KARPOV.COURSES >>>
КОНСПЕКТ

Конспект > 2 урок

>Оптимизация нейронных сетей. Метод обратного распространения ошибки

>Градиентный спуск

В глубинном обучении нейросети состоят из слоёв и обучаются с помощью градиентного спуска, а градиенты для него вычисляются с помощью метода обратного распространения ошибки.

Основная идея градиентного спуска заключается в нахождении локального минимума или максимума функции с помощью движения вдоль градиента. Градиент — это вектор, направленный в сторону наискорейшего роста функции, а, поскольку нужно минимизировать функцию потерь, то необходимо двигаться по нему но с минусом. Процесс оптимизации итеративный:

1. Считаем градиент в случайной точке w_t, в окрестности которой функция $f(w)$ должна быть определена и дифференцируема. Обычно это точка 0.

2. Считаем градиент в новой точке и опять сдвигаемся по нему. Таким образом чередуется обновление параметров и вычисление градиента

$$w_t = w_{(t-1)} - \eta \nabla Q(w_{t-1})$$

3. Ведем оптимизацию до какого-то критерия останова

Критерий останова выбирается самостоятельно и обозначается буквой ξ.
Варианты нахождения критерия останова:

1. Определенное минимальное значение производной. Если достигаем выбранного значения производной, то останавливаем спуск. $||\nabla Q(w_t)|| \leq \xi$
2. Маленькая длина шага. Если шаг градиента становится маленьким, то останавливаем спуск. $|w_t - w_{t-1}| \leq \xi$
3. В глубинном обучении оптимизация ведётся до тех пор, пока ошибка на отложенной выборке продолжает падать.

Поскольку функция потерь состоит из суммы некоторых функций потерь на отдельных объектах обучающей выборки, то и градиент этой функции потерь будет состоять из суммы градиентов по отдельным объектам это называется полный градиентый спуск,он вычисляет среднее значение для всего набора данных. Т.е для каждого шага оптимизации нужно будет посчитать все градиенты и поделить на количество объектов — это очень долго и дорого.

Взамен этого можно оценить градиент по одному объекту или по подмножеству.

Оптимизации нейронных сетей и ведётся следующим образом: На каждом шаге выбираются некоторые случайные объекты из обучающей выборки. Для них считаются градиенты, затем, суммируются и усредняются. Выбранные объекты называются batch, а размер - batch size

>Методы оптимизации

У градиентного спуска есть несколько модификаций, которые помогают решить его проблемы.

>Momentum

Бывает, что направление градиента от шага к шагу может сильно меняться, это приводит к лишнему шуму. Чтобы этого избежать можно усреднить векторы градиентов с предыдущих шагов. Т.е вести историю градиентов и их усреднять. Если для какого-то признака знак постоянно меняется, то усреднённая сумма будет стремиться к нулю, а значит, градиент двигаться не будет. Если же знак постоянный, то шаг по координате будет расти.

$h_0 = 0$

$h_k = ah_{k-1} + \eta_k \triangledown Q(w_{k-1})i^2$

$w_k = w_{k-1} - h_k$

>AdaGrad

У градиентного спуска есть большая чувствительность к размеру шага, т.е если шаг большой, есть риск перешагнуть через точку оптимума, а если шаг маленький, то понадобится больше итераций и велик риск вовсе не добраться до точки оптимума. Метод AdaGrad предлагает ввести адаптивную для каждой координаты длину шага. Т.е шаг будет тем меньше, чем более длинные шаги были раньше.

$G_{ki} = G_{k-1,i} + (\triangledown Q(w_{t-1}))_i^2$

$w_j^k = w_t^{k-1} - \frac{\eta_t}{\sqrt{G_{ki} + \epsilon}}(\triangledown Q(w_{t-1}))_i$

ϵ сглаживающий параметр, необходимый, чтобы избежать численной неустойчивости и не делить на ноль.

Этот метод хорошо подходит для задач, в которых у объектов много нулей в признаках. Т.е для признаков, у которых ненулевые значения встречаются часто, будут большие шаги

>Adam

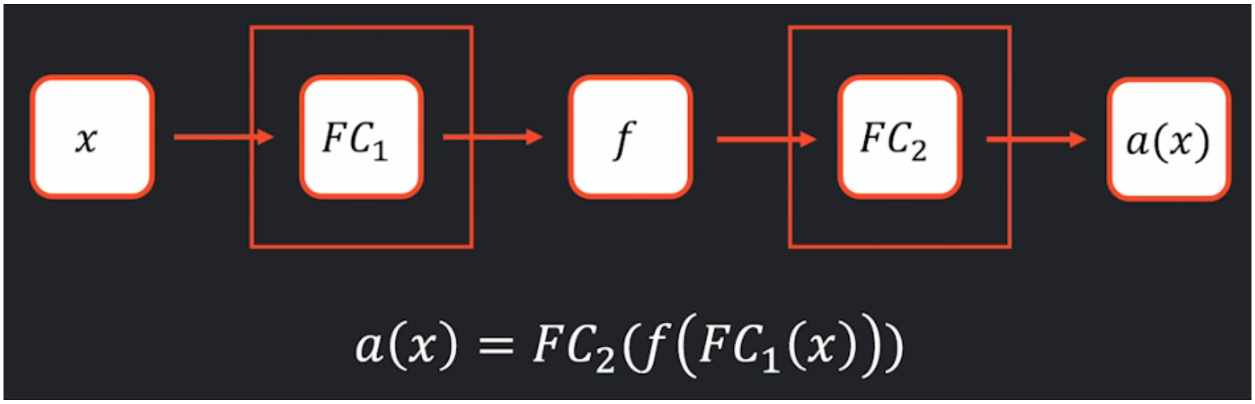
Adam— adaptive moment estimation, ещё один оптимизационный алгоритм. Он объединяет в себе идеи предыдущих методов: адаптивность шага и усреднение градиентов. Метод Adam самый популярный метод обучения нейронных сетей в глубинном обучении в силу своей неприхотливости в использовании.

$$
\begin{aligned}
&\textbf{input} : \gamma \text{ (lr)}, \beta_1, \beta_2 \text{ (betas)}, \theta_0 \text{ (params)}, f(\theta) \text{ (objective)} \\
&\hspace{13mm} \lambda \text{ (weight decay)}, \: \textit{amsgrad}, \: \textit{maximize} \\
&\textbf{initialize} : m_0 \leftarrow 0 \text{ (first moment)}, v_0 \leftarrow 0 \text{ (second moment)}, \: \widehat{v_0}^{max} \leftarrow 0 \\[-1.ex]
&\rule{110mm}{0.4pt} \\
&\textbf{for} \: t=1 \: \textbf{to} \: \ldots \: \textbf{do} \\
&\hspace{5mm}\textbf{if} \: \textit{maximize}: \\
&\hspace{10mm} g_t \leftarrow -\nabla_{\theta} f_t (\theta_{t-1}) \\
&\hspace{5mm}\textbf{else} \\
&\hspace{10mm} g_t \leftarrow \nabla_{\theta} f_t (\theta_{t-1}) \\
&\hspace{5mm}\textbf{if} \: \lambda \neq 0 \\
&\hspace{10mm} g_t \leftarrow g_t + \lambda \theta_{t-1} \\
&\hspace{5mm} m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t \\
&\hspace{5mm} v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g^2_t \\
&\hspace{5mm} \widehat{m_t} \leftarrow m_t/\big(1-\beta_1^t \big) \\
&\hspace{5mm} \widehat{v_t} \leftarrow v_t/\big(1-\beta_2^t \big) \\
&\hspace{5mm}\textbf{if} \: \textit{amsgrad} \\
&\hspace{10mm} \widehat{v_t}^{max} \leftarrow \mathrm{max}(\widehat{v_t}^{max}, \widehat{v_t}) \\
&\hspace{10mm} \theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m_t}/\big(\sqrt{\widehat{v_t}^{max}} + \epsilon \big) \\
&\hspace{5mm}\textbf{else} \\
&\hspace{10mm} \theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m_t}/\big(\sqrt{\widehat{v_t}} + \epsilon \big) \\
&\rule{110mm}{0.4pt} \\[-1.ex]
&\textbf{return} \: \theta_t
\end{aligned}
$$

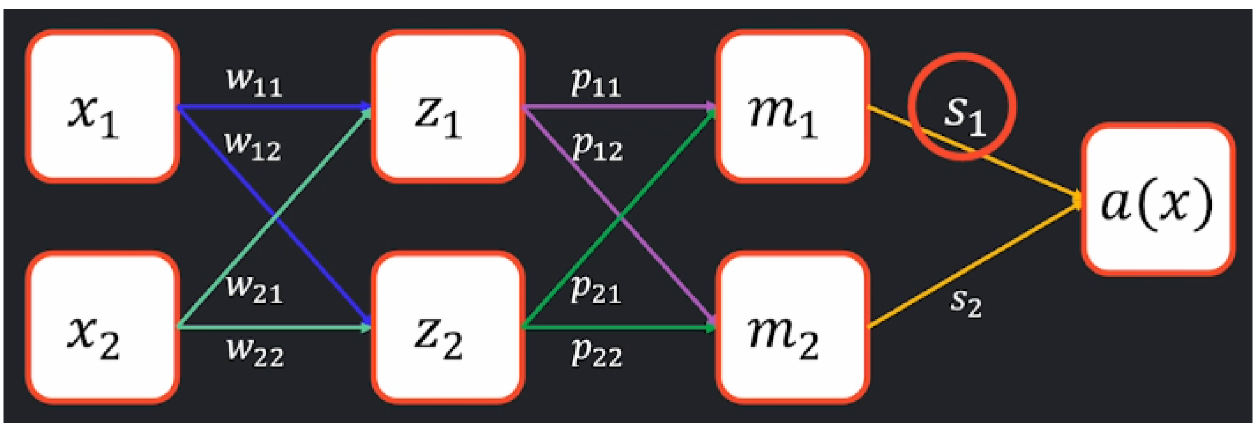
[Подробнее о методе Adam](#)

>Метод обратного распространения ошибки

Как считать градиенты в нейронных сетях? Параметры в нейронных сетях содержатся в слоях, каждый слой дифференцируем и содержит какое-то количество параметров, эти параметры обучаемы. И чтобы обучать нейронную сеть с помощью градиентного спуска, для начала нужно посчитать градиент функции потерь для каждого параметра.



Давайте на примере попробуем ощутить масштаб производных выхода нейронной сети по параметрам. Поменяем какие-то конкретные параметры и посмотрим что получится (параметры изображены в виде стрелок, а промежуточные состояния в виде квадратов):



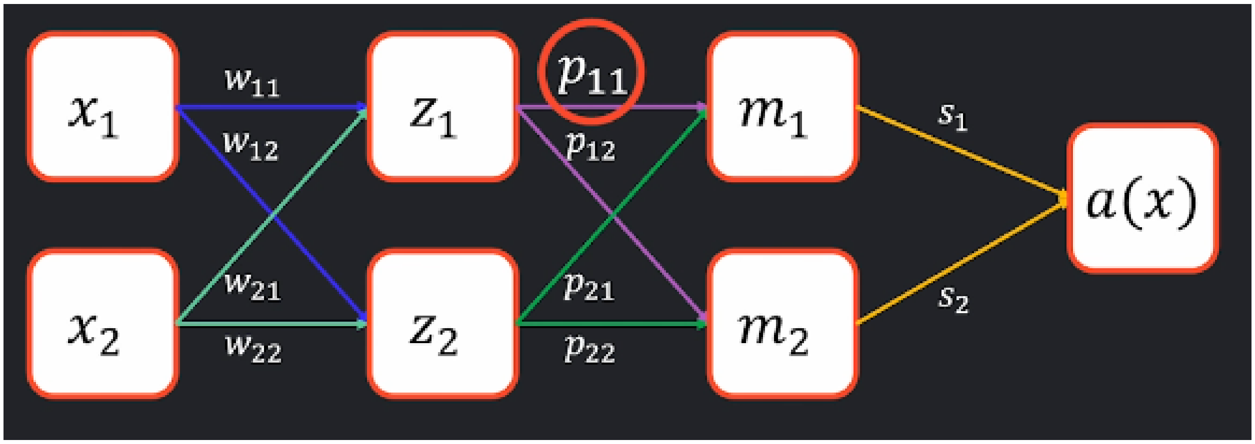
Запишем выход модели a в аналитическом виде. Как сумму двух функций от входных данных $m_1(x)$ и $m_2(x)$. Суммируем с коэффициентами s_1 и s_2.

$a(x) = s_1 m_1(x) + s_2 m_2(x)$

$\frac{\partial a}{\partial s_1} = m_1(x)$ $\frac{\partial a}{\partial m_1} = s_1$

Частная производная выхода модели по коэффициенту s_1 это $m_1(s)$. И чем больше $m_1(x)$, тем сильнее влияет изменение веса s_1 на итоговый выход.

Второй слой, вес p_{11}:



Все p_{11} влияет на выход нейронной сети менее тривиальным образом:

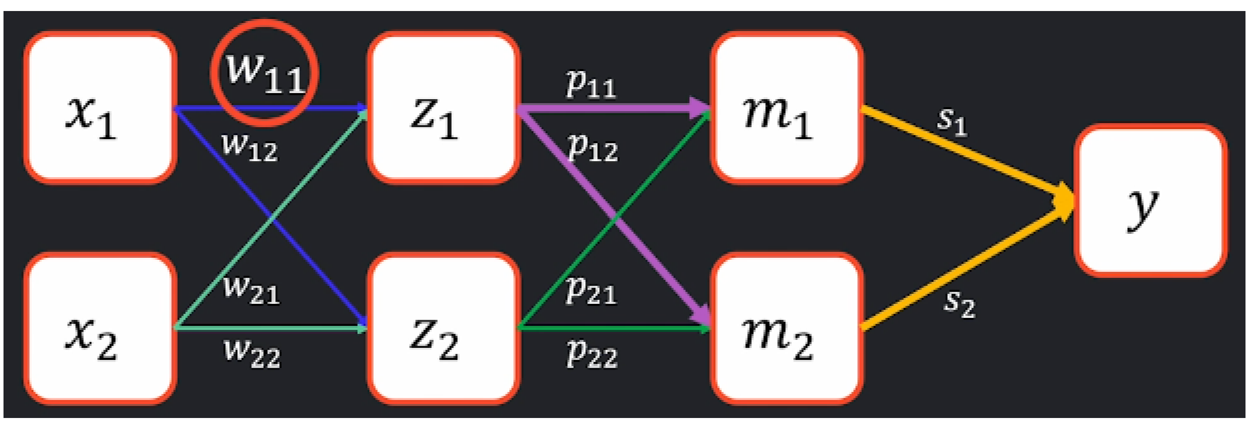
$a(x) = s_1 f(p_{11} z_1 + p_{21} z_2) + s_2 m_2(x)$

Производная выхода модели по параметру p_{11} будет равна производной выхода модели по m_1 умноженная на производную m_1 по p_{11}

$\frac{\partial a}{\partial p_{11}} = \frac{\partial a}{\partial m_1} \frac{\partial m_1}{\partial p_{11}}$

Отметим, что для того чтобы посчитать производную для параметра на каком-то слое нам понадобилась производная на более позднем слое.

Производная по параметру w_{11}:



В данном случае аналитическая формула будет уж через чур большой. Влияет ли изменение w_{11} на выход нейронной сети? Т.е будет ли меняться влияние параметра p_{11} на то, как влияет w_{11} на выход? Можно записать все пути, через которые w_{11} идёт к выходу модели, и вдоль этого пути посчитать частные производные и перемножить:

$$\frac{\partial a}{\partial w_{11}} = \frac{\partial a}{\partial m_1}\frac{\partial m_1}{\partial z_1}\frac{\partial z_1}{\partial w_{11}} + \frac{\partial a}{\partial m_2}\frac{\partial m_2}{\partial z_1}\frac{\partial z_1}{\partial w_{11}}$$

Заметим, что в вычислении производных на конкретном слое нейросети участвуют все производные всех более поздних слоев

Метод обратного распространения ошибки (Backpropagation) - метод вычисления производной сложной функции, примененной в нейронных сетях.

Граф вычислений - это последовательность каких-либо математических операций, которые идут друг за другом.

Суть метода заключается в том, чтобы идти по графу вычислений в обратную сторону (т.е начиная с конца нейросети) и на каждом слое считать производные

backward pass - Вычисление производных и градиентов функции потерь по параметрам нейронной сети

forward pass - Вычисление промежуточных состояний и ответа нейронной сети