- **Metin ve sıralı verilerin derin öğrenme ile işlenmesi (Deep learning for text and sequences)**

Metin (text) verilerle çalışmak

- Metin-vektör dönüşmü, bazı token dönüşüm işlemleri ve üretilen tokenlar için sayısal vektörlerin belirlenmesinden oluşur.
- Bu vektörler dizi tensörleri içerisinde saklanır ve yapay sinir ağına uygulanır.
- Bir vektörü bir token ile ilişkilendirmenin bir çok yolu vardır.
- Bunlardan en çok kullanılan ikisi,
  - **one-hot encoding**
  - ***word embedding*** *(veya token embedding)*

One-hot encoding

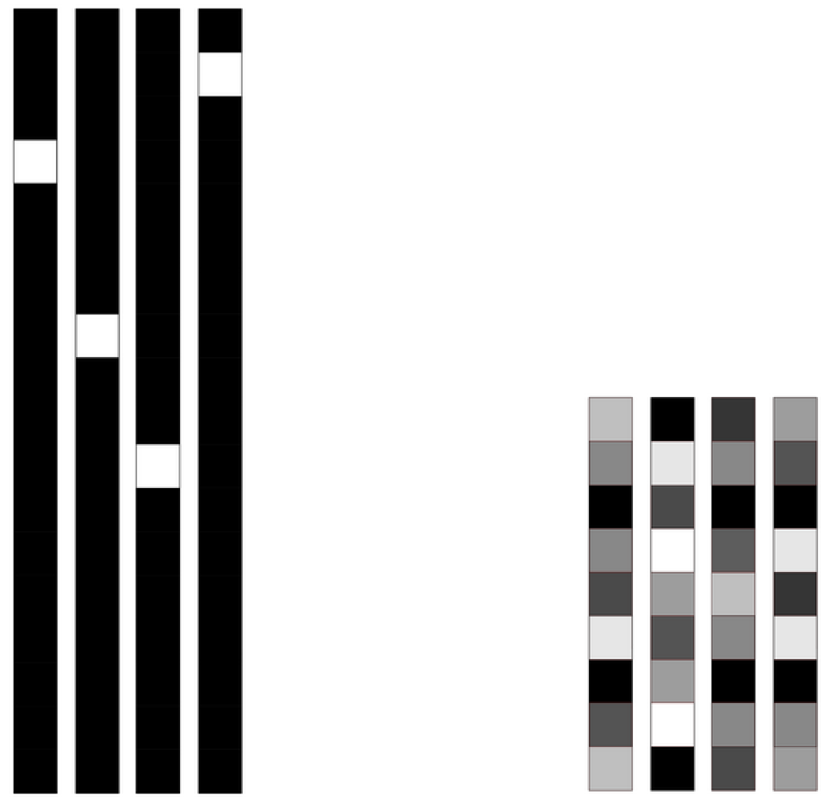
Programda verilen cümledeki
farklı kelimeler bulunup,
her birine farklı bir indis
atanıyor.

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

token_index = {} #boş dictionary

print('Kelimelere atanan indisler:')
for sample in samples:
    for word in sample.split():# sıradaki kelimeyi al
        if word not in token_index:# token indeks verilmemişse
            token_index[word] = len(token_index) + 1 # sıraki tamsayıyı ata
            print('\t',word,'=',token_index[word])

max_length = 10
results = np.zeros(shape=(len(samples),
                          max_length,
                          max(token_index.values()) + 1))

print('Kelimelere vektör ata:')
for i, sample in enumerate(samples):
    for j, word in list(enumerate(sample.split()))[:max_length]:
        index = token_index.get(word)
        print(word,'=',token_index[word])
        results[i, j, index] = 1.
        print('\t',results[i,j,:])
```

```
The = 1
	[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
cat = 2
	[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
sat = 3
	[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
on = 4
	[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
the = 5
	[0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
mat. = 6
	[0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
The = 1
	[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
dog = 7
	[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
ate = 8
	[0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
my = 9
	[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
homework. = 10
	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
```

Kelime gömme (word embeddings)

- One-hot encoding kelime adedine göre (genelde 0’lardan)oluşturulan vektör üzerinde ilgili kelimenin belirtilmesi (genelde 1) ile yapılır.
- Word embedding ile oluşturulan vektörler genelde ağın eğitimi sırasında belirlenir.
- One-hot ile tanımlanan kelime vektörleri Word embedding ile tanımlananlara göre daha düşük boyutludur.



One-hot word vectors:
- Sparse
- High-dimensional
- Hard-coded

Word embeddings:
- Dense
- Lower-dimensional
- Learned from data

Word embedding

Text
“kedi halının üzerinde oturuyor.”

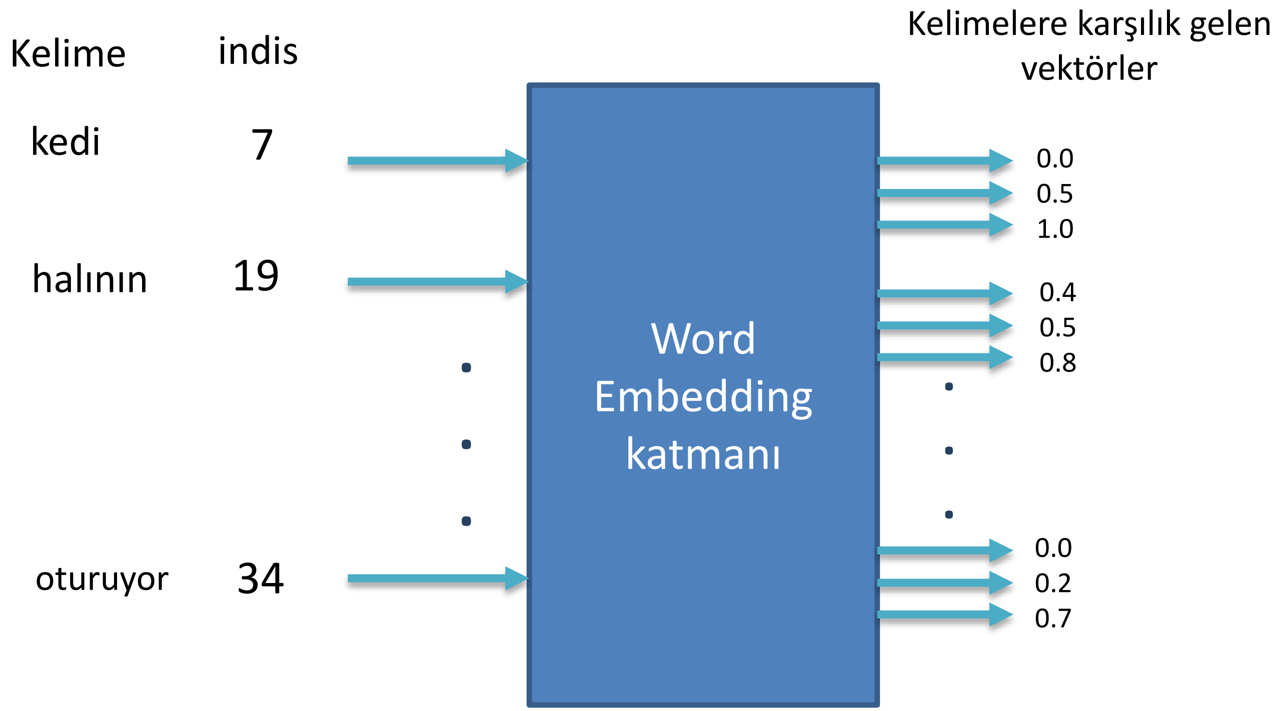
Tokens
“kedi”, halının”, “üzerinde”, “oturuyor”, “.”

vector encoding of the tokens

0.0	0.4	0.0	0.0	0.0
0.5	0.5	0.2	0.5	0.0
1.0	1.0	1.0	1.0	0.0
kedi	halının	üzerinde	oturuyor	.

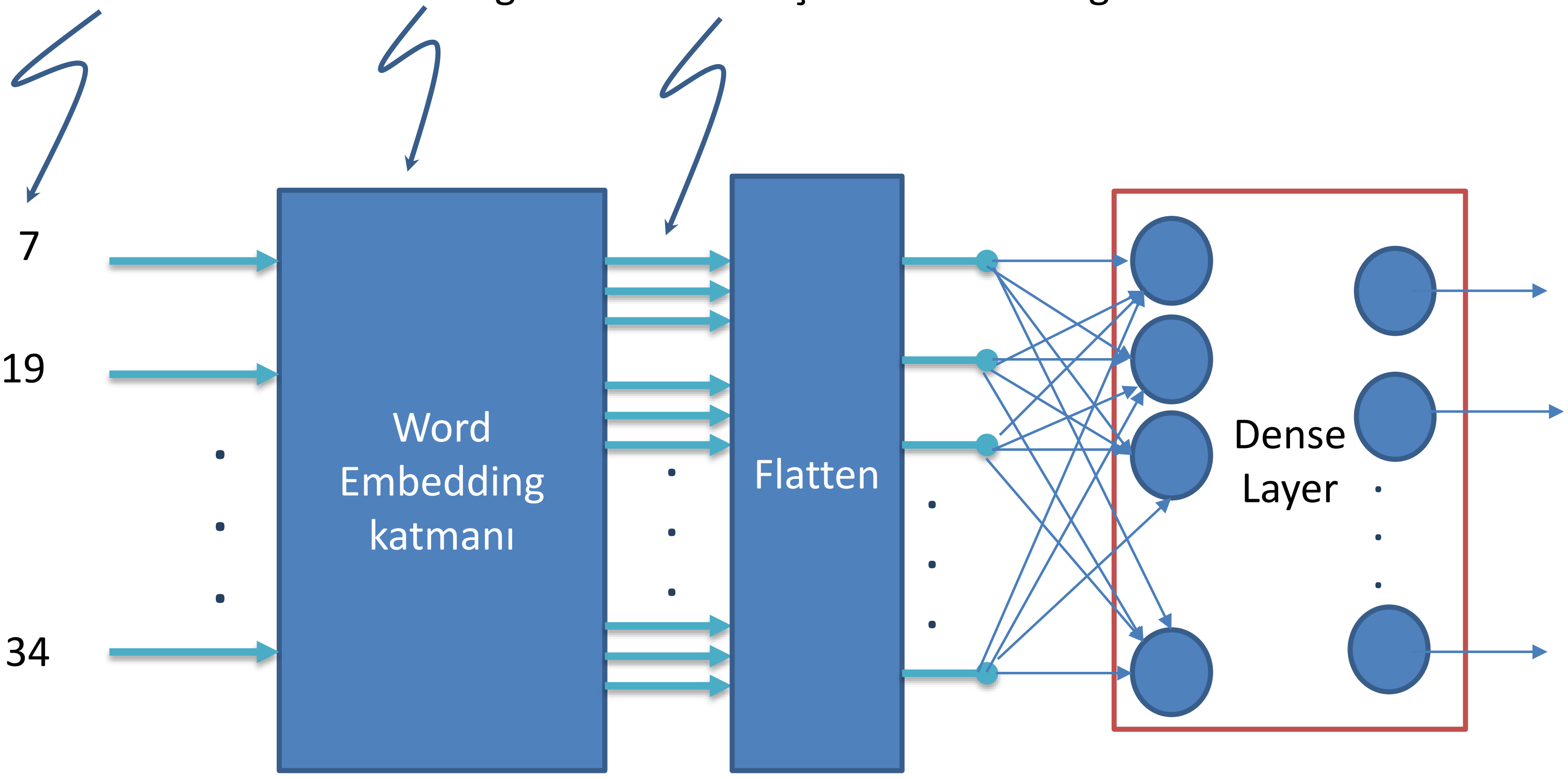
Embedding katmanı

- Embedding katmanı tanımlanmış her bir kelimeye karşılık belirlenmiş sayı vektörlerini verir.
- Girişe kelimelere ait tamsayı indisler uygulanır. Bir iç sözlüğe bakarak girilen indislerin vektör karşılıklarının verir



Embedding katmanı ve sınıflandırıcı

- Kelime indeksi →Embedding katman → Girişteki kelimelere göre kelime vektörleri



IMDB üzerinde Embedding katmanı ve sınıflandırıcı

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

# Özellik olarak düşnülecek ke
max_features = 10000
#20 kelimeden sonra me    kes
maxlen = 20

#veriyi tamsayılar listesi olarak
(x_train, y_train), (x_test, y_test) = \
imdb.load_data(num_words=max_features)

# veriyi 2D tensor olarak yükle (samples, maxlen)
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

Yorumların 20 kelimeden sonrasını keserek oluşturulacak ağın embedding katmanını 20 giriş ile sınırlandırdık. Kelime sınırı örneğin 100 veya daha fazla yapılarak başarıma olan etkisine bakılabilir.

20 kelimeden az yorumları sıfır ile dolgulayıp 20 kelimeye tamamla.

IMDB üzerinde Embedding katmanı ve sınıflandırıcı

```python
#Bir embedding katmanı ve sınıflandırıcı kullan
from keras.models import Sequential
from keras.layers import Flatten, Dense, Embedding

model = Sequential()
model.add(Embedding(10000, 8, input_length=maxlen))

model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
model.summary()

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

10000 kelime 8 elemanlı vektörler ile temsil edilecek.

IMDB üzerinde Embedding katmanı ve sınıflandırıcı

- Model.summary()

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 20, 8)             80000
_________________________________________________________________
flatten_1 (Flatten)          (None, 160)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 161
=================================================================
Total params: 80,161
Trainable params: 80,161
Non-trainable params: 0
```

10000 kelime 8 elemanlı vektörler ile temsil ediliyor toplam 80000 eğitilecek parametre

Yorum sayısı 20 kelime ve her kelime 8 parametreden oluşuyor. Flatten işlemi sonrası dense layer girişi toplam=20*8=160

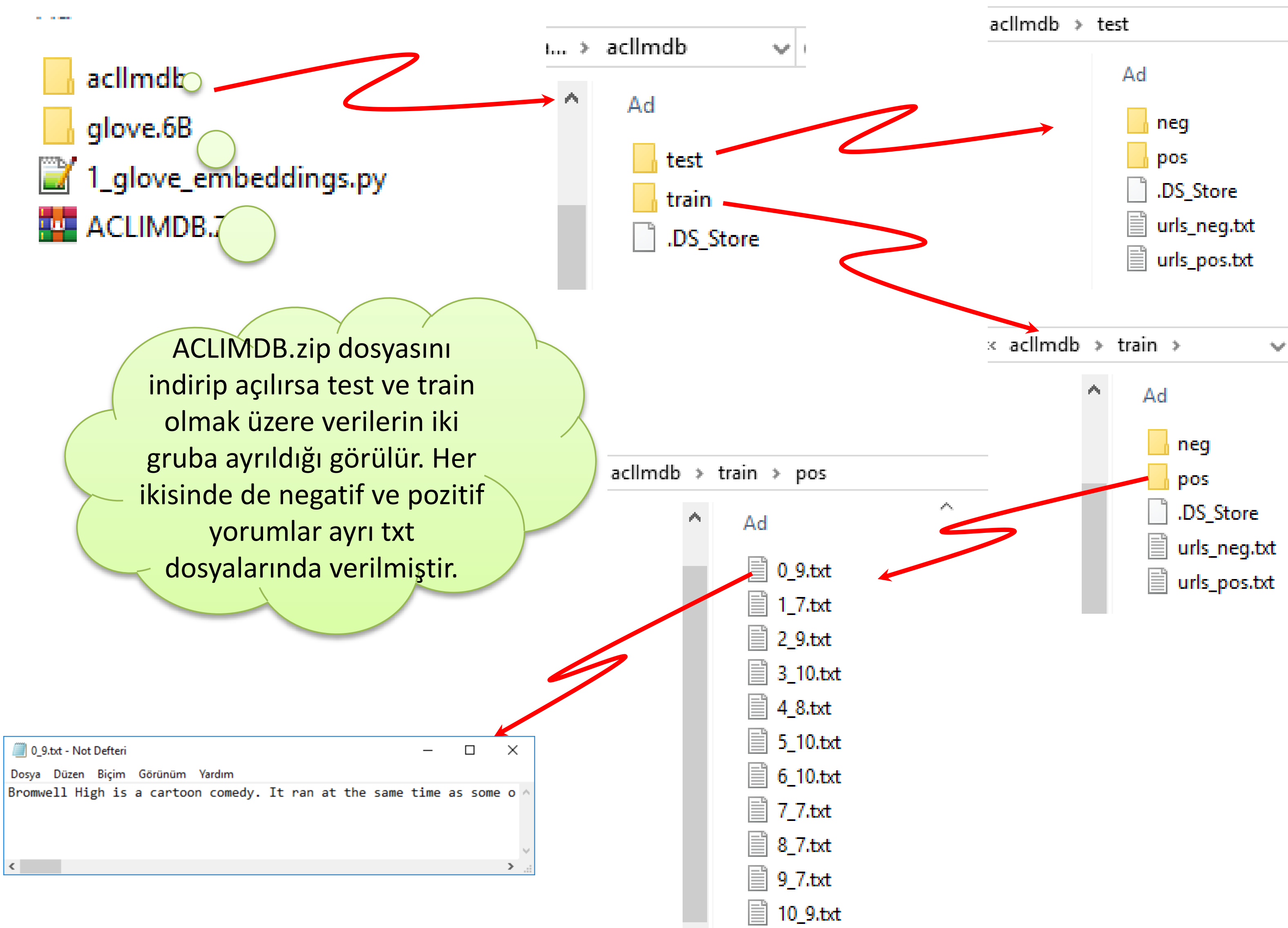
Dense layer 160 girişli bir elemandan oluşuyor. Bu katmanda, bias ile birlikte toplam eğitilecek 161 parametre var

IMDB üzerinde Embedding katmanı ve sınıflandırıcı

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 8s 379us/step - loss:
0.6759 - acc: 0.6050 - val_loss: 0.6398 - val_acc: 0.6814
Epoch 2/10
20000/20000 [==============================] - 2s 107us/step - loss:
0.5657 - acc: 0.7427 - val_loss: 0.5467 - val_acc: 0.7206
Epoch 3/10
20000/20000 [==============================] - 2s 109us/step - loss:
0.4752 - acc: 0.7808 - val_loss: 0.5113 - val_acc: 0.7384
Epoch 4/10
20000/20000 [==============================] - 2s 105us/step - loss:
0.4263 - acc: 0.8077 - val_loss: 0.5008 - val_acc: 0.7452
Epoch 5/10
20000/20000 [==============================] - 2s 105us/step - loss:
0.3930 - acc: 0.8258 - val_loss: 0.4981 - val_acc: 0.7538
Epoch 6/10
20000/20000 [==============================] - 2s 105us/step - loss:
0.3668 - acc: 0.8395 - val_loss: 0.5014 - val_acc: 0.7530
Epoch 7/10
20000/20000 [==============================] - 2s 106us/step - loss:
0.3435 - acc: 0.8533 - val_loss: 0.5052 - val_acc: 0.7520
Epoch 8/10
20000/20000 [==============================] - 2s 108us/step - loss:
0.3223 - acc: 0.8657 - val_loss: 0.5132 - val_acc: 0.7486
Epoch 9/10
20000/20000 [==============================] - 2s 108us/step - loss:
0.3022 - acc: 0.8766 - val_loss: 0.5213 - val_acc: 0.7490
Epoch 10/10
20000/20000 [==============================] - 2s 111us/step - loss:
0.2839 - acc: 0.8860 - val_loss: 0.5303 - val_acc: 0.7466
```

IMDB yorumlarının işlenmesi

ACLIMDB.zip dosyasını indirip açılırsa test ve train olmak üzere verilerin iki gruba ayrıldığı görülür. Her ikisinde de negatif ve pozitif yorumlar ayrı txt dosyalarında verilmiştir.



IMDB yorumlarının işlenmesi

```python
import os
imdb_dir = 'aclImdb'
train_dir = os.path.join(imdb_dir, 'train')

labels = []
texts = []

for label_type in ['neg', 'pos']:
    dir_name = os.path.join(train_dir, la
    for fname in os.listdir(dir_name):
        print(label_type,fname)
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname),encoding="utf8")
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
```

Yorumları text isimli listeye ekle.

label_type negatif ise 0, pozitif ise 1.

```
In [240]: texts[5]
Out[240]: '"It appears that many critics find the idea of a Woody Allen
drama unpalatable." And for good reason: they are unbearably wooden and
pretentious imitations of Bergman. And let\'s not kid ourselves: critics
were mostly supportive of Allen\'s Bergman pretensions, Allen\'s whining
accusations to the contrary notwithstanding. What I don\'t get is this: why
was Allen generally applauded for his originality in imitating Bergman, but
the contemporaneous Brian DePalma was excoriated for "ripping off"
Hitchcock in his suspense/horror films? In Robin Wood\'s view, it\'s a
strange form of cultural snobbery. I would have to agree with that.'

In [241]: labels[5]
Out[241]: 0
```

IMDB yorumlarının işlenmesi

```python
#Tokenize the data -------------------------------------------------------
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 200 samples
validation_samples = 10000  # We will be validating on 10000 samples
max_words = 10000  # We will only consider the top 10,000 words in the dataset

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)
```

Sık kullanılan 10000 kelimeyi numaralandır

```
In [276]: word_index.get('this')
Out[276]: 11

In [277]: word_index.get('apple')
Out[277]: 7671

In [278]: word_index.get('funny')
Out[278]: 160
```

100 kelimeden az yorumları sıfırlarla doldur.

```
Found 88562 unique tokens.
Shape of data tensor: (25000, 100)
Shape of label tensor: (25000,)
```

IMDB yorumlarının işlenmesi

- Veriyi karıştır.
- Eğitim ve geçerleme olmak üzere iki kümeye ayır.

```
# Split the data into a training set and a validation set
# But first, shuffle the data, since we started from data
# where sample are ordered (all negative first, then all positive).
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

```
In [287]: indices
Out[287]: array([    0,     1,     2, ..., 24997, 24998, 24999])

In [288]: np.random.shuffle(indices)

In [289]: indices
Out[289]: array([ 2355,   370, 24190, ...,  6176, 20823,  2243])
```

Önceden eğitilmiş (Pretrained) Word embeddings

- Daha önce eğitilmiş (pretraine) kelime vektörlerine ait ağırlıklar kullanılarak eğitim hızlandırılabilir.
- 2Dconv ağında olduğu gibi eğitilmiş ağa ait ağırlıklar yüklenirse ilgili katman için

  trainable = False yapılır.
- Kelime ağındaki tüm kelimeleri kullanmak yerine ilgilenilen kelimeler bir ağırlık matrisine yazılır ve Embedding layer ağırlıkları olarak yüklenir.

GloVe word embeddings

- GloVe, kelimeler için vektör gösterimleri elde etmek için denetimsiz bir öğrenme (unsupervised learning) algoritmasıdır.
- Önceden eğitilmiş ağırlıkları kullanmak için aşağıda verilen bağlantıya gidip Dosya 822MB boyutlu, *glove.6B.zip* 2014 English Wikipedia dosyası indirilmelidir.
- Bu dosya içerisinden 400.000 kelimeye ait 100-boyutlu embedding vektörleri kullanılacaktır.

https://nlp.stanford.edu/projects/glove/

GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning

Introduction

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-wor
co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

Getting started (Code download)

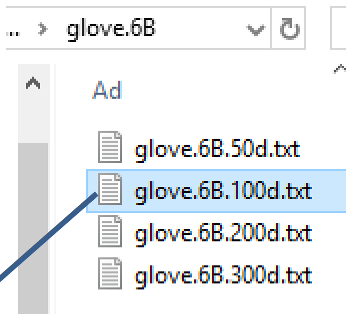
- Download the code (licensed under the Apache License, Version 2.0)
- Unpack the files: unzip GloVe-1.2.zip
- Compile the source: cd GloVe-1.2 && make
- Run the demo script: ./demo.sh
- Consult the included README for further usage details, or ask a question
- The code is also available on GitHub

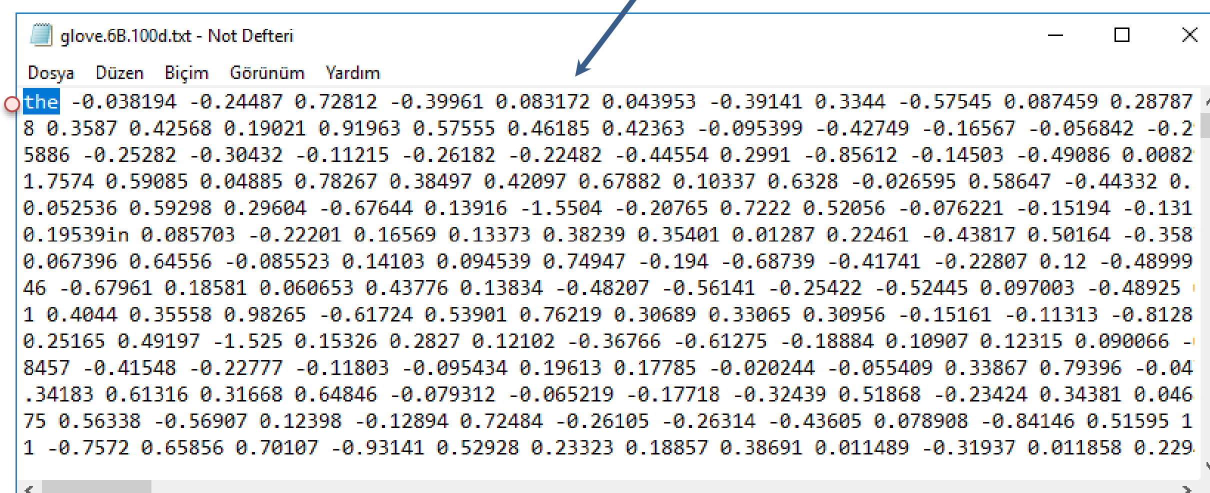
Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip

Önceden eğitilmiş (Pretrained) Word embeddings

```python
#Pre-process the embeddings Let's parse the un-zipped file (it's a txt file)
#to build an index mapping words (as strings) to their vector
#representation (as number vectors).

glove_dir = 'glove.6B'

embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.6B.100d.txt'),encoding="utf8")
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))
```




glove.6B.100d.txt - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

```
the -0.038194 -0.24487 0.72812 -0.39961 0.083172 0.043953 -0.39141 0.3344 -0.57545 0.087459 0.28787
8 0.3587 0.42568 0.19021 0.91963 0.57555 0.46185 0.42363 -0.095399 -0.42749 -0.16567 -0.056842 -0.2
5886 -0.25282 -0.30432 -0.11215 -0.26182 -0.22482 -0.44554 0.2991 -0.85612 -0.14503 -0.49086 0.0082
1.7574 0.59085 0.04885 0.78267 0.38497 0.42097 0.67882 0.10337 0.6328 -0.026595 0.58647 -0.44332 0.
0.052536 0.59298 0.29604 -0.67644 0.13916 -1.5504 -0.20765 0.7222 0.52056 -0.076221 -0.15194 -0.131
0.19539in 0.085703 -0.22201 0.16569 0.13373 0.38239 0.35401 0.01287 0.22461 -0.43817 0.50164 -0.358
0.067396 0.64556 -0.085523 0.14103 0.094539 0.74947 -0.194 -0.68739 -0.41741 -0.22807 0.12 -0.48999
46 -0.67961 0.18581 0.060653 0.43776 0.13834 -0.48207 -0.56141 -0.25422 -0.52445 0.097003 -0.48925
1 0.4044 0.35558 0.98265 -0.61724 0.53901 0.76219 0.30689 0.33065 0.30956 -0.15161 -0.11313 -0.8128
0.25165 0.49197 -1.525 0.15326 0.2827 0.12102 -0.36766 -0.61275 -0.18884 0.10907 0.12315 0.090066 -
8457 -0.41548 -0.22777 -0.11803 -0.095434 0.19613 0.17785 -0.020244 -0.055409 0.33867 0.79396 -0.04
.34183 0.61316 0.31668 0.64846 -0.079312 -0.065219 -0.17718 -0.32439 0.51868 -0.23424 0.34381 0.046
75 0.56338 -0.56907 0.12398 -0.12894 0.72484 -0.26105 -0.26314 -0.43605 0.078908 -0.84146 0.51595 1
1 -0.7572 0.65856 0.70107 -0.93141 0.52928 0.23323 0.18857 0.38691 0.011489 -0.31937 0.011858 0.229
```

Kelimeler listesi ve
100 elemanlı vektör
karşılıkları

Pretrained Word embeddings

```python
glove_dir = 'glove.6B

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

print('Found %s
```

Kelimeler ve vektör karşılıkları sırayla seçilir. Kelimeler key olarak ve ona ait vektörler value olarak kaydedilir.

```
transitioning 0.20251 0.12278 -0.29583 0.29518 0.028708 0.31636 -0.046019
-0.016049 0.33441 0.16679 0.40064 -0.29935 0.42104 -0.17614 0.65392 -0.81993
0.50021 0.068609 0.7578 -0.29849 -0.49697 0.29507 0.48266 0.1083 -0.055123 0.32345
0.23867 -0.046655 0.2335 0.21204 -0.7196 0.074292 -0.27075 -0.36241 -0.12487
-0.34504 -0.31462 -0.36922 0.00089996 -0.0053808 -0.90473 0.39511 -0.16349 0.64374
-0.23603 0.84891 0.081826 0.67047 -0.020202 -0.25858 -0.82844 -0.27441 -0.33778
0.56883 0.17465 0.41678 0.52558 -0.48956 0.24056 0.057252 0.063338 0.54847
-0.12381 0.093183 0.15027 0.21815 -0.37329 -0.30709 -0.096194 0.28707 0.37655
-0.38157 -0.33439 -0.30448 0.031091 -0.082888 -0.539 -0.097052 0.23306 -0.96377
-0.92684 -0.054472 -0.79825 0.6423 -0.013918 0.054843 0.34311 -0.37162 -0.69595
-0.40519 -0.48418 0.39324 -0.70599 -0.46123 0.60166 -0.85254 0.50901 0.47864
0.14736 0.29343

enquiries -0.14904 -0.21957 -0.31082 -0.065191 -0.41996 0.50185 0.11259 0.18736
0.52458 0.15915 0.25132 0.38871 0.34784 0.049581 0.096843 0.21427 -0.37692 0.50058
0.021539 0.76779 -0.17127 0.17142 -0.28802 0.048461 -0.28317 -0.5767 1.1215 1.0056
0.030221 -0.46243 -0.53596 -0.52128 -0.082707 0.10012 -0.067101 0.48861 -0.48095
-0.43809 -0.13637 -0.27922 -0.32265 -0.20612 -0.16561 -0.63729 -0.032788 -0.078506
0.2115 -0.37213 0.68803 0.21017 0.56802 0.21348 0.20787 0.28016 -0.58384 0.97988
0.13051 -0.56958 0.40867 0.082901 0.047878 0.56558 -0.35077 0.76387 0.28555
-0.29691 -0.38145 0.096004 0.83453 0.058359 0.054206 0.0017946 0.68252 -1.6077
0.32038 0.47594 -0.24858 0.57482 -1.3976 -0.51926 0.34805 0.7156 0.81954 0.46885
-0.67936 -0.77978 -0.091613 0.050902 -0.63979 -0.029186 -0.69195 0.14074 -0.78406
-0.34735 1.1656 -0.078694 -0.014208 0.21814 -0.14069 0.32493

questioner -0.074675 0.057146 0.88043 0.20997 -0.84641 0.38579 0.14395 0.11415
0.80489 0.32272 -0.5392 -0.013676 -0.38251 -0.41375 0.3559 -0.066215 -0.6975
-0.37779 0.11049 -1.0119 -0.17246 -0.32064 -0.98775 -0.26829 0.75113 0.16563
-0.16766 0.1512 0.056067 0.22888 -0.11186 0.34037 0.82496 0.17217 -0.16655 0.47789
-0.81997 -0.39386 0.93369 -0.012237 -0.73768 0.26562 -0.15671 0.16761 -0.79503
-0.14487 -0.023603 -0.0095582 -0.4701 0.10814 0.039796 0.055251 0.54247 0.40603
```

```
In [299]: embeddings_index['car']
Out[299]:
array([-0.1684  , -0.53827 ,  0.31155 , -0.53218 ,  0.26678 , -0.13638 ,
        0.36621 ,  0.68383 ,  0.77726 ,  0.68049 ,  0.69137 ,  0.2103  ,
        0.091065,  0.24845 , -0.16157 ,  0.46291 , -0.1503  ,  0.2562  ,
       -0.1199  ,  0.5913  ,  1.0351  , -0.2052  ,  0.30244 , -0.34101 ,
        0.6326  , -0.31603 , -0.9959  , -0.33583 ,  0.25161 ,  0.10323 ,
        0.019611,  0.54893 , -0.33433 ,  0.29617 ,  0.41218 ,  0.4207  ,
        0.25775 ,  0.12709 ,  0.80269 ,  0.61944 ,  0.54316 , -0.5941  ,
        0.87551 , -0.063686, -0.29117 ,  0.61609 ,  0.33376 ,  0.14488 ,
       -0.039021, -1.1849  , -0.45951 ,  0.15631 , -0.50818 ,  1.2357  ,
        0.30965 , -1.958   , -1.1872  ,  1.2027  ,  2.1138  ,  0.083629,
        0.54319 ,  0.78883 ,  0.35416 ,  0.87736 ,  0.54007 , -0.10454 ,
        0.075371, -0.45727 , -0.27466 ,  0.11838 , -0.49412 , -0.61325 ,
        0.071519, -0.57665 ,  0.21371 ,  0.62137 ,  1.4404  , -0.34033 ,
       -0.89958 , -0.69605 ,  0.74058 ,  0.52105 , -0.19224 , -0.20366 ,
       -0.22409 , -0.3708  , -0.34663 , -0.86018 , -0.89182 , -0.43871 ,
        0.19424 ,  0.17073 ,  0.43663 , -0.11295 , -0.51156 ,  0.34186 ,
       -0.10274 ,  0.39684 ,  1.734   , -0.70787 ], dtype=float32)
```

Pretrained Word embeddings

word_index imdb database'a ait kelimeleri içerir. For döngüsü ile bunların karşılıkları ön eğitimli ağdan seçilerek tanımlanacak ağda embedding katmanında kullanılmak üzere embedding_matrix'e yazılır.

```
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 all-zeros.
            embedding_matrix[i] = embedding_vector

```

Aranan kelime ön eğitimli ağda yoksa vektör sıfırlardan oluşuyor.

Pretrained Word embeddings

```python
#Define a model
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

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'))


#Load the GloVe embeddings in the model
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False

model.summary()

#Train and evaluate
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')
```

- Öneğitimli ağdan seçilen ağırlık değerlerini yükle.
- layers[0].trainable=false yapılarak embedding katmanı eğitim dışı tutuluyor.

Pretrained Word embeddings

```
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
embedding_12 (Embedding)     (None, 100, 100)          1000000
________________________________________________________________
flatten_9 (Flatten)          (None, 10000)             0
________________________________________________________________
dense_11 (Dense)             (None, 32)                320032
________________________________________________________________
dense_12 (Dense)             (None, 1)                 33
================================================================
Total params: 1,320,065
Trainable params: 320,065
Non-trainable params: 1,000,000
```

Embedding katmanı donduruldu.

```
Epoch 7/10
200/200 [==============================] - 1s 3ms/step - loss: 0.1218 - acc: 0.9850 -
val_loss: 0.7754 - val_acc: 0.5490
Epoch 8/10
200/200 [==============================] - 1s 3ms/step - loss: 0.1852 - acc: 0.9100 -
val_loss: 0.7782 - val_acc: 0.5583
Epoch 9/10
200/200 [==============================] - 0s 2ms/step - loss: 0.0391 - acc: 1.0000 -
val_loss: 0.8255 - val_acc: 0.5433
Epoch 10/10
200/200 [==============================] - 1s 3ms/step - loss: 0.0266 - acc: 1.0000 -
val_loss: 0.8657 - val_acc: 0.5490
```

Eğitim 200 örnek üzerinde gerçekleştirildiği için geçerleme düşük.

Pretrained Word embeddings

```python
def loadImdb():
    imdb_dir = 'aclImdb'
    maxlen = 100
    max_words = 10000
    tokenizer = Tokenizer(num_words=max_words)
    test_dir = os.path.join(imdb_dir, 'test')
    labels = []
    texts = []
    for label_type in ['neg', 'pos']:
        dir_name = os.path.join(test_dir, label_type)
        for fname in sorted(os.listdir(dir_name)):
            if fname[-4:] == '.txt':
                f = open(os.path.join(dir_name, fname),
                         encoding="utf8")
                texts.append(f.read())
                f.close()
                if label_type == 'neg':
                    labels.append(0)
                else:
                    labels.append(1)
    tokenizer.fit_on_texts(texts)
    sequences = tokenizer.texts_to_sequences(texts)
    x_test = pad_sequences(sequences, maxlen=maxlen)
    y_test = np.asarray(labels)
    return (x_test,y_test)

(x_test, y_test)=loadImdb()
model=models.load_model('pre_trained_glove_model.h5')
[loss,acc]=model.evaluate(x_test, y_test)
print("loss=",loss,"acc=",acc)
```

Eğitim verilerine benzer şekilde test verilerinin de vektör karşılıklarını elde et.

Eğitilmiş modeli yükle. Test verileri ile başarımını ölç.