Стандартные оптимизаторы

Пусть необходимо минимизировать некий функционал $\mathscr{L}$ по "весам" w:

$$\mathscr{L}(w) \to \min_{w}, \text{ где } w = (w_1, w_2, \dots, w_M)^T$$

Рассмотрим несколько наиболее распространенных итеративных методов для решения поставленной оптимизационной задачи.

Примем, что известно значение w_t в момент времени t, тогда необходимо вычислить значение w_{t+1} в следующий момент времени.

Градиентный спуск

Итерационный метод, в простейшей реализации которого шаг выполняется в сторону антиградиента функции потерь, вычисленного по всему датасету.

$$w_{t+1} = w_t - \alpha \nabla_w \mathscr{L}(w_t)$$

Во избежание массивных вычислений на больших наборах данных может быть использована модификация градиентного спуска — *стохастический градиентный спуск*, суть которого заключается в том, что на каждой итерации градиент вычисляется лишь по одному обучающему примеру.

$$w_{t+1} = w_t - \alpha \nabla_w \mathscr{L}(w_t, x^i, y^i)$$

Наиболее оптимальным является метод, совмещающий в себе идеи обоих вышеописанных методов — *mini-batch градиентый спуск.*

$$w_{t+1} = w_t - \alpha \nabla_w \mathscr{L}(w_t,\, x^{(i:i+n)},\, y^{(i:i+n)})$$

Градиентный спуск с импульсом

Для борьбы с неэффективностью метода градиентного спуска при оптимизации функционалов, сильно "растянутых" вдоль одной из компонент w, может быть использован *градиентный спуск с импульсом.* Основной идеей является использование "скорости" v для борьбы с почти противоположно направленными шагами.

$$\begin{cases} v_{t+1} = \beta v_t - \alpha \nabla_w \mathscr{L}(w) \\ w_{t+1} = w_t - v_{t+1} \end{cases}$$

Экспоненциальное слользящее среднее

Ту же самую идею, что и в методе градиентного спуска с импульсом можно реализовать с помощью *экспоненциального скользящего среднего* (EMA).

$$w_{t+1} = w_t - \alpha \cdot EMA\big(\mathscr{L}(w_t)\big),$$

$$\text{где } EMA_\beta\big(f\big)^t = (1-\beta) f^t + \beta \cdot EMA_\beta\big(f\big)^{t-1}$$

RProp

Оптимизационный метод с адаптивным шагом для каждого параметра, основная мысль которого заключается в том, что мы будем увеличивать размер шага, если последние две итерации мы двигались в одном направлении и, напротив, уменьшать размер шага, если направление сменилось.

$$w_i^{t+1} = w_i^t - \alpha_i^t \cdot sign\big(\nabla_w \mathscr{L}_i((w^t)\big)$$

$$\begin{cases} \alpha_i^{t+1} = 1.2 \cdot \alpha_i^t, \text{ если } \ sign\big(\nabla_w \mathscr{L}_i(w^t) \cdot \nabla_w \mathscr{L}_i(w^{t-1})\big) > 0 \\ \alpha_i^{t+1} = 0.6 \cdot \alpha_i^t, \text{ если } \ sign\big(\nabla_w \mathscr{L}_i(w^t) \cdot \nabla_w \mathscr{L}_i(w^{t-1})\big) < 0 \end{cases}$$

RMSProp

Метод *RProp* показывает неудовлетворительные результаты при работе с батчами, к тому же он слишком уж эмпирический. Продолжением алгоритмов с адаптивным шагом является $RMSProp$, в котором мы фактически "штрафуем" шаги за большие изменения и, наоборот, увеличиваем шаг, если изменения малы.

$$w_{t+1} = w_t - \alpha \frac{\nabla_w \mathscr{L}(w_t)}{\sqrt{EMA_\gamma\big(\nabla_w \mathscr{L}^2(w_t)\big)}}$$

Adam

Пользуясь теми же соображениями, что при выводе *градиентного спуска с импульсом*, используем экспоненциальное скользящее среднее в числителе выражения для обновления параметров в алгоритме *RProp* и получим новый алгоритм — *Adam*.

$$w_{t+1} = w_t - \alpha \frac{EMA_{\gamma_1}\big(\nabla_w \mathscr{L}(w_t)\big)}{\sqrt{EMA_{\gamma_2}\big(\nabla_w \mathscr{L}^2(w_t)\big)} + \varepsilon}$$