Dealing with Texts

- Words are categorical objects (non-continuous)

Suppose: ["the", "quick", "brown"]

x_1 = "the"

x_2 = "quick" ← x_t not numerical

$$h_t = \sigma\left(W_{xh}^T x_t + W_{hh}^T h_{t-1} + B_h\right)$$

Problem with One-hot encoding:

- Need to create Array representing all english words
  - ↳ All values 0 except for 1.
- Input = T x V ← size of array containing all Eng words (> 200 000 words)
  - T ↑ No. words
- All 1-hot encoded words have euclidean distance of $\sqrt{2}$
  - ↳ due to '1'

Solution: Embedding Layer

- represent words, tokens, textual elements as dense vectors (opposite is sparse vectors (most elements are 0))
- Vector is low-dimensional (↳ most elements are not 0)
  - ↳ instead of 10,000 for size 10,000 vocab
  - ↳ size 50, 100, 300 instead.

Step 1: Convert words to integers

["I", "Like", "Cats"] → [50, 25, 3] (represent index for Embedding matrix (between 0 & vocab-size) : E[50], E[25], E[3]

Step 2: Integers represent the index to retrieve corresponding dense vector from Embedding Matrix

Embedding Matrix E = (vocab-size x embedding_dim)

- vocab-size: unique tokens, eg: 10 000
- embedding_dim: no. of dim, eg: 50

E = [
[0.11, -0.34 ... x_{50}], ← row 0 (50 dim.)
⋮
[0.45, 0.67 ... x_{50}], ← row 9999
]

Step 3: Receive output

↳ shape = (T x D)

- T ↑ No. of words
- D ↑ Dim of embedding matrix

Step 4: Training Model

- Dense vector can be used to train model
- During training, words are positioned with similar meanings together.
  - i.e "king" & "queen" vector values will be very close.

Pretrained Word vectors

- precomputed embedded matrix trained on large text corpus
- Embedding layer weights fixed, only other layers are trained
- reduces training time & enhances accuracy (esp. if data is insufficient)

* index to map words to vectors should not be '0'

'0' is used for padding

Tensor Flow handling of text

Document → token

1) Break down array of strings into single word strings
   - ↳ known as Tokenization
   - ↳ Each word in array is known as token.
   - ↳ Tensor Flow Tokenizer
     - ↳1) Break down string into tokens
     - 2) convert list of tokens to list of integers.

* Tokenizer (num_words = K)

↑ only keep K most impt words.

the rest assigned to <unknown> token.

2) Padding
   - L sequence of integers have different lengths (due to different sentence length)
   - L RNN do not accept different sequence length.

pad_sequences() method.

padding = "pre" | "post"

L depending on task, padding location matters.

Spam detection - want padding at start - can forget info.

language translation - want padding at end - need info at start to translate.

Data = N × T

↑ No. sentence; ↖ sentence length. (padded)

3) Embedding Layer.
   - N × T data passed through embedding layer to get corresponding dense vector
   - Result is N × T × D

↑ No. sentences (Samples); ↑ sentence length — Each sample have T indices; ↖ vector shape. — Each index have D sized vector