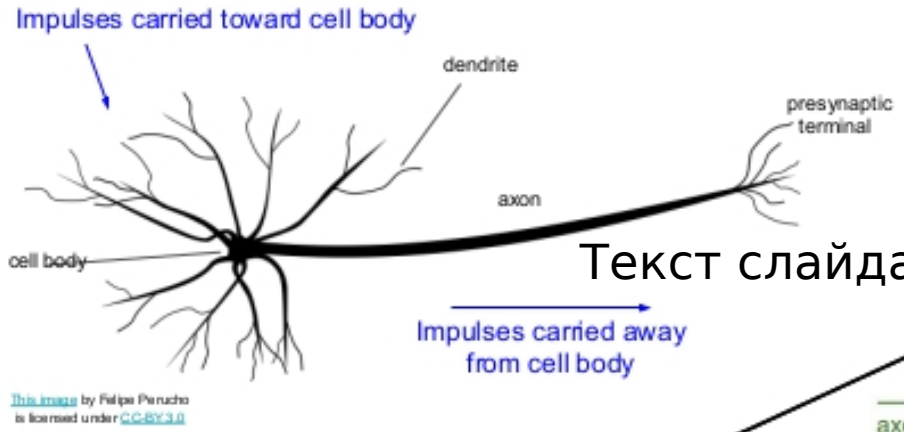
Современные НС технологии. Сверточные сети

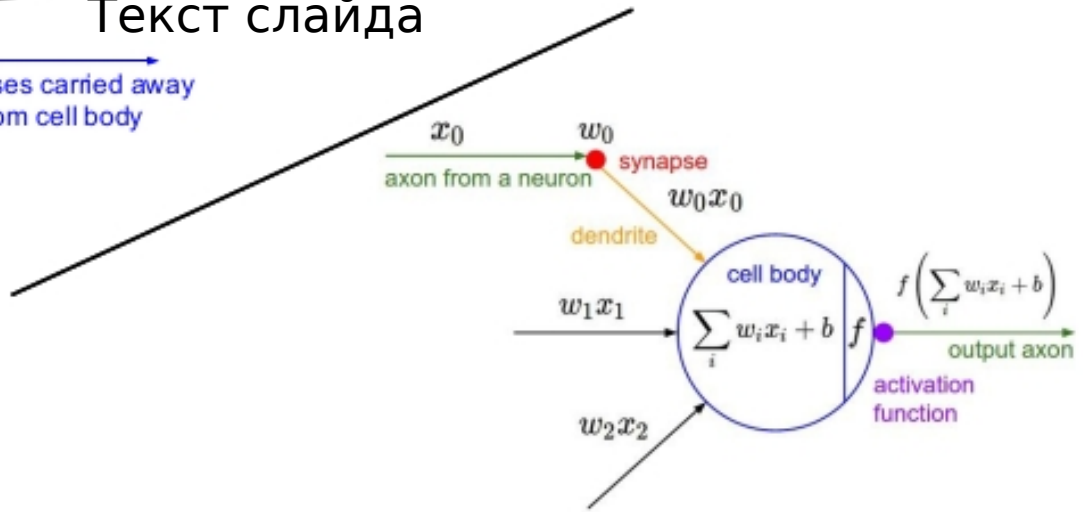
МАЛЕВА Т.В. ЛЕКЦИЯ 2

Что было

1. Искусственный нейрон - модель биологического нейрона

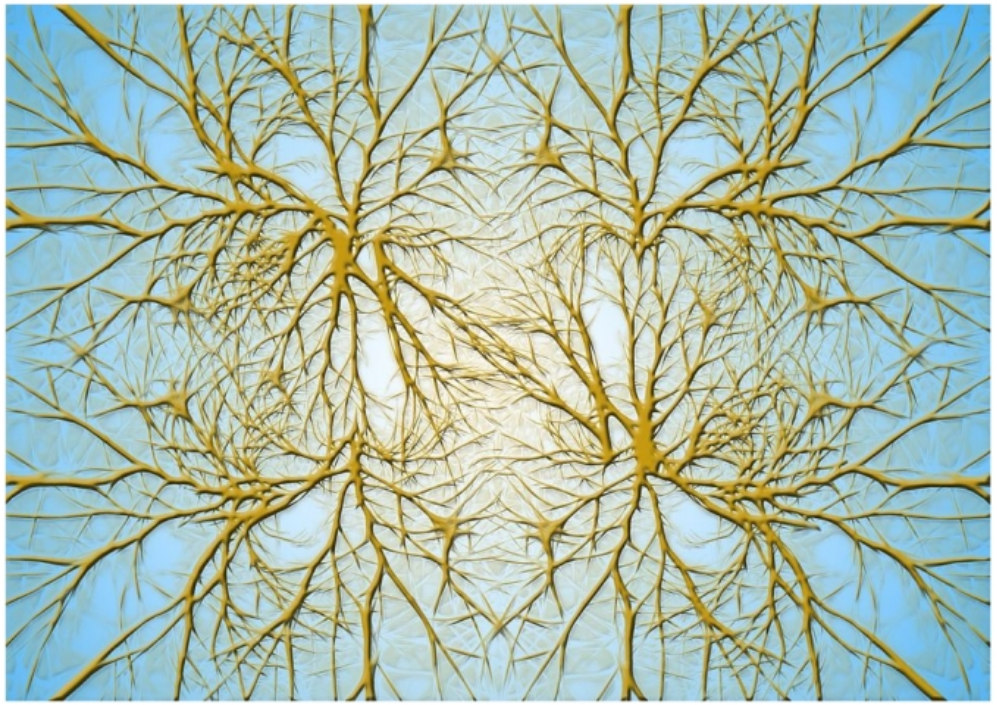


Текст слайда



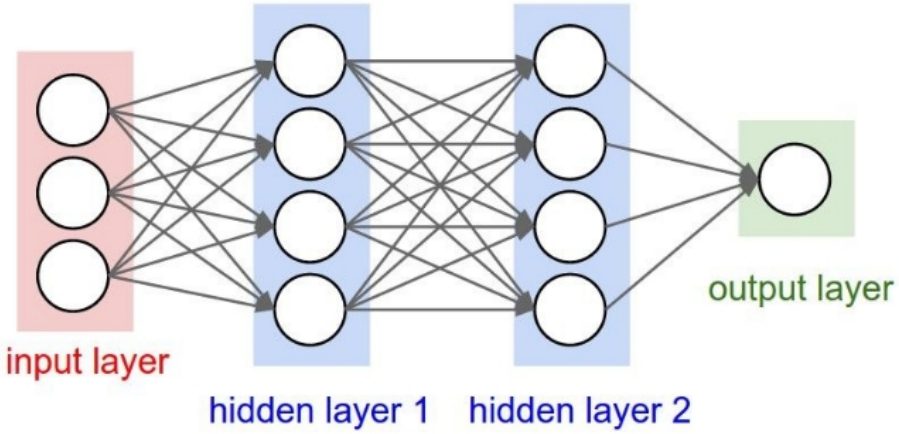
CS231_2019

Многослойный персептрон - имитация нейронной сети



This image is CC0 Public Domain

Трехслойный персептрон (2 скрытых+1 выходной, входной слой - не считаем)



Нейронная сеть

- Функция активации слоя
- Функция потерь (Loss function) — измеряет точность модели во время обучения. Необходимо минимизировать эту функцию чтоб "направить" модель в верном направлении.
- Оптимизатор (Optimizer) — показывает каким образом обновляется модель на основе входных данных и функции потерь (алгоритм обучения).
- Метрики (Metrics) — используются для мониторинга тренировки и тестирования модели.

Перекрестная энтропия (categorical_crossentropy, binary_crossentropy)

$$CCE(p,t) = -\sum_{c=1}^{C} t_{o,c} \log\left(p_{o,c}\right)$$

$$BCE\ (t,p) = -(t * \log(p) + (1-t) * \log\left(1-p\right))$$

T — эталонное значение, p — выход нейрона после активации

Классическое определение перекрестной энтропии

$$\mathrm{H}(p, q) \stackrel{\mathrm{df}}{=} \mathrm{E}_p[-\log q] = \mathrm{H}(p) + D_{\mathrm{KL}}(p \| q),$$

где $H(p)$ — энтропия p, и $D_{\mathrm{KL}}(p||q)$ — расстояние Кульбака—Лейблера от p до q (также известная как *относительная энтропия*).

Для дискретного p и q это означает

$$\mathrm{H}(p, q) = -\sum_{x} p(x) \log q(x).$$

Ситуация для непрерывного распределения аналогична:

$$\mathrm{H}(p, q) = -\int_{X} p(x) \log q(x)\, dx.$$

Регулизатор для функции потерь

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

SGD (обучение по минибатчам)

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

ADAM vs SGD

Algorithm 1 SGDM with Gradient Centralization

Input: Weight vector $\mathbf{w}^0$, step size α, momentum factor β, $\mathbf{m}^0$

Training step:

1: **for** $t = 1, ...T$ **do**
2: $\mathbf{g}^t = \nabla_{\mathbf{w}^t}\mathcal{L}$
3: $\widehat{\mathbf{g}}^t = \Phi_{GC}(\mathbf{g}^t)$
4: $\mathbf{m}^t = \beta\mathbf{m}^{t-1} + (1-\beta)\widehat{\mathbf{g}}^t$
5: $\mathbf{w}^{t+1} = \mathbf{w}^t - \alpha\mathbf{m}_t$
6: **end for**

Algorithm 2 Adam with Gradient Centralization

Input: Weight vector $\mathbf{w}^0$, step size α, β_1, β_2, ϵ, $\mathbf{m}^0$,$\mathbf{v}^0$

Training step:

1: **for** $t = 1, ...T$ **do**
2: $\mathbf{g}^t = \nabla_{\mathbf{w}^t}\mathcal{L}$
3: $\widehat{\mathbf{g}}^t = \Phi_{GC}(\mathbf{g}^t)$
4: $\mathbf{m}^t = \beta_1\mathbf{m}^{t-1} + (1-\beta_1)\widehat{\mathbf{g}}^t$
5: $\mathbf{v}^t = \beta_2\mathbf{v}^{t-1} + (1-\beta_2)\widehat{\mathbf{g}}^t \odot \widehat{\mathbf{g}}^t$
6: $\widehat{\mathbf{m}}^t = \mathbf{m}^t/(1-(\beta_1)^t)$
7: $\widehat{\mathbf{v}}^t = \mathbf{v}^t/(1-(\beta_2)^t)$
8: $\mathbf{w}^{t+1} = \mathbf{w}^t - \alpha\frac{\widehat{\mathbf{m}}^t}{\sqrt{\widehat{\mathbf{v}}^t}+\epsilon}$
9: **end for**

Структура нейронной сети Sequential

1. model = Sequential()

2.входной слой — Input либо указать input_shape в первом слое Dense:
3.model.add(keras.Input(shape=(4,)))

4. Dense — плотный слой, обязательно указать количество нейронов в слое
5.model.add(Dense(n))

6.Оптимизатор
7.model.add(Optimizier(n))

8. model.add(Dropout или BatchNormilize)

9. Dense-Activation

Слой BatchNormalization

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

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Пакетная нормализация применяет преобразование, которое поддерживает среднее значение вывода, близкое к 0, и стандартное отклонение вывода, близкое к 1.

https://arxiv.org/abs/1502.03167

Слой BatchNormalization

```
tf.keras.layers.BatchNormalization(
    axis=-1,
    momentum=0.99,
    epsilon=0.001,
    center=True,
    scale=True,
    beta_initializer="zeros",
    gamma_initializer="ones",
    moving_mean_initializer="zeros",
    moving_variance_initializer="ones",
    beta_regularizer=None,
    gamma_regularizer=None,
    beta_constraint=None,
    gamma_constraint=None,
    renorm=False,
    renorm_clipping=None,
    renorm_momentum=0.99,
    fused=None,
    trainable=True,
    virtual_batch_size=None,
    adjustment=None,
    name=None,
    **kwargs
)
```

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Слой BatchNormalization во время обучения

Во время обучения (т.е. при использовании fit() или вызове слоя / модели с аргументом training=True) уровень нормализует свой вывод, используя среднее значение и стандартное отклонение текущего пакета входных данных. Другими словами, для каждого нормализованного канала возвращается слой

(batch - mean(batch)) / (var(batch) + epsilon) * gamma + beta, где:

1. epsilon небольшая константа (настраивается как часть аргументов конструктора)
2. gamma- коэффициент масштабирования (инициализированный как 1), который можно отключить, передав scale=False.
3. Beta - это коэффициент смещения (инициализированный как 0), который можно отключить, передав center=False.

Слой BatchNormalization во время вывода

При использовании evaluate() или predict() или при вызове слоя / модели с аргументом training=False(который является значением по умолчанию), уровень нормализует свой вывод, используя скользящее среднее среднего значения и стандартного отклонения пакетов, которые он видел во время обучения:

(batch - self.moving_mean) / (self.moving_var + epsilon) * gamma + beta.

self.moving_mean и self.moving_var являются необучаемыми переменными, которые обновляются каждый раз при вызове слоя в режиме обучения:

moving_mean = moving_mean * momentum + mean(batch) * (1 - momentum)
moving_var = moving_var * momentum + var(batch) * (1 — momentum)

Таким образом, уровень будет нормализовать свои входные данные во время вывода только после обучения на данных, которые имеют такую же статистику, что и данные вывода

Компиляция, обучение, предсказание сети

```
model.ompile(

optimizer="rmsprop",
    loss=None,
    metrics=None,
    loss_weights=None,

weighted_metrics=No
ne,
    run_eagerly=None,

steps_per_execution=
None,
    **kwargs
)
```

```
model.fit(
    x=None,  y=None,
    batch_size=None,
    Epochs=1,  verbose=1,
    callbacks=None,
validation_split=0.0,
    validation_data=None,
    shuffle=True, class_weight=None,
    sample_weight=None,
    initial_epoch=0,
steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
max_queue_size=10,
    Workers=1,
use_multiprocessing=False,
)
```

```
Model.predict(
    x,
    batch_size=None,
    verbose=0,
    steps=None,
    callbacks=None,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False,
)
```

Обратные вызовы во время обучения

```
my_callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=2),
    tf.keras.callbacks.ModelCheckpoint(filepath='model.{epoch:02d}-
{val_loss:.2f}.h5'),
    tf.keras.callbacks.TensorBoard(log_dir='./logs'),
]
model.fit(dataset, epochs=10, callbacks=my_callbacks)
```

Снижение скорости обучения

```
tf.keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss",
    factor=0.1,
    patience=10,
    verbose=0,
    mode="auto",
    min_delta=0.0001,
    cooldown=0,
    min_lr=0,
    **kwargs
)
```

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
factor=0.2,
patience=5,min_lr=0.001)


model.fit(X_train, Y_train, callbacks=[reduce_lr])
```

Снижение скорости обучения

- monitor : отслеживаемый параметр.
- factor : коэффициент, на который будет снижена скорость обучения. new_lr = lr * factor.
- patience : количество эпох без улучшения, после которых скорость обучения будет снижена.
- verbose : int. 0: нет сообщений, 1: выводит сообщения.
- mode : один из {'auto', 'min', 'max'}. В 'min'режиме скорость обучения будет снижена, когда отслеживаемое количество перестанет уменьшаться; в 'max' режиме он будет уменьшен, когда отслеживаемое количество перестанет увеличиваться; в 'auto' режиме направление автоматически определяется по названию контролируемой величины.
- min_delta : порог для измерения нового оптимума, чтобы сосредоточиться только на значительных изменениях.
- cooldown : количество эпох перед возобновлением нормальной работы после уменьшения lr.
- min_lr : нижняя граница скорости обучения.

ModelCheckpoint

- Обратный вызов для сохранения модели Keras или веса модели с определенной периодичностью.
- ModelCheckpoint используется в сочетании с обучением с использованием model.fit()для сохранения модели или весов (в файле контрольной точки) через некоторый интервал, поэтому модель или веса могут быть загружены позже, чтобы продолжить обучение из сохраненного состояния.
- Этот обратный вызов предоставляет несколько вариантов:
- Следует ли сохранять только ту модель, которая на данный момент достигла «наилучших характеристик», или сохранять модель в конце каждой эпохи, независимо от производительности.
- Определение «лучший»; какое количество контролировать и следует ли его максимизировать или минимизировать.
- Частота сохранения. В настоящее время обратный вызов поддерживает сохранение в конце каждой эпохи или после фиксированного количества пакетов обучения.
- Сохраняются ли только веса или вся модель.

ModelCheckpoint

```
tf.keras.callbacks.ModelCheckpoint(
    filepath,
    monitor="val_loss",
    verbose=0,
    save_best_only=False,
    save_weights_only=False,
    mode="auto",
    save_freq="epoch",
    options=None,
    **kwargs
)
```

```
EPOCHS = 10
checkpoint_filepath = '/tmp/checkpoint'
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='val_acc',
    mode='max',
    save_best_only=True)

model.fit(epochs=EPOCHS, callbacks=[model_checkpoint_callback])

model.load_weights(checkpoint_filepath)
```

Снижение скорости обучения

- filepath : строка или PathLike, путь для сохранения файла модели. filepath может содержать именованные параметры форматирования, которые будут заполнены значением epochи ключами logs(переданы on_epoch_end). Например: если filepath есть weights.{epoch:02d}-{val_loss:.2f}.hdf5, то контрольные точки модели будут сохранены с номером эпохи и потерями проверки в имени файла.
- monitor : количество для мониторинга.
- verbose : режим детализации, 0 или 1.
- save_best_only : если save_best_only=True, последняя лучшая модель в соответствии с отслеживаемым количеством не будет перезаписана. Если filepath не содержит параметров форматирования, таких как {epoch},то filepath будет перезаписываться каждой новой лучшей моделью.
- mode : один из {auto, min, auto}. Если save_best_only=True решение перезаписать текущий файл сохранения принимается на основе максимизации или минимизации отслеживаемого количества. Для val_acc этого должно быть auto, для val_loss этого должно быть min и т. Д. В auto режиме направление автоматически выводится из названия контролируемой величины.

Снижение скорости обучения

- save_weights_only : если True, то будут сохранены только веса модели (model.save_weights(filepath)), иначе сохраняется полная модель (model.save(filepath)).
- save_freq : 'epoch' или целое число. При использовании 'epoch' обратный вызов сохраняет модель после каждой эпохи. При использовании целого числа обратный вызов сохраняет модель в конце этого количества пакетов. Если Model скомпилирован с steps_per_execution=N, то критерии сохранения будут проверяться каждый N-й пакет. Обратите внимание, что если сохранение не выровнено по эпохам, отслеживаемая метрика потенциально может быть менее надежной (она может отражать всего 1 пакет, поскольку метрики сбрасываются каждую эпоху). По умолчанию 'epoch'.

EarlyStopping

- Прекращение обучение, когда отслеживаемый показатель перестал улучшаться.
- Предполагая, что цель обучения - минимизировать потери. При этом контролируемая метрика будет 'loss', а режим будет 'min'. Цикл model.fit() обучения будет проверять в конце каждой эпохи, не уменьшаются ли потери. Как только выясняется, что потери не уменьшаются, то model.stop_training помечается как True, и обучение прекращается.
- Количество, которое нужно отслеживать, должно быть доступно в logsdict. Для этого передайте потерю или метрики в model.compile().

EarlyStopping

```
tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=0,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)
```

```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

model = tf.keras.models.Sequential([tf.keras.layers.Dense(10)])
model.compile(tf.keras.optimizers.SGD(), loss='mse')
history = model.fit(np.arange(100).reshape(5, 20), np.zeros(5),
...                     epochs=10, batch_size=1, callbacks=[callback],
...                     verbose=0)
```

EarlyStopping

- monitor : параметр, который необходимо контролировать.
- min_delta : Минимальное изменение отслеживаемого параметра, которое квалифицируется как улучшение, т. е. абсолютное изменение менее min_delta, не будет считаться улучшением.
- patience : количество эпох без улучшения, после которых обучение будет остановлено.
- verbose : режим детализации.
- режим : Один из {"auto", "min", "max"}. В min режиме обучение остановится, когда отслеживаемое количество перестанет уменьшаться; в "max" режиме он остановится, когда контролируемое количество перестанет увеличиваться; в "auto" режиме направление автоматически определяется по названию контролируемой величины.
- baseline : базовое значение для отслеживаемого количества. Обучение остановится, если модель не покажет улучшения по сравнению с базовым уровнем.
- restore_best_weights : следует ли восстанавливать веса модели из эпохи с наилучшим значением наблюдаемой величины. Если False, используются веса модели, полученные на последнем этапе обучения.

Сохранение модели

```
from tensorflow.keras.models import load_model


model.save('my_model')  #  'my_model.h5'
del model
model = load_model('my_model')
```

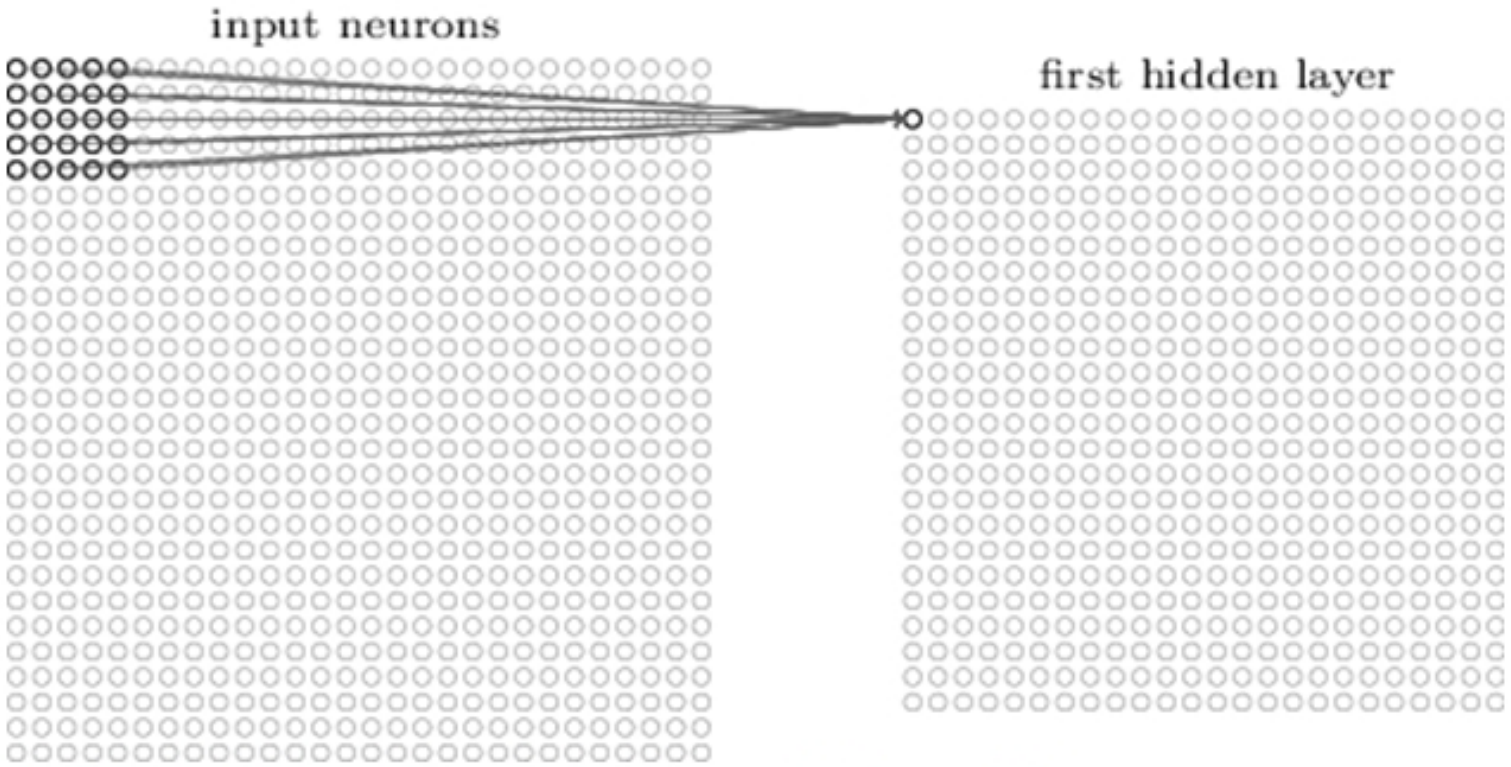
Сохранить только веса

```
model.save_weights('my_model_weights.h5')
model.load_weights('my_model_weights.h5')
```

Персептрон для двумерного входного изображения

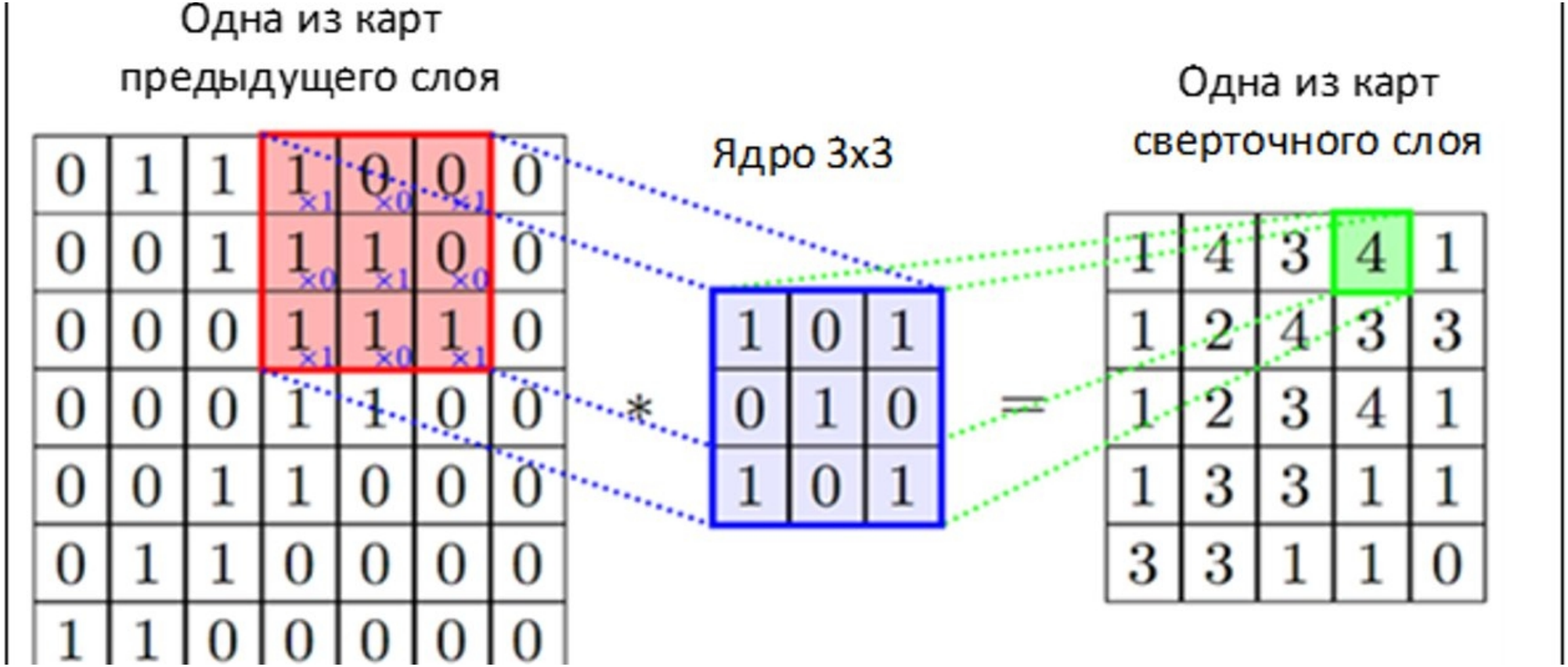
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

Свертка



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

Свертка


Одна из карт
предыдущего слоя
Ядро 3x3
Одна из карт
сверточного слоя
0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 1 1 0 0 0 0
1 1 0 0 0 0 0
1 0 1
0 1 0
1 0 1
*
=
1 4 3 4 1
1 2 4 3 3
1 2 3 4 1
1 3 3 1 1
3 3 1 1 0

Сверточные сети

Изображение или карты признаков в рамках одного слоя могут сканироваться не одним, а несколькими независимыми фильтрами, давая таким образом на выход не одну карту, а несколько (их ещё называют «каналами»). Настройка весов каждого фильтра происходит при помощи процедуры backpropagation.

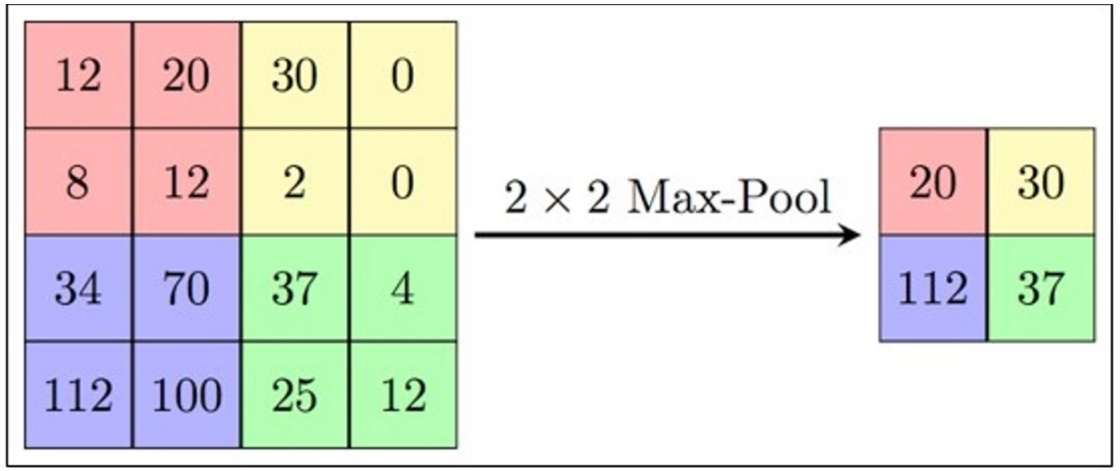
Paddings и strides

1. Очевидно, если ядро фильтра при сканировании не выходит за пределы изображения, размерность карты признаков будет меньше, чем у исходной картинки. Если нужно сохранить тот же размер, применяют paddings — значения, которыми дополняется изображение по краям и которые потом захватываются фильтром вместе с реальными пикселями картинки.
2. Помимо paddings на изменение размерности так же влияют strides — значения шага, с которым окно перемещается по изображению/карте.

MaxPooling

Свёртка не является единственным способом получения обобщённой характеристики группы пикселей. Самый простой способ для этого — выбрать один пиксель по заданному правилу, например — максимальный. Именно это и делает слой MaxPooling.

В отличие от convolution, maxpooling обычно применяется к непересекающимся группам пикселей.



Класс Conv2D

```
tf.keras.layers.Conv2D(
    filters,  kernel_size,     strides=(1, 1),
    padding="valid",   data_format=None,
    dilation_rate=(1, 1),   groups=1, activation=None,

    use_bias=True,  kernel_initializer="glorot_uniform",
    bias_initializer="zeros", kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,   **kwargs
 )
```

Класс Conv2D

filters: целое число, размерность выходного пространства (т. е. количество выходных фильтров в свертке).

kernel_size : целое число или кортеж / список из 2x целых чисел, определяющий высоту и ширину окна двумерной свертки. Может быть одним целым числом, чтобы указать одно и то же значение для всех пространственных измерений.

strides: целое число или кортеж / список из 2x целых чисел, определяющих шаги свертки по высоте и ширине. Может быть одним целым числом, чтобы указать одно и то же значение для всех пространственных измерений. Указание любого значения шага ! = 1 несовместимо с указанием любого dilation_rate значения ! = 1.

padding: одно из "valid"или "same". "valid" означает отсутствие отступов. "same" приводит к равномерному заполнению слева / справа или вверх / вниз от ввода, так что вывод имеет ту же высоту / ширину, что и ввод.

Класс Conv2D

data_format : строка, принимающая одно из значений channels_last(по умолчанию) или channels_first. Порядок размеров во входных данных channels_last соответствует входам с формой, (batch_size, height, width, channels), а channels_first соответствует входам с формой (batch_size, channels, height, width).

dilation_rate : целое число или кортеж / список из 2х целых чисел, определяющий скорость расширения, используемую для расширенной свертки. Может быть одним целым числом, чтобы указать одно и то же значение для всех пространственных измерений.

activation: функция активации для использования. Если ничего не указать, активация не применяется.

use_bias : логическое значение, определяет, использует ли слой вектор смещения.

Класс Conv2D

kernel_initializer : Инициализатор для kernel матрицы весов.

bias_initializer : Инициализатор вектора смещения.

kernel_regularizer : функция регуляризатора, примененная к kernel матрице весов.

bias_regularizer : функция регуляризатора, примененная к вектору смещения.

activity_regularizer : функция регуляризатора, примененная к выходу слоя (его «активация»).

kernel_constraint : функция ограничения, применяемая к матрице ядра.

bias_constraint : функция ограничения, применяемая к вектору смещения.

Класс Conv2D

Входное изображение w*h*d

Гиперпараметры сверточного слоя:

K — число фильтров

F — размер ядра

S — stride

P — padding

Выход

W2 = (w-F+2P)/S+1

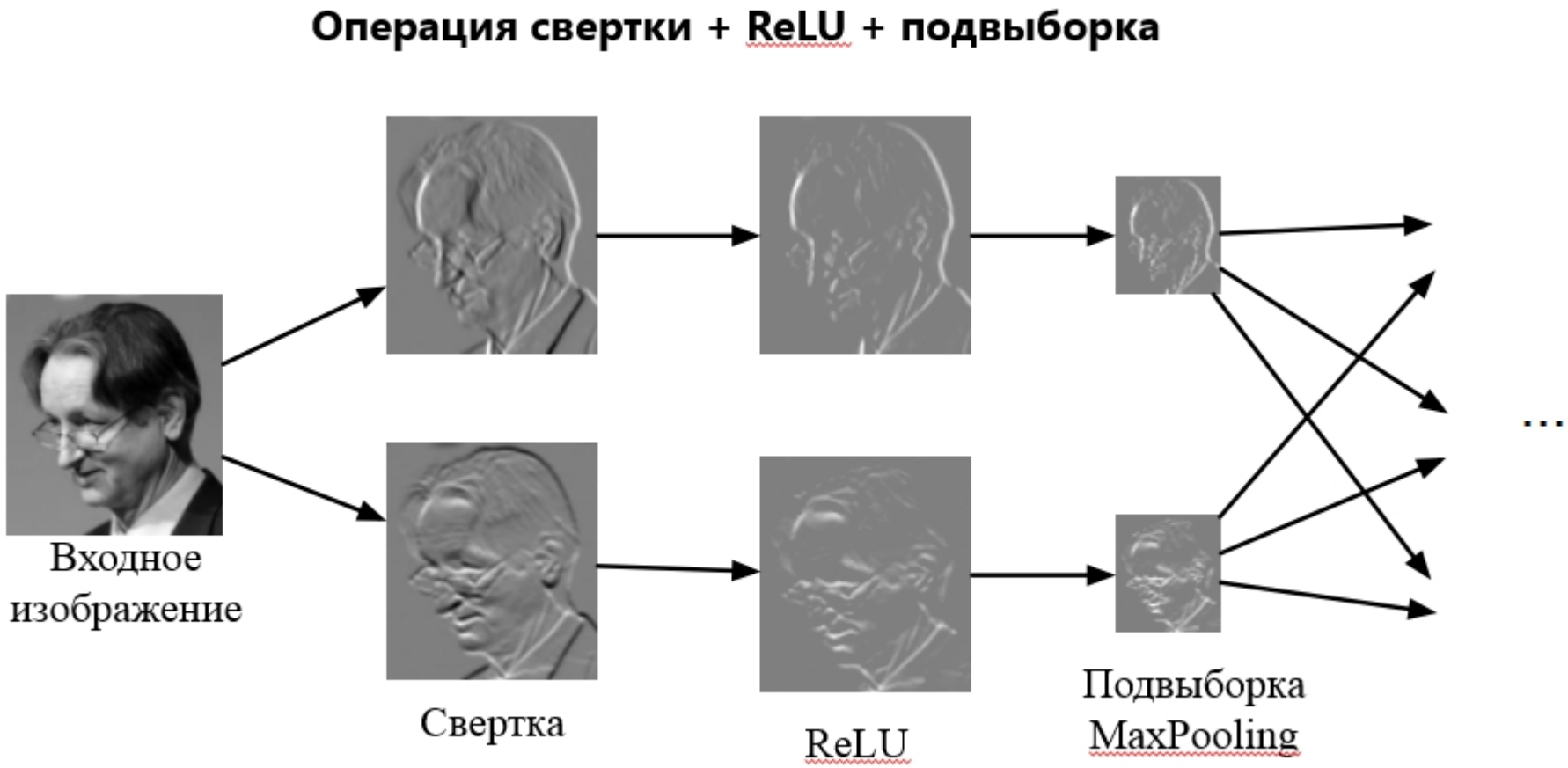
H2 = (h-F+2P)/S+1

D2 = K

Пример сверточной сети

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ])
```

Демонстрация работы сверточной сети

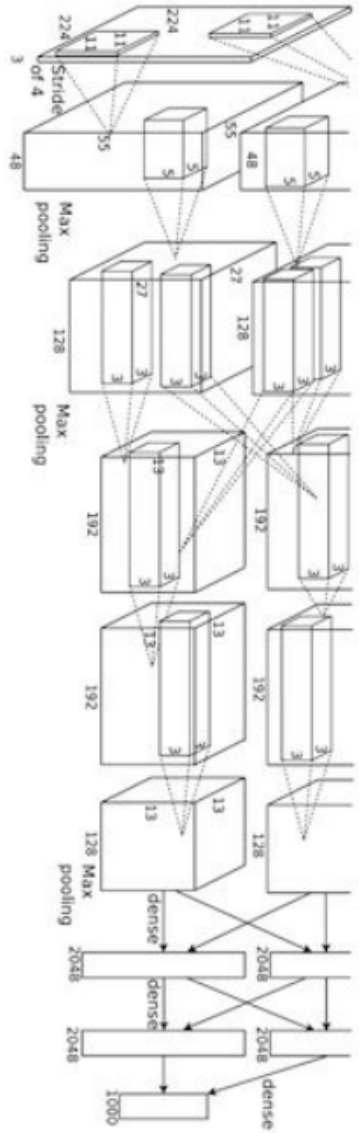


Convolutional network (AlexNet)

input image

weights

loss



Задание

1. создайте сверточную сеть для классификации данных mnist, оцените точность полученных результатов и ошибки по классам. Сохраните модель, сохраните наиболее эффективные веса.

2. создайте многослойный персептрон и сверточную сеть для классификации данных fashion_mnist, оцените точность полученных результатов и ошибки по классам. Сохраните модель, сохраните наиболее эффективные веса.

3. Продемонстрируйте работу обратных вызовов в процессе обучения