduced matrix factorization as a way to construct embeddings directly from statistics.

A picture containing animal

Description automatically generated

What they did was to train a model on a task that required an understanding of the domain, and then treated the first layer of the model as the embedding, in effect, using transfer learning. One recent and influential example of this approach is called Word2Vec.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Instead, these approaches used the contents of the sliding window to transform the sequence of words in the corpus into features and labels for their machine learning task.

However, unlike what we did in module one, where we where we use the final event in the window as the label, these researchers use the word at the center of the window as the feature, and its surrounding context as the label.

We call these words that surround the central word, the positive words for a particular example, and the remaining words in the corpus as negative words. The model's task is to maximize the likelihood of positive words and minimize the likelihood of negative words.

Positive Words and Negative Words:

A close up of a map

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

What the authors of Word2Vec realize, is that because of the size of the context window relative to the size of the vocabulary, the vast majority of words will be negative for a given training example. Their idea was to compute the softmax using all the positive words and a random sample of the negative ones, and that's where this technique got its name.

A close up of a map

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Part of the reason that Word2Vec is become so widely known, is because the embeddings that produced exhibited semantic compositionality.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

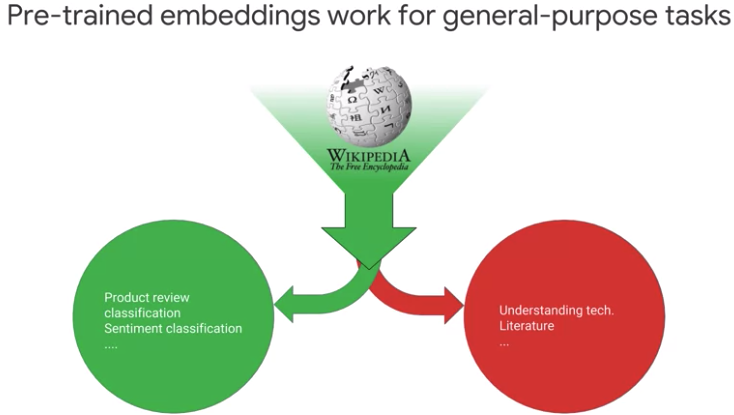
Description automatically generated

A screenshot of a cell phone

Description automatically generated

A close up of text on a white background

Description automatically generated



A picture containing text

Description automatically generated

Tensorflow hub:

A screenshot of text

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Encoder and Decoder Models:

A close up of a piece of paper

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A close up of a device

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

A close up of a device

Description automatically generated