회귀 신경망과 자연언어

7장

엣지있게 설명한 텐서플로우

최우진

2018/07/10

Wikipedia 말뭉치 준비하기

```python
import bz2
import collections
import os
import re
from lxml import etree

from helpers import download


class Wikipedia:

    TOKEN_REGEX = re.compile(r'[A-Za-z]+|[!?.:,()]')

    def __init__(self, url, cache_dir, vocabulary_size=10000):
        pass
    def __iter__(self):
        """Iterate over pages represented as lists of word indices."""
        pass

    @property
    def vocabulary_size(self):
        pass

    def encode(self, word):
        """Get the vocabulary index of a string word."""
        pass
```

```python
def decode(self, index):
    """Get back the string word from a vocabulary index."""
    pass

def _read_pages(self, url):
    """
    Extract plain words from a Wikipedia dump and store them to the pages
    file. Each page will be a line with words separated by spaces.
    """
    pass

def _build_vocabulary(self, vocabulary_size):
    """
    Count words in the pages file and write a list of the most frequent
    words to the vocabulary file.
    """
    pass

@classmethod
def _tokenize(cls, page):
    pass
```

1. 뭉치 데이터를 다운로드하고 페이지와 단어들을 추출한다.
2. 가장빈번하게 사용되는 단어들의 모음(단어장)을 만들기 위해서 단어 수를 센다.
3. 단어집을 이용해서 추출된 페이지를 인코딩한다.

Wikipedia 말뭉치 준비하기

```
def __init__(self, url, cache_dir, vocabulary_size=10000):
    self._cache_dir = os.path.expanduser(cache_dir)
    self._pages_path = os.path.join(self._cache_dir, 'pages.bz2')
    self._vocabulary_path = os.path.join(self._cache_dir, 'vocabulary.bz2')
    if not os.path.isfile(self._pages_path):
        print('Read pages')
        self._read_pages(url)
    if not os.path.isfile(self._vocabulary_path):
        print('Build vocabulary')
        self._build_vocabulary(vocabulary_size)
    with bz2.open(self._vocabulary_path, 'rt') as vocabulary:
        print('Read vocabulary')
        self._vocabulary = [x.strip() for x in vocabulary]
    self._indices = {x: i for i, x in enumerate(self._vocabulary)}

def __iter__(self):
    """Iterate over pages represented as lists of word indices."""
    with bz2.open(self._pages_path, 'rt') as pages:
        for page in pages:
            words = page.strip().split()
            words = [self.encode(x) for x in words]
            yield words
```

```
@property
def vocabulary_size(self):
    return len(self._vocabulary)

def encode(self, word):
    """Get the vocabulary index of a string word."""
    return self._indices.get(word, 0)

def decode(self, index):
    """Get back the string word from a vocabulary index."""
    return self._vocabulary[index]
```

Wikipedia 말뭉치 준비하기

```
def _read_pages(self, url):
    """
    Extract plain words from a Wikipedia dump and store them to the pages
    file. Each page will be a line with words separated by spaces.
    """
    wikipedia_path = download(url, self._cache_dir)
    with bz2.open(wikipedia_path) as wikipedia, ₩
            bz2.open(self._pages_path, 'wt') as pages:
        for _, element in etree.iterparse(wikipedia, tag='{*}page'):
            if element.find('./{*}redirect') is not None:
                continue
            page = element.findtext('./{*}revision/{*}text')
            words = self._tokenize(page)
            pages.write(' '.join(words) + '₩n')
            element.clear()
```

```
@classmethod
def _tokenize(cls, page):
    words = cls.TOKEN_REGEX.findall(page)
    words = [x.lower() for x in words]
    return words
```

_read_pages() 는 압축된 XML파일 형식의 Wikipedia 뭉치를 다운로드하고, 각 페이지마다 포매팅 정보를 제거하기 위해서 보통의 단어들을 추출하는 작업을 반복한다.

Wikipedia 말뭉치 준비하기

```
def _build_vocabulary(self, vocabulary_size):
    """
    Count words in the pages file and write a list of the most frequent
    words to the vocabulary file.
    """
    counter = collections.Counter()
    with bz2.open(self._pages_path, 'rt') as pages:
        for page in pages:
            words = page.strip().split()
            counter.update(words)
    common = ['<unk>'] + counter.most_common(vocabulary_size - 1)
    common = [x[0] for x in common]
    with bz2.open(self._vocabulary_path, 'wt') as vocabulary:
        for word in common:
            vocabulary.write(word + '₩n')
```

오타나 자주사용하지 않는 것을 줄이기 위해
vocabulary_size – 1을 한다.
단어집에 없는 값은 unk 토큰으로 처리한다.

Wikipedia 말뭉치 준비하기

```
import random


def skipgrams(pages, max_context):
    """Form training pairs according to the skip-gram model."""
    for words in pages:
        for index, current in enumerate(words):
            context = random.randint(1, max_context)
            for target in words[max(0, index - context): index]:
                yield current, target
            for target in words[index + 1: index + context + 1]:
                yield current, target
```

Wikipedia 말뭉치 준비하기

- 원-핫-인코딩
- Word2vec model

-CBOW(문맥을 통해 단어를 찾는 것)

-Skip-gram(단어를 통해 문맥 유추)

Wikipedia 말뭉치 준비하기

CBOW(Continuous Bag-of-Words)

하나의 문맥 단어가 주어지면 하나의 타켓 단어를 예측하는 모델이다.

Input : 단어를 수치화한 one-hot-vector

Ex) context word : bark, target word : dog

Input에 대해 target word가 나올 확률 : y_i

Output word 가 target word 이면 1 , 아니면 0 을 t_i에 저장

Error = y_i – t_i (if error > 0 : 실제 단어가 아닌데 모델이 잘 못 예측, else : 실제 단어인데 모델이 잘 못 예측)

모델 구조

```python
import tensorflow as tf
import numpy as np

from helpers import lazy_property


class EmbeddingModel:

    def __init__(self, data, target, params):
        self.data = data
        self.target = target
        self.params = params
        self.embeddings
        self.cost
        self.optimize

    @lazy_property
    def embeddings(self):
        initial = tf.random_uniform(
            [self.params.vocabulary_size, self.params.embedding_size],
            -1.0, 1.0)
        return tf.Variable(initial)

    @lazy_property
    def optimize(self):
        optimizer = tf.train.MomentumOptimizer(
            self.params.learning_rate, self.params.momentum)
        return optimizer.minimize(self.cost)

    @lazy_property
    def cost(self):
        embedded = tf.nn.embedding_lookup(self.embeddings, self.data)
        weight = tf.Variable(tf.truncated_normal(
            [self.params.vocabulary_size, self.params.embedding_size],
            stddev=1.0 / self.params.embedding_size ** 0.5))
        bias = tf.Variable(tf.zeros([self.params.vocabulary_size]))
        target = tf.expand_dims(self.target, 1)
        return tf.reduce_mean(tf.nn.nce_loss(
            weight, bias, embedded, target,
            self.params.contrastive_examples,
            self.params.vocabulary_size))
```

시퀀스 분류

- 전체 입력 시퀀스를 위한 클래스를 예측하기 위한 문제 셋팅
- Ex) 유전학, 재정, 감정분석 등에서 사용
- Imdb Movie 비평 데이터셋

⇒ Positive(T) or Negative(F)

시퀀스 레이블링 모델

1. Input = word

2. Cell(LSTM, GRU 등)

3. Output

4. softmax(T or F)

모델 학습

```
import tensorflow as tf

from helpers import AttrDict

from Embedding import Embedding
from ImdbMovieReviews import ImdbMovieReviews
from preprocess_batched import preprocess_batched
from SequenceClassificationModel import SequenceClassificationModel

IMDB_DOWNLOAD_DIR = './imdb'
WIKI_VOCAB_DIR = '../01_wikipedia/wikipedia'
WIKI_EMBED_DIR = '../01_wikipedia/wikipedia'

params = AttrDict(
    rnn_cell=tf.nn.rnn_cell.GRUCell,
    rnn_hidden=300,
    optimizer=tf.train.RMSPropOptimizer(0.002),
    batch_size=20,
)

reviews = ImdbMovieReviews(IMDB_DOWNLOAD_DIR)
length = max(len(x[0]) for x in reviews)

embedding = Embedding(
    WIKI_VOCAB_DIR + '/vocabulary.bz2',
    WIKI_EMBED_DIR + '/embeddings.npy', length)
batches = preprocess_batched(reviews, length, embedding, params.batch_size)

data = tf.placeholder(tf.float32, [None, length, embedding.dimensions])
target = tf.placeholder(tf.float32, [None, 2])
model = SequenceClassificationModel(data, target, params)

sess = tf.Session()
sess.run(tf.initialize_all_variables())
for index, batch in enumerate(batches):
    feed = {data: batch[0], target: batch[1]}
    error, _ = sess.run([model.error, model.optimize], feed)
    print('{}: {:3.1f}%'.format(index + 1, 100 * error))
```

기울기 자르기

```python
@lazy_property
def cost(self):
    cross_entropy = -tf.reduce_sum(self.target * tf.log(self.prediction))
    return cross_entropy

@lazy_property
def optimize(self):
    gradient = self.params.optimizer.compute_gradients(self.cost)
    try:
        limit = self.params.gradient_clipping
        gradient = [
            (tf.clip_by_value(g, -limit, limit), v)
            if g is not None else (None, v)
            for g, v in gradient]
    except AttributeError:
        print('No gradient clipping parameter specified.')
    optimize = self.params.optimizer.apply_gradients(gradient)
    return optimize
```

Cost = cross_entropy

Optimize

- apply_gradients()로 기울기를 수정하고 웨이트 변화를 적용.
- 기울기를 자르기 위해 값을 설정 후, 이 값보다 작으면 –limit, 크면 +limit 으로 설정.

정리

- 이 모델의 학습 성공은 네트워크의 구조와 하이퍼 파라미터뿐만 아니라 단어 임베딩의 품질에도 영향을 받는다.

7장

감사합니다