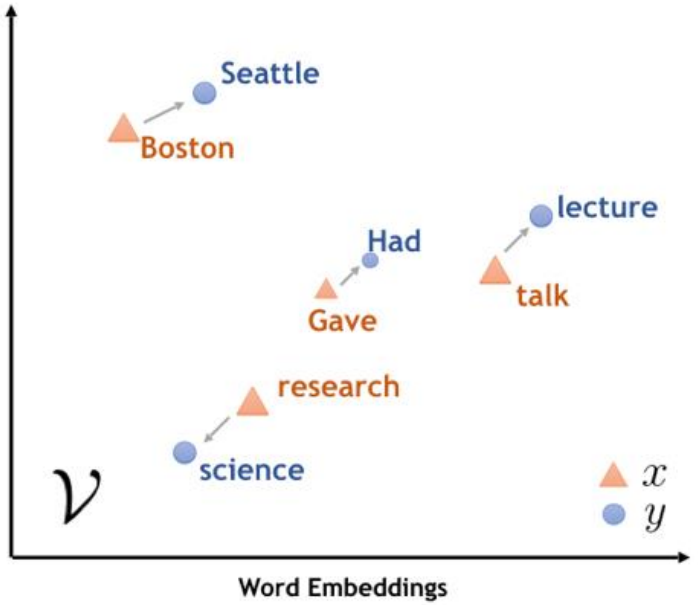
Data Analysis

Practice 7: NLP with Neural Nets

Dr. Nataliya K. Sakhnenko

Word Embeddings

Word embedding is the collective name for a set of feature learning techniques in NLP where words or phrases from the vocabulary are mapped to dense vectors of real numbers.



The most common solution is to represent each word in the vocabulary using a fairly small and dense vector called embedding.

Keras Embedding Layer

Keras offers an **Embedding** layer that can only be used as the first layer in a model.

It turns positive integers (indexes) into dense vectors of fixed size.

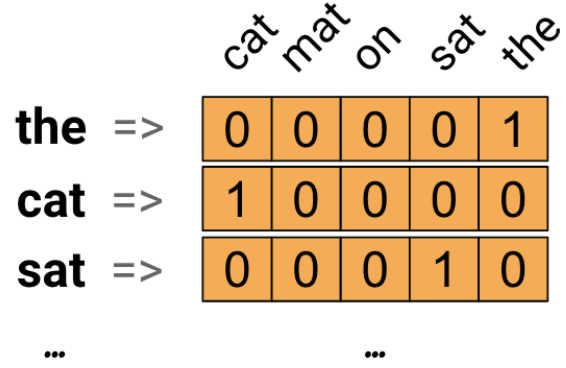
- It can be used as part of a deep learning model where the embedding is learned along with the model itself.
- It can be used to load a pre-trained word embedding model, a type of transfer learning.

Arguments:

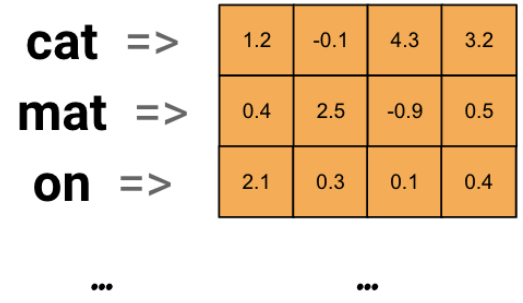
- input_dim: the size of the vocabulary
- output_dim: the size of the vector space in which words will be embedded
- input_length: the length of input sequences

```
tf.keras.layers.Embedding(input_dim, output_dim, … )
```

One-hot encoding

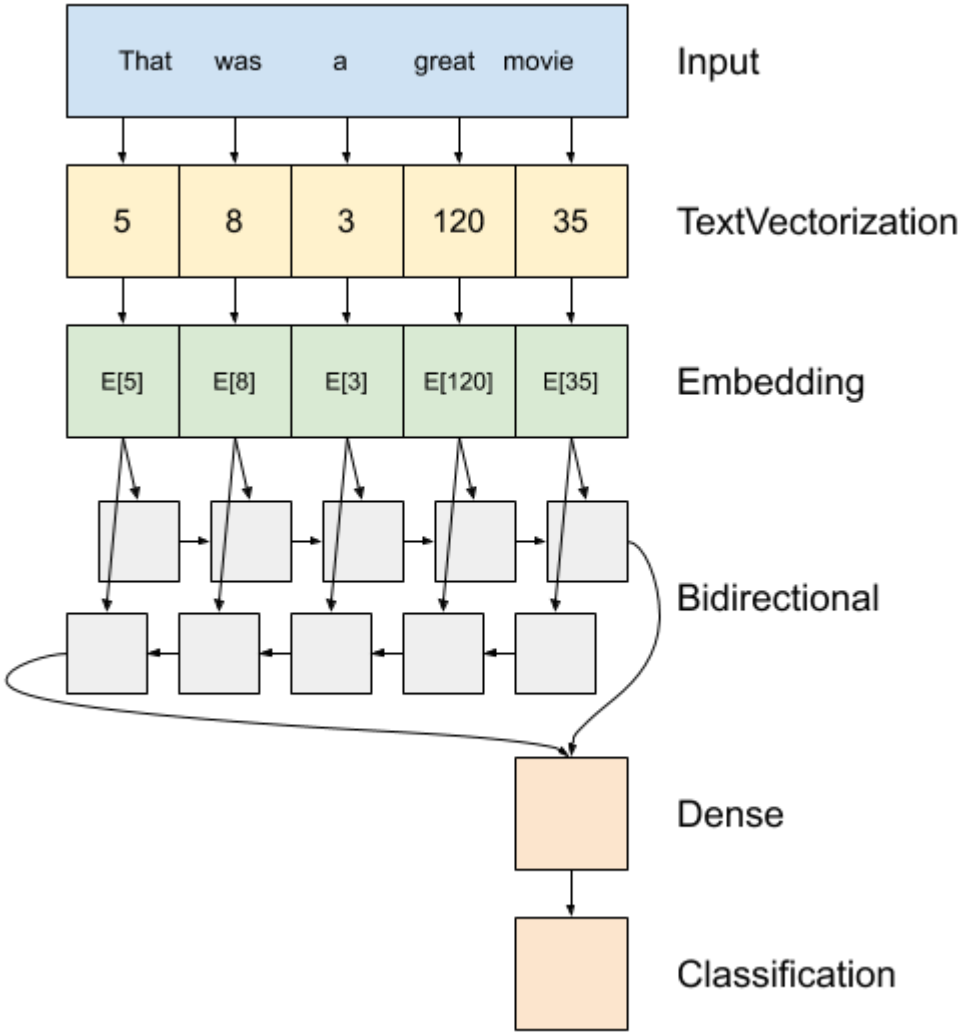


A 4-dimensional embedding



Example: text classification

Text classification with RNN



Sentiment Analysis, IMDB movies

http://ai.stanford.edu/~amaas/data/sentiment/

- ✓ We'll be using a dataset of 50,000 movie reviews taken from IMDb.
- ✓ The data is split evenly with 25k reviews intended for training and 25k for testing your classifier.
- ✓ Each set has 12.5k positive and 12.5k negative reviews.
- ✓ IMDb lets users rate movies on a scale from 1 to 10. To label these reviews the curator of the data labeled anything with ≤ 4 stars as negative and anything with ≥ 7 stars as positive. Reviews with 5 or 6 stars were left out.

Sentiment Analysis, IMDB, LSTM

https://keras.io/examples/nlp/bidirectional_lstm_imdb/

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import imdb
```

```
max_features = 10000
(train_data, train_labels), (test_data,
test_labels) =
imdb.load_data(num_words=max_features)
```

Set the vocabulary size and load in training and test data (among top max_features most common words)

```
25000 train sequences
25000 test sequences
```

Sentiment Analysis, IMDB, LSTM

Padding

In order to feed data into our RNN, all input documents must have the same length. We will limit the maximum review length to max_seq_len by truncating longer reviews and padding shorter reviews with a null value (0)

```
max_seq_len = 80  # cut texts after this number of words
x_train = keras.preprocessing.sequence.pad_sequences (x_train, maxlen=max_seq_len)
x_test = keras.preprocessing.sequence.pad_sequences (x_test, maxlen=max_seq_len)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
```

Pad sequences
x_train shape: (25000, 80)
x_test shape: (25000, 80)

```
print('data ', train_data[0][:10])
print('labels ', train_labels[0])
```

data [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65]
labels 1

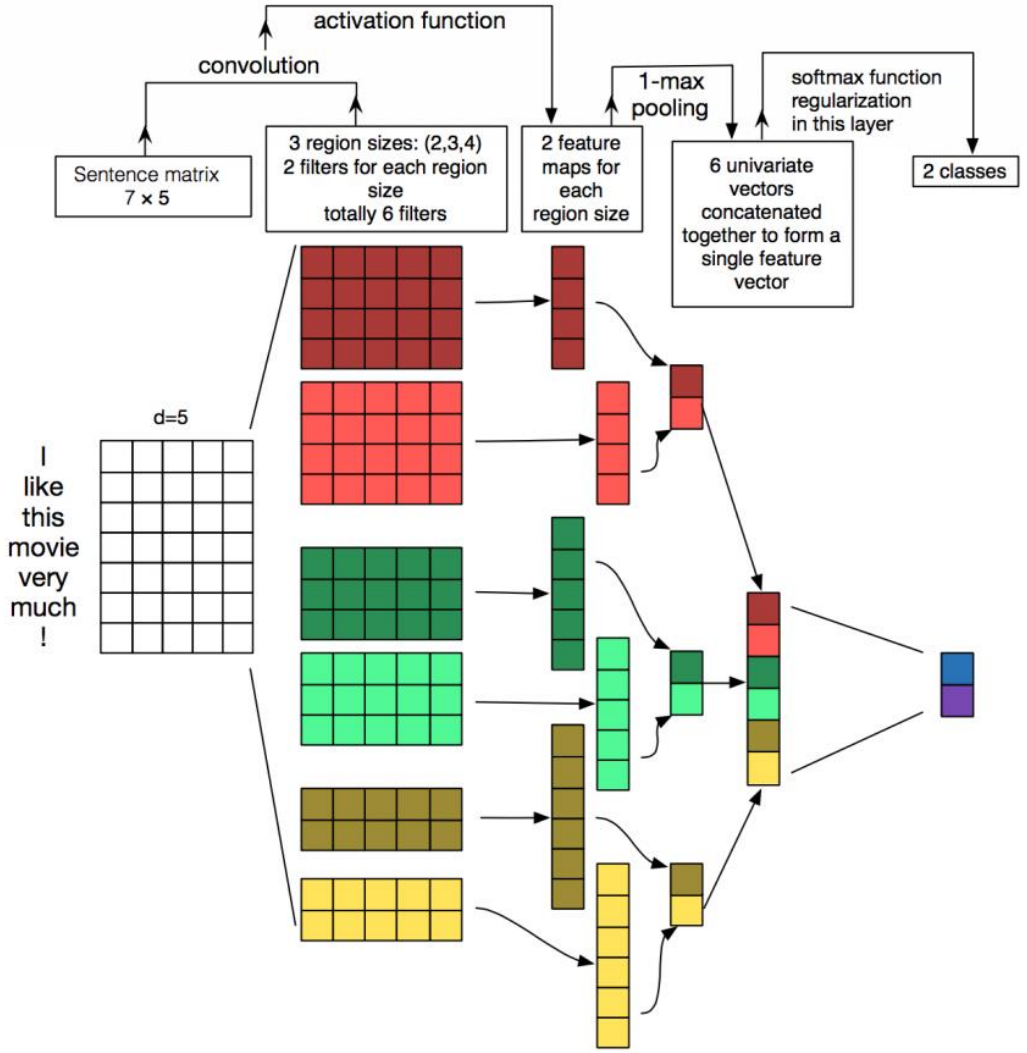
Sentiment Analysis, IMDB, LSTM

```
emb_dim = 128

model = keras.Sequential()
model.add(layers.Embedding(max_features, emb_dim))
model.add(layers.LSTM(128))
model.add(layers.Dense(1, activation='sigmoid’))

model.compile(optimizer="adam", loss="binary_crossentropy",
          metrics=["accuracy"])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 128)         1280000
_________________________________________________________________
lstm (LSTM)                  (None, 128)               131584
_________________________________________________________________
dense (Dense)                (None, 1)                 129
=================================================================
Total params: 1,411,713
Trainable params: 1,411,713
Non-trainable params: 0
_________________________________________________________________
None
```

Text classification with CNN

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1204/slides/cs224n-2020-lecture11-convnets.pdf



- Every filter performs convolution on the sentence matrix and generates feature maps.
- Then 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded.
- Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer.
- The final softmax layer then receives this feature vector as input and uses it to classify the sentence

- Each filter has width = emb_dim, spliting the embedding dim does not make sense
- We make one max for each filter, position of a feature in a sentence is less important
- 3 different size of filters, to capture different scale of correlation between words

Sentiment Analysis, IMDB, CNN

```
emb_dim = 128

model = keras.Sequential()
model.add(layers.Embedding(max_features, emb_dim))
model.add(layers.Conv1D(64, 3, activation = 'relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1, activation='sigmoid’))

model.compile(optimizer="adam", loss="binary_crossentropy",
          metrics=["accuracy"])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, None, 128)         1280000
_________________________________________________________________
conv1d_2 (Conv1D)            (None, None, 64)          24640
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 1,304,705
Trainable params: 1,304,705
Non-trainable params: 0
```

Using pre-trained word embeddings

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
```

The archive contains text-encoded vectors of various sizes: 50-dimensional, 100-dimensional, 200-dimensional, 300-dimensional. We'll use the 100D ones.

Let's make a dict mapping words (strings) to their NumPy vector representation:

```
path_to_glove_file = os.path.join(
    os.path.expanduser("~"), ".keras/datasets/glove.6B.100d.txt"
)

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print("Found %s word vectors." % len(embeddings_index))
```

```
Found 400000 word vectors.
```

https://keras.io/examples/nlp/pretrained_word_embeddings/

Glove Embedding

Using pre-trained word embeddings

```
# Prepare embedding matrix
embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in embedding index will be all-zeros.
        # This includes the representation for "padding" and "OOV"
        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1
print("Converted %d words (%d misses)" % (hits, misses))
```

https://keras.io/examples/nlp/pretrained_word_embeddings/

Glove Embedding

Next, we load the pre-trained word embeddings matrix into an `Embedding` layer.

Note that we set `trainable=False` so as to keep the embeddings fixed (we don't want to update them during training).

```
from tensorflow.keras.layers import Embedding

embedding_layer = Embedding(
    num_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
)
```