Алгоритм градиентного спуска для обучения нейронной сети.

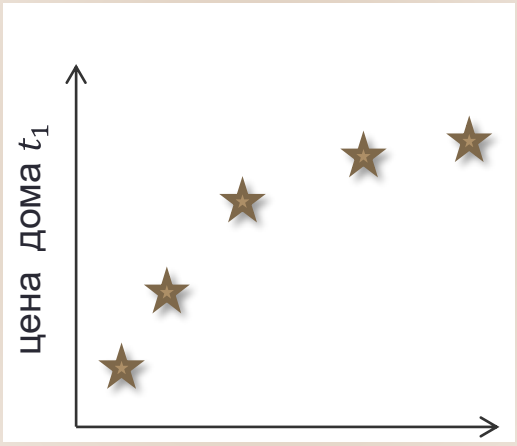
Краткое содержание темы

- Задача обучения нейронной сети
- Функции ошибки для задач регрессии и классификации
- Алгоритм градиентного спуска
- Проблемы алгоритма градиентного спуска

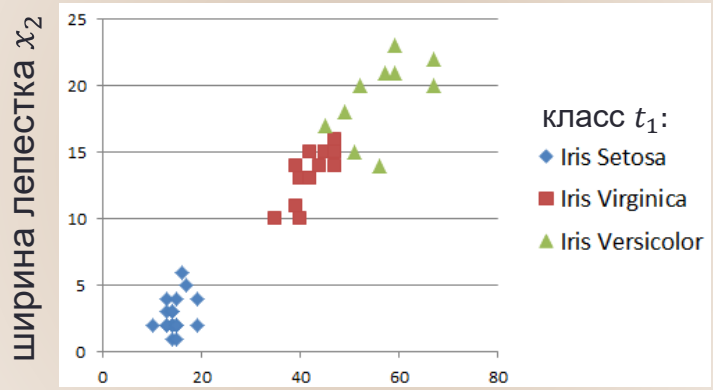
Обучающая выборка

- Каждый элемент обучающей выборки включает:
  - набор значений независимых переменных $\boldsymbol{x^{(i)} = x_1^{(i)}, \ldots, x_n^{(i)}}$
  - соответствующий набор значений зависимых переменных $\boldsymbol{t^{(i)} = t_1^{(i)}, \ldots, t_m^{(i)}}$

Задача регрессии



Задача классификации



Требования к обучающей выборке

● ■ - входят в выборку, ● ■ - не входят в выборку

Задача обучения нейронной сети

- **Необходимо построить нейронную сеть с параметрами, которые отражают зависимость из обучающей выборки**

Критерий ошибки в задаче регрессии

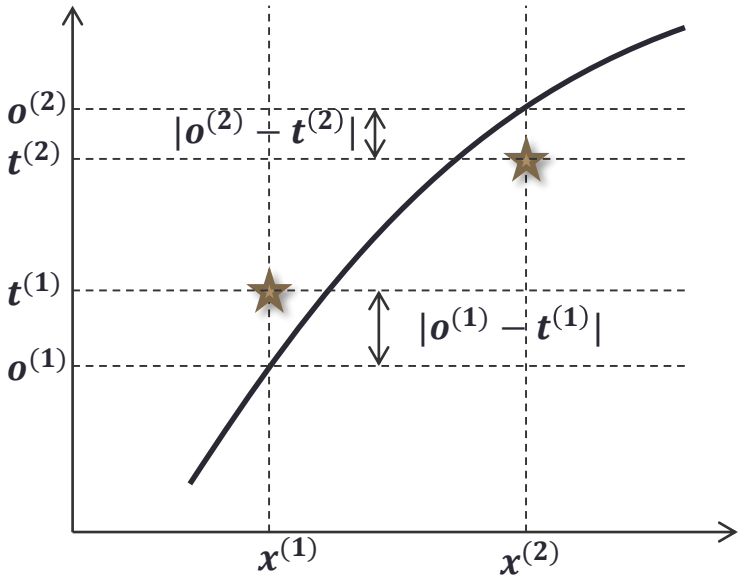
Обучающая выборка

$x^{(1)} \to t^{(1)}$

$x^{(2)} \to t^{(2)}$

…

$x^{(n)} \to t^{(n)}$



Нейронная сеть

$x^{(1)} \to o^{(1)}$

$x^{(2)} \to o^{(2)}$

…

$x^{(n)} \to o^{(n)}$

Критерий ошибки – половина суммы квадратов разности фактических и выданных сетью значений:

$$E_{SS} = \frac{1}{2}\sum_{i=1}^{n}\left(o^{(i)} - t^{(i)}\right)^2$$

так есть на выходе

так должно быть

Критерий ошибки в задаче классификации

Обучающая выборка					
x_1	x_2	класс	t_1	t_2	t_3
45	17	3	0	0	1
47	15	2	0	1	0
16	6	1	1	0	0

Выход сети			
o_1	o_2	o_3	Кросс-энтропия
0.1	0.1	0.9	$0\log 0.1 + 0\log 0.1 + 1\log 0.9 \approx -0.1$
0.1	0.4	0.5	$0\log 0.1 + 1\log 0.4 + 0\log 0.5 \approx -0.9$
0.1	0.8	0.1	$1\log 0.1 + 0\log 0.8 + 0\log 0.1 \approx -2.3$

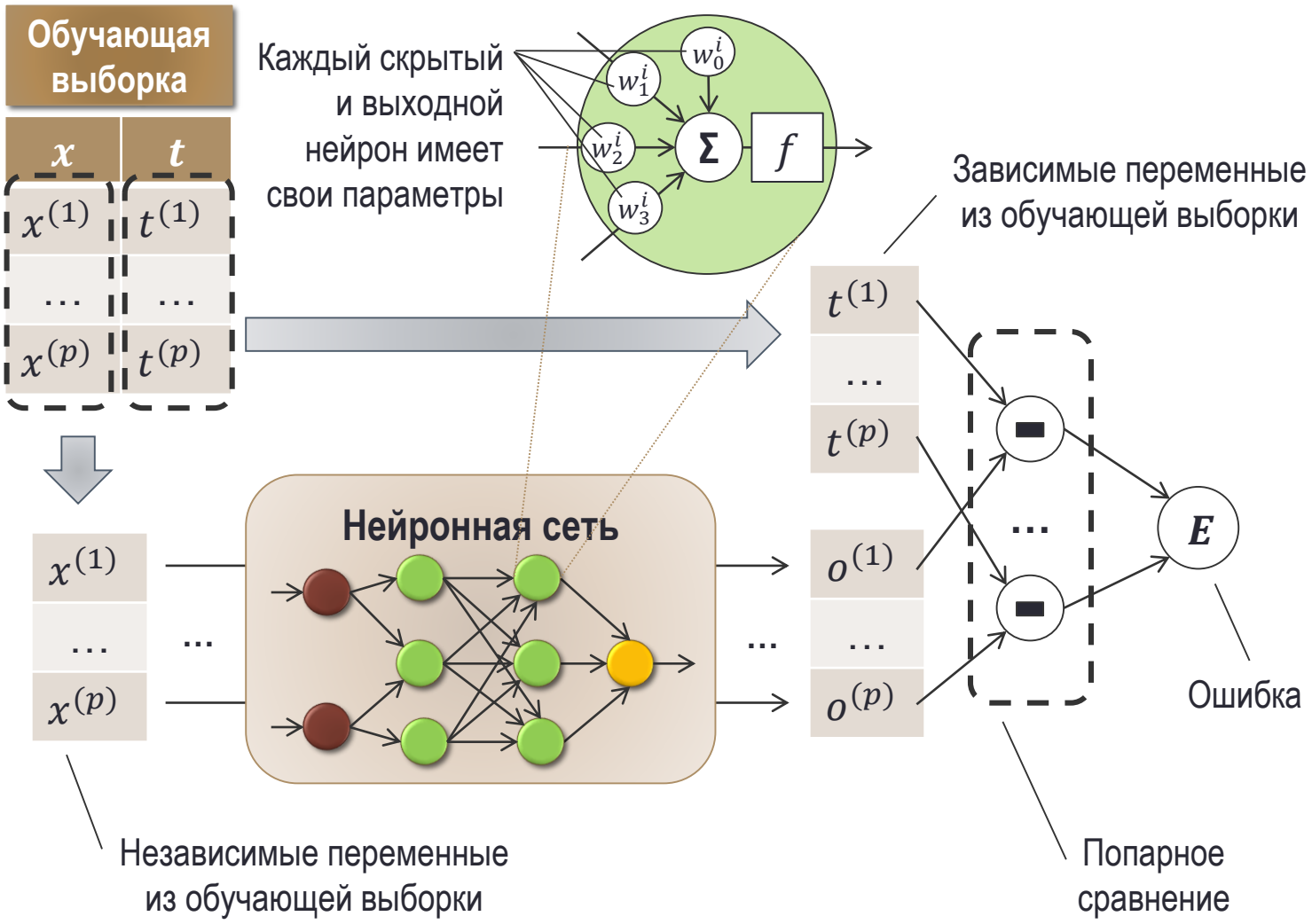
Критерий ошибки – кросс-энтропия:

Сумма по обучающей выборке

Сумма по классам

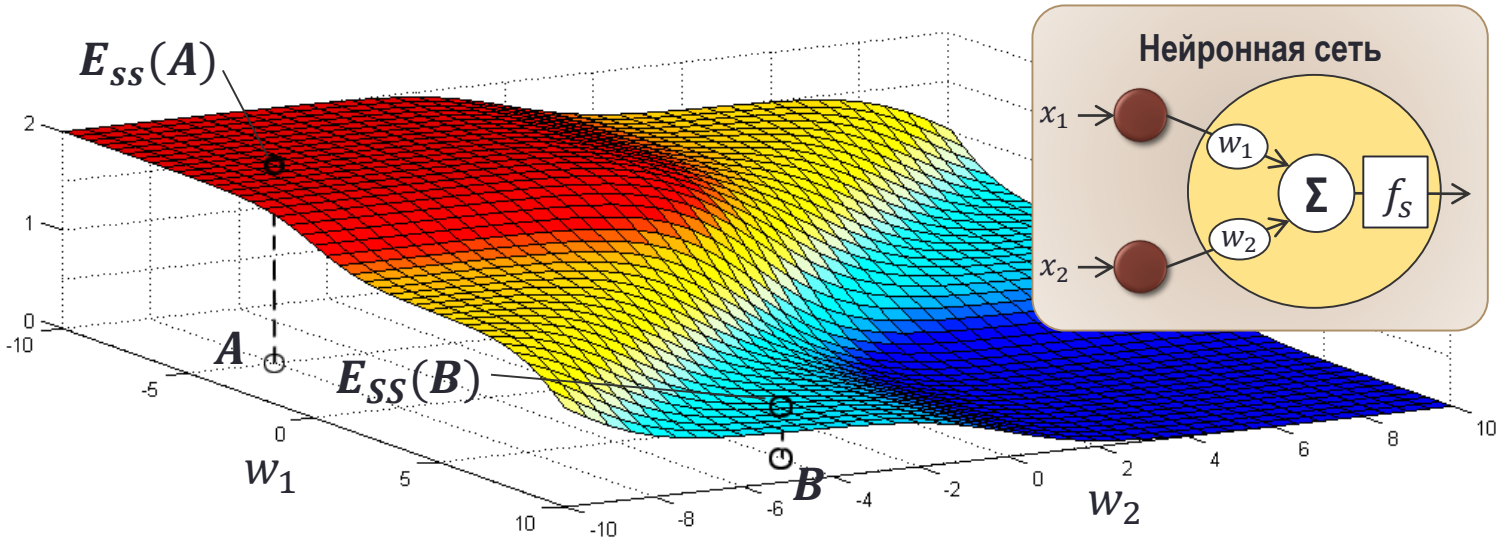
$$E_{CE} = -\sum_{i=1}^{n}\sum_{j=1}^{m} t_j^{(i)} log(o_j^{(i)})$$

Функция ошибки и ее аргументы

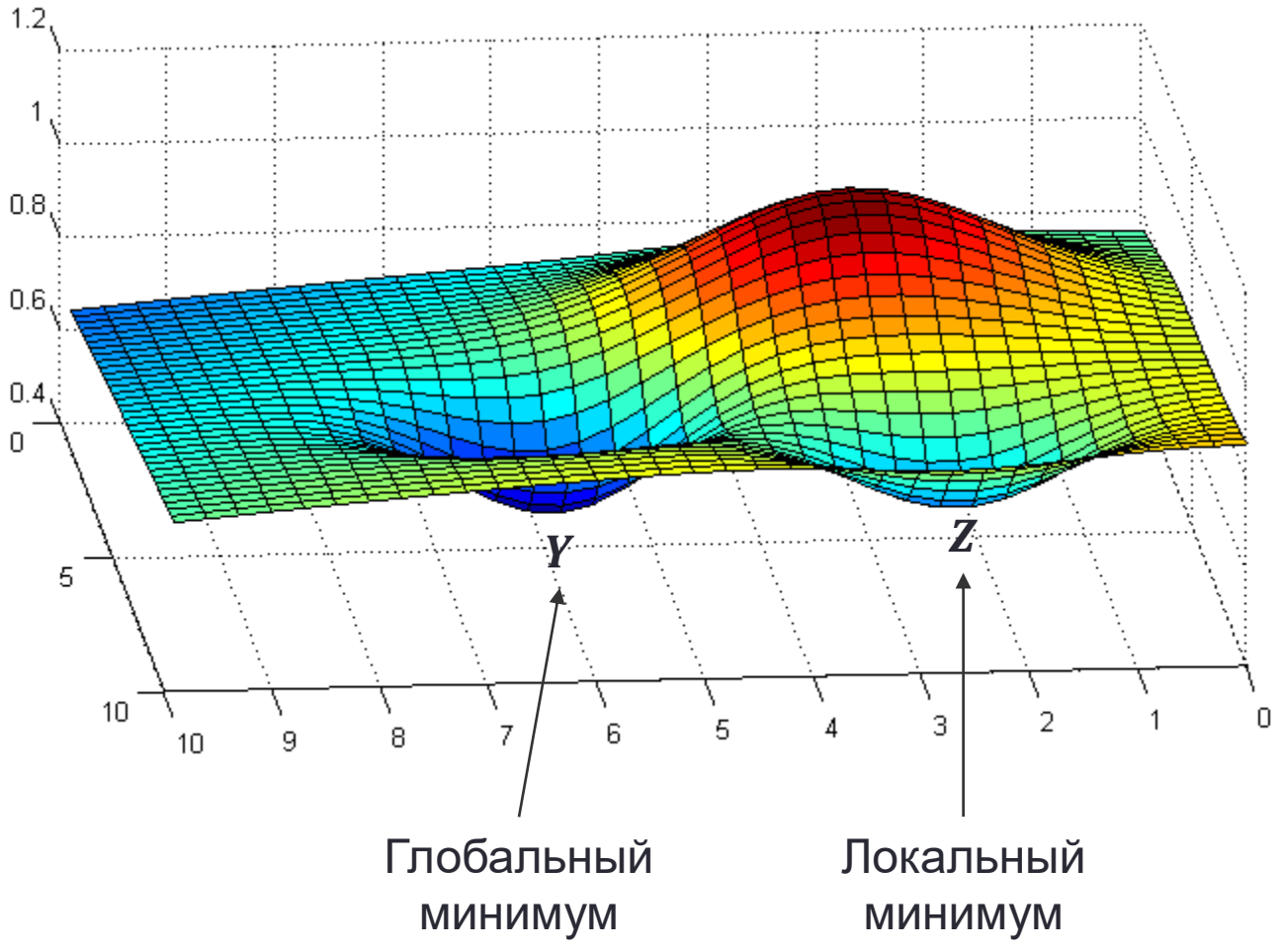


Понятие спуска по функции ошибки

Обучающая выборка			$A: w_1 = -5, w_2 = -8$		$B: w_1 = 8, w_2 = -4$	
x_1	x_2	t	Выход сети	Ошибка	Выход	Ошибка
-1	-1	0	$f_s((-1)*(-5)+(-1)*(-8)) = f_s(13) \approx 1$	$(1-0)^2 = 1$	≈ 0.018	≈ 0.0003
0	1	1	$f_s(-8) \approx 0.0003$	≈ 0.9994	≈ 0.018	≈ 0.9643
1	0	1	$f_s(-5) \approx 0.0067$	≈ 0.9866	≈ 0.997	≈ 0.00001
1	1	1	$f_s(-13) \approx 0$	≈ 1	≈ 0.982	≈ 0.0003
			$E_{SS}(A) \approx (1 + 0.9994 + 0.9866 + 1)/2 \approx 2$		$E_{SS}(B) \approx 0.5$	

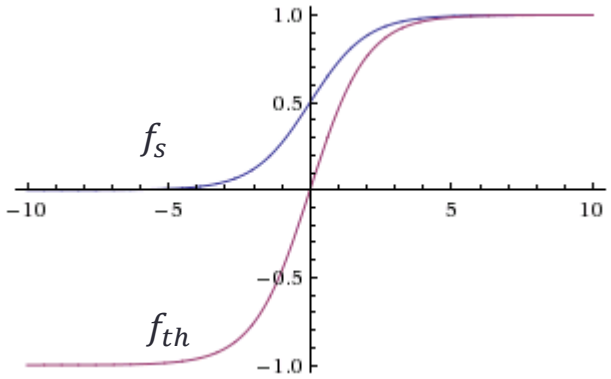


Минимумы функции ошибки

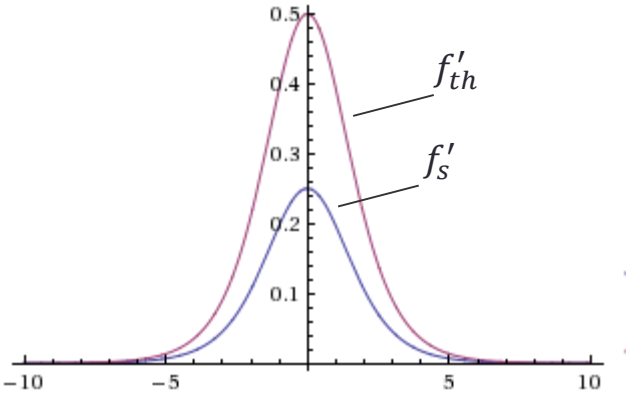


Производная и градиент функции ошибки

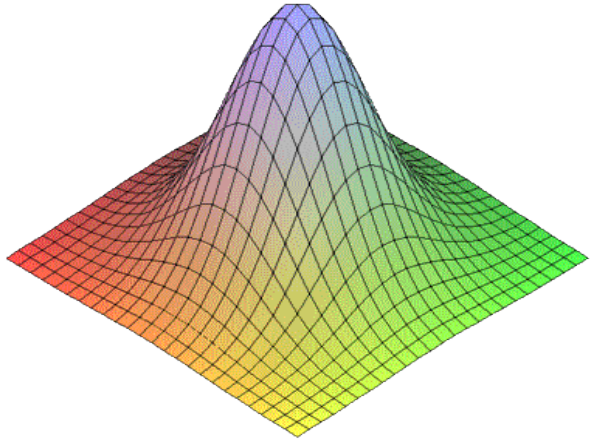
Сигмоида f_s и гиперболический тангенс f_{th}



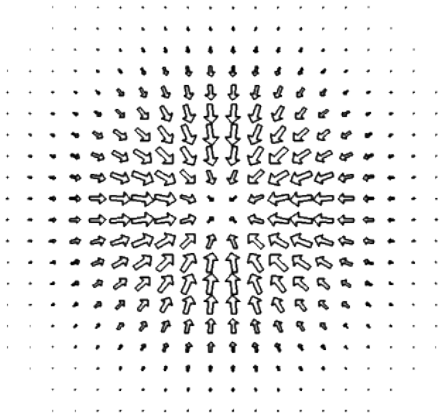
Производные сигмоиды f_s' и гиперболического тангенса f_{th}'



Функция от двух параметров g

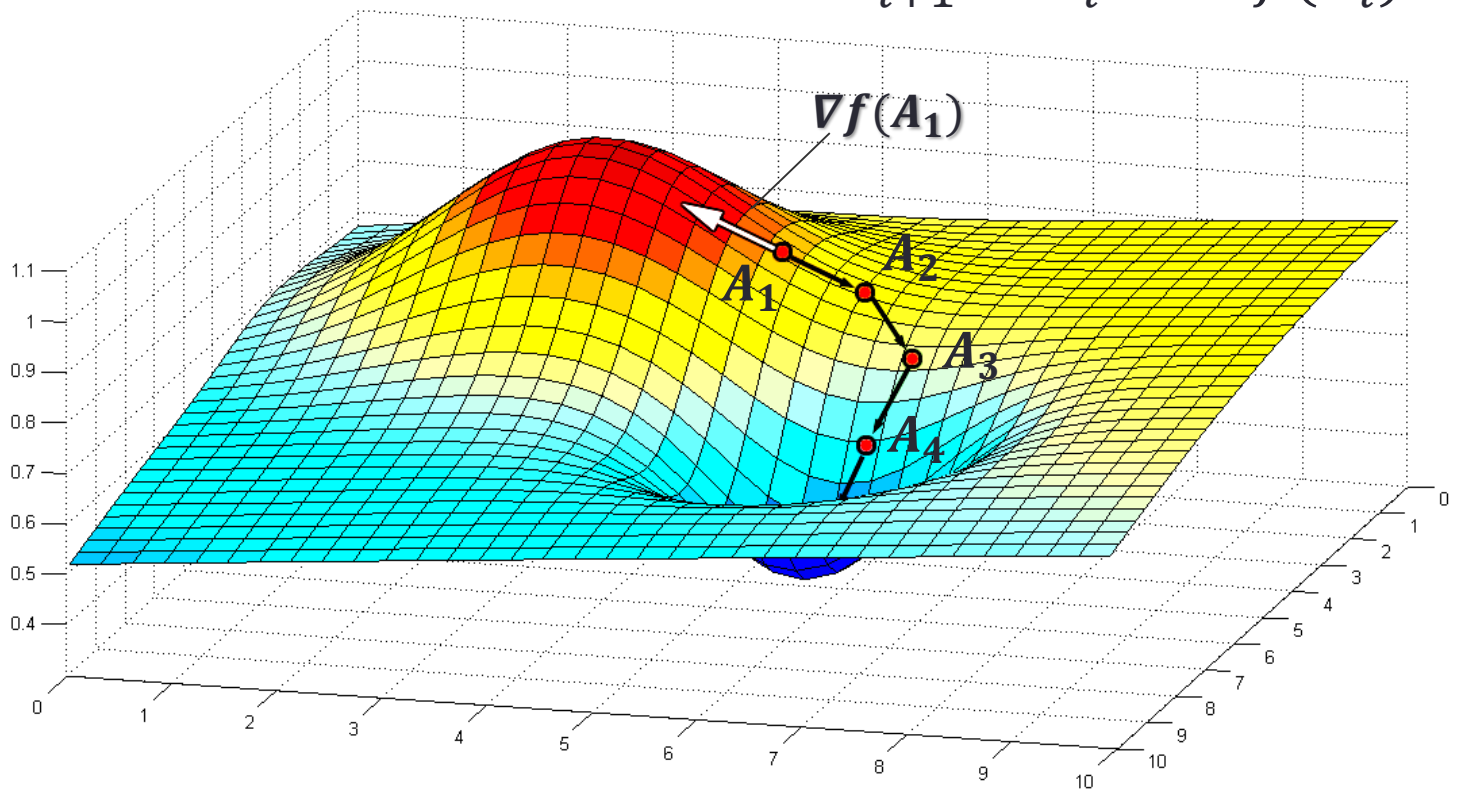


Градиент ∇g

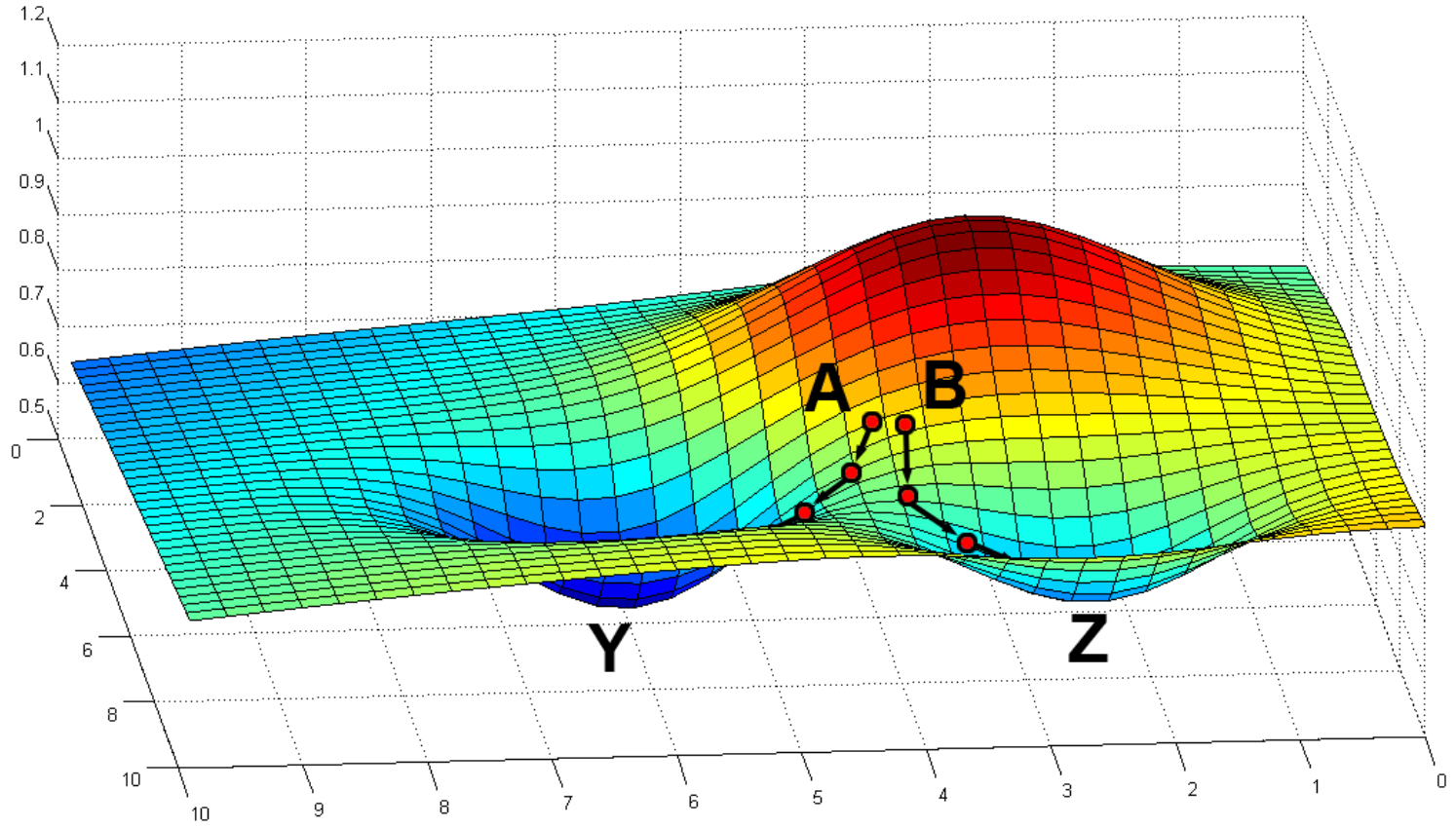


Градиентный спуск

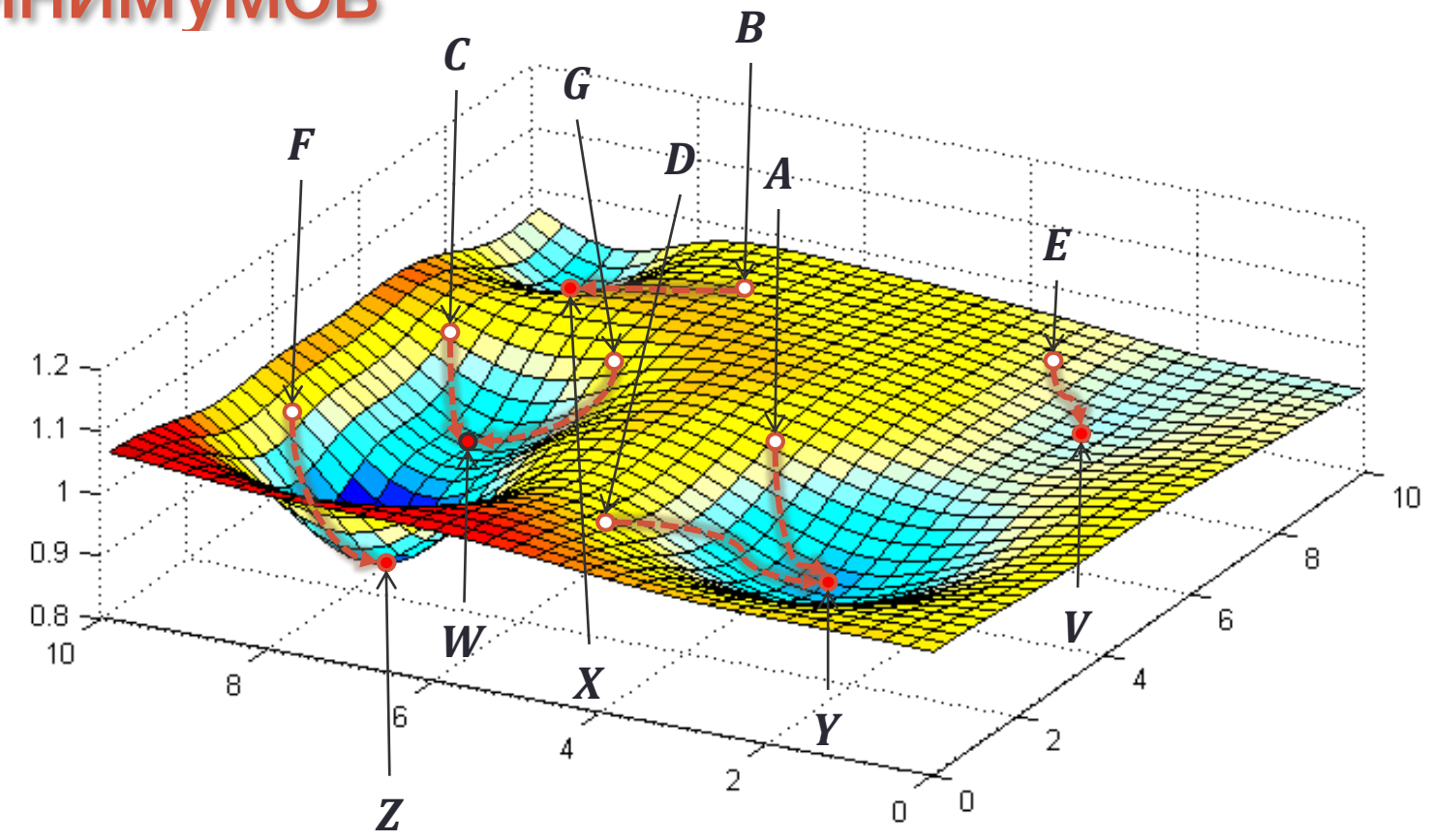
$$A_{i+1} = A_i - k\nabla f(A_i)$$



Ограничения метода градиентного спуска

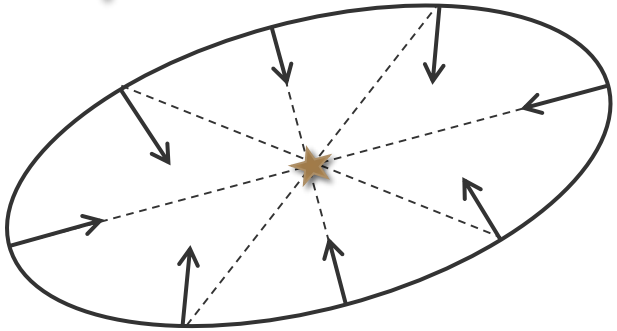


Решение проблемы локальных минимумов



$A \ldots G$ – начальные значения, $V \ldots Z$ – локальные минимумы

Градиентный спуск на квадратичной поверхности

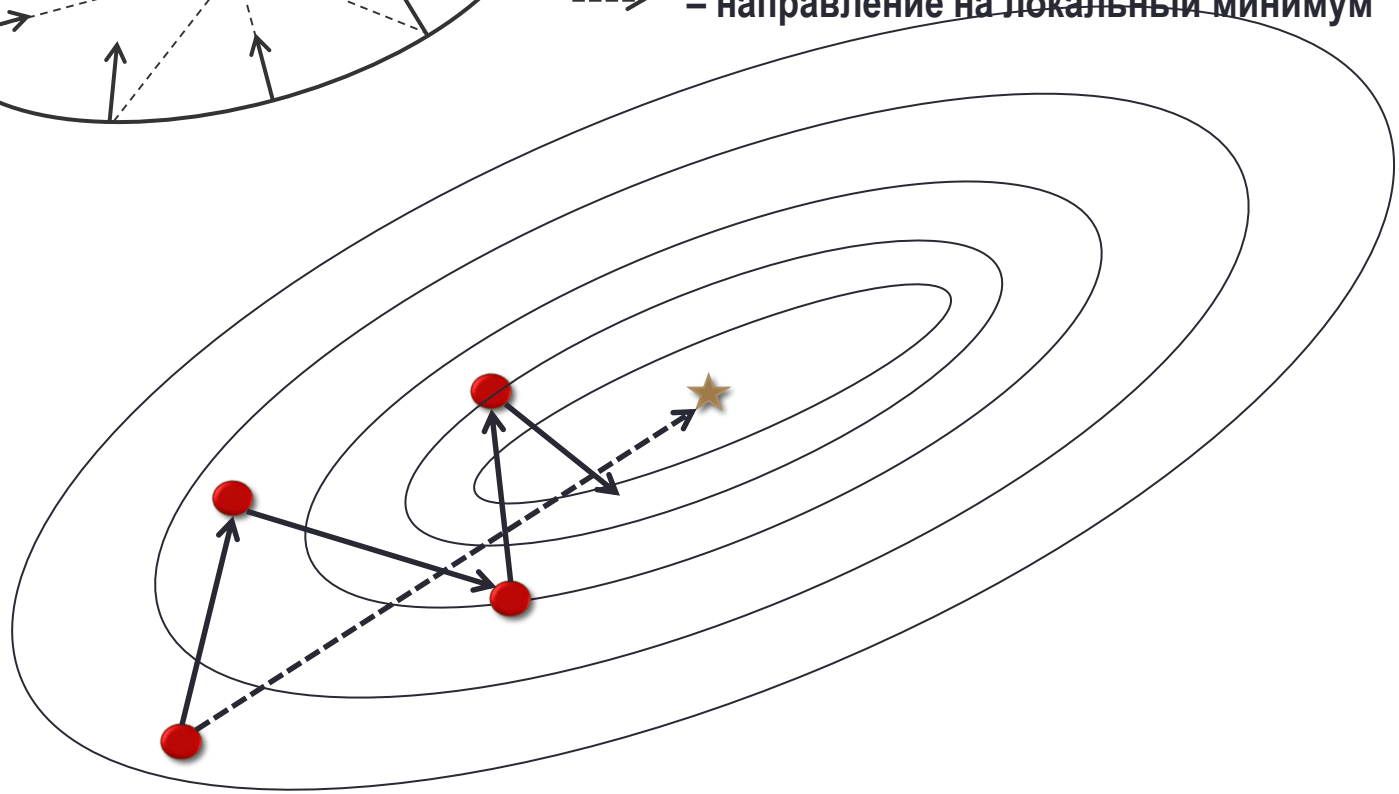


– линии равного уровня функции ошибки

– локальный минимум

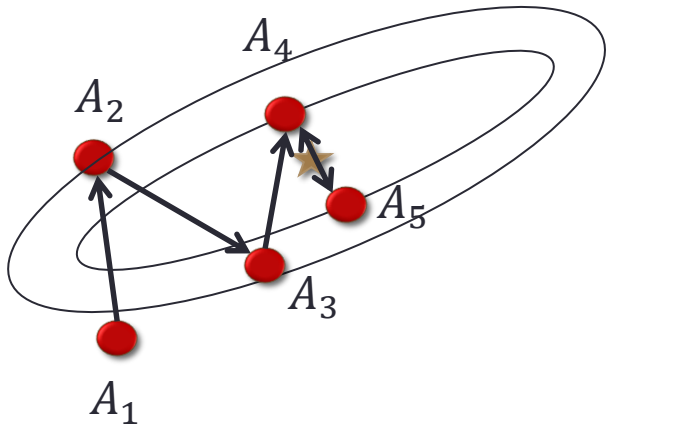
– направление градиентного спуска

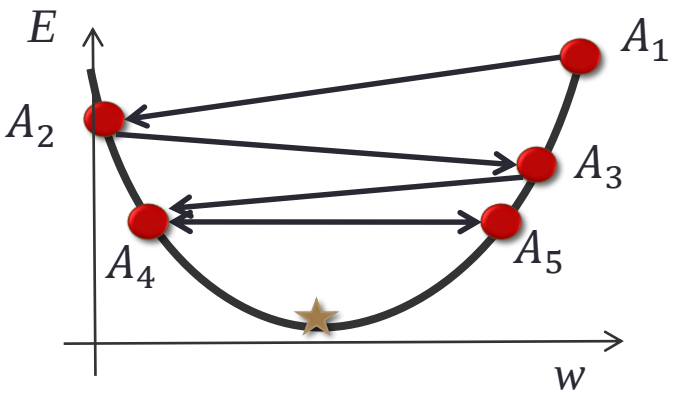
– направление на локальный минимум



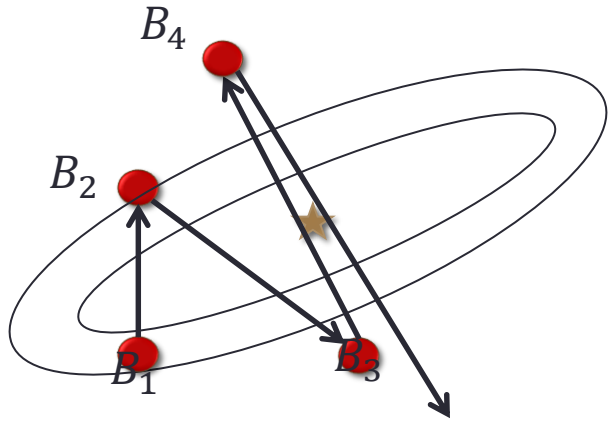
Влияние скорости на градиентный спуск

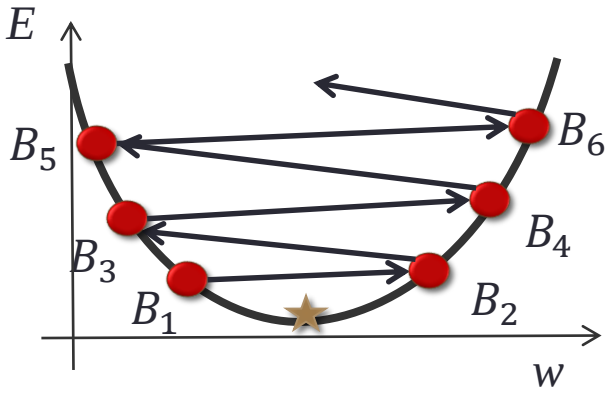
При большой скорости обучения алгоритм не может спуститься



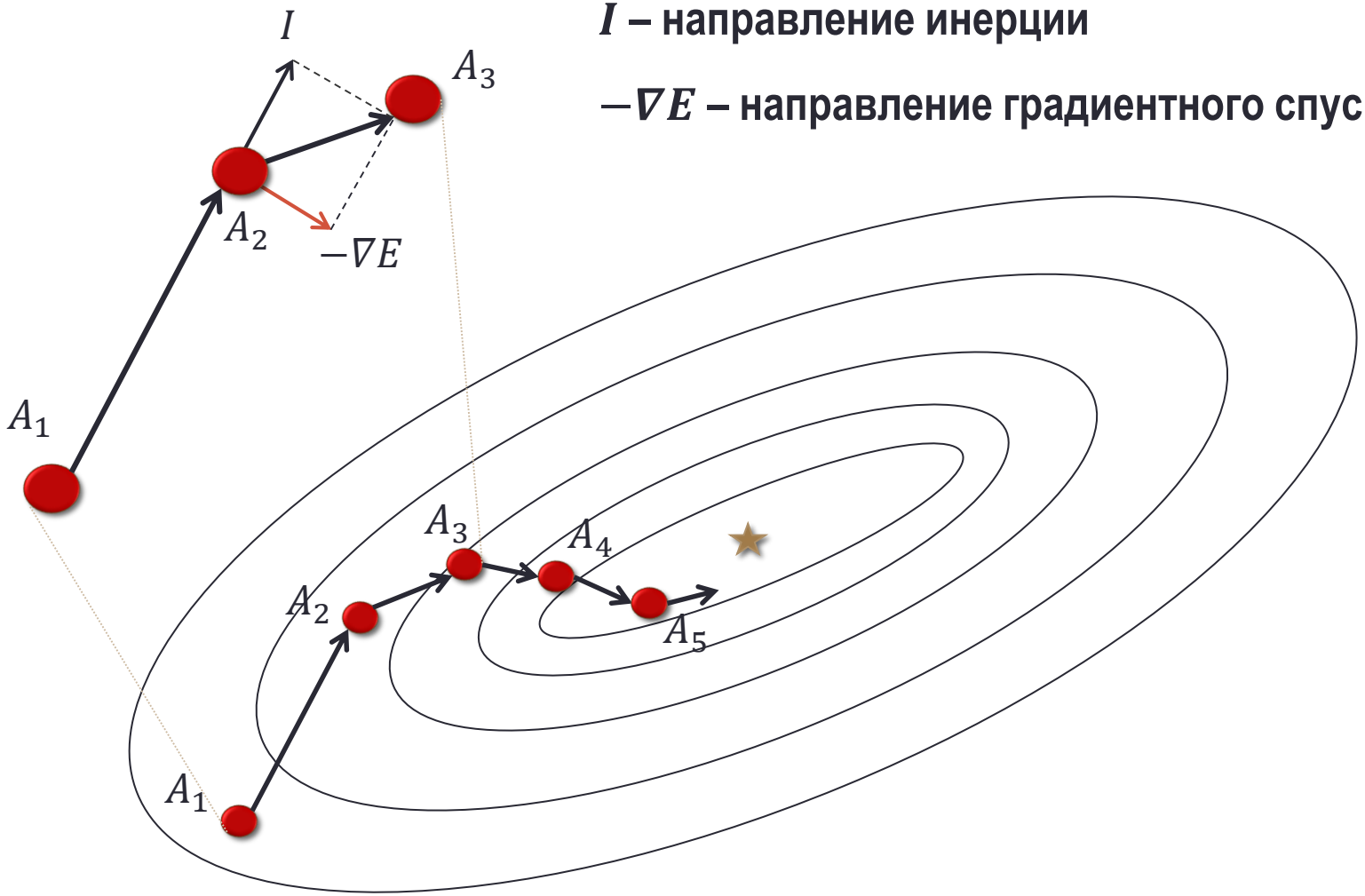


При большой скорости обучения алгоритм расходится





Градиентный спуск с инерцией



$\boldsymbol{I}$ **– направление инерции**

$\boldsymbol{-\nabla E}$ **– направление градиентного спуска**

Теперь Вы знаете

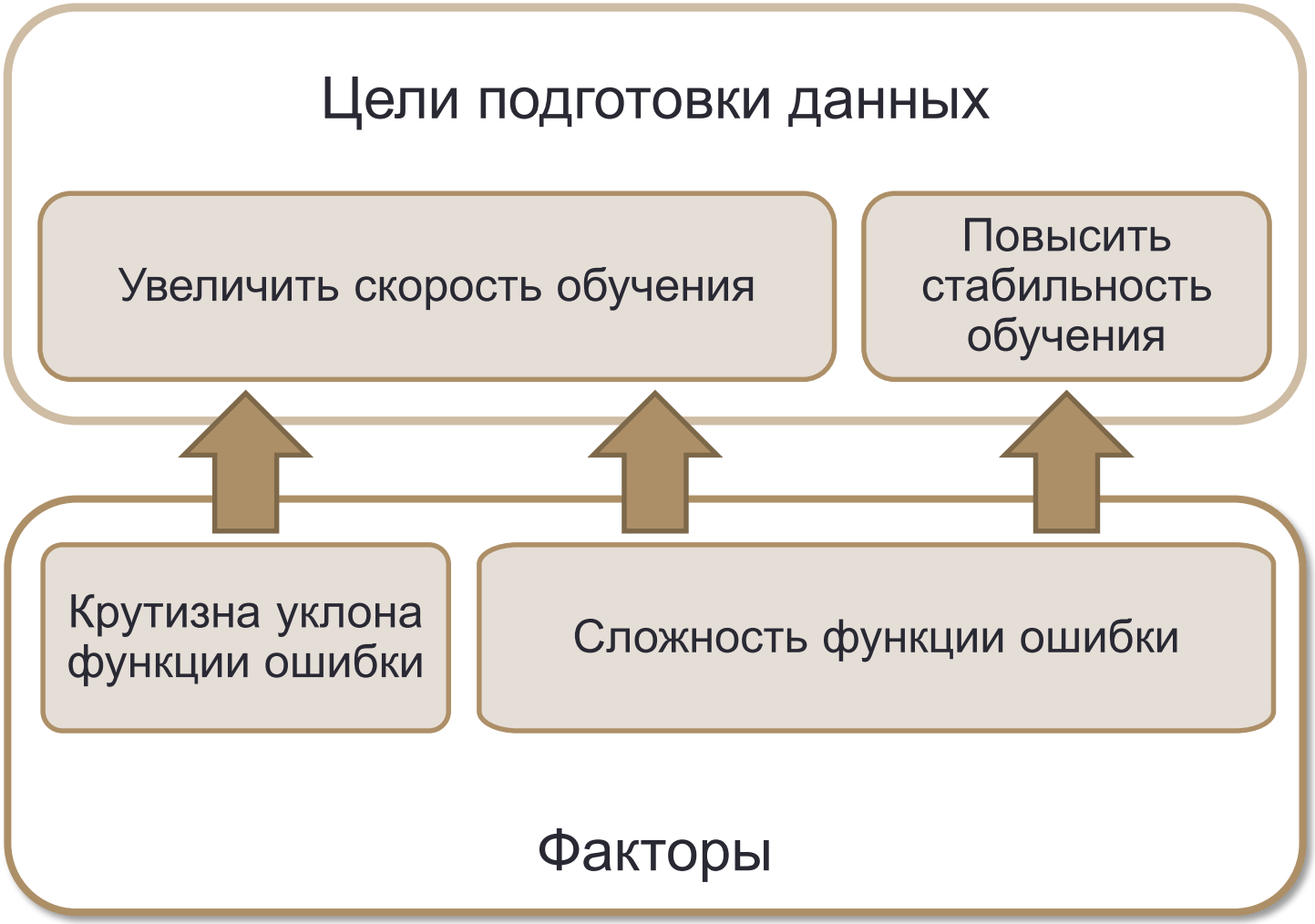
- Что такое функция ошибки нейронной сети
- От чего зависит значений функции ошибки
- Как алгоритм градиентного спуска применятся для обучения нейронных сетей
- Недостатки алгоритма градиентного спуска и его модификация

НАСТРОЙКА ОБУЧЕНИЯ НЕЙРОННЫХ СЕТЕЙ

Краткое содержание темы

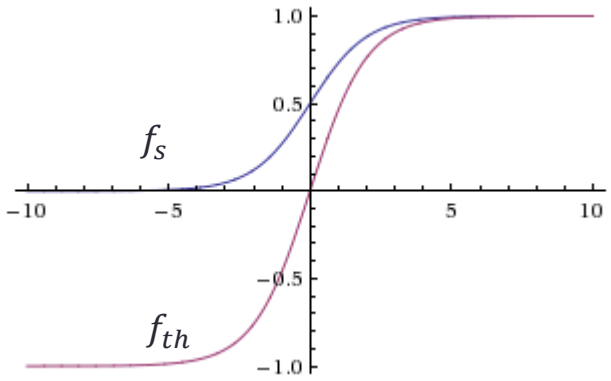
- Подготовка данных для анализа
- Режимы обучения нейронных сетей
- Проверка и улучшение качества модели

Подготовка данных перед обучением нейронной сети

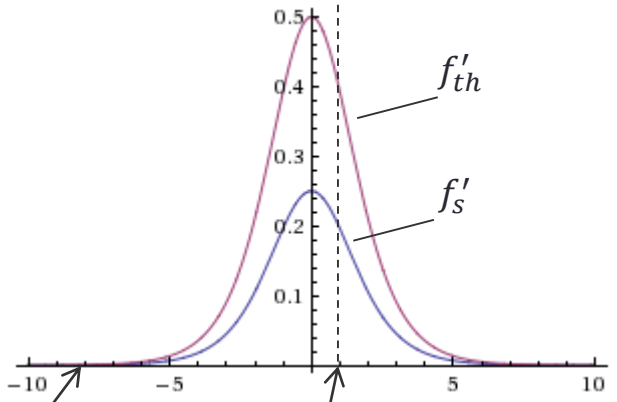


Влияние уклона функции ошибки на обучение сети

Сигмоида f_s и гиперболический тангенс f_{th}



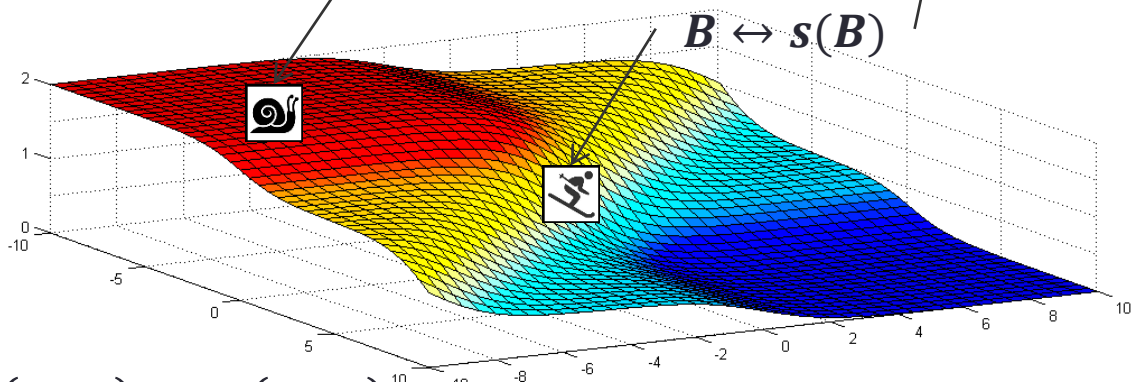
Производные сигмоиды f_s' и гиперболического тангенса f_{th}'



Функция ошибки E

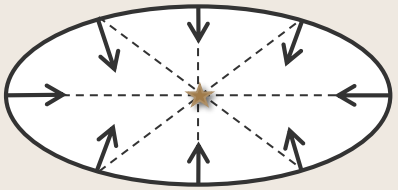
$\boldsymbol{A \leftrightarrow s(A)}$

$\boldsymbol{B \leftrightarrow s(B)}$



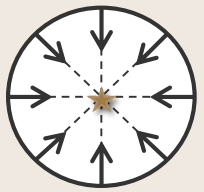
$$|\nabla E(A)| = f_s'\big(s(A)\big) \ll f_s'\big(s(B)\big) = |\nabla E(B)|$$

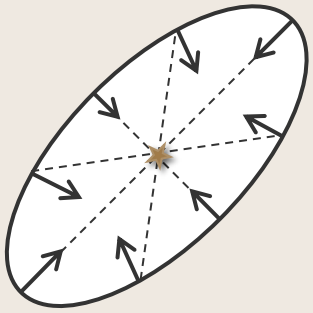
Влияние формы функции ошибки на обучение сети

Обучающая выборка			Функция ошибки
x_1	x_2	t	
0.1	10	2	
0.1	-10	0	

Изменить масштаб
$\tilde{x}_1 = x_1 * 10, \tilde{x}_2 = x2 * 0.1$

Сдвинуть
$\tilde{x}_1 = x_1 - 100, \tilde{x}_2 = x_2 - 100$

Обучающая выборка			Функция ошибки
$\tilde{x}_1$	$\tilde{x}_2$	t	
1	1	2	
1	-1	0	

Обучающая выборка			Функция ошибки
x_1	x_2	t	
101	101	2	
101	99	0	

Способы подготовки данных

Сдвиг данных $\tilde{x} = x + offset$

- Среднее значение всех переменных должно быть равно 0

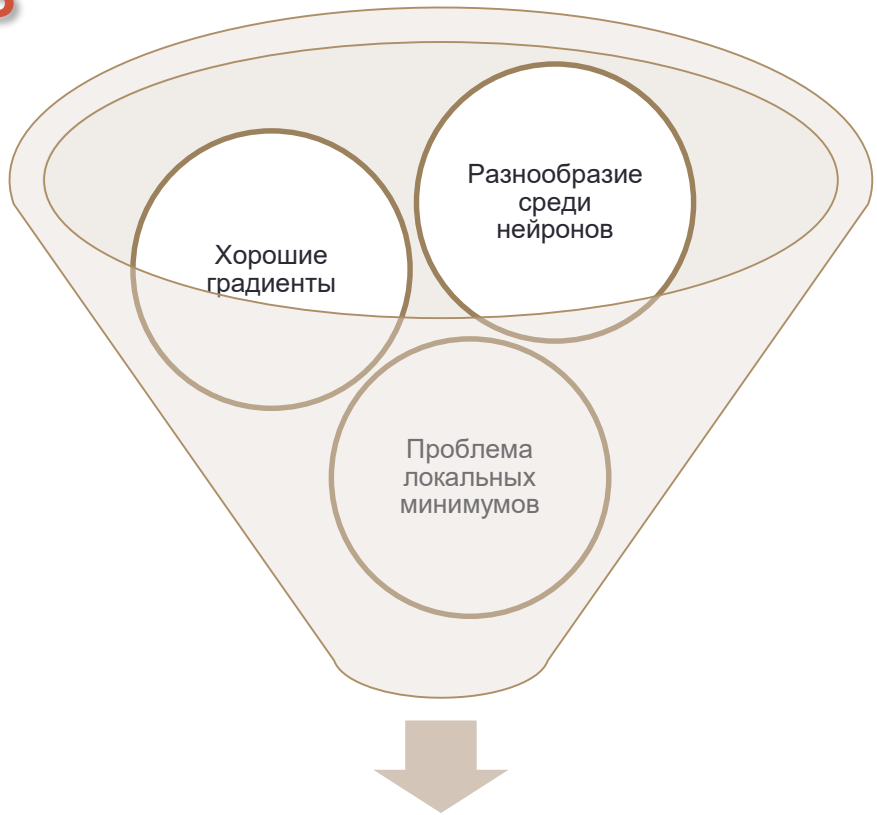
Масштабирование данных $\tilde{x} = scale * x$

- Диапазоны изменения переменных должны совпадать
- Переменные не должны принимать большие по модулю значения

Декорреляция данных

- Не должно быть тесной взаимосвязи между различными независимыми переменными

Выбор начальных параметров нейронов

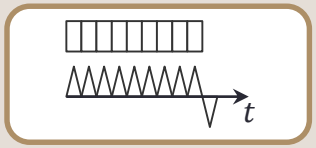


Задание параметров нейронов случайным образом

Выбор режима обучения сети

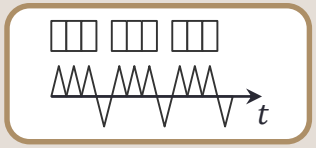
Полное пакетное обучение

- Вычислить направление шага по всей выборке
- Изменить параметры нейронов сети



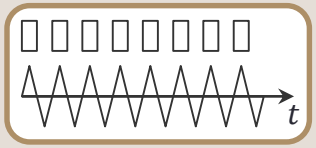
Мини-пакетное обучение

- Разбить обучающую выборку на части (пакеты)
- Для каждого пакета:
  - Вычислить направление шага по одному пакету
  - Изменить параметры нейронов сети



Интерактивное обучение

- Для всех примеров обучающей выборки:
  - Вычислить направление шага по одному примеру
  - Изменить параметры нейронов сети



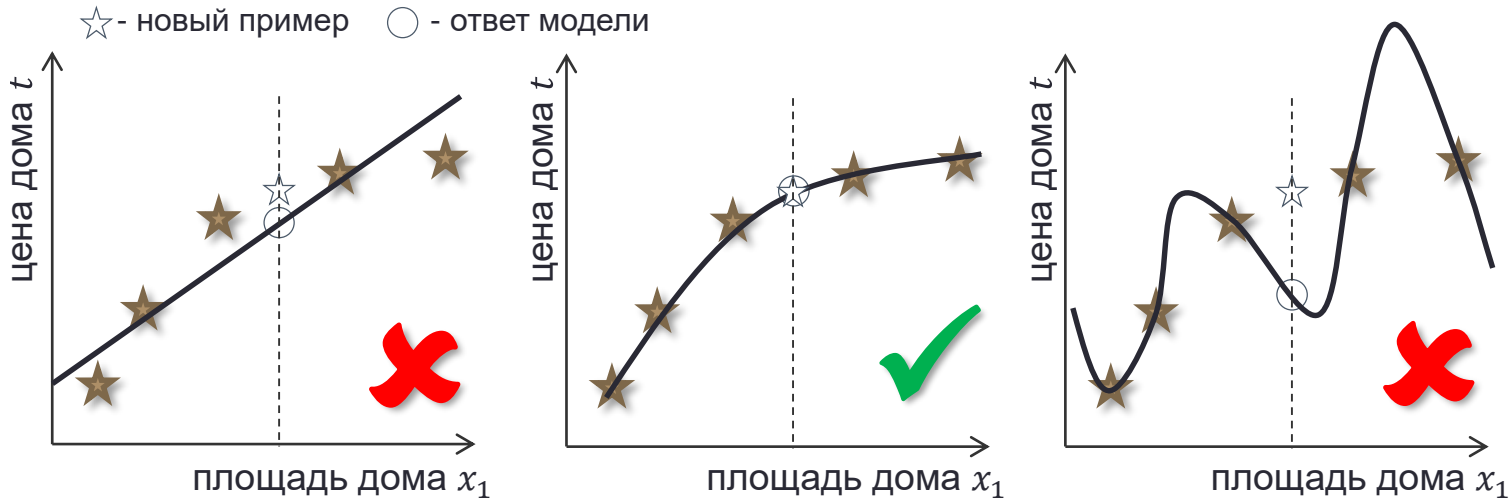
▯ - пример обучающих данных △ - вычисление направления шага ▽ - изменение параметров сети

Свойства режимов обучения

- **Полное пакетное обучение**
  - позволяет использовать усложненные алгоритмы спуска
- **Мини-пакетное обучение**
  - быстрее полного пакетного режима
  - распределение данных в каждом пакете должно быть близко к распределению в выборке в целом
  - рекомендуется для работы с большими избыточными наборами данных
- **Интерактивное обучение**
  - как правило, медленнее мини-пакетного метода

Проблемы переобучения и недообучения модели

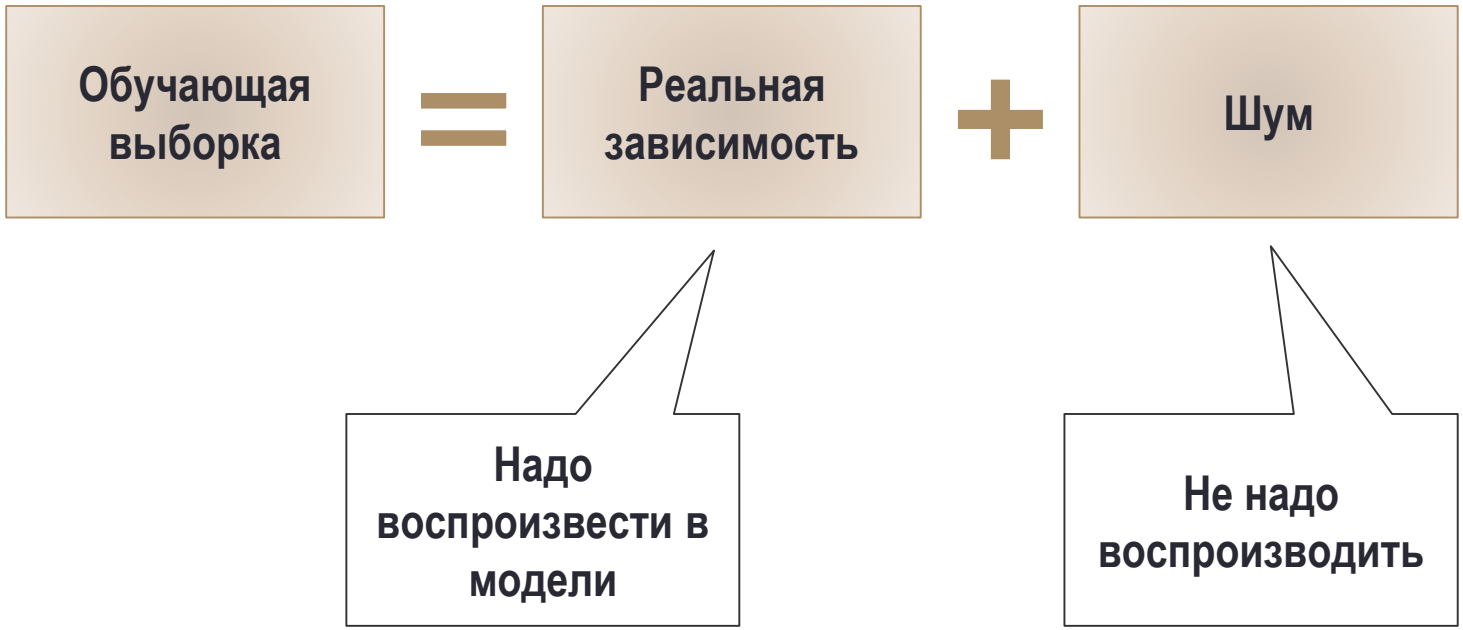
Модели полиномиальной регрессии



Проще — **Сложность модели** — Сложнее

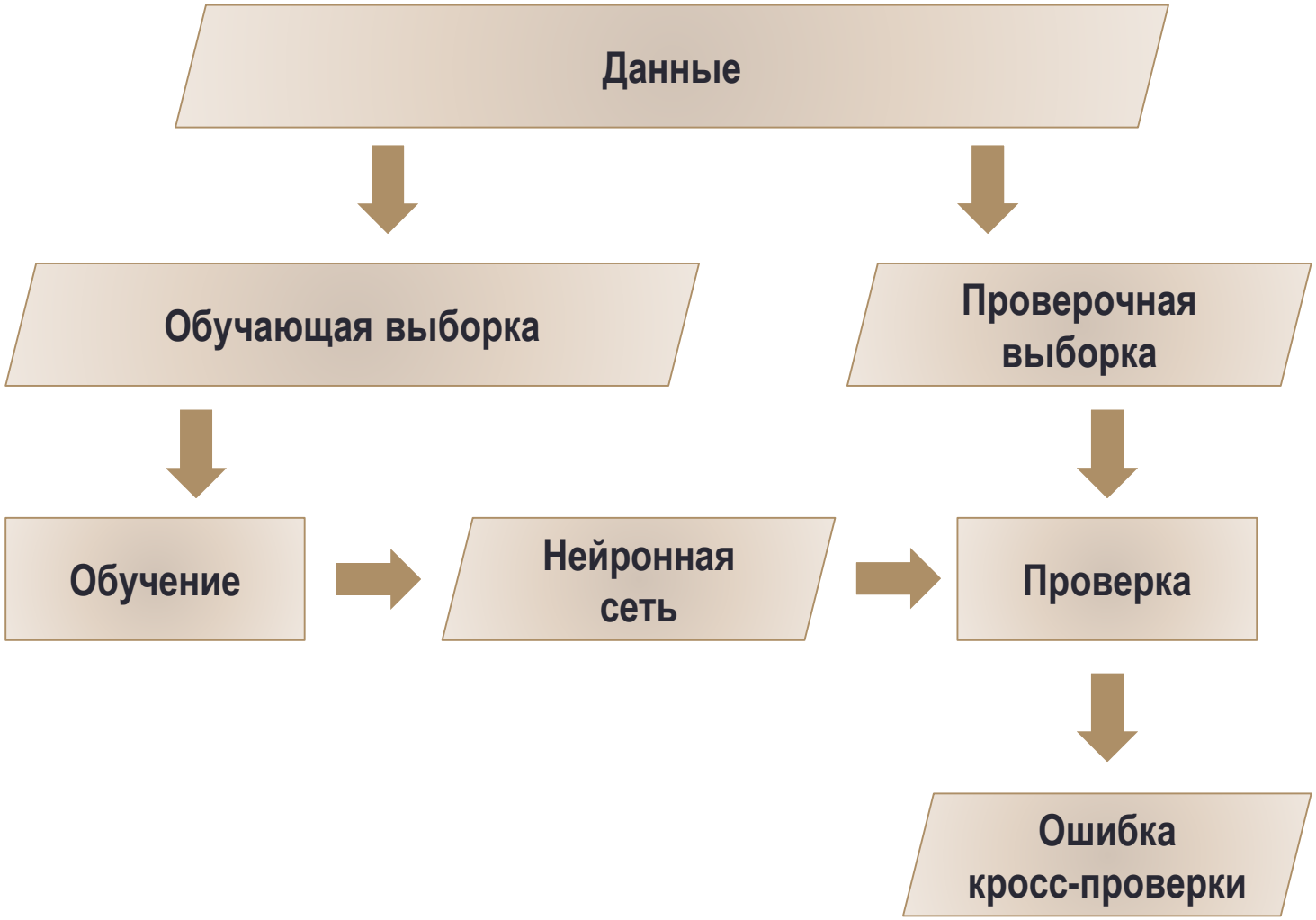
Линейная	Квадратичная	Полином 4 степени
$y = k_1x + k_0$	$y = k_2x^2 + k_1x + k_0$	$y = k_4x^4 + k_3x^3 + +k_2x^2 + k_1x + k_0$
Недообучение – модель слишком проста, чтобы передать зависимость	Сложность модели соответствует зависимости в данных	Переобучение – модель обучилось зависимости, которая сложнее, чем надо

Причины переобучения модели

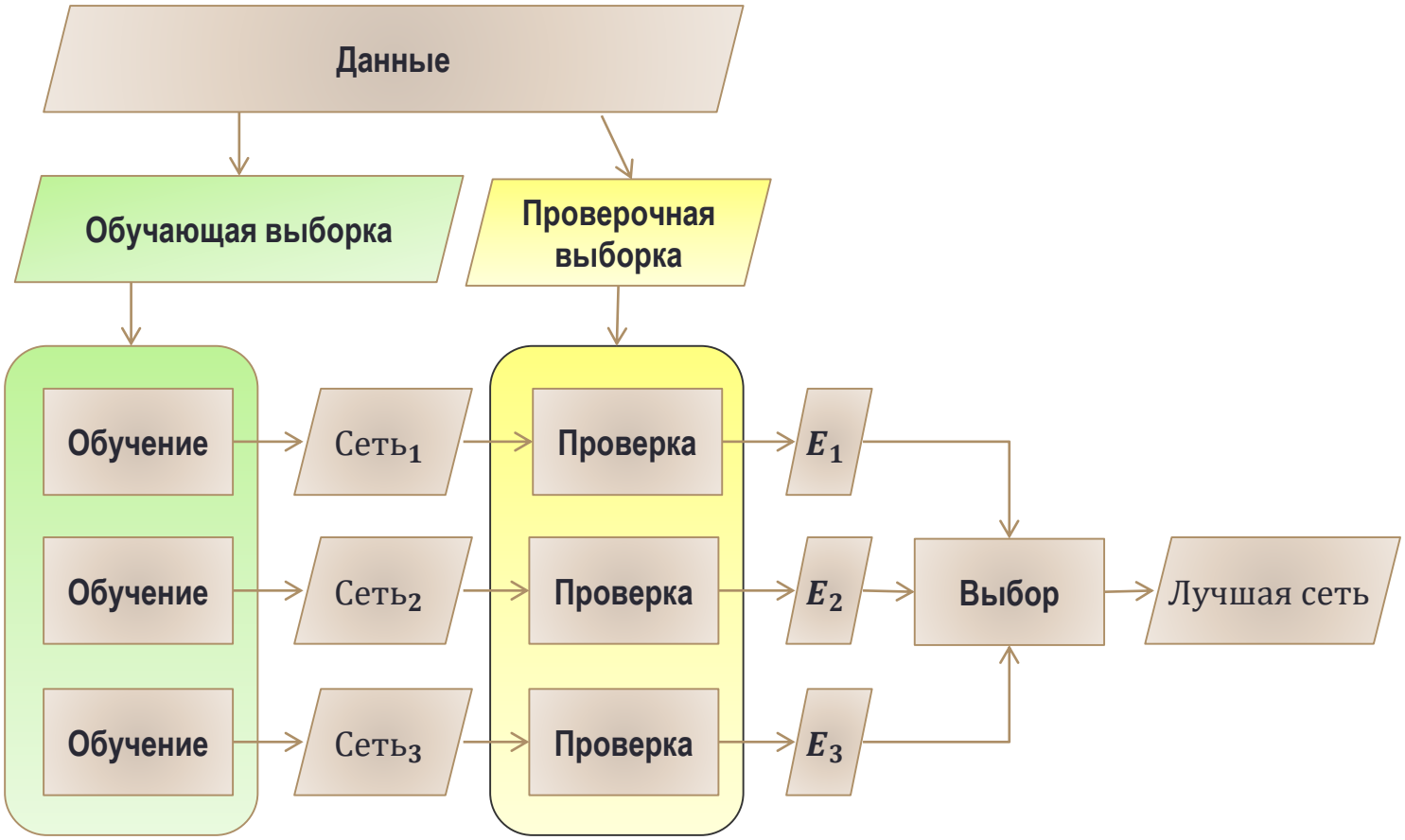


Склонность многослойных сетей к переобучению

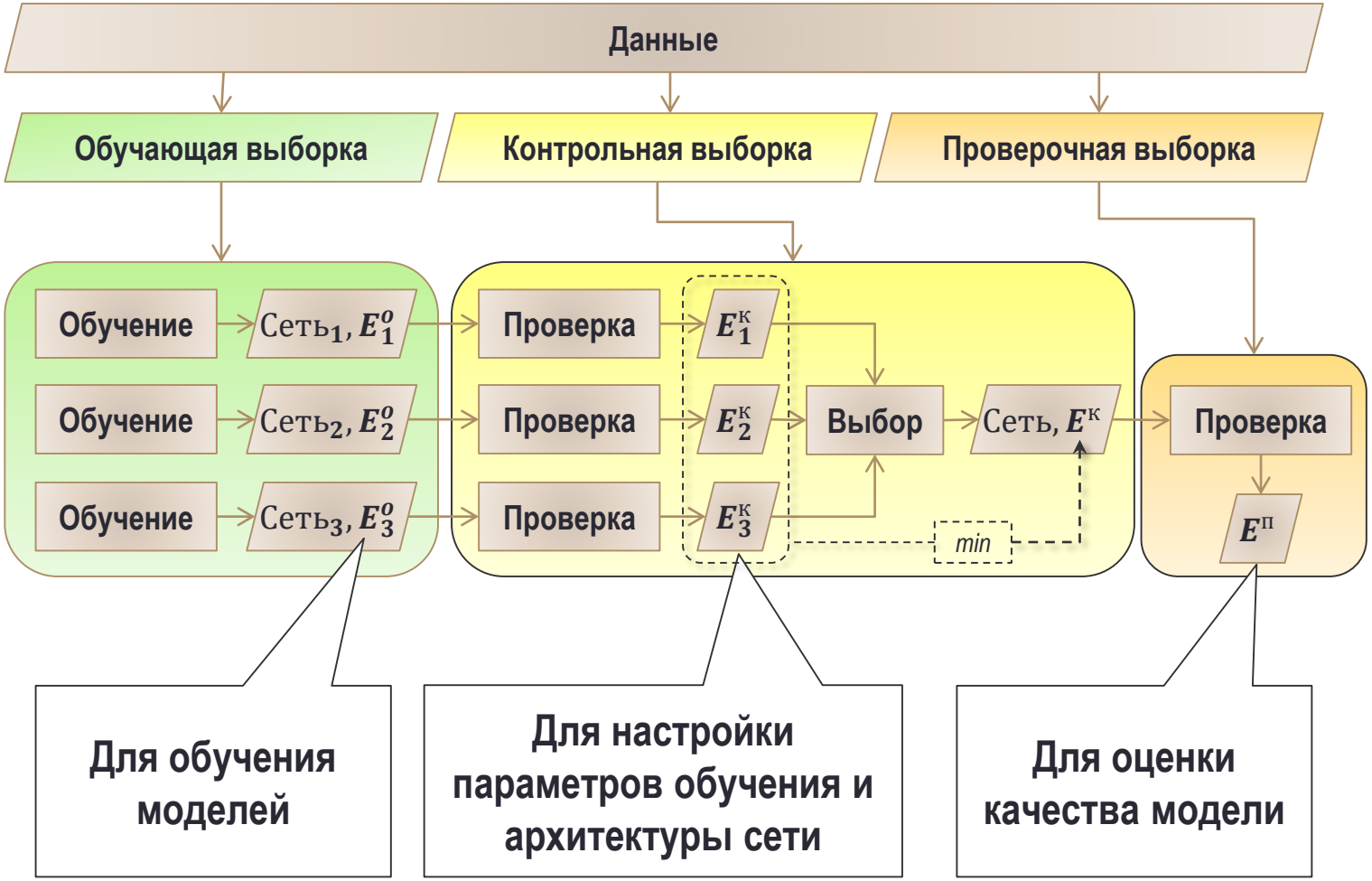
Метод кросс-проверки



Выбор архитектуры сети с помощью кросс-проверки



Оценка качества нейросети с помощью кросс-проверки



Анализ ошибок в процессе обучения модели

Ошибка на обучающей выборке	Ошибка на проверочной выборке	Состояние модели
Большая	Большая	Недообучение
Маленькая	Большая	Переобучение
Приемлемая	Приемлемая (выше ошибки на обучающей)	Хорошая модель
Большая	Маленькая	Так не бывает

Устранение недообучения и переобучения модели

Ранняя остановка

- Начать с малых значений весов и остановить обучение перед тем, как произойдет переобучение.

- Почему это работает?
  - Когда веса малы, все нейроны скрытых слоёв работают в линейном режиме – ёмкость сети мала.
    - Ёмкость равна ёмкости линейной сети.
  - В процессе обучения веса растут, скрытые нейроны начинают работать нелинейно, ёмкость сети повышается.

Другие способы борьбы с переобучением

- Регуляризация
- Усреднение моделей
- Байесовский метод
- Метод отсева (Dropout)
- ….
- Кросс-проверка!!!

L2-регуляризация

- Добавляем к функции ошибки компонент, который пропорционален квадрату значений весов

$$C = E + \frac{\lambda}{2}\sum_i {w_i}^2$$

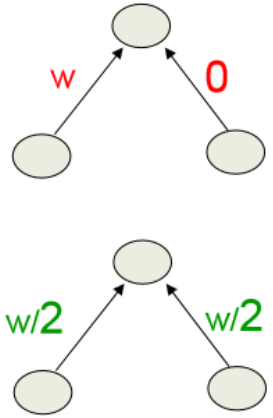
$$\frac{\partial C}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda w_i$$

- Это заставляет веса быть маленькими, кроме тех случаев, когда градиент ошибки велик.

Влияние L2-регуляризации

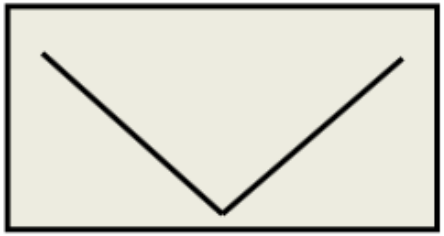
- Предотвращает использование весов, которые не нужны сети.
  - Обычно это сильно улучшает генерализацию.
  - Делает модель более гладкой, выходное значение меняется более плавно при изменении входа.

- Если у сети есть два почти одинаковых входа, то она будет предпочитать поровну распределить веса, а не концентрировать его на одном из вариантов.

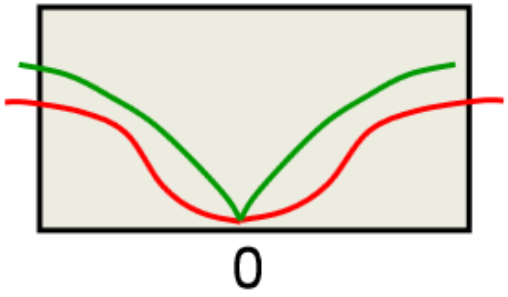


Другие виды регуляризации

- Можно штрафовать модуль веса, а не его квадрат.
  - Многие веса будут иметь значение ровно 0, что упрощает интерпретацию.

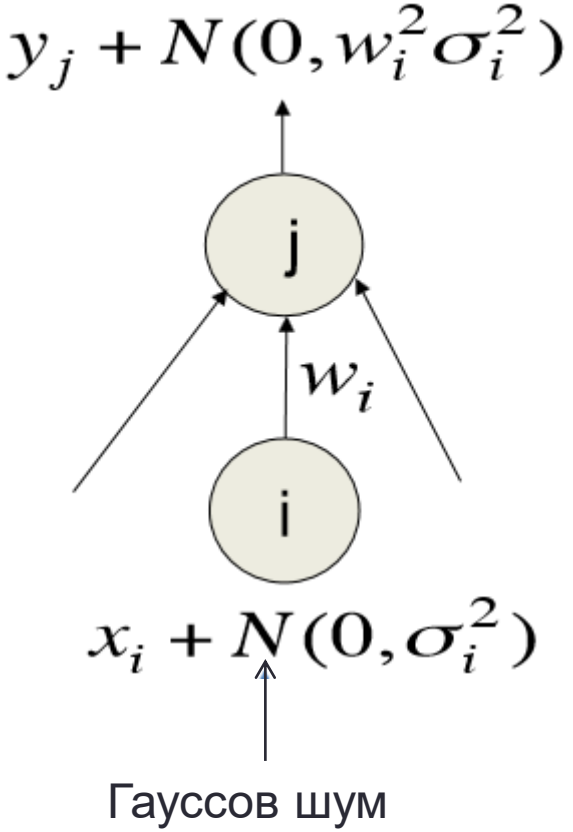


- Можно использовать пенализацию, которая оказывает незначительное влияние на большие веса.
  - Тогда в сети может быть несколько больших весов.



Регуляризация с помощью шума

- Допустим, мы добавим Гауссов шум ко входу нейрона.
  - Дисперсия шума усиливается пропорционально квадрату веса перед тем, как попасть в следующий слой.
- Это приводит к аддитивной добавке к ошибке.
  - Минимизация ошибки приводит к минимизации квадратов весов.



Регуляризация с помощью шума

- В линейной сети регуляризация с помощью шума эквивалентна L2-регуляризации:

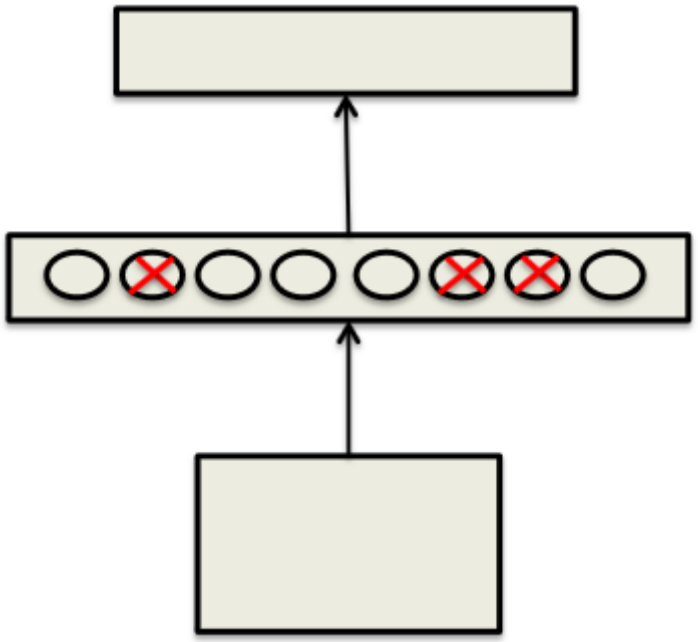
$$E\left[(y^{noisy} - t)^2\right] = (y - t)^2 + \sum_i w_i^2 \sigma_i^2 .$$

- Для нелинейных сетей эта эквивалентность не выдерживается в точности.
  - На практике шум может работать лучше, особенно для рекурсивных сетей.

МЕТОД ОТСЕВА

Отсев: эффективный метод усреднения для больших сетей

- Рассмотрим сеть с одним скрытым слоем
- Каждый раз подавая пример обучающего сигнала мы будем с вероятностью 0.5 исключать нейроны из сети.
- Таким образом мы случайно выбираем сеть из одной из 2^H архитектур.
  - Все архитектуры разделяют веса.
  - Разделение весов означает, что модели будут очень сильно регуляризованы.
  - Этот способ регуляризации значительно лучше L2 или L1 регуляризации, которые тянут веса к 0.
- Отсев замедляет обучение.



как делать прогноз?

- Можно выбрать много моделей и вычислить среднее геометрическое их выходных распределений.
- Проще использовать все нейроны скрытого слоя, но поделить выходящий из них сигнал пополам.
  - Это в точности соответствует вычислению среднего геометрического всех 2^H моделей.

Отсев в многослойных сетях

- Используем отсев с вероятностью 0.5 в каждом слое.
- Во время прогнозирования используем полную сеть с поделенными пополам весами.
  - Это не совсем соответствует точному среднему, но является достаточно хорошим приближением и считается быстро.
- Альтернативно, посчитать модель с отсевом несколько раз для одного входа и усреднить.
  - Заодно можно оценить степень уверенности в ответе.

Другой взгляд на отсев

- Если скрытые нейроны знают, кто есть рядом, то они могут кооперироваться на обучающих данных.
  - Сложные кооперации скорее всего будут работать неправильно на новых данных.
  - Большие, сложные заговоры не робастны.
- Если каждый скрытый нейрон может хорошо работать с комбинаторно-большим числом наборов соседей, то более вероятно, что он самостоятельно делает что-то полезное.
  - Но он также будет стремиться делать, что-то косвенно полезное при наличии результатов работы соседей.

Отсев во входном слое

- Можно использовать отсев и во входном слое, но сохранять нейроны с большей вероятностью.
  - Эта идея использована в «шумоподавляющих автоэнкодерах» (англ. denoising autoencoder).

Насколько хорошо работает отсев?

- Если сеть переобучается, то отсев обычно существенно помогает.
  - Любая сеть, для которой полезна ранняя остановка выиграет от использования отсева (но за счёт времени обучения)
- Если сеть не переобучается, то надо использовать сеть большего размера.

ГЛУБОКОЕ ОБУЧЕНИЕ

Глубокое обучение

- Алгоритм обратного распространения ошибки позволяет обучать сети глубиной 2 - 3 слоя. В более глубоких слоях градиент затухает и скорость обучения падает.

- Обучение глубоких сетей было достигнуто за счёт:
  - Усовершенствования алгоритмов обучения
  - Предобучения
  - Хорошей начальной инициализации
  - Более сложных архитектур
  - Использования функций активации ReLU, CReLU, LeakReLU
  - Увеличения числа итераций градиентного спуска за счёт увеличения вычислительных мощностей (GPU)
  - Transfer Learning

Обучение с разделением весов

- Допустим, нам требуется обеспечить линейную зависимость меду весами, например $w_i = w_j$.

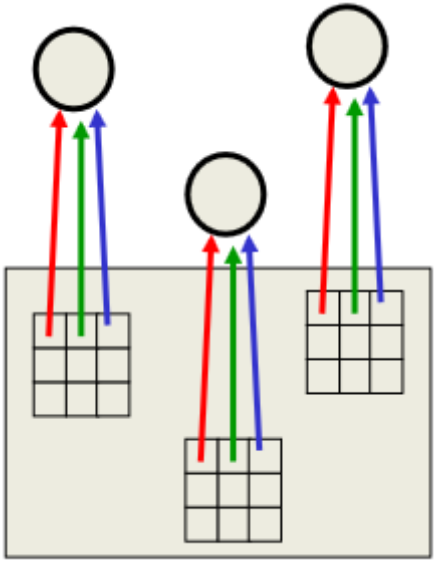
1. Инициализируем веса так, чтобы исходно наше ограничение выполнялось.
2. Вычисляем градиенты $\frac{\partial E}{\partial w_1}$ и $\frac{\partial E}{\partial w_2}$ обычным способом.
3. Для вычисления коррекций для этих весов используем значение градиента $\frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2}$.

- Это гарантирует, что $\Delta w_i = \Delta w_j$ и, соответственно, $w_i = w_j$ на всех шагах оптимизации.

Свёрточные сети

- Используем много копий одного детектора свойств для разных позиций.
  - Можно попробовать для разных поворотов/ориентаций.
  - Копирование сильно снижает число параметров.

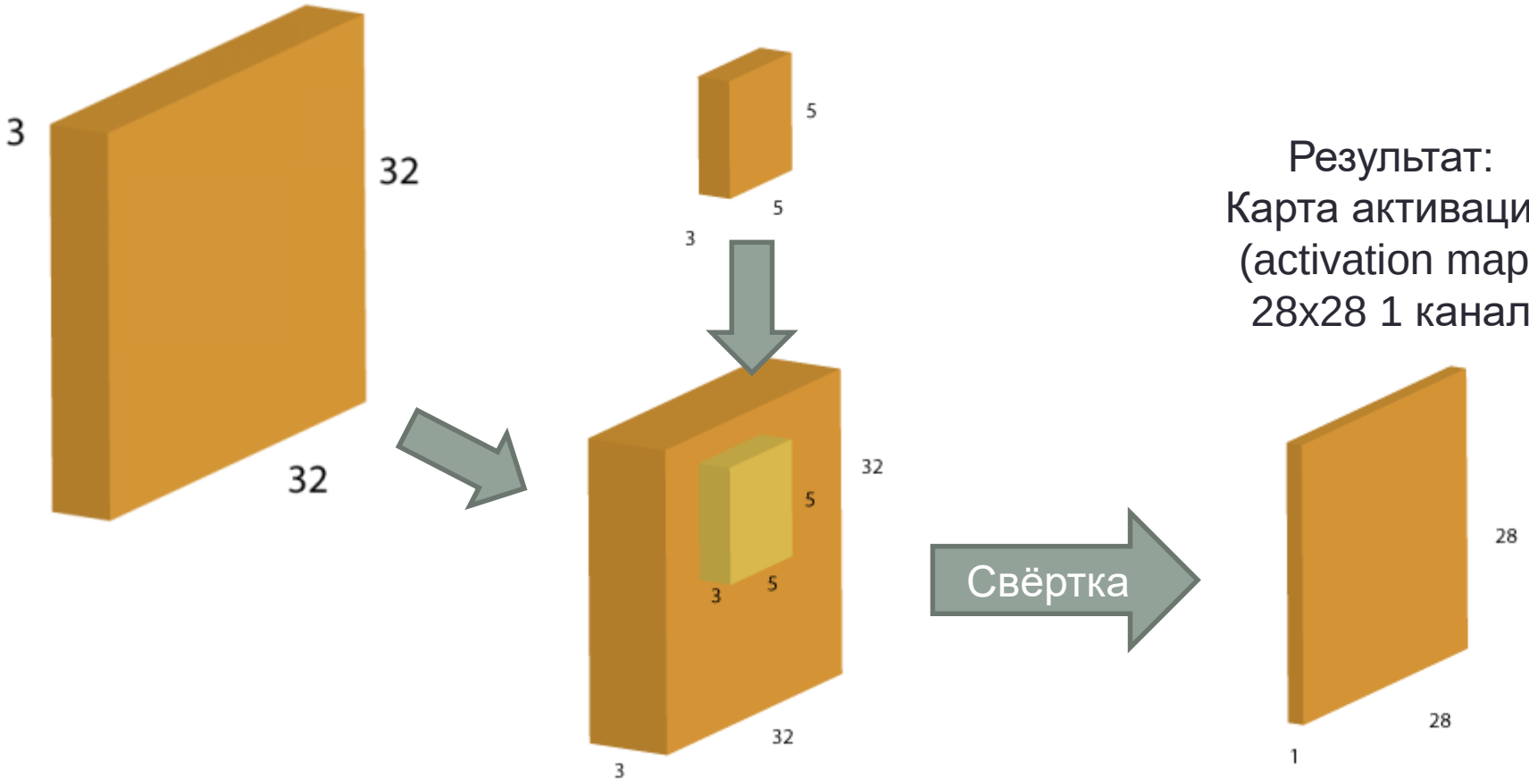
- Используем различные типы свойств, для каждого из которых есть отдельное поле реплицированных детекторов.
  - Позволяет каждый фрагмент изображения представить в нескольких разных представлениях.



Свёрточный слой

Вход сети: изображение
32x32 пиксела
3 канала (цвета)

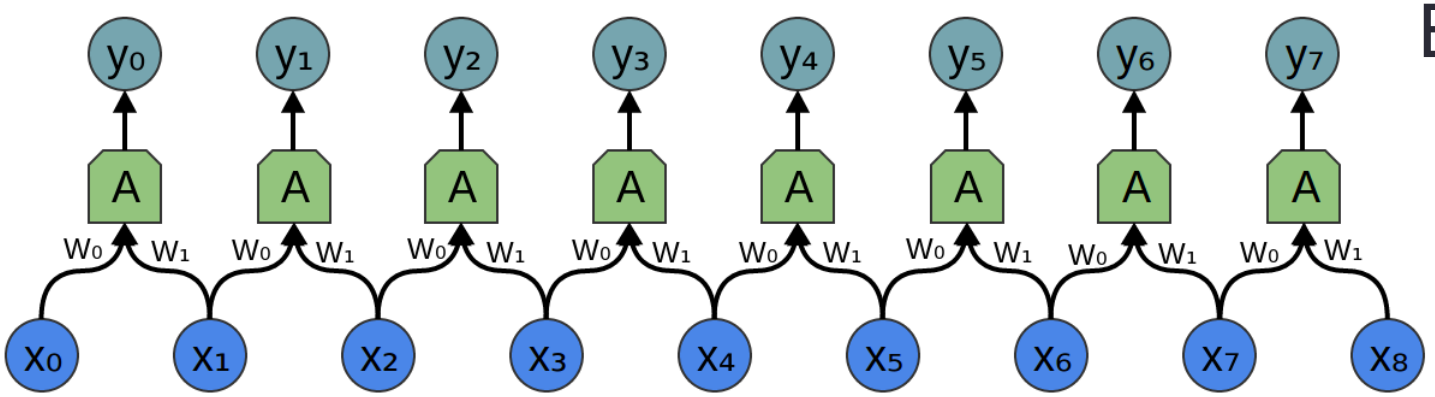
Фильтр: набор весов
меньшего размера
той же глубины



Результат:
Карта активации
(activation map)
28x28 1 канал

Свёрточный слой. Где нейроны?

Свёрточный слой для одномерных данных.
Свёртка фильтром размера 2.



Выход слоя

Копии нейрона

Вход слоя

Свёрточный слой

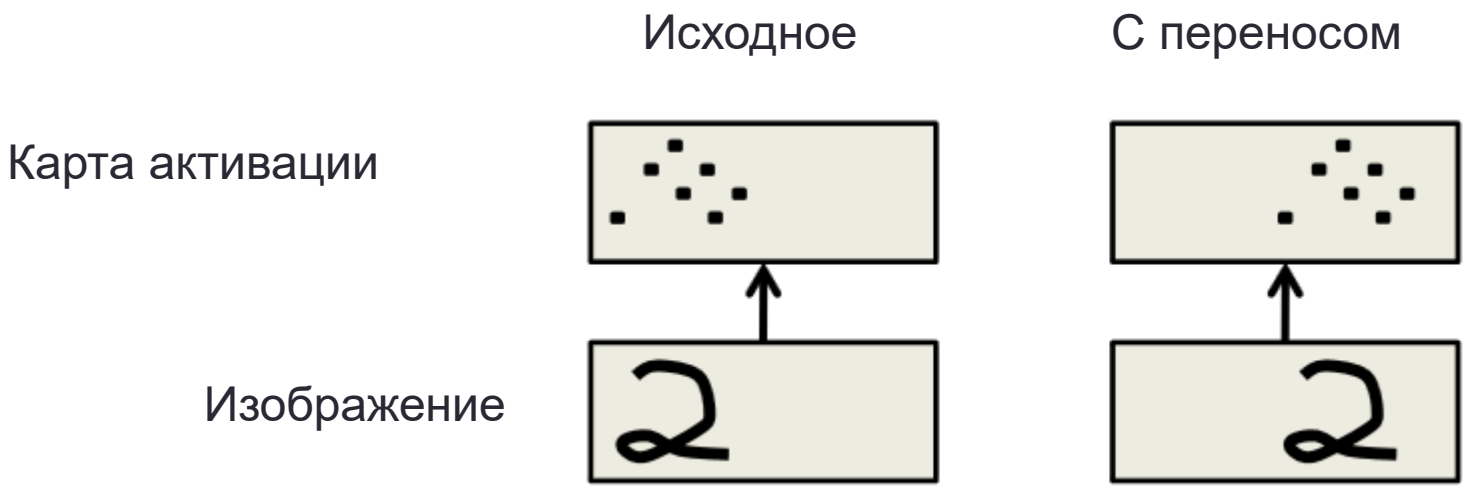
- Обычно свёрточный слой включает несколько разных фильтров
- Каждый фильтр выдает свою карту активации, которые вместе образуют многоканальный вход для следующего слоя.

Свёрточный слой. Параметры

- Размер входа: $W_1 \times H_1 \times D_1$
- Гиперпараметры слоя:
  - K – число фильтров;
  - F – размер фильтра ($F \times F$);
  - S – шаг применения фильтра (Stride);
  - P – размер «добивки» (Padding).
- Размер выхода: $W_2 \times H_2 \times D_2$
  - $W_2 = (W_1 + P - F)/S + 1$
  - $H_2 = (H_1 + P - F)/S + 1$
  - $D_2 = K$
- Число настраиваемых параметров: $(F * F * D_1 + 1) * K$

Что даёт копирование детекторов?

- Копирование не делает активности нейронов инвариантными к переносу.



- Инвариантное знание: если свойство полезно во время обучения в одной точке, то оно может быть полезно в другой во время тестирования.

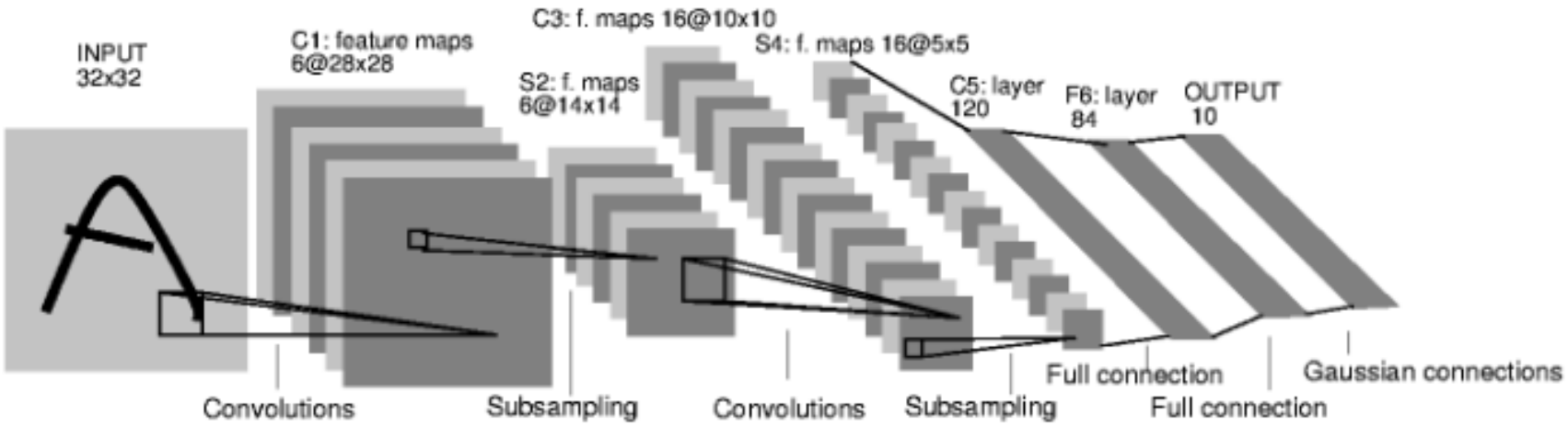
Концентрация свойств (pooling, subsampling)

- Чтобы получить небольшую инвариантность к переносу усредним выходные значения четырёх ближайших детекторов, чтобы получить один выход для следующего уровня.
  - Это уменьшает число входов следующего уровня и позволяет использовать больше различных полей детекторов.
  - Лучше использовать максимум, а не среднее.

- Проблема: после нескольких слоёв концентрации теряется информация о точных местах расположения свойств.
  - Становится невозможным использовать точные пространственные отношения между свойствами высокого уровня.

Архитектура LeNet5

Yann LeCun разработал архитектуру и обучил сеть для распознавания рукописного текста.



Ошибки LeNet5

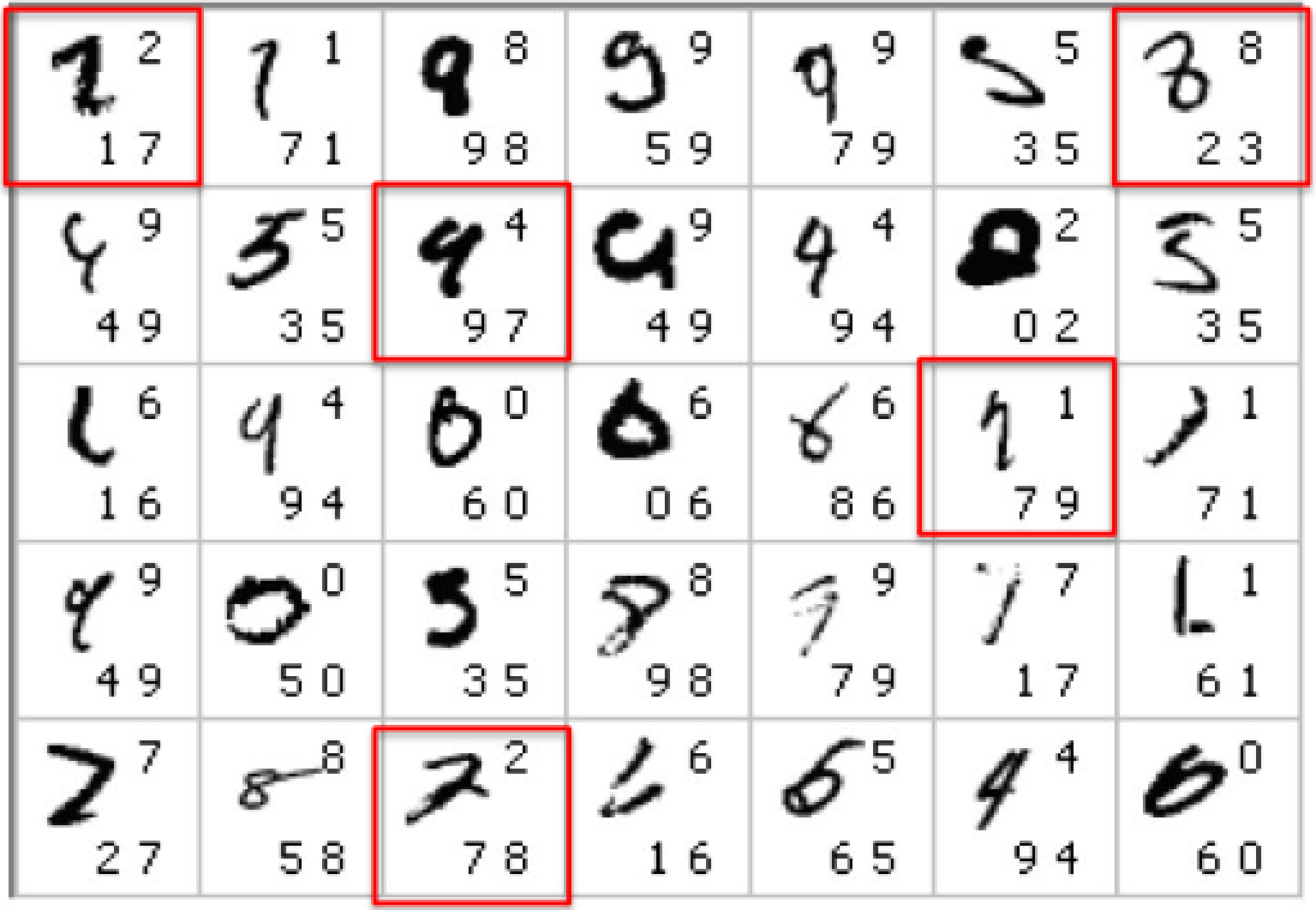
Другие способы достигнуть инвариантности

- Мы можем вложить наши знания в сеть, используя соответствующие:
  - Архитектуру связей.
  - Разделение весов.
  - Функции активации нейронов.
- Это менее жёсткий способ, чем ручной выбор свойств, но он, тем не менее, диктует сети определённый способ решения задачи.
- Альтернативно, мы можем использовать наши знания чтобы создать больше обучающих данных.
  - Это может потребовать много работы.
  - Это увеличит время обучения.
- Это позволит оптимизации найти хитрые способы использования многослойной архитектуры, о которых мы не догадывались.
  - И мы может быть никогда не поймём, какие именно.

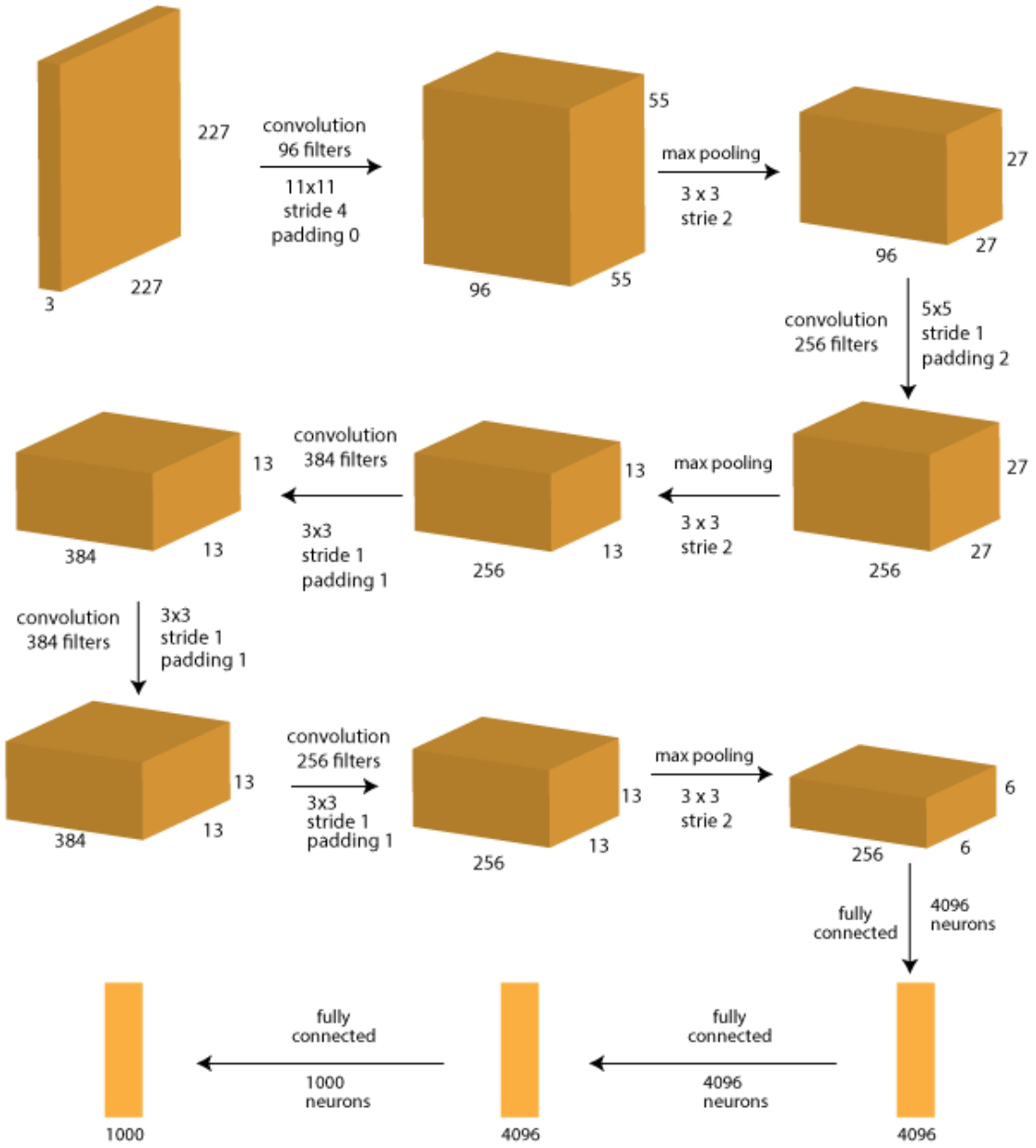
Метод грубой силы (Data Augumentation)

- LeNet использует знания о инвариантностях, выраженные следующими способами:
  - Локальные связи.
  - Разделение весов.
  - Концентрация свойств.
- Это даёт 80 ошибок на базе NMIST.
  - Используя преобразования входа и другие приёмы можно довести до ~40 ошибок.
- Ciresan и другие (2010) использовал знание о инвариантностях создав огромный объём обучающих данных.
  - Для каждого изображения они создали множество дополнительных примеров используя различные трансформации.
  - Затем они обучили на GPU большую многослойную сеть простой архитектуры без заметного переобучения.
- Они получили 35 ошибок.

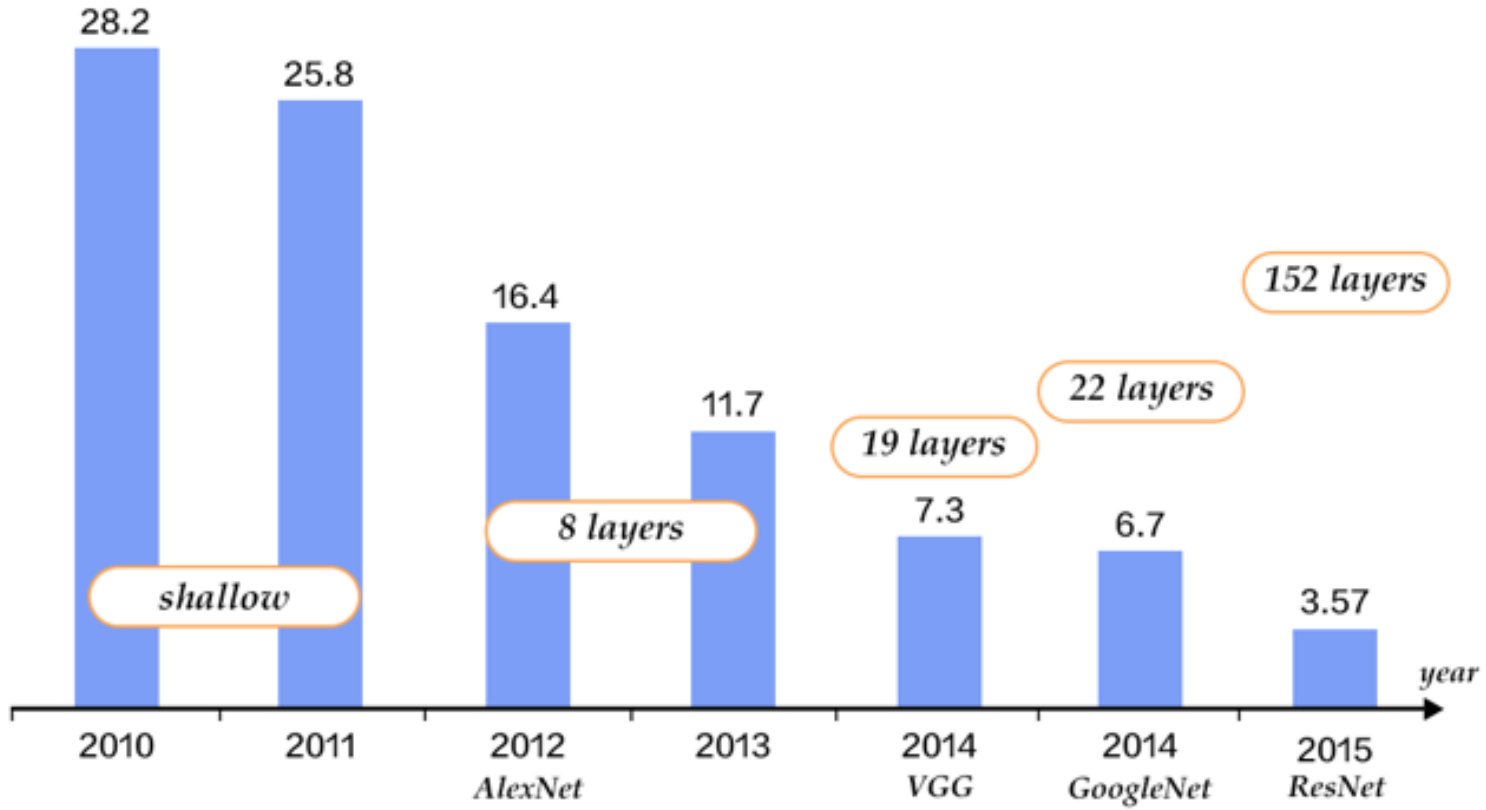
Ошибки сети Ciresan



AlexNet


227
227
3
convolution
96 filters
11x11
stride 4
padding 0
55
55
96
max pooling
3 x 3
strie 2
27
27
96
convolution
256 filters
5x5
stride 1
padding 2
27
27
256
max pooling
3 x 3
strie 2
13
13
256
convolution
384 filters
3x3
stride 1
padding 1
13
13
384
convolution
384 filters
3x3
stride 1
padding 1
13
13
384
convolution
256 filters
3x3
stride 1
padding 1
13
13
256
max pooling
3 x 3
strie 2
6
6
256
fully
connected
4096
neurons
fully
connected
4096
neurons
4096
fully
connected
1000
neurons
1000

Глубокие нейронные сети в конкурсе ImageNet



КОМИТЕТЫ СЕТЕЙ

Комитеты сетей

- Обучающий набор

  $T=\{(x^1,t_1),\ldots,(x^m,t_m)\}$

  $x^i=(x^i_1,\ldots,x^i_n)$

- Пусть у нас есть N сетей, обученных на этом наборе. Каждая сеть вычисляет функцию f_i.

- Тогда комитет сетей вычисляет функцию

$$f = \frac{1}{N}\sum_{i=1}^{N} f_i.$$

Комитеты сетей - ошибка

Пусть e^i_j – ошибка *i*-ой сети на *j*-ом примере обучающей выборки.

Ошибка комитета на всей выборке есть

$$Q = \sum_{i=1}^{m} \left(t_i - \frac{1}{N} \sum_{j=1}^{N} f_j(\mathbf{x}^i) \right)^2 .$$

Определим матрицу ошибок

$$\mathbf{E} = \begin{pmatrix} e_1^1 & e_2^1 & \cdots & e_m^1 \\ \vdots & \vdots & \ddots & \vdots \\ e_1^N & e_2^N & \cdots & e_m^N \end{pmatrix}$$

Комитеты сетей - ошибка

Ошибка комитета на всей выборке есть

$$Q = \sum_{i=1}^{m} \left(t_i - \frac{1}{N} \sum_{j=1}^{N} f_j(\mathbf{x}^i) \right)^2 .$$

Определим матрицу ошибок

$$\mathbf{E} = \begin{pmatrix} e_1^1 & e_2^1 & \cdots & e_m^1 \\ \vdots & \vdots & \ddots & \vdots \\ e_1^N & e_2^N & \cdots & e_m^N \end{pmatrix}$$

Тогда ошибка комитета есть

$$Q = \left| \frac{1}{N}(1, 1, \ldots, 1)\mathbf{E} \right|^2 = \frac{1}{N^2}(1, 1, \ldots, 1)\mathbf{E}\mathbf{E}^{\mathrm{T}}(1, 1, \ldots, 1)^{\mathrm{T}}$$

Комитеты сетей - ошибка

Пусть e^i_j – независимы.

Тогда матрица EE^T – диагональная, и на

i-ом месте на диагонали находится значение Q_i – сумма квадратов ошибок

i-ой сети, т.е. $Q_i=||e^i||^2$.

Тогда ошибка комитета есть

$$Q = \frac{1}{N}\left(\frac{1}{N}(Q_1 + \cdots + Q_N)\right)$$

– примерно в N раз меньше чем средняя ошибка членов комитета.

Комитеты сетей

Если условие независимости не выполняется, то можно использовать взвешенную сумму:

$$f = \sum_{i=1}^{N} w_i f_i.$$

Здесь $w_1 + \cdots + w_N = 1$.

Требуется выбрать веса, минимизирующие ошибку.

Комитеты сетей

В этом случае ошибка комитета есть

$$Q = \frac{1}{N^2}(w_1, w_2, \ldots, w_N)\mathbf{E}\mathbf{E}^{\mathrm{T}}(w_1, w_2, \ldots, w_N)^{\mathrm{T}}.$$

Чтобы учесть ограничение на сумму весов введём множитель Лагранжа:

$$\begin{aligned} Q' &= \frac{1}{N^2}\mathbf{w}\mathbf{E}\mathbf{E}^{\mathrm{T}}\mathbf{w}^{\mathrm{T}} + \lambda(1, 1, \ldots, 1)\mathbf{w}^{\mathrm{T}} \\ &= \frac{1}{N^2}\mathbf{w}\mathbf{E}\mathbf{E}^{\mathrm{T}}\mathbf{w}^{\mathrm{T}} + \lambda\mathbf{1}\mathbf{w}^{\mathrm{T}} \end{aligned}$$

Приравняем производную этой функции по *w* к 0.

Комитеты сетей

Приравняем производную этой функции по *w* к 0. Получим

$$\frac{1}{N^2}\mathbf{w}\mathbf{E}\mathbf{E}^{\mathrm{T}} + \lambda\mathbf{1} = 0\,.$$

Если матрица $\mathbf{E}\mathbf{E}^{\mathrm{T}}$ имеет обратную, то

$$\mathbf{w} = -\lambda N^2\mathbf{1}(\mathbf{E}\mathbf{E}^{\mathrm{T}})^{-1}.$$

Из ограничения $\mathbf{w}\mathbf{1}^{\mathrm{T}}=1$

$$\mathbf{w}\mathbf{1}^{\mathrm{T}} = -\lambda N^2\mathbf{1}(\mathbf{E}\mathbf{E}^{\mathrm{T}})^{-1}\mathbf{1}^{\mathrm{T}} = 1\,,$$

следовательно

$$\lambda = -\frac{1}{N^2\mathbf{1}(\mathbf{E}\mathbf{E}^{\mathrm{T}})^{-1}\mathbf{1}^{\mathrm{T}}}.$$

Тогда

$$\mathbf{w} = \frac{\mathbf{1}(\mathbf{E}\mathbf{E}^{\mathrm{T}})^{-1}}{\mathbf{1}(\mathbf{E}\mathbf{E}^{\mathrm{T}})^{-1}\mathbf{1}^{\mathrm{T}}}$$

Комитеты сетей

- Как сделать сети разными?
  - Сделать так, чтобы алгоритм обучения остановился в разных локальных минимумах.
    - Хак, но работает.
  - Использовать разные модели, в том числе не являющиеся нейронными сетями.
    - Деревья решений, SVM, …
  - Использовать разные модели нейронных сетей:
    - Разные архитектуры
      - Число слоёв
      - Число нейронов в слое
      - Вид нейронов
    - Разные способы регуляризации
    - Разные алгоритмы обучения.

Комитеты сетей

- Как сделать модели разными, используя разные обучающие данные:
  - Бэггинг:
    - Разные обучающие наборы создаются путём выбора с повторениями из исходного.
    - Очень хорошо работает с деревьями решений.
    - Очень долго считать для нейронных сетей.
  - Бустинг:
    - Обучаем последовательность простых моделей.
    - Каждая следующая модель обучается на данных, вес которых зависит от текущей ошибки модели: больше ошибка – больше вес.
      - Фокусирует внимание на моделировании сложных случаев.

Темы для докладов

1. Алоритмы обучения нейронных сетей Adagrad, RMSProp, Adadelta, Adam, Adamax.
2. Dropout
3. Архитектура Inception: GoogLeNet
4. ResNet (Residual Network), HighwayNet и DenseNet
5. Рекурсивные сети и LSTM
6. Model stacking
7. Автоэнкодеры
8. Generative Adversarial Networks (генеративные состязающиеся сети)
9. Байесовские методы обучения нейронных сетей. Вариационный байес.