Лекция 4. Оптимизация. Регуляризация

Денис Деркач, Дмитрий Тарасов

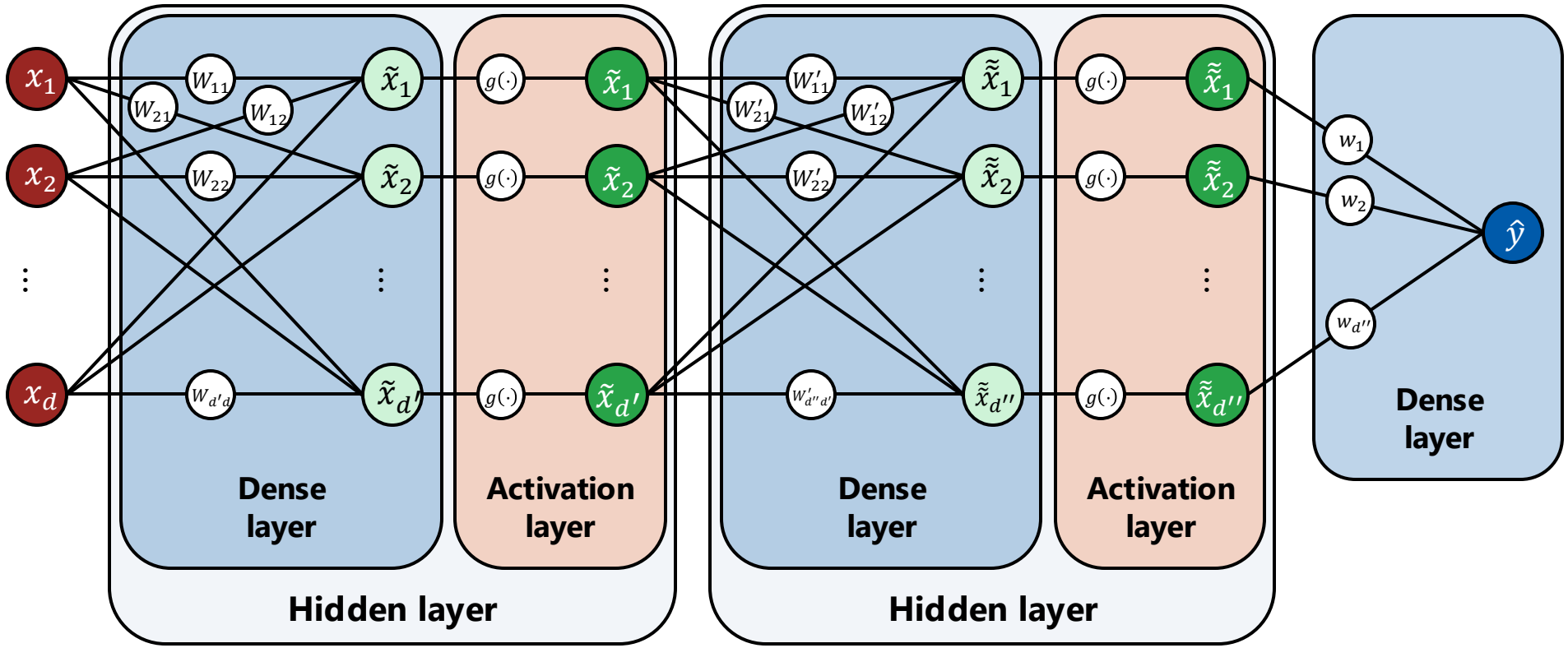
Слайды от А. Маевского, М. Гущина, А. Кленицкого, М Борисяка

LAMBDA • HSE

20 января 2025 года

Оптимизация

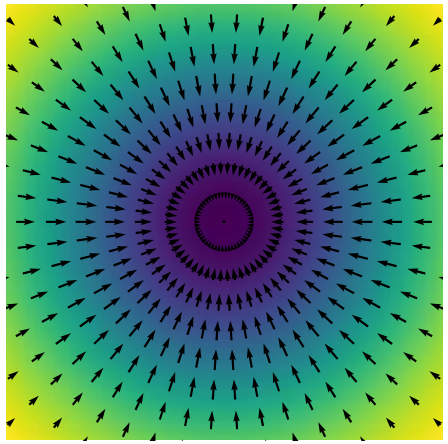
Нейронная сеть как функция

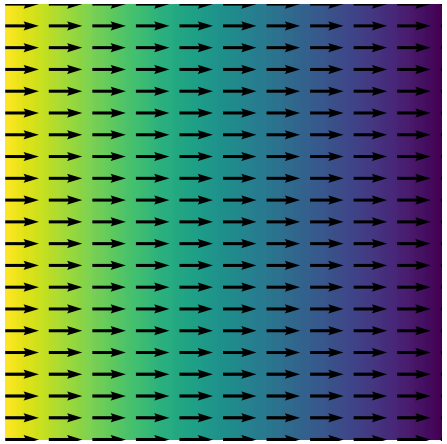


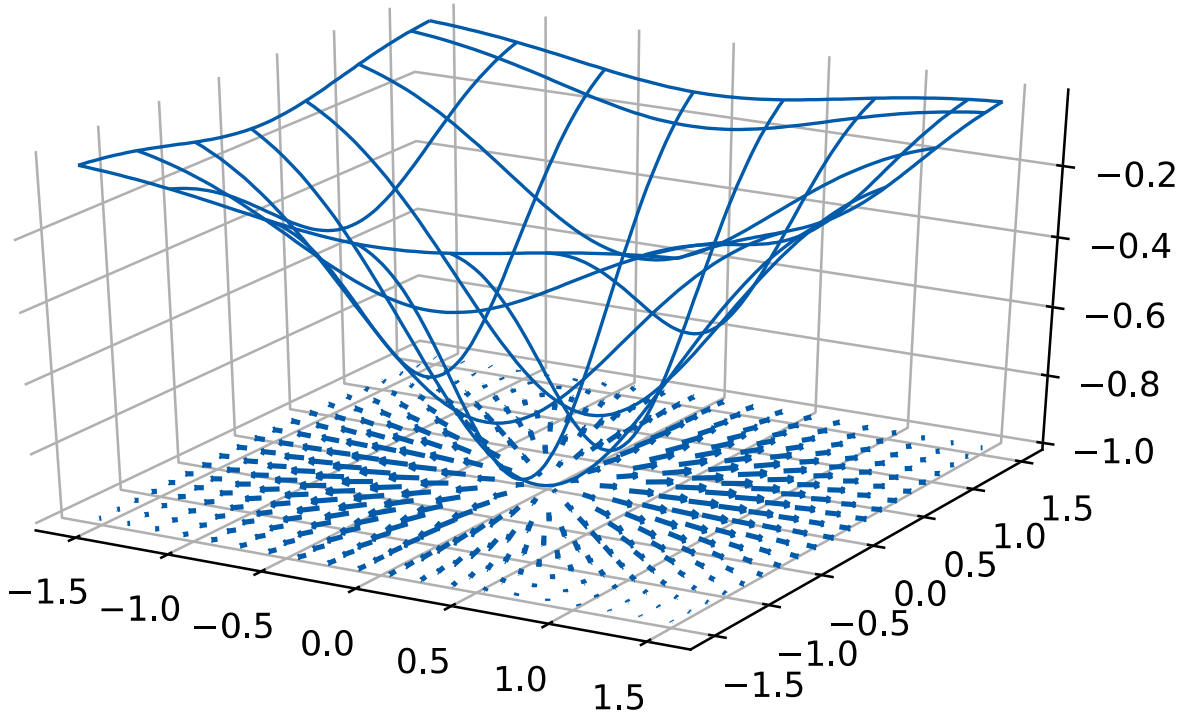
$$\hat{y} = W_{out} \cdot \ldots \cdot W_{h2} \cdot W_{h1} x$$

Градиент

- Градиент: $\nabla_x f(x) \equiv \left(\frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_d}\right)$
- Указывает на самый быстрый рост функции







Оптимизация градиентного спуска

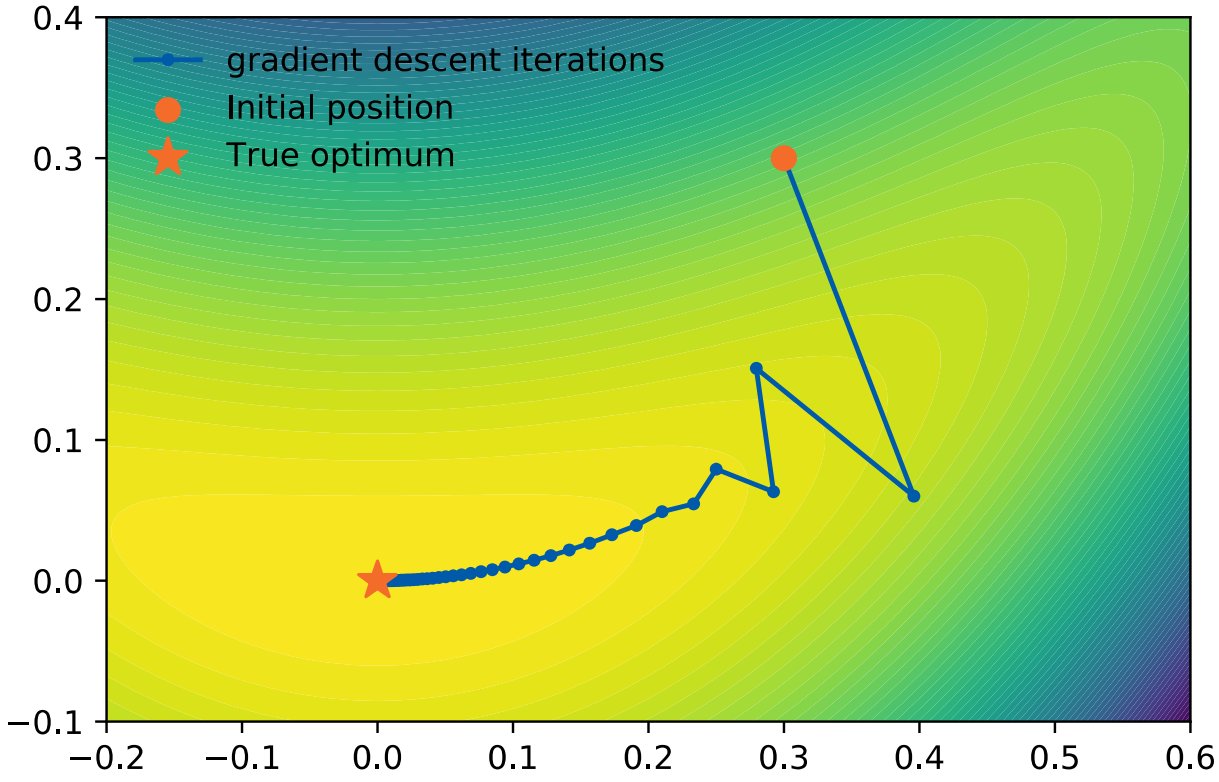
- Можно оптимизировать функции, начиная с некоторой начальной точки $x^{(0)}$ и двигаясь против градиента

$$x^{(k)} \longleftarrow x^{(k-1)} - \alpha \nabla_x f\left(x^{(k-1)}\right)$$

где $\alpha \in \mathbb{R},\ \alpha > 0$ – **темп обучения** (learning rate).

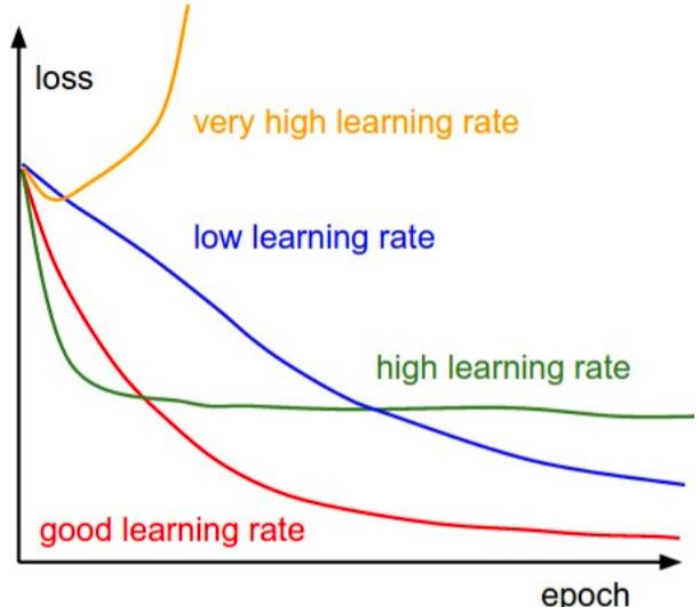
- Для гладких выпуклых функций с единственным минимумом x^*:

$$f\left(x^{(k)}\right) - f(x^*) = \mathcal{O}\left(\frac{1}{k}\right)$$

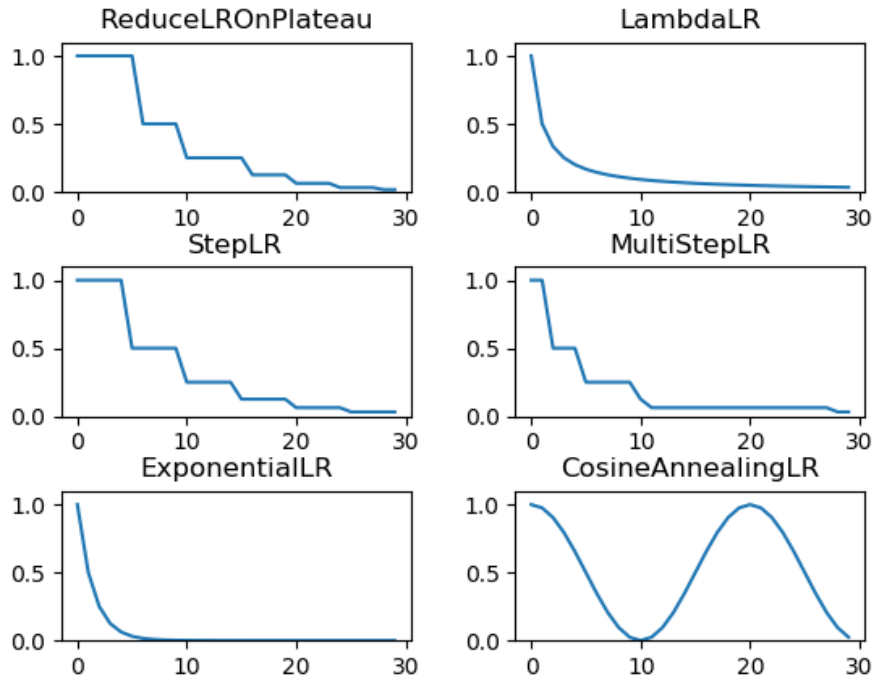


Сценарии

- Важно правильно настроить learning rate и придумать расписание

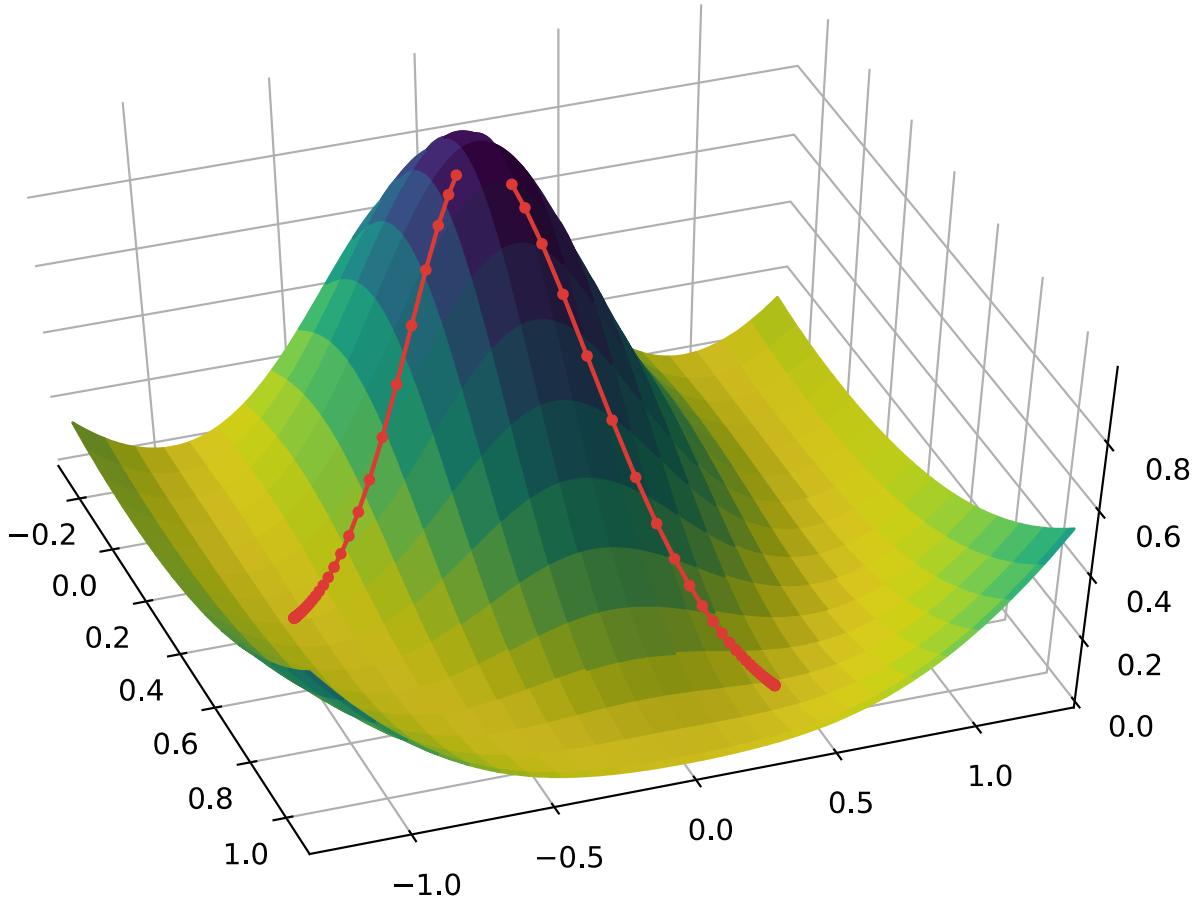


- Можно уменьшать learning rate со временем по расписанию
  - В начале хотим активно исследовать пространство параметров, чтобы найти хорошую область
  - В конце хотим более детально исследовать найденную область
  - Используем ReduceLROnPlateau для зависимости от ошибок на валидации.



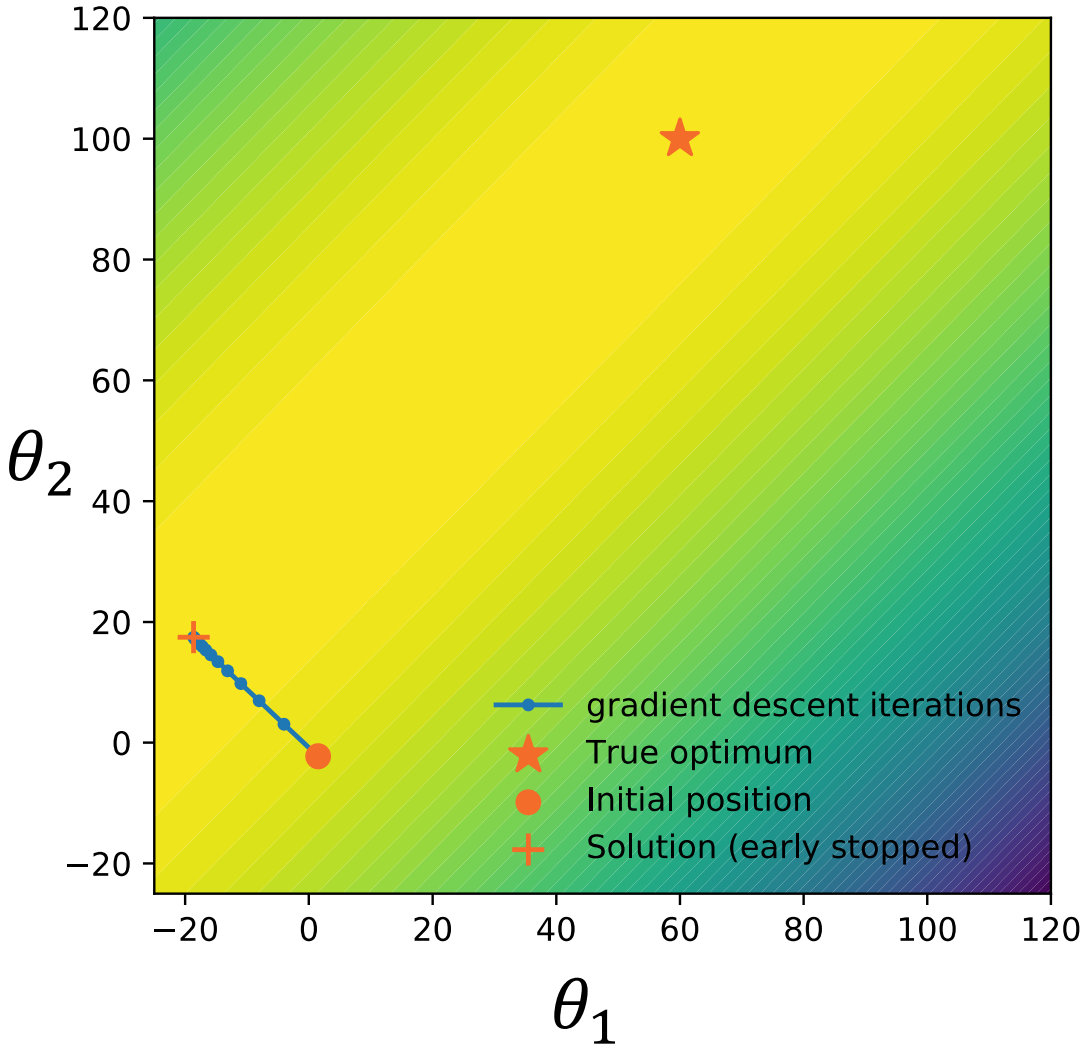
https://greeksharifa.github.io/pytorch/2018/11/10/pytorch-usage-03-How-to-Use-PyTorch/

Градиентный спуск для невыпуклых функций



- Может достичь минимума, который не является глобальным
- Результат зависит от начальной точки

Градиентный спуск как средство регуляризации



- Большие значения параметров обычно означают переобучение
- Можно избежать этой проблемы, инициализируя параметры малыми значениями и останавливая градиентный спуск на ранней стадии (early stopping)

Стохастический градиентный спуск (SGD)

- В машинном обучении мы оптимизируем функции потерь, которые обычно являются средними по объектам:

$$L = \frac{1}{N} \sum_{i = 1 \ldots N} \mathcal{L}\left(y_i, \widehat{f_\theta}(x_i)\right)$$

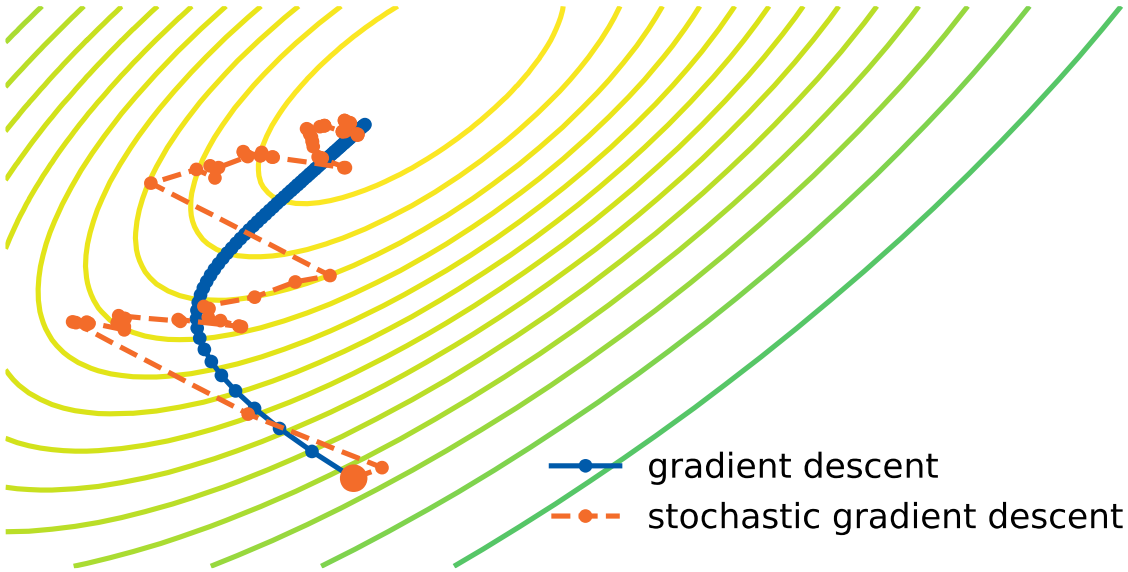
- Для больших N, градиентный спуск неэффективен с точки зрения вычислений и может быть неосуществим с точки зрения потребления памяти
- Альтернатива:
  - На каждом шаге k взять случайный $l_k \in \{1, \ldots, N\}$
  - Оптимизировать: $\theta^{(k)} \leftarrow \theta^{(k-1)} - \alpha \nabla_\theta \mathcal{L}\left(y_{l_k}, \widehat{f_\theta}(x_{l_k})\right)\Big|_{\theta = \theta^{(k-1)}}$

SGD и мини-батчи

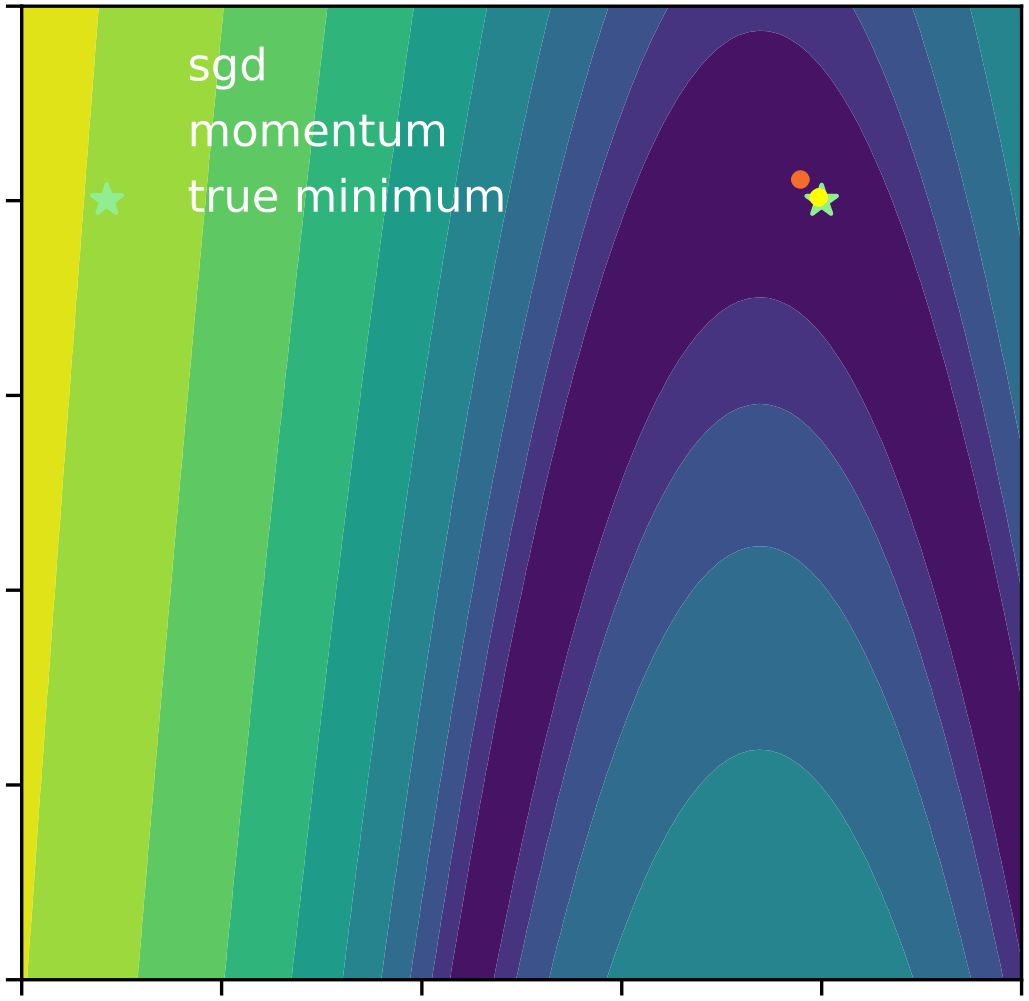
- SGD:
  - На каждом шаге k взять случайный $l_k \in \{1, \dots, N\}$, тогда изменение:
  - $\theta^{(k)} \leftarrow \theta^{(k-1)} - \eta \nabla_\theta \mathcal{L}\left(y_{l_k}, \hat{f}_\theta(x_{l_k})\right)\Big|_{\theta = \theta^{(k-1)}}$
- Мини-батч SGD:
  - Перемешаем обучающий НД, затем итерируем по нему пакетами (батчами) фиксированного размера.
  - На каждой итерации оцениваем градиенты потерь на данном участке.

$$B: g = \sum_{i \in B} \nabla_\theta \mathcal{L}\left(y_i, \hat{f}_\theta(x_i)\right) \quad \theta^{(k)} \leftarrow \theta^{(k-1)} - \eta \cdot g$$

  - Модифицируем параметры:



Импульс (momentum) SGD



- Идея: ввести инерцию (как у мяча, катящегося с горы)
  - Помогает выйти из небольших локальных минимумов
  - Позволяет использовать более широкий диапазон скоростей обучения*

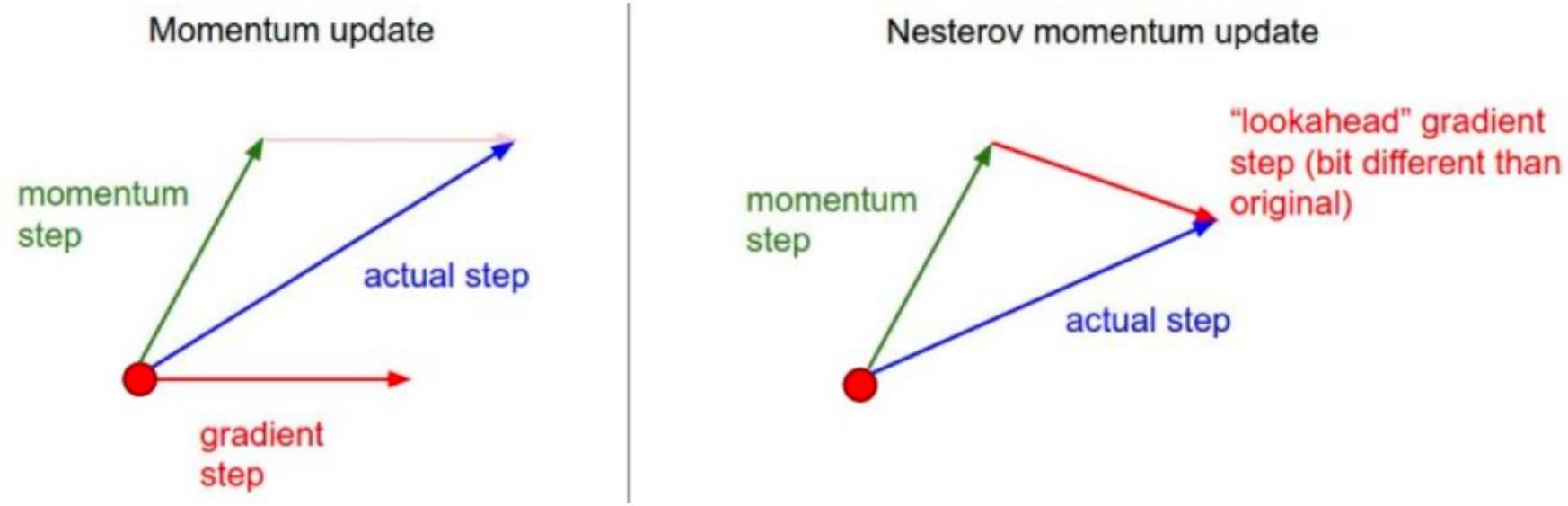
$$m^{(k)} \leftarrow \beta \cdot m^{(k-1)} + (1-\beta) \cdot \left.\frac{\partial L}{\partial \theta}\right|_{\theta=\theta^{(k-1)}}$$

$$\theta^{(k)} \leftarrow \theta^{(k-1)} - \eta \cdot m^{(k)}$$

  - Фактически, экспоненциальное сглаживание.

* https://distill.pub/2017/momentum/

Метод Нестерова



- На самом деле мы уже знаем, что сдвинемся на β_{mt-1} на промежуточном шаге
- Можем вычислить градиент в этой точке, на полпути

$$m_t = \beta m_{t-1} + (1-\beta)\nabla_\theta L(\theta_t - \beta m_{t-1})$$

AdaGrad

- **Идея**: давайте быстрее двигаться по тем параметрам, которые не сильно меняются, и медленнее по быстро меняющимся параметрам:

- Обозначим $g_t = \nabla_\theta L(\theta_t)$

- Накапливаем историю градиентов

$$v_t = \sum_{\tau=1}^{t} g_t^2$$

и учитываем ее

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} g_t$$

- Learning rate все время уменьшается, но с разной скоростью для разных параметров

RMSprop

- Идея: настроить скорость обучения отдельно для разных компонентов вектора параметров, избегая **проблемы** Adagrad с исчезающим градиентом
- Считать скользящее среднее

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2$$

- обновлять параметры с его использованием

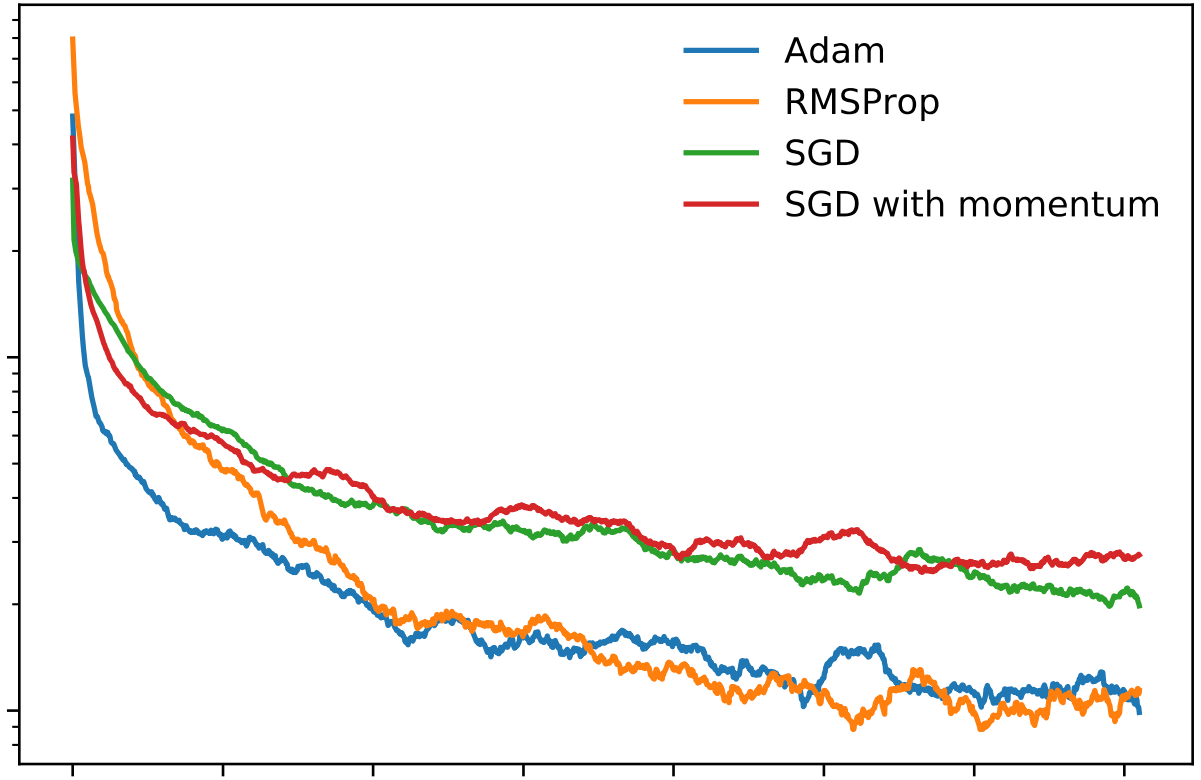
$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} g_t$$

Adam

- Скомбинируем идеи (momentum + RMSprop)
- Обычно хороший алгоритм для начала оптимизации

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} m_t$$



Schedule-free optimization

$$
\begin{aligned}
y_t &= (1-\beta)z_t + \beta x_t, \\
z_{t+1} &= z_t - \gamma \nabla f(y_t, \zeta_t), \\
x_{t+1} &= (1-c_{t+1})\, x_t + c_{t+1} z_{t+1}, \\
c_{t+1} &= \frac{1}{t+1}
\end{aligned}
$$

x - последовательность, в которой должны происходить оценки потерь test/val, которая отличается от первичных итераций z и мест оценки градиента y. Обновления в z соответствуют базовому оптимизатору, в данном случае простому градиентному спуску / SGD.

Полученный алгоритм не должен иметь расписания уменьшения LR, при этом ведёт себя оптимальнее других.

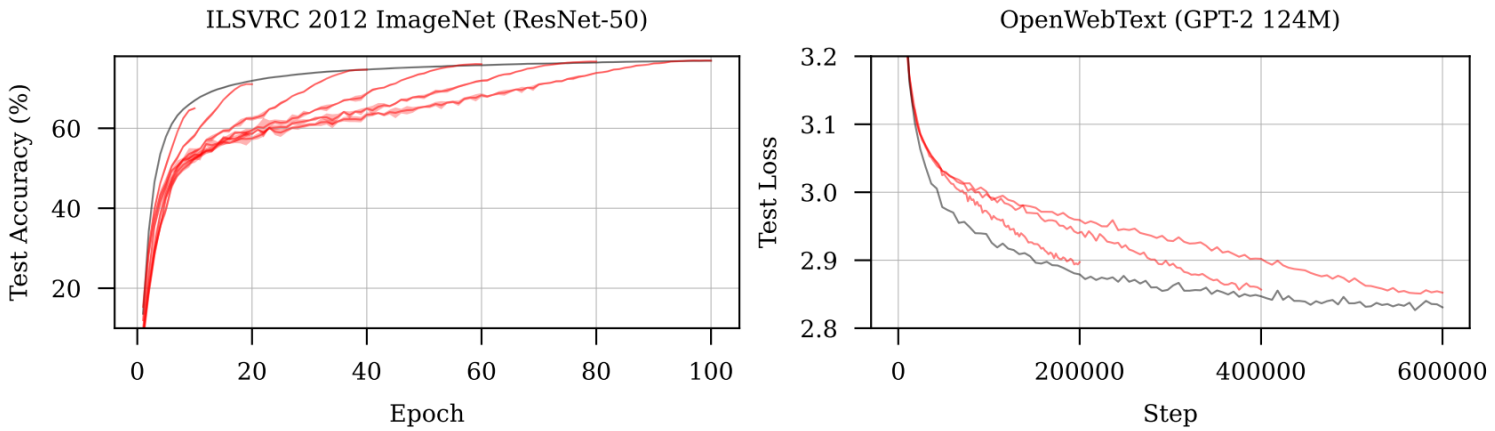


Figure 1: Schedule-Free methods (black) closely track the Pareto frontier of loss v.s. training time in a single run. Both Schedule-Free SGD (left) and AdamW (right) match or exceed the performance of cosine learning rate schedules of varying lengths (red).

https://arxiv.org/pdf/2405.15682

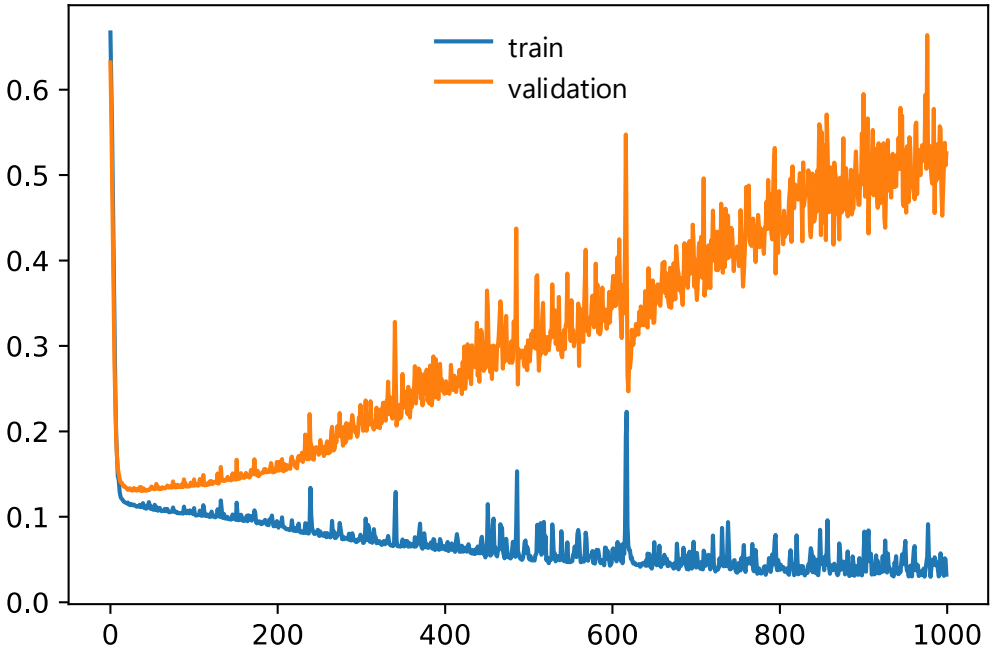
Саммари

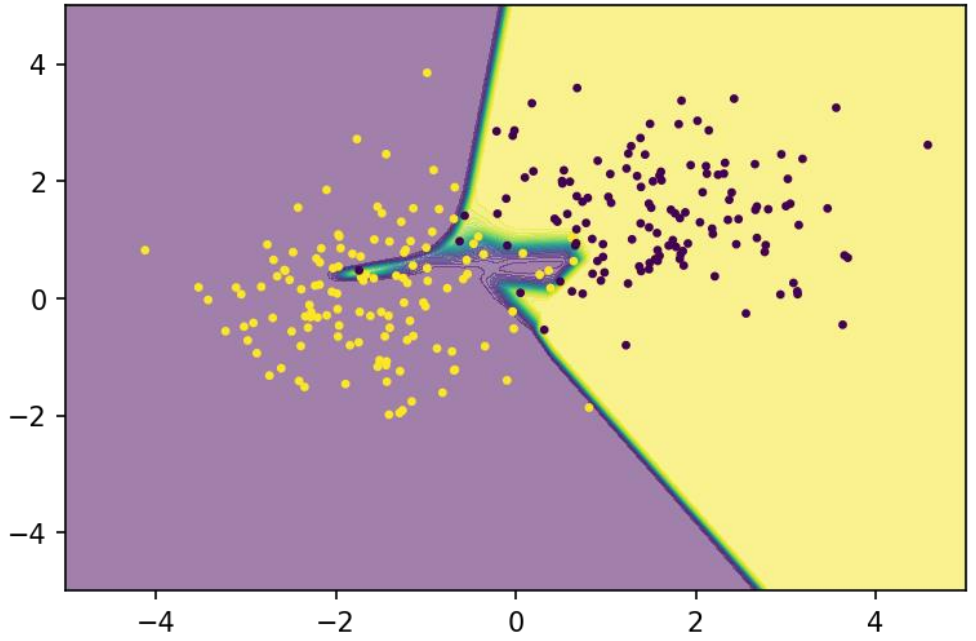
- Множество разнообразных алгоритмов оптимизации
- Хороший старт с Adam

Переобучение

Проблема переобучения

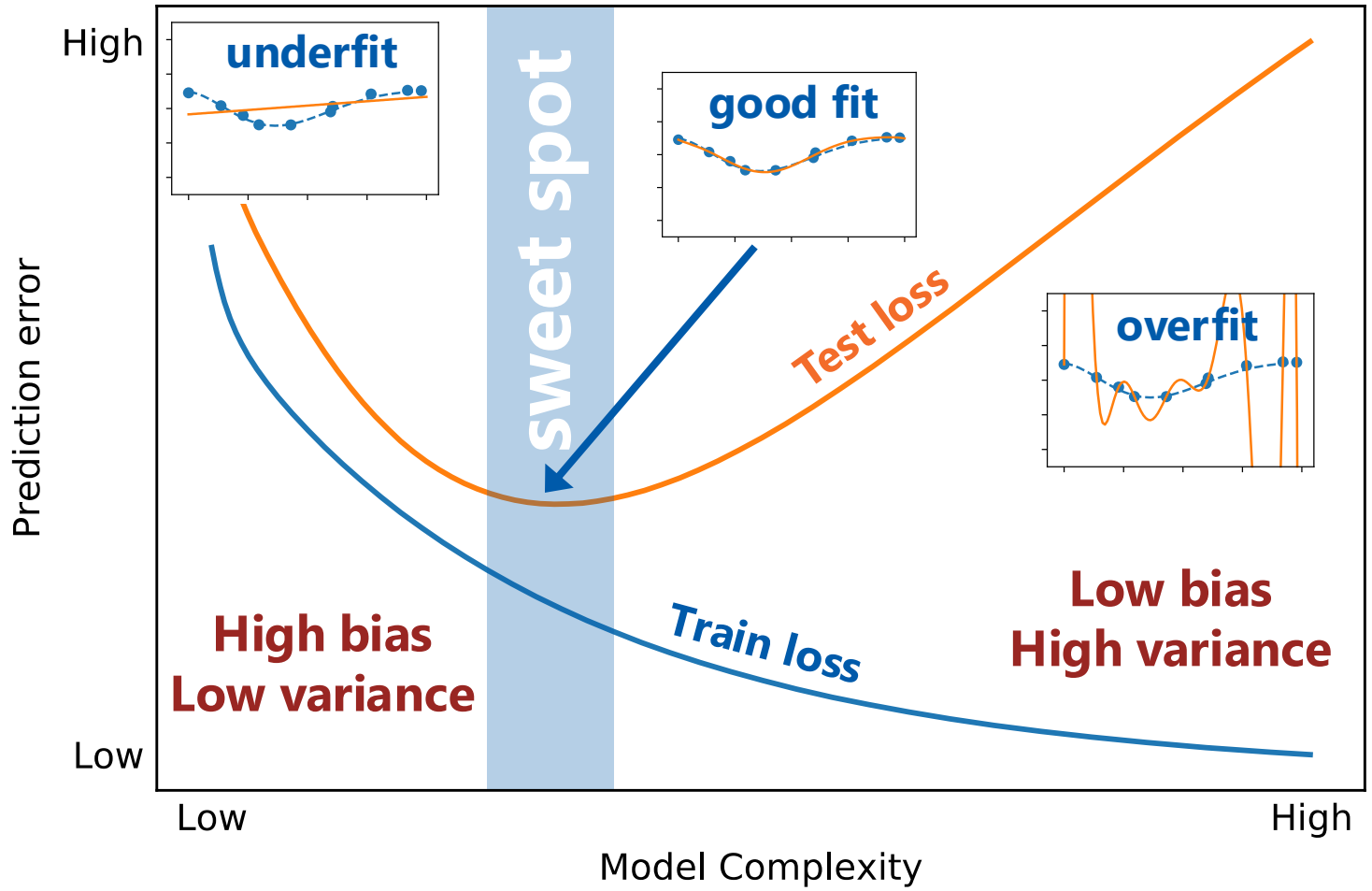
- Будучи очень сложными моделями, нейронные сети склонны к переобучению





Bias-variance

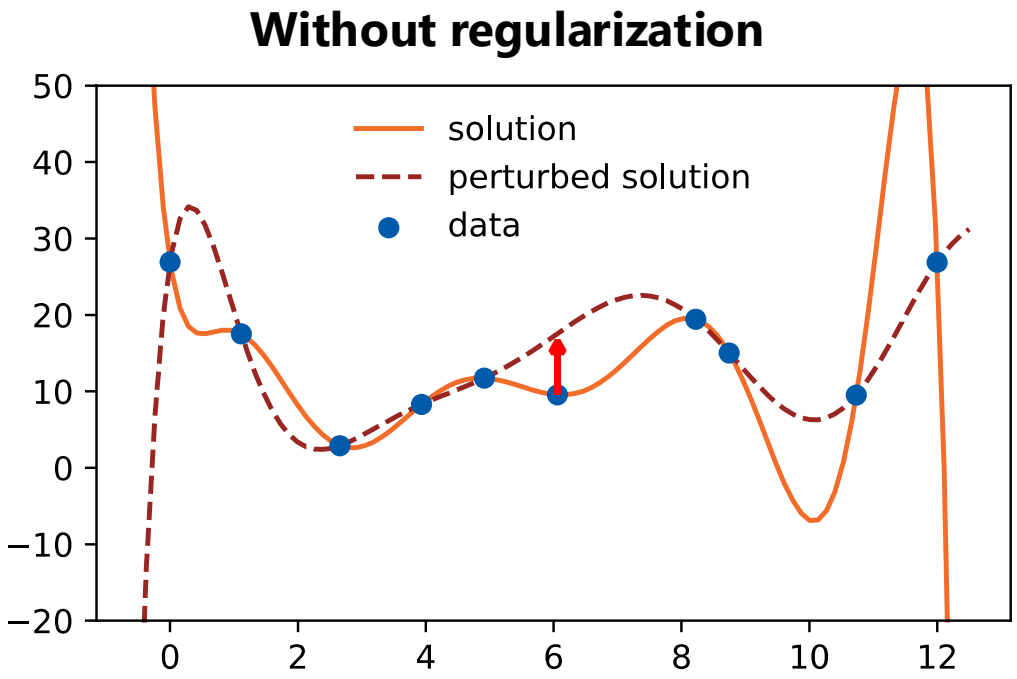
- Будучи очень сложными моделями, нейронные сети склонны к переобучению

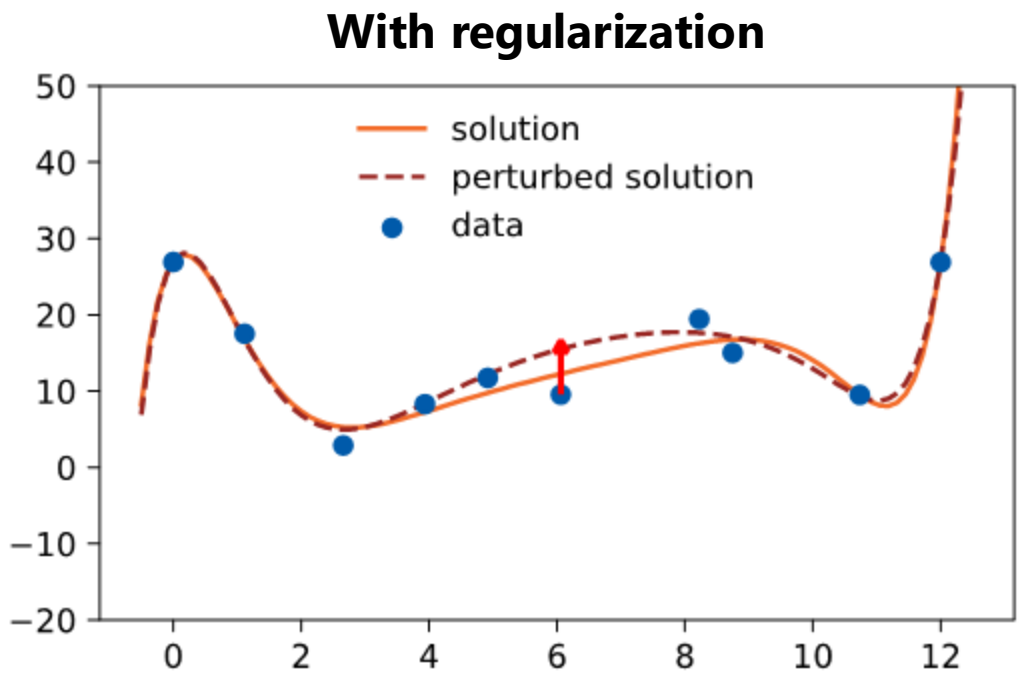


Обычно есть выбор между смещением и дсиперсией.

Bias-Variance trade-off

Регуляризация





NB: регуляризованная модель больше не является несмещённой!

То есть мы увеличили смещение, чтобы уменьшить дисперсию.

Разные методы регуляризации

L2 regularization (Ridge):

$$\mathcal{L} = \|X\theta_\tau - y_\tau\|^2 + \alpha\|\theta_\tau\|^2$$

L2 norm:

$$\|x\|^2 \equiv \sum_{i=1\ldots d} x_i^2$$

L1 regularization (Lasso):

$$\mathcal{L} = \|X\theta_\tau - y_\tau\|^2 + \alpha\|\theta_\tau\|_1$$

L1 norm:

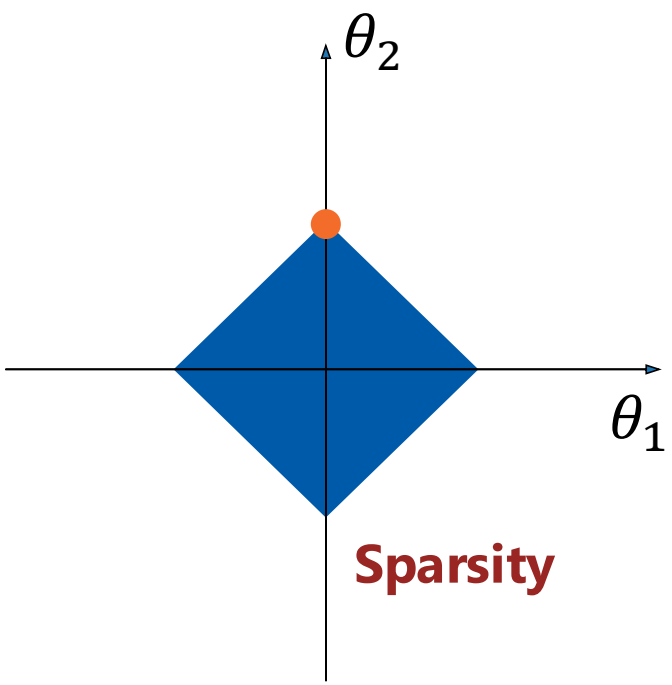
$$\|x\|_1 \equiv \sum_{i=1\ldots d} |x_i|$$

Elastic net:

$$\mathcal{L} = \|X\theta_\tau - y_\tau\|^2 + \alpha\|\theta_\tau\|^2 + \beta\|\theta_\tau\|_1$$

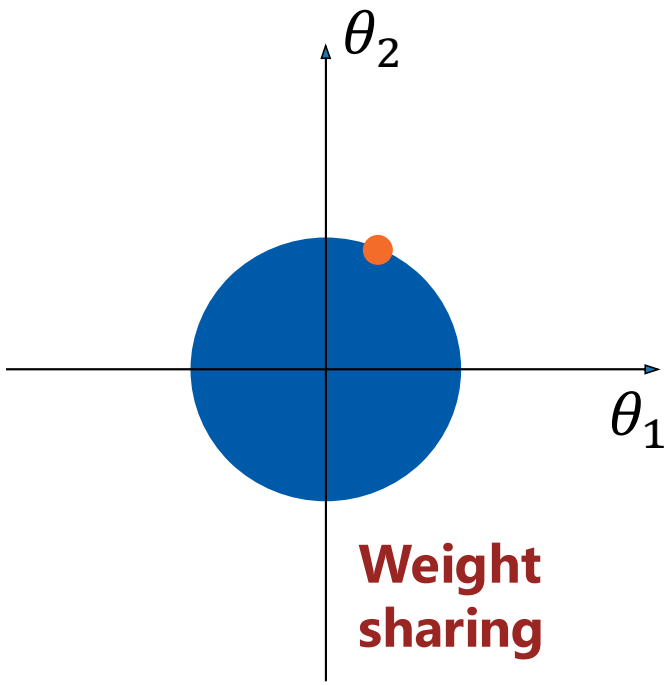
Свойства регуляризации

L1



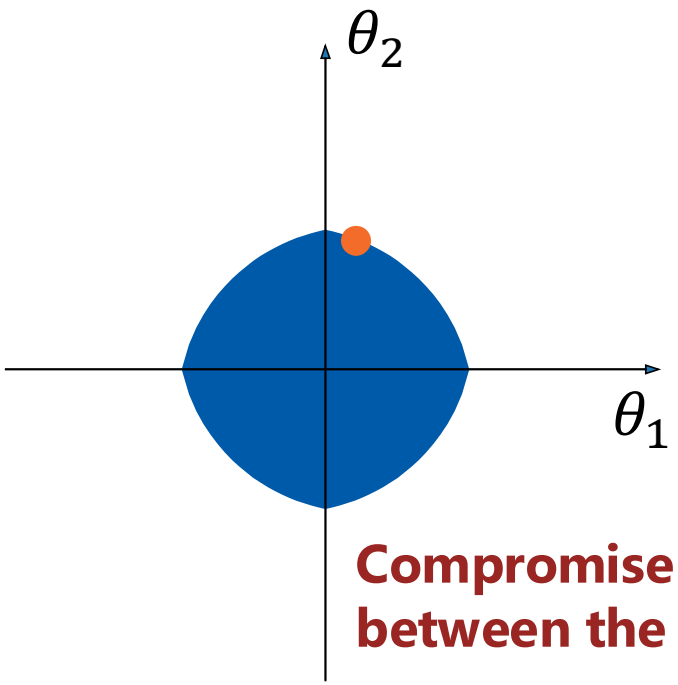
Sparsity

L2



Weight sharing

Elastic Net

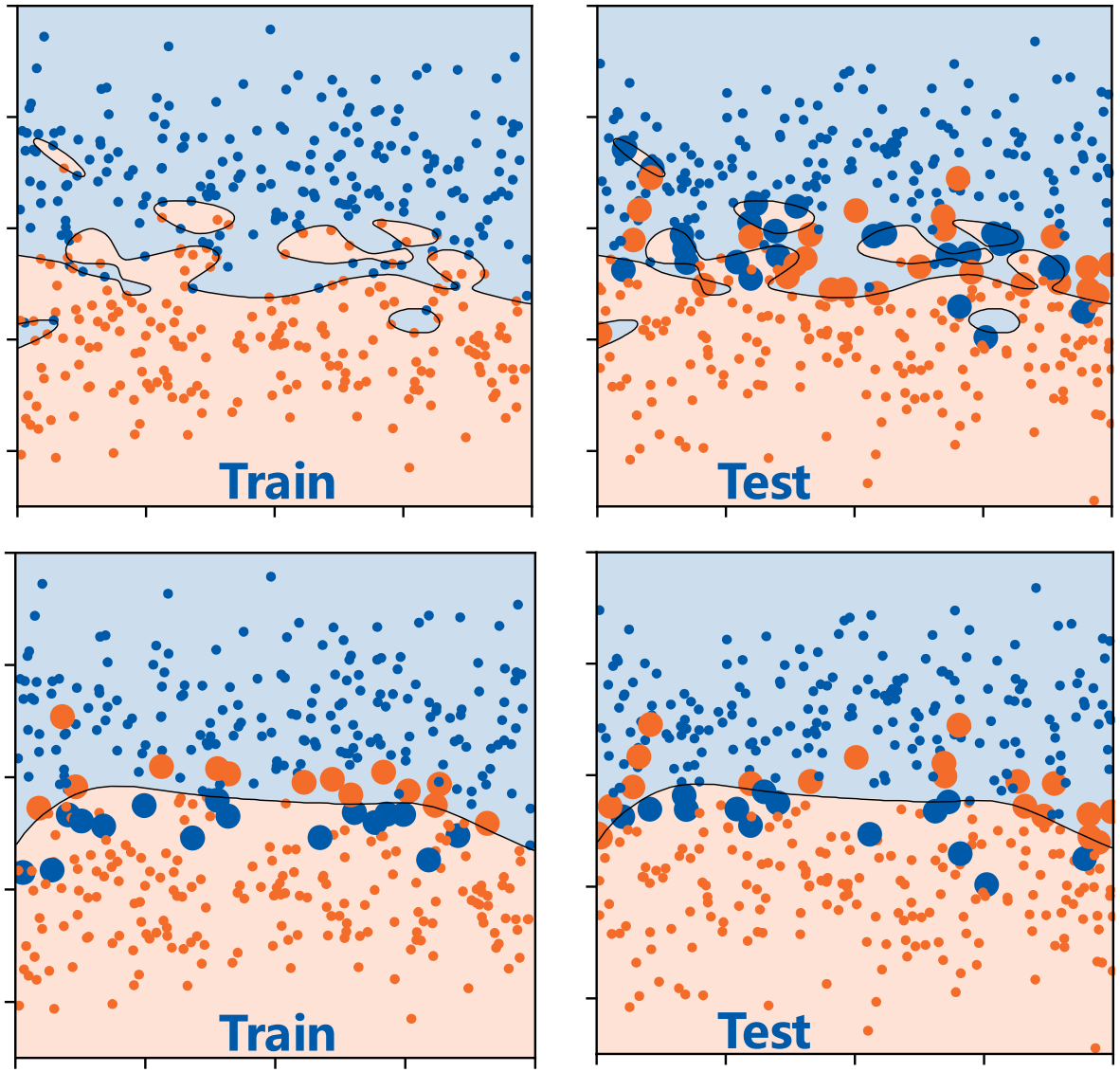


Compromise between the two

Все они смещают вес в сторону меньших значений.

Однако они вызывают различные свойства решения.

Пример: L2-регуляризованная классификация



Without regularization

Регуляризируя модель, мы увеличиваем функцию потерь в обучении и уменьшаем потери при тестировании.

With regularization

Это улучшает обобщаемость модели.

Регуляризация и подходы

MSE loss ⇔ Prob. model with normal label noise!

Частотный подход:
Регуляризация эквивалентна предположению в модели о распределении данных

L2 регуляризация это разрешение модели учитывать нормальный шум

Normal prior ⇔ L2 regularization

Байесовский подход:
Регуляризация эквивалентна априорному предположению о распределении данных

Байесовские нейронные сети = регуляризация

L2 регуляризация это априорное нормальное распределение

Аугментация данных

Что если брать регуляризацию из данных, зная некоторые свойства

https://www.baeldung.com/cs/ml-data-augmentation

Примеры аугментаций:

- Для изображений - flip, rotation, crop, rescale,...
- Для звука - добавляем фоновый шум, меняем высоту звука
- Для текстов - замена на синонимы, backtranslation
- Для последовательностей - пропускаем элементы, меняем местами

Зашумление данных

Добавляем шум в данные

- на входе
- зашумление весов
- на выходе - зашумляем метки

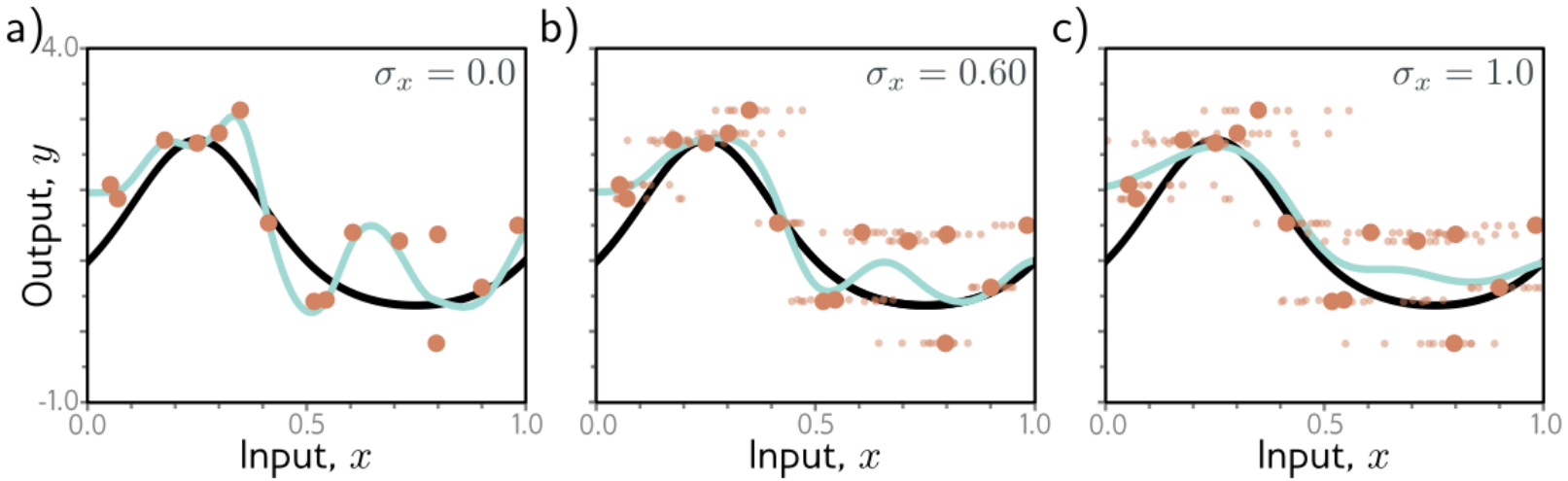
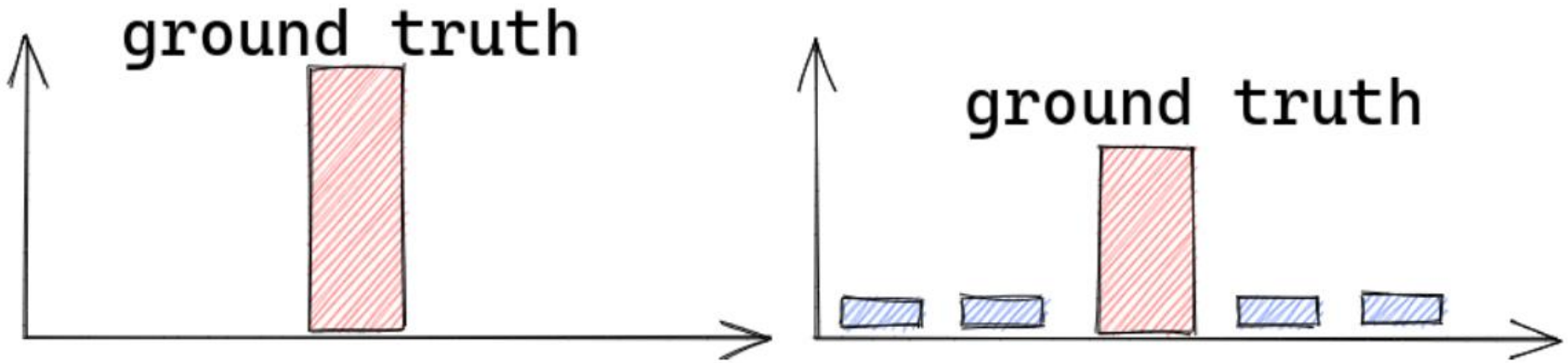


Image credit

Зашумление меток

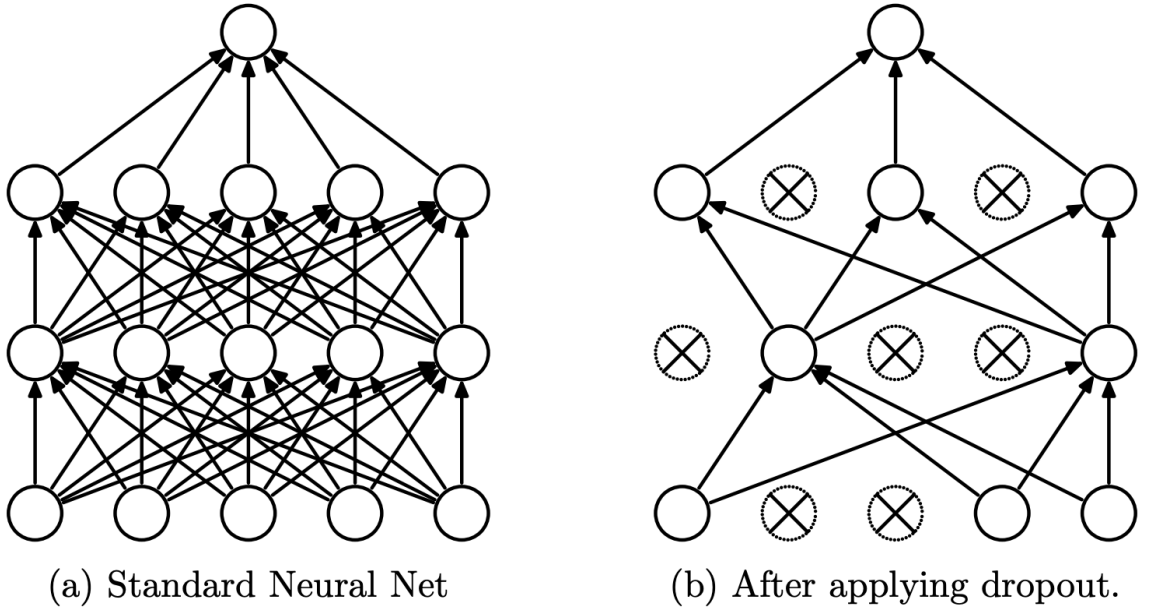


Вероятность правильной метки $1 - \epsilon$, всех остальных - $\epsilon/(K-1)$

Dropout

- При обучении – ставим активацию нейронов 0 с вероятностью p
- При тестировании – умножаем на $1 - p$
  - то есть приравниваем матожиданию
- Заставляет нейрон учиться работать со случайно выбранной выборкой других нейронов
- Заставляет его создавать полезные признаки, а не полагаться на другие нейроны для исправления своих ошибок.

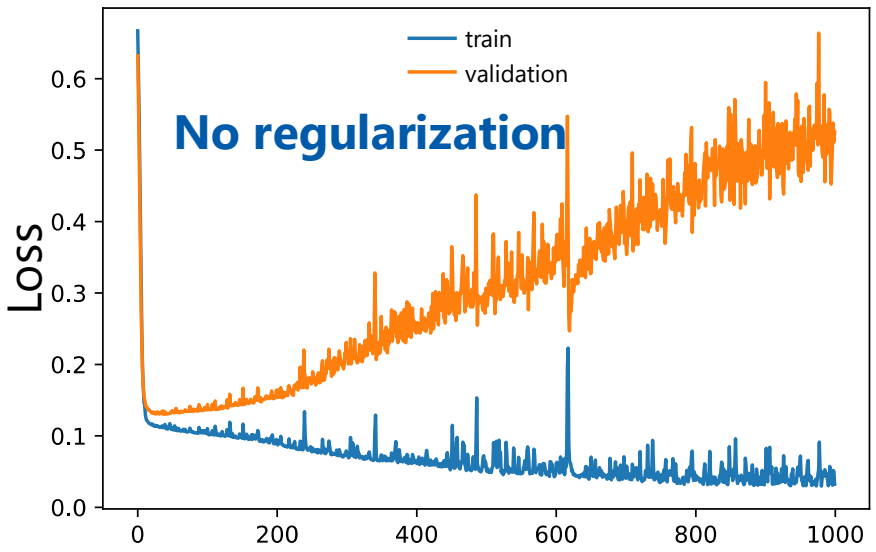
Image from:
http://jmlr.org/papers/v15/srivastava14a.html

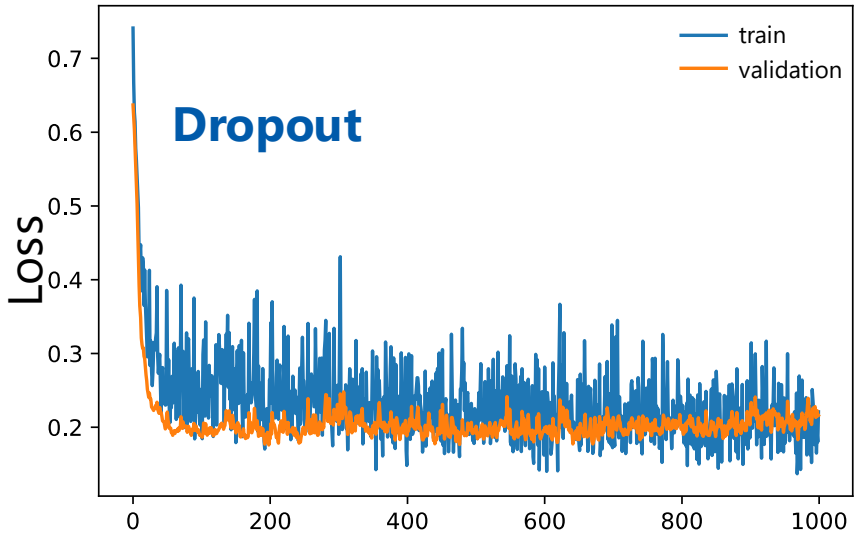


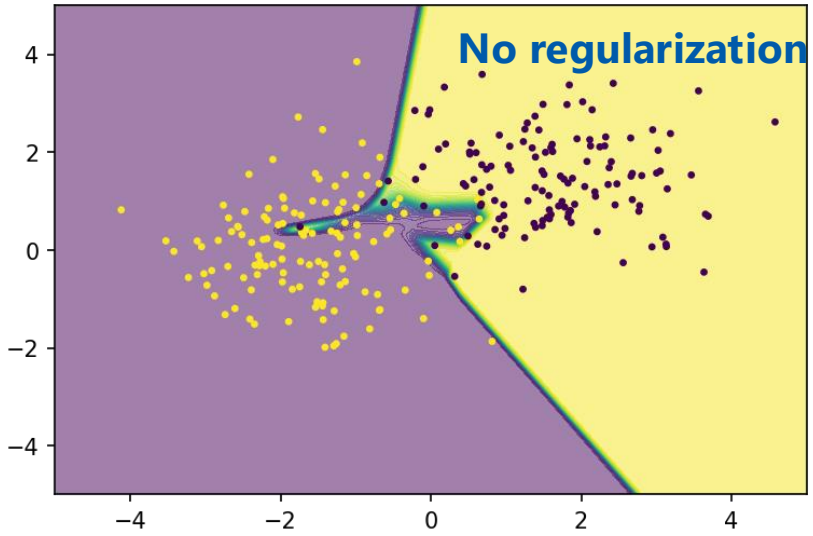
(a) Standard Neural Net

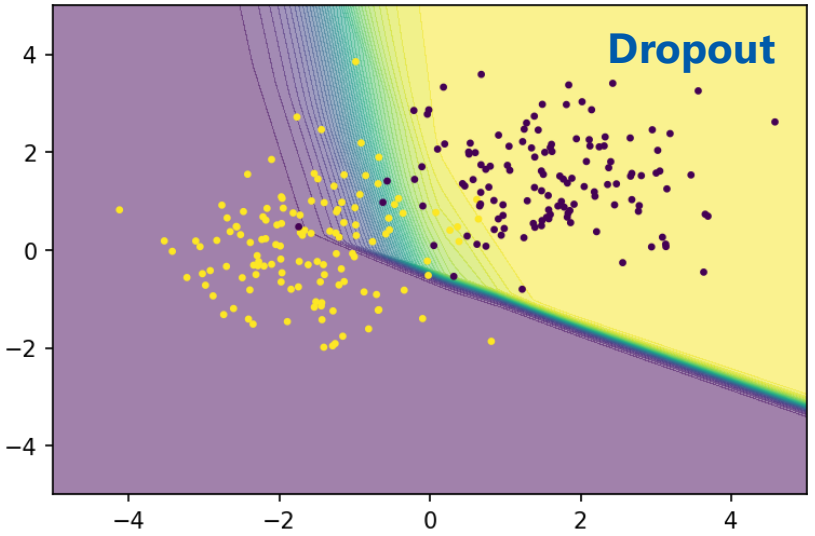
(b) After applying dropout.

Пример прореживания (dropout)









В этом примере dropout приводит к гораздо лучшему (хотя все еще не идеальному) соответствию с меньшей ошибкой теста.

Normalization layers

Пакетная нормализация (Batch normalization)

- Первоначально этот метод был предложен для смягчения «внутреннего ковариационного сдвига»
- Работает следующим образом (входы слоя x_i, выходы y_i):
  - рассчитать выборочное среднее значение и дисперсию входных данных для одного батча B

$$\mu_B = \frac{1}{|B|}\sum_{i \in B} x_i \qquad \sigma_B^2 = \frac{1}{|B|}\sum_{i \in B}(x_i - \mu_B)^2$$

  - нормализуем входные данные, затем масштабируем и сдвигаем (с обучаемыми параметрами (γ, β):

$$y_i = \gamma \cdot \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

internal covariate shift

обновления в одном слое изменяют входные распределения последующих слоев

Batch normalization

- Turned out to be **extremely powerful** in many cases
  - Faster and more stable convergence
- Later was proved to **not** reduce the internal covariate shift

internal covariate shift

the updates in one layer change the input distributions of the subsequent layers

- Effectively **removes** the ‘shift’ and ‘scale’ degrees of freedom from the previous layer

$$y_i = \gamma \cdot \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

Batch normalization

- Which dimension to normalize over? Typically like this:
  - Batch of 1D vectors [Batch_dim x Features_dim]
    - separately for each component in Features_dim, i.e. over Batch_dim
  - Batch of ND objects [Batch_dim x Spacial_dim1 x … x Channel_dim]
    - separately for each component in Channel_dim, i.e. over Batch_dim x Spacial_dim1 x …

Batch normalization

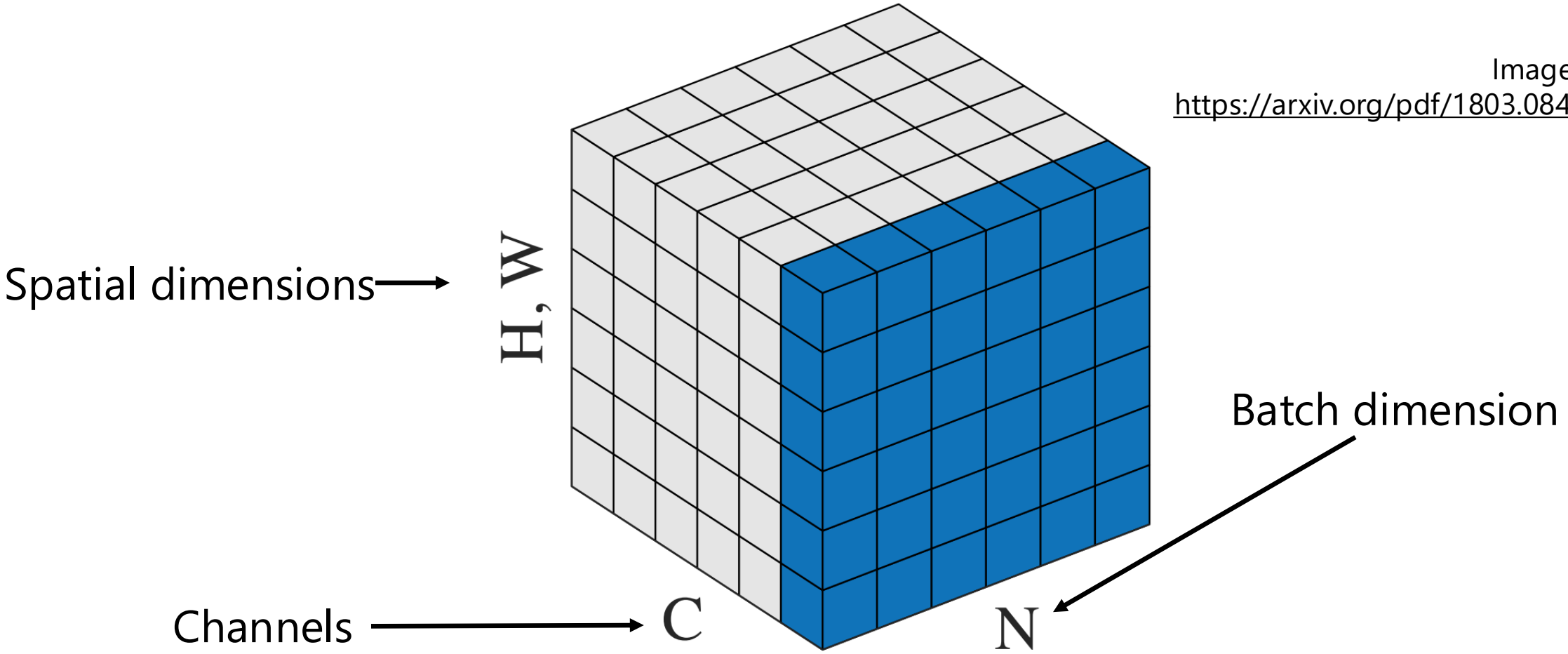
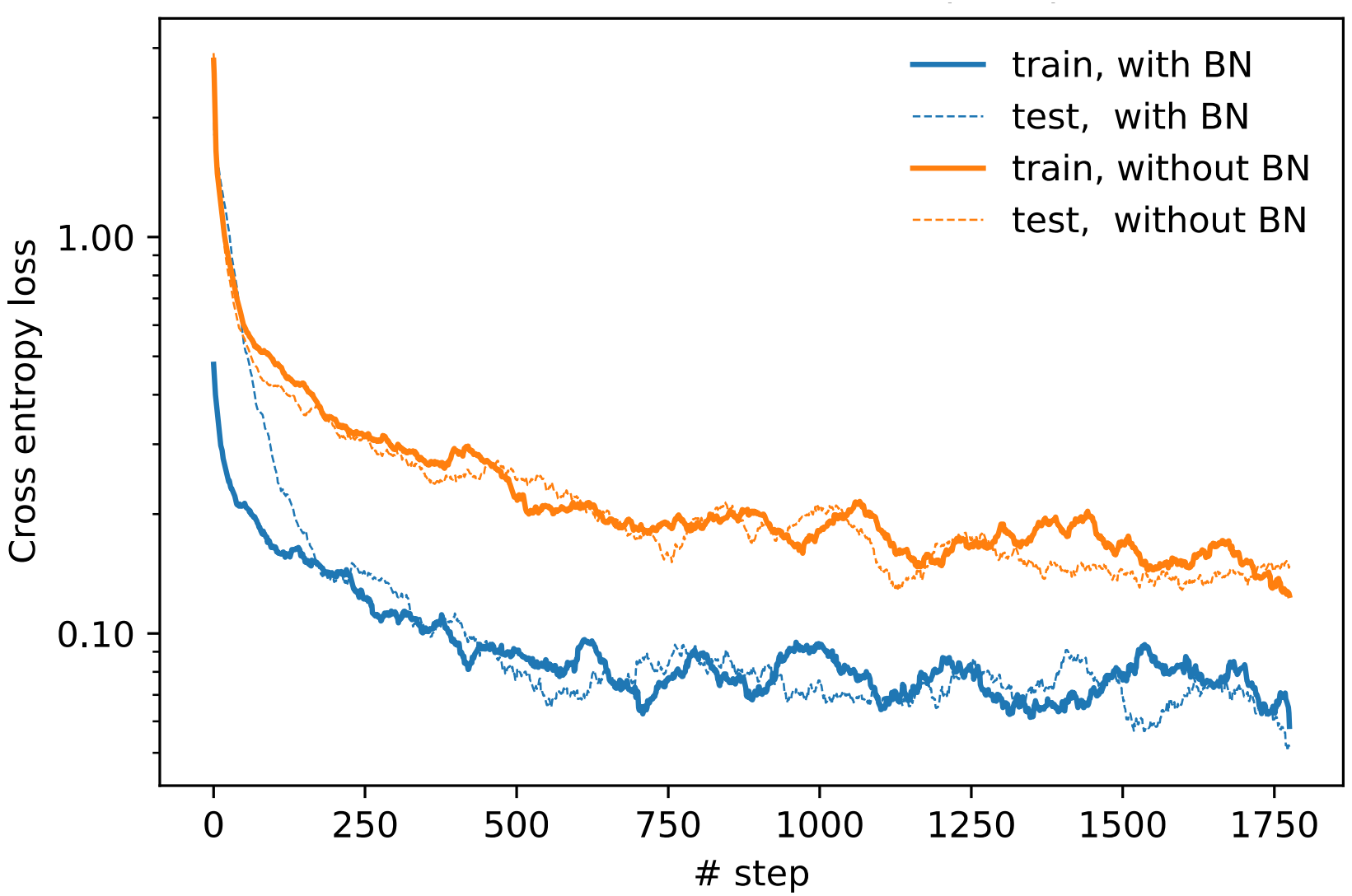


Image from https://arxiv.org/pdf/1803.08494.pdf

Batch normalization at inference time

- Calculating batch statistics at test time may be problematic
  - e.g. when there's a single object to predict
- Instead: calculate running mean and variance during training, apply at test time

Example: CNN on MNIST



(shown: moving average loss)

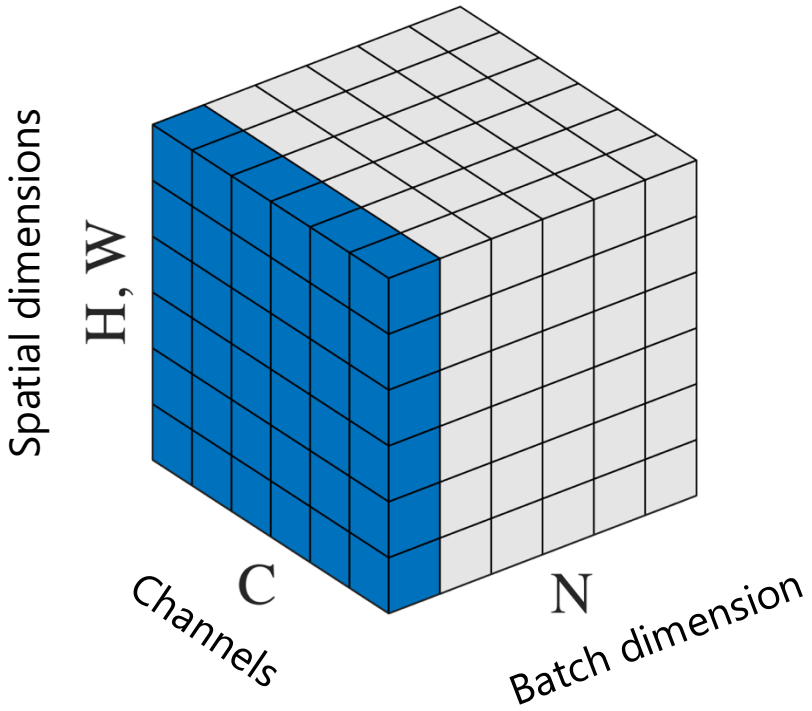
Layer Normalization

- Batch normalization imposes limits on the batch size
  - if too small, the variance of the sample statistics will be too high
- Problematic to use in recurrent networks

Layer Normalization

- Batch normalization imposes limits on the batch size
  - if too small, the variance of the sample statistics will be too high
- Problematic to use in recurrent networks
- Alternative: **Layer Normalization**
  - the math is same, except statistics is calculated over channels rather than batch elements
  - the effect is quite different though
    - e.g. Layer Normalization "entangles" different neurons within a layer

Image from
https://arxiv.org/pdf/1803.08494.pdf



Summary

- Neural networks can be regularized with L1/L2 penalties or early stopping
- Dropout makes neurons **create useful features** rather than rely on other neurons to correct their mistakes
- Batch normalization is an **extremely powerful** regularization technique, though the reason for that is not entirely clear