Линейные модели и нейронные сети

1. Линейные модели в задачах классификации

Отступ (margin)

Отступом алгоритма $a(x) = sign\{f(x)\}$ на объекте x_i называется величина

$$M_i = y_i f(x_i)$$

(y_i - класс, к которому относится x_i)

$$M_i \leq 0 \Leftrightarrow y_i \neq a(x_i)$$
$$M_i > 0 \Leftrightarrow y_i = a(x_i)$$

Функция потерь

$$Q(w) = \sum_{i=1}^{\ell} [M_i(w) \leq 0]$$

Функция потерь

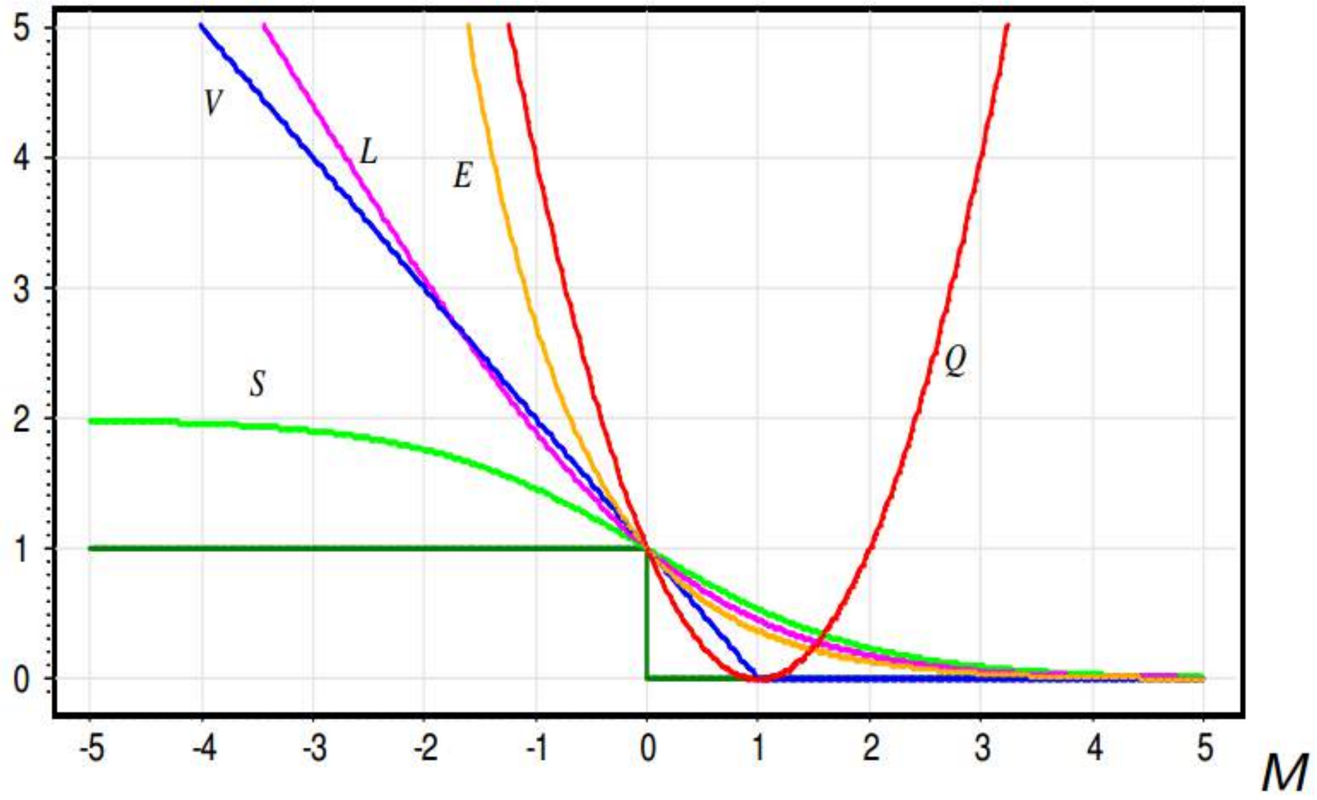
$$Q(w) = \sum_{i=1}^{\ell} [M_i(w) \leq 0] \leqslant \widetilde{Q}(w) = \sum_{i=1}^{\ell} \mathscr{L}(M_i(w)) \to \min_w;$$

Функция эмпирического риска

Функция потерь

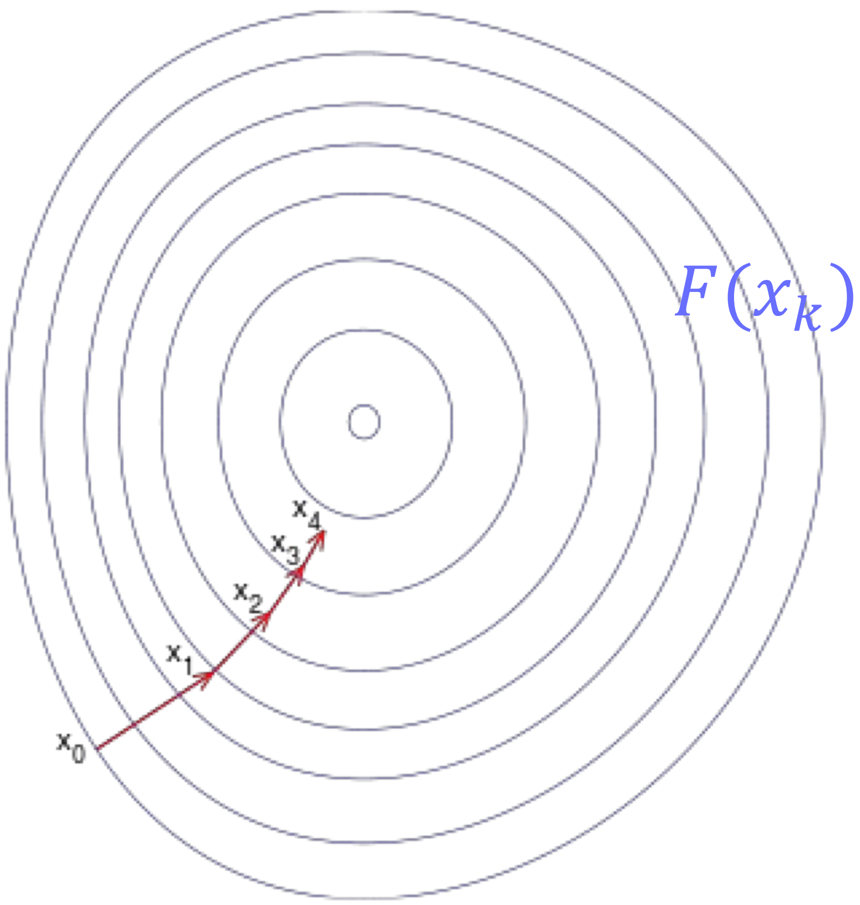
Функция потерь

$$Q(w) = \sum_{i=1}^{\ell} [M_i(w) \le 0] \leqslant \widetilde{Q}(w) = \sum_{i=1}^{\ell} \mathscr{L}(M_i(w)) \to \min_w;$$



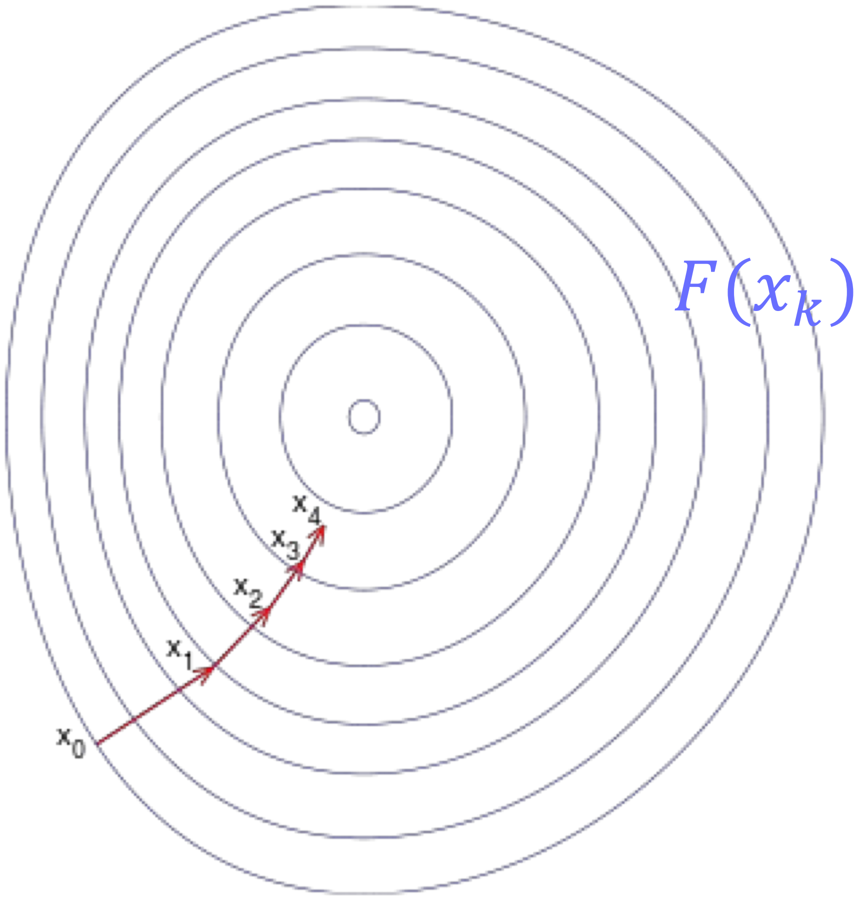
$$Q(M) = (1 - M)^2$$
$$V(M) = (1 - M)_+$$
$$S(M) = 2(1 + e^M)^{-1}$$
$$L(M) = \log_2(1 + e^{-M})$$
$$E(M) = e^{-M}$$

Градиентный спуск (GD, Gradient Decent)



Градиентный спуск (GD, Gradient Decent)

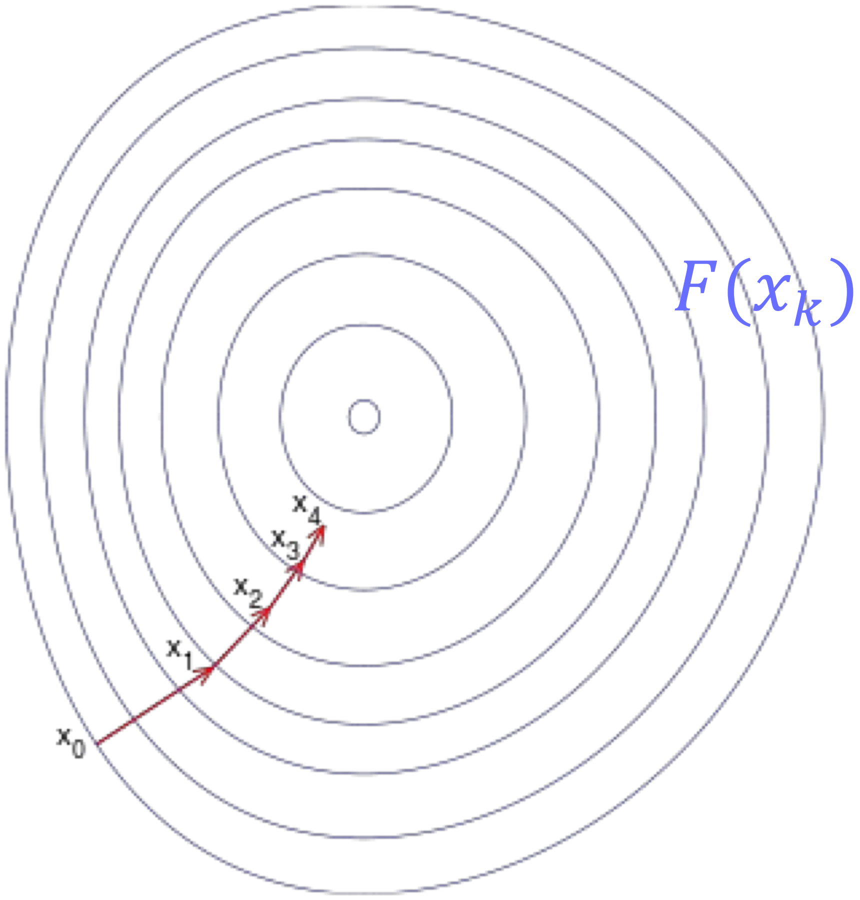
$$x_{k+1} = x_k - \gamma_k \nabla F(x_k)$$



Градиентный спуск (GD, Gradient Decent)

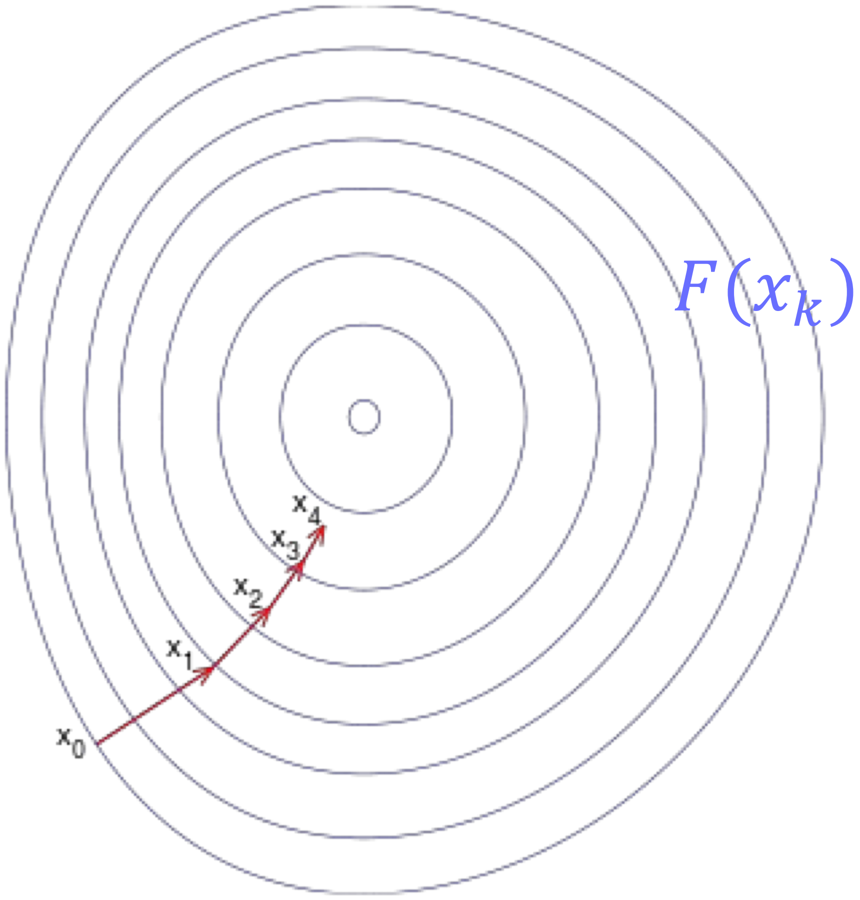
$$x_{k+1} = x_k - \gamma_k \nabla F(x_k)$$

$$\nabla_w \tilde{Q} = \sum_{i=1}^{l} \nabla L(M_i) = \sum_{i=1}^{l} L'(M_i) \frac{\partial M_i}{\partial w}$$



Градиентный спуск (GD, Gradient Decent)

$$x_{k+1} = x_k - \gamma_k \nabla F(x_k)$$

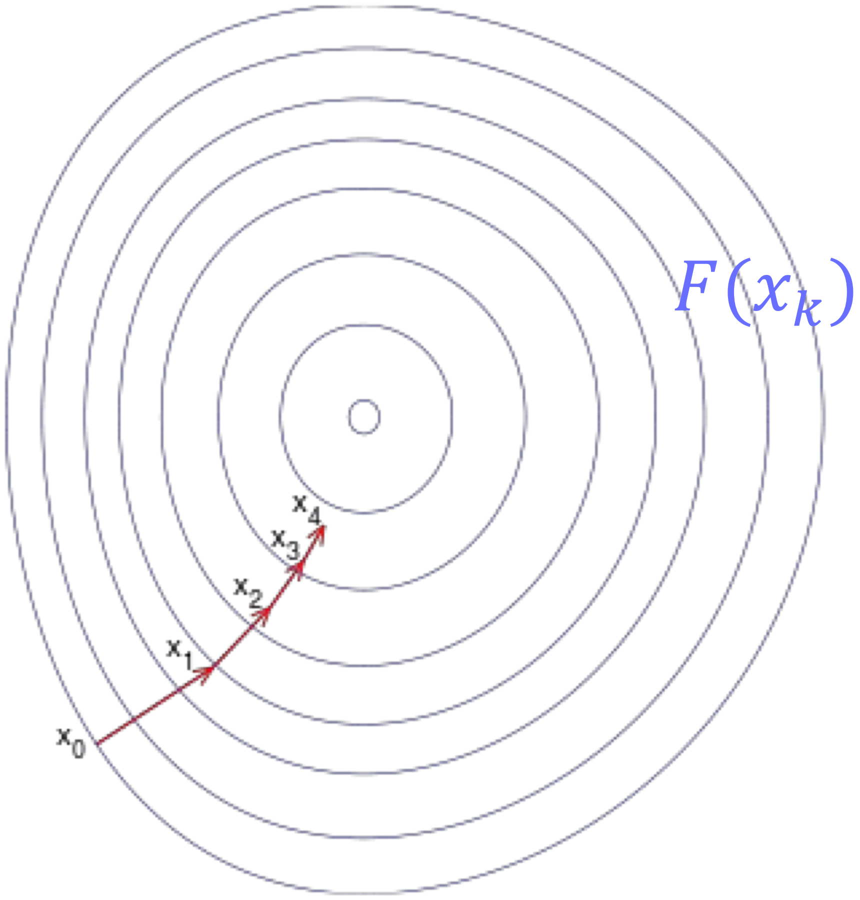


$$\nabla_w \tilde{Q} = \sum_{i=1}^{l} \nabla L(M_i) = \sum_{i=1}^{l} L'(M_i) \frac{\partial M_i}{\partial w}$$

$$M_i = y_i \langle w, x_i \rangle \Longrightarrow \frac{\partial M_i}{\partial w} = y_i x_i$$

Градиентный спуск (GD, Gradient Decent)

$$x_{k+1} = x_k - \gamma_k \nabla F(x_k)$$



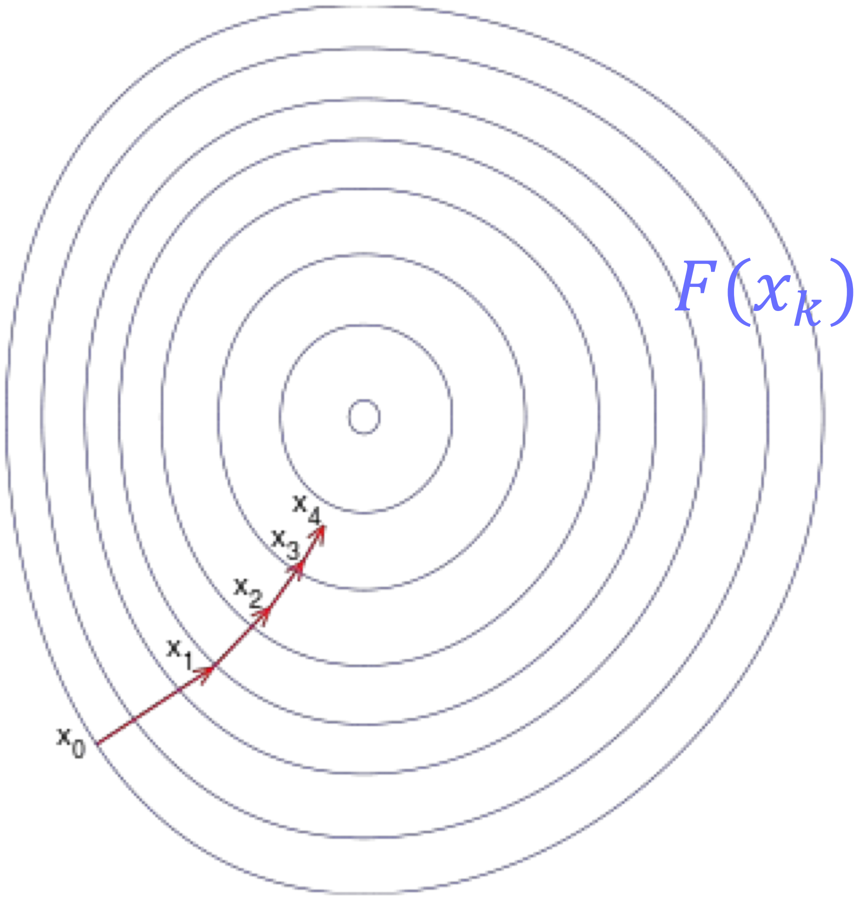
$$\nabla_w \tilde{Q} = \sum_{i=1}^{l} \nabla L(M_i) = \sum_{i=1}^{l} L'(M_i) \frac{\partial M_i}{\partial w}$$

$$M_i = y_i \langle w, x_i \rangle \implies \frac{\partial M_i}{\partial w} = y_i x_i$$

$$\nabla \tilde{Q} = \sum_{i=1}^{l} y_i x_i L'(M_i)$$

Градиентный спуск (GD, Gradient Decent)

$$x_{k+1} = x_k - \gamma_k \nabla F(x_k)$$



$$\nabla_w \tilde{Q} = \sum_{i=1}^{l} \nabla L(M_i) = \sum_{i=1}^{l} L'(M_i) \frac{\partial M_i}{\partial w}$$

$$M_i = y_i \langle w, x_i \rangle \implies \frac{\partial M_i}{\partial w} = y_i x_i$$

$$\nabla \tilde{Q} = \sum_{i=1}^{l} y_i x_i L'(M_i)$$

$$\boxed{w_{k+1} = w_k - \gamma_k \sum_{i=1}^{l} y_i x_i L'(M_i)}$$

Стохастический градиент (SGD)

$$w_{k+1} = w_k - \gamma_k \cancel{\sum_{i=1}^{l}} y_i x_i L'(M_i)$$

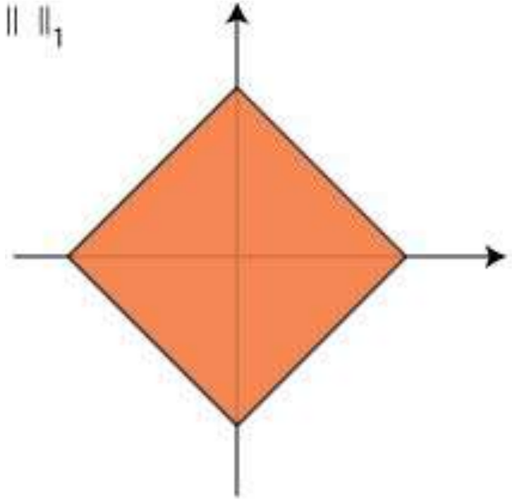
$$\boxed{w_{k+1} = w_k - \gamma_k y_i x_i L'(M_i)}$$

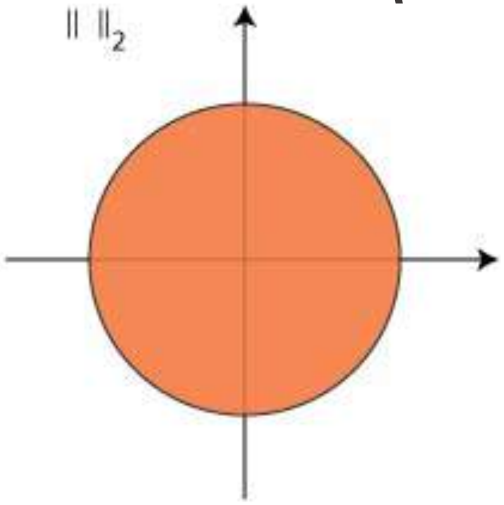
x_i – случайный элемент обучающей выборки

Регуляризация

$$\begin{cases} \tilde{Q} = \sum_{i=1}^{l} L(M_i) \to min \\ \sum_{n=1}^{d} |w_n| \le \tau \end{cases}$$

$$\begin{cases} \tilde{Q} = \sum_{i=1}^{l} L(M_i) \to min \\ \sum_{n=1}^{d} {w_n}^2 \le \tau \end{cases}$$

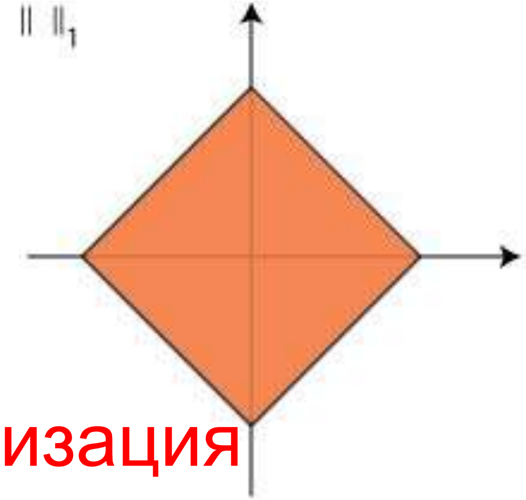


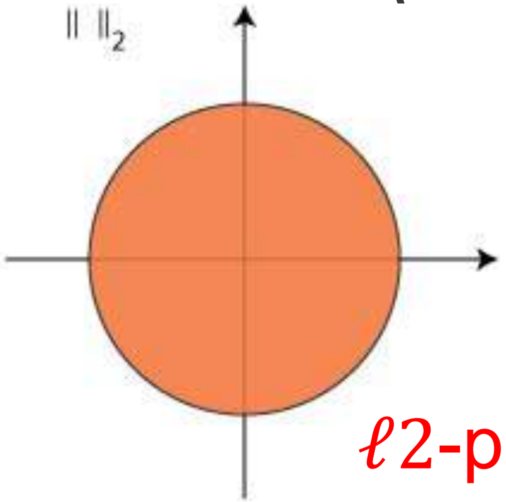


Регуляризация

$$\begin{cases} \tilde{Q} = \sum_{i=1}^{l} L(M_i) \to min \\ \sum_{n=1}^{d} |w_n| \leq \tau \end{cases}$$

$$\begin{cases} \tilde{Q} = \sum_{i=1}^{l} L(M_i) \to min \\ \sum_{n=1}^{m} {w_n}^2 \leq \tau \end{cases}$$



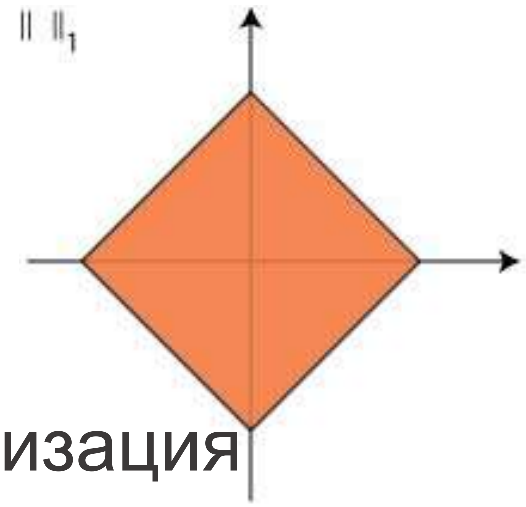


$\ell 1$-регуляризация

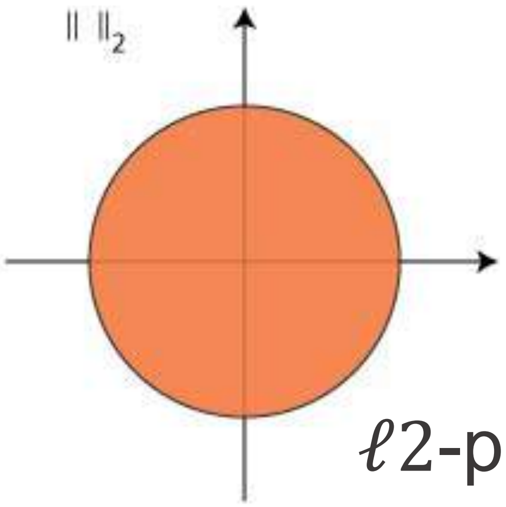
$\ell 2$-регуляризация

Регуляризация

$$\sum_{i=1}^{l} L(M_i) + \gamma \sum_{n=1}^{d} |w_n| \to min \qquad \sum_{i=1}^{l} L(M_i) + \gamma \sum_{n=1}^{d} {w_n}^2 \to min$$



$\ell 1$-регуляризация



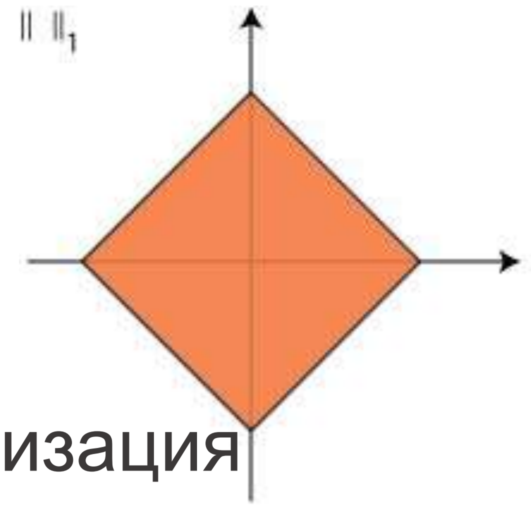
$\ell 2$-регуляризация

Регуляризация

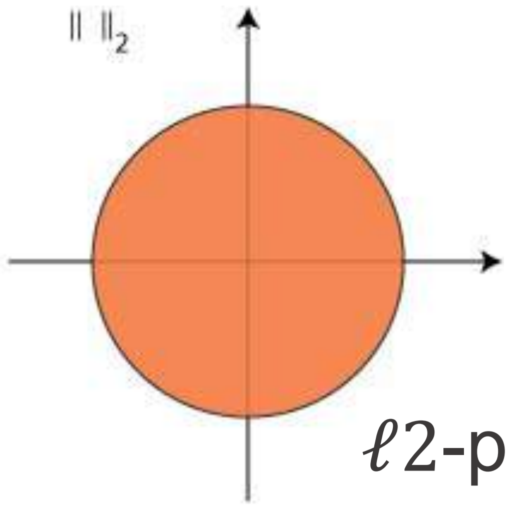
$$\sum_{i=1}^{l} L(M_i) + \gamma \sum_{n=1}^{d} |w_n| \to min$$

$$\sum_{i=1}^{l} L(M_i) + \gamma \sum_{n=1}^{d} {w_n}^2 \to min$$

Вопрос:
вы заметили, что в регуляризатор не включается вес w_o?



$\ell 1$-регуляризация



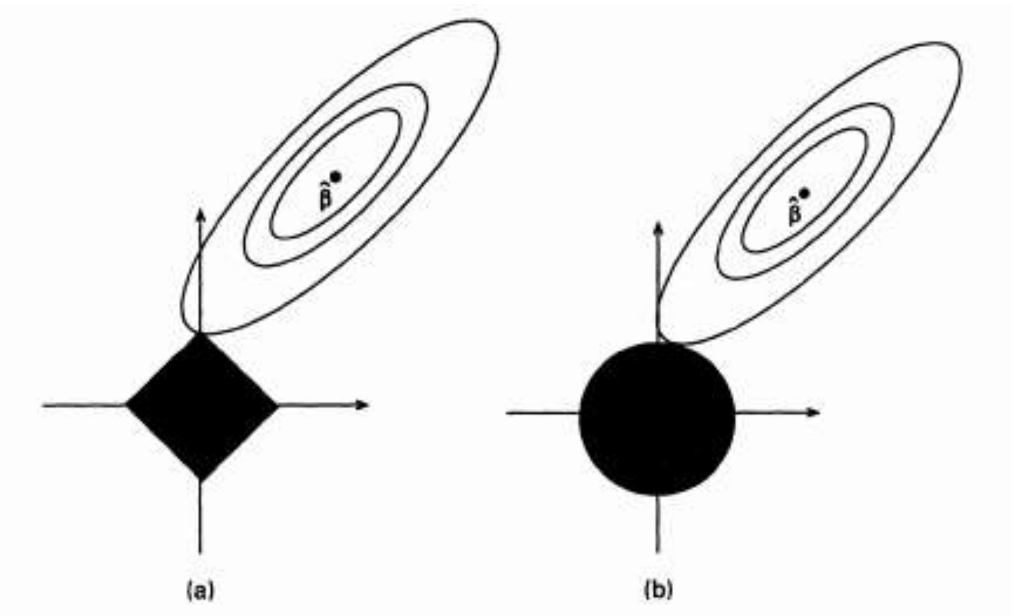
$\ell 2$-регуляризация

Различия между $\ell 1$ и $\ell 2$

- Разреженность – $\ell 1$-регуляризация делает вектор весов более разреженным (содержащим больше нулей)
- В случае линейной классификации это означает отбор признаков: признаки с нулевыми весами не используются в классификации

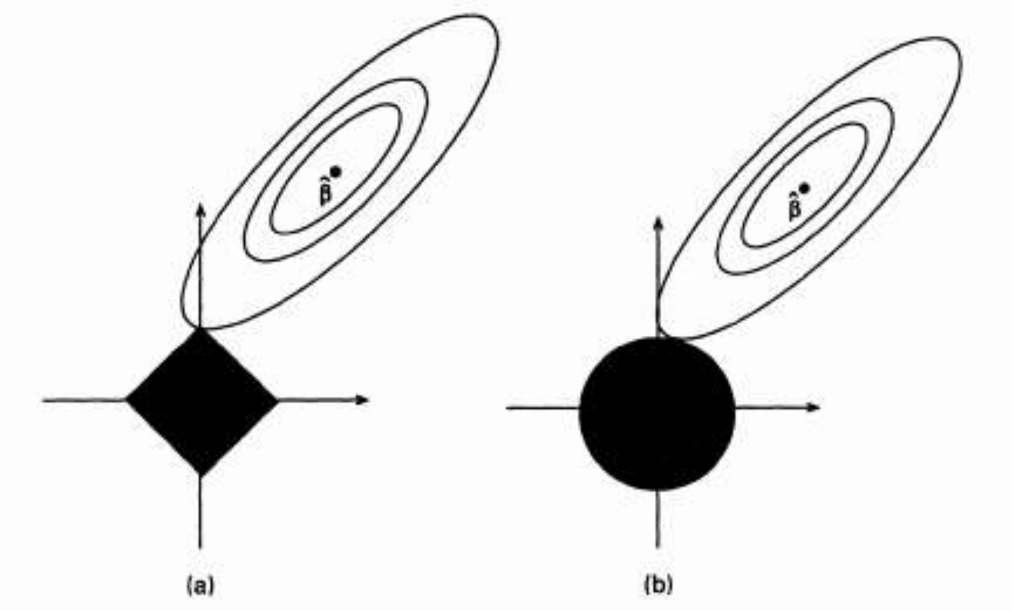
Различия между $\ell 1$ и $\ell 2$

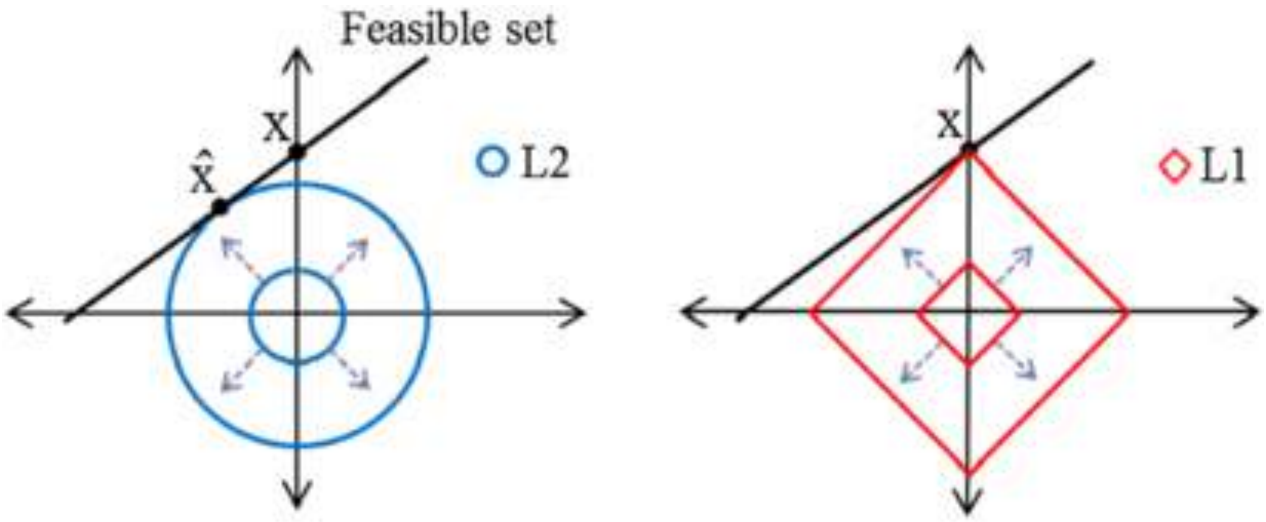
- Разреженность – $\ell 1$-регуляризация делает вектор весов более разреженным (содержащим больше нулей)
- В случае линейной классификации это означает отбор признаков: признаки с нулевыми весами не используются в классификации



Различия между $\ell 1$ и $\ell 2$

- Разреженность – $\ell 1$-регуляризация делает вектор весов более разреженным (содержащим больше нулей)
- В случае линейной классификации это означает отбор признаков: признаки с нулевыми весами не используются в классификации





Стандартные линейные классификаторы

Классификатор	Функция потерь	Регуляризатор
SVM (Support vector machine, метод опорных векторов)	$L(M) = \max\{0, 1 - M\} = (1 - M)_+$	$\sum_{k=1}^{m} {w_k}^2$
Логистическая регрессия	$L(M) = \log(1 + e^{-M})$	Обычно $\sum_{k=1}^{m} {w_k}^2$ или $\sum_{k=1}^{m} \lvert w_k \rvert$

Общий случай

$$Q = \sum_{i=1}^{\ell} L\big(y_i, f(x_i)\big) + \gamma V(w) \to \min_w$$

Функция потерь

Регуляризатор

Коэффициент регуляризации

2. Линейные модели в задачах регрессии

Линейные модели в регрессии

$$a(x) = \langle w, x \rangle + w_0$$

Линейные модели в регрессии

$$a(x) = \langle w, x \rangle + w_0$$

$$Q = \sum_{i=1}^{l} L\big(y_i, a(x_i)\big) \to \min_{w}$$

Линейные модели в регрессии

$$a(x) = \langle w, x \rangle + w_0$$

$$Q = \sum_{i=1}^{l} L\big(y_i, a(x_i)\big) \to \min_{w}$$

$$L\big(y_i, a(x_i)\big) = \ (y_i - a(x_i))^2 \qquad L\big(y_i, a(x_i)\big) = |y_i - a(x_i)|$$

Линейные модели в регрессии

$$a(x) = \langle w, x \rangle + w_0$$

$$Q = \sum_{i=1}^{l} L\big(y_i, a(x_i)\big) + \gamma V(w) \to \min_w$$

$$L\big(y_i, a(x_i)\big) = \left(y_i - a(x_i)\right)^2$$

$$L\big(y_i, a(x_i)\big) = |y_i - a(x_i)|$$

$$V(w) = \|w\|_{l2}^2 = \sum_{n=1}^{d} w_n^2$$

$$V(w) = \|w\|_{l1} = \sum_{n=1}^{d} |w_n|$$

Линейные модели в регрессии

$$a(x) = \langle w, x \rangle + w_0$$

$$Q = \sum_{i=1}^{l} L\big(y_i, a(x_i)\big) + \gamma V(w) \to \min_w$$

Гребневая регрессия (Ridge regression):

$$V(w) = \|w\|_{l2}^2 = \sum_{n=1}^{d} w_n^2$$

LASSO (least absolute shrinkage and selection operator):

$$V(w) = \|w\|_{l1} = \sum_{n=1}^{d} |w_n|$$

Линейная регрессия

$$a(x) = \langle w, x \rangle + w_0$$

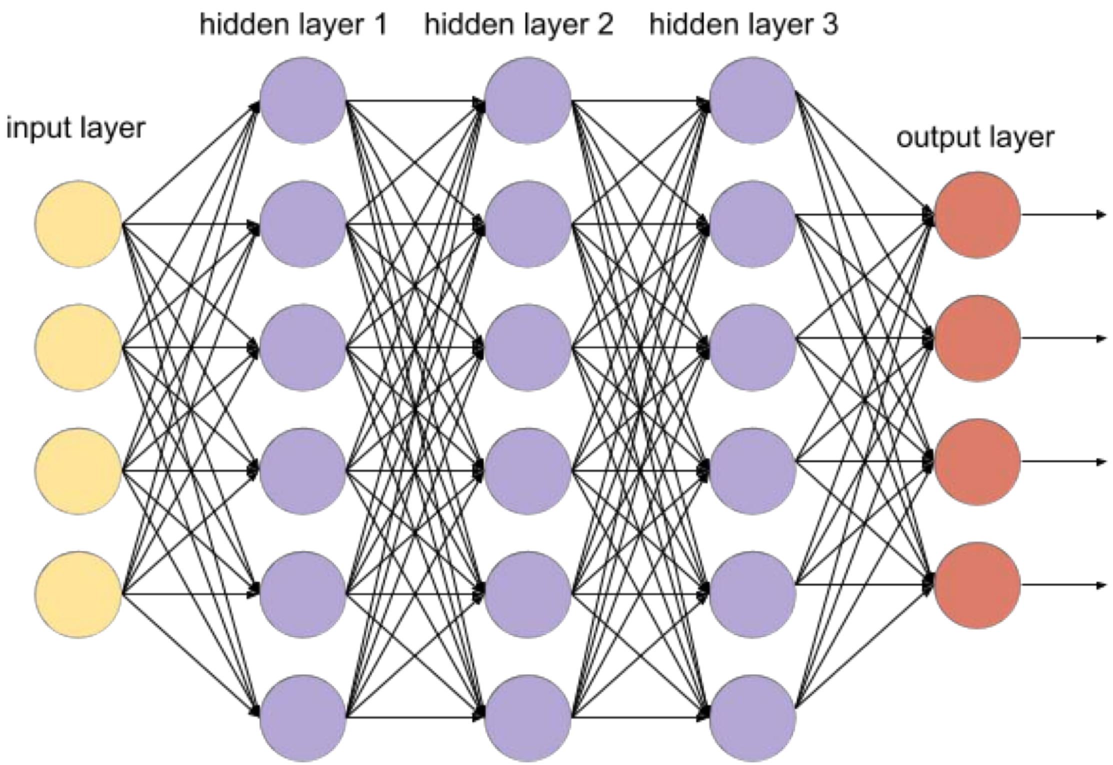
$$Q = \sum_{i=1}^{l} L\big(y_i, a(x_i)\big) \to \min_{w}$$

$$L\big(y_i, a(x_i)\big) = \left(y_i - a(x_i)\right)^2$$

А без регуляризатора и с квадратичными потерями получаем привычную нам линейную регрессию

3. Нейронные сети

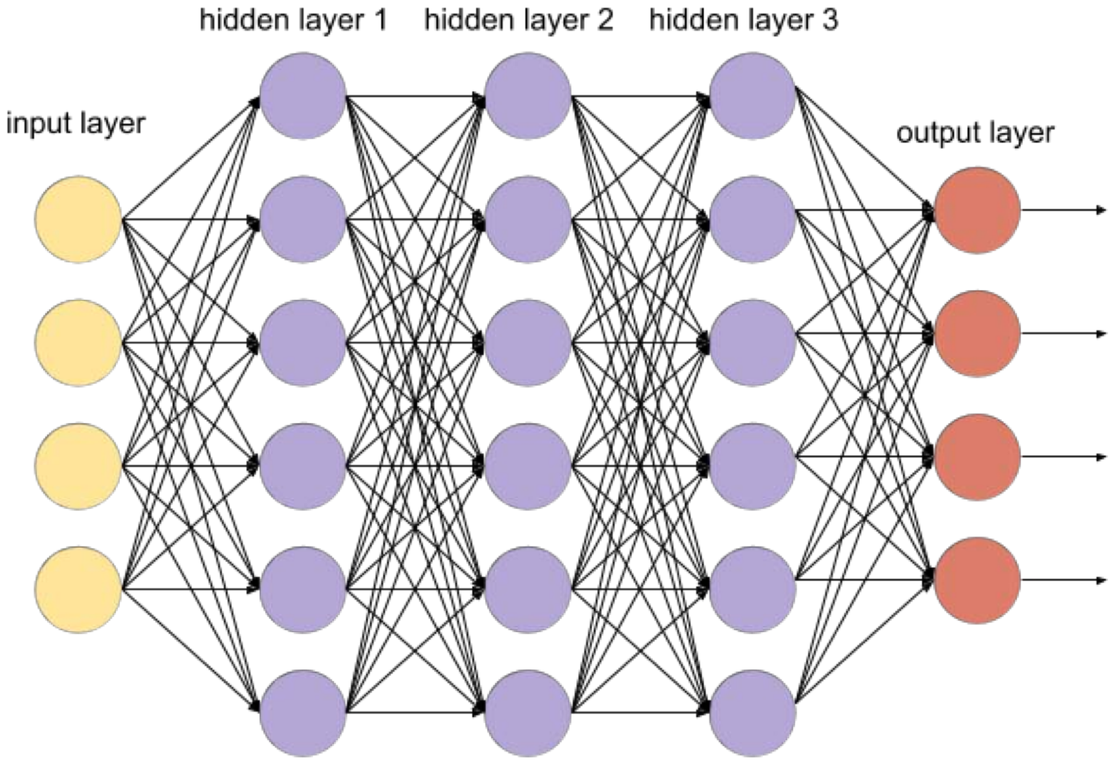
Нейронная сеть



$$h_t = f(W h_{t-1} + b)$$

$$a(x) = f_3(W_3 f_2(W_2 f_1(W_1 f_0(W_0 x + b_0) + b_1) + b_2) + b_3)$$

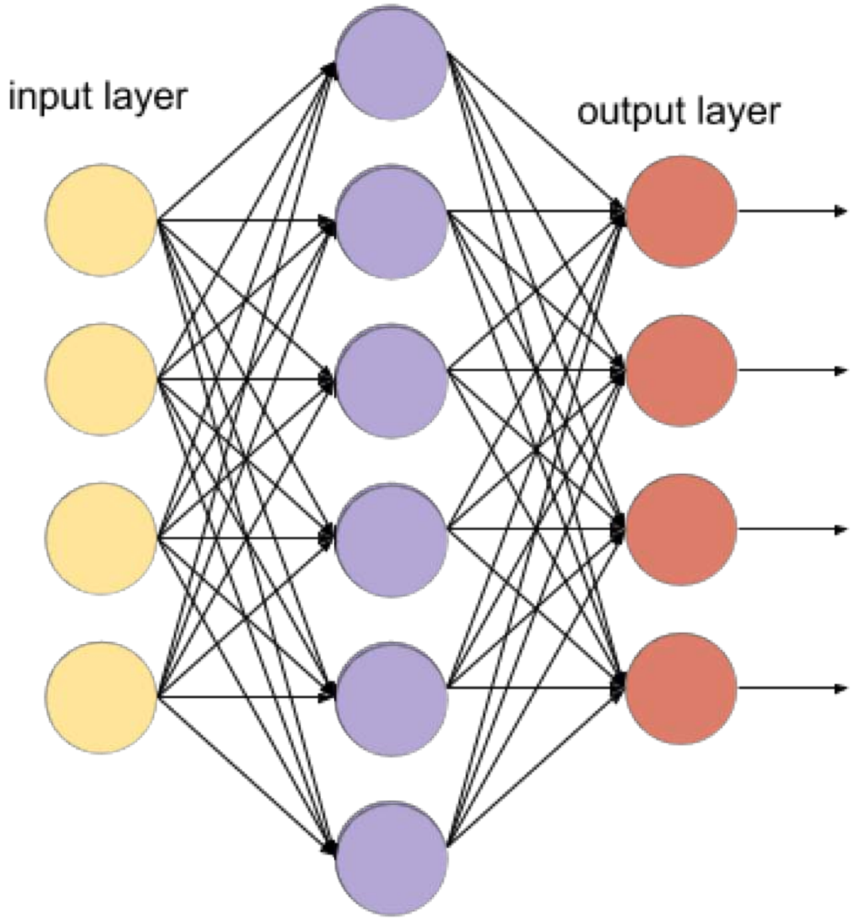
Нейронная сеть



$$h_t = f(W h_{t-1} + b)$$

$$a(x) = f_3(W_3 f_2(W_2 f_1(W_1 f_0(W_0 x + b_0) + b_1) + b_2) + b_3)$$

Универсальная теорема аппроксимации



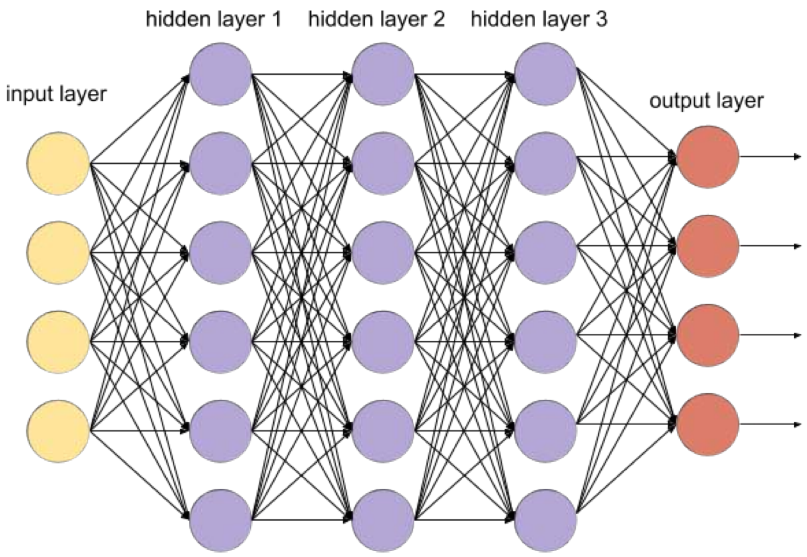
Капелька оптимизма перед обсуждением обучения:
В 1989 г. Джорджем Цыбенко (George Cybenko) была доказана Universal Approximation Theorem

Нестрого:
Если нам дана функция f и сказано, с какой точностью ее нужно приблизить (какой бы эта точность ни была) – мы всегда справимся с задачей даже однослойной нейросетью, т.е. сможем подобрать подходящее количество нейронов и веса

Чуть более строго (для математиков):
f должна быть непрерывна на некотором компакте в R^n и условия теоремы выполняются на нём же

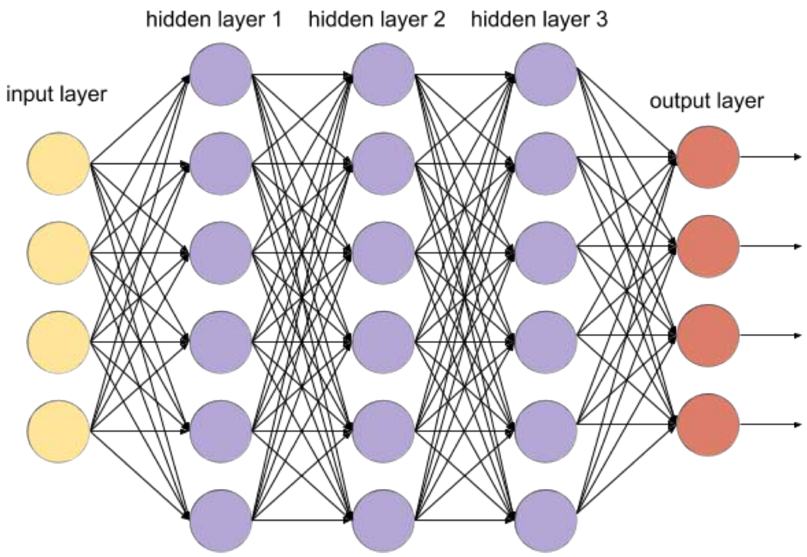
4. Обучение сети

Обучение сети



Задача обучения – настроить веса связей между нейронами на основе обучающей выборки

Обучение сети

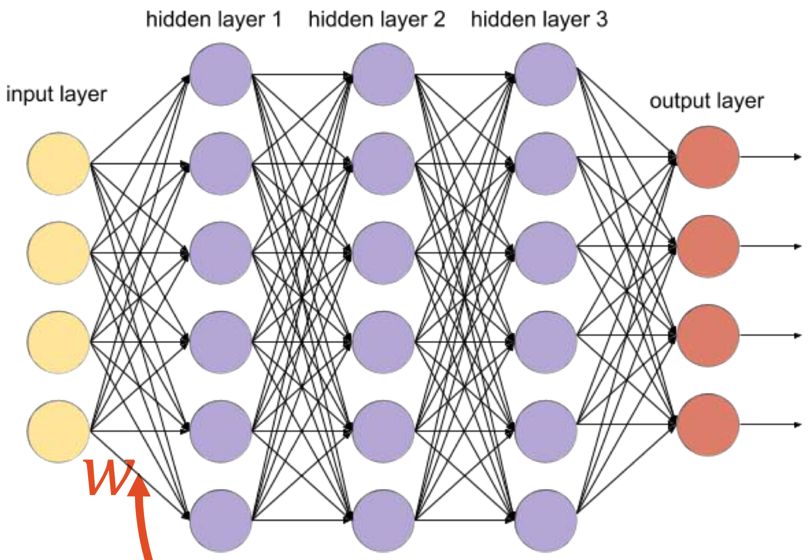


L

Задача обучения – настроить веса связей между нейронами на основе обучающей выборки

1. Выбираем функцию потерь

Обучение сети



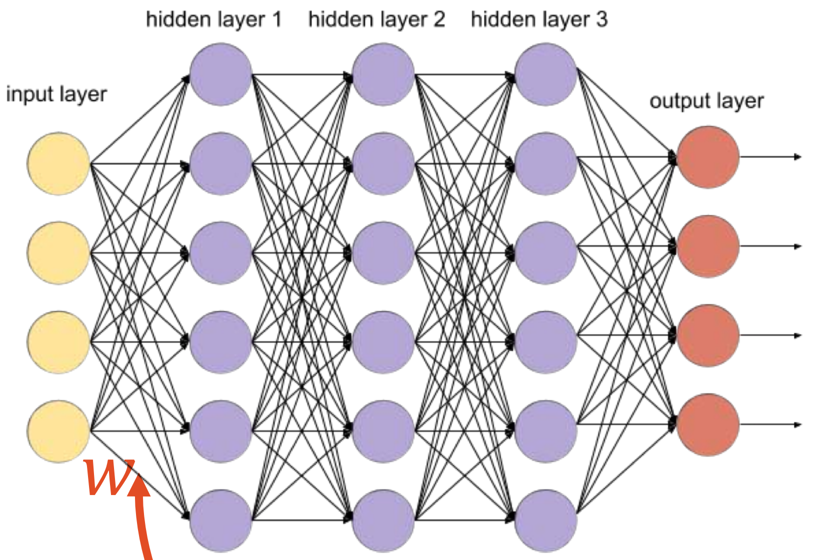
$$L$$

$$w^{(t+1)} = w^{(t)} - \gamma_t \frac{\partial L}{\partial w}$$

Задача обучения – настроить веса связей между нейронами на основе обучающей выборки

1. Выбираем функцию потерь
2. Обучаем веса с помощью SGD

Обучение сети



L

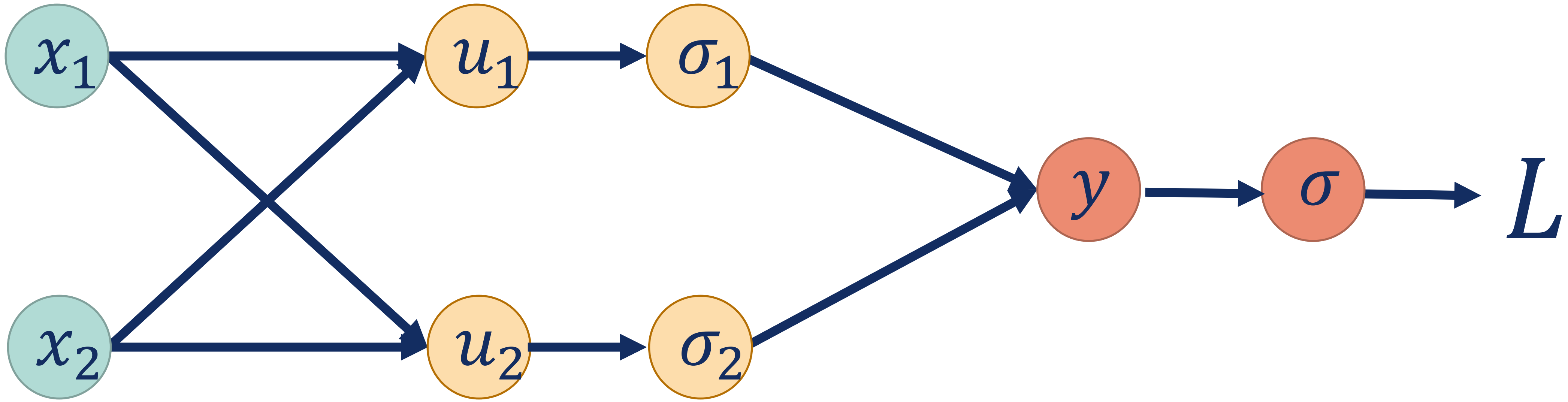
Задача обучения – настроить веса связей между нейронами на основе обучающей выборки

1. Выбираем функцию потерь
2. Обучаем веса с помощью SGD

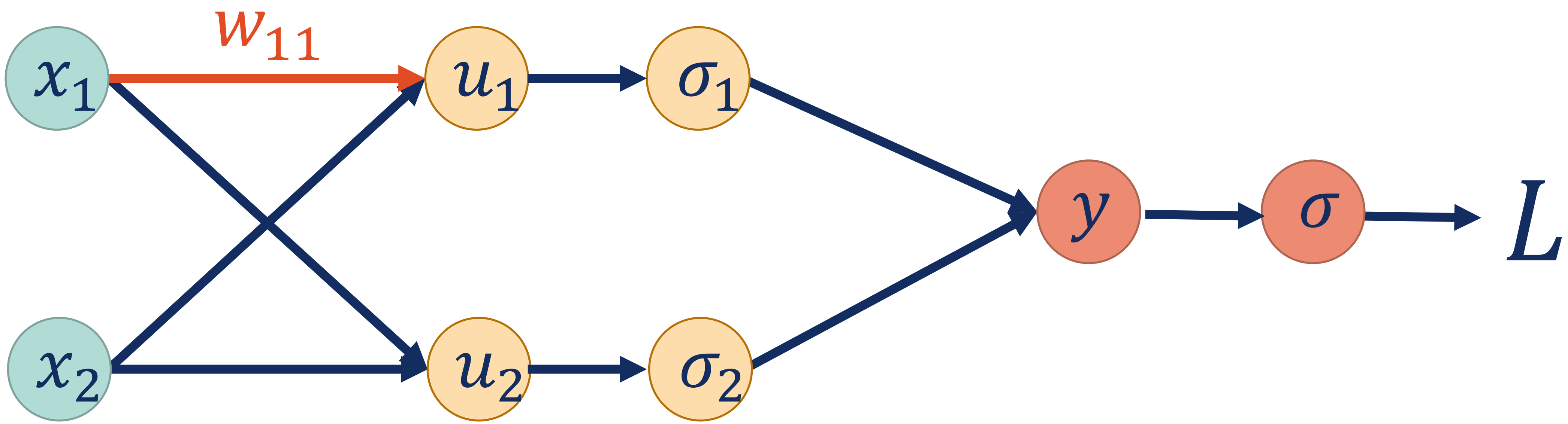
$$w^{(t+1)} = w^{(t)} - \gamma_t \boxed{\frac{\partial L}{\partial w}}$$

Проблема: не выписывать же нам все производные аналитически?!

Граф вычислений для нейросети

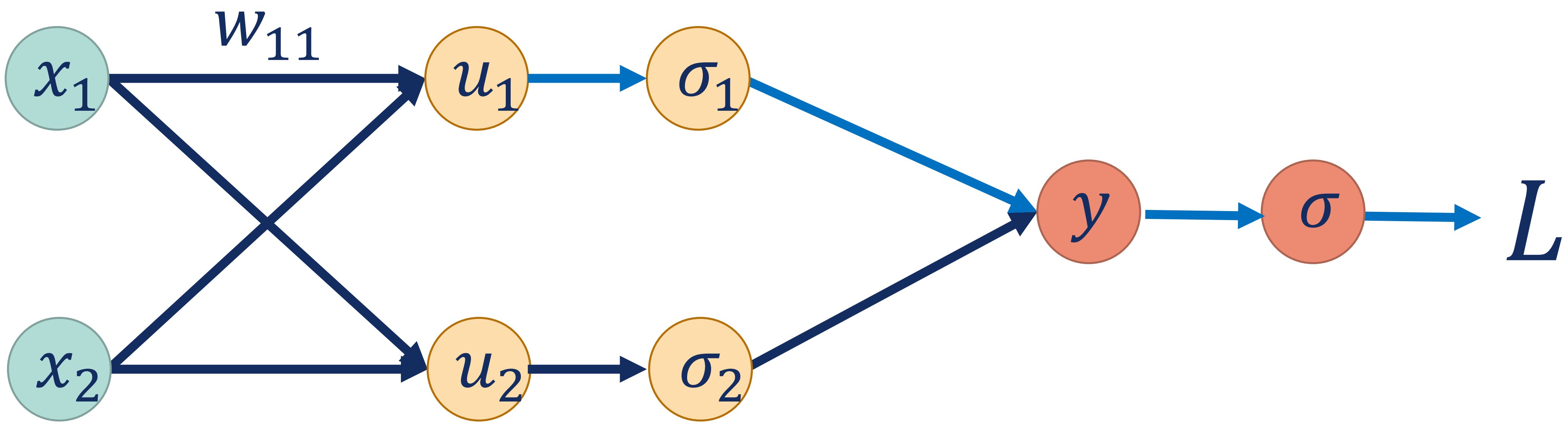


Граф вычислений для нейросети



$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

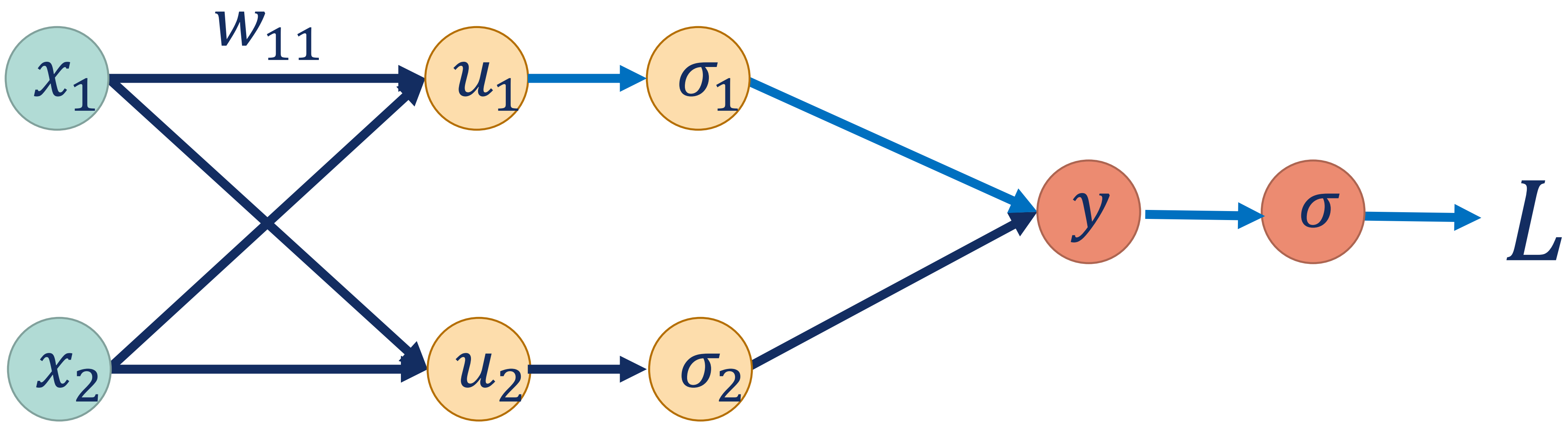
Граф вычислений для нейросети



$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

$$\boxed{\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial u_1} \frac{\partial u_1}{\partial w_{11}} = \frac{\partial L}{\partial u_1} x_1}$$

Граф вычислений для нейросети



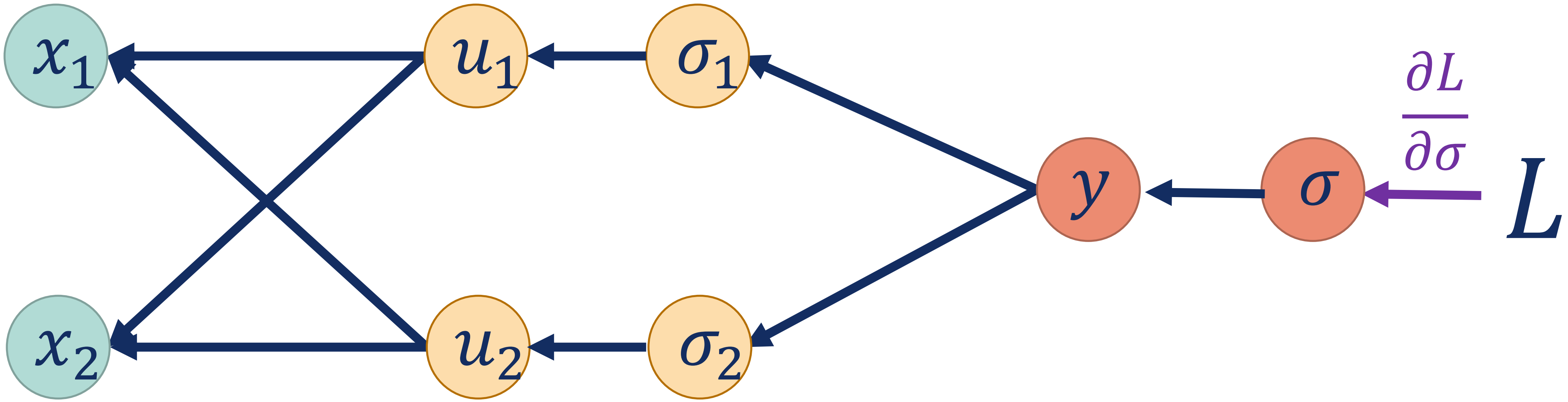
$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

$$\boxed{\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial u_1}\frac{\partial u_1}{\partial w_{11}} = \frac{\partial L}{\partial u_1} x_1}$$

Вывод: для обучения нужны производные L по выходам всех нейронов

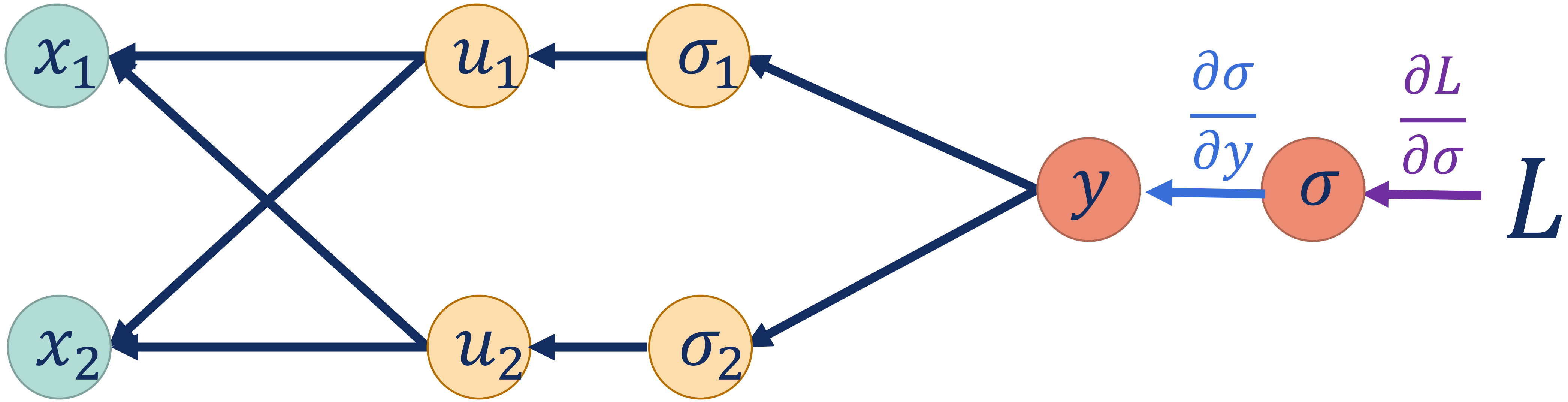
Backpropagation

Backprop – эффективный способ посчитать производные L по нейронам:



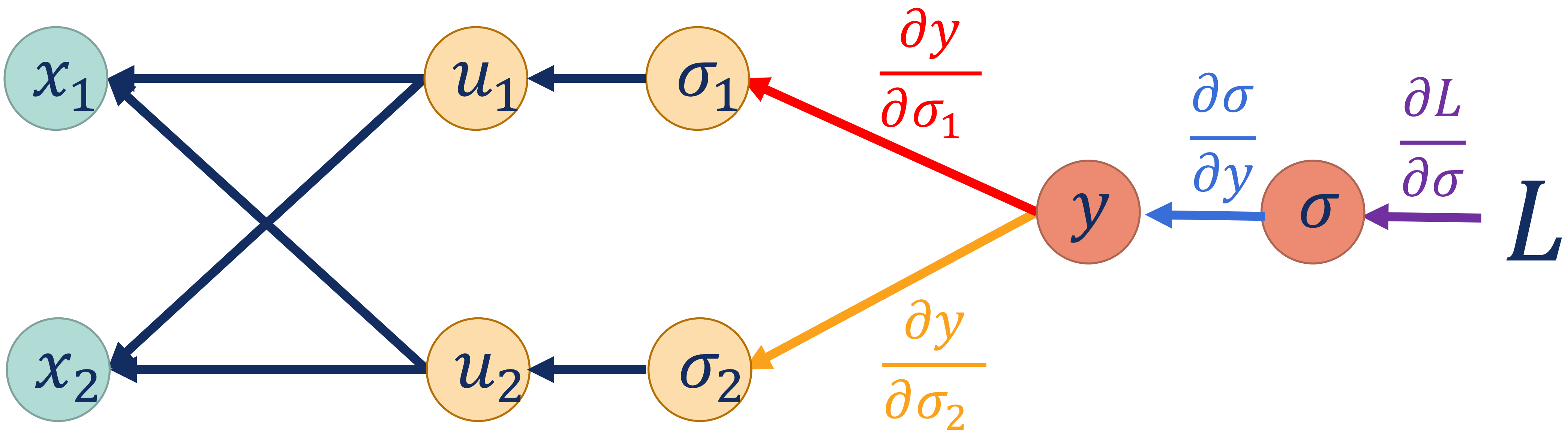
Backpropagation

Backprop – эффективный способ посчитать производные L по нейронам:

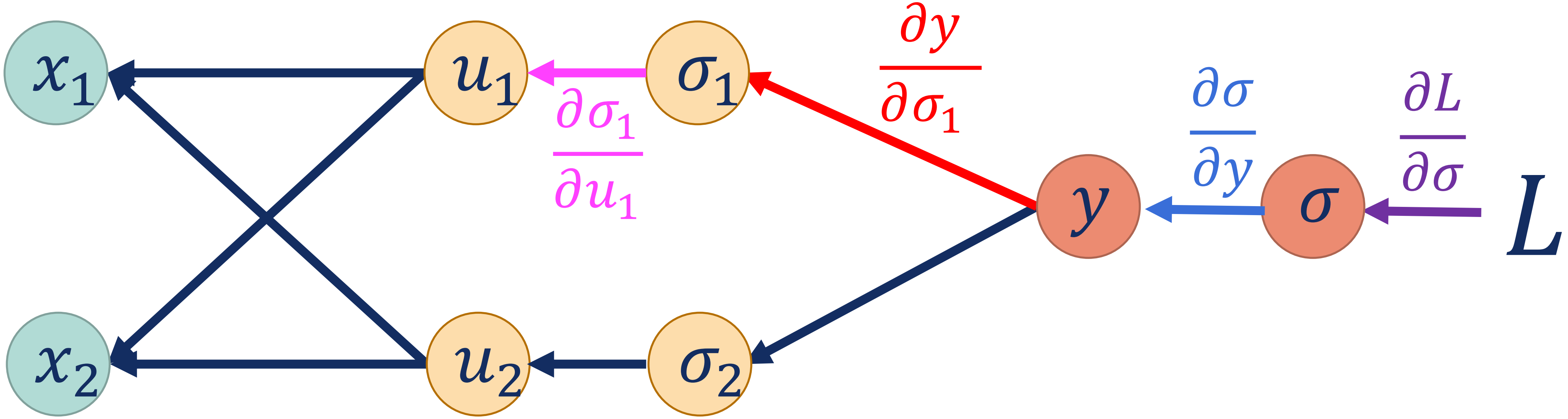


Backpropagation

Backprop – эффективный способ посчитать производные L по нейронам:

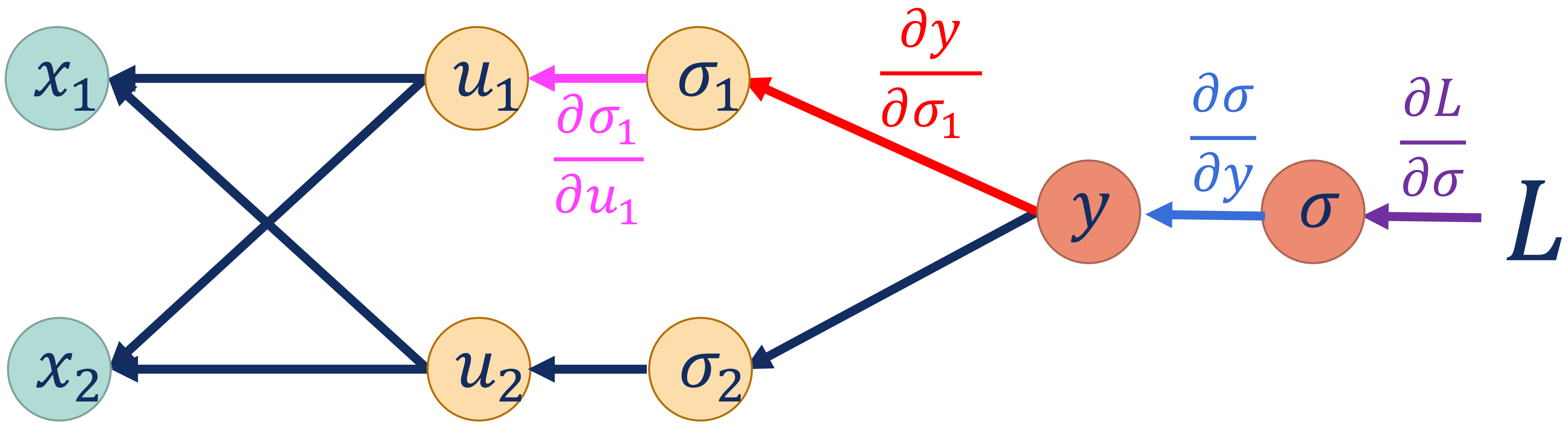


Как вычисляем производную



$$\frac{\partial L}{\partial u_1} = \frac{\partial L}{\partial \sigma} \frac{\partial \sigma}{\partial y} \frac{\partial y}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial u_1}$$

Как вычисляем производную

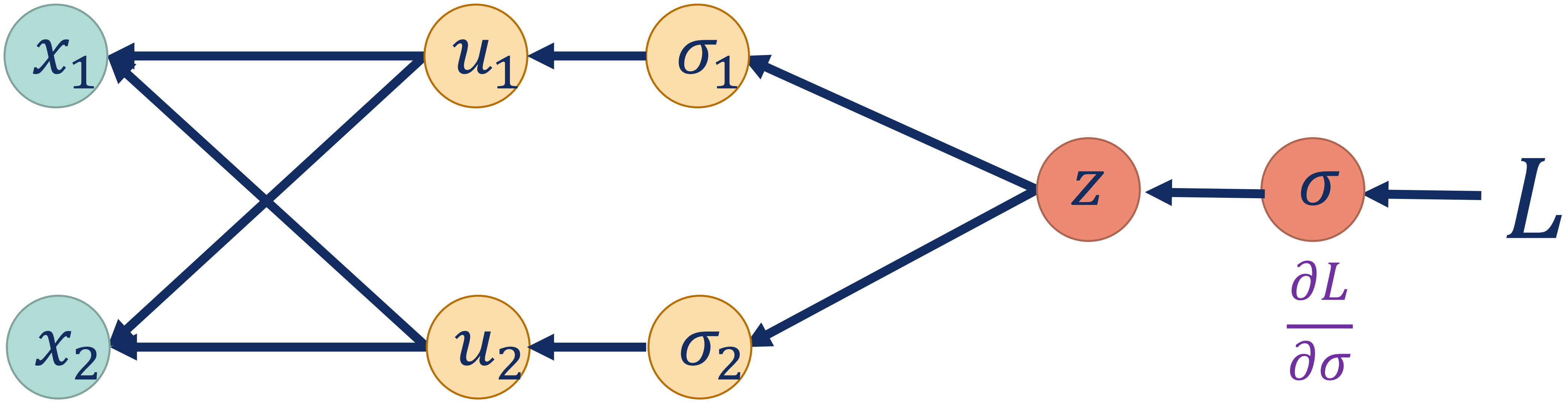


$$\frac{\partial L}{\partial u_1} = \frac{\partial L}{\partial \sigma} \frac{\partial \sigma}{\partial y} \frac{\partial y}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial u_1}$$

Смысл backprop – проходиться по графу с конца и записывать в вершинах графа эти произведения, а не пересчитывать произведение каждый раз заново

Другое название: Error Backpropagation

Раньше этот метод часто называли обратным распространением **ошибки**

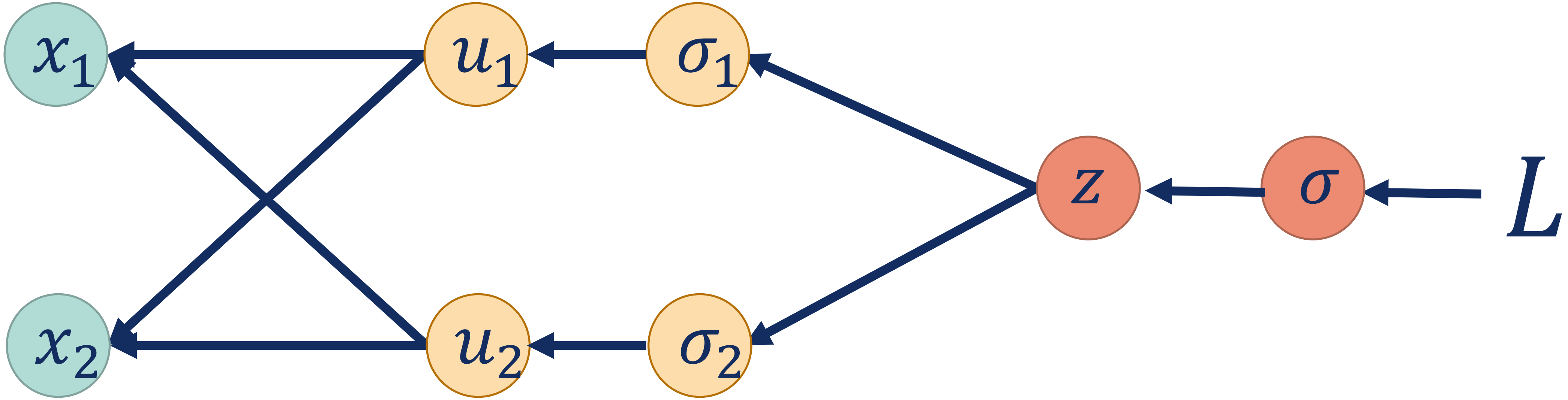


Если $L = \frac{1}{2}(\sigma - y_{true})^2$,

$\frac{\partial L}{\partial \sigma} = \sigma - y_{true}$ - ошибка прогноза

Другое название: Error Backpropagation

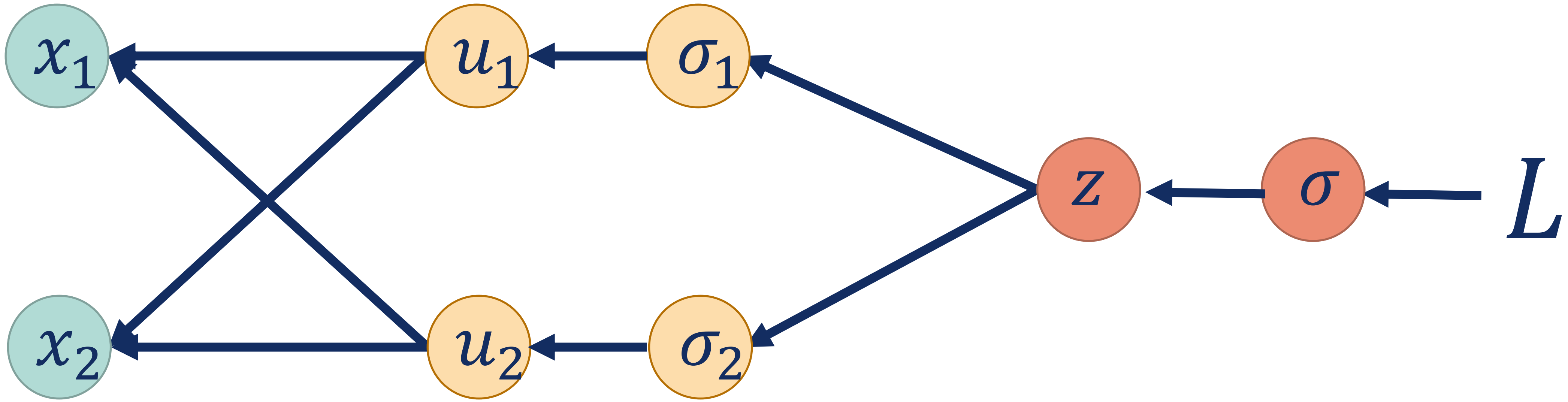
Раньше этот метод часто называли обратным распространением **ошибки**



При градиентном спуске мы бы «подправляли» u_2 на величину $\frac{\partial L}{\partial u_2}$ - значит эта производная – аналог «ошибки» в этом нейроне

Другое название: Error Backpropagation

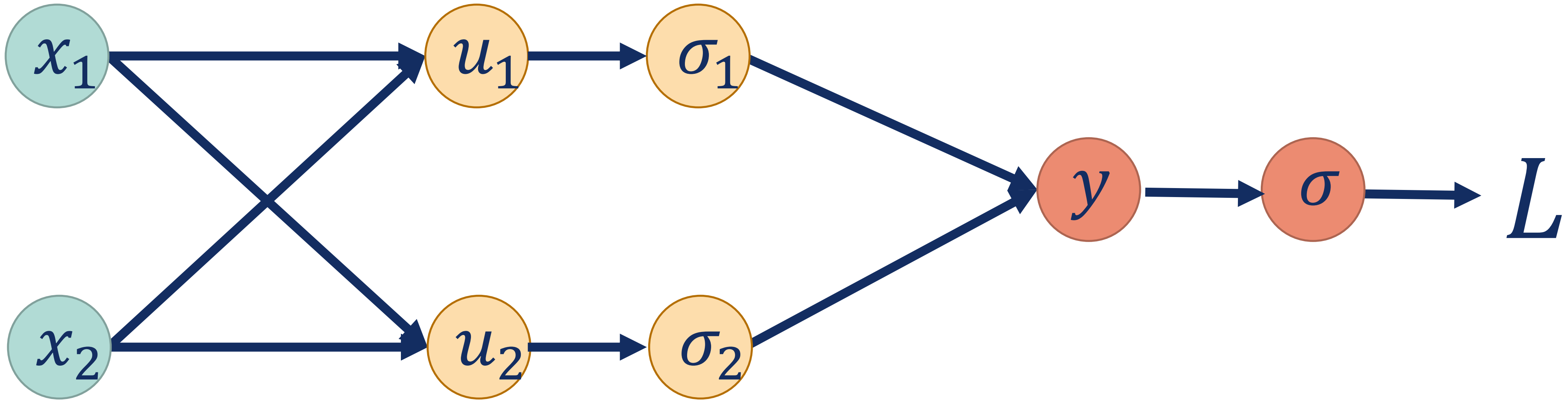
Раньше этот метод часто называли обратным распространением **ошибки**



Получается, мы вычисляем ошибку на выходе и «распространяем» ее в обратном направлении (ко входу), вычисляя «ошибки» во всех нейронах по пути

Backprop: как делать

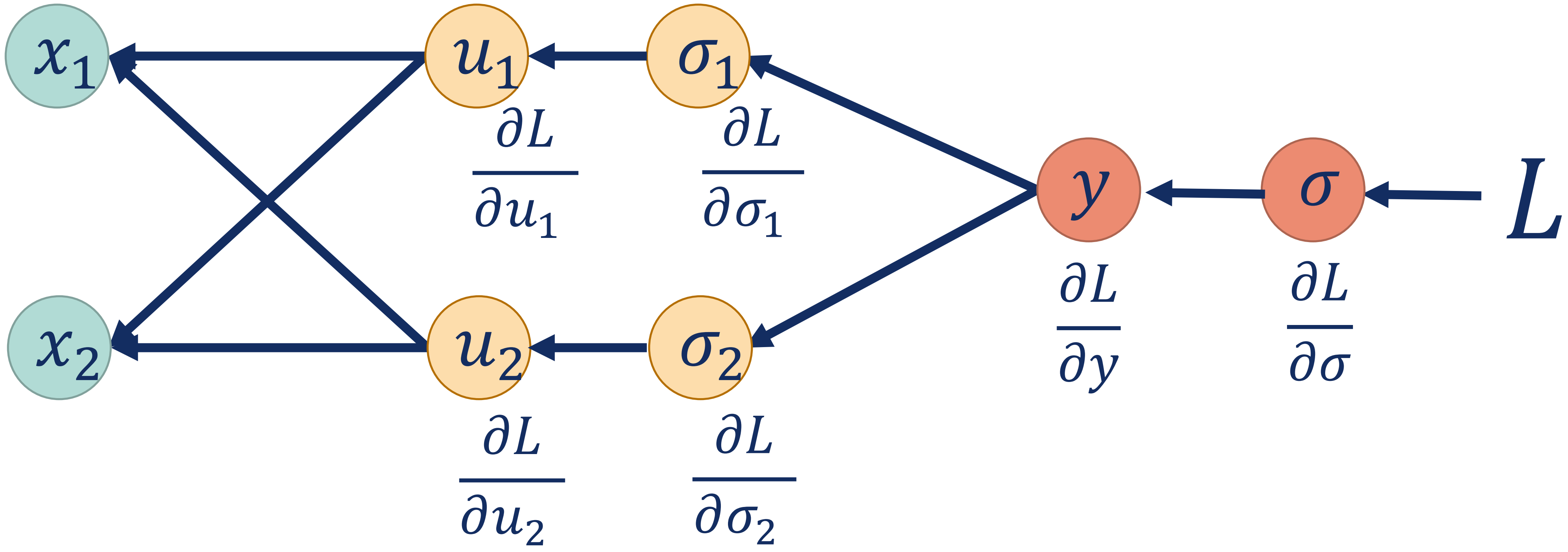
Чередуем **forward pass** (вычисление значений в нейронах)



Этот шаг нам нужен, чтобы знать, в каких точках считать производные

Backprop: как делать

И **backward pass** (вычисление производных):



Как это реализовано в библиотеках

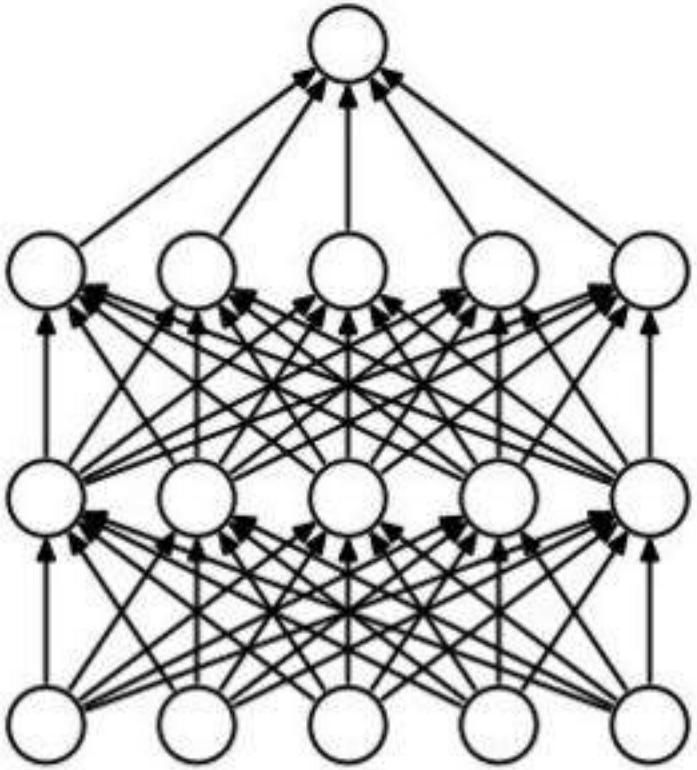
1. Для каждого типа слоя написан forward pass и backward pass
2. Операции оптимизированы за счет матричной записи и алгоритмов быстрых матричных вычислений (см. BLAS)

5. Проблемы backprop

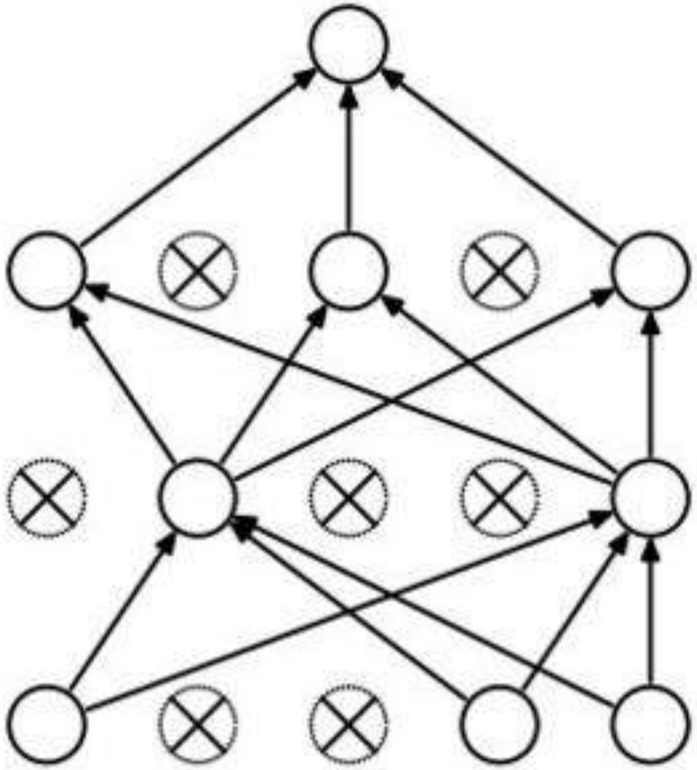
Проблемы backprop

1. Все проблемы SGD, в частности – застревание в острых локальных минимумах и легкое переобучение

Пример регуляризации: dropout



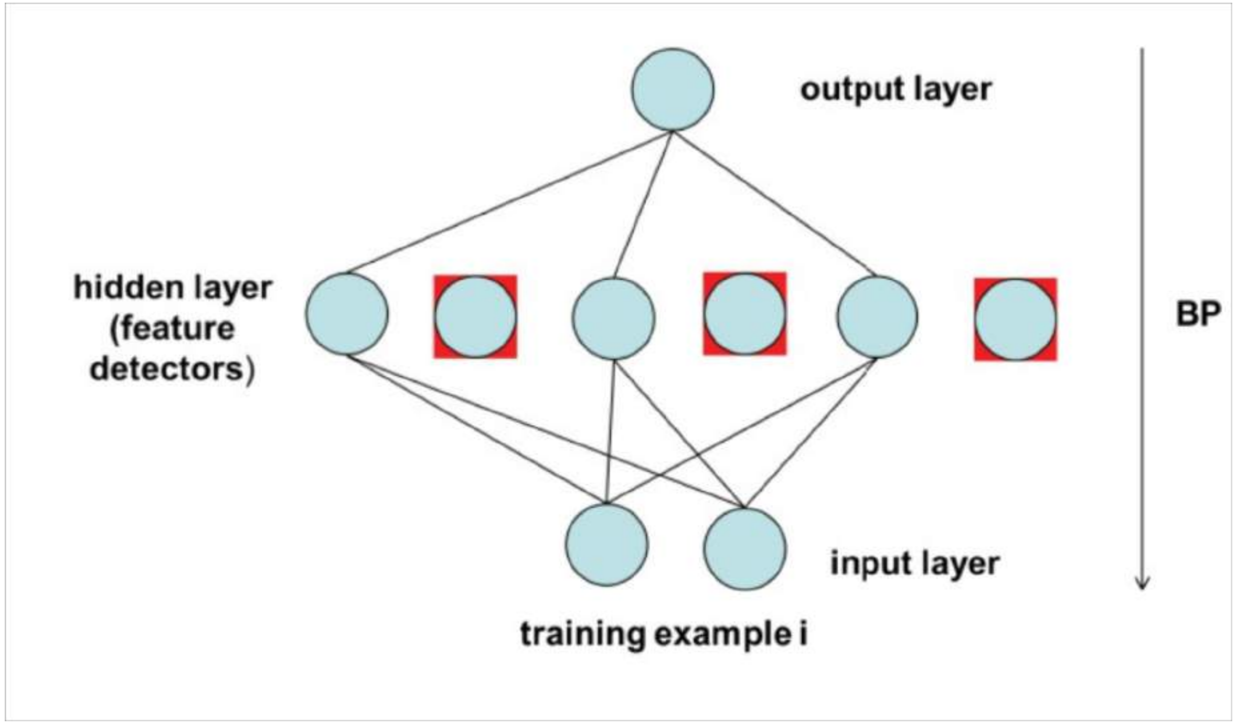
(a) Standard Neural Net



(b) After applying dropout.

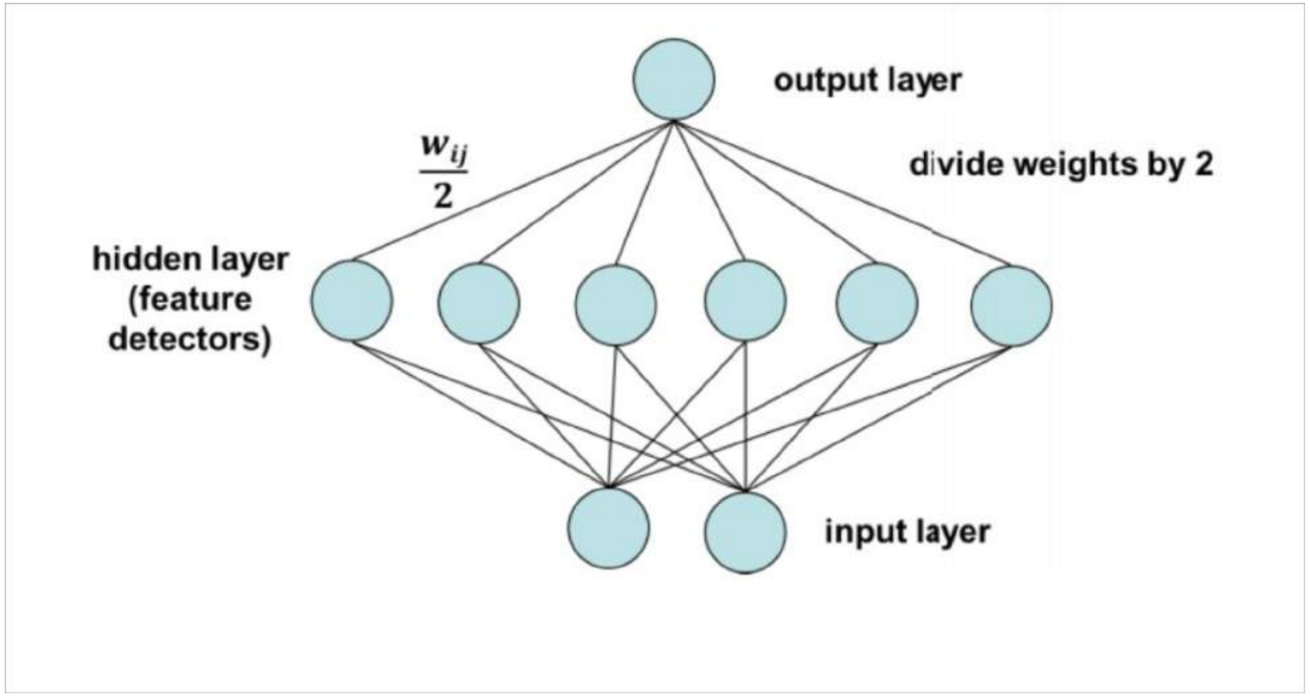
Dropout: обучение

С вероятностью p зануляем выход каждого нейрона на слое

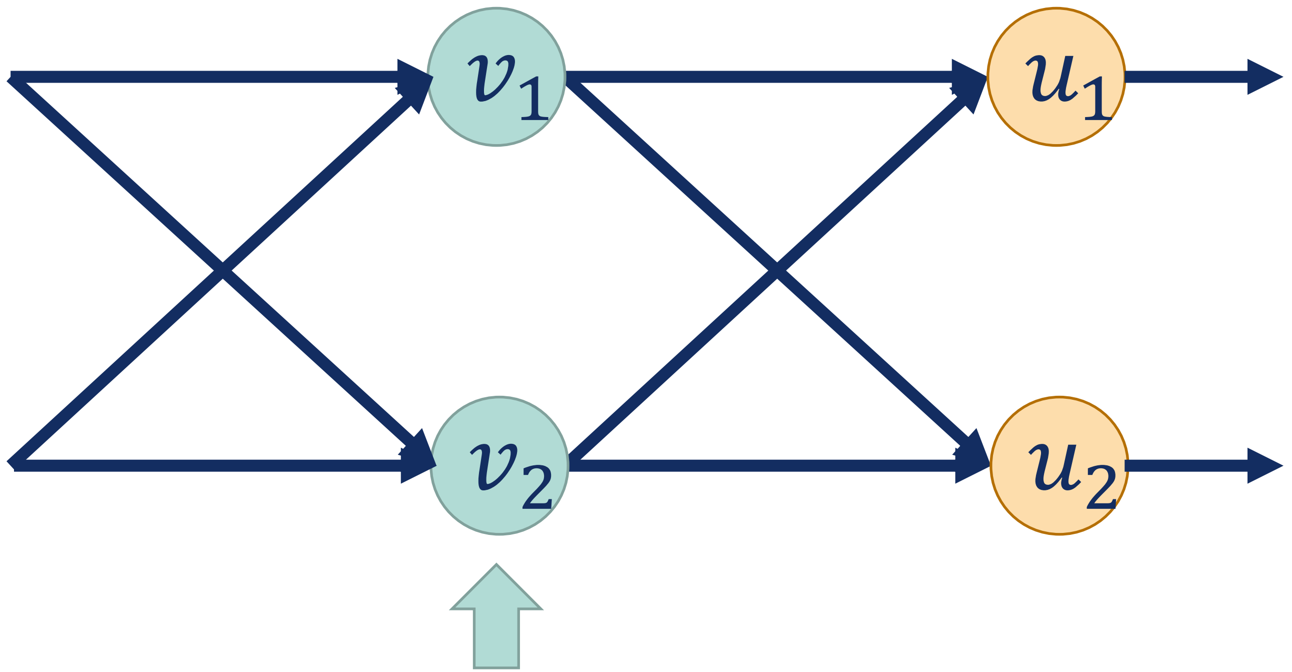


Dropout: применение

Домножаем выход каждого нейрона на (1-p)



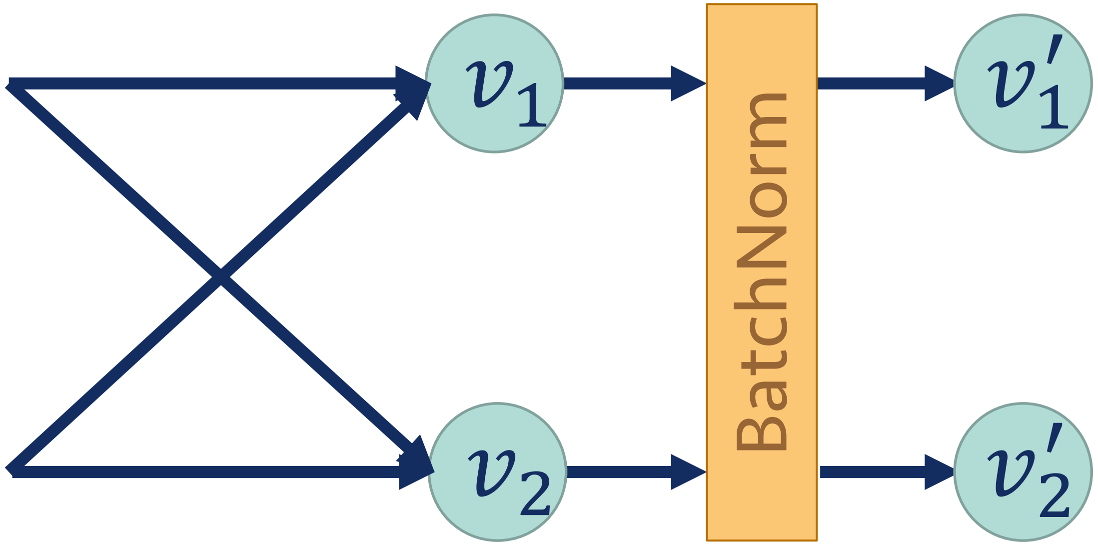
Еще один пример борьбы с переобучением



На прошлой итерации здесь могло быть другое распределение,
т.к. веса на предыдущих слоях тоже поменялись.

Это различие распределений называется *internal covariate shift* (ICS)

Еще один пример борьбы с переобучением



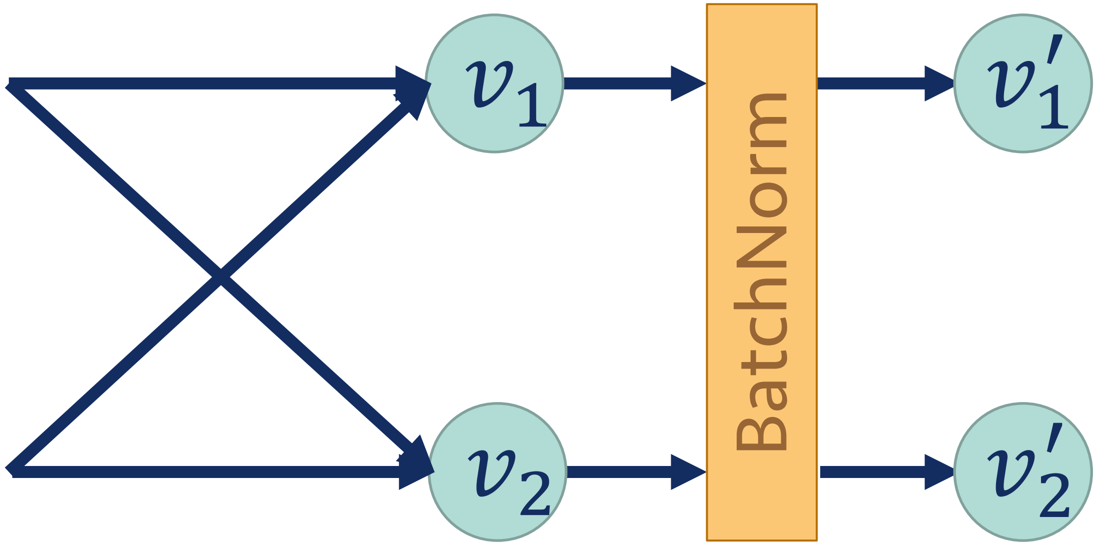
Batch Normalization:
Будем добавлять слои, в которых каждый признак будет нормироваться на среднее и дисперсию по батчу

$v_1^{(1)}, \dots, v_1^{(m)}$ – значения признака v_1 по батчу размера m

Среднее по батчу: $\mu = \frac{1}{m}\sum_{i=1}^{m} v_1^{(i)}$

Дисперсия по батчу: $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}\left(v_1^{(i)} - \mu\right)^2$

Еще один пример борьбы с переобучением



Batch Normalization:
Будем добавлять слои, в которых каждый признак будет нормироваться на среднее и дисперсию по батчу

Нормировка: $\tilde{v}_1 = \frac{v_1 - \mu}{\sqrt{\sigma^2 + \varepsilon}}$

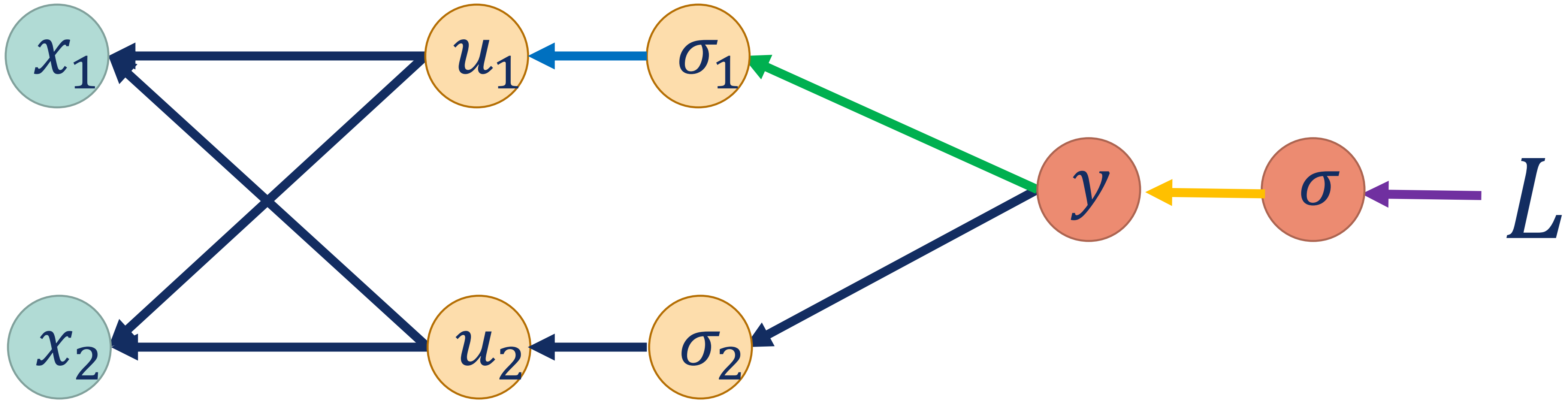
Масштабирование и сдвиг: $v'_1 = \gamma \tilde{v}_1 + \beta$

Проблемы backprop

1. Все проблемы SGD, в частности – застревание в локальных минимумах и легкое переобучение
2. Взрыв и затухание градиента (но на самом деле – это не совсем проблема backprop)

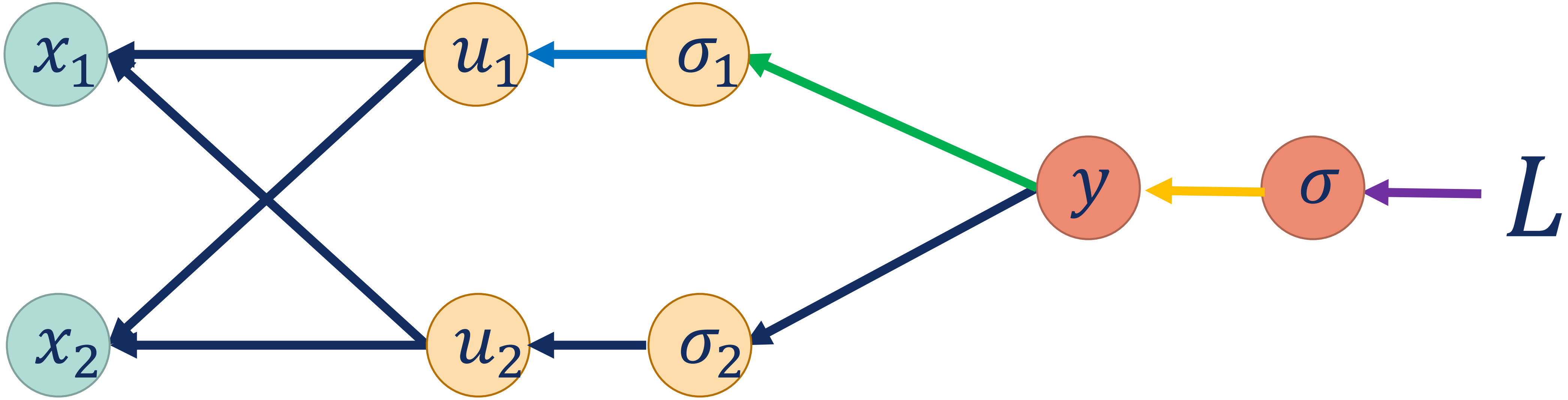
Затухание градиента (Gradient vanishing)

Производная по нейрону по chain rule получается из произведения производных по пути к нему от выхода



Затухание градиента (Gradient vanishing)

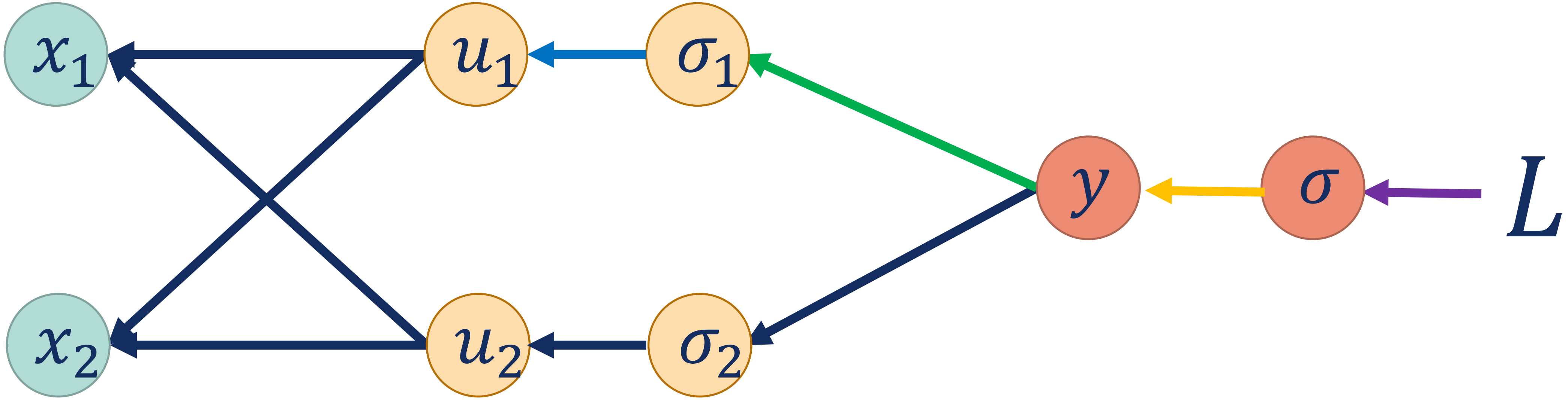
Производная по нейрону по chain rule получается из произведения производных по пути к нему от выхода



Если каждая из производных небольшая по модулю – произведение тоже будет маленьким. Чем больше слоев, тем меньше.

Затухание градиента (Gradient vanishing)

Производная по нейрону по chain rule получается из произведения производных по пути к нему от выхода

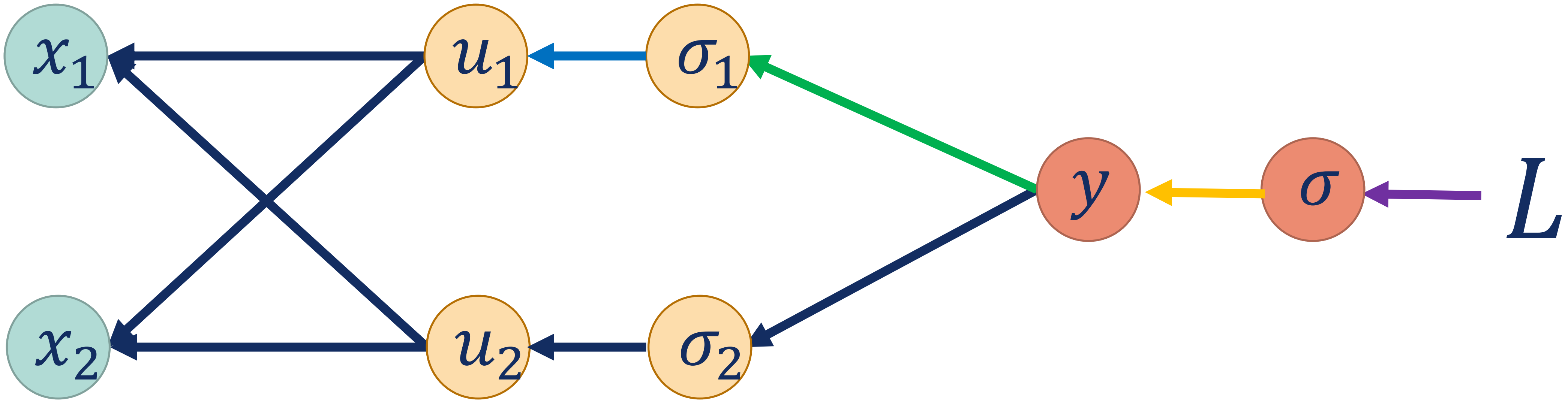


Если каждая из производных небольшая по модулю – произведение тоже будет маленьким. Чем больше слоев, тем меньше.

Значит в «глубине» веса не будут меняться!

Взрыв градиентов (Gradient explosion)

Аналогично для больших по модулю производных:



Модуль произведения производных растет экспоненциально с числом слоев. Чем больше слоев, тем больше будут градиенты.

В «глубине» веса будет кидать из стороны в сторону с огромным шагом.

Deep learning: что изменилось?

Нейросети и backpropagation известны давно, почему же последние достижения происходят только сейчас?

Два фактора:

1. Выросли вычислительные мощности и развились вычисления на GPU
2. Тем временем люди придумали много полезных трюков и архитектур

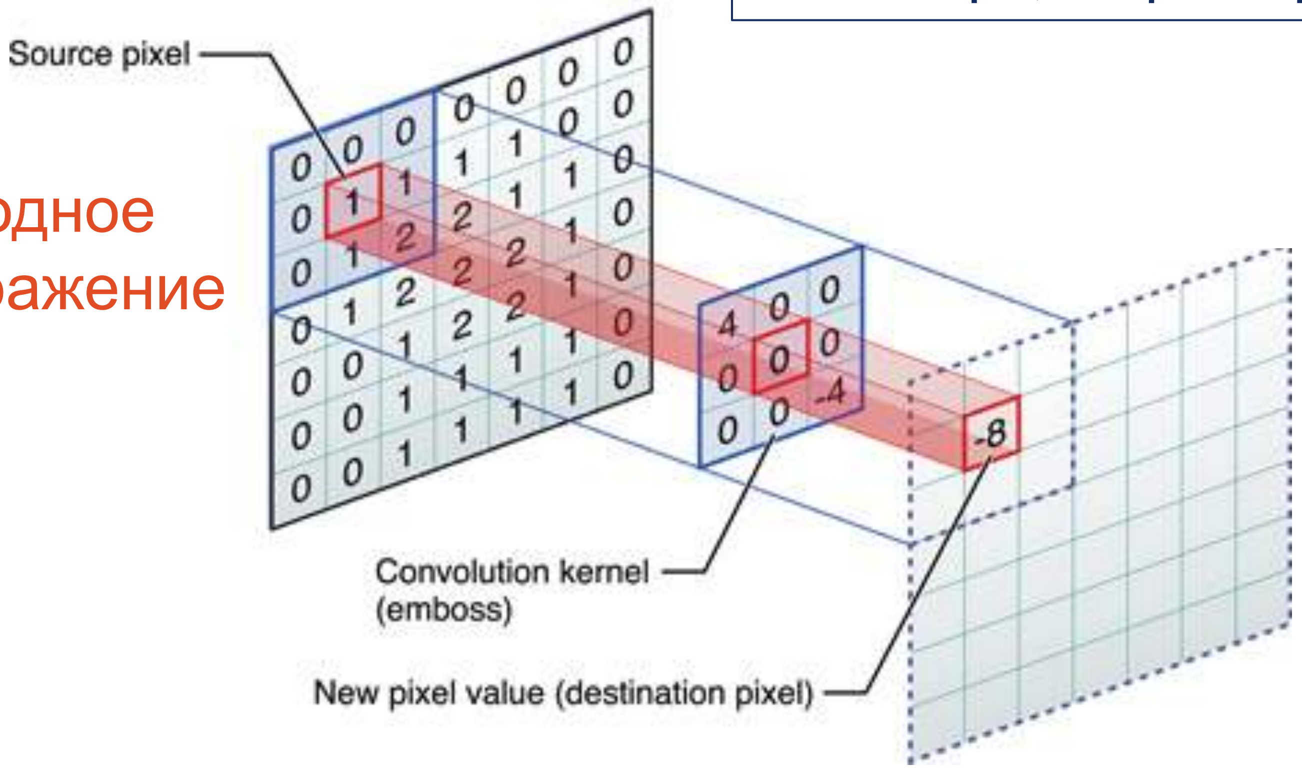
6. Слои

Convolutional

1. Т.к. фильтр – это настраиваемые backprop веса, то **сеть сама «подберет» фильтры**

2. Но т.к. фильтр «замечает» только один паттерн, то фильтров нужно **больше**

Входное изображение



Нейроны слоя

Pooling

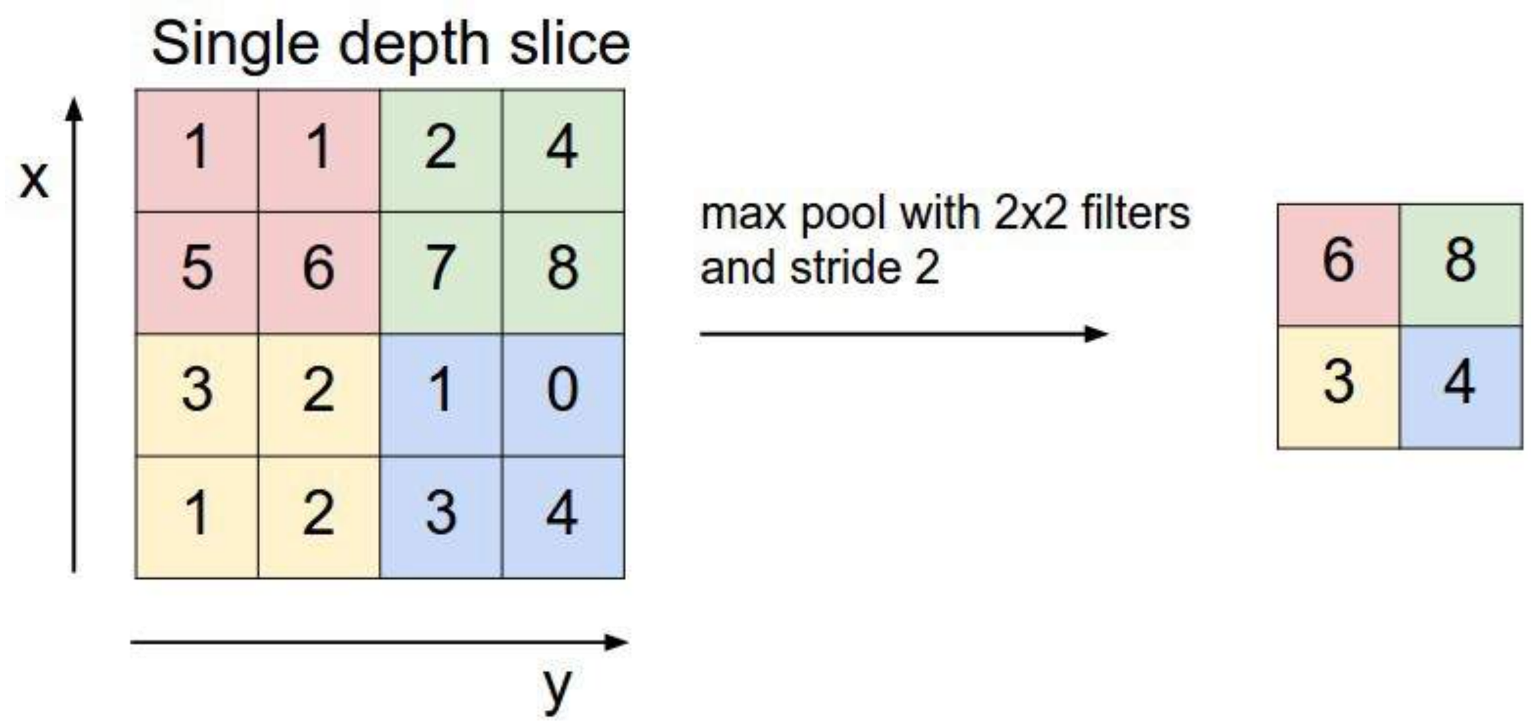
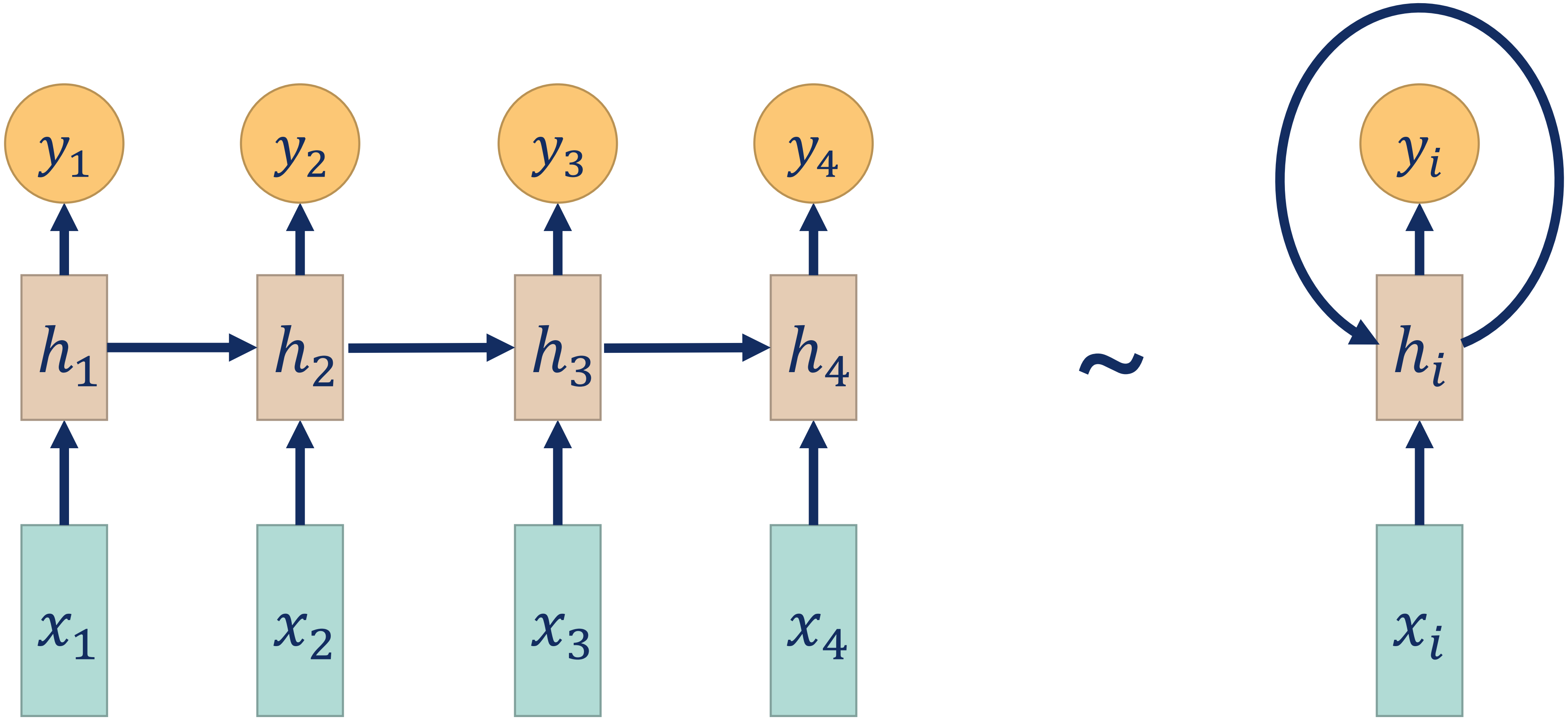
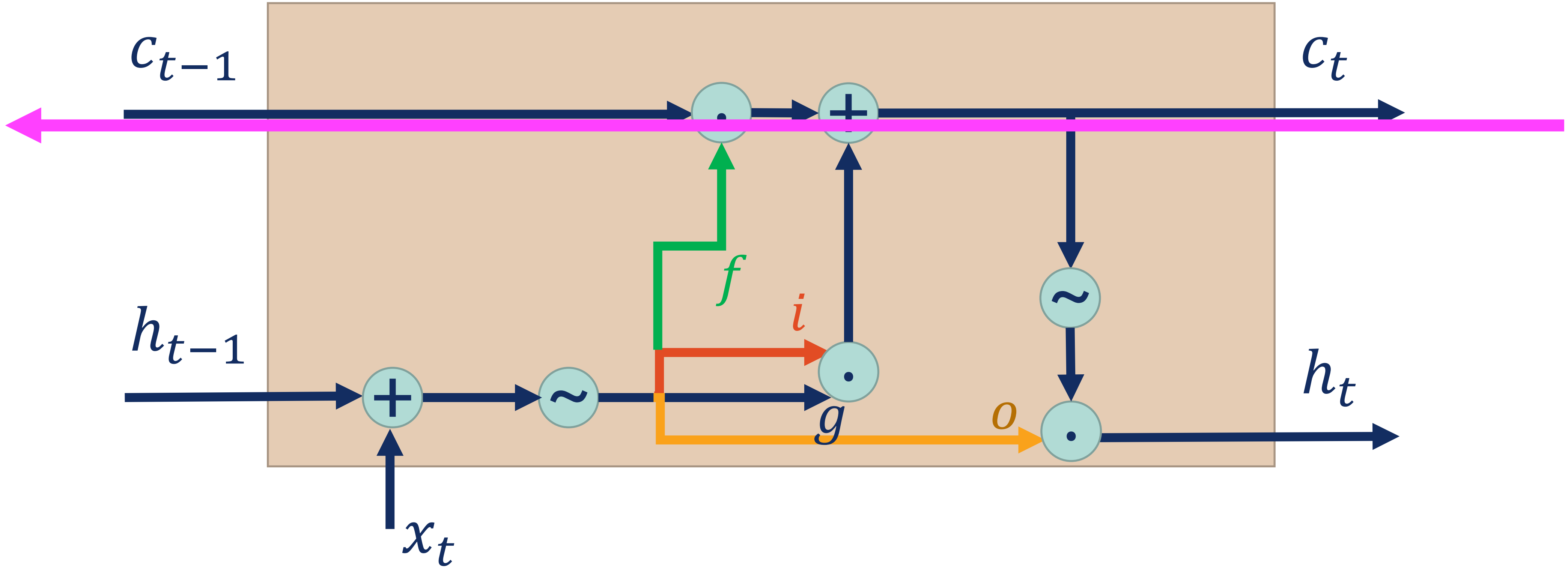

Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Схема RNN



Вчера телефон перестал работать

Схема LSTM

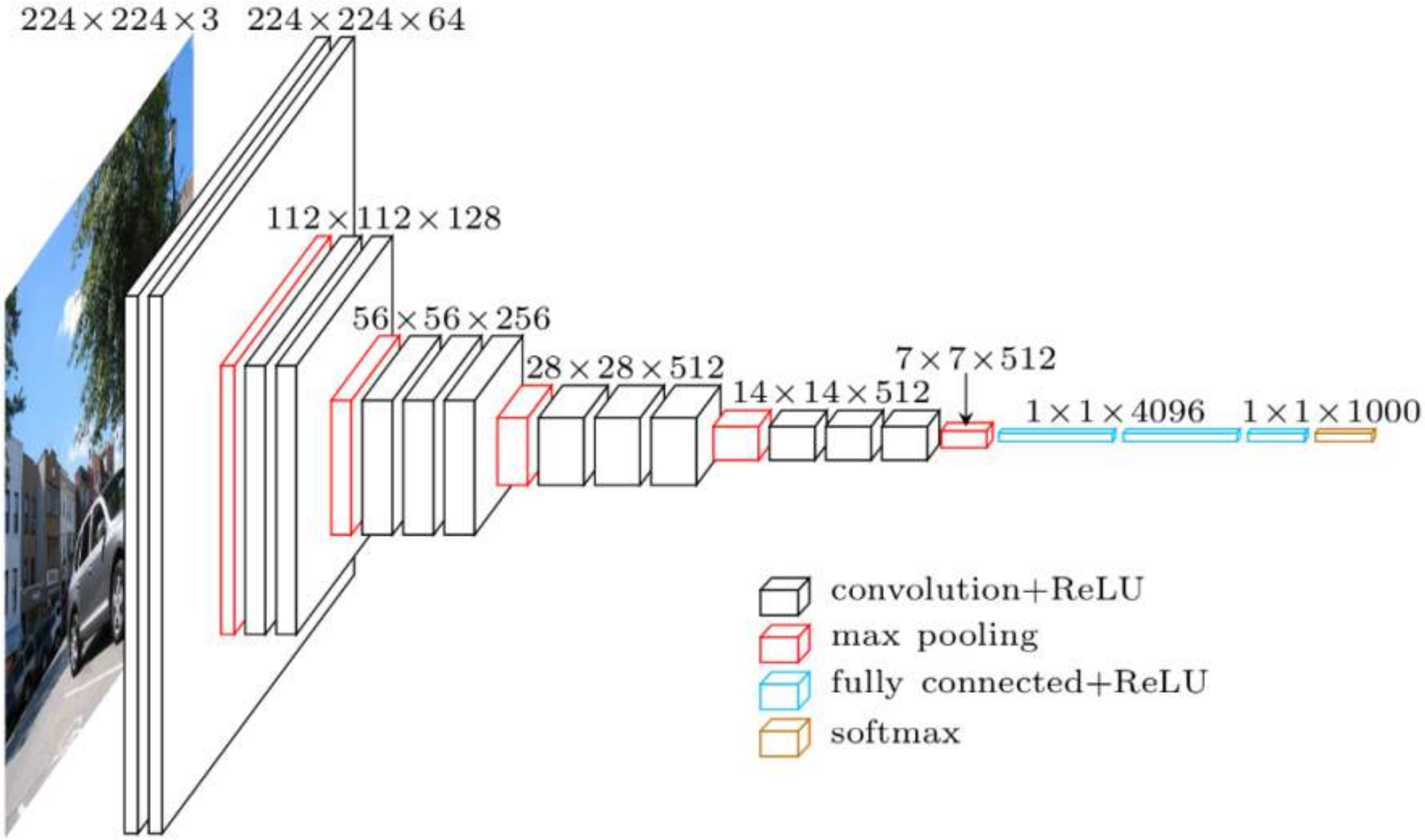


$$\begin{pmatrix} g_t \\ i_t \\ o_t \\ f_t \end{pmatrix} = \begin{pmatrix} \varphi \\ \sigma \\ \sigma \\ \sigma \end{pmatrix} (W_x x_t + W_h h_{t-1} + b)$$

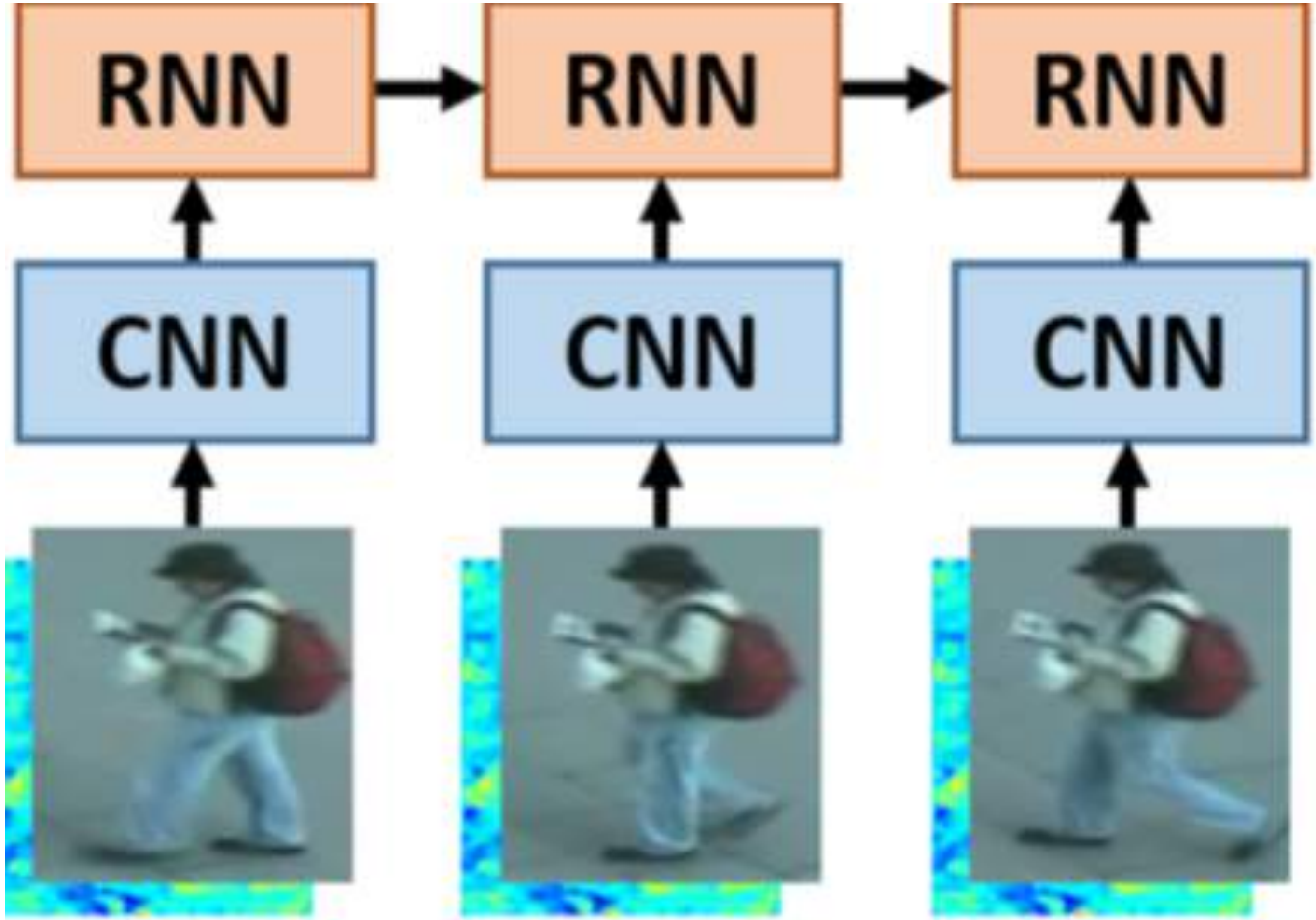
$$c_t = f_t \cdot c_{t-1} + i_t \cdot g_t$$
$$h_t = o_t \cdot \phi(c_t)$$

7. Примеры задач

Conv – Pool – Repeat x n – Dense



Рекуррентный слой после сверточного



Распознавание речи

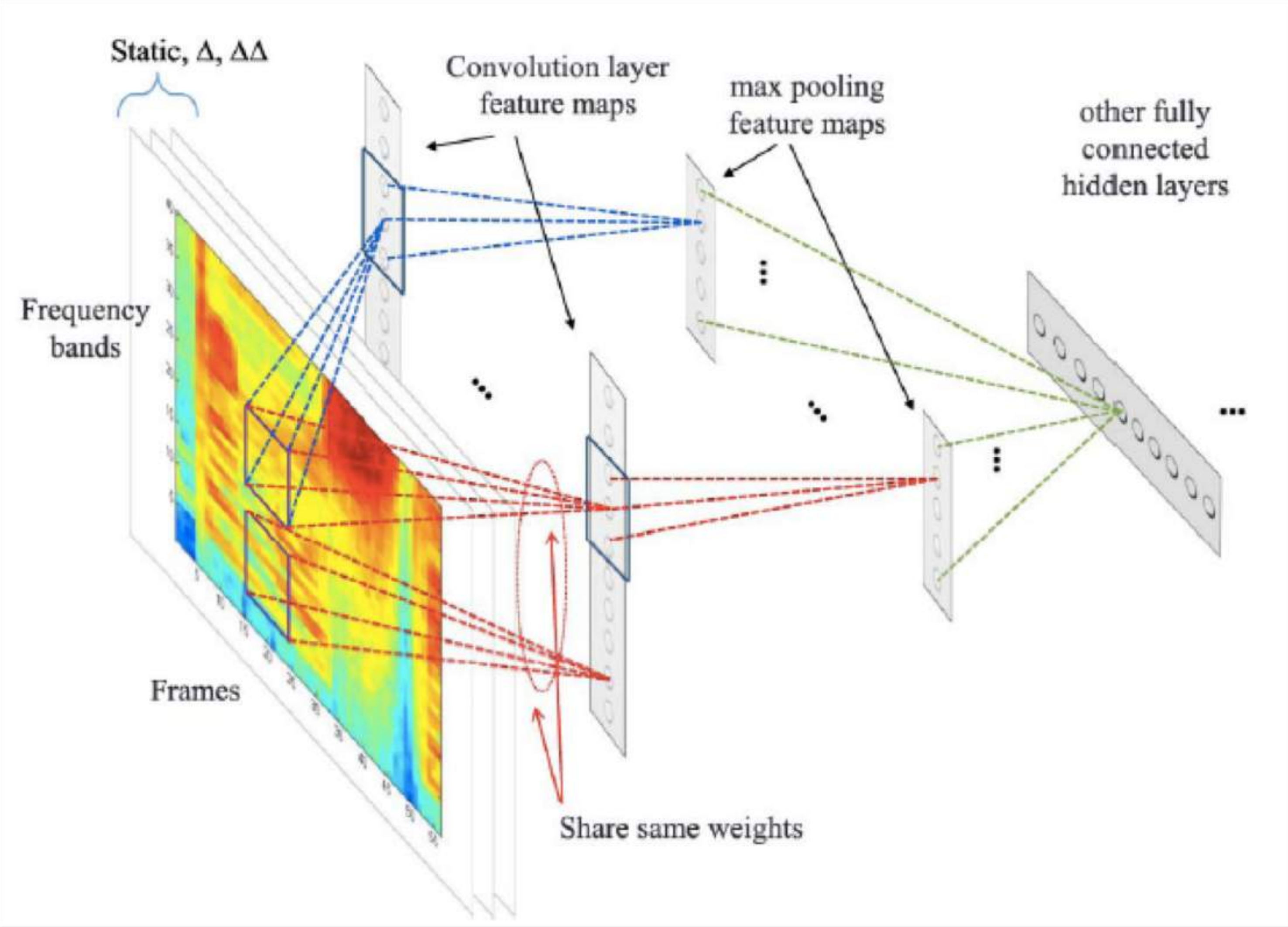
Сверточные или рекуррентные сети?

Распознавание речи

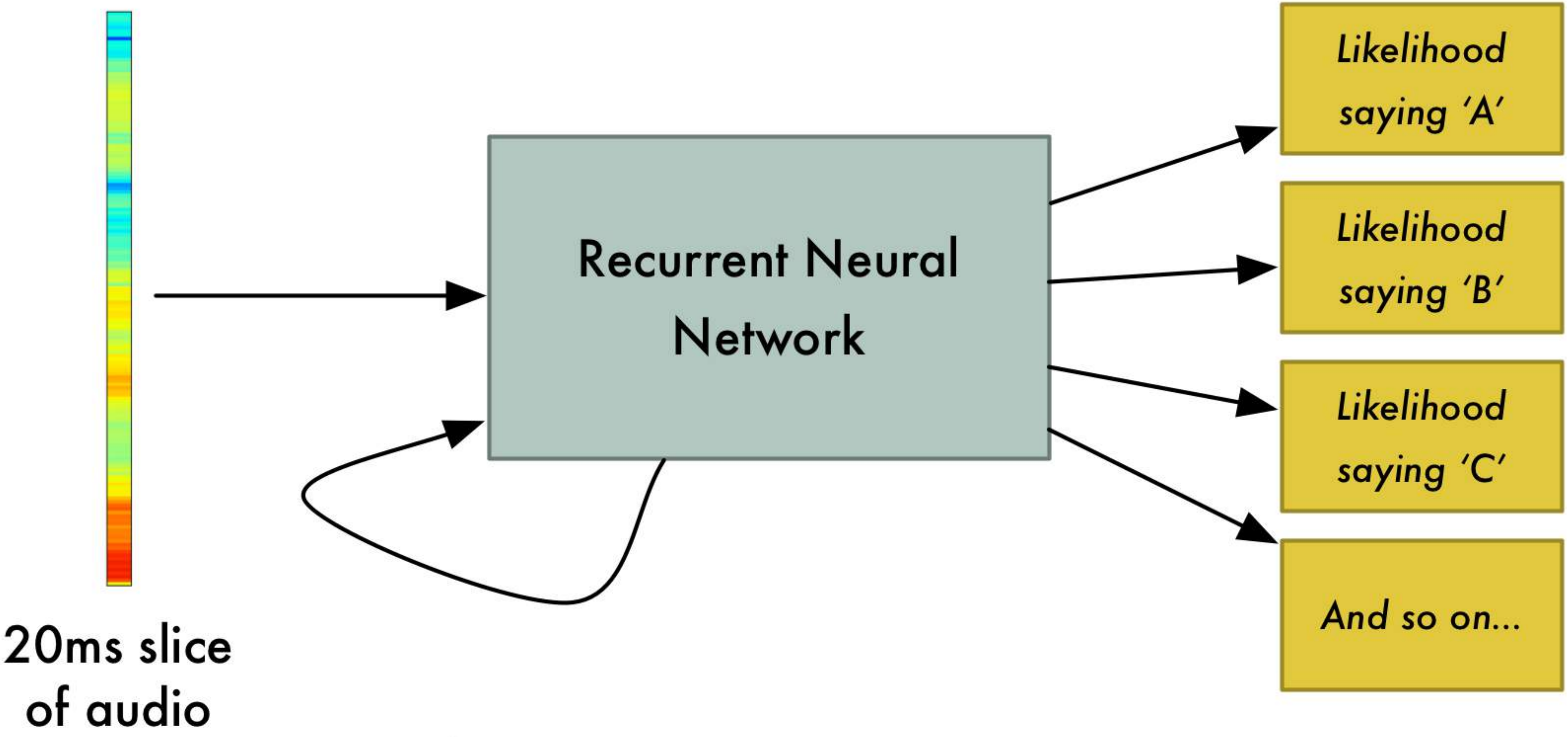
Сверточные или рекуррентные сети?

И то и другое!

Распознавание речи: сверточные сети



Распознавание речи: рекуррентные сети



Классификация текста

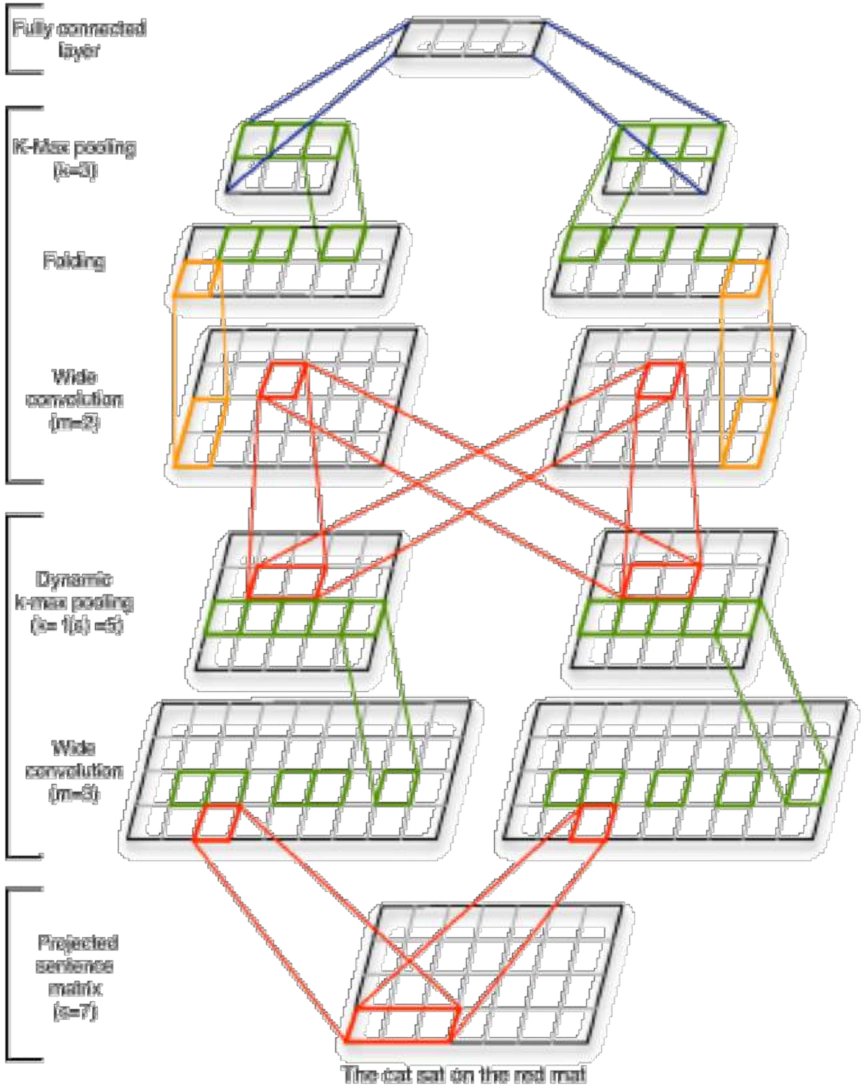
Сверточные или рекуррентные сети?

Классификация текста

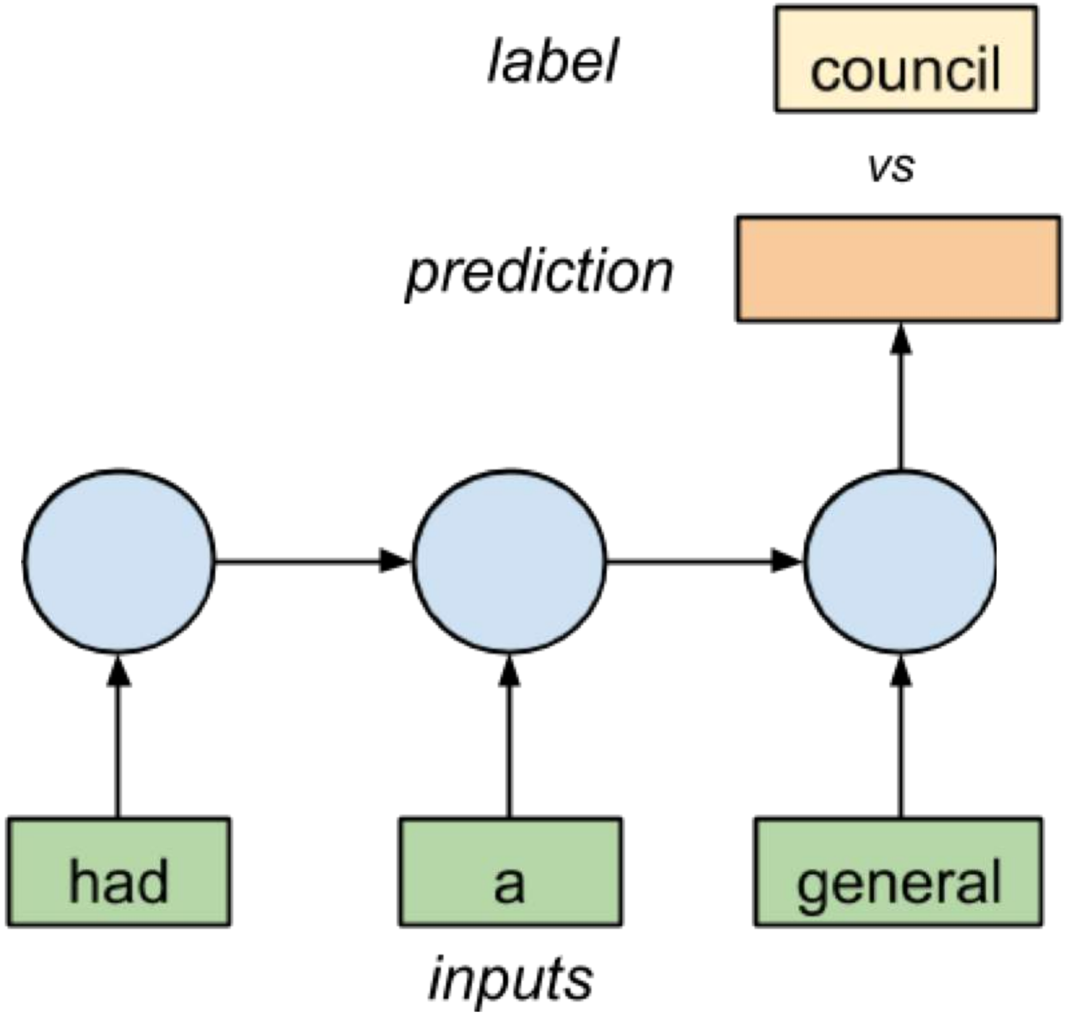
Сверточные или рекуррентные сети?

И то и другое!

Классификация текстов: сверточные сети



Классификация текстов: рекуррентные сети



Классификация изображений

Сверточные или рекуррентные?

Классификация изображений

Сверточные или рекуррентные?

В основном, сверточные

Классификация изображений: сверточные сети