Вопрос Инфо

Всем привет! В сегодняшнем семинаре мы рассмотрим задачу генерации текста с помощью рекуррентных нейронных сетей. Наверняка, многие из вас сталкивались со следующей проблемой — вы пишете код и не можете придумать хорошее название для новой переменной. Или другая проблема — вы регистрируете личный кабинет в каком-либо сервисе и хотите придумать короткий и красивый, запоминающийся логин... С решением обеих этих проблем нам может помочь подход, разобранный в лекции, а именно рекуррентные нейронные сети. Итак, в сегодняшнем семинаре мы разберём проблему генерации новых имён с помощью рекуррентных нейронных сетей. А именно: сегодня мы напишем рекуррентную нейронную сеть с нуля, мы не будем использовать какие-либо готовые решения из pytorch, tensorflow или любых других библиотек, а будем писать рекуррентную нейронную сеть руками. А затем мы научим нашу нейронную сеть генерировать новые имена, обучив её на небольшой коллекции. Для начала, давайте посмотрим на данные, которые у нас есть. Наш датасет состоит из 9000 имён, которые написаны латиницей. Они лежат в файле под названием russian_names.txt, давайте посмотрим на наш файл. Как вы можете видеть, в этом это датасете содержатся не только имена, но и фамилии, и при этом — это русские имена. Все имена отсортированы в алфавитном порядке. Вы можете заметить небольшую особенность — каждое имя в нашем датасете будет начинаться с пробела, и вот в этой строке кода мы добавляем пробел перед каждым именем — зачем это делается? Кажется, это довольно странное решение.

Вопрос 1

Выполнен

Баллов: 1,00 из 1,00

Зачем нам нужен пробел перед каждым словом?

Выберите один ответ:

- a. если не будет специального символа, с которого начинается генерация, то мы лишим нашу модель способности выбирать первый символ последовательности
- b. это выравнивающий токен (паддинг), чтобы длины последовательностей совпадали

Ваш ответ верный.

Вопрос 2

Выполнен

Баллов: 1,00 из 1,00

С помощью какой встроенной функции из pytorch можно делать padding для последовательностей?

(см. документацию torch.nn.utils.rnn).

Выберите один ответ:

- a. torch.nn.utils.rnn.pad_packed_sequence()
- b. torch.nn.utils.rnn.pack_sequence()
- c. torch.nn.utils.pad_sequence()
- d. torch.nn.utils.rnn.pad_sequence()

Ваш ответ верный.

Вопрос Инфо

Здесь на экране вы можете увидеть изображение из лекции, а именно — основной принцип работы и рекуррентной нейронной сети: небольшую схему, которая поможет нам писать код дальше. Будем писать класс, который мы назовём "CharRnnCell". Что у нас есть в этом классе? У нас есть функции "forward" и есть функция "initial state". "initial state" просто заполняет векторы нулями, выход функции "initial state" соответствует переменной "h0", или нулевому скрытому состоянию нашей нейронной сети. Рассмотрим более подробно, что происходит внутри функции "forward". Для начала мы преобразовываем наши входные векторы в эмбеддинги с помощью слоя, который мы берём из библиотеки pytorch, а именно "nn embedding". Дальше мы конкатенируем текущий входной вектор из переменной "x embedding" и скрытое состояние из предыдущего шага, и дальше, с помощью "rnn update" (именно так мы назвали линейный слой функции "init"), мы делаем следующий шаг — предсказываем следующее скрытое состояние. Дальше в ход идёт нелинейность — мы применяем гиперболический тангенс к выходу этого слоя. В итоге, на выход мы передаём следующее скрытое состояние и вероятности для следующего символа. А именно, мы получим 52 вероятности по каждому символу из нашего словаря.

Вопрос Инфо

Теперь мы можем переходить в тренировке нашей нейронной сети. После того, как мы написали один шаг RNN, мы можем вызвать его в цикле и предсказывать символы на каждом шаге нашего цикла. Теперь мы можем начинать тренировку нашей сети. Минимизируя кросс-энтропию, либо максимизируя логарифм правдоподобия нашей модели (что — то же самое) — обучать нашу сеть. Берём матрицу ID токенов, сдвинутую на "i" символов влево, так, чтобы именно "i"-ый символ был следующем символом для предсказания на "i"-ом шаге. Такая матрица хранится в переменной "batch index" (а именно, вот здесь) и, дальше, мы можем переходить к обучению сети. Мы делаем "backward pass" — именно здесь мы вычисляем градиенты нашей лосс-функции по параметрам, делаем шаг с помощью "opt.step" и не забываем сделать "zero_grad", если мы этого не сделаем то градиенты из предыдущих шагов будут накапливаться, аккумулироваться, и это приведёт к неправильному обучению сети. Кроме того, будем визуализировать процесс обучения нашей нейронной сети с помощью библиотеки matplotlib, без каких-либо дополнительных инструментов. Запускаем обучение!

Вопрос 3

Выполнен

Баллов: 1,00 из 1,00

Зачем нужно на каждой итерации обучения в коде семинара вызывать zero_grad()?

Выберите один ответ:

a. zero_grad() существенно ускоряет процесс обучения

b. zero_grad() переводит модель в режим обучения, без вызова этой функции градиенты не будут вычисляться (будет работать режим валидации)

c. По умолчанию градиенты при каждом вызове loss.backward() аккумулируются, с помощью zero_grad() мы обнуляем градиенты перед новым вызовом backward()

Ваш ответ верный.

Вопрос 4

Выполнен

Баллов: 1,00 из 1,00

В семинаре при рассказе о генерации текста упоминается "температура". Что же имеется в виду?

Температура в softmax - параметр, который отвечает за “случайность” итогового распределения. Если устремить температуру к нулю, итоговое распределение вырождается в one-hot (элемент с максимальным значением выбирается с вероятностью 1.0). Такое поведение согласуется и с физическими явлениями. С возрастанием температуры энтропия (мера хаотичности) системы растет, а значит события становятся более случайными (т.е. распределение стремится к равномерному).

Исходя из информации выше, выберите правильные утверждения:

Выберите один или несколько ответов:

- [x] a. При температуре, стремящейся к бесконечности, генерация любого символа имеет примерно одинаковую вероятность
- [x] b. Чем ниже температура генерации, тем выше уверенность модели в сгенерированных результатах и ниже разнообразие сгенерированной выборки
- [ ] c. При высокой температуре генерации модель выдает варианты, уверенность в которых наиболее высока
- [ ] d. При увеличении температуры генерации модель начинает отличаться меньшим разнообразием
- [x] e. При очень маленькой температуре есть вероятность получать каждый раз один и тот же результат

Ваш ответ верный.