

Лекция 2. Борьба с переобучением

Бабич Павел Васильевич

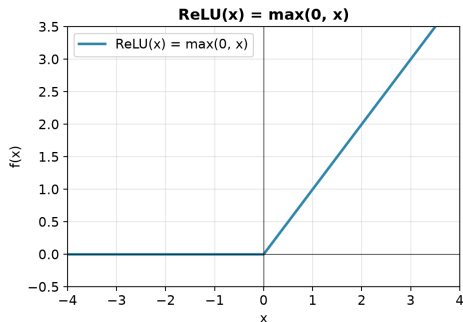
ИММиКН

2026

ReLU — Rectified Linear Unit

$$x = \max(0, x)$$

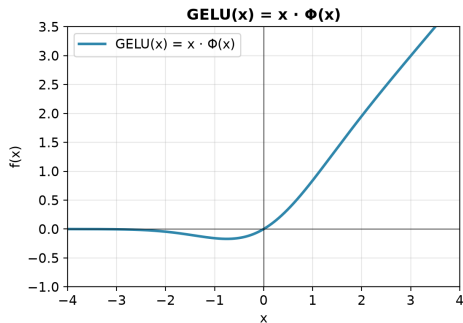
- Самый распространённый выбор для скрытых слоёв
- Быстрый, устраняет «исчезающие градиенты»
- **Минус:** «мёртвые нейроны»



Интерпретация: «вероятностный гейт»

- $\Phi(x)$ = вероятность того, что $z \sim \mathcal{N}(0, 1)$ меньше x
- $x > 0$: $\Phi(x) \approx 1 \Rightarrow$ нейрон почти полностью **пропускает** сигнал
- $x < 0$: $\Phi(x) \approx 0 \Rightarrow$ нейрон почти полностью **подавляет** сигнал
- $x \approx 0$: $\Phi(0) = 0.5 \Rightarrow$ **неопределённость**, плавный переход

В отличие от ReLU («резкий» порог $x = 0$), GELU плавно «решает», пропускать ли сигнал.

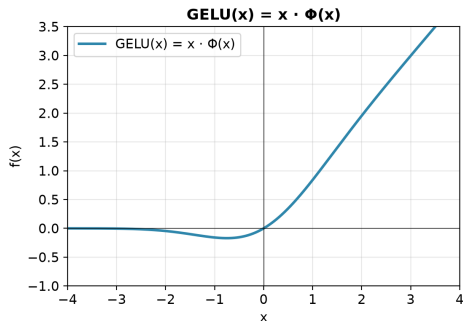


GELU — Gaussian Error Linear Unit

$$\text{GELU}(x) = x \cdot \Phi(x)$$

где $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt$ — функция распределения нормального закона.

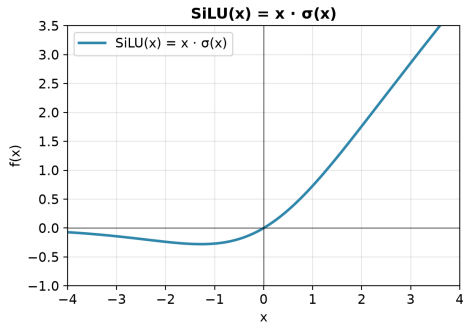
- Плавная аппроксимация ReLU
- Стандарт трансформеров (BERT, GPT, LLaMA)
- Лучше сходимость при преобразовании данных



SiLU (Swish)

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

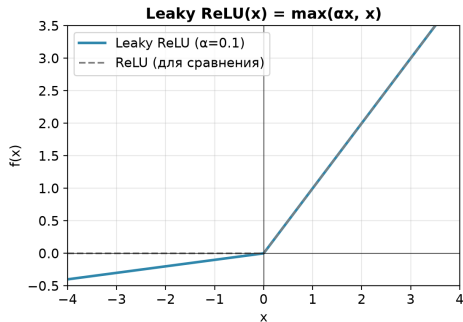
- Плавная, **немонотонная**
- EfficientNet, MobileNetV3, ResNeXt
- Альтернатива ReLU в современных CNN



Leaky ReLU

$$\text{LReLU}(x) = \max(\alpha x, x), \quad \alpha \approx 0.01$$

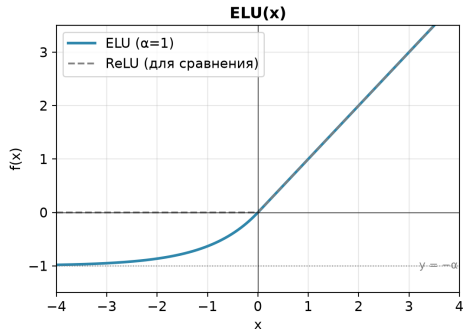
- Избегает «мёртвых нейронов»
- Небольшой наклон при $x < 0$
- Популярен в GAN (DCGAN, StyleGAN)



ELU — Exponential Linear Unit

$$\text{ELU}(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases}$$

- Выбросы ближе к нулю $\rightarrow$ лучше сходимость
- $\alpha = 1$ по умолчанию
- Чуть медленнее ReLU (экспонента)



Формула

Вход: $x \in \mathbb{R}^n$, параметры: $W \in \mathbb{R}^{n \times 2m}$, $b \in \mathbb{R}^{2m}$

$$\underbrace{h = xW + b}_{\in \mathbb{R}^{2m}} = \left[\underbrace{h_1}_{\in \mathbb{R}^m} \mid \underbrace{h_2}_{\in \mathbb{R}^m} \right] \implies \text{GLU}(x) = h_1 \odot \sigma(h_2)$$

h_1 — «сигнал», h_2 — «гейт» (управляет, сколько сигнала пропустить). $\sigma = \text{Sigmoid} \in (0, 1) \Rightarrow$ элементная модуляция.

Численный пример

$$x = [1 \quad -0.5], \quad W = \left[\begin{array}{cc|cc} 0.5 & 1.0 & -0.3 & 0.8 \\ 0.2 & -0.7 & 0.4 & -0.1 \end{array} \right], \quad b = [0.1 \quad 0.0 \quad | \quad -0.2 \quad 0.3]$$

$$h = xW + b = [0.5 \quad 1.35 \quad | \quad -0.7 \quad 1.15] \implies h_1 = \begin{bmatrix} 0.5 \\ 1.35 \end{bmatrix}, \quad h_2 = \begin{bmatrix} -0.7 \\ 1.15 \end{bmatrix}$$

$$\sigma(h_2) = \begin{bmatrix} \sigma(-0.7) \\ \sigma(1.15) \end{bmatrix} = \begin{bmatrix} 0.332 \\ 0.759 \end{bmatrix}$$

$$\text{GLU}(x) = \begin{bmatrix} 0.5 \\ 1.35 \end{bmatrix} \odot \begin{bmatrix} 0.332 \\ 0.759 \end{bmatrix} = \begin{bmatrix} 0.166 \\ 1.025 \end{bmatrix}$$

GLU — Gated Linear Unit

$$\text{GLU}(h) = h_1 \odot \sigma(h_2)$$

- Двухветочная конструкция с «гейтом»
- GPT-2, BERT (в FFN-блоках)
- $\sigma = \text{Sigmoid}$
- + 50% параметров по сравнению с ReLU-FFN

SwiGLU

$$\text{SwiGLU}(h) = h_1 \odot \text{SiLU}(h_2)$$

- GLU с SiLU вместо Sigmoid
- LLaMA 1/2/3, Mistral, Mixtral
- Компенсирует лишние параметры качеством

ССС		
Задача / архитектура	Рекомендация	Примеры
Трансформер (LLM)	SwiGLU, GeGLU	LLaMA, Mistral
Трансформер (BERT-стиль)	GELU	BERT, GPT-2
CNN (комп. зрение)	ReLU, SiLU	EfficientNet, ResNet
GAN	Leaky ReLU, Mish	StyleGAN
RNN / LSTM	Tanh, Sigmoid (gate)	LSTM, GRU
Бинарный классификатор	Sigmoid (выход)	—
Множественный классификатор	Softmax (выход)	—

Почему важна инициализация весов?

- Нейросеть обучается через градиентный спуск
- Качество сходимости сильно зависит от начальных весов
 - Слишком малые веса $\rightarrow$ градиенты затухают (vanishing gradient)
 - Слишком большие веса $\rightarrow$ градиенты взрываются (exploding gradient)
- Цель: сохранить дисперсию активаций и градиентов по слоям

Ключевая идея

Дисперсия сигнала (активаций) и градиентов должна оставаться $\mathcal{O}(1)$ при прохождении через каждый слой сети.

Математическая постановка задачи

Линейный слой: $\mathbf{h}^{(l)} = W^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$

Пусть входы $\mathbf{h}^{(l-1)}$ — независимые случайные величины с дисперсией $\text{Var}(\mathbf{h}^{(l-1)}) = \sigma_{\text{in}}^2$, а веса $W^{(l)}$ — независимые с дисперсией $\text{Var}(W^{(l)}) = \sigma_w^2$.

$$\text{Var}(\mathbf{h}^{(l)}) \approx n_{\text{in}} \cdot \sigma_w^2 \cdot \sigma_{\text{in}}^2 \quad (1)$$

Требование: $\text{Var}(\mathbf{h}^{(l)}) \approx \sigma_{\text{in}}^2$ для всех слоёв.

$$\boxed{\sigma_w^2 = \frac{1}{n_{\text{in}}}} \quad (2)$$

Проблема

Это решение справедливо для **линейных** функций активации. Реальные нелинейности (ReLU, tanh) меняют картину.

Равномерное распределение:

$$W \sim \mathcal{U}(-a, a), \quad a = \sqrt{\frac{3}{n_{\text{in}}}}$$

Нормальное распределение:

$$W \sim \mathcal{N}\left(0, \frac{1}{n_{\text{in}}}\right)$$

Недостатки

- Учитывают только размер входов
- Не учитывают размер выходов
- Не учитывают тип нелинейности
- Для глубоких сетей — затухание/взрыв

Инициализация Ксавье (Xavier / Glorot) (2010)

Мотивация

Учёт и входов, и выходов слоя: $\sigma_w^2 = \frac{2}{n_{in} + n_{out}}$

Формулировки:

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{in} + n_{out}}}, \sqrt{\frac{6}{n_{in} + n_{out}}}\right), \quad W \sim \mathcal{N}\left(0, \frac{2}{n_{in} + n_{out}}\right) \quad (3)$$

Где применять

Слои с симметричными нелинейностями: tanh, sigmoid, elu

Ограничение

Не учитывает, что ReLU обнуляет половину нейронов $\Rightarrow$ дисперсия уменьшается вдвое на каждом слое.

Геометрическая интерпретация

- Вектор весов $\mathbf{w}$ для одного нейрона — случайный вектор в $\mathbb{R}^{n_{\text{in}}}$
- При $n_{\text{in}} \rightarrow \infty$ направление $\mathbf{w}$ становится изотропным
- Дисперсия скалярного произведения $\mathbf{w} \cdot \mathbf{x}$ определяется произведением дисперсий:

$$\text{Var}(\mathbf{w} \cdot \mathbf{x}) = n_{\text{in}} \cdot \sigma_w^2 \cdot \sigma_x^2$$

- Ксавье выбирает σ_w^2 так, чтобы эта дисперсия не менялась при переходе слоёв

Критерий	Формула
Forward pass	$\sigma_w^2 = \frac{1}{n_{\text{in}}}$
Backward pass	$\sigma_w^2 = \frac{1}{n_{\text{out}}}$
Ксавье	$\sigma_w^2 = \frac{2}{n_{\text{in}} + n_{\text{out}}}$

Инициализация Хе (He / Kaiming) (2015)

Ключевая наблюдение

ReLU обнуляет $\approx 50\%$ нейронов $\Rightarrow$ эффективная дисперсия активаций падает в 2 раза на каждом слое.

Коррекция:

$$\sigma_w^2 = \frac{2}{n_{\text{in}}} \quad (\text{вместо } \frac{1}{n_{\text{in}}}) \quad (4)$$

Формулировки:

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{\text{in}}}}, \sqrt{\frac{6}{n_{\text{in}}}}\right) \quad W \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right) \quad (5)$$

Инициализация Хе (He / Kaiming) (2015)

Метод	Сигма	Для нелинейности
Классическая	$\frac{1}{n_{in}}$	tanh, sigmoid
He (fan-in)	$\frac{2}{n_{in}}$	ReLU, Leaky ReLU
He (fan-out)	$\frac{2}{n_{out}}$	обратный проход

В PyTorch

```
torch.nn.init.kaiming_uniform_ или kaiming_normal_  
torch.nn.init.kaiming_normal_(tensor, mode='fan_in', nonlinearity='relu')
```

Ортогональная инициализация

- Выбираем W так, что $W^T W = I$ (или близкое)
- Векторы весов взаимно ортогональны и единичной длины
- Свойство: $\|Wx\|_2 = \|x\|_2$ — норма сохраняется!

Алгоритм (QR-разложение):

- 1 Генерируем матрицу A из $\mathcal{N}(0, 1)$ размером $n_{\text{out}} \times n_{\text{in}}$
- 2 $A = QR \Rightarrow Q$ — ортогональная (если $n_{\text{out}} \leq n_{\text{in}}$)
- 3 Ставим $W = Q$ (или $W = Q^T$, если $n_{\text{out}} > n_{\text{in}}$)

Преимущества

- Отлично для RNN и GRU (стабильность во времени)
- Для CNN — ортогональность по каналам (свертка с ортогональным ядром)

```
torch.nn.init.orthogonal_(weight)
```

Сравнение методов

Метод	Сигма	Нелинейность	RNN	CNN	MLP
Равномерная	$\sqrt{1/n_{in}}$	✓	×	✓	✓
Ксавье	$\sqrt{2/(n_{in} + n_{out})}$	tanh, sig	✓	✓	✓
He	$\sqrt{2/n_{in}}$	ReLU	✓	✓	✓
Ортогональная	$\ W\ = 1$	любая	✓	✓	✓

Практические рекомендации:

- ReLU-сети → **Хе**
- tanh/sigmoid → **Ксавье**
- RNN/LSTM → **Ортогональная**
- BatchNorm → любой (BN нормализует)

Современные подходы:

- **Orthogonal + He scale** (LSTM)
- **Glorot orthogonal** (Transformer)
- **Xavier normal + $\sqrt{2/\text{depth}}$** (DeepNet)
- **Truncated normal** (BERT, T5)

Stochastic Gradient Descent (SGD)

Правило обновления

$$W_{t+1} = W_t - \eta \nabla_W L(W_t)$$

где η — learning rate (скорость обучения).

Идеи:

- Двигаемся в направлении отрицательного градиента
- η контролирует размер шага
- **Стохастический** вариант: используем градиент на мини-батче

Проблемы:

- Один η на все параметры $\rightarrow$ плохо для разнородных градиентов
- Осцилляции в узких долинах
- Медленная сходимость в «плоских» регионах
- Чувствителен к выбору η

SGD с инерцией (Momentum) (Polyak, 1964)

Правило обновления (двухстрочная форма):

$$v_{t+1} = \beta v_t + \nabla_W L_t \quad (6)$$

$$W_{t+1} = W_t - \eta v_{t+1} \quad (7)$$

где:

- v_t — вектор «скорости» (инерция)
- $\beta \in [0, 1)$ — коэффициент затухания (обычно 0.9)

$$v_{t+1} = \nabla L_t + \beta \nabla L_{t-1} + \beta^2 \nabla L_{t-2} + \dots$$

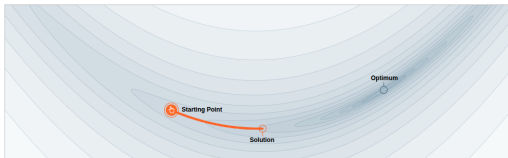
Без momentum (SGD):

- Шар без массы — мгновенно останавливается
- Каждый градиент — новый импульс с нуля
- В узкой долине — зигзаги

С momentum:

- Шар имеет массу (инерцию)
- Накопленная инерция несёт сквозь шум
- В узкой долине — инерция гасит поперечные колебания

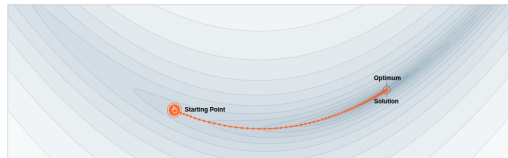
<https://distill.pub/2017/momentum/>



Step-size $\alpha \approx 0.0020$

Momentum $\beta = 0.0$

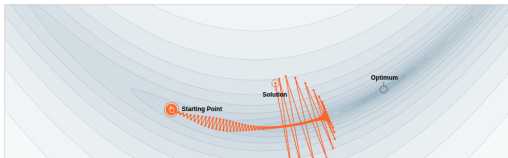
We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?



Step-size $\alpha \approx 0.0020$

Momentum $\beta = 0.90$

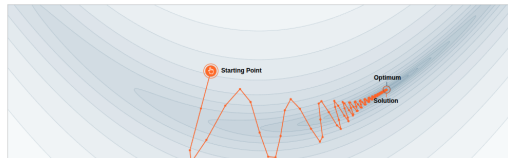
We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?



Step-size $\alpha \approx 0.0053$

Momentum $\beta = 0.0$

We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?



Step-size $\alpha \approx 0.0020$

Momentum $\beta = 0.90$

We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

Минутка кода

```
optimizer = torch.optim.SGD(  
    model.parameters(),  
    lr=1e-2,          # \eta  
    momentum=0.9,     # \beta  
    dampening=0,      # 0  
    weight_decay=0,   # \lambda  
    nesterov=False    # True = Nesterov momentum  
)
```

Идея

Адаптивно уменьшать η для координат с **большими** накопленными градиентами.

Правило обновления:

$$G_t = G_{t-1} + (\nabla_W L_t)^2 \quad (\text{накопленный квадрат градиента}) \quad (8)$$

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{G_t} + \epsilon} \odot \nabla_W L_t \quad (9)$$

где $\epsilon > 0$ — малая константа для числовой стабильности.

Преимущества:

- Не нужно подбирать η вручную
- Хорошо работает с разреженными градиентами (NLP)

Недостатки:

- G_t монотонно растёт $\Rightarrow$ эффективный $\eta_t \rightarrow 0$
- В конце обучения шаг слишком мал $\Rightarrow$ застревание

Минутка кода

```
optimizer = torch.optim.Adagrad(  
    model.parameters(),  
    lr=1e-2,          # \eta  
    lr_decay=0,  
    weight_decay=0, # \lambda  
    initial_accumulator_value=0  
)
```

Идея

Комбинация **Momentum** (сглаживание направления) + **Adagrad** (адаптивный шаг).

Правило обновления:

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1) \nabla_W L_t \quad \text{— 1-й момент (среднее)} \quad (10)$$

$$u_t = \beta_2 u_{t-1} + (1 - \beta_2) (\nabla_W L_t)^2 \quad \text{— 2-й момент (дисперсия)} \quad (11)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_1^t}, \quad \hat{u}_t = \frac{u_t}{1 - \beta_2^t} \quad \text{— bias correction} \quad (12)$$

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{u}_t} + \epsilon} \odot \hat{v}_t \quad (13)$$

Adam: интуиция

- 1 1-й момент v_t : сглаживает шум градиента $\rightarrow$ устойчивое направление (аналог momentum).
- 2 2-й момент u_t : координаты с большой дисперсией градиентов получают **меньший** шаг.
- 3 Bias correction: при $t \gg 1$ оценки v_t, u_t смещены к нулю $\rightarrow$ делим на $(1 - \beta^t)$.

	SGD + Momentum	Adagrad	Adam
Momentum (1-й момент)	✓	—	✓
Адаптивный шаг (2-й момент)	—	✓	✓
Bias correction	—	—	✓
Гиперпараметры	η, β	η	η, β_1, β_2

Задание (1 балл)

Можно ли подобрать β_1 и β_2 , чтобы получить классический SGD, SGD с инерцией, Adagrad? Баллы получают первые 5 студентов.

Weight Decay и L2-регуляризация

Weight Decay (в правиле обновления)

Добавляем к градиенту саму матрицу параметров:

$$\nabla_W L_t \longrightarrow \nabla_W L_t + \lambda W_t$$

где $\lambda > 0$ — коэффициент weight decay.

L2-регуляризация (в функции потерь)

$$\tilde{L}(W) = L(W) + \frac{\lambda}{2} \|W\|_F^2$$

$$\nabla_W \tilde{L} = \nabla_W L + \lambda W$$

Weight Decay и L2-регуляризация

При SGD (без momentum) это одна и та же вещь

$$\text{SGD + weight decay: } W_{t+1} = W_t - \eta(\nabla_W L_t + \lambda W_t)$$

$$\text{SGD + L2: } W_{t+1} = W_t - \eta(\nabla_W \tilde{L}_t)$$

Тот же результат.

Но при адаптивных оптимизаторах (Adam) эквивалентность нарушается!

L2-регуляризация проходит **через** адаптивный шаг:

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{u}_t} + \epsilon} \odot \hat{v}_t \quad \text{где} \quad \hat{v}_t \text{ содержит } \lambda W_t$$

⇒ регуляризатор λW_t делится на $\sqrt{\hat{u}_t}$, т. е. его влияние зависит от истории градиентов и **различно** для разных координат.

AdamW — Decoupled Weight Decay (Loshchilov & Hutter, 2019)

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1) \nabla_W L_t \quad (14)$$

$$u_t = \beta_2 u_{t-1} + (1 - \beta_2) (\nabla_W L_t)^2 \quad (15)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_1^t}, \quad \hat{u}_t = \frac{u_t}{1 - \beta_2^t} \quad (16)$$

$$W_{t+1} = W_t \underbrace{- \frac{\eta}{\sqrt{\hat{u}_t} + \epsilon} \odot \hat{v}_t}_{\text{адаптивный шаг (без } \lambda W_t)} \underbrace{- \eta \lambda W_t}_{\text{weight decay (отделён)}} \quad (17)$$

Минутка кода

```
optimizer = torch.optim.Adam(  
    model.parameters(),  
    lr=1e-3,                # \eta  
    betas=(0.9, 0.999),    # \beta_1, \beta_2  
    eps=1e-8               # \epsilon for zero division  
)  
  
optimizer = torch.optim.AdamW(  
    model.parameters(),  
    lr=1e-3,                # \eta  
    betas=(0.9, 0.999),    # \beta_1, \beta_2  
    eps=1e-8,              # \epsilon for zero division  
    weight_decay=0.01      # \lambda  
)
```

Dropout (Srivastava et al., 2014)

Определение

На каждом forward pass с вероятностью p **отключаем** нейрон (обнуляем его активацию).

Training:

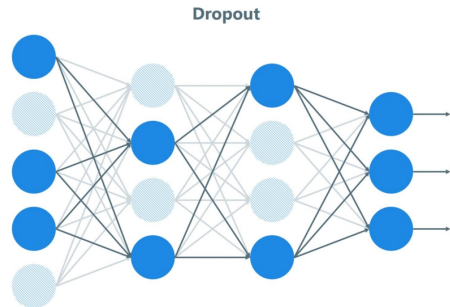
$$m_i \sim \text{Bernoulli}(1 - p), \quad i = 1, \dots, n \quad (18)$$

$$\tilde{h}_i = m_i \odot h_i \quad (19)$$

Inference:

$$\tilde{h}_i = (1 - p)h_i \quad (20)$$

p — dropout rate (обычно 0.1–0.5)



Inverted Dropout (Srivastava et al., 2014)

Training (inverted dropout):

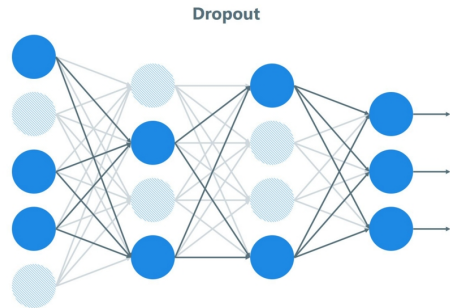
$$m_i \sim \text{Bernoulli}(1 - p), \quad i = 1, \dots, n \quad (21)$$

$$\tilde{h}_i = \frac{m_i}{1 - p} h_i \quad (22)$$

Inference:

$$\tilde{h}_i = h_i \quad (23)$$

Масштаб $\frac{1}{1-p}$ — **inverted dropout**, на inference ничего менять не нужно.



Почему Dropout работает?

- 1 Предотвращение co-adaptation
- 2 Ensemble interpretation
- 3 Регуляризация через эффективную глубину

$$\mathbb{E}[\|\tilde{\mathbf{h}}\|_2^2] = (1-p) \|\mathbf{h}\|_2^2 \implies \text{«размазывает» активации, ограничивает их величину.}$$

Аналог L2-регуляризации, но **структурный** (разрывает зависимости).

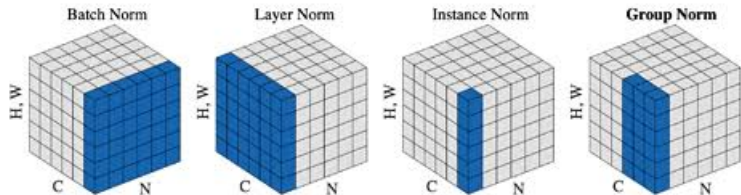
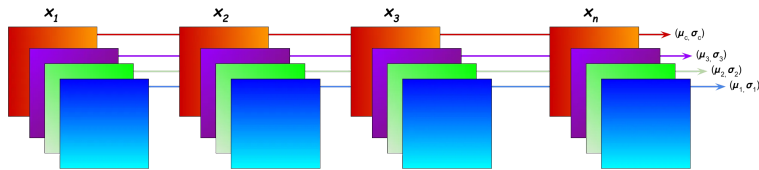


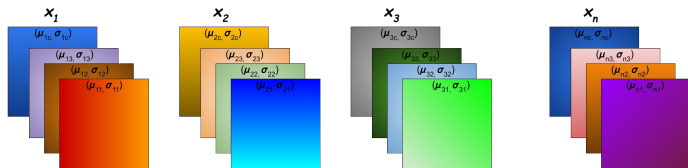
Figure 2. **Normalization methods.** Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Batch Normalization



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

Instance Normalization

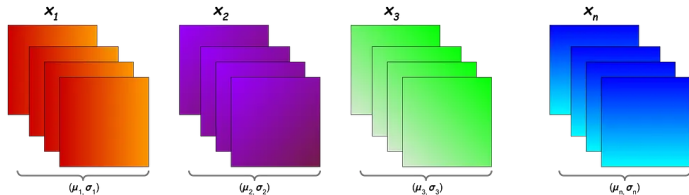


$$\mu_{nc} = \frac{1}{HW} \sum_{j=1}^H \sum_{k=1}^W x_{ncjk}$$

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

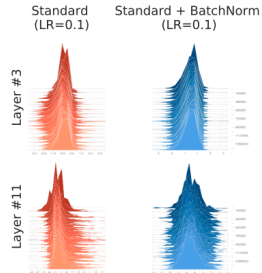
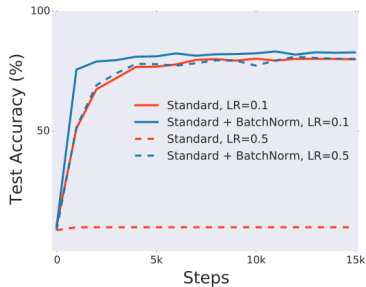
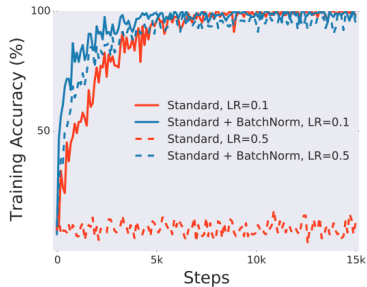
$$\hat{x} = \frac{x - \mu_{nc}}{\sqrt{\sigma_{nc}^2 + \epsilon}}$$

Layer Normalization



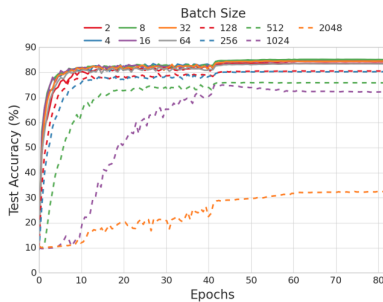
$$\mu_n = \frac{1}{CHW} \sum_{i=1}^C \sum_{j=1}^H \sum_{k=1}^W x_{nijk}$$
$$\sigma_n^2 = \frac{1}{CHW} \sum_{i=1}^C \sum_{j=1}^H \sum_{k=1}^W (x_{nijk} - \mu_n)^2$$
$$\hat{x} = \frac{x - \mu_n}{\sqrt{\sigma_n^2 + \epsilon}}$$

Работает ли это?



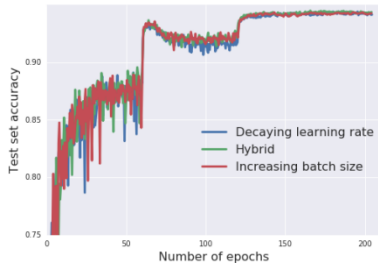
<https://arxiv.org/pdf/1805.11604>

Еще пару слов про Batch Size



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

Еще пару слов про Batch Size



(a)



(b)

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