Chpt4 Real-world Data Representations Using Tensors

Augest 9, 2021

Often, our raw data won't be perfectly formed for the problem we'd like to solve. Each section in this chapter will describe a data type, and each will come with its own dataset. We'll also cover tabular data, time series, and text. Starting with image data, we'll then demonstrate working with a three-dimensional array using medical data that represents patient anatomy as a volume.

In every section, we will stop where a deep learning researcher would start: right before feeding the data to model. We encourage you to keep these datasets; they will constitute excellent material for when we start learning how to train neural network models in next chapter.

4.1 Working with images

An image is represented as a collection of scalars arranged in a regular grid with a height and width. We might have a single scalar per gird point, which would be represented as a grayscale image; or multiple scalars per grid point, which would typically represent different colors, or different *features* like depth from a depth camera.

4.3 Representing tabular data

Continuous, ordinal, and categorical values:

We should be aware of three different kinds of numerical values as we attempt to make scence of our data. The first kind is *continuous* values. If you are counting or measuring something with units, it's probably a continuous value.

Next, we have ordinal data. The strict ordering we have with continuous values remains, but the fixed relationship between values no long applies.

Finally, categoraical values have neither ordering nor numerical meaning to their values. These are oftern just enumerations of possibilities assigned arbitrary numbers. Because the numerical values bear no meaning, they are said to be on a *nominal* scale.

4.3.5 When to categorize

We summarize our data mapping in a small flow chart in the following:

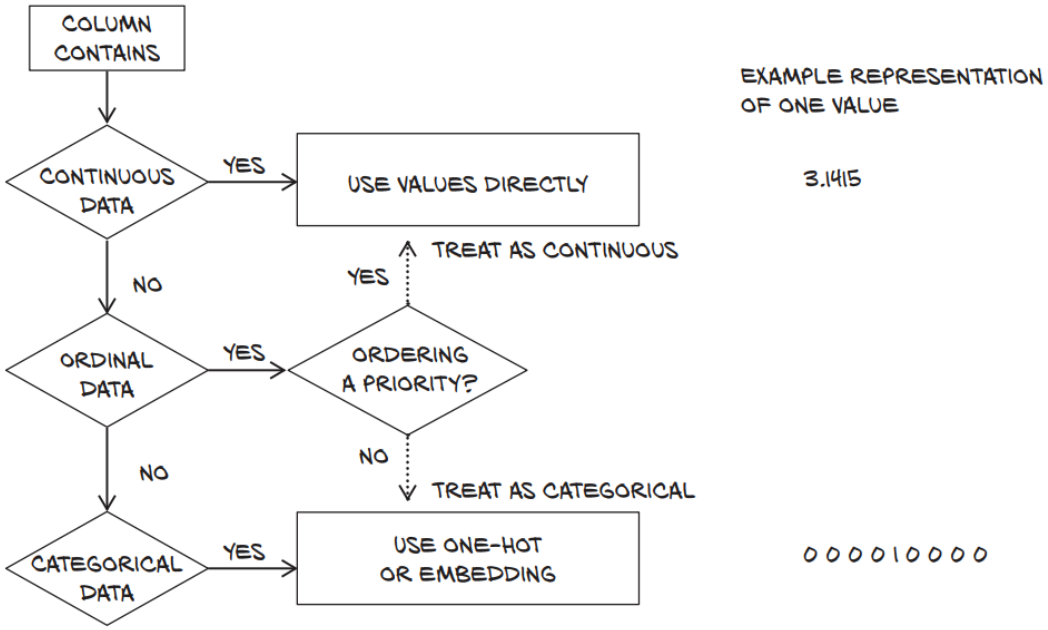

Figure 4.4 How to treat columns with continuous, ordinal, and categorical data

4.4 Working with timeseries

4.4.2 Shaping the data by time period

We might want to break up the two-year dataset into wider observation periods, like days. This way we'll have **N** *(for numbers of samples)* collections of **C** sequences of length **L**. The **C** would remain our 17 channels, while **L** would be 24: 1 per hour of the day. There's no particular reason why we must use chunks of 24 hours.

4.5 Representing text

Recurrent neural networks have been applied with great success to text categorization, text generation, and automated translation system.

Our goal in this section is to turn text into something a neural network can process: a tensor of numbers. We can achieve state-of-art performance on a number of tasks in different domains *with the same PyTorch tools*; we just need to cast our problem in the right form. The first part of this job is reshaping the data.

Given the stark contrast between these two options, it is perhaps unsurprising that intermediate ways have been sought, found, and applied with great success: for example, the *byte pair encoding* method starts with a dictionary of individual letters but then iteratively adds the most frequently observed pairs to the dictionary until it reaches a prescribed dictionary size

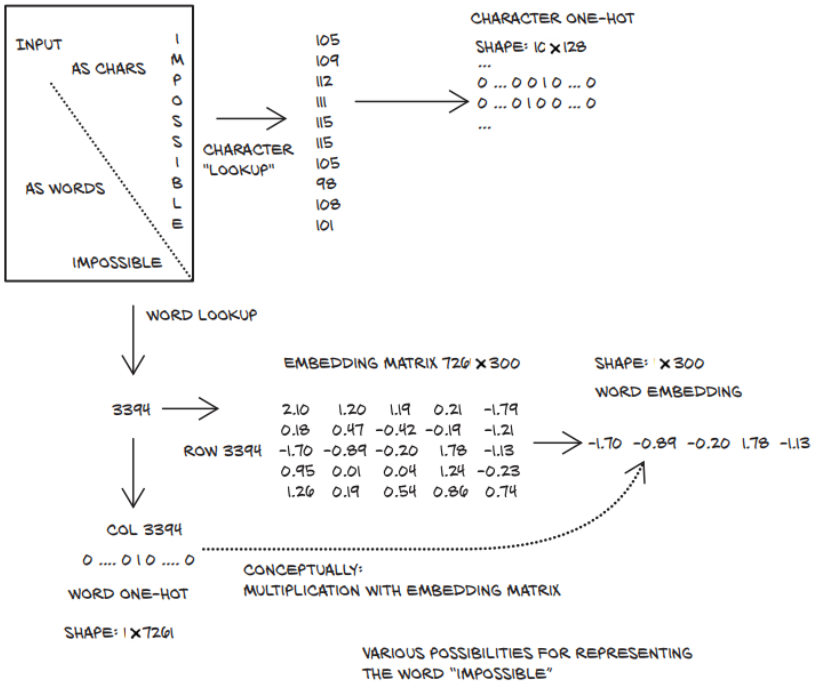

Figure 4.6 Three ways to encode a word

4.5.4 Text embeddings

How can we compress our encoding down to a more managebale size and put a cap on the size growth? Well, instead of vectors of many zeros and a single one, we can use vectors of floating-point numbers. A vector of say, 100 floating-point numbers can indeed represent a large number of words. The trick is to find an effective way to map individual words into this 100-dimensional space in a way that facilitates downstream learning. This is called an *embedding*.

An ideal solution would be to generate the embedding in such a way that words used in similar contexts mapped to nearby regions of the embedding. We'd just like to mention that embeddings are often generated using neural networks, trying to predict a word from nearby words in a sentence. In this case, we could start from one-hot-encoded words and use a nn to generate embedding. Once the embedding was available, we could use it for downstream tasks.

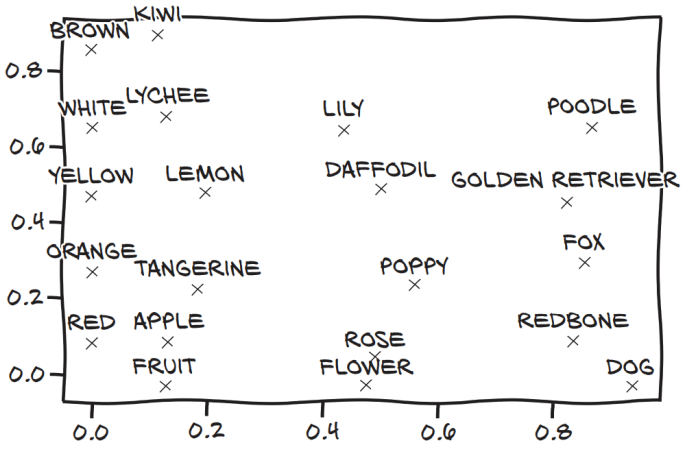

Figure 4.7 Our manual word embeddings

Similar words end up not only clustered together, but also having consistent spatial relationships with other words. For example, if we were to take the embedding vector for *apple* and begin to add and subtract the vectors for other words, we could begin to perform analogies like *apple - red - sweet + yellow + sour* and end up with a vector very similar to the one for *lemon*.

4.6 Conclusion

Now that we're familiar with tensors and how to store data in them, we can move on to the next step towards the goal of the book: teaching us to train neural networks! The next chapter covers the mechanics of learning for simple linear models.

4.8 Summary

1. NN require data to be represented as multidimensional numerical tensors, often 32-bit floating-point.

2. Coverting spreadsheets to tensors can be very straightforward. Categorical and ordinal-valued columns should be handled differently from interval-valued columns.

3. Text or categorical data can be encoded to a one-hot representation through the use of dictionaries. Very ofter, embeddings give good, efficient representations.