Нейронные сети. Case study: обучение ранжирований

Сергей Николенко

Центр Речевых Технологий, 2012

Центр Речевых Технологий

Outline

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 $w_1, w_2, \ldots, w_n$;
  - 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 $x_0 = 1$:

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

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.

Центр Речевых Технологий

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 $f = \mathrm{id}$ 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 $x_1, \ldots, x_D$

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

- 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.

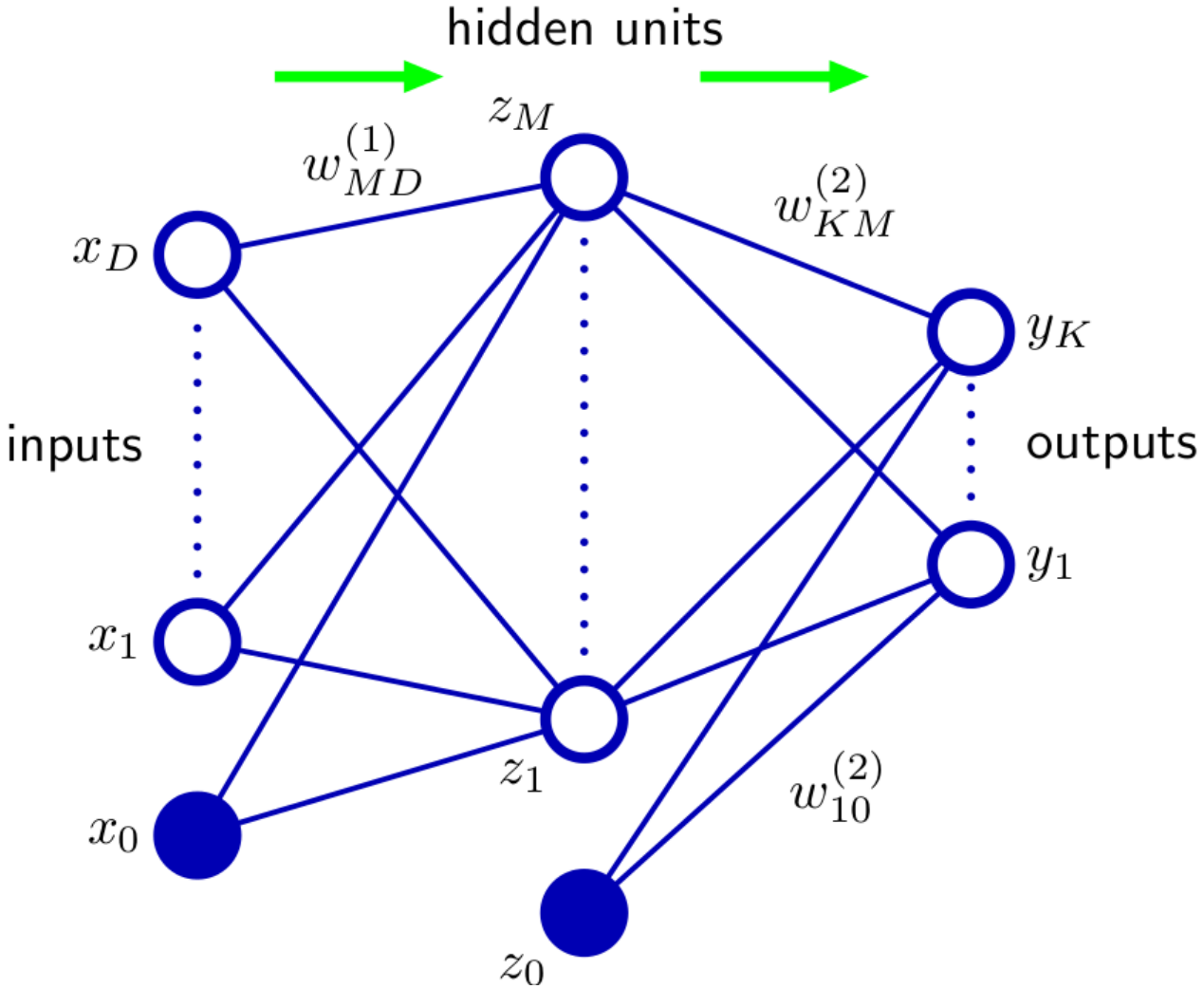
Двухуровневые сети

- 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 σ 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

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

Центр Речевых Технологий

Функции ошибки

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

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

предполагаем независимость тестовых примеров

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

и берём в качестве ошибки минус логарифм:

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

Центр Речевых Технологий

Функции ошибки

- Тогда правдоподобие максимизируется там, где минимизируется квадратическая ошибка:

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

а потом можно и дисперсию максимального правдоподобия найти:

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

Центр Речевых Технологий

Функции ошибки

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

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

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

Центр Речевых Технологий

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 $p(C_1 \mid \mathbf{x})$, и получается распределение Бернулли

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

- Аналогично, берём минус логарифм, получаем функцию ошибки

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

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

Функции ошибки

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

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

- 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.

Центр Речевых Технологий

Outline

Обучение нейронных сетей

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

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

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

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

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

Обучение нейронных сетей

- Надо подсчитать ∇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:

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

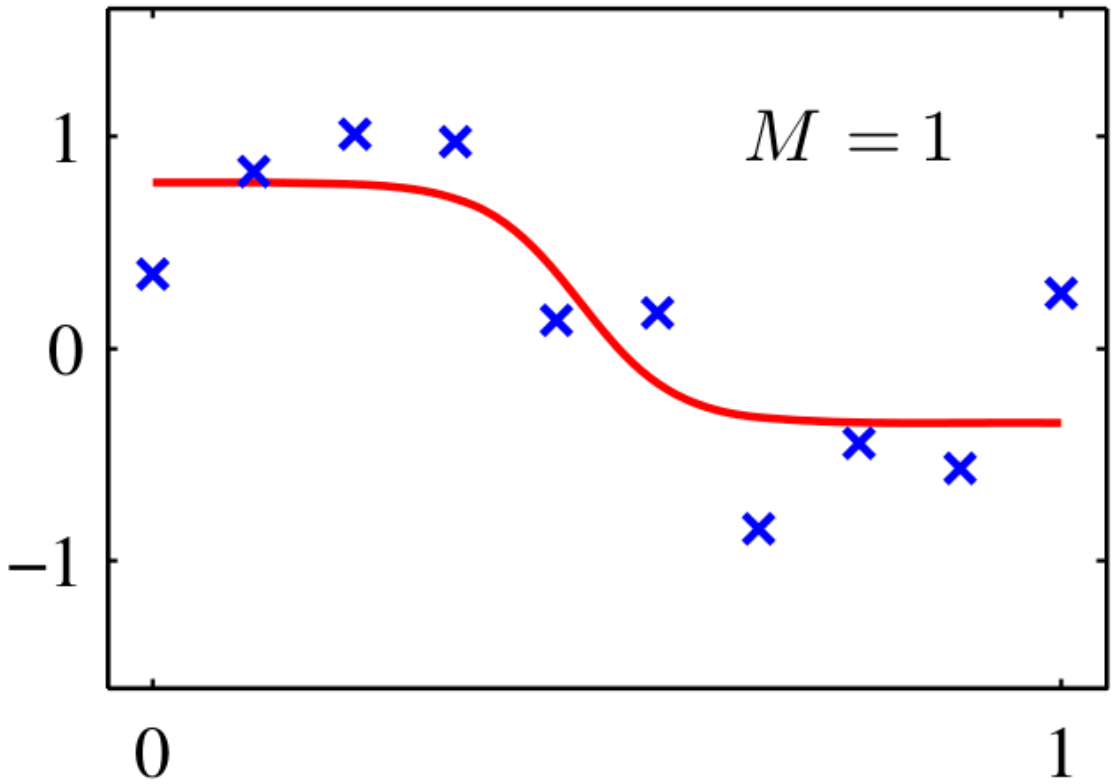
  где сумма берётся по всем выходам узла j.
- Отсюда и получается формула *обратной пропагации ошибки* (backpropagation):

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

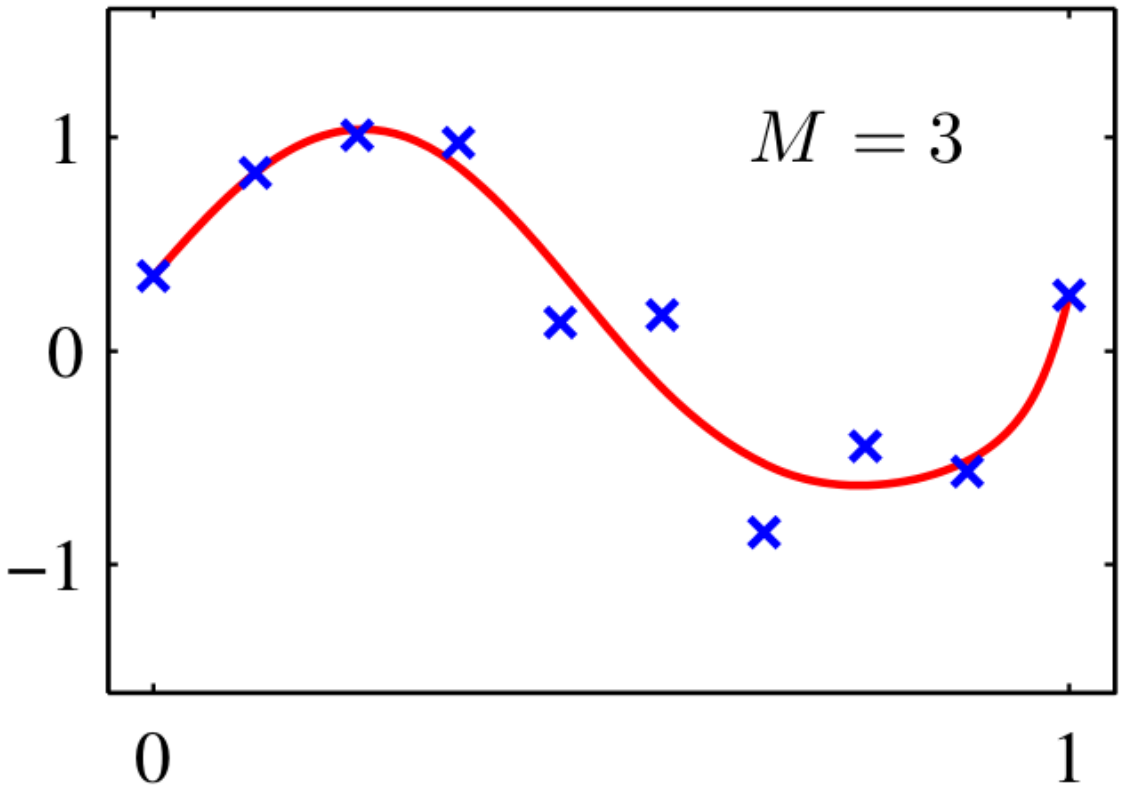
Регуляризация

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

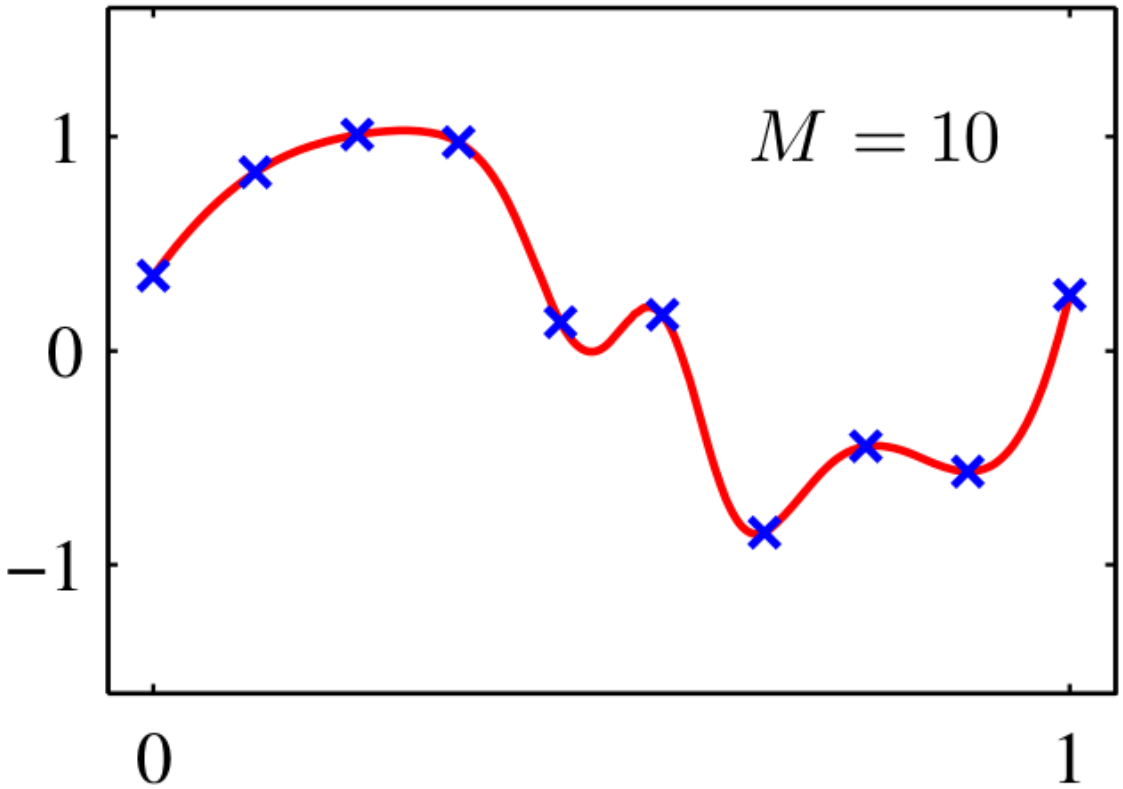
Регуляризация



Регуляризация



Регуляризация



Регуляризация

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

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

- 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:

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

- Выберем априорное распределение:

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

- Построим функцию правдоподобия:

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

Байесовские нейронные сети

- Получим апостеориорное распределение

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

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

Байесовские нейронные сети

- Тут можно (точно так же, с регуляризатором) найти максимум $\mathbf{w}_{\text{MAP}}$, а потом приблизить в точке $\mathbf{w}_{\text{MAP}}$ 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

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

Байесовские нейронные сети

- Предсказательное распределение

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

тоже надо приближать, раскладывая в ряд Тейлора:

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

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

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

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

Байесовские нейронные сети

- А можно и гиперпараметры оптимизировать:

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

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

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

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

Байесовские нейронные сети

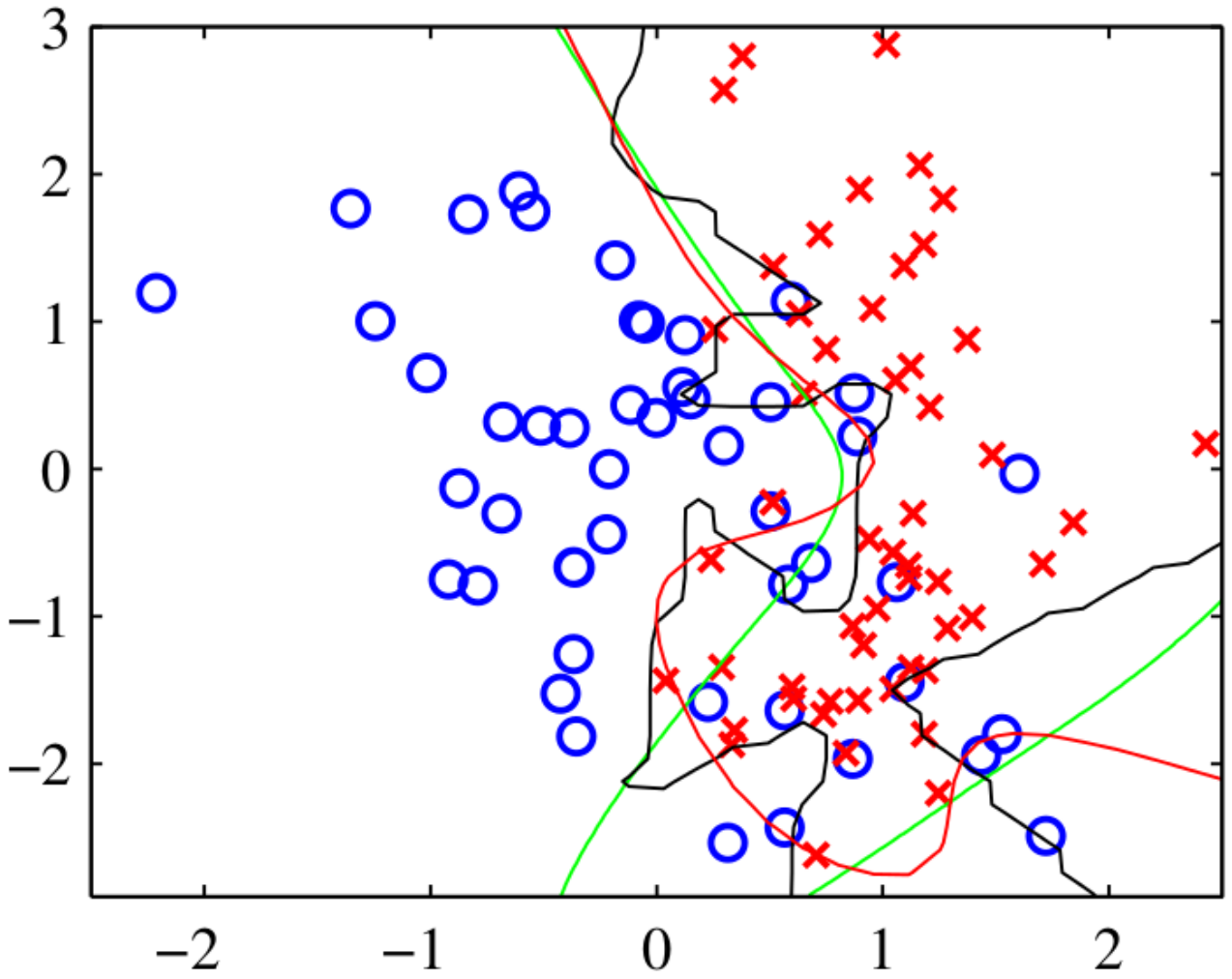
- Максимизировать по α можно как в линейной регрессии: найти собственные числа $\beta\mathbf{H}\mathbf{u}_i = \lambda_i\mathbf{u}_i$ и получить

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

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

- И можно запускать аналогичную итеративную процедуру: посчитали $\mathbf{w}_{\text{MAP}}$, переоценили α, β, 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 сетью 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 $\mathbf{x}_i$ и $\mathbf{x}_j$ модель считает $s_i = f(\mathbf{x}_i)$ и $s_j = f(\mathbf{x}_j)$, а затем оценивает

$$p_{ij} = p(\mathbf{x}_i \rhd x_j) = \frac{1}{1 + e^{-\sigma(s_i - s_j)}}.$$

- А данные – это на самом деле $q(\mathbf{x}_i \rhd x_j)$, 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

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

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

- Т.е. ошибка симметрична, что уже добрый 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 $\mathbf{x}_i \triangleright \mathbf{x}_j$ и $S_{ij} = 1$, получим

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

Центр Речевых Технологий

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 $\mathbf{x}_i \triangleright \mathbf{x}_j$, $(i, j) \in I$, то суммарный апдейт для веса w_k будет

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

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

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

Центр Речевых Технологий

Outline

Деревья регрессии

- Начнём с того, что вспомним, что такое деревья принятия решений в случае регрессии.
- Рассмотрим датасет $\mathbf{X} = \{\mathbf{x}_i, y_i\}$.
- Можно определить *regression stump* (регрессионный пень) так: выбираем одну координату (атрибут) j (т.е. берём x_{ij}) и ищем там оптимальное разбиение – такое значение порога t, что

$$S_j = \sum_{i \in \text{Left}} \left(y_i - \mu_{\text{Left}}\right)^2 + \sum_{i \in \text{Right}} \left(y_i - \mu_{\text{Right}}\right)^2$$

минимизируется, где Left и Right – множества точек слева и справа от t по j-й координате.

Деревья регрессии

- Если повторить эту процедуру L раз (как-то выбирая для расщепления листья – например, по максимальной дисперсии), получится регрессионное дерево с L листьями.
- В каждом листе определим γ_l – среднее по y_i из этого листа.
- Тогда, чтобы применить регрессионное дерево, надо по нему спуститься до листа и взять γ_l из этого листа.

MART

- MART – это бустинг, сделанный на регрессионных деревьях.
- Иначе говоря, окончательная модель будет, опять же, по $\mathbf{x} \in \mathbb{R}^d$ искать $y \in \mathbb{R}$, и искать в виде

$$F_M(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m f_m(\mathbf{x}),$$

где $f_m(\mathbf{x})$ задаётся регрессионным деревом, а $\alpha_m \in \mathbb{R}$ – веса бустинга, и в процессе обучения обучаются одновременно f_m и α_m.

MART

- Нам нужно понять, как обучать новое дерево F_{m+1}, если мы уже обучили m деревьев.
- Зафиксируем функцию ошибки C (она дана свыше).
- Идея: следующее дерево моделирует производные ошибки по текущей модели в точках из датасета:

$$\Delta C \approx \frac{\partial C(F_m)}{\partial F_m} \Delta F,$$

и ΔC будет отрицательным, если взять для $\eta > 0$

$$\Delta F = -\eta \frac{\partial C(F_m)}{\partial F_m}.$$

- Непараметрический метод: мы рассматриваем F_m *в каждой точке из датасета* как «параметр» и по ним оптимизируем.

Центр Речевых Технологий

MART

- Пример: бинарная классификация, $y_i \in \{\pm 1\}$, $\mathbf{x} \in \mathbb{R}^n$, $F(\mathbf{x})$.
- Обозначим $p_+ = p(y = 1 \mid \mathbf{x})$, $p_- = p(y = -1 \mid \mathbf{x})$, $I_+(\mathbf{x}_i) = [y_i = 1]$, $I_-(\mathbf{x}_i) = [y_i = -1]$.
- Будем использовать (как и в RankNet, кстати) перекрёстную энтропию

$$L(y, F) = -I_+ \log p_+ - I_i \log p_i.$$

MART

- Логистическая регрессия моделирует log odds:

$$F_N(\mathbf{x}) = \frac{1}{2}\log\left(\frac{p_+}{p_-}\right),$$

$$p_+ = \frac{1}{1+e^{-2\sigma F_m(\mathbf{x})}}, \quad p_- = \frac{1}{1+e^{2\sigma F_m(\mathbf{x})}},$$

и мы получаем

$$L(y, F) = \log\left(1 + e^{-2y\sigma F_m(\mathbf{x})}\right).$$

MART

- $L(y, F) = \log\left(1 + e^{-2y\sigma F_m(\mathbf{x})}\right)$.
- От этой функции легко взять производную по $F(\mathbf{x})$:

$$\bar{y}_i = \left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_m(\mathbf{x})} = \frac{2y_i\sigma}{1 + e^{2y_i\sigma F_m(\mathbf{x})}}.$$

- И мы строим новое регрессионное дерево, которое пытается смоделировать $\bar{y}_i$.

MART

- Осталось только выбрать вес, с которым это новое дерево войдёт в сумму.
- Мы хотим выбрать (примерно) оптимальный шаг для каждого листа, т.е. минимизировать потери:

$$\gamma_{lm} = \arg\min_\gamma \log\left(1 + e^{2y_i\sigma(F_m(\mathbf{x})+\gamma)}\right) = \arg\min_\gamma g(\gamma).$$

- Минимизировать можно итеративно по методу Ньютона-Рапсона: $\gamma_{n+1} = \gamma_n - \frac{g'(\gamma_n)}{g''(\gamma_n)}$.

Центр Речевых Технологий

MART

- И мы аппроксимируем одним шагом этого метода, начиная с нуля:

$$\gamma_{lm} = -\frac{g'}{g''} = \frac{\sum_{x_i \in R_{lm}} \bar{y}_i}{\sum_{x_i \in R_{lm}} |\bar{y}_i| \left(2\sigma - |\bar{y}_i|\right)}.$$

Упражнение. Проверьте эту формулу.

Центр Речевых Технологий

LambdaMART

- Теперь уже понятно, как из MART сделать LambdaMART.
- Мы просто добавим в градиенты целевую метрику, например

$$\lambda_{ij} = S_{ij} \left| \Delta \mathrm{NDCG} \frac{\partial C_{ij}}{\partial o_{ij}} \right|, \quad o_{ij} = F(x_i) - F(x_j).$$

- Функция ошибки нам тоже уже известна:

$$C_{ij} = C(o_{ij}) = s_j - s_i + \log\left(1 + e^{s_i - s_j}\right),$$
$$\frac{\partial C_{ij}}{\partial o_{ij}} = \frac{\partial C_{ij}}{\partial s_i} = -\frac{1}{1 + e^{o_{ij}}}.$$

Центр Речевых Технологий

LambdaMART

- Получается, что знак λ_{ij} зависит только от меток i и j, и в каждой точке мы можем собрать все «действующие силы» как

$$\lambda_i = \sum_j \lambda_{ij}.$$

- Это и называется LambdaMART.
- Вариант: LambdaSMART (submodel): мы инициализируем первое дерево какой-нибудь обученной хорошей базовой моделью, а всё дальнейшее – это её уточнение.

LambdaMART

- Остался только один вопрос: откуда взять веса α_i (при $f_i(\mathbf{x})$)?
- Базовый LambdaMART выбирает эти веса индивидуально для каждого листа дерева.
- Мы хотим сдвинуться в сторону минимума ошибки; значит, надо идти в сторону нуля производной. Один шаг метода Ньютона-Рапсона:

$$F_m(x_i) = F_{m-1}(x_i) + \upsilon \sum_l \gamma_{lm} \left[x_i \in R_{lm}\right],$$

где $\gamma_{lm} = \dfrac{F'_m(x_i)}{F''_m(x_i)}$, а υ – регуляризатор.

LambdaMART

- Первая производная – это просто λ_i; вторая производная – λ_i':

$$\gamma_{lm} = \frac{\sum_{x_i \in R_{lm}} \lambda_i}{\sum_{x_i \in R_{lm}} \frac{\partial \lambda_i}{\partial F_m(x_i)}}$$

- И теперь можно собрать весь алгоритм целиком.

LambdaSMART: алгоритм

1. $F_0(x_i) = \text{BaseModel}(x_i),\ i = 0..N.$
2. Для m от 1 до M (числа деревьев в сумме):
   1. $y_i = \lambda_i = \sum_j \lambda_{ij},\ i = 0..N$ (считаем градиенты);
   2. $w_i = \frac{\partial y_i}{\partial F(x_i)},\ i = 0..N$ (вторые производные);
   3. строим новое дерево $\{R_{lm}\}_{l=1}^{L}$ на вершинах $\{y_i, x_i\}_{n=1}^{N}$;
   4. $\gamma_{lm} = \frac{\sum_{x_i \in R_{lm}} y_i}{\sum_{x_i \in R_{lm}} w_i},\ l = 1..L$ (веса узлов);
   5. $F_m(x_i) = F_{m-1}(x_i) + v \sum_l \gamma_{lm} \left[x_i \in R_{lm}\right],\ i = 0..N.$

Как оптимизировать α_m

- В Lambda[S]MART веса бустинга подбираются как шаг метода Ньютона для каждого листа дерева.
- Но благодаря тому, что наши IR-метрики дискретные, можно просто подобрать оптимальный α_m.
- Давайте рассмотрим общую задачу: предположим, что у нас есть две ранжирующих функции, R и R', и мы хотим их оптимально линейно скомбинировать, т.е. подобрать оптимальный коэффициент α в

$$s_i = (1 - \alpha) s_i^R + \alpha s_i^{R'}.$$

Как оптимизировать α_m

- Идея простая: представьте себе, как α меняется от 0 (в чистом виде R) до 1 (в чистом виде R').
- Тогда метрики вроде NDCG реально меняются только в точках пересечения (да и то не всегда, а только если пересеклись документы с разными метками).
- Ну вот и давайте просто переберём все такие пары, найдём их точки пересечения и составим список интересных значений α.
- А затем отсортируем список, подсчитаем метрику в каждом интервале и найдём оптимальный α; для ситуации бустинга просто надо это делать на каждом шаге.
- Кажется, что это очень тупо, но на самом деле это квадратичный алгоритм, что не так уж страшно.

Thank you!

Спасибо за внимание!