УДК 004.8
Код РИНЦ 20.00.00 DOI: 10.46554/Russian.science-2021.09-1-16/21

ПЕРСПЕКТИВЫ ИСПОЛЬЗОВАНИЯ ТЕХНОЛОГИЙ ГЛУБОКОГО ОБУЧЕНИЯ ДЛЯ АВТОМАТИЧЕСКОЙ ОБРАБОТКИ ТЕКСТОВЫХ ДОКУМЕНТОВ

© 2021 Воеков Артур Андреевич*
студент
Самарский государственный экономический университет
E-mail: ArturVoekov@mail.ru

Ключевые слова: глубокое обучение, машинное обучение, обработка естественного языка, текстовые документы, TensorFlow.

Статья посвящена краткому описанию перспектив и приемов применения технологий глубокого обучения с использованием языка Python и фреймворка TensorFlow для автоматической обработки больших массивов текстовых данных, например, платежных поручений, финансовых отчетов и т.д.

В настоящее время объём данных растёт в геометрической прогрессии. Значительная часть этих данных относится к языковым данных – текстовым или устным, – таким как электронные письма, сообщения, статьи в интернете и различные документы, в том числе финансовые. Первичный анализ таких документов различными специалистами, в том числе высокооплачиваемыми, может стоить существенных временных затрат, следовательно, имеет смысл автоматизировать данный процесс.

Для дальнейшей работы следует определиться с таким понятием как *корпус. Корпус* – это массив взаимосвязанных текстов (документов) на естественном языке. Корпус может быть и небольшим, но обычно состоит из десятков и сотен гигабайт данных в тысячах документов. Корпусы могут быть аннотированными, то есть текст или документы могут быть снабжены специальными пометками для алгоритмов обучения с учителем (например, для фильтров ошибочных платёжных поручений), или неаннотированными, что делает их кандидатами на тематическое моделирование и кластеризацию.

Корпус можно разбить на категории документов или на отдельные документы. Документы в корпусе могут различаться размерами, от отдельных сообщений до многотомных книг, но все они содержат текст (а иногда метаданные) и представляют набор связанных тем. Документы, в свою очередь, можно разбить на абзацы – *смысловые единицы* речи, которые обычно выражают одну идею. Абзацы также можно разбить на предложения – *синтаксические* единицы; законченное предложение структурно звучит как конкретное выражение. Предложения состоят из слов и знаков препинания – *лексических* единиц, которые определяют общий смысл, но гораздо более полезных в сочетаниях. Наконец, сами слова состоят из слогов, фонем, аффиксов и символов, то есть единиц, имеющих смысл, только когда они объединены в слова[1].

* Научный руководитель – **Ефимова Татьяна Борисовна**, кандидат экономических наук, доцент.

Далее следует кратко описать механизм классификации предложений с помощью сверточных нейронных сетей. *Сверточные нейронные сети* (convolutional neural network, CNN), существенно отличаются от полносвязных нейронных сетей и достигают высокого уровня точности в таких задачах как: классификация изображений, обнаружение объектов, распознавание речи и, конечно, классификацию предложений. Одно из главных преимуществ CNN заключается в том, что по сравнению с полностью связанным слоем сверточный слой в CNN имеет гораздо меньшее количество параметров. Это позволяет обучать более глубокие модели, не беспокоясь о переполнении памяти.

В сущности, CNN – это стек слоев трех типов, таких как сверточные слои, объединяющие слои и полностью связанные слои. Входные данные располагаются в *сверточном слое* (convolution layer). Сверточные слои могут чередоваться с *объединяющими слоями* (pooling layer) и слоями подвыборки (subsampling layer), уменьшающими размерность входных данных. Наконец, сверточные данные передаются на набор полностью связанных слоев, с выхода которых данные поступают на окончательный уровень классификации/регрессии, например, для классификации предложений.

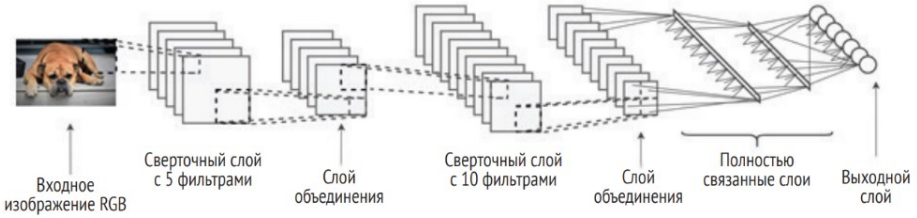

Рис. 1

Операция *свёртки*, используемая в свёрточных слоях, является центральной частью CNN. Пусть есть вход размером $n \times n$ и матрица весов (фильтр) $m \times m$, где $n \geq m$. Операция свертки перемещает фильтр по водному пространству. Обозначим вход буквой X, матрицу весов – W и выход – H. В каждой позиции i, j выход рассчитывается следующим образом:

$$h_{i,j} = \sum_{k=1}^{m} \sum_{l=1}^{m} w_{k,l} x_{i+k-1,j+l-1},$$

где $1 \leq i, j \leq n - m + 1$.

Здесь $x_{i,j}, w_{i,j}$ и $h_{i,j}$ – значения в (i, j)-й точке X, W и H соответственно. Из уравнения следует, что размерность выхода будет $(n - m + 1) \times (n - m + 1)$. Параметр m известен как *размер ядра*.[2]

Пример стандартной операции свертки визуально представлен на рис. 2.

Смещать фильтр на одну позицию не обязательно. Мы можем использовать и большее смещение. Величина смещения обозначается термином *шаг*. Модифицируем формулу операции стандартной свёртки, добавив шаги s_i и s_j:

$$h_{i,j} = \sum_{k=1}^{m} \sum_{l=1}^{m} w_{k,l} x_{(i-1)\times s_i + k, (j-1)\times s_j + l},$$

где $1 \le i \le floor[(n-m)/s_i]$ и $1 \le j \le floor[(n-m)/s_j]$.

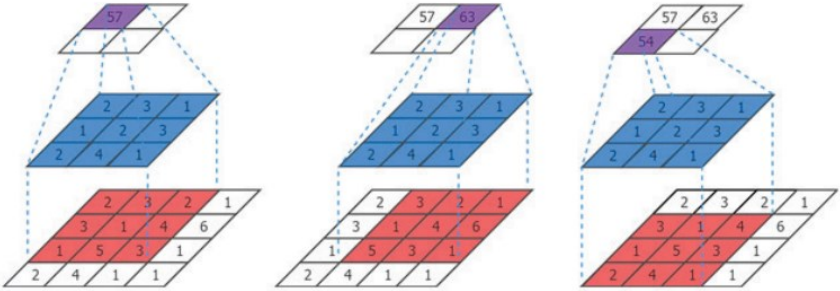

Рис. 2

Если фильтр не может выходить за края области данных, то даже при единичном шаге размерность выхода неизбежно уменьшается. Это нежелательное свойство, которое сильно ограничивает количество слоев нейронной сети. Для борьбы с естественной убылью размерности применяют так называемое заполнение, т. е. добавление нулей вокруг границы области входа, чтобы размерность выхода совпадала с размерностью входа:

$$h_{i,j} = \sum_{k=1}^{m}\sum_{l=1}^{m} w_{k,l} x_{i+k-(m-1),j+l-(m-1)},$$

где $x_{i,j} = 0$, если $i, j < 1$ или $i, j > n$.

Хотя операция свертки выглядит сложной с точки зрения математики, ее можно упростить до умножения матриц. Операция транспонированной свертки играет важную роль в CNN для сохранения градиентов во время обратного распространения. Рассмотрим пример, обозначения X, W и H те же. Выход h можно рассчитать как матричное умножение следующем образом. Пусть $n = 4$ и $m = 3$, развернём вход X слева направо и сверху вниз, получив представление:

$$x^{(16,1)} = x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}, x_{2,1}, x_{2,2}, x_{2,3}, x_{2,4}, \dots, x_{4,1}, x_{4,2}, x_{4,3}, x_{4,4}.$$

Далее определим новую матрицу A из W:

$$A^{(4,16)} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & 0 & w_{2,1} & w_{2,2} & w_{2,3} & 0 & w_{3,1} & w_{3,2} & w_{3,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{1,1} & w_{1,2} & w_{1,3} & 0 & w_{2,1} & w_{2,2} & w_{2,3} & 0 & w_{3,1} & w_{3,2} & w_{3,3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{1,1} & w_{1,2} & w_{1,3} & 0 & w_{2,1} & w_{2,2} & w_{2,3} & 0 & w_{3,1} & w_{3,2} & w_{3,3} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{1,1} & w_{1,2} & w_{1,3} & 0 & w_{2,1} & w_{2,2} & w_{2,3} & 0 & w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix}.$$

Выполнив перемножение матриц получаем H:

$$H^{(4,1)} = A^{(4,16)} X^{(16,1)}.$$

Изменив выход $H^{(4,1)}$ на $H^{(2,2)}$, получаем свёрнутый результат. Спроецируем этот результат обратно на n и m. Развёртывая вход $X^{(n,n)}$ в $X^{(n^2,1)}$ и создавая матрицу $A^{((n-m+1)^2,n^2)}$ из w, как показано выше, мы получаем $H^{((n-m+1)^2,1)}$, который затем преобразуем в $H^{(n-m+1,n-m+1)}$. Чтобы получить транспонированную свёртку, транспонируем A и получаем:

$$\hat{X}^{(n^2,1)} = (A^T)^{(n^2,(n-m+1)^2)} H^{((n-m+1)^2,1)},$$

где $\hat{X}$ – результат операции транспортирования свёртки.

Теперь рассмотрим реализацию свёрточной нейронной сети на практике с использованием пакета TensorFlow. В качестве исследуемых данных возьмём базу вопросов, доступную по адресу **http://cogcomp.org/Data/QA/QC/**, где каждому вопросу присвоена метка класса в соответствии с тем, о чем идет речь. Например, вопрос «Кто был одним из основателей группы Pink Floyd?» снабжён меткой личность.

Определим входы и выходы. На вход будет поступать пакет предложений, в котором слова представлены векторами с унитарным кодированием:

```
sent_inputs = tf.placeholder(shape=[batch_size,sent_length,vocabulary_
size],dtype=tf.float32,name='sentence_inputs')
sent_labels = tf.placeholder(shape=[batch_size,num_classes],dtype=tf.
float32,name='sentence_labels')
```

Далее определим три разных одномерных сверточных слоя с тремя разными размерами фильтров 3, 5 и 7 (предоставленными в виде списка в filter_sizes) и их соответствующими смещениями:

```
w1 = tf.Variable(tf.truncated_normal([filter_sizes[0],vocabulary_
size,1],stddev=0.02,dtype=tf.float32),name='weights_1')
b1 = tf.Variable(tf.random_uniform([1],0,0.01,dtype=tf.float32),name='bias_1')

w2 = tf.Variable(tf.truncated_normal([filter_sizes[1],vocabulary_
size,1],stddev=0.02,dtype=tf.float32),name='weights_2')
b2 = tf.Variable(tf.random_uniform([1],0,0.01,dtype=tf.float32),name='bias_2')

w3 = tf.Variable(tf.truncated_normal([filter_sizes[2],vocabulary_
size,1],stddev=0.02,dtype=tf.float32),name='weights_3')
b3 = tf.Variable(tf.random_uniform([1],0,0.01,dtype=tf.float32),name='bias_3')
```

Теперь рассчитаем три выхода, каждый из которых принадлежит одному из только что определенных слоев свертки.

```
h1_1 = tf.nn.relu(tf.nn.conv1d(sent_inputs,w1,stride=1,padding='SAME') + b1)
h1_2 = tf.nn.relu(tf.nn.conv1d(sent_inputs,w2,stride=1,padding='SAME') + b2)
h1_3 = tf.nn.relu(tf.nn.conv1d(sent_inputs,w3,stride=1,padding='SAME') + b3)
```

Рассчитаем максимальное значение каждого скрытого вывода, производимого каждым слоем свертки, и получим по одному скаляру для каждого слоя:

```
h2_1 = tf.reduce_max(h1_1,axis=1)
h2_2 = tf.reduce_max(h1_2,axis=1)
h2_3 = tf.reduce_max(h1_3,axis=1)
```

Затем объединяем полученные выходные данные по оси 1 (ширина), чтобы получить выходные данные с размерностью $(\text{размер пакета} \times q)$:

```
h2 = tf.concat([h2_1,h2_2,h2_3],axis=1)
```

Далее определяем полностью связанные слои, которые будут полностью подключены к полученному выше выходу:

```
w_fc1 = tf.Variable(tf.truncated_normal([len(filter_sizes),num_
classes],stddev=0.5,dtype=tf.float32),name='weights_fulcon_1')
b_fc1 = tf.Variable(tf.random_uniform([num_classes],0,0.01,dtype=tf.
float32),name='bias_fulcon_1')
logits = tf.matmul(h2,w_fc1) + b_fc1
predictions = tf.argmax(tf.nn.softmax(logits),axis=1)
```

Опеределяем функцию потерь, вычисляемой как перекрестная энтропия:

```
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_
v2(labels=sent_labels,logits=logits))
```

Для оптимизации сети будем использовать встроенный оптимизатор TensorFlow под названием **MomentumOptimizer**:

```
optimizer = tf.train.MomentumOptimizer(learning_
rate=0.01,momentum=0.9).minimize(loss)
```

Выполнение упомянутых выше операций для оптимизации нейросети и оценки тестовых данных дает нам точность на проверочных данных, близкую к 90 % (500 проверочных предложений).

К сожалению, объём данной статьи не позволяет подробнее описать процессы предобработки данных и другие алгоритмы глубокого обучения, но даже описанное выше можем стать базисом для дальнейших исследований.

---

[1] Benjamin Bengfort, Rebecca Bilbro, and Tony Ojeda. Applied Text Analysis with Python; Enabling Language-Aware Data Products with Machine Learning. 2019. С. 43.

[2] Thushan Ganegedara. Natural Language Processing with TensorFlow; Teach language to machines using Python's deep learning library. 2020. С. 138.

PROSPECTS FOR USING DEEP LEARNING TECHNOLOGIES FOR AUTOMATIC PROCESSING OF TEXT DOCUMENTS

© 2021 Voekov Artur Andreevich
Student
Samara State University of Economics
E-mail: ArturVoekov@mail.ru

Keywords: deep learning, machine learning, natural language processing, text documents, TensorFlow.

The article is devoted to a brief description of the prospects and techniques for using deep learning technologies using the Python language and the TensorFlow framework for automatic processing of large arrays of text data, for example, payment orders, financial reports, etc.