Лекция 2. Нейронные сети и обратное распространение ошибки

Денис Деркач, Дмитрий Тарасов

Слайды от А. Маевского, М. Гущина, А. Кленицкого, М Борисяка

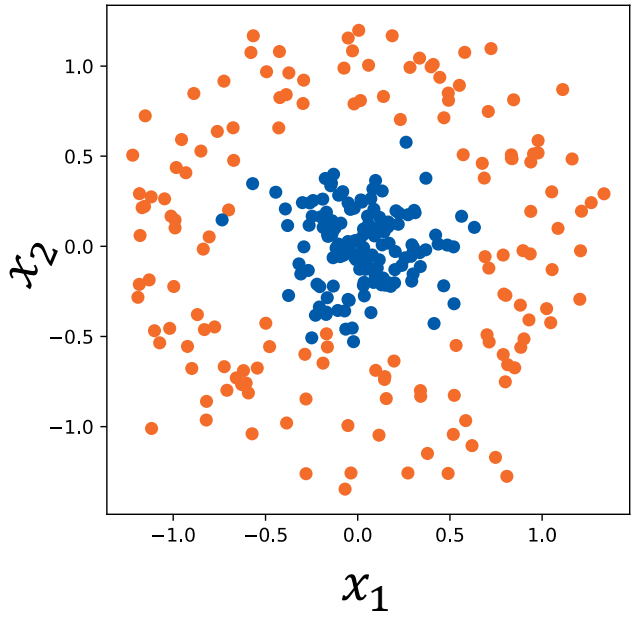
LAMBDA • HSE

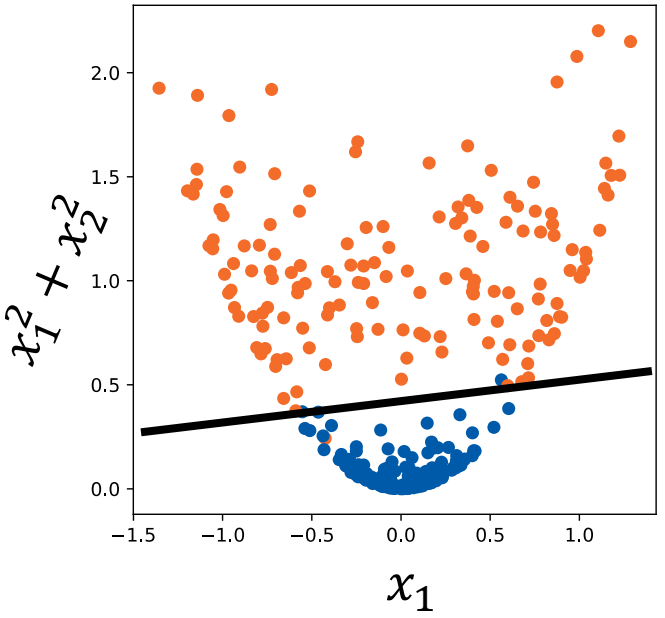
20 января 2025 года

От линейной модели к нейронной сети

Линейные модели b извлечение признаков

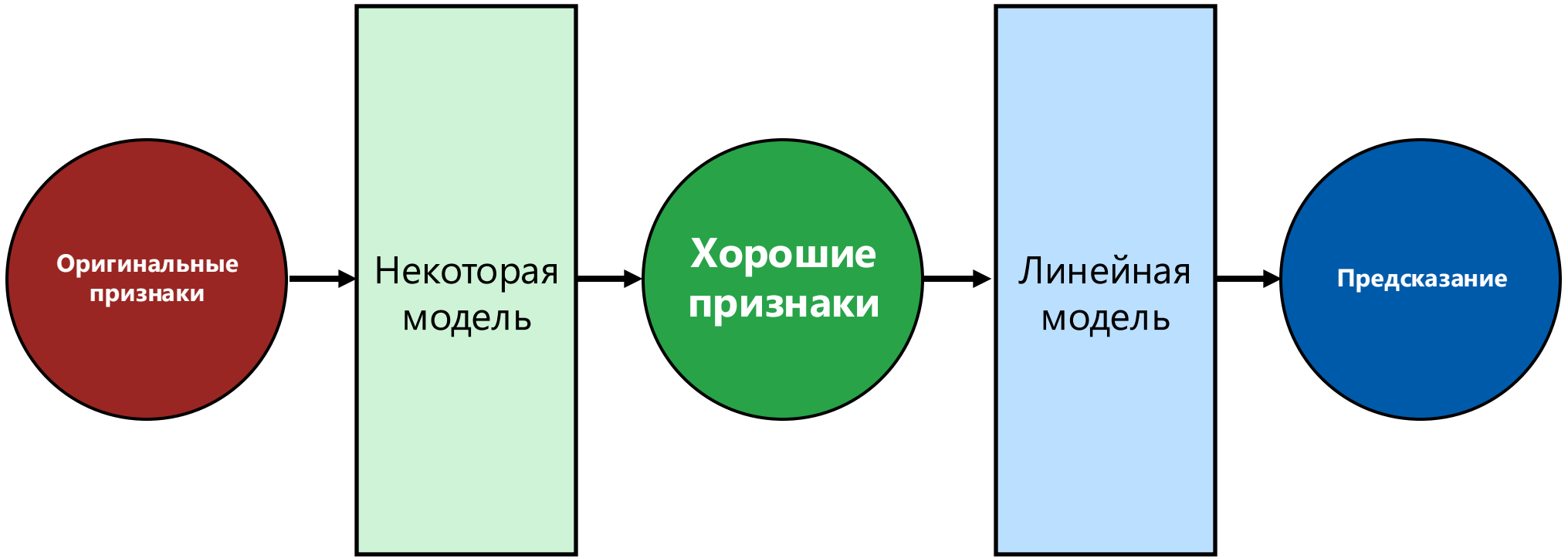
- Для линейных моделей мы **придумываем** новые признаки, чтобы сделать модель более мощной
- Поиск хороших функций (так называемая **feature engineering**) - задача крайне нетривиальная.





Автоматизация конструирования признаков – основная сила DL.

Идея: стэкинг моделей

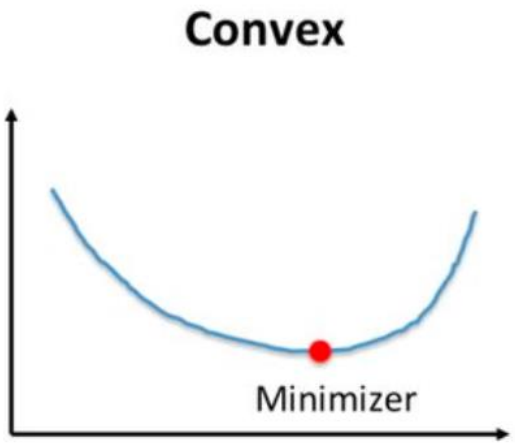


- Добавить ещё одну модель
- Всё обучим одновременно
  - Если модели **дифференцируемые**, можем использовать градиентный спуск

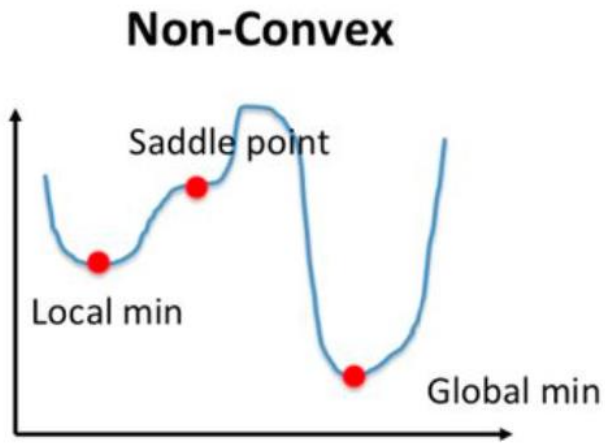
Замечание: такое суммирование моделей, вероятно, делает задачу невыпуклой
⇒ **нет гарантий сходимости**

Convex vs non-convex optimization

Выпуклая оптимизация



Невыпуклая оптимизация

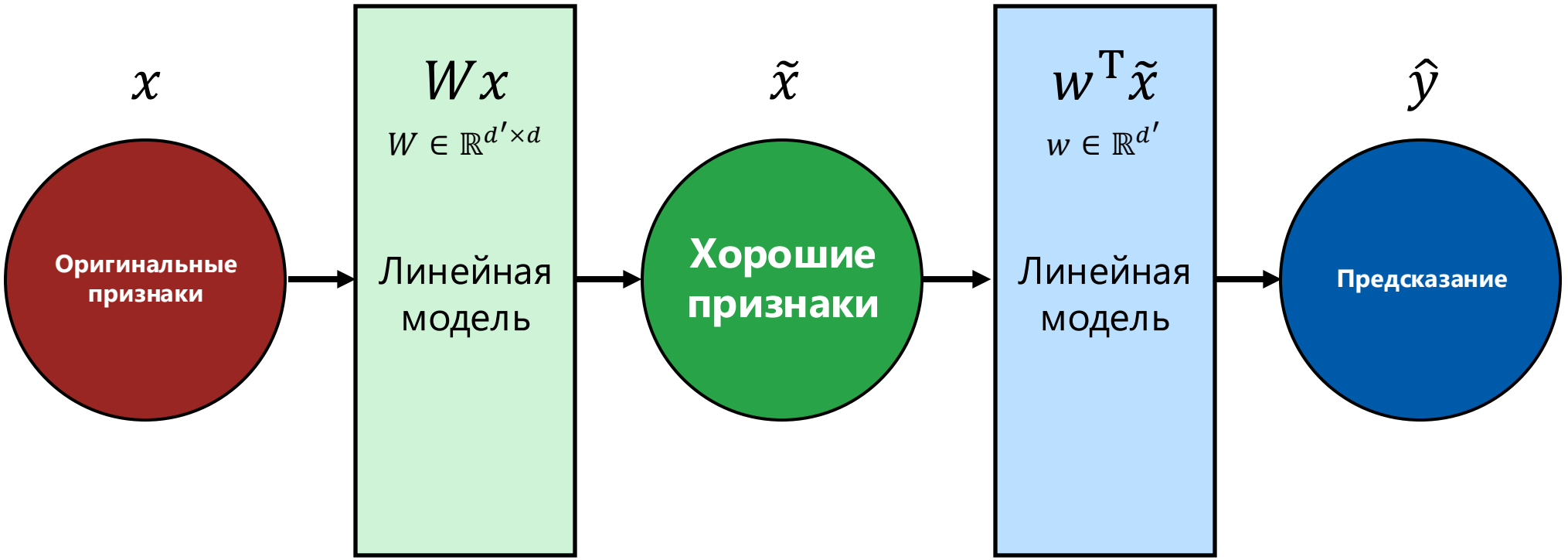


https://ru.pinterest.com/pin/convex-nonconvex-functions--672232681861372201/

Множество минимумов, неявные точки перегиба

Замечание: такое суммирование моделей, вероятно, делает задачу невыпуклой
⇒ нет гарантий сходимости

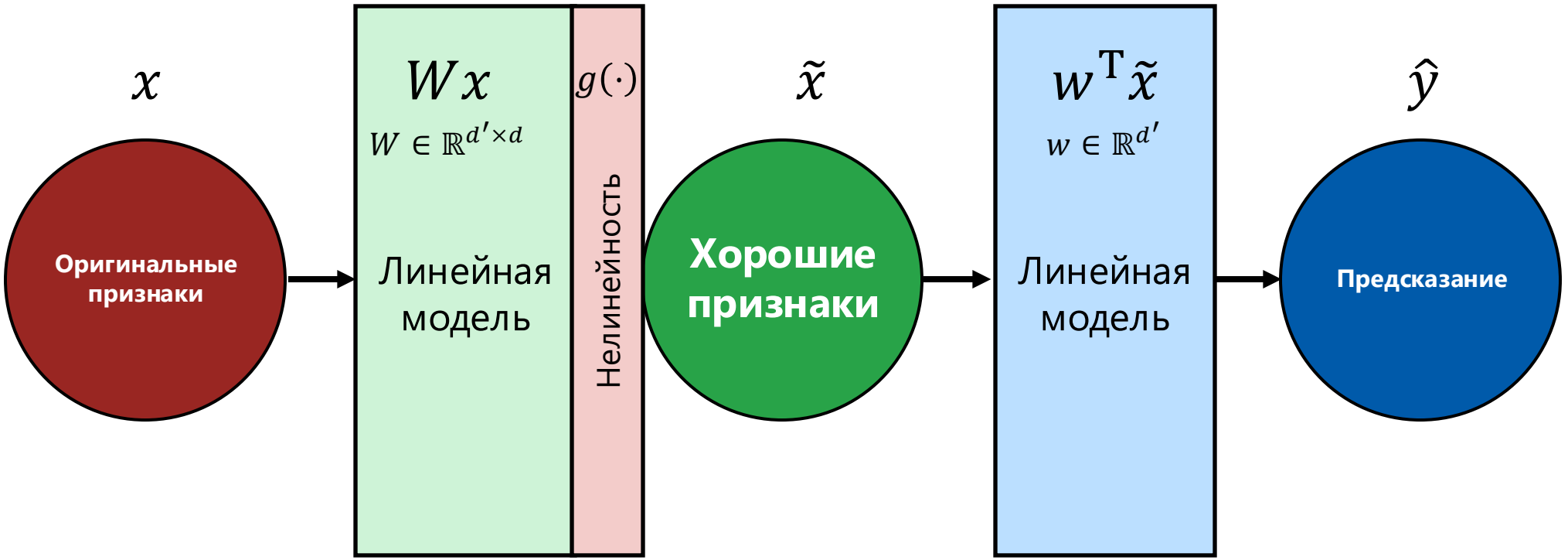
Какие модели идут в стэк?



$$\hat{y} = w^{\mathrm{T}}\tilde{x} = w^{\mathrm{T}}(Wx) = \left(w^{\mathrm{T}}W\right)x = w'^{\mathrm{T}}x$$

- Всё становится линейной моделью

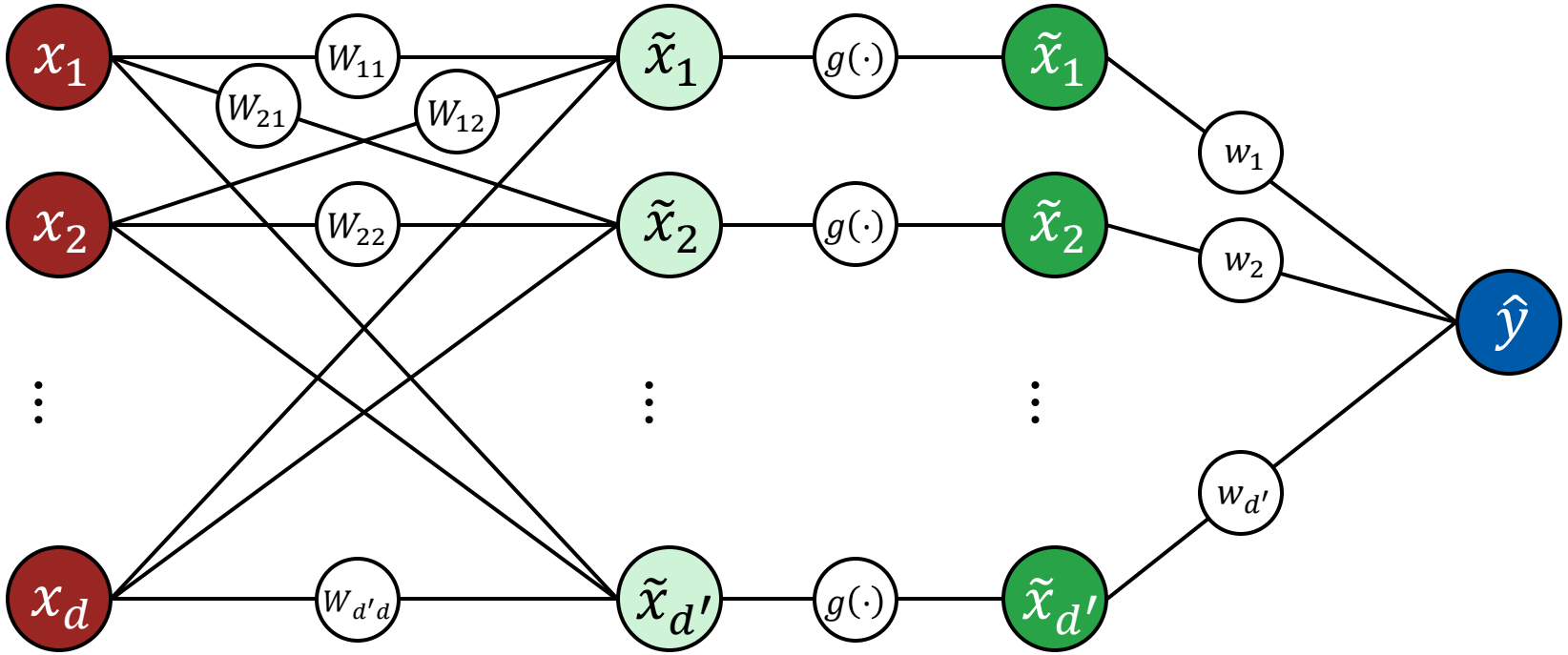
Введём нелинейность



$$\hat{y} = w^{\mathrm{T}}\tilde{x} = w^{\mathrm{T}} g(Wx)$$

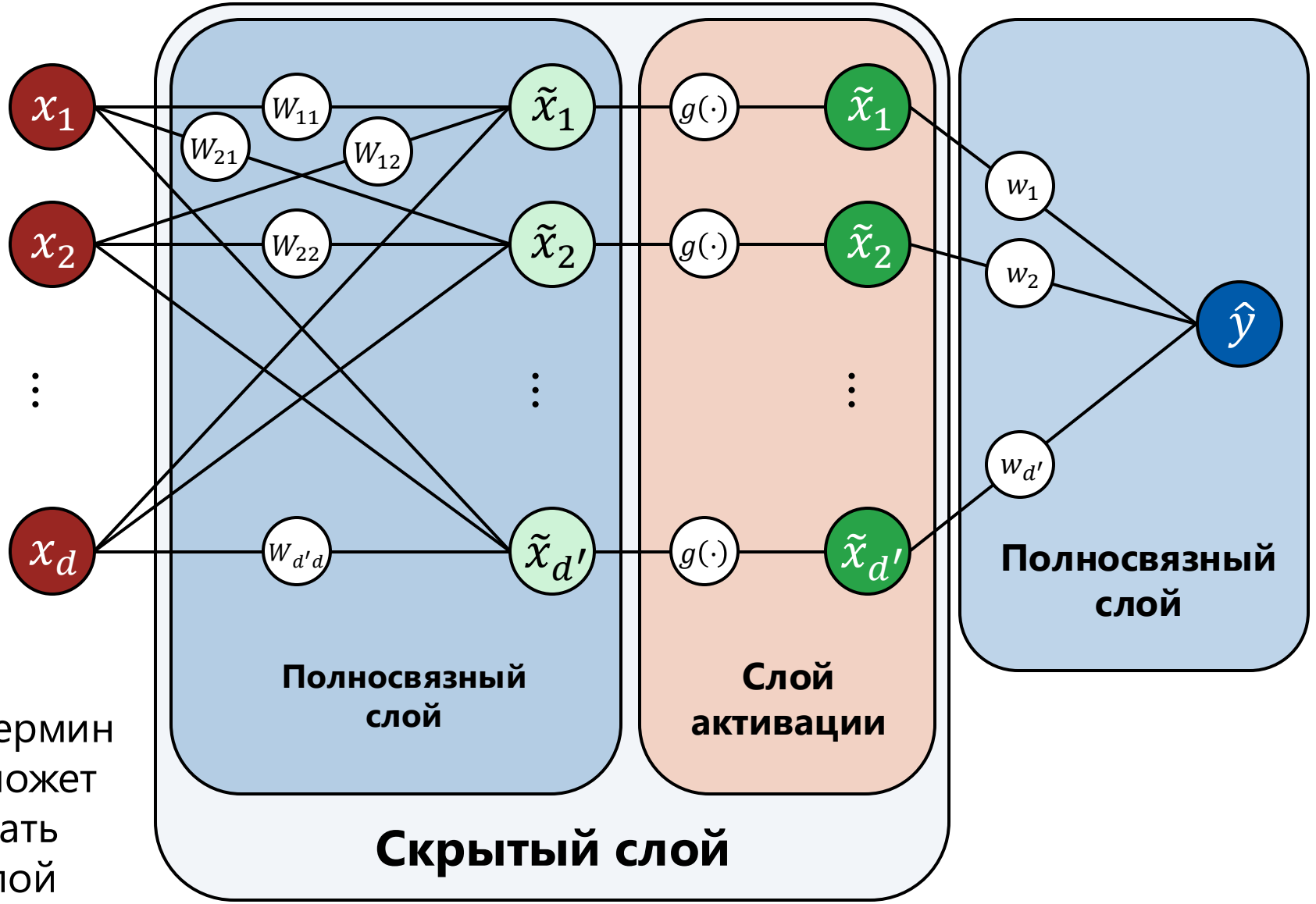
$g(\cdot)$ – некоторая **нелинейная** скалярная функция (поэлементная)

Детализированный вид



$$\hat{y} = w^{\mathrm{T}}\tilde{x} = w^{\mathrm{T}}g(Wx) = \sum_j \left[w_j g\left(\sum_i W_{ji} x_i \right) \right]$$

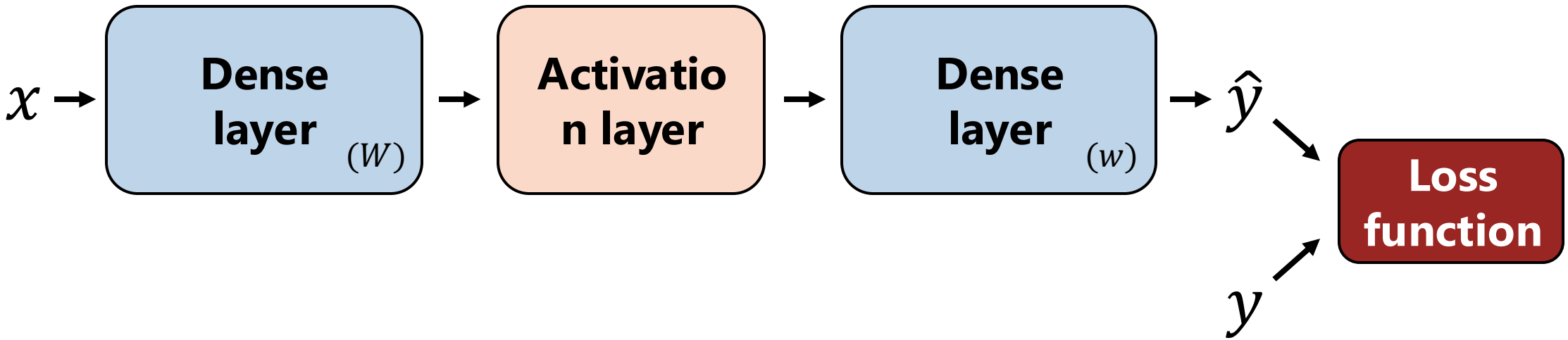
Терминология



Примечание: термин «активация» может также означать выходной слой

Обратное распространение ошибки

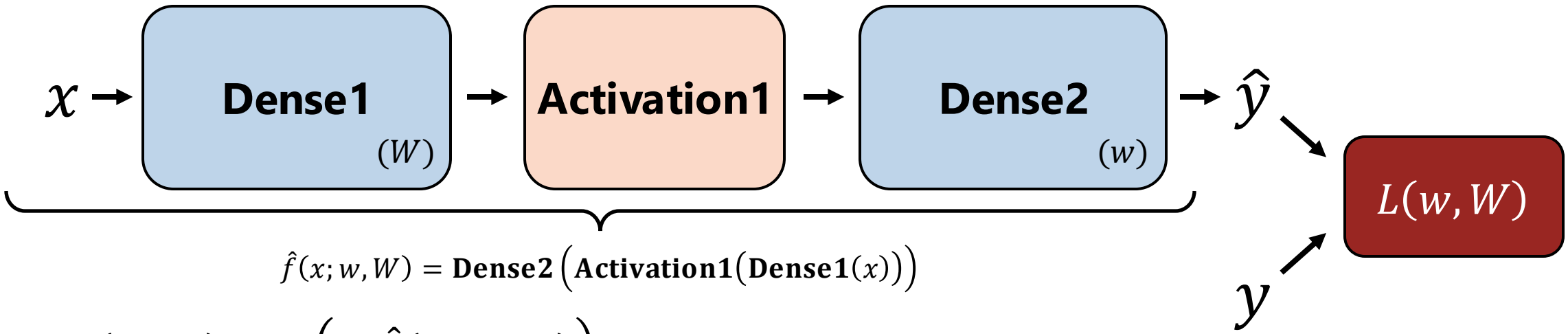
Функция потерь



- Например: квадрат ошибки

$$L = \frac{1}{N} \sum_{i=1\ldots N} \left(y_i - w^{\mathrm{T}} g(W x_i) \right)^2$$

Производные

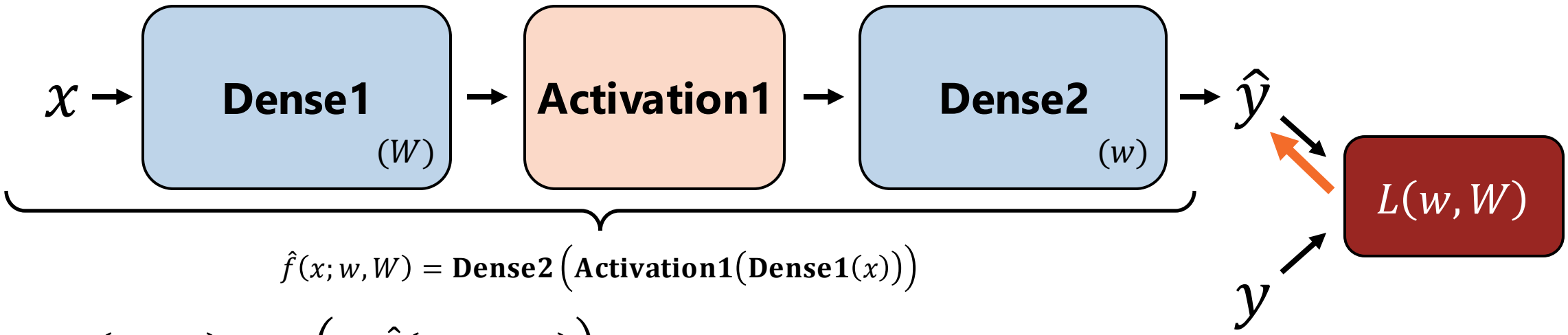


$$\hat{f}(x; w, W) = \mathbf{Dense2}\left(\mathbf{Activation1}\left(\mathbf{Dense1}(x)\right)\right)$$

$$L(w, W) \equiv L\left(y, \hat{f}(x; w, W)\right)$$

$$\frac{\partial L}{\partial W} = \frac{\partial L\left(y, \hat{f}\right)}{\partial \hat{f}} \cdot \frac{\partial \hat{f}}{\partial W} =$$

Производные

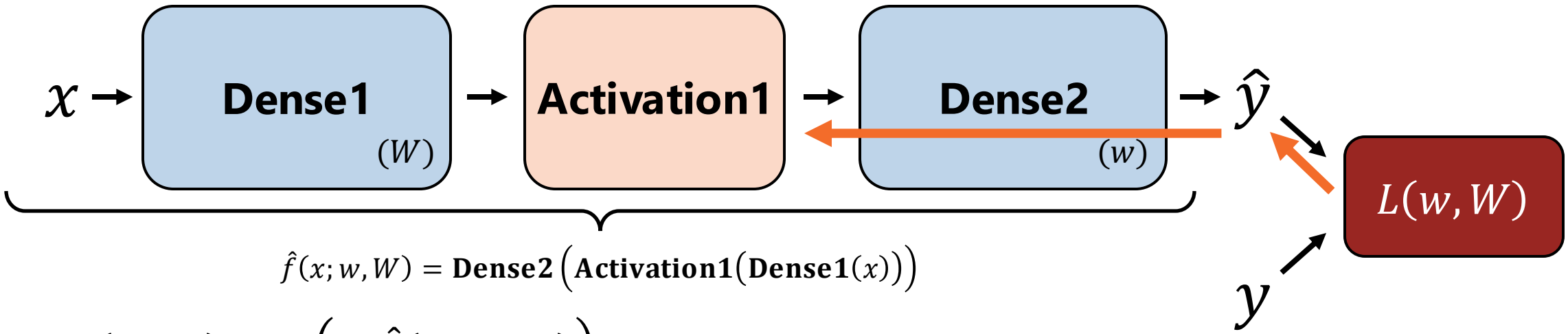


$$\hat{f}(x; w, W) = \mathbf{Dense2}\left(\mathbf{Activation1}\left(\mathbf{Dense1}(x)\right)\right)$$

$$L(w, W) \equiv L\left(y, \hat{f}(x; w, W)\right)$$

$$\frac{\partial L}{\partial W} = \frac{\partial L\left(y, \hat{f}\right)}{\partial \hat{f}} \cdot \frac{\partial \hat{f}}{\partial W} = \frac{\partial L\left(y, \hat{f}\right)}{\partial \hat{f}} \cdot$$

Снова производные

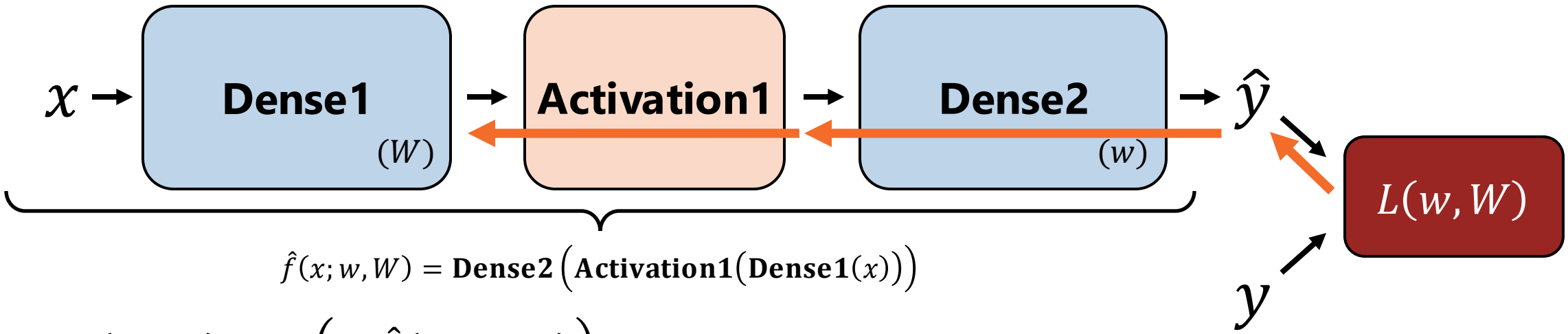


$$\hat{f}(x; w, W) = \mathbf{Dense2}\left(\mathbf{Activation1}\left(\mathbf{Dense1}(x)\right)\right)$$

$$L(w, W) \equiv L\left(y, \hat{f}(x; w, W)\right)$$

$$\frac{\partial L}{\partial W} = \frac{\partial L\left(y, \hat{f}\right)}{\partial \hat{f}} \cdot \frac{\partial \hat{f}}{\partial W} = \frac{\partial L\left(y, \hat{f}\right)}{\partial \hat{f}} \cdot \frac{\partial \mathbf{Dense2}}{\partial \mathbf{Activation1}} \cdot$$

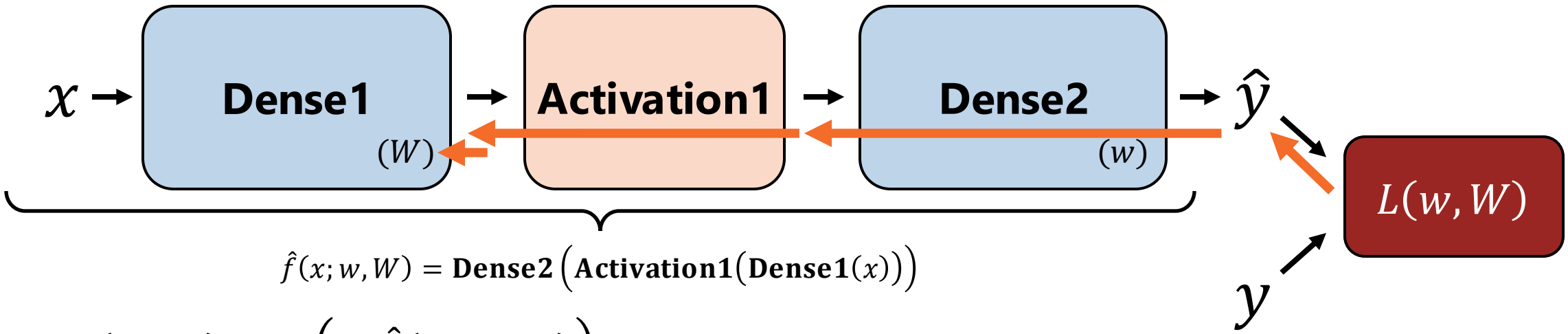
Calculating derivatives



$$L(w,W) \equiv L\left(y, \hat{f}(x;w,W)\right)$$

$$\frac{\partial L}{\partial W} = \frac{\partial L\left(y,\hat{f}\right)}{\partial \hat{f}} \cdot \frac{\partial \hat{f}}{\partial W} = \frac{\partial L\left(y,\hat{f}\right)}{\partial \hat{f}} \cdot \frac{\partial \mathbf{Dense2}}{\partial \mathbf{Activation1}} \cdot \frac{\partial \mathbf{Activation1}}{\partial \mathbf{Dense1}} \cdot$$

Calculating derivatives

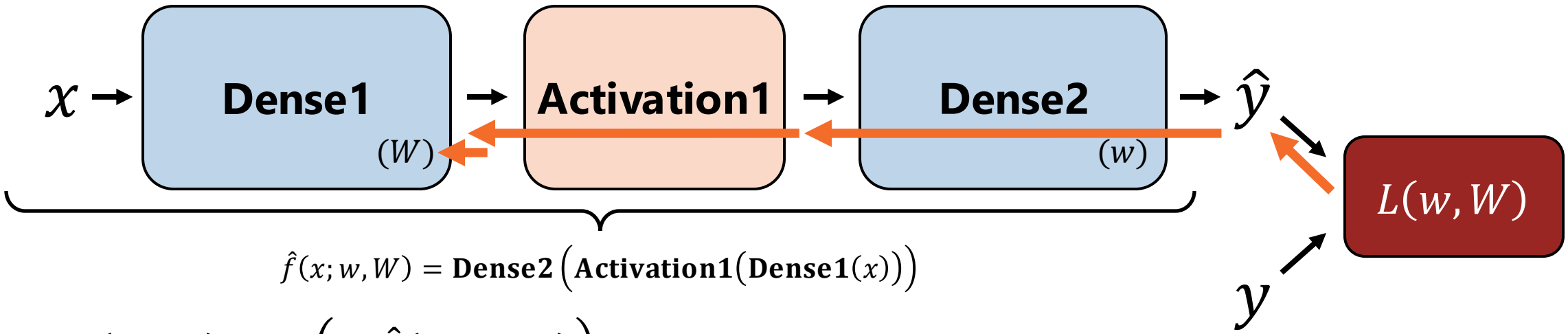


$$L(w, W) \equiv L\left(y, \hat{f}(x; w, W)\right)$$

$$\frac{\partial L}{\partial W} = \frac{\partial L\left(y, \hat{f}\right)}{\partial \hat{f}} \cdot \frac{\partial \hat{f}}{\partial W} = \frac{\partial L\left(y, \hat{f}\right)}{\partial \hat{f}} \cdot \frac{\partial \mathbf{Dense2}}{\partial \mathbf{Activation1}} \cdot \frac{\partial \mathbf{Activation1}}{\partial \mathbf{Dense1}} \cdot \frac{\partial \mathbf{Dense1}}{\partial W}$$

- Backpropagation algorithm ≈ applying the chain rule
  - The actual algorithm states how to do it efficiently

Calculating derivatives



$$\hat{f}(x; w, W) = \mathbf{Dense2}\left(\mathbf{Activation1}\left(\mathbf{Dense1}(x)\right)\right)$$

$$L(w, W) \equiv L\left(y, \hat{f}(x; w, W)\right)$$

$$\frac{\partial L}{\partial W} = \frac{\partial L(y, \hat{f})}{\partial \hat{f}} \cdot \frac{\partial \hat{f}}{\partial W} = \underbrace{\frac{\partial L(y, \hat{f})}{\partial \hat{f}}}_{\text{scalar}} \cdot \underbrace{\frac{\partial \mathbf{Dense2}}{\partial \mathbf{Activation1}}}_{d'\text{-vector}} \cdot \underbrace{\frac{\partial \mathbf{Activation1}}{\partial \mathbf{Dense1}}}_{d' \times d'\text{-matrix}} \cdot \underbrace{\frac{\partial \mathbf{Dense1}}{\partial W}}_{d' \times d' \times d\text{-tensor}}$$

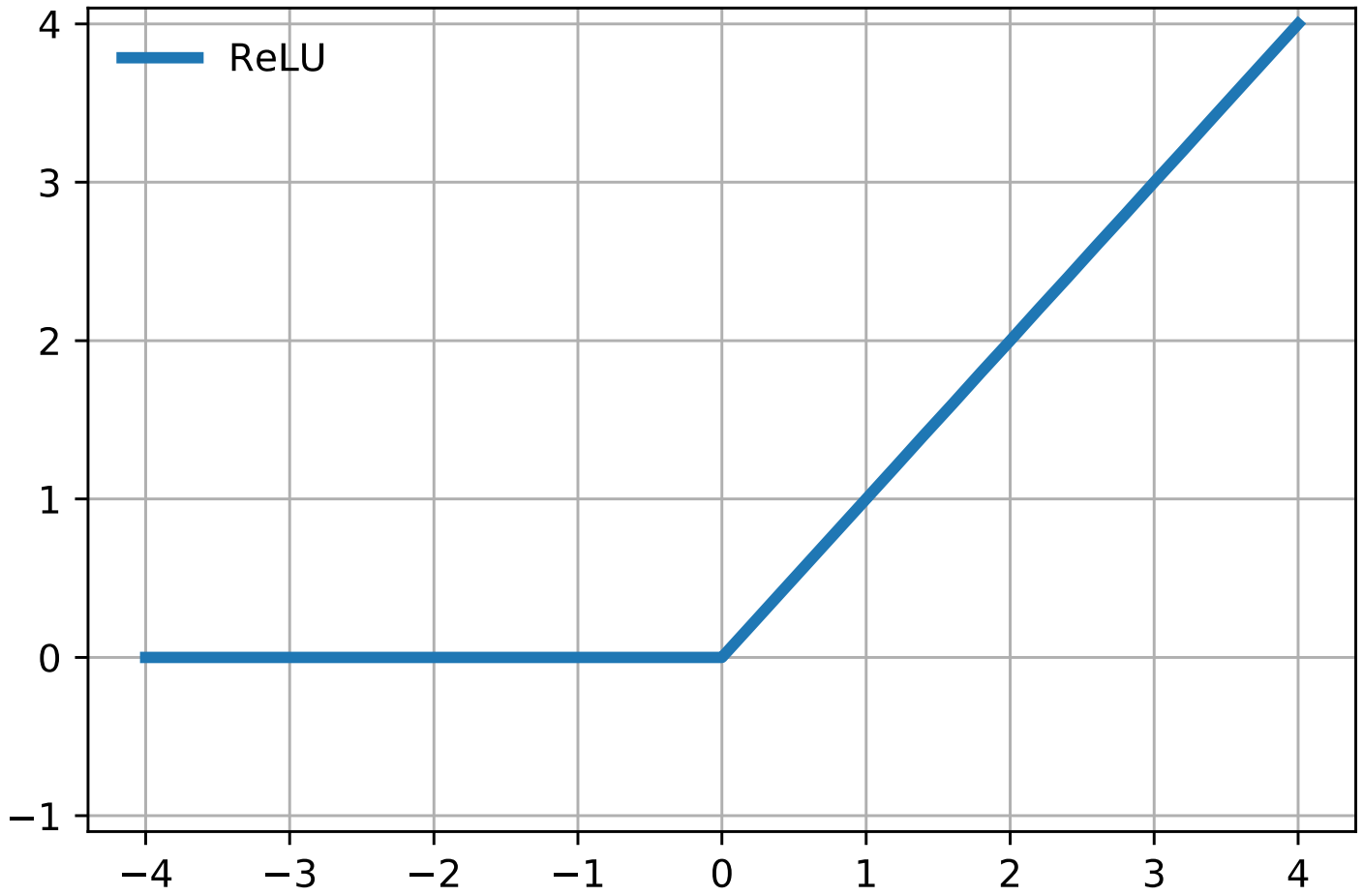
(scalar · unit matrix)

$= d' \times d$-matrix

- Обратное распространение ошибки ≈ цепное правило
  - Алгоритм показывает как делать это эффективно

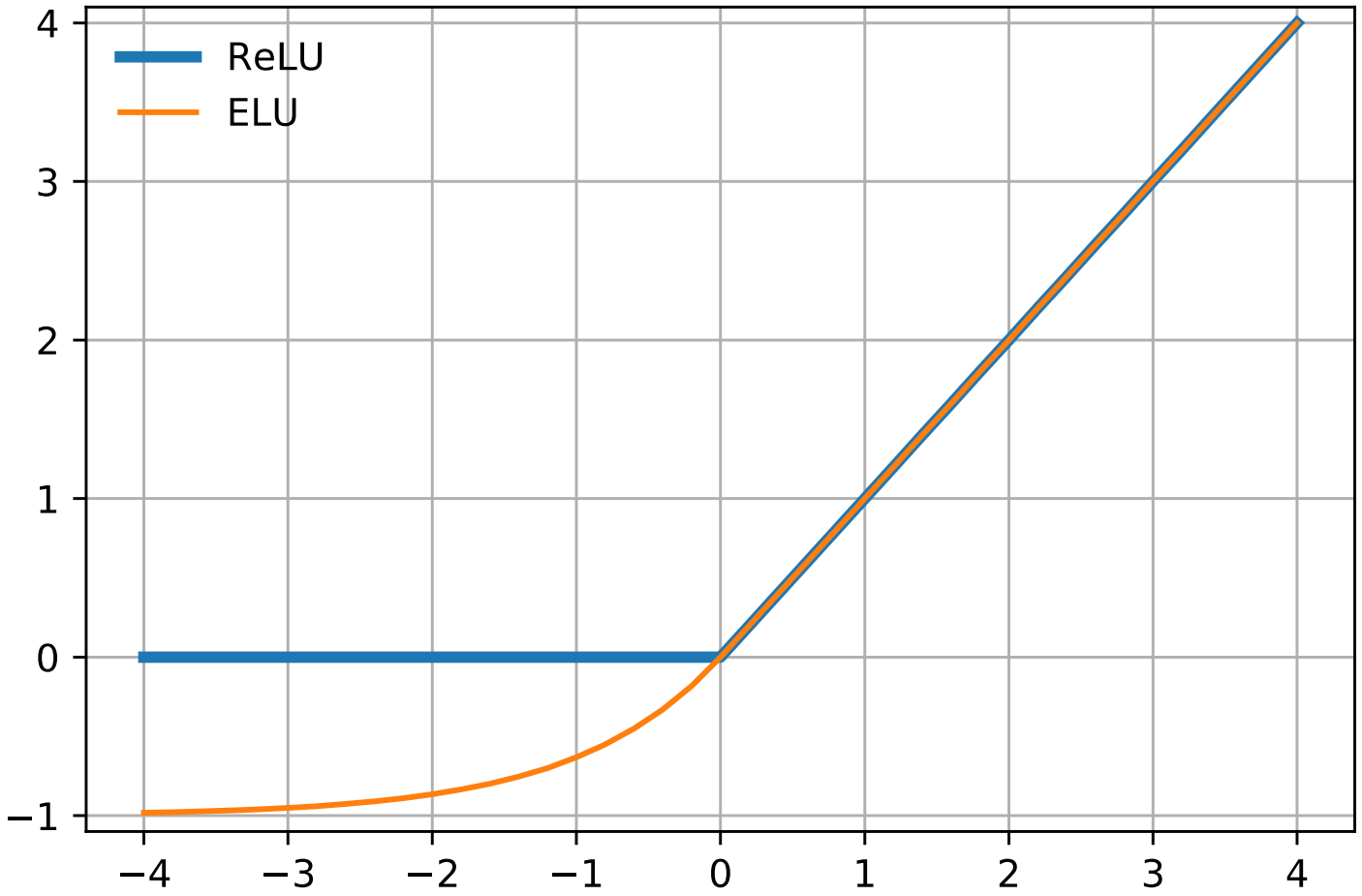
Функции активации

Activation functions



$$\mathrm{ReLU}(x) = \max(0, x)$$

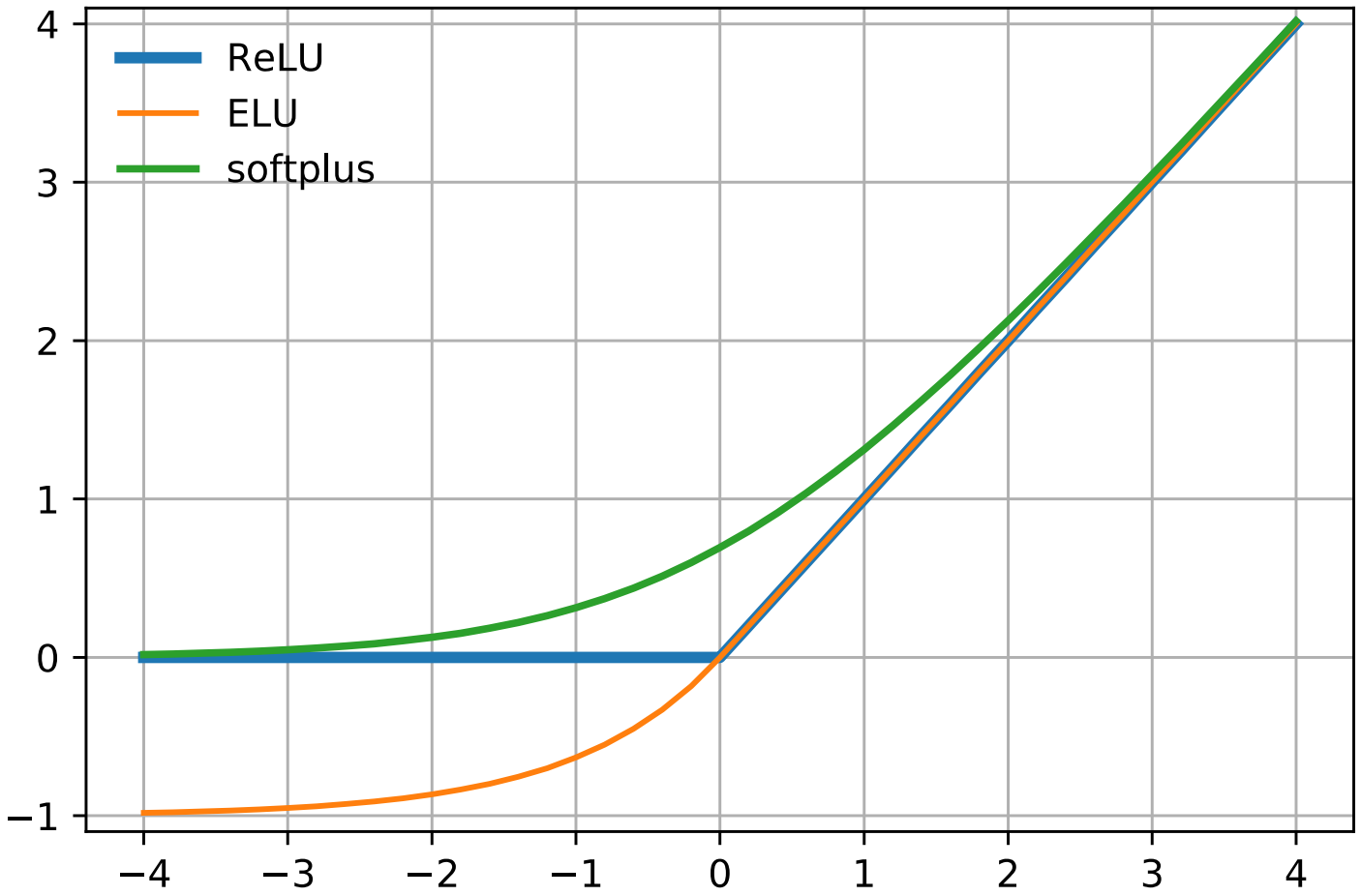
Activation functions



$$\mathrm{ReLU}(x) = \max(0, x)$$

$$\mathrm{ELU}(x) = \begin{cases} x & x \geq 0 \\ e^x - 1 & x < 0 \end{cases}$$

Activation functions

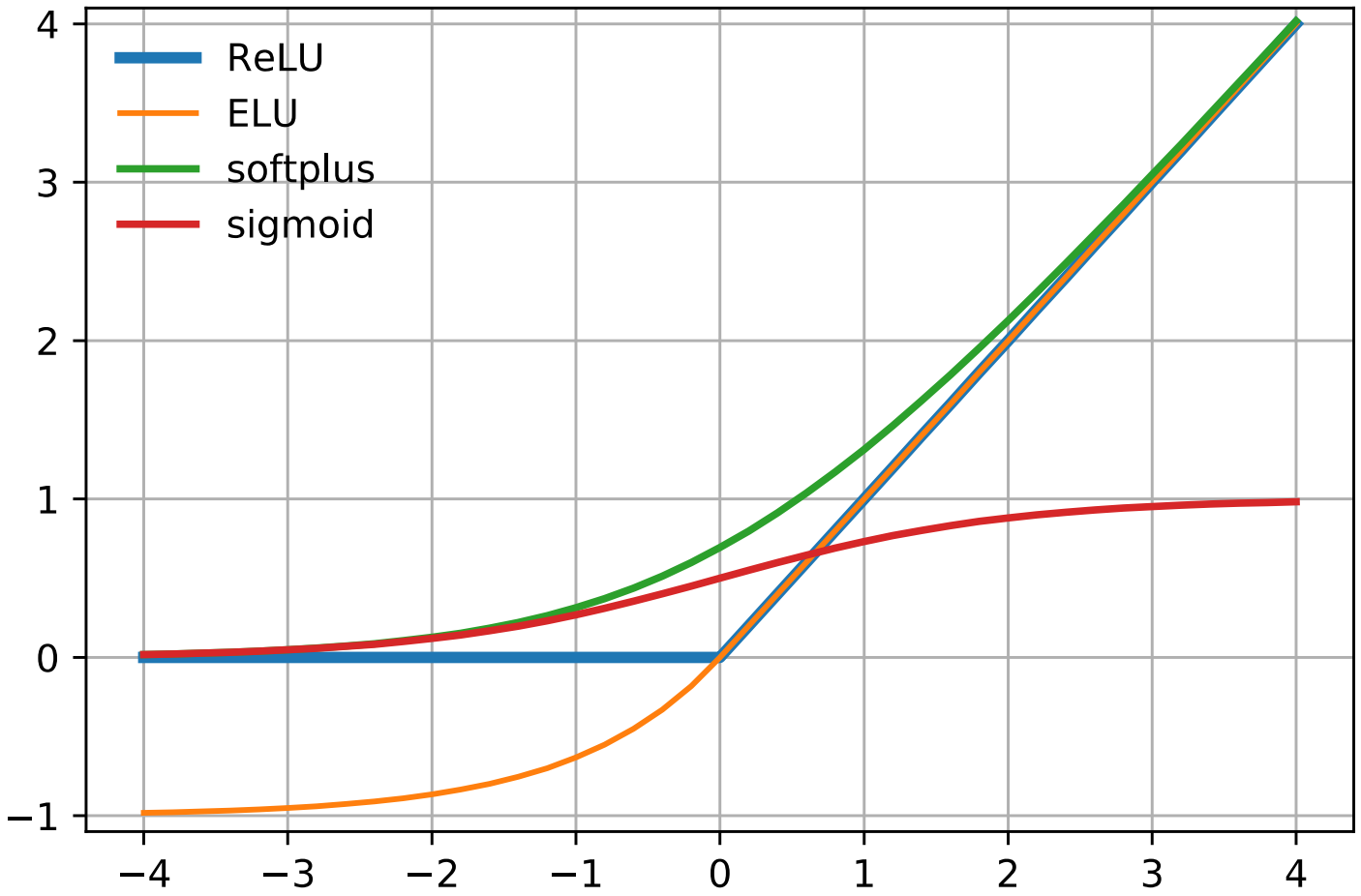


$$\mathrm{ReLU}(x) = \max(0, x)$$

$$\mathrm{ELU}(x) = \begin{cases} x & x \geq 0 \\ e^x - 1 & x < 0 \end{cases}$$

$$\mathrm{softplus}(x) = \log(1 + e^x)$$

Activation functions



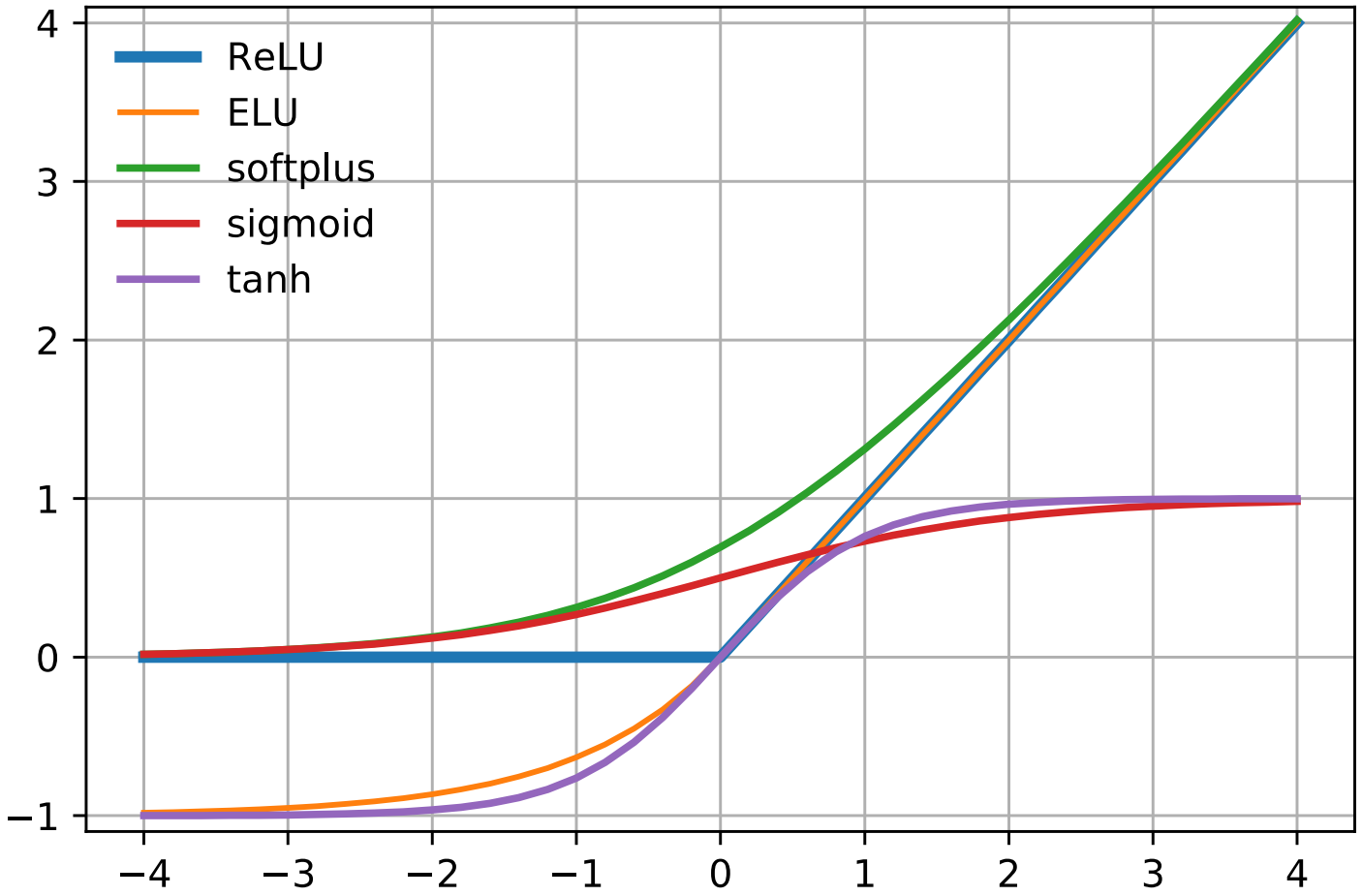
$$\mathrm{ReLU}(x) = \max(0, x)$$

$$\mathrm{ELU}(x) = \begin{cases} x & x \geq 0 \\ e^x - 1 & x < 0 \end{cases}$$

$$\mathrm{softplus}(x) = \log(1 + e^x)$$

$$\mathrm{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

Activation functions



$$\mathrm{ReLU}(x) = \max(0, x)$$

$$\mathrm{ELU}(x) = \begin{cases} x & x \geq 0 \\ e^x - 1 & x < 0 \end{cases}$$

$$\mathrm{softplus}(x) = \log(1 + e^x)$$

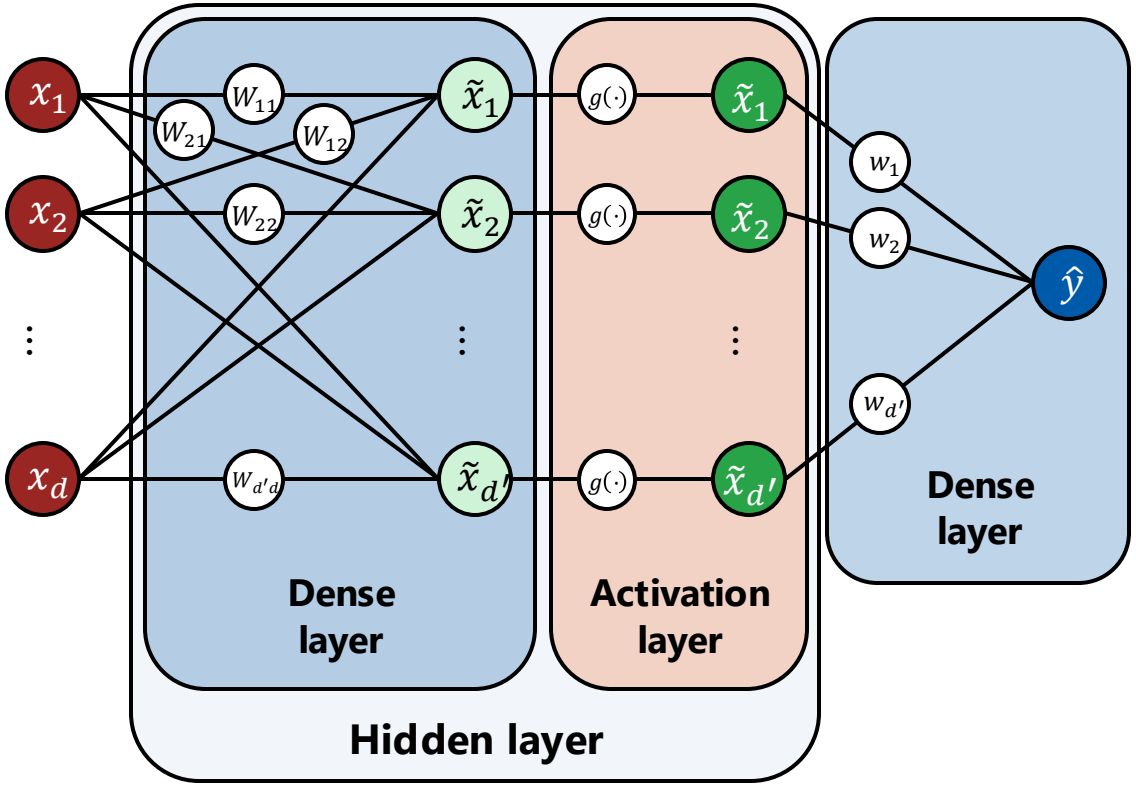
$$\mathrm{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Универсальный аппроксиматор

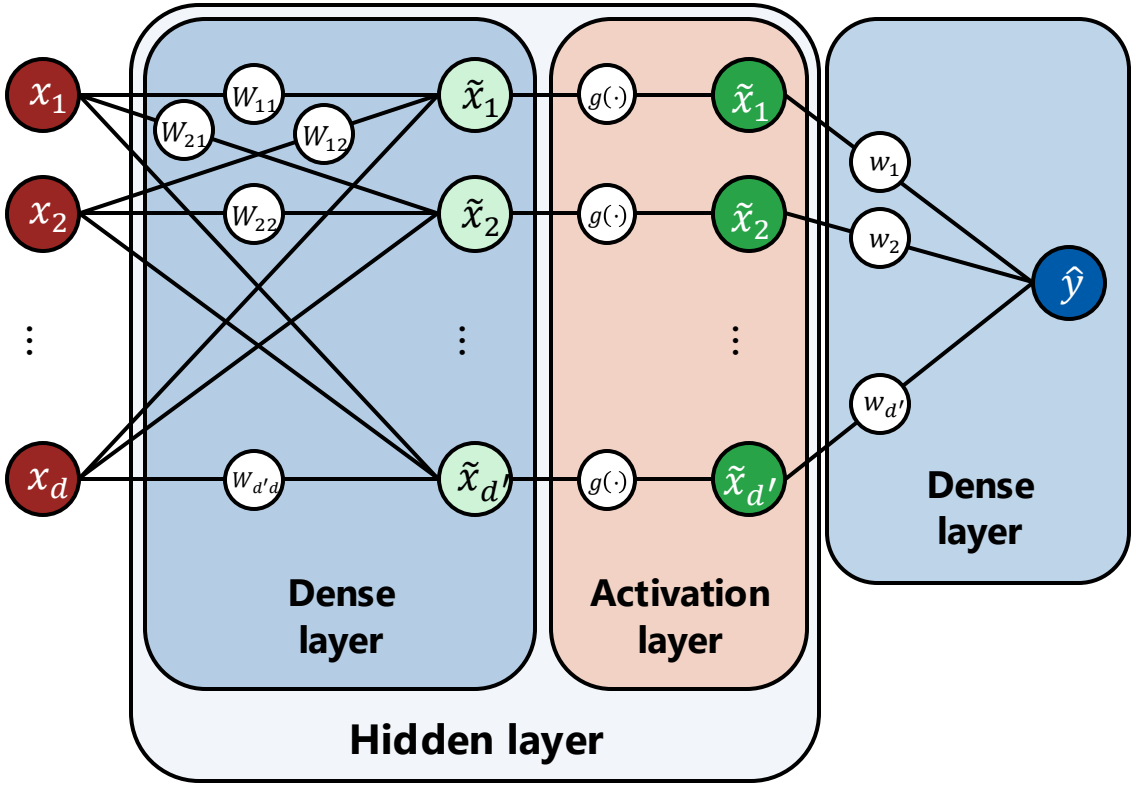
Universal approximator

- Всего один скрытый слой с нелинейностью делает эту модель универсальным аппроксиматором



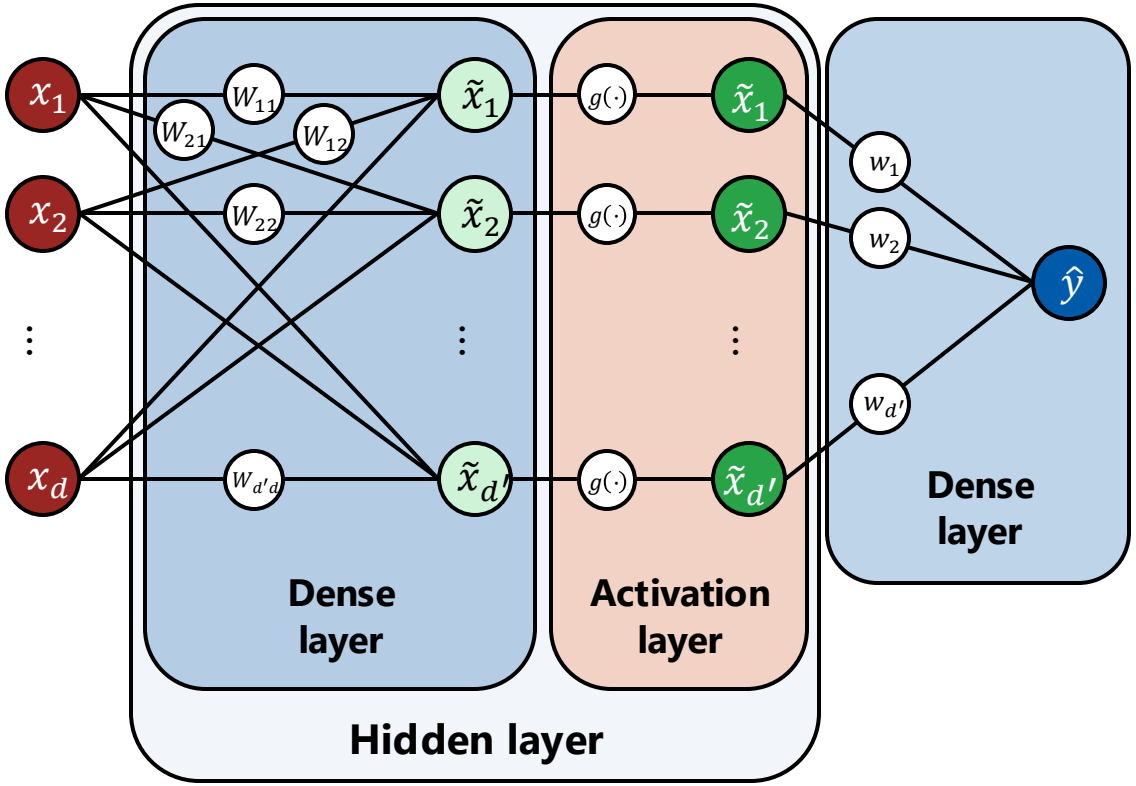
Universal approximator

- Всего один скрытый слой с нелинейностью делает эту модель универсальным аппроксиматором
  - любая функция может быть аппроксимирована с произвольной точностью при достаточно широком скрытом слое (достаточно большом d')

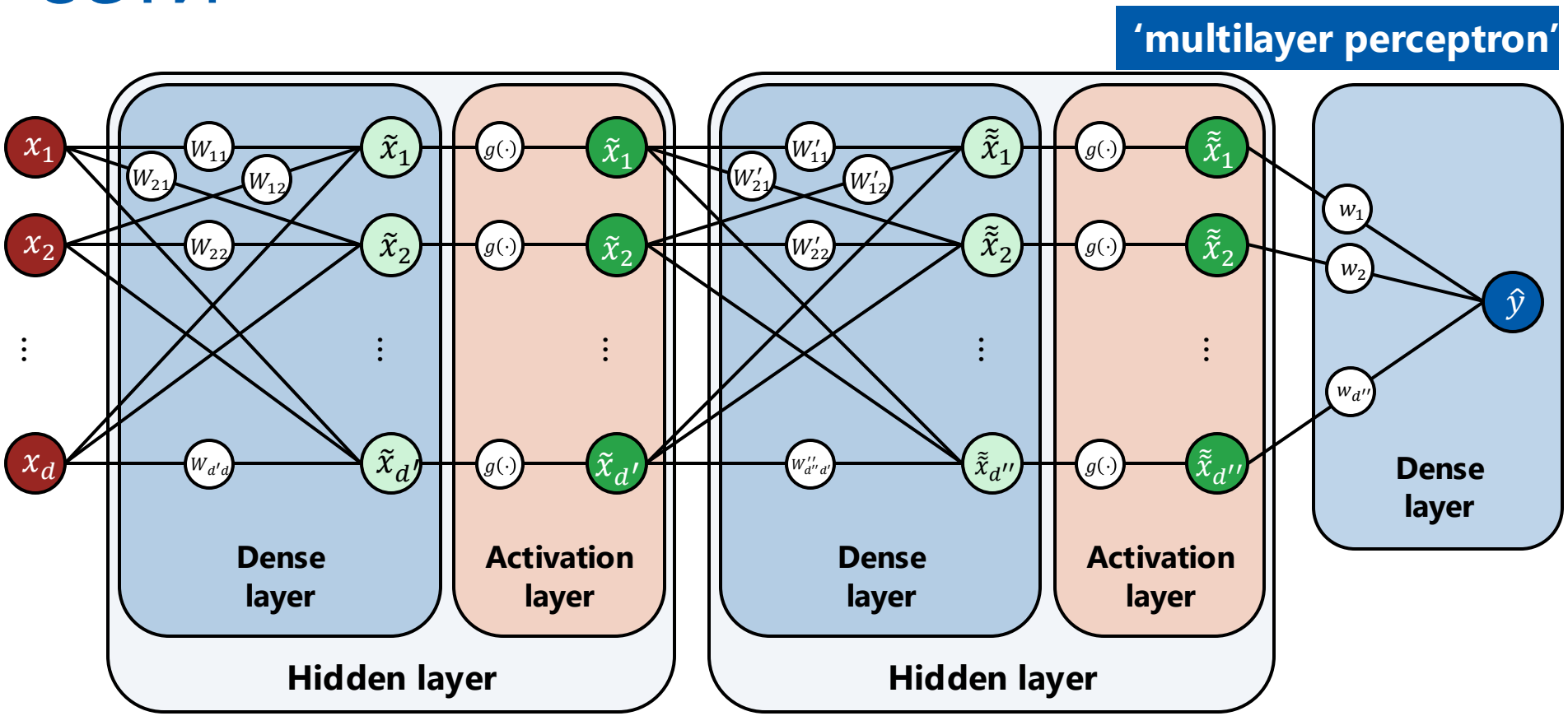


Universal approximator

- Всего один скрытый слой с нелинейностью делает эту модель универсальным аппроксиматором
  - любая функция может быть аппроксимирована с произвольной точностью при достаточно широком скрытом слое (достаточно большом d')
  - Примечание: на практике мы можем не найти такого приближения
    - например, из-за сильно невыпуклой функции потерь, невыполнимо большого d', чрезмерной подгонки



Глубокие сети



- На практике стэкинг большего числа скрытых слоев часто уменьшает количество нейронов, необходимых для представления заданной функции

Теорема о бесплатных обедах
No free lunch
(немного философии)

Тест IQ

Если следовать схеме, показанной в последовательности чисел ниже, какое число отсутствует?

1, 8, 27, ?, 125, 216

- **36;**
- **45;**
- **46;**
- **64;**
- **99.**

Тест IQ (ML edition)

Если следовать схеме, показанной в последовательности чисел ниже, какое число отсутствует?

X_{train}	1	2	3	5	6
y_{train}	1	8	27	125	216

$$X_{\text{test}} = (4,)$$

Тест IQ (ML edition): решений

Предлагаю решение:

$$f(x) = \frac{1}{12}\left(91x^5 - 1519x^4 + 9449x^3 - 26705x^2 + 33588x - 14940\right);$$

$$\begin{aligned} f(1) &= 1; \\ f(2) &= 8; \\ f(3) &= 27; \\ f(4) &= 99; \\ f(5) &= 125; \\ f(6) &= 216. \end{aligned}$$

Определения

A supervised learning algorithm A:

- is defined on sample set $\mathcal{X}$ and target set $\mathcal{Y}$;
- receives training dataset $D_{\text{train}} = (X, y) \subseteq \mathcal{X} \times \mathcal{Y}$;
- does ~~magic~~ learning;
- returns prediction function f:

$$f = A(D_{\text{train}});$$
$$f : \mathcal{X} \mapsto \mathcal{Y}.$$

By this definition, decision trees are a family of algorithms.

Теорема

All supervised learning algorithms are equally **useless**:

- for all binary datasets D_{train} of size n with m features:
  - $\mathcal{X} = \{0,1\}^m$, $\mathcal{Y} = \{0,1\}$;
- averaged by all test datasets D_{test}: $X_{\text{test}} = \mathcal{X} \setminus X_{\text{train}}$;

$$\text{gP}(A, D_{\text{train}}) = \frac{1}{|D_{\text{test}}|} \sum_{x,y \in D_{\text{test}}} \mathbb{I}\big[A(D_{\text{train}})(x) = y\big] - \frac{1}{2};$$

$$\boxed{\sum_{D_{\text{train}}} \text{gP}(A, D_{\text{train}}) = 0;}$$

Some algorithms are better at memorization.

Шокирующее следствие

Тесты IQ измеряют вашу способность думать так же, как их авторы... возможно, с поправкой на вашу скорость.

Ещё следствие

Чтобы решить IQ-тест, нужно думать, как автор теста.

Совсем шокирующее следствие

Чтобы решить ML задачу, нужно мыслить как... природа.

Критика теоремы

- Не все задачи одинаково вероятны:
  - непрерывность;
  - человеческая предвзятость: предварительный отбор признаков;
  - предварительное знание рассматриваемой проблемы;
- запоминание также важно:
  - особенно в сочетании с непрерывностью.

Утверждение остаётся в любом случае:

Чтобы улучшить результаты в решении одного класса задач, нужно пожертвовать эффективностью в решении других.

Идеальный алгоритм/семейство

Для одной задачи:

- максимизирует производительность на задачах, соответствующих предыдущим знаниям;
- жертвует производительностью в других местах.

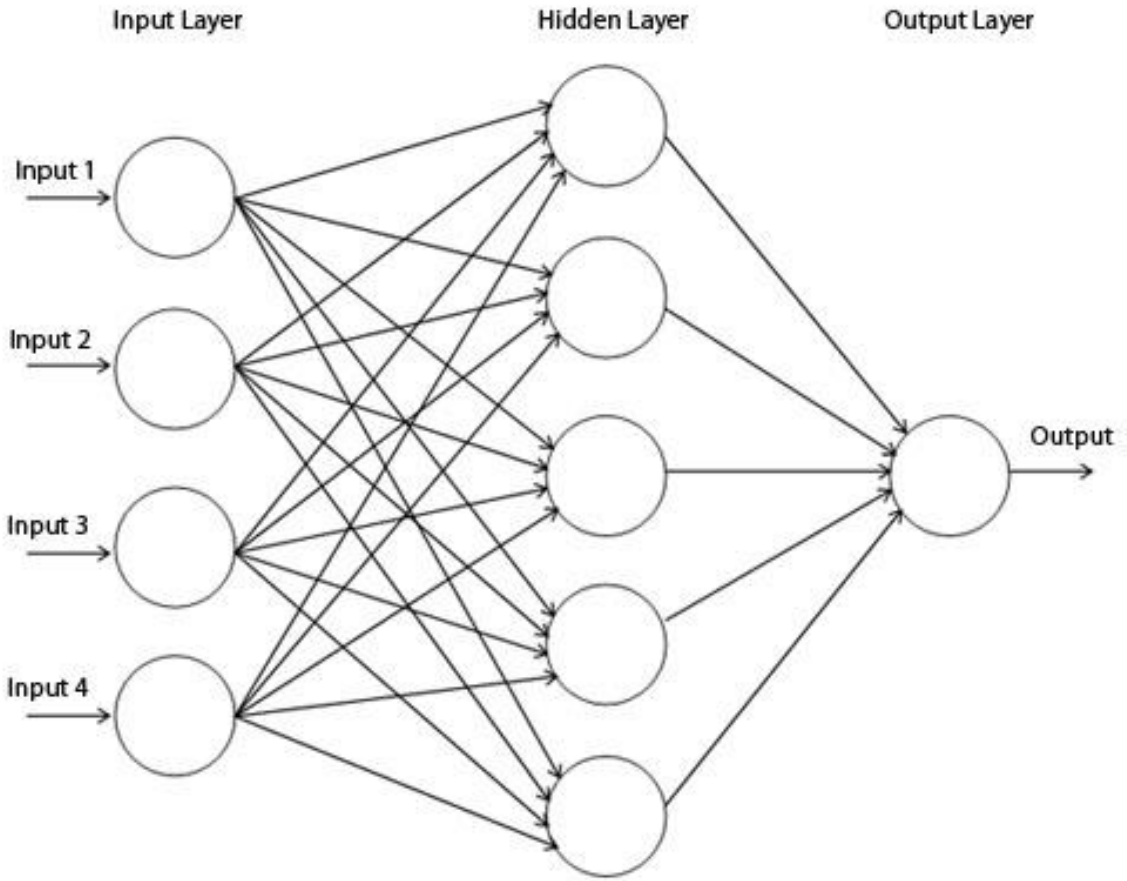
Для класса задач:

- максимизирует производительность на классе проблем;
- легко модифицировать/ограничить.

Архитектуры нейронных сетей

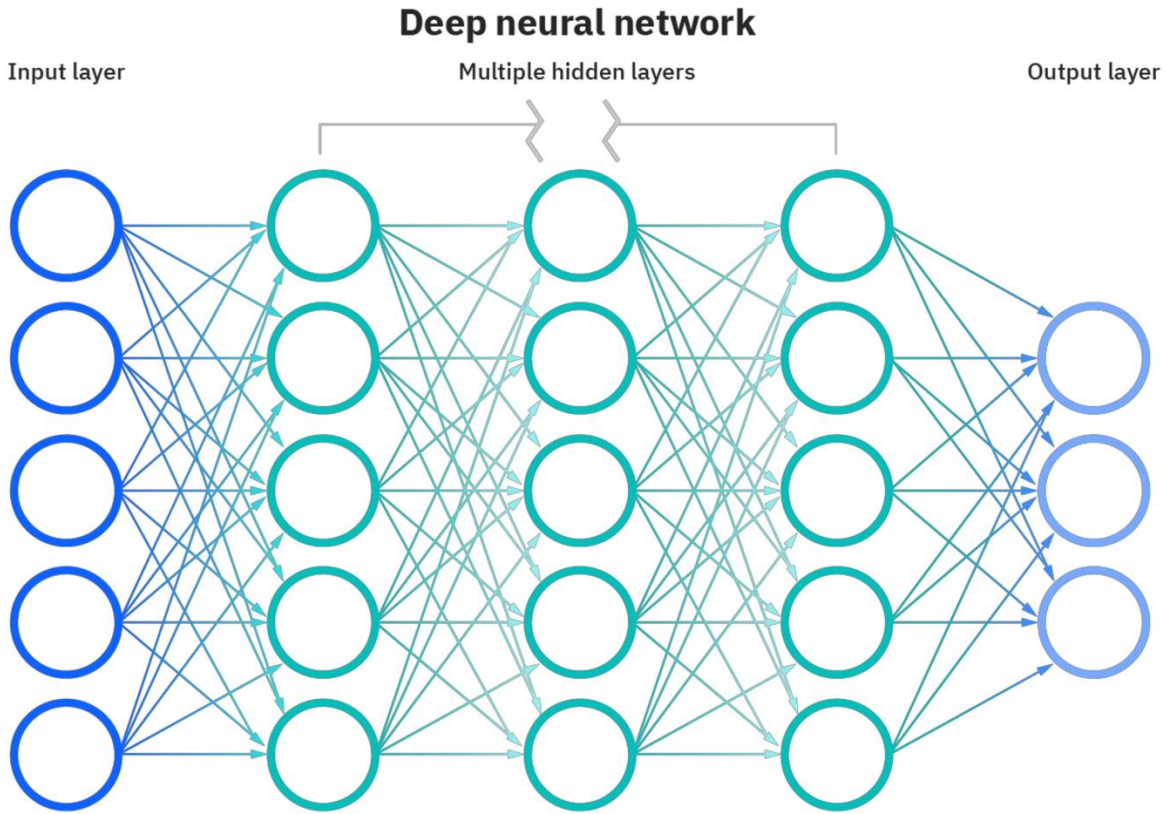
Один скрытый слой

- легко оптимизировать;
- универсальный аппроксиматор;
- часто требуется большое количество скрытых нейронов.



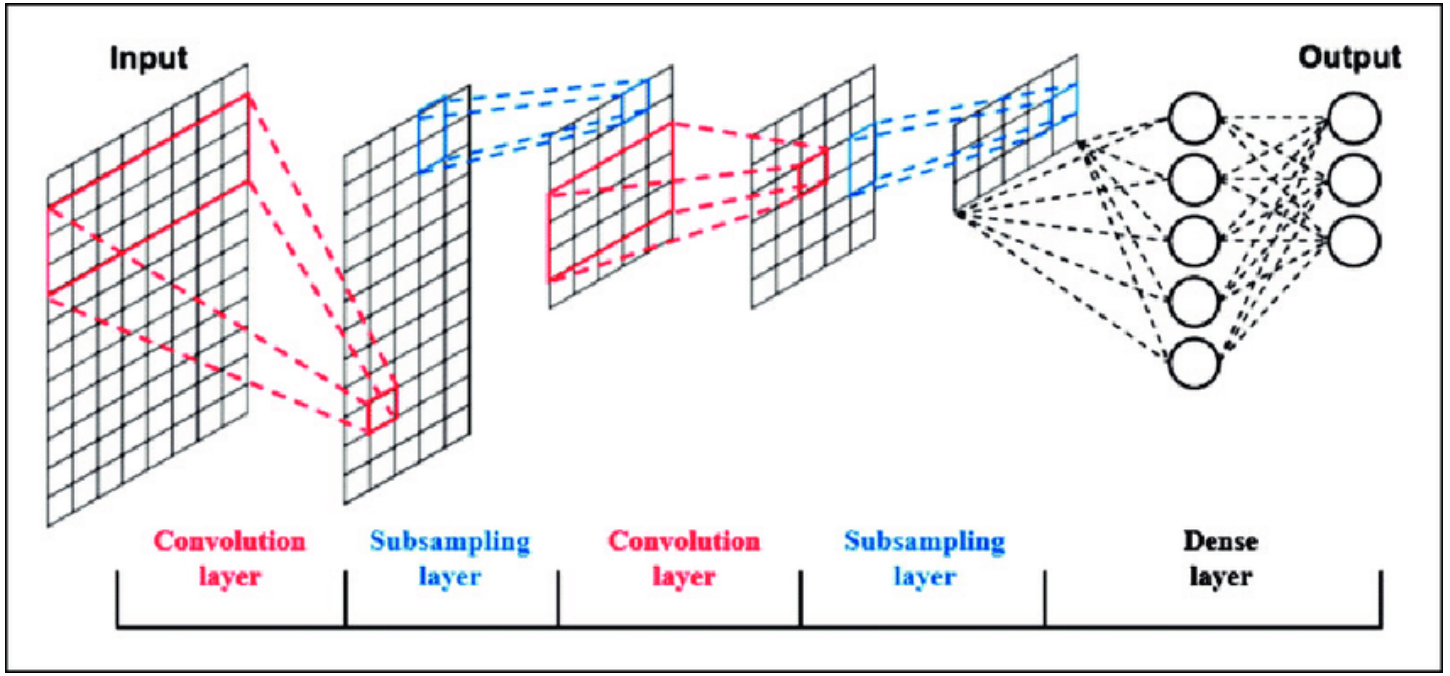
Полносвязная сеть

- труднее оптимизировать;
- лучше подходит для решения реальных задач;
- типичная сеть имеет 4-7 слоев;
- 10+ слоев - сложнее оптимизировать;
- подходит для широкого класса задач;
- начальная точка для других архитектур.



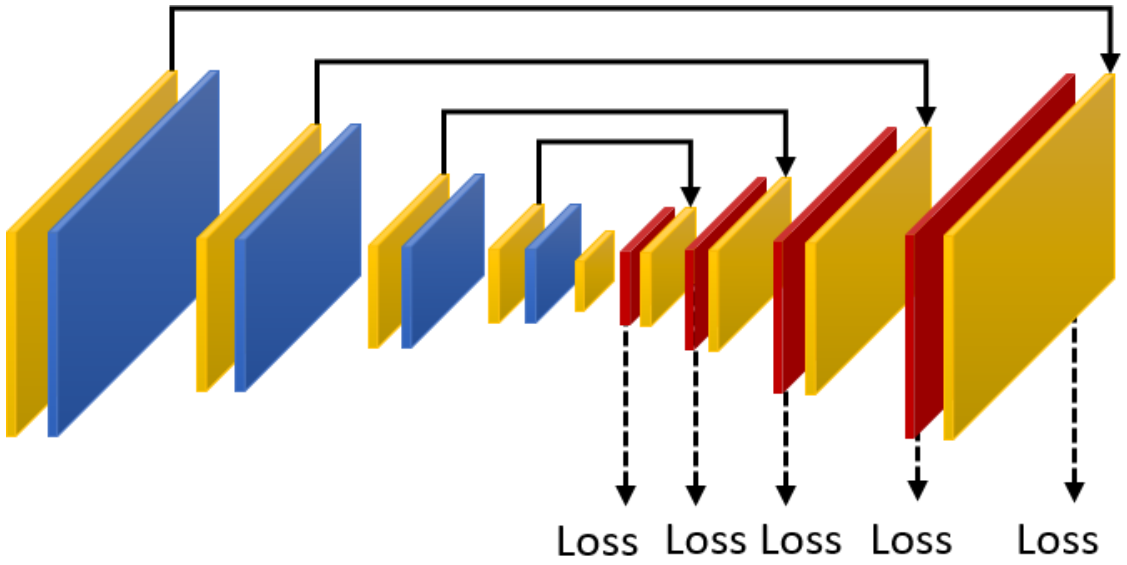
Свёрточные сети

- пространственная/временная структура;
- коммутативный с переносом;
- локальный.



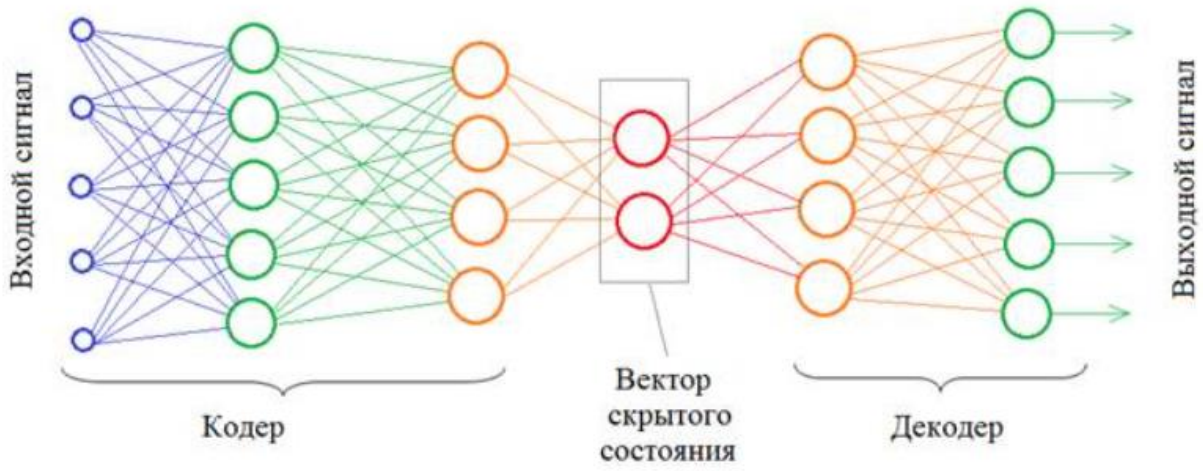
Deeply supervised

- заставляет скрытые слои создавать прогностические характеристики:
  - не всегда хорошая идея;
- сильный регуляризатор;
- прост в реализации;
- отсутствие исчезающих градиентов.



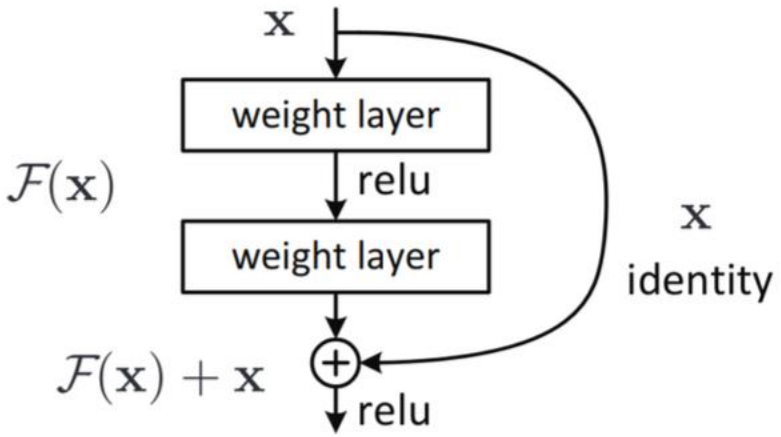
Автокодировщики

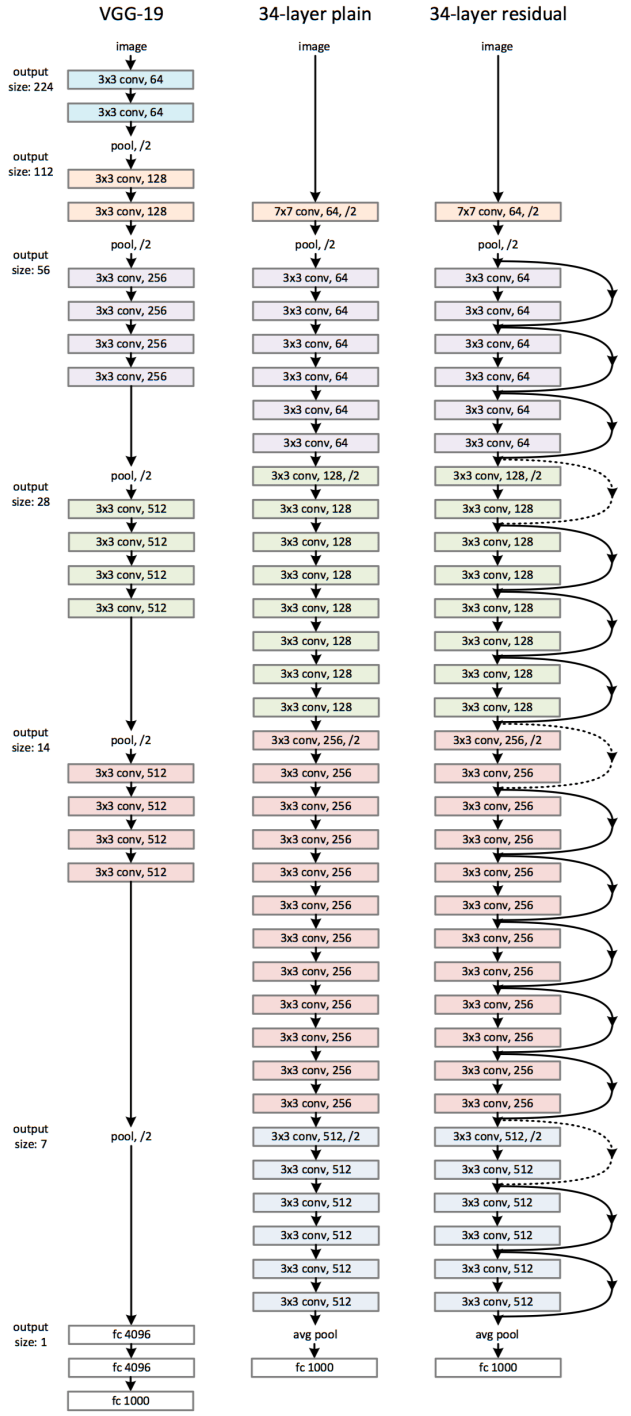
- dimensionality reduction:
  - $\mathcal{X} \mapsto \mathcal{Z}$;
- image to image:
  - $\mathcal{X} \mapsto \mathcal{X}$;
- anomaly detection:
  - $\text{score}(x) = \|\text{decoder}\big(\text{encoder}(x)\big) - x\|^2$;
- auxiliary:
  - denoising;
  - pretraining;
  - auxiliary loss, regularization.



ResNet

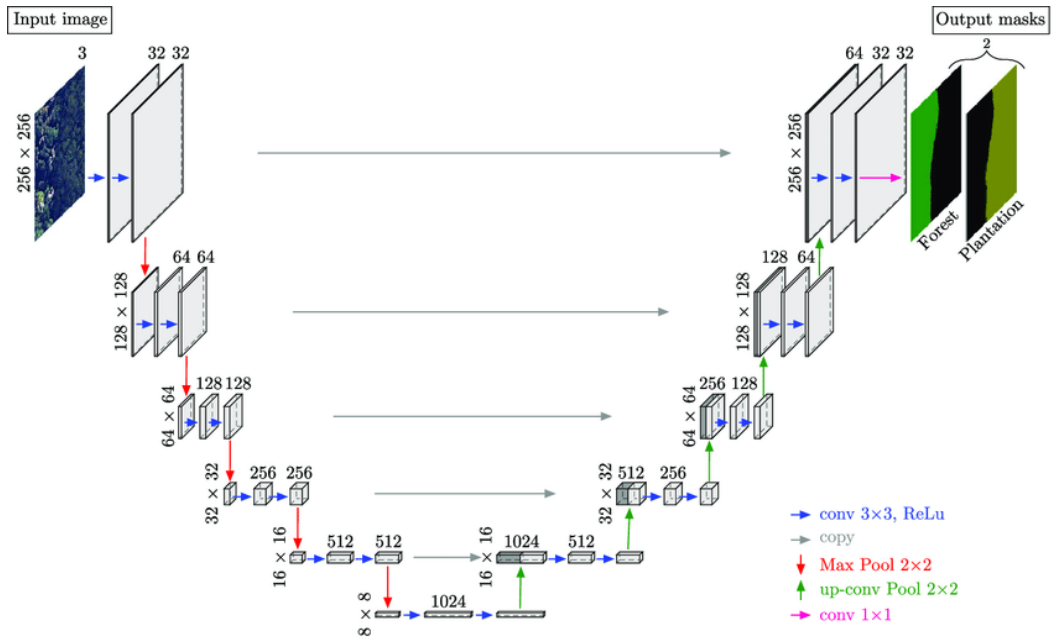
- boosting, neural edition;
- adaptive depth:
  - shallow initially;
- 1000+ layers!





U-NET

- изображение к изображению:
  - увеличение/трансформация; - сегментация;
- легче обучается;
- объединяет глобальную и локальную информацию.

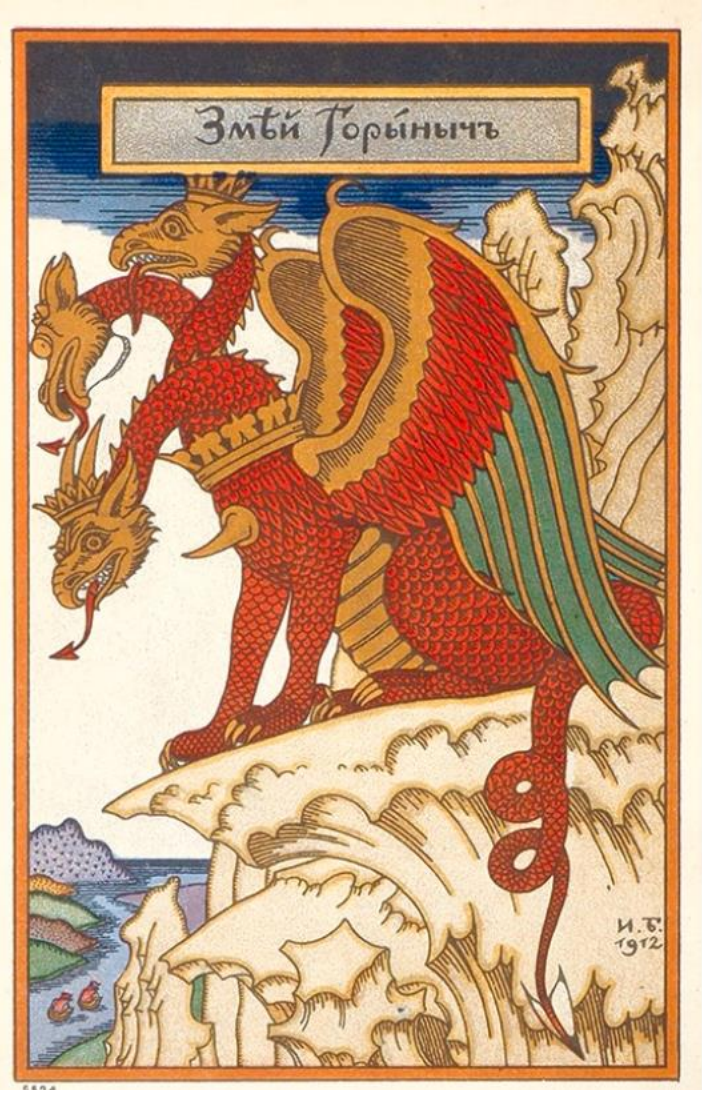


Дополнительные задачи

- similar task tends to share similar features;

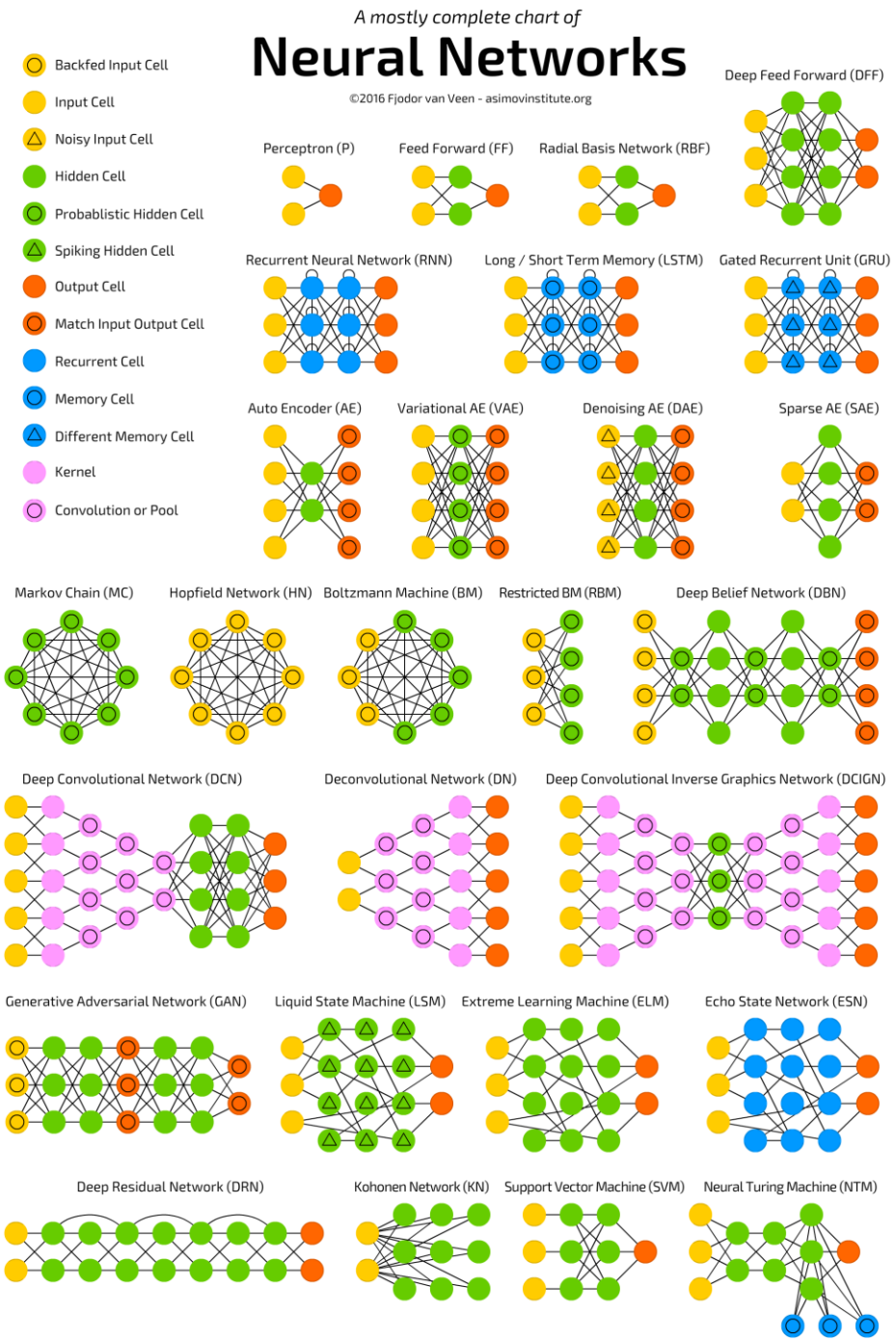
$$\mathcal{L} = \mathcal{L}_{\text{main}} + \sum_i \alpha_i \cdot \mathcal{L}^i_{\text{aux}}$$

- α_i can be decreased during training.



Зоопарк моделей

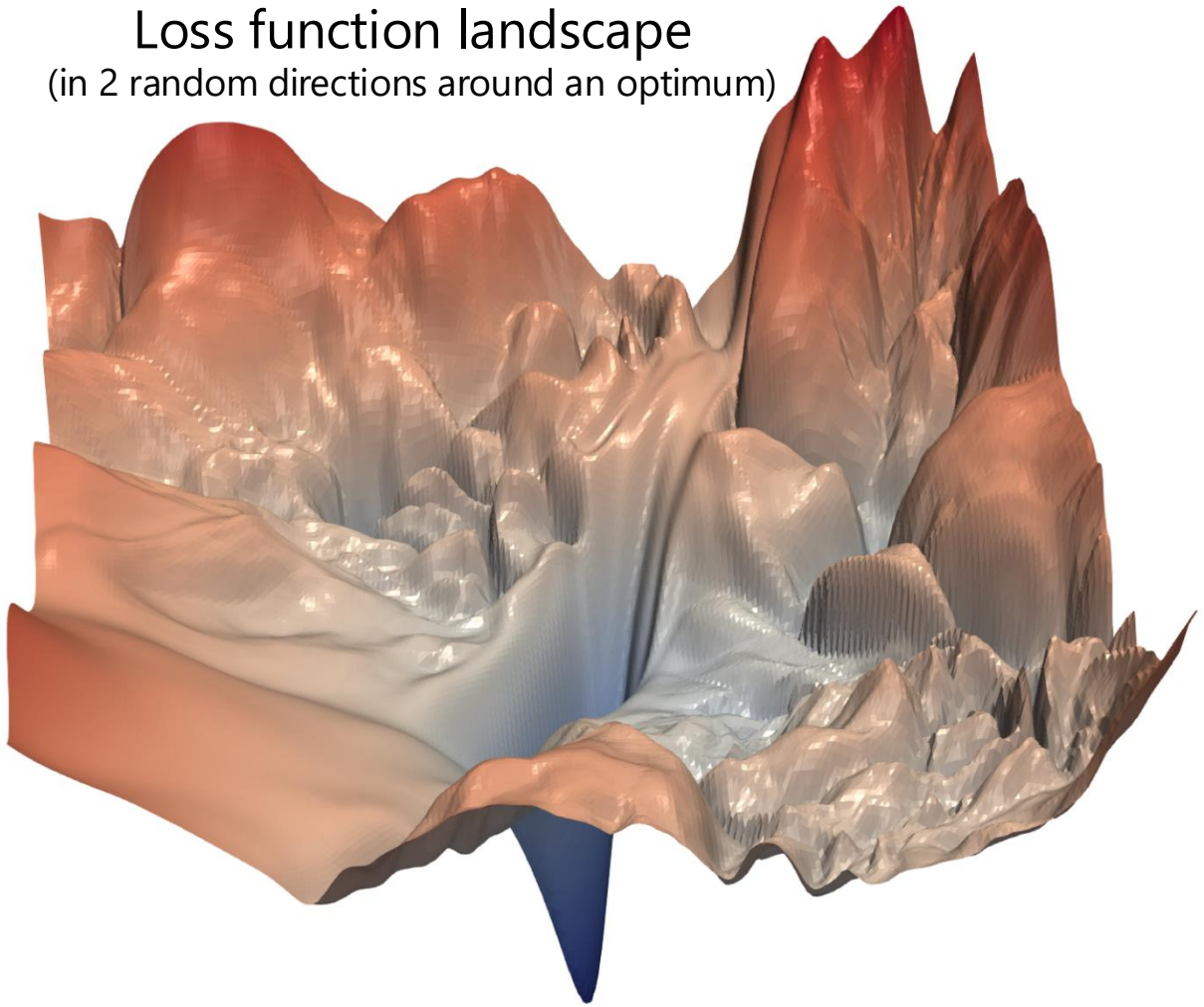
бесчисленное множество других архитектур,
и трюков, и эвристик.



Оптимизация

How to optimize such functions?

Loss function landscape
(in 2 random directions around an optimum)

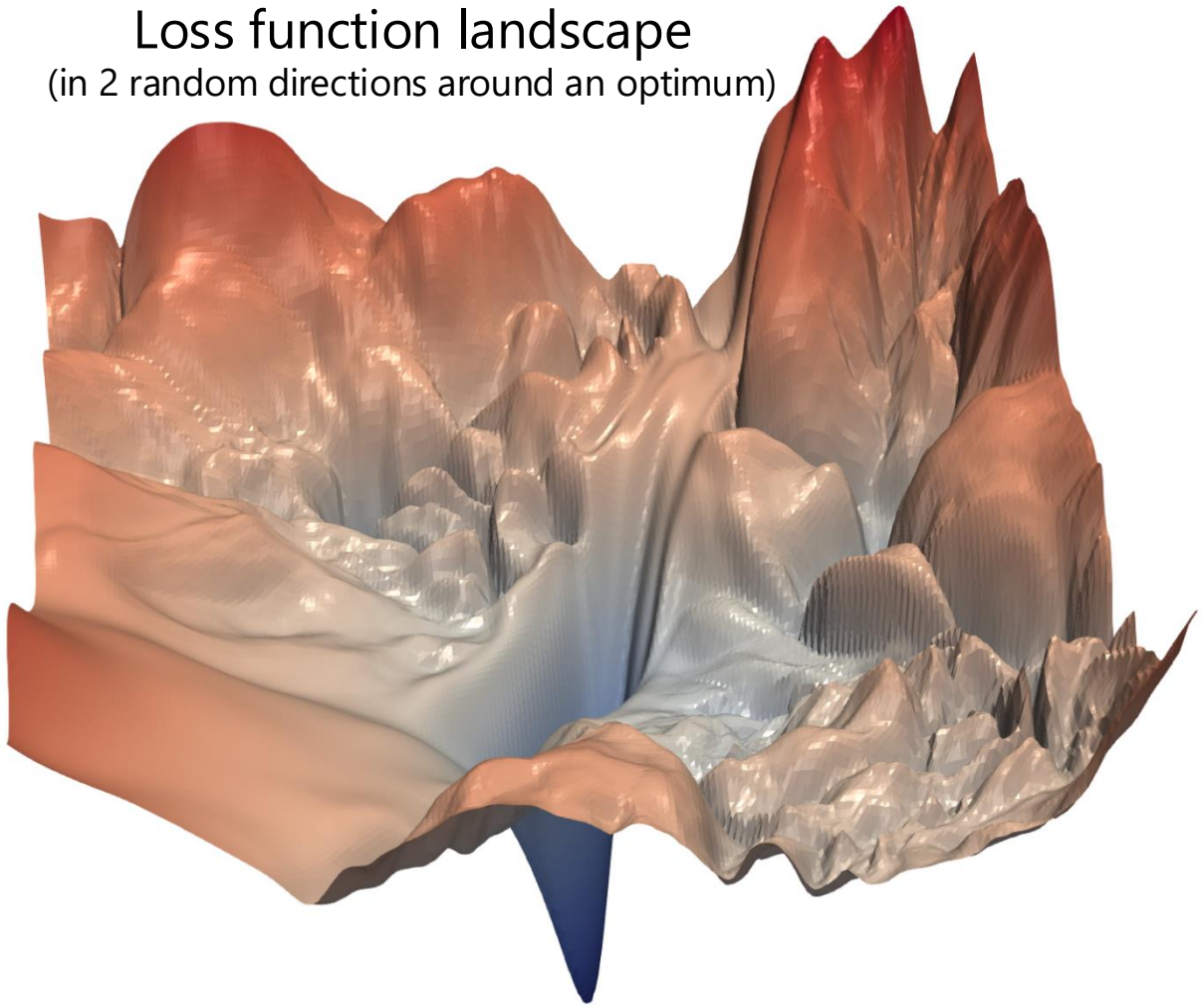


https://papers.nips.cc/paper/7875-visualizing-the-loss-landscape-of-neural-nets

- No convergence guarantees for the stochastic gradient descent

How to optimize such functions?

Loss function landscape
(in 2 random directions around an optimum)

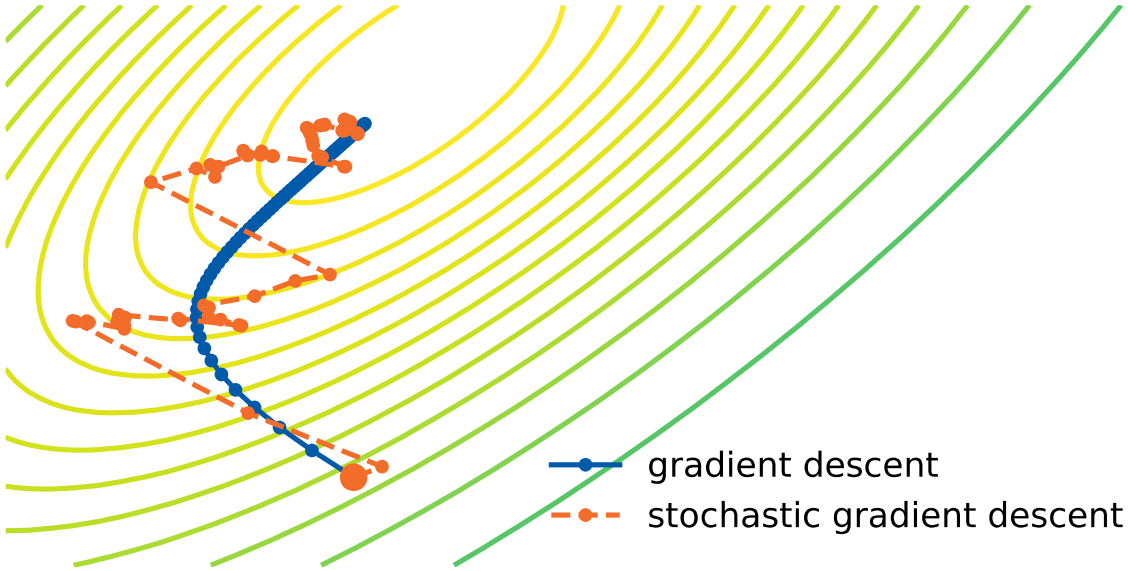


https://papers.nips.cc/paper/7875-visualizing-the-loss-landscape-of-neural-nets

- No convergence guarantees for the stochastic gradient descent
- There's a number of modifications to improve training

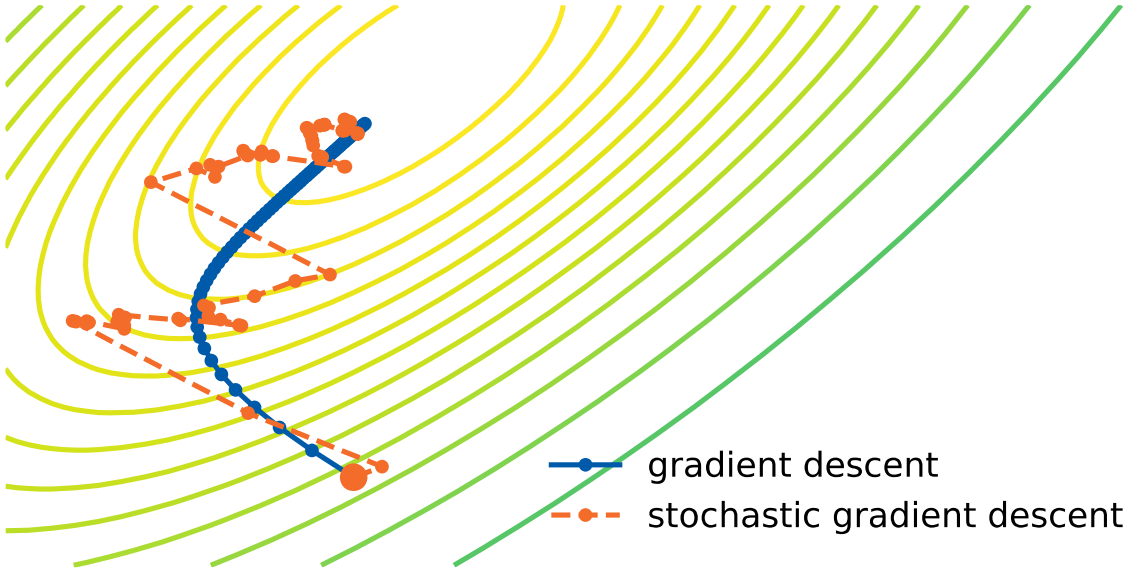
SGD on mini-batches

- SGD:
  - At each step k pick $l_k \in \{1, \ldots, N\}$ at random, then update:
  - $\theta^{(k)} \leftarrow \theta^{(k-1)} - \eta \nabla_\theta \mathcal{L}\left(y_{l_k}, \widehat{f}_\theta\left(x_{l_k}\right)\right)\Big|_{\theta = \theta^{(k-1)}}$



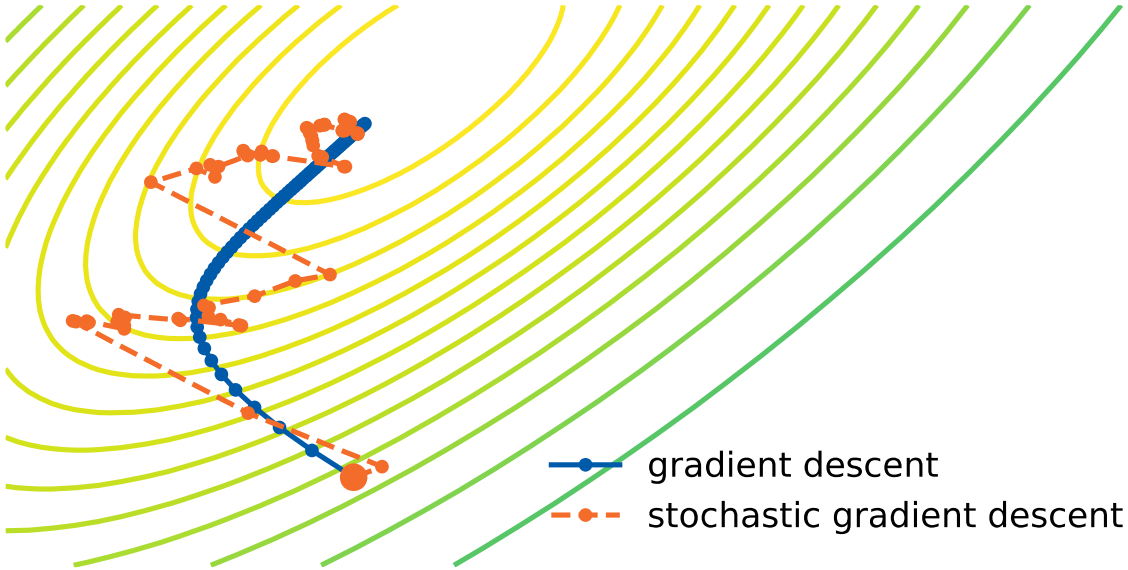
SGD on mini-batches

- SGD:
  - At each step k pick $l_k \in \{1, \ldots, N\}$ at random, then update:
  - $\theta^{(k)} \leftarrow \theta^{(k-1)} - \eta \nabla_\theta \mathcal{L}\left(y_{l_k}, \widehat{f}_\theta(x_{l_k})\right)\Big|_{\theta = \theta^{(k-1)}}$
- Mini-batch SGD:
  - Shuffle the training set, then iterate through it in chunks (batches) of fixed size



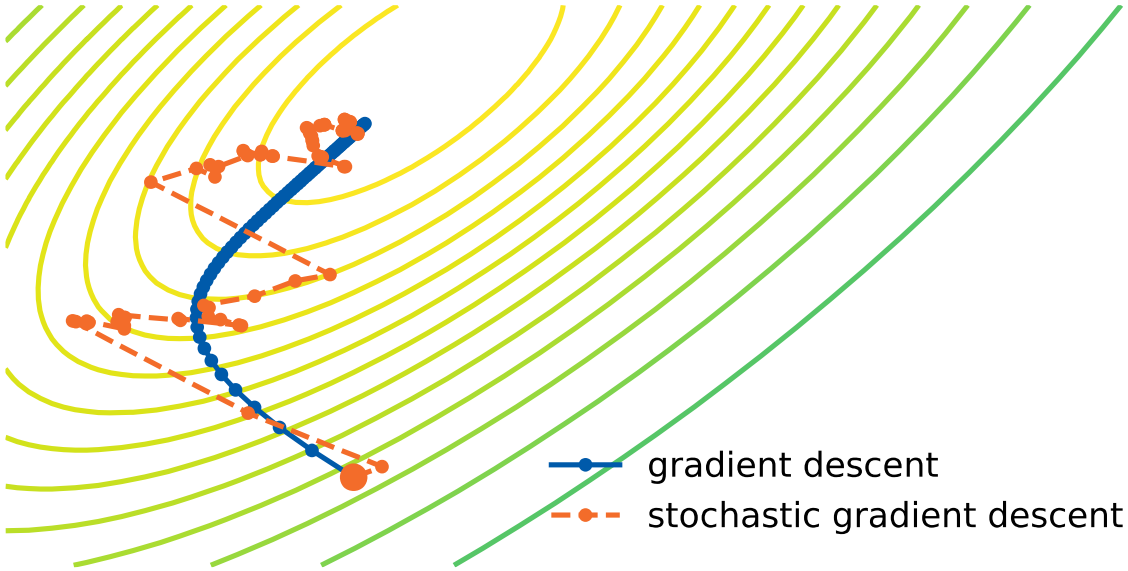
SGD on mini-batches

- SGD:
  - At each step k pick $l_k \in \{1, \ldots, N\}$ at random, then update:
  - $\theta^{(k)} \leftarrow \theta^{(k-1)} - \eta \nabla_\theta \mathcal{L}\left(y_{l_k}, \hat{f}_\theta(x_{l_k})\right)\Big|_{\theta = \theta^{(k-1)}}$
- Mini-batch SGD:
  - Shuffle the training set, then iterate through it in chunks (batches) of fixed size
  - At each iteration evaluate the loss gradients on the given chunk B: $g = \sum_{i \in B} \nabla_\theta \mathcal{L}\left(y_i, \hat{f}_\theta(x_i)\right)$



SGD on mini-batches

- SGD:
  - At each step k pick $l_k \in \{1, \ldots, N\}$ at random, then update:
  - $\theta^{(k)} \leftarrow \theta^{(k-1)} - \eta \nabla_\theta \mathcal{L}\left(y_{l_k}, \widehat{f}_\theta(x_{l_k})\right)\Big|_{\theta = \theta^{(k-1)}}$
- Mini-batch SGD:
  - Shuffle the training set, then iterate through it in chunks (batches) of fixed size
  - At each iteration evaluate the loss gradients on the given chunk B: $g = \sum_{i \in B} \nabla_\theta \mathcal{L}\left(y_i, \widehat{f}_\theta(x_i)\right)$
  - Update the model parameters: $\theta^{(k)} \leftarrow \theta^{(k-1)} - \eta \cdot g$



Summary

- Neural networks are essentially **stacked linear models** with scalar nonlinearities in between
- Earlier layers extract useful features s.t. the problem **becomes solvable** with a linear model (the last layer)
- Neural networks can approximate any function arbitrarily well, given they are deep/wide enough
- Loss functions typically become highly **non-convex** for neural networks
  - this makes the optimization process harder
- A variety of **SGD modifications** are available to mitigate this problem
- Food for thought: being the 'universal approximators', can neural nets really solve every possible supervised learning problem?