Assignment 2. Language Model Pre-training for Text Classification with Recurrent Neural Networks in Pytorch

April 2021

В данном задании предлагается реализовать на Pytorch рекуррентную языковую модель Zaremba [1], а также ряд улучшений предложенных в последующих статьях и рассмотренных на лекции. Затем предлагается реализовать классификатор текстов, предобучающийся на задаче языкового моделирования и дообучающийся на задаче анализа тональности текстов.

1 Теоретическая чать

1.1 Прямой проход рекуррентной языковой модели (hypothesis)

Рассмотрим последовательность: $x_1, x_2, \ldots, x_T$, где x_i - индекс токена из словаря V. Предположим, мы хотим обучать 2-х слойную языковую модель на основе LSTM, которая на i-ом шаге, для каждого слова из словаря выдает оценку $\hat{y}_i[w] = \hat{P}(w|x_1, \ldots, x_{i-1})$ вероятности того, что токен w следует за токенами $x_1, \ldots, x_{i-1}$.

1. Выпишите формулы прямого прохода указанной модели: как вычисляется оценка вероятности $\hat{y}_i[w] = \hat{P}(w|x_1, \ldots, x_{i-1})$? Для упрощения выпишите сначала формулы для отдельных слоев, а затем, воспользовавшись ими, выпишите полную формулу прямого прохода.

2. Чему равна оценка вероятности последовательности $x_1, x_2, \ldots, x_T$ языковой моделью, то есть чему равен
$\hat{P}(x_1, \ldots, x_T)$?

1.2 Оценочная функция для языковой модели (loss)

Задача языкового моделирования на каждом шаге сводится к задаче многоклассовой классификации, где количество классов равно количеству слов в словаре. Важной особенностью языкового моделирвания является то, что входы и выходы формируются из одного текста (не нужна размеченная выборка). Тренировочная выборка состоит из текстов $Texts = \{Text_1, \ldots, Text_N\}$

Обычно в задачах классификации минимизируют некоторый функционал потерь во время обучения модели. В языковом моделировании стандартной функцией потерь является пословная кросс-энтропия (усреднение по каждому шагу последовательности).

Для простоты выпишем функционал потерь, для одного текста из выборки:

$$CE(\theta, Text_i) = \frac{1}{T_i} \sum_{t=1}^{T_i} ce(oh(x_{i,t}), \hat{P}(w|x_{i,1}, \ldots, x_{i,t-1})) \to \min_{\theta},$$

где θ – веса языковой модели, $ce(y, \hat{y})$ – кросс-энтропия между предсказанным распределением и one-hot вектором для следующего слова.

[1] Wojciech Zaremba, Ilya Sutskever, Oriol Vinyals. Recurrent Neural Network Regularization, 2015. https://arxiv.org/abs/1409.2329

1. Покажите, что минимизация кросс-энтропии эквивалентна максимизации правдоподобия правильных текстов.

2. Какие специальные токены (не входящие в текст) используют в языковой модели и какую роль они играют при вычислении вероятности текста / генерации нового текста?

1.3 LSTM.

Основной проблемой классических рекурентных сетей является проблема затухающих градиентов (vanishing gradients). В отличие от классических моделей, языковая модель на основе LSTM меньше страдает от этой проблемы.

1. Выпишите формулы прямого прохода для LSTM слоя. Считайте, что на входе слоя – последовательность векторов $e_1, ..., e_T$.

2. Объясните, что помогает LSTM бороться с проблемой затухающих градиентов и почему?

1.4 Перплексия.

Стандартной метрикой для оценки качества языковой модели является пословная перплексия (per word perplexity). Для последовательности токенов $x_1, \ldots, x_T$ метрика записывается следующим образом:

$$Perplexity(x_1, \ldots, x_T) = \exp(-\frac{1}{T}\sum_{t=1}^{T} \log \hat{P}(x_t|x_1, \ldots, x_{t-1}))$$

1. Выразите пословную перплексию через кросс-энтропию.

1.5 Языковое моделирование для предобучения рекуррентной сети.

Для более стабильного обучения и улучшения качества классификатора текста на основе рекурентных сетей часто используется предобучение весов рекуррентных слоев и эмбедингов слов на задаче языкового моделировани. Особенно критично такое предобучение в случае классификации длинных текстов.

1. Как вы думаете, какие причины приводят к нестабильному обучению рекуррентных сетей даже с LTSM ячейкой на задаче классификации длинных текстов при случайной начальной инициализации всех весов?

2. Предположите с чем связано улучшение финального качества классификаторов текстов при предобучении их языковому моделированию?

2 Практическая чать

2.1 Загрузка датасета.

Загрузите PennTreeBank(PTB) датасет, он содержится в репозитории `https://github.com/nvanva/filimdb_evaluation`, для этого выполните следующие команды:

1. git clone `https://github.com/nvanva/filimdb_evaluation.git`

2. cd filimdb_evaluation

3. ./init.sh

Нужные файлы: PTB/ptb.train.txt, PTB/ptb.valid.txt, PTB/ptb.test.txt. Напишите ответы на следующие вопросы:

1. Составьте таблицу, в которой указано число токенов, уникальных токенов, предложений для каждой из трех частей датасета.

2. Приведите 10 самых частотных и 10 самых редких токенов с их частотами.

3. Какие специальные токены уже есть в выборке, что они означают?

2.2 Генерацей батчей.

В посимвольной языковой модели, рассмотренной на семинаре, примеры были независимы, и каждый батч состоял ровно из одного примера. В PTB примеры - это последовательные предложения из текста. Знание предыдущего предложения помогает предсказать слова из следующего. Поэтому стандартный способ формирования батчей для PTB следующий:

1. Разбиваем каждое предложение на токены (поскольку данные уже предобработаны, токенизация сводится к разделению по пробельным символам). В конец каждого предложения добавляем специальный токен `<eos>`. Конкатенируем все в один длинный список токенов L.

2. Делим L на $batch\_size$ списков равной длины, а остаток выкидываем. Формируем батчи так, что i-ая строка каждого батча - это подсписок i-ого списка, и каждая i-ая строка в в следующем батче является продолжением i-ой строки в предыдущем батче.

3. Все батчи имеют одинаковый размер $(batch\_size, num\_steps)$. Паддинги при данном способе формирования батчей не нужны.

4. Таргет текст Y получается аналогичным образом, но из списка $L[1:]$.

5. Так как продолжение предложений из одного батча будет находиться в следующем батче, рекомендуется в качестве начального скрытого состояния (h_0, c_0) использовать последнее скрытое состояния на предыдущем батче $(h_{num\_steps}, c_{num\_steps})$. Для самого первого батча в качестве (h_0, c_0) используйте нулевые векторы.

6. Для первого батча начальное скрытое состояние рекуррентных слоев будет нулевым, то есть информация о предыдущих токенах не используется, что ухудшает перплексию. При оценке модели можно это исправить (пожертвовав скоростью), если использовать $batch\_size = 1$.

Напишите функцию $batch\_generator(data\_path)$, которая принимает на вход путь к файлу, а на выходе выдаёт итератор по батчам (батчи должны быть разбиты описанным выше способом). Советы:

1. Проверьте, что каждая строка в следующем батче является продолжением соответствующей строки в предыдущем батче.

2. Проверьте, что для каждого батча Y получается из X сдвигом на один токен вправо.

2.3 Реализация LSTM LM.

В этой части задания Вам необходимо реализовать архитектуру языковой модели на основе LSTM. Для этого мы рекомендуем Вам реализоать следующие классы:

1. Класс LSTMCell - ячейка обрабатывающая один шаг последовательности.

2. Класс LSTMLayer - один слой рекурентной сети, который обрабатывет последовательность целиком. Используйте свою реализацию LSTMCell внутри.

3. Класс LSTM - многослойная однонаправленная сеть, основанная на LSTM. Используйте внутри свою реализацию LSTMLayer.

4. Реализуйте LSTM LM для задачи языкового моделирования на датасете PTB. Реализация не сильно отличается от показанной на семинаре.

Далее опишем подробнее, что Вам нужно сделать в каждом пункте.

2.3.1 Класс LSTMCell

На семинаре был показан пример, как реализовать свою RNN ячейку. Там мы сделали отдельный класс, наследованный от класса torch.nn.Module, который содержал параметры ячейки, а именно 2 матрицы весов и 2 вектора сдвига (байесы). Мы рекомендуем Вам реализовать свою LSTM ячейку по формулам из документации PyTorch: `https://pytorch.org/docs/stable/nn.html#torch.nn.LSTMCell`. Там ячейка LSTM содержит 8 матриц и векторов смещения тоже 8. Заводить столько параметров, конечно, можно, но лучше не стоит. Можно заметить, что в LSTM матрицы умножаются на одни и те же вектора, 4 матрицы умножаются на вектор скрытого состояния с предыдущего шага и 4 матрицы умножаются на векторное представление слова на текущем шаге. А значит мы можем каждую четверку матриц конкатенировать в одну большую матрицу (исходный размер матриц (h, h), а после конкатенации $(4h, h)$). С векторами сдвига все аналогично. Таким образом, у нас снова 2 матрицы весов и 2 вектора байеса. Также такая реализация более быстрая, чем наивная, потому что лучше параллелить немного больших матриц, чем много матриц поменьше. Но, ведь после умножения там разные функции активации?! Да, там есть 3 сигмоиды и 1 гиперболический тангенс. Тогда нам нужно разделить результат умножения на 4 куска, чтобы к каждому применить правильную функцию активации. В этом Вам поможет функция torch.Tensor.chunk: `https://pytorch.org/docs/stable/tensors.html#torch.Tensor.chunk`. Она применяется прямо к тензору (результату умножения). Вообщем, к этому моменту Вы должны уже понимать, как написать LSTM ячейку (свой класс LSTMCell). Не забывайте, все операции, которые ячейка делает над входом, нужно реализовывать в методе forward(...). Этот метод должен быть обязательно определен в классе, когда Вы наследуетесь от torch.nn.Module!

2.3.2 Класс LSTMLayer

После реализации ячейки, которая обрабатывает только лишь 1 шаг последовательсности, Вам необходимо реализовать слой сети (класс LSTMLayer), который будет обрабатывать батч последовательностей целиком. Не забывайте, что Вам где-то нужно создавать начальное состояние для сети, если оно не было передано (например, на первом шаге последовательности).

2.3.3 Класс LSTM

Последний класс, который Вам необходимо реализовать - это сама модель LSTM. Этот класс уже способен комбинировать несколько слоев рекурентной сети, делать dropout между слоями. Этот класс должен использовать объекты классов, реализованных Вами ранее.

2.3.4 Класс PTBLM

После реализации базовых блоков, все готово для реализации полноценной многослойной LSTM LM. Вам нужно добавить слой эмбедингов, реализованную LSTM и линейный слой для получения распределения вероятностей.

2.4 Обучение языковой модели.

Теперь Вам предстоит испытать реализованную модель в действии. Обучите модель на PTB датасете. Для начала можете попробовать дефолтные параметры:

1. batch_size = 64 примера
2. num_steps = 35
3. num_layers = 2
4. emb_size = 256
5. hidden_size = 256
6. vocab_size = (размер словаря, должен быть равен числу уникальных токенов в PTB)
7. dropout_rate = 0.2
8. num_epochs = 13
9. learning_rate = 0.01
10. lr_decay = 0.9 (после 6 эпохи) - То есть learning_rate умножается на это значение после 6 эпохи
11. Оптимизатор - Adam

В процессе обучения следите за перплексией на тренировочной и на валидационной выборках. Не дайте Вашей модели переобучиться. После проверки работоспособности модели Вы можете приступить к подбору гиперпараметров. Показателем работоспобности модели могут служить показатели перплексии:

1. На тренировочной выборке < 70.
2. На валидационной и тестовой выборках $< 140 - 150$.

Попробуйте улучшить перплексию за счет увеличения размерности скрытого слоя и подбора dropout, learning rate, схемы изменения learning rate в процессе обучения, числа эпох обучения, распределения для инициализации весов. При подборе ориентируйтесь на значения гиперпараметров из статьи Zaremba.

Вопрос. Приведите графики изменения перплексии на обучающей и валидационной выборки в процессе обучения модели на PTB для различных значений гиперпараметров, которые вы пробовали. Приведите таблицу со всеми метриками (из лидерборда) для лучшей модели, а также гиперпараметры этой модели.

2.5 Генерация предложений.

Написать функцию для генерации предложений с помощью обученной модели. Используйте $batch\_size$=1, num_steps=1. Для генерации первого слова подаем модели в качестве начального скрытого состояния нулевой вектор, в качестве входа - случайное слово из словаря. Вычисляем температурный softmax $softmax(logits/Temp)$ и в качестве следующего слова сэмплируем слово из полученного распределения. Для генерации следующего слова используем последнее скрытое состояние рекурентного слоя в качестве начального состояния, а сгенерированное слово в качестве входа.

Вопрос. Приведите по 10 примеров сгенерированных предложений при разных температурах.

2.6 Tied input-output embeddings / Tied Softmax

Реализуйте связывание входных и выходных эмбедингов слов – для этого используйте одну и ту же матрицу эмбедингов в первом и последнем слое. Вектор смещений (bias) в последнем слое остается независимым.

Вопрос. Приведите на одном графике кривые изменения перплексии на обучающей и валидационной выборки в процессе обучения модели на PTB для модели с независимыми и общими входными и выходными эмбедингами.

2.7 Dropout

Базовая версия рекуррентной языковой модели, предложенная в статье Zaremba, использует dropout на входе первого, выходе второго, и между двумя рекуррентными слоями. Попробуйте добавить в вашу реализацию дополнительные варианты dropout и добиться улучшения перплексии с их помощью. Для этого стоит попробовать увеличить размер скрытых представлений LSTM до 1000, а размер эмбедингов до 300, а после этого попробовать подобрать оптимальную регуляризацию. Для согласования размерностей может потребоваться дополнительный полносвязный слой между LSTM слоем и softmax классификатором. Не забудьте, что при изменении размера сети или регуляризации меняются оптимальный learning rate, число эпох обучения. Обычно усиление регуляризации требует более длительного обучения и часто уменьшения learning rate. Также не забудьте, что распределения для инициализации весов того или иного слоя должны зависеть от размера слоя.

1. Recurrent Dropout применяется к выходам рекуррентной ячейки на каждом шаге перед подачей их на вход рекуррентной ячейки на следующем шаге. Для одного примера рекомендуется обнулять одни и те же элементы векторов h_t и c_t, т.е. одну и ту же маску dropout на каждом шаге (shared dropout mask / variational LSTM). Для разных примеров из одного батча, а также для одного примера в разных эпохах обучения маска может отличаться.

2. Одинаковую маску dropout на разных шагах можно также применять к входам и выходам LSTM слоев.

3. Weight dropout (Dropconnect), предложенный в модели AWS-LSTM. Перед прямым проходом на очередном батче применяется dropout к матрицам весов рекуррентного слоя.

4. Embeddings dropout целиком зануляет эмбединги для части слов перед прямым проходом.

Вопрос. Приведите графики изменения перплексии на обучающей и валидационной выборки в процессе обучения модели на PTB для ваших экспериментов с различными версиями dropout. Добавьте в таблицу с финальными метриками (из лидерборда) результаты лучшей модели и ее гиперпараметры.

2.8 Text classification with LSTMs

Доработайте написанную рекуррентную сеть так, чтобы ее можно было обучить как языковому моделированию, так и классификации текстов. Для этого потребуется добавить режим, в котором:

1. одна строка батча содержит один пример из датасета; размер батча равен $batch\_size \times max\_len$, где max_len – максимальная длина примеров в данном батче; в словарь добавляется специальный токен <PAD>, который используется для увеличения длины отзывов до max_len.

2. классификатор применяется только к выходу на последнем токене примера (<eos> или предыдущем); учитывайте, что его индекс будет отличаться для разных примеров в батче в зависимости от длины примера. В качестве альтернативы можно брать вектора с последнего токена <PAD>, но этом может давать результаты существенно хуже.

Вопрос. Приведите графики изменения лосс функции и точности на обучающей и валидационной выборке FILIMDB в процессе обучения при различных значениях гиперпараметров (рекомендуем подобрать как минимум learning rate, dropout, число эпох обучения).

2.9 Pre-training LSTMs for text classification

Известно, что предобучение весов класификатора на задаче языкового моделирования положительным образом сказывается на скорости и стабильности обучения классификатора, а также финальном качестве классификации. Попробуйте предобучить сеть в режиме языковой модели на FILIMDB (включая unlabeled часть), а затем дообучить в режиме классификатора. Критическим для скорости обучения языковой модели на FILIMDB является размер словаря, который на порядки больше, чем в PTB (где многие не самые частотные слова уже заменены на `<unk>`). Рекомендуем заменить слова, встречающиеся меньше 3-5 раз, на `<unk>`, более агресивная фильтрация может привести к потере существенных для задаче слов и ухудшению качества классификации. Также рекомендуем попробовать реализовать один из двух способов ускорения softmax: Sampled Softmax или Adaptive Softmax.

Вопрос. Приведите графики изменения лосс функции и точности на обучающей и валидационной выборке FILIMDB в процессе обучения при различных значениях гиперпараметров (рекомендуем подобрать как минимум learning rate, dropout, число эпох обучения). Сравните с лучшим графиком обучения при случайной инициализации весов.

3 Исследовательская часть

Эта часть опциональна, но за нее даются дополнительные баллы.

3.1 QRNN

Попробуйте заменить LSTM слои на QRNN слои, скомбинировав сверточный слой из Pytorch и самописный pooling.

3.2 Average SGD

Попробуйте вместо весов сети на послдне шаге градиентного спуска использовать экспоненциально взвешенное среднее весов на последних нескольких шагах.

3.3 LSTM outputs pooling

В модели UlmFiT предлагается вместо выходов с последнего слова подавать классификатору вектор, полученный в результате конкатенации выхода с последнего слова, среднего выхода со всех слов (mean pooling) и покомпонентного максимума выходов со всех слов (max pooling). Попробуйте оценить, насколько такой вариант улучшает качество классификации.