1. Embeddings

* Dealing with text:
* Text is also sequence data >< not continuous
* X(t)’s are not numerical
* Can’t do regular RNN
* One-hot encoding?
* Create an array of size equal to the number of words in the English language
* Map each word to an index
* A sequence of T words, each word get a one-hot encoded vector of size V
* One sentence – T x V matrix
* Same generic shape as a T x D matrix
* Problem:
* Some English-language datasets have ~ 1 million possible tokens / words
* Large input and large input-hidden weight matrix
* We want our input features to have some structure
  + ML is nothing but a geometry problem
  + This rule relies on the fact that there’s some geometrical structure in the data – that data vectors in the same class are probably near each other
* Problem with one-hot encoding:
* Take any 2 one-hot encoded vectors
* Euclidean distance is sqrt(2)
* All words are an equal distance apart – useless
* Better solution: Embeddings
* Assign each word to a D-dimensional vector (not one-hot encoded)
* How are these vectors found?
* Coding trick: Suppose we multiple a one-hot encoded vector with a matrix
* One-hot encoding an integer k and multiplying that by a matrix is the same as selecting the k’th row of the matrix
* Index the weight matrix (W[k]) – more efficient
* This is exactly what the Embedding layer does
* Tensorflow embedding
* Step 1: convert words into integers
  + [I, like, cats] -> [50, 25, 3]
* Step 2: use integers to index the word embedding matrix to get word vectors for each word
  + [50, 25, 3] -> [[0.3, -0.5], [1.2, -0.7], [-2.1, 0.9]]
  + T length array -> T x D matrix
* How are the weights found?
* We know intuitively that the word vectors must have some useful structure
* Same story as with convolutional filters
* These weights are found automatically with gradient descent when you call model.fit()
* Pre-trained word vectors
* Sometimes, we use pre-trained word vectors (trained using some other algorithm)
* We freeze (fix) the embedding layer’s weights, so only the other layers are trained when we call model.fit()



* Extra reading: word2vec, GloVe
* Summary:
* It’s not possible to multiple a word by a weight matrix, there is no concept of multiplying words
* We must turn words into some kind of numerical representation first
* One-hot encoding – fine first guess
  + Too much space
  + Not geometrically useful (cat is just as close to feline as to airplane)
* Better: convert words to unique integers
* Use those integers to index a weight matrix (embedding matrix)
* Much faster than matrix multiplication
* Each input sentence thus becomes a T x D sequence which we know an RNN can accept
* The embedding is trained like any other layer -> just need to call model.fit()
* Alternative methods for training embedding: word2vec, GloVe

1. Code preparation