21) Working with Text Data: Word Embedding

Vitor Kamada

August 2019

Reference

Chollet (2018): Ch 6.1

https://www.manning.com/books/deep-learning-with-python

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/6.1-one-hot-encoding-of-words-or-characters.ipynb

From Text to Tokens to Vectors

Text
"The cat sat on the mat."

↓

Tokens
"the", "cat", "sat", "on", "the", "mat", "."

↓

Vector encoding of the tokens

0.0	0.0	0.4	0.0	0.0	1.0	0.0
0.5	1.0	0.5	0.2	0.5	0.5	0.0
1.0	0.2	1.0	1.0	1.0	0.0	0.0
the	cat	sat	on	the	mat	.

Chollet (2018: 180)

Bag-of-Words

2-grams:

{"The", "The cat", "cat", "cat sat", "sat", "sat on", "on", "on the", "the", "the mat", "mat"}

3-grams:

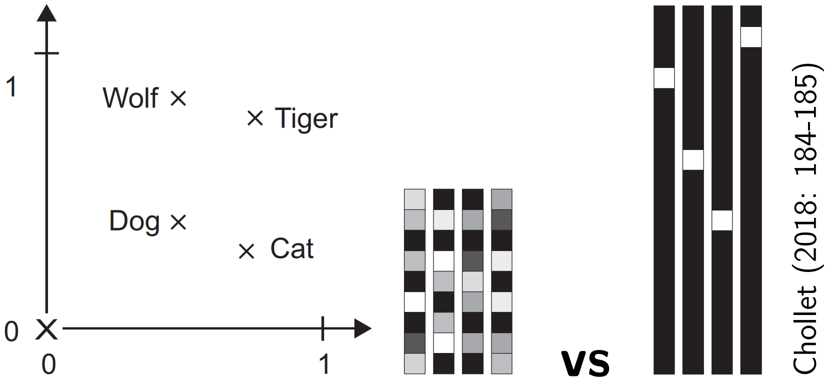
{"The", "The cat", "cat", "cat sat", "The cat sat", "sat", "sat on", "on", "cat sat on", "on the", "the", "sat on the", "the mat", "mat", "on the mat"}

Word-Level One-Hot Encoding

```
from keras.preprocessing.text import Tokenizer

samples = ['The cat sat on the mat.', 'The dog ate my homework.']

# We create a tokenizer, configured to only take
# into account the top-1000 most common words
tokenizer = Tokenizer(num_words=1000)
# This builds the word index
tokenizer.fit_on_texts(samples)

# This turns strings into lists of integer indices.
sequences = tokenizer.texts_to_sequences(samples)

# You could also directly get the one-hot binary representations.
# Note that other vectorization modes than one-hot encoding are sup
one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')

# This is how you can recover the word index that was computed
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
Found 9 unique tokens.
```

Word Embedding vs One-Hot



Dense vs Sparse

Lower-Dimensional vs High-Dimensional

Learned from Data vs Hardcoded

IMDB Movie-Review

```python
from keras.datasets import imdb
from keras import preprocessing
```

```python
# Number of words to consider as features
max_features = 10000
# Cut texts after this number of words
# (among top max_features most common words)
maxlen = 20

# Load the data as lists of integers.
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# This turns our lists of integers
# into a 2D integer tensor of shape `(samples, maxlen)`
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

```
/usr/local/lib/python3.6/dist-packages/numpy/lib/format.py in read_array(fp, allow_pic
    694         # The array contained Python objects. We need to unpickle the data.
    695         if not allow_pickle:
--> 696             raise ValueError("Object arrays cannot be loaded when "
    697                              "allow_pickle=False")
    698         if pickle_kwargs is None:

ValueError: Object arrays cannot be loaded when allow_pickle=False
```

New Code to Load IMDB

```
import numpy as np
# save np.load
np_load_old = np.load

# modify the default parameters of np.load
np.load = lambda *a,**k: np_load_old(*a,
                                      allow_pickle=True, **k)

# call load_data with allow_pickle implicitly set to true
(train_data, train_labels), (test_data,
          test_labels) = imdb.load_data(num_words=10000)

# restore np.load for future normal usage
np.load = np_load_old
```

Reviews	Total	Positive	Negative
Training	25,000	50%	50%
Testing	25,000	50%	50%

```
from keras.models import Sequential
from keras.layers import Flatten, Dense
```

```
model = Sequential()
# We specify the maximum input length to our Embedding layer
# so we can later flatten the embedded inputs
model.add(Embedding(10000, 8, input_length=maxlen))
# After the Embedding layer,
# our activations have shape `(samples, maxlen, 8)`.

# We flatten the 3D tensor of embeddings
# into a 2D tensor of shape `(samples, maxlen * 8)`
model.add(Flatten())

# We add the classifier on top
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy',
              metrics=['acc'])
model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 20, 8)             80000
_________________________________________________________________
flatten_1 (Flatten)          (None, 160)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 161
=================================================================
Total params: 80,161
```

Validation Accuracy of 75%

```
Epoch 1/10
20000/20000 loss: 0.6560 - acc: 0.6482 - val_loss: 0.5906 - val_acc: 0.7146
Epoch 2/10
20000/20000 loss: 0.5189 - acc: 0.7595 - val_loss: 0.5117 - val_acc: 0.7364
Epoch 3/10
20000/20000 loss: 0.4512 - acc: 0.7933 - val_loss: 0.4949 - val_acc: 0.7470
Epoch 4/10
20000/20000 loss: 0.4190 - acc: 0.8069 - val_loss: 0.4905 - val_acc: 0.7538
Epoch 5/10
20000/20000 loss: 0.3965 - acc: 0.8198 - val_loss: 0.4914 - val_acc: 0.7572
Epoch 6/10
20000/20000 loss: 0.3784 - acc: 0.8311 - val_loss: 0.4953 - val_acc: 0.7594
Epoch 7/10
20000/20000 loss: 0.3624 - acc: 0.8419 - val_loss: 0.5004 - val_acc: 0.7574
Epoch 8/10
20000/20000 loss: 0.3474 - acc: 0.8484 - val_loss: 0.5058 - val_acc: 0.7572
Epoch 9/10
20000/20000 loss: 0.3330 - acc: 0.8582 - val_loss: 0.5122 - val_acc: 0.7528
Epoch 10/10
20000/20000 loss: 0.3194 - acc: 0.8669 - val_loss: 0.5183 - val_acc: 0.7554
```

GloVe Word-Embeddings

https://nlp.stanford.edu/projects/glove (822 MB zip)

Precomputed Embeddings from 2014 English Wikipedia

100-dimensional embedding vectors for 400,000 words

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np
```

```
maxlen = 100  # We will cut reviews after 100 words
training_samples = 200  # We will be training on 20
validation_samples = 10000  # We will be validating
max_words = 10000  # We will only consider the top

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

data = pad_sequences(sequences, maxlen=maxlen)

labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)
```

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 100)
Shape of label tensor: (25000,)
```

Parsing and Preparing the GloVe

```
glove_dir = '/home/ubuntu/data/'

embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.6B.100d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

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

```
Found 400000 word vectors.
```

```
embedding_dim = 100

embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if i < max_words:
        if embedding_vector is not None:
            # Words not found in embedding index will be
            embedding_matrix[i] = embedding_vector
```

```python
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
```

```python
model = Sequential()
model.add(Embedding(max_words, embedding_dim,
                    input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

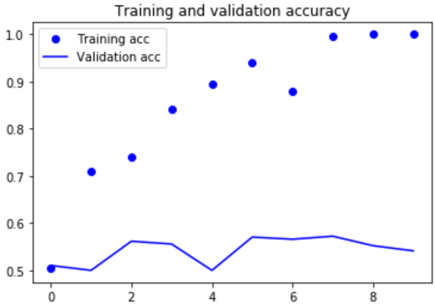
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
flatten_3 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                320032
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,320,065
```

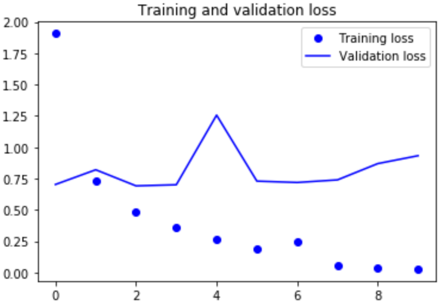
Load and Freeze the Embedding Layer

```
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')
```

Validation Accuracy of 57%





Without the Pre-Trained Word Embeddings

```
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
flatten_4 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_6 (Dense)              (None, 32)                320032
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,320,065
```

Validation Accuracy of 52%