Обзор трех подходов к решению задачи "Paper Pairwise Sizes": Отжиг, Adam + регуляризация, BNN.

Авторы: Матвеев Артем Сергеевич(206 гр.), Никишкина Евгения Геннадьевна(201 гр.)

Аннотация – Во время проведения соревнования из всех подходов, которые были опробованы в рамках данной задачи, лучше всего себя показал подход с поиском максимальных клик. С ним получилось занять первое место с результатом в 91762. В данной работе описаны еще два подхода, которые были опробованы в данной задаче: Adam + регуляризация и отжиг, которые из-за больших размеров входных данных давали результаты хуже (точнее просто не могли найти хоть какой-то ответ). Также был опробован новый подход, основанный на BNN(Binarized Neural Networks). Он показал себя в несколько раз лучше (если сравнивать по абсолютной ошибке) обычного Adam + регуляризация, но тоже не смог получить ответ по аналогичным причинам.

I. ОТЖИГ

Данный подход реализован следующим алгоритмом:

1. Фиксируем некоторую матрицу $C \in \{0,1\}^{878\times10000}$
2. Делаем какое-то случайное изменение матрицы (т.е. случайно выбираем пару (i,j) и в этой позиции инвертируем элемент).
3. Если суммарная ошибка уменьшилась $\delta E \leq 0$, то фиксируем изменение, иначе оставляем с вероятностью $P(\delta E) = exp(-\delta E/t_i)$
4. $t_{i+1} = T_0/i$, где T_0 - начальная температура (в представленном алгоритме $T_0 = 10$ давало наискорейшее убывание ошибки).

Данный алгоритм запускался на большом числе итераций $\approx 300e6$, но уже после такого количества ошибка практически не изменялась. Причиной этого является тот факт, что в задаче имеется очень большое число переменный $878 * 10000$, поэтому, начиная с некоторого момента, любое улучшение требовало большого числа случайных выборов произвольного элемента, и даже если получалось релаксировать ответ, мы попадали в какой-то "локальный минимум". Наилучший результат, который получается достичь с помощью этого метода: $AbsoluteError = 469690$. Данное решение было реализовано на C++ ввиду большого количества вычислений. Реализация представлена здесь: `https://github.com/matfu-pixel/algorithms_for_paper_pairwise_sizes/blob/main/Annealing.cpp`.

II. ADAM + РЕГУЛЯРИЗАЦИЯ

Теперь рассмотрим подход, основанный на минимизации функционала ошибки с помощью оптимизатора ADAM. Рассмотрим следующую интерпретацию задачи: пусть дана матрица J попарных пересечений множеств. Необходимо найти такую матрицу G, например, принадлежащую $\{0,1\}^{878\times2000}$, что:

$$|(GG^T - J) \circ P| = 0, \text{ где } P = [1]_{N\times N} - I_{N\times N}$$

Для того, чтобы решить эту задачу, будем пытаться минимизировать следующий функционал ошибки:

$$|GG^T \circ P - J| + (|G| - |G-1| - |G-0.5| - 0.5) -> min_G,$$
$$\text{где } P = [1]_{N\times N} - I_{N\times N}$$

Как видно, здесь была добавлена регуляризация для того, чтобы "штрафовать"за веса, отличные от 0 или 1. Ниже изображен график:

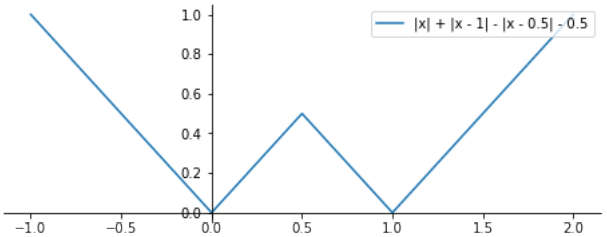


Рис. 1. Регуляризация для того, чтобы "поощрять"веса, близкие к 0 или 1.

Проблема данного подхода заключается в том, что ADAM (также были опробованы SGD, Adamax) не сходятся из-за большого числа переменных. Рещение данной проблемы могло бы быть исправление недифференцируемых функций на гладкие (MSE и гладкую регуляризацию), однако это не решает проблему, результаты получаются аналогичные. Лучший результат, полученный с помощью данного метода: $AbsoluteError = 2000000$. Реализация представлена здесь: `https://github.com/matfu-pixel/algorithms_for_paper_pairwise_sizes/blob/main/ADAM%20%2B%20regularization.ipynb`.

III. BNN

Проблема предыдущего подхода заключалась в том, что мы пытаемся минимизировать функционал ошибки по дискретному пространству бинарных матриц. Методы, основанные на идее градиентного спуска, работают на входных данных из вещественных пространств и для того, чтобы учесть бинарную структуру входных данных, мы и добавляли регуляризацию.

Существует иной подход к минимизации таких функционалов от бинарных переменных,

описанный в следующей статье: https://papers.nips.cc/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf. Здесь рассматриваются нейронные сети, у которых все веса и активации из множества $\{-1,1\}$. Соответственно, после каждой итерации обучения сети, релаксирующий все веса, происходит уменьшение ошибки (важным фактом является то, что веса остаются бинарными).

Основное преимущество данных нейронных сетей заключается в многократной экономии памяти за счет того, что все веса и активации бинарные. Помимо этого, так как теперь вместо тяжелой вещественной арифметике используются быстрые битовые операции, увеличивается скорость прямого прохода.

Авторы статьи утверждают, что данный подход является оптимальным решением для мобильных устройств (то есть устройств с относительно небольшой вычислительной производительностью и небольшим объемом оперативной памяти).

Краткое описание самого алгоритма обучения данной нейронной сети (ниже представлены основные шаги, позволяющие понять отличия прямого и обратного прохода в обычных сетях от бинарных, с подробными деталями можно ознакомиться в представленной статье выше):

Вход: minibatch, выходы a^*, предыдущие веса W, параметр для BatchNorm θ и learning_rate η, L - слоев.

Выход: обновленные веса W, обновленная θ, обновленный learning_rate.

Algorithm 1 Обучения BNN

1: Прямой ход:
2: for k = 1 to L do
3: $W_k{}^b = Binarize(W_k), s_k = a^b{}_{k-1} W_k{}^b$
4: $a_k = BatchNorm(s_k, \theta_k)$
5: Обратный ход (сами градиенты не бинарные):
6: Считаем $g_{a_L} = \frac{\partial c}{\partial a_L}$, где c = Loss
7: for k = L to 1 do
8: if k < L then $g_{a_k} = g_{a_k{}^b} \circ 1_{|a_k| \le 1}$
9: $(g_{s_k}, g_{\theta_k}) = BackBatchNorm(g_{a_k}, s_k, \theta_k)$
10: $g_{a_{k-1}{}^b} = g_{s_k} W_k{}^b, g_{W_k{}^b} = g_{s_k}{}^T a_{k-1}{}^b$
11: Суммируем градиенты:
12: for k = 1 to L do
13: $\theta_k = update(\theta_k, \eta, g_{\theta_k})$
14: $W_k = Clip(update(W_k, \gamma_k \eta, g_{W_k{}^b}), \eta = \lambda \eta$

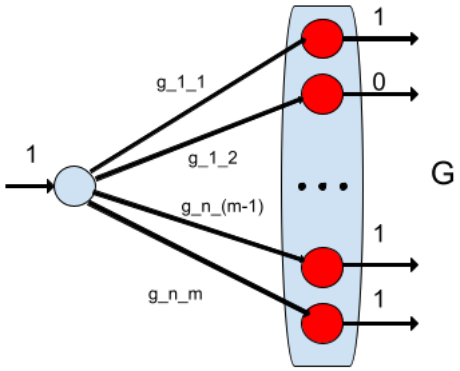
Проблемы, возникшие в ходе применения данного подхода к нашей задаче:

1. Гораздо удобнее использовать уже готовые проверенные реализации таких нейронных сетей, чем совсем отойти от данной концепции: самостоятельно реализовывать оценку градиента, представленную выше, и делать градиентный спуск. Поэтому решение требует переформулирования в терминах нейронных сетей.
2. Все реализации таких нейросетей, представленные в открытом доступе, являются сверточными и решают задачу классификации изображений. В нашем же случае требуется скорее решать задачу регрессии и не нужны никакие слои свертки. Соответственно, нужно переделать одно из таких решений под нашу задачу.

A. Формулировка задачи в терминах BNN

Рассмотрим однослойную нейронную сеть из слоя $BNNLinear(1, 878 \times 10000)$, где в качестве весов будут выступать элементы нашей матрицы G. Тогда, если в случае подачи на вход 1, наша модель только посчитает градиент, согласно алгоритму, описанному в статье, обновит веса и бинаризует их так, что функционал ошибки будет теперь иметь меньшее значение.

То есть процесс обучения нашей модели представляет многократную передачу на вход сети 1.



Покажем, что представляет из себя функция потерь:

Пусть $G \in \{0,1\}^{878 \times 10000}$ текущие веса модели, $J \in N^{878 \times 878}$ данная матрица, $P \in \{0,1\}^{878 \times 878}$ матрица, в которой все 1, кроме диагональных элементов (именно такой объект представляет интерес, учитывая постановку задачи), тогда:

$Loss = |(GG^T - J) \circ P|$, где $\circ$ поэлементное умножения матриц, $|\cdot|$ сумма модулей элементов матрицы

B. Реализация BNN для нашей задачи

За основу была взята следуюая реализация сверточной бинарной нейронной сети: https://github.com/lucamocerino/Binary-Neural-Networks-PyTorch-1.0.

Наша реализация представлена здесь: https://github.com/matfu-pixel/algorithms_for_paper_pairwise_sizes/blob/main/BNN.ipynb

C. Выводы

Данный метод показал хорошие результаты на матрицах небольшого размера (тестировалось на матрицах, размерами 10×10), абсолютная ошибка на них была не больше 3).

Конкретно в нашей задаче, была взята матрица $G \in \{0,1\}^{878 \times 10000}$, проинициализирована так, как в реализации, ссылка на которую представлена выше. С помощью оптимизатора Adam с оптимальными гипперпараметрами удалось достичь результата в $AbsoluteError = 589912$. Аналогично подходу с регуляризацией данный метод столкнулся с проблемой очень большого числа переменных. Можно заметить, что результат получился заметно лучше метода Adam + регуляризация, но немного не догнал отжиг.

IV. ССЫЛКИ

[1] Binary Neural Networks: A Survey. Authors: Haotong Qina, Ruihao Gonga, Xianglong Liu, Xiao Baie,Jingkuan Songc, Nicu Sebed. Доступно: `https://arxiv.org/pdf/2004.03333.pdf`

[2] Binarized Neural Networks, Authors: Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, Yoshua Bengio. Доступно: `https://papers.nips.cc/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf`

[3] Пример реализации сверточных BNN для задачи классификации изображений. Доступно: `https://github.com/lucamocerino/Binary-Neural-Networks-PyTorch-1.0`