# Основы практического использования нейронных сетей.

## Лекция 4. Методы повышения эффективности алгоритмов обучения для глубоких НС.

Дмитрий Буряк.
к.ф.-м.н.
dyb04@yandex.ru

# Пример НС для распознавания изображений

- Распознавание изображений кошек и собак (база cats vs dogs)
  - RGB, различное разрешение, 25000 изображений.

The model expects RGB images of size 180 × 180.

```
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Rescale inputs to the [0, 1] range by dividing them by 255.

- 991К параметров

# Подготовка обучающих и тестовых данных

- Использование объекта Dataset.

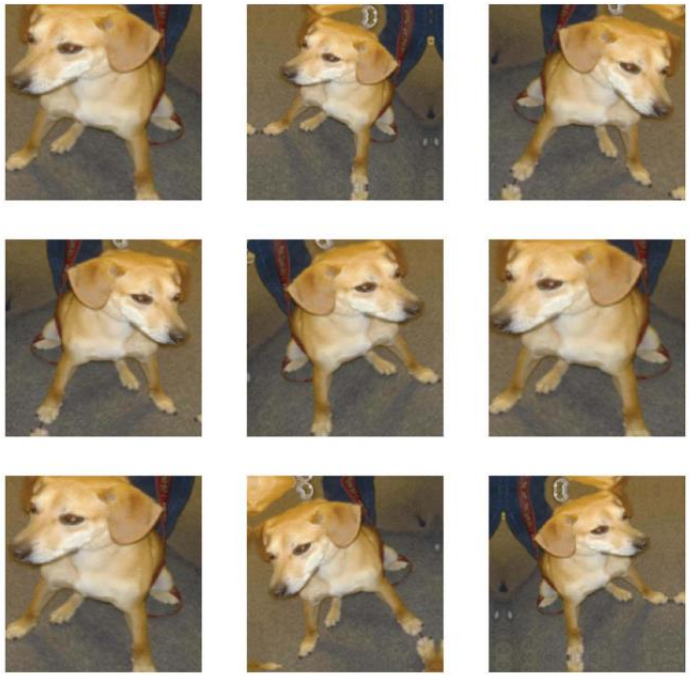
```
from tensorflow.keras.utils import image_dataset_from_directory

train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

```
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

# Аугментация данных

- Формирование синтетических данных для повышения разнообразия данных при обучении.
  - Необходимо учитывать, какие реальные искажения данных возможны/допустимы

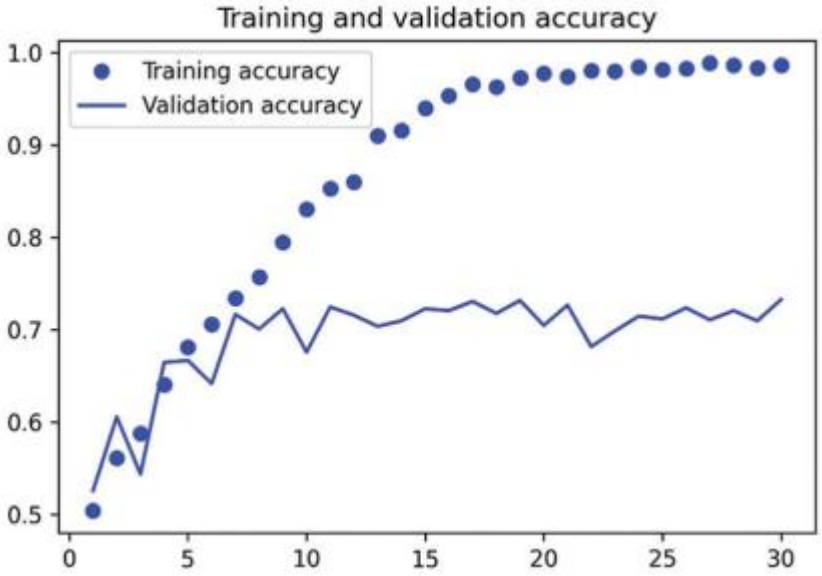
```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```



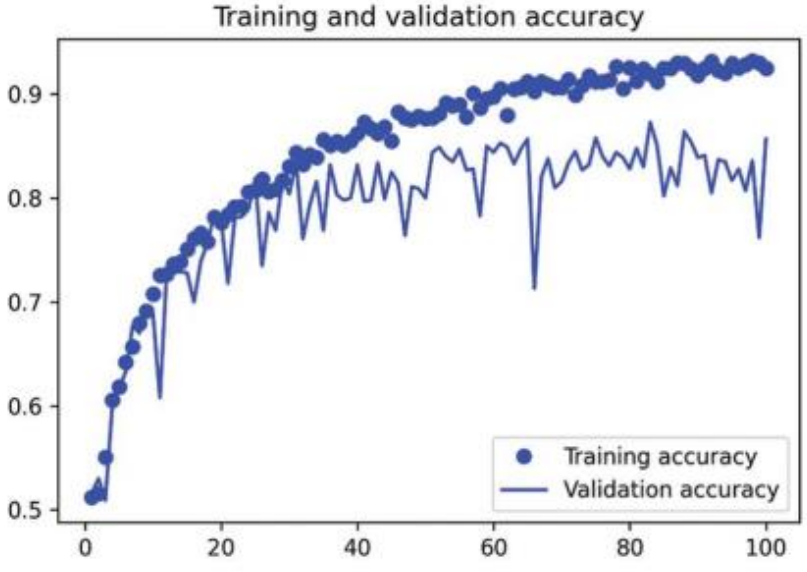
# НС с аугментацией данных

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
```

# Влияние аугментации на точность на тестовых данных




Без аугментации




С аугментацией

# Регуляризация L2

- Регуляризация - метод предотвращения переобучения НС.
- Введение штрафа для больших весов.

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

- λ - коэффициент регуляризации.

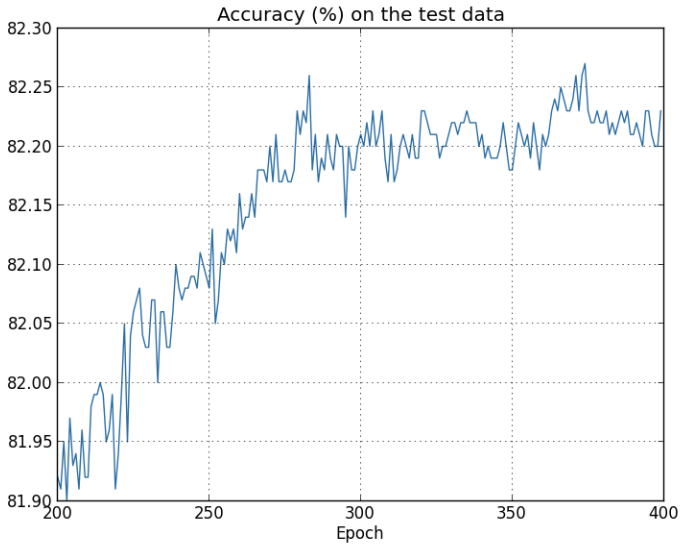
$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w$$
$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b}.$$

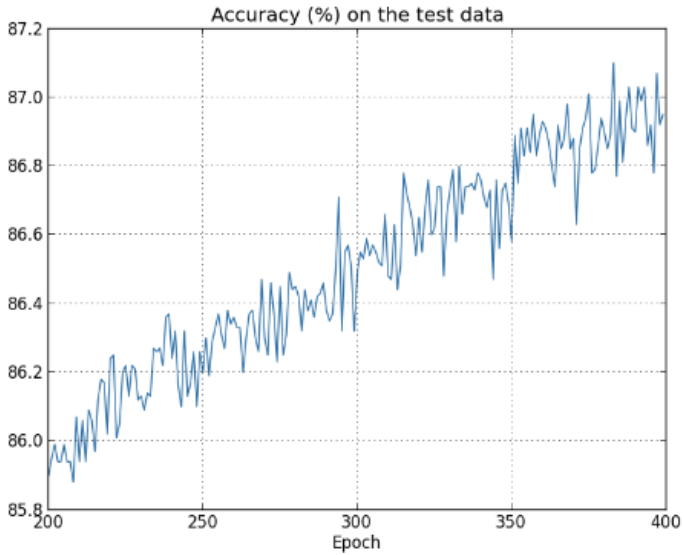
$$b \to b - \eta \frac{\partial C_0}{\partial b}$$
$$w \to w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta\lambda}{n} w$$
$$= \left(1 - \frac{\eta\lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}$$

- Масштабирование веса перед коррекцией по градиентному спуску.

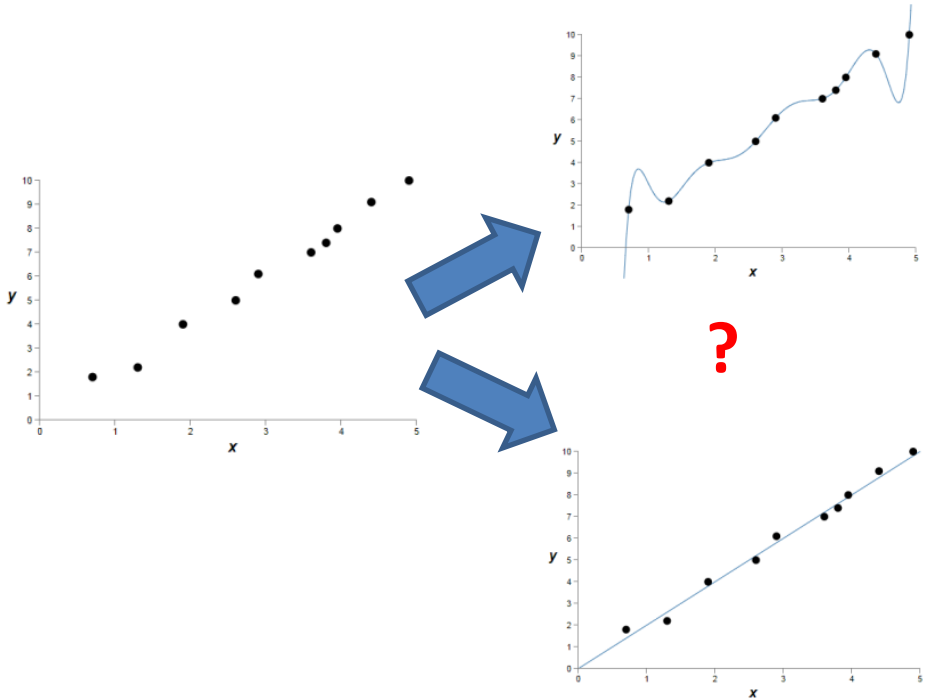
# Пример применения регуляризации L2

- Классификация MNIST
- Сеть 784x30x10, 1000 обучающих примеров



# Регуляризация → снижение переобучения




- Нет однозначного решения без дополнительной инфоормации.
- Большие значения параметров → увеличение чувствительности к шуму.

$$y = a_0 x^9 + a_1 x^8 + \dots$$

$$y = a_0 x + a_1$$

# Регуляризация L1

- Введение штрафа для больших весов.

$$C = C_0 + \frac{\lambda}{n} \sum_w |w|$$

- λ - коэффициент регуляризации.

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} \operatorname{sgn}(w) \qquad w \to w' = w - \frac{\eta\lambda}{n} \operatorname{sgn}(w) - \eta \frac{\partial C_0}{\partial w}$$

- Уменьшение веса на фиксированную величину

- Для регуляризации L2 значение уменьшения веса зависит от его величины.

$$w \to w' = w \left(1 - \frac{\eta\lambda}{n}\right) - \eta \frac{\partial C_0}{\partial w}$$

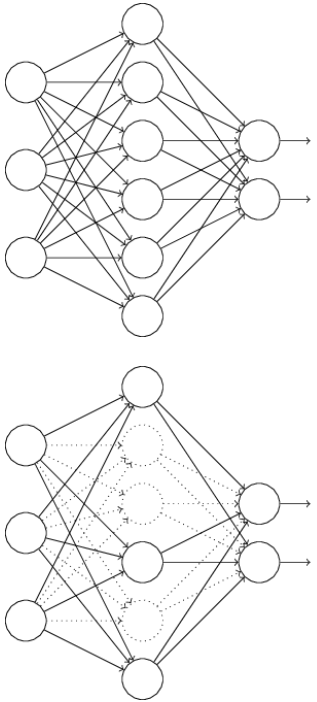
# Регуляризация в Keras

- keras.layers.Dense( units, …, **kernel_regularizer=None, bias_regularizer=None**, ….)
- keras.layers.Conv2D( filters, kernel_size, …, **kernel_regularizer=None, bias_regularizer=None**, … )

- Доступные регуляризаторы:
  - keras.regularizers.L1(l1=0.01)
  - keras.regularizers.L2(l2=0.01)
  - keras.regularizers.L1L2(l1=0.01, l2=0.01)
  - keras.regularizers.OrthogonalRegularizer(factor=0.01, mode="rows")
  - реализация собственных регуляризаторов

# Dropout

- Инструмент регуляризации.
- Модификация архитектуры сети в процессе обучения.
- Упрощенная схема Dropout
    1. Временно удалить из НС половину случайно выбранных внутренних нейронов с соответствующими связями.
    2. Провести итерацию обучение на пакете: обновление связей оставшихся нейронов.
    3. Восстановить удаленные нейроны и их связи.
    4. Повторить п. 1 – 3.
- Перед применением сети уменьшить внутренние веса в 2 раза.



N. Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting, 2014.

# Dropout. Пример использования.

- MNIST.

Входной вектор 784 элемента.

10 классов.

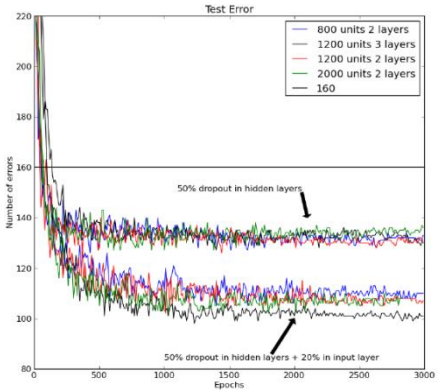
10000 тестовых изображений.

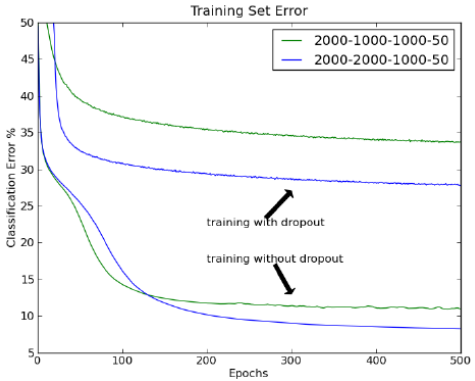
- Reuters. Классификация документов.

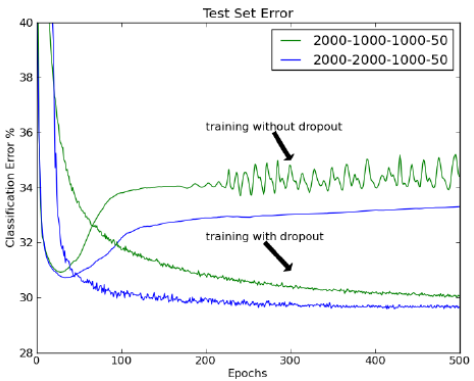
Входной вектор 2000 элементов.

50 классов.

~200000 тестовых документов.



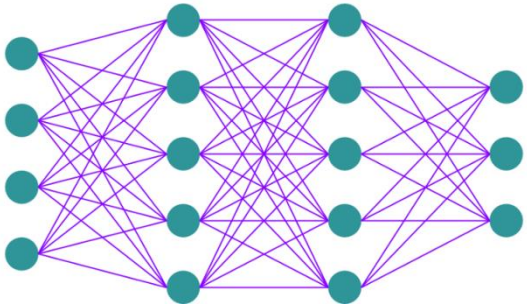



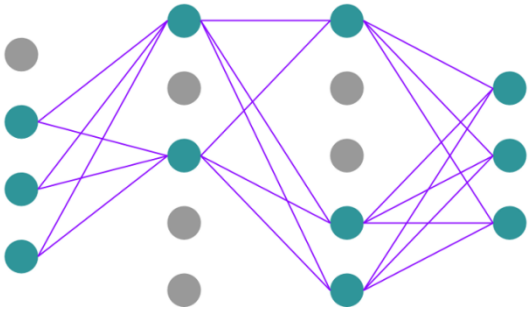




G.E. Hinton et al, Improving neural networks by preventing co-adaptation of feature detectors, 2012

# Варианты Dropout. DropConnect.

- Удаление связей.



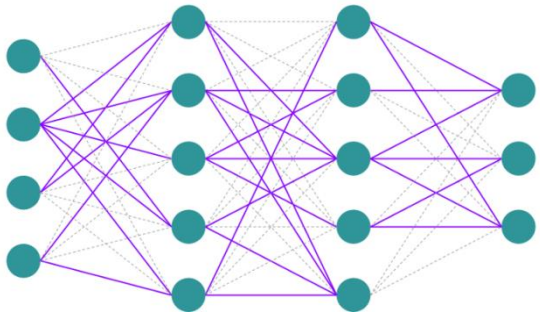




Training Phase :

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \circ \mathbf{m}, \quad m_i \sim Bernoulli(p)$$

Testing Phase :

$$\mathbf{y} = (1-p)f(\mathbf{W}\mathbf{x})$$




Training Phase :

$$\mathbf{y} = f((\mathbf{W} \circ \mathbf{M})\mathbf{x}), \quad M_{i,j} \sim Bernoulli(p)$$

Testing Phase :

$$\mathbf{y} = (\mathbf{W}\mathbf{x}) \circ \hat{\mathbf{m}}(\mathbf{Z})$$

$$\text{where } \hat{m}_i(Z) = \frac{1}{Z}\sum_{z=0}^{Z} f(\hat{x}_{i,z}), \quad \hat{x}_{i,z} \sim \mathcal{N}(\mu_i, \sigma_i^2)$$

$$\text{and } \boldsymbol{\mu} = p\mathbf{W}\mathbf{x}, \quad \boldsymbol{\sigma}^2 = p(1-p)(\mathbf{W} \circ \mathbf{W})(\mathbf{x} \circ \mathbf{x}), \quad Z \in \mathbb{N}^+$$

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

# Варианты Dropout. Standout.

- Вероятность удаления нейрона зависит от величины весов.

- Пример.

Training Phase :

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \circ \mathbf{m}, \quad m_i \sim Bernoulli(g(\mathbf{W_s}\mathbf{x}))$$

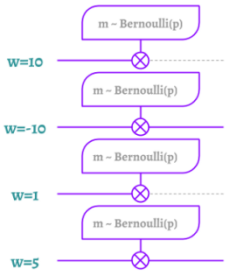
Testing Phase :

$$\mathbf{y} = (1 - g(\mathbf{W_s}\mathbf{x})) \circ f(\mathbf{W}\mathbf{x})$$

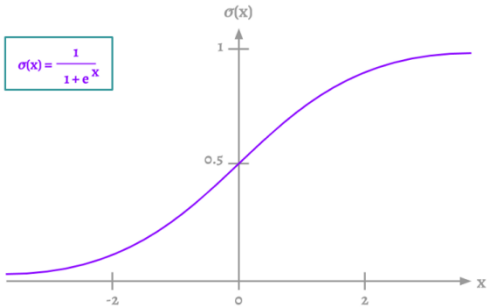
where $\mathbf{W_s}$ is the belief network's weights
and $g$ is the belief network's activation function

$$g(x) = |\sigma(x)|$$

$$\mathbf{W_s} = \alpha\mathbf{W} + \beta$$








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

# Варианты Dropout. Gaussian Dropout.

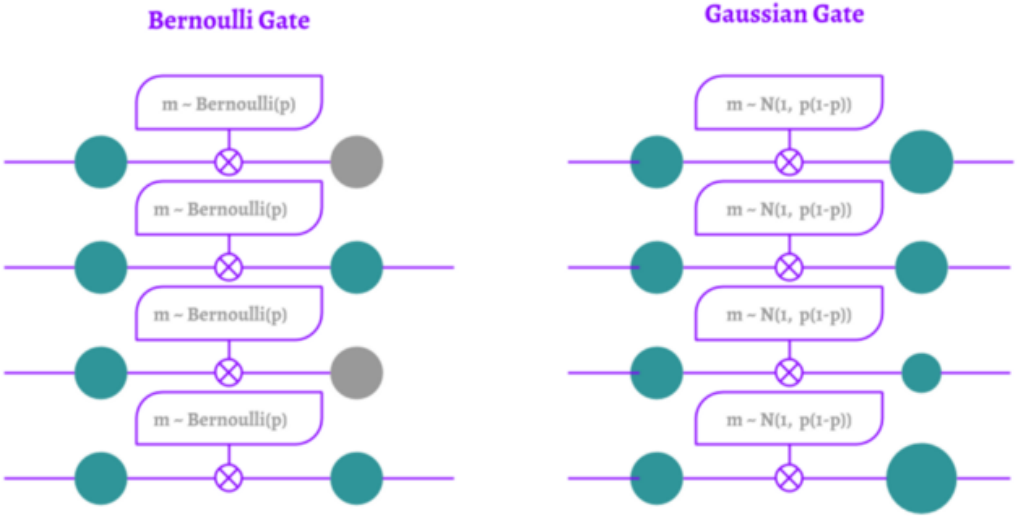
- ❑ Нейроны не удаляются, а «завешиваются» с помощью нормального распределения.
- ❑ Выше скорость сходимости.

Training Phase :

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \circ \mathbf{m}, \quad m_i \sim \mathcal{N}(1,\ p(1-p))$$

Testing Phase :

$$\mathbf{y} = f(\mathbf{W}\mathbf{x})$$




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

# Варианты Dropout. Pooling Dropout.

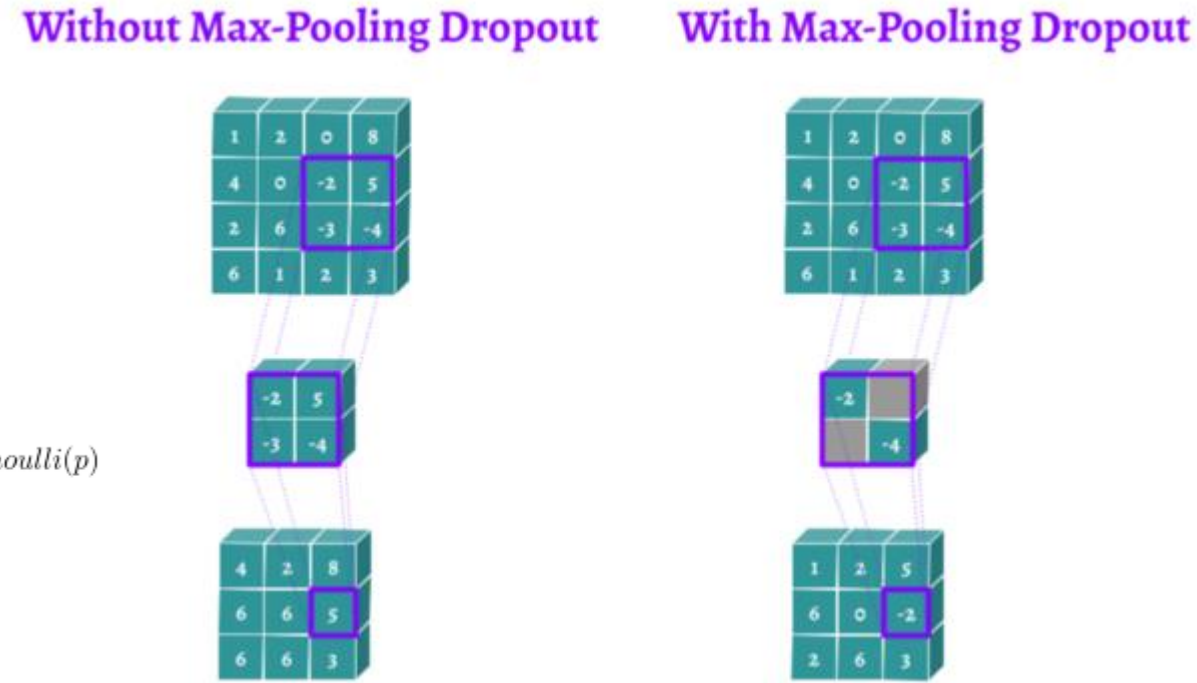
- ❑ Стандартный dropout не эффективен для изображений
- ❑ Pooling Dropout применяют для сверточных сетей

Training Phase :

$$\mathbf{Y} = max\{Pool_{size}(\mathbf{Y}) \circ \mathbf{M_{size}}\} \quad M_{ij} \sim Bernoulli(p)$$

Testing Phase :

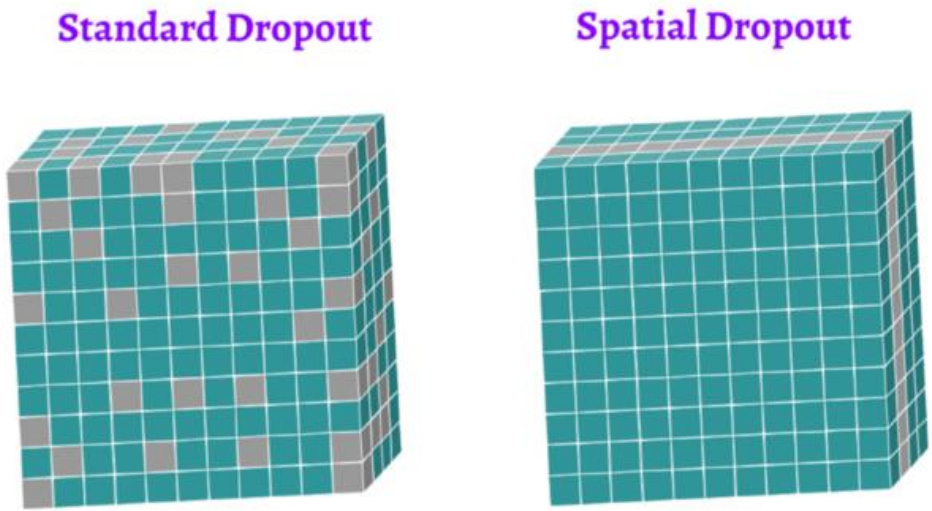
$$\mathbf{Y} = (1-p)\ max\{Pool_{size}(\mathbf{Y})\}$$




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

# Варианты Dropout. Spatial Dropout.

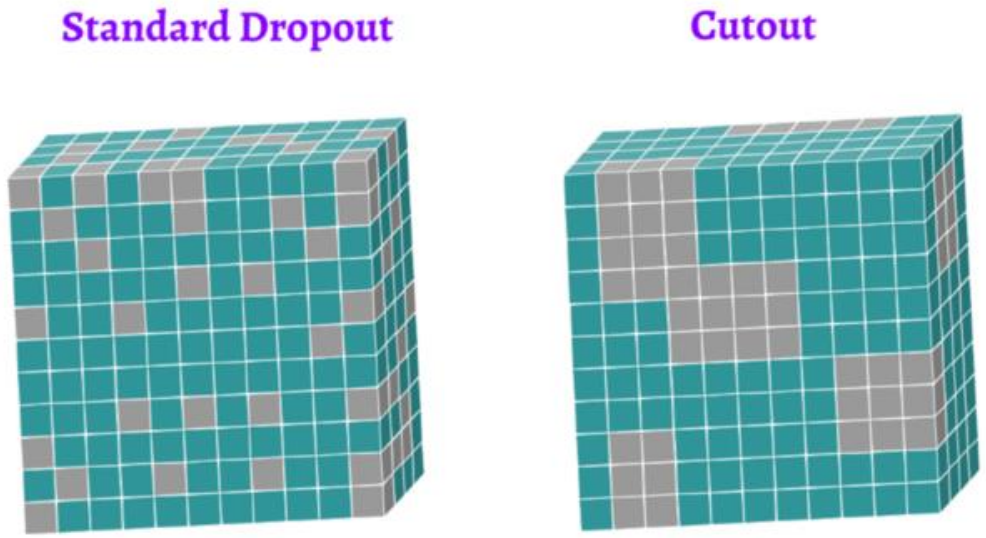
- ❑ Удаление карт признаков
- ❑ Spatial Dropout применяют для сверточных сетей




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

# Варианты Dropout. Cutout.

- Удаление фрагментов карт
- Cutout применяют для сверточных сетей




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

# Dropout в Keras

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

- ❑ Dropout
- ❑ SpatialDropout1D
- ❑ SpatialDropout2D
- ❑ SpatialDropout3D
- ❑ GaussianDropout
- ❑ AlphaDropout

# Batch normalization

- Нормализация + декорреляция входных данных → повышение эффективности обучения.
- Нарушение полученных свойств входных векторов для промежуточных данных во внутренних слоях (internal covariance shift).
- Идея: проводить предобработку входных данных для каждого внутреннего слоя.
- Оптимизация вычислительных затрат → нормализация внутренних данных (без декорреляции).
- **Эффект от Batch normalization:**
  - **влияние на распределение данных (контроль internal covariance shift)**
  - **стимуляция градиента.**

Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015

# Batch normalization. Алгоритм

**Input:** Network $N$ with trainable parameters $\Theta$;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$
**Output:** Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \dots K$ **do**
3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \dots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size $m$, and average over them:

$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$
12: **end for**

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: $\gamma$, $\beta$
**Output:** $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

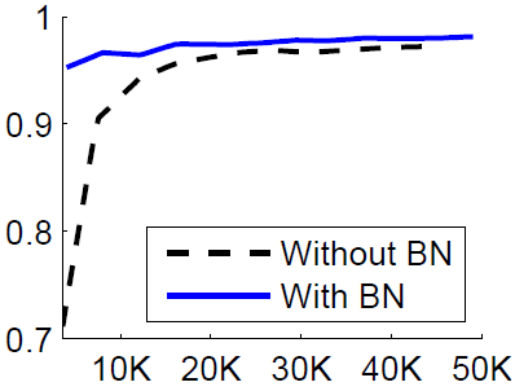
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Реализовано через экспоненциальное скользящее среднее, вычисляемое на этапе обучения

# Batch normalization. Примеры.

- Классификация MNIST
- Сеть 784x100x100x100x10,

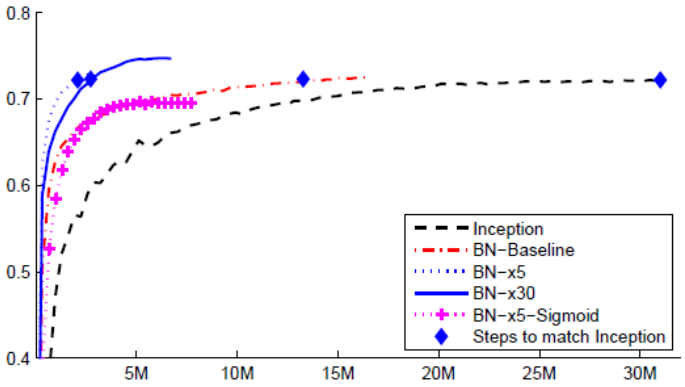
50000 обучающих примеров




- Классификация ImageNet
- Сеть $13.6*10^6$ параметров,

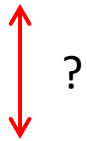
1000 классов

| Model | Steps to 72.2% | Max accuracy |
|---|---|---|
| Inception | $31.0 \cdot 10^6$ | 72.2% |
| *BN-Baseline* | $13.3 \cdot 10^6$ | 72.7% |
| *BN-x5* | $2.1 \cdot 10^6$ | 73.0% |
| *BN-x30* | $2.7 \cdot 10^6$ | 74.8% |
| *BN-x5-Sigmoid* | | 69.8% |



# Реализация Batch normalization

```
x = layers.Conv2D(32, 3, activation="relu")(x)
x = layers.BatchNormalization()(x)
```



?

```
x = layers.Conv2D(32, 3, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
```

# Реализация Batch normalization

```
x = layers.Conv2D(32, 3, activation="relu")(x)
x = layers.BatchNormalization()(x)
```

```
x = layers.Conv2D(32, 3, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
```

# Вопросы

- ❑ Для чего нужна аугментация данных?
- ❑ Какие типы Dropout применяют для сверточных сетей?
- ❑ Почему Batch Normalization повышает эффективность обучения?