

**курс «Глубокое обучение»**

# **Борьба с переобучением в нейронных сетях**

**Александр Дьяконов**

**23 сентября 2021 года**

## Борьба с переобучением / сложностью

**Очень много параметров  $\Rightarrow$  переобучение**

- Нормировки (**Normalization of Data**)
- Инициализация весов
- Верификация – ранний останов (**Early Stopping**)
- Настройка темпа обучения (**Learning Rate**)
- Мини-батчи (**Mini-Batches**) / Batch-обучение
- Продвинутая оптимизация
- Регуляризация + **Weight Decay**
- Max-norm-регуляризация
- **Dropout**
- Увеличение выборки + Расширение выборки (**Data Augmentation**)
- Обрезка градиентов (**Gradient clipping**)

## Борьба с переобучением / сложностью

- Доучивание уже настроенных нейросетей (**Pre-training**)
- **Unsupervised Learning**
- Техника зануления весов (**разреживания НС**)
- Использование специальных архитектур под задачу **другая лекция**  
(например, использующих локальность – свёрточных НС)
- **зашумление (inject noise)**
- **Разделение параметров (Parameter Sharing) по всему курсу**

## Нормировки (Normalization of Data)

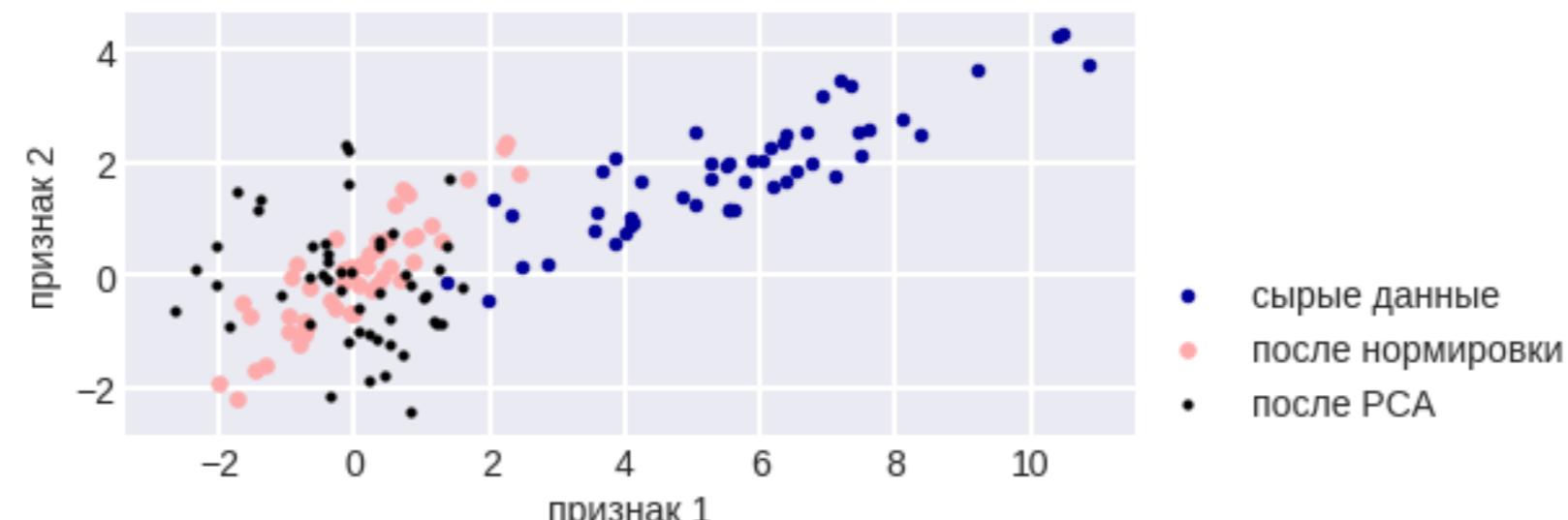
**Признак**  $X = (X_1, \dots, X_m)$

иногда м.б. РСА

$$\mu = \frac{1}{m} \sum_{i=1}^m X_i \text{ среднее признака}$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (X_i - \mu)^2 \text{ дисперсия признака}$$

$$X = \frac{X - \mu}{\sqrt{\sigma^2}} \text{ нормировка}$$



## Минутка кода: нормировки

```
# простое (но не очень хорошее) вычисление статистик
loader = DataLoader(train_set, batch_size=len(train_set), num_workers=1)
data = next(iter(loader))
data[0].mean(), data[0].std()

(tensor(0.2860), tensor(0.3530))

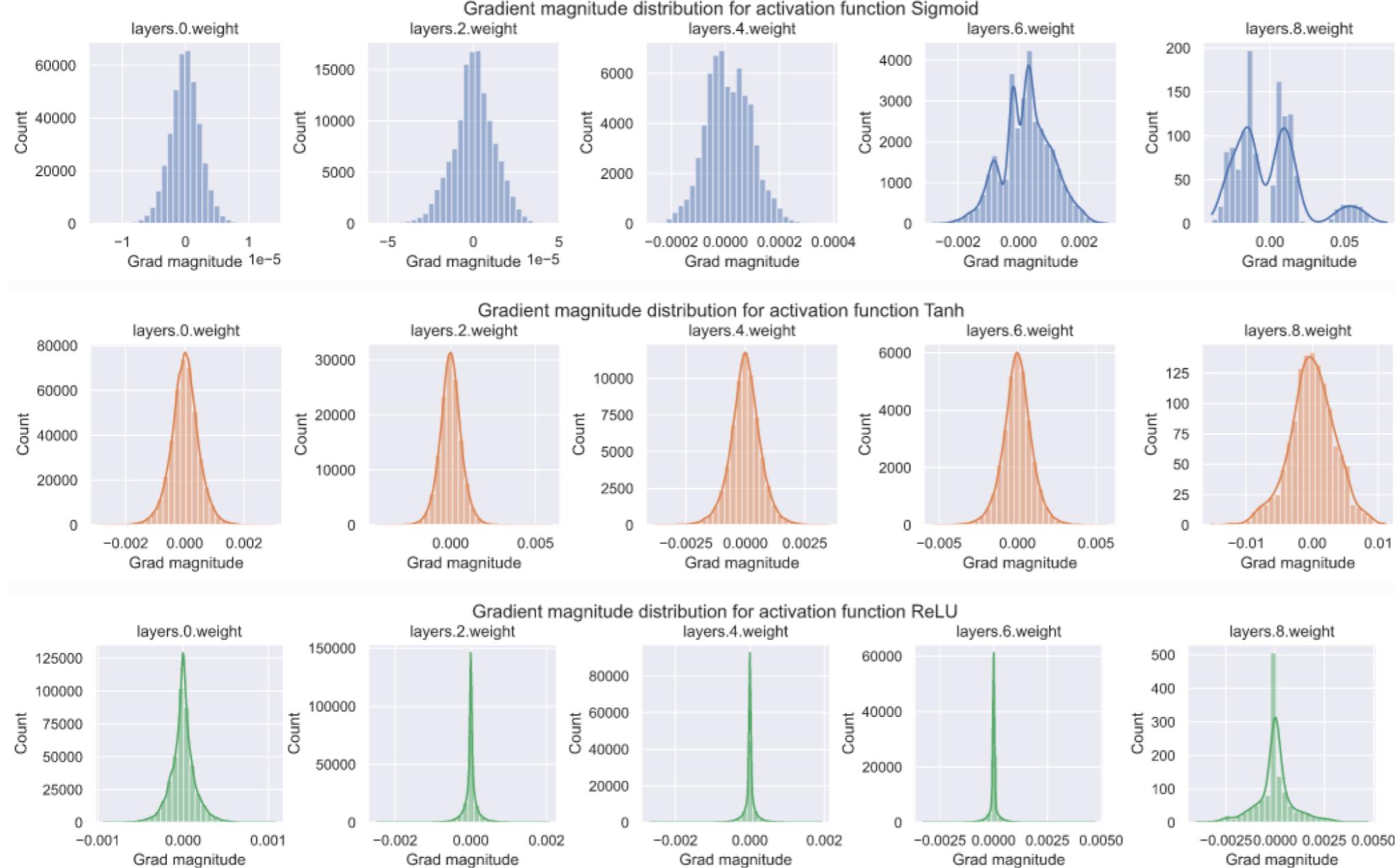
# новый нормированный датасет
train_set_normal = torchvision.datasets.FashionMNIST(root='./data',
                                                       train=True,
                                                       download=True,
                                                       transform=transforms.Compose([transforms.ToTensor(),
                                                       transforms.Normalize(mean, std)]))
```

<https://deeplizard.com/learn/video/Iu7TCu7HeYc>

## Инициализация весов

- **нарушение симметричности**  
(чтобы нейроны были разные)
  - **недопустить «насыщенности» нейрона**  
(почти всегда близок к нулю или 1)
  - **ключевая идея – входы на все слои должны иметь одинаковую дисперсию**  
(для избегания «насыщения» нейронов)
- смещения := 0** (зависит от того, где смещение; если на выходе...)

## Распределения и дисперсии в нейронных сетях

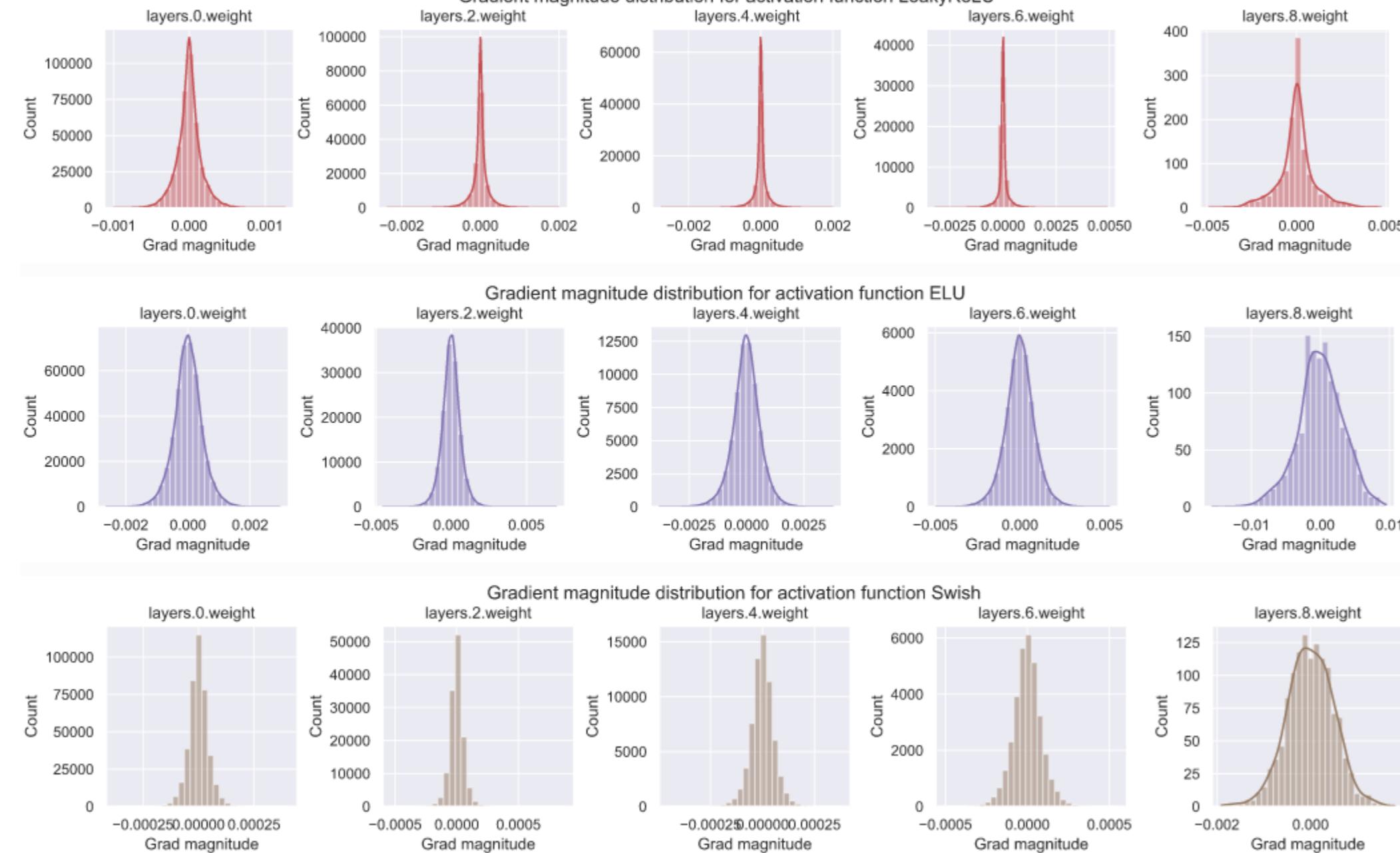


**У сигмоиды очень  
большие градиенты  
на выходе – плохо!**

**И маленькие на  
входе.**

**У RELU пик в нуле –  
понятно**

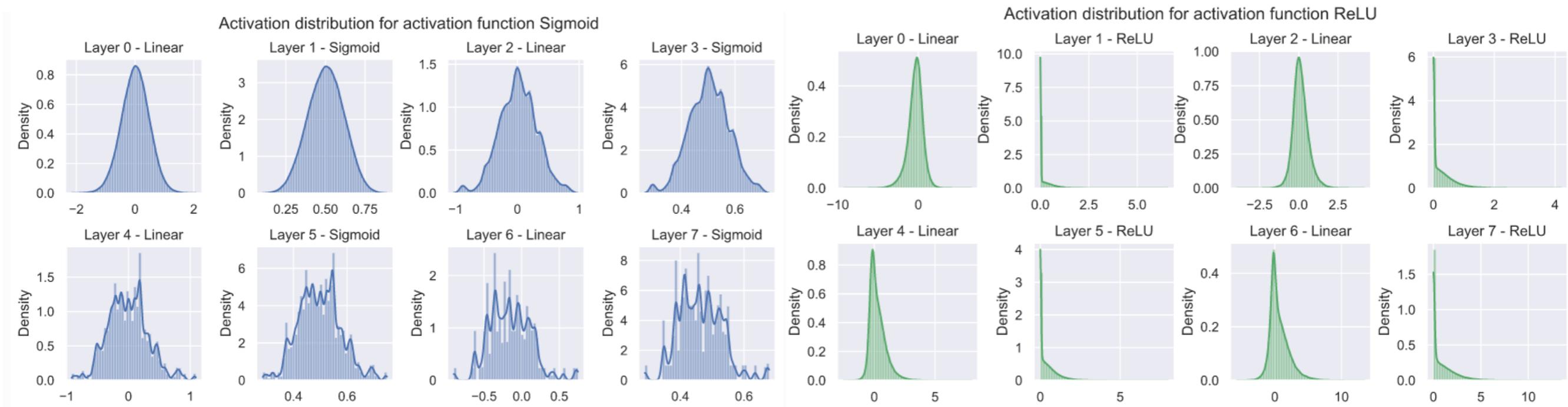
## Распределения и дисперсии в нейронных сетях



**Кроме сигмоиды  
плохих эффектов  
нет**

**Но тут важно,  
какая была  
инициализация!**

## Распределения и дисперсии в нейронных сетях



**Здесь будут разные картинки для разных функций активаций  
(при этом качество НС может быть схожим)**

[https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial\\_notebooks/tutorial3/Activation\\_Functions.html](https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial3/Activation_Functions.html)

## Xavier uniform (Glorot) initialization

$$w_{ij}^{(k)} \sim U\left[-\sqrt{\frac{6}{n_{\text{in}}^{(k)} + n_{\text{out}}^{(k)}}}, +\sqrt{\frac{6}{n_{\text{in}}^{(k)} + n_{\text{out}}^{(k)}}}\right]$$

[Glorot & Bengio, 2010]

**Распределение, чтобы**

$$Dw_{ij}^{(k)} = \frac{2}{n_{\text{in}}^{(k)} + n_{\text{out}}^{(k)}}$$

**Формула выведена в предположении, что нет нелинейностей, w – i.i.d., z – i.i.d...**

$$z^{(k+1)} = f(W^{(k)} z^{(k)}) \equiv W^{(k)} z^{(k)}$$

**МОЖНО ПОСЧИТАТЬ СМ.**

[https://www.youtube.com/watch?v=PjS2y8LBMLc&list=PLrCZzMib1e9oOGNLh6\\_d65HyfdqIJwTQP&index=5](https://www.youtube.com/watch?v=PjS2y8LBMLc&list=PLrCZzMib1e9oOGNLh6_d65HyfdqIJwTQP&index=5)

**Плохо для ReLu – [He et al., 2015]**

## Xavier normal (Glorot) initialization

$$w_{ij}^{(k)} \sim \text{norm}\left(0, \frac{2}{n_{\text{in}}^{(k)} + n_{\text{out}}^{(k)}}\right)$$

## Kaiming (He) uniform initialization

$$w_{ij}^{(k)} \sim U\left[-\sqrt{\frac{3}{n_{\text{in}}^{(k)}}}, +\sqrt{\frac{3}{n_{\text{in}}^{(k)}}}\right]$$

## Инициализация весов: код

```
w = torch.empty(3, 5)
nn.init.uniform_(w, a=0.0, b=1.0)

nn.init.normal_(w, mean=0.0, std=1.0)

nn.init.constant_(w, val=0.3)

nn.init.eye_(w)

nn.init.xavier_uniform_(tensor, gain=1.0)

gain = nn.init.calculate_gain('leaky_relu', 0.2)

# по умолчанию в Pytorch (опт для tanh)
nn.init.kaiming_uniform_(w, a=0, mode='fan_in', nonlinearity='leaky_relu')

nn.init.orthogonal_(w, gain=1)

nn.init.sparse_(w, sparsity, std=0.01)
```

$$w_{ij}^{(k)} \sim U[-a, +a]$$

$$a = \text{gain} \cdot \sqrt{\frac{6}{n_{\text{in}}^{(k)} + n_{\text{out}}^{(k)}}}$$

## Верификация – ранний останов (Early Stopping)

**Смотрим ошибку на отложенной выборке!**

**Выбираем итерацию, на которой наименьшая ошибка.**



**Настройка темпа обучения (Learning Rate)**

## Мини-батчи (mini-batches) / Batch-обучение

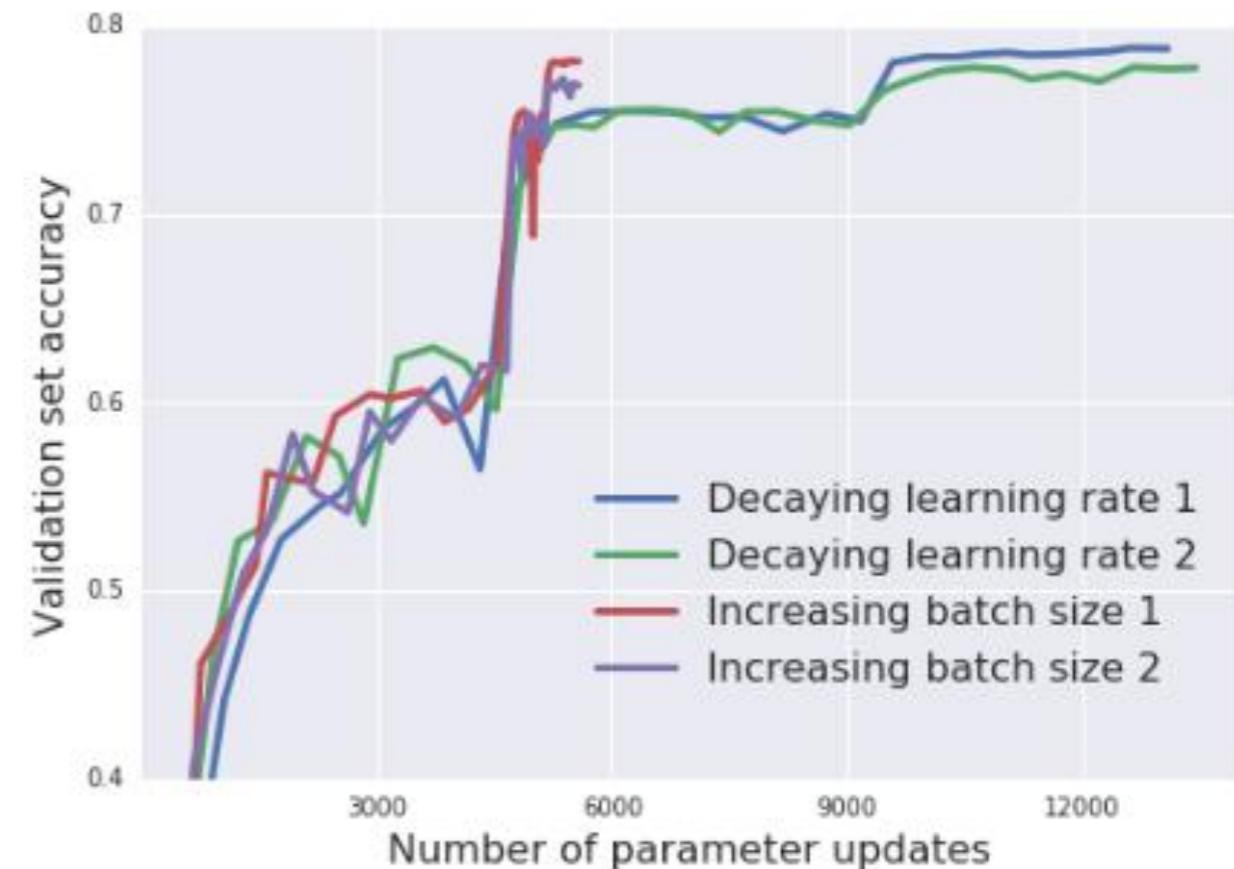
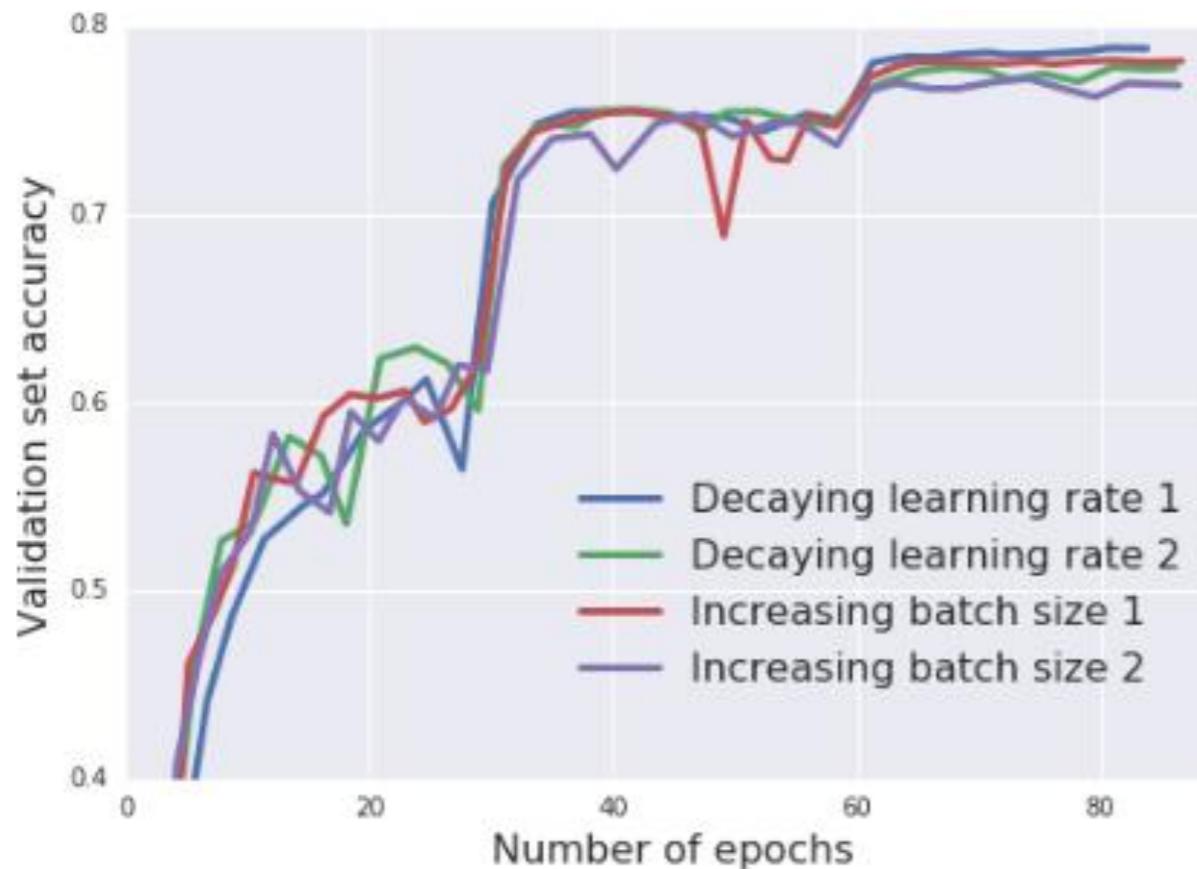
$$w^{(t+1)} = w^{(t)} - \frac{\eta}{|I|} \sum_{i \in I} \nabla[l(a(x_i | w^{(t)}), y_i) + \lambda R(w^{(t)})]$$

- Градиенты не такие случайные (оцениваем по подвыборке)
- Можно вычислять быстрее (на современных архитектурах)  
Можно делать максимальный батч, который влезает в память,  
лучше – при котором максимальное ускорение обучения (см. дальше)
- Можно делать нормировку по батчу
- Немного противоречит теории / рекомендациям

м.б. специально организовывать батчи  
(ex: должны содержать представителей всех классов)

появляется новый гиперпараметр – размер батча  
он связан с другими гиперпараметрами

## Мини-батчи (mini-batches) / Batch-обучение



**увеличение размера батча – тот же эффект, что и уменьшение темпа обучения  
при больших батчах может учиться медленнее! – м.б. проблемы со сходимостью**

**см. дальше**

Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, Quoc V. Le «Don't Decay the Learning Rate, Increase the Batch Size»

<https://arxiv.org/abs/1711.00489>

## Мини-батчи: размер батча

**Есть мнение, что оптимальный размер батча на [1, 32]**

**Yann LeCun's facebook:**

Training with large minibatches is bad for your health. More importantly, it's bad for your test error. Friends dont let friends use minibatches larger than 32. Let's face it: the *only* people have switched to minibatch sizes larger than one since 2012 is because GPUs are inefficient for batch sizes smaller than 32. That's a terrible reason. It just means our hardware sucks.

**Dominic Masters, Carlo Luschi, «Revisiting Small Batch Training for Deep Neural Networks»**

<https://arxiv.org/abs/1804.07612>

## Плоские минимумы

**Считается, что SGD, в отличие от GD, лучше находит «плоские минимумы»  
(более надёжно будет и на teste)**

Chiyuan Zhang «Theory of Deep Learning III: Generalization Properties of SGD»  
<https://cbmm.mit.edu/sites/default/files/publications/CBMM-Memo-067.pdf>

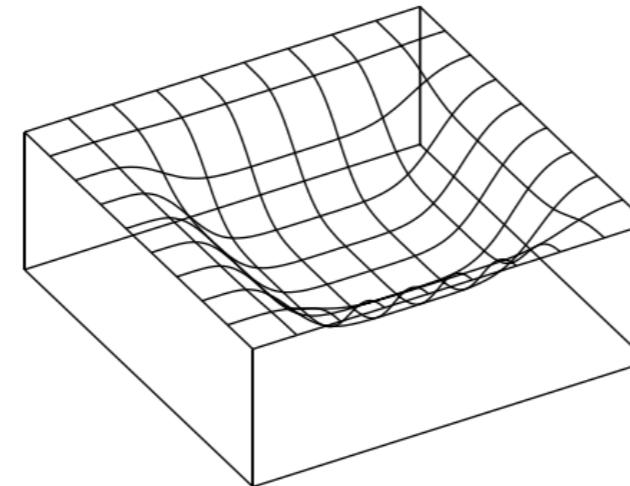


Figure 1: Example of a “flat” minimum.

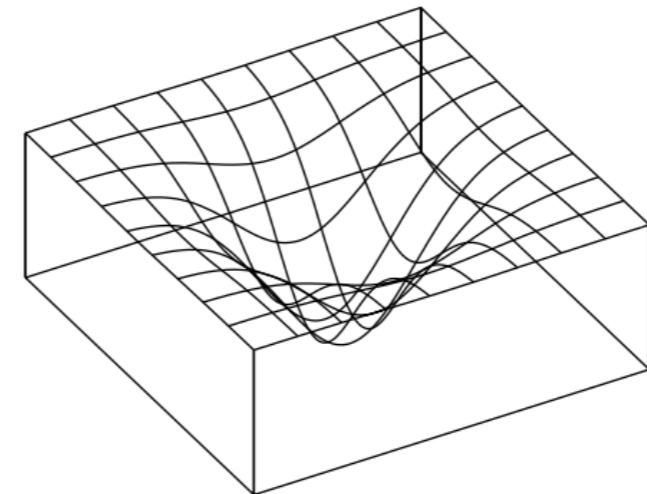


Figure 2: Example of a “sharp” minimum.

<https://www.inference.vc/everything-that-works-works-because-its-bayesian-2/>

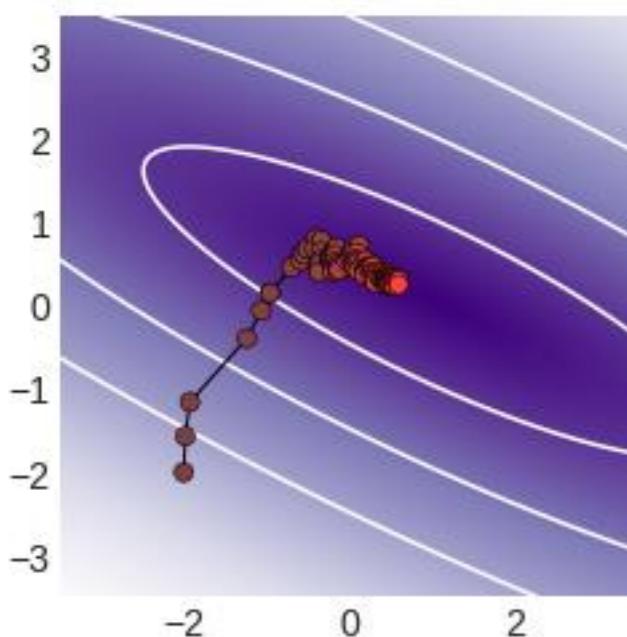
## Продвинутая оптимизация

### Обучение: стохастический градиент

$$w^{(t+1)} = w^{(t)} - \eta \nabla L^{(t)}(w^{(t)})$$

**Эпоха** – проход по всей обучающей выборке

Теоретически SGD – выбор случайного объекта, практически – проход по перестановке



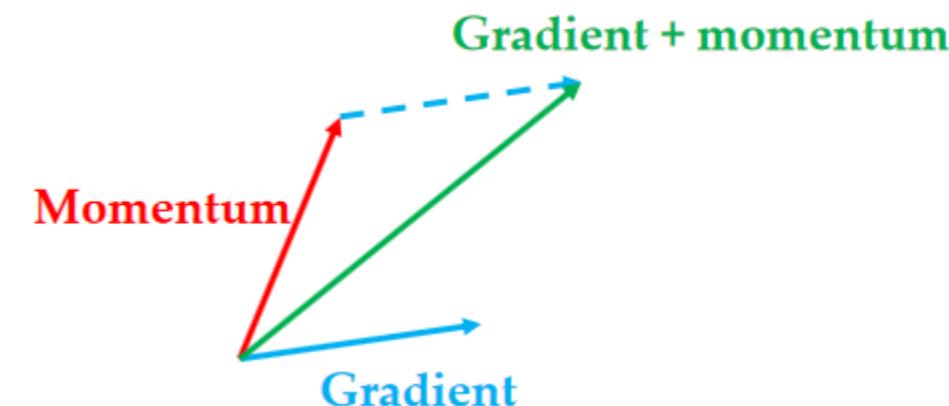
- надо случайно перемешивать данные перед каждой эпохой
- уровни функции могут быть сильно вытянуты – adam (далее)
  - все параметры разные – скорость обучения для каждого параметра
  - градиент случаен (зависит от батча) – добавляем градиенту инертность

## Продвинутая оптимизация: инерция

### стохастический градиент с инерцией (momentum)

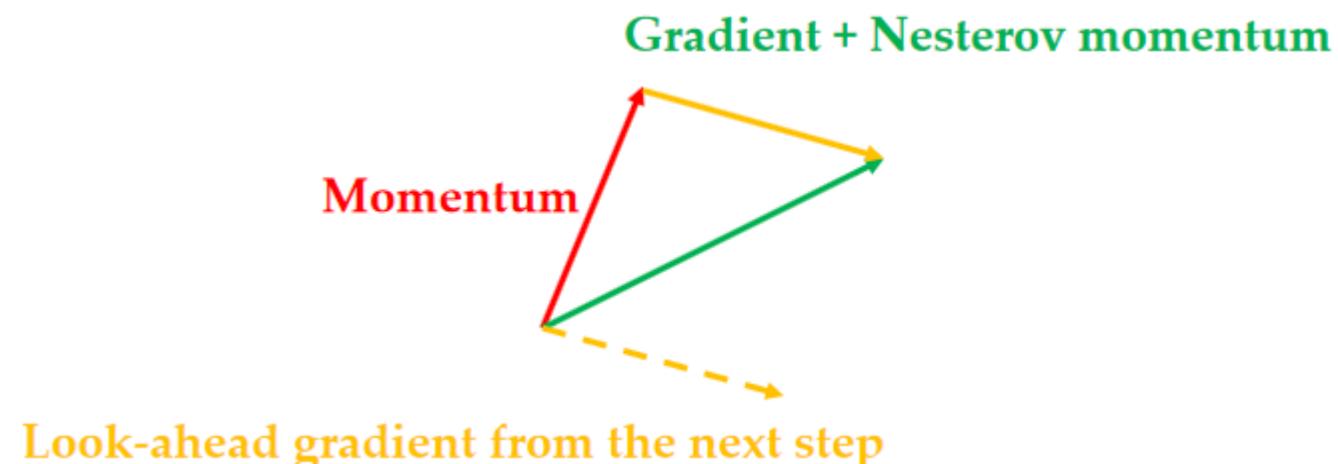
$$\begin{aligned} m^{(t+1)} &= \rho m^{(t)} + \nabla L^{(t)}(w^{(t)}) \\ w^{(t+1)} &= w^{(t)} - \eta m^{(t+1)} \end{aligned}$$

**добавление инерции**



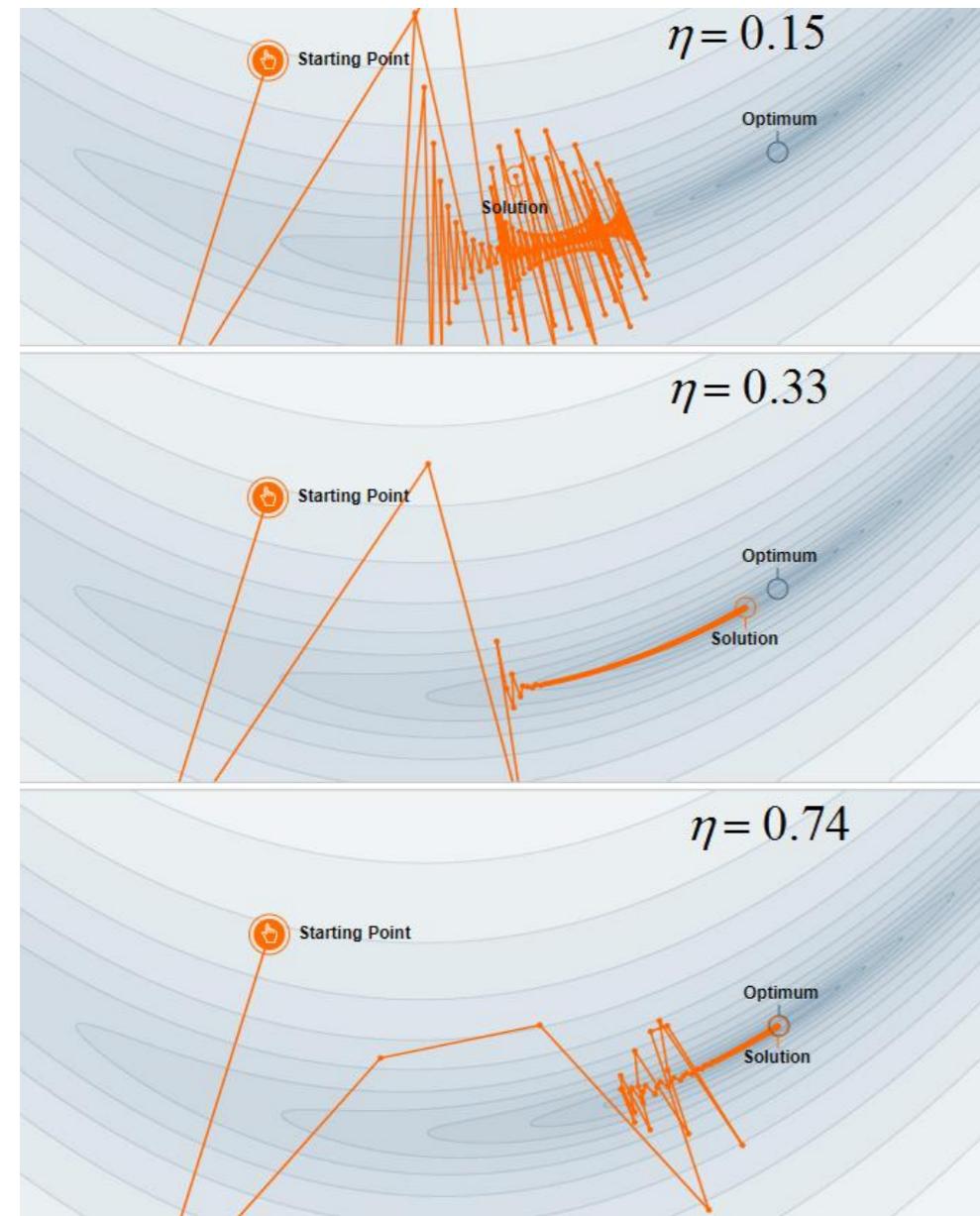
### метод Нестерова

$$\begin{aligned} m^{(t+1)} &= \rho m^{(t)} + \nabla L^{(t)}(w^{(t)} - \eta m^{(t)}) \\ w^{(t+1)} &= w^{(t)} - \eta m^{(t+1)} \end{aligned}$$



<https://uvadlc.github.io/lectures/dec2020/lecture3.1.pdf>

## Иллюстрация СГ с моментом



**Добавление инерции может помочь, когда**

- **линии уровня вытянуты...**
- **для проскачивания седловых точек**

<https://distill.pub/2017/momentum/>

## Продвинутая оптимизация – Адаптивная

### Adagrad [Duchi и др., 2011]

$$v_i^{(t+1)} = v_i^{(t)} + (\nabla_i L^{(t)}(w^{(t)}))^2$$

$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{v_i^{(t+1)} + \epsilon}} \nabla_i L^{(t)}(w^{(t)})$$

### RMSprop = «Root Mean Squared Propagation» = Adagrad + MA [Hinton, 2012]

$$v_i^{(t+1)} = \beta v_i^{(t)} + (1-\beta)(\nabla_i L^{(t)}(w^{(t)}))^2$$

$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{v_i^{(t+1)} + \epsilon}} \nabla_i L^{(t)}(w^{(t)})$$

## Продвинутая оптимизация – Адаптивная

**Adam = «Adaptive Moment Estimation» = RMSprop + momentum + correction bias**

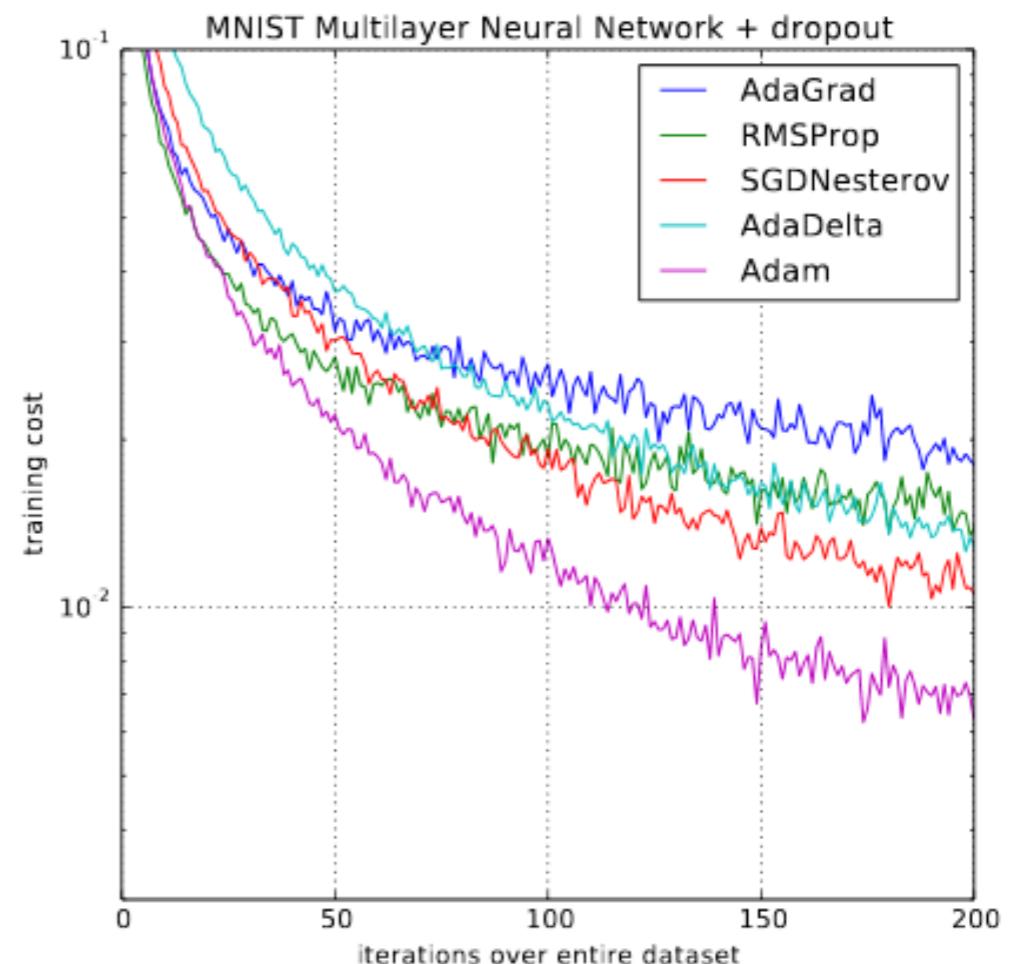
$$\begin{aligned} m_i^{(t+1)} &= \alpha m_i^{(t)} + (1 - \alpha) \nabla_i L^{(t)}(w^{(t)}) \\ v_i^{(t+1)} &= \beta v_i^{(t)} + (1 - \beta) (\nabla_i L^{(t)}(w^{(t)}))^2 \end{aligned}$$

**корректировка смещения:**

$$\hat{m}_i^{(t+1)} = m_i^{(t+1)} / (1 - \alpha^t)$$

$$\hat{v}_i^{(t+1)} = v_i^{(t+1)} / (1 - \beta^t)$$

$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{\hat{v}_i^{(t+1)} + \epsilon}} \hat{m}_i^{(t)}$$



[Kingma, Ba, 2014 <https://arxiv.org/abs/1412.6980>]

18k ссылок, неверное доказательство сходимости

## Продвинутая оптимизация

### AdaMax [Kingma and Ba, 2015]

**вместо**

$$\begin{aligned}v_i^{(t+1)} &= \beta v_i^{(t)} + (1 - \beta)(\nabla_i L^{(t)}(w^{(t)}))^2 \\ \hat{v}_i^{(t+1)} &= v_i^{(t+1)} / (1 - \beta^t)\end{aligned}$$

**будет**

$$v_i^{(t+1)} = \max[\beta v_i^{(t)}, |\nabla_i L^{(t)}(w^{(t)})|]$$

### AdaDelta (модификация Adagrad)

$$\begin{aligned}v_i^{(t+1)} &= v_i^{(t)} + (\nabla_i L^{(t)}(w^{(t)}))^2 \\ \Delta w_i^{(t+1)} &= -\frac{\sqrt{\eta_i^{(t)} + \epsilon}}{\sqrt{v_i^{(t+1)} + \epsilon}} \nabla_i L^{(t)}(w^{(t)}) \\ w_i^{(t+1)} &= w_i^{(t)} + \Delta w_i^{(t+1)} \\ \eta_i^{(t+1)} &= \gamma \eta_i^{(t)} + (1 - \gamma)(\Delta w_i^{(t+1)})^2\end{aligned}$$

## Продвинутая оптимизация

**Подбор гиперпараметров очень важен!**

**Подбор метода оптимизации очень важен!**

### Хорошие обзоры

**«An overview of gradient descent optimization algorithms»**

<https://ruder.io/optimizing-gradient-descent/>

**«An updated overview of recent gradient descent algorithms»**

<https://johnchenresearch.github.io/demon/>

**Rectified Adam (RAdam, 2019)**

<https://medium.com/@lessw/new-state-of-the-art-ai-optimizer-rectified-adam-radam-5d854730807b>

## Продвинутая оптимизация: код

```
from torch.utils.data import DataLoader, TensorDataset

X_dataset = TensorDataset(torch.tensor(X, dtype=torch.float32).to(device),
                          torch.tensor(Y, dtype=torch.float32).to(device))
X_dataloader = DataLoader(X_dataset, batch_size=512, shuffle=True)

optimizer = torch.optim.Adam(model.parameters(),
                             lr=0.001,
                             betas=(0.9, 0.999),
                             eps=1e-08,
                             weight_decay=0,
                             amsgrad=False)

for input, target in X_dataset:
    optimizer.zero_grad()
    output = model(input)
    loss = loss_fn(output, target)
    loss.backward()
    optimizer.step()
```

## Что выбрать на практике

**1) посмотрите, что использовали авторы нейросетевой архитектуры**

**2) Adam почти всегда хорош**  
считается даже «over-optimize»

**3) Если нет – попробуйте SGD + momentum**  
можно попробовать разные методы

**torch.optim** довольно консервативен

## Зашумление градиента (Gradient Noise)

Например, добавление к градиенту шума (вываливаемся из седловых точек)

$$g^{(t)} \leftarrow g^{(t)} + \text{norm} \left( 0, \frac{\eta}{(1+t)^\gamma} \right)$$

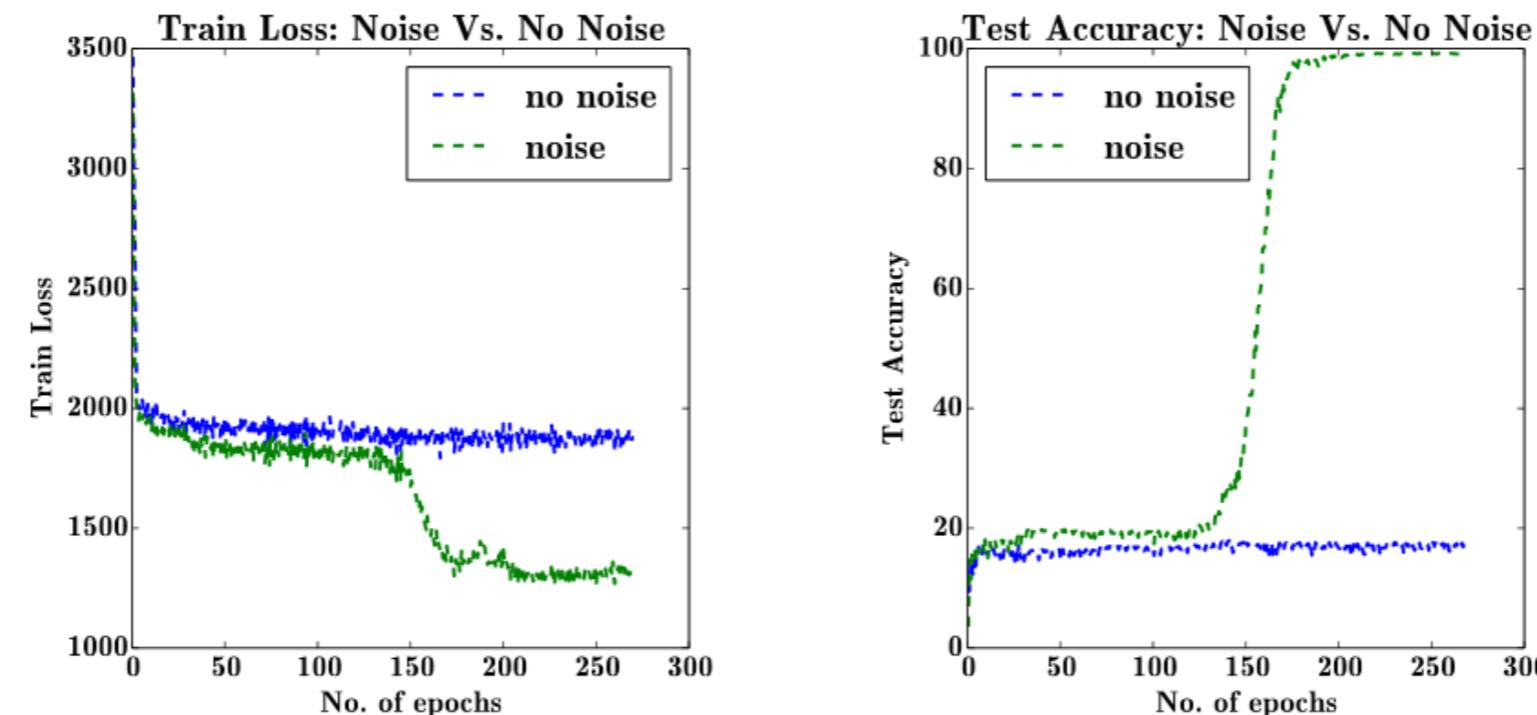


Figure 2: Noise vs. No Noise in our experiment with tables containing 5 columns. The models trained with noise generalizes almost always better.

Neelakantan et al. «Adding gradient noise improves learning for very deep networks» // <https://arxiv.org/abs/1511.06807>

Anandkumar and Ge «Efficient approaches for escaping higher order saddle points in non-convex optimization» <https://arxiv.org/abs/1602.05908>

## Регуляризация + Weight Decay

**Не применяется к весам к константным входам (смещениям)**

**L2, L1 – регуляризация**

**Уменьшение весов (вид регуляризации)**

$$w^{(t+1)} = (1 - \lambda)w^{(t)} - \eta \nabla L^{(t)}(w^{(t)})$$

**На самом деле это L2-регуляризация (в SGD):**

$$\nabla(L(w) + \lambda \| w \|^2) = \nabla L(w) + \lambda w$$

$$w^{(t+1)} = w^{(t)} - \eta(\nabla L(w^{(t)}) + \lambda w^{(t)}) = (1 - \lambda\eta)w^{(t)} - \eta \nabla L(w^{(t)})$$

```
optimizer = torch.optim.SGD(model.parameters(), lr=0.1,  
                           momentum=0.9, weight_decay=0)
```

## Max-norm-регуляризация

**Для каждого нейрона ограничиваем норму весов:**

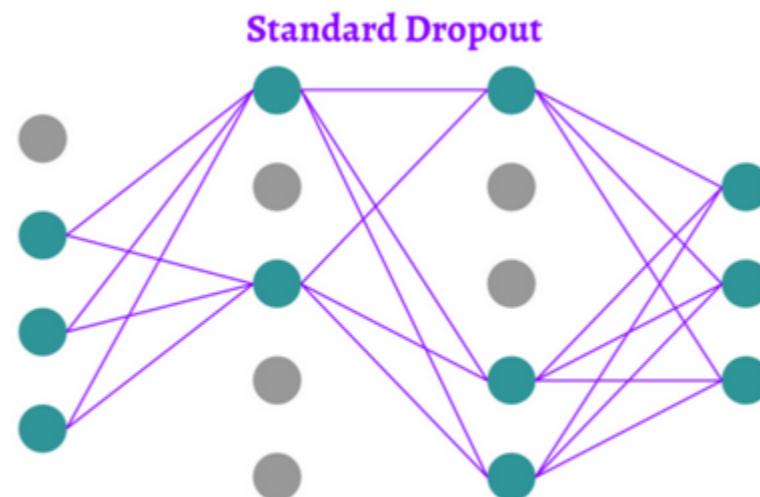
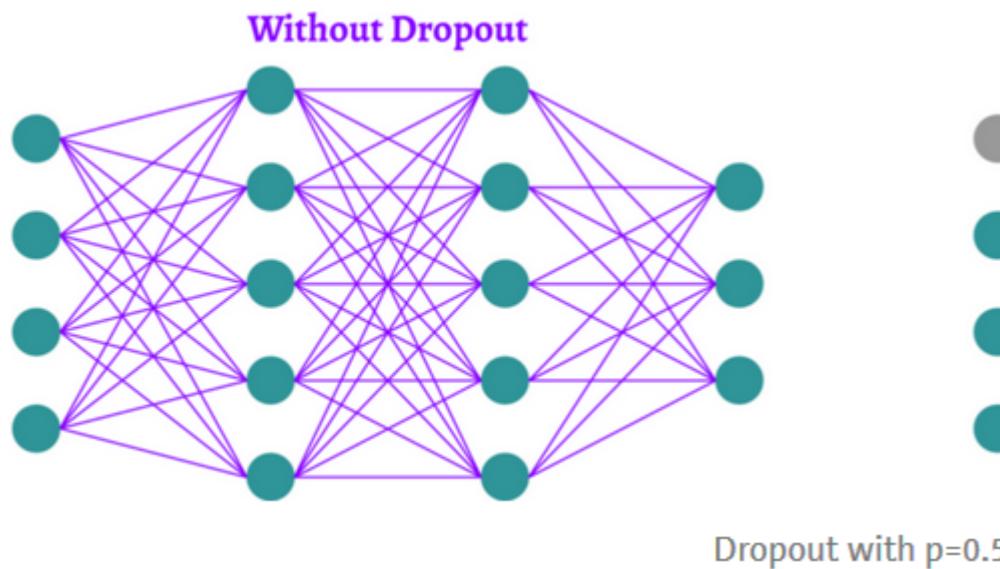
$$\| w_{\text{neuron}} \| \leq c$$

**если превысила – проекция**

Results from MNIST (handwritten digit recognition)

Method	Unit Type	Architecture	Error %
Standard Neural Net (Simard et al., 2003)	Logistic	2 layers, 800 units	1.60
SVM Gaussian kernel	NA	NA	1.40
Dropout NN	Logistic	3 layers, 1024 units	1.35
Dropout NN	ReLU	3 layers, 1024 units	1.25
Dropout NN + max-norm constraint	ReLU	3 layers, 1024 units	1.06
Dropout NN + max-norm constraint	ReLU	3 layers, 2048 units	1.04
Dropout NN + max-norm constraint	ReLU	2 layers, 4096 units	1.01
Dropout NN + max-norm constraint	ReLU	2 layers, 8192 units	0.95
Dropout NN + max-norm constraint (Goodfellow et al., 2013)	Maxout	2 layers, (5 × 240) units	0.94

<http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>

**(Standard) Dropout**

**обучение**  
 $y = f(Wx) \circ m$   
 $m \sim \text{Bernoulli}(1 - p)$

**тест**  
 $y = (1 - p)f(Wx)$

**«отключают в режиме теста»**

**случайное обнуление активаций**

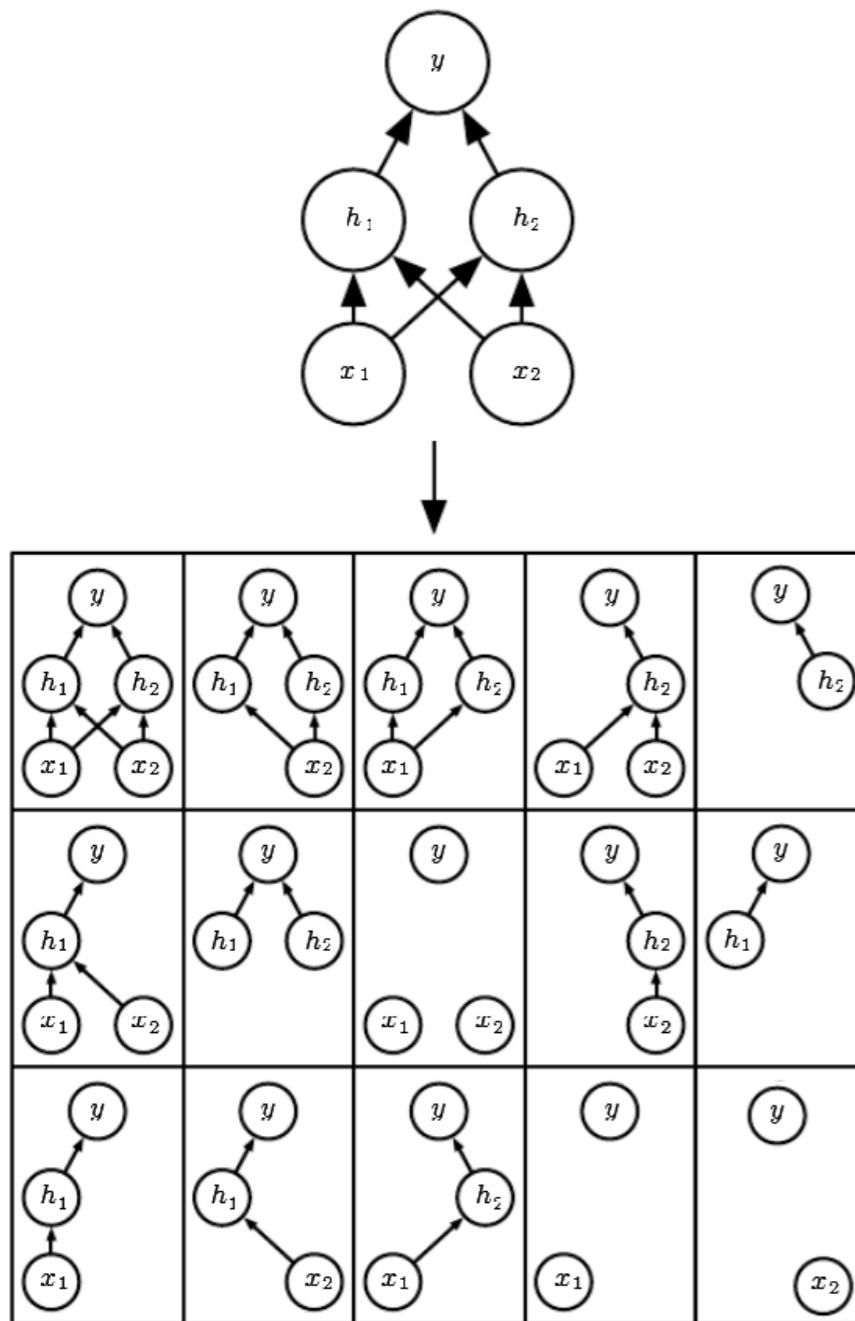
**~ выбрасывание нейронов из сети с вероятностью  $p$**

**Обычно в полносвязных слоях, до последнего!**

**Бонус: оценка уверенности в ответе (а не только регуляризация)  
также оценивают возможность удаления нейронов (Model Compression)**

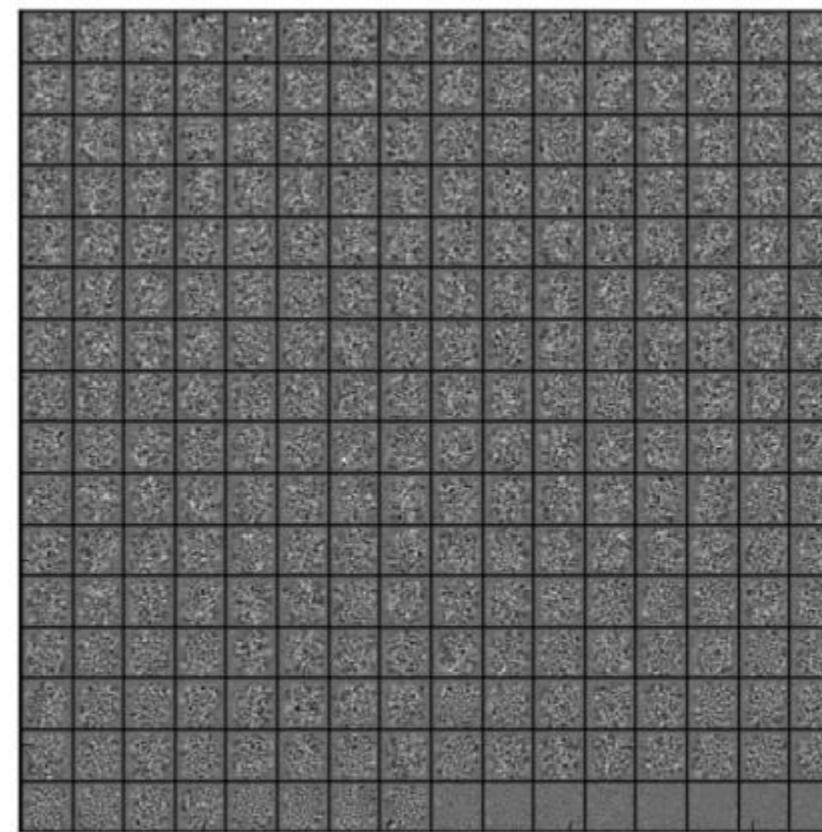
<https://towardsdatascience.com/12-main-dropout-methods-mathematical-and-visual-explanation-58cdc2112293>

## Dropout: обоснования

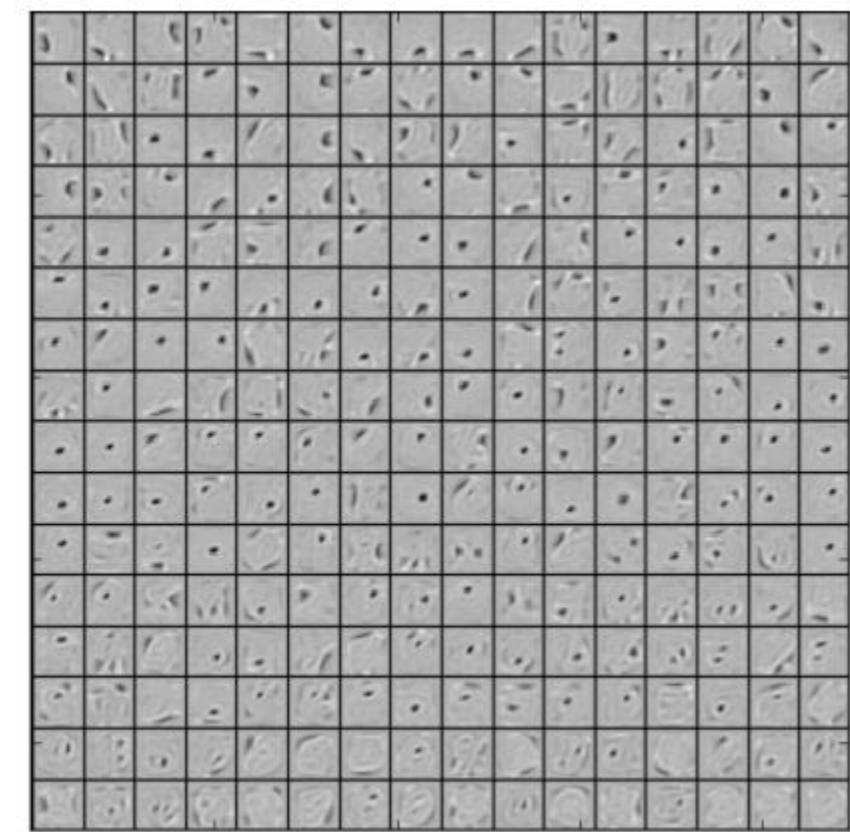


- аналогия с ансамблированием
  - байесовский подход
  - связь с регуляризацией / аугментацией / теорией информацией
  - уменьшение *co-adaptations / co-dependencies* соседних нейронов
  - большая робастность нейронов
- на рисунке:  
В отличие от бэгинга все модели делят параметры  
(не независимы) [DLbook]

## Dropout



(a) Without dropout

(b) Dropout with  $p = 0.5$ .

**MNIST: автокодировщик с 1 скрытым слоем и 256 линейными нейронами  
[Srivastava, 2014]**

## Inverted Dropout

### обучение

$$y = \frac{1}{1-p} f(Wx) \circ m$$

### Тест

$$y = f(Wx)$$

Во время тестирования ничего не делаем...

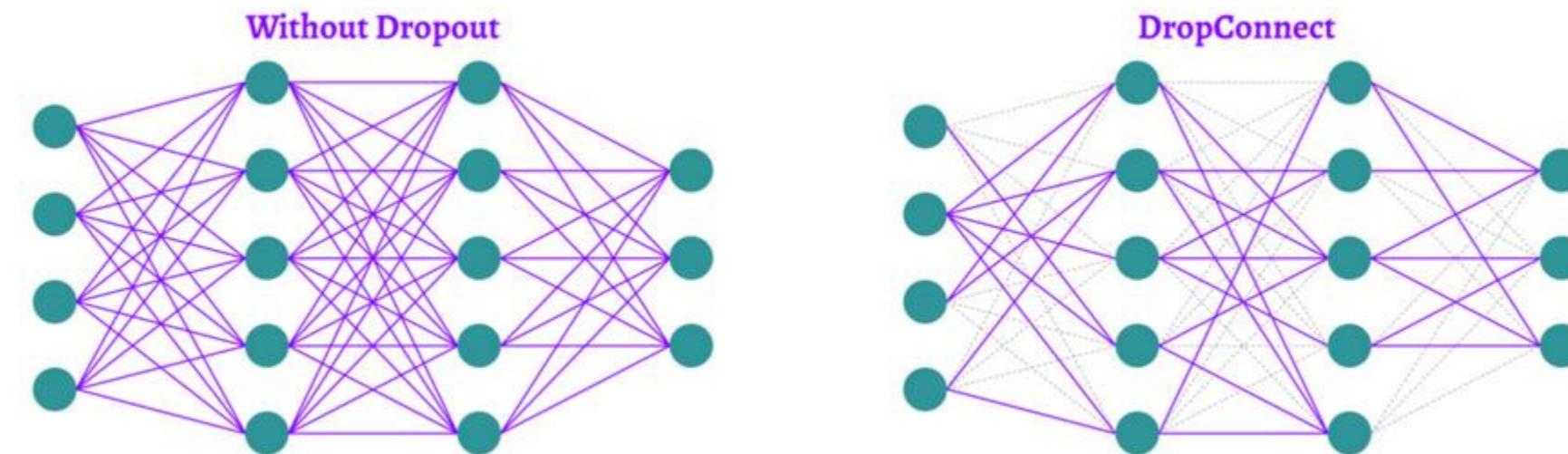
## Standout

$$m \sim \text{Bernoulli}(g(W_s x))$$

L. J. Ba and B. Frey, Adaptive dropout for training deep neural networks //

<https://papers.nips.cc/paper/5032-adaptive-dropout-for-training-deep-neural-networks.pdf>

## DropConnect



**Зануление отдельных весов, а не нейронов**

$$y = f((W \circ M)x)$$

<https://cs.nyu.edu/~wanli/dropc/>

<https://towardsdatascience.com/12-main-dropout-methods-mathematical-and-visual-explanation-58cdc2112293>

## DropConnect

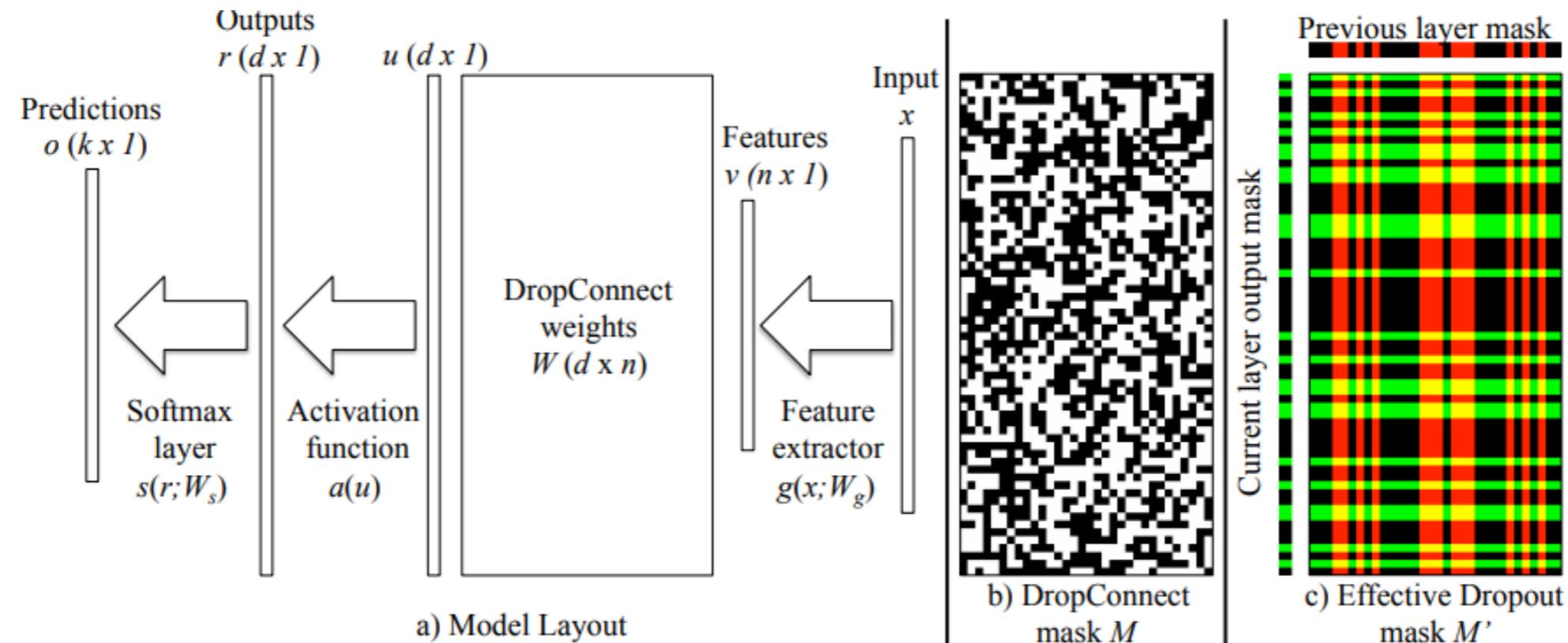
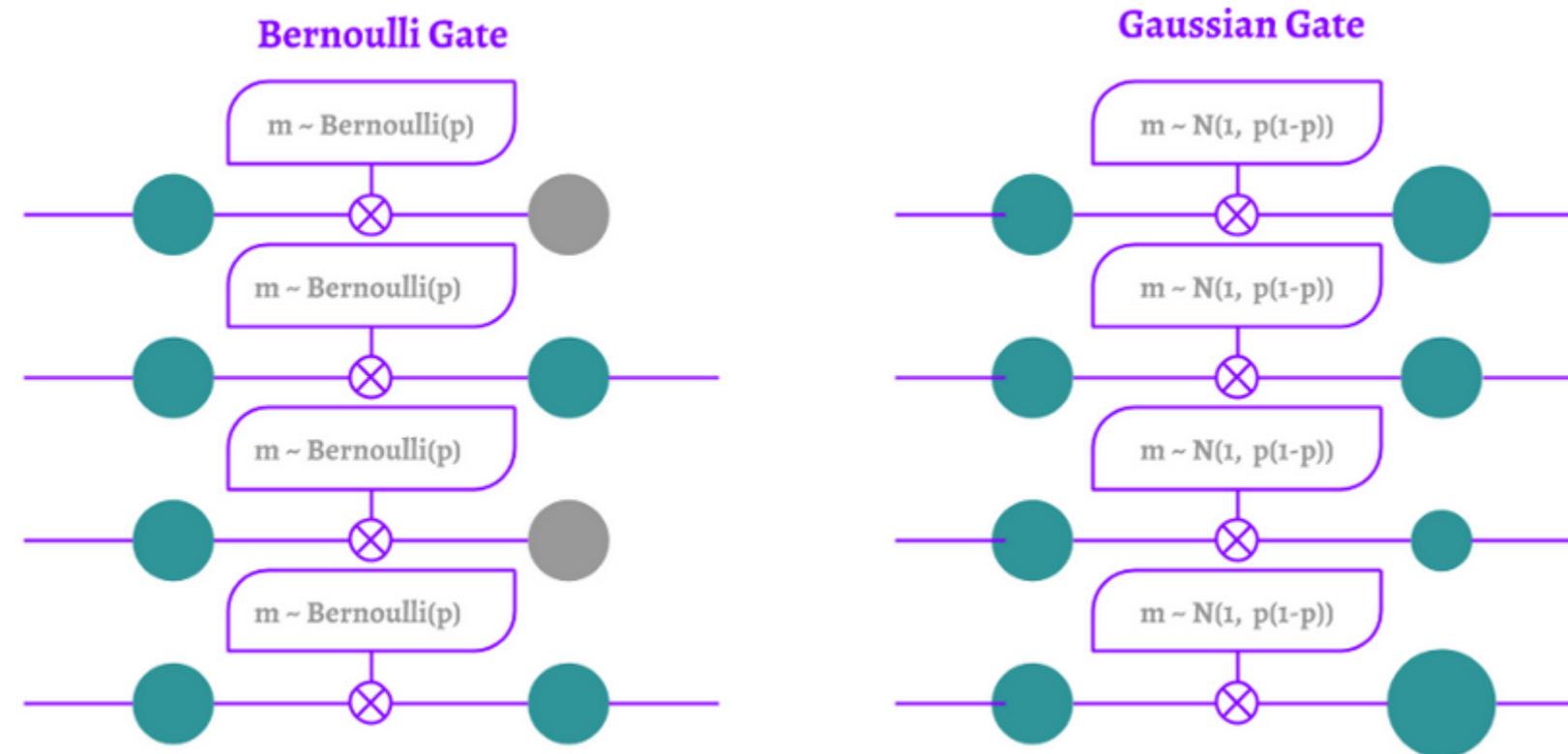


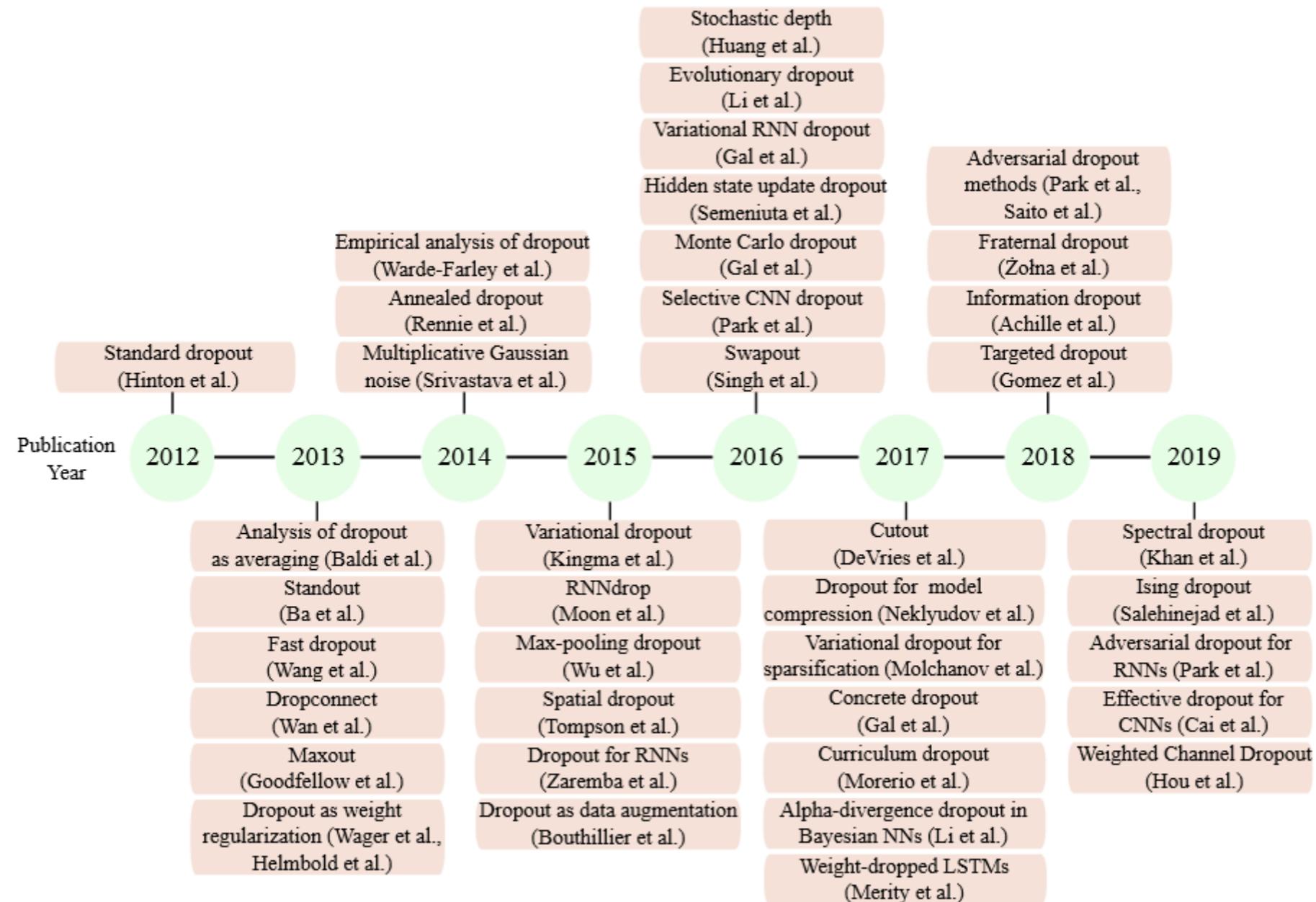
Figure 1. (a): An example model layout for a single DropConnect layer. After running feature extractor  $g()$  on input  $x$ , a random instantiation of the mask  $M$  (e.g. (b)), masks out the weight matrix  $W$ . The masked weights are multiplied with this feature vector to produce  $u$  which is the input to an activation function  $a$  and a softmax layer  $s$ . For comparison, (c) shows an effective weight mask for elements that Dropout uses when applied to the previous layer's output (red columns) and this layer's output (green rows). Note the lack of structure in (b) compared to (c).

## Gaussian Dropout



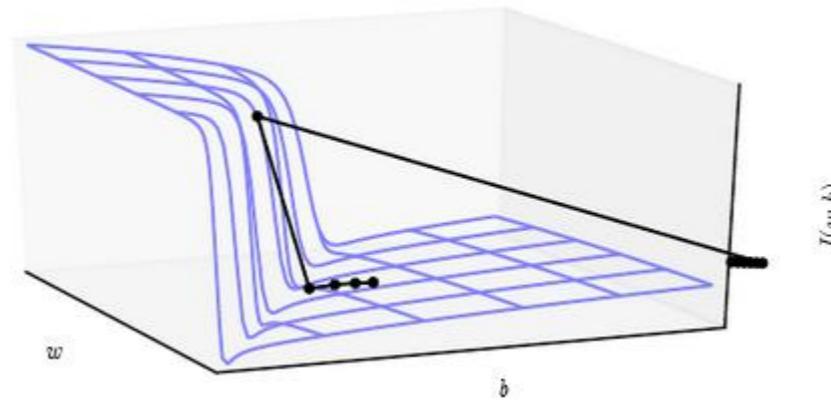
## Fast Dropout / Variational Dropout / Concrete Dropout

**Все эти приёмы неявной регуляризации – ухудшаем оптимизацию на train,  
но уменьшается переобучение**



## Обрезка градиентов (Gradient clipping)

**Если «попали на утёс», может слишком уйти от минимума**



**Выход – укоротить вектор градиента  
(направление сохраняется)**

$$g = \frac{\partial L}{\partial w}$$

$$g^{\text{new}} = \frac{\min(\theta, \|g\|)}{\|g\|} g$$

**Работает пока дисперсия градиента маленькая...**  
**(если градиент ~ колоколообразно распределён, то ясно как ограничить его,  
но если колокол начинает «расширяться» ...)**

**Pascanu, Mikolov, Bengio для RNN**

## Минутка кода

```
optimizer.zero_grad()  
loss, hidden = model(data, hidden, targets)  
loss.backward() # после этого  
  
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm)  
optimizer.step() # до этого
```

## Батч-нормализация (Batch normalization)

– метод адаптивной перепараметризации

**Проблема:**

**градиент – как изменять параметры,  
при условии, что вся остальная сеть **не меняется****

**трудно предсказать, насколько изменится какое-то значение  
(оно зависит от всех предыдущих в суперпозиции)**

**Covariate shift – изменение распределений входов во время обучения  
Надо уменьшить это изменение в скрытых слоях!**

**Ioffe and Szegedy, 2015 <https://arxiv.org/abs/1502.03167>**

## Батч-нормализация (Batch normalization)

**минибатч**  $\{x_i\}_{i=1}^m$

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i \text{ **среднее по мини-батчу**}$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2 \text{ **дисперсия по мини-батчу**}$$

$$x_i^{\text{new}} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \text{ **нормировка**}$$

$$y_i = \gamma x_i^{\text{new}} + \beta \text{ **растяжение и сдвиг**}$$

**Надо определить параметры  $\gamma$  и  $\beta$**

**Зачем центрировать, а потом смещать?**

**Так эффективнее обучать: смещение является параметром в чистом виде**

## Батч-нормализация (Batch normalization)

**При обучении:**

**среднее и дисперсия**

- по мини-батчу

**При teste:**

**среднее и дисперсия –**

- по обучению
- усреднение значений, что были во время обучения
- экспоненциальное среднее  $= \gamma \cdot \text{текущее} + (1 - \gamma) \cdot \text{предыдущее}$

**нормализация перед входом в каждый слой (иногда до активации, иногда после)**

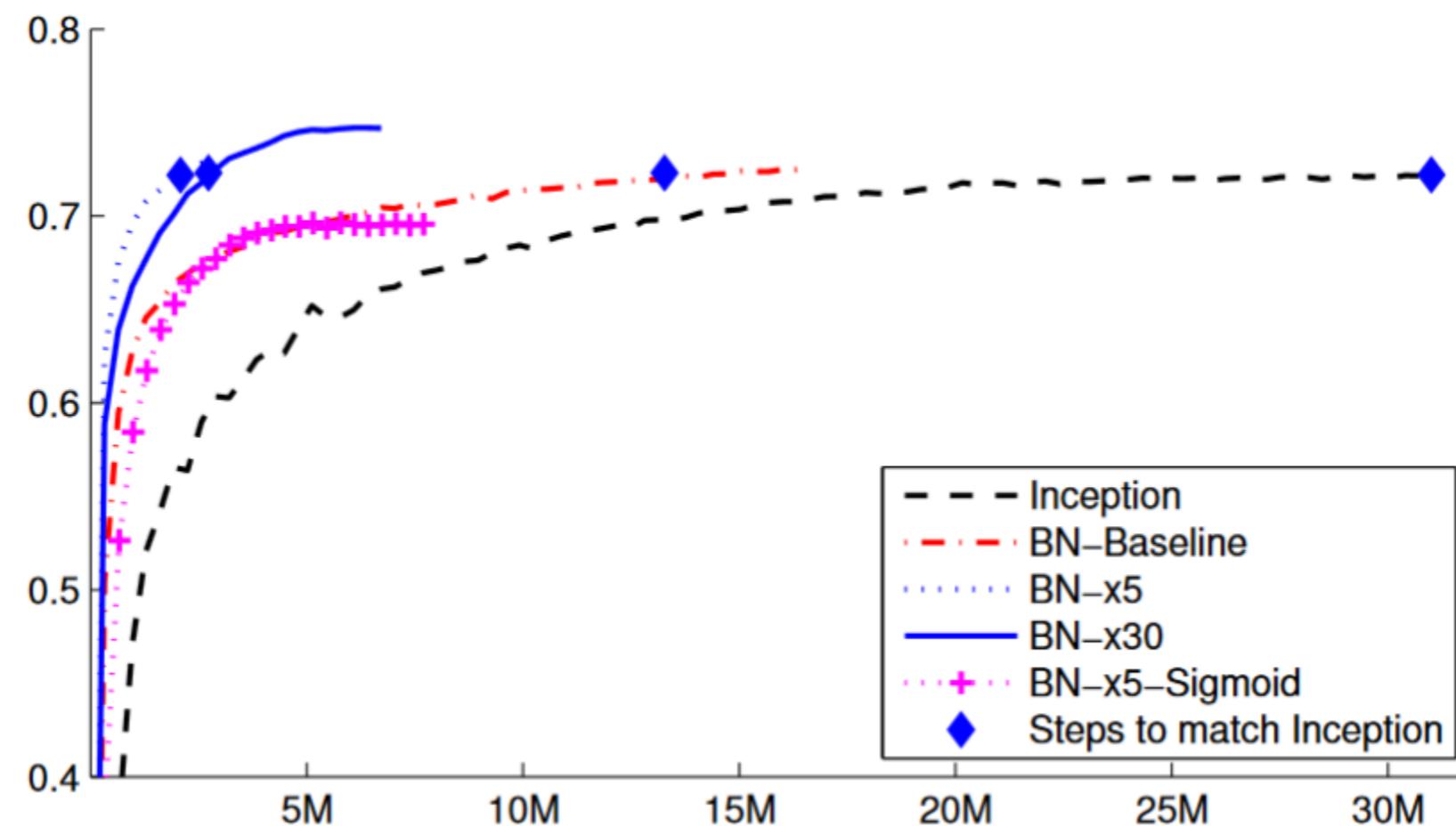
**если Linear + BN,  
то линейный слой делать без смещения**

## Батч-нормализация (Batch normalization)

- можно увеличить скорость обучения
  - можно убрать dropout
  - можно уменьшить регуляризацию и так есть шум из-за случайных мини-батчей
  - можно использовать более глубокие сети
  - меньше чувствительности к инициализации

**когда сеть обучена и не будет меняться м.б. BN можно устраниТЬ,  
например свёртка + BN = свёртка**

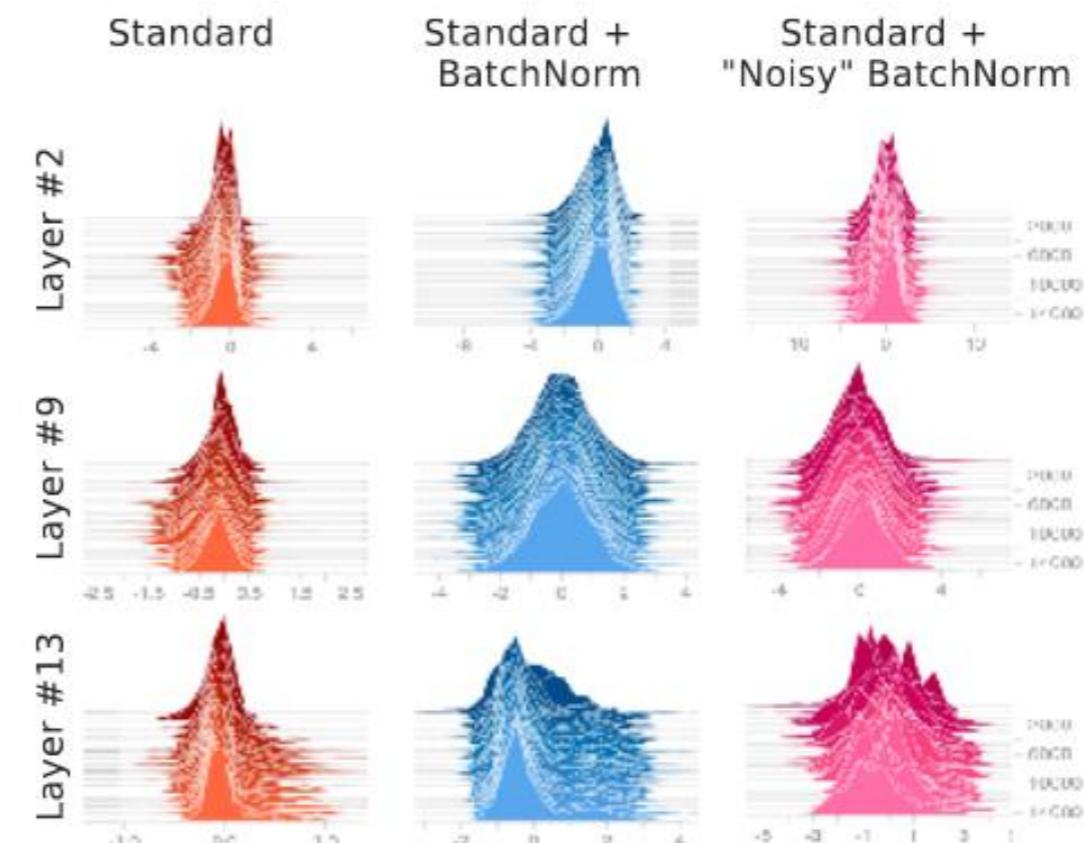
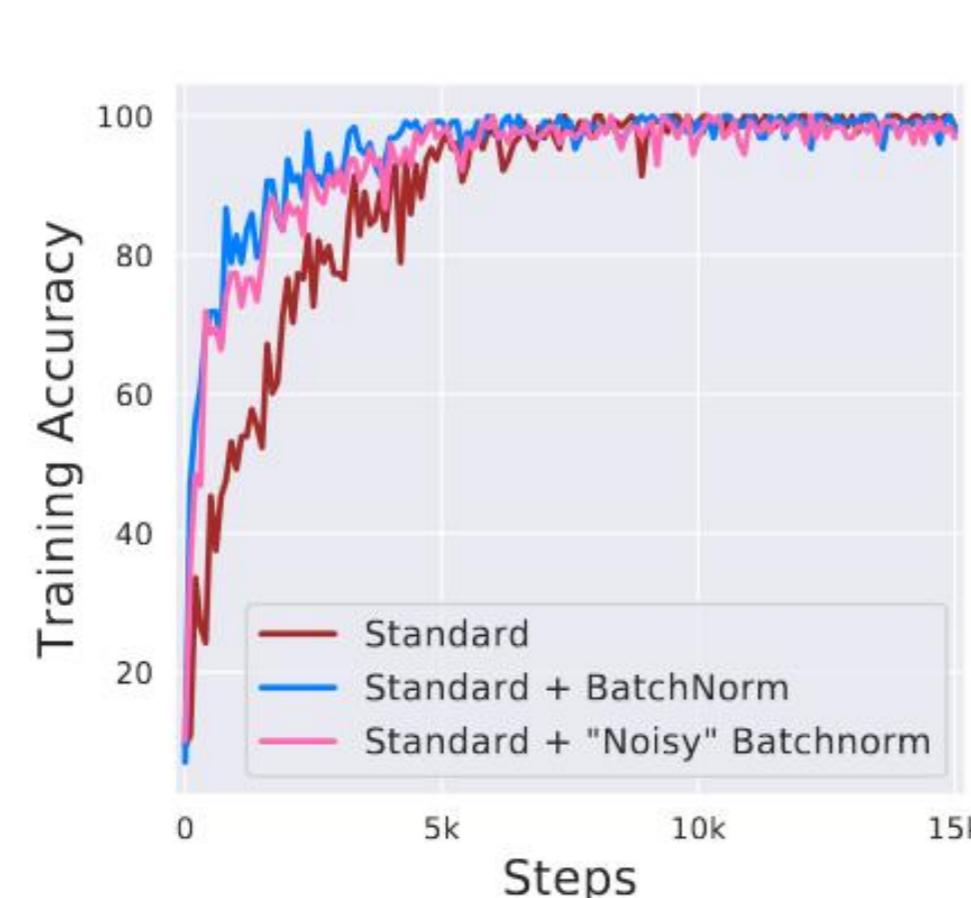
## Батч-нормализация (Batch normalization)



**Обучение Inception с / без батч-нормализацией  
+ варианты с увеличением темпа обучения**

## Батч-нормализация (Batch normalization)

– обоснование эффективности – открытая проблема



дело не в изменении распределения, а в сглаживании функции

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, Aleksander Madry

How Does Batch Normalization Help Optimization? 2018 // <https://arxiv.org/abs/1805.11604>

## Батч-нормализация (Batch normalization)

**DropOut и BatchNorm реализуются как отдельные слои:**

```
self.bn1 = nn.BatchNorm1d (num_features=n1, eps=1e-05,  
                         momentum=0.1, affine=True,  
                         track_running_stats=True)  
self.do1 = nn.Dropout(p=0.1, inplace=False)  
self.ln1 = nn.Linear(n1, n2)  
self.ph1 = nn.ReLU()
```

**Batch Normalization (BN) – если размер батча маленький, то всё плохо...**

мало статистики

а иногда необходимы / желательны маленькие батчи (bs=1)

Хотя есть мнение, что маленький батч сильнее регуляризирует!

далее ещё варианты нормализации

## Минутка кода

```
from torch import nn
H = torch.arange(1, 17).reshape(4, 4).float()

drop = nn.Dropout(p=0.5) # Dropout
print(H, drop(H))

tensor([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.],
       [ 9., 10., 11., 12.],
       [13., 14., 15., 16.]])  
  
tensor([[ 0.,  4.,  0.,  0.],
       [ 0.,  0., 14., 16.],
       [18.,  0., 22., 24.],
       [ 0., 28.,  0.,  0.]])  
  
bn = nn.BatchNorm1d(4, affine=False) # BN1D
print(bn(H))  
  
tensor([[-1.341, -1.341, -1.341, -1.341],
       [-0.447, -0.447, -0.447, -0.447],
       [ 0.447,  0.447,  0.447,  0.447],
       [ 1.341,  1.341,  1.341,  1.341]])
```

## Ещё нормализации

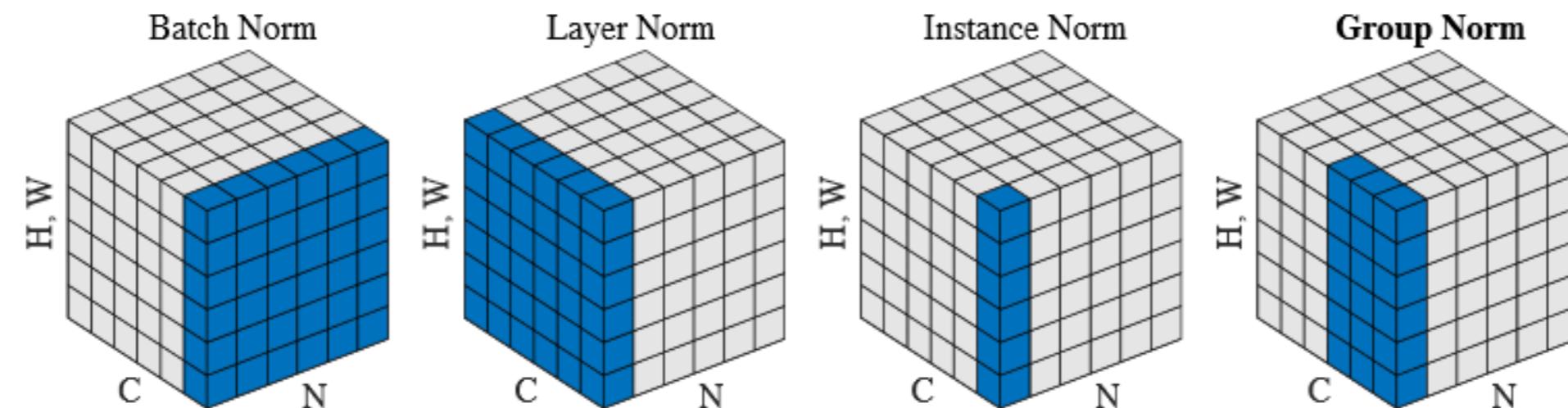


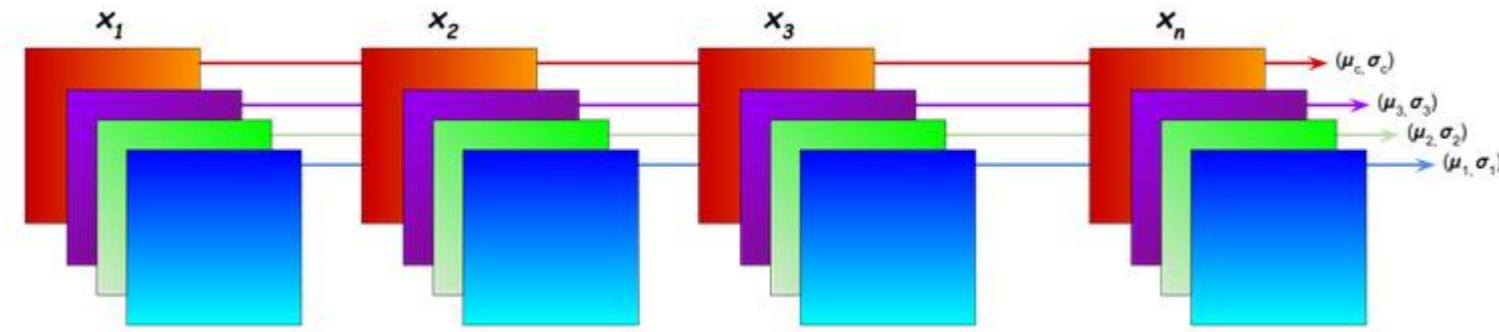
Figure 2. **Normalization methods.** Each subplot shows a feature map tensor, with  $N$  as the batch axis,  $C$  as the channel axis, and  $(H, W)$  as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

**Layer Normalization** – Ba, Kiros, and Hinton, «Layer Normalization», arXiv 2016

**Instance Normalization** – Ulyanov et al «Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis», CVPR 2017

**Group Normalization** – Wu and He, «Group Normalization» arXiv 2018

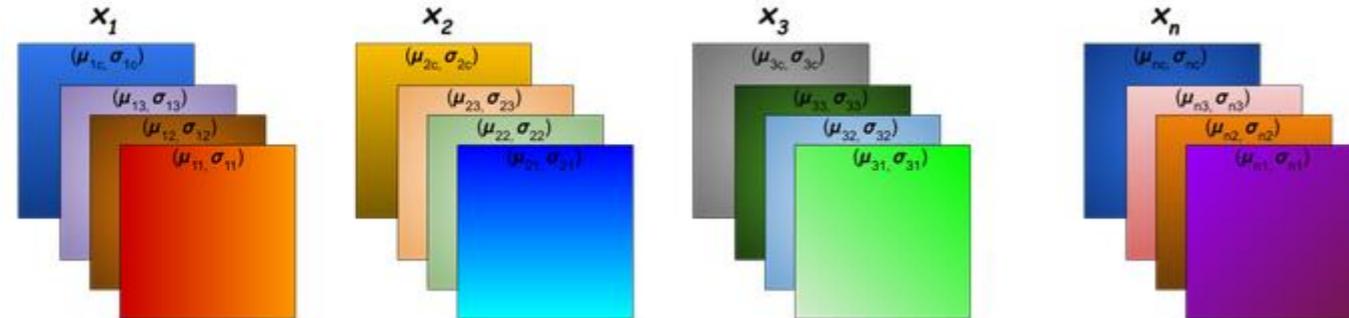
<https://becominghuman.ai/all-about-normalization-6ea79e70894b>

**BN – средние и дисперсии по каналам (признакам)**

$$\mu_c = \frac{1}{NHW} \sum_{i=1}^N \sum_{j=1}^H \sum_{k=1}^W x_{icjk}$$

$$\sigma_c^2 = \frac{1}{NHW} \sum_{i=1}^N \sum_{j=1}^H \sum_{k=1}^W (x_{icjk} - \mu_c)^2$$

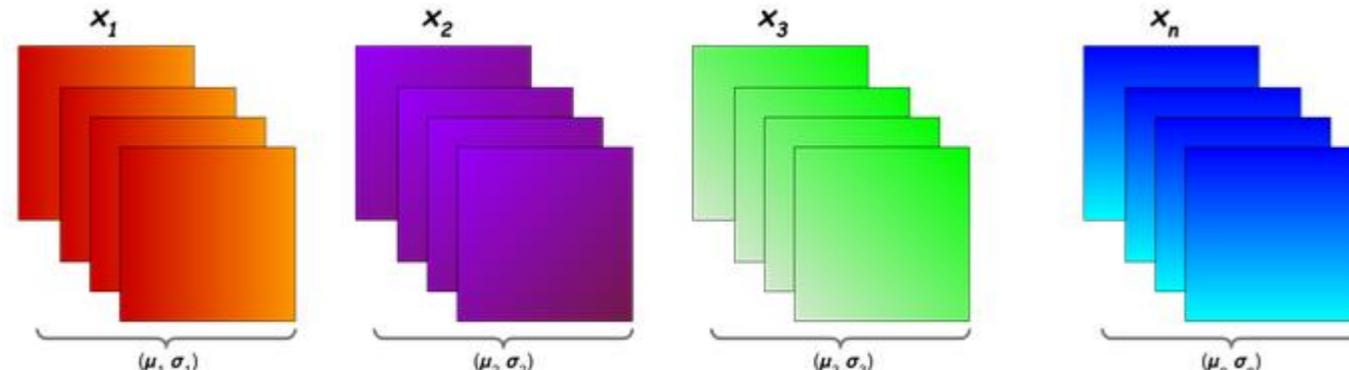
$$\hat{x} = \frac{x - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

**IN – для пары (объект, канал) (лучше ~style transfer)**

$$\mu_{nc} = \frac{1}{HW} \sum_{j=1}^H \sum_{k=1}^W x_{ncjk}$$

$$\sigma_{nc}^2 = \frac{1}{HW} \sum_{j=1}^H \sum_{k=1}^W (x_{ncjk} - \mu_{nc})^2$$

$$\hat{x} = \frac{x - \mu_{nc}}{\sqrt{\sigma_{nc}^2 + \epsilon}}$$

**LN – для объекта (лучше ~RNN)**

$$\mu_n = \frac{1}{CHW} \sum_{i=1}^C \sum_{j=1}^H \sum_{k=1}^W x_{nijk}$$

$$\sigma_n^2 = \frac{1}{CHW} \sum_{i=1}^C \sum_{j=1}^H \sum_{k=1}^W (x_{nijk} - \mu_n)^2$$

$$\hat{x} = \frac{x - \mu_n}{\sqrt{\sigma_n^2 + \epsilon}}$$

## Ещё нормализации

**Batch renormalization = BN + MA (4 inference)**

**Batch-Instance Normalization ~ ЛК batch normalization + instance normalization  
(коэффициент выучивается)**

## Минутка кода: если бы BN с нуля...

```
def batch_norm(X, gamma, beta, moving_mean, moving_var, eps, momentum):
    if not torch.is_grad_enabled():
        # prediction mode - directly use the mean and variance obtained by moving average
        X_hat = (X - moving_mean) / torch.sqrt(moving_var + eps)
    else:
        assert len(X.shape) in (2, 4)
        if len(X.shape) == 2:
            # fully-connected layer => calculate the mean and variance on the feature dimension
            mean = X.mean(dim=0)
            var = ((X - mean) ** 2).mean(dim=0)
        else:
            # two-dimensional convolutional layer => mean and variance on axis=1 (channel)
            mean = X.mean(dim=(0, 2, 3), keepdim=True)
            var = ((X - mean) ** 2).mean(dim=(0, 2, 3), keepdim=True)
        # In training mode, the current mean and variance are used for the standardization
        X_hat = (X - mean) / torch.sqrt(var + eps)
        # Update the mean and variance using moving average
        moving_mean = momentum * moving_mean + (1.0 - momentum) * mean
        moving_var = momentum * moving_var + (1.0 - momentum) * var
    Y = gamma * X_hat + beta # Scale and shift
    return Y, moving_mean.data, moving_var.data
```

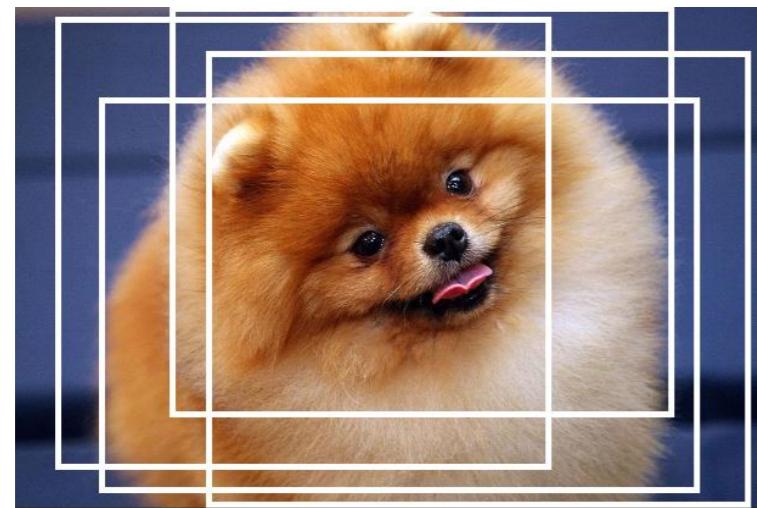
## Минутка кода: если бы BN с нуля...

```
class BatchNorm(nn.Module):
    # `num_features`: # outputs for a fully-connected layer or # output channels for a convolutional layer.
    # `num_dims`: 2 for a fully-connected layer and 4 for a convolutional layer
    def __init__(self, num_features, num_dims):
        super().__init__()
        if num_dims == 2:
            shape = (1, num_features)
        else:
            shape = (1, num_features, 1, 1)
        # The scale parameter and the shift parameter (model parameters)
        self.gamma = nn.Parameter(torch.ones(shape))
        self.beta = nn.Parameter(torch.zeros(shape))
        # The variables that are not model parameters are initialized to 0 and 1
        self.moving_mean = torch.zeros(shape)
        self.moving_var = torch.ones(shape)

    def forward(self, X):
        # copy `moving_mean` and `moving_var` to the device where `X` is located
        if self.moving_mean.device != X.device:
            self.moving_mean = self.moving_mean.to(X.device)
            self.moving_var = self.moving_var.to(X.device)
        # Save the updated `moving_mean` and `moving_var`
        Y, self.moving_mean, self.moving_var = batch_norm(X, self.gamma, self.beta, self.moving_mean,
                                                          self.moving_var, eps=1e-5, momentum=0.9)
        return Y
```

[https://d2l.ai/chapter\\_convolutional-modern/batch-norm.html](https://d2l.ai/chapter_convolutional-modern/batch-norm.html)

## Расширение обучающего множества (Data Augmentation)



### звук

- + фоновый шум
- тональность

### текст

- замена синонимов
- перестановки, удаления слов
- смесь с другим текстом
- преобразования (ex:  
переводчик и обратно)

**Аугментация – построение дополнительных данных из исходных**

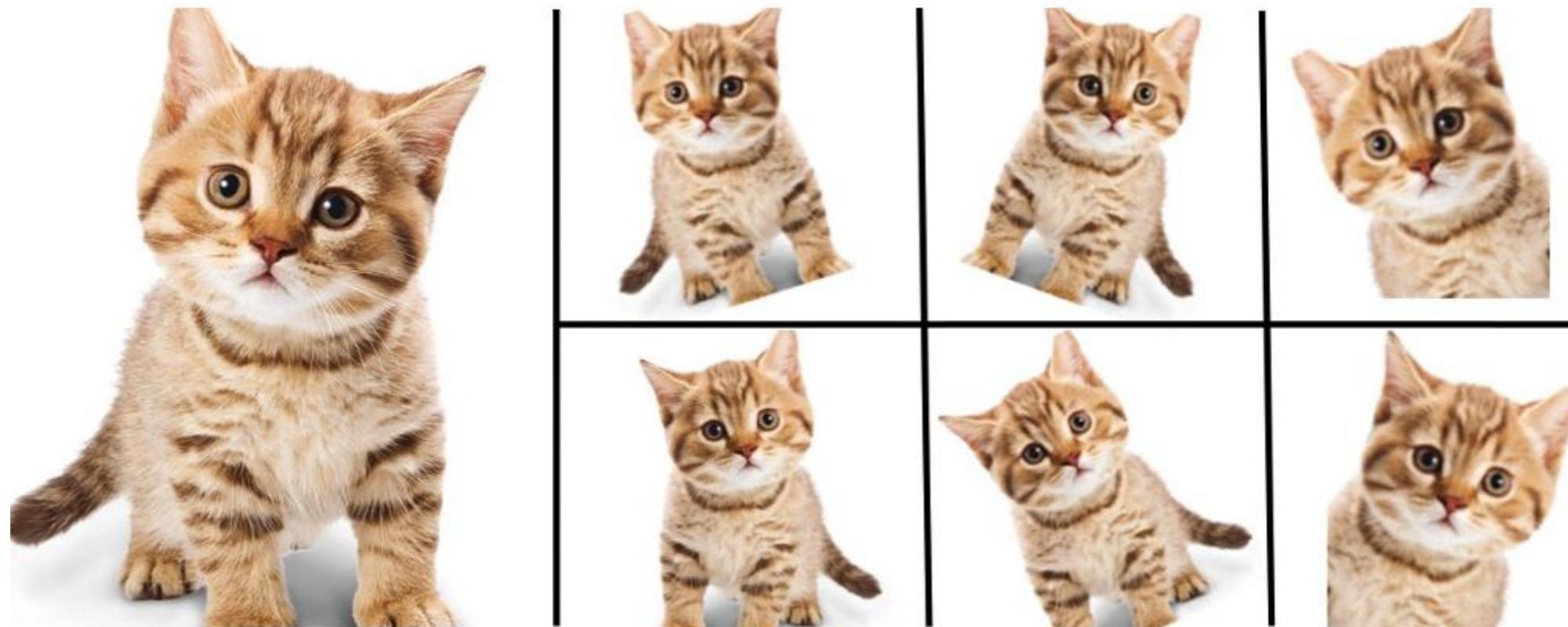
### изображения

- симметрии (flip)
- вырезки (crop)
- изменение масштаба (rescaling)
- случайные модификации (+шум)
- повороты (rotation)
- сдвиги (shift)
- изменение яркости, контраста, палитры
- эффекты линзы
- синтез / перерисовка изображений (ex GAN)

### Тонкость:

**преобразования могут переводить объект в другой класс, например повороты «6» и «9»**

## Расширение обучающего множества (Data Augmentation)



- **простая** (естественные изменения объектов, ех: небольшие повороты изображений)
- **агрессивная** («порча» объектов, ех: накладывание масок, объектов других классов)
- **креативная** (симуляция, GANы и т.п.)

<https://github.com/aleju/imgaug>

<https://medium.com/nanoneets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced>

## Расширение обучающего множества (Data Augmentation): код

```
import torch
import torchvision

transforms = torchvision.transforms.Compose([
    torchvision.transforms.Resize((224, 224)),
    torchvision.transforms.ColorJitter(hue=.05, saturation=.05),
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.RandomRotation(20, resample=PIL.Image.BILINEAR)
])

dataset = torchvision.datasets.ImageFolder('/data/' , transform=transforms)
```

**может быть online- и offline- аугментации**

**м.б. test time- аугментация (TTA)**

<https://colab.research.google.com/drive/109vu3F1LTzD1gdVV6cho9fKGx7lzbF11#scrollTo=wpWSjR-HBVso>

## Аугментация: Mixup

**Contribution** Motivated by these issues, we introduce a simple and data-agnostic data augmentation routine, termed *mixup* (Section 2). In a nutshell, *mixup* constructs virtual training examples

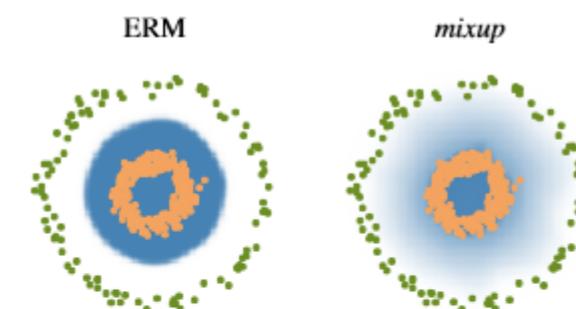
$$\tilde{x} = \lambda x_i + (1 - \lambda)x_j, \quad \text{where } x_i, x_j \text{ are raw input vectors}$$

$$\tilde{y} = \lambda y_i + (1 - \lambda)y_j, \quad \text{where } y_i, y_j \text{ are one-hot label encodings}$$

$(x_i, y_i)$  and  $(x_j, y_j)$  are two examples drawn at random from our training data, and  $\lambda \in [0, 1]$ .

```
# y1, y2 should be one-hot vectors
for (x1, y1), (x2, y2) in zip(loader1, loader2):
    lam = numpy.random.beta(alpha, alpha)
    x = Variable(lam * x1 + (1. - lam) * x2)
    y = Variable(lam * y1 + (1. - lam) * y2)
    optimizer.zero_grad()
    loss(net(x), y).backward()
    optimizer.step()
```

(a) One epoch of *mixup* training in PyTorch.



(b) Effect of *mixup* ( $\alpha = 1$ ) on a toy problem. Green: Class 0. Orange: Class 1. Blue shading indicates  $p(y = 1|x)$ .

Figure 1: Illustration of *mixup*, which converges to ERM as  $\alpha \rightarrow 0$ .

<https://github.com/facebookresearch/mixup-cifar10>

<https://arxiv.org/abs/1710.09412>

## Аугментация: Mixup

Model	Method	Epochs	Top-1 Error	Top-5 Error
ResNet-50	ERM (Goyal et al., 2017)	90	23.5	-
	<i>mixup</i> $\alpha = 0.2$	90	<b>23.3</b>	<b>6.6</b>
ResNet-101	ERM (Goyal et al., 2017)	90	22.1	-
	<i>mixup</i> $\alpha = 0.2$	90	<b>21.5</b>	<b>5.6</b>
ResNeXt-101 32*4d	ERM (Xie et al., 2016)	100	21.2	-
	ERM	90	21.2	5.6
	<i>mixup</i> $\alpha = 0.4$	90	<b>20.7</b>	<b>5.3</b>
ResNeXt-101 64*4d	ERM (Xie et al., 2016)	100	20.4	5.3
	<i>mixup</i> $\alpha = 0.4$	90	<b>19.8</b>	<b>4.9</b>
ResNet-50	ERM	200	23.6	7.0
	<i>mixup</i> $\alpha = 0.2$	200	<b>22.1</b>	<b>6.1</b>
ResNet-101	ERM	200	22.0	6.1
	<i>mixup</i> $\alpha = 0.2$	200	<b>20.8</b>	<b>5.4</b>
ResNeXt-101 32*4d	ERM	200	21.3	5.9
	<i>mixup</i> $\alpha = 0.4$	200	<b>20.1</b>	<b>5.0</b>

Table 1: Validation errors for ERM and *mixup* on the development set of ImageNet-2012.

**SOTA:**  
**CIFAR-10**  
**CIFAR-100**  
**ImageNet-2012**

- **learning from corrupt labels**
- **facing adversarial examples**
- **generalization on speech / tabular data**
- **to stabilize the training of GANs**

## Аугментация: Cutout, CutMix

	ResNet-50	Mixup	Cutout	CutMix
Image				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4
ImageNet Cls (%)	76.3 (+0.0)	77.4 (+1.1)	77.1 (+0.8)	<b>78.4</b> <b>(+2.1)</b>
ImageNet Loc (%)	46.3 (+0.0)	45.8 (-0.5)	46.7 (+0.4)	<b>47.3</b> <b>(+1.0)</b>
Pascal VOC Det (mAP)	75.6 (+0.0)	73.9 (-1.7)	75.1 (-0.5)	<b>76.7</b> <b>(+1.1)</b>

Sangdoo Yun et al «CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features» // <https://arxiv.org/pdf/1905.04899.pdf>

## Аугментация: Cutout, CutMix

Let  $x \in \mathbb{R}^{W \times H \times C}$  and  $y$  denote a training image and its label, respectively. The goal of CutMix is to generate a new training sample  $(\tilde{x}, \tilde{y})$  by combining two training samples  $(x_A, y_A)$  and  $(x_B, y_B)$ . The generated training sample  $(\tilde{x}, \tilde{y})$  is used to train the model with its original loss function. We define the combining operation as

$$\begin{aligned}\tilde{x} &= \mathbf{M} \odot x_A + (1 - \mathbf{M}) \odot x_B \\ \tilde{y} &= \lambda y_A + (1 - \lambda) y_B,\end{aligned}\tag{1}$$

where  $\mathbf{M} \in \{0, 1\}^{W \times H}$  denotes a binary mask indicating where to drop out and fill in from two images,  $\mathbf{1}$  is a binary mask filled with ones, and  $\odot$  is element-wise multiplication. Like Mixup [48], the combination ratio  $\lambda$  between two data points is sampled from the beta distribution  $\text{Beta}(\alpha, \alpha)$ . In our all experiments, we set  $\alpha$  to 1, that is  $\lambda$  is sampled from the uniform distribution  $(0, 1)$ . Note that the major difference is that CutMix replaces an image region with a patch from another training image and generates more locally natural image than Mixup does.

```
lam = np.random.beta(args.beta, args.beta)
rand_index = torch.randperm(input.size()[0]).cuda()
target_a = target
target_b = target[rand_index]
bbx1, bby1, bbx2, bby2 = rand_bbox(input.size(), lam)
input[:, :, bbx1:bbx2, bby1:bby2] = input[rand_index, :, bbx1:bbx2, bby1:bby2]

# adjust lambda to exactly match pixel ratio
lam = 1 - ((bbx2 - bbx1) * (bby2 - bby1) /
            (input.size()[-1] * input.size()[-2]))
output = model(input)
loss = criterion(output, target_a) * lam +
       criterion(output, target_b) * (1. - lam)
```

**есть несоответствие между кодом и статьёй**

## Аугментация: Attentive CutMix

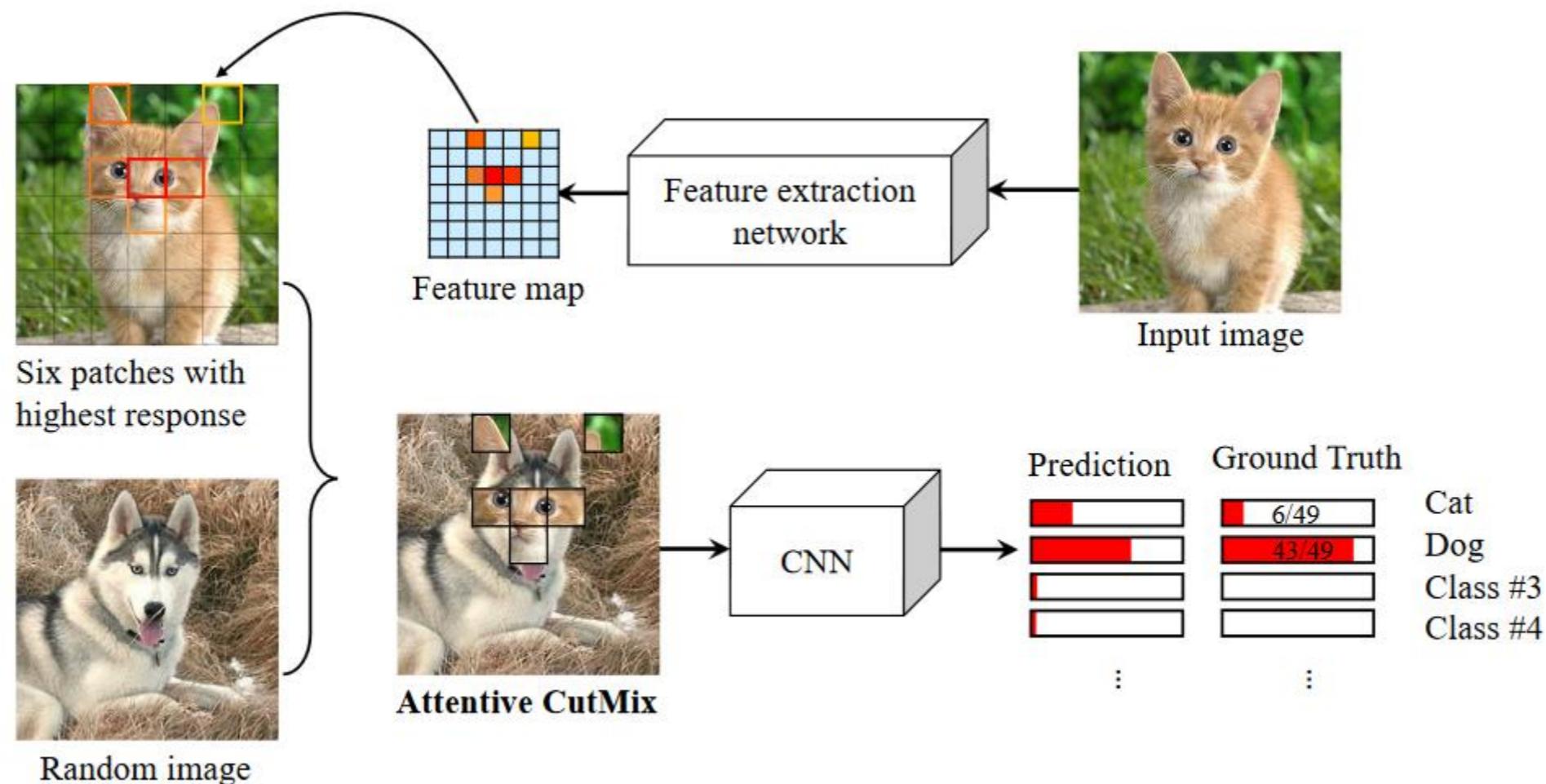


Figure 2: Framework overview of proposed *Attentive CutMix*.

**Devesh Walawalkar, Zhiqiang Shen, Zechun Liu, Marios Savvides «Attentive CutMix: An Enhanced Data Augmentation Approach for Deep Learning Based Image Classification» //**

<https://arxiv.org/pdf/2003.13048.pdf>

## Аугментация: GridMask

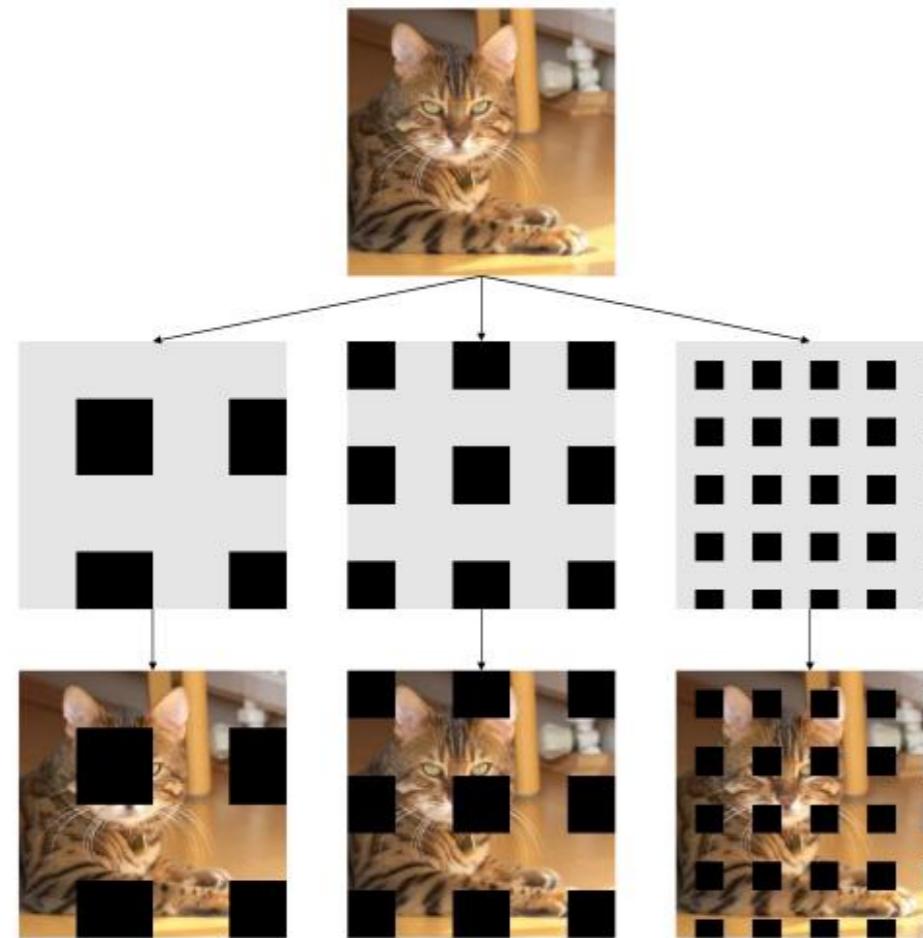


Figure 3. This image shows examples of GridMask. First, we produce a mask according to the given parameters  $(r, d, \delta_x, \delta_y)$ . Then we multiply it with the input image. The result is shown in the last row. In the mask, gray value is 1, representing the reserved regions; black value is 0, for regions to be deleted.

Model	Accuracy (%)
ResNet18 [10]	95.28
+ Randomerasing [29]	95.32
+ Cutout [4]	96.25
+ HaS [17]	96.10
+ AutoAugment [3]	96.07
+ GridMask (Ours)	96.54
+ AutoAugment & Cutout [3]	96.51
+ AutoAugment & GridMask (Ours)	<b>96.64</b>
WideResNet-28-10 [24]	96.13
+ Radnomerasing [29]*	96.92
+ Cutout [4]	97.04
+ HaS [17]	96.94
+ AutoAugment [3]	97.01
+ GridMask (Ours)	97.24
+ AutoAugment & Cutout [3]	97.39
+ AutoAugment & GridMask (Ours)	<b>97.48</b>
ShakeShake-26-32 [5]	96.42
+ Randomerasing [29]	96.46
+ Cutout [4]	96.96
+ Has [17]	96.89
+ Autoaugment [3]	96.96
+ GridMask (Ours)	97.20
+ AutoAugment & Cutout [3]	97.36
+ AutoAugment & GridMask (Ours)	<b>97.42</b>

Table 3. Results on CIFAR10 are summarized in this table. We achieve the best accuracy on different models. \* means results reported in the original paper.

P Chen «GridMask Data Augmentation» <https://arxiv.org/pdf/2001.04086.pdf>

## Ансамбль нейросетей

- **псевдо-ансамбль (pseudo-ensemble) (общая свёрточная часть, разные FC-блоки)**
- **несколько независимых моделей, усредняем результат (как всегда +2%)**
- **несколько независимых моделей, усредняем веса (не работает п.в.)**
- **Snapshot Ensemble – усреднение одной НС на разных эпохах обучения (аналог усреднения Поляка)**
- **усреднение параметров на одной траектории оптимизации – Stochastic Weight Averaging (SWA)**

+ более сложные ансамбли...

Model	Prediction method	Test Accuracy
Baseline (10 epochs)	Single model	0.837
True ensemble of 10 models	Average predictions	0.855
True ensemble of 10 models	Voting	0.851
Snapshots (25) over 10 epochs	Average predictions	0.865
Snapshots (25) over 10 epochs	Voting	0.861
Snapshots (25) over 10 epochs	Parameter averaging	0.864

## Ансамбль нейросетей: Snapshot Ensembles

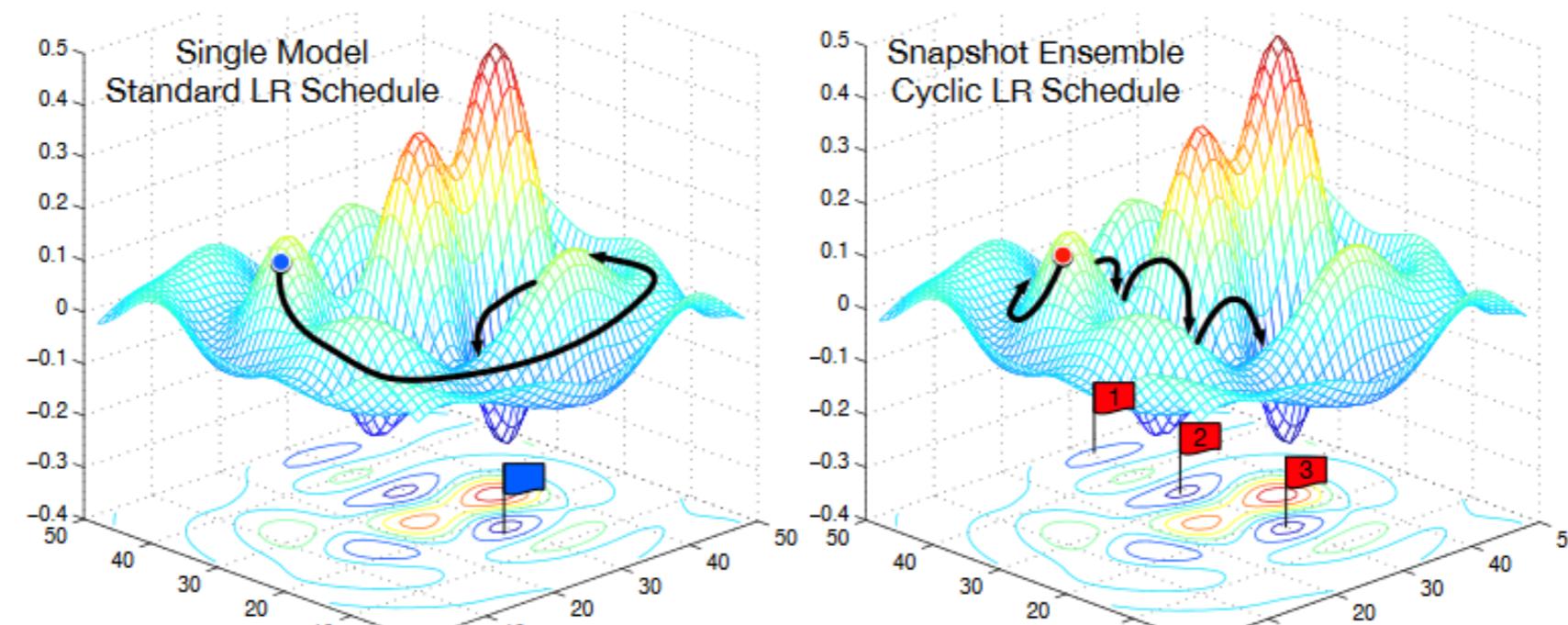


Figure 1: **Left:** Illustration of SGD optimization with a typical learning rate schedule. The model converges to a minimum at the end of training. **Right:** Illustration of Snapshot Ensembling. The model undergoes several learning rate annealing cycles, converging to and escaping from multiple local minima. We take a snapshot at each minimum for test-time ensembling.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, Kilian Q. Weinberger «**Snapshot Ensembles: Train 1, get M for free**» // <https://arxiv.org/pdf/1704.00109.pdf>

## Ансамбль нейросетей : Stochastic Weight Averaging (SWA)

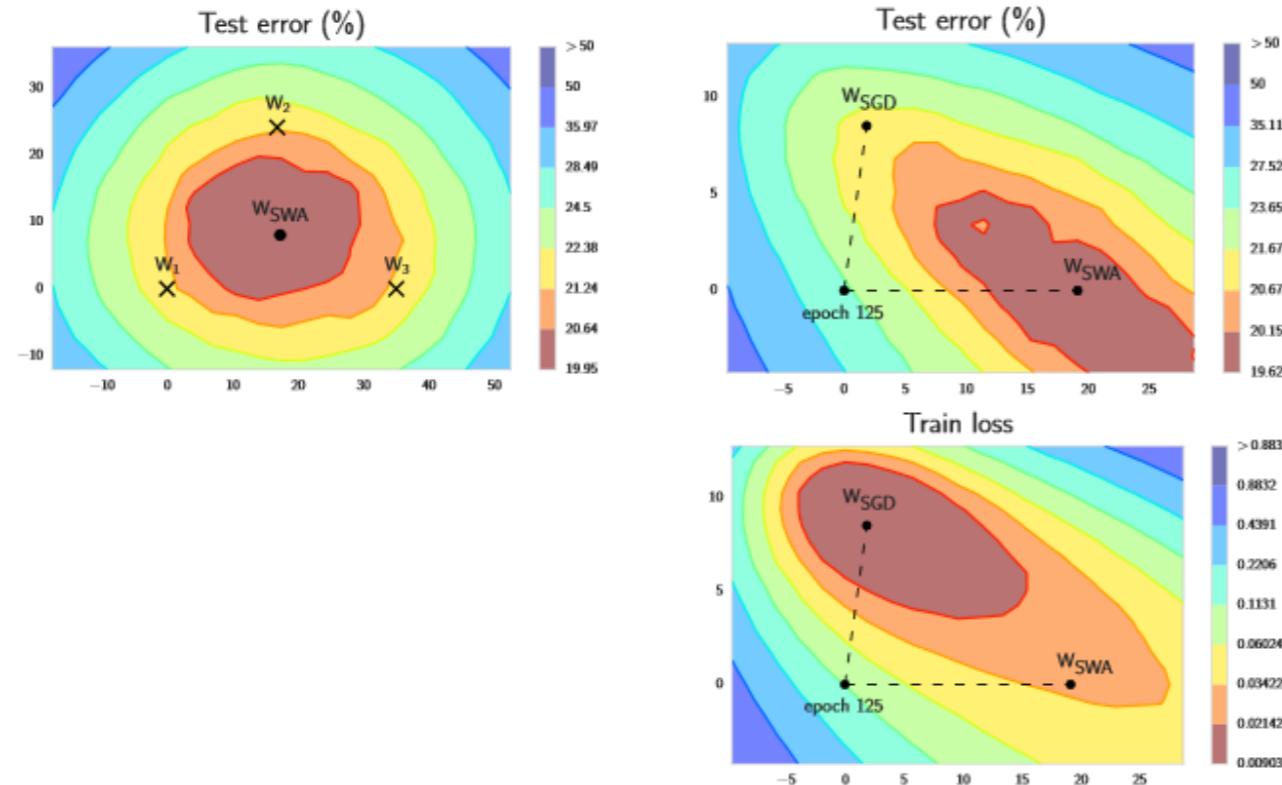


Figure 1: Illustrations of SWA and SGD with a Preactivation ResNet-164 on CIFAR-100<sup>1</sup>. **Left:** test error surface for three FGE samples and the corresponding SWA solution (averaging in weight space). **Middle and Right:** test error and train loss surfaces showing the weights proposed by SGD (at convergence) and SWA, starting from the same initialization of SGD after 125 training epochs.

### Algorithm 1 Stochastic Weight Averaging

**Require:**

weights  $\hat{w}$ , LR bounds  $\alpha_1, \alpha_2$ , cycle length  $c$  (for constant learning rate  $c = 1$ ), number of iterations  $n$

**Ensure:**  $w_{SWA}$

$w \leftarrow \hat{w}$  {Initialize weights with  $\hat{w}$ }

$w_{SWA} \leftarrow w$

**for**  $i \leftarrow 1, 2, \dots, n$  **do**

$\alpha \leftarrow \alpha(i)$  {Calculate LR for the iteration}

$w \leftarrow w - \alpha \nabla \mathcal{L}_i(w)$  {Stochastic gradient update}

**if**  $\text{mod}(i, c) = 0$  **then**

$n_{\text{models}} \leftarrow i/c$  {Number of models}

$w_{SWA} \leftarrow \frac{w_{SWA} \cdot n_{\text{models}} + w}{n_{\text{models}} + 1}$  {Update average}

**end if**

**end for**

{Compute BatchNorm statistics for  $w_{SWA}$  weights}

Pavel Izmailov et. al. «Averaging Weights Leads to Wider Optima and Better Generalization» //

<https://arxiv.org/abs/1803.05407>

## Ансамбль нейросетей: Stochastic Weight Averaging (SWA)

Table 2: Top-1 accuracies (%) on ImageNet for SWA and SGD with different architectures.

DNN	SGD	SWA	
		5 epochs	10 epochs
ResNet-50	76.15	76.83 ± 0.01	76.97 ± 0.05
ResNet-152	78.31	78.82 ± 0.01	78.94 ± 0.07
DenseNet-161	77.65	78.26 ± 0.09	78.44 ± 0.06

For all 3 architectures SWA provides consistent improvement by 0.6-0.9% over the pretrained models.

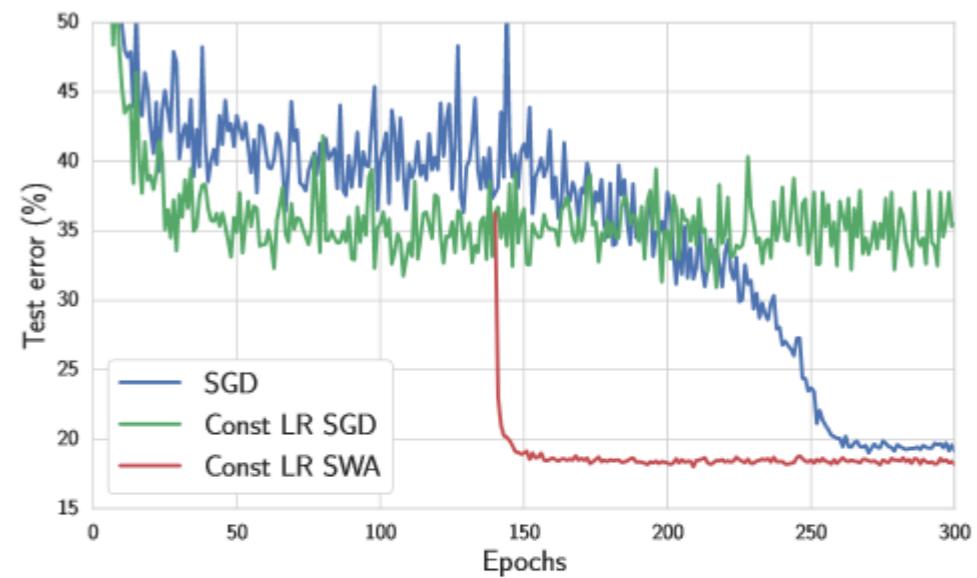


Figure 7: Test error as a function of training epoch for constant (green) and decaying (blue) learning rate schedules for a Wide ResNet-28-10 on CIFAR-100. In red we average the points along the trajectory of SGD with constant learning rate starting at epoch 140.

## Ансамбль нейросетей: более сложные

---

**Algorithm 1** Pseudocode of the training procedure for our method

---

- 1: ▷ Let each neural network parametrize a distribution over the outputs, i.e.  $p_\theta(y|\mathbf{x})$ . Use a proper scoring rule as the training criterion  $\ell(\theta, \mathbf{x}, y)$ . Recommended default values are  $M = 5$  and  $\epsilon = 1\%$  of the input range of the corresponding dimension (e.g 2.55 if input range is [0,255]).
  - 2: Initialize  $\theta_1, \theta_2, \dots, \theta_M$  randomly
  - 3: **for**  $m = 1 : M$  **do** ▷ train networks independently in parallel
  - 4:   Sample data point  $n_m$  randomly for each net ▷ single  $n_m$  for clarity, minibatch in practice
  - 5:   Generate adversarial example using  $\mathbf{x}'_{n_m} = \mathbf{x}_{n_m} + \epsilon \text{ sign}(\nabla_{\mathbf{x}_{n_m}} \ell(\theta_m, \mathbf{x}_{n_m}, y_{n_m}))$
  - 6:   Minimize  $\ell(\theta_m, \mathbf{x}_{n_m}, y_{n_m}) + \ell(\theta_m, \mathbf{x}'_{n_m}, y_{n_m})$  w.r.t.  $\theta_m$  ▷ adversarial training (optional)
- 

We treat the ensemble as a uniformly-weighted mixture model and combine the predictions as  $p(y|\mathbf{x}) = M^{-1} \sum_{m=1}^M p_{\theta_m}(y|\mathbf{x}, \theta_m)$ . For classification, this corresponds to averaging the predicted probabilities. For regression, the prediction is a mixture of Gaussian distributions. For ease of computing quantiles and predictive probabilities, we further approximate the ensemble prediction as a Gaussian whose mean and variance are respectively the mean and variance of the mixture. The mean and variance of a mixture  $M^{-1} \sum \mathcal{N}(\mu_{\theta_m}(\mathbf{x}), \sigma_{\theta_m}^2(\mathbf{x}))$  are given by  $\mu_*(\mathbf{x}) = M^{-1} \sum_m \mu_{\theta_m}(\mathbf{x})$  and  $\sigma_*^2(\mathbf{x}) = M^{-1} \sum_m (\sigma_{\theta_m}^2(\mathbf{x}) + \mu_{\theta_m}^2(\mathbf{x})) - \mu_*^2(\mathbf{x})$  respectively.

### Использование «состязательных примеров»

Balaji Lakshminarayanan «Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles» // <https://arxiv.org/pdf/1612.01474.pdf>

## Диагностика проблем с НС

- 1. Численная проверка**
- 2. Визуализация**
- 3. Маленькая выборка**
- 4. Насыщенность нейронов**
- 5. Динамика ошибки**
- 6. Изменения весов**

## Диагностика проблем с НС: Численная проверка

### 1. Численно проверить градиенты (с помощью конечных разностей)

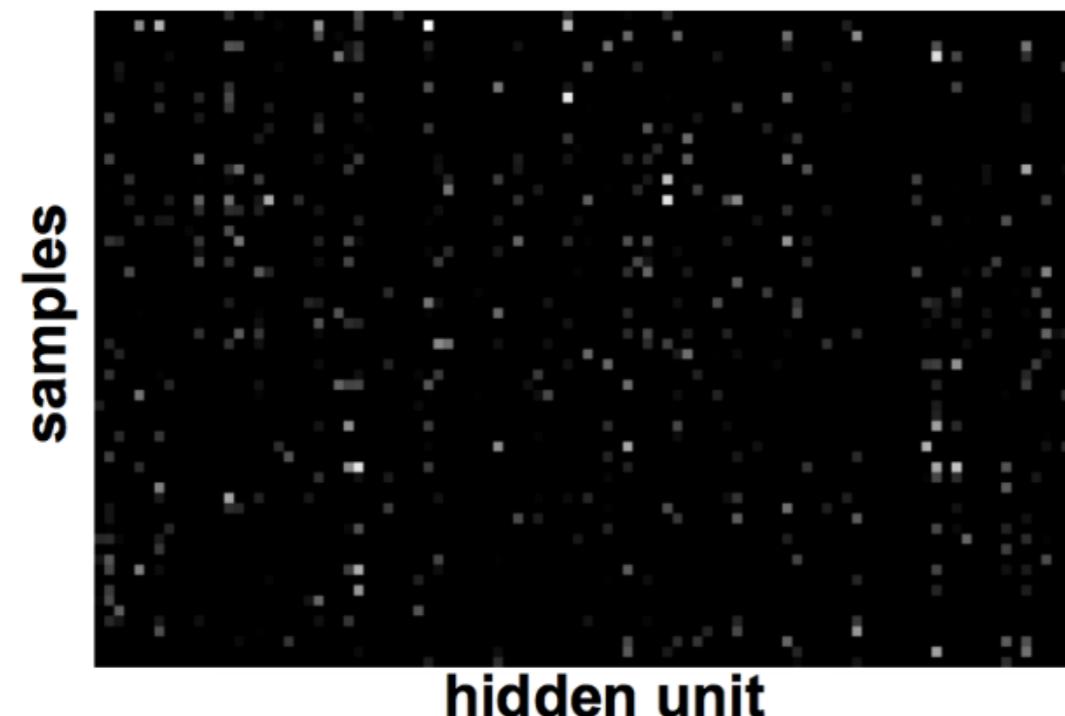
$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x + \varepsilon) - f(x - \varepsilon)}{2\varepsilon}$$

проверка реализации обратного и прямого распространения

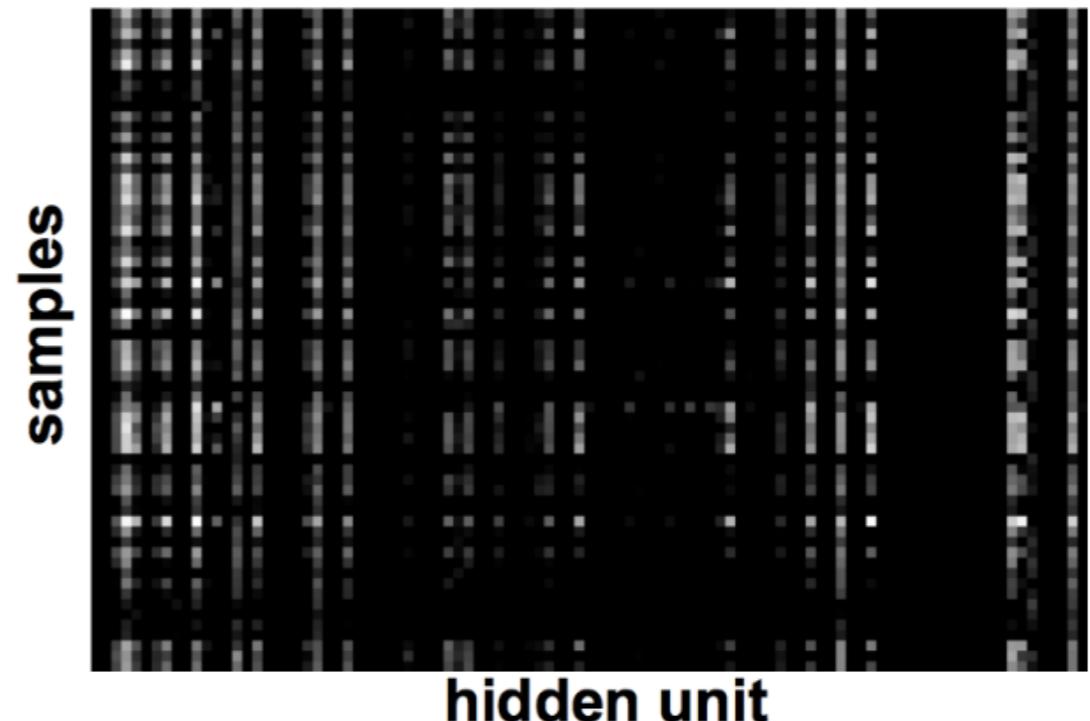
## Диагностика проблем с НС: Визуализация

Признаки (в общем смысле) должны быть некоррелированными и с большой дисперсией

Хорошо:



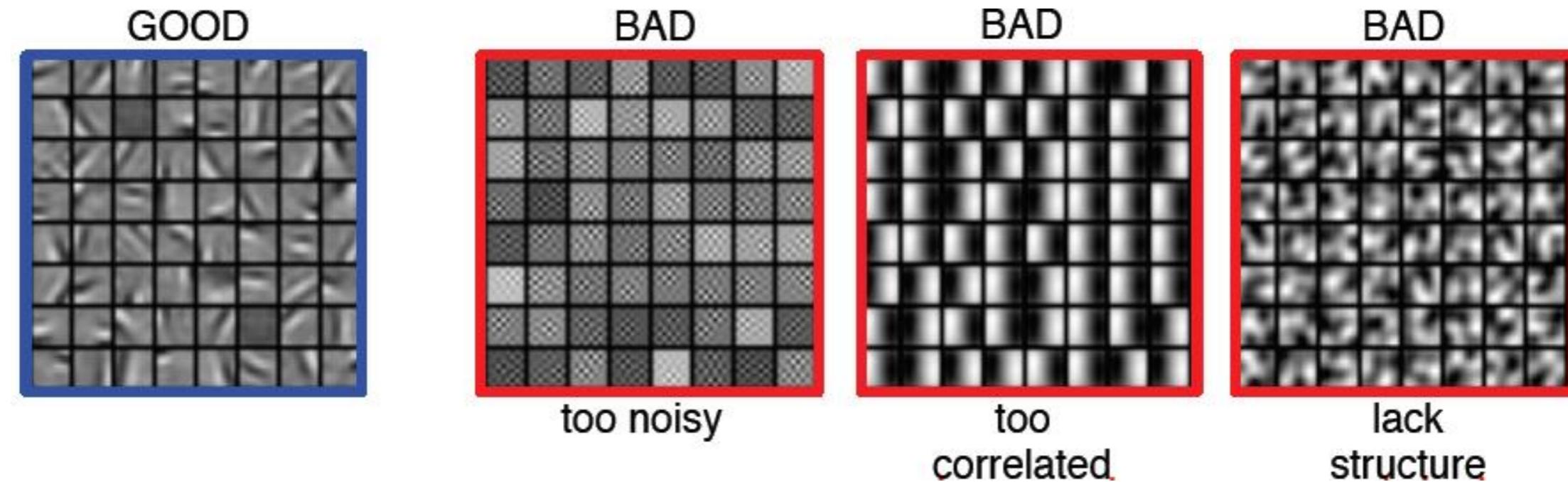
Плохо:



Marc'Aurelio Ranzato, CVPR 2014

## Диагностика проблем с НС: Визуализация

Хорошие фильтры имеют структуру и некоррелированные



## Диагностика проблем с НС: Маленькая выборка

**Убедиться, что сеть работает на небольшом куске данных  
(~ 100 – 500 объектов)**

---

## Диагностика проблем с НС: Насыщенность нейронов

**Насыщены ли нейроны ещё до обучения?  
Нормировка!**

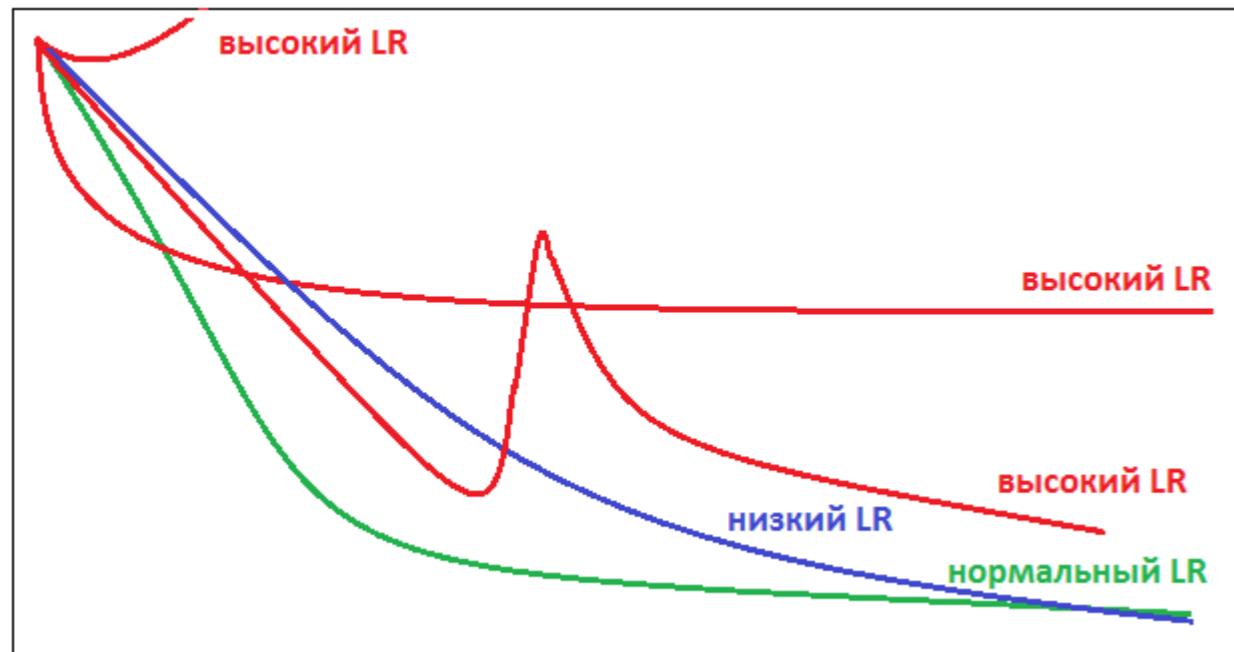
---

## Диагностика проблем с НС: Изменения весов

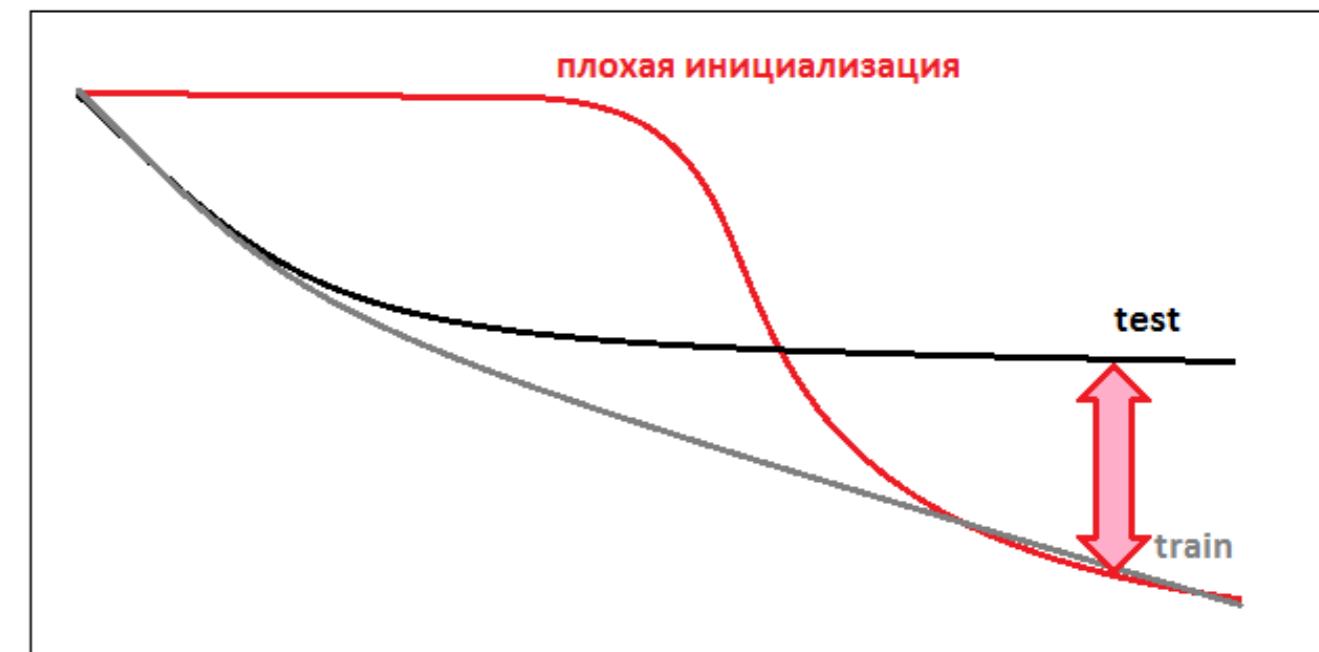
**Насколько меняются веса за итерацию (~ 0.1%)**

## Диагностика проблем с НС: Динамика ошибки

**Как ведёт себя ошибка обучения – настройте темп обучения!**



настройка темпа



плохая инициализация  
м.б. плохая архитектура,  
ex: нет прокидывания связей

**Большой зазор  $\Rightarrow$  переобучение  $\Rightarrow$  усилить регуляризацию**  
**Маленький  $\Rightarrow$  усложнить модель**

## Диагностика проблем с НС: Динамика ошибки

**На что ещё смотреть при обучении:**

- **ошибка / точность (метрика задачи) на обучении / валидации**
- **нормы градиентов по слоям (распределения)**
- **активации (гистограммы активаций)**
  - **время одной итерации**  
*(ускоряйте обучение: GPU, данные с SSD)*
- **прогресс обучения: когда заметны какие-то паттерны**

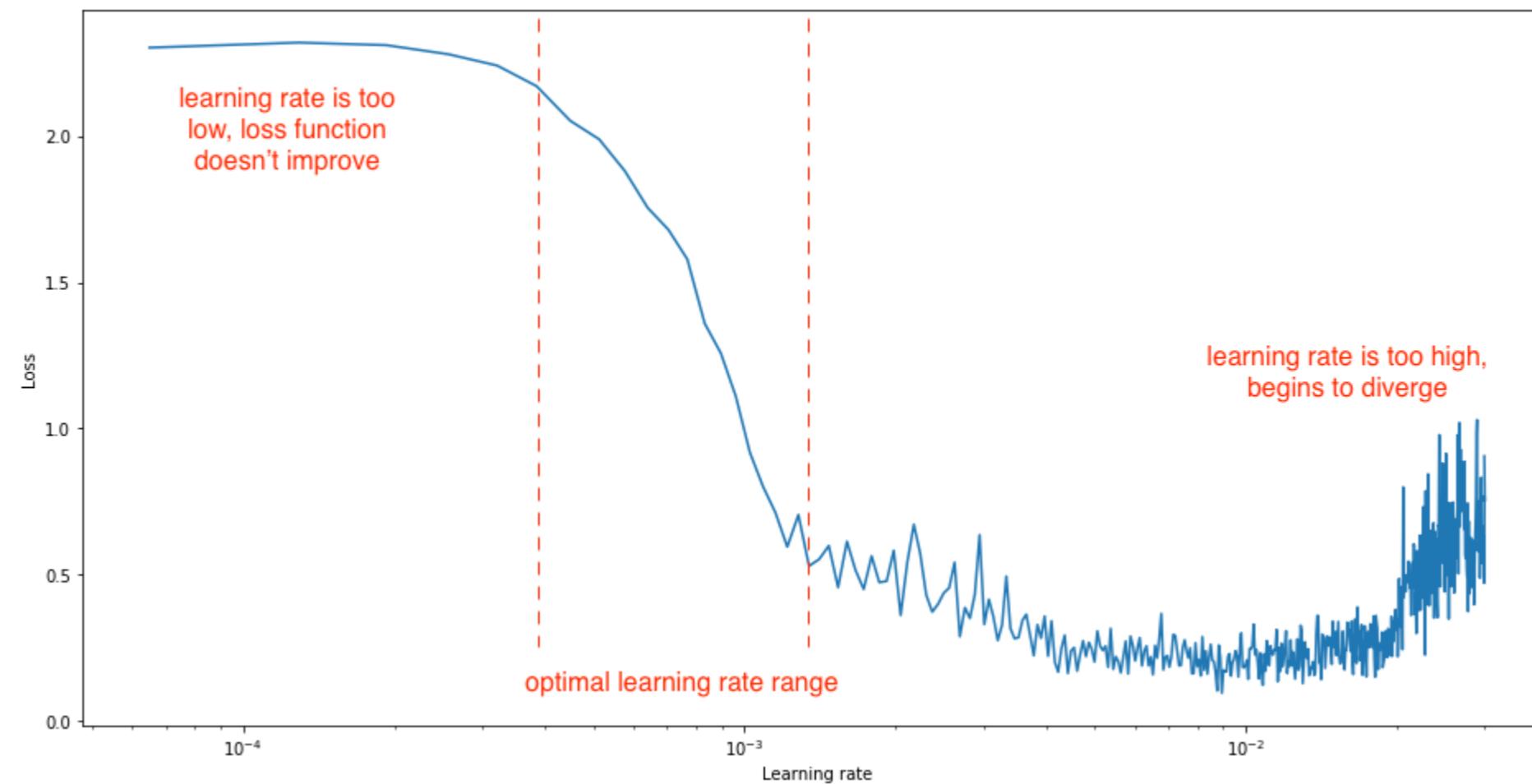
## Настройка темпа обучения

**При фиксированном темпе:  
чем больше – быстрее идём,  
но и сильнее перескакиваем  
(будем прыгать вокруг оптимума)**

**на практике снижают темп!  
начинают с такого, что метод не расходится**

**рекомендации:  
глубже – меньше  
меньше батч – меньше темп  
SGD: 0.1  
Adam: 1e-3  
Adam + transformer: 1e-5**

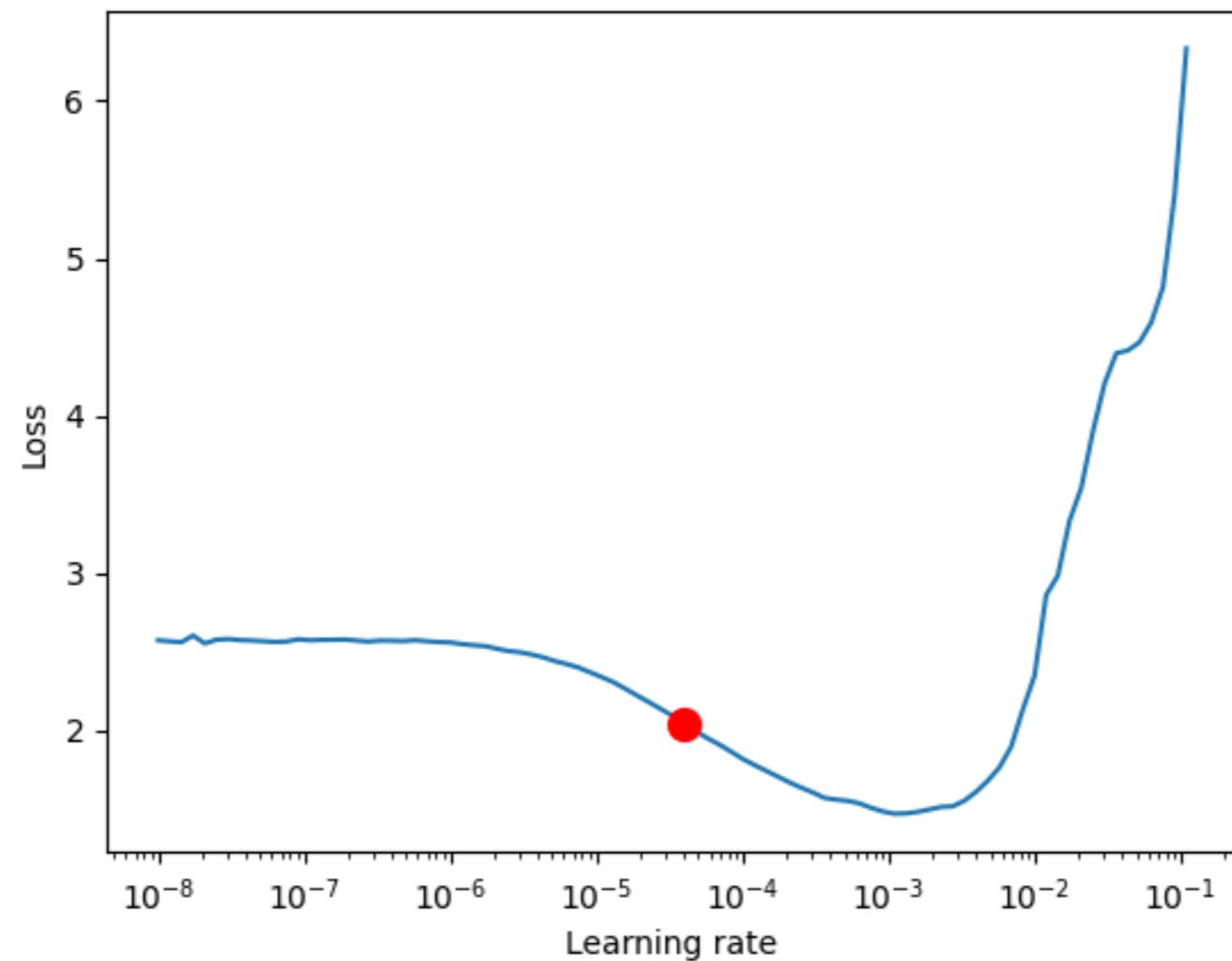
## Настройка темпа обучения



<https://www.jeremyjordan.me/nn-learning-rate/>

**Аналогичная идея – увеличивать темп после каждого батча, по графику ошибки найти оптимальный темп: «Cyclical Learning Rates for Training Neural Networks» <https://arxiv.org/abs/1506.01186>**

## Настройка темпа обучения

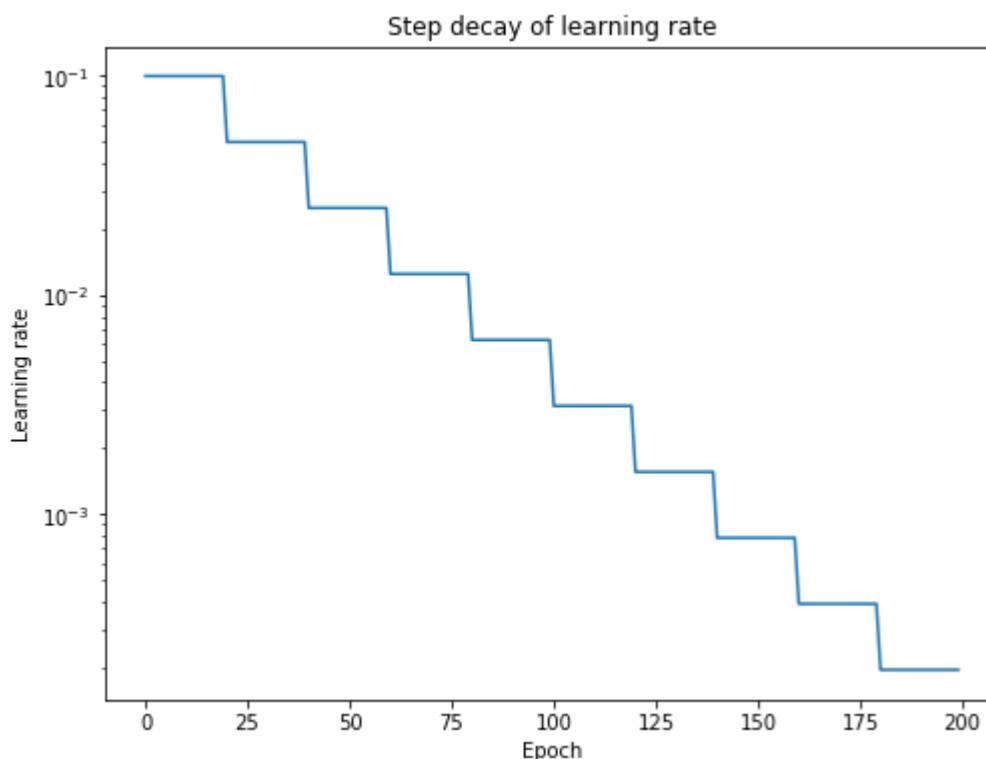


[https://pytorch-lightning.readthedocs.io/en/latest/advanced/lr\\_finder.html](https://pytorch-lightning.readthedocs.io/en/latest/advanced/lr_finder.html)

## Настройка темпа обучения: Learning Rate Annealing

**Всегда смотрите на кривые ошибок во время обучения!**

**уменьшать через каждые N эпох** `torch.optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1, last_epoch=-1, verbose=False)`



```
>>> # lr = 0.05      if epoch < 30  
>>> # lr = 0.005    if 30 <= epoch < 60  
>>> # lr = 0.0005   if 60 <= epoch < 90
```

<https://www.jeremyjordan.me/nn-learning-rate/>

## Настройка темпа обучения: Learning Rate Schedule

**уменьшать после выхода на плато**  
(качество не улучшается в течение N эпох)  
**или какого-то другого специфического условия**

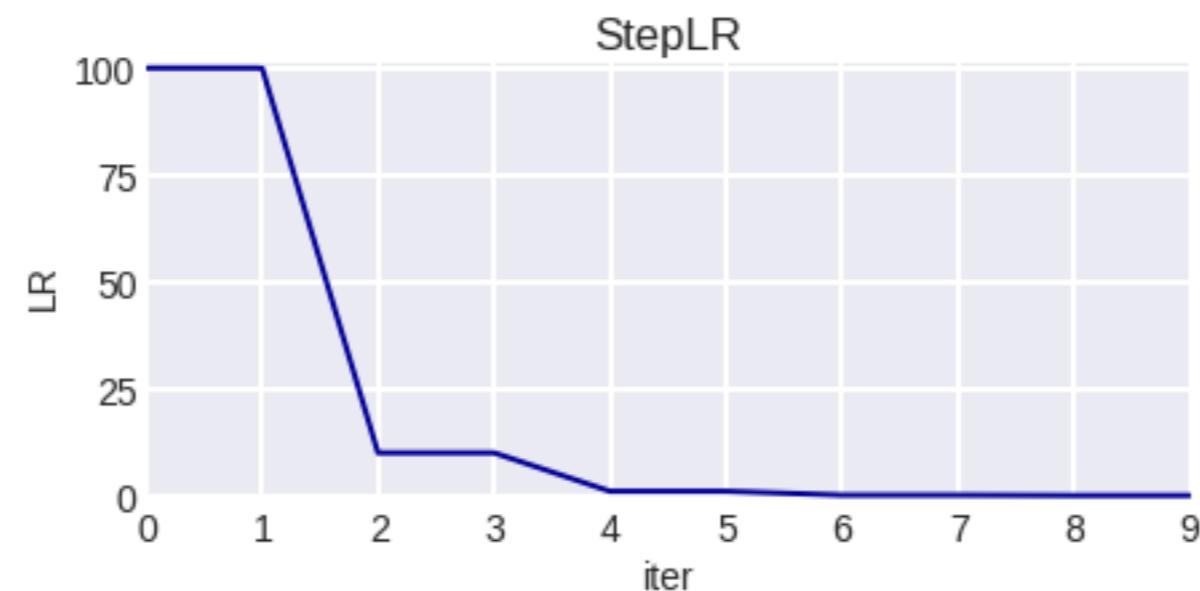
**увеличивать при попадании в седловую точку**  
(отсутствии прогресса)

`torch.optim.lr_scheduler.`

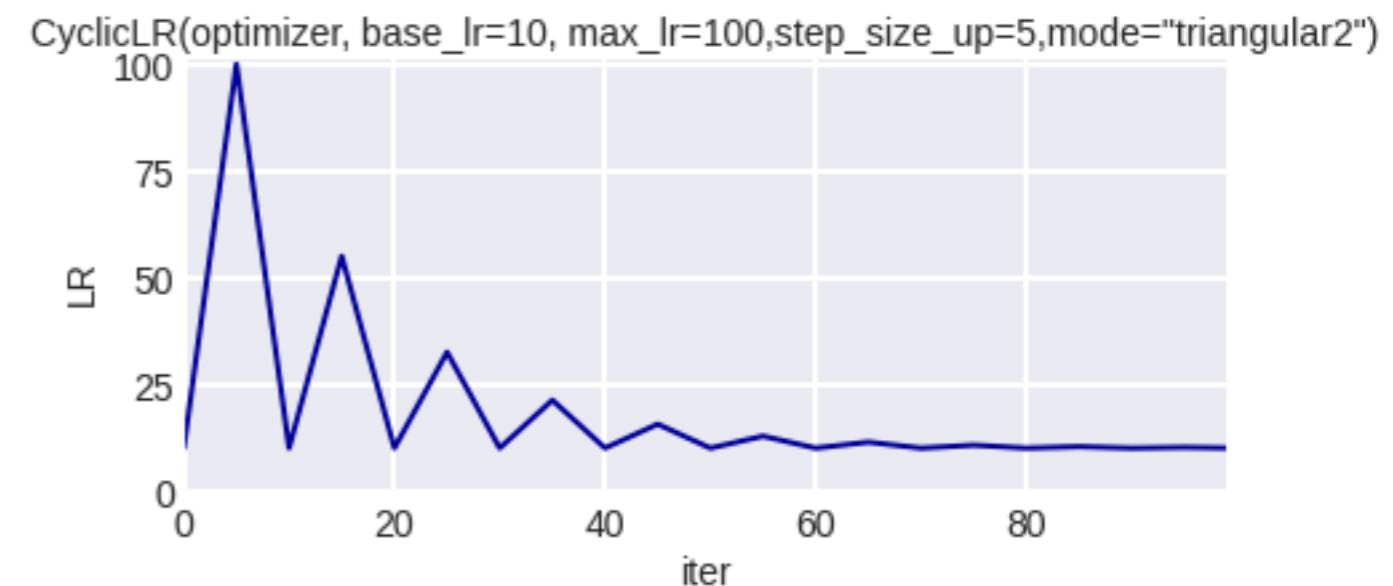
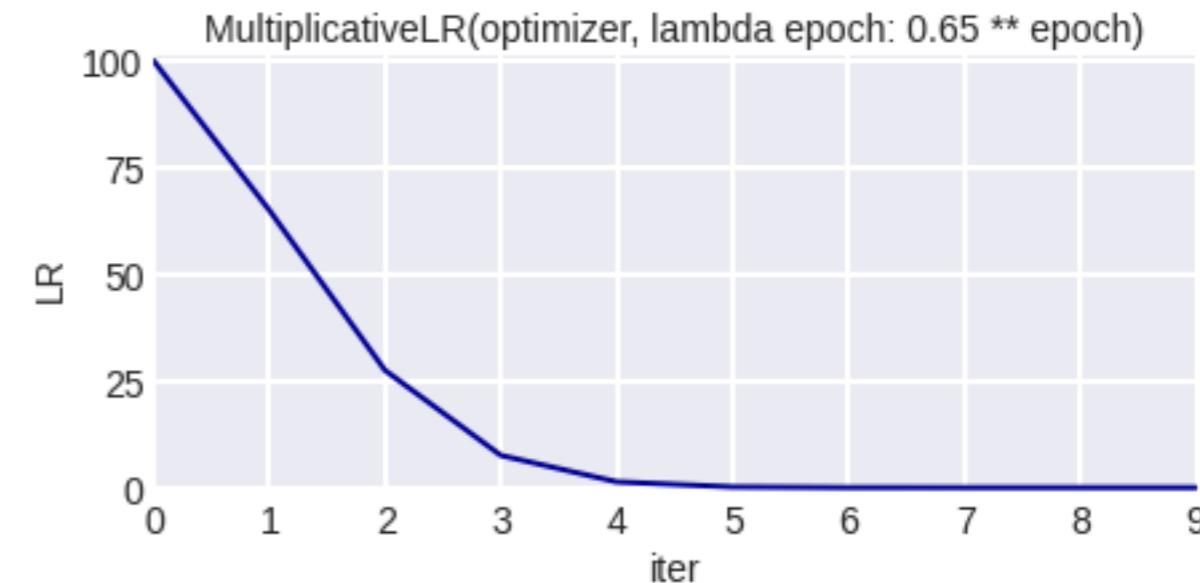
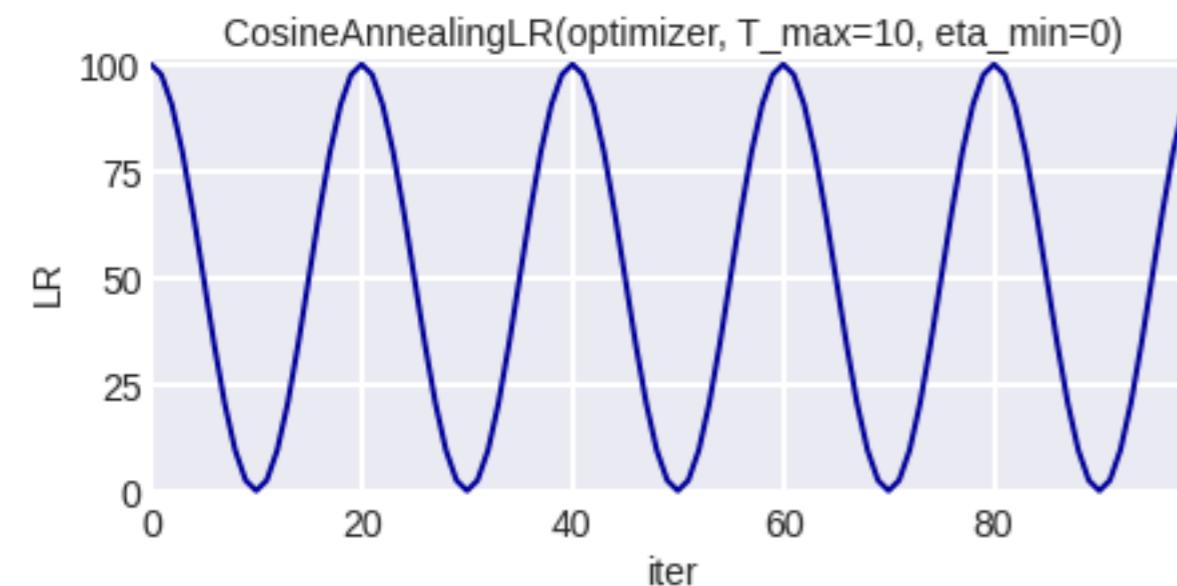
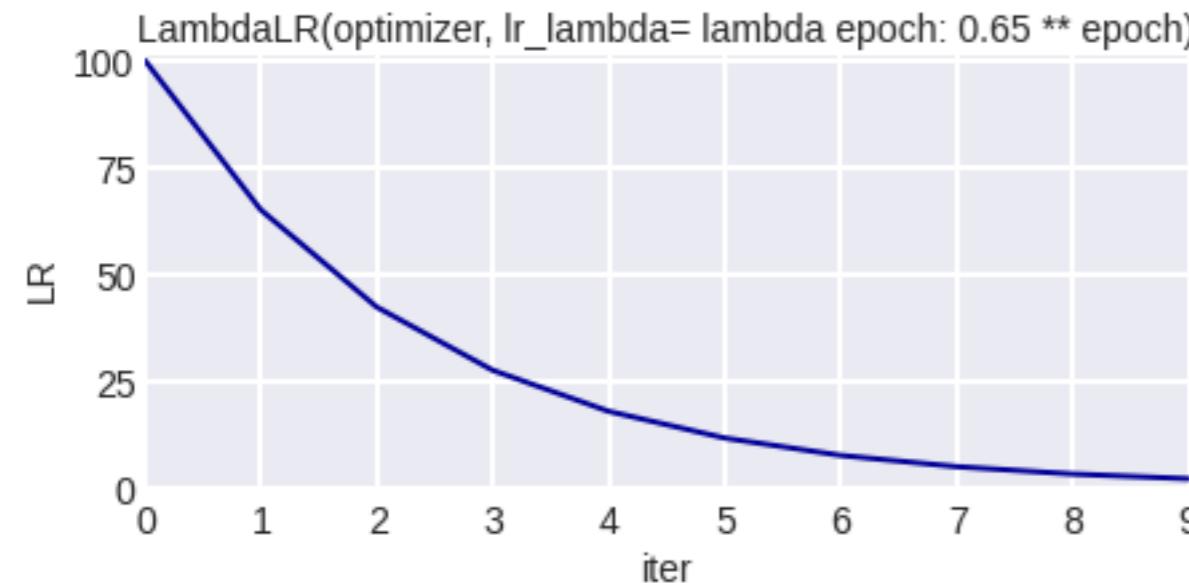
## Минутка кода: сценарий изменения темпа

```
model = torch.nn.Linear(2, 1)
optimizer = torch.optim.SGD(model.parameters(), lr=100)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=2, gamma=0.1)
lrs = []

for i in range(10):
    optimizer.step()
    lrs.append(optimizer.param_groups[0]["lr"])
    scheduler.step()
```



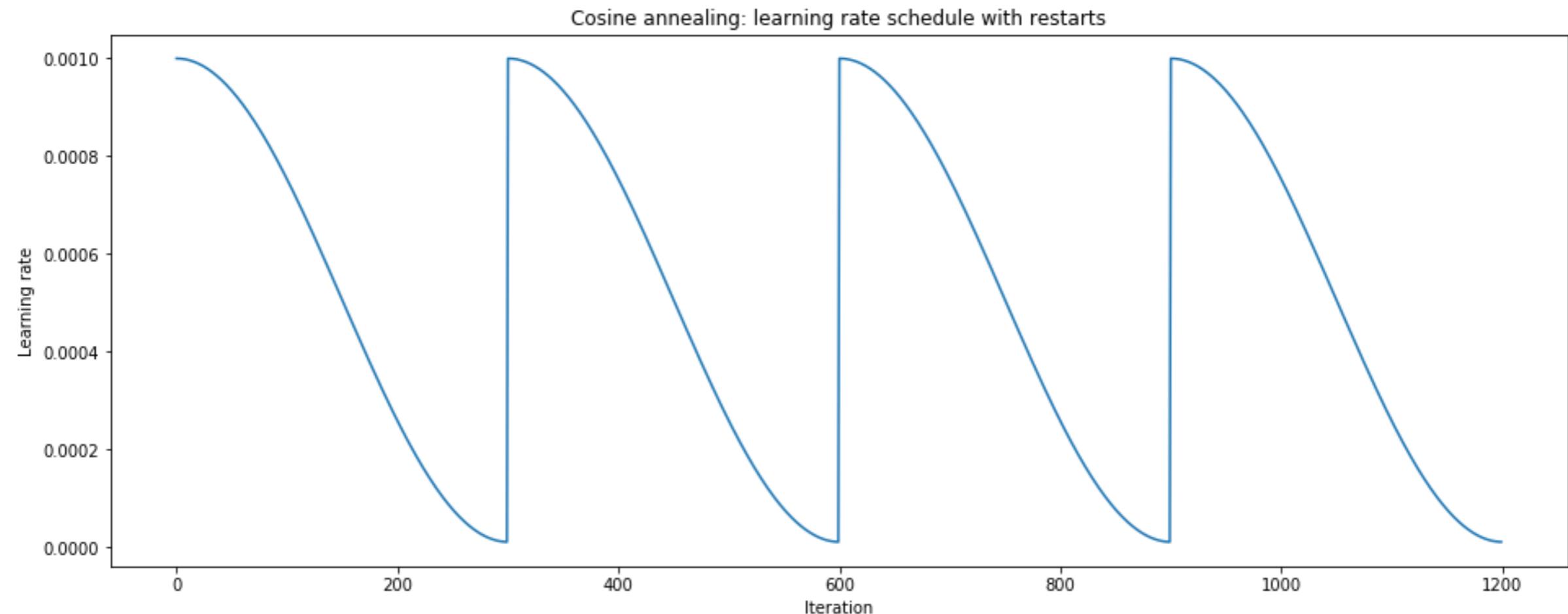
## Сценарий изменения темпа



<https://www.kaggle.com/isbhargav/guide-to-pytorch-learning-rate-scheduling>

## Настройка темпа обучения

### Stochastic Gradient Descent with Warm Restarts (SGDR)



и можно строить ансамбль моделей в точках затухания!

<https://www.jeremyjordan.me/nn-learning-rate/>

## Optimized learning rate schedule

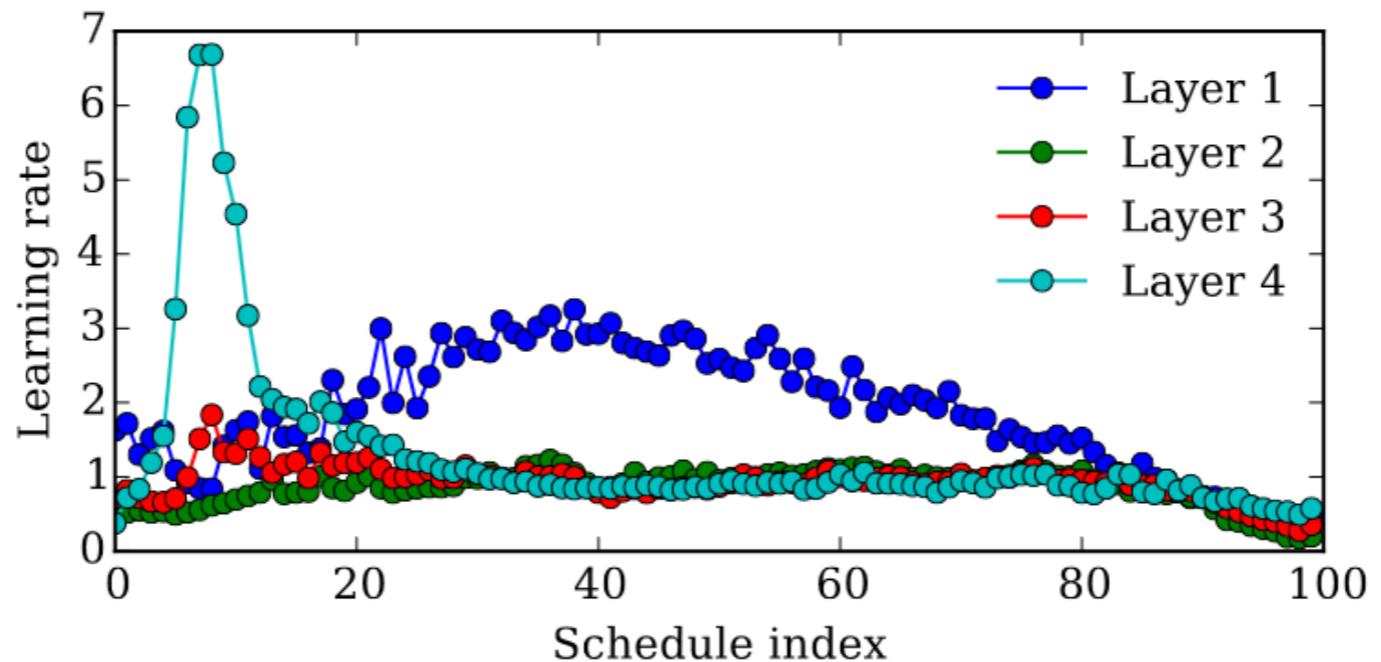


Figure 2. A learning-rate training schedule for the weights in each layer of a neural network, optimized by hypergradient descent. The optimized schedule starts by taking large steps only in the topmost layer, then takes larger steps in the first layer. All layers take smaller step sizes in the last 10 iterations. Not shown are the schedules for the biases or the momentum, which showed less structure.

**Dougal Maclaurin, David Duvenaud, Ryan P. Adams «Gradient-based Hyperparameter Optimization through Reversible Learning» // <https://arxiv.org/abs/1502.03492>**

## Диагностика проблем с НС

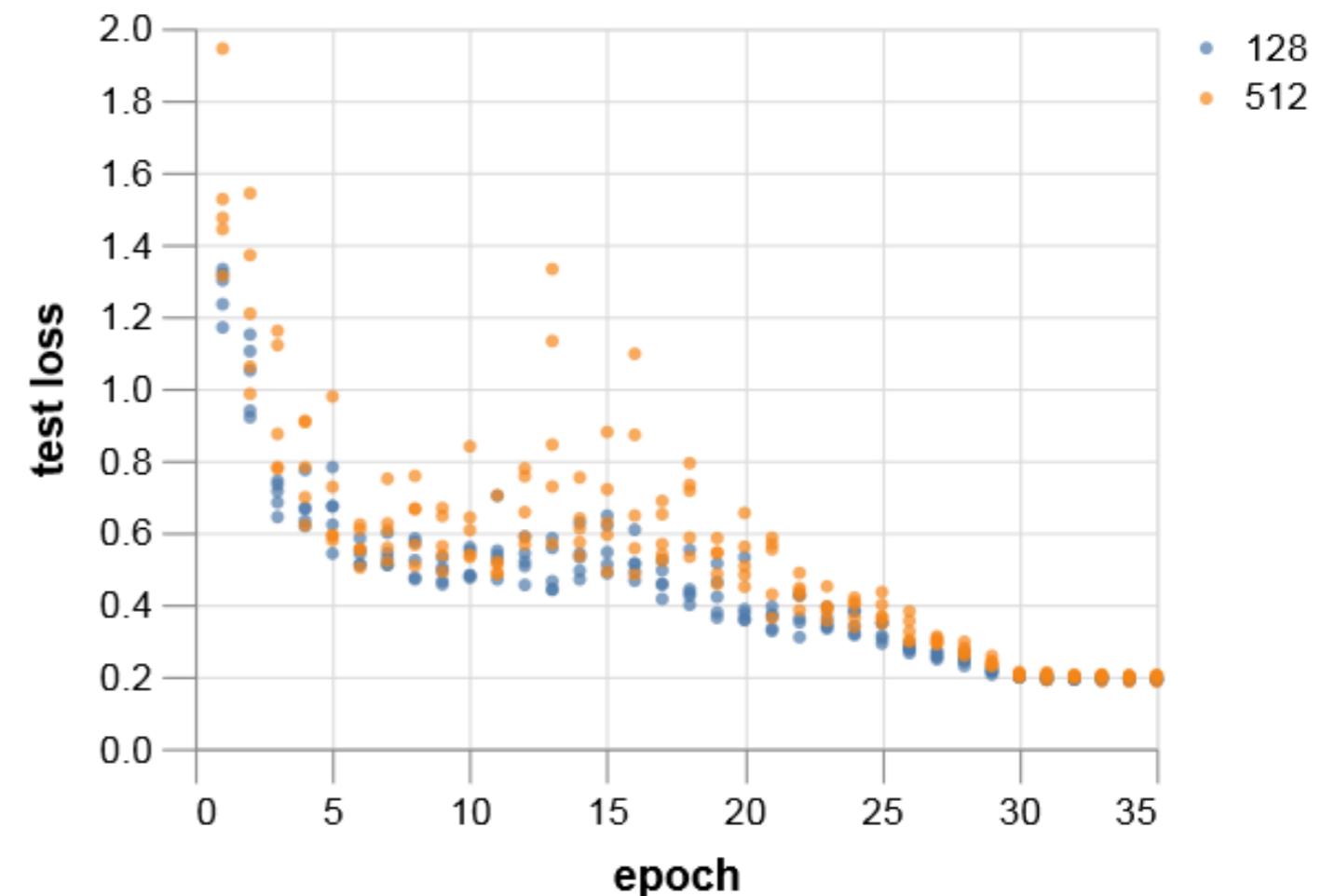
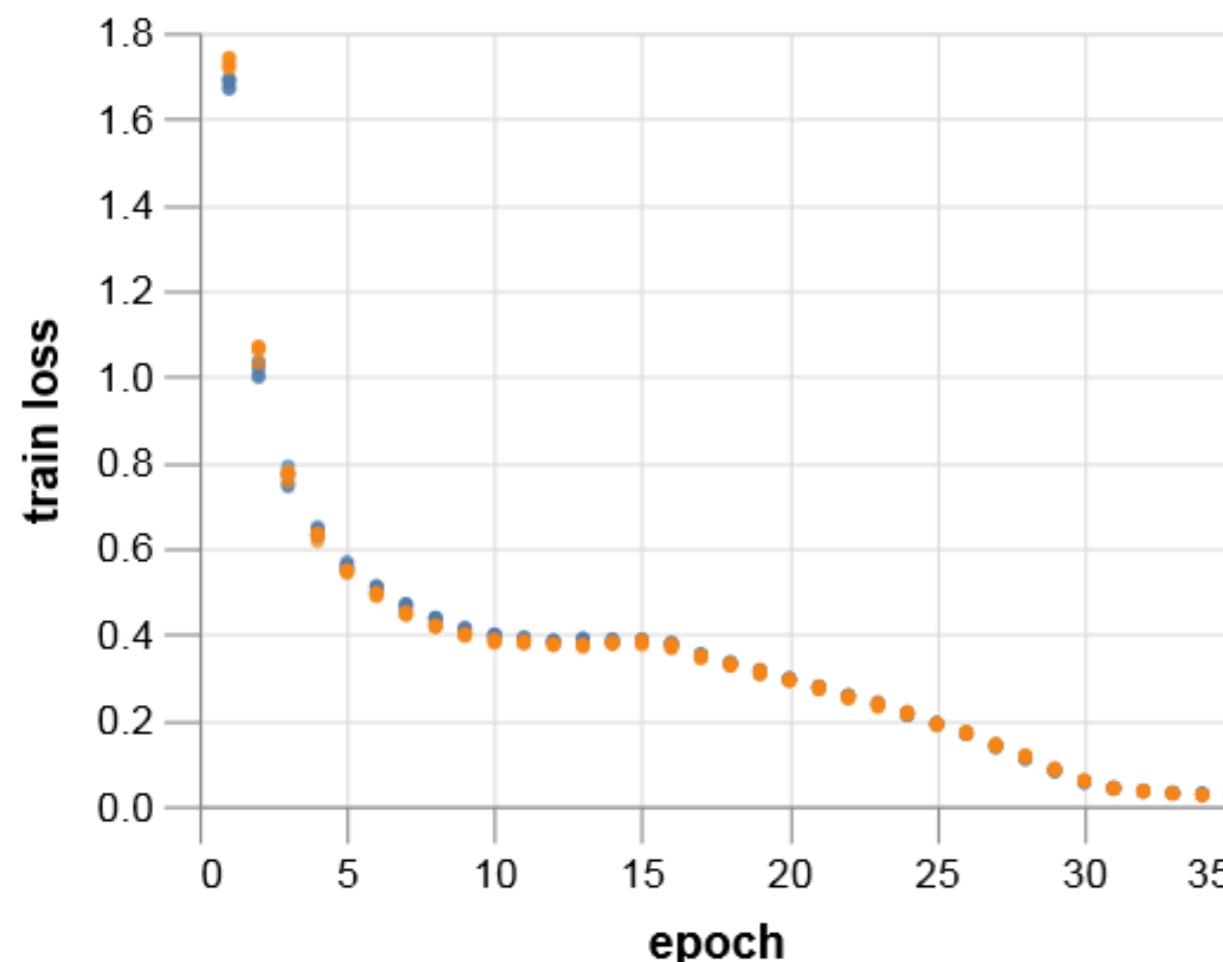
**Недообучение**

**Другие методы оптимизации**  
**GPU**  
**Сложнее сеть**

**Переобучение**

**Упрощение сети**  
**Регуляризация (+Dropout...)**  
приёмы, которые были  
**Обучение без учителя**  
**(более сложная задача ⇒ меньше переобучения)**

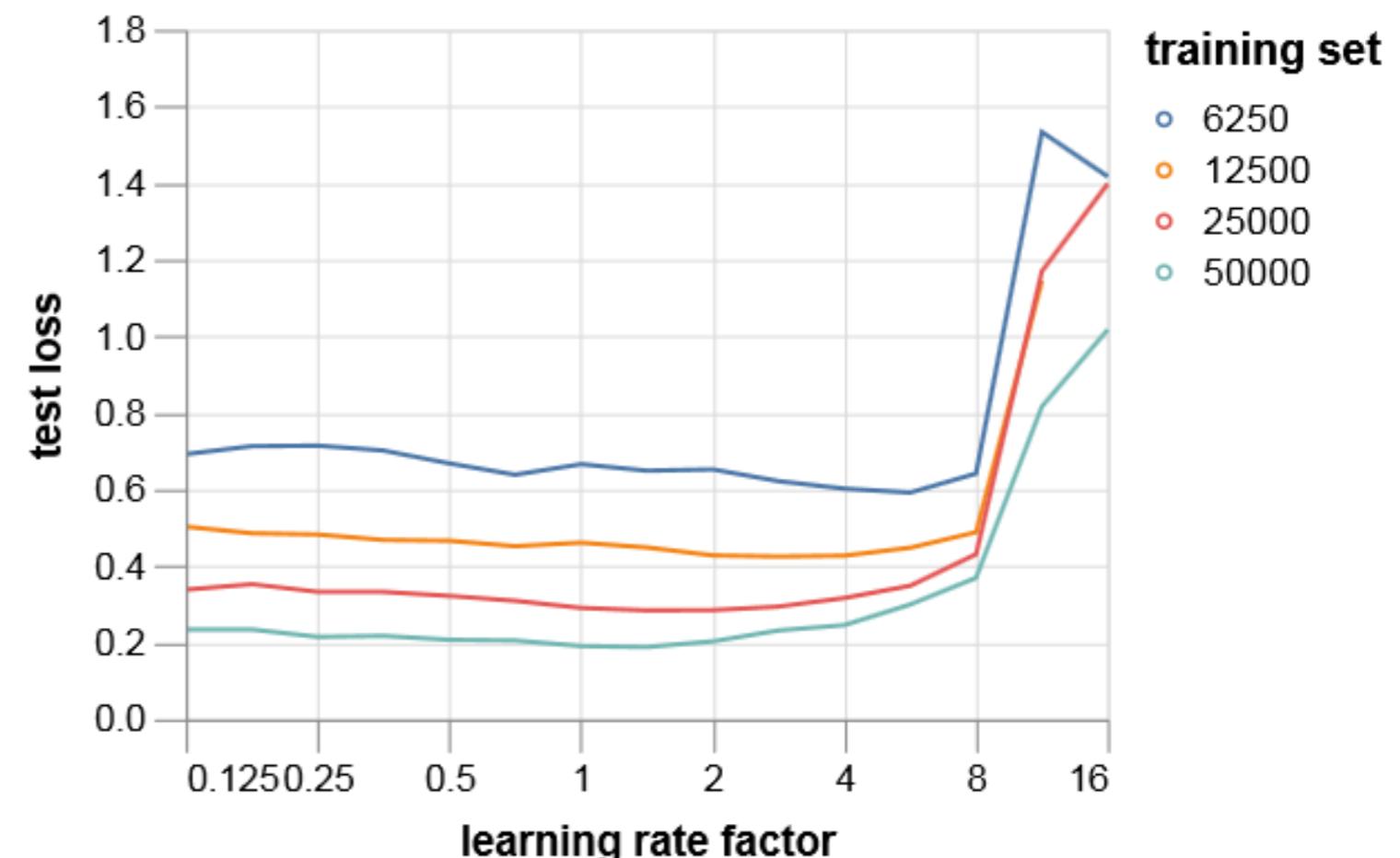
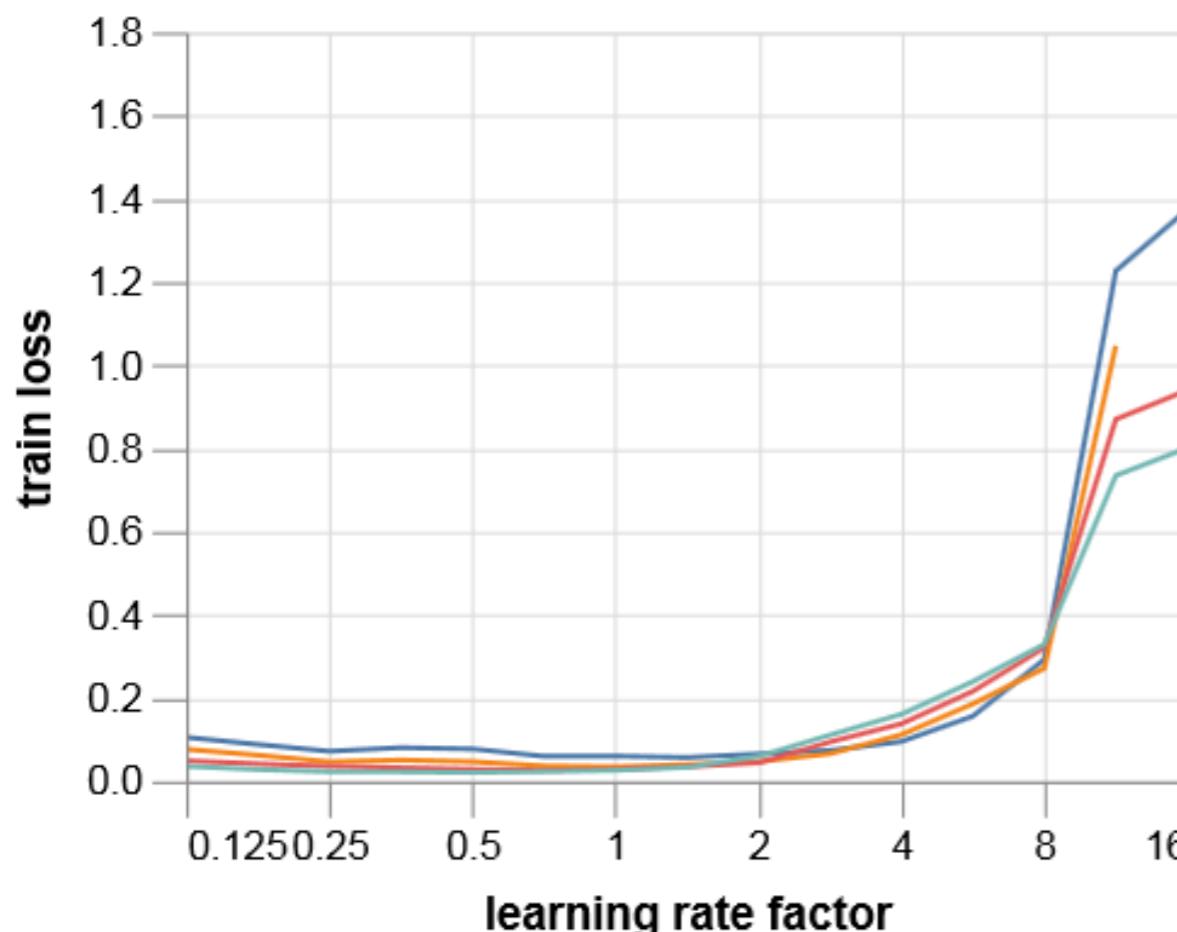
## Обучение нейронных сетей



**зависимость от размера батча**

<https://myrtle.ai/learn/how-to-train-your-resnet-2-mini-batches/>

## Обучение нейронных сетей

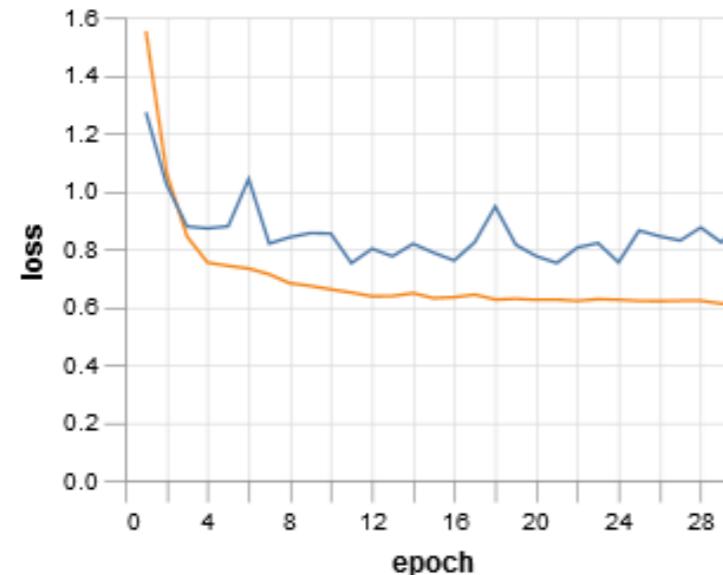


**зависимость от темпа обучения при разном размере обучения**

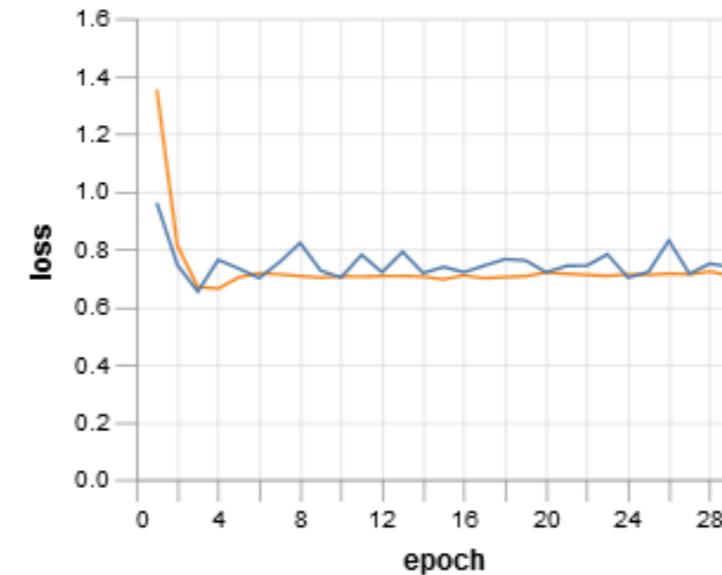
## Обучение нейронных сетей

maximum learning rate 4x higher than the original training setup:

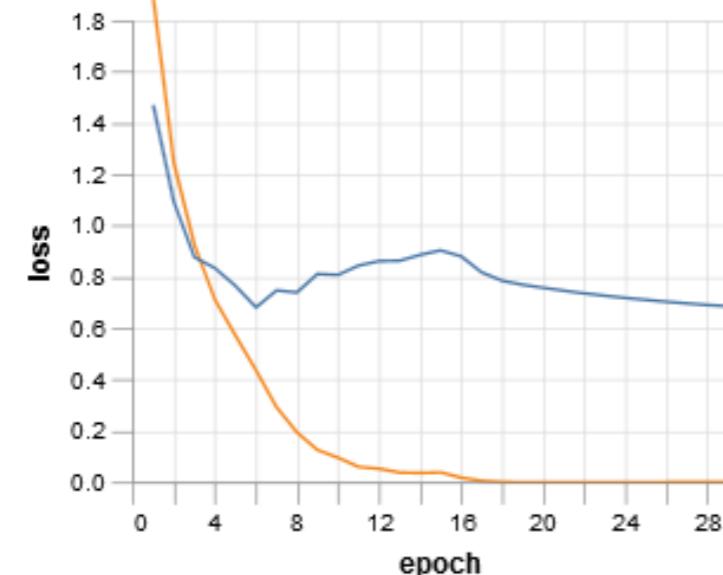
a) Half dataset no augmentation



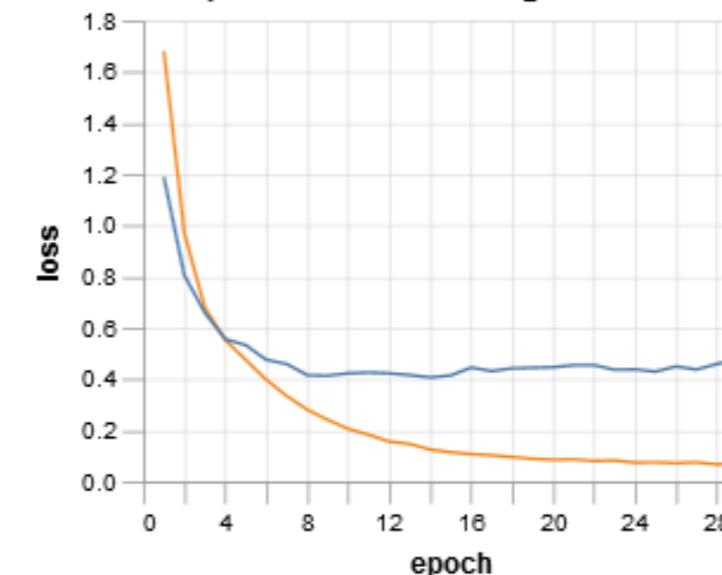
b) Full dataset + augmentation



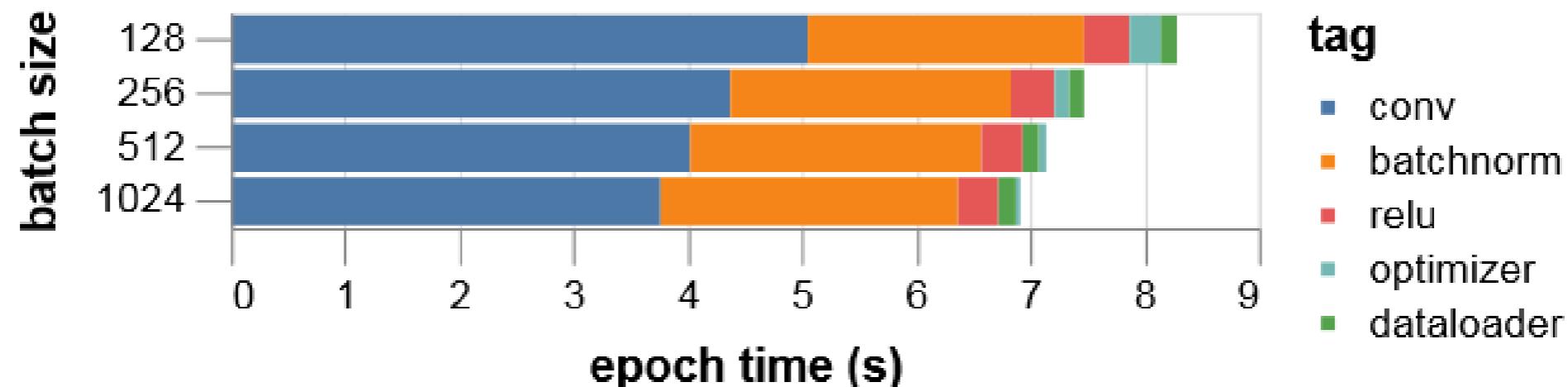
a) Half dataset no augmentation



b) Full dataset + augmentation

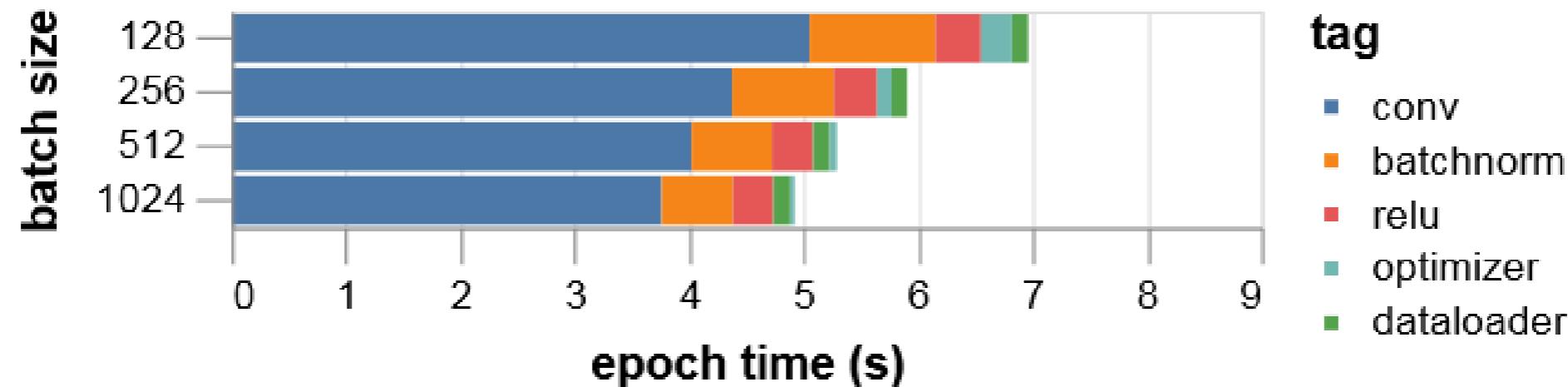


## Обучение нейронных сетей

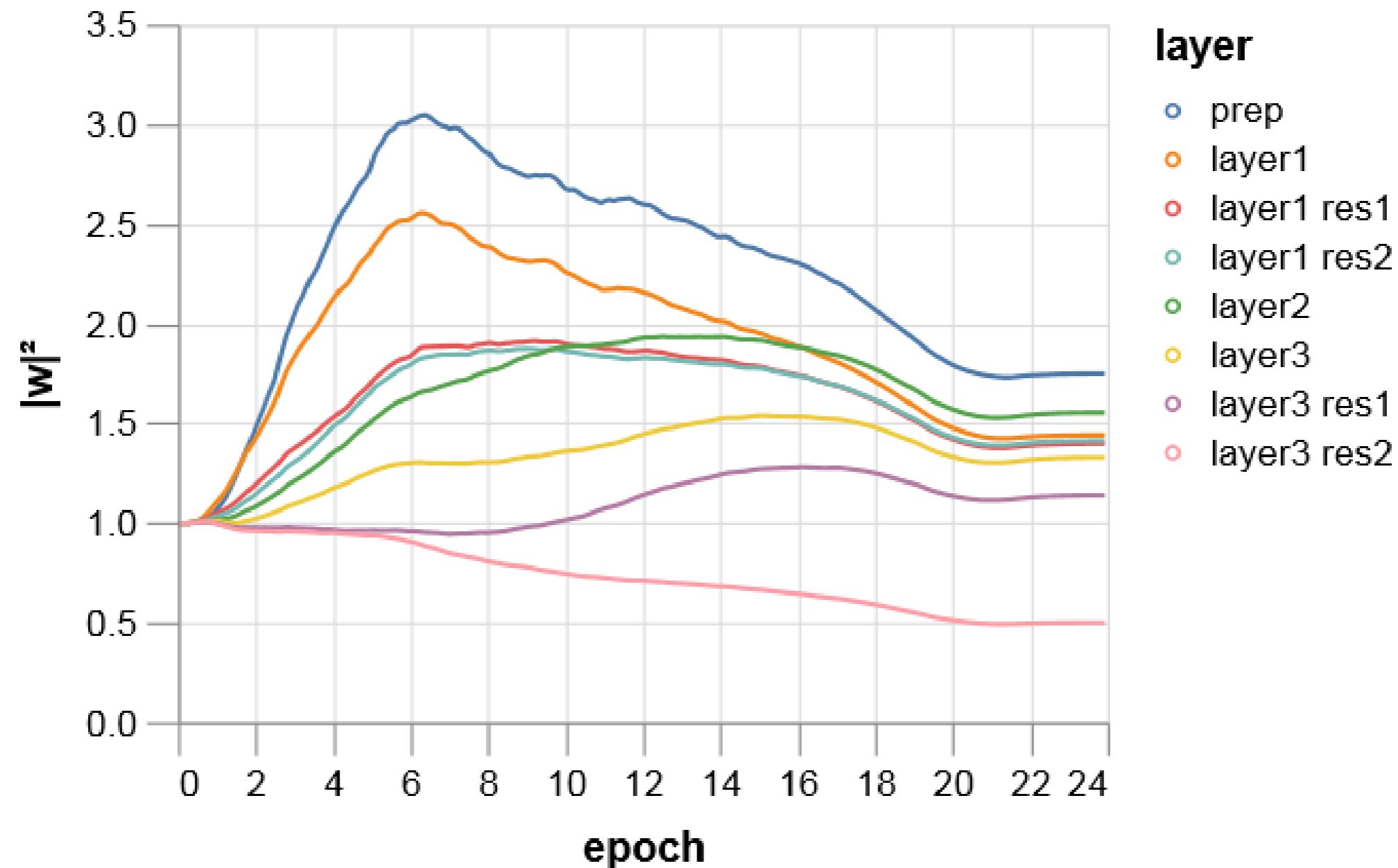


**convert batch norm weights back to single precision:**

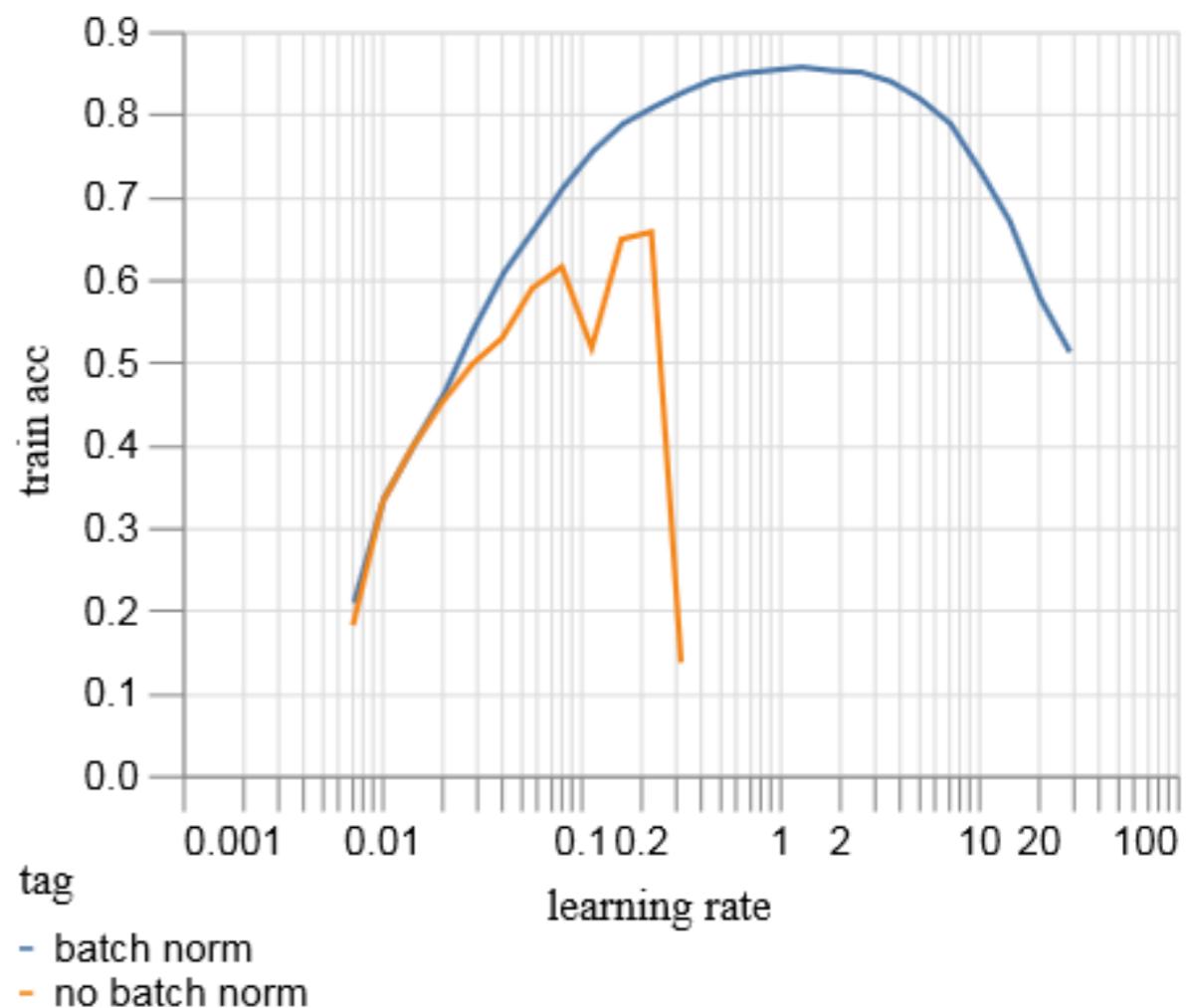
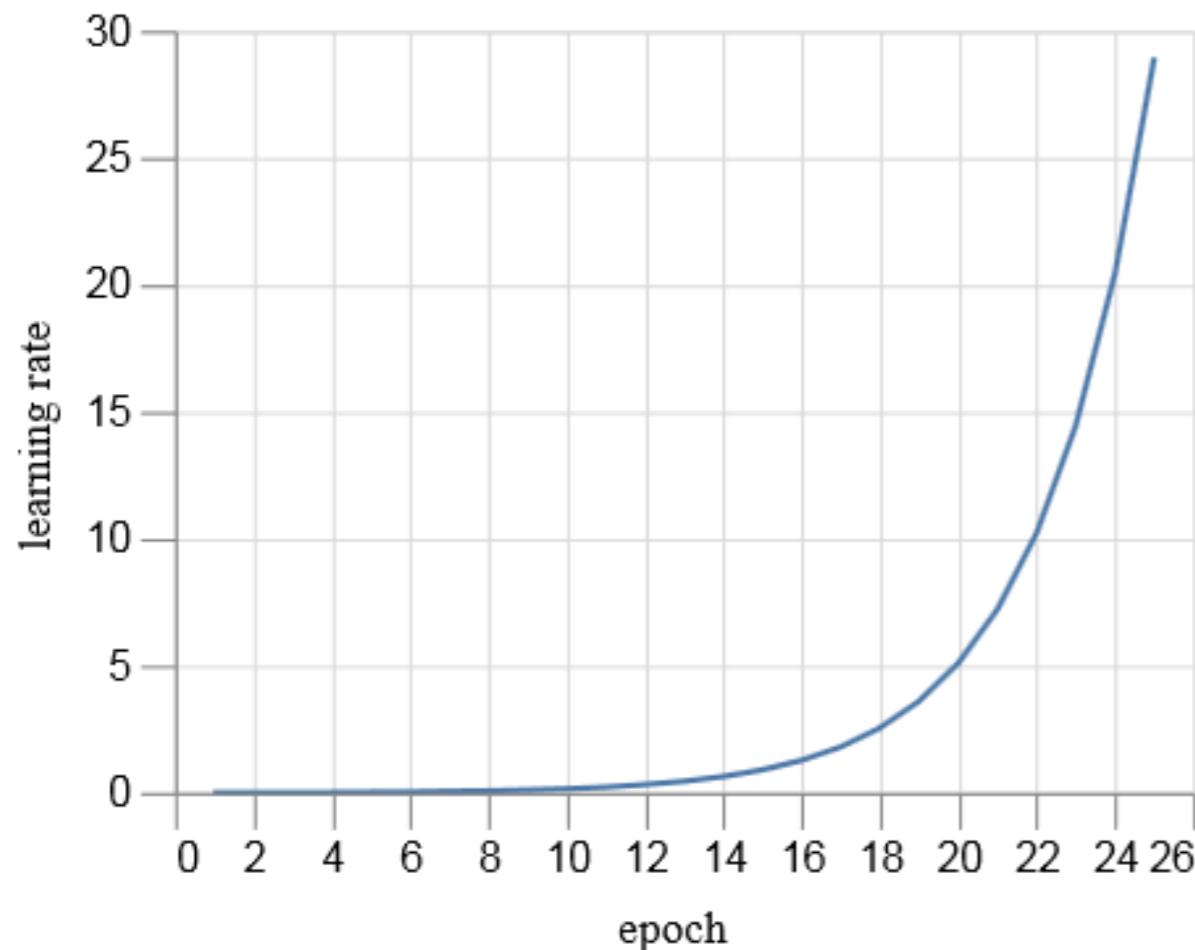
(небольшой хак с реализацией)



## Обучение нейронных сетей



## Обучение нейронных сетей



**Эксперимент, в котором увеличивали темп обучения (слева – как)**  
<https://myrtle.ai/learn/how-to-train-your-resnet-7-batch-norm/>

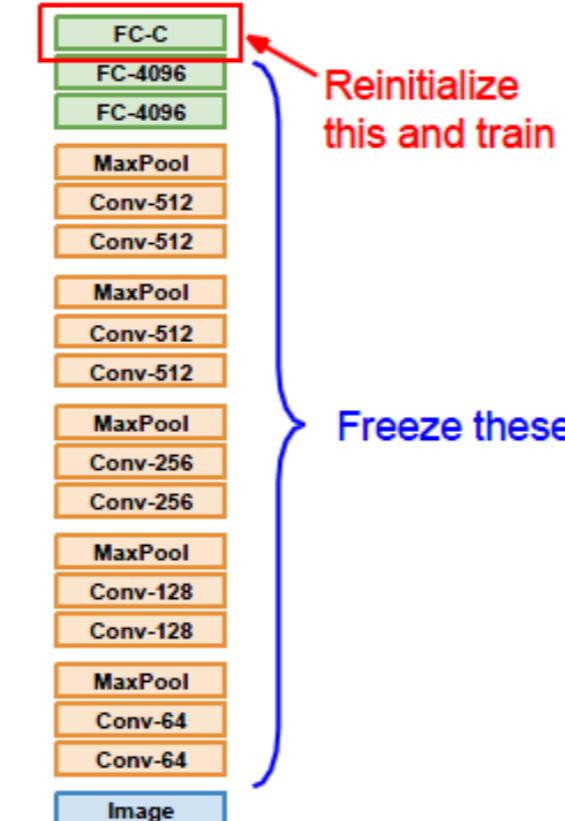
## Transfer Learning

**Чтобы решать задачи нужны данные...  
если данных мало, берём предобученную НС**

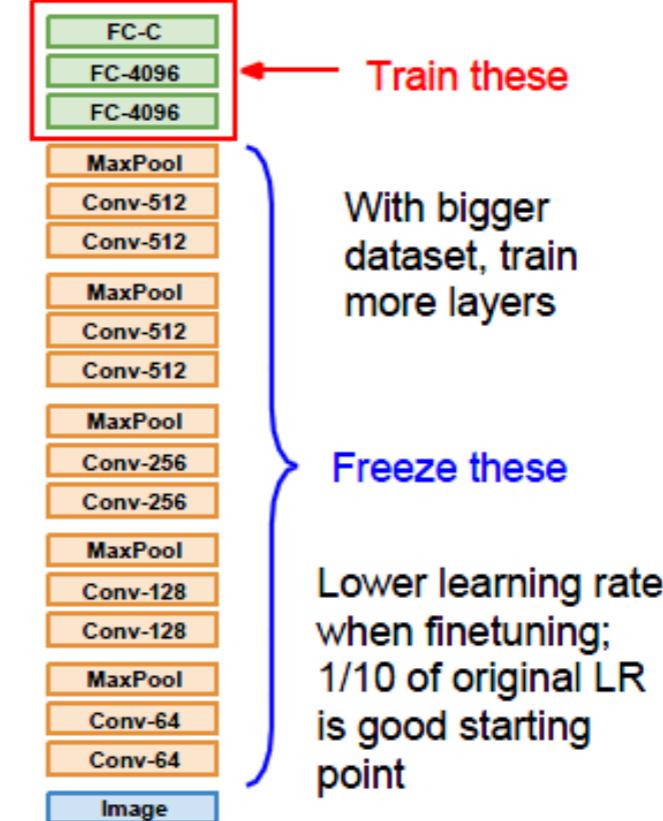
1. Train on Imagenet



2. Small Dataset (C classes)



3. Bigger dataset



**Можно не морозить слои, а обучать с меньшим темпом [cs231]**

## Transfer Learning

**Можно решать и другие задачи – для этого перестраивается голова сети  
(как бы получаем признаки обученной сетью)**

**Всю обученную сеть можно:**

- **полностью переучивать (можно с маленькими темпами на первых слоях)**
  - **переучивать с какого-то слоя**
    - **оставить как есть**

**Узкие глубокие сети учат с помощью уже обученных неглубоких широких**

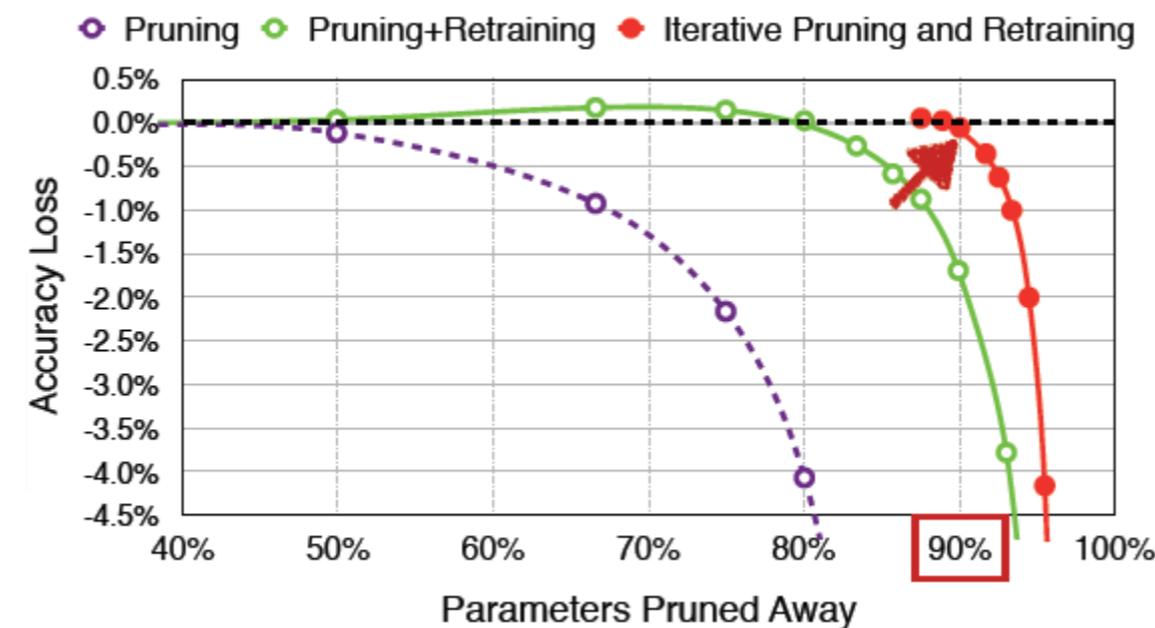
**Есть предтренированные НС:**

**Caffe:** <https://github.com/BVLC/caffe/wiki/Model-Zoo>

**TensorFlow:** <https://github.com/tensorflow/models>

**PyTorch:** <https://github.com/pytorch/vision>

## Упрощение НС (Pruning)



[Han et al. NIPS'15]



- **Original** : a man is riding a surfboard on a wave
- **Pruned 90%**: a man in a wetsuit is riding a wave **on a beach**



- **Original** : a soccer player in red is running in the field
- **Pruned 95%**: a man in **a red shirt and black and white black shirt** is running through a field

## Оптимизация гиперпараметров

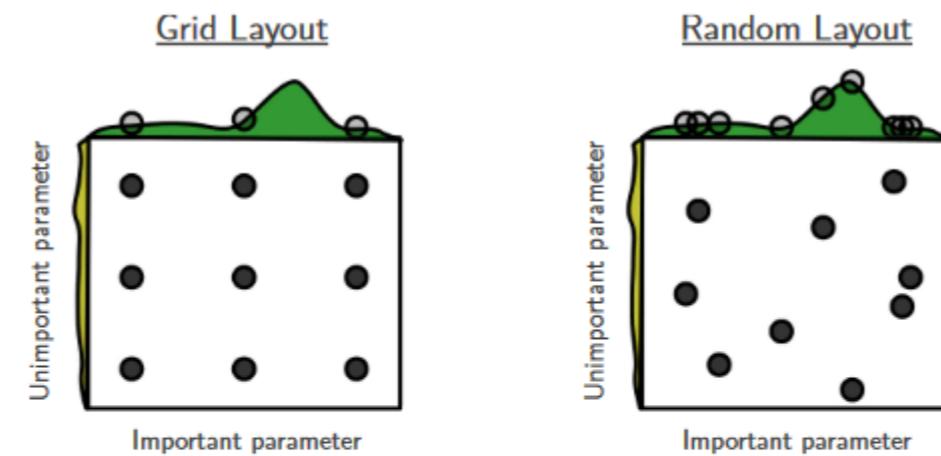
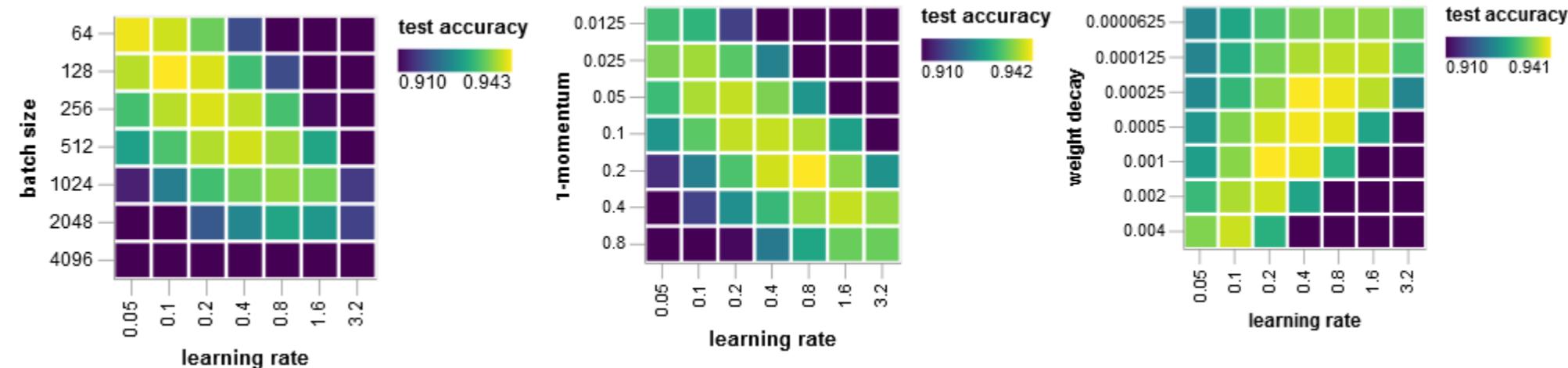


Figure 1: Grid and random search of nine trials for optimizing a function  $f(x,y) = g(x) + h(y) \approx g(x)$  with low effective dimensionality. Above each square  $g(x)$  is shown in green, and left of each square  $h(y)$  is shown in yellow. With grid search, nine trials only test  $g(x)$  in three distinct places. With random search, all nine trials explore distinct values of  $g$ . This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

**можно вести более интенсивный поиск в окрестности уже наёденного решения**

**есть другие методы оптимизации!**

## Оптимизация параметров



**можно подойти к поиску оптимальных гиперпараметров  
как к отдельной задаче оптимизации**

**покоординатный спуск в пространстве**  $\left( \frac{\lambda\alpha}{1-\rho}, \rho, \alpha \right)$

$\lambda$  - the maximal learning rate     $N$  - batch size     $\rho$  - momentum

<https://myrtle.ai/learn/how-to-train-your-resnet-5-hyperparameters/>

## Практические советы

- начинайте с простых архитектур и методов (оптимизации)
- начинайте с небольшого набора данных (для начальных экспериментов, попробуйте переобучиться на батче)
- выбирайте правильную архитектуру (классификация изображений – CNN, последовательности – LSTM/GRU и т.п.)
- Добавление параметров  $\Rightarrow$  усложнение сети (больше времени на обучение, риск переобучения)
- Используйте средства борьбы с переобучением (см. выше)
- Если данных слишком много средства могут не понадобиться. Если можно – собирайте данные!
- Используйте уже натренированные модели (не геройствуйте – берите проверенное)
- Learning rate часто самый важный параметр – лучше уменьшать (рекомендуют Adam)
- Есть методы настройки параметров лучше, чем structured (grid) search
- Визуализируйте!
- Смотрите на несколько метрик (обязательно на интерпретируемые)
- 0.1 / 1% Rule – веса должны меняться на 1% от своих значений

## Практические советы

- **Правильная инициализация**  
ex: при дисбалансе в последнем слое надо так, чтобы выдавалась малая верть
- **если что-то не получается можно понизить размерность**  
по-умному (PCA и т.п.) или просто уменьшить картинки
- **если что-то не получается можно упростить / усложнить сеть**
- **меняйте сеть поэтапно (не вносите более одного изменения)**
- **осторожней со смешиванием техник (например, dropout и BN)**  
Li et al. «Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift» // <https://arxiv.org/abs/1801.05134>
- **тренируйте дольше**  
«One time I accidentally left a model training during the winter break and when I got back in January it was SOTA»
- **не доверяйте встроенным программам понижения темпа**

## Глупые ошибки

**не перевести в режим train / eval**

**забыть zero\_grad() до backward()**

**сделать softmax для функции, ожидающей логиты (оценки)**

`nn.CrossEntropyLoss = nn.LogSoftmax + nn.NLLLoss`

**не нужен softmax**

**размерности, по которым делаются операции (nn.Softmax(dim=-1))**

**не создавайте лишнего в forward pass**

**(например, тогда nn.Embedding будет каждый раз новым)**

**не тестировать с разными параметрами**

**например, batch\_size, т.к. ориентацию матриц можно перепутать,  
если матрица квадратная, то ошибки не будет**

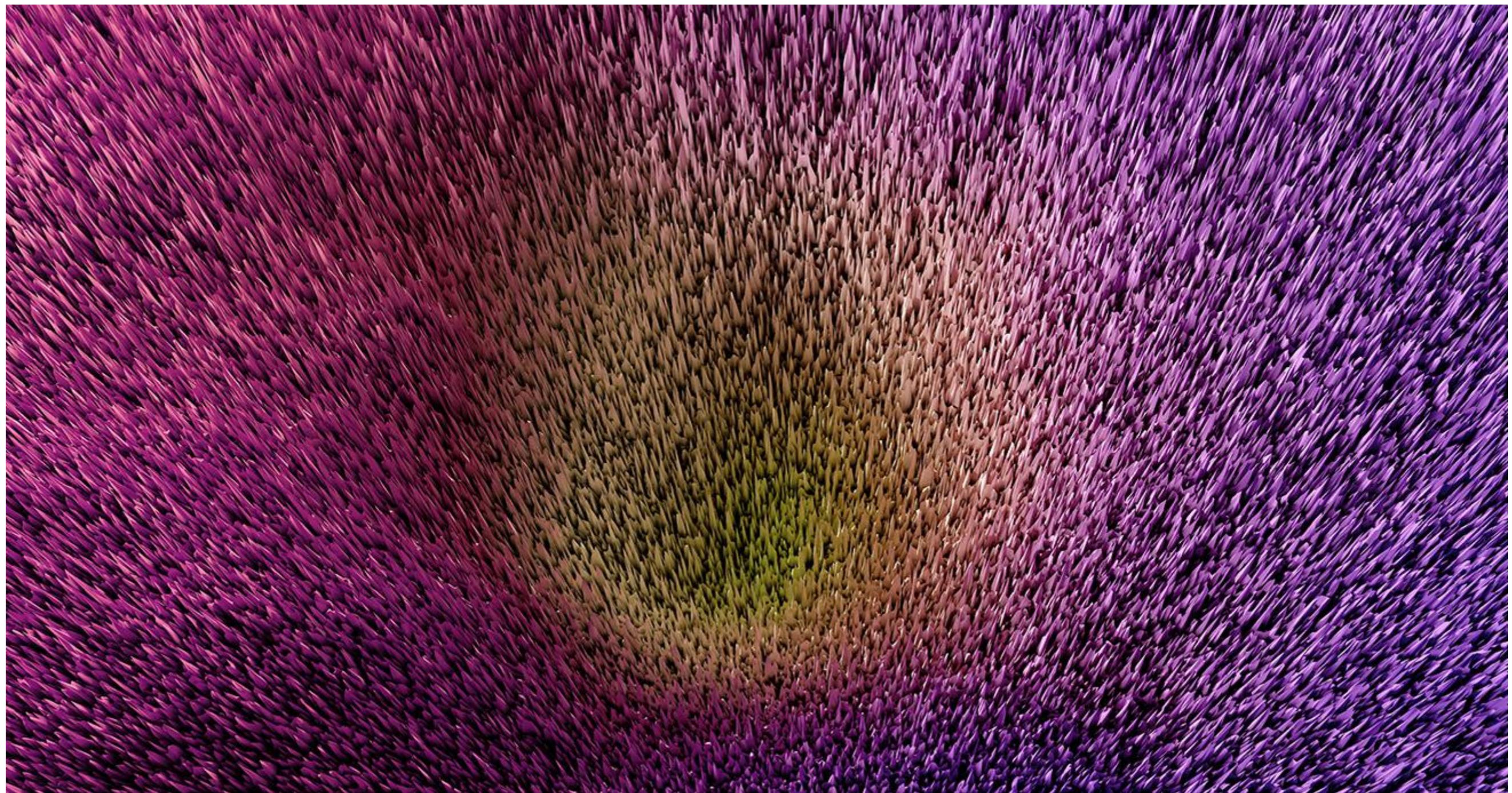
## Глупые ошибки

**не оформлять последовательность модулей в  
`nn.ModuleList / nn.Sequential`  
если это внутри другого модуля,  
то их параметры автоматически не определяются как параметры обёртки**

**вызывать `.to(device)` внутри инициализации НС**

**пренебрегать правильной инициализацией в глубоких сетях**





## Ссылки

### **Советы по настройке сетей**

<https://karpathy.github.io/2019/04/25/recipe/>

### **Debugging in PyTorch**

[https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial\\_notebooks/guide3/Debugging\\_PyTorch.html](https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/guide3/Debugging_PyTorch.html)

### **Поверхности функций ошибки**

<https://losslandscape.com/gallery/>