Лекция 4. Оптимизация. Регуляризация.

Глубинное обучение

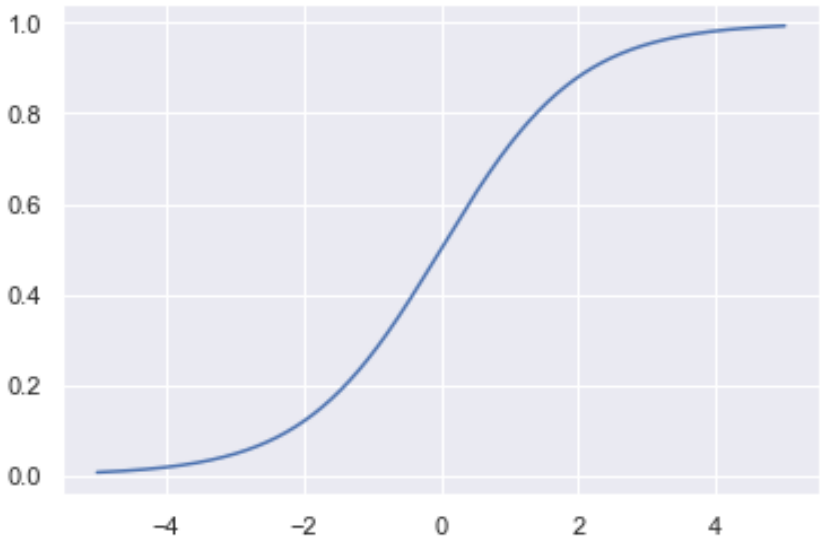
Антон Кленицкий

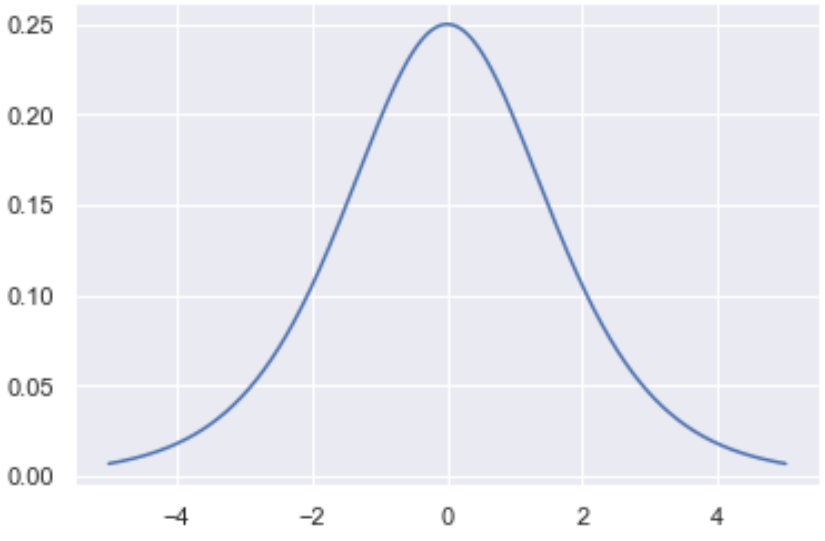
Recap

Функции активации - Sigmoid

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

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$





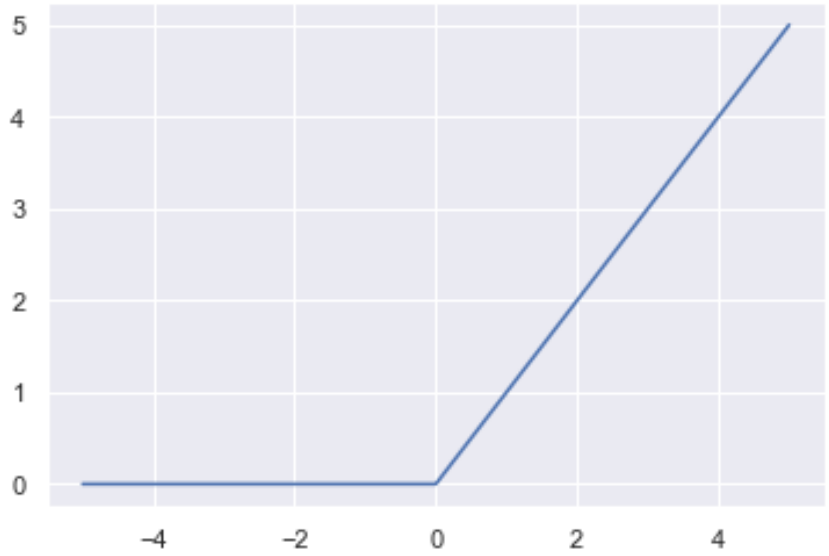
- Насыщение на краях, градиент близок к нулю
- Максимальное значение производной 0.25
- Выход не центрирован вокруг нуля

Функции активации - ReLU

Rectified Linear Unit

$$f(x) = \max(0, x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

$$f'(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$



Плюсы

- Нет насыщения, ненулевые градиенты
- Вычислительно очень дешево

Минус

- “Dead neurons” - нейроны, для которых всегда $x < 0$

Функции активации - выводы

Модификации ReLU

- LeakyReLU
- PReLU
- ELU
- GELU
- Swish (SiLU)

Выводы

- Избегать сигмоиды
- ReLU - хороший выбор в большинстве случаев
- Модификации ReLU могут чуть-чуть улучшить качество
- Подбор функции активации - не первый приоритет

Инициализация весов

На каждом слое умножаем на матрицу весов и при прямом проходе, и при обратном проходе:

$$f_k = W_k h_{k-1} + b_k$$

$$\frac{\partial f_k}{\partial h_{k-1}} = W_k^T$$

Большие значения весов

- Выходы последних слоев становятся слишком большими
- exploding gradients

Маленькие значения весов

- Выходы последних слоев становятся слишком маленькими
- vanishing gradients

Xavier (Glorot) initialization

Understanding the difficulty of training deep feedforward neural networks - Glorot, X. & Bengio, Y. (2010)

- Для симметричных функций активации
- Идея - дисперсии выходов и градиентов на всех слоях должны быть одинаковыми

$$Var[w_i] = \frac{2}{n_{in} + n_{out}}$$

$$w_i \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}\right]$$

(Kaiming) He initialization

Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification - He, K. et al. (2015)

- Для несимметричных функций активации (ReLU)
- Идея - дисперсии выходов или градиентов на всех слоях должны быть одинаковыми

$$Var[w_i] = \frac{2}{n_{in}}$$

$$w_i \sim N(0, \sqrt{2/n_{in}})$$

или

$$w_i \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in}}}, \frac{\sqrt{6}}{\sqrt{n_{in}}}\right]$$

Maximum likelihood

Модель $f(x, \theta)$ определяет параметры распределения вероятности $P(y|x)$:

$$P(y|x) = P(y|f(x, \theta))$$

Метод максимального правдоподобия (maximum likelihood estimation):

$$\theta = \underset{\theta}{\operatorname{argmax}}\, L(\theta) = \underset{\theta}{\operatorname{argmax}} \left[\prod_{i=1}^{N} P(y_i|f(x_i, \theta)) \right]$$

Negative log-likelihood loss:

$$\mathcal{L} = -\sum_{i=1}^{N} \log\left(P(y_i|f(x_i, \theta))\right)$$

Оптимизация

Stochastic gradient descent

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\theta_t)$$

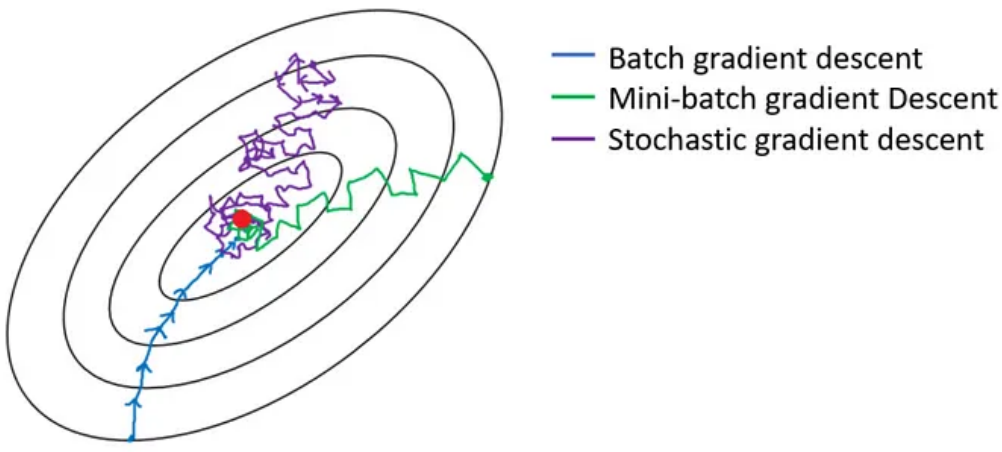


Image credit

Learning rate

Важно правильно настроить learning rate

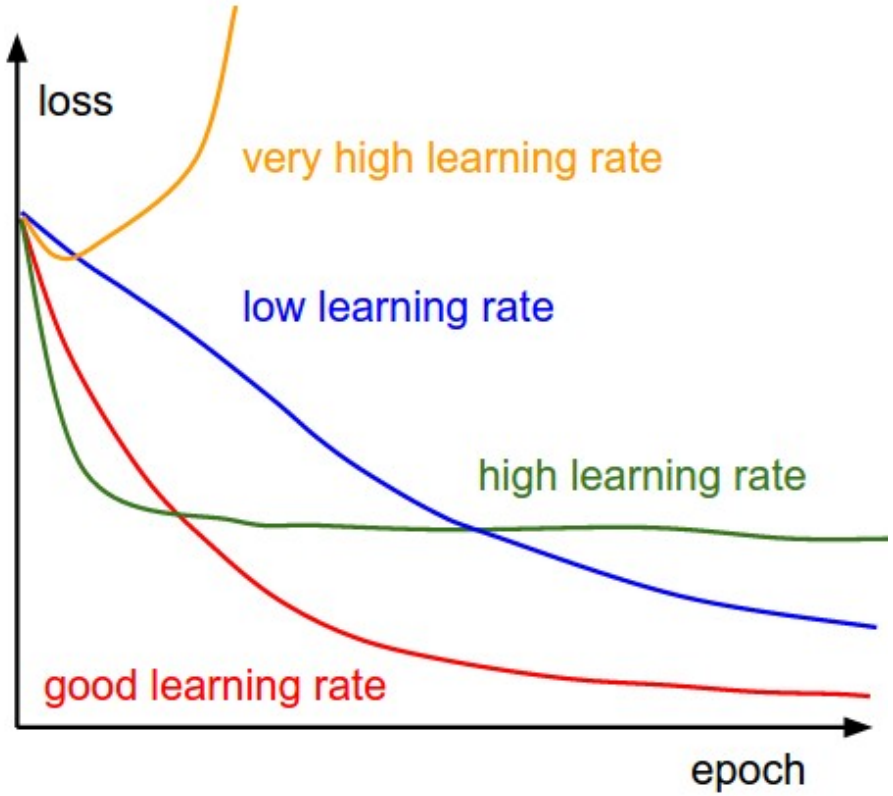


Image credit

Learning rate schedulers

Можно уменьшать learning rate со временем по расписанию

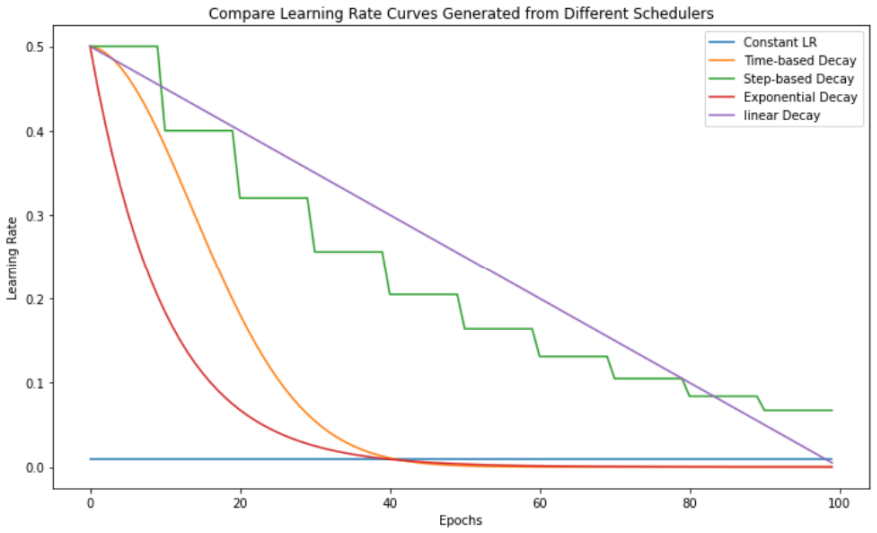


Image credit

- В начале хотим активно исследовать пространство параметров, чтобы найти хорошую область
- В конце хотим более детально исследовать найденную область

Learning rate schedulers

Но как понять, как быстро уменьшать learning rate?

Можно использовать стратегию ReduceLROnPlateau:

- Смотрим на ошибку на валидации
- Уменьшаем learning rate, если она перестала падать

Learning rate schedulers

Cyclic learning rate

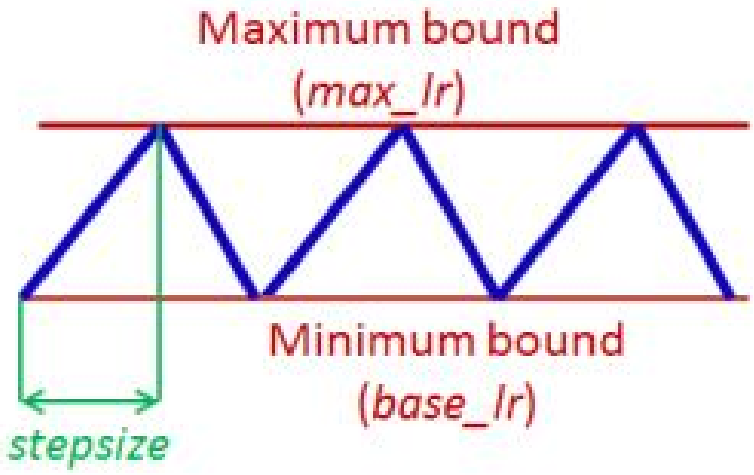
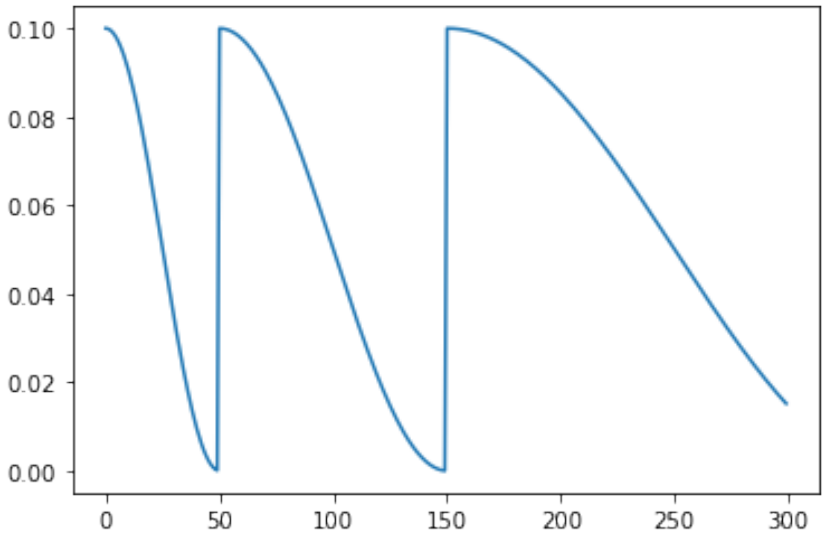


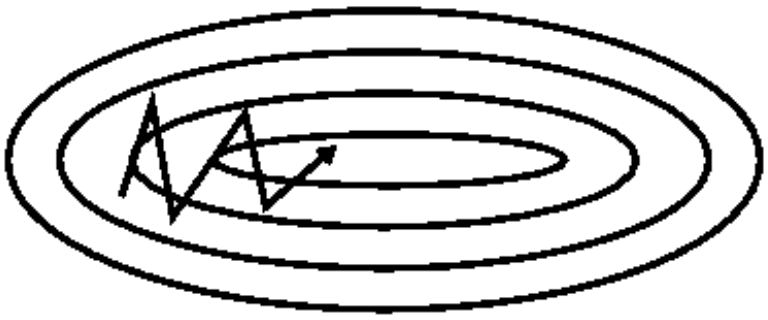
Image credit

Cosine annealing with warm restarts



Momentum

SGD without momentum



SGD with momentum

Добавляем «инерцию», сохраним часть «скорости» с предыдущего шага

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

$$\theta_{t+1} = \theta_t - \alpha m_t$$

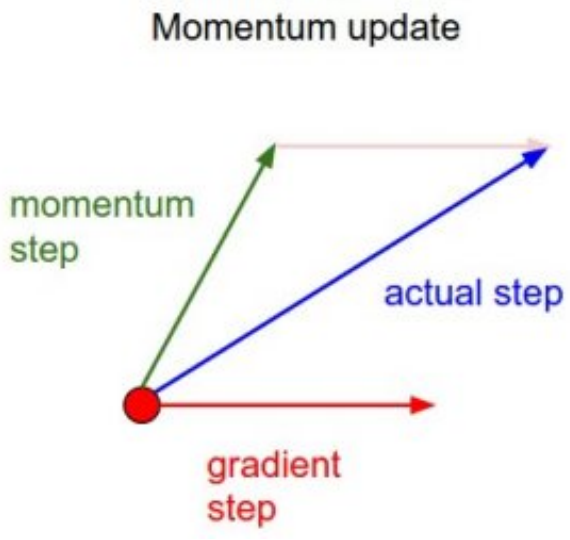
По сути считаем экспоненциальное скользящее среднее градиента

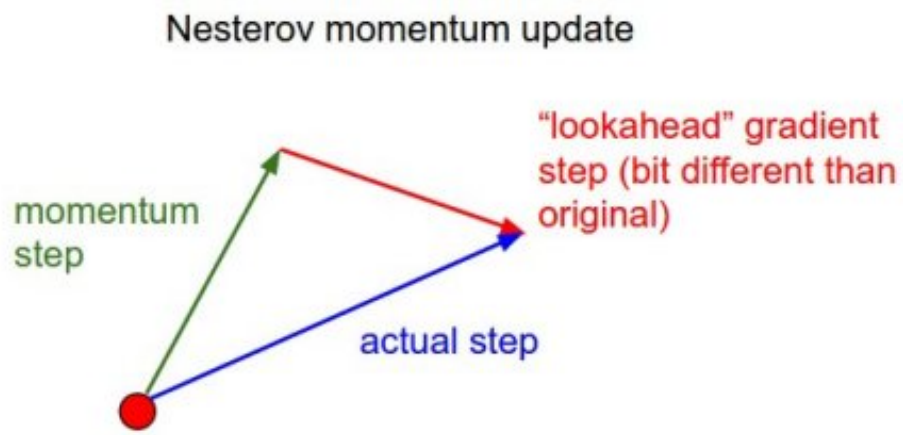
Nesterov's momentum

Метод Нестерова

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

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





Адаптивные методы градиентного спуска

Можно использовать разные learning rate для разных параметров

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

Adagrad

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

Проблема Adagrad - learning rate может слишком быстро уменьшиться до нуля (раньше, чем модель обучится)

Что делать?

RMSProp

Проблема Adagrad - learning rate может слишком быстро уменьшиться до нуля (раньше, чем модель обучится)

Что делать?

Считать скользящее среднее:

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

и обновлять параметры с его использованием:

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

Adam

RMSProp + momentum

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

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

Другие оптимизаторы

Существует много других оптимизаторов, например

- AdamW
- NAdam
- RAdam
- Lion

Но

- Adam обычно работает хорошо

Регуляризация

L1/L2 regularization

L2 регуляризация

$$\frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i, \theta)) + \frac{\lambda}{2}\sum_{j} \theta_j^2 \to \min_{\theta}$$

Также называют weight decay, так как

$$\theta_j' = (1 - \alpha\lambda)\theta_j - \frac{\alpha}{N}\sum_{i=1}^{N} \frac{\partial L(y_i, f(x_i, \theta))}{\partial \theta_j}$$

Но для адаптивных оптимизаторов нет полного соответствия!

L1 регуляризация

$$\frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i, \theta)) + \lambda\sum_{j} |\theta_j| \to \min_{\theta}$$

L2 regularization

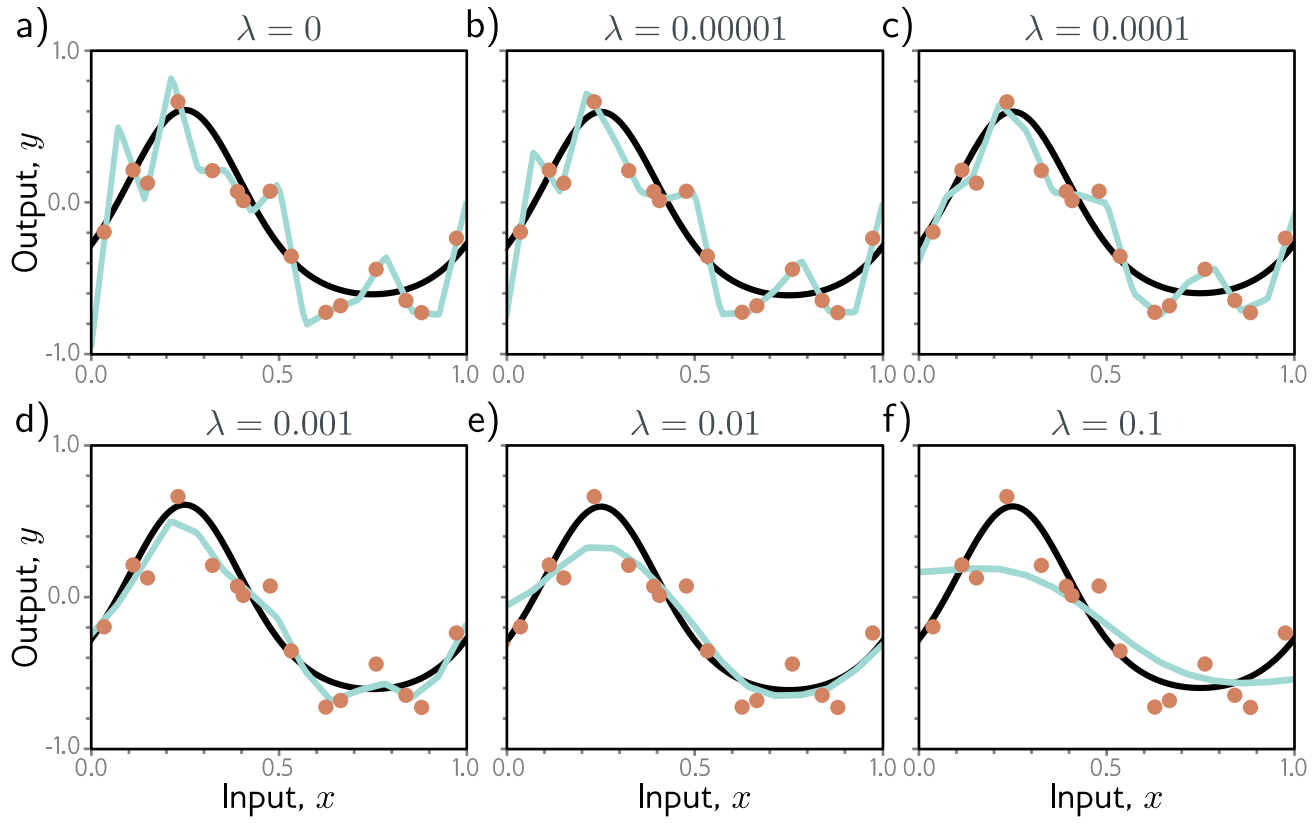


Image credit

- Меньше значения весов
- Более гладкая функция на выходе

Early stopping

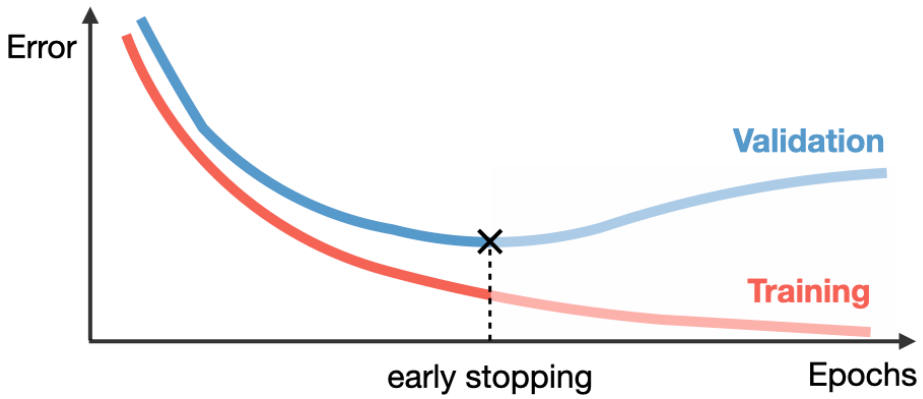


Image credit

- Смотрим на ошибку на валидации
- Останавливаемся, когда перестает убывать

Early stopping

Похоже на L2 регуляризацию - не даем весам стать слишком большими

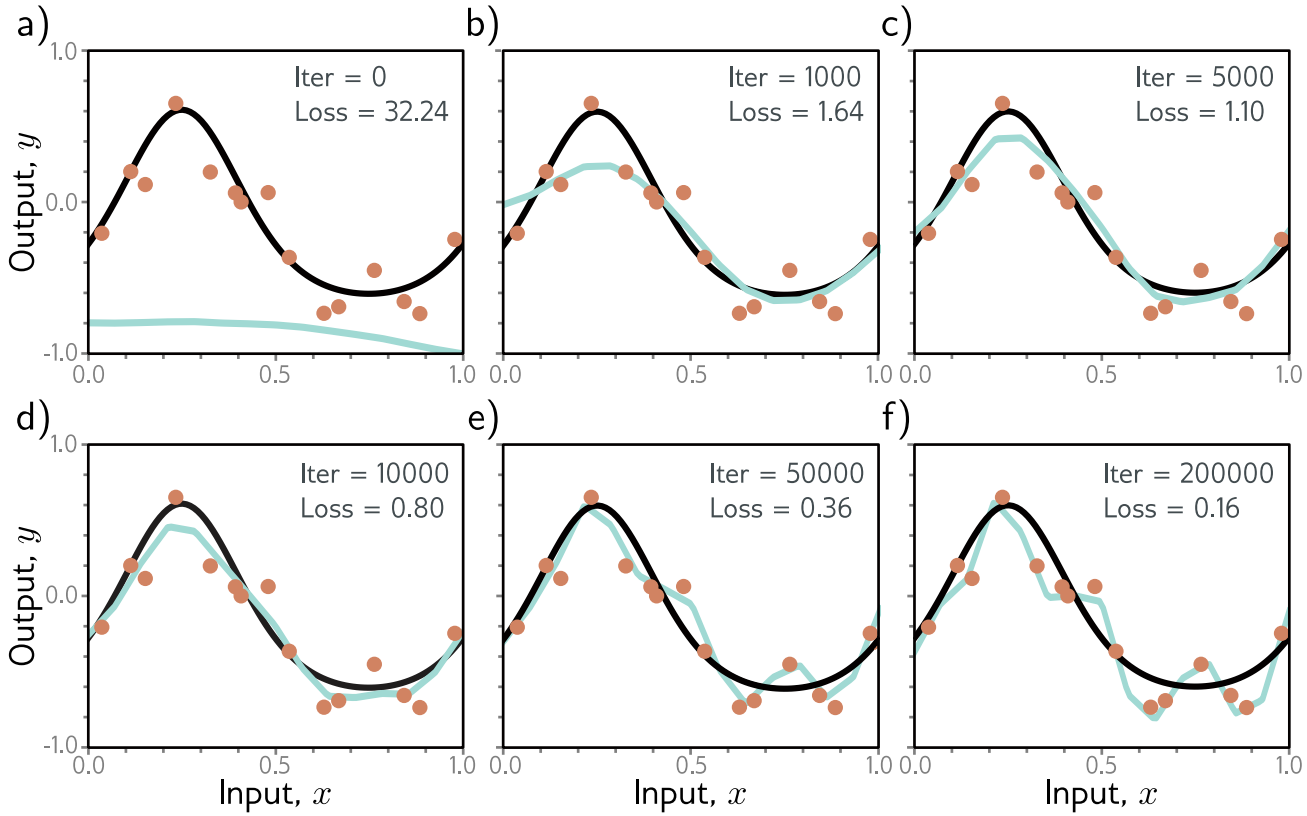


Image credit

Dropout

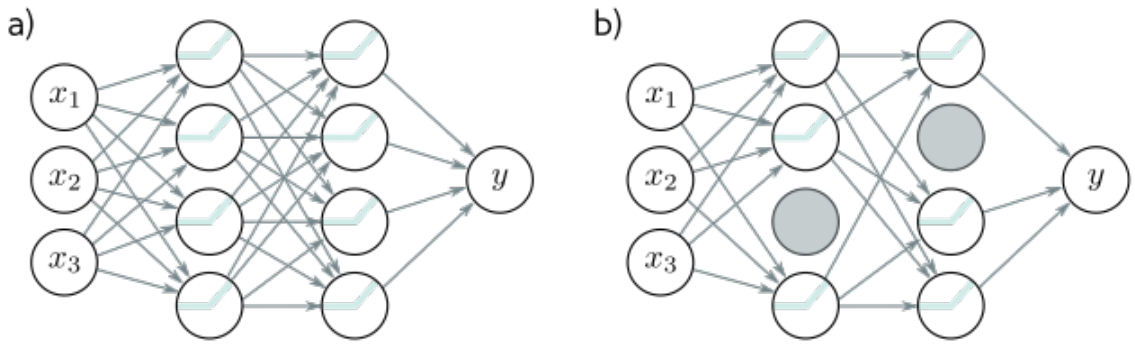


Image credit

При обучении:

- "Выбрасываем"некоторые нейроны случайным образом с вероятностью p (обнуляем активации)

Dropout

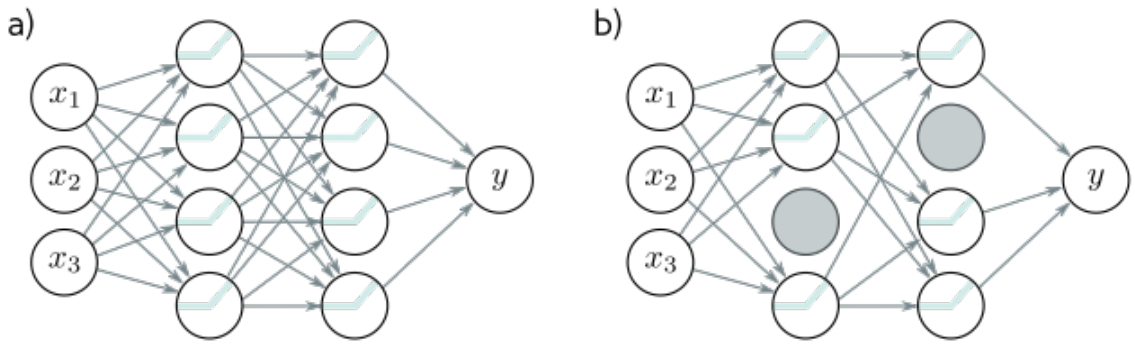


Image credit

При обучении:

- "Выбрасываем"некоторые нейроны случайным образом с вероятностью p (обнуляем активации)

При предсказании:

- Не обнуляем активации
- Умножаем выход каждого нейрона на $1 - p$, чтобы сохранялось мат. ожидание

Dropout

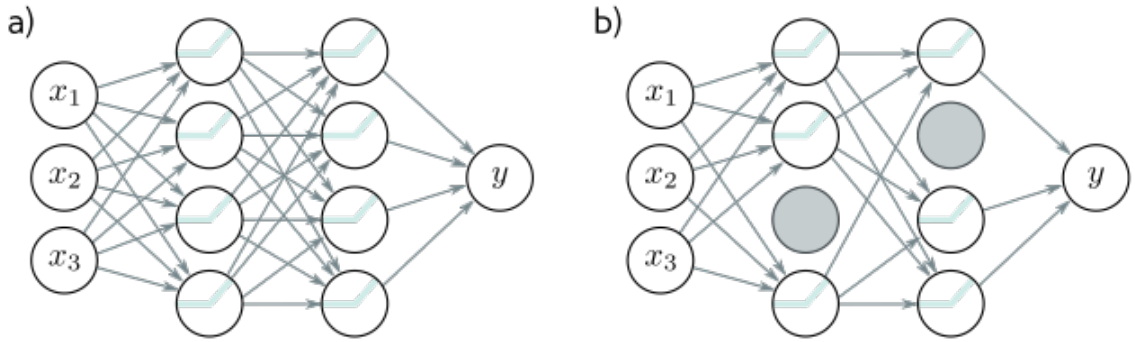


Image credit

- Получается более устойчивая модель, меньше зависящая от конкретных нейронов
- Как бы обучаем ансамбль большого количества сетей (каждую обучаем один шаг)

Data augmentation

Image credit

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

Noise injection

Добавляем шум в данные

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

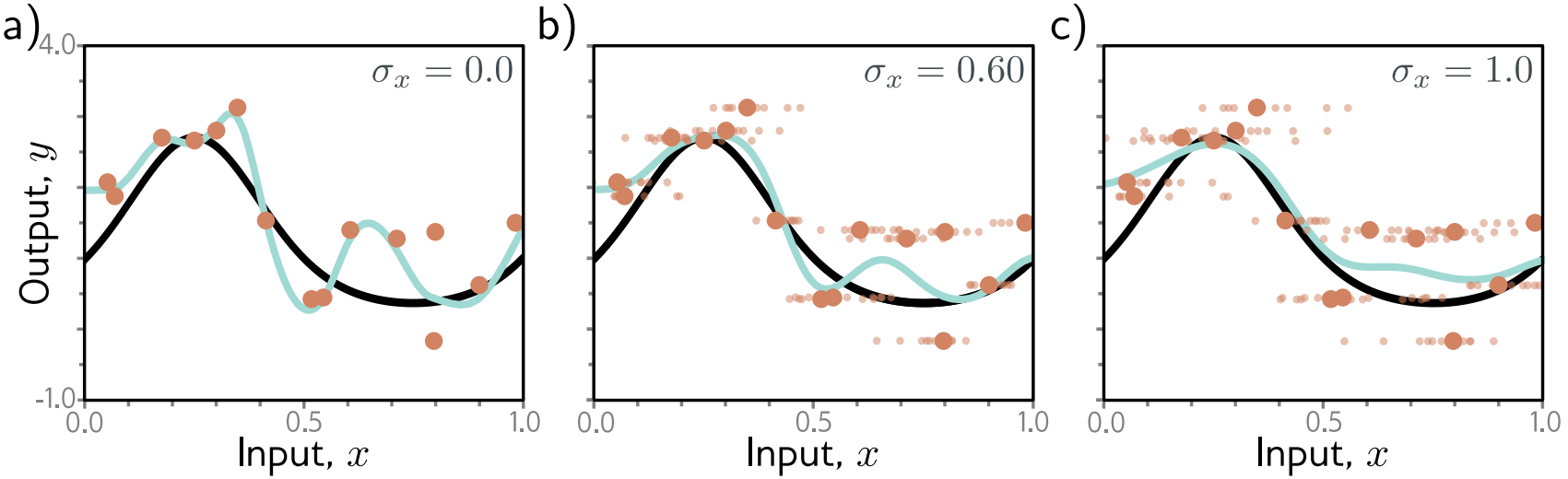
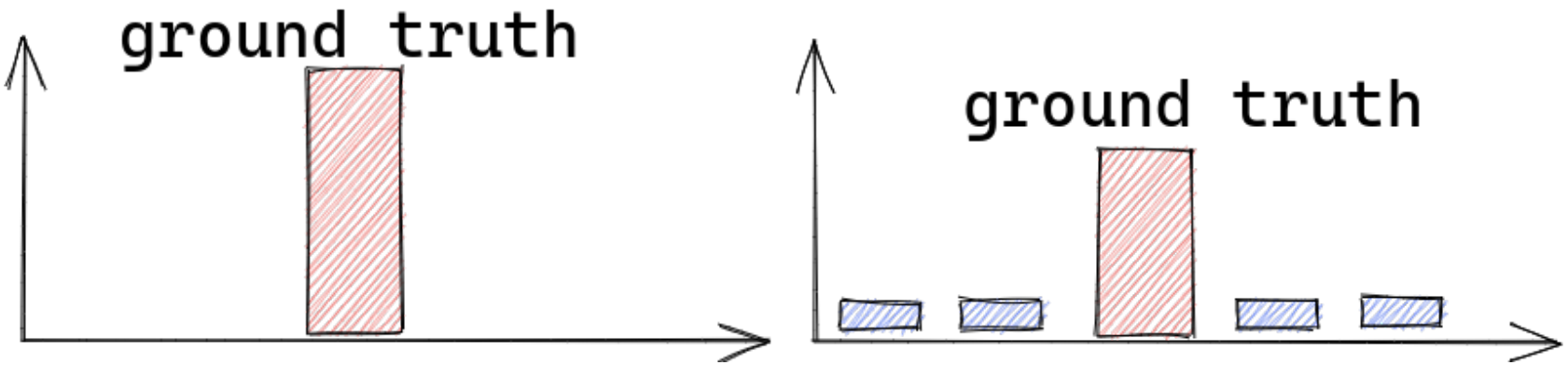


Image credit

Label smoothing



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

Batch normalization

Internal covariate shift (внутренний сдвиг переменных)

- Когда меняются веса слоя, меняется распределение его выходов
- Следующий слой уже подстроился под предыдущее распределение
- Придется следующему слою переучиваться заново

Batch normalization

Batch normalization (BatchNorm)

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Batch normalization

Batch normalization (BatchNorm)

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Для большей выразительности

$$y_i = \gamma\hat{x}_i + \beta$$

γ, β - обучаемые параметры

Batch normalization

Batch normalization (BatchNorm)

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Для большей выразительности

$$y_i = \gamma\hat{x}_i + \beta$$

γ, β - обучаемые параметры

На тесте используем оценки средних и дисперсий, посчитанные во время обучения

$$\hat{\mu} = \alpha\mu_B + (1-\alpha)\hat{\mu}$$

$$\hat{\sigma^2} = \alpha\sigma_B^2 + (1-\alpha)\hat{\sigma^2}$$

Batch normalization

Можно использовать до или после нелинейности

Преимущества Batch normalization

- Лучше и быстрее сходимость
- Позволяет использовать большие значения lr
- Позволяет обучать очень глубокие модели

Почему работает?

- Internal covariate shift
- Делает более гладкой поверхность функции потерь
- Регуляризация, так как на каждом батче статистики разные

Batch normalization

Недостатки Batch normalization

- При малом размере батча статистики становятся ненадежными
- Не работает для рекуррентных сетей
- Неудобно при распараллеливании на разные машины

Layer normalization

Layer normalization - то же самое, только считаем статистики для одного примера, но по всем нейронам

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} x_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(x_i^l - \mu^l)^2}$$

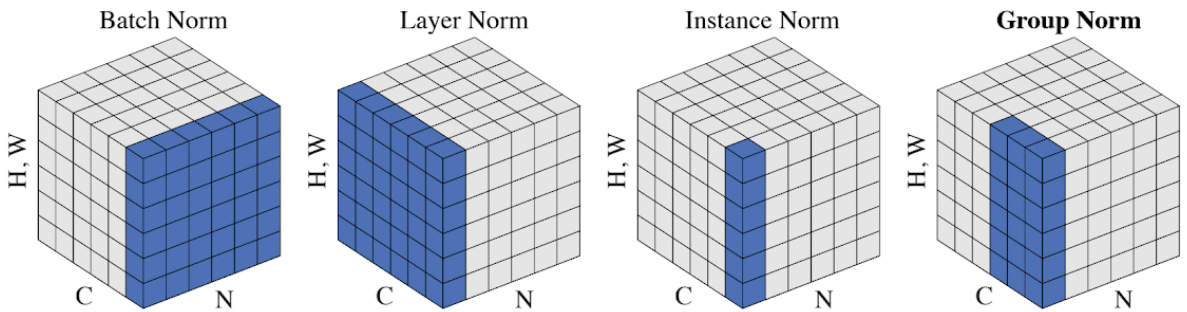


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Image credit

Batch vs Layer normalization

BatchNorm

- CNN, Computer Vision

LayerNorm

- NLP, RNN, Transformers

Нормализация входных данных

Как и в многих других методах машинного обучения, важно нормализовать входные данные

SGD - implicit regularization

Неявная регуляризация в стохастическом градиентном спуске

$$\tilde{L}_{SGD}(\theta) = L(\theta) + \frac{\alpha}{4}\left\|\frac{\partial L}{\partial \theta}\right\|^2 + \frac{\alpha}{4B}\sum_{b=1}^{B}\left\|\frac{\partial L_b}{\partial \theta} - \frac{\partial L}{\partial \theta}\right\|^2$$

Отдает предпочтение траекториям, где

- Меньше норма градиента (более пологие траектории)
- Меньше дисперсия градиентов между батчами (подгоняемся под все данные одинаково хорошо)

Understanding Deep Learning, chapter 9

Полезные ссылки

- An overview of gradient descent optimization algorithms (Ruder)
- A journey into Optimization algorithms for Deep Neural Networks
- A Recipe for Training Neural Networks (Karpathy)
- Deep Learning Tuning Playbook
- How to train your Deep Neural Network

В следующий раз

- Сверточные нейронные сети
- Computer vision