

# Нейронные сети

Сергей Николенко

Академический Университет, 2012

# Outline

- 1 **Постановка задачи, разные сигмоиды**
  - Что мы уже знаем
  - Сети и функции ошибки
- 2 **Обучение и варианты**
  - Backpropagation
  - Пример: RankNet и LambdaRank

# Почему мы лучше?

- Компьютер считает быстрее человека
- Но гораздо хуже может:
  - понимать естественный язык
  - узнавать людей
  - обучаться в широком смысле этого слова
  - ...
- Почему так?

# Строение мозга

Как человек всего этого добивается?

- В мозге много нейронов
- Но цепочка нейронов, которые успевают поучаствовать в принятии решения, не может быть длиннее нескольких сот штук!
- Значит, мозг очень хорошо структурирован в этом смысле

# Искусственные нейронные сети

- Основная мысль позаимствована у природы: есть связанные между собой нейроны, которые передают друг другу сигналы.
- Есть нейронные сети, которые стараются максимально точно моделировать головной мозг; это уже не AI и не наша тема; мы будем рассматривать нейронные сети как аппарат машинного обучения, и только.
- John Denker: «neural networks are the second best way of doing just about anything».

# История ANN

- Warren McCulloch & Walter Pitts, 1943: идея.
- Marvin Minsky, 1951: первая реализация.
- 1960-е годы: долго изучали перцептрон, выяснили, что ничего существенного одним перцептроном не сделать.
- 1980-е: появились многоуровневые ANN, была разработана современная теория.

# Линейный перцептрон

- Мы уже умеем обучать линейный перцептрон. У него заданы:
  - $n$  весов  $w_1, w_2, \dots, w_n$ ;
  - лимит активации  $w_0$ ;
  - выход перцептрона  $o(x_1, \dots, x_n)$  вычисляется так:

$$o(x_1, \dots, x_n) = \begin{cases} 1, & \text{если } w_0 + w_1x_1 + \dots + w_nx_n > 0, \\ -1 & \text{в противном случае.} \end{cases}$$

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

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

# Правило обучения перцептрона

- Правило обучения перцептрона:

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

где:

- $t$  — значение целевой функции,
- $o$  — выход перцептрона,
- $\eta > 0$  — скорость обучения.



## Двухуровневые сети

- Мы уже давно и упорно изучаем линейные модели, основанные на комбинации базисных функций:

$$y(\mathbf{x}, \mathbf{w}) = f \left( \sum_{j=1}^M w_j \phi_j(\mathbf{x}) \right),$$

где  $f = \text{id}$  для задачи регрессии и, например,  $\sigma(x) = 1/(1 + e^{-x})$  для задачи классификации.

- Суть двухуровневых нейронных сетей – в том, что мы задаём  $\phi_j(\mathbf{x})$  в параметрической форме и пытаемся обучать эти параметры.

## Двухуровневые сети

- На первом уровне мы строим  $M$  линейных комбинаций входных переменных  $x_1, \dots, x_D$

$$a_j = \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} = \sum_{i=0}^D w_{ji}^{(1)} x_i \quad (x_0 = 1).$$

- Затем пропускаем их через дифференцируемую функцию активации:  $z_j = h(a_j)$ .
- На втором уровне – опять линейная комбинация

$$a_k = \sum_{j=1}^M w_{kj}^{(2)} z_j + w_{k0}^{(2)} = \sum_{j=0}^M w_{kj}^{(2)} z_j \quad (z_0 = 1).$$

- И выход – это  $y_k = a_k$  для регрессии,  $y_k = \sigma(a_k)$  для бинарной классификации (сразу несколько), и softmax  $y_k = \frac{\exp(a_k)}{\sum_s \exp(a_s)}$  для классификации на несколько классов.

## Двухуровневые сети

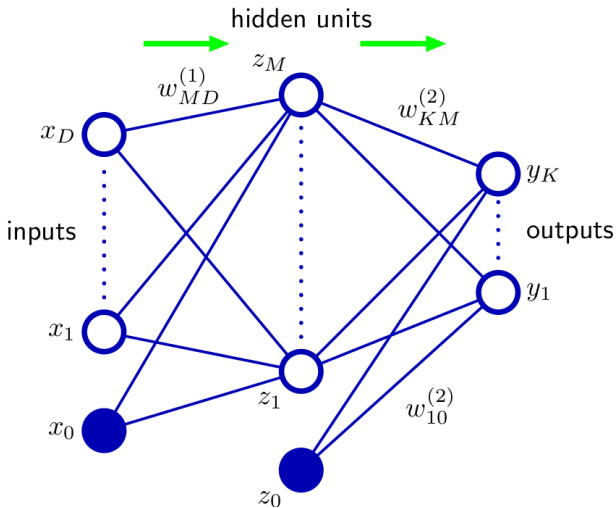
- Таким образом, если всё свернуть в одну функцию, получится

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left( \sum_{j=0}^M w_{kj}^{(2)} h \left( \sum_{i=0}^D w_{ji}^{(1)} x_i \right) \right),$$

и мы хотим найти оптимальные параметры этой функции.

- Это иногда называют *multilayer perceptron*, хотя здесь как раз важно, чтобы  $\sigma$  и  $h$  были дифференцируемы, а не ступенька, как в перцептроне.
- Конечно, нам никто не запрещает рассматривать и другие топологии (лишь бы циклов не было): больше уровней, skip-layer networks, в которых веса перепрыгивают через уровни, разреженные сети и т.д.

# Структура сети



# Функции ошибки

- Чтобы оптимизировать, надо определить функцию ошибки.
- Если есть датасет  $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N$  с правильными ответами  $\{\mathbf{t}_n\}$  (ответы теперь могут быть векторами), мы можем определить квадратическую ошибку

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2.$$

# Функции ошибки

- Мы уже знаем, как её мотивировать вероятностно: для регрессии с одним выходом  $t$  вводим предположение о нормально распределённом шуме

$$p(t | \mathbf{x}, \mathbf{w}) = \mathcal{N}(t | \mathbf{y}(\mathbf{x}, \mathbf{w}), \beta^{-1}),$$

предполагаем независимость тестовых примеров

$$p(\mathbf{t} | \mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^N p(t_n | \mathbf{x}_n, \mathbf{w})$$

и берём в качестве ошибки минус логарифм:

$$\frac{\beta}{2} \sum_{n=1}^N (\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n)^2 - \frac{N}{2} \ln \beta + \frac{N}{2} \ln(2\pi).$$

# Функции ошибки

- Тогда правдоподобие максимизируется там, где минимизируется квадратическая ошибка:

$$\mathbf{w}_{\text{ML}} = \arg \max_{\mathbf{w}} E(\mathbf{w}) = \arg \max_{\mathbf{w}} \sum_{n=1}^N (\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n)^2,$$

а потом можно и дисперсию максимального правдоподобия найти:

$$\frac{1}{\beta_{\text{ML}}} = \frac{1}{N} \sum_{n=1}^N (\mathbf{y}(\mathbf{x}_n, \mathbf{w}_{\text{ML}}) - \mathbf{t}_n)^2.$$

# Функции ошибки

- Для нескольких выходных переменных можно предположить, что они условно независимы при условии  $\mathbf{x}$ ,  $\mathbf{w}$  и оценить так же:

$$p(\mathbf{t} | \mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{t} | \mathbf{y}(\mathbf{x}, \mathbf{w}), \beta^{-1}\mathbf{I}),$$

$\mathbf{w}_{ML}$  и  $\beta_{ML}$  – как выше, только суммируем по всем выходам.



# Функции ошибки

- Для задачи бинарной классификации теперь  $y = \sigma(a) = \frac{1}{1 + \exp(-a)}$ .
- Мы интерпретируем  $y(\mathbf{x}, \mathbf{w})$  как оценку вероятности  $p(C_1 | \mathbf{x})$ , и получается распределение Бернулли

$$p(t | \mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t (1 - y(\mathbf{x}, \mathbf{w}))^{1-t}.$$

- Аналогично, берём минус логарифм, получаем функцию ошибки

$$E(\mathbf{w}) = - \sum_{n=1}^N (t_n \ln y_n + (1 - t_n) \ln(1 - y_n)).$$

- Это, кстати, и есть относительная энтропия двух распределений – из данных и модельного.

# Функции ошибки

- Для  $K$  отдельных задач бинарной классификации будет то же самое ( $y_{kn} = y_k(\mathbf{x}_n, \mathbf{w})$ ):

$$E(\mathbf{w}) = - \sum_{k=1}^K \sum_{n=1}^N (t_{kn} \ln y_{kn} + (1 - t_{kn}) \ln(1 - y_{kn})).$$

- И для классификации на несколько классов тоже всё понятно:

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))}.$$

- Однако, хотя функции ошибки различаются, у них всегда одни и те же  $a_k$ . Важное свойство:

$$\frac{\partial E}{\partial a_k} = y_k - t_k.$$

- Им мы будем активно пользоваться при обучении.

# Outline

- 1 Постановка задачи, разные сигмоиды
  - Что мы уже знаем
  - Сети и функции ошибки
- 2 Обучение и варианты
  - Backpropagation
  - Пример: RankNet и LambdaRank

# Обучение нейронных сетей

- Как обучать? Хочется сделать градиентный спуск по  $E(\mathbf{w})$ :

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}).$$

- Градиент считать дорого – надо суммировать по всем тестовым примерам.
- *Стохастический градиентный спуск* (stochastic gradient descent, online gradient descent): давайте менять веса после каждого примера:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}), \quad E(\mathbf{w}) = \sum_{n=1}^N E_n(\mathbf{w}).$$

- Это гораздо быстрее приведёт к цели, и может помочь выбраться из локальных минимумов.

# Обучение нейронных сетей

- Надо подсчитать  $\nabla E_n$ . В простой линейной модели  $y_k = \sum_i w_{ki} x_i$  и квадратической ошибкой  $E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2$  мы получим

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni}.$$

- Но при любой топологии один узел  $j$  вычисляет

$$a_j = \sum_i w_{ji} z_i,$$

а потом прогоняет через нелинейную функцию:

$$z_j = h(a_j).$$

# Обучение нейронных сетей

- Давайте потихоньку разворачивать:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i,$$

где  $\delta_j = \frac{\partial E_n}{\partial a_j}$  называются *ошибками*.

- Мы уже видели, что для выходов  $\delta_k = y_k - t_k$ .

# Обучение нейронных сетей

- А для скрытых уровней можно разворачивать дальше:

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j},$$

где сумма берётся по всем выходам узла  $j$ .

- Отсюда и получается формула *обратной пропагации ошибки* (backpropagation):

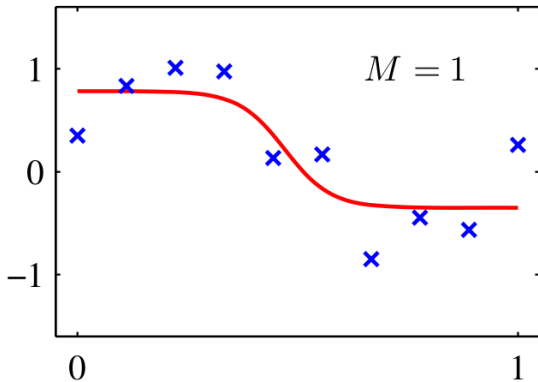
$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k.$$

# Регуляризация

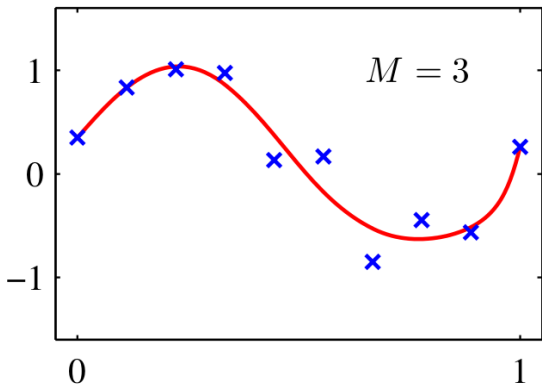
- В нейронных сетях тоже могут возникнуть проблемы с оверфиттингом.
- Дело в том, что мы сами выбираем число нейронов скрытого уровня, т.е. сложность модели.



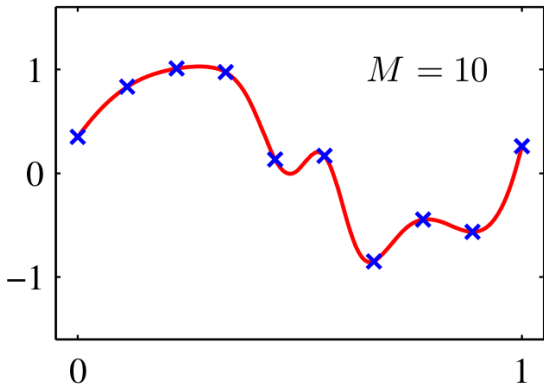
# Регуляризация



# Регуляризация



# Регуляризация



# Регуляризация

- Поэтому надо регуляризовать (в нейронных сетях это называется *weight decay*):

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}.$$

- Альтернатива, которой часто пользуются, – просто следить за ошибкой на валидационном наборе и останавливаться, когда ошибка увеличится.

# Байесовские нейронные сети

- А можно посмотреть на нейронные сети и нашим любимым байесовским способом:

$$p(t | \mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t | y(\mathbf{x}, \mathbf{w}), \beta^{-1}).$$

- Выберем априорное распределение:

$$p(\mathbf{w} | \alpha) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \alpha^{-1}\mathbf{I}).$$

- Построим функцию правдоподобия:

$$p(D | \mathbf{w}, \beta) = \prod_{n=1}^N \mathcal{N}(t_n | y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}).$$

# Байесовские нейронные сети

- Получим апостериорное распределение

$$p(\mathbf{w} \mid D, \alpha, \beta) \propto p(\mathbf{w} \mid \alpha) p(D \mid \mathbf{w}, \beta),$$

$$\ln p(\mathbf{w} \mid D, \alpha, \beta) = -\frac{\alpha}{2} \mathbf{w}^\top \mathbf{w} - \frac{\beta}{2} \sum_{n=1}^N (y(\mathbf{x}_n, \mathbf{w}) - t_n)^2 + \text{const.}$$

# Байесовские нейронные сети

- Тут можно (точно так же, с регуляризатором) найти максимум  $w_{\text{MAP}}$ , а потом приблизить в точке  $w_{\text{MAP}}$  гауссианом.
- Получится (точно так же, как когда-то уже получалось)

$$\mathbf{A} = -\nabla\nabla \ln p(\mathbf{w} \mid D, \alpha, \beta) = \alpha\mathbf{I} + \beta\mathbf{H},$$

где  $\mathbf{H}$  – гессиан, матрица вторых производных функции ошибки по компонентам  $\mathbf{w}$ .

- Значит, мы получили гауссовскую аппроксимацию в виде

$$q(\mathbf{w} \mid D) = \mathcal{N}(\mathbf{w} \mid \mathbf{w}_{\text{MAP}}, \mathbf{A}^{-1}).$$

# Байесовские нейронные сети

- Предсказательное распределение

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

тоже надо приближать, раскладывая в ряд Тейлора:

$$y(\mathbf{x}, \mathbf{w}) \approx y(\mathbf{x}, \mathbf{w}_{\text{MAP}}) + \mathbf{g}^\top (\mathbf{w} - \mathbf{w}_{\text{MAP}}) \text{ для}$$

$$\mathbf{g} = \nabla_{\mathbf{w}} y(\mathbf{x}, \mathbf{w}) |_{\mathbf{w}=\mathbf{w}_{\text{MAP}}} .$$

- Опять же, когда-то мы это уже считали, так что просто ответ:

$$p(t | \mathbf{x}, D, \alpha, \beta) = \mathcal{N}(t | y(\mathbf{x}, \mathbf{w}_{\text{MAP}}), \sigma^2(\mathbf{x})), \text{ где}$$
$$\sigma^2(\mathbf{x}) = \beta^{-1} + \mathbf{g}^\top \mathbf{A}^{-1} \mathbf{g}.$$



# Байесовские нейронные сети

- А можно и гиперпараметры оптимизировать:

$$p(D | \alpha, \beta) = \int p(D | \mathbf{w}, \beta) p(\mathbf{w} | \alpha) d\mathbf{w},$$

$$\ln p(D | \alpha, \beta) = -E(\mathbf{w}_{\text{MAP}}) - \frac{1}{2} \ln |\mathbf{A}| + \frac{W}{2} \ln \alpha + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi)$$

где  $W$  – общее число параметров в  $\mathbf{w}$ , а  $E(\mathbf{w}_{\text{MAP}})$  регуляризована:

$$E(\mathbf{w}_{\text{MAP}}) = \frac{\beta}{2} \sum_{n=1}^N (y(\mathbf{x}_n, \mathbf{w}_{\text{MAP}}) - t_n)^2 + \frac{\alpha}{2} \mathbf{w}_{\text{MAP}}^T \mathbf{w}_{\text{MAP}}.$$

# Байесовские нейронные сети

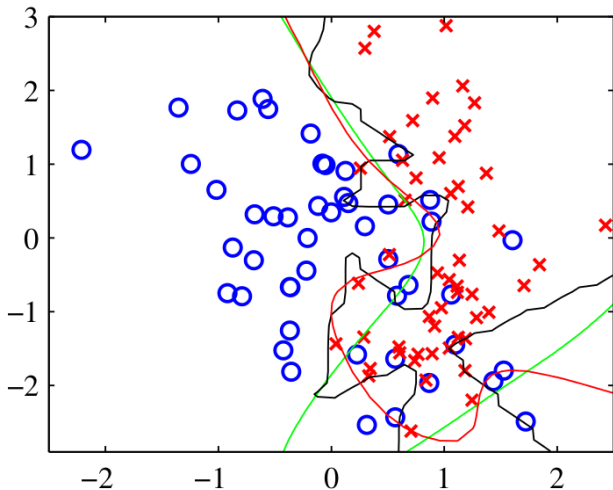
- Максимизировать по  $\alpha$  можно как в линейной регрессии: найти собственные числа  $\beta \mathbf{H} \mathbf{u}_i = \lambda_i \mathbf{u}_i$  и получить

$$\alpha = \frac{\gamma}{\mathbf{w}_{\text{MAP}}^\top \mathbf{w}_{\text{MAP}}}, \text{ где } \gamma = \sum_{i=1}^W \frac{\lambda_i}{\alpha + \lambda_i},$$

$$\frac{1}{\beta} = \frac{1}{N - \gamma} \sum_{n=1}^N (y(\mathbf{x}_n, \mathbf{w}_{\text{MAP}}) - t_n)^2.$$

- И можно запускать аналогичную итеративную процедуру: посчитали  $\mathbf{w}_{\text{MAP}}$ , переоценили  $\alpha, \beta$ , опять пересчитали  $\mathbf{w}_{\text{MAP}}$  и т.д.

# Вот что Байес животворящий делает



# RankNet

- Приведём пример применения нейронных сетей в information retrieval.
- Задача поиска: выдать ранжированный список наиболее релевантных документов по запросу.
- Пусть у нас есть кое-какие прямые данные для обучения (т.е. про некоторые подмножества документов эксперт сказал, какие более релевантны, какие менее).
- Подход к решению: давайте нейронной сети обучать функцию, которая по данному вектору атрибутов  $\mathbf{x} \in \mathbb{R}^n$  выдаёт  $f(\mathbf{x})$  и ранжирует документы по значению  $f(\mathbf{x})$ .

# RankNet

- Итак, для тестовых примеров  $x_i$  и  $x_j$  модель считает  $s_i = f(x_i)$  и  $s_j = f(x_j)$ , а затем оценивает

$$p_{ij} = p(x_i \triangleright x_j) = \frac{1}{1 + e^{-\sigma(s_i - s_j)}}.$$

- А данные – это на самом деле  $q(x_i \triangleright x_j)$ , либо точные из  $\{0, 1\}$ , либо усреднённые по нескольким экспертам.
- Поэтому разумная функция ошибки – кросс-энтропия

$$C = -q_{ij} \log p_{ij} - (1 - q_{ij}) \log(1 - p_{ij}).$$

# RankNet

- Ошибка:  $C = -q_{ij} \log p_{ij} - (1 - q_{ij}) \log(1 - p_{ij})$ .
- Для самого частого случая, когда оценки релевантности точные, и  $q_{ij} = (1 + S_{ij})/2$  для  $S_{ij} \in \{-1, 0, +1\}$ , мы получаем

$$C = \frac{1}{2}(1 - S_{ij})\sigma(s_i - s_j) + \log\left(1 + e^{-\sigma(s_i - s_j)}\right), \text{ т.е.}$$

$$C = \begin{cases} \log(1 + e^{-\sigma(s_i - s_j)}), & \text{если } S_{ij} = 1, \\ \log(1 + e^{-\sigma(s_j - s_i)}), & \text{если } S_{ij} = -1. \end{cases}$$

- Т.е. ошибка симметрична, что уже хороший знак.

# RankNet

- Ошибка:  $C = -q_{ij} \log p_{ij} - (1 - q_{ij}) \log(1 - p_{ij})$ .
- Давайте подсчитаем градиент по  $s_i$ :

$$\frac{\partial C}{\partial s_i} = \sigma \left( \frac{1 - S_{ij}}{2} - \frac{1}{1 + e^{\sigma(s_i - s_j)}} \right) = -\frac{\partial C}{\partial s_j}$$

- И теперь осталось использовать этот подсчёт для градиента по весам:

$$\frac{\partial C}{\partial w_k} = \sum_i \frac{\partial C}{\partial s_i} \frac{\partial s_i}{\partial w_k} + \sum_j \frac{\partial C}{\partial s_j} \frac{\partial s_j}{\partial w_k}$$

# RankNet

- Основной пафос RankNet – в том, что это можно факторизовать:

$$\frac{\partial C}{\partial w_k} = \sum_i \frac{\partial C}{\partial s_i} \frac{\partial s_i}{\partial w_k} + \sum_j \frac{\partial C}{\partial s_j} \frac{\partial s_j}{\partial w_k} = \lambda_{ij} \left( \frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k} \right),$$

где

$$\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = \sigma \left( \frac{1 - S_{ij}}{2} - \frac{1}{1 + e^{\sigma(s_i - s_j)}} \right).$$

- Переупорядочив пары так, чтобы всегда было  $x_i \triangleright x_j$  и  $S_{ij} = 1$ , получим

$$\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = -\sigma \frac{1}{1 + e^{\sigma(s_i - s_j)}}.$$



# RankNet

- $\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = -\sigma \frac{1}{1 + e^{\sigma(s_i - s_j)}}$ .
- Значит, если для данной выдачи есть множество пар  $I$ , в которых известно, что  $x_i \triangleright x_j$ ,  $(i, j) \in I$ , то суммарный апдейт для веса  $w_k$  будет

$$\Delta w_k = -\eta \left[ \sum_{(i,j) \in I} \lambda_{ij} \frac{\partial s_i}{\partial w_k} - \sum_{(i,j) \in I} \lambda_{ij} \frac{\partial s_j}{\partial w_k} \right] = -\eta \sum_i \lambda_i \frac{\partial s_i}{\partial w_k},$$

$$\text{где } \lambda_i = \sum_{j:(i,j) \in I} \lambda_{ij} - \sum_{j:(j,i) \in I} \lambda_{ij}.$$

- И можно просто считать  $\lambda_i$  по таким mini-batches от каждого запроса, а потом уже апдейтить.
- Иначе говоря,  $\lambda_i$  «тянет» ссылку в выдаче вверх или вниз, и мы апдейтим веса на основе этого.

# LambdaRank

- Казалось бы, это очень прямолинейный пример, ничего не происходит.
- Но в нём есть мощная идея, которая приводит к концепции LambdaRank: идея о том, что  $\lambda_{ij}$  можно отделить от сети и оценивать по парам тестовых примеров.
- Давайте начнём с того, что поставим типичную information retrieval задачу.

# LambdaRank

- В information retrieval есть разные интересные метрики:

- 1 WTA (winner takes all): 1, если первый документ релевантен, и 0, если нет;
- 2 DCG (discounted cumulative gain): для уровня релевантности  $l_i \in \{0, \dots, L\}$  и отсечки  $T$  (например,  $T = 10$ ),

$$DCG_T = \sum_{i=1}^T \frac{2^{l_i} - 1}{\log(1 + i)};$$

- 3 NDCG – нормализованный DCG (по максимально возможному для данного запроса):  $NDCG_T = \frac{DCG_T}{\max(DCG_T)}$ ;
- 4 ERR (expected reciprocal rank):

$$ERR = \sum_{r=1}^n \frac{1}{r} R_r \prod_{i=1}^{r-1} (1 - R_i), \quad R_i = \frac{2^{l_i} - 1}{2^L};$$

5 ...

- Как их оптимизировать?

# LambdaRank

- Мы хотели бы оптимизировать, скажем, NDCG, обучая ранжирующую функцию.
- Всё, что мы контролируем тут – это функция ошибки  $C$ .
- Значит, надо подобрать такую новую функцию ошибки  $C'$ , что она оптимизирует NDCG – совершенно негладкую дискретную функцию, зависящую от мест в выдаче.
- А ранжирует она по порядку значений  $s_i$ .
- Есть предложения о том, как такую придумать?

# LambdaRank

- Вот и у меня нету.
- Но тут приходит на помощь главная мысль LambdaRank: нам не надо задумываться о функции  $C'$ !
- Всё, что нам нужно для обучения, – это уметь считать градиенты

$$\lambda'_{ij} = \frac{\partial C'(s_i - s_j)}{\partial s_i}.$$

- Если мы зададим градиенты так, чтобы  $\frac{\partial^2 C'}{\partial s_i \partial s_j} = \frac{\partial^2 C'}{\partial s_j \partial s_i}$ , то такая функция  $C'$  будет существовать (лемма Пуанкаре); но нам не обязательно её строить явно.

# LambdaRank

- Ну так давайте попробуем добавить в наши градиенты  $\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = -\sigma \frac{1}{1 + e^{\sigma(s_i - s_j)}}$  разность  $\Delta_{\text{NDCG}}$ : то, насколько изменится NDCG, если поменять местами  $i$  и  $j$  в выдаче (типа «градиент» NDCG по  $s_i - s_j$ ):

$$\lambda'_{ij} = \frac{\partial C'(s_i - s_j)}{\partial s_i} = -\sigma \frac{1}{1 + e^{\sigma(s_i - s_j)}} |\Delta_{\text{NDCG}}|.$$

- И дальше будем делать точно такой же стохастический градиентный спуск с mini-batches, как в исходном RankNet.
- Естественно, вместо NDCG можно подставить что угодно.
- Это и есть LambdaRank – один из лучших инструментов для ранжирования выдачи, реально применяется.

Thank you!

**Спасибо за внимание!**