Теормин по машинному обучению

Мстители ЭФ МГУ

Последнее обновление: 7 января 2023 г.

Содержание

1 Матричное дифференцирование. Определение $Df(x)[\Delta x]$. Связь $Df(x)[\Delta x]$ с градиентом ($\frac{\partial f}{\partial x}$) в случаях: 1) $f : \mathbb{R}^{n\times 1} \to \mathbb{R}$ 2) $f : \mathbb{R}^{m\times n} \to \mathbb{R}$.

$Df(x)[\Delta x]$ обозначают дифференциал функции f (линейную часть приращения функции). В свою очередь производная функция f в точке x_0 – линейный оператор $Df(x)$, который лучше всего аппроксимирует приращение функции:

$$f(x + \Delta x) - f(x) = Df(x)[\Delta x] + \mathcal{O}(\Delta x)$$

В случае 1) $f : \mathbb{R}^{n\times 1} \to \mathbb{R}$:

$$Df(x)[\Delta x] = \langle \nabla f(x), \Delta x \rangle, \text{ где } \underbrace{\nabla f(x) = \left(\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x), \ldots, \frac{\partial f}{\partial x_n}(x)\right)^T}_{\text{градиент функции – столбец частных производных}} \in \mathbb{R}^{n\times 1}$$

Вектор приращения Δx тоже вектор столбец $\in \mathbb{R}^{n\times 1}$. Теперь пару примеров:

- $f(x) = a^T \cdot x : \ Df(x)[\Delta x] = a^T \cdot \Delta x = \langle a, \Delta x \rangle, \ \nabla f(x) = a$
- $f(x) = x^T A x : \ Df(x)[\Delta x] = x^T (A + A^T)\Delta x = \langle (A + A^T)x, \Delta x \rangle = \langle \nabla f(x), \Delta x \rangle$
- $f(x) = \frac{1}{2}||Ax - b||_2^2 : \ Df(x)[\Delta x] = \langle \nabla f(x), \Delta x \rangle, \ \nabla f(x) = A^T \cdot (Ax - b)$

В случае 2) $f : \mathbb{R}^{m\times n} \to \mathbb{R}$:

$$Df(x)[\Delta x] = Tr(\langle \nabla f(x), \Delta x \rangle), \text{ где } \underbrace{\nabla f(x) = \left(\frac{\partial f}{\partial x_{ij}}(x)\right)_{i=1,j=1}^{m,n}}_{\text{матрица частных производных}} \in \mathbb{R}^{m\times n}, \ \Delta x \in \mathbb{R}^{m\times n}$$

Напоминание: скалярное произведение матриц: $\langle A, B \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{i,j} \cdot b_{i,j}$.

2 Градиентный спуск (GD), стохастический градиентный спуск (SGD), метод Ньютона.

Направление максимального роста функции задаётся градиентом
Направление максимального падения функции задаётся антиградиентом

1. Градиентный спуск:
$$w^{(t+1)} = w^{(t)} - \eta \nabla L(w^{(t)})$$
$\eta > 0$ – шаг/темп обучения, t – итерация, L – функция потерь (или любая другая функция), w – по какому набору (весам) оптимизируем, $-\nabla L(w^{(t)})$ – антиградиент функции потерь при наборе $w^{(t)}$ (Важно: сначала берётся антиградиент функции, потом в него подставляется набор $w^{(t)}$).

2. Стохастический градиентный спуск отличается от простого градиентного тем, что мы ищем градиент не для $L(w^{(t)}) = \sum\limits_{i=1}^{N} L_i(w^{(t)})$, а для $L_R(w^{(t)}) = \sum\limits_{i \in R} L_i(w^{(t)})$, где R – случайное подмножество элементов выборки, называемое minibatch:

$$w^{(t+1)} = w^{(t)} - \eta \sum_{i \in R} \nabla L_i(w^{(t)})$$

3. Метод Ньютона:

$$w^{(t+1)} = w^{(t)} - H_{(t)}^{-1} \nabla L(w^{(t)})$$

$H(L)$ – матрица Гессе для функции потерь:

$$H(L) = \frac{\partial}{\partial w} \left(\frac{\partial L(w)}{\partial w} \right)^T = \begin{bmatrix} \frac{\partial^2 L}{\partial w_1^2} & \frac{\partial^2 L}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 L}{\partial w_1 \partial w_d} \\ \frac{\partial^2 L}{\partial w_2 \partial w_1} & \frac{\partial^2 L}{\partial w_2^2} & \cdots & \frac{\partial^2 L}{\partial w_2 \partial w_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 L}{\partial w_d \partial w_1} & \frac{\partial^2 L}{\partial w_d \partial w_2} & \cdots & \frac{\partial^2 L}{\partial w_d^2} \end{bmatrix}$$

На примере множественной регрессии: модель $y = X\beta + \epsilon$, функция потерь:

$$L(\beta) = \sum_{i=1}^{N} (y_i - \langle x_i, \beta \rangle)^2 = (y - X\beta)^T (y - X\beta) = y^T y - 2y^T X\beta + \beta^T X^T X\beta$$

$$\nabla L(\beta) = -2X^T y + 2X^T X\beta$$

$$H = 2X^T X$$

1. GD: $\beta^{(t+1)} = \beta^{(t)} - \eta(-2X^T y + 2X^T X\beta^{(t)})$
2. SGD: случайно выбираем один элемент $i : (x_i, y_i)$ – minibatch из 1-го элемента,

$$L_i(\beta) = (y_i - \langle x_i, \beta \rangle)^2$$

$$\nabla L_i(\beta) = -2(y_i - \langle x_i, \beta \rangle) x_i$$

$$\beta^{(t+1)} = \beta^{(t)} - 2\eta(\langle x_i, \beta^{(t)} \rangle - y_i) x_i$$

3. Метод Ньютона: $\beta^{(t+1)} = \beta^{(t)} - \eta(2X^T X)^{-1}(-2X^T + 2X^T X\beta^{(t)})$

3 Настройка гиперпараметров с помощью кросс-валидации. K-fold crossvalidation, leave-one-out, кросс-валидация в работе с временными рядами.

Сначала стоит написать о видах настройки гиперпараметров:

1. grid-search: перебираем **все возможные** комбинации гиперпараметров из того пространства гиперпараметров, которые мы задали, – выбираем комбинацию, лучшую по качеству на кросс-валидации;

2. randomized-search: если пространство гиперпараметров слишком большое, стоит рассмотреть **случайные n** комбинаций – снова выбираем комбинацию, лучшую по качеству на кросс-валидации;

3. байесовская оптимизация: в данном случае предполагается наличие функции качества (на кросс-валидации) от гиперпараметров и алгоритм пытается найти её оптимум.

Теперь к видам кросс-валидации:

1. **k-fold cross validation:** выборка делится на k фолдов (обычно, имеются в виду непересекающиеся подвыборки). Алгоритм настраивается на $(k-1)$ фолдах и оценивается на оставшимся фолде. Получается лист (массив) из k оценок качества, который обычно усредняется;

2. **stratified k-fold validation:** алгоритм аналогичен предыдущему, только деление на фолды происходит с условием, что распределение классов в подвыборке соответствует распределению по всей выборке (данный способ полезен при дисбалансе классов для корректной валидации);

3. **leave-one-out:** предельный случай k-fold crossvalidation, где k равно числу наблюдений в располагаемой выборке. Алгоритм оценивается на $(n-1)$ наблюдениях и оценивается качество на одном оставшимся. Итоговое качество считается как среднее из n оценок;

4. **time-series cross validation:** заглядывать в будущее кажется плохой идеей, поэтому предыдущие способы не годятся. Действуем теперь так: опять делим на k фолдов, только теперь деление не случайное, а по времени. Условно, в $(k+1)$-ом фолде будут присутствовать более поздние по времени наблюдения по сравнению с k-м. Модель оценивается на k первых фолдах и оценивается на $(k+1)$-ом ($\{1\}$ и $\{2\}, \{1,2\}$ и $\{3\}, \ldots, \{1,\ldots,k\}$ и $\{k+1\}$), и снова массив оценок усредняется.

4 Формула классификации и регрессии в модели KNN. Примеры весов. Примеры метрик.

Алгоритм K-nearest neighbours является метрическим алгоритмом для классификации и регрессии, принимающий решение на основе найденных K-ближайших соседей:

- в случае многоклассовой классификации: $y_{pred} = \underset{c \in C}{\operatorname{argmax}} \sum_{k=1}^{K} [y_k = c] \cdot w_k$;
- в случае регрессии: $y_{pred} = \sum_{k=1}^{K} y_k \cdot w_k$

$\{y_k \mid k = 1, \ldots, K\}$ – ближайшие соседи с соответствующими им весами w_k.

Веса в обоих случаях уже нормированы ($\sum_{k=1}^{K} w_k = 1$). Какие могут быть веса?

- $\tilde{w}_i = \left(\frac{d-i+1}{d}\right)^{\delta},\ i = \{1, \ldots, d\},\ \delta \in [0, +\infty)$;

- $\tilde{w}_i = \lambda^i,\ i = \{1, \dots, d\},\ \lambda \in (0, 1]$;
- $\tilde{w}_i = \frac{1}{\gamma^i},\ i = \{1, \dots, d\},\ \gamma \in [1, +\infty)$;
- любые другие «осмысленные» вариации (вряд ли стоит давать больший вес более далёким наблюдениям), которые потом будут нормированы: $w_i = \frac{\tilde{w}_i}{\sum_{i=1}^{d} \tilde{w}_i}$.

Как находить ближайших соседей? По вычислению расстояния между наблюдениями на основе имеющихся характеристик. Примеры метрик расстояния:

1. $\rho_2(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$ – Евклидово расстояние;
2. $\rho_1(x, y) = \sum_{i=1}^{n} |x_i - y_i|$ – Манхэттенское расстояние;
3. $\rho_\infty(x, y) = \max\limits_{i \in 1,\dots,n} |x_i - y_i|$ – Чебышёва расстояние;
4. для отличников: $\rho_p(x, y) = (\sum_{i=1}^{n}(x_i - y_i)^p)^{\frac{1}{p}}$ – расстояние Минковского при $p \geq 1$ является метрикой, при $p < 1$ нарушается нер-во треугольника.

Для категориальных признаков можно поступать как с численными, предварительно применив кодирование: если в признаке есть осмысленный порядок (например, между бакалавром и магистром расстояние ближе, чем между бакалавром и PhDшником), стоит применить ordinal encoder. Если нет определённого порядка, one-hot-encoder выглядит предпочтительнее. Второй метод их анализа описан чуть ниже и не требует применять какое-либо кодирование, просто работа с множествами.

Есть ряд специфичных метрик:

- $p(X, Y) = 1 - \dfrac{|X \cap Y|}{|X \cup Y|}$ – расстояние Джаккарда (на множествах, удобно использовать в случае наличия только категориальных признаков);
- $p(x, y) = 1 - \dfrac{\langle x, y \rangle}{||x|| \cdot ||y||}$ – косинусное расстояние (обычно используется для сравнительной оценки схожести какой-либо пары текстов);
- $p(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)}$ – расстояние Махалонобиса (если рассмотрим диагональную ковариационную матрицу, увидим автоматическую стандартизацию по признакам, которую, к слову, полезно делать при использовании метрических подходов, чтобы учитывать каждый признак в равной степени).

5 Определение линейного классификатора. Понятие отступа (Margin). Примеры верхних оценок на долю ошибок. Как обучаются веса?

- Линейный классификатор: $a(x) = sign(\langle w, x \rangle + b)$

- Отступ (margin) на i-ом объекте выборки: $M_i = y_i\langle w, x_i\rangle$. Отрицательное значение отступа свидетельствует об ошибочной классификации, положительное – о верной. Чем больше абсолютное значение отступа, тем более уверенная классификация с точки зрения нашего алгоритма.

- Изначально имеем такую задачу:

$$Q(a,X) = \frac{1}{l}\sum_{i=1}^{l}[a(x_i) \neq y_i] = \frac{1}{l}\sum_{i=1}^{l}[sign(\langle w, x_i\rangle) \neq y_i]$$
$$= \frac{1}{l}\sum_{i=1}^{l}[y_i sign(\langle w, x_i\rangle) \neq 1] = \frac{1}{l}\sum_{i=1}^{l}[M_i < 0] \to \min_w$$

  Однако, данный функционал от M_i имеет ступенчатый график и не является дифференцируемым. Поэтому оценим $L(M) = [M < 0]$ сверху так, что $L(M) \leq \hat{L}(M)$ и $\hat{L}(M)$ – гладкая. Минимизируя $\frac{1}{l}\sum\limits_{i=1}^{l}\hat{L}(y_i\langle w, x_i\rangle)$ мы добиваемся того, что исходный функционал также минимизируется. В качестве верхних оценок можно использовать:

  1. $\hat{L}(M) = ln(1 + e^{-M})$ – логистическая функция потерь
  2. $\hat{L}(M) = max(0, 1 - M)$ – кусочно-линейная функция потерь (hinge loss из SVM)
  3. $\hat{L}(M) = e^{-M}$ – экспоненциальная функция потерь (встречалась в AdaBoost)
  4. $\hat{L}(M) = max(0, -M)$
  5. $\hat{L}(M) = \frac{2}{1+e^M}$ – сигмоидная функция потерь

- Веса обучаются, когда мы минимизируем сумму верхних оценок. Если нет аналитического решения используются градиентные методы.

6 Подходы one-vs-one и one-vs-rest для линейных классификаторов.

- Подход **one-vs-one**

  One-vs-one – метод, используемый для многоклассовой классификации, заключающийся в разбиении k-мерного множества на $\frac{k(k-1)}{2}$ задач бинарной классификации. На примере задачи разбиения множества на 4 класса, приходится решать 6 задач попарного разделения на класса (пары классов каждый с каждым). Ещё пример (чуть более наглядный): задача присвоения классов «красный», «синий», «зеленый» разбивается на три задачи:

  1. Бинарная классификация «красный», «синий»
  2. Бинарная классификации «красный», «зеленый»
  3. Бинарная классификация «зеленый», «синий»

Есть несколько вариантов принятия решения:

- для каждой пары классов для объекта делается выбор класса, на их основе (классов «победителей») берётся argmax (самый частотный класс из победителей);
- можно выбрать класс на основе самого уверенного прогноза (выбираем класс, который имеют наибольшую вероятность из всех пар);
- можно усреднить вероятность принадлежности к классу по количеству пар, в которых они участвовали, и выбрать класс с наибольшим средним.

- Подход **one-vs-rest** (также называется **one-vs-all**)

  One-vs-rest или One-vs-all – также метод, используемый для многоклассовой классификации. Подход заключается в решении k отдельных задач бинарной классификации. Задачей каждого из них будет отделение данного класса от всех остальных. Ещё пример: задача присвоения классов «красный», «синий», «зеленый» разбивается на три задачи:

  1. Бинарная классификация «красный», «некрасный»
  2. Бинарная классификации «синий», «несиний»
  3. Бинарная классификация «зеленый», «незеленый»

  После обучения на каждой такой задаче, класс выбирается на основе наибольшей уверенности в классе (в примере $k = 3$):

  $$a(x) = \underset{i \in \{1,2,\ldots,k\}}{\operatorname{argmax}} a_i(x)$$

  Проблема метода заключается в значительном дисбалансе классов, которая только усугубляется с ростом числа классов и объема множества. Таким образом, приходится проводить дополнительную работу с калибровкой весов.

7 Переобучение. Способы борьбы с переобучением.

Переобучение - ситуация, когда алгоритм начинает находить в тренеровочной выборке закономерности, которых нет в остальном датасете. То есть хорошо работает на обучающей выборке и плохо на тестовой. Обнаружить переобучение можно с помощью кроссвалидации. Способы борьбы с переобучением

1. Регуляризация
2. Ранняя остановка (например в градиентном бустинге)
3. dropout (для нейронных сетей)

8 Проклятье размерности.

Под «проклятием размерности» подразумевается разного рода явления и эффекты, происходящие в высокоразмерных пространствах (в нашем случае при большом количестве признаков).

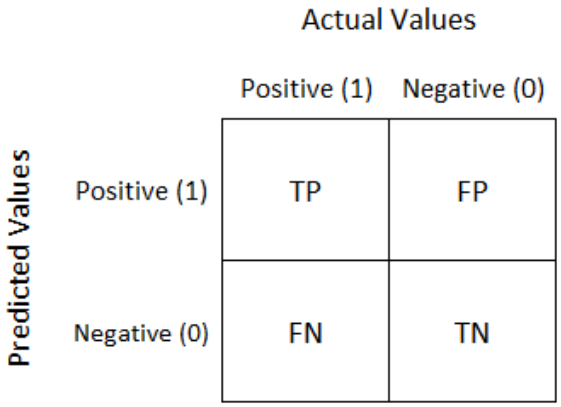
Если используемая метрика основана на суммировании различий по всем признакам, а число признаков очень велико, то все точки выборки могут оказаться практически одинаково далеки друг от друга.

Ещё один пример: рассмотрим вектора $(x_1, x_2, \ldots, x_n)$, $(x_1 + \epsilon, x_2 + \epsilon, \ldots, x_n + \epsilon)$ и $(x_1, x_2 + \Delta, \ldots, x_n)$, где ϵ – небольшой шум, а Δ – большая константа. При большой размерности данных второй вектор может быть дальше от первого, чем третий вектор, что кажется не желаемым эффектом.

И заключительный: невозможность эффективного поиска ближайших соседей для заданной точки (показано, что сложность всех популярных методов решения этой задачи становится линейной по размеру выборки по мере роста размерности).

9 Матрица ошибок (confusion matrix). Определение accuracy, precision, recall, f1-measure.

Матрица ошибок – матрица, размера KxK, где K – количество классов, которая показывает точность классификации модели. В случае двух классов:



Многосклассовый случай. Пусть есть выборка X размера n, y_i – метки класса для каждого объекта i (всего K-штук классов). Матрицей ошибок называется следующая матрица:

$$M = m_{jk}, \text{ размерности KxK}$$

$$m_{jk} = \sum_{i=0}^{N} [a(x_i) = j][y_i = k], \text{ где j – класс, предсказанный классификатором.}$$

Accuracy – доля правильных ответов алгоритма.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$Precision$, $recall$, $f1-measure$ рассчитываются для каждого отдельного класса.

$Recall$ – какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм (должно быть как можно выше).

$$Recall = \frac{TP}{TP + FN}$$

$Precision$ – долю объектов, названных классификатором положительными и при этом действительно являющимися положительными (должно быть как можно выше).

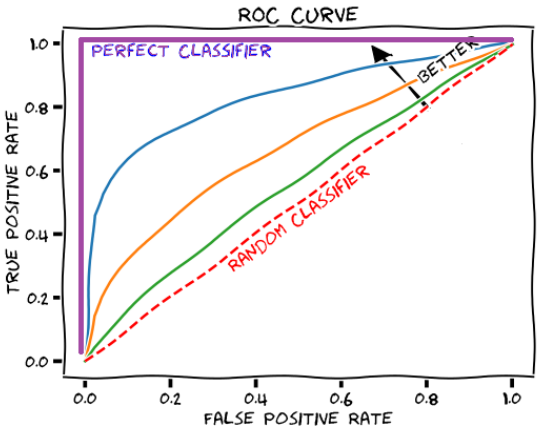
$$Precision = \frac{TP}{TP + FP}$$

Хороший классификатор имеет обе метрики (precision и recall) высокими. При этом ориентируясь только на одну из них качество второй может падать (но не обязательно). Чтобы оптимизировать одновременно обе величины используют f1-меру, которая включает обе метрики и достигает максимума при при полноте (recall) и точности (precision), равными единице. $f1-measure$ – среднее гармоническое precision и recall с весом точности = 1 (должно быть как можно выше). f-мера близка к нулю, если один из аргументов близок к нулю.

$$f1-measure = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

10 Рок кривая, AUC-ROC.

ROC кривая – кривая зависимости $False\ Positive\ Rate\ = \frac{FP}{FP + TN}$ и $True\ Positive\ Rate\ = \frac{TP}{TP + FN}$. AUC ROC – площадь под этой кривой, значение характеризует долю правильно классифицируемых пар. Для ее построения ранжируем вероятность присвоения класса ($b(x)$), для каждого порога считаем FPR и TPR.



$$AUC = \frac{\sum_i \sum_j [y_i < y_j] \times [a(x_i) < a(x_j)]}{\sum_i \sum_j [y_i < y_j]} = \frac{1}{n_- \times n_+} \times \sum_{i<j} [y_i < y_j]$$

ROC может пойти по диагонали (не ступенькой), если у одинаковых наблюдений будут разные метки. Самый плохой случай AUC: AUC = 0.5, т.к. по сути ничего не упорядочили и угадываем в 50% случаев. Если AUC ROC = 0, то значит, что присвоенные метки полностью перепутаны.

Критерий AUC-ROC имеет большое число интерпретаций: например, он равен вероятности того, что случайно выбранный положительный объект окажется позже случайно выбранного отрицательного объекта в ранжированном списке, порожденном $b(x)$.

11 Определение решающего дерева. Критерии расщепления в случае задачи классификации и регрессии.

Рассмотрим бинарное дерево, в котором:

- каждой внутренней вершине v приписана функция (предикат) $\beta_v : \mathbb{X} \to \{0, 1\}$, обычно $\beta_v(x; j,t) = [x_j < t]$ – бинарное правило;
- каждой листовой вершине v приписан прогноз $c_v \in Y$ (в случае с классификацией листу также может быть приписан вектор вероятностей).

Рассмотрим теперь алгоритм $a(x)$, который стартует из корневой вершины v_0 и вычисляет значение функции β_{v_0}. Если оно равно нулю, то алгоритм переходит в левую дочернюю вершину, иначе в правую, вычисляет значение предиката в новой вершине и делает переход или влево, или вправо. Процесс продолжается, пока не будет достигнута листовая вершина; алгоритм возвращает тот класс, который приписан этой вершине. Такой алгоритм называется **бинарным решающим деревом**.

Критерий расщепления:

$$Q(R, \theta) = H(R) - \frac{|R_{left}|}{|R|} \cdot H(R_{left}) - \frac{|R_{right}|}{|R|} \cdot H(R_{right}), \text{ где меры неоднородности}$$

- для классификации:
  1. $H(R) = \sum_{k=1}^{K} p_k \cdot (1 - p_k)$ – критерий Джини
  2. $H(R) = -\sum_{k=1}^{K} p_k \cdot \log p_k$ – энтропийный критерий
  3. $H(R) = 1 - \max\limits_{1,\dots,K} p_k$ – miss-classification criteria
- для регрессии:
  1. $H(R) = \frac{1}{|R|} \sum_{(x_i,y_i)\in R} \left(y_i - \frac{1}{|R|} \sum\limits_{(x_j,y_j)\in R} y_j \right)^2$ – MSE
  2. $H(R) = \frac{1}{|R|} \sum_{(x_i,y_i)\in R} \left| y_i - \underset{(x_j,y_j)\in R}{Median}\ y_j \right|$ – MAE

Напоминание: параметр θ обозначает рассматриваемое правило для ветвления.

12 Постановка задачи PCA. Как выбирать оптимальную размерность маломерного пространства?

Суть алгоритма в том, что мы хотим перейти из пространства размерности D в пространство размерности d ($D > d$)

Постановка задачи:

$$\begin{cases} \sum\limits_{k=1}^{d} a_k^T S a_k \to \max\limits_{a_1,\ldots,a_d} \\ ||a_k||^2 = 1 \\ \langle a_l, a_m \rangle = 0 \quad l, m : l \neq m, \quad l \in \{1,\ldots,d\}, \quad m \in \{1,\ldots,d\} \end{cases}$$

Где $S = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T$, а x_i – это вектор столбец длины D

Эквивалентно d задачам:

$$\begin{cases} a_k^T S a_k \to \max\limits_{a_k} \\ ||a_k||^2 = 1 \end{cases}$$

Эквивалентность получается, так как для максимизации суммы необходима максимизация каждого слагаемого, а условие на ортогональность отпадает (почему написано чуть далее).

Ортогональность отпадает, так как нахождение максимума соответсвует процессу нахождения d собственных векторов с максимальными собственными значениями, а так как матрица симметричная, то она является самосопряжённым оператором, и его собственные вектора являются ортогональными, поэтому это условие всегда выполняется, а потому не нужно.

Есть два способа определения размерности (размерность соответсвует количеству собственных значений):

1) **Внутренний:** изобразить график состоящий из собственных значений и брать те собственные значения, которые находятся до резкого падения собственных значений (на графике выглядит, как резкое падение). Альтернативный способ брать отношение накопленной суммы первых i максимальных собственных значений к сумме всех собственных значений и смотреть, сколько процентов вариации они объясняют, например, если 3 объясняют 80% вариации, то можно взять как раз три.

2) **Внешний:** следует из каких-то наших интересов, например, мы хотим с его помощью что-то визуализировать, тогда размерность не больше 3 (обычно 2), или мы хотим всё в один общий показатель агрегировать и т.д.

13 Постановка задачи кластеризации. Алгоритм K-means.

Задача кластеризации: выявить в данных K кластеров – областей, где объекты внутри одного кластера похожи друг на друга, а объекты из разных кластеров друг на

друга не похожи. Похожесть объектов определяется расстоянием между этими объектами, $\rho(x_i, x_j)$, для этого могут быть использованы различные метрики: Евклидова (Минковского - общий вариант), Манхэттена, косинусная (для текстов), Хэмминга и другие.

Построить алгоритм $a : X \to \{1, ...K\}$, определяющий для каждого объекта номер его кластера

K-means

$$Q = \sum_{k=1}^{K} \sum_{i:x_i \in A_k} ||x_i - c_k||^2 \to \min_{c_1,...c_K}$$

Хотим подобрать $c_1, ...c_K$ - центры кластеров, такие что расстояния до $x_i \in A_k$, где $A_k = \{x_i : \rho(x_i, c_k) \leqslant \rho(x_i, c_j) \forall j \neq k\}$ было как можно меньше.

Алгоритм такой (Решение задачи происходит в повторении шагов 1 и 2):

0. Выбор произвольных центров и множеств кластеров

1. Оптимизация функционала по A_k

2. Выбор новых центров c_k, соответсвующих кластерам A_k

Повторяя эти шаги до **сходимости** (пока центры станут неподвижными = нулевое изменение внутриклассового расстояния), мы получим некоторое распределение объектов по кластерам. Новый объект относится к тому кластеру, чей центр является ближайшим.

14 Сингулярное разложение (SVD) определение, его связь с PCA.

Сингулярным разложением матрицы называется такое разложение:

$$\underset{n\times m}{X} = \underset{n\times n}{U} \cdot \underset{n\times m}{\Sigma} \cdot \underset{m\times m}{V^T}, \text{ где}$$

- Σ – матрица с неотрицательными элементами, у которой элементы, лежащие на главной диагонали – сингулярные числа (а все элементы, не лежащие на главной диагонали, являются нулевыми)

- матрицы U и V – две унитарные ($M^TM = MM^T = I$) матрицы, состоящие из левых и правых сингулярных векторов соответственно (то есть собственных векторов матриц XX^T и X^TX соответственно)

Теперь покажем связь с PCA:

$$X^T \cdot X = (U \cdot \Sigma \cdot V^T)^T \cdot U \cdot \Sigma \cdot V^T = V \cdot \Sigma^T \cdot U^T \cdot U \cdot \Sigma \cdot V^T = V \cdot \Sigma^T \cdot \Sigma \cdot V^T$$

$$\frac{1}{n-1} \cdot X^T \cdot X = \frac{1}{n-1} \cdot V \cdot \Sigma^T \cdot \Sigma \cdot V^T \iff S = V \cdot \frac{\Sigma^T \cdot \Sigma}{n-1} \cdot V^T$$

Результатом перемножения $\Sigma^T \cdot \Sigma$ будет квадратная матрица, на главной диагонали которой будут стоять соответствующие квадраты сингулярных значений, которые стояли на главной диагонали Σ. Мы получили, такое соотношение между сингулярным значением σ_i и собственным значением ковариационной матрицы λ_i:

$$\lambda_i = \frac{\sigma_i^2}{n-1}, \ \sigma_i = \sqrt{(n-1) \cdot \lambda_i}$$

В свою очередь главными компонентами являются собственные векторы ковариационной матрицы – векторы-столбцы матрицы V.

15 l1, l2 регуляризация в задаче линейной регрессии и классификации. Какая из них позволяет отбирать признаки и почему?

Регуляризация является методом добавления ограничения на коэффициенты / параметры модели (w), что эквивалентно добавлению в оптимизируемый функционал штрафа за большие значения коэффициентов. Есть два основных вида регуляризации:

- l_1-регуляризация: $\sum_{j=1}^{d} |w_j|$ (соответствующая регрессия называется lasso regression)
- l_2-регуляризация: $\sum_{j=1}^{d} w_j^2$ (соответствующая регрессия называется ridge regression)

Стандартно они также домножаются на коэффициент λ, контролирующий степень регуляризации.

Отбирать признаки позволяет l_1-регуляризация из-за угловатости области допустимых значений в случае её применения (рис. 1), минимум функционала намного вероятнее попадёт в углы квадрата, соответствующие занулению одного из весов (эффект становится ешё сильнее/вероятнее при большем количестве признаков):

16 Постановка задачи обучения логистической регрессии

$$\frac{1}{l} \sum_{i=1}^{l} log(1 + exp(-y_i \langle w, x_i \rangle)) + \frac{\lambda}{2} reg \to \min_w$$

где reg – l_1 или l_2 регуляризация, $\lambda > 0$ – сила регуляризации (подбирается на кросс-валидации):

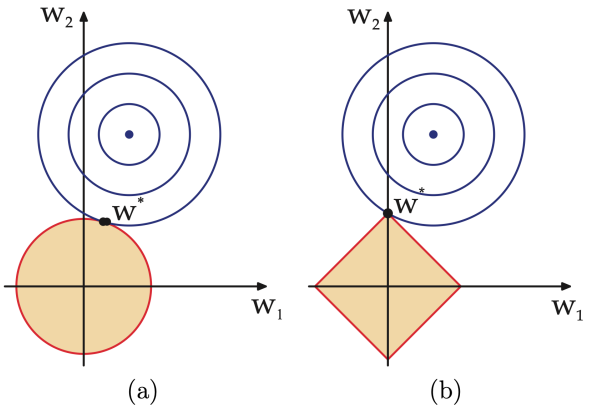


Рис. 1. a) l_2-регуляризация, b) l_1-регуляризация

- l_1-регуляризация: $\sum_{j=1}^{d} |w_j|$
- l_2-регуляризация: $\sum_{j=1}^{d} w_j^2$

Зачем нужна регуляризация? Предположим, что уже найдена идеально разделяющая выборку гиперплоскость $\langle w^*, x\rangle = 0$. Умножив w^* на любое положительное число мы не изменим положение гиперплоскости, а вот $\frac{1}{l}\sum_{i=1}^{l} log(1 + exp(-y_i\langle w, x_i\rangle))$ станет ближе к нулю. То есть без регуляризации обучение закончится, а веса продолжат меняться. Большие веса могут ломать адекватность модели, приводя к резкому изменению вероятности при небольшом изменении значения признака (когда мы ожидаем, что объекты близкие по значению признаков должны иметь один класс).

17 Постановка задачи SVM (Hard и Soft margin).

Hard-margin SVM:

$$\begin{cases} \frac{1}{2}\|w\|^2 \to \min\limits_{w,b} \\ y_i(\langle w, x\rangle + b) \geq 1 \end{cases}$$

$a(x) = sign(\langle w, x\rangle + b)$
Если $\lambda_i > 0$ и $y_i(\langle w,x_i\rangle + b) = 1$, то такой объект обучающей выборки x_i называется *опорным вектором*. Решение задачи зависит только от опорных векторов.

Soft-margin SVM:

$$\begin{cases} \frac{1}{2}\|w\|^2 + C\sum\xi_i \to \min\limits_{w,b,\xi_i} \\ y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{cases}$$

Здесь объекты выборки можно разделить на 3 категории:

1. Эталонные
   $\lambda = 0, \xi = 0$, Не влияют на веса и на решение.

2. Опорные $\xi = 0, 0 < \lambda < c$

3. Нарушители $\xi > 0, \lambda = c$. Причем нарушители бывают 2 видов:

   (a) $\xi < 1$ Правильно классифицированы, но маленькая margin.

   (b) $\xi > 1$ Неправильно классифицированы.

Решение задачи зависит от опорных векторов и векторов-нарушителей.

Чем больше значение коэффициента регуляризации C, тем на меньшее количество объектов будет опираться гиперплоскость, чтобы не занижать функционал, но тогда переобучение.

Откуда берется $\frac{1}{2}\|w\|^2 \to \min\limits_{w,b}$?
Хотим максимизировать расстояние от x_i до разделяющей гиперплоскости, то есть $\frac{|w^T x+b|}{\|w\|} \to \max\limits_{w,b}$. Веса в этом выражении подбираются так, чтобы числитель=1, тогда это сводится к задаче $\|w\|^2 \to \min$ и отсюда же возникает ограничение ≥ 1. Сама найденная гиперплоскость никак не зависит от этой нормировки числителя.

18 Определение ядровой функции (K(x, z)). Обобщение SVM с помощью ядер.

$K(x, z) = \langle \phi(x), \phi(z) \rangle$ – **ядровая функция**, где $\phi : X \to H$, (функция переводящая пространство изначальных признаков X в спрямляющее пространство H).

Обобщение SVM с помощью ядер:
В случае использования ядра меняется оптимизационная задача поиска λ:

$$\begin{cases} g(\lambda) = -\frac{1}{2}\sum\limits_j \sum\limits_i \lambda_i \lambda_j y_i y_j \underbrace{\langle \phi(x_i), \phi(x_j) \rangle}_{K(x_i, x_j)} + \sum\limits_i \lambda_i \to \max\limits_{\lambda} \\ \dots \end{cases}$$

Веса в задаче SVM теперь находятся как:

$$w = \sum_{i=1}^{n} \lambda_i y_i \phi(x_i)$$

Сам прогноз же выглядит следующим образом:

$$a(x) = sign(\langle w, \phi(x) \rangle + b) = sign(\sum_{i=1}^{n} \lambda_i y_i \langle \phi(x_i), \phi(x) \rangle + b)$$

Данные преобразования позволят решить задачу с нелинейной разделимостью.

19 Постановка задачи гребневой регрессии (Ridge-regression). Формула оптимальных весов для неё.

Техника регуляризации, применяемая в линейных моделях классификации и регрессии, чтобы решить проблему мультиколлинеарности и переобучения. Вид: добавление к среднеквадратичной ошибке штрафа за увеличение нормы вектора весов $\|w\|$:

$$L(X, \vec{y}, \vec{w}) = \|\vec{y} - X\vec{w}\|^2 + \frac{\lambda}{2}\|\vec{w}\|^2,$$

где λ – неотрицательный параметр регуляризации.
Формула оптимальных весов:

$$\vec{w}^*_\lambda = (X^T X + \lambda E)^{-1} X^T y,$$

где λE – гребень.
(При мультиколлинеарности матрица $X^T X$ близка к сингулярной или вырожденной, то есть некоторые собственные значения будут близки к нулю, а в обратной матрице $X^T X^{-1}$ появятся экстремально большие собственные значения, что будет сильно изменять решение при добавлении 1 наблюдения).

20 Алгоритм Word2Vec. Какой функционал и как оптимизируется в варианте Skip-Gram Negative Sampling.

Алгоритм Word2Vec обучает векторные представления слов для предсказания «окружающих» слов – контекста (либо наоборот, для предсказания слова с помощью «окружающих» его в предложении). Функция правдоподобия и соотвествующая ей функция потерь такие:

$$Likelihood = L(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m, j \neq 0} P(w_{t+j} | w_t, \theta)$$

$$Loss = J(\theta) = -\frac{1}{T} \cdot \log L(\theta) = -\frac{1}{T} \cdot \sum_{t=1}^{T} \sum_{-m \le j \le m, j \neq 0} \log P(w_{t+j} | w_t, \theta), \text{ где}$$

- T – количество позиций в корпусе текстов;
- m – размер окрестности (количество «окружающих» слов слева и справа)
- w_i – обозначение i-го слова (у нас w_t – центральное слово, w_{t+j} – окружающее);
- θ – оптимизируемые параметры векторов матриц U и V.

Инициализируются две матрицы U (контекстные слова) и V (центральные слова) размерности $|Vocab| \times h$ (размер словар на выбранную размерность размера вектора слова). Далее, для каждой комбинации центрального и контекстного слов отдельно оптимизуруем функционал:

$$J_{t,j}(\theta) = -\log P(w_{t+j} | w_t, \theta) = -\log \frac{\exp(u^T_{t+j} v_t)}{\sum_{u_{t+j} \in Vocab} \exp(u^T_{t+j} v_t)} = -u^T_{t+j} v_t + \log \sum_{u_{t+j} \in Vocab} \exp(u^T_{t+j} v_t)$$

Проблема в этом подходе в том, что для оптимизации одной пары нам надо сделать шаг для каждого контекстного слова из словаря (+ для самого центрального), для решения этой проблемы можно рандомно выбирать только k слов из контекстной матрицы (их и называют условно «negative» examples) и придётся делать градиентный шаг лишь для (k+2) векторов слов (функционал может быть тот же, только в сумме участвуют (k+1) элементов). Это и есть Skip-Gram Negative Sampling:

$$J_{t,j}(\theta) = -u_{t+j}^T v_t + \log \sum_{u_{t+j} \in SubVocab} \exp(u_{t+j}^T v_t), \text{ где}$$

$SubVocab$ – случайные k векторов из U + контекстный из рассматриваемой пары.

21 Байесовский подход в машинном обучении. Априорное/Апостериорное распределение на параметры. Связь априорного распределения с регуляризацией. Виды точечных оценок (MAP-оценка/байесовская оценка)

Основная идея подхода - найти не саму точечную оценку, а ее распределение и на его основе выбирать что вам нужно (моду, медиану и тд.). То есть найти распределение $p(w|x,y)$, где w - веса, x - фичи, y - таргеты. Распределение - это максимум информации об оценке!

Априорные и апостериорные суждения

- Предположим, мы пытаемся изучить некоторое явление
- У нас имеются некоторые знания, полученные до (лат. a priori) наблюдений/эксперимента. Это может быть опыт прошлых наблюдений, какие-то модельные гипотезы, ожидания
- В процессе наблюдений эти знания подвергаются постепенному уточнению. После (лат. a posteriori) наблюдений/эксперимента у нас формируются новые знания о явлении

Пусть X - это сразу x и y. Тогда формула Байеса будет иметь вид

$$p(w|X) = \frac{p(X|w) * p(w)}{p(X)}$$

$p(w)$ - априорное распределение на параметры
$p(w|X)$ - апостерирное распределение на параметры
MAP-оценка - оценка с максимальной апостериорной вероятностью

$$\hat{w}_{MAP} = arg\max_w \ p(w|X)$$

Появление l2 регуляризации в MAP-оценке как раз показывает, что мы учитываем больше информации, когда ищем именно распределение оценки, а не точечную оценку.

Байесовская оценка

$$\hat{w}_{bayes} = E_{p(w|X)} w$$

22 EM-алгоритм. На какие компоненты раскладывается неполное правдоподобие. Как выглядят шаги EM-алгоритма.

Expectation maximization:
EM-алгоритм – итерационный метод максимизации правдоподобия выборки. Метод оптимизации в вероятностных моделях с исходной задачей: $\log p(x|\theta) \to \max\limits_{\theta}$, где $x \in R^n$, θ – набор параметров. Такой функционал не всегда легко оптимизировать, проще оптимизировать другой функционал, являющейся его нижней оценкой, используя введение скрытой переменной z с плотностью q и функцию полного правдоподобия ($\log p(x,z|\theta)$).
$KL(p||q)$ – дивергенция Кульбака-Лейблера, характеризующая расстояние между двумя распределениями.

$$\log p(x|\theta) = \log p(x|\theta) \cdot \int q(z)dz = \int q(z) \cdot \log p(x|\theta)dz = \{p(x,y) = p(x|y) \cdot p(y)\} =$$
$$= \int q(z) \cdot \log \frac{p(x,z|\theta)}{p(z|x,\theta)}dz = \int q(z) \cdot \log \frac{p(x,z|\theta) \cdot q(z)}{q(z) \cdot p(z|x,\theta)}dz =$$
$$= \int q(z) \cdot \log \frac{p(x,z|\theta)}{q(z)}dz + \int q(z) \cdot \log \frac{q(z)}{p(z|x,\theta)}dz =$$
$$= \mathcal{L}(q,\theta) + KL(q(z)||p(z|x,\theta)) \geqslant \mathcal{L}(q,\theta)$$

$\mathcal{L}(q,\theta)$ – это ELBO (evidence lower bound), его и будем оптимизировать.

Так, можем оптимизировать нижнюю границу в 2 шага по q и θ
Шаг E: $\mathcal{L} \to \max\limits_{q} \sim KL(p||q) \to \min\limits_{q}$, т.к. $\mathcal{L} = \log p(x|\theta) - KL(q(z)||p(z|x,\theta))$, минимум достигается при совпадении распределений q и p: $q^* = p(z|x,\theta^{old})$
Шаг M: $\mathcal{L}(q^*,\theta) \to \max\limits_{\theta}, \mathcal{L}(q^*,\theta) = \int q^*(z) \times \log \frac{p(x,z|\theta)}{q^*(z)}dz = E_{q^*(z)} \log p(x,z|\theta) + H(q^*)$:
$\theta^{new} = \underset{\theta}{\operatorname{argmax}} E_{q^*(z)} \log p(x,z|\theta)$

23 Bias-variance-decomposition формула разложения. Интерпретация компонент.

Пусть есть какое-то распределение фичей (x) и соответствующих им ответов (y): $p(x,y)$ – *плотность распределения*. Из этого распределения генерируется выборка X и ответы на ней. Будем рассматривать задачу регрессии (y – вещественное число). Пусть a – наш алгоритм. В качестве функции потерь будем рассматривать квадротичную функцию : $L(y,a) = (y - a(x))^2$, которой соответствует среднеквадратическая ошибка: $E_{x,y}(y - a(x))^2$ (это MSE). При минимизации MSE оптимальным алгоритмом будет $a = E(y|x)$. Чтобы использовать такой алгоритм нам необходимо знать распределение, но мы его не знаем $\implies$ переходим к выбору алгоритма на основе **метода обучения**.

Из обучающей выборки генерируем выборки (произвольным способом) с фиксированным размером n. Для каждой выборки выбираем метод обучения – μ (он выбирается из некоторого семейства алгоритмов A), обучаем μ на соответствующей

ему выборке и получаем предикты:

$$a(x)_{X^n} = \mu(X^n)(x) \text{ – это уже ответы}$$

Для оценки качества **метода обучения** берем среднее по всем выборкам значение среднеквадратической ошибки (усредняем MSE по всем X^n):

$R(a) = E_{X^n}(E_{x,y}(y - a(x))^2)$ – ошибка, разложение на компоненты которой хотим понять

Разложение ошибки:

$$R(a) = \underbrace{E_{x,y}(y - E(y|x))^2}_{\text{шум}} + \underbrace{E_{x,y}[(E(y|x) - E_{X^n}\mu(X^n)(x))^2]}_{\text{смещение}} + \underbrace{E_{x,y}[E_{X^n}[(\mu(X^n)(x) - E_{X^n}\mu(X^n)(x))^2]]}_{\text{дисперсия}}$$

Логика:

- Шум показывает ошибку **лучшей модели** $= E(y|x)$, это шум исходных данных, устранить не выйдет
- Смещение показывает на сколько наша модель (среднее по обученным алгоритмам $\mu(X^n)$) отличается от лучшего алгоритма $E(y|x)$
- Дисперсия показывает разброс ответов обученных алгоритмов относительно среднего ответа

Разложение верно почти для любой функции потерь (у нас была квадратичная).

24 Бэггинг и случайный лес.

Бэггинг (**B**ootstrap **Agg**regat**ing**) – способ построения ансамбля следующим образом: мы строем базовую модель на бутстреп подвыборке, делаем это несколько на разных подвыборках, получаем разные модели, после чего применяем к данным и усредняем ответы.

При отборе подвыборки с помощью бутстрепа вероятность какого-либо элемента из изначального множества попасть в подвыборку составляет при каком-либо взятии $\frac{1}{n}$, тогда вероятность не попасть выборку при этом взятии $1 - \frac{1}{n}$. Отсюда получаем, что вероятность не попасть в бутстреп подвыборку того же размера, что и изначальная составляет $\left(1 - \frac{1}{n}\right)^n$, значит, вероятность попасть хотя бы раз будет:

$$1 - \left(1 - \frac{1}{n}\right)^n \underset{n\to\infty}{\longrightarrow} 1 - \frac{1}{e} \approx 0{,}63$$

Получается, что каждая подвыборка состоит примерно из 63% оригинальной выборки, на которой и обучается, остальная часть называется out-of-bag-наблюдениями. На этих наблюдениях можно оценивать качество моделей, считая для них ответы, а с их помощью и ошибки. Ответ для таких наблюдений получаем следующим образом:

$$a_{OOB}(x_j) = \frac{1}{|\{i : x_j \in OOB_i\}|} \sum_{i:x_j \in OOB_i} b_i(x_j)$$

Случайный лес
Алгоритм:

1) Берём бутстрап подвыборку

2) Строим на подвыборке случайное дерево следующим образом: сначала выбираем случайное подмножество фичей, потом по ним ищем оптимальный сплит, после чего опять выбираем случайное подмножество фичей и уже по ним делаем оптимальный сплит и так далее

3) Повторяем шаги 1), 2) для постройки необходимого нам количества деревьев

4) Получаем ответы случайного леса, как среднее предсказание по всем деревьям

25 Алгоритм градиентного бустинга. Особенность градиентного бустинга над деревьями.

Градиентный бустинг – метод ансамблирования, в котором мы хотим получить алгоритм следующего вида:

$$a_N(x) = \sum_{j=0}^{N} \gamma_j b_j(x)$$

$b_j(x)$ – это простые алгоритмы (weak learners), которые обучаются итеративно (то есть сначала мы назначаем нулевой алгоритм $b_0(x)$, а потом последовательно обучаем первый, второй и т.д.), а γ_j – это веса, а $\mathcal{L}(y,z)$ пусть будет нашей loss-функцией

Алгоритм градиентного бустинга:

1) Вычисляем антиградиенты нашей функции потерь в точках ответов нашей предыдущей композиции (которая уже настроена)

$$s_i^{(N)} = -\frac{\partial}{\partial z}\mathcal{L}(y_i, z)\Big|_{z=a_{N-1}(x_i)}$$

2) Настраиваем наши слабые модели (weak learners)

$$b_N(x) = \underset{b\in\mathcal{A}}{\operatorname{argmin}} \sum_{i=1}^{n} (b(x_i) - s_i^{(N)})^2$$

3) Настраиваем вес (гамму)

$$\gamma_N = \underset{\gamma\in R}{\operatorname{argmin}} \sum_{i=1}^{n} \mathcal{L}(y_i, a_{N-1}(x_i) + \gamma b_N(x_i))$$

Особенности градиентного бустинга над деревьями:

Когда мы используем деревья, то мы можем представить простую модель (дерево) в следующем виде:

$$b(x) = \sum_{t=1}^{T} w_t[x \in R_t]$$

Тогда наша функция потерь выглядит следующим образом:

$$\sum_{i=1}^{n} \mathcal{L}(y_i, a_{N-1}(x) + \gamma_N \sum_{t=1}^{T} w_{Nt}[x \in R_t])$$

Однако и γ_N и w_{Nt} представляют из себя константы, а потому нет смысла сначала искать одну, потом другую, проще их сразу объединить, после чего мы можем разбить нашу задачу на эквивалентные подзадачи, так как константа ищется для каждого листа отдельно, независимо от других:

$$\sum_{i:(x_i,y_i)\in R_t} \mathcal{L}(y_i, a_{N-1}(x_i) + w_{Nt})$$

Тогда алгоритм получается следующий:

1) Вычисляем антиградиенты нашей функции потерь в точках ответов нашей предыдущей композиции (тут ничего не поменялось, так как антиградиенты зависят от функции потерь, а не от моделей)

$$s_i^{(N)} = -\frac{\partial}{\partial z}\mathcal{L}(y_i, z)\Big|_{z=a_{N-1}(x_i)}$$

2) Настраиваем наши деревья (которые соответсвуют weak learners, и которые обучаются по антиградиентам)

$$b_N(x) = \underset{b\in DecisionTree}{\operatorname{argmin}} \sum_{i=1}^{n} (b(x_i) - s_i^{(N)})^2$$

3) Обновляем константы в листьях в соответствии с нашей функцией потерь (то есть деревья у нас обучаются и выдают константу оптимальную для MSE, а мы их меняем а не оптимальные с точки зрения функции потерь)

$$w_{Nt}^*(x) = \underset{w\in R}{\operatorname{argmin}} \sum_{i:(x_i,y_i)\in R_t} \mathcal{L}(y_i, a_{N-1}(x_i) + w)$$

Пример для понимания: из второго шага мы получили константы, и так как у нас во втором шаге минимизируется квадрат разницы, то константа равна среднему значению, теперь представим, что в качестве функции потерь у нас взят модуль отклонения, тогда получим, что оптимальная константа для функции потерь должна быть равна медианному значению

26 Оценка качества кластеризации.

Существует два подхода к оценке качества кластеризации: внутренний (основан на некоторых свойствах выборки и кластеров) и внешний (использует дополнительные данные, например, об истинных кластерах). Несколько примеров внутренних метрик качества (через c_k обозначим центр кластера k):

1. $\sum_{k=1}^{K}\sum_{i}^{n}[a(x_i)=k]\cdot\rho(x_i,c_k)$ – внутрикластерное расстояние (минимизируется);
2. $\sum_{i<j}^{n}[a(x_i)\neq a(x_j)]\cdot\rho(x_i,x_j)$ – межкластерное расстояние (максимизируется);
3. $\frac{min_{1\leq k<k'\leq K}d(k,k')}{max_{1\leq k\leq K}d(k)}$ – индекс Данна (Dunn Index): $d(k,k')$ – расстояние между кластерами, $d(k)$ – внутрикластерное расстояние. Индекс максимизируется.
4. для особых ценителей (здесь C_i обозначает кластер, к которому принадлежит наблюдение x с индексом i):

$$Silhouette(x_i)=\begin{cases}\frac{b(x_i)-a(x_i)}{\max(a(x_i),b(x_i))},\ \text{если } |C_i|>1\\ 0,\ \text{если } |C_i|=1\end{cases},\ \text{где}$$

$$a(x_i)=\frac{1}{|C_i|-1}\sum_{j\neq i,x_j\in C_i}p(x_i,x_j),\quad b(x_i)=\min_{k\neq i}\frac{1}{|C_k|}\sum_{j\in C_k}p(x_i,x_j)$$

Далее силуэт можно усреднить по всем точкам и получить оценку качества.

27 Многослойный персептрон. Основные функции активации. Активация на выходном слое в случае задачи классификации (бинарной/многоклассовой) и регрессии. Типичные функции потерь для задачи классификации и регрессии. Автокодировщик.

Многослойный персептрон – класс искусственных нейронных сетей прямого распространения (ациклический граф), состоящих как минимум из трех слоёв: входного (input layer), скрытого (hidden layer) и выходного (output layer) (рис. 2).

Каждый слой имеет какое-то количество нейронов (кружочки на рисунке). Количество нейронов на входном слое равно количеству располагаемых признаков (размерности признакового пространства), количество нейронов на выходном слое зависит от задачи: в случаи обычной регрессии и бинарной классификации нейрон будет один, в случае многоклассовой классификации количество нейронов соответствует

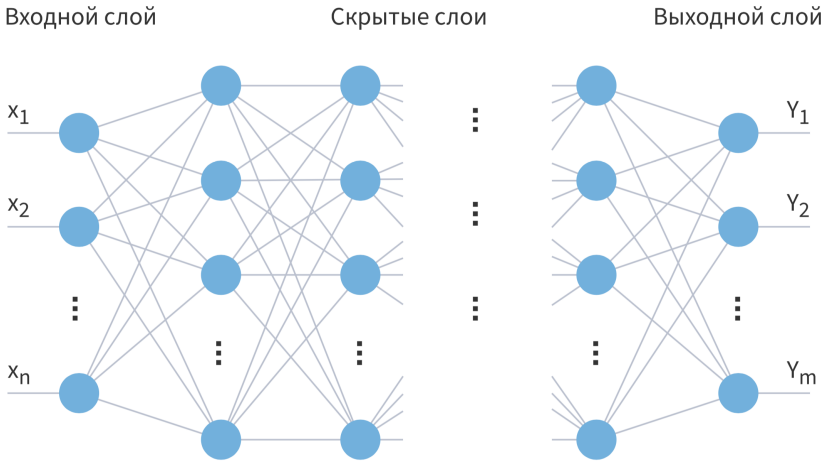


Рис. 2. Красота какая

количеству классов (для бинарной тоже будет верно). Нейроны на скрытых и выходном слоях аккумулируют информацию с предыдущего им слоя (суммируются в виде линейной комбинации со сдвигом) и применяют **функции активации**, например:

- $\sigma(z) = \frac{1}{1+e^{-z}}$ – сигмоида;
- $\text{th}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ – гиперболический тангенс;
- $f(z) = \max(0, z)$ – Rectified linear unit, ReLU

В качестве активации на выходном слое в случае бинарной классификации используется сигмоидная функция, в случае регрессии годится и просто тождественное преобразование ($f(z) = z$), а в случае многоклассовой используется Softmax преобразование:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

На словах, в каждом нейроне берётся экспоненциальная функция от z_j (z_j уже является взвешенной суммой значений нейронов с прошлого слоя со сдвигом, то есть $z_j = \sum_{i=1}^{d} w_i \cdot x_i + b$, где d – количество нейронов в предыдущем слое) и нормируется.

В качестве **функций потерь** обычно используют:

- в случае регрессии пойдёт классический MSE;
- в случае многоклассовой классификации (бинарная будет частным случаем) используется кросс-энтропия:

$$logloss = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{l} y_{ij} \cdot \log a_{ij}, \text{ где}$$

n – число наблюдений, l – число классов, y_{ij} равен 1, если i-ое наблюдение принадлежит классу j, и 0 иначе, a_{ij} – модельная вероятность принадлежности i-ое наблюдения к j-ому классу.

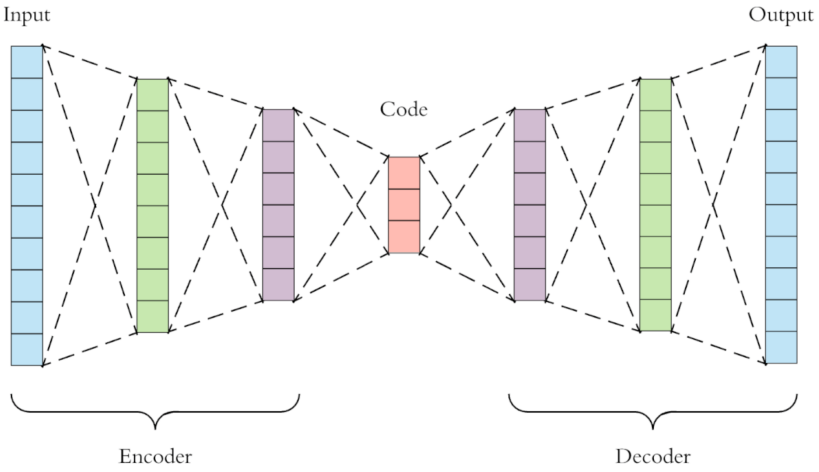


Рис. 3. Ешё одна красота

Автоэнкодер – специальный вид нейросетей, где количество нейронов в выходном слое соответствует количеству нейронов в входном: нейросеть пытается копировать входные данные на выход как можно точнее (рис. 3). Encoder часть автоэнкодера сжимает входные данные для представления их в latent-space (скрытое пространство), а затем decoder часть восстанавливает из этого представления output (выходные данные). Частный случай персептрона вполне себе пример такого автокодера.

Данный вид нейросетей используется для некоторых задач обучения без учителя: уменьшения шума в данных и снижения размерности многомерных данных для визуализации (оказывается лучше классических подходов типа PCA и других). Для целей визуализации используется скрытое представление (оранжевые нейроны на картинке), которое пытается сохранить в себе как можно больше информации (так как дальше на её основе происходило восстановление).

28 Chain rule для производной сложной функции. Алгоритм Backpropagation.

Chain rule – правило дифференцирования сложной функции:

$$\frac{\partial L}{\partial x} = \sum_{i=1}^{n} \frac{\partial L}{\partial z_i} \cdot \frac{\partial z_i}{\partial x}$$

Алгоритм Backpropagation – алгоритм, который используют для тренировки нейросетей. Идея в том, что с помощью разложения по chain rule мы сможем утилизировать информацию о производных с верхнего слоя, что упрощает счёт на нижних слоях и позволяет делать меньше операций и следовательно позволяет быстрее обучать миллионы параметров. Алгоритм такой: имея после прямого прохода по сети все промежуточные ответы и ошибку на функции потерь (или просто конечный ответ функции интереса), считаем все необходимые градиенты с конца в начало, «распространяем ошибку». На основе этих градиентов можем обновить все веса.

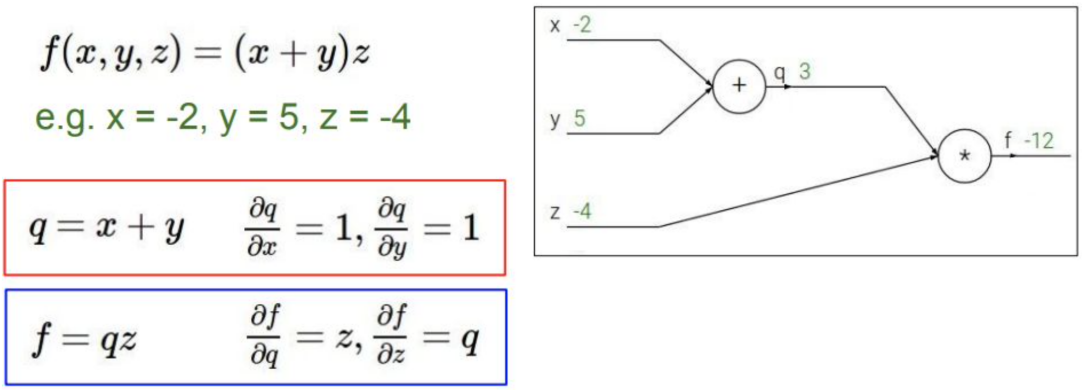


Рис. 4. Пример для разгона

Рассмотрим пример с оптимизацией функции $f(x, y, z)$ (рис. 4):

Делаем рандомную инициализацию (зелёные чиселки в первом столбике) и производим прямое распространение (слева направо делаем прописанные на графе операции и получаем все остальные зелёные чиселки). Вот мы и пришли в конец, теперь можем идти в начало (рассмотрим только $\frac{\partial f}{\partial x}$, для $\frac{\partial f}{\partial y}$ ответ такой же, а пример с z запутывающе простой):

$$\frac{\partial f}{\partial q} = \frac{\partial f}{\partial (x+y)} = -4 \Longrightarrow \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x} = -4 \cdot 1 = -4$$

Теперь более сложный пример (рис. 5). Представим, что мы хотим оптимизировать параметры w_i для сигмоиды.

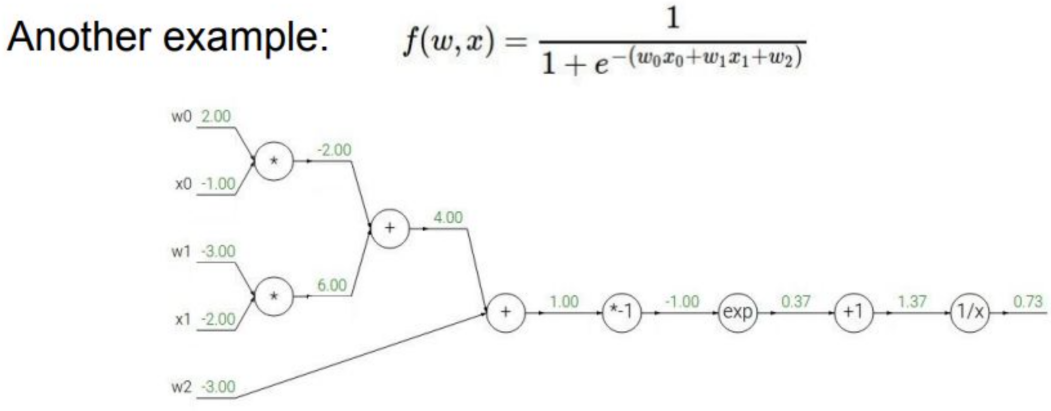


Рис. 5. Пример с логистической функцией (сигмоидой) – простейшая нейросеть

Также сделаем рандомную инициализацию и произведём прямое распространение. Далее посчитаем необходимые производные (на экзамене лучше вводить обозначения для упрощения конструкций).

$$\frac{\partial\left(\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}\right)}{\partial(1+e^{-(w_0x_0+w_1x_1+w_2)})} = -\frac{1}{(1+e^{-(w_0x_0+w_1x_1+w_2)})^2} = -\frac{1}{1.37^2} = -0.53$$

$$\frac{\partial(1+e^{-(w_0x_0+w_1x_1+w_2)})}{\partial(e^{-(w_0x_0+w_1x_1+w_2)})} = 1$$

$$\frac{\partial(e^{-(w_0x_0+w_1x_1+w_2)})}{\partial(-(w_0x_0+w_1x_1+w_2))} = e^{-(w_0x_0+w_1x_1+w_2)} = 0.37$$

$$\frac{\partial(-(w_0x_0+w_1x_1+w_2))}{\partial(w_0x_0+w_1x_1+w_2)} = -1$$

$$\frac{\partial(w_0x_0+w_1x_1+w_2)}{\partial w_2} = 1$$

Теперь у нас есть все необходимые производные для вычисления производной сигмоиды по w_2 с помошью chain rule.

$$\frac{\partial\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}}{\partial w_2} = \frac{\partial\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}}{\partial(1+e^{-(w_0x_0+w_1x_1+w_2)})} \cdot \frac{\partial(1+e^{-(w_0x_0+w_1x_1+w_2)})}{\partial(e^{-(w_0x_0+w_1x_1+w_2)})} \cdot$$
$$\cdot \frac{\partial(e^{-(w_0x_0+w_1x_1+w_2)})}{\partial(-(w_0x_0+w_1x_1+w_2))} \cdot \frac{\partial(-(w_0x_0+w_1x_1+w_2))}{\partial(w_0x_0+w_1x_1+w_2)} \cdot \frac{\partial(w_0x_0+w_1x_1+w_2)}{\partial w_2} =$$
$$= (-0.53) \cdot 1 \cdot 0.37 \cdot (-1) \cdot 1 = 0.2$$

Теперь досчитаем пару производных для подсчёта производной сигмоиды по w_0:

$$\frac{\partial(w_0x_0+w_1x_1+w_2)}{\partial(w_0x_0)} = 1 \quad \frac{\partial(w_0x_0)}{\partial w_0} = x_0 = -1$$
$$\frac{\partial\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}}{\partial w_0} = \underbrace{(-0.53) \cdot 1 \cdot 0.37 \cdot (-1)}_{\text{известны с прошлых расчётов}} \cdot 1 \cdot (-1) = -0.2$$

И пару производных для подсчёта производной сигмоиды по w_1:

$$\frac{\partial(w_0x_0+w_1x_1+w_2)}{\partial(w_1x_1)} = 1 \quad \frac{\partial(w_1x_1)}{\partial w_1} = x_1 = -2$$
$$\frac{\partial\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}}{\partial w_1} = \underbrace{(-0.53) \cdot 1 \cdot 0.37 \cdot (-1)}_{\text{известны с прошлых расчётов}} \cdot 1 \cdot (-2) = -0.4$$

29 Конволюционная нейронная сеть. Устройство свёртки (какие есть параметры). Стандартный свёрточный слой (свёртка, нелинейность, пулинг). Идея transfer learning'a.

Идея transfer learning'a состоит в том, чтобы взять нужные слои из уже обученной нейронной сети и использовать их в своей нейронке. Это позволяет значительно экономить время и вычислительные мощности, так как нужно обновлять меньше весов или не придумывать архитектуру самому. Например, есть нейронка, которая обучалась классифицировать картинки животных по 1000 категорий. Перед нами стоит задача классификации картинок 10 видов животных. Мы можем взять "тело"готовой нейронки и добавить нужные нам дополнительные слои ("голова"нейронки). Почему это работает? Так как нейронка обучалась на большем множестве картинок, то, возможно, она научилась распознавать нужные нам объекты.
Что конкретно делают CNN? Берётся изображение, пропускается через серию свёрточных, нелинейных слоев, слоев объединения и полносвязных слоёв, и генерируется вывод.
Название архитектура сети получила из-за наличия операции свёртки, суть которой в том, что каждый фрагмент изображения умножается на матрицу (ядро) свёртки поэлементно, а результат суммируется и записывается в аналогичную позицию выходного изображения. Свертка - это линейная операция!!!
Параметры:

- Глубина (depth) / число каналов
- Высота (height) и ширина (width) каждого ядра
- Шаг (stride) — на сколько смещается ядро при вычислении свёрток (чем больше, тем меньше размер итогового изображения)
- Отступ (padding) – для дополнения изображения нулями по краям

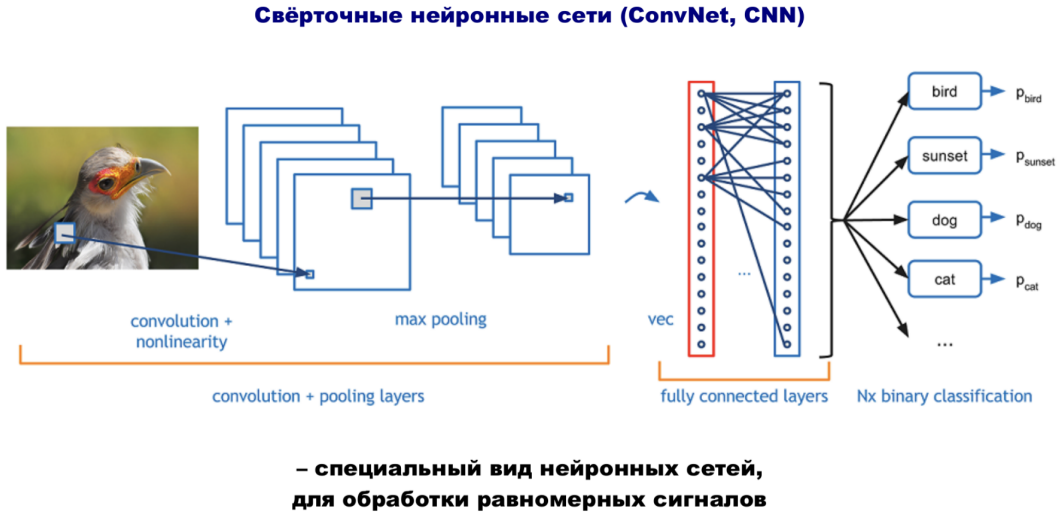


Рис. 6. Стандартный свёрточный слой (свёртка, нелинейность, пулинг)

Пулинг используется для снижения размерности!

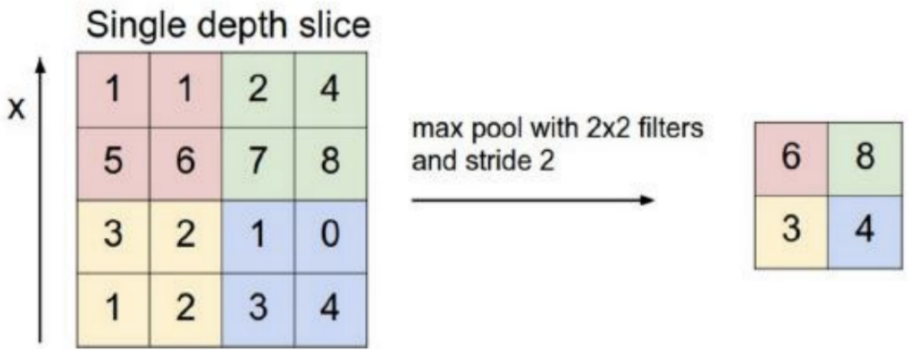


Рис. 7. Пулинг

30 Концептуальное устройство RNN. Почему в базовом варианте затухают/взрываются градиенты?

Рекуррентная нейросеть (RNN = Recurrent neural network) – для обработки последовательностей использование выхода (output) / скрытого состояния (hidden state). Легко масштабируется при увеличении длины последовательностей. Главная идея – разделение параметров (Parameter sharing) как и в свёртках;) Матрицы весов одинаковые при обработке любого элемента последовательности (символ, слово, ...) Учим одну модель, которая применяется на каждом шаге к последовательности любой длины

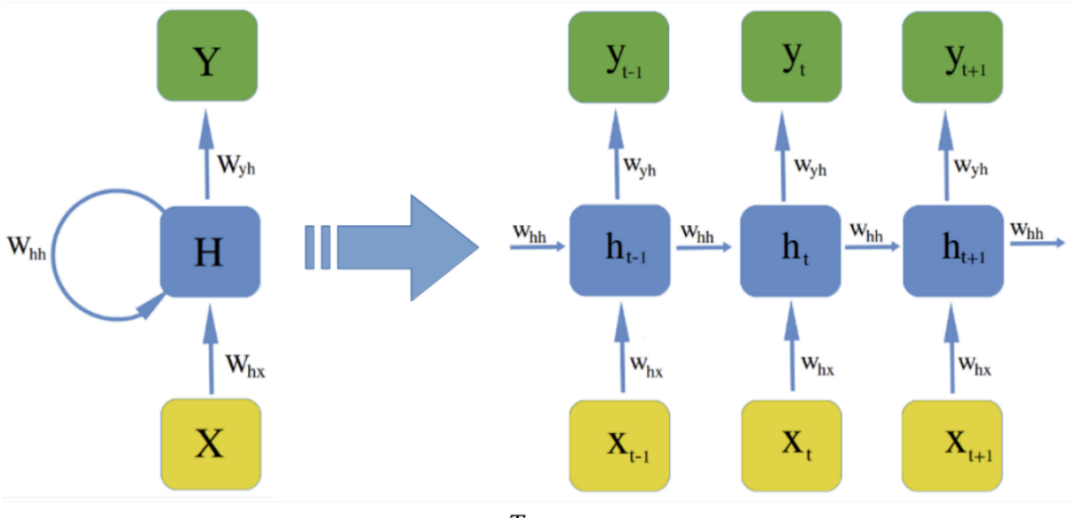


Рис. 8. RNN

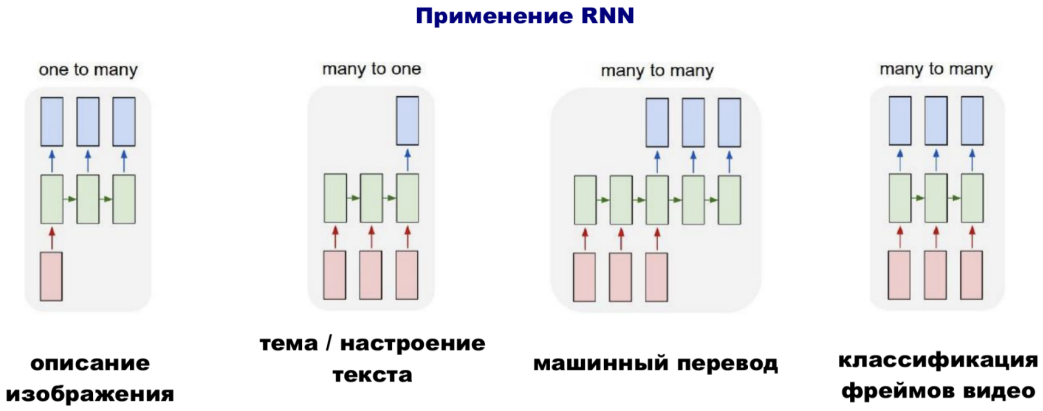


Рис. 9. Применение RNN

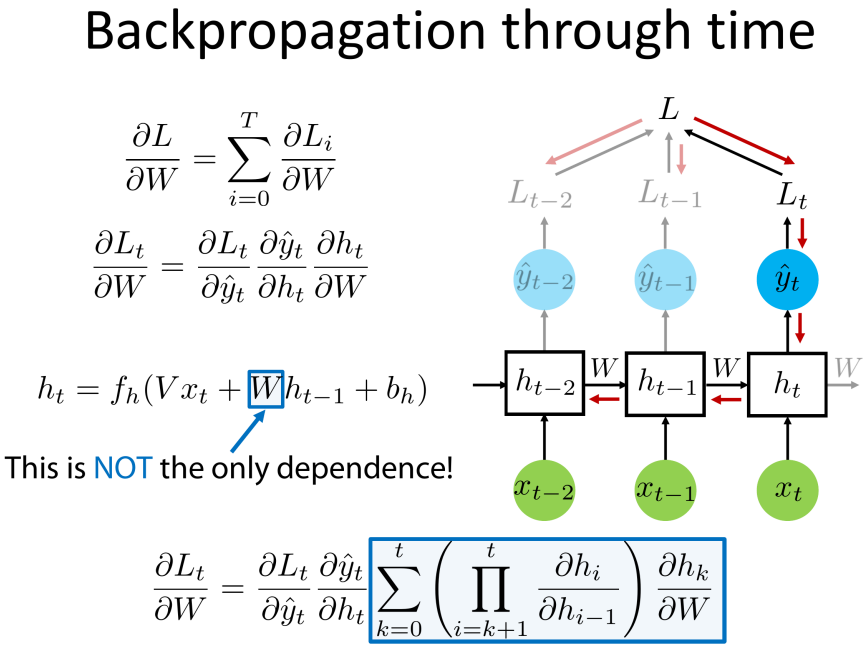


Рис. 10. Градиенты

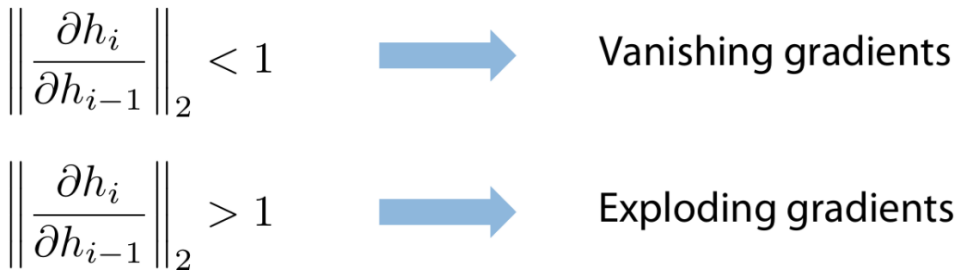


Рис. 11. Затухающие / взрывающиеся градиенты