

курс «Глубокое обучение»

Борьба с переобучением в нейронных сетях

Александр Дьяконов

14 февраля 2022 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
- Dropout
- Увеличение выборки + Расширение выборки (Data Augmentation)
- Обрезка градиентов (Gradient clipping)
- Доучивание уже настроенных нейросетей (Pre-training)
- Техника зануления весов (разреживания НС)

Нормировки (Normalization of Data)

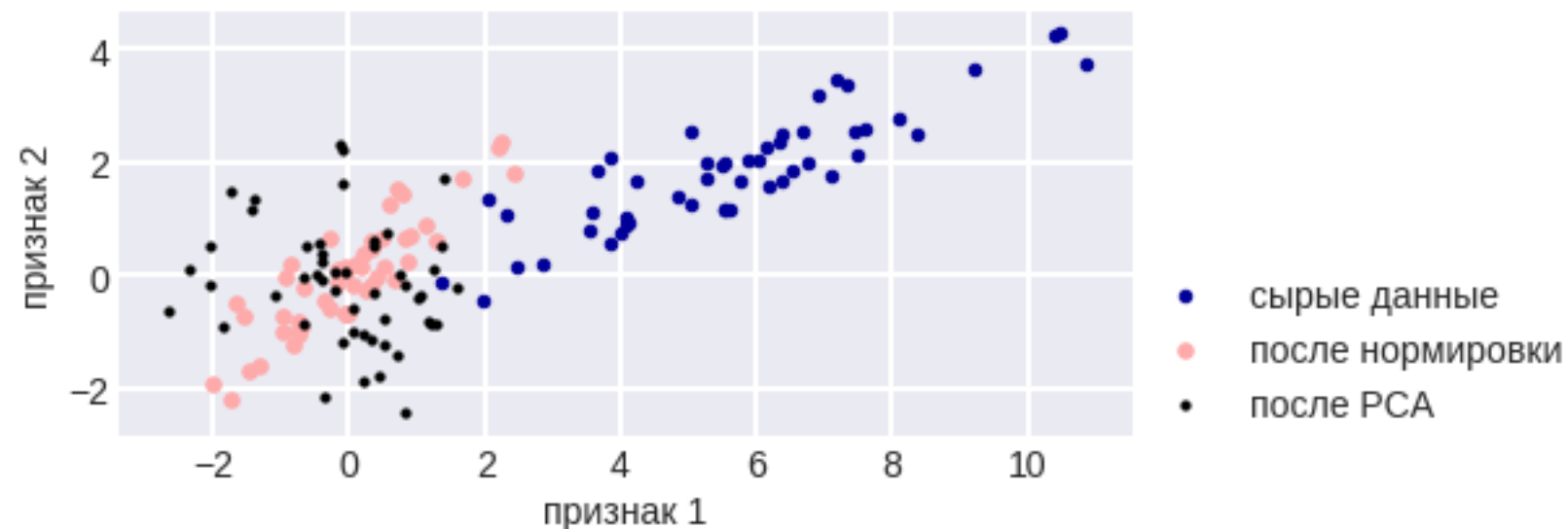
Признак $X = (X_1, \dots, X_m)$

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

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

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

иногда м.б. PCA



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/lu7TCu7HeYc>

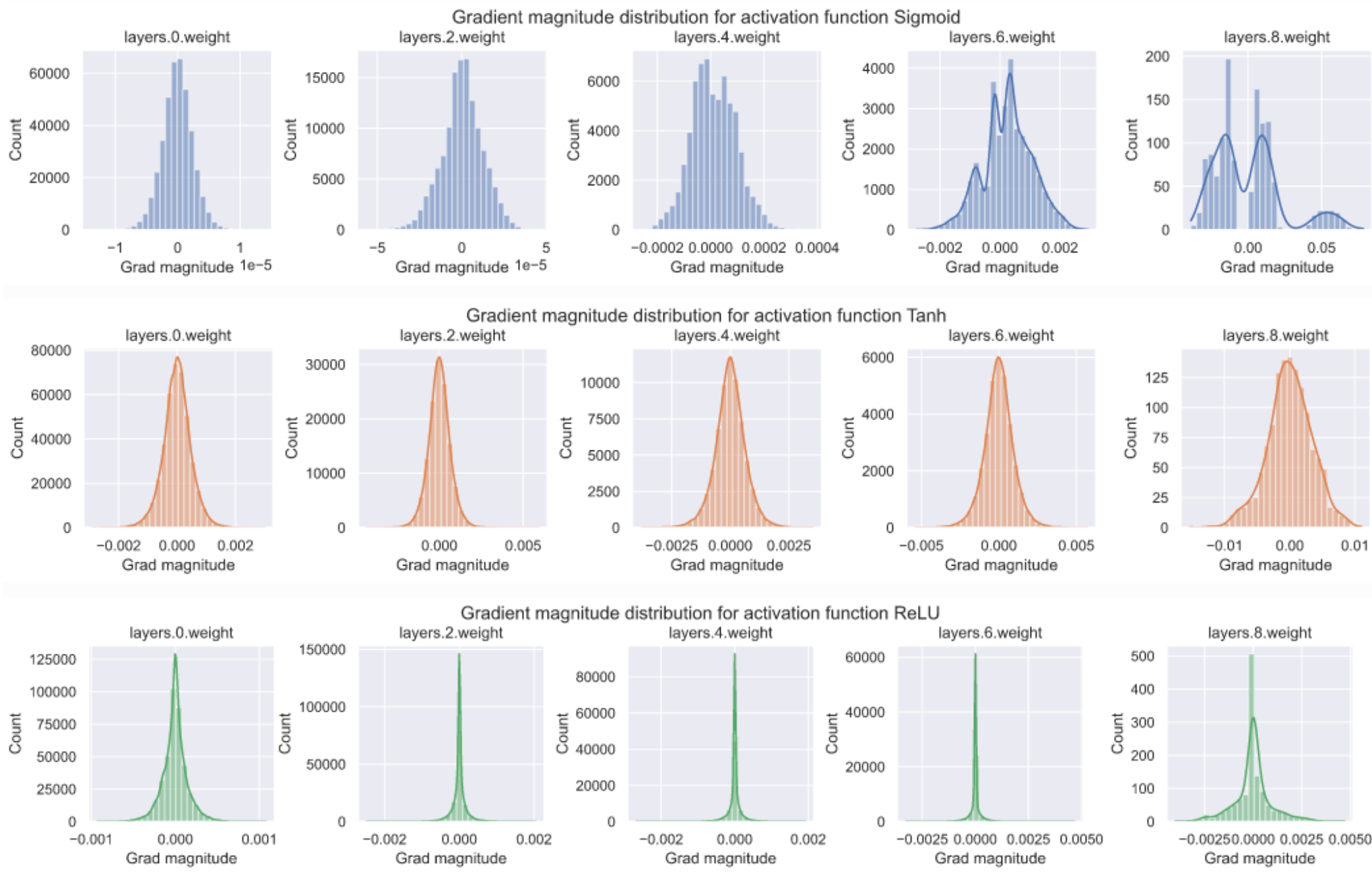
Инициализация весов

- **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)
это показатель, что сигнал проходит и не затухает / увеличивается
- **другая идея – градиенты на всех слоях должны иметь одинаковую дисперсию**
тогда подберём темп обучения сразу для всех слоёв

смещения := 0 (зависит от того, где смещение; если на выходе...)

Поиграйтесь! – <https://deeplearning.ai/ai-notes/initialization/>

Распределения и дисперсии в нейронных сетях

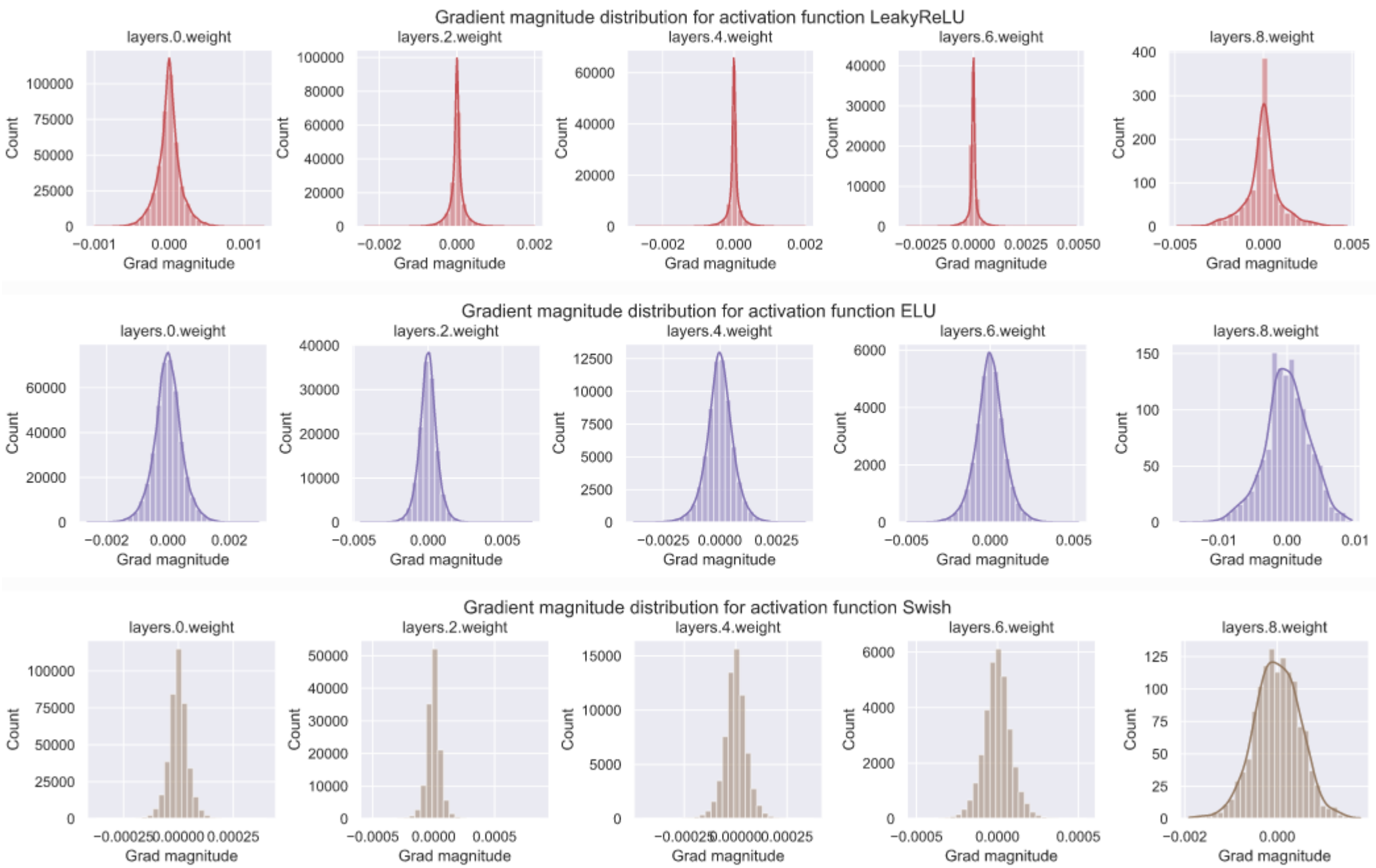


У сигмоиды очень большие градиенты на выходе – плохо!

И маленькие на входе.

У RELU пик в нуле – понятно

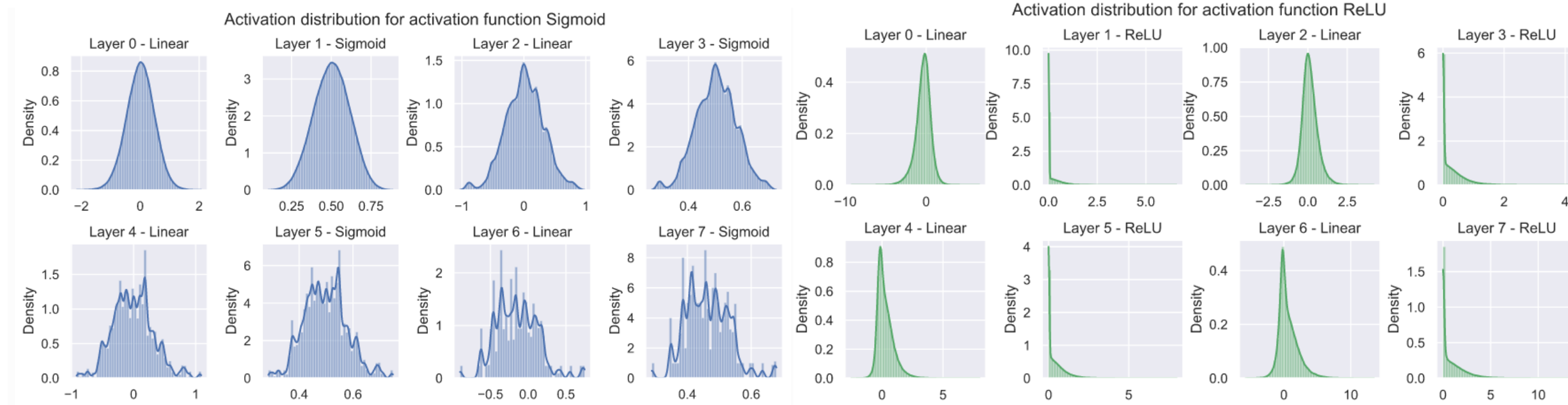
Распределения и дисперсии в нейронных сетях



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

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=PLrCZzMib1e9oOGLh6_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)

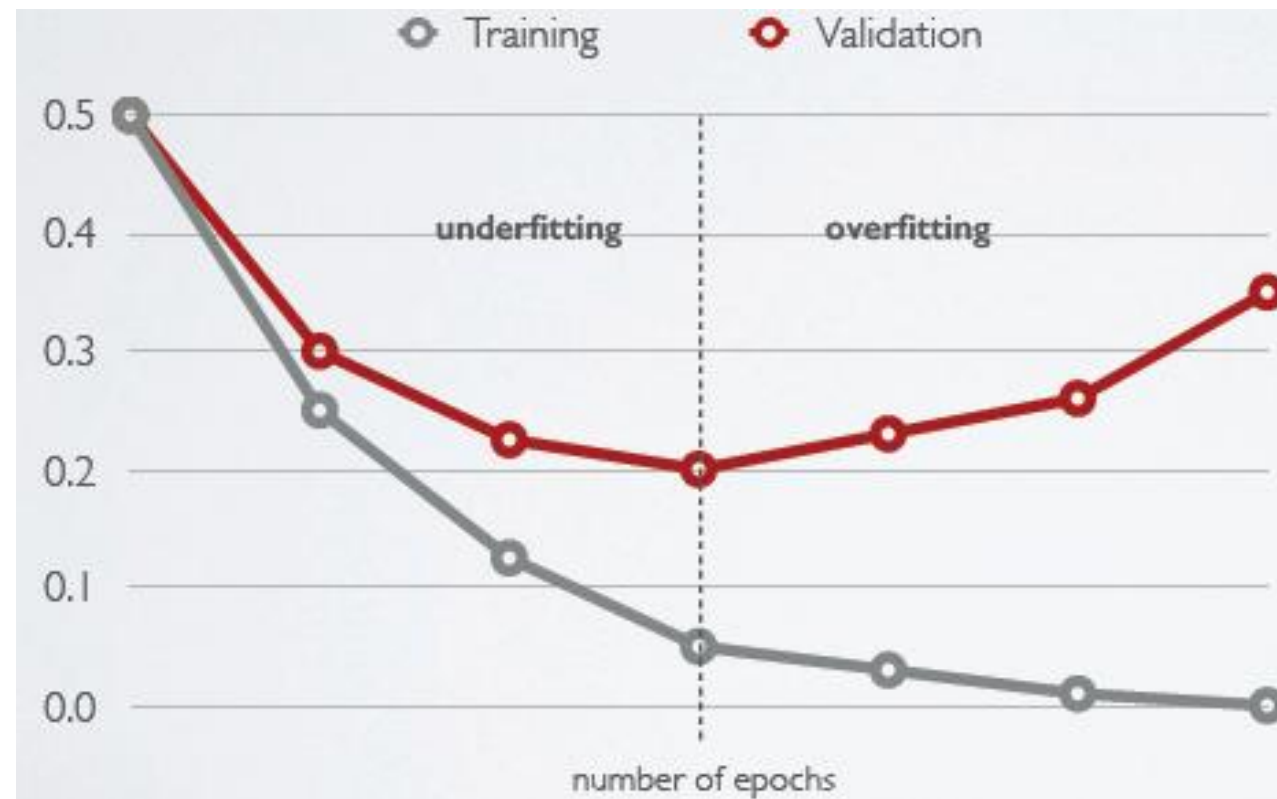
nn.init.sparse_(w, sparsity, std=0.01)
```

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

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

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
(ех: должны содержать представителей всех классов)

появляется новый гиперпараметр – размер батча
он связан с другими гиперпараметрами

Плоские минимумы

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

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

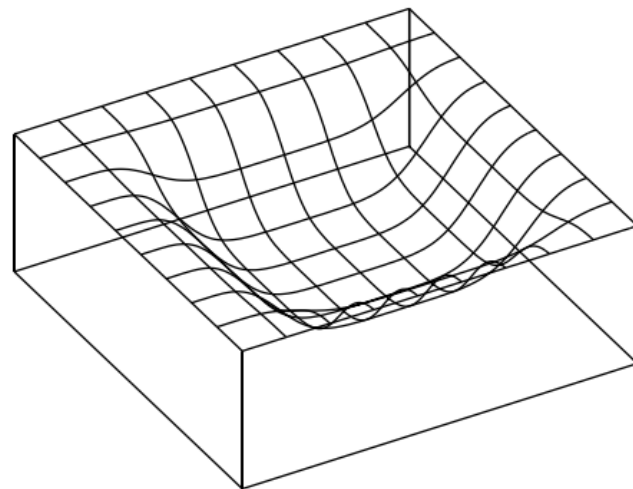


Figure 1: *Example of a “flat” minimum.*

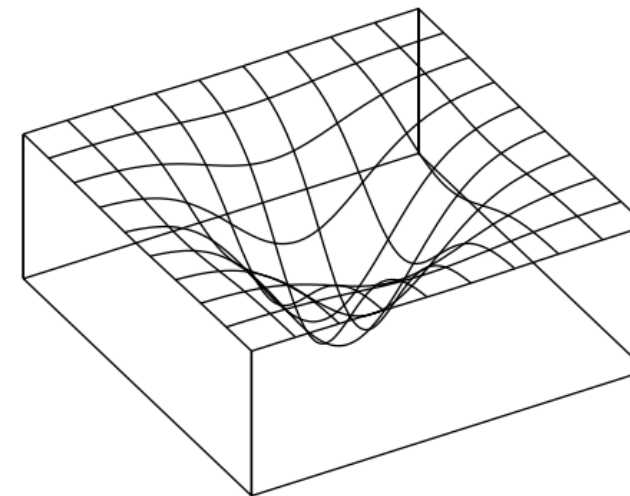


Figure 2: *Example of a “sharp” minimum.*

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

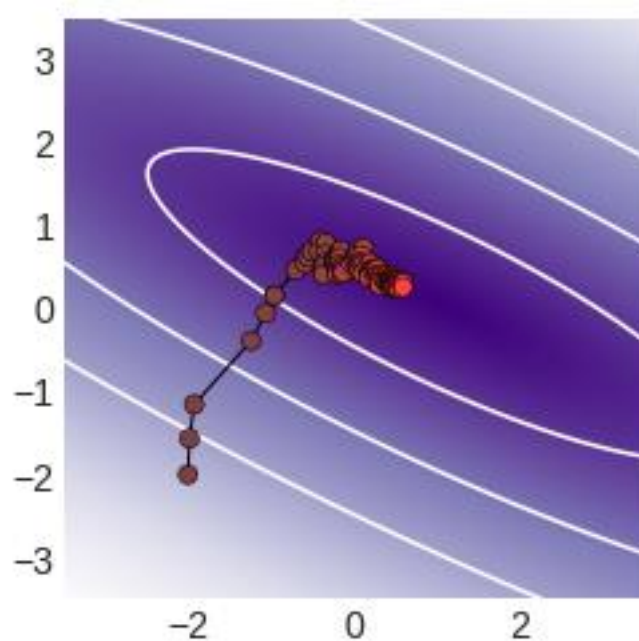
Продвинутая оптимизация

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 (дальше)**
 - **все параметры разные –**
скорость обучения для каждого параметра
 - **градиент случаен (зависит от батча) –**
добавляем градиенту инертность

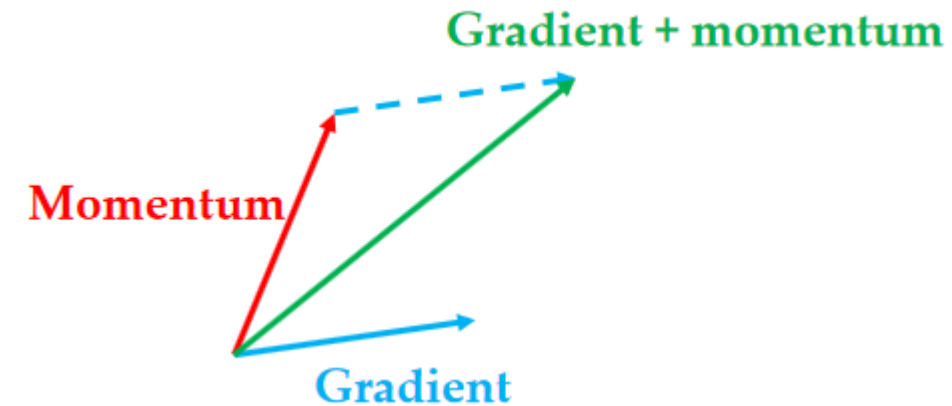
Продвинутая оптимизация: инерция

стохастический градиент с инерцией (momentum)

$$m^{(t+1)} = \rho m^{(t)} + \nabla L^{(t)}(w^{(t)})$$

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

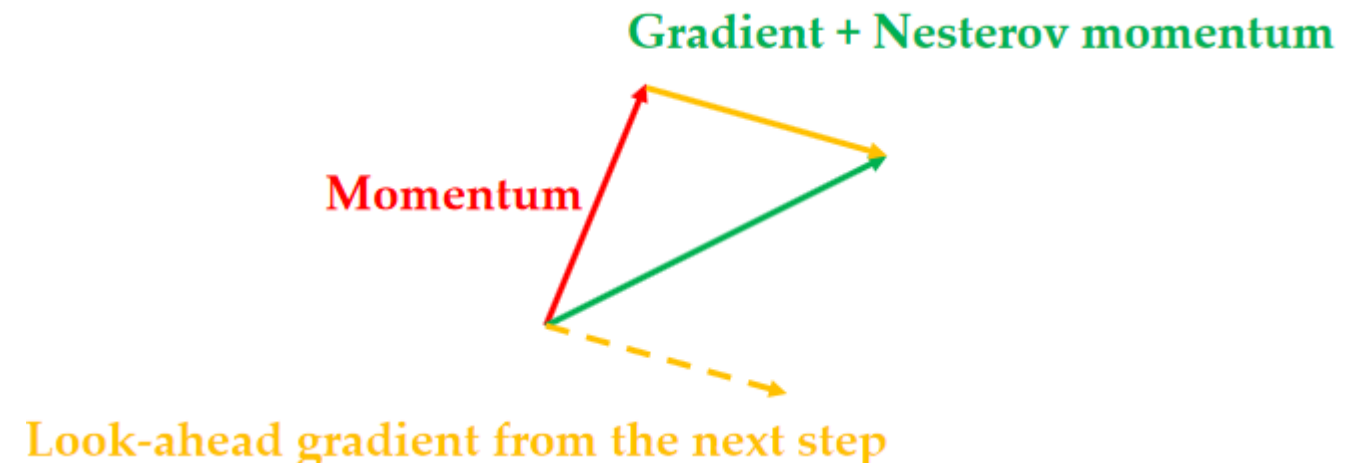
добавление инерции



метод Нестерова

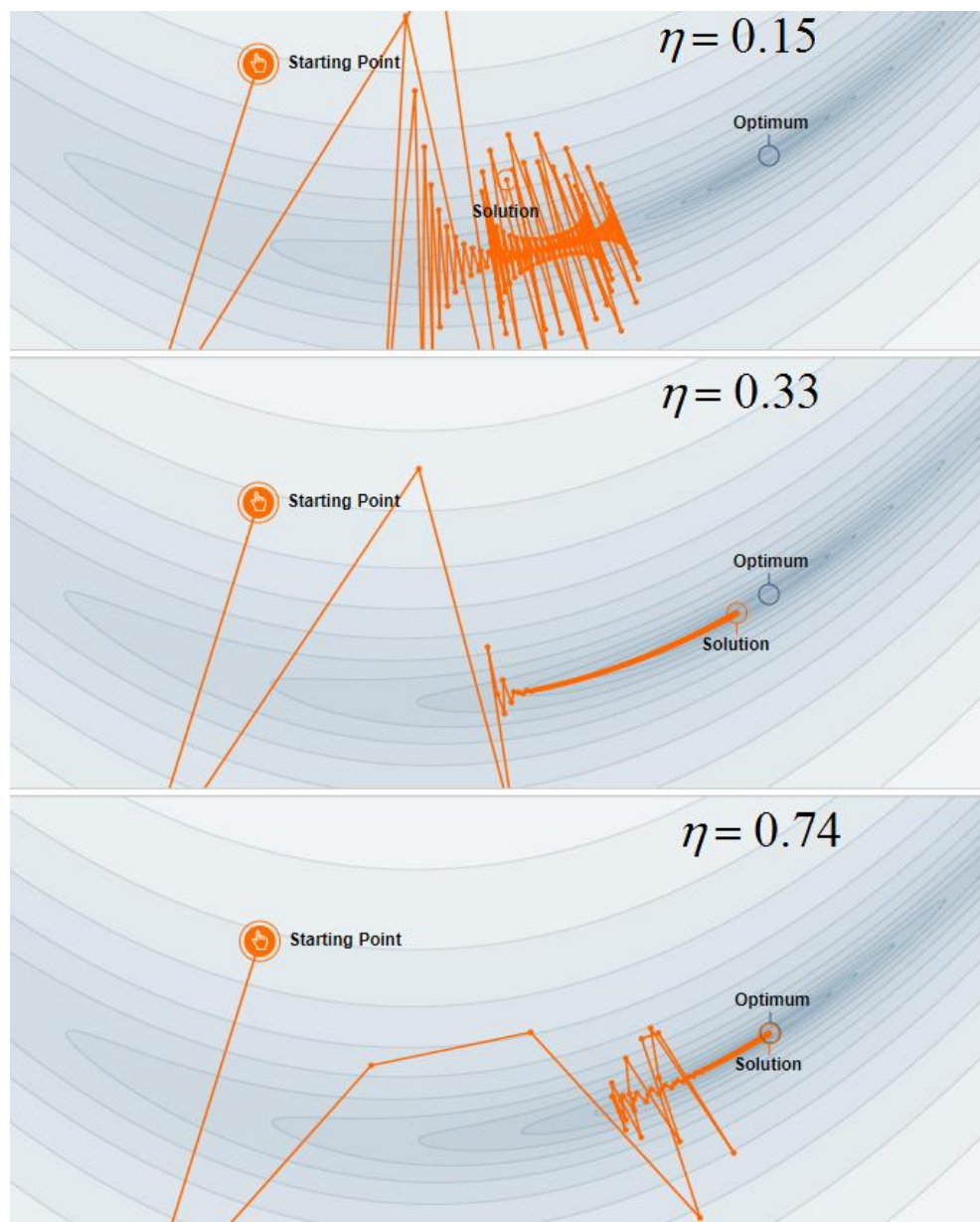
$$m^{(t+1)} = \rho m^{(t)} + \nabla L^{(t)}(w^{(t)} - \eta m^{(t)})$$

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



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

Иллюстрация СГ с моментом



Добавление инерции может помочь, когда

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

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

Продвинутая оптимизация – Адаптивная

Adagrad [Duchi и др., 2011]

$$v_i^{(t+1)} = v_i^{(t)} + (\nabla_i L^{(t)}(w^{(t)}))^2$$
$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{v_i^{(t+1)} + \varepsilon}} \nabla_i L^{(t)}(w^{(t)})$$

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

$$v_i^{(t+1)} = \beta v_i^{(t)} + (1 - \beta)(\nabla_i L^{(t)}(w^{(t)}))^2$$
$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{v_i^{(t+1)} + \varepsilon}} \nabla_i L^{(t)}(w^{(t)})$$

Продвинутая оптимизация – Адаптивная

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

$$m_i^{(t+1)} = \alpha m_i^{(t)} + (1 - \alpha) \nabla_i L^{(t)}(w^{(t)})$$

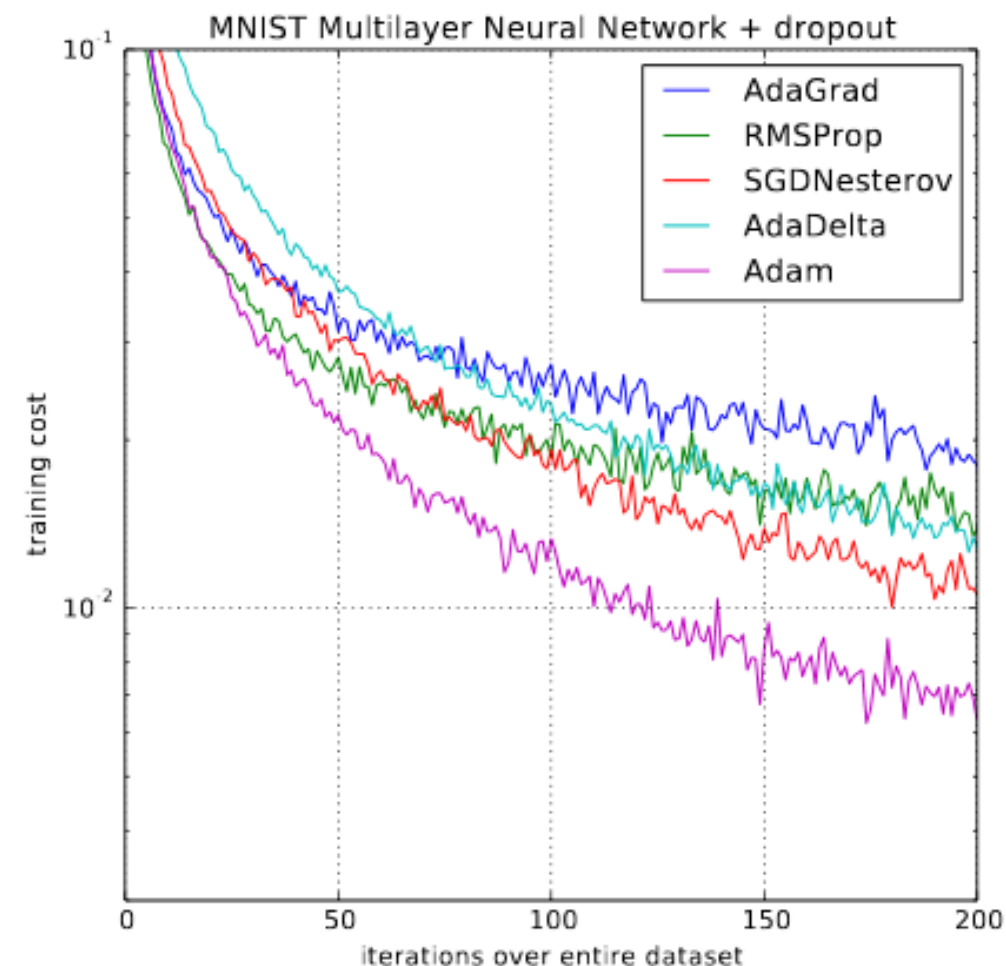
$$v_i^{(t+1)} = \beta v_i^{(t)} + (1 - \beta) (\nabla_i L^{(t)}(w^{(t)}))^2$$

корректировка смещения:

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

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

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



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

18k ссылок, неверное доказательство сходимости

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

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)**:

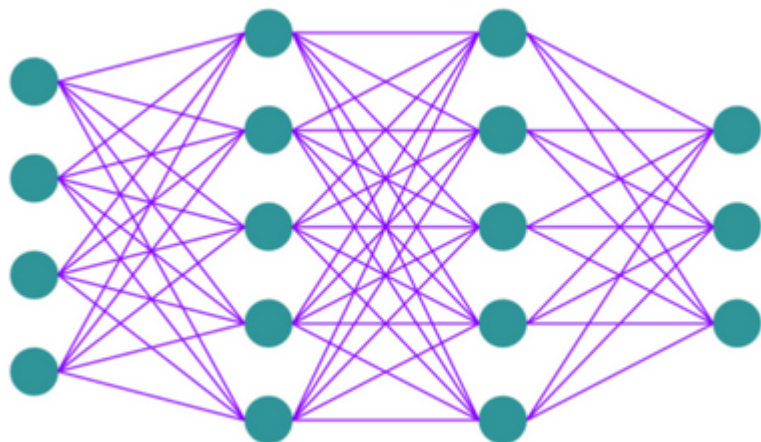
$$\nabla(L(w) + \lambda \|w\|^2) = \nabla L(w) + \lambda w$$

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

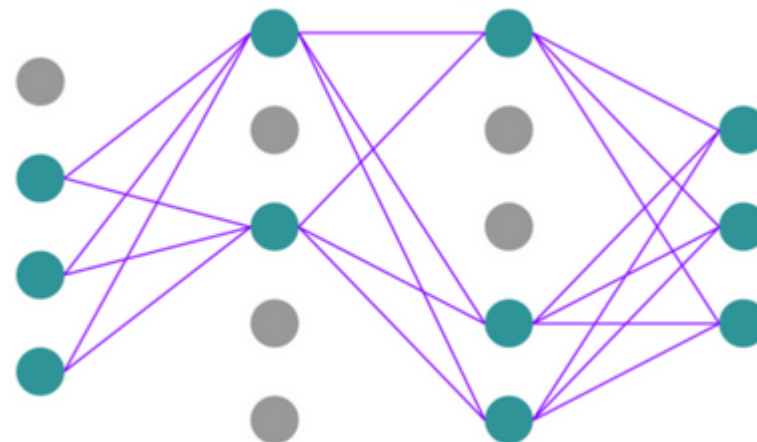
```
optimizer = torch.optim.SGD(model.parameters(), lr=0.1,  
                             momentum=0.9, weight_decay=0)
```

(Standard) Dropout

Without Dropout



Standard Dropout

Dropout with $p=0.5$

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

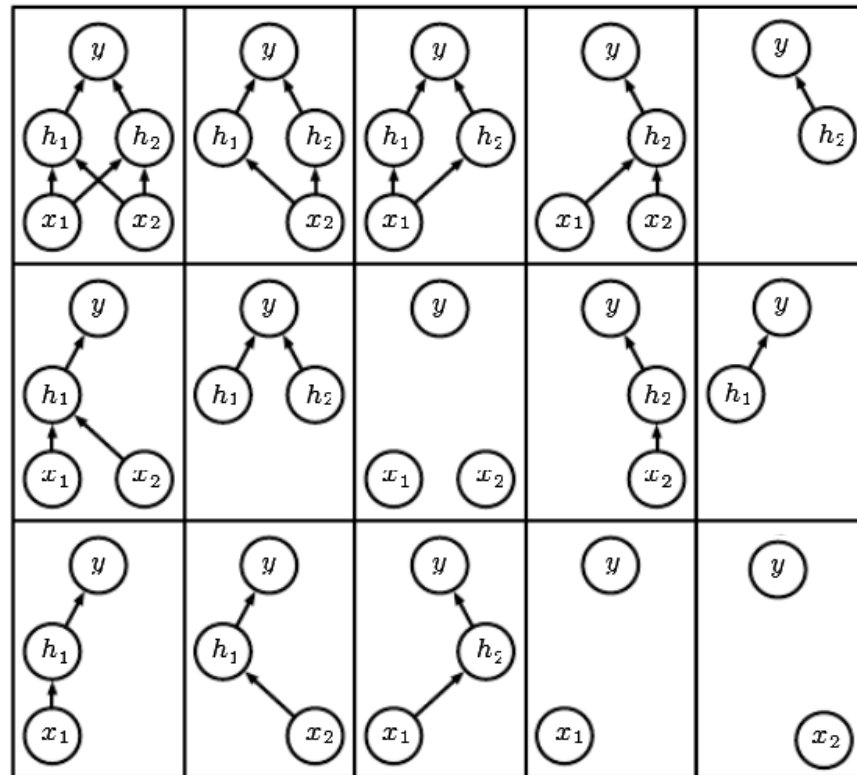
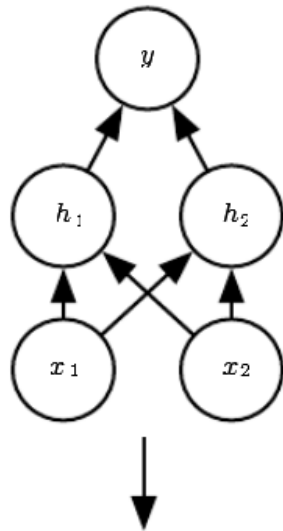
~ выбрасывание нейронов из сети с вероятностью p

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 информации

- уменьшение co-adaptations / co-dependencies соседних нейронов

- большая робастность нейронов

на рисунке:

В отличие от бэгинга все модели делят параметры (не независимы) [DLbook]

Inverted Dropout

обучение

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

Тест

$$y = f(Wx)$$

Во время тестирования ничего не делаем...

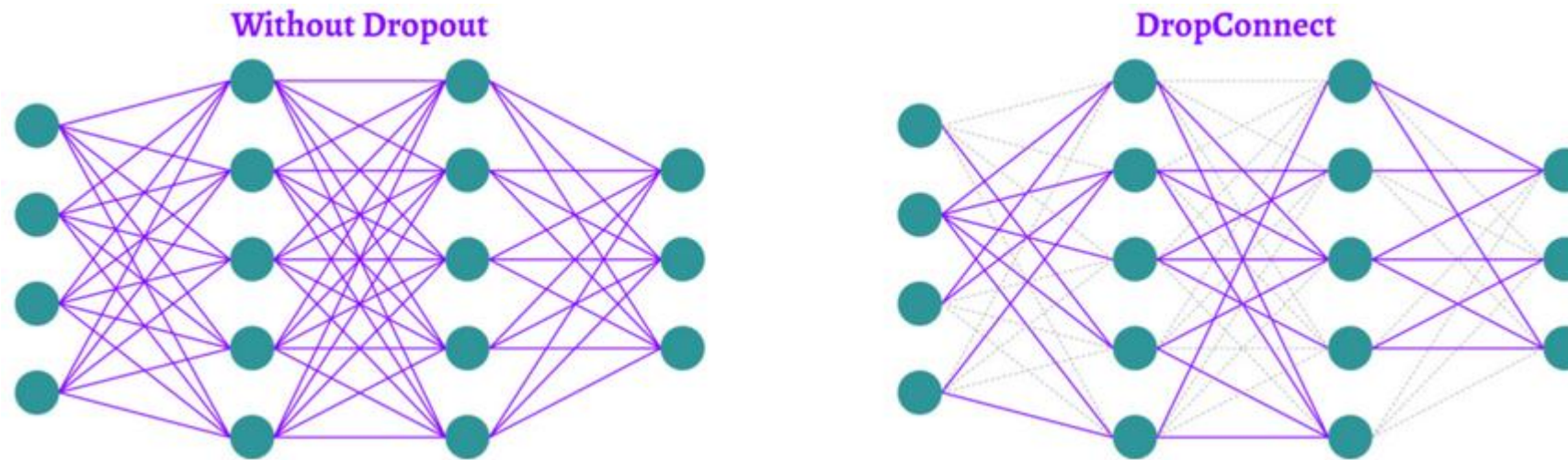
Standout

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

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

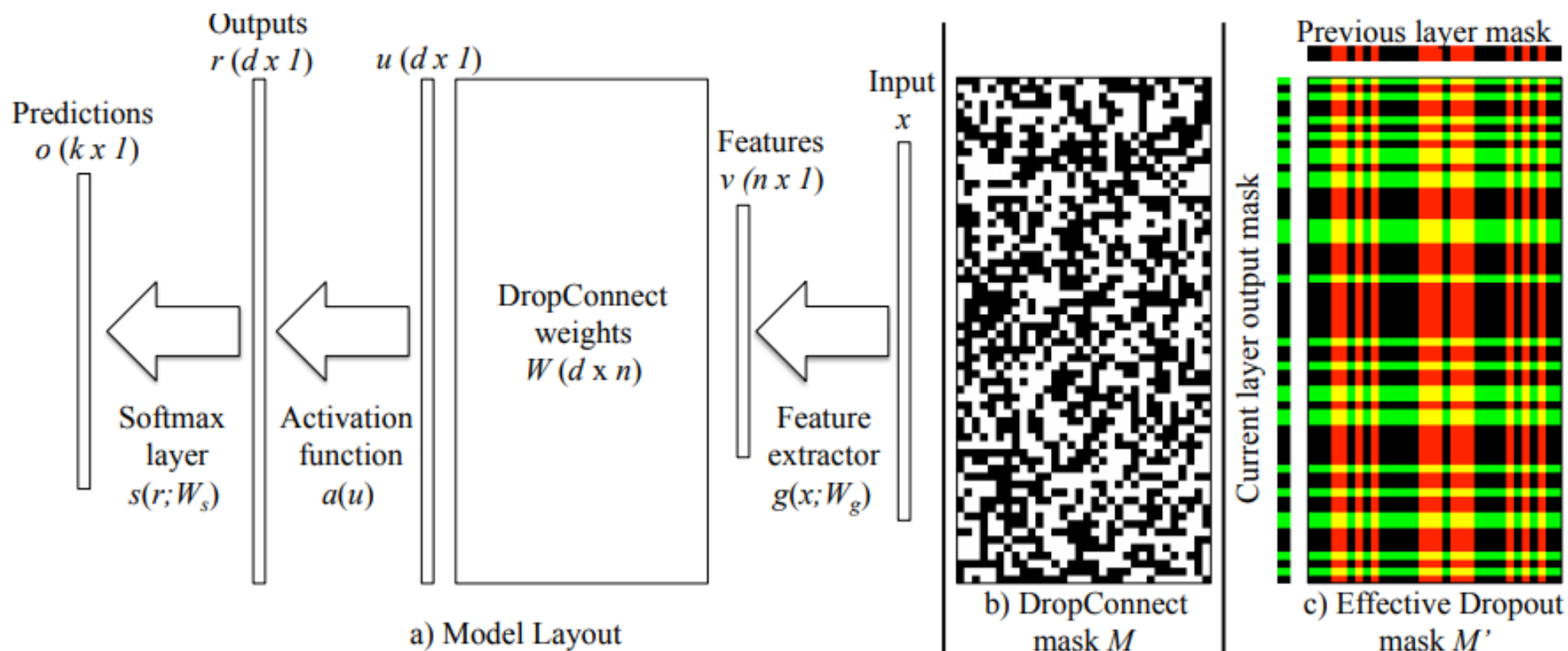
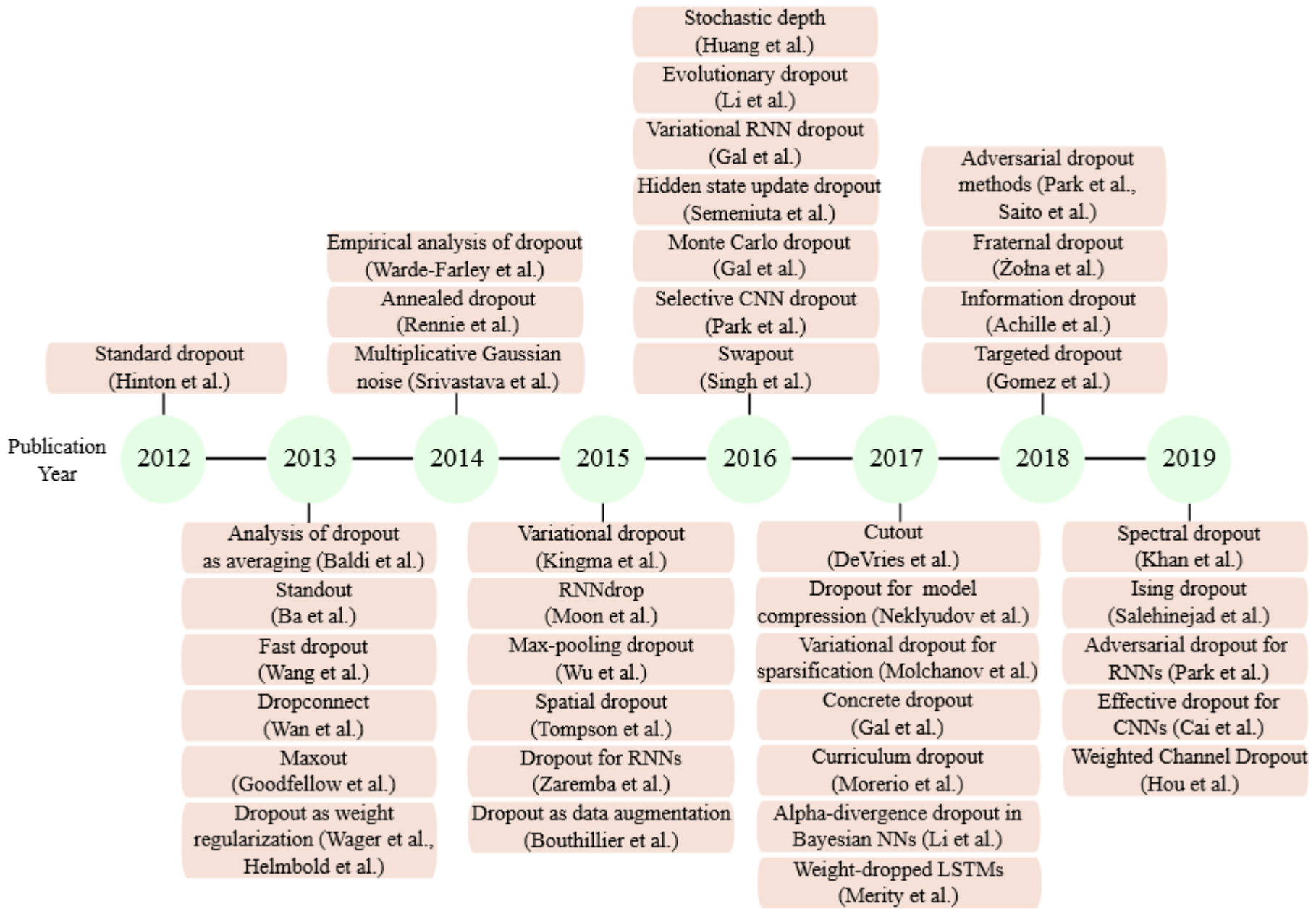
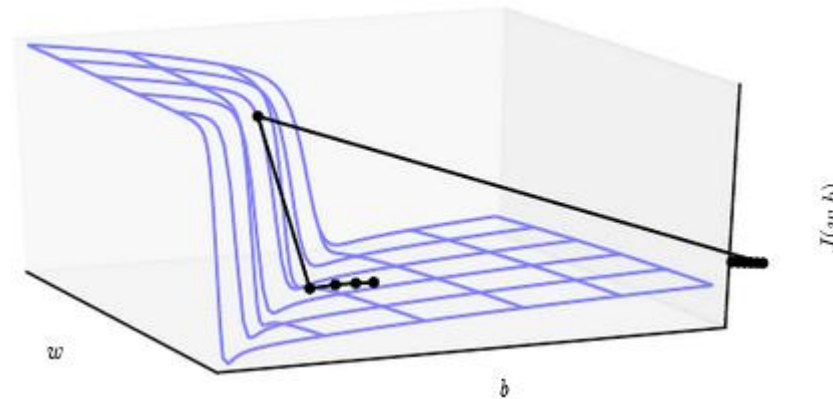


Figure 1. (a): An example model layout for a single DropConnect layer. After running feature extractor $g()$ on input x , a random instantiation of the mask M (e.g. (b)), masks out the weight matrix W . The masked weights are multiplied with this feature vector to produce u which is the input to an activation 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).



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)

мини-батч $\{x_i\}_{i=1}^m$

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

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

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

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

Надо определить параметры γ и β

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 тесте:

среднее и дисперсия –

- по обучению
- усреднение значений, что были во время обучения
- экспоненциальное среднее $=*$

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

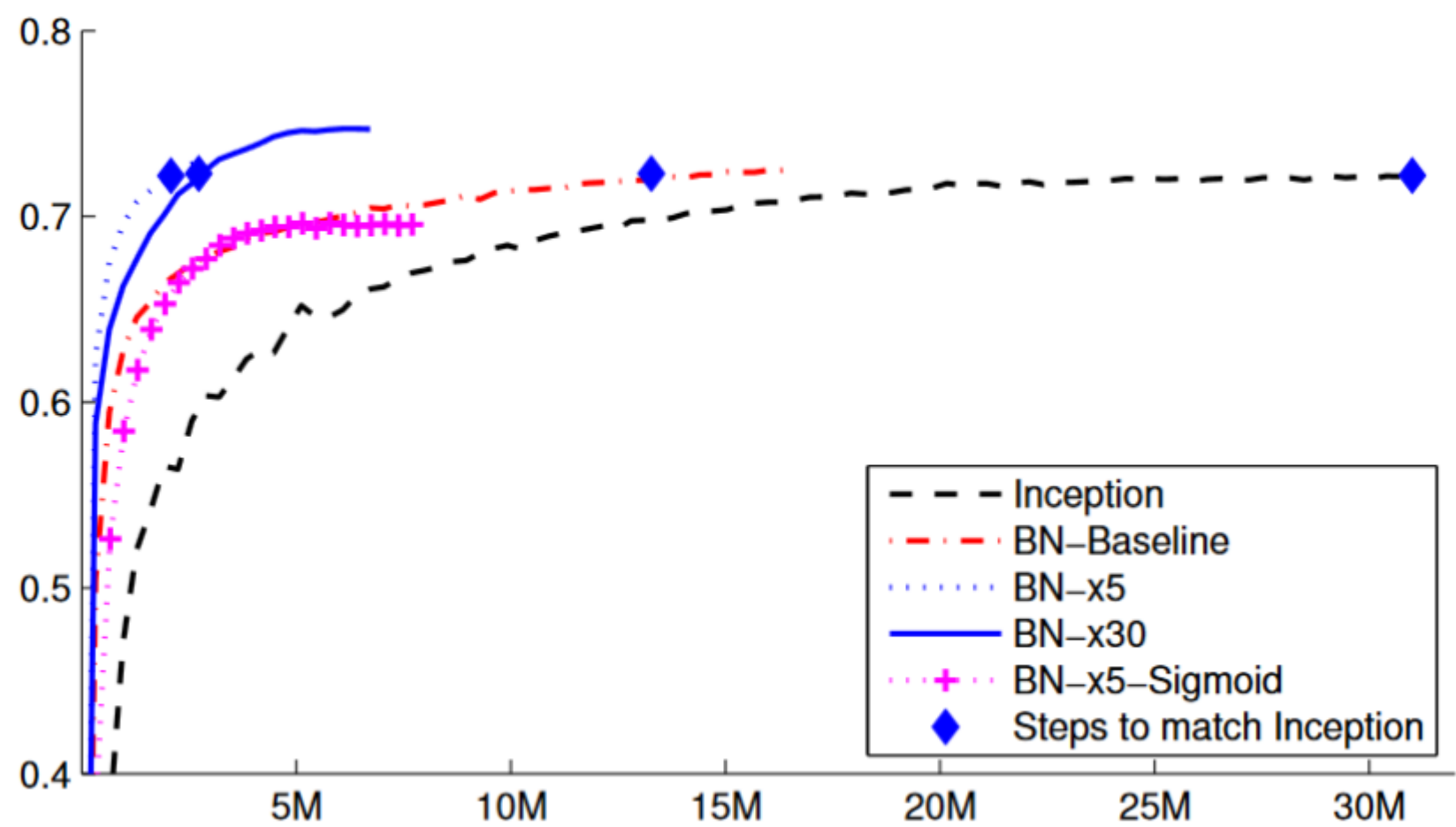
если Linear + BN,
то линейный слой делать без смещения

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 / выбору активации
- когда сеть обучена и не будет меняться м.б. BN можно устранить,
например свёртка + BN = свёртка

Удалось убрать LRN (Local Response Normalization),
но надо тщательнее перемешивать обучение

Батч-нормализация (Batch normalization)



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

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 регуляризует!

дальше 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.]])
```

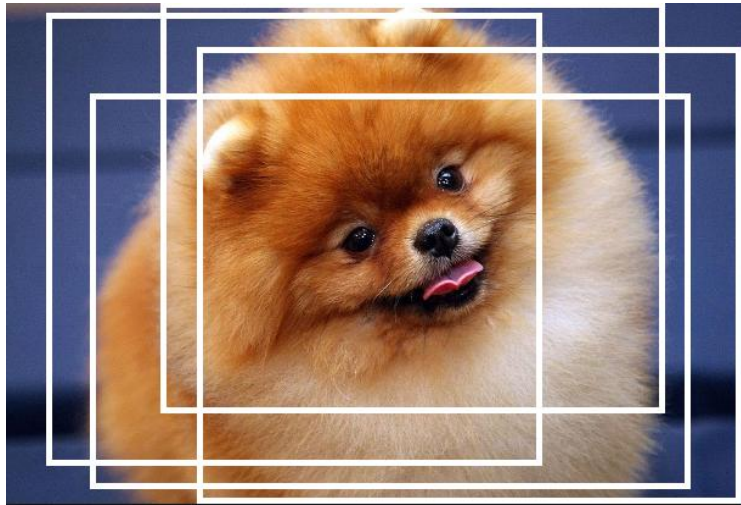
```
tensor([[ 0.,  4.,  0.,  0.],
        [ 0.,  0., 14., 16.],
        [18.,  0., 22., 24.],
        [ 0., 28.,  0.,  0.]])
```

```
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]]])
```

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

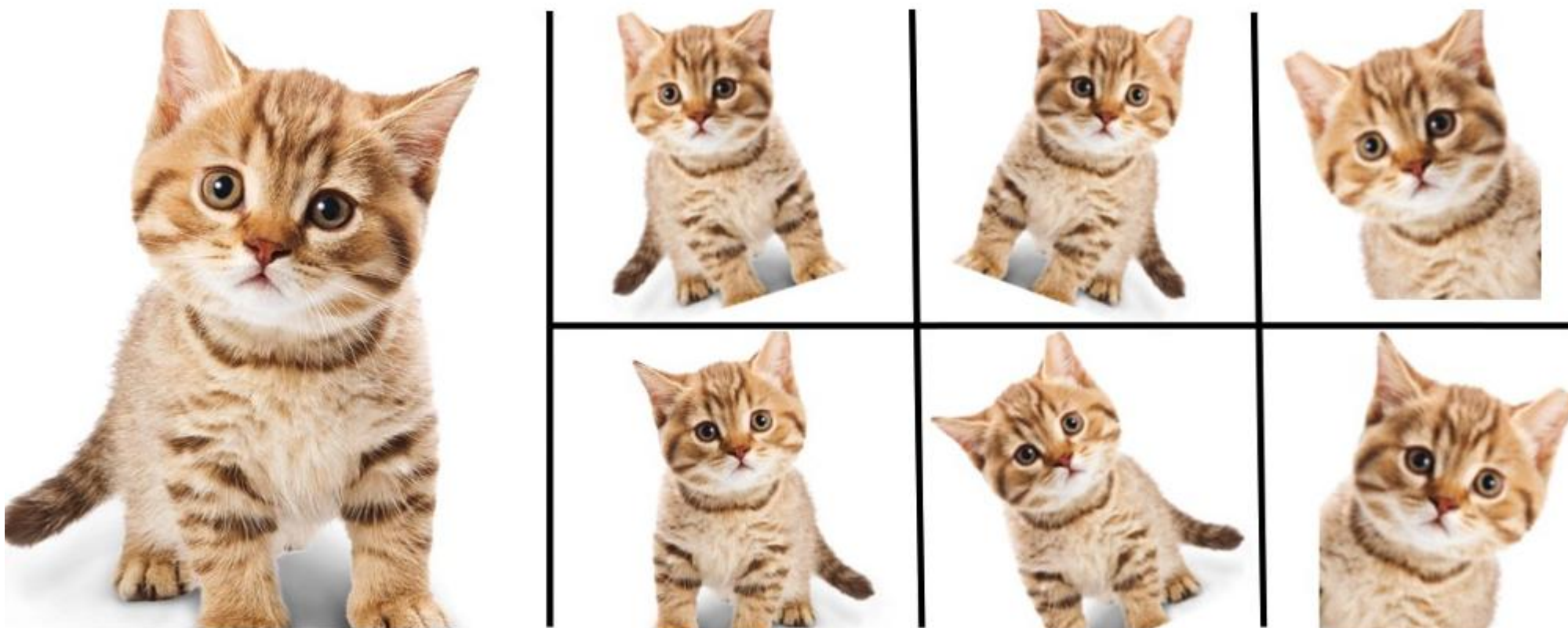
изображения

- симметрии (flip)
- вырезки (crop)
- изменение масштаба (resizing)
- 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/nanonets/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/109vu3F1LTzD1gdVV6cho9fKGx71zbF1l#scrollTo=wpWSjR-HBVso>

Аугментация: Mixup

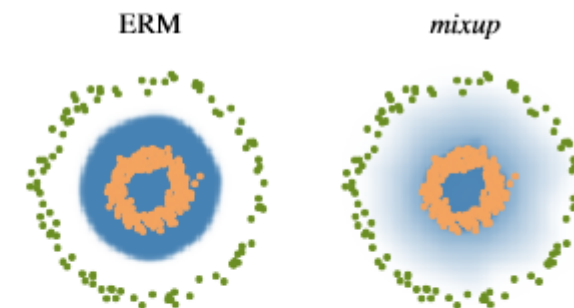
Contribution Motivated by these issues, we introduce a simple and data-agnostic data augmentation routine, termed *mixup* (Section 2). In a nutshell, *mixup* constructs virtual training examples

$$\begin{aligned}\tilde{x} &= \lambda x_i + (1 - \lambda)x_j, & \text{where } x_i, x_j \text{ are raw input vectors} \\ \tilde{y} &= \lambda y_i + (1 - \lambda)y_j, & \text{where } y_i, y_j \text{ are one-hot label encodings}\end{aligned}$$

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

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)	78.4 (+2.1)
ImageNet Loc (%)	46.3 (+0.0)	45.8 (-0.5)	46.7 (+0.4)	47.3 (+1.0)
Pascal VOC Det (mAP)	75.6 (+0.0)	73.9 (-1.7)	75.1 (-0.5)	76.7 (+1.1)

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 λ between two data points is sampled from the beta distribution $\text{Beta}(\alpha, \alpha)$. In our all experiments, we set α to 1, that is λ 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

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 на разных эпохах обучения (аналог усреднения Поляка)

+ 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

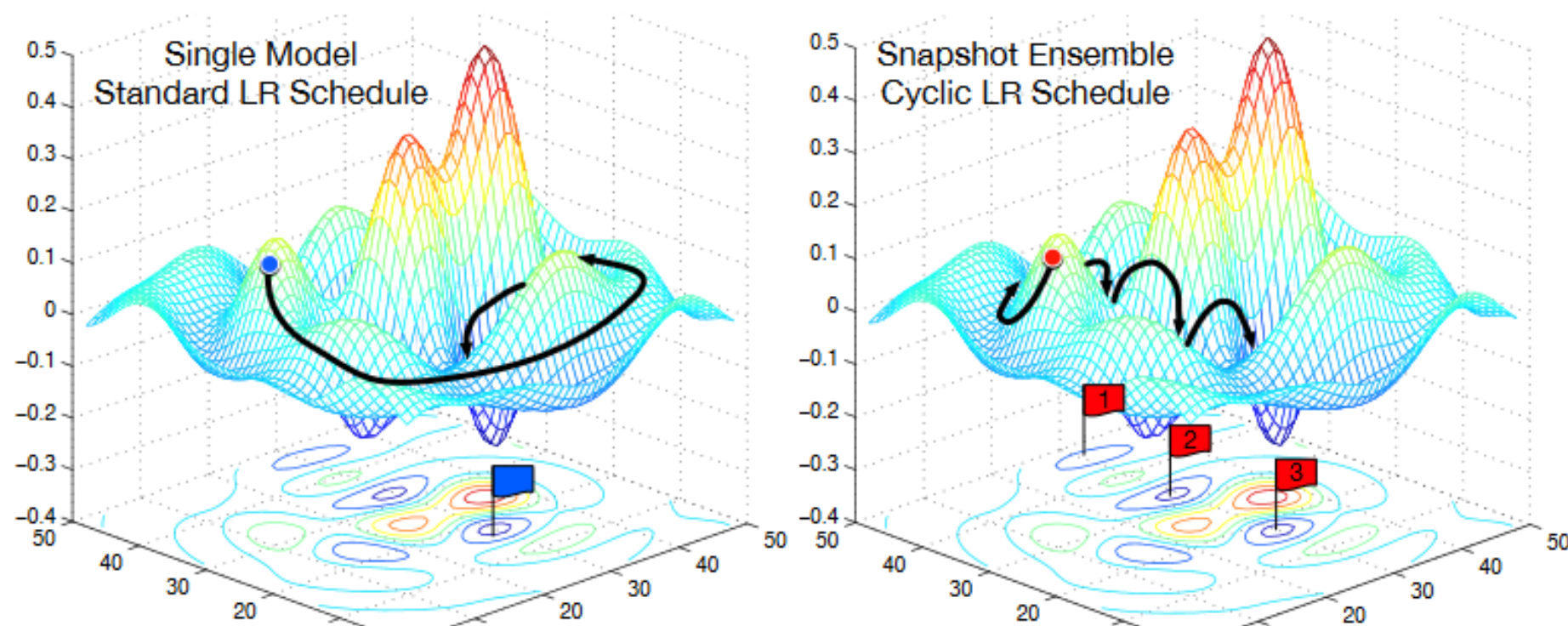


Figure 1: **Left:** Illustration of SGD optimization with a typical learning rate schedule. The model converges to a minimum at the end of training. **Right:** Illustration of Snapshot Ensembling. The model undergoes several learning rate annealing cycles, converging to and escaping from multiple local 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>

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)

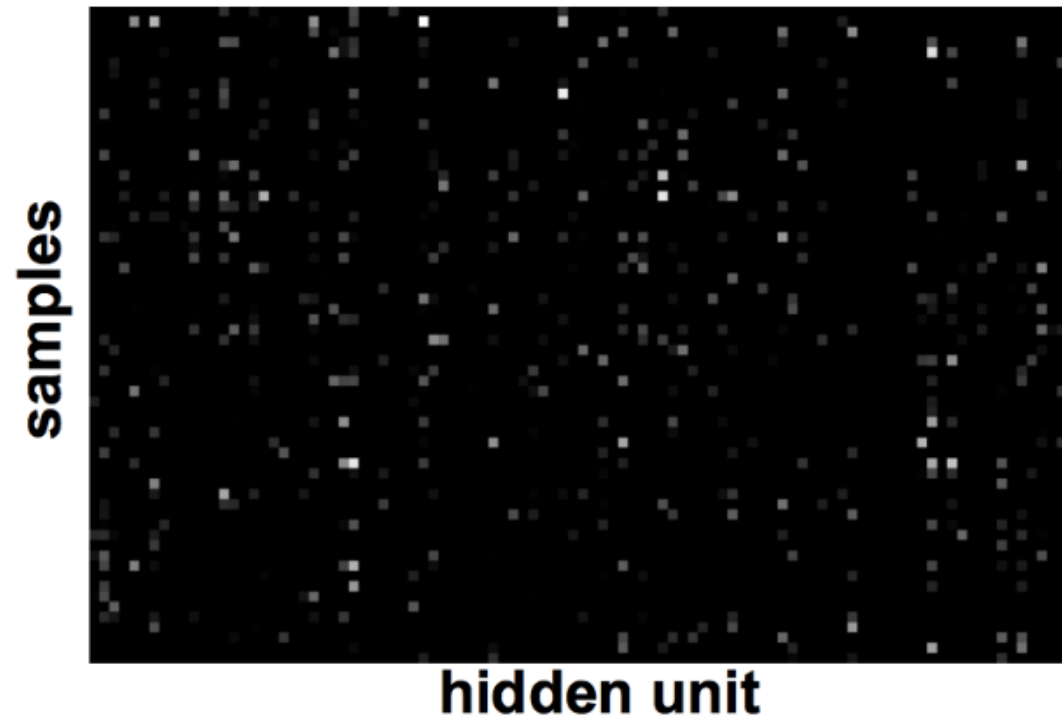
$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x + \varepsilon) - f(x - \varepsilon)}{2\varepsilon}$$

проверка реализации обратного и прямого распространения

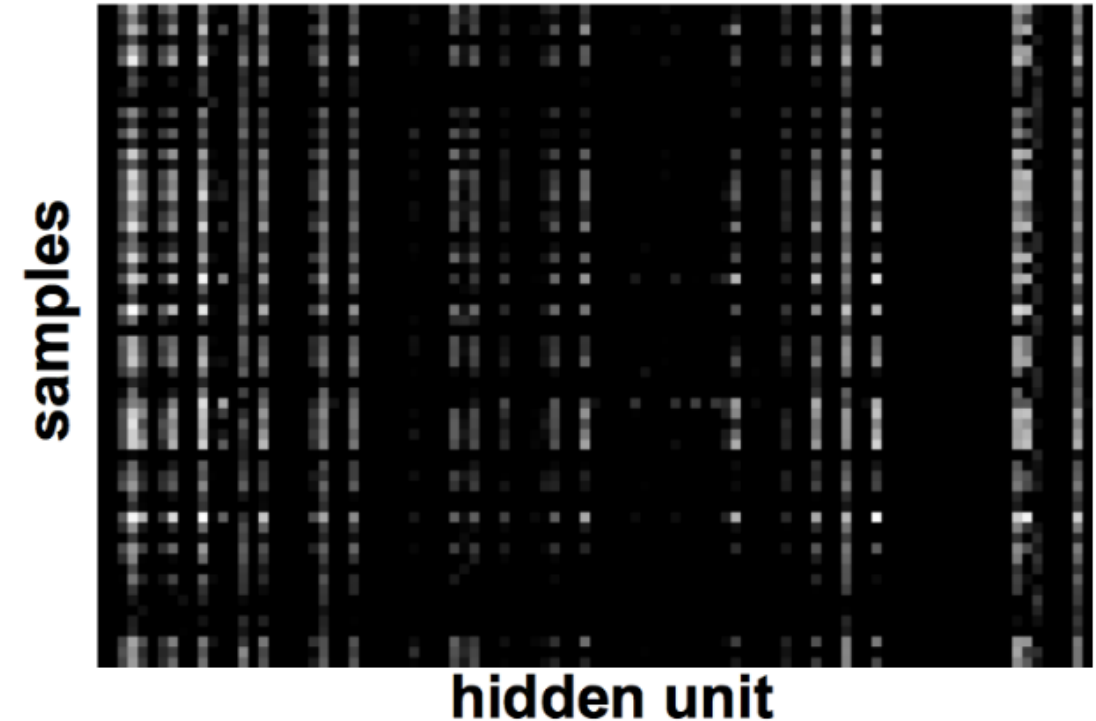
Диагностика проблем с НС: Визуализация

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

Хорошо:



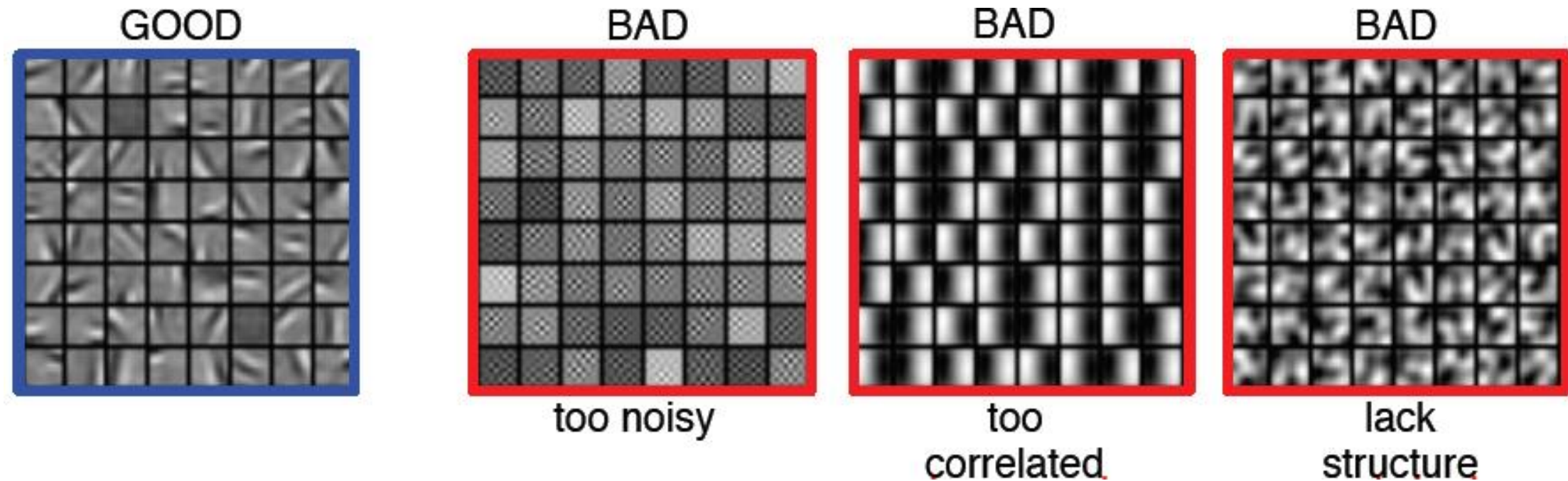
Плохо:



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

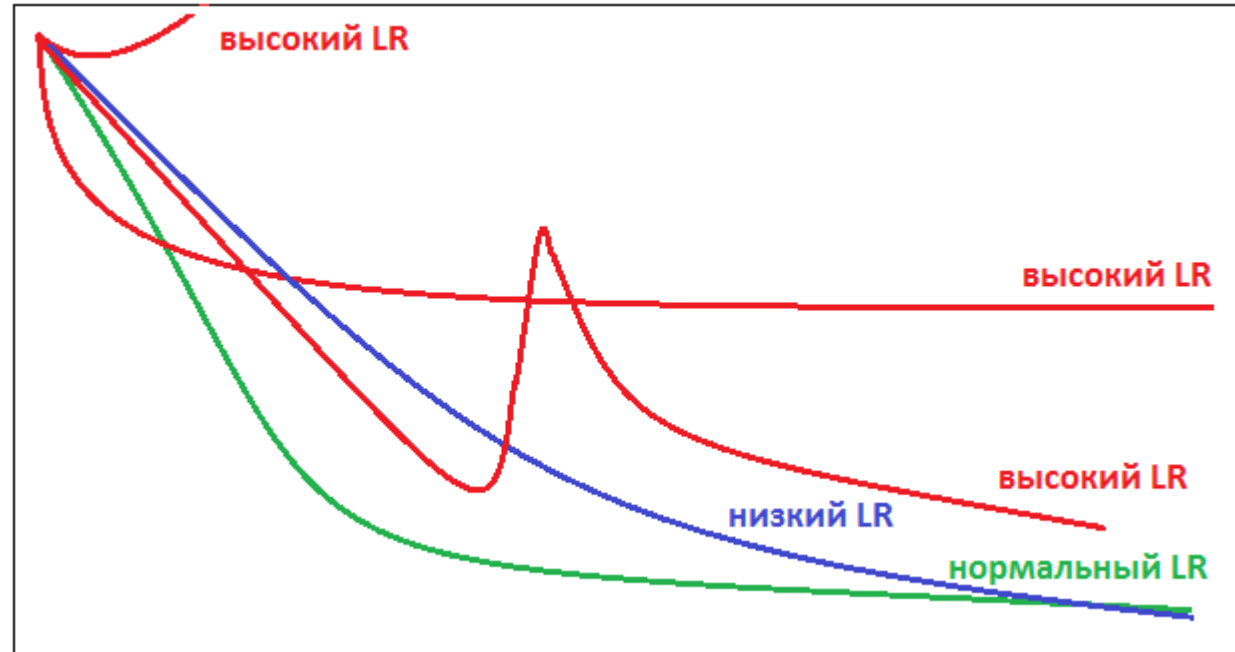
Насыщены ли нейроны ещё до обучения?
Нормировка!

Диагностика проблем с НС: Изменения весов

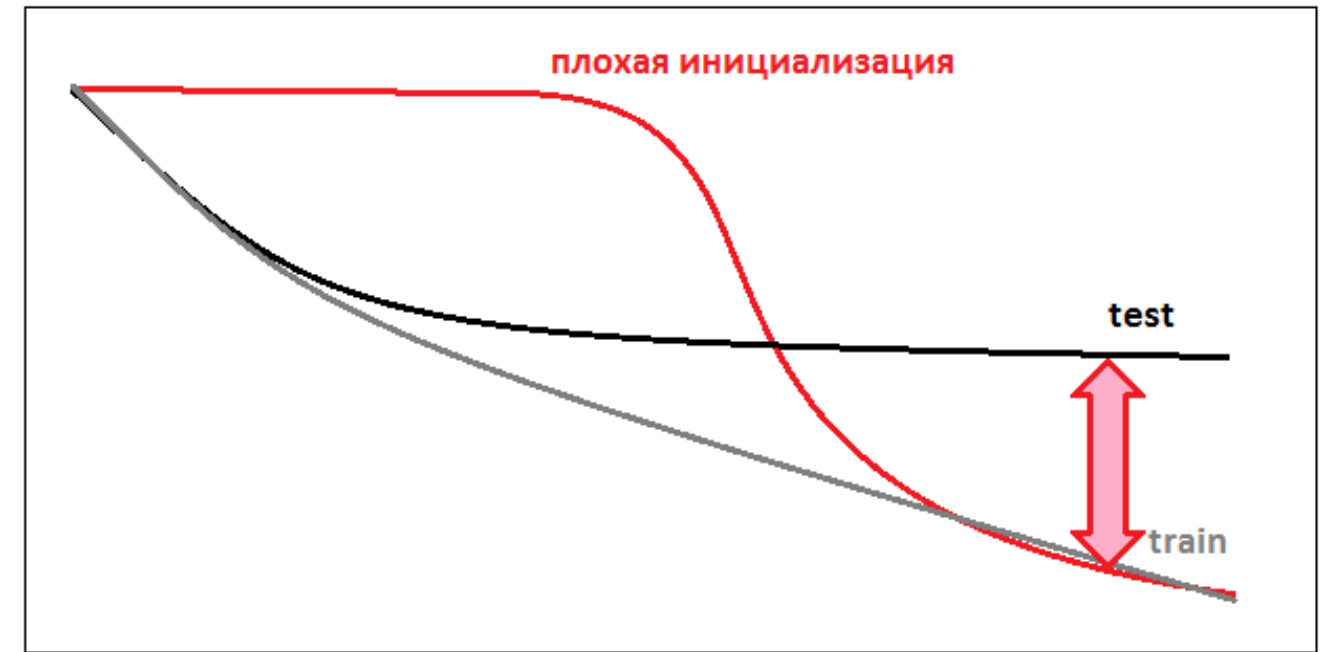
Насколько меняются веса за итерацию (~ 0.1%)

Диагностика проблем с НС: Динамика ошибки

Как ведёт себя ошибка обучения – настроить 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)**

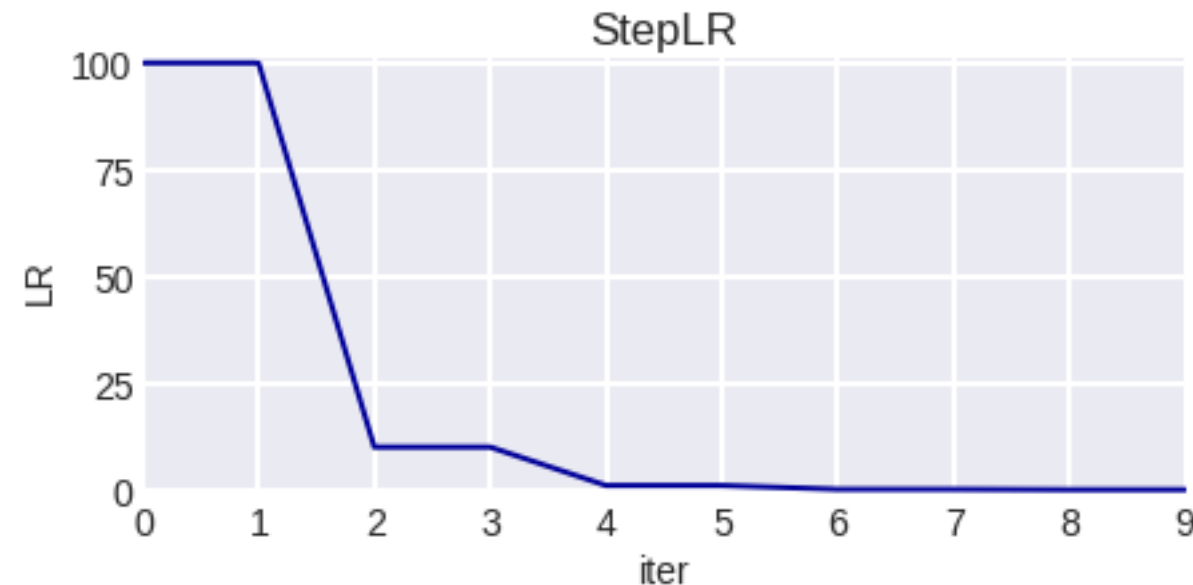
**на практике снижают темп!
начинают с такого, что метод не расходится**

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

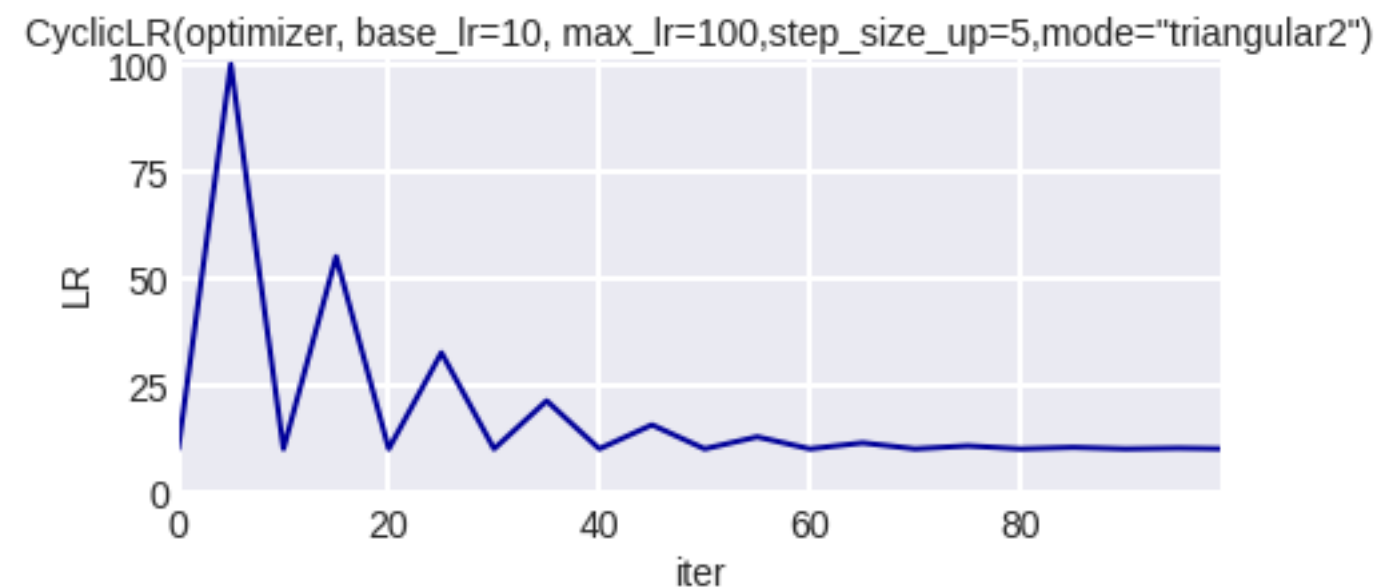
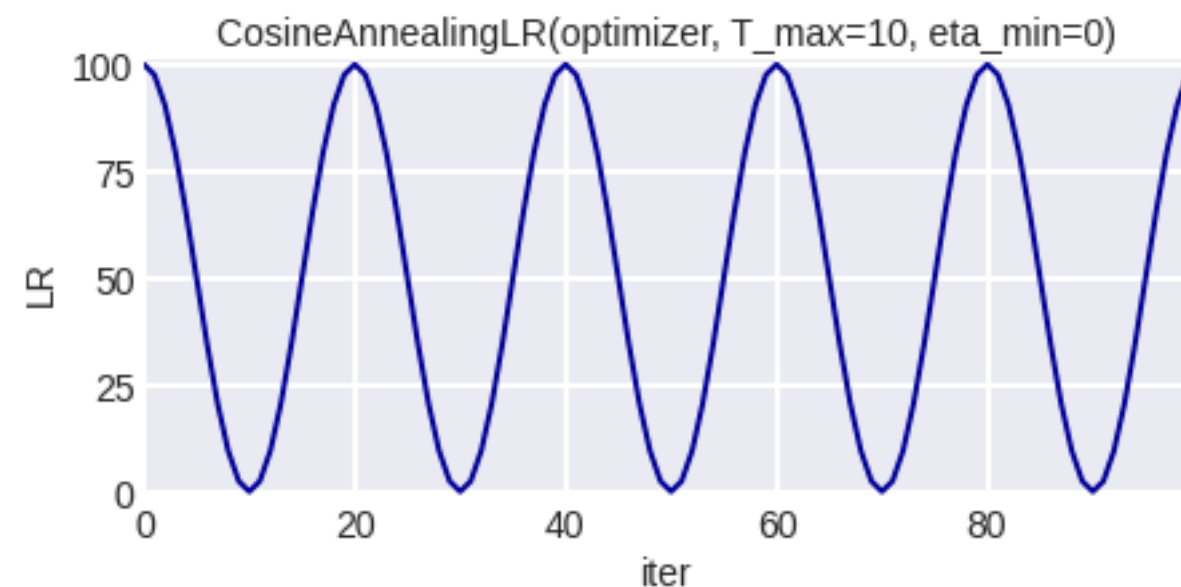
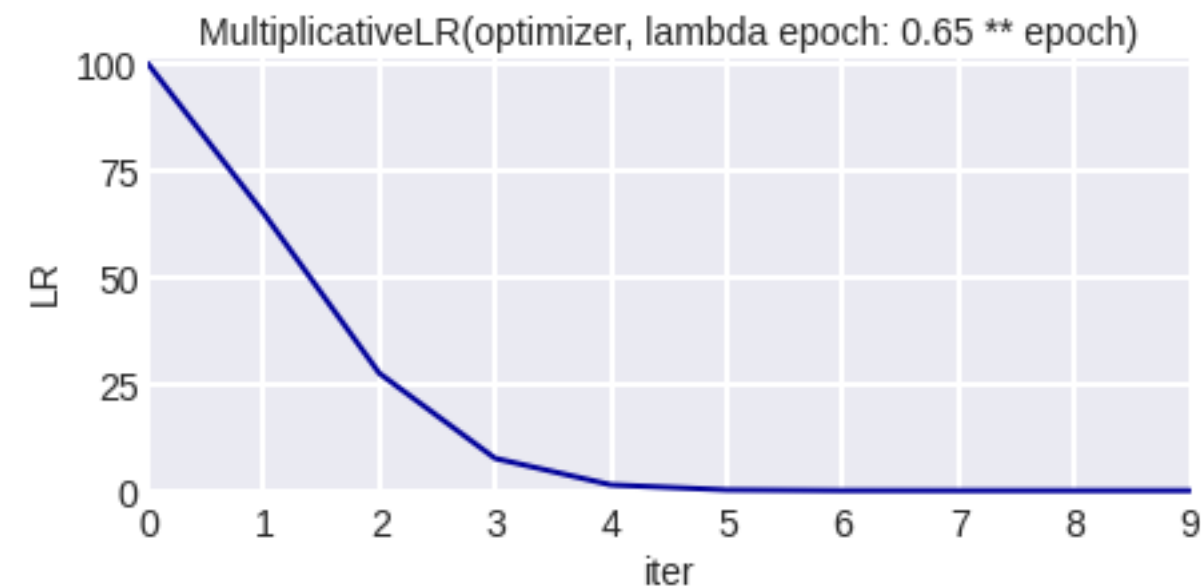
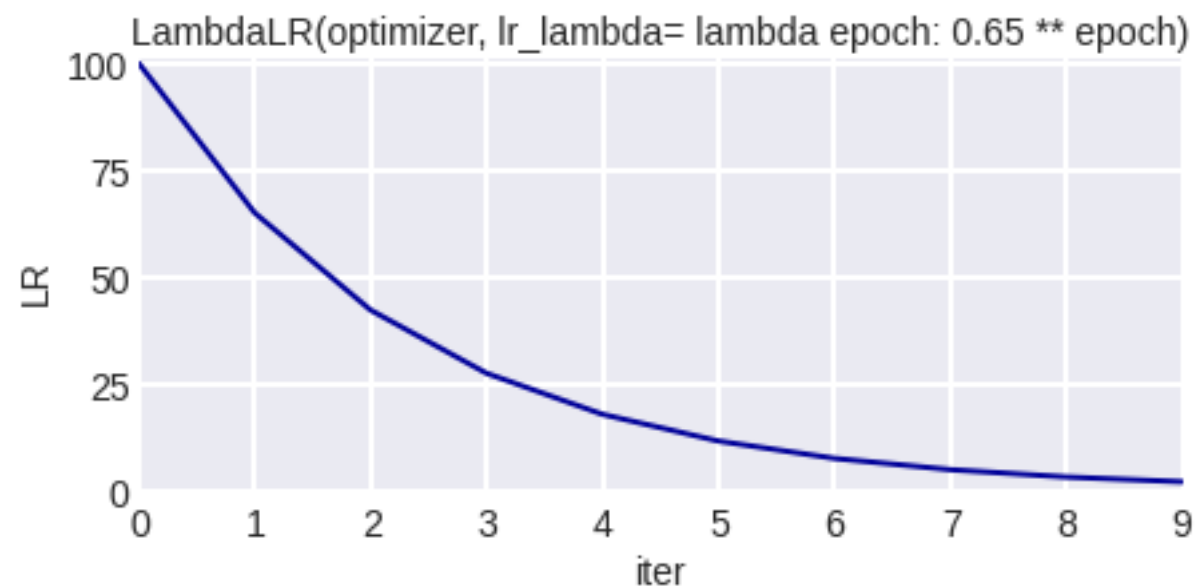
Минутка кода: сценарий изменения темпа

```
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>

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)

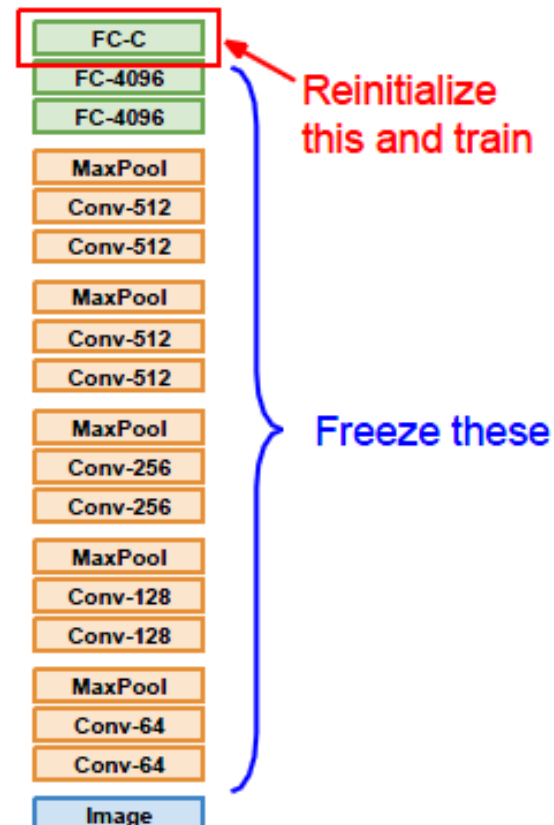
Transfer Learning

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

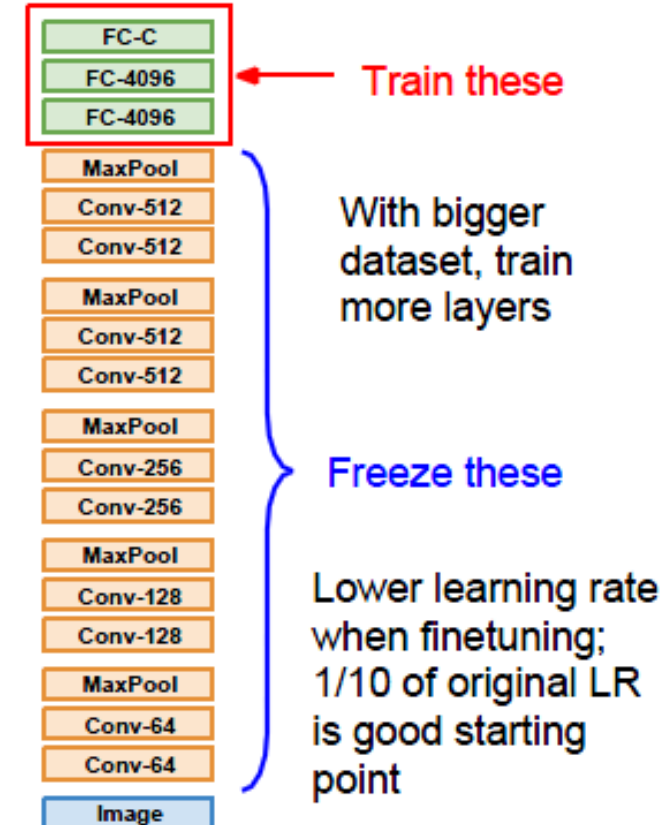
1. Train on Imagenet



2. Small Dataset (C classes)



3. Bigger dataset



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

Transfer Learning

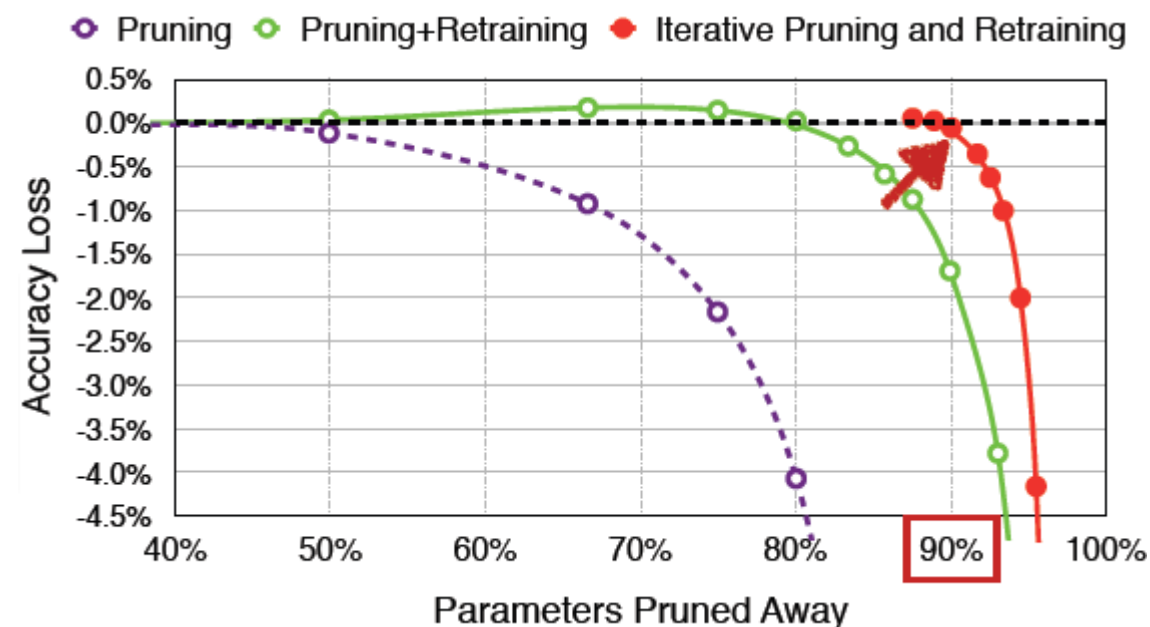
Можно решать и другие задачи – для этого перестраивается голова сети
(как бы получаем признаки обученной сетью)

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

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

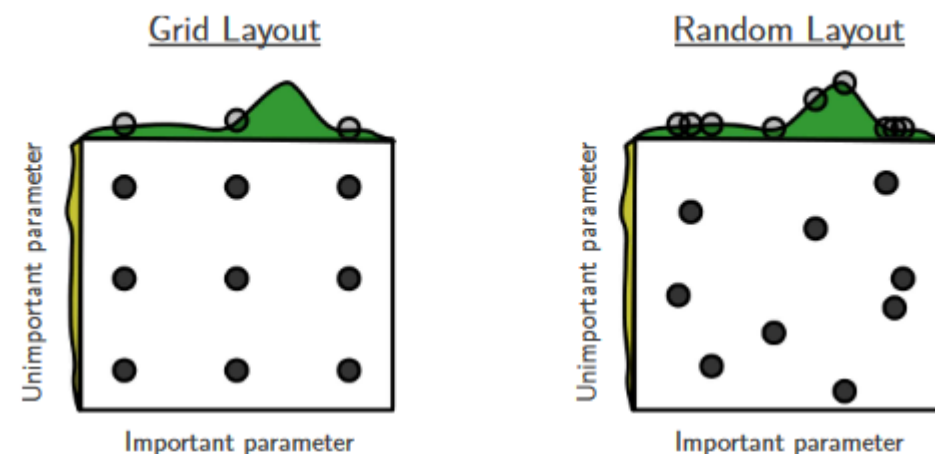


Figure 1: Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct 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 найденного решения

есть другие методы оптимизации!

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**

ех: при дисбалансе в последнем слое надо так, чтобы выдавалась малая вер-ть

- **если что-то не получается можно понизить размерность**

по-умному (РСА 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

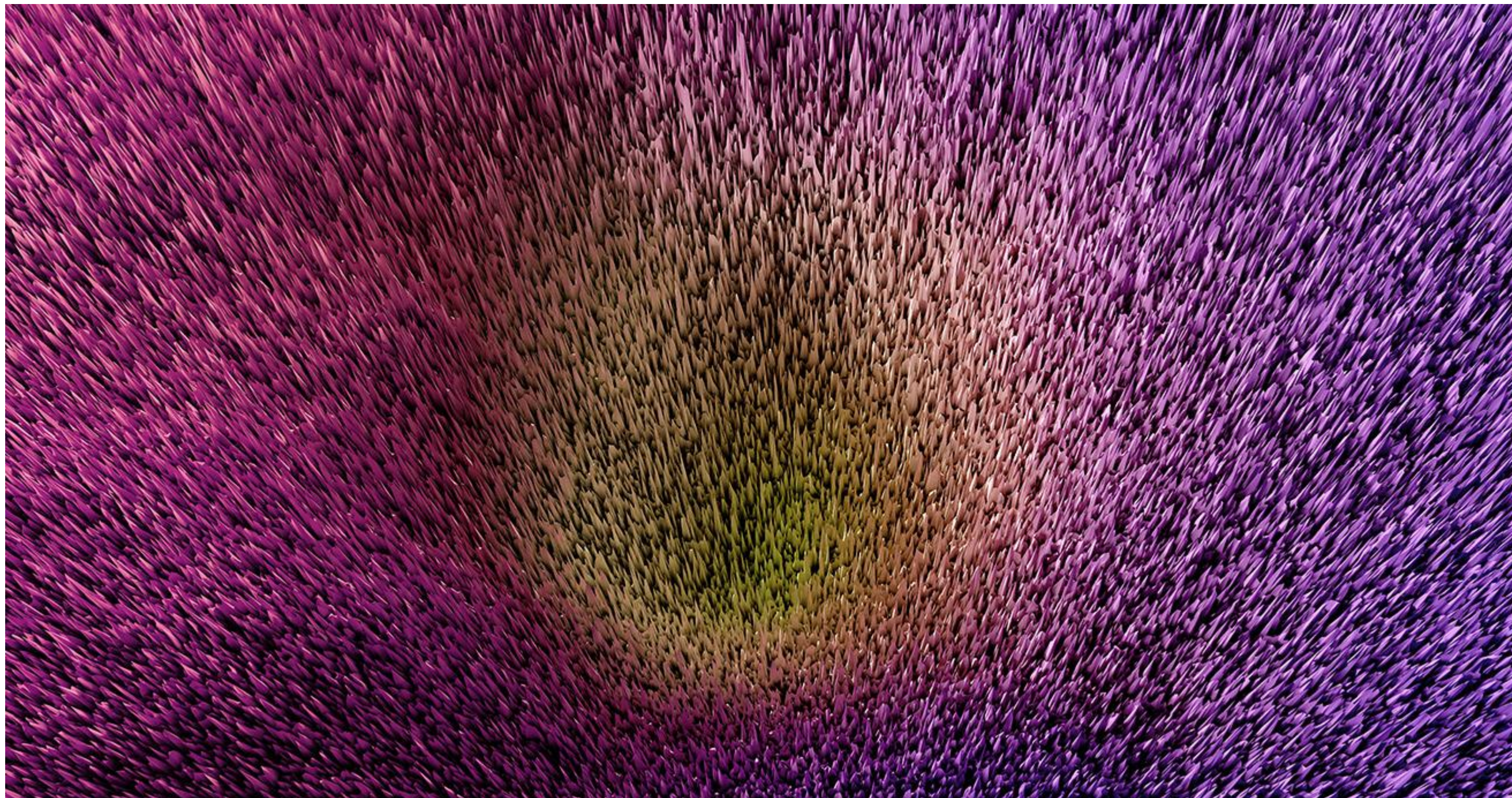
Глупые ошибки

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

Поверхности функций ошибки

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