ЦЕНТРАЛЬНЫЙ УНИВЕРСИТЕТ

Стохастический градиентный спуск. Адаптивные методы. Оптимизация нейронных сетей

МЕТОДЫ ВЫПУКЛОЙ ОПТИМИЗАЦИИ

НЕДЕЛЯ 14

Даня Меркулов
Пётр Остроухов

Стохастический градиентный спуск. Адаптивные методы

Семинар

Оптимизация для всех! ЦУ

Задача конечной суммы

Задача конечной суммы

Рассмотрим классическую задачу минимизации среднего на конечной выборке:

$$\min_{x \in \mathbb{R}^p} f(x) = \min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

Градиентный спуск действует следующим образом:

$$x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(x) \tag{GD}$$

Задача конечной суммы

Рассмотрим классическую задачу минимизации среднего на конечной выборке:

$$
\min_{x \in \mathbb{R}^p} f(x) = \min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} f_i(x)
$$

Градиентный спуск действует следующим образом:

$$
x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(x) \tag{GD}
$$

- Сходимость с постоянным α или линейным поиском.

Задача конечной суммы

Рассмотрим классическую задачу минимизации среднего на конечной выборке:

$$
\min_{x \in \mathbb{R}^p} f(x) = \min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} f_i(x)
$$

Градиентный спуск действует следующим образом:

$$
x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(x) \tag{GD}
$$

- Сходимость с постоянным α или линейным поиском.
- Стоимость итерации линейна по n. Для ImageNet $n \approx 1.4 \cdot 10^7$, для WikiText $n \approx 10^8$.

Задача конечной суммы

Рассмотрим классическую задачу минимизации среднего на конечной выборке:

$$
\min_{x \in \mathbb{R}^p} f(x) = \min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} f_i(x)
$$

Градиентный спуск действует следующим образом:

$$
x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(x) \tag{GD}
$$

- Сходимость с постоянным α или линейным поиском.
- Стоимость итерации линейна по n. Для ImageNet $n \approx 1.4 \cdot 10^7$, для WikiText $n \approx 10^8$.

Задача конечной суммы

Рассмотрим классическую задачу минимизации среднего на конечной выборке:

$$\min_{x \in \mathbb{R}^p} f(x) = \min_{x \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n f_i(x)$$

Градиентный спуск действует следующим образом:

$$x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^n \nabla f_i(x) \tag{GD}$$

- Сходимость с постоянным α или линейным поиском.
- Стоимость итерации линейна по n. Для ImageNet $n \approx 1.4 \cdot 10^7$, для WikiText $n \approx 10^8$.

Перейдем от вычисления полного градиента к его несмещенной оценке, когда мы случайно выбираем индекс i_k точки на каждой итерации равномерно:

$$x_{k+1} = x_k - \alpha_k \nabla f_{i_k}(x_k) \tag{SGD}$$

При $p(i_k = i) = \frac{1}{n}$ стохастический градиент является несмещенной оценкой градиента, определяемой как:

$$\mathbb{E}[\nabla f_{i_k}(x)] = \sum_{i=1}^n p(i_k = i) \nabla f_i(x) = \sum_{i=1}^n \frac{1}{n} \nabla f_i(x) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(x) = \nabla f(x)$$

Это указывает на то, что математическое ожидание стохастического градиента равно фактическому градиенту $f(x)$.

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

Если ∇f является Липшицевым, то мы имеем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

Если ∇f является Липшицевым, то мы имеем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- Стохастический метод имеет низкую стоимость итерации, но медленную скорость сходимости.

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

Если ∇f является Липшицевым, то мы имеем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- Стохастический метод имеет низкую стоимость итерации, но медленную скорость сходимости.
    - Сублинейная скорость даже в сильно выпуклом случае.

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

Если ∇f является Липшицевым, то мы имеем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- Стохастический метод имеет низкую стоимость итерации, но медленную скорость сходимости.
    - Сублинейная скорость даже в сильно выпуклом случае.
    - Оценки неулучшаемы при стандартных предположениях.

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

Если ∇f является Липшицевым, то мы имеем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- Стохастический метод имеет низкую стоимость итерации, но медленную скорость сходимости.
  - Сублинейная скорость даже в сильно выпуклом случае.
  - Оценки неулучшаемы при стандартных предположениях.
  - Оракул возвращает несмещенную аппроксимацию градиента с ограниченной дисперсией.

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

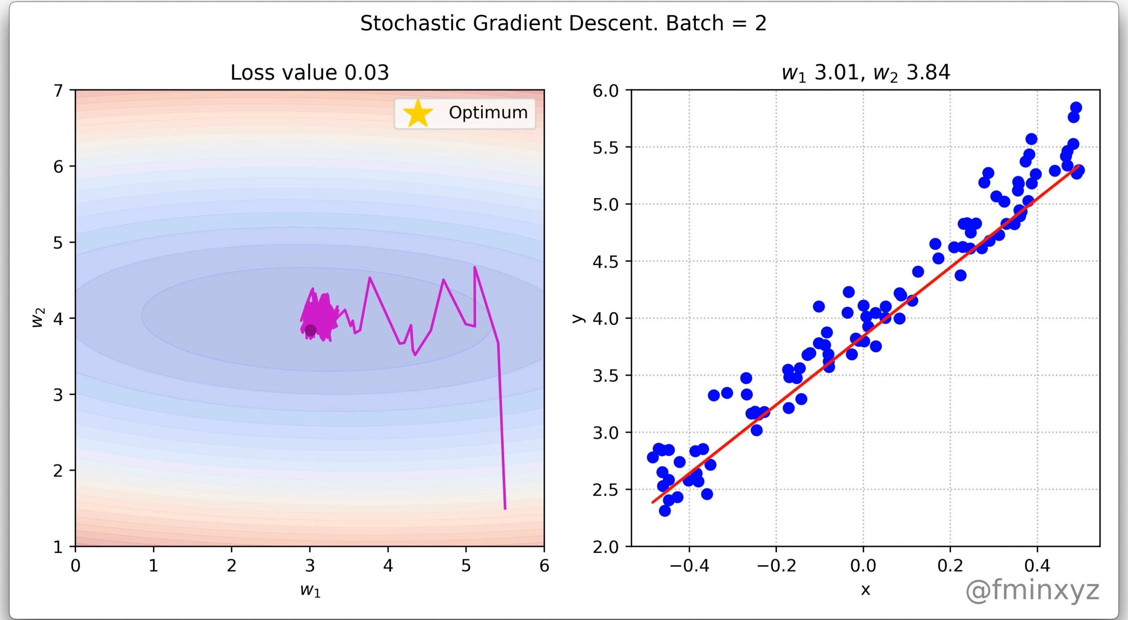
Если ∇f является Липшицевым, то мы имеем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- Стохастический метод имеет низкую стоимость итерации, но медленную скорость сходимости.
    - Сублинейная скорость даже в сильно выпуклом случае.
    - Оценки неулучшаемы при стандартных предположениях.
    - Оракул возвращает несмещенную аппроксимацию градиента с ограниченной дисперсией.
- Momentum и квазиньютоновские методы не улучшают скорость сходимости в стохастическом случае. Могут улучшить только константы (узким местом является дисперсия, а не число обусловленности).

Стохастический градиентный спуск (SGD)

Типичное поведение



Вычислительные эксперименты

Вычислительные эксперименты

Визуализация SGD.

Посмотрим на вычислительные эксперименты для SGD.

Адаптивность или масштабирование

Adagrad (Duchi, Hazan, and Singer 2010)

Очень популярный адаптивный метод. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$, и обновление для $j = 1, \dots, p$:

$$
v_j^{(k)} = v_j^{k-1} + (g_j^{(k)})^2
$$

$$
x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}
$$

Adagrad (Duchi, Hazan, and Singer 2010)

Очень популярный адаптивный метод. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$, и обновление для $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{k-1} + (g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Заметки:

- AdaGrad не требует настройки скорости обучения: $\alpha > 0$ — фиксированная константа, и скорость обучения естественным образом уменьшается с итерациями.

Adagrad (Duchi, Hazan, and Singer 2010)

Очень популярный адаптивный метод. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$, и обновление для $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{k-1} + (g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Заметки:

- AdaGrad не требует настройки скорости обучения: $\alpha > 0$ — фиксированная константа, и скорость обучения естественным образом уменьшается с итерациями.
- Скорость обучения редких информативных признаков уменьшается медленно.

Adagrad (Duchi, Hazan, and Singer 2010)

Очень популярный адаптивный метод. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$, и обновление для $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{k-1} + (g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Заметки:

- AdaGrad не требует настройки скорости обучения: $\alpha > 0$ — фиксированная константа, и скорость обучения естественным образом уменьшается с итерациями.
- Скорость обучения редких информативных признаков уменьшается медленно.
- Может значительно превосходить SGD в разреженных задачах.

Adagrad (Duchi, Hazan, and Singer 2010)

Очень популярный адаптивный метод. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$, и обновление для $j = 1, \ldots, p$:

$$
v_j^{(k)} = v_j^{k-1} + (g_j^{(k)})^2
$$

$$
x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}
$$

Заметки:

- AdaGrad не требует настройки скорости обучения: $\alpha > 0$ — фиксированная константа, и скорость обучения естественным образом уменьшается с итерациями.
- Скорость обучения редких информативных признаков уменьшается медленно.
- Может значительно превосходить SGD в разреженных задачах.
- Основным недостатком является монотонное накопление градиентов в знаменателе. AdaDelta, Adam, AMSGrad и др. улучшают это, популярны при обучении глубоких нейронных сетей.

Adagrad (Duchi, Hazan, and Singer 2010)

Очень популярный адаптивный метод. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$, и обновление для $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{k-1} + (g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Заметки:

- AdaGrad не требует настройки скорости обучения: $\alpha > 0$ — фиксированная константа, и скорость обучения естественным образом уменьшается с итерациями.
- Скорость обучения редких информативных признаков уменьшается медленно.
- Может значительно превосходить SGD в разреженных задачах.
- Основным недостатком является монотонное накопление градиентов в знаменателе. AdaDelta, Adam, AMSGrad и др. улучшают это, популярны при обучении глубоких нейронных сетей.
- Константа ϵ обычно устанавливается равной 10^{-6}, чтобы избежать деления на ноль или слишком больших шагов.

RMSProp (Tieleman and Hinton, 2012)

Улучшение AdaGrad, решающее проблему агрессивного, монотонно убывающего темпа обучения. Использует скользящее среднее квадратов градиентов для настройки скорости обучения для каждого веса. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ и правило обновления для $j = 1, \dots, p$:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)(g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

RMSProp (Tieleman and Hinton, 2012)

Улучшение AdaGrad, решающее проблему агрессивного, монотонно убывающего темпа обучения. Использует скользящее среднее квадратов градиентов для настройки скорости обучения для каждого веса. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ и правило обновления для $j = 1, \dots, p$:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)(g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Заметки:

- RMSProp делит скорость обучения для веса на скользящее среднее величин последних градиентов для этого веса.

RMSProp (Tieleman and Hinton, 2012)

Улучшение AdaGrad, решающее проблему агрессивного, монотонно убывающего темпа обучения. Использует скользящее среднее квадратов градиентов для настройки скорости обучения для каждого веса. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ и правило обновления для $j = 1, \dots, p$:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)(g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Заметки:

- RMSProp делит скорость обучения для веса на скользящее среднее величин последних градиентов для этого веса.
- Позволяет более тонко настраивать скорость обучения, чем AdaGrad, что делает его подходящим для неизотропных задач.

RMSProp (Tieleman and Hinton, 2012)

Улучшение AdaGrad, решающее проблему агрессивного, монотонно убывающего темпа обучения. Использует скользящее среднее квадратов градиентов для настройки скорости обучения для каждого веса. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ и правило обновления для $j = 1, \dots, p$:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)(g_j^{(k)})^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \epsilon}}$$

Заметки:

- RMSProp делит скорость обучения для веса на скользящее среднее величин последних градиентов для этого веса.
- Позволяет более тонко настраивать скорость обучения, чем AdaGrad, что делает его подходящим для неизотропных задач.
- Часто используется при обучении нейронных сетей, особенно рекуррентных.

Adadelta (Zeiler, 2012)

Расширение RMSProp, направленное на уменьшение зависимости от вручную задаваемой глобальной скорости обучения. Вместо накопления всех прошлых квадратов градиентов, Adadelta ограничивает окно накопленных прошлых градиентов некоторым фиксированным размером w. Механизм обновления не требует скорости обучения α:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)(g_j^{(k)})^2$$

$$\tilde{g}_j^{(k)} = \frac{\sqrt{\Delta x_j^{(k-1)} + \epsilon}}{\sqrt{v_j^{(k)} + \epsilon}} g_j^{(k)}$$

$$x_j^{(k)} = x_j^{(k-1)} - \tilde{g}_j^{(k)}$$

$$\Delta x_j^{(k)} = \rho \Delta x_j^{(k-1)} + (1-\rho)(\tilde{g}_j^{(k)})^2$$

Adadelta (Zeiler, 2012)

Расширение RMSProp, направленное на уменьшение зависимости от вручную задаваемой глобальной скорости обучения. Вместо накопления всех прошлых квадратов градиентов, Adadelta ограничивает окно накопленных прошлых градиентов некоторым фиксированным размером w. Механизм обновления не требует скорости обучения α:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)(g_j^{(k)})^2$$

$$\tilde{g}_j^{(k)} = \frac{\sqrt{\Delta x_j^{(k-1)} + \epsilon}}{\sqrt{v_j^{(k)} + \epsilon}} g_j^{(k)}$$

$$x_j^{(k)} = x_j^{(k-1)} - \tilde{g}_j^{(k)}$$

$$\Delta x_j^{(k)} = \rho \Delta x_j^{(k-1)} + (1-\rho)(\tilde{g}_j^{(k)})^2$$

Заметки:

- Adadelta адаптирует скорость обучения на основе скользящего окна обновлений градиента, а не накапливает все прошлые градиенты. Таким образом, настроенные скорости обучения более устойчивы к изменениям динамики модели.

Adadelta (Zeiler, 2012)

Расширение RMSProp, направленное на уменьшение зависимости от вручную задаваемой глобальной скорости обучения. Вместо накопления всех прошлых квадратов градиентов, Adadelta ограничивает окно накопленных прошлых градиентов некоторым фиксированным размером w. Механизм обновления не требует скорости обучения α:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)(g_j^{(k)})^2$$

$$\tilde{g}_j^{(k)} = \frac{\sqrt{\Delta x_j^{(k-1)} + \epsilon}}{\sqrt{v_j^{(k)} + \epsilon}} g_j^{(k)}$$

$$x_j^{(k)} = x_j^{(k-1)} - \tilde{g}_j^{(k)}$$

$$\Delta x_j^{(k)} = \rho \Delta x_j^{(k-1)} + (1-\rho)(\tilde{g}_j^{(k)})^2$$

Заметки:

- Adadelta адаптирует скорость обучения на основе скользящего окна обновлений градиента, а не накапливает все прошлые градиенты. Таким образом, настроенные скорости обучения более устойчивы к изменениям динамики модели.
- Метод не требует установки начальной скорости обучения, что упрощает настройку.

Adadelta (Zeiler, 2012)

Расширение RMSProp, направленное на уменьшение зависимости от вручную задаваемой глобальной скорости обучения. Вместо накопления всех прошлых квадратов градиентов, Adadelta ограничивает окно накопленных прошлых градиентов некоторым фиксированным размером w. Механизм обновления не требует скорости обучения α:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)(g_j^{(k)})^2$$

$$\tilde{g}_j^{(k)} = \frac{\sqrt{\Delta x_j^{(k-1)} + \epsilon}}{\sqrt{v_j^{(k)} + \epsilon}} g_j^{(k)}$$

$$x_j^{(k)} = x_j^{(k-1)} - \tilde{g}_j^{(k)}$$

$$\Delta x_j^{(k)} = \rho \Delta x_j^{(k-1)} + (1-\rho)(\tilde{g}_j^{(k)})^2$$

Заметки:

- Adadelta адаптирует скорость обучения на основе скользящего окна обновлений градиента, а не накапливает все прошлые градиенты. Таким образом, настроенные скорости обучения более устойчивы к изменениям динамики модели.
- Метод не требует установки начальной скорости обучения, что упрощает настройку.
- Часто используется в глубоком обучении, где масштабы параметров значительно различаются по слоям.

Adam (Kingma and Ba, 2014) [1 2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

EMA:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1-\beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1-\beta_2)\left(g_j^{(k)}\right)^2$$

Исправление смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1-\beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1-\beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Adam (Kingma and Ba, 2014) [1 2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

EMA:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1-\beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1-\beta_2) \left(g_j^{(k)}\right)^2$$

Исправление смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1-\beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1-\beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Заметки:

- Он исправляет смещение к нулю в начальные моменты, наблюдаемое в других методах, таких как RMSProp, делая оценки более точными.

Adam (Kingma and Ba, 2014) [1 2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

EMA:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1-\beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1-\beta_2)\left(g_j^{(k)}\right)^2$$

Исправление смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1-\beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1-\beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Заметки:

- Он исправляет смещение к нулю в начальные моменты, наблюдаемое в других методах, таких как RMSProp, делая оценки более точными.
- Одна из самых цитируемых научных статей в мире

Adam (Kingma and Ba, 2014) [1 2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

EMA:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1-\beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1-\beta_2) \left(g_j^{(k)}\right)^2$$

Исправление смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1-\beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1-\beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Заметки:

- Он исправляет смещение к нулю в начальные моменты, наблюдаемое в других методах, таких как RMSProp, делая оценки более точными.
- Одна из самых цитируемых научных статей в мире
- В 2018-2019 годах были опубликованы статьи, указывающие на ошибки в оригинальной статье

Adam (Kingma and Ba, 2014) [1,2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

EMA:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1-\beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1-\beta_2) \left(g_j^{(k)}\right)^2$$

Исправление смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1-\beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1-\beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Заметки:

- Он исправляет смещение к нулю в начальные моменты, наблюдаемое в других методах, таких как RMSProp, делая оценки более точными.
- Одна из самых цитируемых научных статей в мире
- В 2018-2019 годах были опубликованы статьи, указывающие на ошибки в оригинальной статье
- Не сходится на некоторых простых задачах (даже выпуклых)

Adam (Kingma and Ba, 2014) [1 2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

EMA:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1-\beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1-\beta_2) \left(g_j^{(k)}\right)^2$$

Исправление смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1-\beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1-\beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon}$$

Заметки:

- Он исправляет смещение к нулю в начальные моменты, наблюдаемое в других методах, таких как RMSProp, делая оценки более точными.
- Одна из самых цитируемых научных статей в мире
- В 2018-2019 годах были опубликованы статьи, указывающие на ошибки в оригинальной статье
- Не сходится на некоторых простых задачах (даже выпуклых)
- Каким-то образом работает исключительно хорошо для некоторых сложных задач

Adam (Kingma and Ba, 2014) [1] [2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

EMA:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1-\beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1-\beta_2)\left(g_j^{(k)}\right)^2$$

Исправление смещения:

$$\hat{m}_j = \frac{m_j^{(k)}}{1-\beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1-\beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j + \epsilon}}$$

Заметки:

- Он исправляет смещение к нулю в начальные моменты, наблюдаемое в других методах, таких как RMSProp, делая оценки более точными.
- Одна из самых цитируемых научных статей в мире
- В 2018-2019 годах были опубликованы статьи, указывающие на ошибки в оригинальной статье
- Не сходится на некоторых простых задачах (даже выпуклых)
- Каким-то образом работает исключительно хорошо для некоторых сложных задач
- Работает намного лучше для языковых моделей, чем для задач компьютерного зрения — почему?

[1] Adam: A Method for Stochastic Optimization

[2] On the Convergence of Adam and Beyond

AdamW (Loshchilov & Hutter, 2017)

Решает распространенную проблему с ℓ_2 регуляризацией в адаптивных оптимизаторах, таких как Adam. Стандартная ℓ_2 регуляризация добавляет $\lambda\|x\|^2$ к функции потерь, что приводит к слагаемому градиента λx. В Adam это слагаемое масштабируется адаптивной скоростью обучения $\left(\sqrt{\hat{v}_j} + \epsilon\right)$, связывая затухание весов с величинами градиента.

AdamW отделяет затухание весов от шага адаптации градиента.

Правило обновления:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1 - \beta_2) (g_j^{(k)})^2$$

$$\hat{m}_j = \frac{m_j^{(k)}}{1 - \beta_1^k}, \quad \hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k}$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \left(\frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon} + \lambda x_j^{(k-1)} \right)$$

AdamW (Loshchilov & Hutter, 2017)

Решает распространенную проблему с ℓ_2 регуляризацией в адаптивных оптимизаторах, таких как Adam. Стандартная ℓ_2 регуляризация добавляет $\lambda\|x\|^2$ к функции потерь, что приводит к слагаемому градиента λx. В Adam это слагаемое масштабируется адаптивной скоростью обучения $\left(\sqrt{\hat{v}_j} + \epsilon\right)$, связывая затухание весов с величинами градиента.

AdamW отделяет затухание весов от шага адаптации градиента.

Правило обновления:

$$
m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)}
$$

$$
v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1 - \beta_2)(g_j^{(k)})^2
$$

$$
\hat{m}_j = \frac{m_j^{(k)}}{1 - \beta_1^k}, \quad \hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k}
$$

$$
x_j^{(k)} = x_j^{(k-1)} - \alpha \left(\frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon} + \lambda x_j^{(k-1)} \right)
$$

Заметки:

- Слагаемое затухания весов $\lambda x_j^{(k-1)}$ добавляется *после* шага адаптивного градиента.

AdamW (Loshchilov & Hutter, 2017)

Решает распространенную проблему с ℓ_2 регуляризацией в адаптивных оптимизаторах, таких как Adam. Стандартная ℓ_2 регуляризация добавляет $\lambda\|x\|^2$ к функции потерь, что приводит к слагаемому градиента λx. В Adam это слагаемое масштабируется адаптивной скоростью обучения $\left(\sqrt{\hat{v}_j} + \epsilon\right)$, связывая затухание весов с величинами градиента.

AdamW отделяет затухание весов от шага адаптации градиента.

Правило обновления:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1 - \beta_2)(g_j^{(k)})^2$$

$$\hat{m}_j = \frac{m_j^{(k)}}{1 - \beta_1^k}, \quad \hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k}$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \left(\frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon} + \lambda x_j^{(k-1)} \right)$$

Заметки:

- Слагаемое затухания весов $\lambda x_j^{(k-1)}$ добавляется *после* шага адаптивного градиента.
- Широко применяется при обучении трансформеров и других больших моделей. Выбор по умолчанию для huggingface trainer.