Нейронные сети и байесовский подход

Сергей Николенко

Машинное обучение — ИТМО, осень 2006

Outline

В прошлом семестре

- В прошлом семестре мы уже рассматривали нейронные сети.
- Сейчас мы кратко напомним, что там было, а потом будем двигаться дальше.

Общая структура

- Есть сеть из нейронов, соединённых между собой
- Нейроны возбуждаются под действием входов и передают возбуждение (либо как один бит, либо с каким-то значением) дальше
- В результате последний нейрон на выход подаёт ответ

Линейный перцептрон

У линейного перцептрона заданы:

- n весов $w_1, w_2, \ldots, w_n$;
- лимит активации w_0;
- выход перцептрона $o(x_1, \ldots, x_n)$ вычисляется так:

$$o(x_1, \ldots, x_n) = \begin{cases} 1, \text{ если } w_0 + w_1 x_1 + \ldots + w_n x_n > 0, \\ -1 \text{ в противном случае.} \end{cases}$$

- или запишем иначе, введя переменную $x_0 = 1$:

$$o(x_1, \ldots, x_n) = \begin{cases} 1, \text{ если } \sum_i w_i x_i > 0, \\ -1 \text{ в противном случае.} \end{cases}$$

Как обучать перцептрон?

- Всё, что может у перцептрона меняться — это веса w_i, $i = 0..n$.
- Их мы и будем подправлять при обучении.
- Если перцептрон отработал правильно, веса не меняются.
- Если неправильно — сдвигаются в нужную сторону.

Perceptron training rule

Простейшее правило:

$$w_i \leftarrow w_i + \eta(t - o)x_i,$$

где:

- t — значение целевой функции
- o — выход перцептрона
- $\eta > 0$ — небольшая константа (обычно 0.05–0.2), которая задаёт скорость обучения

Перцептрон без лимита активации

Бывают перцептроны, у которых нет лимита активации. Они просто выдают линейную форму от своих входов:

$$o(x_0, \ldots, x_n) = \sum_0^n w_i x_i.$$

Т.е. у такого перцептрона не два, а континуально много возможных значений.

В остальном это всё те же линейные перцептроны.

Функция ошибки

Мы хотим найти перцептрон, который минимизирует ошибку.
Пусть есть m тестовых примеров x_i^j с верными ответами t^j, $j = 1..m$.
Ошибка — из статистики, среднеквадратичное отклонение:

$$E(w_0, \ldots, w_n) = \frac{1}{2} \sum_{j=1}^{m} (t_j - o(x_0^j, \ldots, x_n^j))^2.$$

Задача: минимизировать функцию E на пространстве возможных весов $\{w_i\}$.

Градиент от функции ошибки

В нашем случае подсчитать градиент совсем просто:

$$\frac{\partial E}{\partial w_i} = \frac{1}{2}\sum_{j=1}^{m}\frac{\partial}{\partial w_i}\left(t^j - \sum_{0}^{n} w_i x_i^j\right)^2 = \\ = \sum_{j=1}^{m}\left(t^j - \sum_{0}^{n} w_i x_i^j\right)(-x_i^j).$$

Изменения весов примут вид:

$$w_i \leftarrow w_i + \eta\sum_{j}\left(t^j - \sum_{0}^{n} w_i x_i^j\right)x_i^j.$$

Нелинейные перцептроны

- Всё то же самое, но функция выхода уже не линейная.
- Но от линейной формы всё же не отказываемся, а берём её результат и сглаживаем.
- Популярная функция — *сигмоид*:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

- То есть выход перцептрона подсчитывается по формуле:

$$o(x_1, \ldots, x_n) = \frac{1}{1 + e^{-\sum_i w_i x_i}}.$$

Градиент

- Почему сигмоид? Потому что легко считать производную:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)).$$

- Соответственно, правило модификации веса для одного перцептрона выглядит как

$$w_i \leftarrow w_i + \eta o(x)(1 - o(x))(t(x) - o(x))x_i,$$

где $t(x)$ — целевое значение функции.

Нейронная сеть

- У сети есть входы $x_1, \ldots, x_n$, выходы Outputs и внутренние узлы.
- Перенумеруем все узлы (включая входы и выходы) числами от 1 до N.
- w_{ij} — вес, стоящий на ребре (i, j).
- o_i — выход i-го узла.
- Даны m тестовых примеров с целевыми значениями выходов $\{t_k^d\}, d=1..m, k \in \text{Outputs}$.

Функция ошибки:

$$E(\{w_{ij}\}) = \frac{1}{2} \sum_{d=1}^{m} \sum_{k \in \text{Outputs}} \left(t_k^d - o_k(x_1^d, \ldots, x_n^d) \right)^2.$$

Градиент

Как подсчитать градиент функции ошибки для каждого тестового примера?

$$E^d(\{w_{ij}\}) = \frac{1}{2} \sum_{k \in \text{Outputs}} \left(t_k^d - o_k^d\right)^2.$$

Случай 1. Считаем $\Delta w_{ij} = -\eta \frac{\partial E^d}{\partial w_{ij}}$, где $j \in \text{Outputs}$.

- w_{ij} влияет на выход o^d только как часть суммы $S_j = \sum_i w_{ij} x_{ij}$, поэтому

$$\frac{\partial E^d}{\partial w_{ij}} = \frac{\partial E^d}{\partial S_j} \frac{\partial S_j}{\partial w_{ij}} = x_{ij} \frac{\partial E^d}{\partial S_j}.$$

- S_j влияет на общую ошибку только в рамках выхода j-го узла o_j (это выход всей сети). Поэтому:

Градиент

Как подсчитать градиент функции ошибки для каждого тестового примера?

$$E^d(\{w_{ij}\}) = \frac{1}{2} \sum_{k \in \text{Outputs}} \left(t_k^d - o_k^d\right)^2.$$

$$\frac{\partial E^d}{\partial S_j} = \frac{\partial E^d}{\partial o_j} \frac{\partial o_j}{\partial S_j} = \left(\frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{Outputs}} (t_k - o_k)^2\right) \left(\frac{\partial \sigma(S_j)}{\partial S_j}\right) =$$

$$= \left(\frac{1}{2} \frac{\partial}{\partial o_j} (t_j - o_j)^2\right) (o_j(1 - o_j)) = -o_j(1 - o_j)(t_j - o_j).$$

Градиент

Как подсчитать градиент функции ошибки для каждого тестового примера?

$$E^d(\{w_{ij}\}) = \frac{1}{2} \sum_{k \in \text{Outputs}} \left(t_k^d - o_k^d\right)^2.$$

Случай 2. Считаем $\Delta w_{ij} = -\eta \frac{\partial E^d}{\partial w_{ij}}$, где $j \notin \text{Outputs}$.

- У узла j есть потомки $\text{Children}(j)$. Тогда

$$\frac{\partial E^d}{\partial S_j} = \sum_{k \in \text{Children}(j)} \frac{\partial E^d}{\partial S_k} \frac{\partial S_k}{\partial S_j}, \text{ и } \frac{\partial S_k}{\partial S_j} = \frac{\partial S_k}{\partial o_j} \frac{\partial o_j}{\partial S_j} = w_{jk} o_j (1 - o_j).$$

- $\frac{\partial E^d}{\partial S_k}$ — это в точности аналогичная поправка, но вычисленная для узла следующего уровня. Так мы и дойдём от выходов ко входам.

Резюме подсчёта градиента

- Для узла последнего уровня

$$\delta_j = -o_j(1 - o_j)(t_j - o_j).$$

- Для внутреннего узла сети

$$\delta_j = -o_j(1 - o_j) \sum_{\text{Outputs}(j)} \delta_k w_{jk}.$$

- Для любого узла

$$\Delta w_{ij} = -\eta \delta_j x_{ij}.$$

Outline

Функция ошибки

- Начнём с одного нейрона. Пусть у него n входов $\vec{x} = \{x_1, \ldots, x_n\}$, веса $\vec{w} = \{w_0, \ldots, w_n\}$, и он вычисляет функцию $o(\vec{x}, \vec{w}) \in [0, 1]$.
- Ему дают данные $\{t^j = f(\vec{x}^j)\}_{j=1}^m$, причём $t^j \in \{0, 1\}$. Какова функция ошибки?
- Мы уже даже доказывали, что в случае данных с нормально распределённым шумом функцией ошибки должно быть среднеквадратичное отклонение.
- А какая должна быть функция ошибки с точки зрения теории информации?

Функция ошибки и информация

- Информационное содержание одного запуска — это относительная энтропия между эмпирическим распределением $(t^j, 1-t^j)$ и распределением выхода нейрона $(o, 1-o)$:

$$G^j(\vec{w}) = -t^j \ln o(\vec{x}^j, \vec{w}) - (1-t^j) \ln \left(1 - o(\vec{x}^j, \vec{w})\right).$$

- То есть мы рассматриваем перцептрон как классификатор, а данные t и выход перцептрона $o(\vec{x}, \vec{w}) \in [0,1]$ — как распределение вероятностей (с какой вероятностью ответ равен 1).
- Общая ошибка, естественно, получается как сумма:

$$G(\vec{w}) = -\sum_{j=1}^{m} \left[t^j \ln o(\vec{x}^j, \vec{w}) + (1-t^j) \ln \left(1 - o(\vec{x}^j, \vec{w})\right)\right].$$

Минимизация относительной энтропии

- Алгоритм от этого не меняется, кстати. Если $o(\vec{x}, \vec{w}) = \frac{1}{1+e^{-\vec{x}\vec{w}}}$, то, продифференцировав $G(\vec{w})$ по w_i, получим
$$\frac{\partial G}{\partial w_i} = -\sum_j (t^j - o^j) x_i^j.$$
- То есть по-прежнему градиентный спуск ходит по направлению, обратному ошибке.
- Однако относительная энтропия позволит нам потом разработать другие методы.

Оверфиттинг

- А пока поборемся с оверфиттингом. Оверфиттинг — это когда модель *слишком* хорошо отвечает данным.
- В случае нейронных сетей такое бывает, когда данным соответствуют всё лучше и лучше, но веса при этом растут до бесконечности.
- Как этого избежать? Можно, конечно, искусственно останавливаться раньше, но это слишком уж ad hoc.

Регуляризация

- Нужно внести в функцию ошибки поправку, которая будет убирать нежелательные эффекты (такой процесс называется *регуляризацией*).
- Чтобы убрать нейронный оверфиттинг, будем рассматривать функцию ошибки

$$M(\vec{w}) = G(\vec{w}) + \alpha E(\vec{w}),$$

где α — константа (*гиперпараметр*), а функция E — *регуляризатор*

$$E(\vec{w}) = \frac{1}{2}\sum_{i=1}^{n} w_i^2.$$

Упражнение. Как изменится алгоритм градиентного спуска?

Суть проблемы

- Во-первых, скорость обучения должна меняться со временем, становиться меньше по мере приближения к минимуму и быть достаточно большой далеко от него.
- Во-вторых, неплохо бы всё-таки научиться вылезать хотя бы из самых мелких локальных минимумов на пути к глобальному.

Момент

- Одно из решений — ввести в правило пересчёта весов так называемый *момент* (momentum). Пересчёт на шаге t имеет вид:

$$\Delta w_{ij}(t) = \delta + m\Delta w_{ij}(t-1),$$

где δ — изменение веса по обычному алгоритму (какому угодно).

- То есть по мере продвижения к минимуму нейронная сеть как бы «разгоняется», одновременно и быстрее к нему приближаясь, и, возможно, перепрыгивая мелкие минимумы и ущелья целевой функции по ходу дела.

Адаптация скорости обучения

- Нужно менять скорость обучения, чтобы быстрее придти к минимуму и там остаться.
- Для этого есть разные методы; мы рассмотрим три из них.

Bold driver

- Модель «бесстрашного водителя» — увеличивать скорость, пока не врежешься во что-нибудь, а потом уменьшать.
- Пока ошибка уменьшается, скорость обучения η ненамного увеличивается (на 5–10%).
- Как только ошибка при очередном шаге увеличилась, скорость обучения тут же резко падает (на $\approx 50\%$).

Уменьшение скорости обучения

- Если проблема не в том, чтобы побыстрее дойти до минимума, а в том, чтобы там остаться, можно просто уменьшать скорость обучения:

$$\eta(t) = \frac{\eta(0)}{1 + \frac{t}{T}},$$

где T — новый параметр.

- В этой модели скорость падает небыстро во время первых T шагов, когда мы достигаем примерно минимума, но при этом уменьшается достаточно быстро, чтобы гарантировать сходимость.

Градиентный спуск для скорости обучения

- Более изощрённый метод — подправлять η при помощи того же алгоритма обучения (градиентного спуска). Пусть веса обновляются как

$$w_{ij}(t+1) = w_{ij}(t) + \eta_{ij}(t)\Delta w_{ij}(t).$$

- Мы хотим изменить $\eta_{ij}(t)$ так, чтобы уменьшить $E(t+1)$ (функцию ошибки):

$$\frac{\partial E(t+1)}{\partial \eta_{ij}(t)} = \frac{\partial E(t+1)}{\partial w_{ij}(t+1)} \frac{\partial w_{ij}(t+1)}{\partial \eta_{ij}(t)} = -\Delta w_{ij}(t)\Delta w_{ij}(t-1).$$

Градиентный спуск для скорости обучения

- То есть можно подправлять η_{ij} по правилу:

$$\eta_{ij}(t) = \eta_{ij}(t-1) + \xi \Delta w_{ij}(t) \Delta w_{ij}(t-1).$$

- Правда, с таким подходом η_{ij} может стать отрицательной. Кроме того, нам надо менять η_{ij} на несколько порядков. Поэтому применим то же самое не к η, а к её логарифму:

$$\log(\eta_{ij}(t)) = \log(\eta_{ij}(t-1)) + \xi \Delta w_{ij}(t) \Delta w_{ij}(t-1).$$

Проблема

- При этом алгоритме значения η_{ij} будут сильно прыгать.
- Поэтому часто заменяют $\Delta w_{ij}(t-1)$ на среднее нескольких последних значений Δw_{ij}, обычно экспоненциальное среднее, которое вычисляется как

$$\bar{u}(t) = m\bar{u}(t-1) + (1-m)u(t),$$

где m — новый параметр.

Outline

Постановка задачи и основная идея

- Вспомним вероятностную постановку задачи обучения.
- Мы интерпретируем результат нейронной сети как вероятность того, что ответ равен 1:

$$p(t=1|\vec{w},\vec{x}) = o(\vec{w},\vec{x}), \quad p(t=0|\vec{w},\vec{x}) = 1 - o(\vec{w},\vec{x}).$$

- При таком подходе $p(t|\vec{w},\vec{x}) = o^t(1-o)^t$, и в целевой функции $M(\vec{w}) = G(\vec{w}) + \alpha E(\vec{w})$

$$G(\vec{w}) = -\sum_{j=1}^{m} \left[t^j \ln o(\vec{x}^j,\vec{w}) + (1-t^j)\ln\left(1-o(\vec{x}^j,\vec{w})\right)\right] =$$

$$= -\log p(t|\vec{w},\vec{x}).$$

- То есть функция ошибки — это как раз функция правдоподобия.

Регуляризатор

- А что такое регуляризатор $E(\vec{w})$? Вспомните формулу Байеса.

Регуляризатор

- А что такое регуляризатор $E(\vec{w})$? Вспомните формулу Байеса.
- Это на самом деле априорное распределение $p(\vec{w}|\alpha) = \frac{1}{Z(\alpha)} e^{-\alpha E}$.
- Если, как раньше, $E(\vec{w}) = \frac{1}{2} \sum_{i=1}^{n} w_i^2$, то это априорное распределение — гауссиан с нулевым средним и вариацией $\sigma^2 = 1/\alpha$.

Задача байесовского обучения

- В чём классическая задача нейронных сетей? Найти лучшее приближение к целевой функции и его потом применять.
- У байесовского подхода есть преимущество: мы можем не просто применять оптимум (который может быть с оверфиттингом, может быть вообще не очень характерный), а усреднять по всем возможным значениям весов.

Задача байесовского обучения

- То есть, чтобы оценить значение функции t на входе $\vec{x}$, нужно подсчитать

$$p(t|\vec{x}, D, \alpha) = \int p(t|\vec{x}, \vec{w}, \alpha) p(\vec{w}|D, \alpha) d\vec{w}, \text{ где}$$

$$p(t = 1|\vec{x}, \vec{w}, \alpha) = o(\vec{x}, \vec{w}),$$

$$p(t = 0|\vec{x}, \vec{w}, \alpha) = 1 - o(\vec{x}, \vec{w}), \text{ а}$$

$$p(\vec{w}|D, \alpha) = \frac{1}{Z_M} e^{-M(\vec{w})}.$$

- Как оценивать такие интегралы?

Оценка интеграла

- Итак, интеграл получается такой:

$$p(t = 1|\vec{x}, D, \alpha) = \int o(\vec{x}, \vec{w}) \frac{1}{Z_M} e^{-M(\vec{w})} d\vec{w}.$$

- Мы оценим интеграл, рассматривая $o(\vec{x}, \vec{w})$ как функцию от $\vec{w}$ по методу Монте-Карло и посчитаем её среднюю

$$\bar{o}(\vec{w}) = \frac{1}{R} \sum_r o(\vec{w}^{(r)},$$

где $w^{(r)}$ мы будем брать по распределению $\frac{1}{Z_M} e^{-M(\vec{w})}$.

- Как получить сэмплы $\{w^{(r)}\}$?

Метод Ланжевена

- Метод Ланжевена — это random walk градиентным спуском с дополнительным шумом.
- Мы генерируем шумовой вектор $\vec{p}$ (по $N(0;1)$), потом подсчитываем градиент целевой функции $\vec{g}$.
- А потом считаем

$$\Delta\vec{w} = -\frac{1}{2}\epsilon^2\vec{g} + \epsilon\vec{p}.$$

- То есть небыстро, но движемся в сторону градиента. Шаги принимаем или отвергаем с такой вероятностью, чтобы выполнялось условие баланса.

Условие баланса

- Напомним условие баланса; в марковском процессе, когда $p^{t+1}(x) = \int T(x,y)p^t(y)dy$,

$$\forall x, y \quad T(x,y)p(y) = T(y,x)p(x).$$

- Т.е. вероятность того, что мы выберем x и дойдём до y, равна вероятности выбрать y и дойти до x.

Упражнение. Выписать условие баланса для данного случая и дать алгоритм его применения.

Гамильтонов метод Монте-Карло

- Этот метод обычно ещё лучше, чем метод Ланжевена.
- Суть его та же, но при этом мы на каждом шаге ходим по градиенту не только по $\vec{w}$, но ещё и по $\vec{p}$, т.е. в пространстве $(\vec{w}, \vec{p})$.
- То есть вместо $\Delta\vec{w} = -\frac{1}{2}\epsilon^2\vec{g} + \epsilon\vec{p}$ мы выбираем случайный $\vec{p}$ и 100–200 раз запускаем алгоритм
  - $\vec{p} := \vec{p} - \epsilon\vec{g}/2$;
  - $\vec{w} := \vec{w} + \epsilon\vec{p}$;
  - $\vec{g} := \nabla M(\vec{w})$;
  - $\vec{p} := \vec{p} - \epsilon\vec{g}/2$.

Ничего не меняется

- На самом деле ничего не меняется. Целевые функции те же, и всё так же нужно оценивать интеграл

$$p(t|\vec{x}, D, \alpha) = \int p(t|\vec{x}, \vec{w}, \alpha) p(\vec{w}|D, \alpha) d\vec{w},$$

  просто весов $\vec{w}$ становится больше, и градиент $\nabla M(\vec{w})$ теперь нужно считать по-другому (вспомните, как именно!).
- Ничего не меняется, потому что мы уже ненавязчиво перешли от нейронных сетей к методам curve fitting: как выучить функцию при помощи параметризованного семейства функций.
- Наш байесовский подход на самом деле разработан для этой, куда более общей ситуации.
- Сейчас мы рассмотрим эту тему более подробно.

Outline

Определение

- Гауссовский процесс — это такое семейство случайных переменных, что любое конечное число из них имеют нормальное совместное распределение.
- Многомерное нормальное распределение задаётся вектором средних $\vec{\mu} = (\mu_1, \ldots, \mu_n)$ и матрицей ковариаций $A = (\mathrm{cov}(X_i, X_j))_{ij} = (E(X_i - \mu_i)(X_j - \mu_j))_{ij}$:

$$p(x_1, \ldots, x_n) = \frac{1}{Z} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^t A^{-1} (\vec{x}-\vec{\mu})}.$$

- Точно так же гауссовский процесс задаётся функцией средних $\mu(x)$ и функцией ковариаций $C(x, x')$.
- Скоро мы увидим, зачем это нам нужно в обучении.

Задача

- Есть данные $\{\vec{x}^{(i)}, t^{(i)}\}_{i=1}^{m}$. Пусть $t^{(i)}$ — вещественные числа, и мы хотим приблизить функцию $y(\vec{x})$, о которой у нас данные.
- Два подхода: параметрический и непараметрический. В параметрическом мы ищем y в виде $y(\vec{x}, \vec{w})$ с явным набором параметров $\vec{w}$ (как в нейронной сети). В непараметрическом — просто ищем $y(\vec{x})$.
- Начнём с параметрических подходов.

Фиксированный базис

- Зафиксируем базис $\{\phi_h(x)\}_{h=1}^{H}$ и будем искать

$$y(\vec{x}, \vec{w}) = \sum_h w_h \phi_h(\vec{x}).$$

- Например, выберем точки $\{\vec{c}_h\}_h$ и возьмём базис

$$\phi_h(\vec{x}) = e^{-\frac{(\vec{x}-\vec{c}_h)^2}{2r^2}}.$$

- Или будем искать в виде полиномов $\phi_h(\vec{x}) = x_i^p x_j^q$. И т.д.

Адаптивный базис

- Это когда базисные функции тоже зависят от параметров. Например:

$$y(\vec{x}, \vec{w}) = w_{02} + \sum_h w_{h2} \frac{1}{1 + e^{-w_{01} - \sum_i w_{i1} x_i}}.$$

- Что это за функция, кстати?

Оптимизация параметров

- Теперь надо найти $\vec{w}$. Это делается так:

$$p(\vec{w}|\vec{t}, y) = \frac{p(\vec{t}|\vec{w}, y)p(\vec{w})}{p(\vec{t}|y)}.$$

- Здесь обычно правдоподобие $p(\vec{t}|\vec{w}, y)$ берут как гауссиан вокруг $y(\vec{x}, \vec{w})$, $p(\vec{w})$ — это априорное распределение параметров, тоже часто нормальное берут (ср. с weight decay).
- А вывод проводится так, как мы уже делали — методами Монте-Карло или минимизацией функции ошибки M.
- Предсказания делаются как раньше:

$$p(t|\vec{x}, D, \alpha) = \int p(t|\vec{x}, \vec{w}, \alpha)p(\vec{w}|D, \alpha)d\vec{w}.$$

Параметрические модели как гауссовские процессы

- Фиксируем точки $\{\vec{x}^{(i)}\}_{i=1}^{n}$ и определим матрицу R: $R_{ih} = \phi_h(\vec{x}^{(i)})$.
- Определим вектор $\vec{y}$ как вектор значений $y(\vec{x})$: $y_i = \sum_h R_{ih} w_h$.
- Если априорное распределение $\vec{w}$ нормально со средним 0: $p(\vec{w}) = N(0, \sigma_w^2 I)$, то y тоже распределён нормально, и его матрица ковариаций

$$Q = \vec{y}\vec{y}^t = R\vec{w}\vec{w}^t R^t = R(\vec{w}\vec{w}^t)R^t = \sigma_w^2 RR^t.$$

Параметрические модели как гауссовские процессы

- То есть у нас получилось, что вектор из n значений $y(\vec{x})$ нормально распределён, и это верно для любых точек и любого n. Это и есть определение гауссовского процесса.

Распределения значений целевой функции

- А как распределены значения t? Пусть t распределено как y плюс гауссов шум. Тогда априорное распределение t:

$$p(\vec{t}) = N(0, Q + \sigma_\nu^2 I) = N(0, C).$$

- В случае фиксированного базиса

$$Q_{ij} = \sigma_w^2 [RR^t]_{ij} = \sigma_w^2 \sum_h \phi_h(\vec{x}^{(i)}) \phi_h(\vec{x}^{(j)}), \text{ и}$$

$$C_{ij} = \sigma_w^2 \sum_h \phi_h(\vec{x}^{(i)}) \phi_h(\vec{x}^{(j)}) + \delta_{ij} \sigma_\nu^2.$$

Пример: радиальный базис

- Рассмотрим базис

$$\phi_h(\vec{x}) = e^{-\frac{(\vec{x}-\vec{c}_h)^2}{2r^2}}, \quad h \in [h_{min}, h_{max}.$$

- Придётся положить $\sigma_w^2 = S/(\Delta H)$, чтобы при увеличении H ничего не портилось. Тогда

$$Q_{ij} = S\int_{h_{min}}^{h_{max}} \phi_h(x^{(i)})\phi_h(x^{(j)})dx = S\int_{h_{min}}^{h_{max}} e^{-\frac{(\vec{x}^{(i)}-\vec{c}_h)^2}{2r^2}} e^{-\frac{(\vec{x}^{(j)}-\vec{c}_h)^2}{2r^2}} dx$$

Пример: радиальный базис

- Если пределы интегрирования устремить в бесконечность, то можно взять интеграл:

$$Q_{ij} = \sqrt{\pi r^2} S e^{-\frac{(\vec{x}^{(i)} - \vec{x}^{(j)})^2}{4r^2}}.$$

- Отметим, что вся модель на самом деле свелась к функции ковариаций. Мы теперь можем для каждого набора точек подсчитать соответствующую матрицу ковариаций.

Сплайны

- *Сплайн* — это кусочно–полиномиальная функция.
- Т.е. у неё производные, начиная с какого-то номера, равны нулю, а последняя ненулевая производная может иметь разрывы.
- Мы будем искать непараметрическое решение в виде сплайна.

Целевая функция

- Чтобы оно в таком виде нашлось, будем искать $y(\vec{x})$, который минимизирует

$$M(y) = -\frac{1}{2}\beta \sum_{i=1}^{n} (y(\vec{x}^{(i)} - t^{(i)})^2 - \frac{1}{2}\alpha \int \left[y^{(p)}(x)\right]^2 dx.$$

- Тогда решение получится сплайном степени p.
- Как это записать в виде задачи байесовского вывода?

Решение в виде сплайнов как задача байесовского вывода

- Нужно просто задать соответствующее априорное распределение:

$$\log p(y(x)|\alpha) = -\frac{1}{2}\alpha \int \left[y^{(p)}(x)\right]^2 dx + \text{const}.$$

- Ну, и функцию правдоподобия тоже:

$$\log p(\vec{t}|y(x), \beta) = -\frac{1}{2}\beta \sum_{i=1}^{n} (y(\vec{x}^{(i)} - t^{(i)})^2 + \text{const}.$$

- Константы зависят от α и β (для нормировки).
- Теперь можно применять наши байесовские методы для поиска решений.

Сплайны как гауссовские процессы

- Здесь тоже гауссовский процесс. Фактически, гауссовский процесс — это такой процесс, который можно записать в виде

$$p(y(x)|\mu(x), C) = \frac{1}{Z} e^{-\frac{1}{2}(y(x)-\mu(x))^t C (y(x)-\mu(x))}.$$

- А наше априорное распределение можно записать как

$$\log p(y(x)|\alpha) = -\frac{1}{2}\alpha \int [D^p y(x)]^2 \, dx + \text{const} =$$

$$= -\frac{1}{2} y(x)^t ([D^p]^t D) y(x) + \text{const}.$$

Постановка задачи

- Мы до сих пор просто доказали, что некоторые известные (и неизвестные) нам методы — это гауссовские процессы.
- Это очень интересно, но мы не знаем пока, как использовать это знание, т.е. можно ли байесовский вывод обобщить на произвольный гауссовский процесс.
- Для этого нужно по заданной матрице (функции) ковариаций $C(\vec{x}, \vec{x}')$ оценить новое значение t_{n+1} по данным $\{t_1, \ldots, t_n\}$.

Вывод

- Всё просто. Поскольку $p(t_{n+1}, t_1, \ldots, t_n)$ — гауссиан, то условное распределение

$$p(t_{n+1}|t_1, \ldots, t_n) = \frac{p(t_{n+1}, t_1, \ldots, t_n)}{p(t_1, \ldots, t_n)}$$

  тоже гауссово.
- Как его подсчитать?

Матрица C

- Для $n+1$ точки матрица C_{n+1} имеет вид

$$C_{n+1} = \begin{pmatrix} C_n & \vec{k} \\ \vec{k}^t & \kappa \end{pmatrix}.$$

- Теперь мы уже можем подсчитать предсказание, просто обратив грубой силой матрицу C_{n+1}, потому что

$$p(t_{n+1}|t_1,\ldots,t_n) \approx e^{-\frac{1}{2}\begin{bmatrix} t_1 & \ldots & t_n & t_{n+1} \end{bmatrix} C_{n+1}^{-1} \begin{bmatrix} t_1 \\ \vdots \\ t_n \\ t_{n+1} \end{bmatrix}}.$$

Упражнение

Упражнение. Доказать, что

$$p(t_{n+1}|t_1,\ldots,t_n) = \frac{1}{Z} e^{-\frac{(t_{n+1}-\hat{t}_{n+1})^2}{2\hat{\sigma}^2_{n+1}}},$$

где

$$\hat{t}_{n+1} = \vec{k}^t C_n^{-1} \vec{t}_{1..n},$$
$$\hat{\sigma}^2_{n+1} = \kappa - \vec{k}^t C_n^{-1} \vec{k}.$$

Спасибо за внимание!

- Lecture notes, слайды и коды программ появятся на моей homepage:

  `http://logic.pdmi.ras.ru/~sergey/index.php?page=teaching`

- Присылайте любые замечания, коды программ на других языках, решения упражнений, новые численные примеры и прочее по адресам:
  `sergey@logic.pdmi.ras.ru`, `smartnik@inbox.ru`