# Loading and Preprocessing Big Data

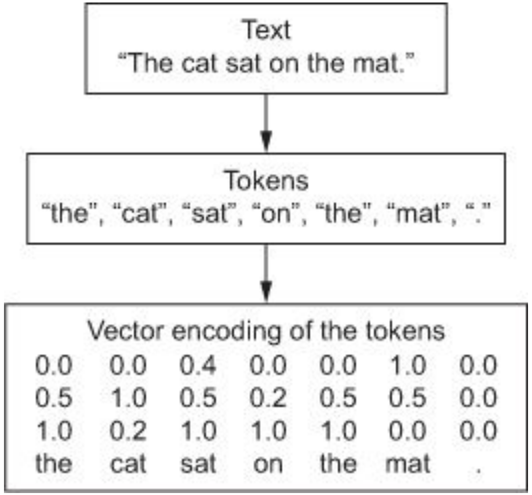
## Encoding Categorical and Textual Features

N. Rich Nguyen, PhD

**SYS 6016**

# Working with text data

- Text is the most widespread form of data
- Text can be processed as a sequence of characters, or a sequence of words
- **Neural nets do not take input as raw text, instead they takes the numeric vectors**
- Vectorizing text can be done is multiple ways:
    - Segment text into words, and transform each word into a vector
    - Segment text into characters, and transform each character into a vector
    - Extract **n-grams,** overlapping groups of multiple consecutive words, and transform each into a vector.



# Encode Categorical Features

- Consider the `ocean_proximity` feature in the California housing dataset: it is a categorical feature with five possible values: “<1H OCEAN”, “INLAND”, “NEAR OCEAN”, “NEAR BAY”, and “ISLAND”.
- If there’s a small number of categories, we can use **one-hot encoding**.
- First, we map each category to its index (0 to 4) using a lookup table
- We also use **out-of-vocabulary buckets** to assign the unknown categories

```
vocab = ["<1H OCEAN", "INLAND", "NEAR OCEAN", "NEAR BAY", "ISLAND"]
indices = tf.range(len(vocab), dtype=tf.int64)
table_init = tf.lookup.KeyValueTensorInitializer(vocab, indices)
num_oov_buckets = 2  # number of out-of-vocabulary buckets
table = tf.lookup.StaticVocabularyTable(table_init, num_oov_buckets)
```

# Encode categorical features to one-hot vectors

- The more unknown categories we expect during training, the more oov buckets we should use.
- Second, we use `tf.one_hot()` to convert lookup index into one-hot vectors

```
>>> categories = tf.constant(["NEAR BAY", "DESERT", "INLAND", "INLAND"])
>>> cat_indices = table.lookup(categories)
>>> cat_indices
<tf.Tensor: id=514, shape=(4,), dtype=int64, numpy=array([3, 5, 1, 1])>
>>> cat_one_hot = tf.one_hot(cat_indices, depth=len(vocab) + num_oov_buckets)
>>> cat_one_hot
<tf.Tensor: id=524, shape=(4, 7), dtype=float32, numpy=
array([[0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.]], dtype=float32)>
```

`keras.layers.TextVectorization` layer can extract the vocabulary from a data example, convert each category to its index, and convert them into one-hot vectors

# Limitations of One-hot Encoding

- Vocabulary $\mathbf{v}$ = [a, aaron, … , zulu, <UNK>] $|\mathbf{v}| = \sim 10{,}000$
- Size of each one-hot vector is the vocabulary length plus oov → **not efficient**
- Also, the order of the categories (alphabetical) has **no intrinsic meaning**.

$$\begin{array}{cccccc}
\text{Man} & \text{Woman} & \text{King} & \text{Queen} & \text{Apple} & \text{Orange} \\
(5391) & (9853) & (4914) & (7157) & (456) & (6257) \\
\begin{bmatrix}0\\0\\0\\0\\\vdots\\1\\\vdots\\0\\0\end{bmatrix} &
\begin{bmatrix}0\\0\\0\\0\\0\\\vdots\\1\\\vdots\\0\end{bmatrix} &
\begin{bmatrix}0\\0\\0\\\vdots\\1\\\vdots\\0\\0\\0\end{bmatrix} &
\begin{bmatrix}0\\0\\0\\0\\0\\\vdots\\1\\\vdots\\0\end{bmatrix} &
\begin{bmatrix}0\\\vdots\\1\\\vdots\\0\\0\\0\\0\\0\end{bmatrix} &
\begin{bmatrix}0\\0\\0\\0\\0\\\vdots\\1\\\vdots\\0\end{bmatrix}
\end{array}$$

I want a glass of orange juice.
I want a glass of apple ______?

There is **no semantic relationship** between apple and orange here.



Source: https://www.coursera.org/learn/nlp-sequence-models/lecture/6Oq70/word-representation

# Word Embedding

Each category is “embedded” into a vectorized feature space.

256 features

| | Man (5391) | Woman (9853) | King (4914) | Queen (7157) | Apple (456) | Orange (6257) |
|---|---|---|---|---|---|---|
| **Gender** | -1 | 1 | -0.95 | 0.97 | 0.00 | 0.01 |
| **Royal** | 0.01 | 0.02 | 0.93 | 0.95 | -0.01 | 0.00 |
| **Age** | 0.03 | 0.02 | 0.7 | 0.69 | 0.03 | -0.02 |
| **Food** | 0.04 | 0.01 | 0.02 | 0.01 | 0.95 | 0.97 |
| **Size & more….** | ... | ... | ... | ... | ... | ... |

**a 256 dimensional vector to denote each category**

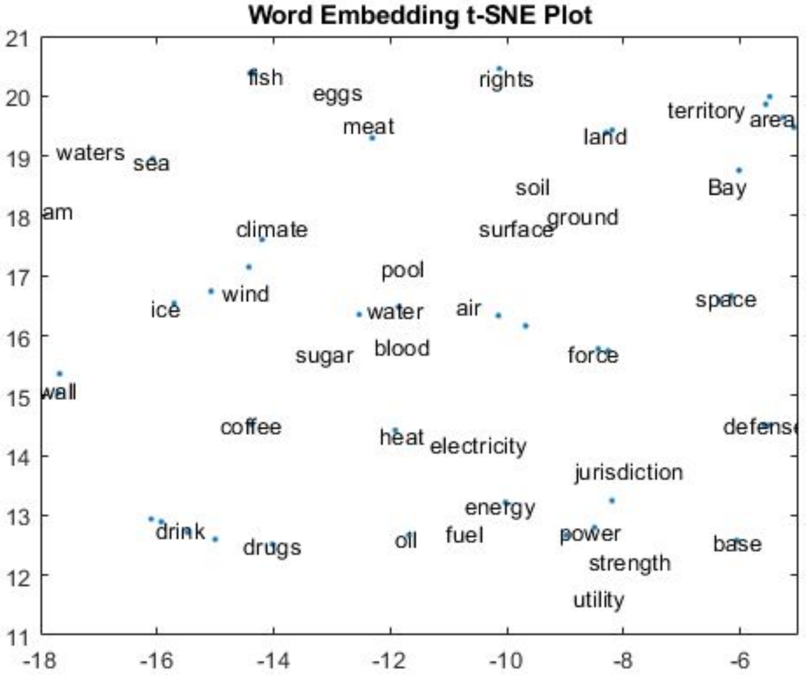
**now they are similar in this embedded space**



Source: https://www.coursera.org/learn/nlp-sequence-models/lecture/6Oq70/word-representation

# Visualizing word embeddings

Project 256 dimensional space into 2-d space → t-SNE (t-distributed stochastic neighbor embeddings), a visualization tool by Maaten and Hinton

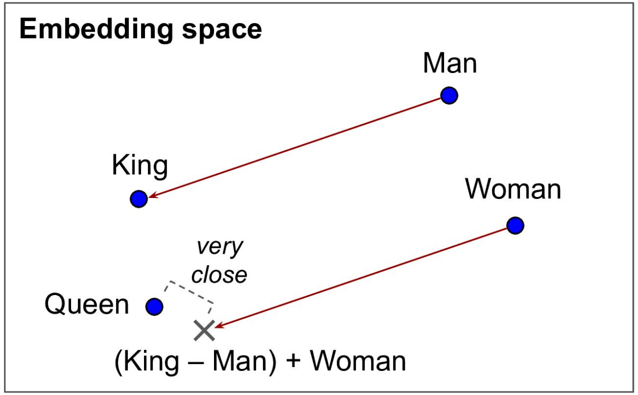



Source: https://www.mathworks.com/help/textanalytics/ug/visualize-word-embedding-using-text-scatter-plot.html

# Properties of a Word Embedding

- In a word embedding space, the geometric relationship between word vectors should reflect the **semantic relationship** between these words (ie. synonyms “accurate” and “exact” should be embedded into similar vectors)
- We can use **vector arithmetic** to map from one category to another

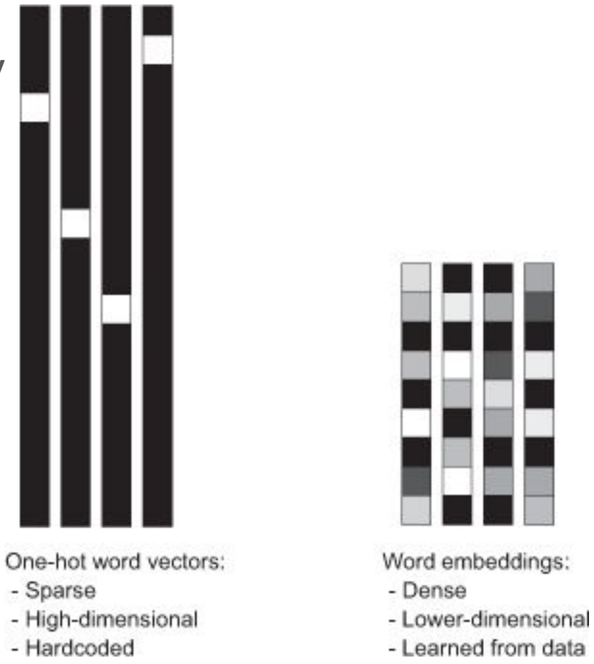
| | Man (5391) | Woman (9853) | King (4914) | Queen (7157) |
|---|---|---|---|---|
| Gender | −1 | 1 | -0.95 | 0.97 |
| Royal | 0.01 | 0.02 | 0.93 | 0.95 |
| Age | 0.03 | 0.02 | 0.70 | 0.69 |
| Food | 0.09 | 0.01 | 0.02 | 0.01 |




- (“tiger” - “cat”) + “dog” ≈ “wolf” → “**feline to canine**” vector
- (“wolf” - “tiger”) + “cat” ≈ “dog” → “**wild animal to pet**” vector

# Word Embeddings vs. One-Hot Encoding

- One-Hot Encoding creates sparse (mostly made of zeros), high dimensional vectors (same dimensionality as the number of words in the vocabulary).
- Word Embeddings are low-dimensional (can be 128, 256, 512 or 1024 dimensions per application) floating points which can be learned from data → they pack more information into far fewer dimensions.
- Two ways to obtain word embeddings:
  - Learn them jointly with the main task
  - Use a pre-trained embedding



# Learning embeddings jointly with the task

- We can implement embedding using `keras.layers.Embedding` layer
- Basically, we create a Keras model that can process categorical features along with numerical ones and learn an embedded weight vector for each category
- Thus, embedded vectors will be learned **in the same way** as regular features (ie. optimized using gradient descent)
- Below we use 2D embeddings, but as a rule of thumb embeddings should have 16 to 512 dimensions, depending on the task and the vocabulary size

```
regular_inputs = keras.layers.Input(shape=[8])
categories = keras.layers.Input(shape=[], dtype=tf.string)
cat_indices = keras.layers.Lambda(lambda cats: table.lookup(cats))(categories)
cat_embed = keras.layers.Embedding(input_dim=6, output_dim=2)(cat_indices)
encoded_inputs = keras.layers.concatenate([regular_inputs, cat_embed])
outputs = keras.layers.Dense(1)(encoded_inputs)
model = keras.models.Model(inputs=[regular_inputs, categories],
                           outputs=[outputs])
```

# Using Pre-trained Embeddings

- Most of the time, you have so little training data available to use it alone to learn an appropriate embedding of your vocabulary → load **embedding vectors** from a powerful pre-trained space
- Most famous and successful word-embedding schemes: **Word2vec** algorithm developed by **Tomas Mikolov** at **Google** in 2013 and trained on a large corpus on Google News
- Another popular one is called **Global Vector for Word Representation** (GloVe) developed by **Stanford** researchers in 2014. GloVe trained on data obtained on Wikipedia and Common Crawl

# Extracting Word Embedding

$$\underbrace{\begin{pmatrix} 0.8 \\ 1.0 \\ 4.2 \\ 7.5 \\ 3.6 \end{pmatrix}}_{\text{Word Embeddings}} = \text{Embedding Size}\ \underbrace{\begin{pmatrix} 1.5 & \cdot & \cdot & \cdot & \cdot & 0.8 & \cdot & \cdot & 2.0 \\ 1.4 & \cdot & \cdot & \cdot & \cdot & 1.0 & \cdot & \cdot & 3.6 \\ 7.8 & \cdot & \cdot & \cdot & \cdot & 4.2 & \cdot & \cdot & 5.4 \\ 5.6 & \cdot & \cdot & \cdot & \cdot & 7.5 & \cdot & \cdot & 8.5 \\ 7.8 & \cdot & \cdot & \cdot & \cdot & 3.6 & \cdot & \cdot & 9.7 \end{pmatrix}}_{\text{Vocabulary Size}} \cdot \underbrace{\begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ 1 \\ \cdot \\ 0 \end{pmatrix}}_{\text{One-hot Vector}}$$

In practice, we use a specialized function to lookup an embedding column

Keras provides a `keras.layers.Embedding` layer that handles embedding matrix

# Addressing Bias* in Word Embeddings

- Unfortunately, word embeddings sometime capture our worst biases.
- For example, although they correctly learn that Man is to King as Woman is to Queen, they also seem to learn that Man is to Doctor as Woman is to Nurse → **quite a sexist bias!**
- Word embedding can reflect gender, ethnicity, age, sexual orientation, and other **biases from the text (written by people) used to train the model**

| Extreme *she* | Extreme *he* |
|---|---|
| 1. homemaker | 1. maestro |
| 2. nurse | 2. skipper |
| 3. receptionist | 3. protege |
| 4. librarian | 4. philosopher |
| 5. socialite | 5. captain |
| 6. hairdresser | 6. architect |
| 7. nanny | 7. financier |
| 8. bookkeeper | 8. warrior |
| 9. stylist | 9. broadcaster |
| 10. housekeeper | 10. magician |

| | **Gender stereotype *she-he* analogies** | |
|---|---|---|
| sewing-carpentry | registered nurse-physician | housewife-shopkeeper |
| nurse-surgeon | interior designer-architect | softball-baseball |
| blond-burly | feminism-conservatism | cosmetics-pharmaceuticals |
| giggle-chuckle | vocalist-guitarist | petite-lanky |
| sassy-snappy | diva-superstar | charming-affable |
| volleyball-football | cupcakes-pizzas | lovely-brilliant |

| | **Gender appropriate *she-he* analogies** | |
|---|---|---|
| queen-king | sister-brother | mother-father |
| waitress-waiter | ovarian cancer-prostate cancer | convent-monastery |



*The bias is as prejudice in favor of or against one thing, person, or group compared with another, not the bias in the context of bias and variance.

# Word Neutralization

An important and active research topic

A paper from Bolukbasi and other in 2016

1. **Identify** bias: averaging to find axis
    a. He - She
    b. Man - Woman
    c. Male - Female
2. **Neutralize**: For every word that is not definitional, **project** to get rid of bias.
3. **Equalize:** For gender appropriate pairs, correct to ensure **equal** distance to axis
    a. Mommy -vs- Daddy
    b. Daughter -vs- Son
    c. Sister -vs- Brother




[Bolukbasi et. al, 2016. Man to computer programmer as woman is to homemaker? Debiasing word embeddings]

# Summary

- When you are working with categorical or textual features, your are often better off using pre-trained word embeddings than training your own.
- With the Data API, TFRecord, TFDS, and word embeddings, you can build highly scalable input pipelines for training. You also benefit from fast and portable data preprocessing in production.
- Ensuring fairness in ML is an important research topic.