Data Mining in Action

Лекция 9. Нейронные сети

Образовательный партнер курса

misis.ru

Генеральный партнер курса

jet.su

Партнеры курса

raiffeisen-digital.ru

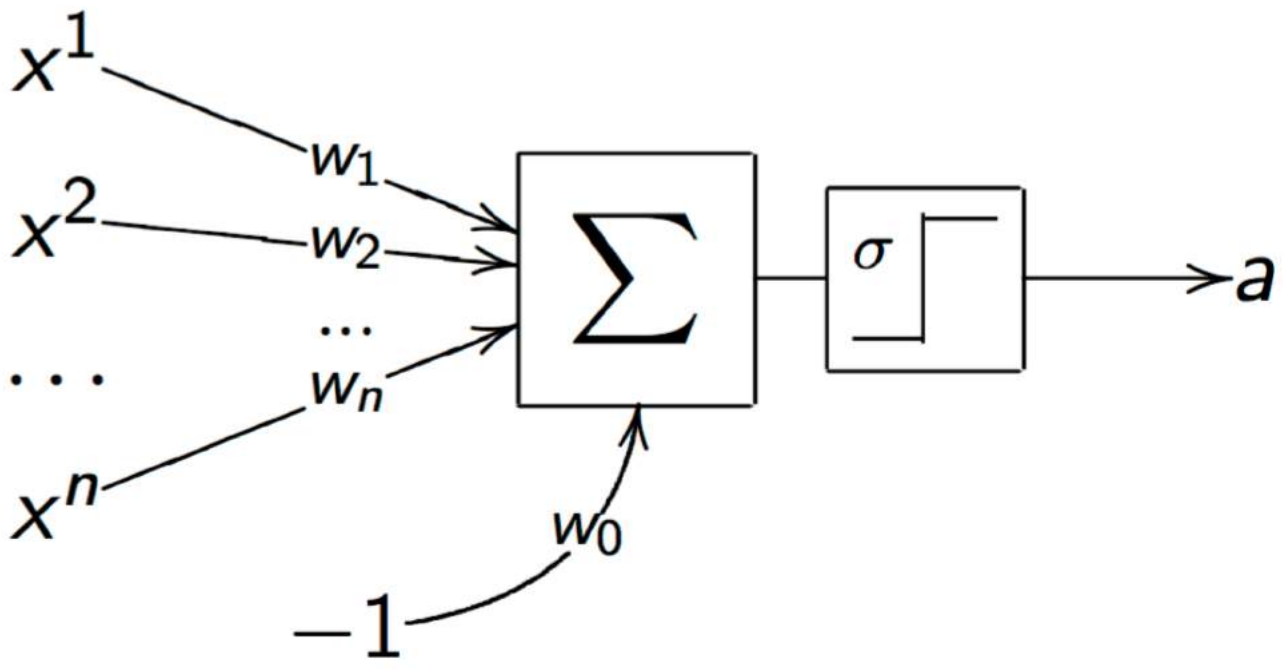
academy.yandex.ru

План

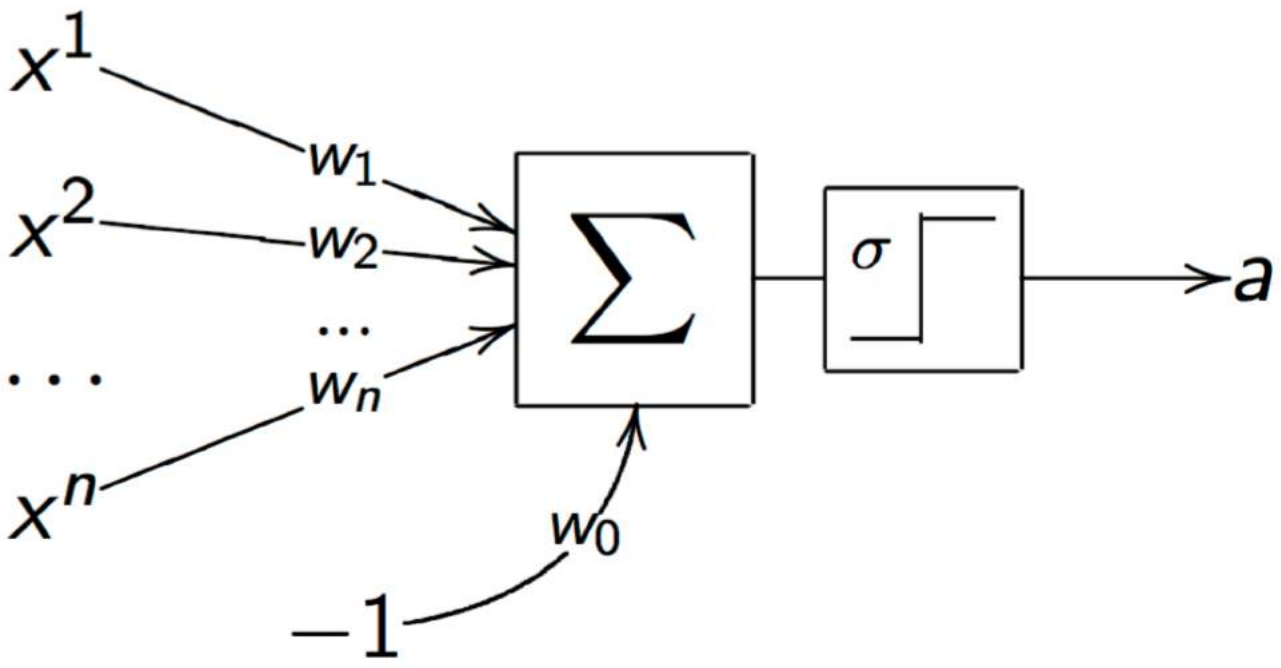
1. Что такое нейронные сети
2. Обучение: backpropagation
3. Регуляризация и другие трюки
4. Методы оптимизации

1. Что такое нейронные сети

Нейрон в машинном обучении

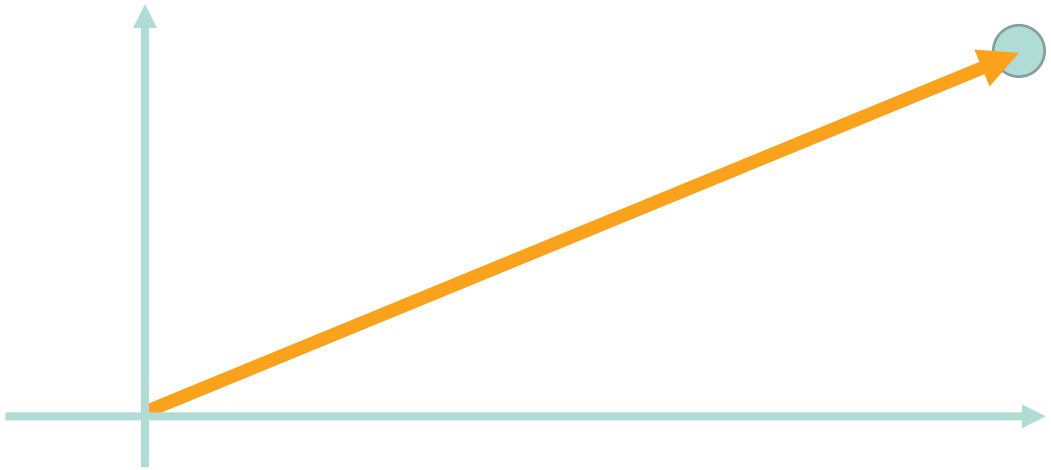


Модель нейрона



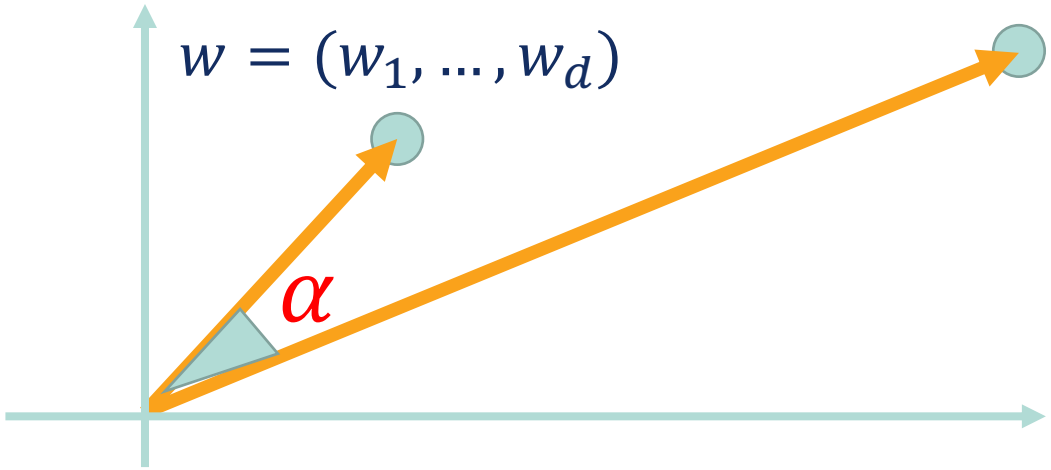
$$a(x) = \sigma(w^T x + b)$$

Векторы



$x = (x_1, \ldots, x_d)$

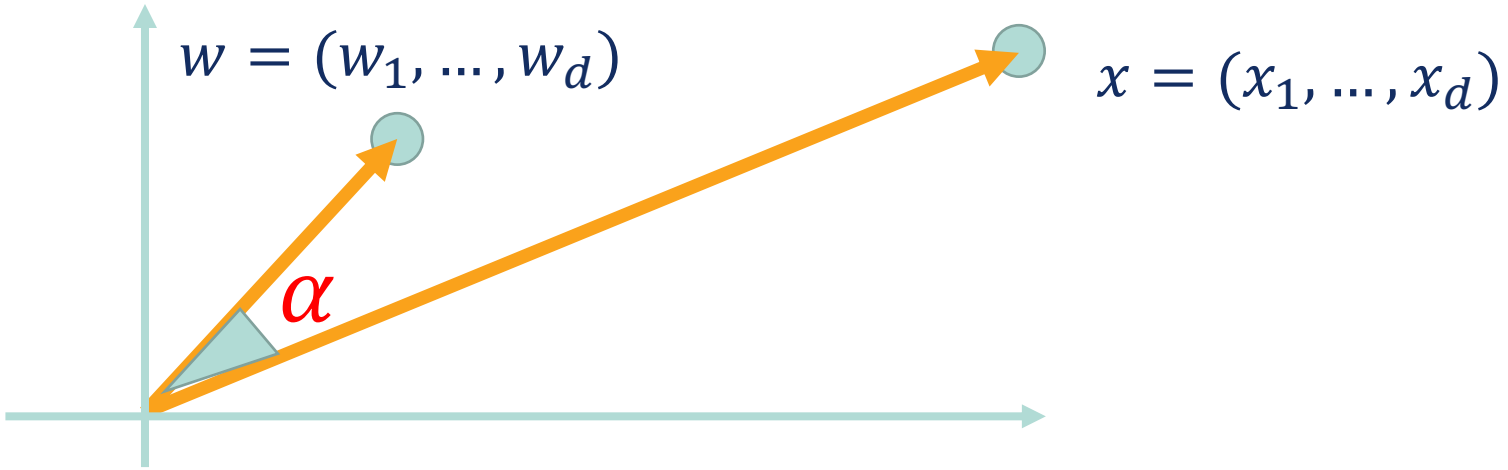
Произведение векторов*



$$\langle w, x \rangle = w_1 \cdot x_1 + \cdots + w_d \cdot x_d$$

*Если говорить более точно, - скалярное произведение векторов

Произведение векторов*



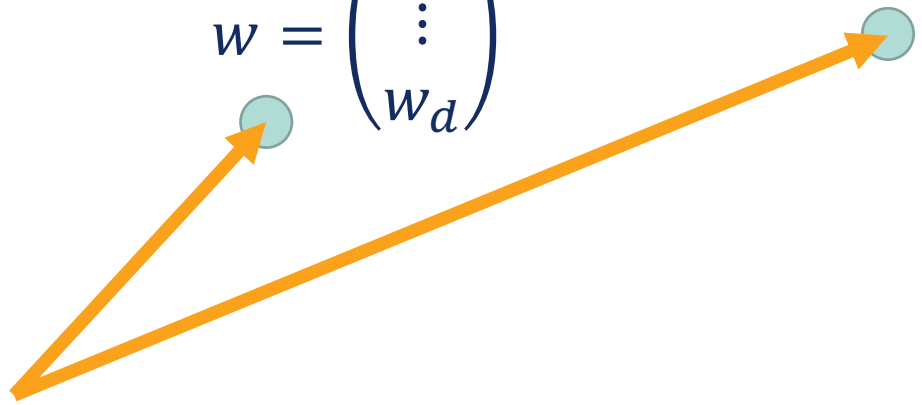
$$\langle w, x \rangle = w_1 \cdot x_1 + \cdots + w_d \cdot x_d$$

$$\langle w, x \rangle = \|w\| \, \|x\| \cos \alpha$$

*Если говорить более точно, - скалярное произведение векторов

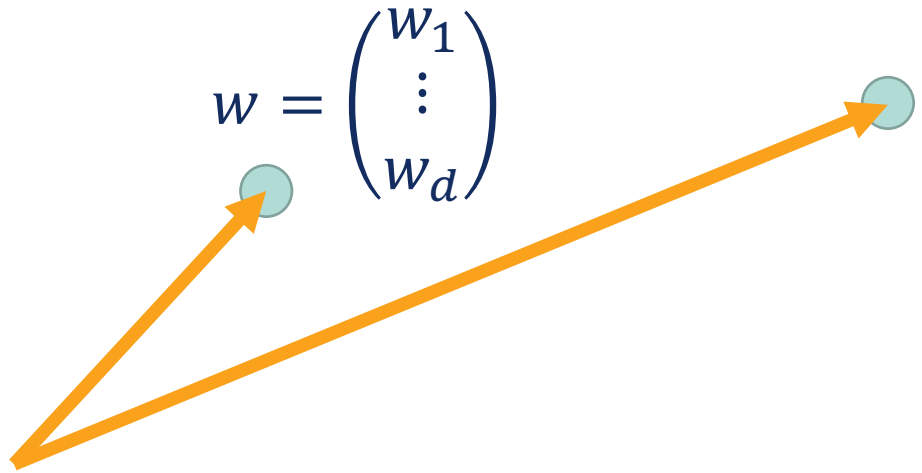
Строки и столбцы

$$w = \begin{pmatrix} w_1 \\ \vdots \\ w_d \end{pmatrix}$$



$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$$

Строки и столбцы

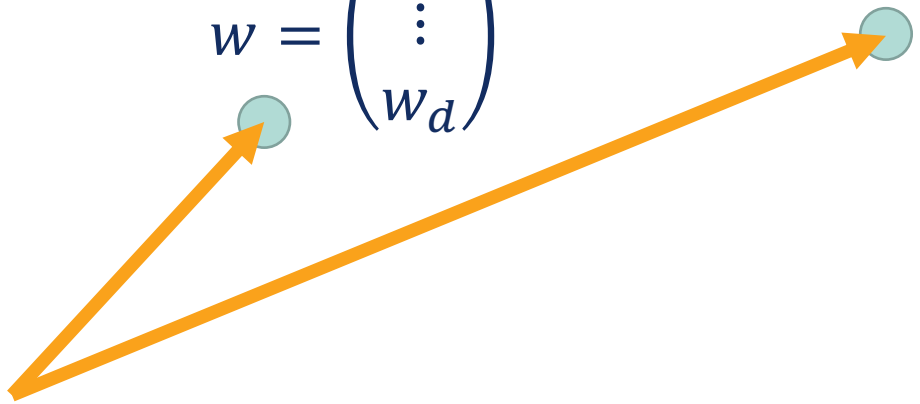


$$w = \begin{pmatrix} w_1 \\ \vdots \\ w_d \end{pmatrix}$$

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$$

$$\langle w, x \rangle = (w_1, \ldots, w_d) \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$$

Строки и столбцы

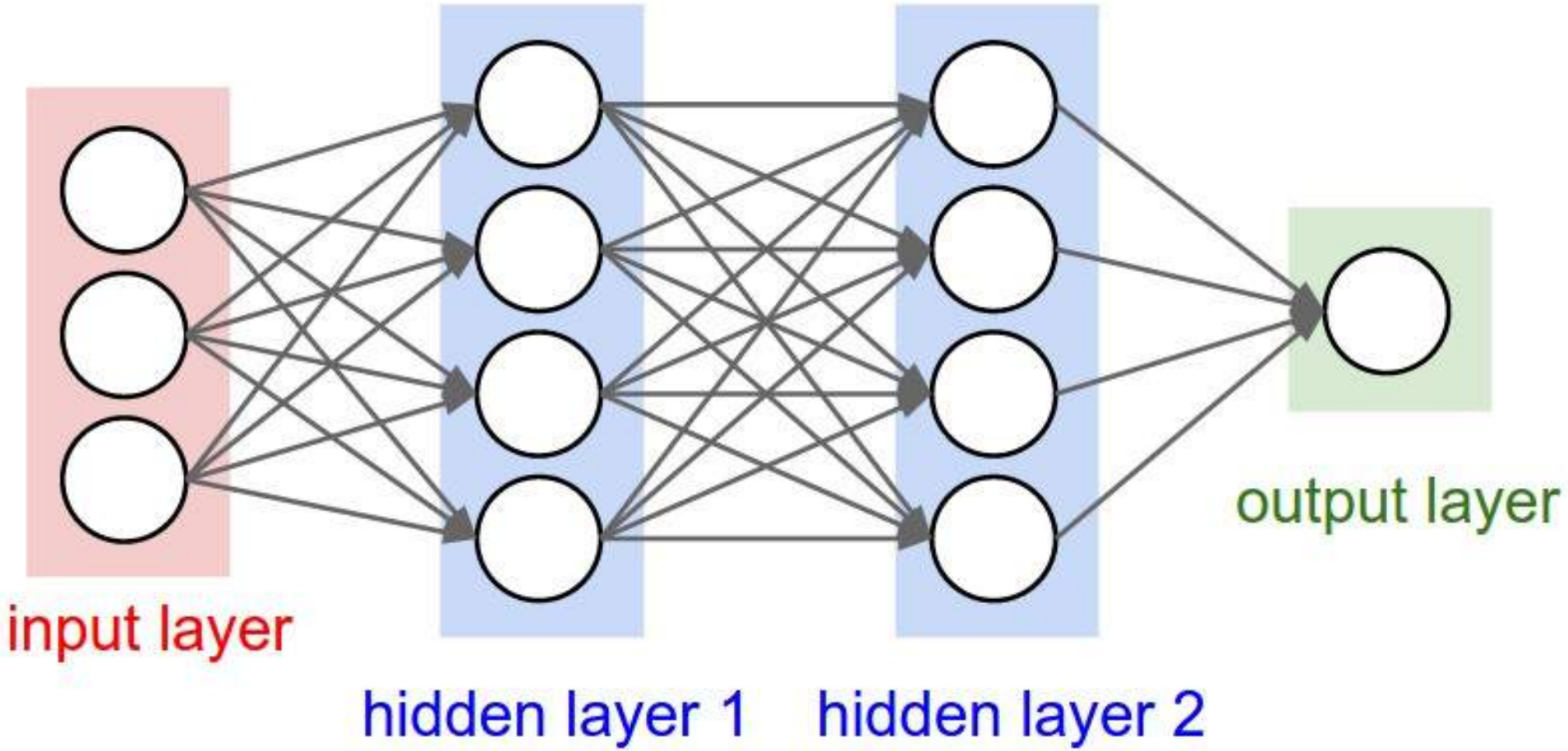


$$w = \begin{pmatrix} w_1 \\ \vdots \\ w_d \end{pmatrix}$$

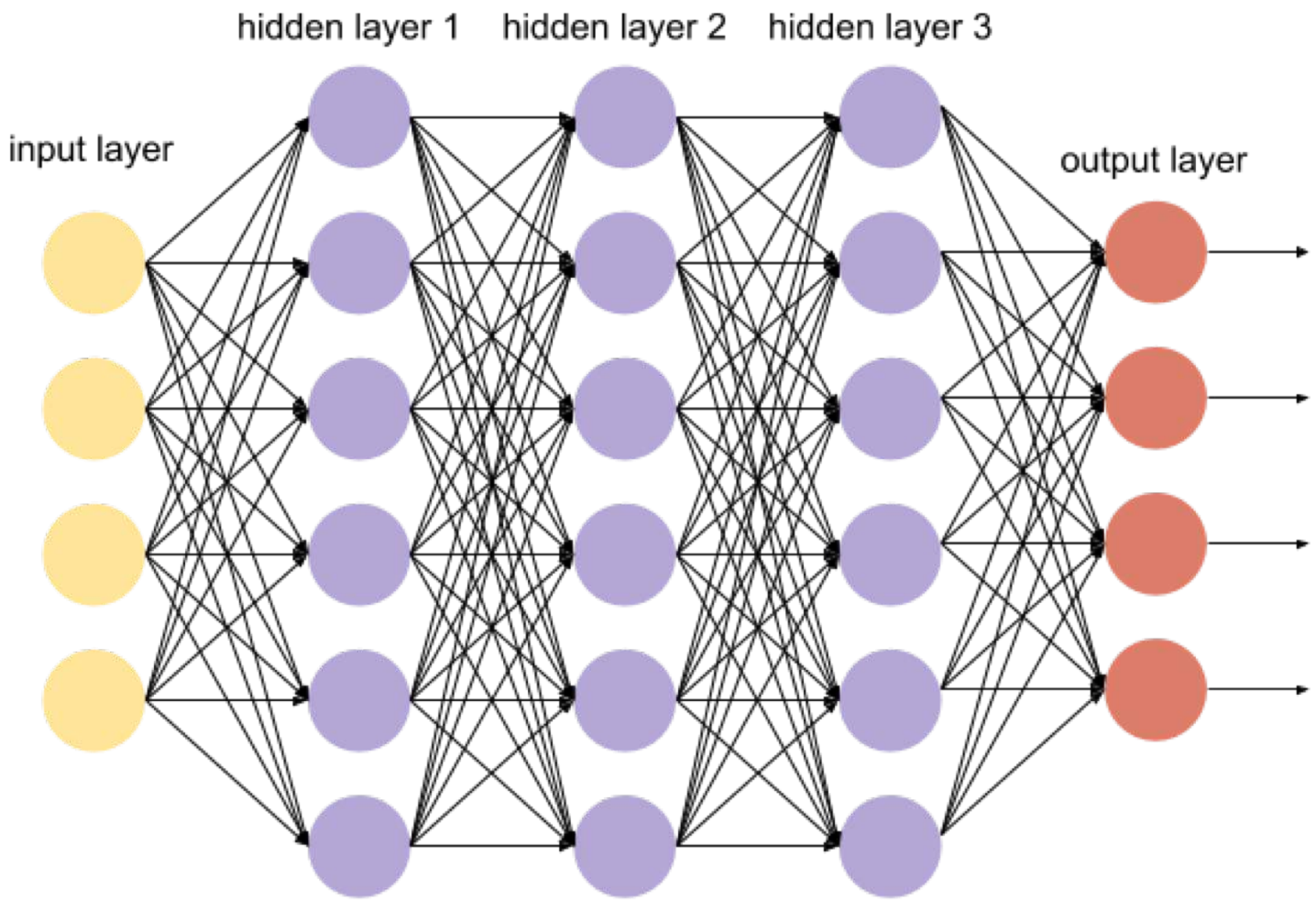
$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$$

$$\langle w, x \rangle = (w_1, \dots, w_d) \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} = w^T x$$

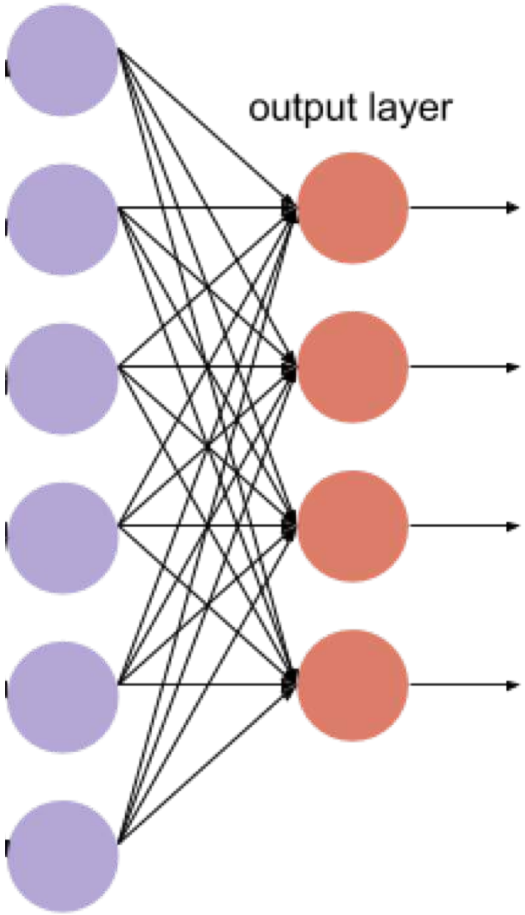
Нейронная сеть (один выход)



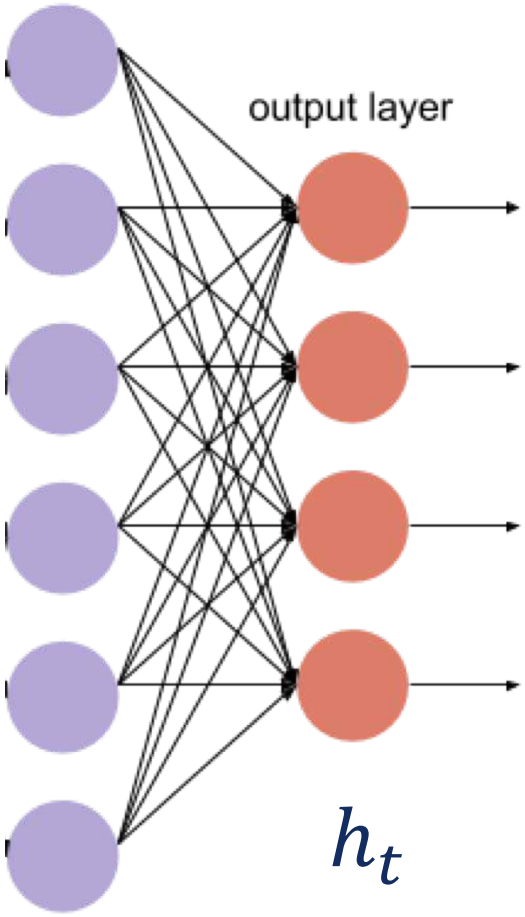
Нейронная сеть (много выходов)



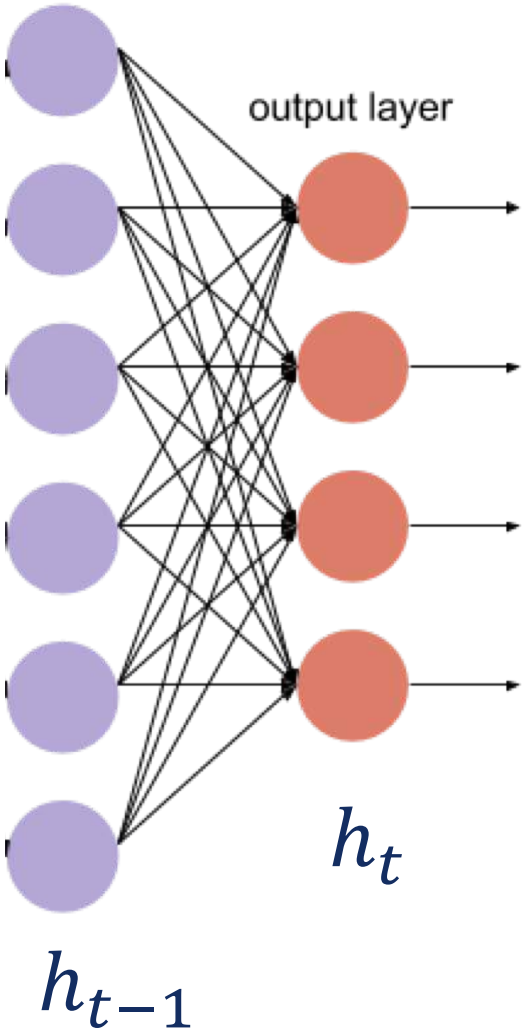
Если в следующем слое много нейронов



Если в следующем слое много нейронов



Если в следующем слое много нейронов



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

Матрицы

Было:

$$\langle w, x \rangle = (w_1, \dots, w_d) \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} = w^T x$$

Стало:

$$\begin{pmatrix} w_{11}, \dots, w_{1d} \\ \dots \\ w_{n1}, \dots, w_{nd} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} = Wx$$

Умножение на матрицу

$$\begin{pmatrix} w_{11}, \dots, w_{1d} \\ \dots \\ w_{n1}, \dots, w_{nd} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} = \begin{pmatrix} \blacksquare \\ \vdots \\ \blacksquare \end{pmatrix}$$

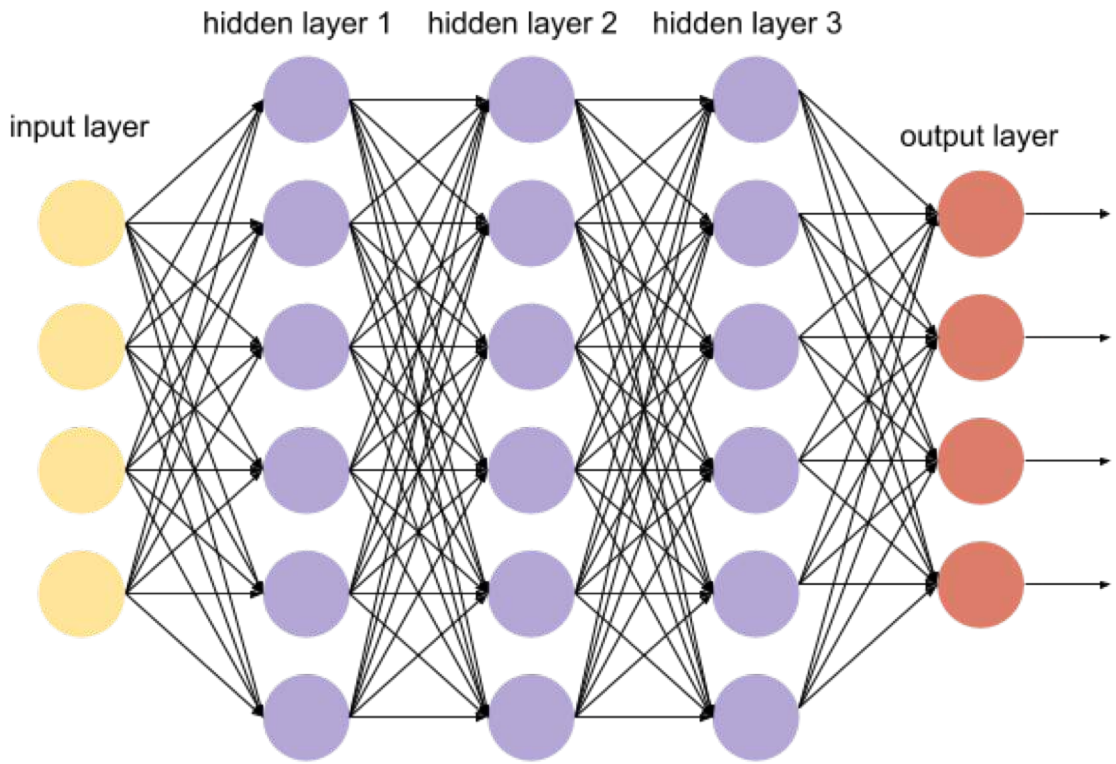
Умножение на матрицу

$$\begin{pmatrix} w_{11}, \dots, w_{1d} \\ \dots \\ w_{n1}, \dots, w_{nd} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} = \begin{pmatrix} \blacksquare \\ \vdots \\ \blacksquare \end{pmatrix}$$

$$\blacksquare = (w_{11}, \dots, w_{1d}) \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$$

$$\blacksquare = (w_{n1}, \dots, w_{nd}) \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$$

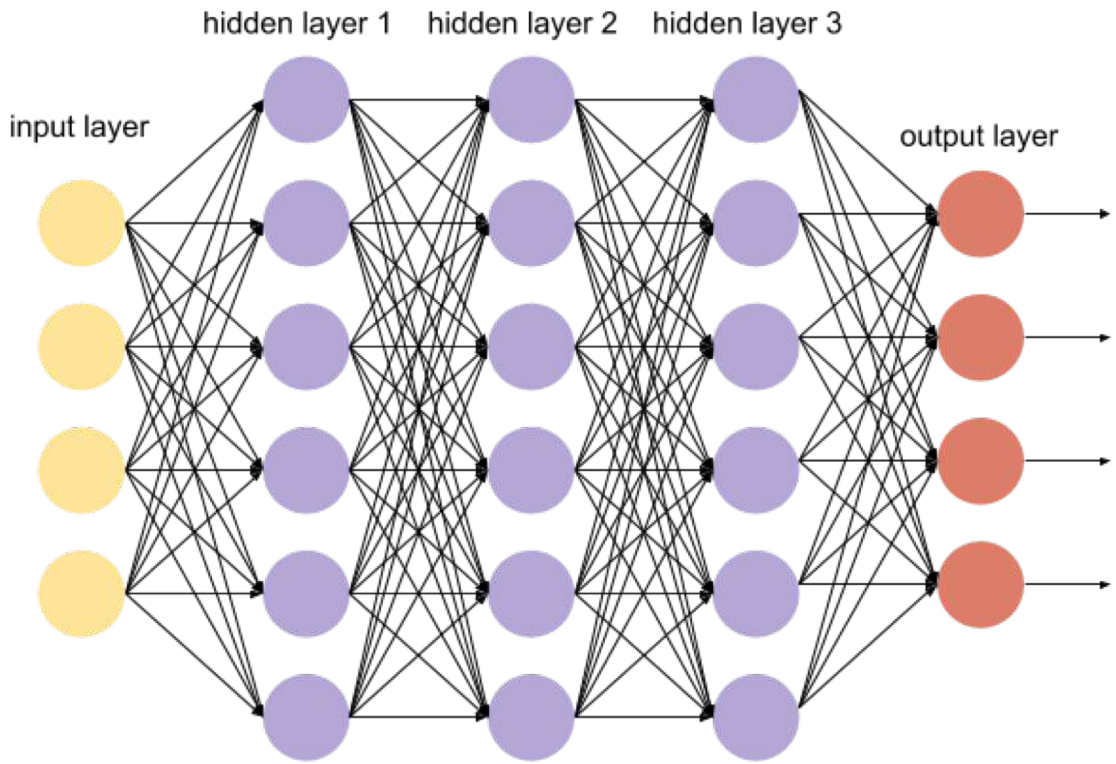
Нейронная сеть



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

$$f_0(W_0 x + b_0)$$

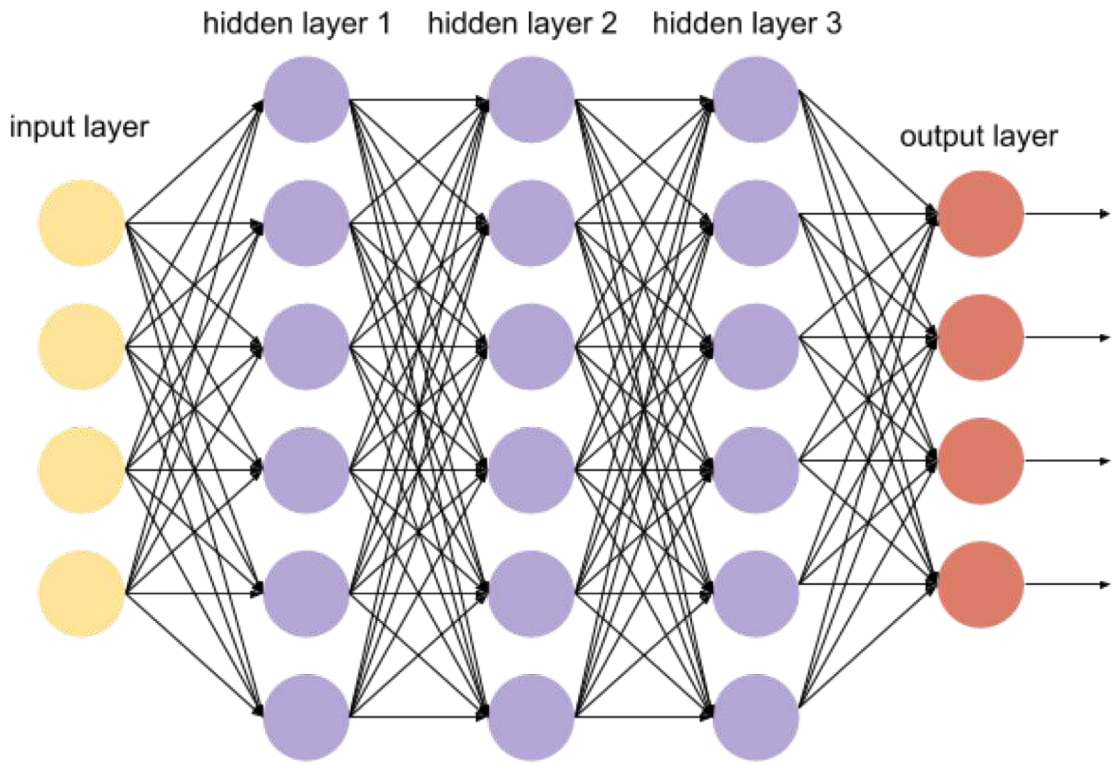
Нейронная сеть



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

$$f_1(W_1 f_0(W_0 x + b_0) + b_1)$$

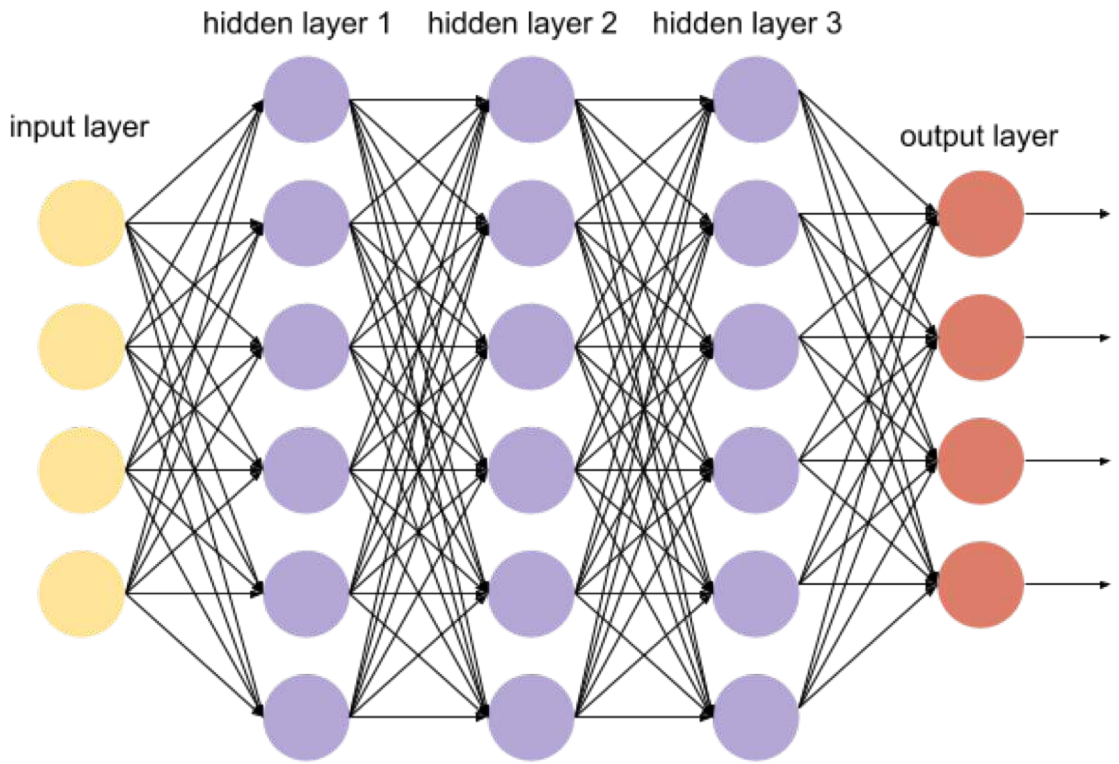
Нейронная сеть



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

$$f_2(W_2 f_1(W_1 f_0(W_0 x + b_0) + b_1) + b_2)$$

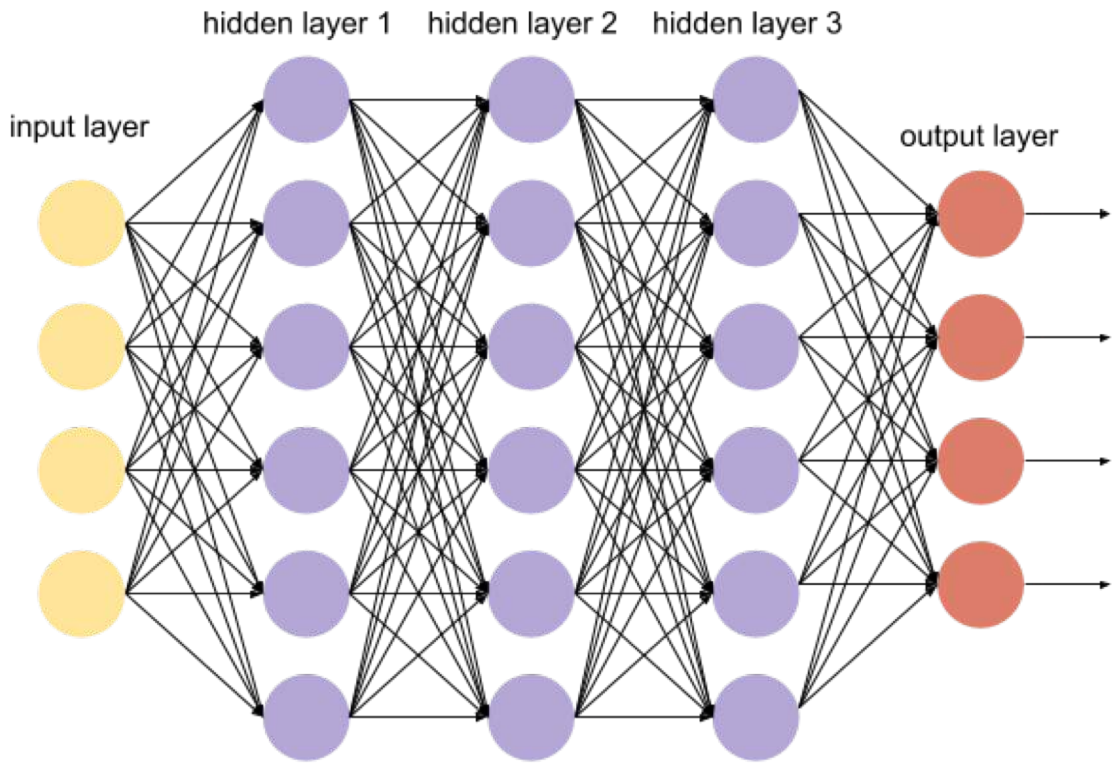
Нейронная сеть



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

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

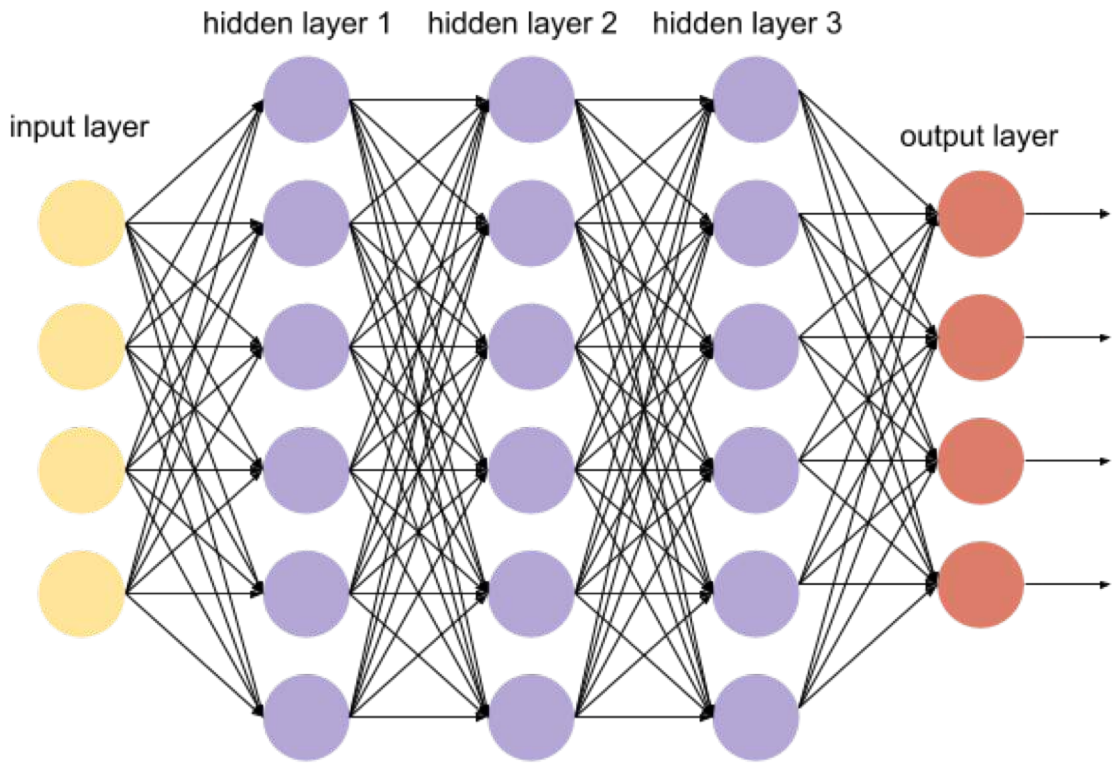
Нейронная сеть



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

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

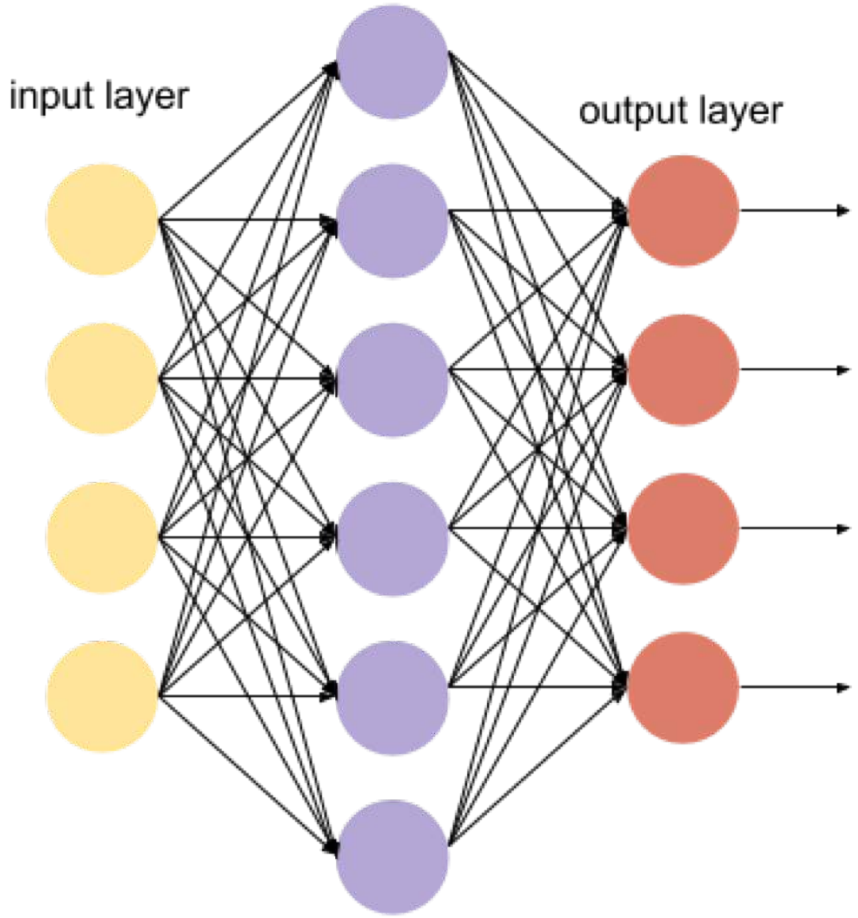
Нейронная сеть



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

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

Универсальная теорема аппроксимации



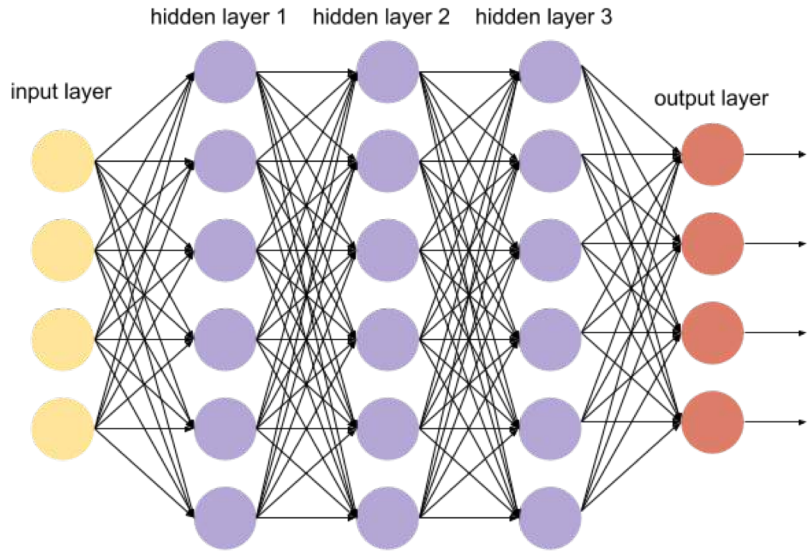
Капелька оптимизма перед обсуждением обучения:
В 1989 г. Джорджем Цыбенко (George Cybenko) была доказана Universal Approximation Theorem

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

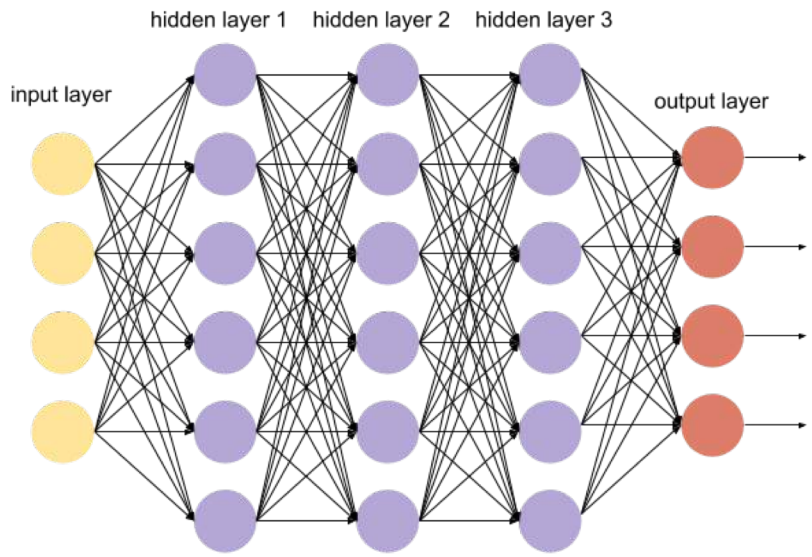
2. Обучение: backpropagation

Обучение сети



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

Обучение сети

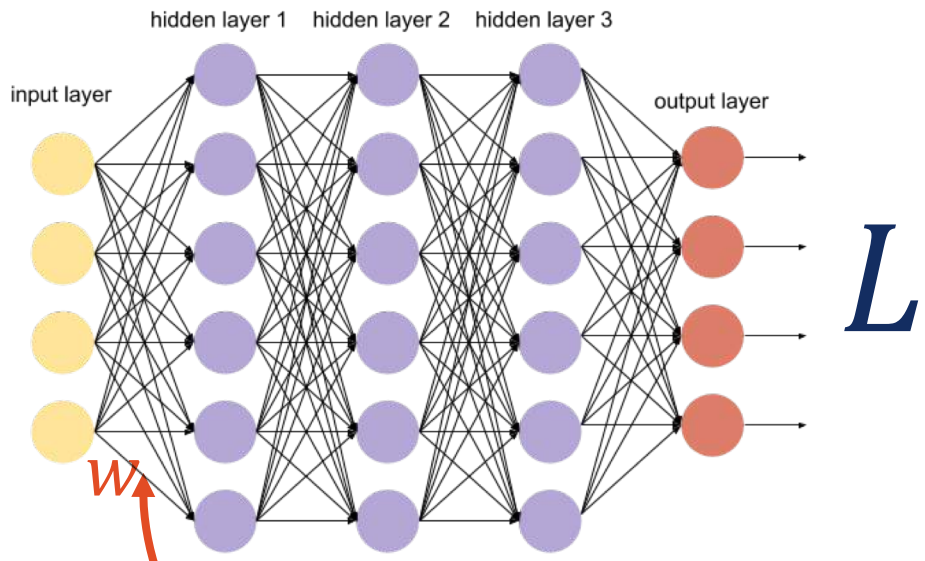


L

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

1. Выбираем функцию потерь

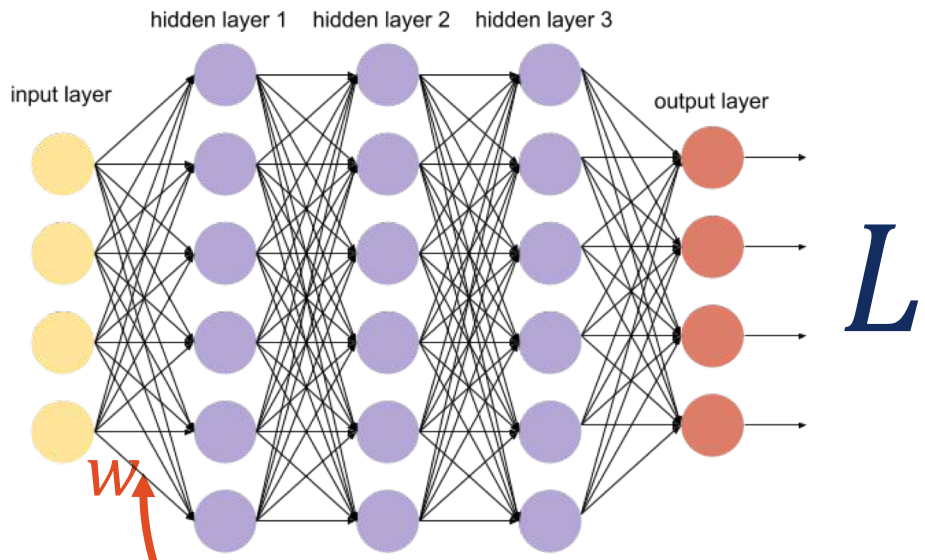
Обучение сети



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

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

Обучение сети

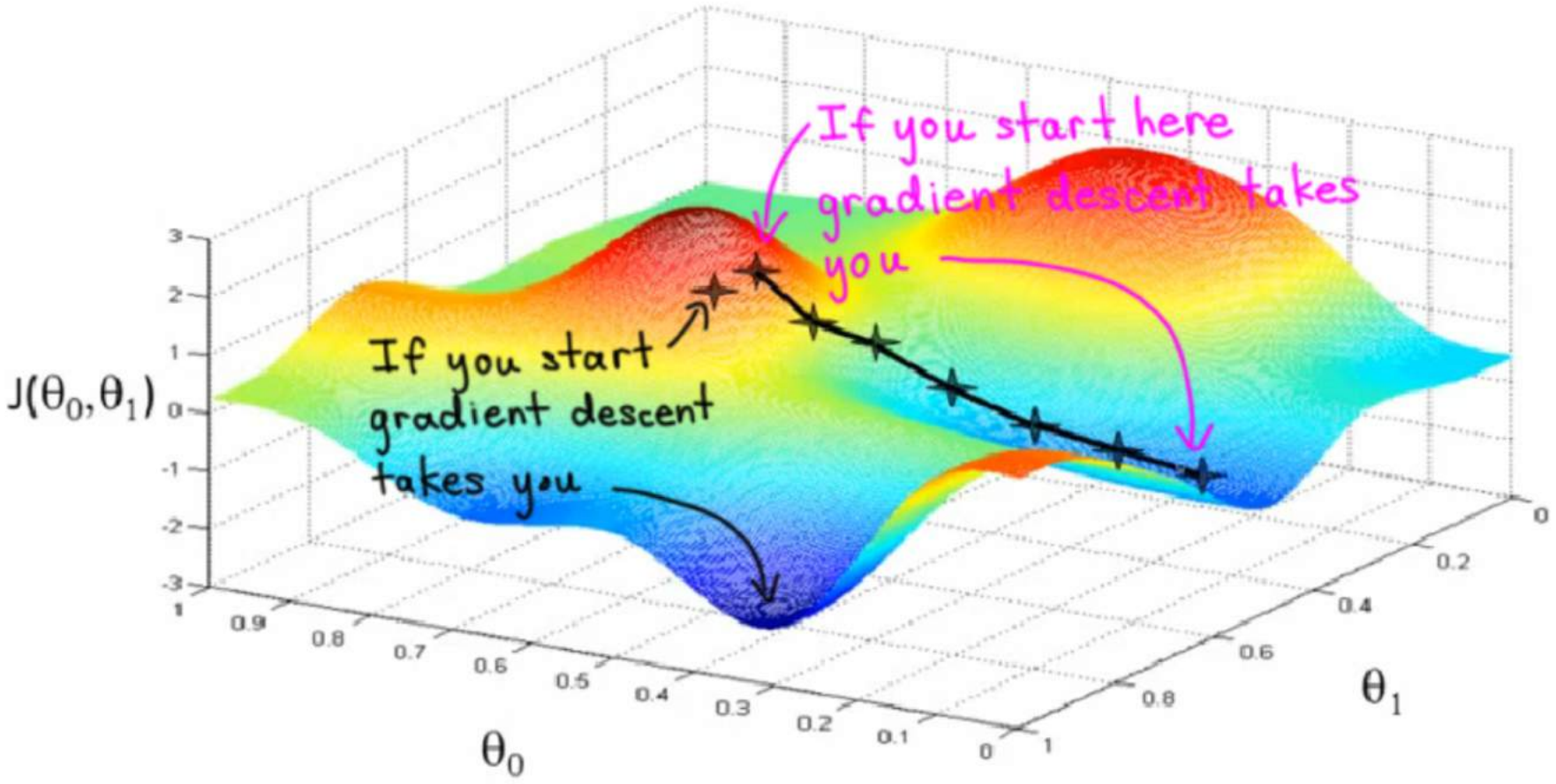


$$w^{(t+1)} = w^{(t)} - \gamma_t \frac{\partial L}{\partial w}$$

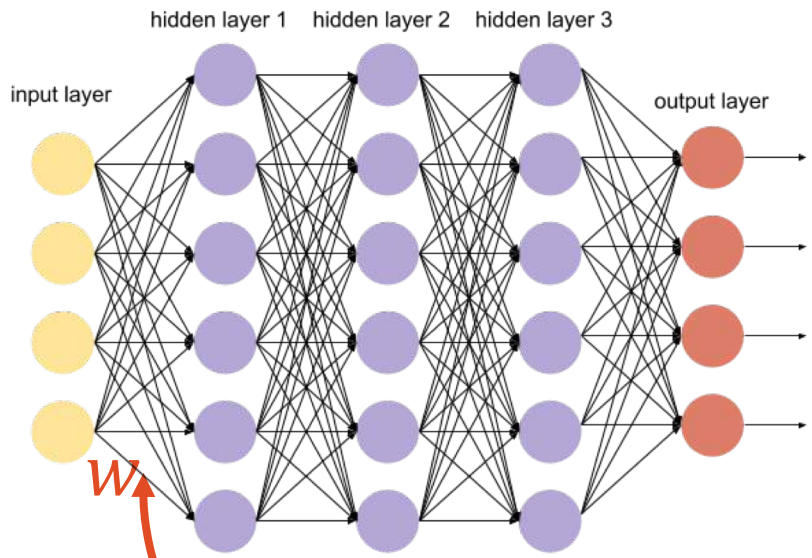
Задача обучения – настроить веса связей между нейронами на основе обучающей выборки

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

Градиентный спуск



Обучение сети



L

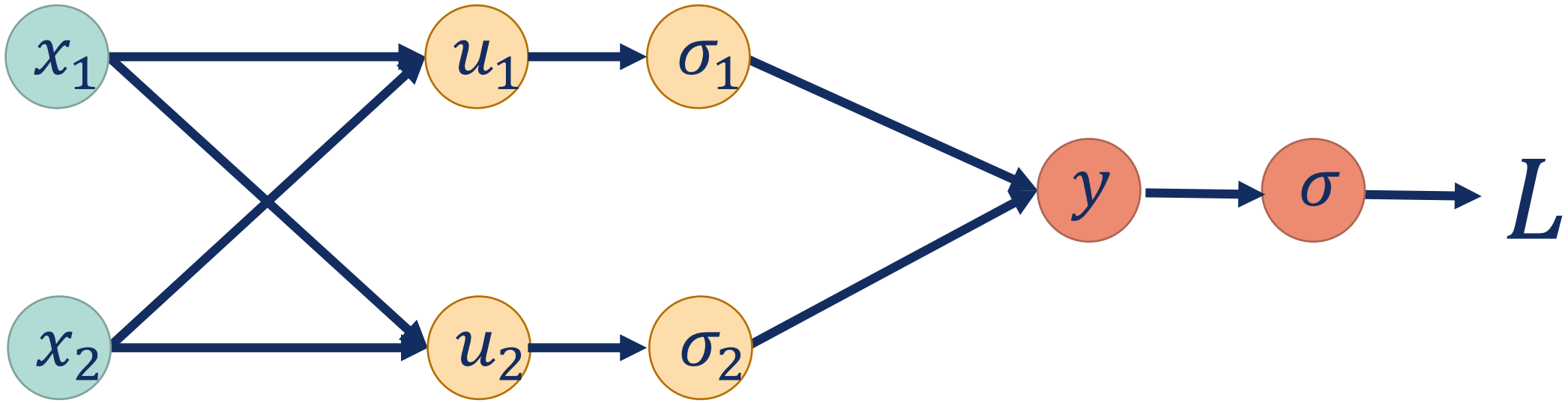
$$w^{(t+1)} = w^{(t)} - \gamma_t \frac{\partial L}{\partial w}$$

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

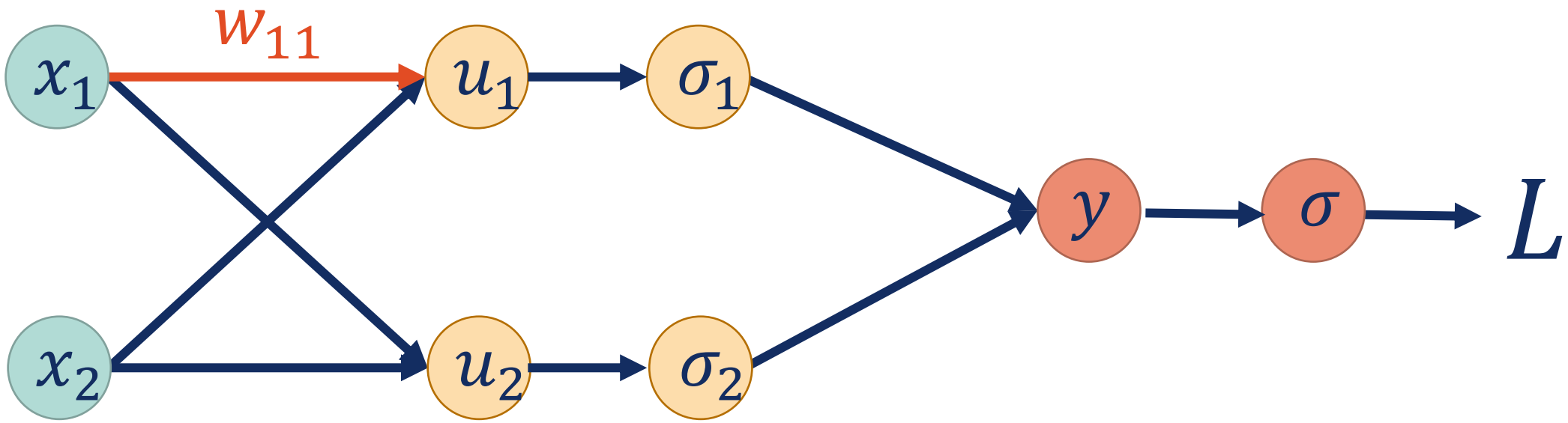
1. Выбираем функцию потерь
2. Обучаем веса с помощью SGD

Проблема: не выписывать же нам все производные аналитически?!

Граф вычислений для нейросети



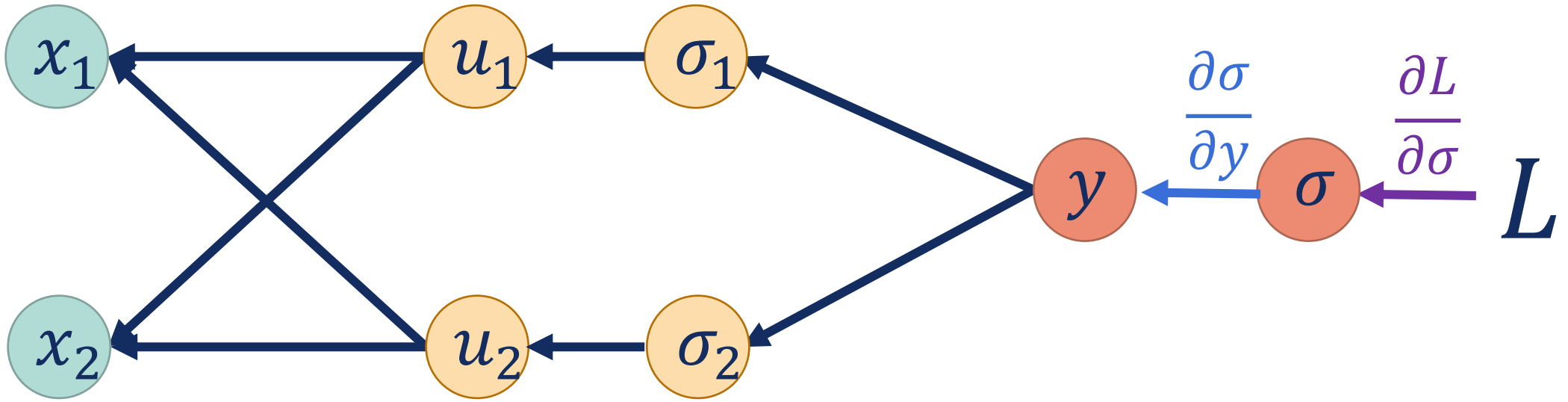
Граф вычислений для нейросети



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

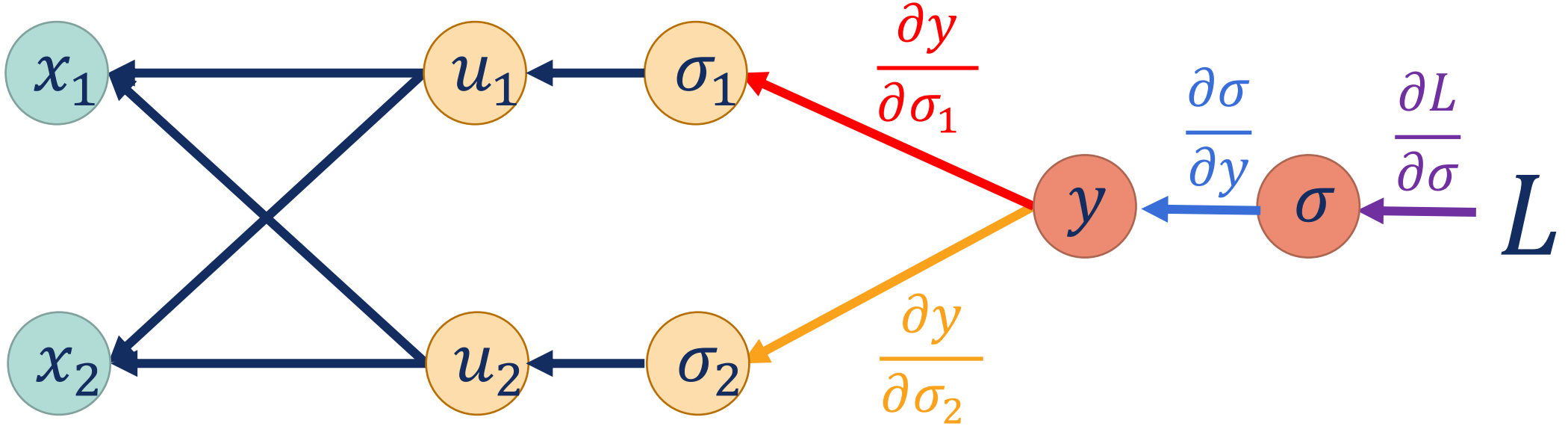
Backpropagation

Backprop – эффективный способ посчитать производные L по нейронам:

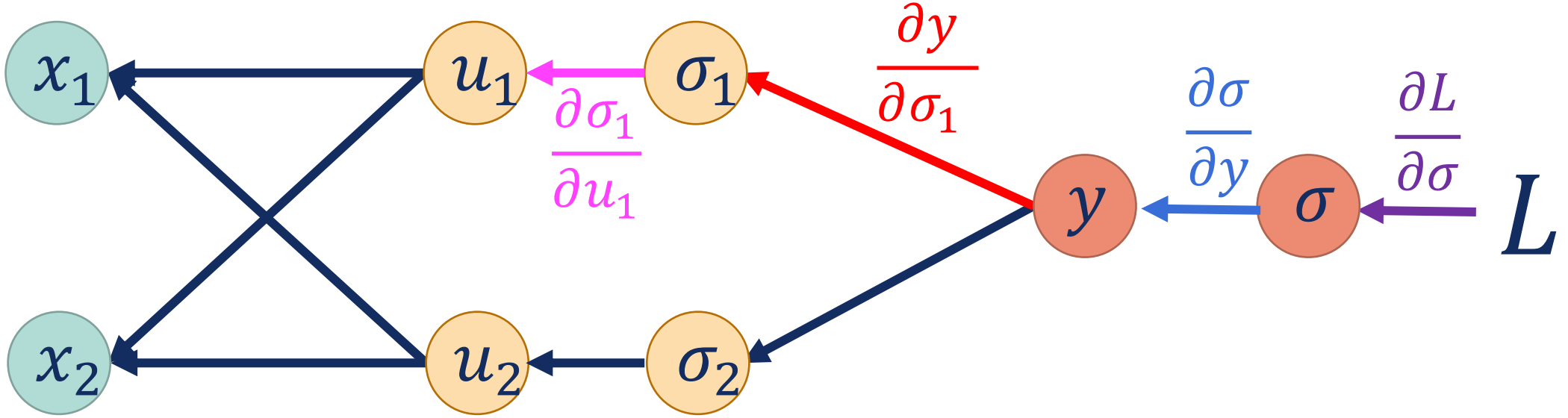


Backpropagation

Backprop – эффективный способ посчитать производные L по нейронам:

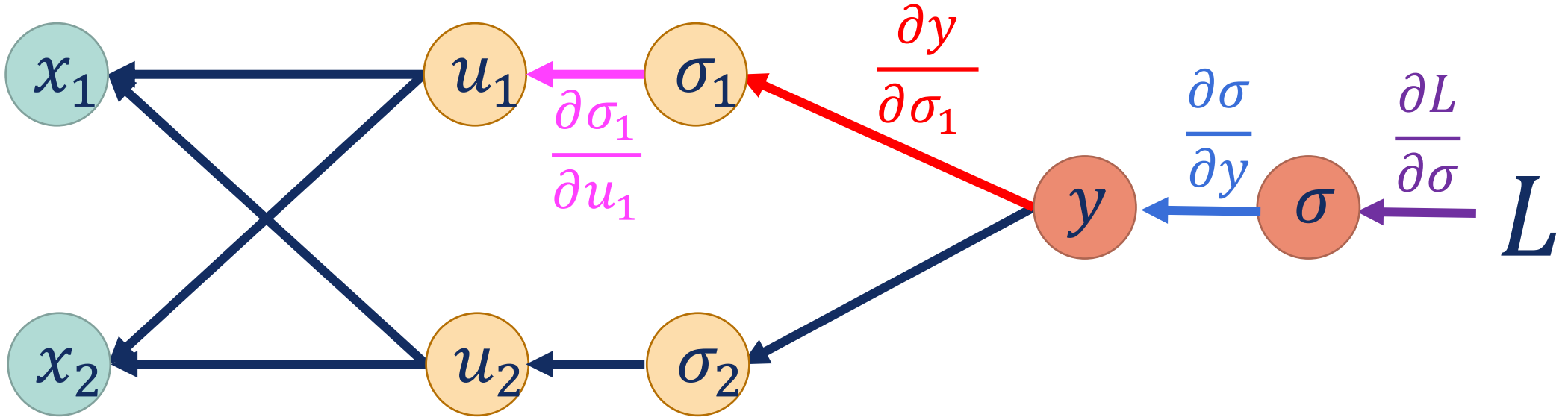


Как вычисляем производную



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

Как вычисляем производную

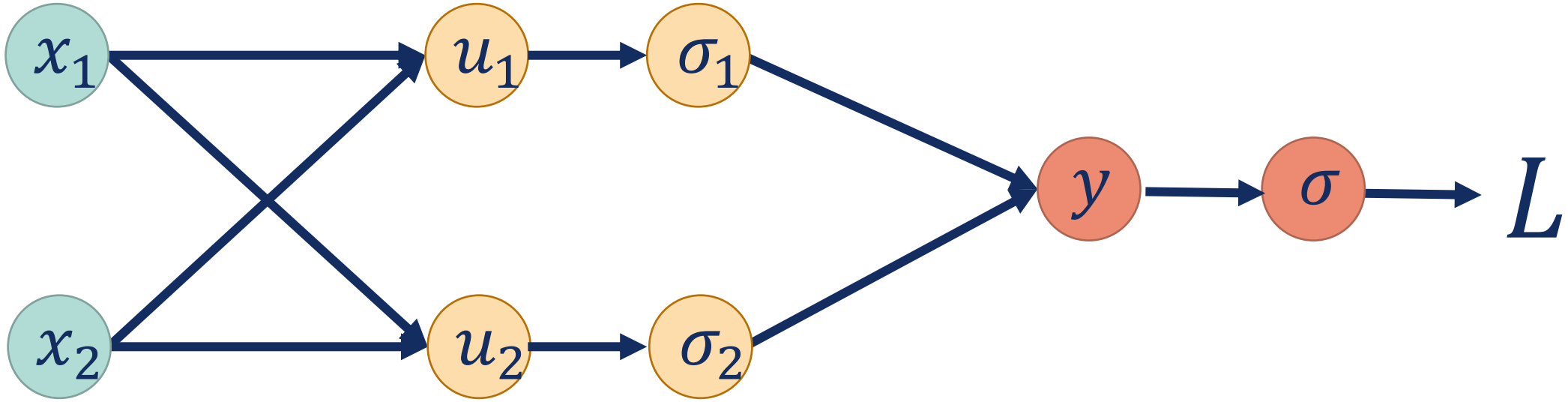


$$\frac{\partial L}{\partial u_1} = \frac{\partial L}{\partial \sigma}\frac{\partial \sigma}{\partial y}\frac{\partial y}{\partial \sigma_1}\frac{\partial \sigma_1}{\partial u_1}$$

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

Как происходит обучение сети

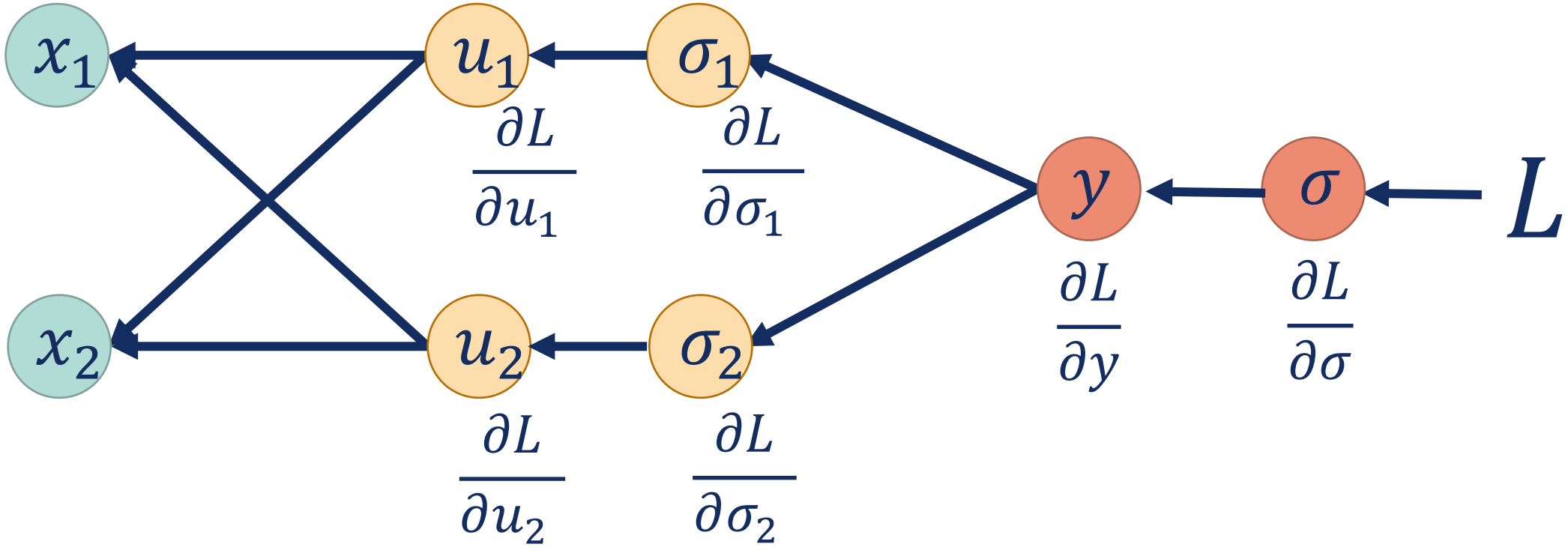
Чередуем **forward pass** (вычисление значений в нейронах)



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

Как происходит обучение сети

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)

Где можно прочитать про backprop

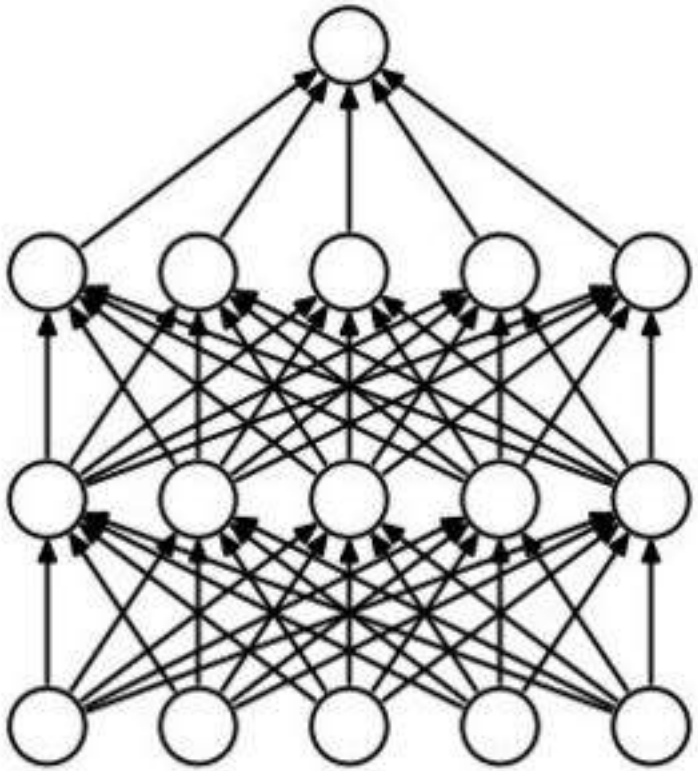
- Если вам ближе код, см. реализацию на Python: https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/
- Если вам ближе формулы, см. стр. 13 в лекциях К.В. Воронцова по нейросетям: http://www.ccas.ru/voron/download/NeuralNets.pdf
- Если нужно неторопливое и подробное объяснение, см. стр. 200-217 в Deep Learning Book: https://www.deeplearningbook.org/contents/mlp.html

3. Регуляризация и другие детали тренировки

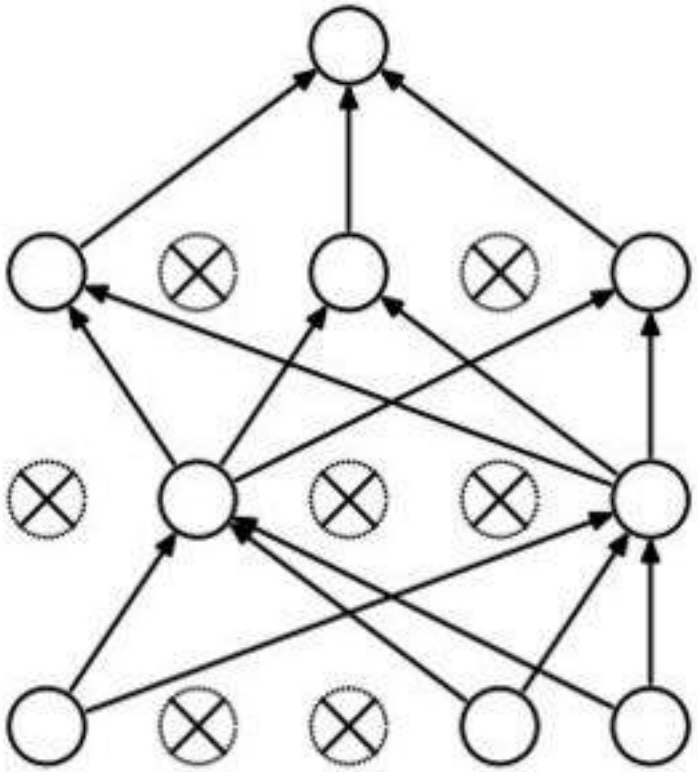
Проблемы backprop

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

Пример регуляризации: dropout



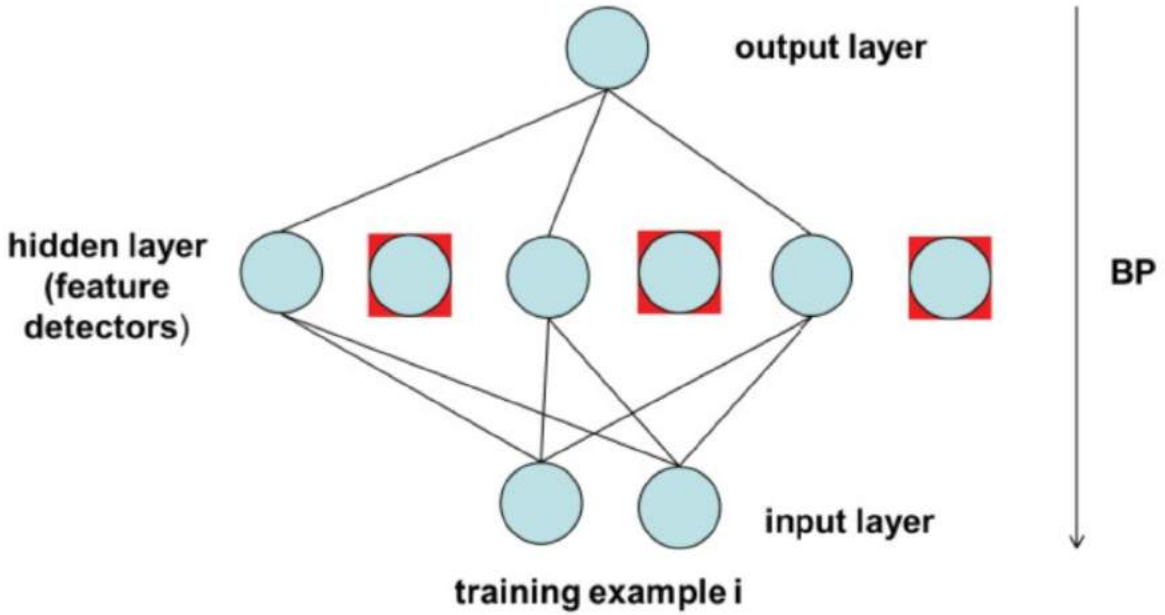
(a) Standard Neural Net



(b) After applying dropout.

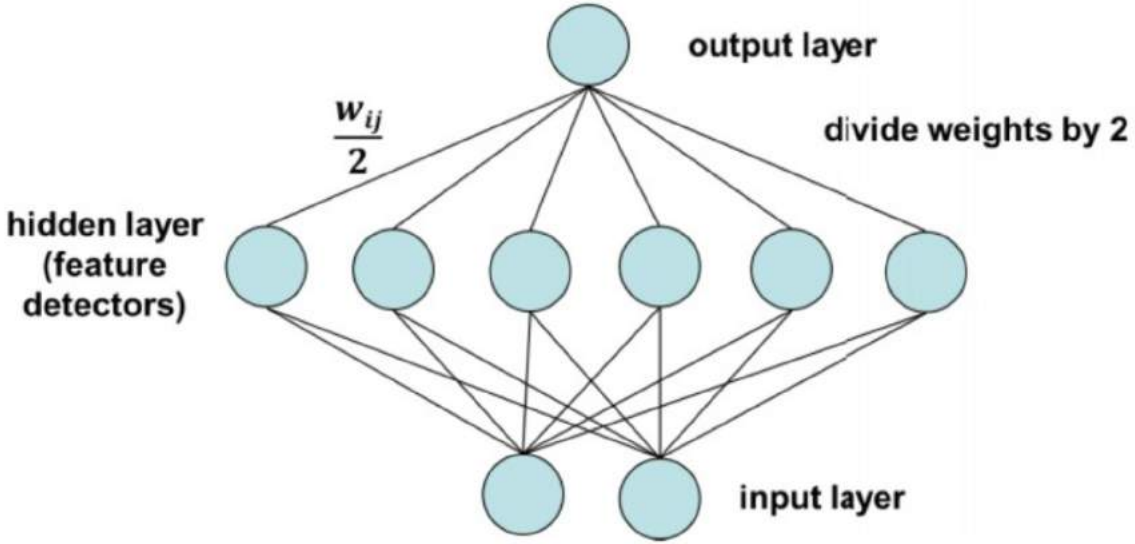
Dropout: обучение

С вероятностью p зануляем
выход каждого нейрона на слое

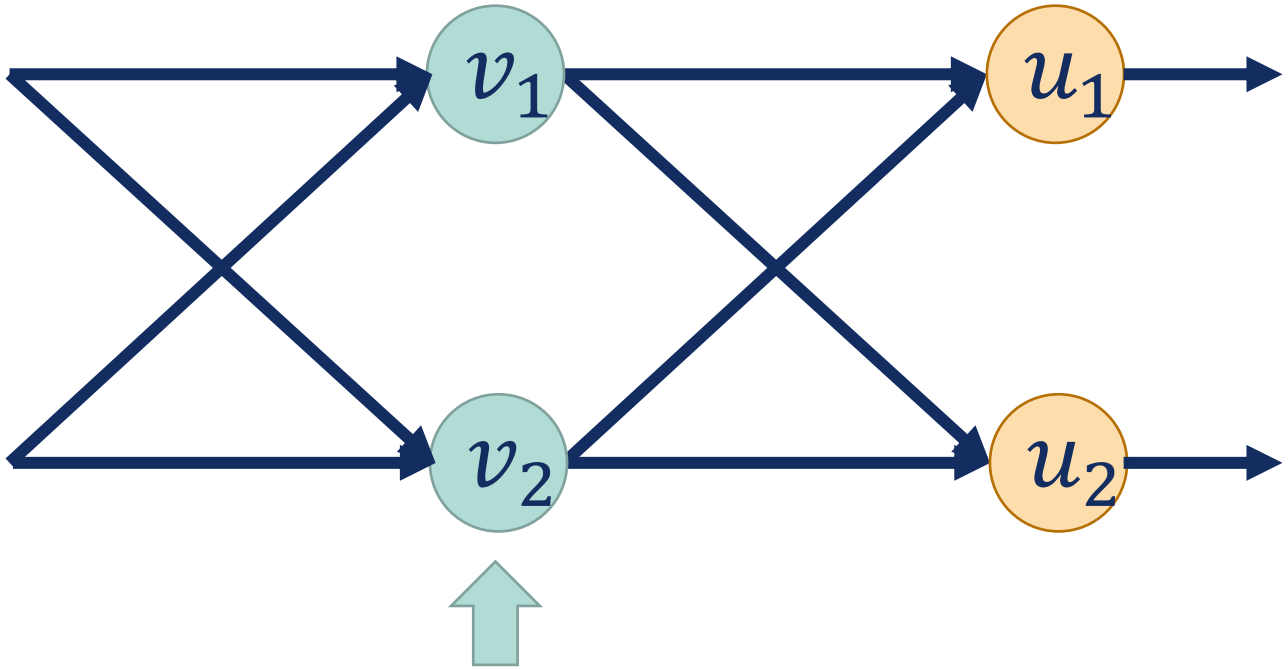


Dropout: применение

Домножаем выход каждого нейрона на (1-p)



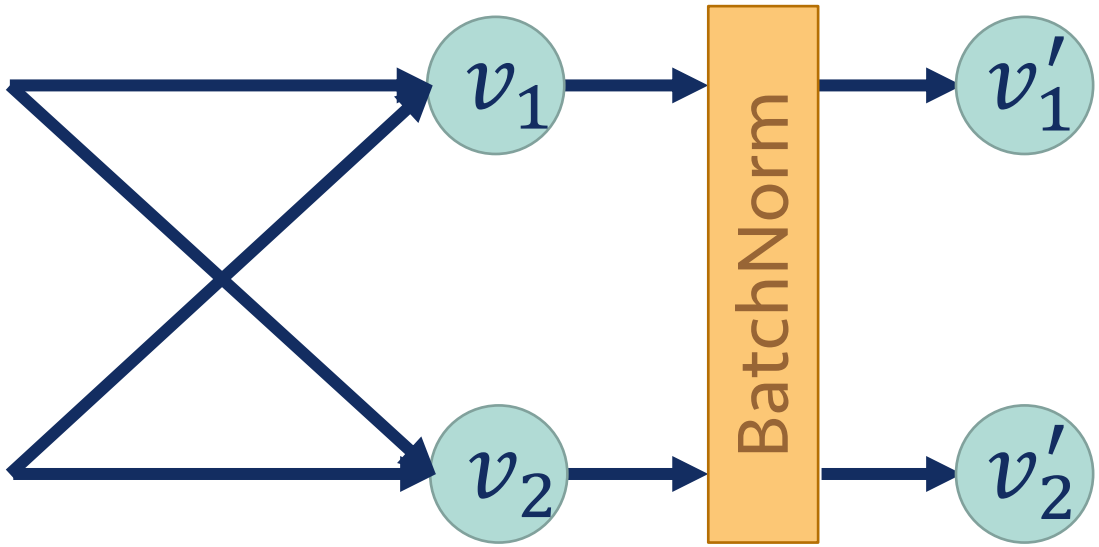
Еще один пример борьбы с переобучением



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



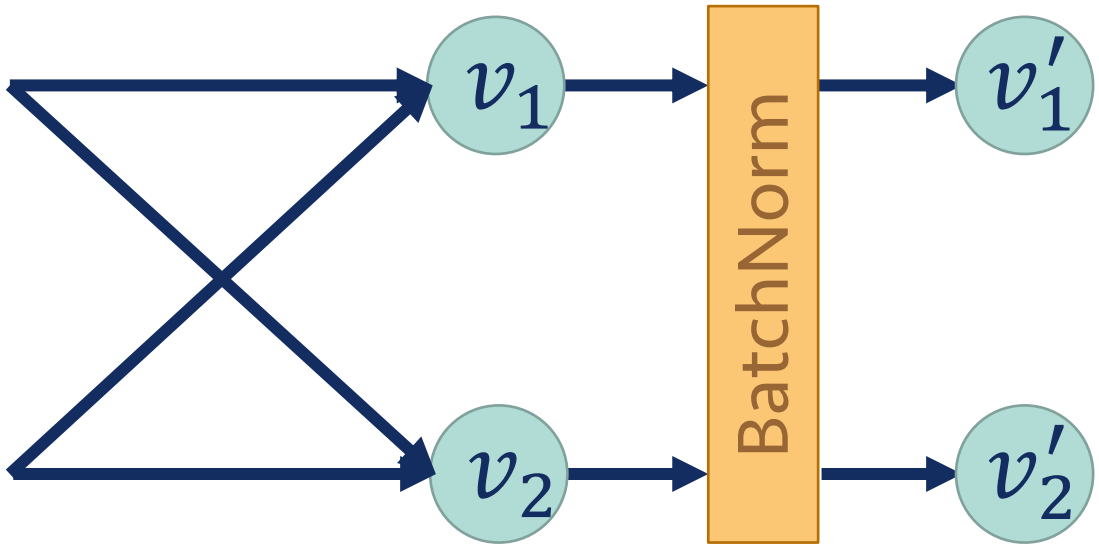
Batch Normalization:
Будем добавлять слои, в которых каждый признак будет нормироваться на среднее и дисперсию по батчу

$v_1^{(1)}, \dots, v_1^{(m)}$ – 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

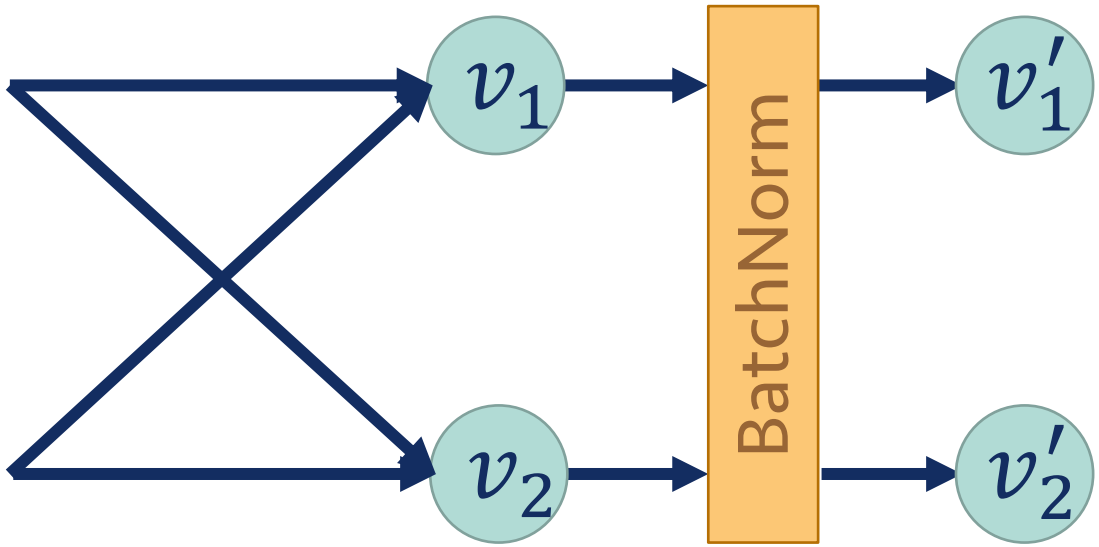


Batch Normalization:
Будем добавлять слои, в которых каждый признак будет нормироваться на среднее и дисперсию по батчу

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

Масштабирование и сдвиг: $v_1' = \gamma \tilde{v}_1 + \beta$

Еще один пример борьбы с переобучением

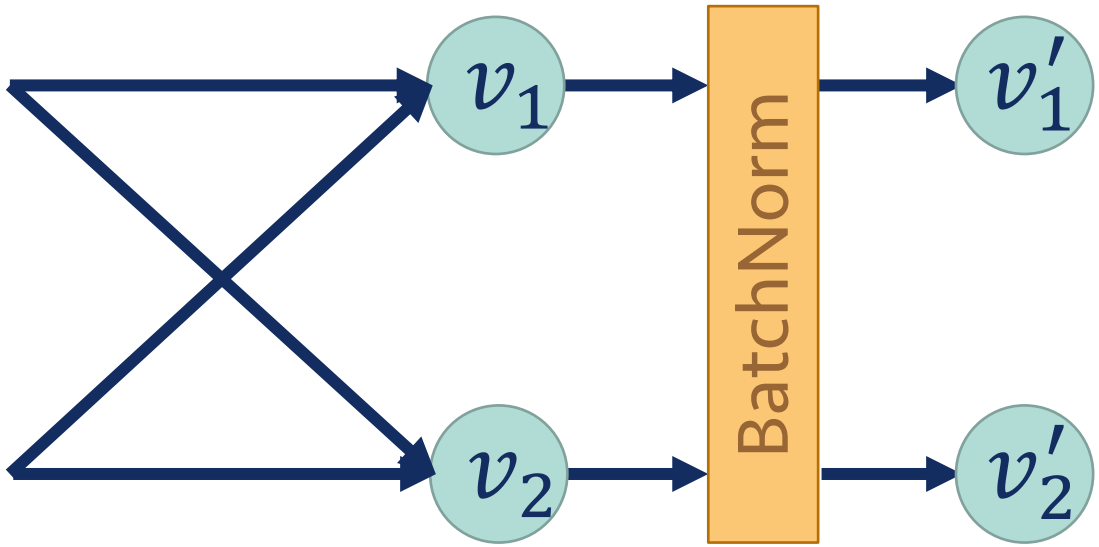


Batch Normalization:
Будем добавлять слои, в которых каждый признак будет нормироваться на среднее и дисперсию по батчу

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

Масштабирование и сдвиг: $v_1' = \gamma \tilde{v}_1 + \beta$

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: $v_1' = \gamma \tilde{v}_1 + \beta$

Полезный вопрос: зачем второй шаг?

Немного драмы: ICS не причём (NIPS 2018)

How Does Batch Normalization Help Optimization?

Shibani Santurkar* MIT shibani@mit.edu
Dimitris Tsipras* MIT tsipras@mit.edu
Andrew Ilyas* MIT ailyas@mit.edu
Aleksander Mądry MIT madry@mit.edu

Abstract

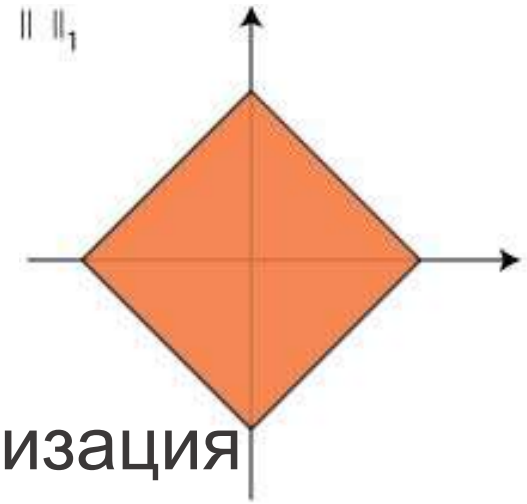
Batch Normalization (BatchNorm) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs). Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood. The popular belief is that this effectiveness stems from controlling the change of the layers' input distributions during training to reduce the so-called "internal covariate shift". In this work, we demonstrate that such distributional stability of layer inputs has little to do with the success of BatchNorm. Instead, we uncover a more fundamental impact of BatchNorm on the training process: it makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.

https://arxiv.org/abs/1805.11604

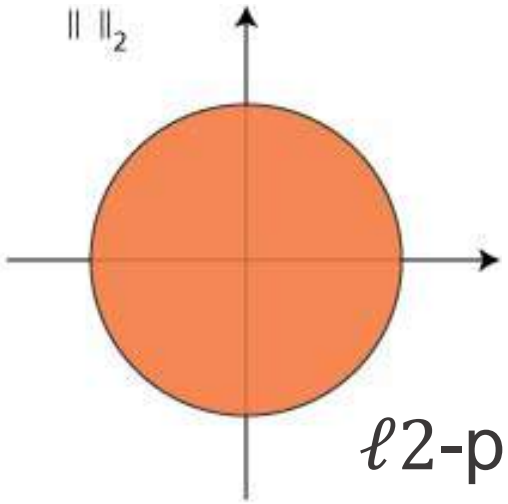
Напоминание: l1 и l2 регуляризаторы

$$\sum_{i=1}^{l} L_i + \gamma \sum_{n=1}^{d} |w_n| \to min$$

$$\sum_{i=1}^{l} L_i + \gamma \sum_{n=1}^{d} {w_n}^2 \to min$$



$\ell 1$-регуляризация



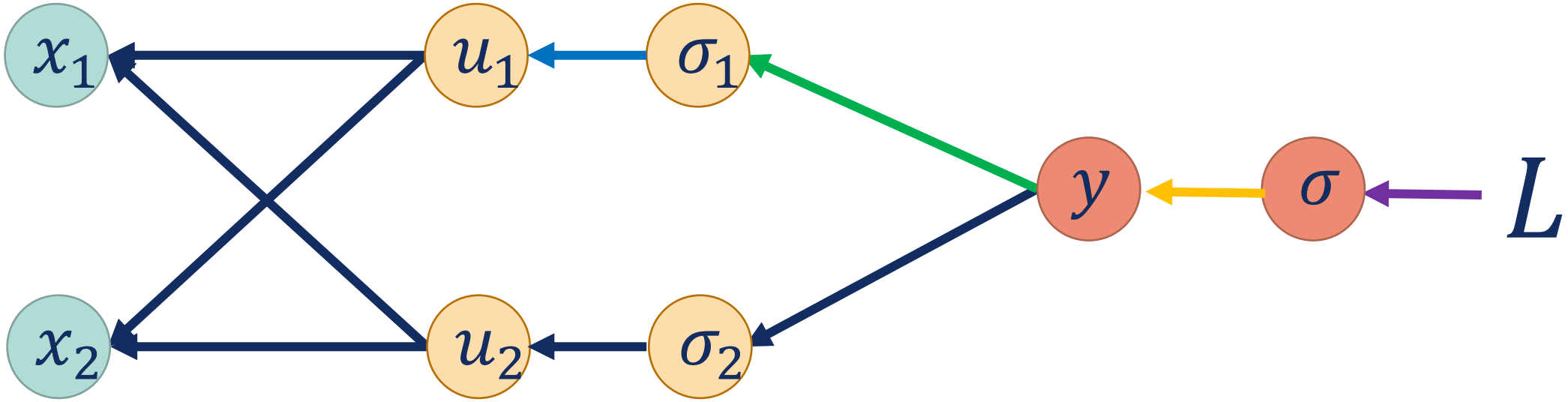
$\ell 2$-регуляризация

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)

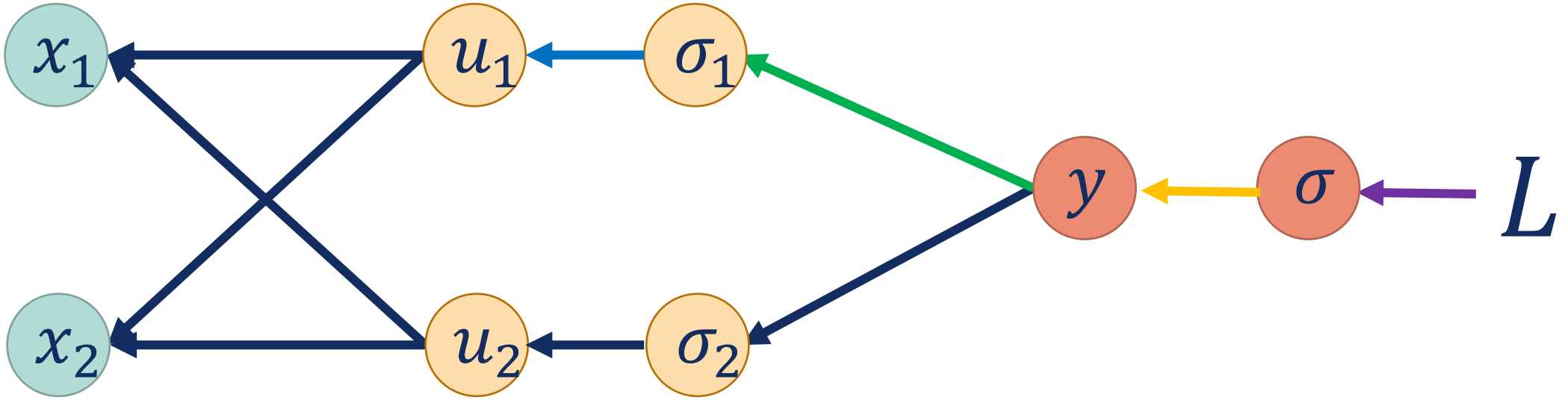
Затухание градиента (Gradient vanishing)

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



Затухание градиента (Gradient vanishing)

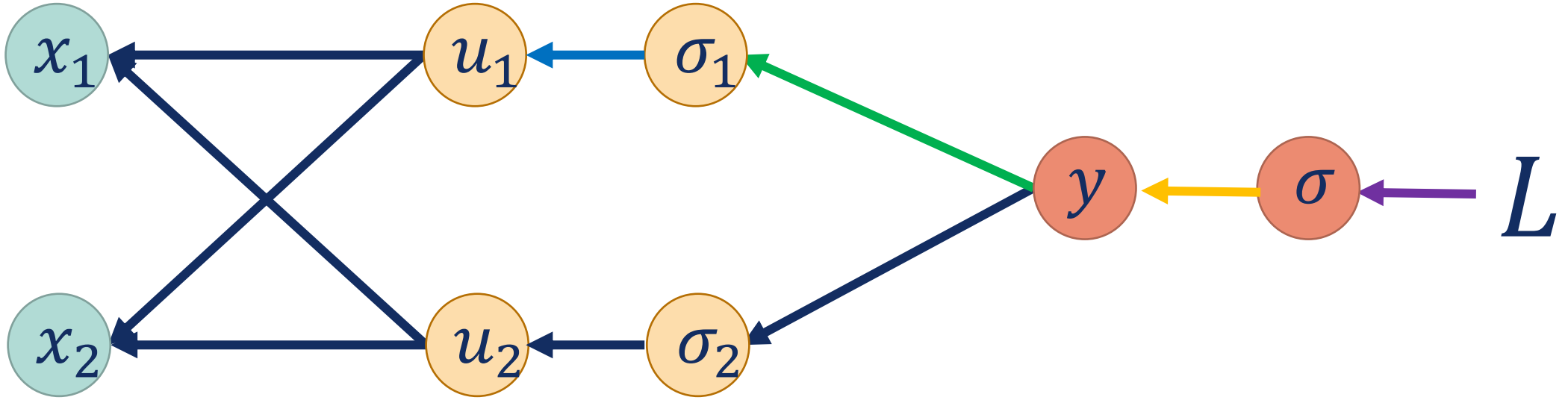
Производная по нейрону по chain rule получается из произведения производных по пути к нему от выхода



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

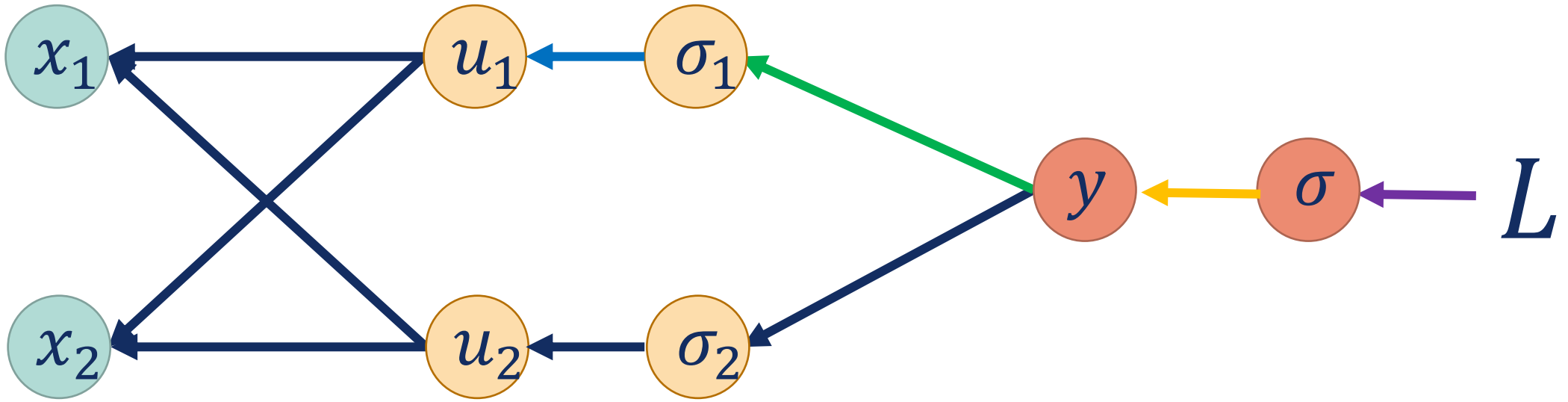


Если каждая из производных небольшая по модулю – произведение тоже будет маленьким. Чем больше слоев, тем меньше.

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?

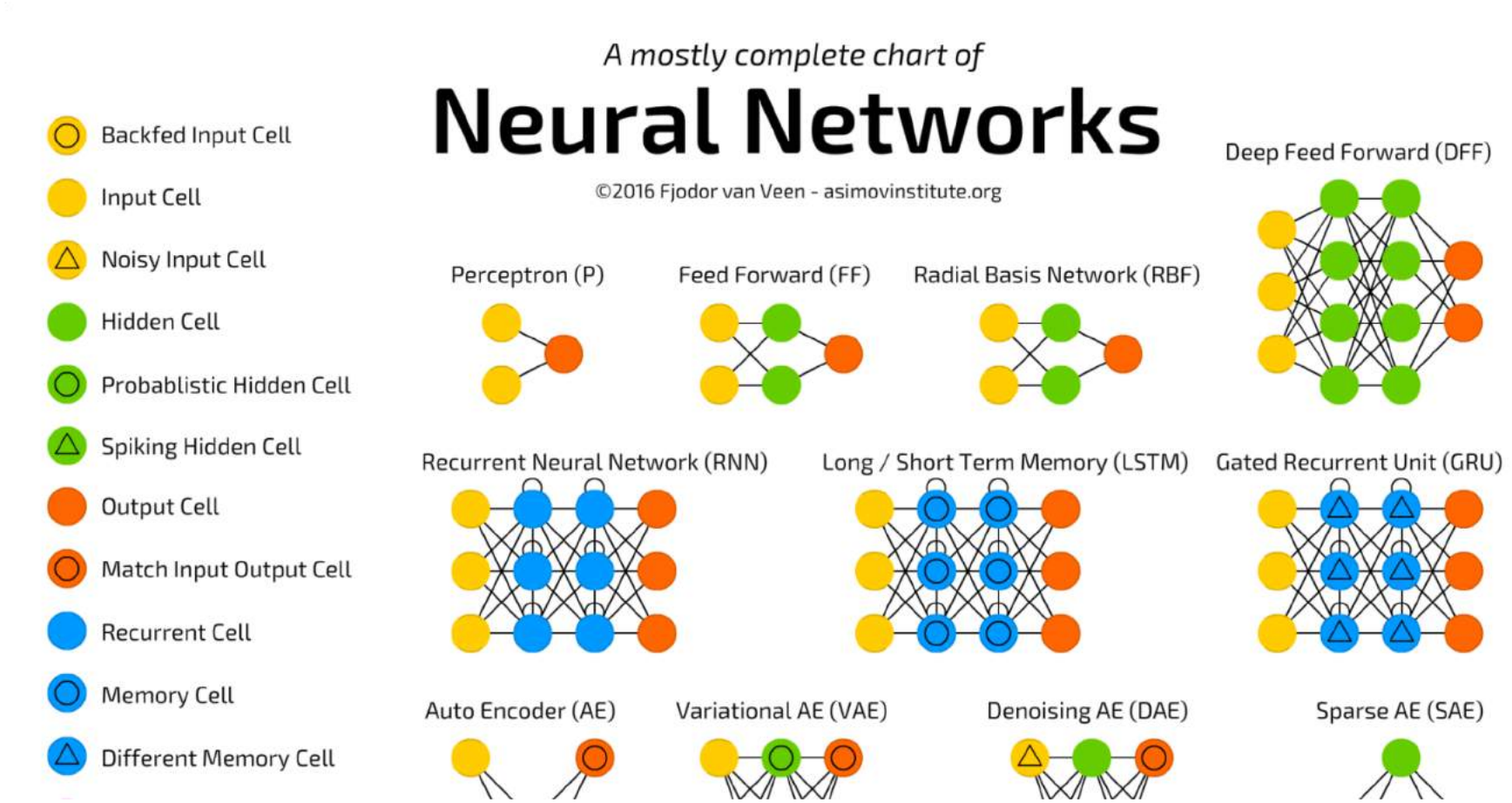
Нейросети и backpropagation известны давно, почему же последние достижения происходят только сейчас?

Два фактора:

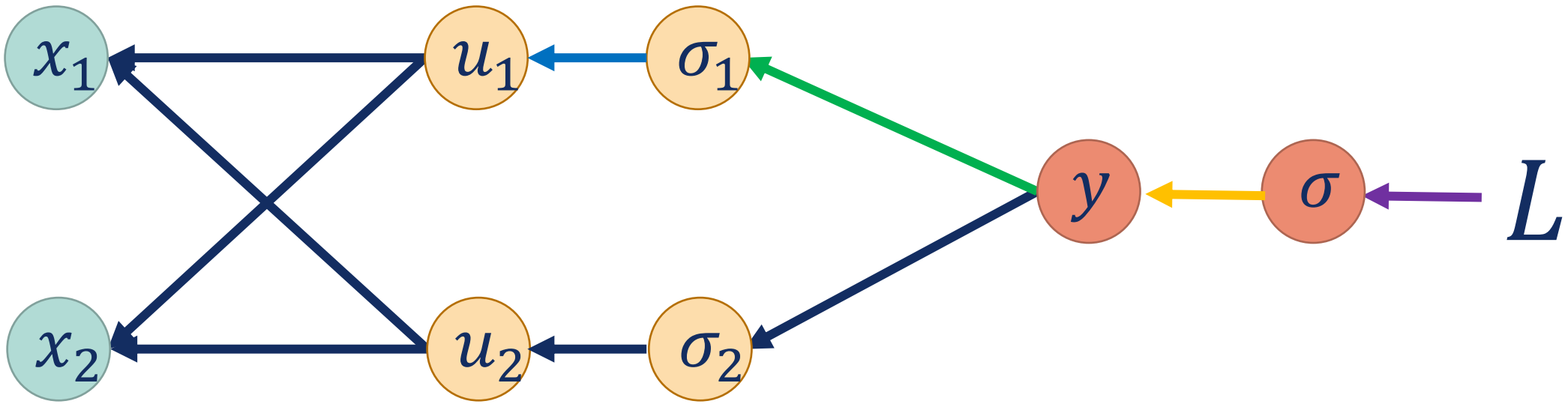
1. Выросли вычислительные мощности и развились вычисления на GPU
2. Тем временем люди придумали много полезных трюков и архитектур

Архитектуры нейросетей

http://www.asimovinstitute.org/neural-network-zoo/

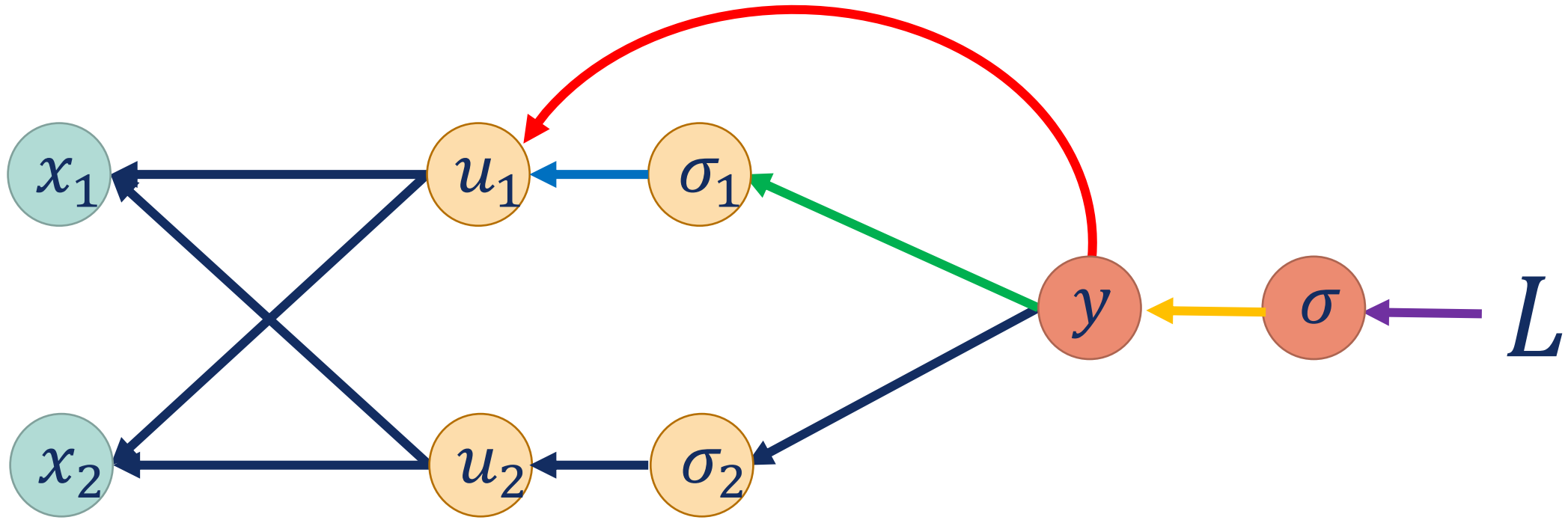


Пример: как архитектура помогает обучению

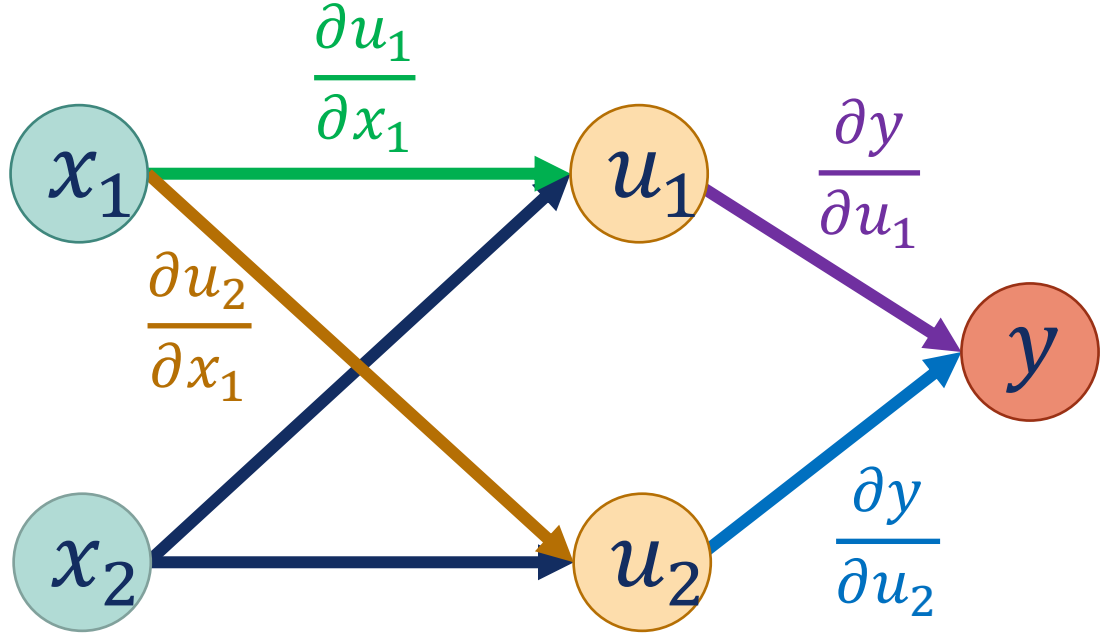


Пример: как архитектура помогает обучению

Добавляем «перепрыгивающие» через слои связи *(residual connection)*: теперь градиент будет «дотекать» не затухая, а большие градиенты просто будем обрезать *(gradient clipping)*



Инициализация весов

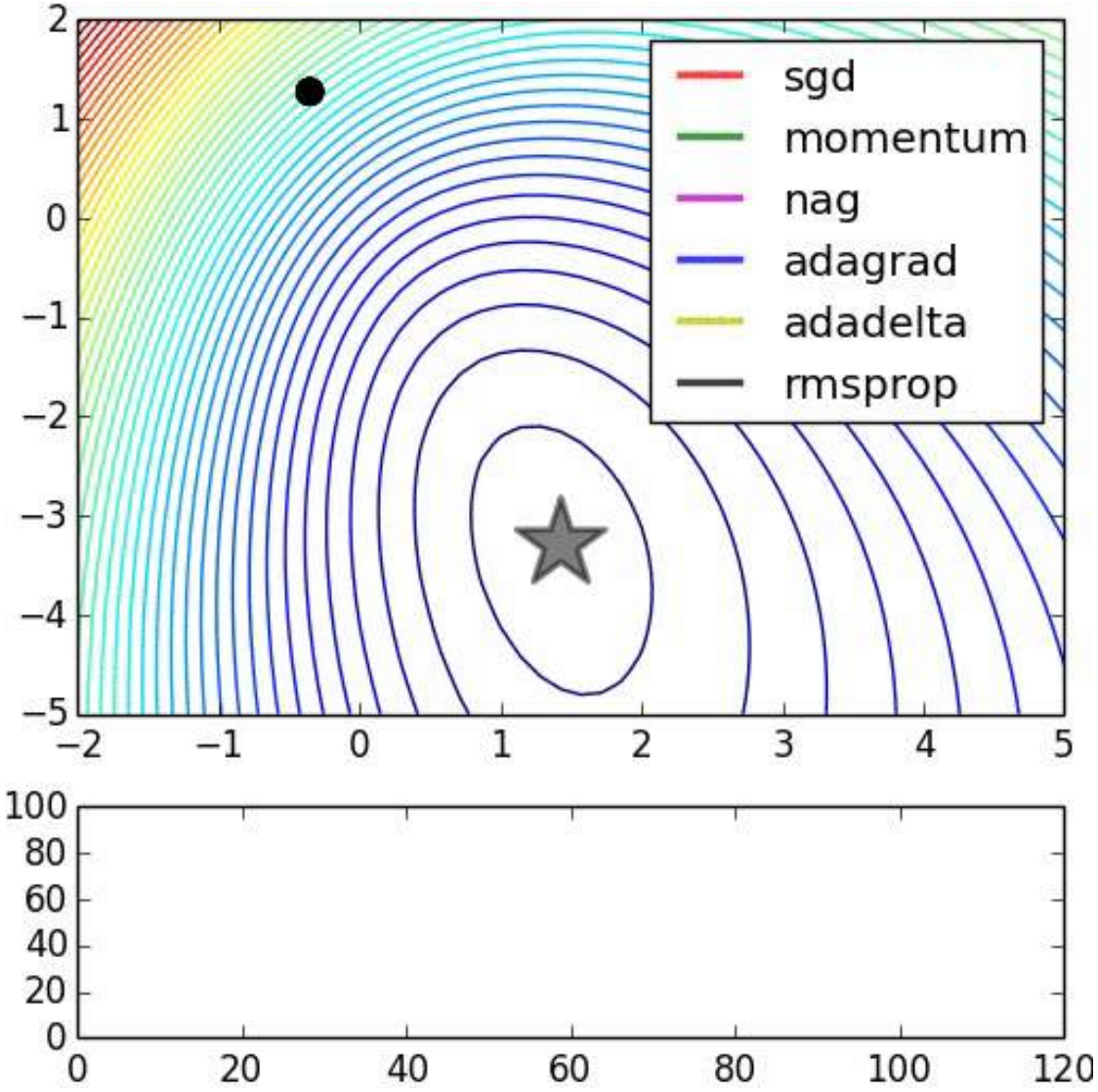


1. Все производные – либо веса между нейронами, либо производные нелинейностей, т.е. производные будут пропорциональны весам по пути к вершине. **Не стоит брать все веса равными (особенно равными нулю)**
2. Инициализация из нормального распределения может приводить к затуханию или взрыву градиента (об этом позже) – но **есть эвристики для предотвращения эффекта***

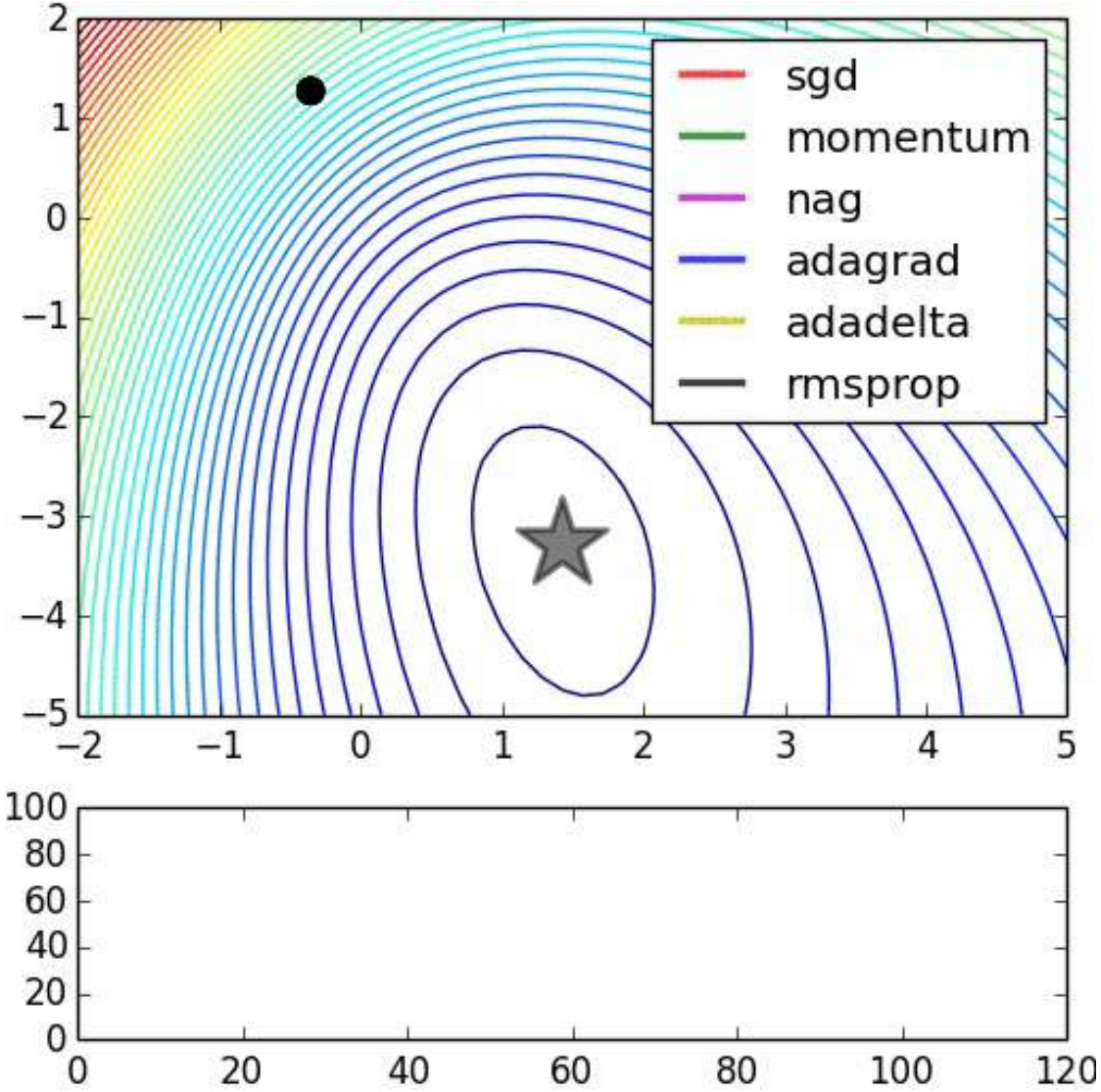
* https://medium.com/usf-msds/deep-learning-best-practices-1-weight-initialization-14e5c0295b94

4. Методы оптимизации

Модификации SGD



Модификации SGD



Momentum:
Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

NAG (Nesterov Accelerated Gradient):
Модификация Momentum: антиградиент берем в той точки, куда шагнули бы по инерции

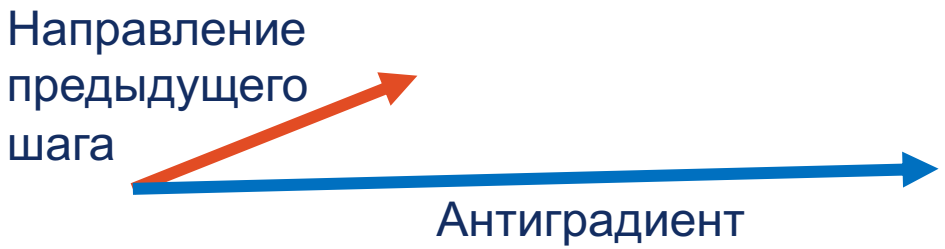
Adagrad, Adadelta, RMSprop, Adam:
Добавлены эвристики для настройки разного шага по разным координатам (весам)

Модификации SGD

Momentum:
Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

An overview of gradient descent optimization algorithms

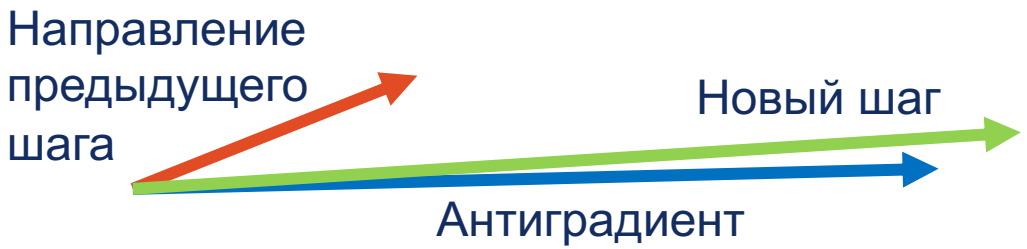
Модификации SGD



Momentum:
Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

[An overview of gradient descent optimization algorithms]

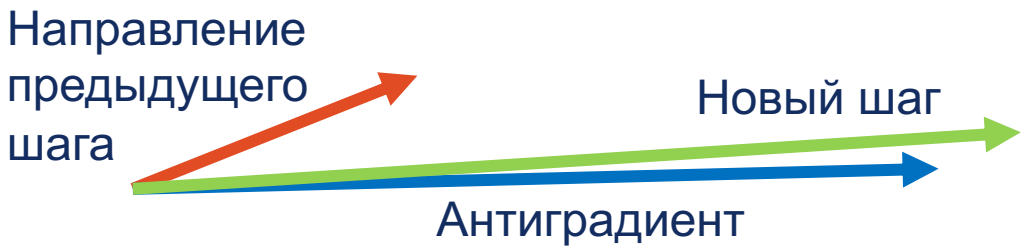
Модификации SGD



Momentum:
Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

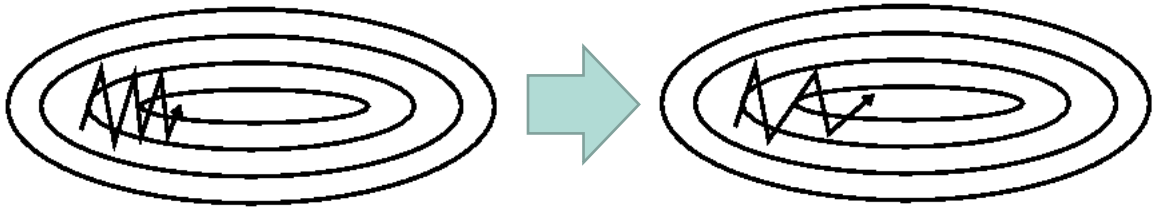
[An overview of gradient descent optimization algorithms]

Модификации SGD



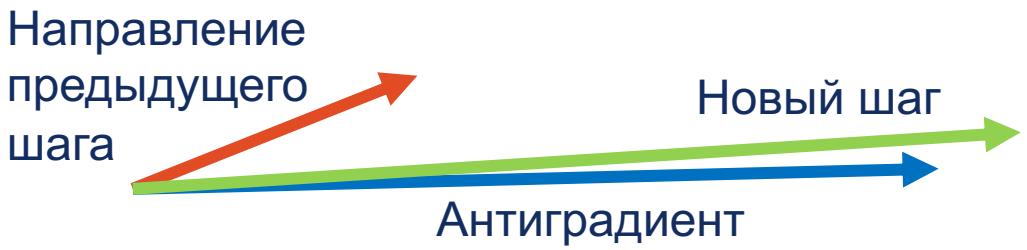
Momentum:
Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

Идея Momentum:



[An overview of gradient descent optimization algorithms]

Модификации SGD

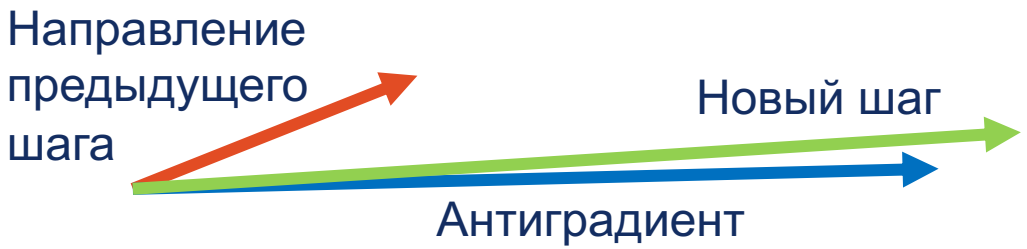


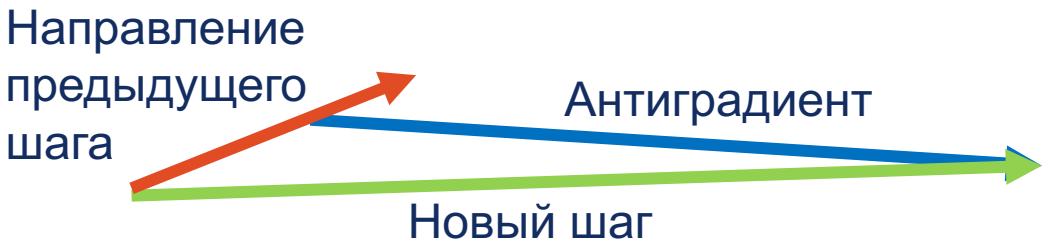
Momentum:
Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

NAG (Nesterov Accelerated Gradient):
Модификация Momentum: антиградиент берем в той точке, куда шагнули бы по инерции

An overview of gradient descent optimization algorithms

Модификации SGD



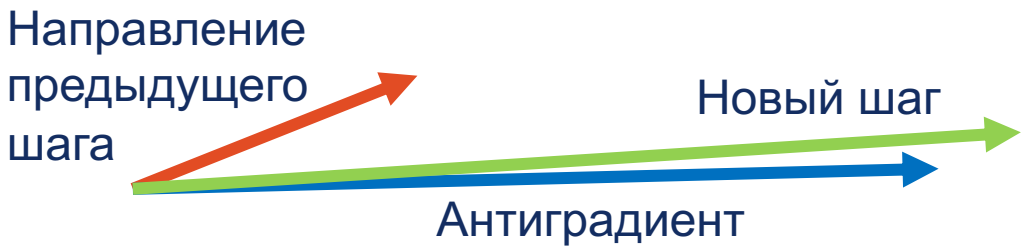


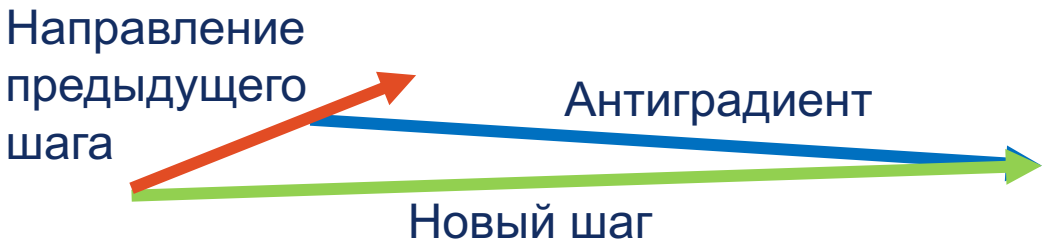
Momentum:
Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

NAG (Nesterov Accelerated Gradient):
Модификация Momentum: антиградиент берем в той точки, куда шагнули бы по инерции

[An overview of gradient descent optimization algorithms]

Модификации SGD





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

Momentum:
Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

NAG (Nesterov Accelerated Gradient):
Модификация Momentum: антиградиент берем в той точки, куда шагнули бы по инерции

Adagrad, Adadelta, RMSprop, Adam:
Добавлены эвристики для настройки разного шага по разным координатам (весам)

An overview of gradient descent optimization algorithms

План

1. Что такое нейронные сети
2. Обучение: backpropagation
3. Регуляризация и другие трюки
4. Методы оптимизации

Data Mining in Action

Лекция 9

Группа курса в Telegram:

https://t.me/joinchat/B1OITk74nRV56Dp1TDJGNA