Converting Documents to Vectors

Computers store words as a series of characters with each character represented as a number. This representation from number to character or character to number is called **character encoding**. The most popular character encoding is the **ASCII** character encoding.

A	65
...	...
Z	90

a	97
...	...
z	122

ASCII A-Z and a-z

In ASCII, an uppercase "A" is represented as 65 and a uppercase "Z" is represented as 90. All other uppercase english characters are represented between 65 and 90. Lowercase letters are represented in the range of 97 to 122.

Just like a computer, a neural network can't understand words. It only sees a series of numbers. Feeding a neural network these series of numbers would require it to learn the english vocabulary. This requires more data than we have. Instead, let's turn the words into something easer for a neural network to learn from.

Just like ASCII encodes a character as a number, you can encode a word as a number. For example, the text `"my bat is the best bat"` could be changed to `[1, 2, 3, 4, 5, 2]`. Each word has been converted to a number where the number assigned to a word is arbitrary.
It's only required that each number corresponds to a single word and vice versa. This is known as a one to one function. In our case, this mapping is:

- `1 <-> my`
- `2 <-> bat`
- etc.

Word2vec

Representing a word as a number is great, but the neural network still has to learn the meaning of the word. Instead of representing a word as a single number, what about using multiple numbers? Each number can represent an abstract feature of the word. Words that are similar in meaning will have values close together. This representation is called **word embeddings**. It's words or phrases that are mapped to vectors.

One of the most popular methods for creating these word embeddings is **Word2vec**. Word2vec is a neural network model that trains on text to create embeddings.

There are two flavors for Word2Vec, continuous bag of words (CBOW) and Skip grams. We'll focus on Skip grams, but a lot of the stuff covered is applicable to CBOW. The skip grams model is a neural network that takes in a word and tries to predict the *n* number of surrounding words. Say you have the following string:

```
the quick brown fox jumps over the lazy dog
```

The input to the neural network will be "jump" and it will try to predict *n* surrounding words, one at a time. For example, if *n* was 2, it would try to predict on "fox" and "over". As this neural network trains on these words, it will learn abstract meanings for the words. Let's say we have the following vectors after training:

	Word Vector	Species	Age
Cat	$\begin{bmatrix}0.13\\0.70\end{bmatrix}$	0.13	0.70
Kitten	$\begin{bmatrix}0.13\\0.01\end{bmatrix}$	0.13	0.01
Dog	$\begin{bmatrix}0.87\\0.70\end{bmatrix}$	0.87	0.70
Puppy	$\begin{bmatrix}0.87\\0.01\end{bmatrix}$	0.87	0.01

Each dimension is an abstract meaning that the neural network has learned from the dataset. We're using a 2 size vector for simplicity, but most vectors are the size of 25 or higher. The word vectors for "Cat" and "Dog" have a larger number for age than "Kitten" and "Puppy". This number isn't a prediction on age, but the relationship between words. Words that mean a similar age have a value closer together.

Linear Relations

Since similar words are also in similar context, linear relations are formed between word vectors as you saw above. This property allows us to answer analogy questions by using vector math with the word vectors. For example, what is an old kitten?

$$\underbrace{\begin{bmatrix}0.87\\0.70\end{bmatrix}}_{\text{Dog}} - \underbrace{\begin{bmatrix}0.87\\0.01\end{bmatrix}}_{\text{Puppy}} + \underbrace{\begin{bmatrix}0.13\\0.01\end{bmatrix}}_{\text{Kitten}} = \underbrace{\begin{bmatrix}0.13\\0.70\end{bmatrix}}_{\text{Cat}}$$

We can answer this question by adding the number that represents old in the first element to the "kitten" word vector. To get this number, let's subtract everything that has to do with the species dog (**Canis lupus familiaris**) from the "Dog" word vector. This can be done by subtracting the "Puppy" word vector from the "Dog" word vector, which gives the following vector:

$$\begin{bmatrix}0.00\\0.69\end{bmatrix}$$

Adding this vector to kitten will give the final result, which is the word vector for "Cat":

$$\begin{bmatrix}0.13\\0.70\end{bmatrix}$$

This all seams really great, but this isn't the end point. This is a representation for words that you'll feed into a neural network to solve more complex problems. In the next section, you'll learn about a model commonly used for NLP problems, Long Short-Term memory networks.