Issues with text: Sparse encoding of tokens

A token is an element of the finite set $\mathbf{V}$

- It is thus a *categorical* variable
- Which is naturally represented by a vector of length $||\mathbf{V}||$
    - One Hot Encoded (OHE)

So, in theory, we already know how to perform NLP

- Encode text as a sequence of OHE vectors
- Apply techniques (e.g., RNN) specialized to sequences

This approach may be both

- Impractical
- Sub-optimal: not taking advantage of properties inherent in text

Let us suppose that we had a function that maps a token to a OHE vector

$$\mathrm{rep} : \mathrm{token} \mapsto \mathbb{R}^{n_{\mathbf{V}}}$$

Thus

$$\mathrm{rep}(\mathbf{w}_{(t)})$$

is the OHE vector for $\mathbf{w}_{(t)} \in \mathbf{V}$

The first issue we encounter:

- $\mathbf{V}$ is big ! A decent vocabulary is easily thousands of token
- So the OHE is a *long* vector ($||\mathbf{V}||$)

Thus, OHE may not be *practical*

It is also potentially failing to take advantage of relationships between tokens

- There is clearly a relationship between tokens
$$\mathrm{dog}, \mathrm{dogs}$$
but *no* relationship between their OHE vectors
$$\mathrm{rep}(\mathrm{dog}), \mathrm{rep}(\mathrm{dogs})$$

OHE vectors are *sparse*

- All zero
- Except for a single element

The sparsity both wastes space and is the cause for not capturing potential relationships between words

We will subsequently demonstrate a *dense* encoding of tokens that solves both issues.

Issues with text: variable length sequences

Another issue with text: the sequence length is variable, not constant.

The reason this may be an issue

- Classical Machine Learning models (e.g. Logistic Regression) can only deal with fixed length inputs
- The final layer in a Neural Network is usually an implementation of a Classical model

Fortunately, we can easily identify two solutions to this issue.

Both involve reducing a variable length sequence to a fixed length encoding.

Once tokens have been encoded as a sequence of numeric vectors the solutions are

- Replace the variable number of tokens by a summary statistic (sum/average) over the tokens
- Use the final state of an RNN as an encoding of the sequence

As a notational convenience we will extend rep to sequences $\mathbf{w}$:

$$\mathrm{rep}(\mathbf{w}) = \left[\mathrm{rep}(\mathbf{w}_{(t)}) | 1 \leq t \leq ||\mathbf{w}||\right]$$

Traditional methods for summarizing a variable length sequence

The simplest way to derive a fixed length encoding of a sequence is by a summarization operation.

This is the approach that was pursued in "traditional" (pre-Neural Network) NLP.

Recall what a Global Pooling layer does

- Each row is a feature over multiple spatial/temporal locations (tokens in the sequence)
- Is transformed to a single value
- Preserving the feature/channel dimension

Global Pooling
3 features over spatial locations
to 3 features over one location

Unfortunately, the summarization (sum/average) of a sequence

- Will lose the ordering relationship among the tokens

Consider the summarization across single tokens of the two sequences

"Machine Learning is easy not hard"

"Machine Learning is hard not easy"

both have the same summary.

We will give a brief introduction to traditional summarization techniques.

Bag of Words (BOW): Pooling

We define a *reduction* operation CBOW

- convert a sequence $\mathbf{w}$ of $||\mathbf{w}||$ elements
- each element of length $||\mathrm{rep}(\mathbf{w}_{(t)})||$
- to a fixed length vector of length $||\mathrm{rep}(\mathbf{w}_{(t)})||$

This will necessarily lose token order: this method is called *Bag of Words (BOW)*

There are many operators to achieve the reduction, which we will group under the name *pooling*

Sum/Average

$$\text{CBOW}(\mathbf{w}) = \sum_{t=1}^{||\mathbf{w}||} \text{rep}(\mathbf{w}_{(t)})$$

Since $\mathbf{w}_{(t)}$ is a vector, the addition operation is element-wise.

So the composite vector for the sequence is the sum of the vectors of each element in the sequence.

We can easily turn the Sum into an average by dividing by $||\mathbf{w}||$

Count vectorization:

In the special case that

$$\text{rep}(\mathbf{w}_{(t)}) = \text{OHE}(\mathbf{w}_{(t)})$$

$\text{CBOW}(\mathbf{w})_j$ is equal to the number of occurrences in sequence $\mathbf{w}$ of the j^{th} word in $\mathbf{V}$.

This is often called *Count Vectorization*.

TF-IDF

Count Vectorization is simple but ignores a basic fact or language

- Word "importance" is often inversely correlated with frequency in $\mathbf{V}$

In English:

- The words "a", "the" and "is" are extremely high frequency (so high counts in most sequences $\mathbf{w}$).
- But are so common as to convey little meaning

On the other hand, a rare word (or sequence of words) may be very distinctive ("Machine Learning").

Term Frequency, Inverse Document Frequency (TF-IDF)

- Is based on the idea that a word that is *infrequent* in the wide corpus (collection of text)
- But is frequent in a particular document (one text in the collection) in the corpus is very meaningful in the context of the document.

So a document

- In which "Machine Learning" occurred a disproportionately high (relative to the broad corpus) number of times
- Is likely to indicate that the document is dealing with the subject of Machine Learning.

Note A similar idea is behind many Web search algorithms (Google).

TF-IDF is similar to the Count Vectorizer, but with modified counts that are the product of

- The frequency of a token within a single document
- The inverse of the frequency of the token relative to all documents
- v is a token
- d is a document (collection of tokens) in set of documents D

$$
\begin{aligned}
\text{tf}(v, d) &= \text{frequency of word } v \text{ in document } d && \text{(Term Frequenc} \\
\text{df}(v) &= \text{number of documents that contain word } v \\
\text{idf}(v) &= \log(\frac{||D||}{\text{df}(v)}) + 1 && \text{Inverse Docume} \\
\\
\text{tf-idf}(v, d) &= \text{tf}(v, d) * \text{idf}(v)
\end{aligned}
$$

Thus, $\text{tf-idf}(v, d)$ is high (token v is *important* to document d):

- For tokens v that occur infrequently in the corpus (resulting in high inverse: $\text{idf}(v)$)
- But which occur frequently in a particular document d: $\text{tf}(v, d)$

Using an RNN to obtain a fixed length encoding of a variable length sequence

As a refresher, here is our picture of the RNN API:

RNN many to one API

Although we don't know exactly what $\mathbf{h}_{(t)}$ represents, recall that

- It *is* a summary of the prefix of input $\mathbf{x}$ ending at step t

Thus $\mathbf{h}_{(T)}$ is a summary of sequence $\left[\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(T)}\right]$

It will be typical to see a Neural Network for an NLP task having an architecture

- That uses an RNN to process input $\mathbf{x}$
- Followed by other Layers
- **Culminating in "head" layer L implementing a Classical Layer (e.g., Logistic Regression)**

Conclusion

Text data is characterized by

- Tokens that are categorical variables
- Variable length sequences of tokens

We presented some "obvious" methods to deal with both issues, but they clearly have limitations.

We will move beyond the limitations in a subsequent module.

In [2]:
```
print("Done")
```

```
Done
```