МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОННОЙ ТЕХНИКИ
Институт системной и программной инженерии
и информационных технологий (Институт СПИНТех)

Лабораторный практикум по курсу
"Свёрточные нейронные сети в компьютерном зрении"
(02/20 – 06/20)

Лабораторная работа 3
«Введение в TensorFlow»

На этом лабораторном занятии вы познакомитесь с библиотекой общего назначения TensorFlow, которая позволяет создавать любой граф вычислений и специализированные надстройки, которые реализуют разные компоненты нейронных сетей: обычные слои, свёрточные, рекуррентные, рекуррентные слои из LSTM или GPU, современные алгоритмы оптимизации и много другое.

Итак, в этом компьютерном практикуме вы изучите основные понятия, операции и принцип работы с TensorFlow.

Прежде чем приступить к выполнению лабораторных заданий, настоятельно рекомендуется ознакомиться с материалом лекций, а также с дополнительными материалами, имеющими отношение к библиотеке общего назначения TensorFlow, а также с языком программирования Python и основами работы с Jupyter Notebook, подробно рассмотренными в первом компьютерном практикуме данного курса.

Замечание: подавляющее большинство библиотек, в том числе и TensorFlow, имеют основной интерфейс к языку программирования Python. Этот язык стал дефакто стандартом в современном машинном обучении и обработке данных. Мы не можем включить в данный курс ещё и руководство по языку Python и в дальнейшем будем просто предполагать, что сам язык и основная его вычислительная библиотека NumPy вам знаком. Если же это не так, то очень рекомендуем вам познакомиться Python.

1. Почему TensorFlow

Среди библиотек общего назначения, которые способны строить граф вычислений и проводить автоматическое дифференцирование, долгое время бесспорным лидером оставалась Theano. Однако в ноябре 2015 года Google выпустила (с открытым исходным кодом) библиотеку TensorFlow, предназначенную для того же самого. Библиотеки Theano и TensorFlow на данный момент остаются двумя бесспорными лидерами в этой области, и сложно уверенно рекомендовать одну из них. Для данного курса лабораторных работ мы выбрали именно TensorFlow, так как можно ожидать, что мощь разработчиков Google в итоге приведет к тому, что новые интересные возможности будут появляться в TensorFlow быстрее, чем в Theano. Время покажет, оправдается ли наше предположение.

2. Граф вычислений

В этом разделе мы введём основополагающее понятие для реализации алгоритмов обучения нейронных сетей. Оказывается, что если у нас получится представить сложную функцию как композицию более простых, то мы сможем эффективно вычислить её производную по любой переменной, что и требуется для градиентного спуска. Самое удобное представление в виде композиции — это представление в виде ***графа вычислений***. Граф вычислений — это граф, узлами которого являются функции (обычно достаточно простые, взятые из заранее фиксированного набора), а ребра связывают функции со своими аргументами.

Это проще увидеть своими глазами, чем формально определять; посмотрите на рис. 2.7, где показаны три графа вычислений для одной и той же функции:

$$f(x, y) = x^2 + xy + (x + y)^2.$$

На рис. 2.7, а граф получился совсем прямолинейный, потому что мы разрешили использовать в качестве вершин унарную функцию «возведение в квадрат». А на рис. 2.7, б граф чуть более хитрый: явное возведение в квадрат мы заменили обычным умножением. Впрочем, он от этого не сильно увеличился в размерах, потому что в графе вычислений на рис. 2.7, б разрешается по несколько раз переиспользовать результаты предыдущих вычислений и достаточно просто подать один и тот же x два раза на вход функции умножения, чтобы вычислить его квадрат. Для сравнения мы нарисовали на рис. 2.7, в граф той же функции, но без переиспользования; теперь он становится деревом, но в нем приходится повторять целые большие поддеревья, из которых раньше могли выходить сразу несколько путей к корню дерева.

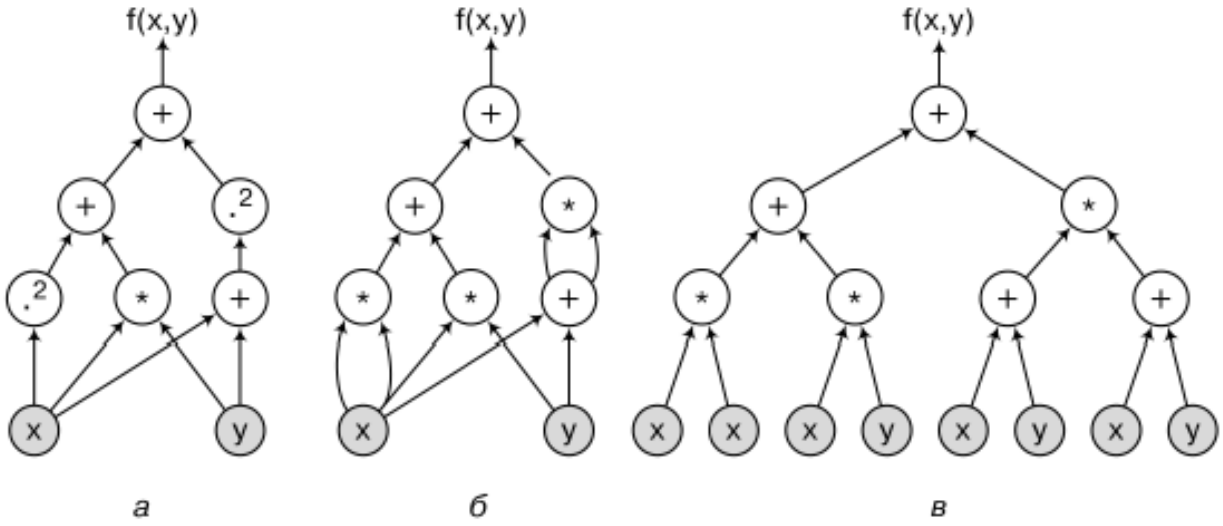

Рис. 2.7. Графы вычислений для функции $f(x, y) = x^2 + xy + (x + y)^2$:
а — с использованием функций $+$, $\times$ и ^{2}; *б* — с использованием функций $+$ и $\times$;
в — в виде дерева с использованием функций $+$ и $\times$

В нейронных сетях в качестве базисных, элементарных функций графа вычислений обычно используют функции, из которых получаются нейроны. Элементарной может служить любая функция, для которой мы сможем легко вычислить ее саму и ее производную по любому аргументу. В частности, прекрасно подойдут любые функции активации нейронов, о которых мы будем подробно говорить в чуть позже.

Вернемся к машинному обучению. Цель машинного обучения — подобрать модель (чаще всего здесь имеются в виду веса модели, заданной в параметрическом виде) таким образом, чтобы она лучше всего описывала данные. Под «лучше всего» здесь, как правило, имеется в виду оптимизация некоторой функции ошибки. Обычно она состоит из собственно ошибки на обучающей выборке (функции правдоподобия) и регуляризаторов (априорного распределения), но сейчас нам достаточно просто считать, что есть некая довольно сложная функция, которая дана нам свыше, и мы хотим ее минимизировать. Один из самых простых и универсальных методов оптимизации сложных функций — это градиентный спуск. Мы также только что выяснили, что один из самых простых и универсальных методов задать сложную функцию — это граф вычислений.

В общем виде алгоритм такой: предположим, что нам задан некоторый направленный ациклический граф вычислений G = (V,E), вершинами которого являются функции $g \in V$, причем часть вершин соответствует входным переменным x1,... ,xn и не имеет входящих ребер, одна вершина не имеет исходящих ребер и соответствует функции f (весь граф вычисляет эту функцию), а ребра показывают зависимости между функциями, стоящими в узлах. Тогда мы уже знаем, как получить функцию f, стоящую

в «последней» вершине графа: для этого достаточно двигаться по ребрам и вычислять каждую функцию в топологическом порядке.

А чтобы узнать частные производные этой функции, достаточно двигаться в обратном направлении.

Такой подход называют алгоритмом ***обратного распространения*** (backpropagation, backprop, bprop), потому что частные производные считаются в направлении, обратном ребрам графа вычислений. А алгоритм вычисления самой функции или производной по одной переменной называют алгоритмом прямого распространения (forward propagation, fprop).

Важное замечание: обратите внимание, что мы ни разу ещё всерьез не упомянули нейронные сети! И действительно, метод вычисления производных/градиентов по графу вычислений сам по себе совершенно никак не связан с нейронными сетями. Дело в том, что библиотеки Theano и TensorFlow, на которых делается большая часть глубокого обучения, — это, вообще говоря, библиотеки для ***автоматического дифференцирования***, а не для обучения нейронных сетей. Все, что они делают, — позволяют вам задать граф вычислений и чертовски эффективно, с распараллеливанием и переносом на видеокарты, вычисляют градиент по этому графу.

3. Знакомство с TensorFlow

Основная абстракция, которая потребуется нам для того, чтобы понять всё, что происходит в коде таких библиотек, как TensorFlow или Theano, – это граф вычислений. Программа, реализующая TensorFlow, обычно просто задаёт граф вычислений, а потом запускает процедуру вроде *session.run*, которая выполняет вышеописанные вычисления и получает собственно результаты.

Основной объект, которым оперирует TensorFlow, - это как можно понять буквально из названия, ***тензор***, или многомерный массив чисел.

Переменная в TensorFlow – это некий буфер в памяти, который содержит тензоры. Переменные нужно явным образом инициализировать. Чтобы объявить переменную, нужно задать способ её инициализации; при желании можно также назначить ей имя, на которое можно будет потом ссылаться. Например:

```
In [3]: w=tf.Variable(tf.random_normal([3,2],mean=0.0,stddev=0.4),name='weights')
        b=tf.Variable(tf.zeros([2]), name='biases')
```

В этом коде мы объявили две переменные: w с именем weights и b с именем biases.

Все переменные обязательно нужно инициализировать. Самый простой способ – сделать это перед началом вычислений:

Упражнение

Запустите Jupyter Notebook и создайте новый блокнот, нажав кнопку «Создать» на панели инструментов в правом верхнем углу и выберите «Python 3». Дайте вашему блокноту соответствующее название из панели управления Jupyter, не забудьте, что для этого сперва необходимо закрыть ядро. Далее запустите заново ваш блокнот и напечатайте следующий код для импортирования библиотеки TensorFlow:

```
In [2]: import tensorflow.compat.v1 as tf
        tf.disable_v2_behavior()
```

Затем проинициализируйте локальную переменную int:

```
In [4]: init = tf.local_variables_initializer()
```

Но это не работает в тех случаях, когда нужно инициализировать переменные из значений других переменных; в таких случаях синтаксис будет следующим:

Упражнение

Объявите переменные w и b, как было описано выше. Затем напечатайте следующий код и выполните его:

```
In [5]: w2 = tf.Variable(w.initialized_value(), name='w2')
```

В задачах машинного обучения необходимо применять одну и ту же последовательность операций к разным наборам данных. В частности, обучение мини-батчами подразумевает периодическое вычисление результата и ошибки на новых примерах. Для удобства передачи новых данных в TensorFlow существуют специальные тензоры – ***заглушки***, *tf.placeholder*, которым нужно сначала передать только тип данных и размерности тензора, а сами данные будут подставлены уже в момент вычислений:

Упражнение

Создайте *заглушки* – специальные тензоры для удобства передачи различных данных, используя следующий код:

```
In [6]: a = tf.placeholder(tf.int16)
        b = tf.placeholder(tf.int16)

In [7]: add = tf.add(a, b)
        mul = tf.multiply(a, b)

In [8]: with tf.Session() as sess:
            print("a=2, b=3")
            print("Addition: %i" % sess.run(add, feed_dict={a: 2, b: 3}))
            print("Multiplication: %i" % sess.run(mul, feed_dict={a: 2, b: 3}))

        a=2, b=3
        Addition: 5
        Multiplication: 6
```

В TensorFlow реализован полный набор операций над тензорами из NumPy с поддержкой матричных вычислений над массивами разной формы (broadcasting). Например, в реальных задачах часто возникает необходимость к каждому столбцу матрицы поэлементно добавить один и тот же вектор. В TensorFlow это делается самым простым из возможных способов:

Упражнение

Прибавьте поэлементно к каждому столбцу матрицы один и тот же вектор, используя следующий код:

```
with tf.Session() as sess:
    m = tf.Variable(tf.random_normal([10,100],mean=0.0,stddev=0.4),name='matrix')
    v = tf.Variable(tf.random_normal([100],mean=0.0,stddev=0.4),name='vector')
    result = m + v
```

Здесь m – это матрица размера 10x100, а v – вектор длины 100, и при сложении v будет прибавлен к каждому столбцу m.

Broadcasting применим для всех поэлементных операций над двумя тензорами и устроен следующим образом. Размерности двух тензоров последовательно сравниваются, начиная с конца; при каждом сравнении необходимо выполнение одного из условий:

- либо размерности равны;
- либо одна из размерностей равна 1.

Помимо бинарных операций в TensorFlow реализован широкий ассортимент унарных операций: возведение в квадрат, взятие экспоненты или логарифма, а также широкий спектр редукций. Например, иногда нам бывает необходимо вычислить среднее значение не по всему тензору, а, скажем, по каждому элементу мини-батча.

Упражнение

Вычислите среднее значение тензора, используя следующий код:

```
In [13]: tensor = tf.placeholder(tf.float32, [10, 100])
         result = tf.reduce_mean(tensor, axis=1)
```

При этом среднее будет вычисляться по второй размерности тензора tensor, то есть мы десять раз усредним 100 чисел и получим вектор длины 10.

```
In [18]: result.shape
Out[18]: TensorShape([Dimension(10)])
```

Напоследок, давайте разберём перемножение матриц в TensorFlow.

Упражнение

Создайте константную матрицу размером 1x2. Операция constant по умолчанию добавляется в качестве узла на вычислительный граф. Для этого введите следующий код:

```
In [ ]: matrix1 = tf.constant([[3., 3.]])
```

Создайте ещё одну константу – матрицу размером 2x1:

```
In [ ]: matrix2 = tf.constant([[2.],[2.]])
```

Создайте операцию умножения матриц - Matmul, которая принимает в качестве входных данных «matrix1» и «matrix2»:

```
In [ ]: product = tf.matmul(matrix1, matrix2)
```

Для запуска операции matmul мы вызываем метод сессии 'run ()', передавая "product", который представляет выходные данные для операции matmul. Все входы, необходимые для matmul, автоматически запускаются сессией. Обычно они работают параллельно. Вызов 'run (product)', таким образом, вызывает выполнение трех операций в графе вычислений: двух constants и matmul:

```
In [22]: with tf.Session() as sess:
             result = sess.run(product)
             print(result)

         [[12.]]
```

```
In [23]: result.shape

Out[23]: (1, 1)
```

Задание к лабораторной работе

1) Выполните все упражнения из данного документа.
2) Представьте отчёт в виде файла **.ipynb** с результатами вашей работы.
3) Расскажите основные особенности TensorFlow.