Московский государственный университет имени
М.В. Ломоносова

Факультет вычислительной математики и
кибернетики

Реферат

Метод натурального градиента в машинном обучении

Исполнитель:
Виктор Януш

Москва
2016 год

Содержание

1 Введение

Многие методы машинного обучения подразумевают необходимость оптимизации некоторых функционалов, зависящих, например, от параметров модели. Часто такие подзадачи решаются методами из градиентного спуска — они просты в использовании и реализации.

Стандартный метод градиентного спуска является наиболее простым из этого класса методов, но он не всегда сходится быстро и зависит от того пространства, в котором находятся параметры оптимизируемой функции. В основном, параметры моделей машинного обучения находятся в пространствах, структура которых совсем не похожа на структуру обычного евклидова пространства. В них градиентный спуск работает не очень хорошо.

В этом реферате рассматривается метод натурального градиента, который иногда позволяет обойти эти ограничения, используя внутреннюю риманову структуру пространства параметров.

2 Градиентный спуск

Предположим, что перед нами стоит задача минимизировать некоторую функцию $f(x) : \mathbb{R}^n \to \mathbb{R}$.
Рассмотрим линейное приближение $f(x)$ в точке x_0 при достаточно малом $||x - x_0||$:

$$f(x) \approx f(x_0) + \nabla f(x_0)^T (x - x_0)$$

Пусть $x - x_0 = \alpha \widetilde{d}$, где $||\widetilde{d}|| = 1,\ \alpha > 0$.
Ясно, что:

$$\underset{\widetilde{d}}{argmin} f(x) \approx \underset{\widetilde{d}}{argmin}(f(x_0) + \alpha \nabla f(x_0)^T \widetilde{d}) = \underset{\widetilde{d}}{argmin} \nabla f(x_0)^T \widetilde{d} = -\frac{\nabla f(x_0)}{||\nabla f(x_0)||}$$

Таким образом, направлением наискорейшего спуска является:

$$\widetilde{d} = -\frac{\nabla f(x_0)}{||\nabla f(x_0)||}$$

Приходим к следующему алгоритму градиентного спуска:

1. Выбираем x_0, ε и инициализируем $k \leftarrow 0$
2. $d_k = -\nabla f(x_k)$.
3. Если $||d_k|| < \varepsilon$, то выход.
4. $x_{k+1} \leftarrow x_k + \alpha_k d_k,\ k \leftarrow k + 1$. Переход к шагу 2.

Проблема этого метода заключается в том, что он зависит от системы координат. К примеру, если мы перейдем к $y : x = Ay$, то получится следующая формула:

$$y_{k+1} = y_k - \alpha_k \nabla_y f(Ay) = y_k - \alpha_k A^T \nabla_x f(Ay)$$
$$Ay_{k+1} = Ay_k - \alpha_k A A^T \nabla_x f(Ay)$$

Проблема в том, что $Ay_{k+1} \neq x_{k+1}$

Отсюда следует, что, вообще говоря, скорость сходимости метода может зависеть от системы координат.

3 Метод Ньютона

Пусть перед нами стоит задача минимизировать некоторую функцию $f(x) : \mathbb{R}^n \to \mathbb{R}$.
Попробуем приблизить функцию $f(x)$ квадратичной формой и рассмотрим на этот раз разложение функции f до второго члена в ряд Тейлора:

$$f(x) \approx f(x_0) + \nabla f(x_0)^T (x - x_0) + \frac{1}{2}(x - x_0)^T H(x_0)(x - x_0)$$

где $H(x_0)$ — гессиан $f(x)$ в точке x_0.

В точке минимума должно выполняться равенство $\nabla f(x) = 0$, следовательно:

$$\nabla f(x) = \nabla f(x_0) + H(x_0)(x - x_0) = 0$$

$$x - x_0 = -H(x_0)^{-1} \nabla f(x_0)$$

Приходим к следующему алгоритму:

1. Выбираем $x_0, \quad \varepsilon$ и инициализируем $k \leftarrow 0$
2. $d_k = -H(x_k)^{-1} \nabla f(x_k)$.
3. Если $||d_k|| < \varepsilon$, то выход.
4. $x_{k+1} \leftarrow x_k + \alpha_k d_k, \ k \leftarrow k + 1$. Переход к шагу 2.

Если мы снова рассмотрим $y : x = Ay$, то получим:

$$H(y) = A^T H(x) A$$

$$\nabla_y f(Ay) = A^T \nabla_x f(Ay)$$

Следовательно:

$$d = -(A^T H(x) A)^{-1} A^T \nabla_x f(Ay) = A^{-1} H(x)^{-1} \nabla_x f(Ay)$$

Таким образом:

$$Ay_{k+1} = Ay_k + \alpha_k A A^{-1} H(x)^{-1} \nabla_x f(Ay_k) = x_{k+1}$$

Значит, этот метод не меняется при аффинных преобразованиях координат. Однако он может меняться при общих преобразованиях.

4 Метод натурального градиента

4.1 Описание метода

Предположим, что мы имеем параметры $w = (w_1, w_2, ..., w_n)$. В обычном случае евклидова пространства для расстояния имеет место следующее соотношение:

$$d(w, w + \delta w)^2 = \sum_{i=1}^{n} \delta {w_i}^2$$

Однако такое расстояние не всегда имеет смысл. Например, расстояние на сфере или на какой-нибудь другой поверхности с кривизной будет иметь другой вид.

Допустим, что расстояние в пространстве задается следующим образом:

$$d(w, w + \delta w)^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} g_{ij}(w)\delta w_i \delta w_j = \delta w^T G(w) \delta w$$

где $G(w)$ — метрический тензор Римана. $G(w) = G(w)^T > 0$. Пространство с таким расстоянием — риманово пространство.

Заметим, что если

$$g_{ij} = \delta_{ij} = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases}$$

то $G(w) = I$, и мы получаем обычный евклидовый ортонормальный случай.

Предположим, что перед нами стоит задача минимизировать некоторую функцию $f(x) : \mathbb{R}^n \to \mathbb{R}$. Как делать градиентный спуск в таком случае? Рассмотрим разложение по формуле Тейлора до члена первого порядка:

$$f(w + \delta w) \approx f(w) + \nabla f(w)^T \delta w$$

Пусть $\delta w = \varepsilon a$, где $||a||^2 = a^T G(w) a = 1$, тогда:

$$f(w + \delta w) \approx f(w) + \varepsilon \nabla f(w)^T a$$

Пользуясь методом множителей Лагранжа можно получить:

$$\frac{\partial}{\partial a}(\nabla f(w)^T a - \lambda a^T G(w) a) = 0$$

Следовательно:

$$\nabla f(w) = 2\lambda G(w)a$$

$$a = \frac{1}{2\lambda} G(w)^{-1} \nabla f(w)$$

Неизвестная λ находится из условия нормировки $||a|| = 1$. Получаем, что $-G(w)^{-1}\nabla f(w)$ является направлением наискорейшего спуска в пространстве с римановой метрикой.
Приходим к следующему алгоритму:

1. Выбираем $x_0,\ \varepsilon$ и инициализируем $k \leftarrow 0$
2. $d_k = -G(x_k)^{-1}\nabla f(x_k)$.
3. Если $||d_k|| < \varepsilon$, то выход.
4. $x_{k+1} \leftarrow x_k + \alpha_k d_k,\ k \leftarrow k + 1$. Переход к шагу 2.

4.2 Примеры

4.2.1 Полярная система координат

Рассмотрим полярную систему координат на плоскости:

$$x = rcos\phi$$

$$y = rsin\phi$$

Предположим, что вектор w получается из v переходом к полярным координатам:

$$v = v(x, y)$$

$$w = w(r, \phi)$$

Должно выполняться равенство:

$$d(w, w + \delta w)^2 = d(v, v + \delta v)^2$$

$$d(v, v + \delta v)^2 = \delta x^2 + \delta y^2$$

$$\begin{aligned} w + \delta w &= \begin{bmatrix} (r + \delta r)cos(\phi + \delta\phi) \\ (r + \delta r)sin(\phi + \delta\phi) \end{bmatrix} \\ &= \begin{bmatrix} rcos\phi + \delta r\ cos\phi - \delta\phi\ rsin\phi \\ rsin\phi + \delta r\ sin\phi + \delta\phi\ rcos\phi \end{bmatrix} \end{aligned}$$

$$\delta w = \begin{bmatrix} \delta r\ cos\phi - \delta\phi\ rsin\phi \\ \delta r\ sin\phi + \delta\phi\ rcos\phi \end{bmatrix}$$

Здесь членами вида $\delta r\delta\phi^n$ и $\delta\phi^{n+1}$ можно пренебречь.
Отсюда:

$$d(w, w + \delta w)^2 = \delta r^2 + r^2\delta\phi^2 = \delta w^T G(w)\delta w$$

где

$$G(w) = \begin{bmatrix} 1 & 0 \\ 0 & r^2 \end{bmatrix}$$

В данном случае получилось, что $G(w)$ зависит от w, однако это не всегда так. Примером является линейное преобразование евклидовых координат.

Примечание: Данный пример взят из [4]

4.2.2 Статистическая оценка функции вероятности

Предположим, что есть неизвестное распределение с функцией вероятности $Q(z)$ и мы пытаемся приблизить ее с помощью функции $P(z, w)$, где $w = (w_1, w_2, ..., w_n)$ — параметр.

Часто в качестве расстояния между двумя распределениями рассматривают дивергенцию Кулльбака-Лейблера:

$$KL(Q(z)||P(z,w)) = \mathbb{E}_q \left[\log \frac{Q(z)}{P(z,w)}\right]$$

Оптимальный параметр $\hat{w}$ характеризуется тем, что $Q(z) = P(x, \hat{w})$. При нем достигается минимум KL-дивергенции, а значит достигается оценка максимального правдоподобия.

Ясно, что

$$\begin{aligned} \underset{w}{argmin}\ KL(Q(z)||P(z,w)) =& \underset{w}{argmin}\ \mathbb{E}_Q \left[\log \frac{Q(z)}{P(z,w)}\right] = \\ =& \underset{w}{argmin}\ -\mathbb{E}_Q \left[\log\ P(z,w)\right] - H_q = \\ =& \underset{w}{argmin}\ -\mathbb{E}_Q \left[\log\ P(z,w)\right] \end{aligned}$$

где H_Q — энтропия $Q(z)$ и не зависит от w. Поэтому достаточно минимизировать $L(w) = -\mathbb{E}_Q \left[\log\ P(z,w)\right]$.

Как минимизировать данную функцию с помощью метода натурального градиента? Необходимо найти метрический тензор Римана.

Обозначим $H\left[f(x)\right]$ — гессиан $f(x)$.
Заметим, что

$$\begin{aligned} & KL(P(z,w), P(z,w+\delta w)) = \\ & = \mathbb{E}\left[\log\ \frac{P(z,w)}{P(z,w+\delta w)}\right] = \\ & = \mathbb{E}\left[\log\ P(z,w) - \log\ P(z,w+\delta w)\right] = \\ & \approx \mathbb{E}\left[\log\ P(z,w) - \log\ P(z,w) - \nabla \log\ P(z,w)^T \delta w - \frac{1}{2}\delta w^T H\left[\log\ P(z,w)\right]\delta w\right] = \\ & = -\mathbb{E}\left[\nabla \log\ P(z,w)^T \delta w + \frac{1}{2}\delta w^T H\left[\log\ P(z,w)\right]\delta w\right] = \\ & = -\mathbb{E}\left[\nabla \log\ P(z,w)^T \delta w\right] - \mathbb{E}\left[\frac{1}{2}\delta w^T H\left[\log\ P(z,w)\right]\delta w\right] \end{aligned}$$

Здесь мы переходим к определению матожидания для дискретного случая:

$$KL(P(z,w), P(z,w+\delta w)) =$$
$$= -\sum_{z\in Z} P(z,w)\nabla \log\, P(z,w)^T \delta w - \frac{1}{2}\sum_{z\in Z} P(z,w)\delta w^T H\left[\log\, P(z,w)\right]\delta w$$

Рассмотрим первую сумму:

$$\begin{aligned}
&-\sum_{z\in Z} P(z,w)\nabla \log\, P(z,w)^T \delta w = \\
&= -\sum_{z\in Z} P(z,w)\frac{\nabla P(z,w)^T}{P(z,w)}\delta w = \\
&= -\sum_{z\in Z} \nabla P(z,w)^T \delta w = \\
&= -\nabla(\sum_{z\in Z} P(z,w))^T \delta w = \\
&= -\nabla 1^T \delta w = \\
&= 0
\end{aligned}$$

Следовательно:

$$\begin{aligned}
&KL(P(z,w), P(z,w+\delta w)) = \\
&= -\frac{1}{2}\delta w^T \sum_{z\in Z} P(z,w) H\left[\log\, P(z,w)\right]\delta w = \\
&= -\frac{1}{2}\delta w^T \sum_{z\in Z} P(z,w)\frac{P(z,w)H\left[P(z,w)\right] - \nabla P(z,w)\nabla P(z,w)^T}{P(z,w)^2}\delta w = \\
&= -\frac{1}{2}\delta w^T \sum_{z\in Z} H\left[P(z,w)\right]\delta w - \frac{1}{2}\delta w^T \sum_{z\in Z} P(z,w)\nabla \log\, P(z,w)\nabla \log\, P(z,w)^T \delta w = \\
&= -\frac{1}{2}\delta w^T H\left[\sum_{z\in Z} P(z,w)\right]\delta w - \frac{1}{2}\delta w^T \sum_{z\in Z} P(z,w)\nabla \log\, P(z,w)\nabla \log\, P(z,w)^T \delta w = \\
&= -\frac{1}{2}\delta w^T H\left[1\right]\delta w - \frac{1}{2}\delta w^T \sum_{z\in Z} P(z,w)\nabla \log\, P(z,w)\nabla \log\, P(z,w)^T \delta w = \\
&= -\frac{1}{2}\delta w^T \sum_{z\in Z} P(z,w)\nabla \log\, P(z,w)\nabla \log\, P(z,w)^T \delta w = \\
&= \frac{1}{2}\delta w^T G(w)\delta w
\end{aligned}$$

Где $G(w)$ — матрица информации Фишера.

$$G(w) = -\mathbb{E}\left[\nabla \log\, P(z,w) \nabla \log\, P(z,w)^T\right]$$

$$g_{ij}(w) = -\sum_{z \in Z} P(z,w) \frac{\partial}{\partial w_i} \left[\log\, P(z,w)\right] \frac{\partial}{\partial w_j} \left[\log\, P(z,w)\right]$$

Таким образом, мы научились приближать заданное распределение с помощью метода натурального градиента. Такие задачи часто встречаются в алгоритмах машинного обучения.

Замечание: Аналогичные рассуждения верны и в случае, когда мы пытаемся приблизить функцию плотности, при условии законности данных преобразований.

Замечание: KL-дивергенция не зависит от параметризации распределений.

Примечание: Данный пример взят из [3]

5 Сравнение методов

5.1 Нормальное распределение

5.1.1 Описание

В качестве примера для сравнения методов стандартного и натурального градиентов возьмем следующую задачу:
Дана выборка $X = (X_1, X_2, ..., X_n),\ X_i \sim \mathcal{N}(a_0, \sigma_0^2)$ из нормального распределения, необходимо подобрать параметры $a_0,\ \sigma_0$.

Для этого можно воспользоваться методом максимального правдоподобия:

$$L(X; a, \sigma) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x_i-a)^2}{2\sigma^2}}$$

$$\log L(X; a, \sigma) = -n\log \sigma - \frac{n}{2}\log 2\pi - \sum_{i=1}^{n} \frac{(x_i - a)^2}{2\sigma^2}$$

$$\nabla\log L(X; a, \sigma) = \begin{bmatrix} \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - a) \\ \frac{1}{\sigma^3}\sum_{i=1}^{n}(x_i - a)^2 - \frac{n}{\sigma} \end{bmatrix}$$

Несложно показать, что оценку максимального правдоподобия в данном случае можно выразить аналитически, однако, для демонстрации работы метода этот пример удобен.

Матрица информации Фишера была выражена на стр. 10. Нужно лишь показать как вычислить ее по выборке:

$$\nabla\log P(z, w) = \begin{bmatrix} \frac{x_i - a}{2\sigma^2} \\ \frac{(x_i - a)^2}{\sigma^3} - \frac{1}{\sigma} \end{bmatrix}$$

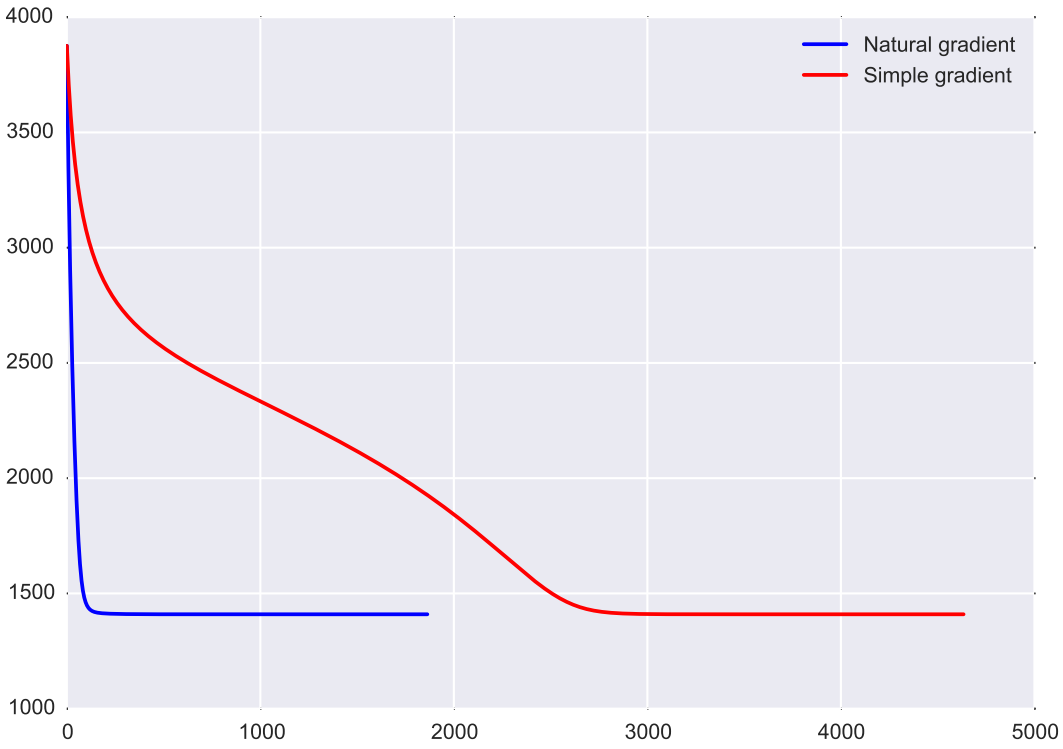
$$G(w) = -\mathbb{E}\left[\nabla\log P(z, w)\nabla\log P(z, w)^T\right] \approx \frac{1}{n}\sum_{i=1}^{n}\left[\nabla\log P(z, w)\nabla\log P(z, w)^T\right]$$

5.1.2 Результаты и графики

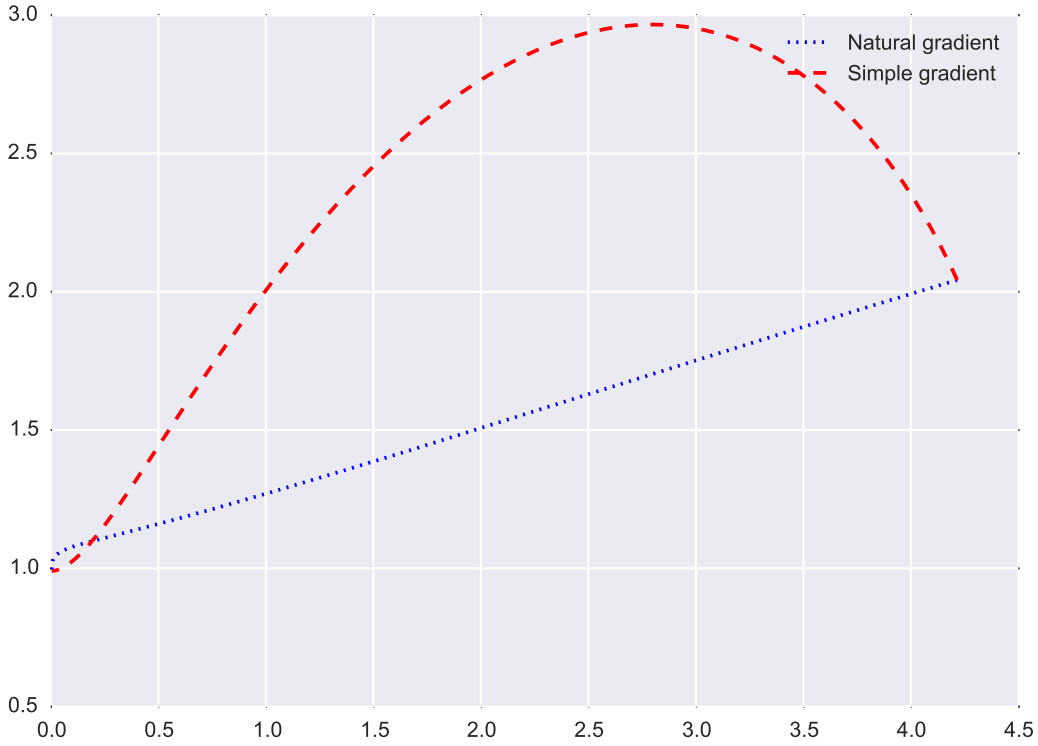
Исходная выборка была сгенерирована из стандартного нормального распределения. Ее размер — n был выбран равным 1000. В качестве начальных значений параметров (откуда мы начинаем идти) выбирались случайным образом в интервалах от -100 до 100 и от 0 до 100 для среднего и дисперсии соответственно. Параметры алгоритмов:

$$\alpha = 0.000005$$
$$\varepsilon = 0.0001$$

Зависимость $-\log L(X; a, \sigma)$ от числа итераций:



Точки, по которым проходили методы стандартного и натурального градиента:



Видно, что метод натурального градиента практически сразу сошелся к нужному значению параметров распределения. Также, метод натурального градиента сразу выбрал правильное направление на плоскости параметров (a, σ), в отличие от метода стандартного градиентного спуска.

5.2 Полярная система координат

5.2.1 Описание

Рассмотрим другой пример: дана функция

$$f(r, \phi) = (r\ cos\phi - 1)^2 + (r\ sin\phi)^2,$$

необходимо найти ее минимум. Ясно, что если мы перейдем к замене

$$x = rcos\phi$$
$$y = rsin\phi,$$

то получим функцию $g(x, y) = (x-1)^2+y^2$. Ее минимум находится в точке $(x,\ y) = (1,\ 0)$. Однако из-за перехода в исходной функции к полярной системе координат стандартный градиентный спуск работает несколько иначе и не находит прямой путь в минимум. Найдем градиент:

$$\nabla f(r, \phi) = \begin{bmatrix} 2(r - cos\phi) \\ 2rsin\phi \end{bmatrix}$$

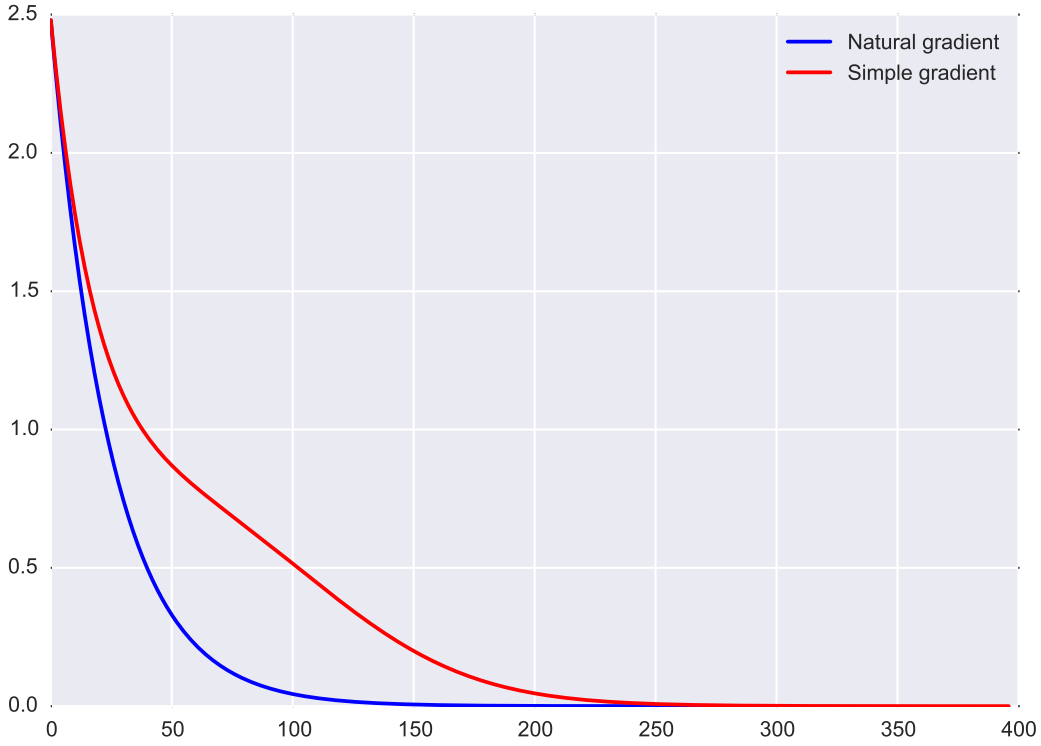
Метрический тензор Римана для полярных координат был найден на стр. 7.

5.2.2 Результаты и графики

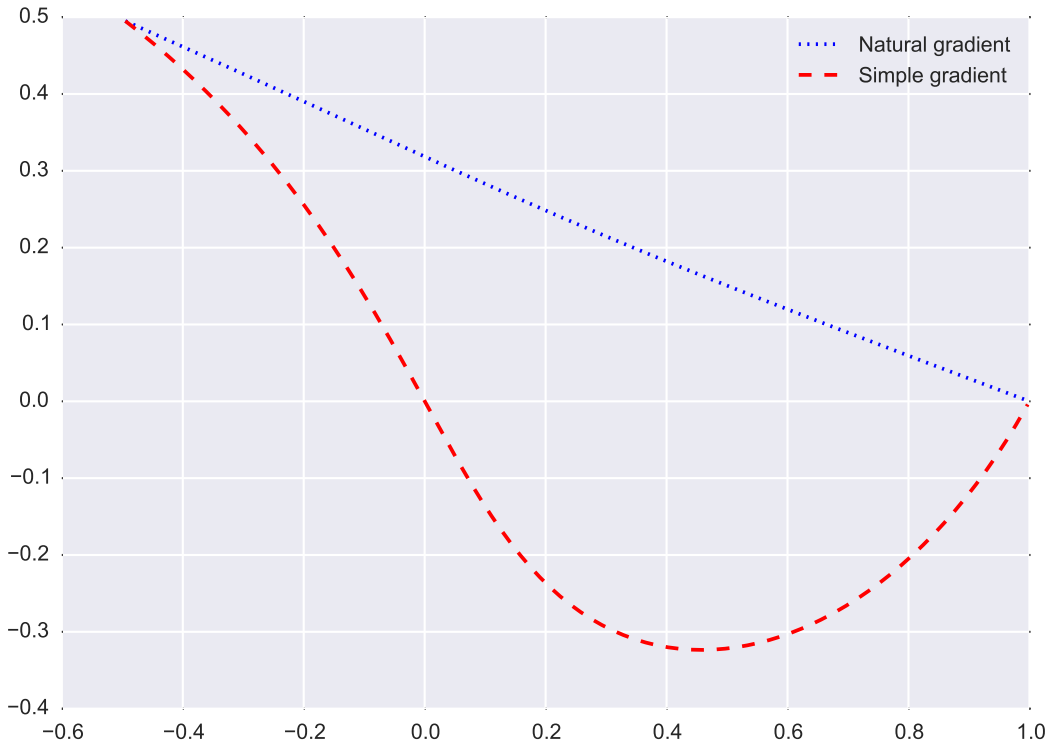
В качестве начальных значений параметров выбирались случайные начальные значение от 0 до 10 и от 0 до 2π для r и ϕ соответственно. Параметры алгоритмов:

$$\alpha = 0.01$$
$$\varepsilon = 0.0001$$

Зависимость значения функции от числа итераций:



Точки, по которым проходили методы стандартного и натурального градиента:



В данном примере метод натурального градиента примерно в 2 раза быстрее сошелся к минимуму. Метод натурального градиента снова сразу выбрал правильное направление на плоскости параметров (r, ϕ), в отличие от метода стандартного градиентного спуска.

6 Применения натурального градиента

1. Обучение нейронных сетей

   Нейронные сети являются широким разделом алгоритмов машинного обучения. Их необходимо обучать подбирая некоторые параметры — веса. Подбор осуществляется методом градиентного спуска. Этот метод можно улучшить применяя метод натурального градиента.

2. Слепое разделение источников сигнала

   Данная задача подразумевает, что мы знаем, что есть несколько источников сигнала, и что эти сигналы могут смешиваться друг с другом. Однако мы не знаем каким именно образом они смешиваются и задача состоит в том, чтобы разделить их.

3. Слепая развертка сигналов

   Эта задача похожа по формулировке на предыдущую, однако, отличается от нее тем фактом, что текущие сигналы могут смешиваться с предыдущими. Тут также используется метод натурального градиента.

   Примечание: Применение натурального градиента для решения приведенных выше проблем описаны в [3]

7 Заключение

Метод натурального градиента имеет свои преимущества над другими методами оптимизации. Он позволяет находить естественное для пространства параметров направление наискорейшего спуска, и следовательно часто сходится быстрее остальных. Однако он не лишен недостатков:

1. Необходимо глубокое понимание проблемы для нахождения $G(w)$.
2. Обращение матрицы — вычислительно довольно затратная операция.

В целом, этот алгоритм является хорошим методом оптимизации функционалов, позволяющим добиться большой скорости сходимости. В современном машинном обучении используется не так часто как раз из-за необходимости обращать матрицу на каждой итерации. Это дает высокие накладные расходы.

8 Список литературы

[1] Pieter Abbeel. *CS 287: Advanced Robotics*. http://www.cs.berkeley.edu/~pabbeel/cs287-fa09/lecture-notes/lecture20-6pp.pdf. 2009.

[2] Nicolas Le Roux. *Using Gradient Descent for Optimization and Learning*. http://www.gatsby.ucl.ac.uk/teaching/courses/ml2-2008/graddescent.pdf. 2009.

[3] S.Amari. "Natural Gradient Works Efficiently in Learning". B: *Neural Computation (Volume:10 , Issue: 2)* (1998).

[4] S.Amari S.C. Douglas. "Why natural gradient?" B: *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on (Volume:2)* (1998).

9 Приложение

Реализация натурального градиента.

```
import numpy as np

def natural_gradient_descent(x0, G_inv, grad, alpha,
↪   eps=0.0001, f=None, iters=None):
    x = x0.copy()
    d = np.dot(G_inv(x), grad(x, 0))

    points = [x.copy()]

    hist = None
    if f is not None:
        hist = [f(x, 0)]

    iter_num = 1

    if iters is None:
        is_end = lambda : np.sum(d**2) < eps
    else:
        is_end = lambda : iter_num >= iters

    while not is_end():
        x -= alpha * d
        d = np.dot(G_inv(x), grad(x))

        points.append(x.copy())

        if f is not None:
            hist.append(f(x))

        iter_num += 1

    points = np.array(points)
    if f is not None:
        return x, points, hist
    return x, points

def simple_gradient_descent(x0, grad, alpha, eps=0.0001,
↪   f=None, iters=None):
```

```
return natural_gradient_descent(x0, lambda x:
 ↪  np.eye(x0.shape[0]), grad, alpha, eps, f, iters)
```

Тест с нормальным распределением.

```
import numpy as np
import matplotlib.pyplot as plt
import math
from natgrad import *

def test():

    n = 1000
    X = np.random.randn(n)

    x0 = 10 * np.random.randn(2)
    x0[1] = np.abs(x0[1])

    print "True mean: ", x0[0]
    print "True variance: ", x0[1]

    def loglikelihood(w, k=None):
        a, sigma = w[0], w[1]
        return sum(((x - a)**2 / (2 * sigma**2) for x in X)) +
         ↪  n * np.log(sigma) + n / 2.0 * np.log(2.0 * math.pi)

    def loglikelihood_grad(w, k=None):
        a, sigma = w[0], w[1]
        print "Mean: ", a, "Variance: ", sigma
        return np.array([
            -1.0 / sigma**2 * sum((x - a for x in X)),
            -1.0 / sigma**3 * sum(((x - a)**2 for x in X)) + n
             ↪  / sigma
        ])

    def G(w):
        a, sigma = w[0], w[1]
        G = np.zeros((2, 2))
        for x in X:
            log_grad = np.array([
                [(x - a) / (2 * sigma ** 2)],
                [(x - a)**2 / sigma ** 3 - 1.0 / sigma]
            ])
            G += np.dot(log_grad, log_grad.T)
```

```
    return G / n

G_inv = lambda w: np.linalg.pinv(G(w))

alpha = 0.000005

x, points, hist = natural_gradient_descent(x0, G_inv,
↪  loglikelihood_grad, alpha, f=loglikelihood)
x_simple, points_simple, hist_simple =
↪  simple_gradient_descent(x0, loglikelihood_grad, alpha,
↪  f=loglikelihood)

plt.plot(hist, color='b', label="Natural gradient")
plt.plot(hist_simple, color='r', label="Simple gradient")
plt.legend()
plt.show()

plt.plot(points[:, 0], points[:, 1], color='b',
↪  label="Natural gradient", linestyle=":")
plt.plot(points_simple[:, 0], points_simple[:, 1],
↪  color='r', label="Simple gradient", linestyle="--")

plt.legend()
plt.show()
```

Тест с полярной системой координат.

```
import numpy as np
import matplotlib.pyplot as plt
import math
from natgrad import *
from pprint import pprint

def polar_G(w):
    return np.array([
        [1.0, 0.0],
        [0.0, w[0]**2]
    ])

def polar_f(w, k=None):
    r, phi = w[0], w[1]
    return (r * math.cos(phi) - 1)**2 + (r * math.sin(phi))**2

def polar_f_grad(w, k=None):
    r, phi = w[0], w[1]
    return np.array([
            2 * (r - math.cos(phi)),
            2 * r * math.sin(phi)
        ])

def polar_to_cartesian(r_phi):
    x = np.zeros(r_phi.shape)
    r, phi = r_phi[:, 0], r_phi[:, 1]

    x[:, 0] = r * np.cos(phi)
    x[:, 1] = r * np.sin(phi)

    return x


def test():

    polar_x0 = np.array([0.7, 3 * math.pi / 4.0])
    polar_G_inv = lambda w: np.linalg.pinv(polar_G(w))
    polar_alpha = 0.01
```

```
polar_x, polar_points, hist =
 ↪  natural_gradient_descent(polar_x0, polar_G_inv,
 ↪  polar_f_grad, polar_alpha, f=polar_f)
polar_x_simple, polar_points_simple, hist_simple =
 ↪  simple_gradient_descent(polar_x0, polar_f_grad,
 ↪  polar_alpha, f=polar_f)

plt.plot(hist, color='b', label="Natural gradient")
plt.plot(hist_simple, color='r', label="Simple gradient")
plt.legend()
plt.show()

polar_points = polar_to_cartesian(polar_points)
polar_points_simple =
 ↪  polar_to_cartesian(polar_points_simple)

print "Iterations (natural): ", len(polar_points)
print "Iterations (simple): ", len(polar_points_simple)

plt.plot(polar_points[:, 0], polar_points[:, 1], color='b',
 ↪  label="Natural gradient", linestyle=':')
plt.plot(polar_points_simple[:, 0], polar_points_simple[:,
 ↪  1], color='r', label="Simple gradient", linestyle='--')

plt.legend()
plt.show()
```

Основной файл.

```
import numpy as np
import matplotlib.pyplot as plt
import math
import seaborn

import polar_test
import likelihood_test

def main():
    polar_test.test()
    likelihood_test.test()

main()
```