یادگیری عمیق

جلسه ۲۱ و ۲۲

یادگیری عمیق
برای متن و دنباله‌ها

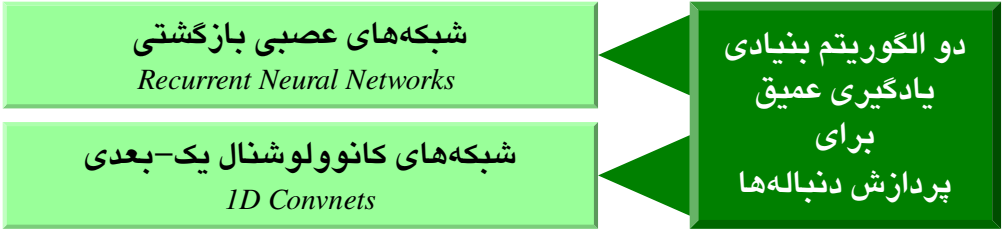
Deep Learning for Text and Sequences

کاظم فولادی قلعه
دانشکده مهندسی، پردیس فارابی
دانشگاه تهران

http://courses.fouladi.ir/deep

یادگیری عمیق برای متن و دنباله‌ها

DEEP LEARNING FOR TEXT AND SEQUENCES



یادگیری عمیق برای متن و دنباله‌ها

کاربردها

طبقه‌بندی سندها *Document Classification*	مانند تعیین موضوع یک مقاله / نویسنده‌ی یک کتاب
طبقه‌بندی سری‌های زمانی *Timeseries Classification*	مانند تعیین سبک یک موسیقی
مقایسه‌ی سری‌های زمانی *Timeseries Comparisons*	مانند تخمین میزان نزدیکی دو سند
یادگیری دنباله-به-دنباله *Sequence-to-Sequence Learning*	مانند ترجمه‌ی جملات انگلیسی به فارسی
تحلیل احساس *Sentiment Analysis*	مانند طبقه‌بندی احساسات در توئیت‌ها یا نقد فیلم‌ها به‌عنوان مثبت/منفی
پیش‌بینی سری‌های زمانی *Timeseries Forecasting*	مانند پیش‌بینی آینده‌ی آب‌وهوا در یک موقعیت خاص با داشتن داده‌های اخیر آب‌وهوا

مثال‌های این درس: (۱) تحلیل احساس بر روی مجموعه داده IMDB (۲) پیش‌بینی دما

یادگیری عمیق برای متن و دنباله‌ها

۱

کار با داده‌های متنی

کار با داده‌های متنی

WORKING WITH TEXT DATA

متن *Text*	دنباله‌ای از نویسه‌ها / کاراکترها
	دنباله‌ای از کلمات

نگاه به متن در قالب دنباله‌ای از کلمات، متداول‌تر است.

مدل‌های یادگیری عمیق، متن را به‌گونه‌ای که انسان درک می‌کند، درک نمی‌کنند؛
بلکه این مدل‌ها **ساختار آماری** زبان نوشتاری را نگاشت می‌کنند
که برای بسیاری از وظایف متنی کافی است.

یادگیری عمیق برای پردازش زبان طبیعی	یادگیری عمیق برای بینایی کامپیوتری
بازشناسی الگوی حاکم بر کلمات، جملات و پاراگراف‌ها	بازشناسی الگوی حاکم بر پیکسل‌ها

کار با داده‌های متنی

برداری‌سازی متن

VECTORIZING TEXT

مدل‌های یادگیری عمیق/شبکه‌های عصبی، متن خام را به‌عنوان ورودی دریافت نمی‌کنند.
برداری‌سازی متن: فرآیند تغییر شکل متن به تانسورهای عددی

روش‌های برداری‌سازی متن

- بخش‌بندی متن به **کلمات** و تبدیل هر کلمه به یک بردار
- بخش‌بندی متن به **نویسه‌ها**/کاراکترها و تبدیل هر کاراکتر به یک بردار
- بخش‌بندی متن به **N-gram**ها:
  استخراج N-gramها از کلمات یا نویسه‌ها و تبدیل هر N-gram به یک بردار

N-gramها گروه‌های همپوشان از چند کلمه یا نویسه‌ی متوالی هستند.

کار با داده‌های متنی

توکن

TOKEN

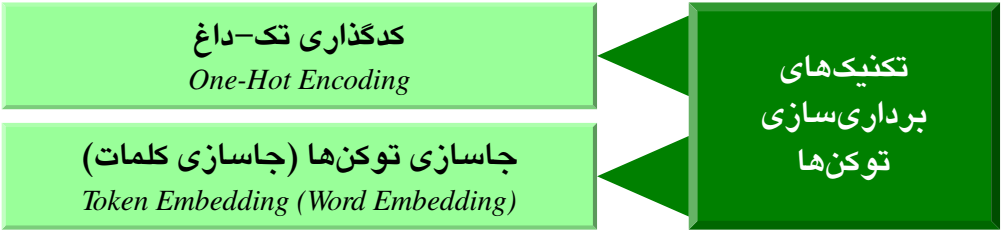
توکن *Token*	واحدی که متن به آن بخش‌بندی می‌شود.
کلمه *Word*	
نویسه *Character*	
N–گرام *N-gram*	
توکن‌بندی *Tokenization*	بخش‌بندی متن به توکن‌ها

تمام فرآیندهای برداری‌سازی متن،
شامل استفاده از یک روش برای بخش‌بندی بر اساس توکن و
سپس مرتبط سازی بردارهای عددی با توکن‌های تولیدشده است.

کار با داده‌های متنی

تکنیک‌های برداری‌سازی توکن‌ها

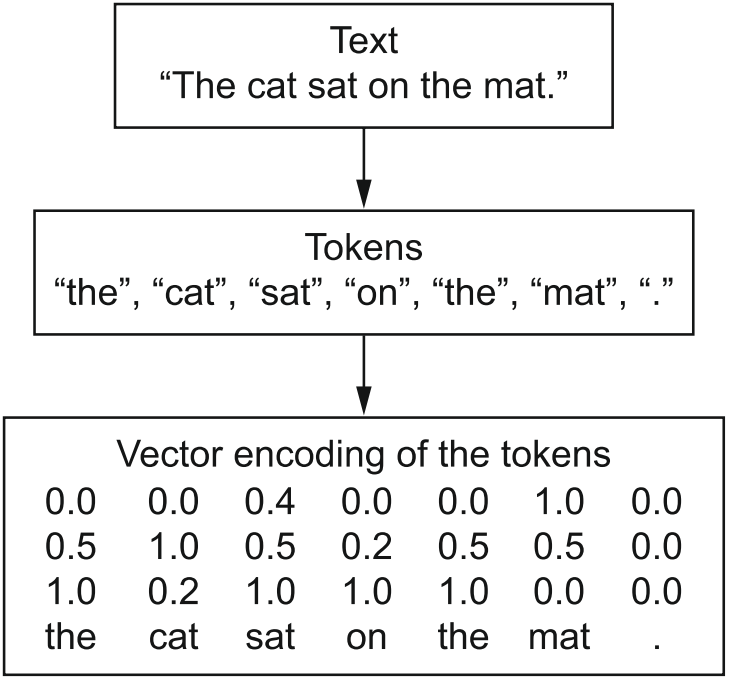
TOKEN VECTORIZATION



تکنیک جاسازی توکن‌ها منحصراً برای کلمات به‌کار می‌رود.

کار با داده‌های متنی

برداری‌سازی متن: مثال



از متن به توکن‌ها به بردارها

Understanding n-grams and bag-of-words

Word n-grams are groups of *N* (or fewer) consecutive words that you can extract from a sentence. The same concept may also be applied to characters instead of words.

Here's a simple example. Consider the sentence "The cat sat on the mat." It may be decomposed into the following set of 2-grams:

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

It may also be decomposed into the following set of 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"}
```

Such a set is called a *bag-of-2-grams* or *bag-of-3-grams*, respectively. The term *bag* here refers to the fact that you're dealing with a *set* of tokens rather than a list or sequence: the tokens have no specific order. This family of tokenization methods is called *bag-of-words*.

Because bag-of-words isn't an order-preserving tokenization method (the tokens generated are understood as a set, not a sequence, and the general structure of the sentences is lost), it tends to be used in shallow language-processing models rather than in deep-learning models. Extracting n-grams is a form of feature engineering, and deep learning does away with this kind of rigid, brittle approach, replacing it with hierarchical feature learning. One-dimensional convnets and recurrent neural networks, introduced later in this chapter, are capable of learning representations for groups of words and characters without being explicitly told about the existence of such groups, by looking at continuous word or character sequences. For this reason, we won't cover n-grams any further in this book. But do keep in mind that they're a powerful, unavoidable feature-engineering tool when using lightweight, shallow text-processing models such as logistic regression and random forests.

کدگذاری تک–داغ کلمات و کاراکترها

ONE-HOT ENCODING OF WORDS AND CHARACTERS

کدگذاری تک–داغ
One-Hot Encoding

به هر توکن (کلمه) یک عدد صحیح یکتا به‌عنوان اندیس نسبت می‌دهیم؛
این اندیس صحیح i را به یک بردار دودویی با طول N (= تعداد کلمات) تبدیل می‌کنیم
که همه‌ی عناصر آن **صفر** است مگر در اندیس i که مقدار آن **یک** است.

کدگذاری تک–داغ می‌تواند در سطح کلمه یا نویسه انجام شود.

کدگذاری تک-داغ کلمات و کاراکترها

کدگذاری تک-داغ در سطح کلمه

Word-level one-hot encoding (toy example)

Builds an index of all tokens in the data

Initial data: one entry per sample (in this example, a sample is a sentence, but it could be an entire document)

Tokenizes the samples via the split method. In real life, you'd also strip punctuation and special characters from the samples.

```
import numpy as np

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

token_index = {}
for sample in samples:
    for word in sample.split():
        if word not in token_index:
            token_index[word] = len(token_index) + 1

max_length = 10

results = np.zeros(shape=(len(samples),
                          max_length,
                          max(token_index.values()) + 1))
for i, sample in enumerate(samples):
    for j, word in list(enumerate(sample.split()))[:max_length]:
        index = token_index.get(word)
        results[i, j, index] = 1.
```

This is where you store the results.

Assigns a unique index to each unique word. Note that you don't attribute index 0 to anything.

Vectorizes the samples. You'll only consider the first max_length words in each sample.

کدگذاری تک-داغ کلمات و کاراکترها

کدگذاری تک-داغ در سطح کاراکتر

Character-level one-hot encoding (toy example)

```
import string

samples = ['The cat sat on the mat.', 'The dog ate my homework.']
characters = string.printable
token_index = dict(zip(range(1, len(characters) + 1), characters))

max_length = 50
results = np.zeros((len(samples), max_length, max(token_index.keys()) + 1))
for i, sample in enumerate(samples):
    for j, character in enumerate(sample):
        index = token_index.get(character)
        results[i, j, index] = 1.
```

All printable ASCII characters

کدگذاری تک-داغ کلمات و کاراکترها

استفاده از کراس برای کدگذاری تک-داغ در سطح کلمه

Using Keras for word-level one-hot encoding

```
from keras.preprocessing.text import Tokenizer

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

tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(samples)

sequences = tokenizer.texts_to_sequences(samples)

one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')

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

Creates a tokenizer, configured to only take into account the 1,000 most common words

Builds the word index

Turns strings into lists of integer indices

How you can recover the word index that was computed

You could also directly get the one-hot binary representations. Vectorization modes other than one-hot encoding are supported by this tokenizer.

کتابخانه‌ی پیشرفته‌ی کراس برای کدگذاری تک-داغ می‌تواند در سطح کلمه یا نویسه عمل کند.
این ابزار از داده‌های متنی خام شروع می‌کند.
دارای امکاناتی چون: حذف نویسه‌های خاص از رشته‌ها، در نظر گرفتن تنها N کلمه‌ی متداول‌تر و ...

کدگذاری تک-داغ کلمات و کاراکترها

استفاده از کراس برای کدگذاری تک-داغ در سطح کلمه با ترفند درهم‌سازی

Word-level one-hot encoding with hashing trick (toy example)

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

dimensionality = 1000
max_length = 10

results = np.zeros((len(samples), max_length, dimensionality))
for i, sample in enumerate(samples):
    for j, word in list(enumerate(sample.split()))[:max_length]:
        index = abs(hash(word)) % dimensionality
        results[i, j, index] = 1.
```

Stores the words as vectors of size 1,000. If you have close to 1,000 words (or more), you'll see many hash collisions, which will decrease the accuracy of this encoding method.

Hashes the word into a random integer index between 0 and 1,000

به‌جای انتساب مستقیم یک اندیس به هر کلمه،
اشاره‌گرهایی به این اندیس‌ها در واژه‌نامه ایجاد می‌کنیم که طول آنها ثابت است.
اشاره‌گر با استفاده از یک تابع درهم‌سازی / Hashing ایجاد می‌شود.
(مناسب برای زمانی که تعداد توکن‌ها بسیار زیاد باشد.)

مزیت: صرفه‌جویی در حافظه / امکان کدگذاری برخط داده‌ها؛ **عیب:** احتمال بروز تصادم در درهم‌سازی

استفاده از جاسازی کلمات

USING WORD EMBEDDINGS

جاسازی کلمات *Word Embedding*	به هر توکن (کلمه) یک بردار ممیزشناور با ابعاد پایین نسبت داده می‌شود.

برخلاف بردارهای کلمات حاصل از کدگذاری تک-داغ، جاسازی‌های کلمات از روی داده‌ها یاد گرفته می‌شوند.

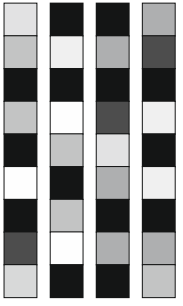
جاسازی‌های متداول: ۲۵۶-بعدی، ۵۱۲-بعدی، ۱۰۲۴-بعدی، ... برای لغت‌نامه‌های بسیار بزرگ

استفاده از جاسازی کلمات

مقایسه‌ی جاسازی کلمات با کدگذاری تک–داغ

جاسازی کلمات
Word Embedding

بردارهای ممیزشناور
بردارهای متراکم
بردارهای با ابعاد پایین
(مثلاً ۲۵۶، ۵۱۲، ۱۰۲۴)
تولید بردارها با یادگیری



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

کدگذاری تک–داغ
One-Hot Encoding

بردارهای دودویی
بردارهای خلوت
بردارهای با ابعاد بالا
(مثلاً ۲۰۰۰۰ به بالا)
تولید بردارها با کدگذاری ثابت

One-hot word vectors:
- Sparse
- High-dimensional
- Hardcoded

استفاده از جاسازی کلمات

روش‌های تولید جاسازی کلمات

روش‌های تولید جاسازی کلمات

یادگیری جاسازی کلمات به‌طور همزمان با وظیفه‌ی اصلی
Learn word embeddings jointly with the main task

با بردارهای کلمات تصادفی شروع می‌کنیم و همزمان با یادگیری وزن‌های شبکه‌ی عصبی، بردارهای کلمات را یاد می‌گیریم.

جاسازی کلمات پیش‌آموزش دیده
Pretrained word embeddings

از جاسازی کلمات که پیش از این با استفاده از یک وظیفه‌ی دیگر یادگیری ماشینی محاسبه شده است، استفاده می‌کنیم.

تکنیک جاسازی توکن‌ها منحصراً برای کلمات به‌کار می‌رود.

استفاده از جاسازی کلمات

روش‌های تولید جاسازی کلمات: یادگیری جاسازی کلمات با لایه‌ی جاسازی

LEARNING WORD EMBEDDINGS WITH THE EMBEDDING LAYER

ساده‌ترین راه برای انتساب یک بردار متراکم به یک کلمه، **انتخاب تصادفی یک بردار** است.

مشکل: فضای جاسازی حاصل هیچ ساختاری ندارد:

(دو کلمه‌ی مترادف ممکن است دارای جاسازی‌های بسیار متفاوتی باشند.

⇐ برای یک شبکه‌ی عصبی عمیق درک این فضای جاسازی بدون ساختار و نویزدار دشوار است)

روابط هندسی میان بردارهای کلمات بایستی منعکس‌کننده‌ی روابط معنایی میان این کلمات باشد.

هدف جاسازی کلمات: **نگاشت زبان انسان به یک فضای هندسی** است.

کلمات هم‌معنی باید در بردارهای کلمات مشابهی جاسازی شوند.

فاصله‌ی هندسی هر دو بردار کلمه باید با فاصله‌ی معنایی میان کلمات متناظر متناسب باشد.

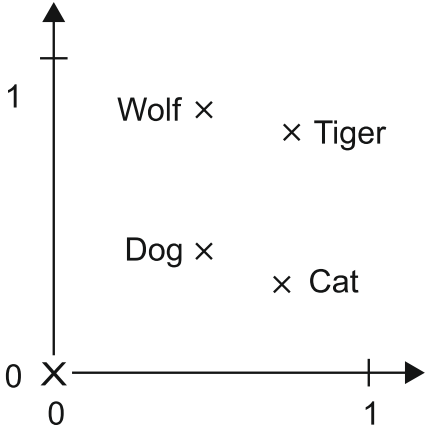
(کلماتی که دارای معانی متفاوتی هستند، به‌صورت نقاطی بسیار دور از یکدیگر جاسازی می‌شوند،

درحالی‌که کلمات مرتبط به یکدیگر نزدیک‌تر هستند)

ممکن است برخی جهت‌های خاص در فضای جاسازی دارای معنای خاصی باشد.

استفاده از جاسازی کلمات

روش‌های تولید جاسازی کلمات: یادگیری جاسازی کلمات با لایه‌ی جاسازی: مثال



چهار کلمه در یک فضای دوبعدی جاسازی شده است: گربه، سگ، گرگ، ببر.

برخی روابط معنایی بین این کلمات

که به‌صورت تبدیل‌های هندسی قابل کدگذاری است:

- بردار یکسانی از گربه به ببر و از سگ به گرگ وجود دارد
  (بردار از حیوان اهلی به حیوان وحشی).
- بردار یکسانی از سگ به گربه و از گرگ به ببر وجود دارد
  (از حیوان سگسان به حیوان گربه‌سان).

نمونه‌های متداول تبدیل‌های هندسی معنادار در فضاهای جاسازی کلمات در دنیای واقعی:

بردارهای جنسیت:

با افزودن یک بردار «مؤنث/female» به بردار «پادشاه/king» به بردار «ملکه/queen» می‌رسیم.

بردارهای جمع:

با افزودن یک بردار «جمع/plural» به بردار «پادشاه/king» به بردار «پادشاهان/kings» می‌رسیم.

استفاده از جاسازی کلمات

روش‌های تولید جاسازی کلمات: یادگیری جاسازی کلمات با لایه‌ی جاسازی

Instantiating an Embedding layer

```
from keras.layers import Embedding

embedding_layer = Embedding(1000, 64)
```

The Embedding layer takes at least two arguments: the number of possible tokens (here, 1,000: 1 + maximum word index) and the dimensionality of the embeddings (here, 64).

روش مناسب این است که برای هر وظیفه‌ی جدید، یک فضای جاسازی جدید یاد گرفته شود.
خوشبختانه الگوریتم پس‌انتشار این کار را ساده می‌کند.
در کراس این کار در قالب یادگیری وزن‌های یک لایه با نام **لایه‌ی جاسازی** انجام می‌شود.

Word index ⟶ Embedding layer ⟶ Corresponding word vector

لایه‌ی جاسازی یک لغت‌نامه است که
اندیس‌های صحیح (نماینده‌ی کلمات خاص) را به بردارهای متراکم نگاشت می‌دهد.
یعنی عدد صحیح را در لغت‌نامه‌ی داخلی جستجو می‌کند و بردار مربوطه را برمی‌گرداند.

استفاده از جاسازی کلمات

روش‌های تولید جاسازی کلمات: یادگیری جاسازی کلمات با لایه‌ی جاسازی: ورودی و خروجی

Word index ⟶ Embedding layer ⟶ Corresponding word vector

لایه‌ی جاسازی، یک تانسور دوبعدی از اعداد صحیح به‌شکل زیر را به‌عنوان ورودی دریافت می‌کند:

`(samples, sequence_length)`

که در آن هر درایه، دنباله‌ای از اعداد صحیح است.

این لایه می‌تواند دنباله‌هایی با طول متغیر را جاسازی کند:

مثلاً دسته‌هایی با شکل (32,10) یا با شکل (64,15) به این لایه قابل تزریق است؛

اما تمام دنباله‌های موجود در یک دسته باید طول یکسانی داشته باشند:

(دنباله‌های کوتاه‌تر باید با صفر پدگذاری شوند و دنباله‌های بلندتر باید کوتاه شوند.)

لایه‌ی جاسازی، یک تانسور سه‌بعدی از اعداد ممیزشناور به‌شکل زیر را به‌عنوان خروجی تحویل می‌دهد:

`(samples, sequence_length, embedding_dimensionality)`

این تانسور سه‌بعدی را می‌توان با یک لایه‌ی RNN یا یک لایه‌ی کانوولوشنی یک‌بعدی پردازش کرد.

وزن‌های این لایه در ابتدا تصادفی هستند.

در طی فرآیند آموزش، این بردارهای کلمات از طریق پس‌انتشار به‌تدریج تنظیم می‌شوند

و ساختار فضا را به چیزی تبدیل می‌کنند که مدل پایین‌دست می‌تواند از آن بهره‌مند شود.

استفاده از جاسازی کلمات

مثال: طبقه‌بندی نقد فیلم‌ها: بارگذاری داده‌ها

Loading the IMDB data for use with an Embedding layer

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

max_features = 10000
maxlen = 20

(x_train, y_train), (x_test, y_test) = imdb.load_data(
    num_words=max_features)

x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

Number of words to consider as features

Cuts off the text after this number of words (among the max_features most common words)

Loads the data as lists of integers

Turns the lists of integers into a 2D integer tensor of shape (samples, maxlen)

نقد فیلم‌ها را به ۱۰۰۰۰ عدد از متداول‌ترین کلمات محدود می‌کنیم.

از هر نقد ۲۰ کلمه‌ی اول را نگه می‌داریم و بقیه را حذف می‌کنیم.

استفاده از جاسازی کلمات

مثال: طبقه‌بندی نقد فیلم‌ها: استفاده از لایه‌ی جاسازی و طبقه‌بندی‌کننده

Using an Embedding layer and classifier on the IMDB data

Specifies the maximum input length to the Embedding layer so you can later flatten the embedded inputs. After the Embedding layer, the activations have shape (samples, maxlen, 8).

Flattens the 3D tensor of embeddings into a 2D tensor of shape (samples, maxlen * 8)

Adds the classifier on top

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

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)
```

این شبکه برای هر یک از ۱۰۰۰۰ کلمه، جاسازی‌های ۸–بعدی را یاد می‌گیرد؛
دنباله‌های صحیح ورودی را به دنباله‌های جاسازی‌شده‌ی تبدیل می‌کند؛
تانسور دوبعدی را مسطح می‌کند و یک لایه‌ی Dense را در بالا برای طبقه‌بندی آموزش می‌دهد.

دقت اعتبارسنجی: تقریباً ۷۶ درصد

استفاده از جاسازی کلمات

روش‌های تولید جاسازی کلمات : استفاده از جاسازی کلمات پیش‌آموزش دیده

USING PRETRAINED WORD EMBEDDINGS

در برخی موارد، داده‌های آموزشی موجود به‌حدی اندک هستند که
نمی‌توانیم از این داده‌ها به‌تنهایی برای یادگیری یک جاسازی مناسب مختص وظیفه برای لغت‌نامه‌ی خود استفاده کنیم.

به‌جای یادگیری جاسازی کلمات همزمان با مسئله‌ای که می‌خواهیم آن را حل کنیم،
می‌توانیم بردارهای جاسازی‌شده از یک فضای جاسازی از پیش‌محاسبه شده را بارگذاری کنیم.
(یادگیری انتقالی)

استفاده از جاسازی کلمات

الگوریتم‌های متداول، جاسازی‌های از پیش‌آموزش دیده

الگوریتم کلمه–به–بردار
Word2Vec Algorithm

یک فضای جاسازی متراکم با ابعاد پایین برای کلمات که به‌شیوه‌ی غیرنظارتی محاسبه شده است.
گوگل، ۲۰۱۳

https://code.google.com/archive/p/word2vec

بردارهای سراسری برای بازنمایی کلمه
GloVe (Global Vector for Word Representation)

تکنیک جاسازی مبتنی بر فاکتورگیری از یک ماتریس حاوی آمارهای هم‌وقوعی کلمات.
دانشگاه استنفورد، ۲۰۱۴

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

از هر یک از پایگاه‌های داده‌ی فوق می‌توان در لایه‌ی جاسازی کراس استفاده کرد.

جمع‌بندی: از متن خام تا جاسازی کلمات

مثال طبقه‌بندی نقد فیلم‌ها

PUTTING IT ALL TOGETHER: FROM RAW TEXT TO WORD EMBEDDINGS

الگویی مشابه قبل:

جاسازی جملات در دنباله‌هایی از بردارها،

مسطح‌سازی بردارها،

آموزش یک لایه‌ی متراکم در بالا

از جاسازی کلمات پیش‌آموزش دیده GloVe استفاده می‌کنیم.

از داده‌های متنی اصلی IMDB استفاده می‌کنیم.

جمع‌بندی: از متن خام تا جاسازی کلمات

مثال طبقه‌بندی نقد فیلم‌ها: پردازش برچسب‌های داده‌های خام IMDB

Downloading the IMDB Data As Raw Text

دانلود پایگاه داده‌ی خام IMDB از http://mng.bz/0tIo و خارج کردن آن از حالت فشرده.
برای هر نقد، یک رشته (همه‌ی نقدها: لیستی از رشته‌ها)؛ برچسب نقدها را درون لیست `labels` قرار می‌دهیم.

Processing the labels of the raw IMDB data

```
import os

imdb_dir = '/Users/fchollet/Downloads/aclImdb'
train_dir = os.path.join(imdb_dir, 'train')

labels = []
texts = []

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

جمع‌بندی: از متن خام تا جاسازی کلمات

مثال طبقه‌بندی نقد فیلم‌ها: توکن‌بندی متن داده‌های خام IMDB

TOKENIZING THE DATA

متن را برداری‌سازی می‌کنیم و جداسازی داده‌های آموزشی و اعتبارسنجی را انجام می‌دهیم.
محدودسازی داده‌های آموزشی به ۲۰۰ نمونه‌ی اول
(برای سودمند واقع شدن بیشتر جاسازی از پیش‌آموزش دیده درشرایطی که داده‌های آموزشی اندکی موجود است؛
در غیر این صورت جاسازی‌های مختص آن وظیفه، احتمالاً آنها را بهتر اجرا می‌کنند.)

Tokenizing the text of the raw IMDB data

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

maxlen = 100                 <-- Cuts off reviews after 100 words
training_samples = 200       <-- Trains on 200 samples
validation_samples = 10000   <-- Validates on 10,000 samples
max_words = 10000            <-- Considers only 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)
```

جمع‌بندی: از متن خام تا جاسازی کلمات

مثال طبقه‌بندی نقد فیلم‌ها: توکن‌بندی متن داده‌های خام IMDB (ادامه)

Tokenizing the text of the raw IMDB data (Cntd.)

```
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)

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]
```

Splits the data into a training set and a validation set, but first shuffles the data, because you're starting with data in which samples are ordered (all negative first, then all positive)

جمع‌بندی: از متن خام تا جاسازی کلمات

دانلود جاسازی کلمات GloVe و پیش‌پردازش جاسازی‌ها

DOWNLOADING THE GLOVE WORD EMBEDDINGS; PREPROCESSING THE EMBEDDINGS

دانلود GloVe پیش‌محاسبه شده از ویکی‌پدیای انگلیسی ۲۰۱۴ با حجم ۸۲۲ مگابایت (glove.6B.zip)
از نشانی: https://nlp.stanford.edu/projects/glove
حاوی بردارهای جاسازی ۱۰۰-بعدی برای ۴۰۰۰۰۰ کلمه (یا توکن‌های غیرکلمه‌ای)، Unzip کردن آن.
پارس کردن فایل متنی حاصل و ایجاد اندیسی برای نگاشت کلمات به بازنمایی‌های برداری آنها.

Parsing the GloVe word-embeddings file

```
glove_dir = '/Users/fchollet/Downloads/glove.6B'

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))
```

جمع‌بندی: از متن خام تا جاسازی کلمات

آماده‌سازی ماتریس جاسازی کلمات GloVe

Preparing the GloVe word-embeddings matrix

```
embedding_dim = 100

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

Words not found in the embedding index will be all zeros.

یک ماتریس جاسازی برای بارگذاری در لایه‌ی Embedding می‌سازیم به شکل:

`(max_words, embedding_dim)`

که در آن هر درایه i، حاوی یک بردار `embedding_dim`–بعدی برای کلمه با اندیس i در شاخص کلمه‌ی مرجع است.

اندیس صفر برای هیچ کلمه یا توکنی در نظر گرفته نمی‌شود و یک جانگهدار است.

جمع‌بندی: از متن خام تا جاسازی کلمات

تعریف یک مدل

Model definition

```
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'))
model.summary()
```

از همان معماری مدل پیشین استفاده می‌کنیم.

جمع‌بندی: از متن خام تا جاسازی کلمات

بارگذاری جاسازی‌های GloVe در مدل

Loading pretrained word embeddings into the `Embedding` layer

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

لایه‌ی Embedding دارای یک ماتریس وزن تنها است:
یک ماتریس دوبعدی ممیزشناور که هر درایه‌ی i در آن بردار کلمه‌ای است که باید به اندیس i متناظر شود.

ماتریس GloVe را که درون لایه‌ی Embedding آماده کردیم، در اولین لایه در مدل بارگذاری می‌کنیم.

به‌علاوه، لایه‌ی Embedding را منجمد می‌کنیم:
بخش‌های پیش‌آموزش دیده نباید در حین آموزش به‌روزرسانی شوند.

جمع‌بندی: از متن خام تا جاسازی کلمات

آموزش و ارزیابی مدل

Training and evaluation

```
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')
```

مدل را کامپایل می‌کنیم و مورد آموزش قرار می‌دهیم.

جمع‌بندی: از متن خام تا جاسازی کلمات

ترسیم نتایج

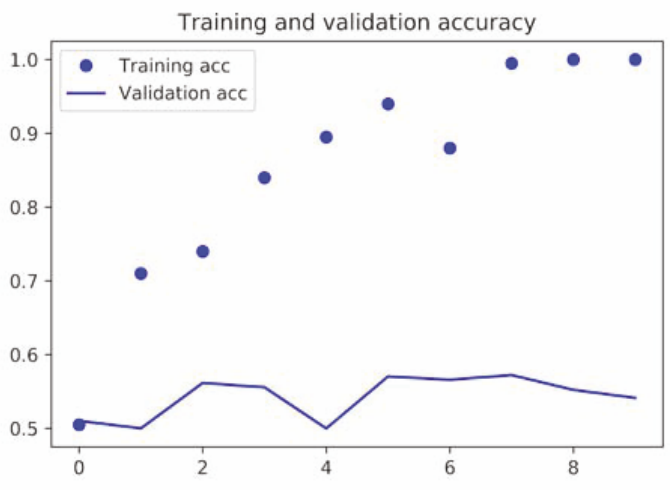
Plotting the results

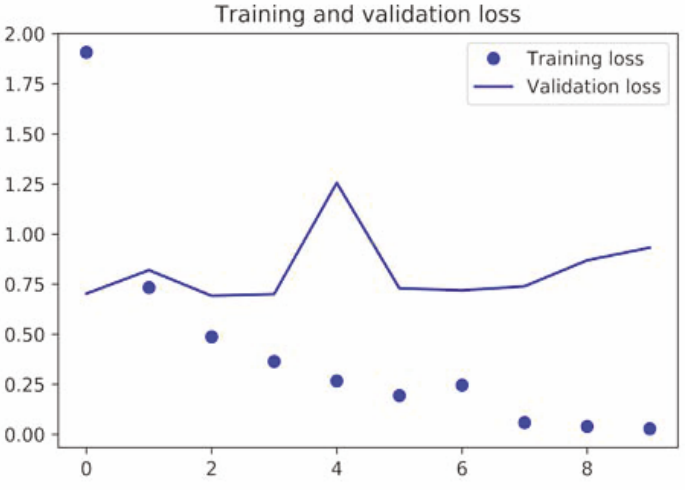
```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

عملکرد مدل را در طول زمان ترسیم می‌کنیم.

جمع‌بندی: از متن خام تا جاسازی کلمات

نمودارهای اتلاف و دقت آموزش و اعتبارسنجی با استفاده از جاسازی‌های کلمات از پیش‌آموزش دیده





این مدل به‌سرعت شروع به بیش‌برازش می‌کند
که با توجه به میزان اندک داده‌های نمونه‌ی آموزشی دور از انتظار نیست.
به همین دلیل، دقت اعتبارسنجی دارای واریانس بالایی است اما به‌نظر می‌رسد به نیمه‌ی بالا می‌رسد.

(می‌توانیم با انتخاب تصادفی ۲۰۰ نمونه‌ی دیگر، عملکرد را آزمایش کنیم.)

جمع‌بندی: از متن خام تا جاسازی کلمات

آموزش بدون بارگذاری جاسازی‌های از پیش‌آموزش دیده‌ی کلمات

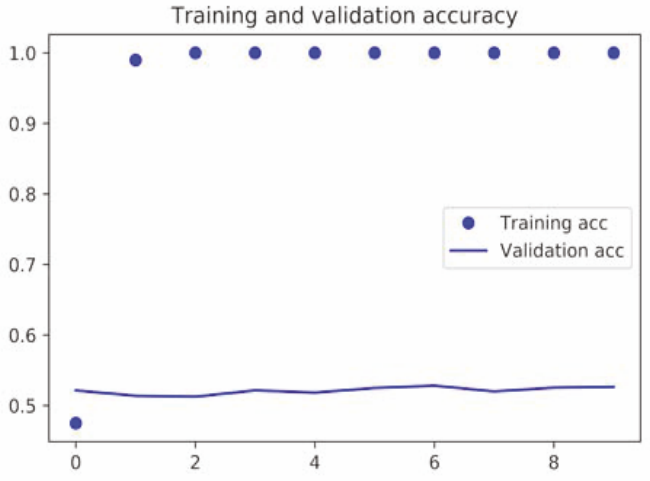
Training the same model without pretrained word embeddings

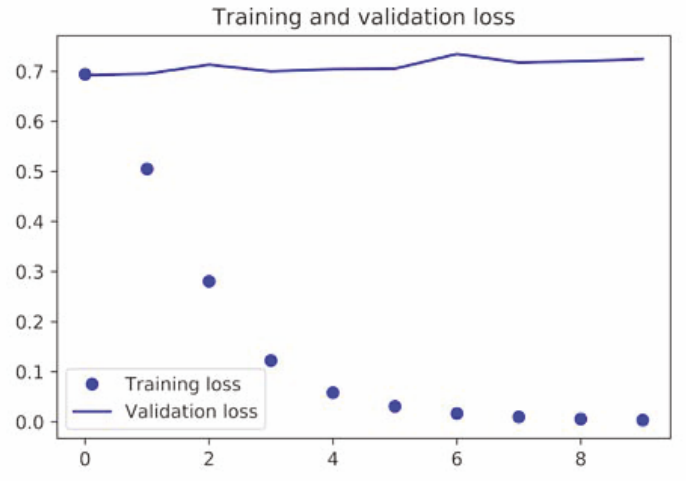
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

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'))
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))
```

لایه‌ی جاسازی را منجمد نمی‌کنیم و جاسازی‌های ازپیش‌آموزش دیده را بارگذاری نمی‌کنیم.
در این حالت یک جاسازی کلمات مختص این وظیفه برای توکن‌های ورودی یاد گرفته می‌شود
(که به‌طور کلی بسیار قدرت‌مندتر از جاسازی‌های کلمات از پیش‌آموزش دیده برای زمانی است که داده‌های زیادی موجود باشد).

جمع‌بندی: از متن خام تا جاسازی کلمات

نمودارهای اتلاف و دقت آموزش و اعتبارسنجی بدون استفاده از جاسازی‌های کلمات از پیش‌آموزش دیده





دقت اعتبارسنجی در نیمه‌ی پایین متوقف می‌شود.
بنابراین در این مورد، جاسازی‌های از پیش‌آموزش دیده‌ی کلمات
بهتر از جاسازی‌های یادگیری شده به‌صورت همزمان عمل می‌کنند.

اگر تعداد نمونه‌های آموزشی را افزایش بدهیم، این نتیجه معکوس می‌شود.

جمع‌بندی: از متن خام تا جاسازی کلمات

توکن‌بندی داده‌های مجموعه‌ی آزمایشی

Tokenizing the data of the test set

```
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))
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)

sequences = tokenizer.texts_to_sequences(texts)
x_test = pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(labels)
```

برای ارزیابی مدل روی داده‌های آزمایشی، ابتدا باید این داده‌ها را بر اساس توکن‌ها بخش‌بندی کنیم.

جمع‌بندی: از متن خام تا جاسازی کلمات

ارزیابی مدل بر روی مجموعه‌ی آزمایشی

Evaluating the model on the test set

```
model.load_weights('pre_trained_glove_model.h5')
model.evaluate(x_test, y_test)
```

مدل اول را بارگذاری و ارزیابی می‌کنیم.

به دقت دلسرد کننده‌ی ۵۶% دست پیدا می‌کنیم.
کار با نمونه‌های آموزشی اندک دشوار است!!

جمع‌بندی: از متن خام تا جاسازی کلمات

خلاصه

WRAPPING UP

Now you're able to do the following:

- Turn raw text into something a neural network can process
- Use the Embedding layer in a Keras model to learn task-specific token embeddings
- Use pretrained word embeddings to get an extra boost on small natural-language-processing problems

یادگیری عمیق برای متن و دنباله‌ها

۲

فهم شبکه‌های عصبی بازگشتی

فهم شبکه‌های عصبی بازگشتی

UNDERSTANDING RECURRENT NEURAL NETWORKS

شبکه‌های عصبی مانند شبکه‌های متصل متراکم و شبکه‌های کانوولوشنال، دارای حافظه نیستند:
هر ورودی ارائه شده به آنها به‌صورت مستقل پردازش می‌شود
و هیچ حالتی بین ورودی‌ها نگهداری نمی‌شود.

⇓

برای پردازش دنباله یا سری زمانی، باید کل دنباله را یکجا به شبکه ارائه بدهیم (تبدیل کل دنباله به یک نقطه داده)
(**شبکه‌های پیش‌خور**: ***Feedforward Networks***)

اما،
هوش بیولوژیکی، اطلاعات را به‌صورت تدریجی پردازش می‌کند:
یک **مدل داخلی** از آنچه پردازش می‌شود، نگهداری می‌شود که از اطلاعات گذشته ساخته شده است
و هنگامی که اطلاعات جدید وارد می‌شود، دائماً به‌روزرسانی می‌شود.

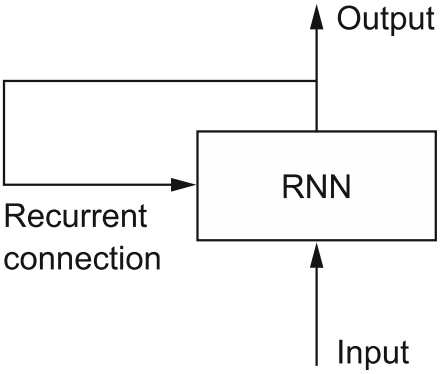
مانند خواندن یک جمله، کلمه به کلمه ...

فهم شبکه‌های عصبی بازگشتی

ساختار RNN

یک شبکه‌ی عصبی بازگشتی،
دنباله‌ها را با انجام تکرار در عناصر دنباله پردازش می‌کند
و یک **حالت** حاوی اطلاعات مرتبط با آنچه تاکنون مشاهده کرده است را نگهداری می‌کند:
RNN یک نوع شبکه‌ی عصبی است که یک حلقه‌ی داخلی دارد.

* حالت RNN میان پردازش دو دنباله‌ی مستقل و متفاوت بازنشانی/reset می‌شود.
⇐ یک دنباله را همچنان یک نقطه‌ی داده‌ای منفرد در نظر می‌گیریم،
اما این نقطه‌ی داده‌ای دیگر طی یک مرحله‌ی واحد پردازش نمی‌شود،
بلکه در عوض شبکه حلقه‌ی تکرار را به‌صورت داخلی روی عناصر دنباله انجام می‌دهد.



فهم شبکه‌های عصبی بازگشتی

شبکه کد RNN

Pseudocode RNN

```
state_t = 0                          <---- The state at t
for input_t in input_sequence:       <---- Iterates over sequence elements
    output_t = f(input_t, state_t)
    state_t = output_t               <---- The previous output becomes the state for the next iteration.
```

شبکه، حلقه را در گام‌های زمانی می‌پیماید
و در هر گام زمانی، حالت کنونی خود در t و ورودی در t را در نظر می‌گیرد؛
آنها را توسط تا بع f ترکیب می‌کند تا خروجی در لحظه‌ی t را به‌دست آورد.
سپس حالت را برای گام بعدی تنظیم می‌کند.

برای اولین گام زمانی، خروجی قبلی و حالت فعلی موجود نیست.
⇐ حالت آغازین را به‌صورت یک بردار تماماً صفر مقداردهی می‌کنیم (حالت آغازین شبکه)

فهم شبکه‌های عصبی بازگشتی

شبکه کد مفصل‌تر RNN

More detailed pseudocode for the RNN

```
state_t = 0
for input_t in input_sequence:
    output_t = activation(dot(W, input_t) + dot(U, state_t) + b)
    state_t = output_t
```

می‌توانیم تابع f را با جزئیات بیشتر ارائه کنیم:
تبدیل ورودی و حالت به یک خروجی، با دو ماتریس $\boldsymbol{W}$ و $\boldsymbol{U}$ و یک بردار بایاس $\boldsymbol{b}$ پارامتری می‌شود.

فهم شبکه‌های عصبی بازگشتی

پیاده‌سازی یک RNN ساده در Numpy

Numpy implementation of a simple RNN

```
import numpy as np

timesteps = 100
input_features = 32
output_features = 64

inputs = np.random.random((timesteps, input_features))

state_t = np.zeros((output_features,))

W = np.random.random((output_features, input_features))
U = np.random.random((output_features, output_features))
b = np.random.random((output_features,))

successive_outputs = []
for input_t in inputs:
    output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)

    successive_outputs.append(output_t)

    state_t = output_t

final_output_sequence = np.concatenate(successive_outputs, axis=0)
```

Number of timesteps in the input sequence

Dimensionality of the input feature space

Dimensionality of the output feature space

Input data: random noise for the sake of the example

Initial state: an all-zero vector

Creates random weight matrices

input_t is a vector of shape (input_features,).

Stores this output in a list

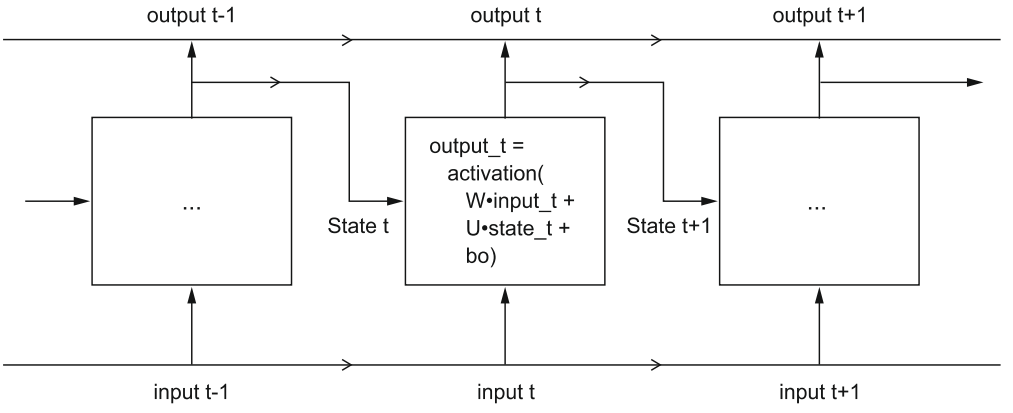
Combines the input with the current state (the previous output) to obtain the current output

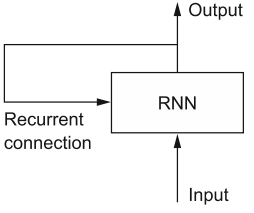
The final output is a 2D tensor of shape (timesteps, output_features).

Updates the state of the network for the next timestep

فهم شبکه‌های عصبی بازگشتی

تاگشایی / بازکردن یک RNN ساده





هر گام زمانی در خروجی، حاوی اطلاعاتی درباره‌ی گام‌های زمانی 0 تا t (کل گذشته) است.

⇐ در بسیاری از موارد به دنباله‌ی کامل خروجی نیاز نداریم و تنها به آخرین خروجی نیاز داریم.

یک لایه‌ی بازگشتی در کراس

A Recurrent Layer in Keras

کد پیاده‌سازی شده، معادل با یک لایه‌ی واقعی در کراس است:

```
from keras.layers import SimpleRNN
```

با این تفاوت که این لایه، دسته‌هایی از دنباله‌ها را پردازش می‌کند (نه یک دنباله‌ی تنها).

شکل ورودی: ⇐

```
(batch_size, timesteps, input_features)
```

لایه‌های بازگشتی را می‌توان به دو شیوه‌ی متفاوت اجرا کرد:

* **برگرداندن دنباله‌های کاملی از خروجی‌های متوالی برای هر گام زمانی:**

```
(batch_size, timesteps, output_features)
```

* **برگرداندن آخرین خروجی برای هر دنباله‌ی ورودی:**

```
(batch_size, output_features)
```

این دو شیوه از طریق آرگومان سازنده‌ی `return_sequences` کنترل می‌شود.

یک لایه‌ی بازگشتی در کراس

یک مثال از به‌کارگیری SimpleRNN و بازگرداندن تنها یک خروجی در آخرین گام زمانی

```
>>> from keras.models import Sequential
>>> from keras.layers import Embedding, SimpleRNN
>>> model = Sequential()
>>> model.add(Embedding(10000, 32))
>>> model.add(SimpleRNN(32))
>>> model.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_22 (Embedding)     (None, None, 32)          320000
_________________________________________________________________
simplernn_10 (SimpleRNN)     (None, 32)                2080
=================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
```

یک لایه‌ی بازگشتی در کراس

یک مثال از به‌کارگیری SimpleRNN و بازگرداندن دنباله‌ی کامل حالت‌ها

```
>>> model = Sequential()
>>> model.add(Embedding(10000, 32))
>>> model.add(SimpleRNN(32, return_sequences=True))
>>> model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_23 (Embedding)     (None, None, 32)          320000
_________________________________________________________________
simplernn_11 (SimpleRNN)     (None, None, 32)          2080
=================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
```

یک لایه‌ی بازگشتی در کراس

یک مثال از به‌کارگیری پشته‌ای از چند لایه‌ی بازگشتی (برای افزایش توان بازنمایی شبکه)

لازم است تمام لایه‌های بازگشتی میانی، دنباله‌ی کامل خروجی را برگردانند.

```
>>> model = Sequential()
>>> model.add(Embedding(10000, 32))
>>> model.add(SimpleRNN(32, return_sequences=True))
>>> model.add(SimpleRNN(32, return_sequences=True))
>>> model.add(SimpleRNN(32, return_sequences=True))
>>> model.add(SimpleRNN(32))
>>> model.summary()
```

Last layer only returns the last output

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_24 (Embedding)     (None, None, 32)          320000
_________________________________________________________________
simplernn_12 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simplernn_13 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simplernn_14 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simplernn_15 (SimpleRNN)     (None, 32)                2080
=================================================================
Total params: 328,320
Trainable params: 328,320
Non-trainable params: 0
```

یک لایه‌ی بازگشتی در کراس

مثال: استفاده از یک مدل بازگشتی برای طبقه‌بندی نقد فیلم‌ها: پیش‌پردازش داده‌ها

Preparing the IMDB data

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

max_features = 10000
maxlen = 500
batch_size = 32

print('Loading data...')
(input_train, y_train), (input_test, y_test) = imdb.load_data(
     num_words=max_features)
print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')

print('Pad sequences (samples x time)')
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)
print('input_train shape:', input_train.shape)
print('input_test shape:', input_test.shape)
```

Number of words to consider as features

Cuts off texts after this many words (among the max_features most common words)

یک لایه‌ی بازگشتی در کراس

مثال: استفاده از یک مدل بازگشتی برای طبقه‌بندی نقد فیلم‌ها: آموزش مدل با لایه‌های Embedding و SimpleRNN

Training the model with Embedding and SimpleRNN layers

```
from keras.layers import Dense

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=128,
                     validation_split=0.2)
```

یک لایه‌ی بازگشتی در کراس

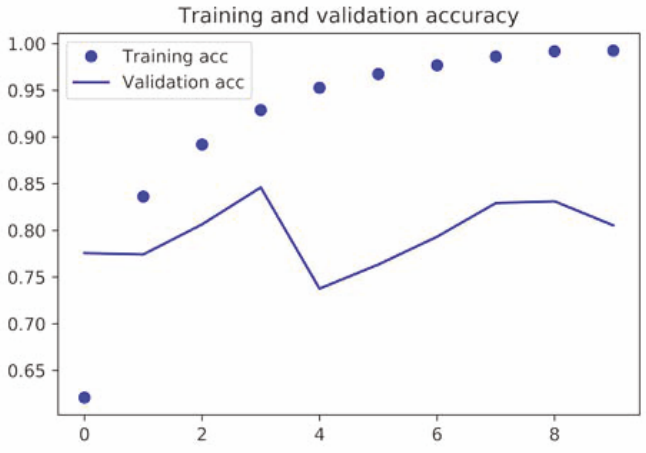
مثال: استفاده از یک مدل بازگشتی برای طبقه‌بندی نقد فیلم‌ها: ترسیم نتایج؛ نمودارهای اتلاف و دقت آموزش و اعتبارسنجی

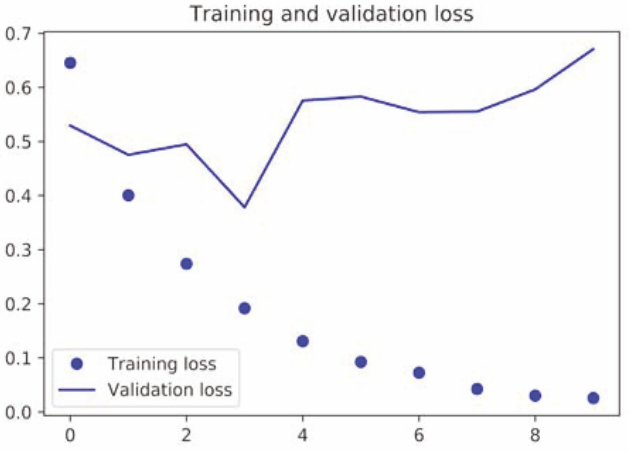
Plotting results

```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

یک لایه‌ی بازگشتی در کراس

مثال: استفاده از یک مدل بازگشتی برای طبقه‌بندی نقد فیلم‌ها: نمودارهای اتلاف و دقت آموزش و اعتبارسنجی





این شبکه‌ی بازگشتی کوچک در مقایسه با خط پایه (فصل۳: ۸۸%) عملکرد خوبی نشان نمی‌دهد: ۸۵%.
بخشی از مشکل: استفاده از ۵۰۰ کلمه‌ی اول به‌جای دنباله‌های کامل
بقیه‌ی مشکل: RNN ساده در پردازش دنباله‌های طولانی مانند متن عملکرد خوبی ندارد.

دیگر انواع لایه‌های بازگشتی عملکرد بهتری دارند.

فهم لایه‌های LSTM و GRU

UNDERSTANDING THE LSTM AND GRU LAYERS

RNN ساده بیش از حد ساده است و برای کاربردهای واقعی چندان مناسب نیست:

اگرچه از لحاظ تئوری باید بتواند در زمان t اطلاعاتی در مورد ورودی‌هایی که چندین گام قبل‌تر دیده است حفظ کند،
در عمل یادگیری این وابستگی‌های بلندمدت غیرممکن است:
به‌دلیل مشکل ناشی از اضمحلال گرادیان‌ها

لایه‌های LSTM و GRU برای حل این مسئله طراحی شدند.

مشکل اضمحلال گرادیان‌ها در شبکه‌هایی با تعداد لایه‌های بالا هم رخ می‌دهد:
هنگامی که به افزودن لایه‌ها به یک شبکه ادامه می‌دهیم، آن شبکه در نهایت غیر قابل آموزش می‌شود.

See, for example, Yoshua Bengio, Patrice Simard, and Paolo Frasconi, “Learning Long-Term Dependencies with Gradient Descent Is Difficult,” *IEEE Transactions on Neural Networks* 5, no. 2 (1994).

لایه‌ی LSTM

حافظه‌ی کوتاه–مدت طولانی

LSTM Layer: Long Short-Term Memory

الگوریتم پایه‌ی حافظه‌ی کوتاه–مدت طولانی (LSTM)
در ۱۹۹۷ توسط **هوچرایتر** و **اشمیدهابر** توسعه یافت.

این لایه شکل تغییر یافته‌ای از RNN ساده است:
این لایه روشی برای حمل اطلاعات طی چندین گام زمانی اضافه می‌کند:

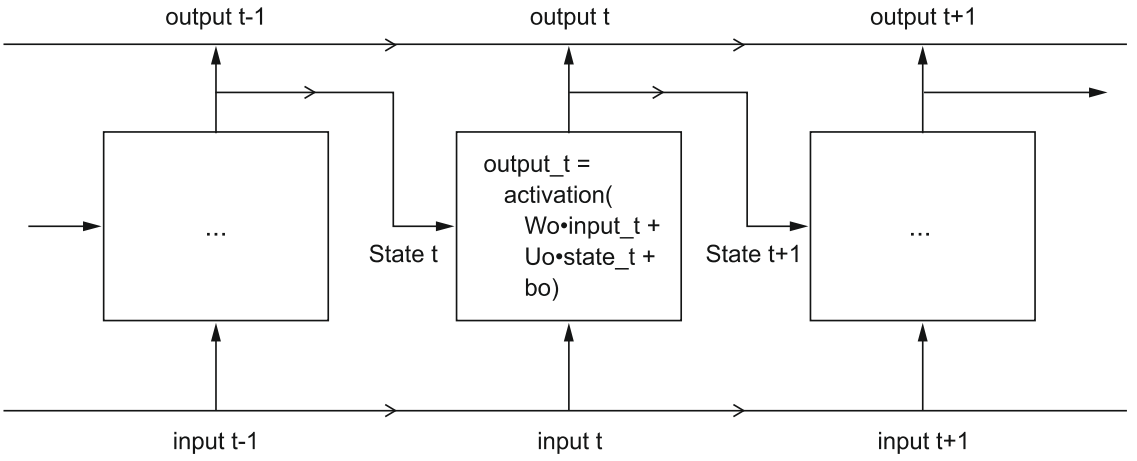
یک نوار حامل (تسمه نقاله) را در نظر می‌گیریم که موازی با دنباله‌ی مورد پردازش حرکت می‌کند.
اطلاعات مربوط به این دنباله، می‌تواند در هر نقطه‌ای روی این نوار حامل پرش کند
و به یک گام زمانی بعدی منتقل شود و سالم و دست‌نخورده از آن خارج شود.
⇐ از اضمحلال تدریجی سیگنال‌های قدیمی‌تر جلوگیری می‌شود.

Sepp Hochreiter and Jürgen Schmidhuber, “Long Short-Term Memory,” Neural Computation 9, no. 8 (1997).

لایه‌ی LSTM

شروع از لایه‌ی RNN ساده

LSTM Layer



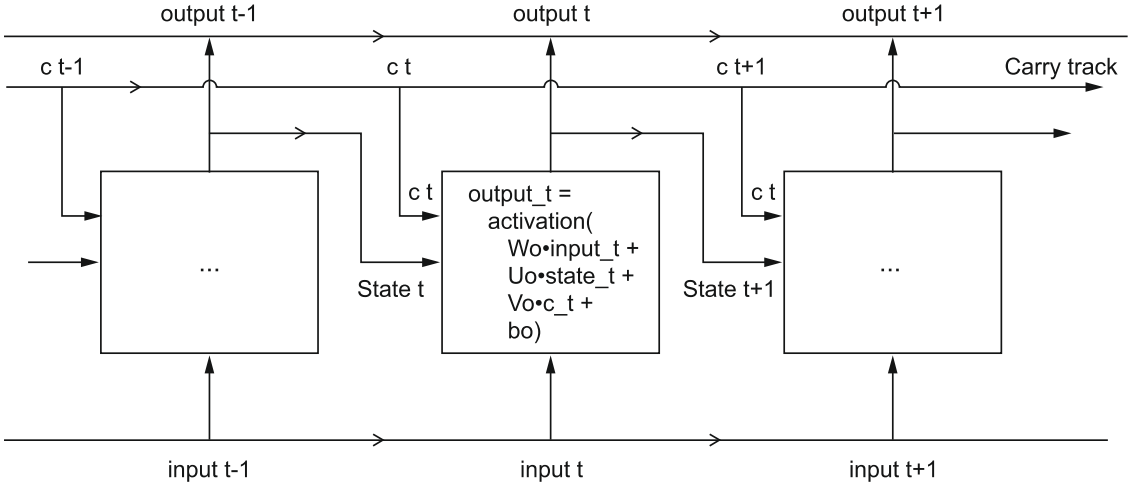
از یک سلول SimpleRNN شروع می‌کنیم.

(اندیس‌گذاری ماتریس‌های وزن خروجی با o)

لایه‌ی LSTM

اضافه کردن تسمه نقاله

LSTM Layer



به ساختار SimpleRNN یک گردش جریان دیگر اضافه می‌کنیم: تسمه نقاله
برای حمل اطلاعات در گام‌های زمانی.

(حالت حمل: مقادیر تسمه نقاله برای گام‌های زمانی مختلف: c_t برای $carry$)

اثر این اطلاعات بر سلول: اتصال ورودی و اتصال بازگشتی از طریق یک تبدیل متراکم ترکیب می‌شوند
و مقدار حالت ارسالی به گام بعدی را تحت تاثیر قرار می‌دهند.

گردش داده‌ی حامل، روشی برای تعدیل خروجی بعدی و حالت بعدی است.

لایه‌ی LSTM

بخش زیرکانه

LSTM Layer

روشی که مقدار بعدی گردش داده‌ی حامل محاسبه می‌شود، شامل سه تبدیل جداگانه است.
هر سه تبدیل به‌شکل یک سلول SimpleRNN است.

Pseudocode details of the LSTM architecture (1/2)

```
output_t = activation(dot(state_t, Uo) + dot(input_t, Wo) + dot(C_t, Vo) + bo)

i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)
```

حالت حمل جدید (c_t بعدی) با ترکیب i_t، f_t و k_t به‌دست می‌آید.

Pseudocode details of the LSTM architecture (2/2)

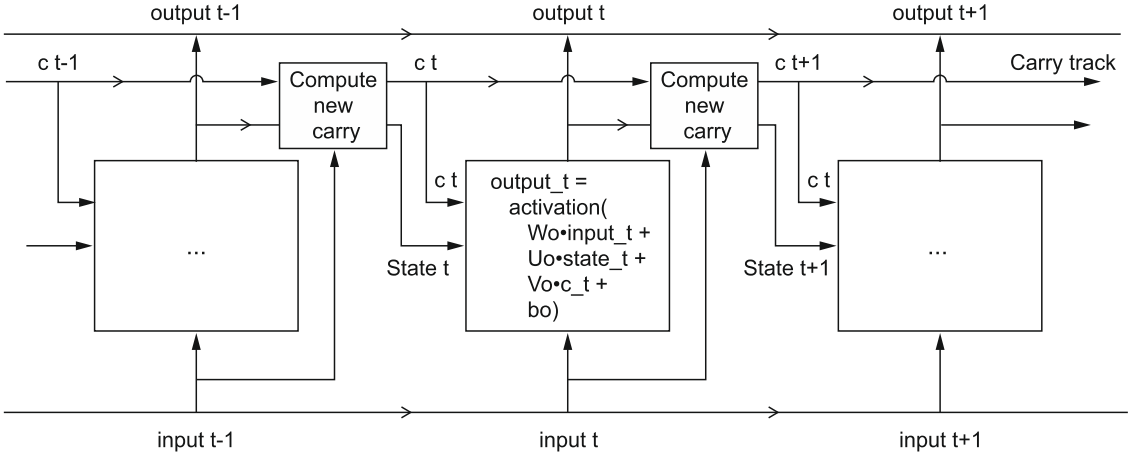
```
c_t+1 = i_t * k_t + c_t * f_t
```

- ضرب c_t و f_t روشی برای فراموش کردن عمدی اطلاعات نامرتبط در گردش داده‌ی حامل است.
- i_t و k_t اطلاعاتی در مورد زمان کنونی فراهم می‌کنند و مسیر حامل را با اطلاعات جدید به‌روزرسانی می‌کنند.

لایه‌ی LSTM

آناتومی لایه

LSTM Layer



در مجموع LSTM به ما این امکان را می‌دهد که:

اطلاعات قبلی در زمانی دیرتر دوباره تزریق شود؛

و از این طریق با مسئله‌ی اضمحلال گرادیان مبارزه می‌شود.

لایه‌ی LSTM

یک مثال واقعی: طبقه‌بندی نقد فیلم‌ها

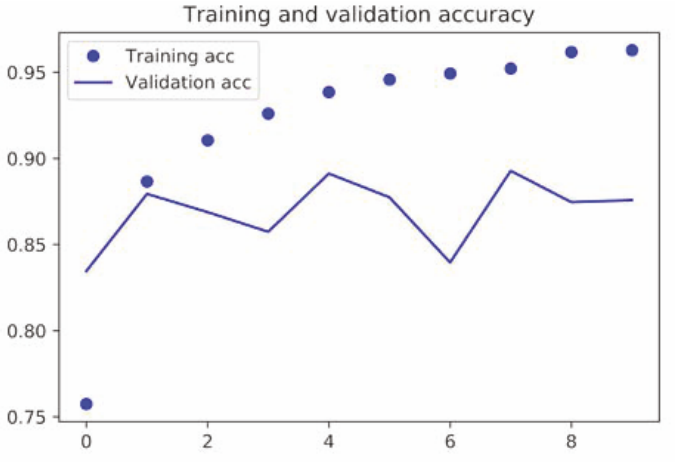
راه‌اندازی یک مدل با استفاده از لایه‌ی LSTM و آموزش آن با داده‌های IMDB.
فقط ابعاد خروجی لایه‌ی LSTM را تعیین می‌کنیم و سایر آرگومان‌ها را با مقادیر پیش‌فرض باقی می‌گذاریم.

Using the LSTM layer in Keras

```
from keras.layers import LSTM

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

لایه‌ی LSTM

یک مثال واقعی: طبقه‌بندی نقد فیلم‌ها: نتایج



این بار به دقت اعتبارسنجی تا مقدار ۸۹% رسیدیم.

بهتر شدن نسبت به RNN ساده به‌دلیل کمتر قرار گرفتن در معرض اضمحلال گرادیان‌هاست.

نتیجه کمی بهتر از روی‌کرد متراکم متصل در فصل ۳ است در حالی که از حجم داده‌ی کمتری استفاده کرده‌ایم.

(پس از ۵۰۰ گام زمانی دنباله‌ها را قطع کردیم، اما در فصل ۳ کل دنباله در نظر گرفته می‌شد.)

برای بهبود نتایج: ❋ تنظیم هایپرپارامترها: ابعاد جاسازی، ابعاد خروجی LSTM، ... ❋ اضافه کردن رگولاریزاسیون

صادقانه: تحلیل ساختاری و طولانی‌مدت نقدها (نقطه قوت *LSTM*) برای مسئله‌ی *sentiment analysis* مناسب نیست.
روش بهتر، بررسی کلمات موجود در نقد و فراوانی آنها (مشابه روش مورد استفاده در روی‌کرد متصل متراکم فصل ۳)

جمع‌بندی: فهم شبکه‌های عصبی بازگشتی

خلاصه

WRAPPING UP

Now you understand the following:

- What RNNs are and how they work
- What LSTM is, and why it works better on long sequences than a naive RNN
- How to use Keras RNN layers to process sequence data

یادگیری عمیق برای متن و دنباله‌ها

۳

استفاده‌ی پیشرفته از شبکه‌های عصبی بازگشتی

استفاده‌ی پیشرفته از شبکه‌های عصبی بازگشتی

ADVANCED USE OF RECURRENT NEURAL NETWORKS

سه تکنیک پیشرفته برای بهبود عملکرد و توان تعمیم‌دهی شبکه‌های عصبی بازگشتی

برون‌اندازی بازگشتی
Recurrent Dropout

شیوه‌ای خاص و پیش‌ساخته برای استفاده از برون‌اندازی برای مقابله با بیش‌برازش در لایه‌های بازگشتی

پشته‌سازی از لایه‌های بازگشتی
Stacking Recurrent Layers

برای افزایش توان بازنمایی شبکه (به‌قیمت بار محاسباتی بیشتر)

لایه‌های بازگشتی دوطرفه
Bidirectional Recurrent Layers

این لایه‌ها اطلاعات یکسانی را به شیوه‌های گوناگون در اختیار شبکه‌ی بازگشتی می‌گذارد.
⇐ افزایش دقت + تسکین مسئله‌ی فراموشی

مسئله‌ی پیش‌بینی دما

A TEMPERATURE-FORECASTING PROBLEM

مسئله‌ی پیش‌بینی دما:

به یک سری زمانی از نقاط داده‌ای حاصل از حسگرهای نصب‌شده در سقف یک ساختمان دسترسی داریم
(مانند دما، فشار، میزان رطوبت) و می‌خواهیم از آنها برای پیش‌بینی دما طی ۲۴ ساعت پس از نقطه‌ی داده‌ای قبلی استفاده کنیم.

We play with a weather timeseries dataset recorded at
the Weather Station at the Max Planck Institute for Biogeochemistry in Jena, Germany.
Olaf Kolle, www.bgc-jena.mpg.de/wetter

در این مجموعه داده، ۱۴ کمیت مختلف (مانند دمای هوا، فشار جوی، رطوبت، جهت باد و ...)
هر ۱۰ دقیقه یک بار طی چند سال ثبت شده است.
داده‌های اصلی از سال ۲۰۰۳ هستند، اما مثال ما به داده‌های ۲۰۰۹ تا ۲۰۱۶ محدود می‌شود.

مدل مورد نظر ما، داده‌های مربوط به گذشته‌ی اخیر (چند روز گذشته) را به‌عنوان ورودی دریافت می‌کند
و دمای هوا را در ۲۴ ساعت آینده پیش‌بینی می‌کند.

مسئله‌ی پیش‌بینی دما

دانلود و معاینه‌ی داده‌ها

```
cd ~/Downloads
mkdir jena_climate
cd jena_climate
wget https://s3.amazonaws.com/keras-datasets/jena_climate_2009_2016.csv.zip
unzip jena_climate_2009_2016.csv.zip
```

Inspecting the data of the Jena weather dataset

```
import os

data_dir = '/users/fchollet/Downloads/jena_climate'
fname = os.path.join(data_dir, 'jena_climate_2009_2016.csv')

f = open(fname)
data = f.read()
f.close()

lines = data.split('\n')
header = lines[0].split(',')
lines = lines[1:]

print(header)
print(len(lines))
```

این کد ۴۲۰۵۵۱ خط داده را به‌عنوان خروجی ارائه می‌دهد (هر خط یک گام زمانی است).

```
["Date Time",
 "p (mbar)",
 "T (degC)",
 "Tpot (K)",
 "Tdew (degC)",
 "rh (%)",
 "VPmax (mbar)",
 "VPact (mbar)",
 "VPdef (mbar)",
 "sh (g/kg)",
 "H2OC (mmol/mol)",
 "rho (g/m**3)",
 "wv (m/s)",
 "max. wv (m/s)",
 "wd (deg)"]
```

سرآیند / header

مسئله‌ی پیش‌بینی دما

تبدیل خطوط داده‌ای به یک آرایه‌ی NumPy

حال، تمام ۴۲۰۵۵۱ خط داده‌ای را به یک آرایه‌ی NumPy تبدیل می‌کنیم.

Parsing the data

```
import numpy as np

float_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(',')[1:]]
    float_data[i, :] = values
```

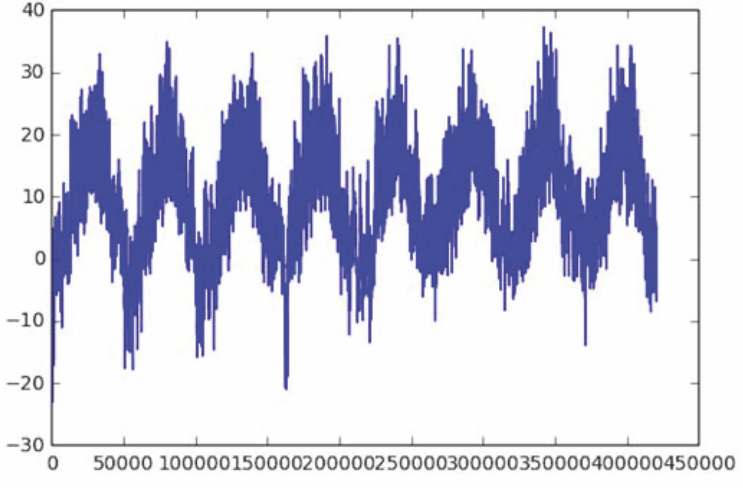
مسئله‌ی پیش‌بینی دما

ترسیم سری زمانی دما

نمودار دما (برحسب درجه‌ی سلیسیوس) در طی زمان را رسم می‌کنیم.

Plotting the temperature timeseries

```
from matplotlib import pyplot as plt

temp = float_data[:, 1]  <1> temperature (in degrees Celsius)
plt.plot(range(len(temp)), temp)
```



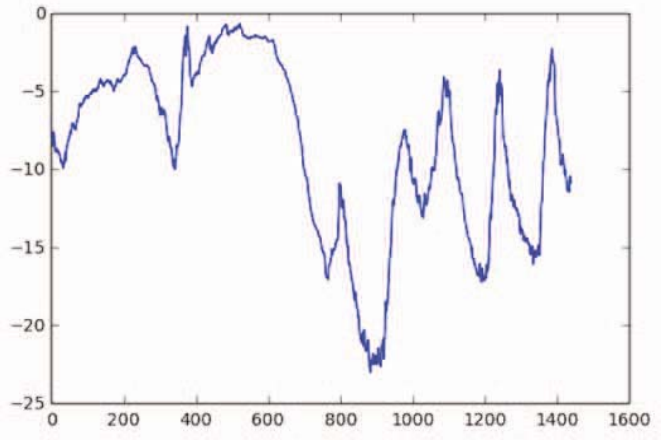
مسئله‌ی پیش‌بینی دما

ترسیم سری زمانی دما برای ۱۰ روز

نمودار دما (برحسب درجه‌ی سلیسیوس) در طی زمان را برای ۱۰ روز رسم می‌کنیم.
چون داده‌ها هر ۱۰ دقیقه یک‌بار ثبت می‌شوند، برای هر روز ۱۴۴ نقطه‌ی داده‌ای به‌دست می‌آید.

Plotting the first 10 days of the temperature timeseries

```
plt.plot(range(1440), temp[:1440])
```



در این نمودار، تناوب روزانه به‌خصوص برای ۴ روز آخر مشهود است. این ۱۰ روز باید از روزهای سرد زمستانی باشد.
پیش‌بینی دمای میانگین ماه بعد براساس داده‌های چند ماه گذشته ساده است (به‌دلیل قابل اطمینان بودن داده‌های دوره‌های سالانه).
اما، به‌نظر می‌رسد داده‌ها دما در مقیاس چند روز، بسیار بی‌قاعده‌تر باشد ...

مسئله‌ی پیش‌بینی دما

آماده‌سازی داده‌ها

PREPARING THE DATA

فرمول‌بندی دقیق این مسئله:

با توجه به داده‌هایی که زمان آنها مربوط به گام‌های زمانی `lookback` است (هر گام زمانی = ۱۰ دقیقه)،
آیا می‌توانیم دما را در گام‌های زمانی `delay` پیش‌بینی کنیم؟

پارامترها

- `lookback = 720`: مشاهدات مربوط به ۵ روز قبل هستند.
- `steps = 6`: مشاهدات در یک نقطه به‌ازای هر ساعت نمونه‌گیری می‌شوند.
- `delay = 144`: تارگت‌ها در ۲۴ ساعت آینده خواهند بود.

برای شروع کار:

۱) داده‌ها را **پیش‌پردازش** می‌کنیم تا به فرمت قابل پذیرش برای یک شبکه‌ی عصبی تبدیل شوند.

داده‌ها عددی هستند، پس برداری‌سازی لازم نیست.

اما هر سری زمانی در داده‌ها مقیاس متفاوتی دارد (مثلاً بازه‌ی مقادیر دما و بازه‌ی مقادیر فشار):

هر سری زمانی را به‌صورت مستقل نرمال‌سازی می‌کنیم تا همگی مقادیر کوچکی در یک مقیاس مشابه پیدا کنند.

۲) یک **مولد** پایتون می‌نویسیم که دسته‌های داده را تولید کند و تحویل بدهد.

این مولد آرایه‌ی کنونی داده‌های ممیزشناور را دریافت می‌کند و دسته‌هایی از داده‌ها مربوط به گذشته‌ی اخیر را به‌همراه دمای تارگت در آینده برمی‌گرداند. چون نمونه‌های موجود در این مجموعه داده شدیداً تکراری است (اشتراک نمونه‌ی N و نمونه‌ی $N+1$ در بخش زیادی از سری زمانی خود)، تخصیص مستقیم هر نمونه سبب هدر رفت می‌شود.

در عوض، در حین اجرا نمونه‌ها را با استفاده از داده‌های اصلی تولید می‌کنیم.

مسئله‌ی پیش‌بینی دما

نرمال‌سازی داده‌ها

نرمال‌سازی: میانگین هر سری زمانی را از آن کم می‌کنیم و بر انحراف استاندارد تقسیم می‌کنیم
۲۰۰,۰۰۰ سری زمانی اول را به‌عنوان داده‌های آموزشی استفاده می‌کنیم.

Normalizing the data

```
mean = float_data[:200000].mean(axis=0)
float_data -= mean
std = float_data[:200000].std(axis=0)
float_data /= std
```

مسئله‌ی پیش‌بینی دما

مولد داده‌ها

مولد داده‌ها، خروجی خود را به‌صورت `(samples, targets)` تحویل می‌دهد:
`samples`: یک دسته از داده‌های ورودی، `targets`: آرایه‌ی متناظر دماهای تارگت

آرگومان‌های مولد عبارتند از:

- `data`—The original array of floating-point data, which you normalized.
- `lookback`—How many timesteps back the input data should go.
- `delay`—How many timesteps in the future the target should be.
- `min_index` and `max_index`—Indices in the data array that delimit which timesteps to draw from. This is useful for keeping a segment of the data for validation and another for testing.
- `shuffle`—Whether to shuffle the samples or draw them in chronological order.
- `batch_size`—The number of samples per batch.
- `step`—The period, in timesteps, at which you sample data. You'll set it to 6 in order to draw one data point every hour.

مسئله‌ی پیش‌بینی دما

مولد تحویل‌دهنده‌ی نمونه‌های سری‌های زمانی و تارگت‌های آنها

Generator yielding timeseries samples and their targets

```
def generator(data, lookback, delay, min_index, max_index,
              shuffle=False, batch_size=128, step=6):
    if max_index is None:
        max_index = len(data) - delay - 1
    i = min_index + lookback
    while 1:
        if shuffle:
            rows = np.random.randint(
                min_index + lookback, max_index, size=batch_size)
        else:
            if i + batch_size >= max_index:
                i = min_index + lookback
            rows = np.arange(i, min(i + batch_size, max_index))
            i += len(rows)

        samples = np.zeros((len(rows),
                            lookback // step,
                            data.shape[-1]))
        targets = np.zeros((len(rows),))
        for j, row in enumerate(rows):
            indices = range(rows[j] - lookback, rows[j], step)
            samples[j] = data[indices]
            targets[j] = data[rows[j] + delay][1]
        yield samples, targets
```

مسئله‌ی پیش‌بینی دما

Preparing the training, validation, and test generators

```
lookback = 1440
step = 6
delay = 144
batch_size = 128
```

```
train_gen = generator(float_data,
                      lookback=lookback,
                      delay=delay,
                      min_index=0,
                      max_index=200000,
                      shuffle=True,
                      step=step,
                      batch_size=batch_size)
val_gen = generator(float_data,
                    lookback=lookback,
                    delay=delay,
                    min_index=200001,
                    max_index=300000,
                    step=step,
                    batch_size=batch_size)
test_gen = generator(float_data,
                     lookback=lookback,
                     delay=delay,
                     min_index=300001,
                     max_index=None,
                     step=step,
                     batch_size=batch_size)

val_steps = (300000 - 200001 - lookback)

test_steps = (len(float_data) - 300001 - lookback)
```

از تابع انتزاعی generator، برای ایجاد سه‌نمونه مولد استفاده می‌کنیم:

- آموزش، ۲۰۰۰۰۰ گام زمانی اول
- اعتبارسنجی، ۱۰۰۰۰۰ گام بعدی
- آزمایش، بقیه‌ی داده‌ها

How many steps to draw from val_gen in order to see the entire validation set

How many steps to draw from test_gen in order to see the entire test set

مسئله‌ی پیش‌بینی دما

یک خط پایه‌ی بدون یادگیری ماشینی مبتنی بر حس مشترک

A COMMON-SENSE, NON-MACHINE-LEARNING BASELINE

ایجاد یک خط پایه برای اثبات کارآیی و سودمندی مدل‌های پیشرفته‌تر یادگیری ماشینی
با فائق آمدن بر این خط مبنا.

مثلاً در مسئله‌ی طبقه‌بندی نامتوازن که برخی کلاس‌ها بسیار متداول‌تر از کلاس‌های دیگر هستند.
اگر مجموعه داده‌ی شما حاوی ۹۰% نمونه از کلاس A و ۱۰% نمونه از کلاس B باشد،
روی‌کرد حس مشترک برای طبقه‌بندی نمونه‌ی جدید انتخاب کلاس A است
که در مجموع ۹۰% درست کار می‌کند!
هر روی‌کرد دیگر یادگیری ماشینی، باید از این ۹۰% عبور کند تا کارآیی آن ثابت شود! و در برخی موارد این بسیار مشکل است !!

در مورد سری‌های زمانی دما،
می‌توان دما را پیوسته فرض کرد (احتمال می‌رود دمای روز بعد به دمای امروز نزدیک باشد)
و می‌توان آن را در یک دوره‌ی روزانه به‌صورت متناوب در نظر گرفت.
⇐ روی‌کرد حس مشترک: پیش‌بینی می‌کنیم که دما در ۲۴ ساعت پس از زمان کنونی برابر با دمای کنونی باشد.

مسئله‌ی پیش‌بینی دما

محاسبه‌ی خطای مطلق میانگین برای خط پایه‌ی حس مشترک

معیار خطای مطلق میانگین (MAE) را برای خط پایه‌ی حس مشترک محاسبه می‌کنیم.

```
np.mean(np.abs(preds - targets))
```

Computing the common-sense baseline MAE

```
def evaluate_naive_method():
    batch_maes = []
    for step in range(val_steps):
        samples, targets = next(val_gen)
        preds = samples[:, -1, 1]
        mae = np.mean(np.abs(preds - targets))
        batch_maes.append(mae)
    print(np.mean(batch_maes))

evaluate_naive_method()
```

مسئله‌ی پیش‌بینی دما

محاسبه‌ی خطای مطلق میانگین برای خط پایه‌ی حس مشترک به درجه‌ی سلیسیوس

معیار خطای مطلق میانگین (MAE) را برای خط پایه‌ی حس مشترک مقدار 0.29 را نتیجه می‌دهد.
چون داده‌های مربوط به دما نرمال‌سازی شده‌اند تا حول نقطه‌ی مرکزی صفر قرار داشته باشند،
و دارای انحراف استاندارد برابر با یک هستند،
پس این عدد بلافاصله قابل تفسیر نیست.
این عدد را به یک خطای مطلق میانگین بر حسب درجه‌ی سلیسیوس تبدیل می‌کنیم.

Converting the MAE back to a Celsius error

```
celsius_mae = 0.29 * std[1]
```

حاصل، 2.57 درجه‌ی سلیسیوس است.
این یک خطای مطلق میانگین نسبتاً بزرگ است.

برای بهتر شدن این خطا از یادگیری عمیق استفاده می‌کنیم.

مسئله‌ی پیش‌بینی دما

یک روی‌کرد پایه‌ی یادگیری ماشینی

A BASIC MACHINE-LEARNING APPROACH

آزمایش مدل‌های ساده و کم‌هزینه برای یادگیری ماشینی (مانند شبکه‌های کوچک متصل متراکم)
پیش از اقدام به بررسی مدل‌های پیچیده و دارای هزینه‌ی محاسباتی بالا.

از این طریق مطمئن می‌شویم که آیا افزودن هرگونه پیچیدگی بیشتر به مسئله مزیتی دارد یا خیر ...

مسئله‌ی پیش‌بینی دما

یک روی‌کرد پایه‌ی یادگیری ماشینی: آموزش و ارزیابی یک مدل متصل متراکم

ابتدا داده‌ها را مسطح‌سازی می‌کنیم و سپس آنها را به دو لایه‌ی متراکم ارسال می‌کنیم.
این یک مسئله‌ی رگرسیون است: تابع فعالیت لایه‌ی آخر: خطی؛ تابع اتلاف: MAE.
(نتایج قابل مقایسه با خط پایه‌ی حس مشترک)

Training and evaluating a densely connected model

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Flatten(input_shape=(lookback // step, float_data.shape[-1])))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1))
model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

مسئله‌ی پیش‌بینی دما

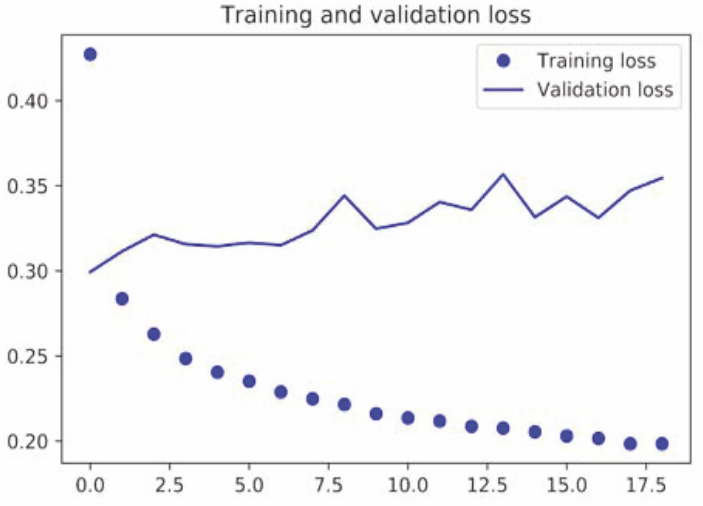
یک روی‌کرد پایه‌ی یادگیری ماشینی: رسم نتایج مدل متصل متراکم

Plotting results

```
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

مسئله‌ی پیش‌بینی دما

یک روی‌کرد پایه‌ی یادگیری ماشینی: رسم نتایج مدل متصل متراکم: نمودار اتلاف آموزش و اعتبارسنجی



برخی از مقادیر اتلاف اعتبارسنجی، نزدیک به خط پایه‌ی بدون یادگیری هستند، اما چندان قابل اطمینان نیستند.
ارزشمند بودن خط مبنا: معلوم می‌شود که ارائه‌ی عملکرد بهتر چندان ساده نیست
«خط پایه‌ی حس مشترک، حاوی اطلاعات ارزشمندی است که یک مدل یادگیری ماشینی به آن دسترسی ندارد.»

چرا مدل در حال آموزش، مدل خط پایه را نمی‌یابد؟
زیرا این راه‌حل ساده در تنظیمات آموزشی (فضای فرضیه ما: همه‌ی شبکه‌های متصل متراکم ...) قرار ندارد
با به‌دلیل پیچیدگی مدل در آن گم می‌شود.

مسئله‌ی پیش‌بینی دما

اولین خط مبنای بازگشتی با استفاده از GRU

A First Recurrent Baseline

روی‌کرد قبلی، سری‌های زمانی را مسطح کرد، یعنی مفهوم زمان را از داده‌های ورودی حذف کرد.
حال با روشی دیگر داده‌ها را با در نظر گرفتن ترتیب و روابط علّی آنها مورد بررسی قرار می‌دهیم:
مدل مناسب برای داده‌های دنباله‌ای: **شبکه‌های بازگشتی**

از لایه‌ی GRU استفاده می‌کنیم.
(مشابه LSTM و با همان اصول کار می‌کند اما تا حدی ساده‌تر است و برای اجرا کم‌هزینه‌تر است؛
اگرچه ممکن است قدرت بازنمایی آن با LSTM یکسان نباشد:
در یادگیری ماشینی، همیشه بین هزینه‌ی محاسباتی و قدرت بازنمایی بده‌بستان وجود دارد.)

Junyoung Chung et al., "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *Conference on Neural Information Processing Systems* (2014), https://arxiv.org/abs/1412.3555.

مسئله‌ی پیش‌بینی دما

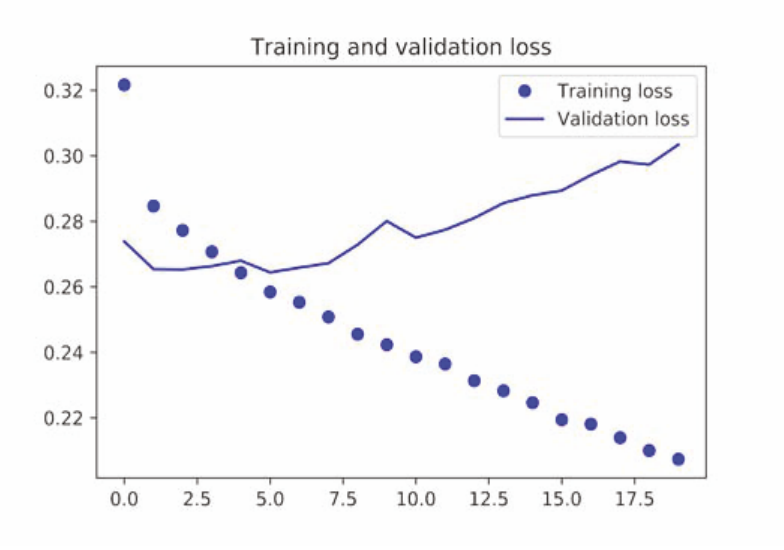
اولین خط مبنای بازگشتی با استفاده از GRU: آموزش و ارزیابی

Training and evaluating a GRU-based model

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.GRU(32, input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

مسئله‌ی پیش‌بینی دما

اولین خط مبنای بازگشتی با استفاده از GRU: نتایج



نتیج بسیار بهتر شده است و می‌توانیم تا حد زیادی مدل پایه‌ی مبتنی بر حس مشترک را شکست بدهیم.

⇐ برای این وظیفه، شبکه‌های بازگشتی در مقایسه با شبکه‌های متراکم با مسطح‌سازی دنباله برتری دارد.

MAE اعتبارسنجی جدید حدوداً 0.265 است.

پس از دنرمال‌سازی به خطای مطلق میانگین 2.35 درجه‌ی سلیسیوس می‌رسیم.

نسبت به خط پایه (2.57) بهتر است اما احتمالاً هنوز جای بهبود وجود دارد.

استفاده از برون‌اندازی بازگشتی برای مقابله با بیش‌برازش

USING RECURRENT DROPOUT TO FIGHT OVERFITTING

برون‌اندازی، واحدهای ورودی یک لایه را به‌صورت تصادفی برابر با صفر قرار می‌دهد
تا همبستگی اتفاقی بین داده‌های آموزشی آن لایه را از بین ببرد.
اما نحوه‌ی صحیح به‌کارگیری برون‌اندازی در شبکه‌های بازگشتی، پرسش مهمی است.

اعمال برون‌اندازی قبل از یک لایه‌ی بازگشتی نه‌تنها به رگولاریزاسیون کمکی نمی‌کند،
بلکه مانع انجام فرآیند یادگیری می‌شود.

روش مناسب برون‌اندازی در شبکه‌های بازگشتی:

به‌جای استفاده از یک ماسک برون‌اندازی که از یک گام زمانی تا گام زمانی دیگر به‌صورت تصادفی تغییر می‌کند،
در هر گام زمانی باید ماسک برون‌اندازی یکسانی (الگوی یکسان برای واحدهای برون‌اندازی شده) به‌کار گرفته شود.
به‌علاوه، برای منظم‌سازی بازنمایی‌های تشکیل‌شده توسط گیت‌ها (در لایه‌هایی چون LSTM و GRU)
باید یک ماسک برون‌اندازی ثابت زمانی به فعالیت‌های بازگشتی داخلی آن لایه اعمال شود
(ماسک برون‌اندازی بازگشتی / recurrent dropout mask).

استفاده از ماسک یکسان برای برون‌اندازی در هر گام زمانی،
به شبکه این امکان را می‌دهد که خطای یادگیری خود را به‌درستی در طول زمان انتشار دهد.
یک ماسک تصادفی زمانی برای برون‌اندازی ممکن است این سیگنال خطا را مختل کند و برای فرآیند یادگیری مضر باشد.

Yarin Gal, "Uncertainty in Deep Learning (PhD Thesis)," October 13, 2016, http://mlg.eng.cam.ac.uk/yarin/blog_2248.html

استفاده از برون‌اندازی بازگشتی برای مقابله با بیش‌برازش

در کراس

Training and evaluating a dropout-regularized GRU-based model

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.GRU(32,
                     dropout=0.2,
                     recurrent_dropout=0.2,
                     input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=40,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

رگولاریزاسیون با برون‌اندازی معمولاً باعث افزایش زمان همگرایی می‌شود. پس، تعداد اپک‌ها برای آموزش را دو برابر می‌کنیم.

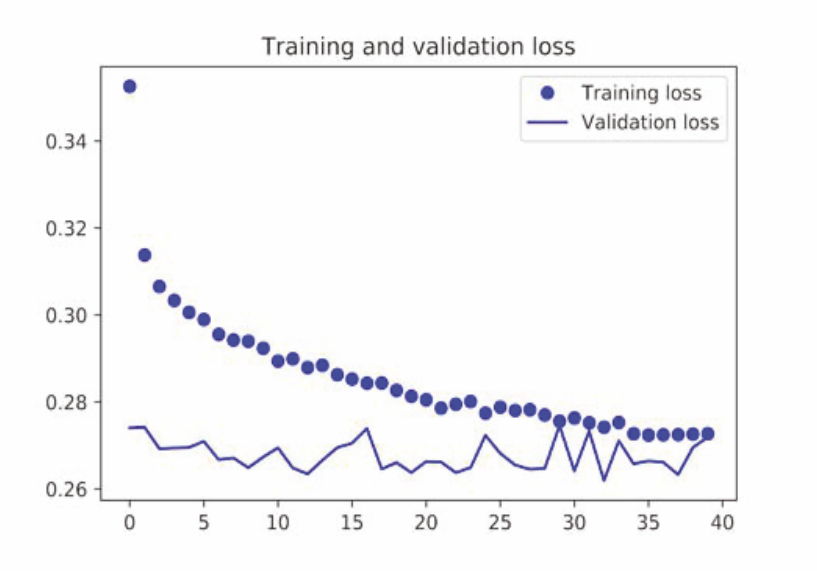
هر لایه‌ی بازگشتی در کراس، دارای دو آرگومان مرتبط به برون‌اندازی است:

`dropout`: یک عدد اعشاری مشخص‌کننده‌ی نرخ برون‌اندازی برای واحدهای ورودی آن لایه

`recurrent_dropout`: مشخص‌کننده‌ی نرخ برون‌اندازی برای واحدهای بازگشتی

مسئله‌ی پیش‌بینی دما

استفاده از برون‌اندازی بازگشتی برای مقابله با بیش‌برازش: نتایج



موفقیت! طی ۳۰ اپک اول بیش‌برازش اتفاق نمی‌افتد.
البته، با وجود اینکه امتیازهای ارزیابی پایدارتری داریم،
بهترین امتیازهای اینجا خیلی بهتر از امتیازهای قبلی نیست.

پشته‌سازی از لایه‌های بازگشتی

STACKING RECURRENT LAYERS

زمانی که بیش‌برازش دیده نمی‌شود و به‌نظر می‌رسد که دارای گلوگاه کارآیی هستیم،
باید ظرفیت شبکه را افزایش دهیم.

(افزایش ظرفیت شبکه تا زمانی ایده‌ی خوبی است که بیش‌برازش به مانع اصلی تبدیل شود؛
با فرض اینکه از قبل گام‌های ابتدایی برای کاهش اثر بیش‌برازش انجام شده باشد: مانند برون‌اندازی.
تا زمانی که با بیش‌برازش بسیار بدی روبرو نشده باشیم، احتمالاً ظرفیت شبکه هنوز تکمیل نشده است.)

روش‌های افزایش ظرفیت شبکه:
افزایش تعداد واحدهای موجود در لایه‌ها یا افزودن لایه‌های بیشتر

پشته‌سازی از لایه‌های بازگشتی، روشی کلاسیک برای ایجاد یک شبکه‌ی بازگشتی قدرت‌مندتر است.
(مثال: الگوریتم ترجمه‌ی گوگل حاوی پشته‌ای از هفت لایه‌ی بزرگ LSTM است.)

پشته‌سازی از لایه‌های بازگشتی

در کراس

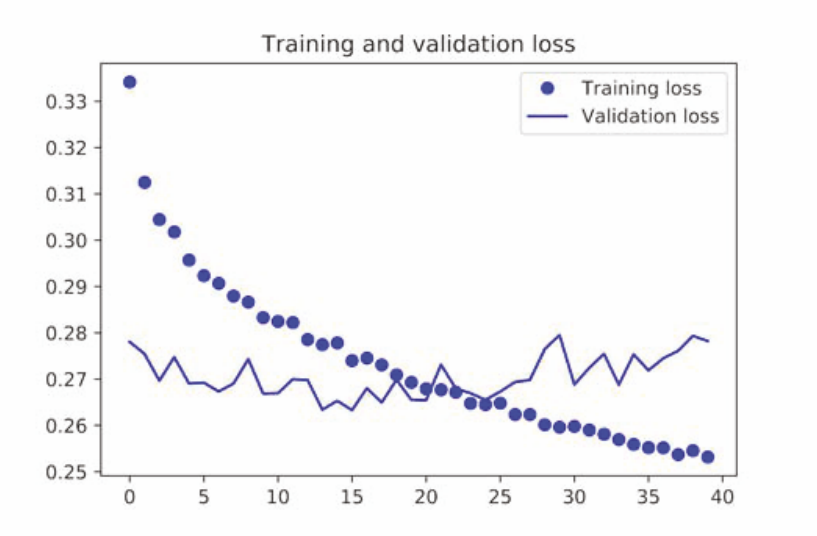
Training and evaluating a dropout-regularized, stacked GRU model

برای پشته‌سازی لایه‌های بازگشتی در کراس، باید لایه‌های میانی، دنباله‌ی کامل خروجی‌های خود را برگردانند (به‌جای خروجی آخرین گام).

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.GRU(32,
                     dropout=0.1,
                     recurrent_dropout=0.5,
                     return_sequences=True,
                     input_shape=(None, float_data.shape[-1])))
model.add(layers.GRU(64, activation='relu',
                     dropout=0.1,
                     recurrent_dropout=0.5))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=40,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

مسئله‌ی پیش‌بینی دما

پشته‌سازی از لایه‌های بازگشتی: نتایج



لایه‌های اضافه شده، نتایج را اندکی بهبود داده‌اند، اگرچه قابل توجه نیست.

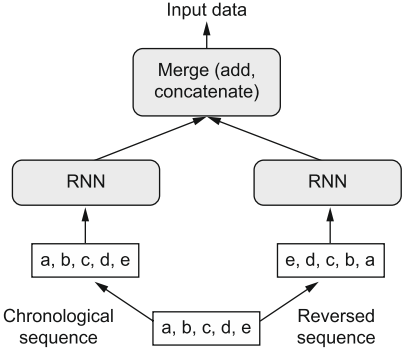
نتایج:

- از آنجا که با بیش‌برازش بسیار بدی روبرو نشده‌ایم، برای بهبود اتلاف اعتبارسنجی می‌توانیم اندازه‌ی لایه‌ها را بزرگ کنیم (با افزایش هزینه‌ی محاسباتی).
- اضافه کردن یک لایه، کمک چندانی نکرد. ممکن است شاهد کاهش بازدهی ناشی از افزایش ظرفیت شبکه باشیم.

استفاده از شبکه‌های عصبی بازگشتی دوطرفه

USING BIDIRECTIONAL RNNS

RNNهای دوطرفه: یک شکل تغییریافته‌ی متداول از RNN
که در برخی کاربردها کارآیی بهتری نسبت به RNN معمولی از خود نشان می‌دهد
(بسیار مناسب برای پردازش زبان طبیعی)

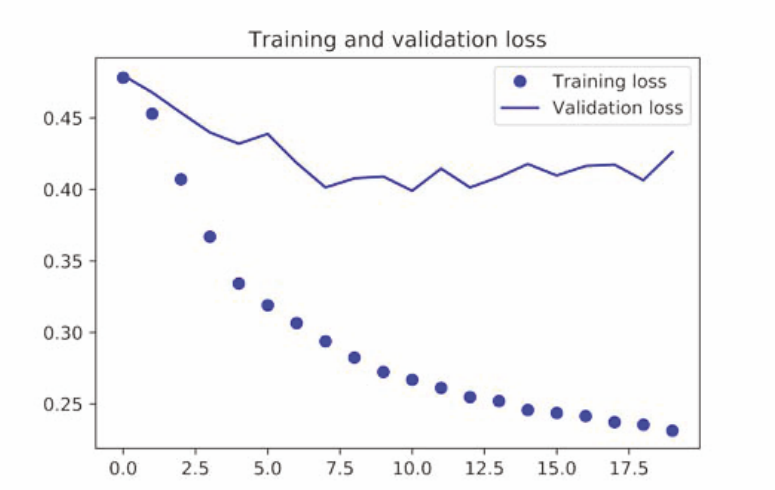


RNN دوطرفه از حساسیت RNNها نسبت به ترتیب بهره می‌برد
و شامل به‌کارگیری دو RNN معمولی (مانند لایه‌های LSTM یا GRU) است
که هر یک از آنها دنباله‌ی ورودی را در یک جهت پردازش می‌کند
(یکی به‌ترتیب زمانی / chronologically
و دیگری به‌ترتیب وارون زمانی / anti-chronologically)؛
سپس بازنمایی‌های حاصل از آنها با هم ادغام می‌شود.

با پردازش یک دنباله از دو طرف،
یک RNN دوطرفه می‌تواند الگوهایی را پیدا کند که ممکن است از دید یک RNN یک‌طرفه دور مانده باشد.

مسئله‌ی پیش‌بینی دما

آموزش و اعتبارسنجی با یک GRU آموزش‌یافته روی دنباله‌های وارون



در مولد داده‌ها، ترتیب زمانی داده‌ها را وارون می‌کنیم (خط آخر: `yield samples[:, ::-1, :], targets`).
و همان شبکه‌ی یک‌لایه‌ای *GRU* (آزمایش اول) را آموزش می‌دهیم.
نتایج فوق، عملکرد شدیداً ضعیفی را نشان می‌دهد حتی نسبت به خط پایه‌ی حس مشترک.

این بدان معناست که پردازش به ترتیب زمانی برای این مجموعه داده، ضروری است.
این مسئله برای بسیاری از مسائل دیگر از جمله پردازش زبان طبیعی برقرار نیست.

مسئله‌ی طبقه‌بندی نقد فیلم

آموزش و ارزیابی یک LSTM با استفاده از دنباله‌های وارون

Training and evaluating an LSTM using reversed sequences

```
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras import layers
from keras.models import Sequential

max_features = 10000
maxlen = 500

(x_train, y_train), (x_test, y_test) = imdb.load_data(
    num_words=max_features)

x_train = [x[::-1] for x in x_train]
x_test = [x[::-1] for x in x_test]

x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

model = Sequential()
model.add(layers.Embedding(max_features, 128))
model.add(layers.LSTM(32))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

Number of words to consider as features

Cuts off texts after this number of words (among the max_features most common words)

Loads data

Reverses sequences

Pads sequences

مسئله‌ی طبقه‌بندی نقد فیلم

آموزش و ارزیابی یک LSTM با استفاده از دنباله‌های وارون

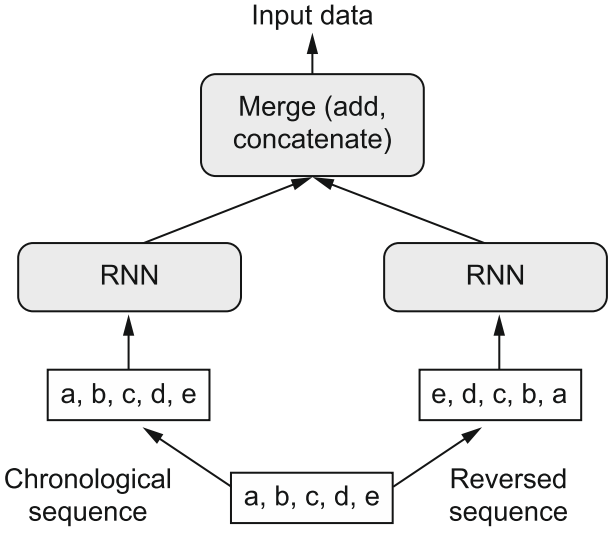
با آموزش و ارزیابی یک LSTM با استفاده از دنباله‌های وارون
به عملکردی تقریباً مشابه با عملکرد LSTM با ترتیب زمانی دست پیدا می‌کنیم.

در چنین مجموعه داده‌ی متنی، پردازش با ترتیب وارون، به‌خوبی پردازش با ترتیب زمانی عمل می‌کند،
و این فرضیه را تایید می‌کند که:
«اگرچه ترتیب کلمه در درک زبان مهم است،
اما اینکه از چه ترتیبی (مستقیم/وارون) استفاده شود، چندان مهم نیست»

مهم‌تر اینکه: یک RNN آموزش‌یافته با دنباله‌های وارون بازنمایی‌های متفاوتی نسبت به بازنمایی‌های یادگیری شده توسط RNN با دنباله‌های اصلی یاد می‌گیرد.
(مثل اینکه وقایع زندگی کسی را از روز آخر به روز اول پخش کنند که باید ایجاد مدل ذهنی متفاوتی می‌شود.)

استفاده از شبکه‌های عصبی بازگشتی دوطرفه

چگونگی عملکرد یک شبکه‌ی بازگشتی دوطرفه



یک RNN دوطرفه، دنباله‌ی ورودی خود را به دو روش مستقیم و وارون بررسی می‌کند؛
و به‌طور بالقوه بازنمایی‌های قوی‌تری به‌دست می‌آورد و الگوهایی را پیدا می‌کند که ممکن است در نسخه‌ی مستقیم از نظر دور مانده باشد.

استفاده از شبکه‌های عصبی بازگشتی دوطرفه

در کراس

برای ایجاد یک نمونه از RNN دوطرفه در کراس، از لایه‌ی Bidirectional استفاده می‌کنیم
که یک نمونه از لایه‌ی بازگشتی را به‌عنوان آرگومان اول خود دریافت می‌کند.
نمونه‌ی دوم و مجزایی برای پردازش دنباله‌ی معکوس از روی این آرگومان ساخته می‌شود.

Training and evaluating a bidirectional LSTM

```
model = Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.Bidirectional(layers.LSTM(32)))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

برای مسئله‌ی طبقه‌بندی نقد فیلم،
LSTM دوطرفه اندکی بهتر از LSTM معمولی عمل می‌کند و به‌دقت ۸۹% می‌رسد.
همچنین با سرعت بیشتری دچار بیش‌برازش می‌شود (زیرا تعداد پارامترها دوبرابر شده است).
با اندکی رگولاریزاسیون به نتایج بهتری می‌رسیم.

مسئله‌ی پیش‌بینی دما

استفاده از یک GRU دوطرفه

Training a bidirectional GRU

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Bidirectional(
    layers.GRU(32), input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=40,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

برای مسئله‌ی پیش‌بینی دما،
GRU دوطرفه تقریباً به‌خوبی GRU معمولی عمل می‌کند.
درک علت آن ساده است: تمام ظرفیت پیش‌بینی کننده باید از نیمه‌ی دارای ترتیب زمانی باشد،
چرا که نیمه‌ی دارای ترتیب وارون، عملکرد به‌شدت نامناسبی در این وظیفه دارد
(به این دلیل که در این مورد، گذشته‌ی نزدیک اهمیت بسیار بیشتری نسبت به گذشته‌ی دور دارد).

مسئله‌ی پیش‌بینی دما

بهبودهای بیشتر

GOING EVEN FURTHER

موارد دیگری که می‌توان برای بهبود کارآیی در مسئله‌ی پیش‌بینی دما آنها را بررسی کرد:

تنظیم تعداد واحدهای موجود در هر لایه‌ی بازگشتی در وضعیت پشته‌ای.
(انتخاب‌های فعلی تا حد زیادی دلخواه و احتمالاً پایین‌تر از حد بهینه هستند.)

تنظیم نرخ یادگیری به‌کار رفته در بهینه‌ساز *RMSprop*

استفاده از لایه‌های *LSTM* به‌جای *GRU*.

استفاده از یک رگرسور متصل متراکم بزرگ‌تر در بالای لایه‌های بازگشتی.
(یعنی یک لایه‌ی *Dense* بزرگ‌تر یا حتی پشته‌ای از این لایه‌ها)

در انتها باید مدل‌های دارای بهترین عملکرد (از نظر *MAE* اعتبارسنجی) را در مجموعه‌ی آزمایشی اجرا کنیم. وگرنه، معماری‌هایی توسعه می‌یابند که نسبت به مجموعه‌ی اعتبارسنجی دچار بیش‌برازش هستند.

جمع‌بندی

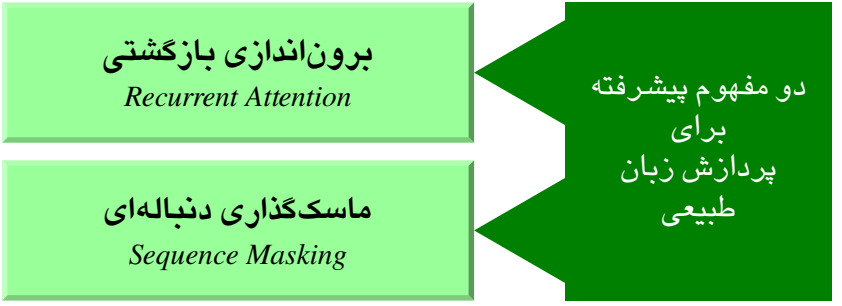
WRAPPING UP

Here's what you should take away from this section:

- As you first learned in chapter 4, when approaching a new problem, it's good to first establish common-sense baselines for your metric of choice.
  If you don't have a baseline to beat, you can't tell whether you're making real progress.
- Try simple models before expensive ones, to justify the additional expense. Sometimes a simple model will turn out to be your best option.
- When you have data where temporal ordering matters, recurrent networks are a great fit and easily outperform models that first flatten the temporal data.
- To use dropout with recurrent networks, you should use a time-constant dropout mask and recurrent dropout mask. These are built into Keras recurrent layers, so all you have to do is use the dropout and recurrent_dropout arguments of recurrent layers.
- Stacked RNNs provide more representational power than a single RNN layer. They're also much more expensive and thus not always worth it. Although they offer clear gains on complex problems (such as machine translation), they may not always be relevant to smaller, simpler problems.
- Bidirectional RNNs, which look at a sequence both ways, are useful on natural language processing problems. But they aren't strong performers on sequence data where the recent past is much more informative than the beginning of the sequence.

استفاده‌ی پیشرفته از شبکه‌های عصبی بازگشتی

مفاهیم پیشرفته برای پردازش زبان طبیعی



Markets and machine learning

Some readers are bound to want to take the techniques we've introduced here and try them on the problem of forecasting the future price of securities on the stock market (or currency exchange rates, and so on). Markets have *very different statistical characteristics* than natural phenomena such as weather patterns. Trying to use machine learning to beat markets, when you only have access to publicly available data, is a difficult endeavor, and you're likely to waste your time and resources with nothing to show for it.

Always remember that when it comes to markets, past performance is *not* a good predictor of future returns—looking in the rear-view mirror is a bad way to drive. Machine learning, on the other hand, is applicable to datasets where the past *is* a good predictor of the future.

یادگیری عمیق

یادگیری عمیق برای متن و دنباله‌ها

۴

پردازش دنباله‌ها با شبکه‌های کانوولوشنال

پردازش دنباله‌ها با شبکه‌های کانوولوشنال

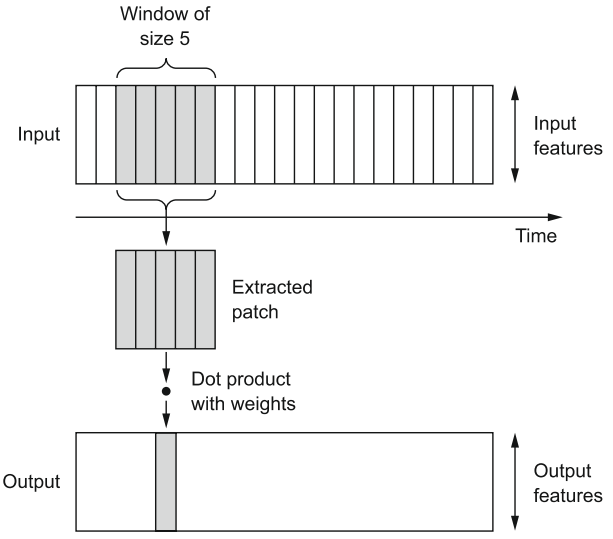
SEQUENCE PROCESSING WITH CONVNETS

همان دلایلی که سبب می‌شد شبکه‌های کانوولوشنال برای بینایی کامپیوتری عملکرد خوبی داشته باشند
[توانایی آنها در انجام کانوولوشن و استخراج ویژگی از قطعات محلی ورودی و امکان ماژولار کردن بازنمایی‌ها]،
باعث می‌شوند در پردازش دنباله‌ها نیز بسیار مناسب باشند.
زمان را می‌توان یک بعد فضایی در نظر گرفت (مثل عرض یا ارتفاع یک تصویر دوبعدی).

شبکه‌های کانوولوشنال یک–بعدی در برخی مسائل پردازش دنباله می‌توانند با RNNها رقابت کنند.
کاربردهای اخیر: تولید صدا و ترجمه‌ی ماشینی + طبقه‌بندی متون، پیش‌بینی سری زمانی، ...

پردازش دنباله‌ها با شبکه‌های کانوولوشنال

شیوه‌ی عملکرد



با استفاده از کانوولوشن‌های یک–بعدی می‌توانیم قطعات یک–بعدی محلی (زیردنباله‌ها) را از دنباله‌ها جداسازی کنیم.
لایه‌های کانوولوشنی می‌توانند الگوهای محلی را در یک دنباله آشکارسازی کنند.
⇐ الگویی که در یک موقعیت خاص از یک دنباله یادگیری شده است را می‌توان بعداً در یک مکان دیگر بازشناسی کرد.

مثال: یک شبکه‌ی کانوولوشنال یک–بعدی که دنباله‌ای از نویسه‌ها را با استفاده از پنجره‌های کانوولوشنی با سایز ۵ پردازش می‌کند، باید بتواند کلمات یا زیرکلمات به طول ۵ یا کمتر را یاد بگیرد و باید بتواند این کلمات را در هر متنی در یک دنباله‌ی ورودی بازشناسی کند.
⇐ توانایی یادگیری موروفولوژی کلمات

پردازش دنباله‌ها با شبکه‌های کانوولوشنال

تلفیق یک-بعدی برای داده‌های دنباله‌ای

1D POOLING FOR SEQUENCE DATA

تلفیق یک-بعدی:

استخراج قطعات یک-بعدی (زیردنباله‌ها) و

ارسال مقدار ماکزیمم (برای max-pooling) یا مقدار میانگین (برای average-pooling) به‌عنوان خروجی.

هدف از عمل تلفیق، کاهش طول ورودی‌های یک-بعدی (زیر نمونه‌برداری/subsampling) است.

پیاده‌سازی یک شبکه‌ی عصبی کانوولوشنال یک-بعدی

در کراس

IMPLEMENTING A 1D CONVNET

در کراس لایه‌ی Conv1D را داریم که واسط آن مشابه Conv2D است.
این لایه یک تانسور سه‌بعدی به شکل
(samples, time, features)
را به‌عنوان ورودی دریافت می‌کند
و تانسورهایی سه‌بعدی به شکل مشابهی را برمی‌گرداند.

پنجره‌ی کانوولوشن، یک کادر یک-بعدی روی محور زمانی (اولین محور تانسور ورودی) است.

مسئله‌ی طبقه‌بندی نقد فیلم‌ها

با استفاده از شبکه‌های کانوولوشنال یک-بعدی: آماده‌سازی داده‌ها

یک شبکه‌های کانوولوشنال یک-بعدی با دو لایه ایجاد می‌کنیم
و آن را برای وظیفه‌ی طبقه‌بندی نقد فیلم‌ها اعمال می‌کنیم.

Preparing the IMDB data

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

max_features = 10000
max_len = 500

print('Loading data...')
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')

print('Pad sequences (samples x time)')
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
```

مسئله‌ی طبقه‌بندی نقد فیلم‌ها

با استفاده از شبکه‌های کانوولوشنال یک-بعدی: آموزش و ارزیابی روی داده‌های IMDB

Training and evaluating a simple 1D convnet on the IMDB data

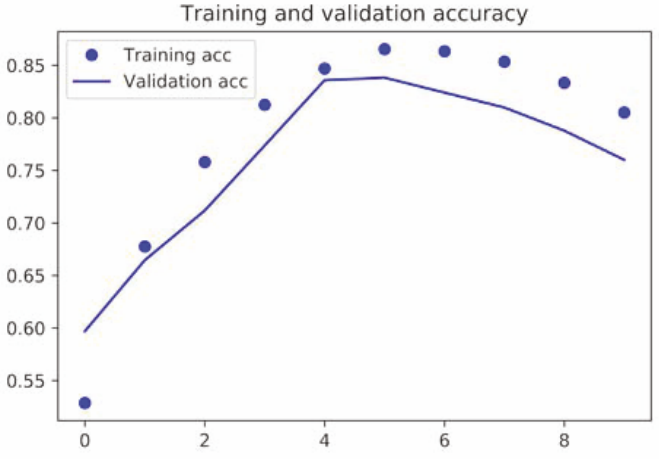
```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Embedding(max_features, 128, input_length=max_len))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))

model.summary()

model.compile(optimizer=RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

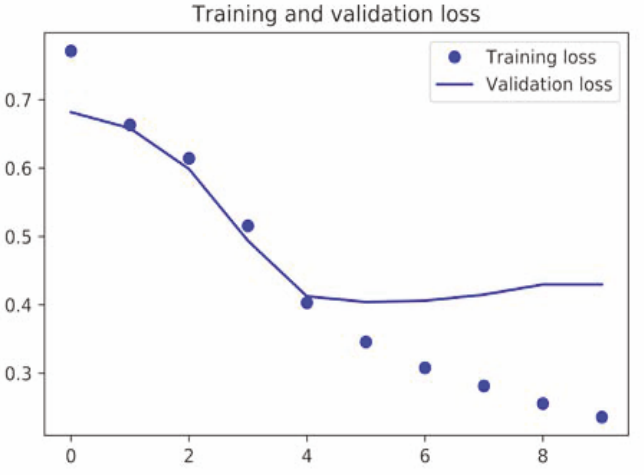
ساختار کای، تناوب لایه‌های کانوولوشنی و تلفیق لایه‌های است.

می‌توانیم از پنجره‌های کانوولوشن با اندازه‌های بزرگ‌تر هم استفاده کنیم (۷ یا ۹ یا ...).

مسئله‌ی طبقه‌بندی نقد فیلم‌ها

با استفاده از شبکه‌های کانوولوشنال یک-بعدی: نتایج





دقت اعتبارسنجی کمی کمتر از دقت اعتبارسنجی LSTM است،
اما زمان اجرا هم در CPU و هم در GPU کمتر است.

⇐ یک شبکه‌ی کانوولوشنال یک-بعدی می‌تواند جایگزینی سریع و کم‌هزینه برای یک شبکه‌ی بازگشتی برای وظیفه‌ی طبقه‌بندی احساسات در سطح کلمه باشد.

ترکیب CNNها و RNNها برای پردازش دنباله‌های طولانی

COMBINING CNNS AND RNNS TO PROCESS LONG SEQUENCES

شبکه‌های کانوولوشنال یک–بعدی قطعات ورودی را به‌صورت مستقل پردازش می‌کنند
و برخلاف *RNN*ها به ترتیب گام‌های زمانی (فراتر از یک مقیاس محلی: اندازه‌ی پنجره‌ها) حساس نیستند.

برای بازشناسی الگوهایی طولانی–مدت، می‌توان
تعداد زیادی لایه‌ی کانوولوشنی و لایه‌ی تلفیق را به‌صورت پشته روی هم قرار داد
که در نتیجه،
لایه‌های بالایی، قطعات طولانی‌تری از ورودی‌های اصلی را خواهند دید.
اما این روش برای ایجاد حساسیت روی ترتیب نسبتاً ضعیف عمل می‌کند.

برای اثبات این ضعف، از شبکه‌های کانوولوشنال یک–بعدی در مسئله‌ی پیش‌بینی دما استفاده می‌کنیم.
(در این مسئله حساسیت به ترتیب برای تولید پیش‌بینی‌های خوب یک عامل کلیدی است.)

مسئله‌ی پیش‌بینی دما

با استفاده از شبکه‌های کانوولوشنال یک-بعدی: آموزش و ارزیابی

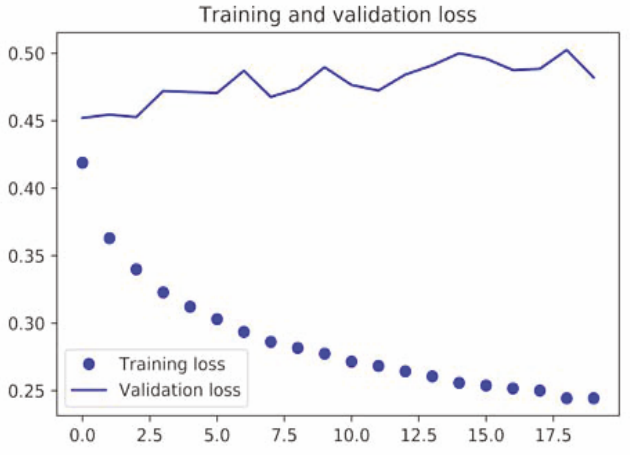
Training and evaluating a simple 1D convnet on the Jena data

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Conv1D(32, 5, activation='relu',
                        input_shape=(None, float_data.shape[-1])))
model.add(layers.MaxPooling1D(3))
model.add(layers.Conv1D(32, 5, activation='relu'))
model.add(layers.MaxPooling1D(3))
model.add(layers.Conv1D(32, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

در این کد از متغیرهای از پیش تعریف شده استفاده شده است.

مسئله‌ی پیش‌بینی دما

با استفاده از شبکه‌های کانوولوشنال یک-بعدی: نتایج



اعتبارسنجی در حد $0.40s$ باقی می‌ماند: پایین‌تر از خط پایه‌ی حس مشترک. MAE

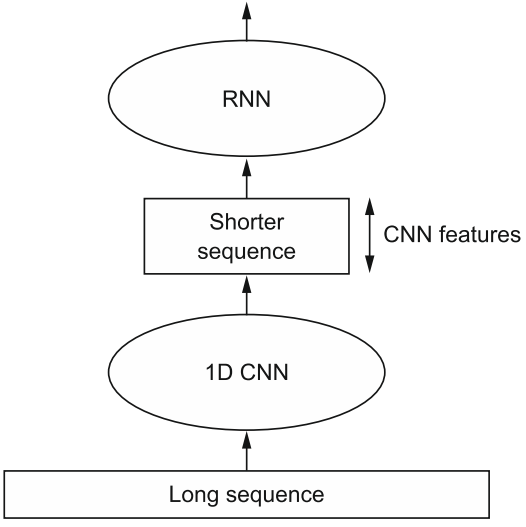
دلیل: شبکه‌ی کانوولوشنال به‌دنبال الگوهایی در هرکجای سری زمانی ورودی می‌پردازد و هیچ دانشی از موقعیت زمانی الگویی که دیده است (ابتدا، انتها، وسط، ...)، ندارد.

چون در این مسئله‌ی خاص پیش‌بینی، تعداد بیشتری از نقاط داده‌ای اخیر باید به‌شیوه‌ای متفاوت نسبت به نقاط داده‌ای قدیمی‌تر تفسیر شوند، از شبکه موفق به تولید نتایج معنادار نمی‌شود.

این محدودیت شبکه‌های کانوولوشنال برای طبقه‌بندی نقد فیلم مشکلی محسوب نمی‌شود، زیرا الگوهای کلمات کلیدی متناظر با نقد مثبت/منفی جدا از مکان وقوع در جمله، حاوی اطلاعات سودمند هستند.

ترکیب CNNها و RNNها برای پردازش دنباله‌های طولانی

COMBINING CNNS AND RNNS TO PROCESS LONG SEQUENCES



یک استراتژی برای ترکیب سرعت و سبکی شبکه‌های کانوولوشنال با حساسیت *RNN*ها به ترتیب،
این است که از یک *CNN* یک-بعدی به‌عنوان یک گام پیش‌پردازش قبل از یک *RNN* استفاده کنیم.

این روش خصوصاً زمانی سودمند است که با دنباله‌هایی سروکار داریم که بسیار طولانی هستند و با نگاه واقع‌گرایانه نمی‌توان آنها را با *RNN*ها پردازش کرد.
CNN دنباله‌های طولانی ورودی را به دنباله‌های بسیار کوچک‌تر از ویژگی‌های سطح بالاتر تبدیل می‌کند (نمونه‌برداری به سمت پایین)؛ سپس این دنباله از ویژگی‌های استخراج شده به ورودی بخش *RNN* تبدیل می‌شود.

این تکنیک در غالب مقالات پژوهشی و کاربردهای عملی دیده نمی‌شود و شاید دلیل آن ناشناخته بودن آن باشد.
این تکنیک مؤثر است و باید رایج شود!

مسئله‌ی پیش‌بینی دما

ترکیب CNNها و RNNها برای پردازش دنباله‌های طولانی: کار با داده‌های دارای رزولوشن بالاتر

Preparing higher-resolution data generators for the Jena dataset

با این تکنیک می‌توانیم دنباله‌های بسیار طولانی‌تری را دستکاری کنیم

Unchanged

```
step = 3
lookback = 720
delay = 144
```

Previously set to 6 (1 point per hour); now 3 (1 point per 30 min)

```
train_gen = generator(float_data,
                      lookback=lookback,
                      delay=delay,
                      min_index=0,
                      max_index=200000,
                      shuffle=True,
                      step=step)
val_gen = generator(float_data,
                    lookback=lookback,
                    delay=delay,
                    min_index=200001,
                    max_index=300000,
                    step=step)
test_gen = generator(float_data,
                     lookback=lookback,
                     delay=delay,
                     min_index=300001,
                     max_index=None,
                     step=step)
val_steps = (300000 - 200001 - lookback) // 128
test_steps = (len(float_data) - 300001 - lookback) // 128
```

مسئله‌ی پیش‌بینی دما

ترکیب CNNها و RNNها برای پردازش دنباله‌های طولانی: کار با داده‌های دارای رزولوشن بالاتر: تعریف مدل

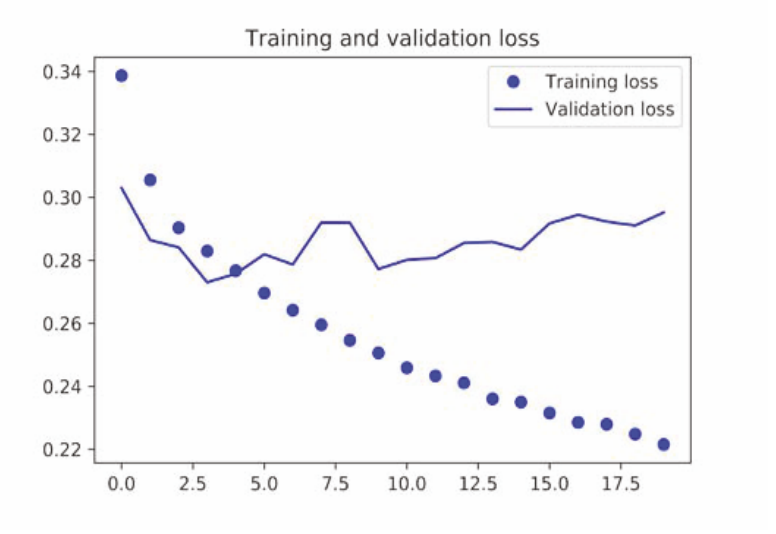
Model combining a 1D convolutional base and a GRU layer

این مدل با دو لایه‌ی کانوولوشنال یک–بعدی شروع می‌شود و با یک لایه‌ی GRU ادامه می‌یابد.

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Conv1D(32, 5, activation='relu',
                        input_shape=(None, float_data.shape[-1])))
model.add(layers.MaxPooling1D(3))
model.add(layers.Conv1D(32, 5, activation='relu'))
model.add(layers.GRU(32, dropout=0.1, recurrent_dropout=0.5))
model.add(layers.Dense(1))

model.summary()

model.compile(optimizer=RMSprop(), loss='mae')

history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

مسئله‌ی پیش‌بینی دما

ترکیب CNNها و RNNها برای پردازش دنباله‌های طولانی: کار با داده‌های دارای رزولوشن بالاتر: تعریف مدل



با قضاوت بر اساس اتلاف اعتبارسنجی مشخص می‌شود که این پیکربندی به‌خوبی GRU رگولاریزه نیست.
اما بسیار سریع‌تر است.
این مدل دو برابر بیشتر داده بررسی می‌کند که به‌نظر می‌رسد در این مورد چندان سودمند واقع نشده است.
اما ممکن است برای مجموعه داده‌های دیگر مفید واقع شود.

جمع‌بندی

WRAPPING UP

Here's what you should take away from this section:

- In the same way that 2D convnets perform well for processing visual patterns in 2D space, 1D convnets perform well for processing temporal patterns. They offer a faster alternative to RNNs on some problems, in particular natural language processing tasks.
- Typically, 1D convnets are structured much like their 2D equivalents from the world of computer vision: they consist of stacks of Conv1D layers and Max-Pooling1D layers, ending in a global pooling operation or flattening operation.
- Because RNNs are extremely expensive for processing very long sequences, but 1D convnets are cheap, it can be a good idea to use a 1D convnet as a preprocessing step before an RNN, shortening the sequence and extracting useful representations for the RNN to process.

یادگیری عمیق برای متن و دنباله‌ها

خلاصه

CHAPTER SUMMARY

- In this chapter, you learned the following techniques, which are widely applicable to any dataset of sequence data, from text to timeseries:
  - How to tokenize text
  - What word embeddings are, and how to use them
  - What recurrent networks are, and how to use them
  - How to stack RNN layers and use bidirectional RNNs to build more-powerful sequence-processing models
  - How to use 1D convnets for sequence processing
  - How to combine 1D convnets and RNNs to process long sequences

یادگیری عمیق برای متن و دنباله‌ها

خلاصه (ادامه)

CHAPTER SUMMARY

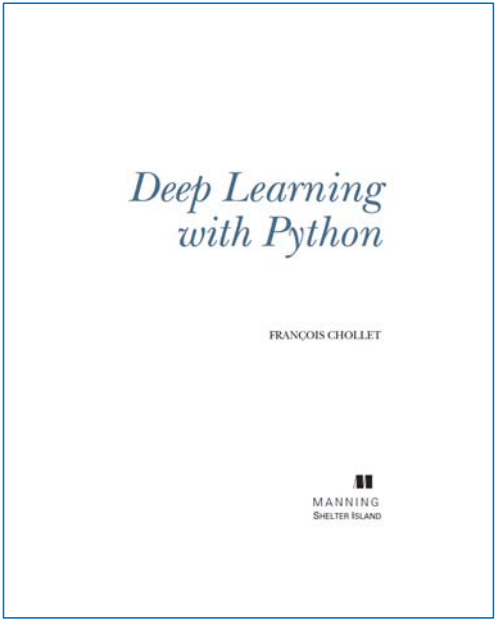
- You can use RNNs for timeseries regression ("predicting the future"), timeseries classification, anomaly detection in timeseries, and sequence labeling (such as identifying names or dates in sentences).
- Similarly, you can use 1D convnets for machine translation (sequence-to-sequence convolutional models, like SliceNeta), document classification, and spelling correction.
- If global order matters in your sequence data, then it's preferable to use a recurrent network to process it. This is typically the case for timeseries, where the recent past is likely to be more informative than the distant past.
- If global ordering isn't fundamentally meaningful, then 1D convnets will turn out to work at least as well and are cheaper. This is often the case for text data, where a keyword found at the beginning of a sentence is just as meaningful as a keyword found at the end.

یادگیری عمیق برای متن و دنباله‌ها

۶

منابع

منبع اصلی



François Chollet,
Deep Learning with Python,
Manning Publications, 2018.

Chapter 6