Лекция 3
Регуляризация и аугментация данных

Разработка нейросетевых систем

Канев Антон Игоревич

Итог по сверточным нейронным сетям

Анализ

- CNN повышает точность на тестовой выборке
- Но точность на обучающей выборке по-прежнему выше точности на тестовой

Решение

- Разнообразие набора данных помогает модели обобщать



Аугментация данных

Аугментация данных

Различают две стратегии аугментации: offline-аугментация и online-агументация. Первый вид предполагает заблаговременное расширение обучающей выборки за счёт добавления в неё преобразованных и искаженных копий.

Так, например, если отразить изображения по горизонтали и вертикали, получится увеличить объём обучающей выборки в 4 раза:

- 1x оригинальный датасет
- 2x - оригинальный + горизонтальное отражение
- 3x - оригинальный + горизонтальное отражение + вертикальное отражение
- 4x - оригинальный + горизонтальное отражение + вертикальное отражение + оба отражения одновременно

После расширения обучающей выборки она используется для обучения.

Offline-аугментация

Такой подход имеет следующие недостатки:

- Применяется лишь ограниченное число преобразований из числа возможных (например, при использовании преобразования поворота на целое число градусов)
- Увеличивается размер выборки, который необходимо держать в оперативной памяти или на жестком диске. Для больших наборов данных с миллионами примеров это непозволительная роскошь.

Есть и преимущество:

- Преобразования выполняются лишь раз, поэтому при обучении нейросети доступно больше вычислительных ресурсов.

Online-аугментация

Такой подход наоборот имеет следующие преимущества:

- Применяется каждую эпоху и итерацию новое случайное преобразование с заданными ограничениями (например, при использовании преобразования поворота на целое число градусов)
- Не нужно тратить память на дублирование данных – храним только оригиналы.

Недостаток:

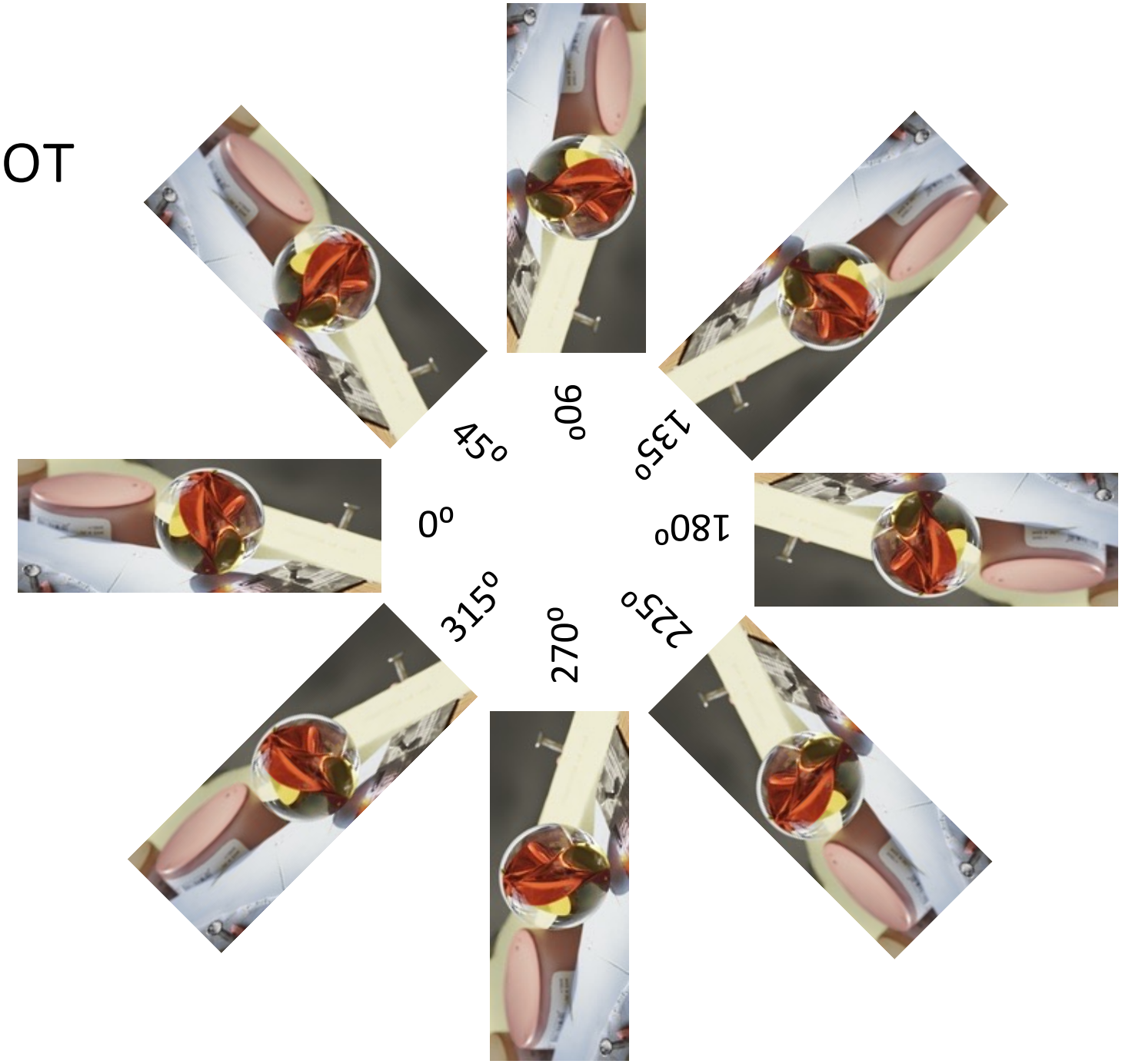
- Преобразования выполняются каждый раз, отнимаем ресурсы от обучения. Но мы можем использовать для обучения GPU, а для аугментации – CPU.

Flipping- переворачивание

Horizontal Flip

Vertical Flip

Rotation - поворот



Rotation vs Flipping

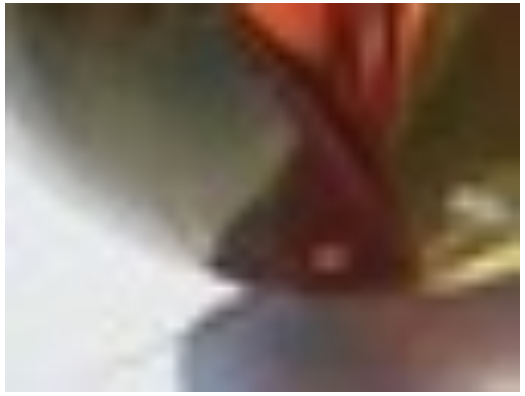
Вращение

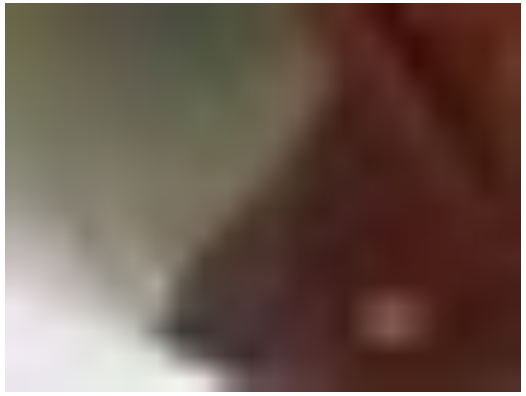
Переворачивание

Zooming - масштабирование



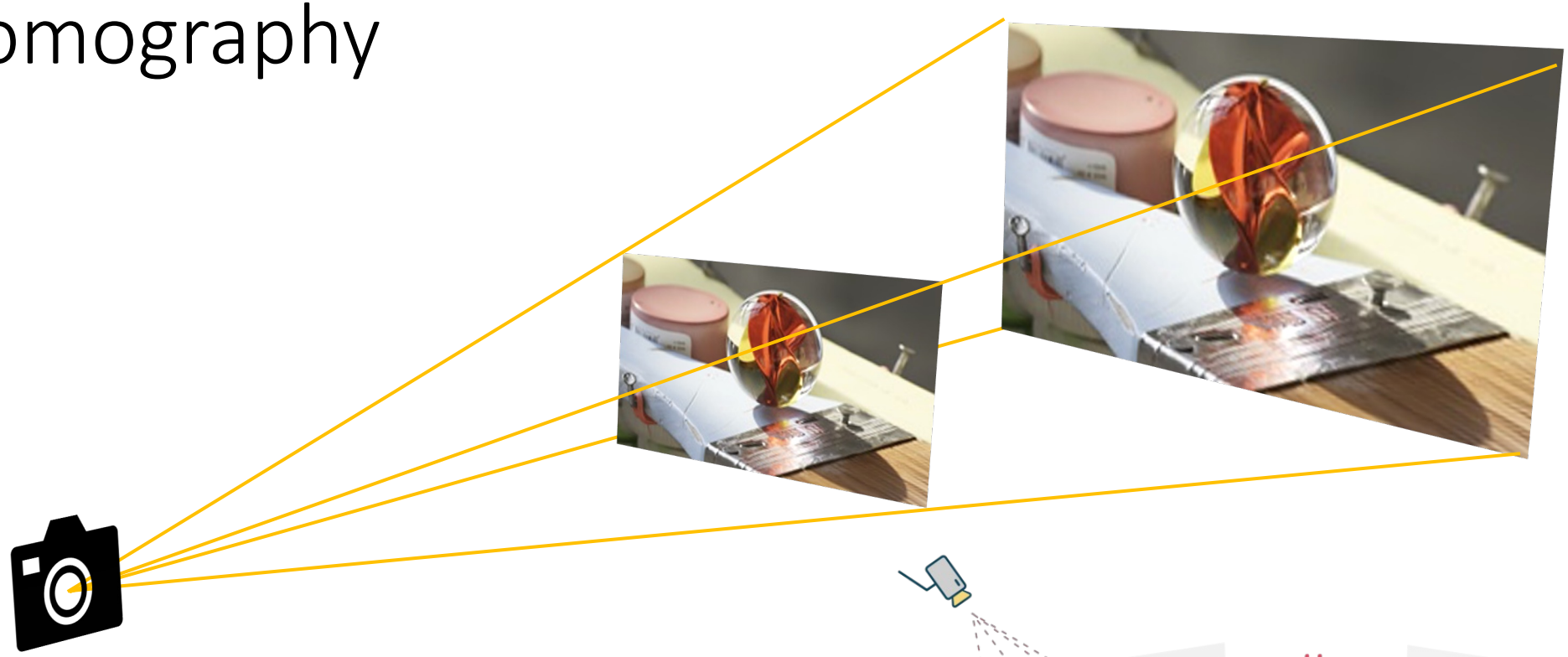


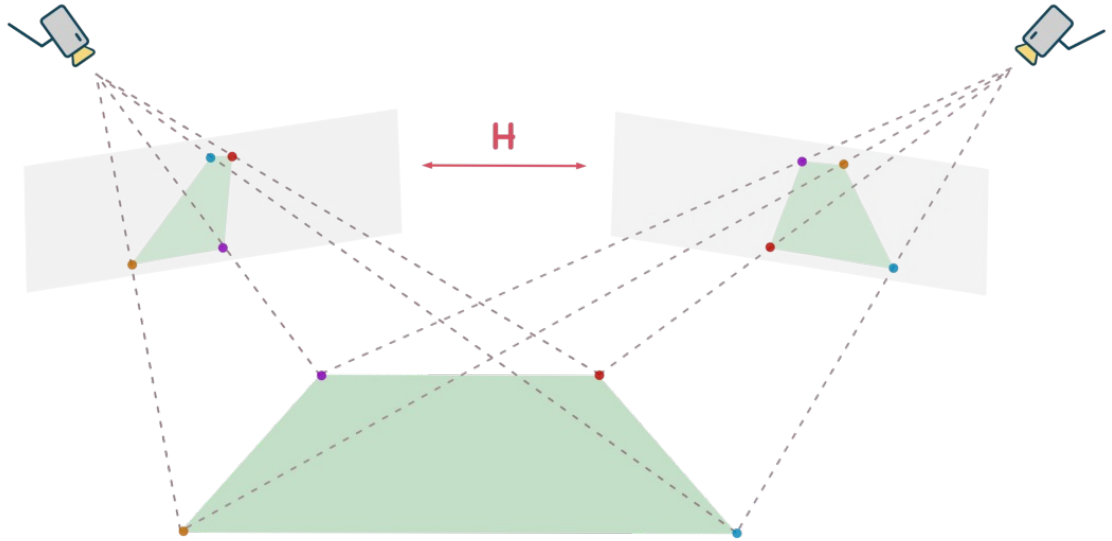





Width And Height Shifting- сдвиги

Homography





Brightness - яркость

Channel Shifting – цветовые каналы

Аугментация данных

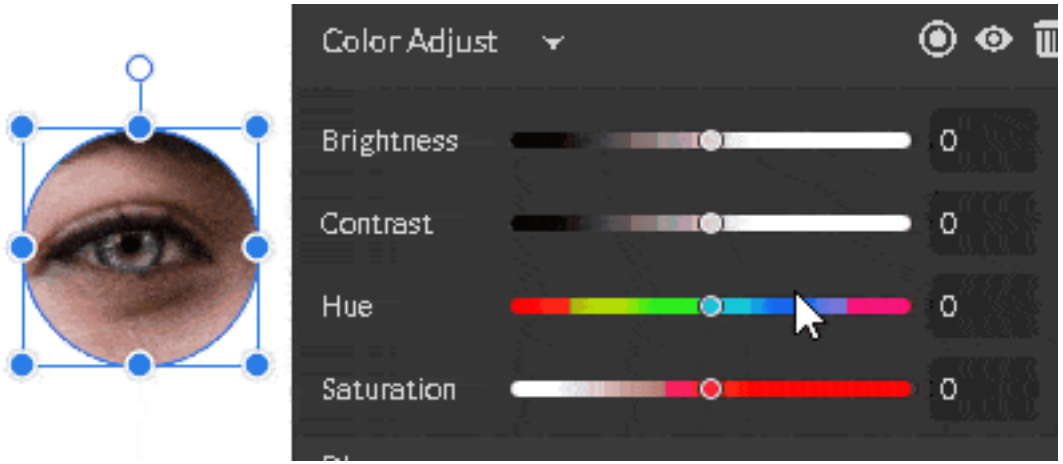
Аугментация данных позволяет разнообразить нашу выборку

```
transform = T.Compose([
     T.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.2, hue=0.0),
     T.RandomAffine(degrees=15, translate=(0.1, 0.1), scale=(0.8, 1.2),
                    shear=5),
])
```

Аугментация данных

T.ColorJitter - Случайным образом изменяет яркость, контрастность, насыщенность и оттенок изображения

- **brightness** – Насколько сильно изменять яркость
- **contrast** – Насколько сильно изменять контраст
- **saturation** – Насколько сильно изменять насыщенность
- **hue** – Как сильно изменять оттенок



Аугментация данных

T.RandomAffine - Случайное аффинное преобразование изображения, сохраняющее инвариантность центра.

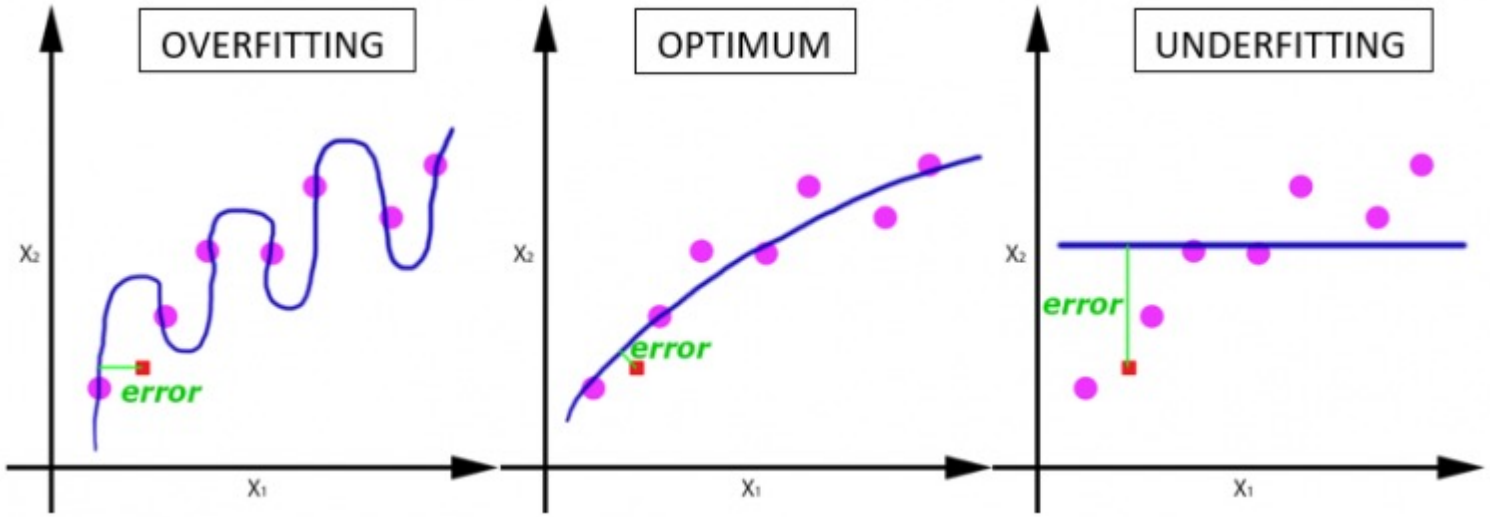
degrees – диапазон градусов для поворота

translate – кортеж для горизонтальных и вертикальных смещений

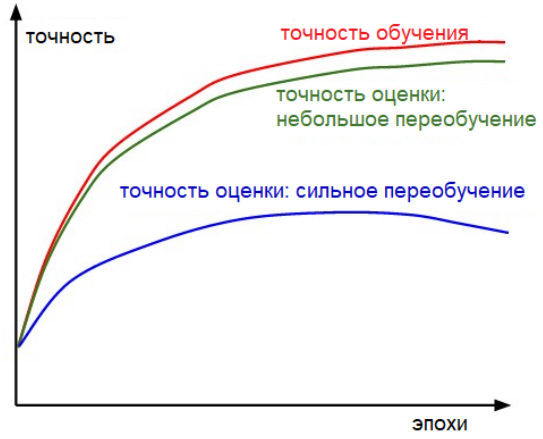
scale – интервал коэффициента масштабирования, например (a, b), затем масштаб выбирается случайным образом из диапазона a <= масштаб <= b

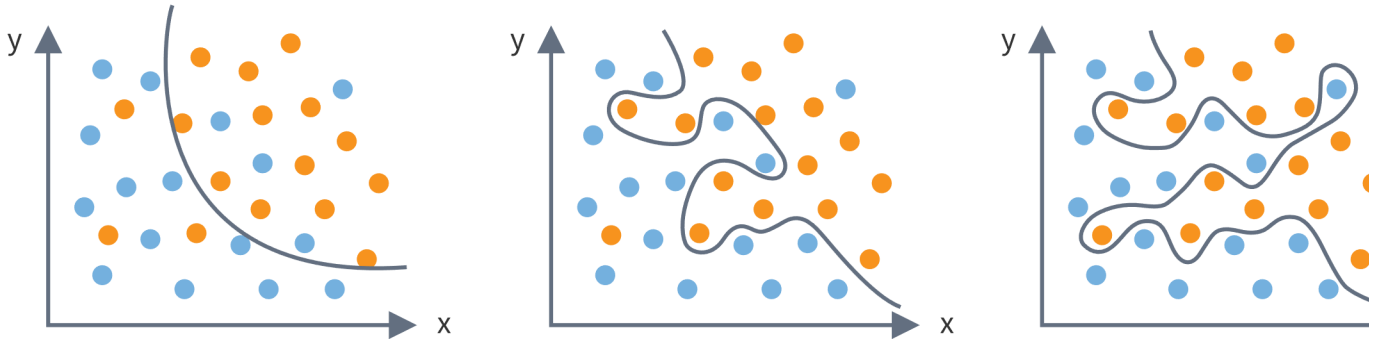
shear – диапазон градусов на выбор. Если сдвиг является числом, то будет применен сдвиг, параллельный оси x в диапазоне (-сдвиг, +сдвиг). В противном случае, если сдвиг представляет собой последовательность из 2 значений, будет применен сдвиг, параллельный оси x в диапазоне (сдвиг [0], сдвиг[1]). В противном случае, если сдвиг представляет собой последовательность из 4 значений, будет применен сдвиг по оси x (сдвиг[0], сдвиг[1]) и сдвиг по оси y (сдвиг[2], сдвиг[3]). По умолчанию сдвиг не применяется.

Точность, переобучение

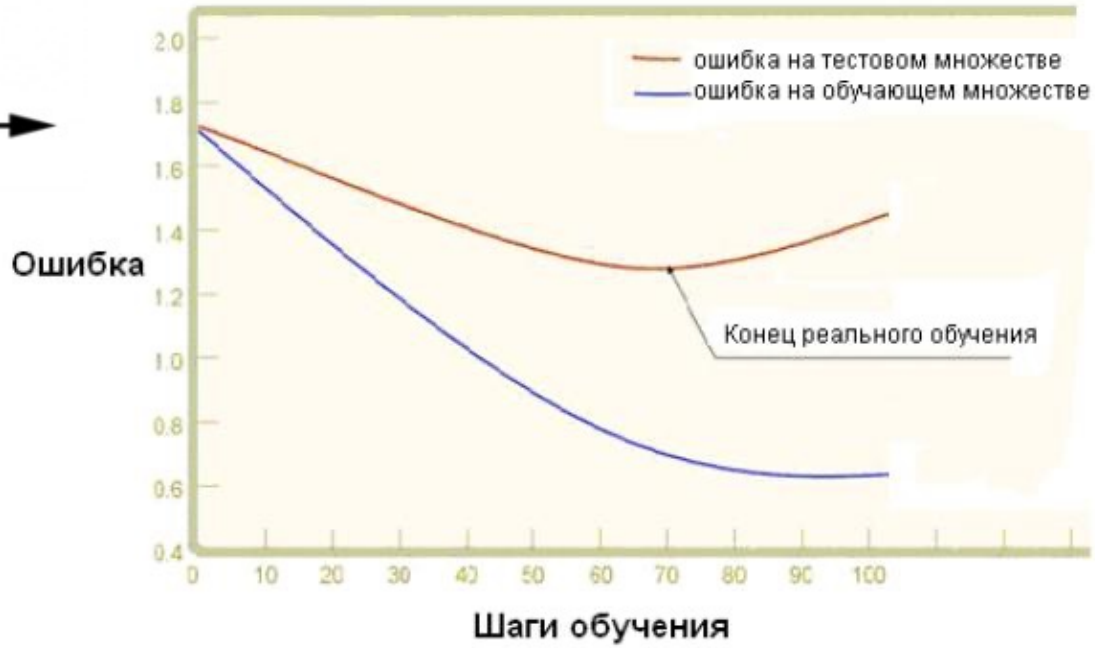


- Переобучение при долгом обучении слишком сложной модели



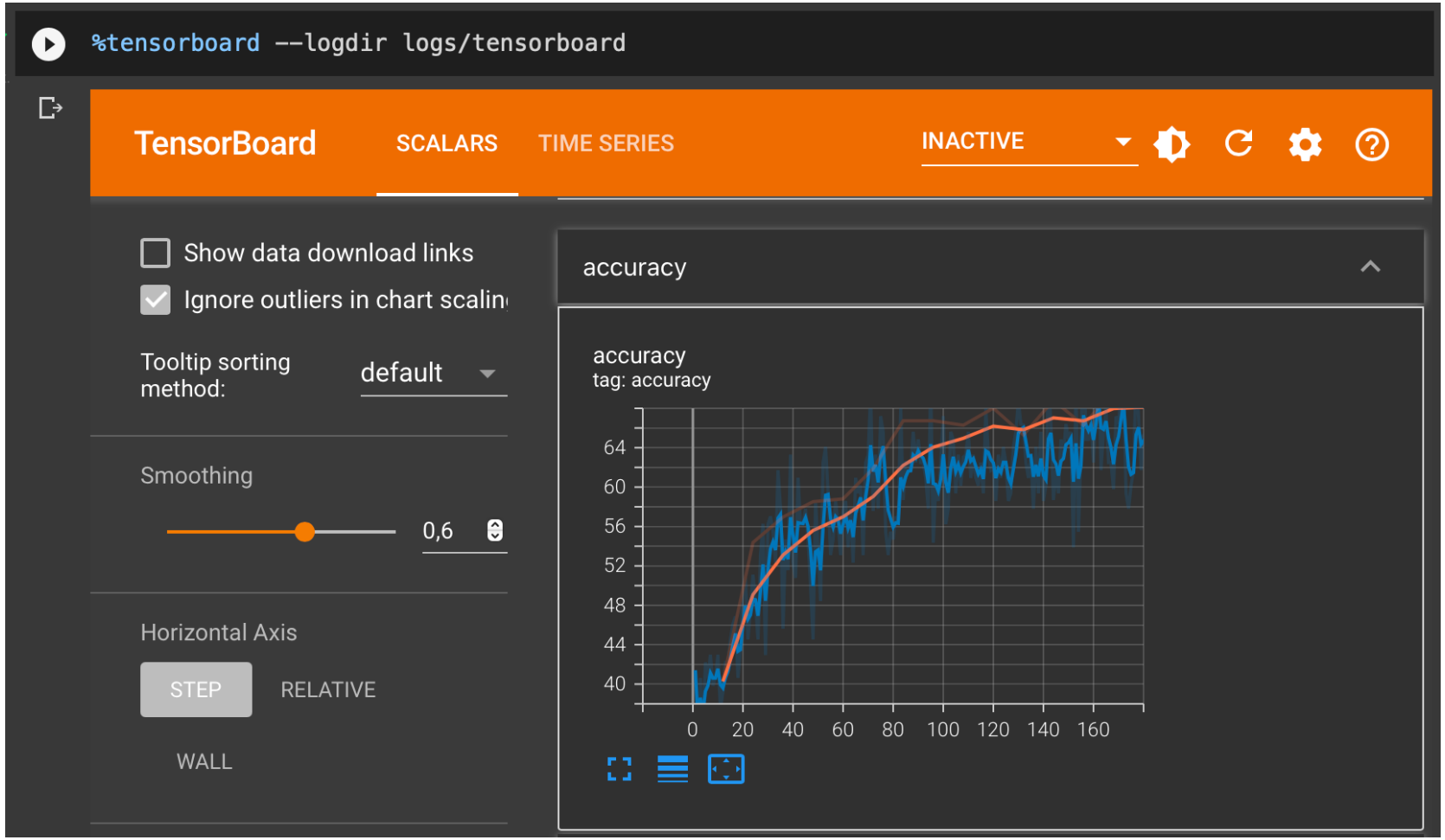


Недообучение | Оптимум | Переобучение



Tensorboard

Tensorboard - инструмент для визуализации различных параметров и переменных в процессе обучения




```
with train_summary_writer.as_default():
        tfsummary.scalar('loss', tmp[-1][0], step=pbar.n)
        tfsummary.scalar('accuracy', tmp[-1][1], step=pbar.n)
```

Регуляризация

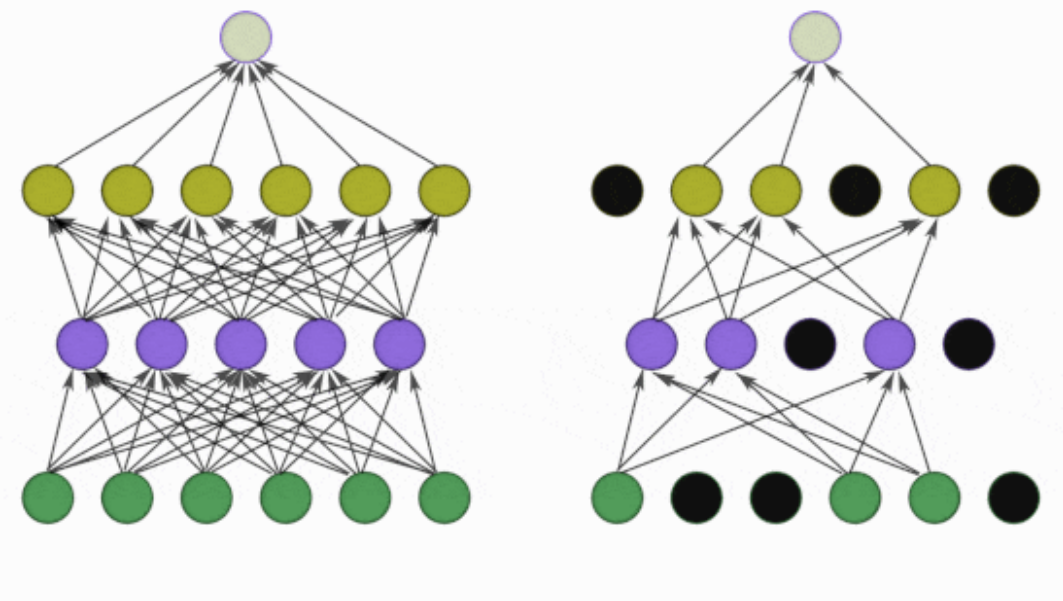
Регуляризация позволяет уменьшить эффект переобучения моделей.

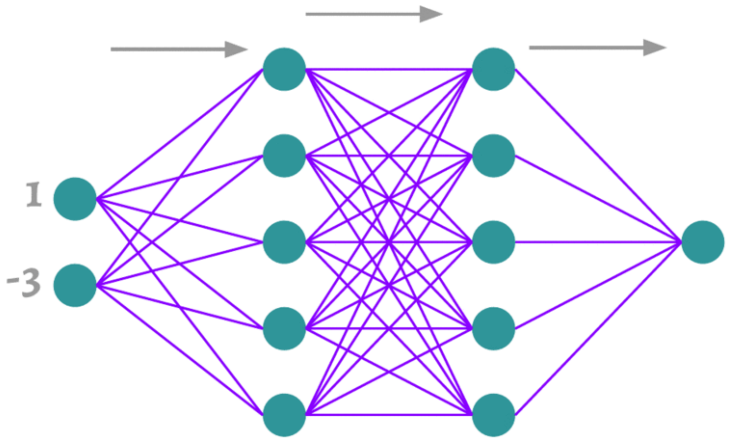
Три вида регуляризации:

- Дропаут
- Штраф за сложность модели
- Label smoothing

Дропаут

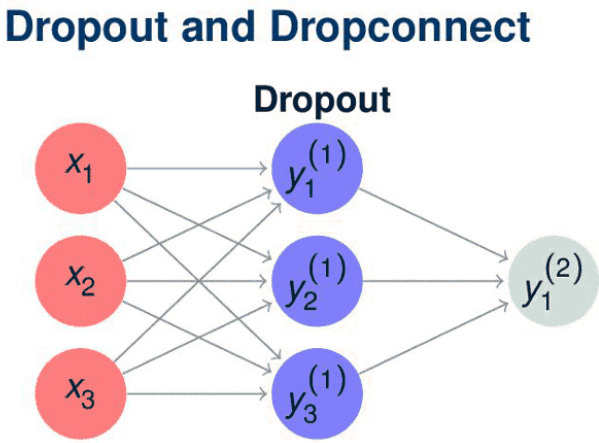
- Дропаут – борьба с переобучением для уменьшения сложности модели





Results : [5.42, 7.89, 4.39, 5.17, 8.01, 6.27,

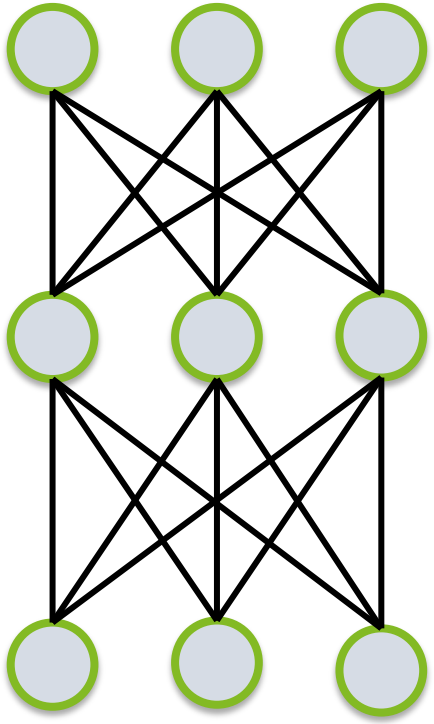
$\mu = 6.19$ $\sigma^2 = 1.85$



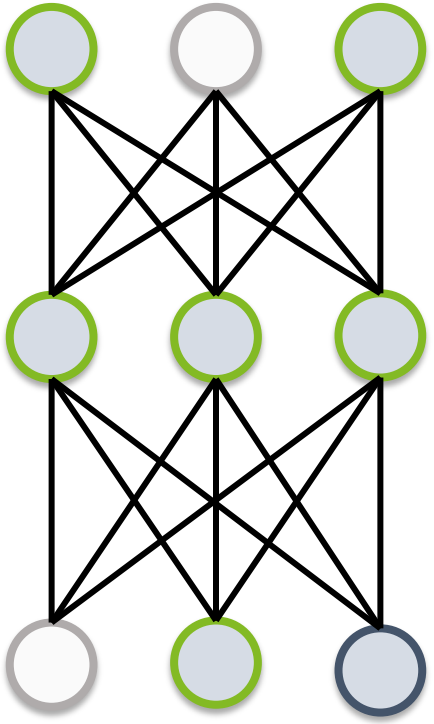
Дропаут

- Слой Droupout позволяет уменьшить эффект переобучения модели. Он достигает этого за счёт "выключения" нейронов предшествующего слоя с вероятностью **p**.
- Под "выключением" подразумевается зануление значений нейронов. На практике p лежит в диапазонах от 0.1 до 0.4, при этом у последующих слоёв большие значения p.
- Для того, чтобы среднее всех нейронов осталось неизменным, значения нетронутых нейронов увеличивают в 1/(1-p) раз. Таким образом, если отключается 50% нейронов, сигнал от остальной половины увеличивается ровно вдвое.
- Поскольку зануление случайной подвыборки нейронов происходит на каждой итерации обучения, достигается эффект "ансамбля" нейронных сетей с меньшим числом параметров.

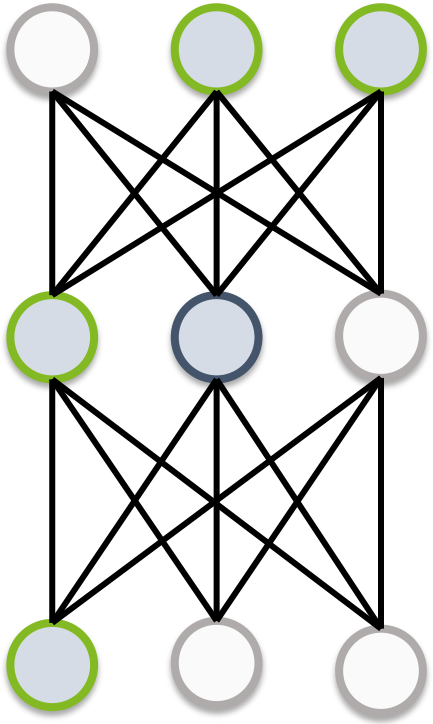
Dropout



rate = 0



rate = .2



rate = .4

Дропаут

```
nn.Conv2d(3, HIDDEN_SIZE, 3, stride=4),
nn.ReLU(),
nn.Dropout2d(p=0.2),

nn.Conv2d(HIDDEN_SIZE, HIDDEN_SIZE*2, 3, stride=1, padding=1),
nn.ReLU(),

nn.AvgPool2d(4),
nn.Dropout2d(p=0.3),
nn.Flatten(),

nn.Linear(HIDDEN_SIZE*8, classes),
```

Штраф за сложность модели

- Второй вариант регуляризации - добавление в функцию потерь второго слагаемого, штрафа за сложность модели.
- Обычно сложность модели выражается взятием нормы её параметров или весов.
- В PyTorch добавление штрафа реализовано на этапе определения оптимизатора градиентного спуска с помощью параметра **weight_decay**:

optimizer = optim.SGD(model.parameters(), lr=1e-3, weight_decay=1e-5)

- В данном примере к функции потерь при вызове метода optimizer.step будет добавлено второе слагаемое, равное сумме норм весов параметров, помноженное на значение параметра **weight_decay**.

$$L_1 = \sum_i \left(y_i - y(t_i)\right)^2 + \lambda \sum_i |a_i|.$$

Label smoothing

Сглаживание меток заключается в изменении разметки для того, чтобы сделать модель менее уверенной в своих предсказаниях.

Рассмотрим пример для трёх классов:

- В таком случае hot-point кодировка класса 2 будет [0, 1, 0].
- Если применить **label smoothing** с параметром 0.3, то сглаженное представление будет равно [0.1, 0.8, 0.1].

К большому везению, данная техника автоматически встроена во многие функции потерь, в частности в **nn.CrossEntropyLoss** в качестве параметра **label_smoothing**.

Также если в обучающей выборке содержатся ошибки в разметке, то функция потерь на сглаженных метках будет придавать им меньший вес.