Основы нейронных сетей

Лекция 1

Введение. Модель нейронов. Нейронные сети

Основы нейронных сетей

- Структура курса и УКД
- Необходимое ПО
- Литература (список + Интернет) и справочная литература

Лекция 1. Введение. Модель нейронов. Нейронные сети

- Нейронная сеть. Основные свойства систем
- Хронология
- Области применения
- Задачи
- Модели нейронов

Нейронная сеть. Основные свойства систем

В основу ИНС положены черты живых нейросетей:

- простой обрабатывающий элемент - нейрон
- большое количество нейронов
- глобальные связи
- изменяющиеся по весу связи между нейронами
- параллелизм обработки информации

Нейронная сеть. Основные свойства систем

- Искусственная нейронная сеть (ИНС) - частный случай методов распознавания образов, анализа признаков, адаптивного управления , алгоритмов для робототехники и др. (машинное обучение);
- Обучение нейронных сетей — это многопараметрическая задача нелинейной оптимизации (математическая дисциплина);
- Нейронная сеть — модель параллельных и распределенных вычислений (вычислительная техника и программирование);
- ИНС является основой философского течения коннекционизма и основным направлением в структурном подходе по изучению возможности построения (моделирования) естественного интеллекта с помощью компьютерных алгоритмов (ИИ).

Нейронная сеть. Основные свойства систем

Этапы развития ИИ
- семиотический — создание экспертных систем, баз знаний и систем логического вывода, имитирующих высокоуровневые психические процессы: мышление, рассуждение, речь, эмоции, творчество и т. д.;
- биологический — изучение нейронных сетей и эволюционных вычислений, моделирующих интеллектуальное поведение на основе биологических элементов, а также создание соответствующих вычислительных систем, таких как нейрокомпьютер или биокомпьютер.

Нейронная сеть. Основные свойства систем

Разработка машин (40-50гг) с интеллектуальным поведением
1 подход - символьно-алгоритмическая парадигма:
а) ПЗ в виде множества атомных семантических объектов или символов;
б) манипуляция ими по формальным алгоритмическим правил
=> основа традиционных ИИ (экспертные системы)

Нейронная сеть. Основные свойства систем

Недостатки:

- не все знания можно формализовать;
- разум не устройство с обработкой информации по определенным правилам.

Знания разделяются на:

- формализуемые (математика)
- интуитивные (экспертные - врача, художника...)

Нейронная сеть. Основные свойства систем

2-ой подход - коннекционистская парадигма

Для реализации некоторых возможностей мозга необходимо реализовать его архитектурные особенности, т.е.

- *НС- многосвязная сеть простых процессоров (нейронов), каждый из которых имеет много входов и выходов.*

Нейронная сеть. Основные свойства систем

- Такие процессоры обычно довольно просты (особенно в сравнении с процессорами, используемыми в персональных компьютерах). Каждый процессор подобной сети имеет дело только с сигналами, которые он периодически получает, и сигналами, которые он периодически посылает другим процессорам.

- Будучи соединёнными в достаточно большую сеть с управляемым взаимодействием, такие по отдельности простые процессоры вместе способны выполнять довольно сложные задачи.

Нейронная сеть. Основные свойства систем

Особенности интеллекта:
- зрение (моторика, обработка органов чувств, распознавание, …)
- творчество (создание, восприятие, оценка, …)
- …
- локатор лет. мыши

Мозг:
- изменение структуры всю жизнь (2 года)
- пластичность мозга - настройка нервной системы в соответствии с окружающими условиями

Нейронная сеть. Основные свойства систем

Схожесть НС с мозгом:
- знания поступают извне и участвуют в процессе обучения
- для накопления знаний используются связи, называемые синаптическими весами
Процедура изменения синаптических весов, называется алгоритмом обучения

Возможность обучения — одно из главных преимуществ нейронных сетей перед традиционными алгоритмами (но и его сложность) .

Нейронная сеть. Основные свойства систем

Этапы разработки систем с НС для решения конкретной задачи:

- Сбор данных для обучения;
- Подготовка и нормализация данных;
- Выбор топологии сети;
- Экспериментальный подбор характеристик сети;
- Экспериментальный подбор параметров обучения;
- Собственно обучение;
- Проверка адекватности обучения;
- Корректировка параметров, окончательное обучение;
- Описание и использование сети

Нейронная сеть. Основные свойства систем

Системы с НС:
1. Вместо программ - набор весов нейронов, вместо программирования - обучение нейронов
2. Обрабатываемые объекты представлены неявно весами, возможность обобщения

Нейронная сеть. Основные свойства систем

1. Нелинейность
2. Отображение входной информации в выходную (аппроксимация)
3. Адаптиность
4. Очевидность ответа (для увеличения достоверности принимаемого решения)
5. Контекстная информация влияет на структуру сети
6. Отказоустойчивость (устойчива к неисправностям отдельных узлов)

Нейронная сеть. Основные свойства систем

7. Эффективность из-за параллельной структуры (способна к быстрым вычислениям)
8. Единообразие анализа и проектирования , т. е.
 НС- достаточно универсальный механизм обработки информации в разных предметных областях:
 - нейроны – это часть любой сети
 - одни и те же сети и алгоритмы обучения в разных приложениях

Нейронная сеть. Хронология

- фон Нейман аналогия процессоров с нейрона ми
- 42 год Норберт Винер - идеи кибернетики - рассмотрение биологических процессов с мат и инженерной точек зрения
- Маккалок и Питтс опубликовали формальное описание ИНС (первой формальной моделью НС была модель МакКаллока-Питтса, уточненная и развитая Клини)
- Хебб (49г) описал способность обучаться - вес должен изменяться для отражения корреляции между входом и выходом

Нейронная сеть. Хронология

- Розенблатт (57) - персептрон
- Бернард Уидроу (60) + студент Хофф на основе дельта-правила (формулы Уидроу) разработали Адалин, который сразу начал использоваться для задач предсказания и адаптивного управления.
- Минский и Пайперт (69) указали класс задач, нельзя решить персептроном - упадок
- далее многослойные сети
- Т. Кохонен и Дж. Андерсон (72) – новый тип нейронных сетей – (82) сети на основе памяти (SOM)
- Пол Дж. Вербос и Галушкин А. И (74) - алгоритм обратного распространения ошибки (дважды!!); Дэвид И. Румельхарт, Дж. Е. Хинтон и Рональд Дж. Вильямс, а так же независимо и одновременно С. И. Барцев и В. А. Охонин (86)
- А. Г. Ивахненко (70-ые) удавалось обучать восьмислойные нейронные сети, в основе которых лежал искусственный нейрон, основанный на интерполяционном полиноме Колмогорова — Габора

Нейронная сеть. Хронология

- Хопфилд (82) - сеть с обратными связями
- Ян Лекун (88) - свёрточная нейронная сеть (англ. convolutional neural network, CNN) — специальная архитектура искусственных нейронных сетей, нацеленная на эффективное распознавание образов
- Джеффри Хинтон (2007) в университете Торонто созданы алгоритмы глубокого обучения многослойных нейронных сетей
- Юрген Шмитхубер – (97) архитектура LSTM
- 2010 - База фотографий Image Net открыла возможности для машинного обучения
- 2012 – AlexNet, далее VGG…, ResNet – неполное обучение (Transfer Learning)
- 2018 – Google BERT, TableNet (19) …
- 2021 - Китай представил собственную генеративную модель глубокого обучения нейросеть Wu Dao с 1,75 трлн параметров (мультимодальная система для разных целей)

Нейронная сеть. Области применения

- распознавание образов
- оптическое распознавание речи, звука
- системы машинного перевода
- системы управления и технического контроля
- инструмент изучения человеческого мозга
- нейробиология

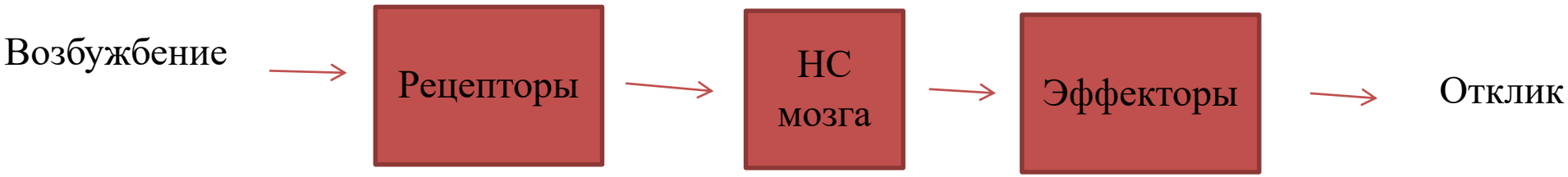
Нейронная сеть. Задачи

- Распознавание образов и классификация
- Принятие решений и управление. Эта задача близка к задаче классификации. Классификации подлежат ситуации, характеристики которых поступают на вход нейронной сети. На выходе сети при этом должен появиться признак решения, которое она приняла. При этом в качестве входных сигналов используются различные критерии описания состояния управляемой системы
- Кластеризация
- Прогнозирование
- Аппроксимация
- Сжатие данных и ассоциативная память
- Анализ данных

Нейронная сеть. Области применения (некоторые примеры)

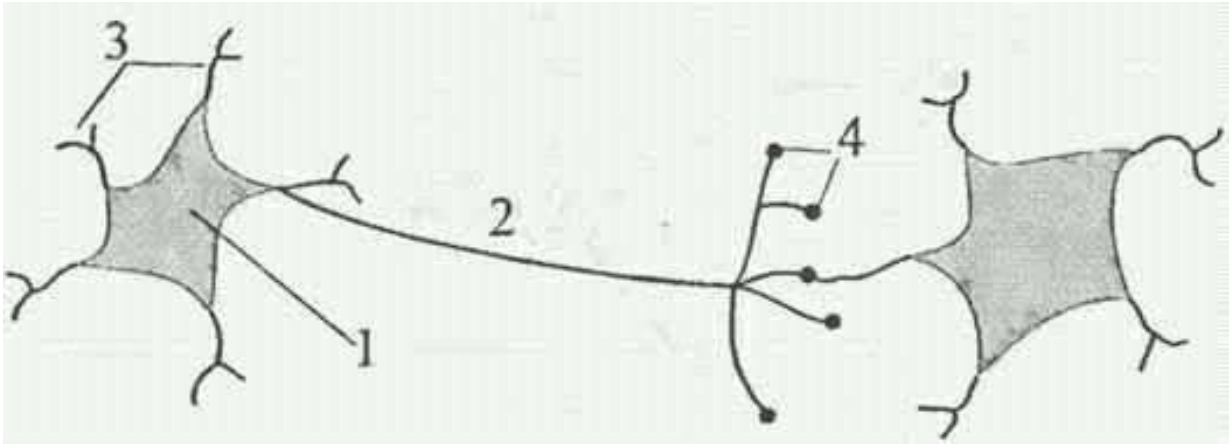
- Алиса, Google Assistant, …
- Microsoft обучает искусственный интеллект Drawing Bot рисовать по текстовому описанию (2018)
- алгоритм Brain на основе нейросетей ежедневно работает над системой рекомендаций Ютуба. Он подбирает для пользователей релевантный контент, изучая их поведение
- нейросеть Яндекса записывает музыкальный альбом
- Яндекс использует алгоритм Yandex Data Factory, чтобы помочь крупным промышленным производствам сокращать издержки. Система анализирует данные о, например, загруженности дорог или подсказывает безопасные расстояния от линий электропередач до высокорастущих деревьев
- нейросеть заставила пользователей соцсетей превращать свои фотографии в подобие шедевров искусства (разработчики из Беларуси - сервис MSQRD, который потом купил Facebook) и …

НС. Модели нейронов. Человеческий мозг



Нейроны медленнее ЛЭ, компенсирует массой (~10биллионов) и количеством связей (60 триллионов)
=> эффективная структура
Энергетические затраты мозга е-16 Дж в среднем на операцию в секунду, а компьютера - е-6 Дж

Упрощенная модель мозга

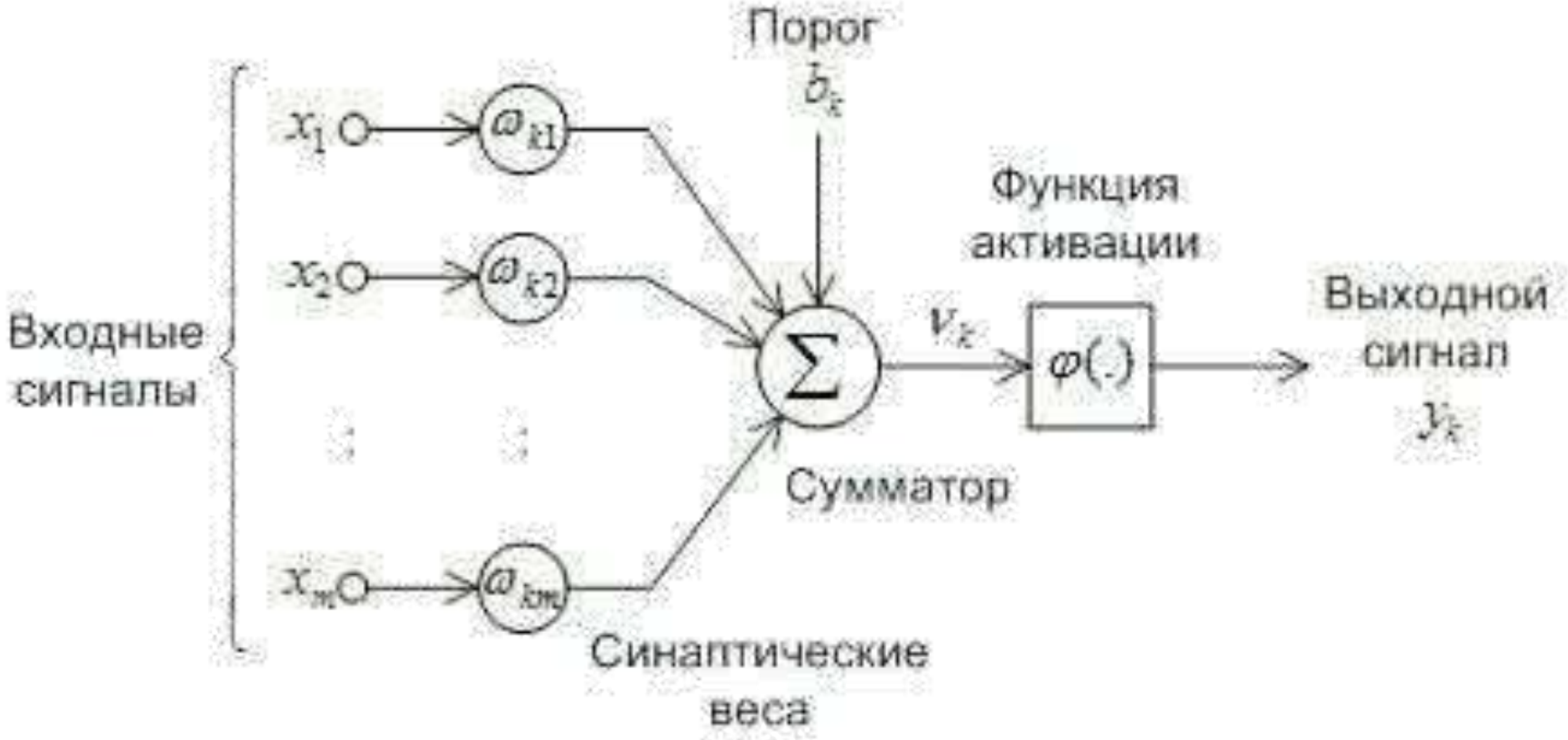


Упрощенная модель нейрона и его соединения с соседним нейроном: 1 - тело клетки, 2 - аксон, 3 - дендриты. 4 - синапсы.

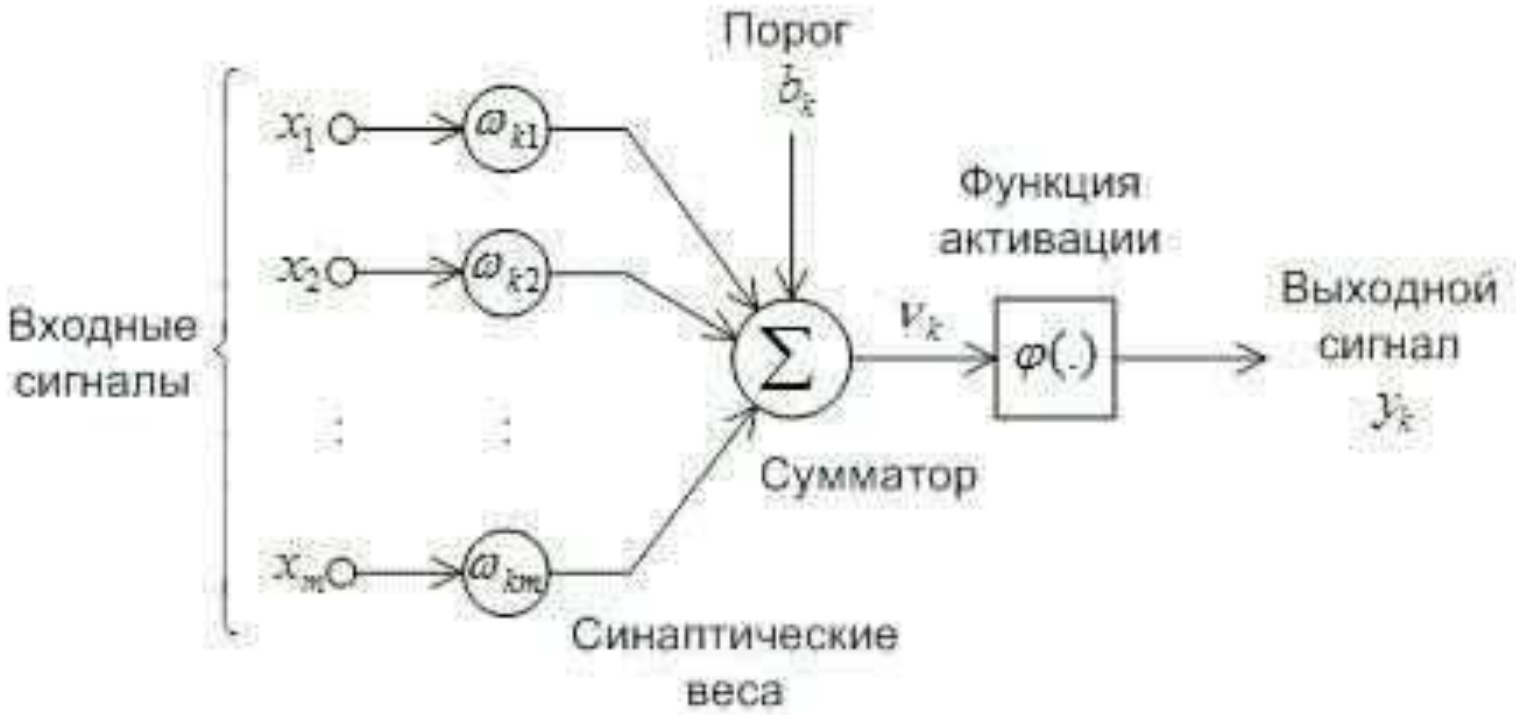
Блочная модель нейрона

1. Набор синапсов или связей, каждый из которых характеризуется своим весом w_{ij}
2. Сумматор - линейная комбинация входных сигналов
3. Функция активация - ограничивает амплитуду выходного сигнала - функция сжатия
- обеспечивает функциональное описание различных элементов

Блочная модель



Блочная модель



Математическая запись

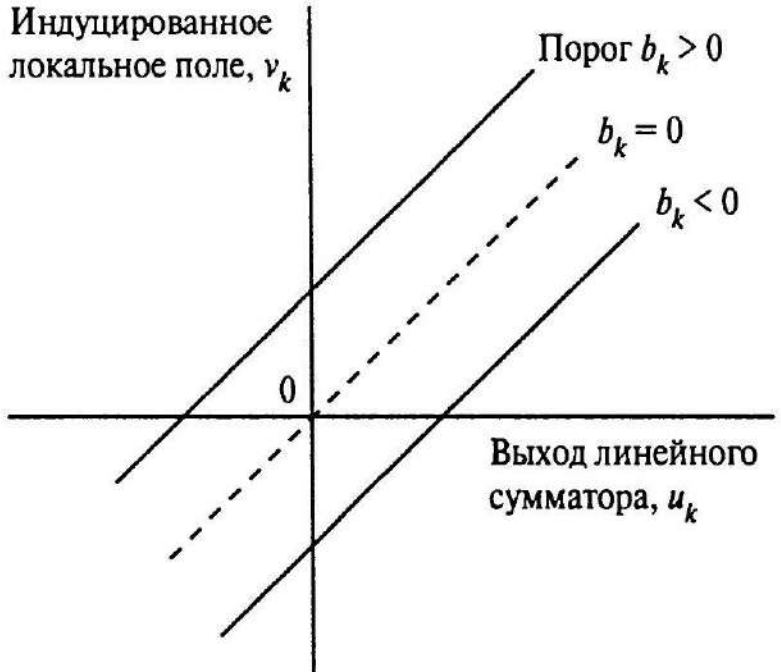
$$u_k = \sum_{j=1}^{m} w_{kj} x_j,$$

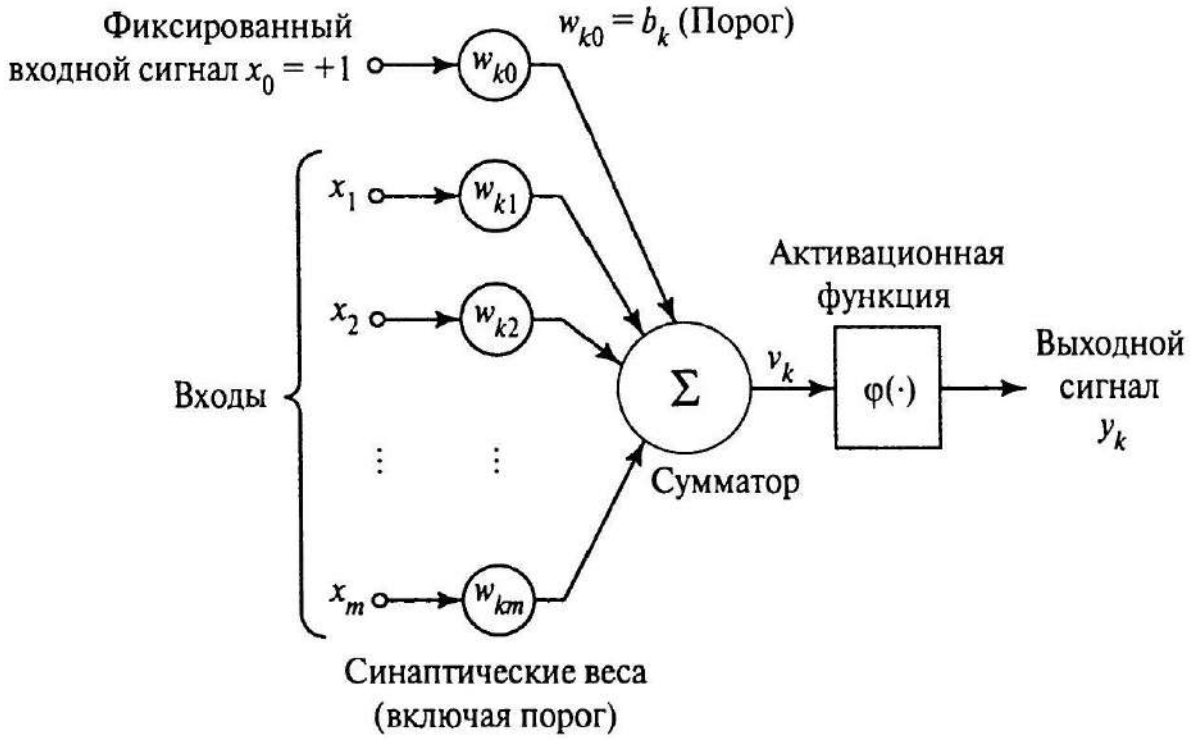
$$v_k = u_k + b_k.$$

$$y_k = \varphi(u_k + b_k)$$

Блочная модель порог

Порог b_k- афинное преобразование выхода, внешний параметр сети, иногда добавляют к весам с фиксированным входным сигналом $x_0=1$ и весом $w_{k0}=b_k$

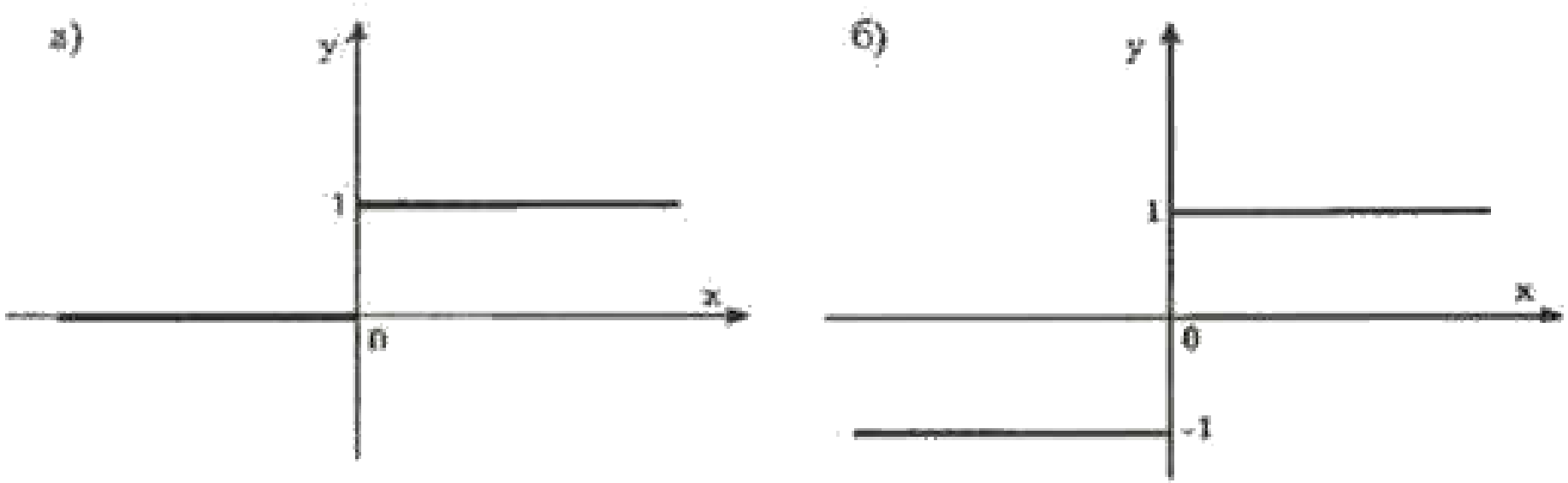




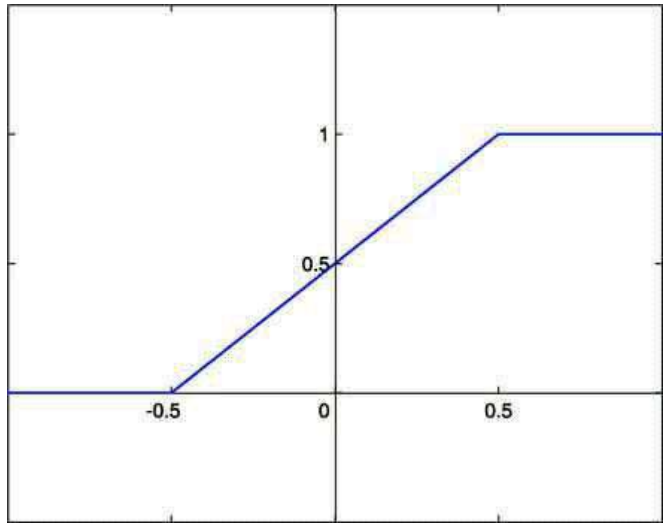
Функции активации

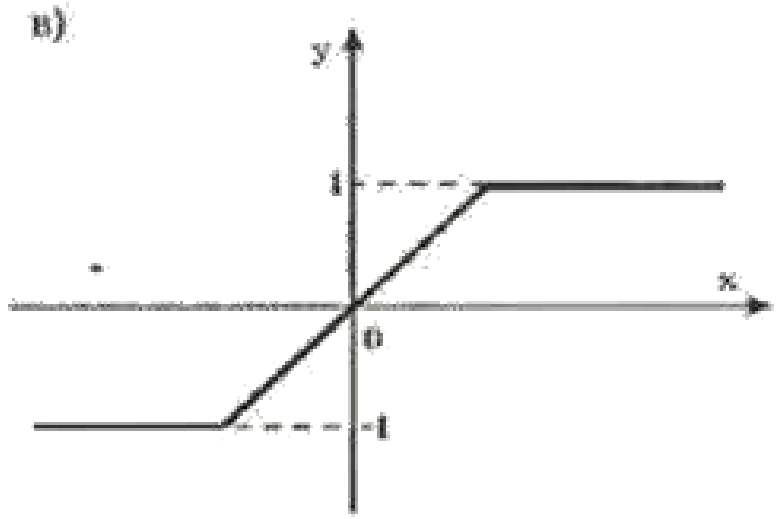
Пороговая передаточная функция

Другое название — функция Хевисайда.



Функции активации





Функции активации

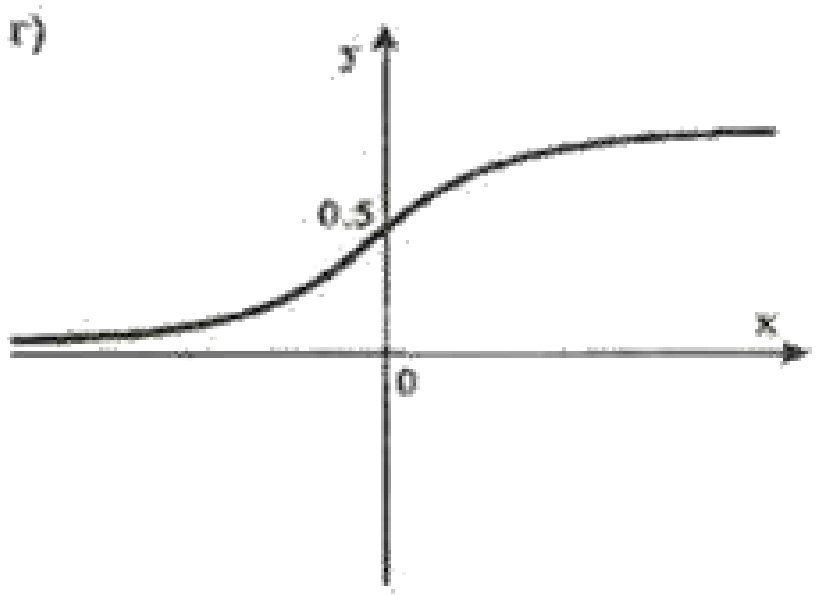
Логистическая функция - сигмоид

Математически логистическую функцию функцию можно выразить так:

f (x)=1/(1+exp (-tx))

Здесь *t* — это параметр функции, определяющий её *крутизну*. Когда *t* стремится к бесконечности, функция вырождается в пороговую. При t=0 сигмоида вырождается в постоянную функцию со значением 0,5. Область значений данной функции находится в интервале (0,1).

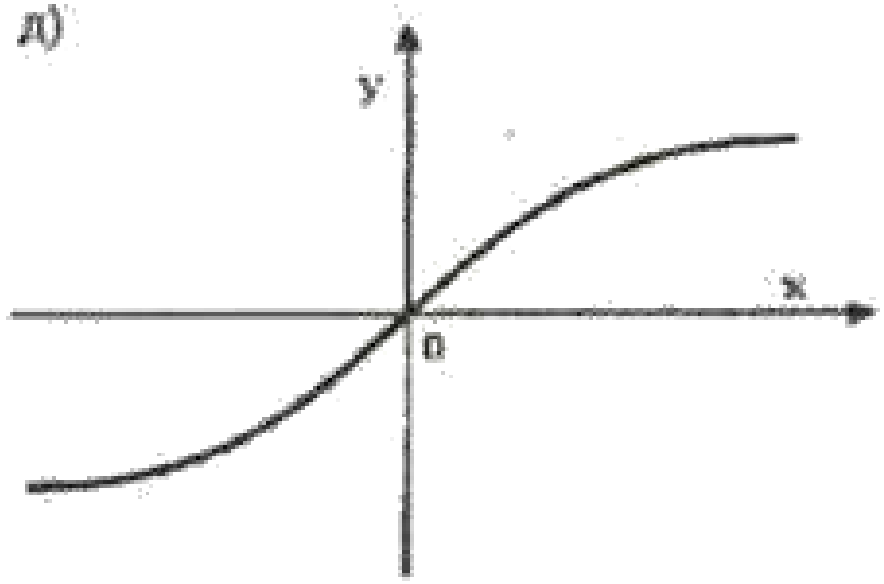
Обобщение - SoftMax



Функции активации

Биполярная функция

f (x)=xtanh (bx), чаще b=1



Функции активации

$$\varphi(v) = \frac{1 - \exp(-av)}{1 + \exp(-av)} = \tanh(\frac{av}{2}),$$

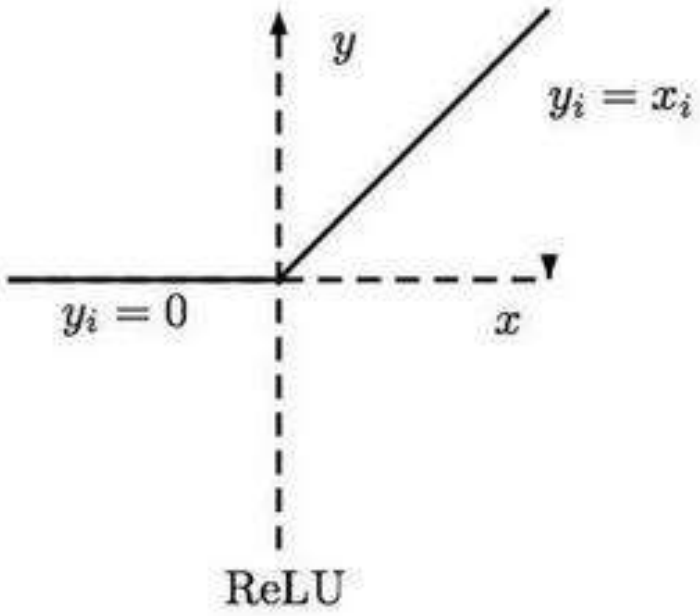
$$\varphi(v) = \frac{v}{\sqrt{1 + v^2}},$$

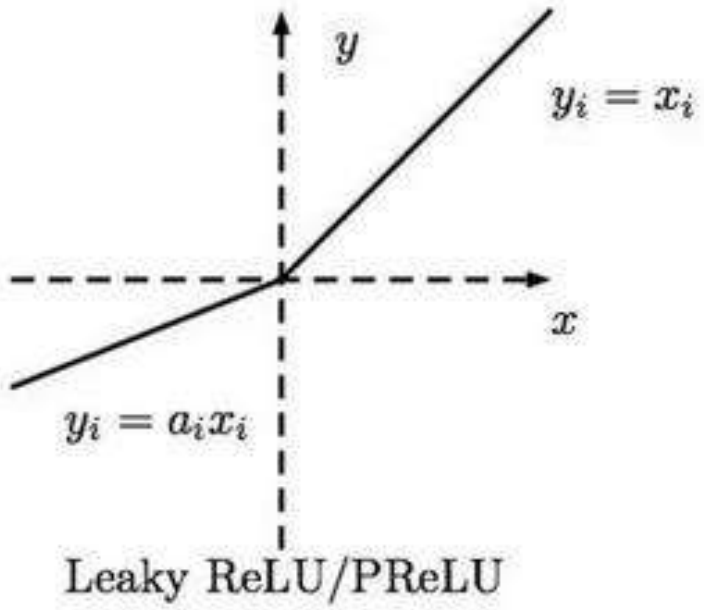
Функции активации

ReLU

$f(u) = \max(0, u)$

Range: [0, ∞)





НС. Стахостическая модель нейрона

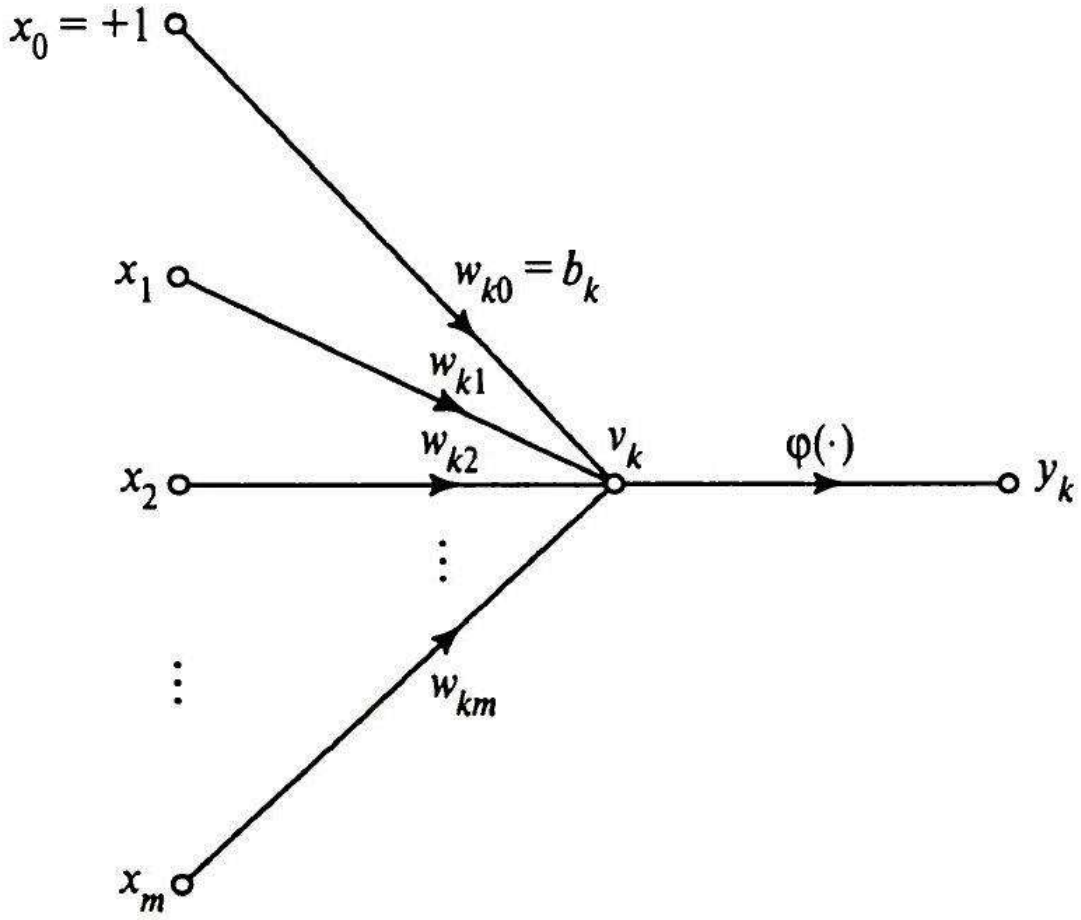
Если преобразован входного сигнала в выходной точно определено для всех значений входного сигнала, эта модель является детерминистской.

Состояние нейрона =1 с вероятностью P (v),

=-1 вероятностью 1-P (v)

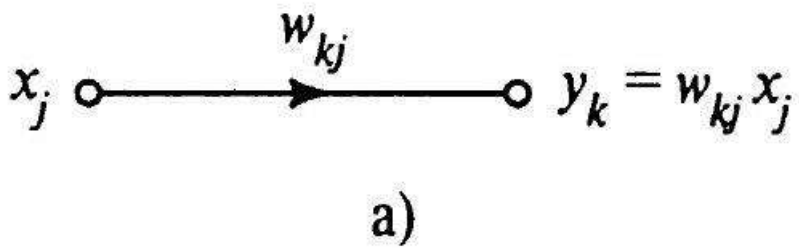
где P (v)=1/(1+exp(-v/t))., t- синаптический шум

НС. Модели нейронов. Ориентированный граф (53г Мейсон)

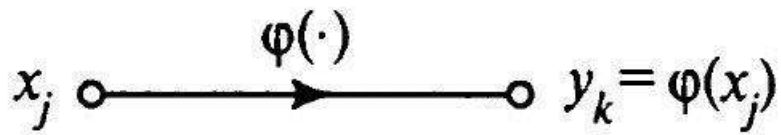


Граф передачи сигнала - это сеть направленных связей, соединяющих отдельные точки-узлы. С каждым узлом j связан сигнал x, направленная связь из него заканчивается в узле k. Связь характеризуется передаточной функцией.
Прохождение сигнала подчиняется трем правилам:

Правило 1



а)

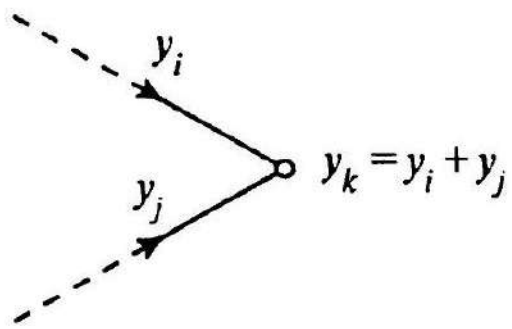


б)

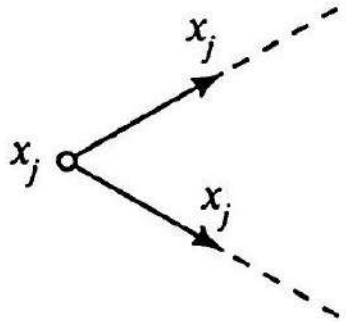
Направление прохождения сигнала вдоль каждой связи определяется направлением стрелки.

Выделяют синаптические (а) и активационные (б) связи.

Правила 2 и 3



в)



г)

Правило 2. Сигнал узла = алгебраической сумме сигналов, поступающих на его вход (в)

Правило 3. Сигнал данного узла передается по каждой исходящей связи без учета передаточных функций исходящих связей (г)

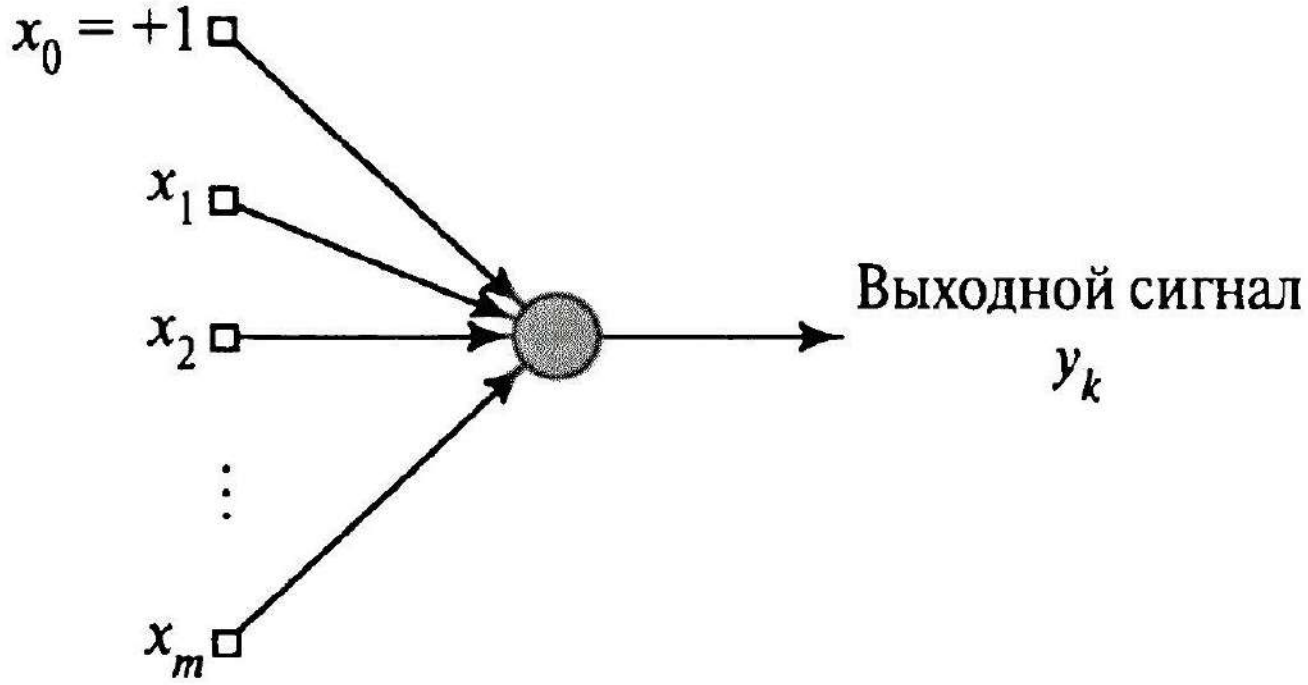
Еще одно определение НС (С. Хайкин)

Нейронная сеть — это направленный граф, состоящий из узлов, соединенных синаптическими и активационными связями, который характеризуется следующими четырьмя свойствами.

1. *Каждый нейрон представляется множеством линейных синаптических связей, внешним порогом и, возможно, нелинейной связью активации. Порог, представляемый входной синаптической связью, считается равным* $+1$.
2. *Синаптические связи нейрона используются для взвешивания соответствующих входных сигналов.*
3. *Взвешенная сумма входных сигналов определяет индуцированное локальное поле каждого конкретного нейрона.*
4. *Активационные связи модифицируют индуцированное локальное поле нейрона, создавая выходной сигнал.*

НС. Модели нейронов. Структурный граф

Частично полный ориентированный граф, в котором нет весов и обозначаются входные, вычислительные и выходные:



Основы нейронных сетей

Лекция 2

Классификация нейронных сетей. Библиотеки

Классификация нейронных сетей. Библиотеки

- Виды классификаций
- Классификация по архитектуре сетей
- Библиотеки для работы с нейронными сетями
- Структуры представления сетей
- Основные методы работы сетей

Виды классификаций НС

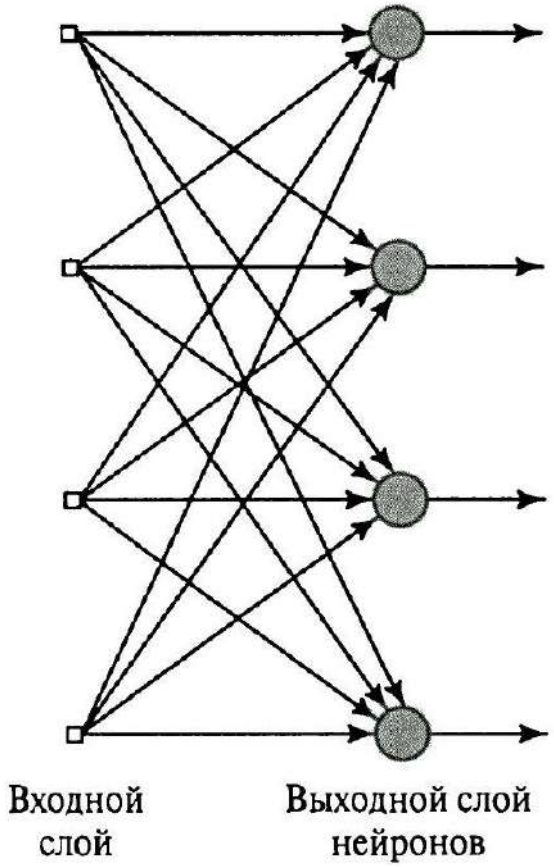
- По архитектуре (характеру связей)
- По количеству связей
- По алгоритму обучения (связь с внешней средой)
- По типу обучения

Классификация НС по характеру связей (архитектуре)

- Нейронные сети прямого распространения (направления):
  - Однослойные
  - Многослойные
  - РБС, …
- Рекуррентные нейронные сети (наличие обратной связи)

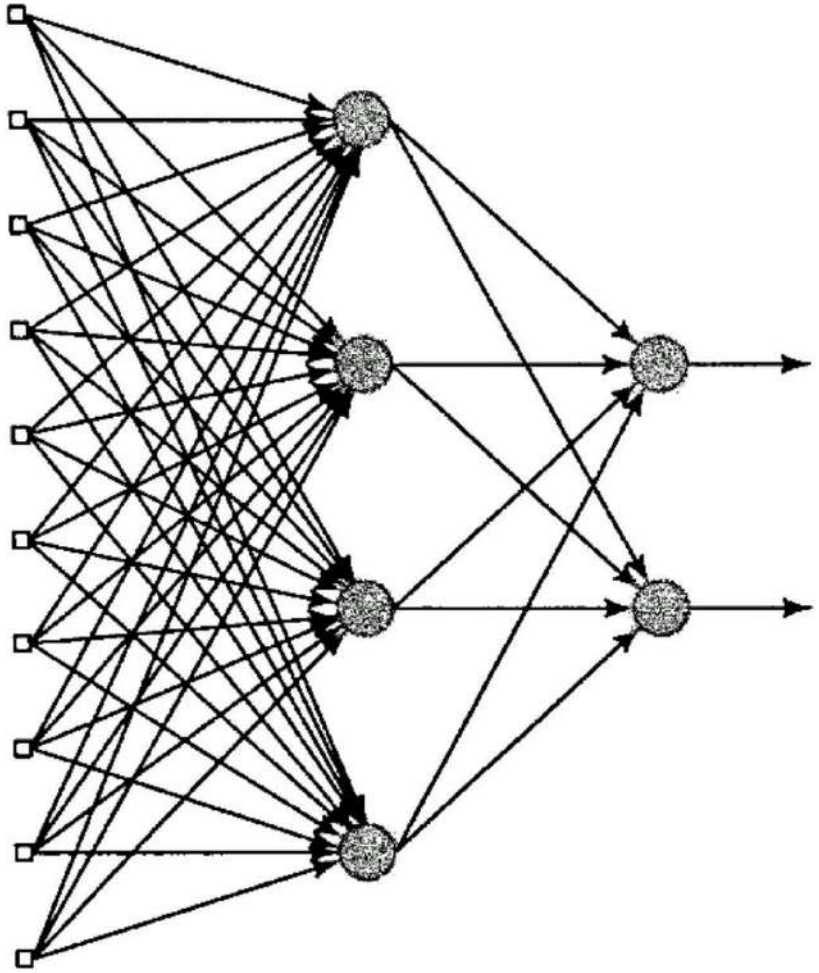
Виды классификаций НС

Однослойные сети (структурный граф)



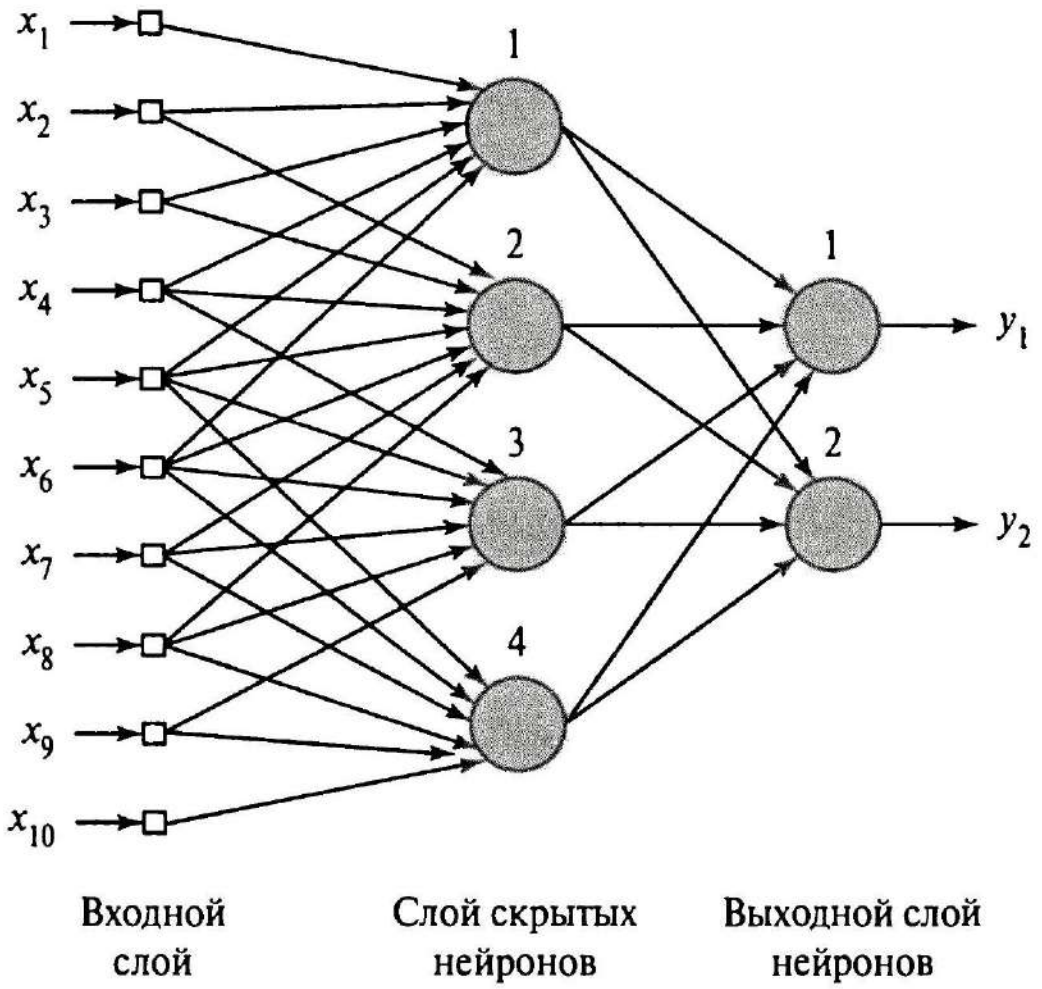
Многослойные сети

Глобальные свойства
данных можно выделить
с помощью скрытых слоев
Сеть m-h1-h2-q
10-4-2
Неполносвязная сеть

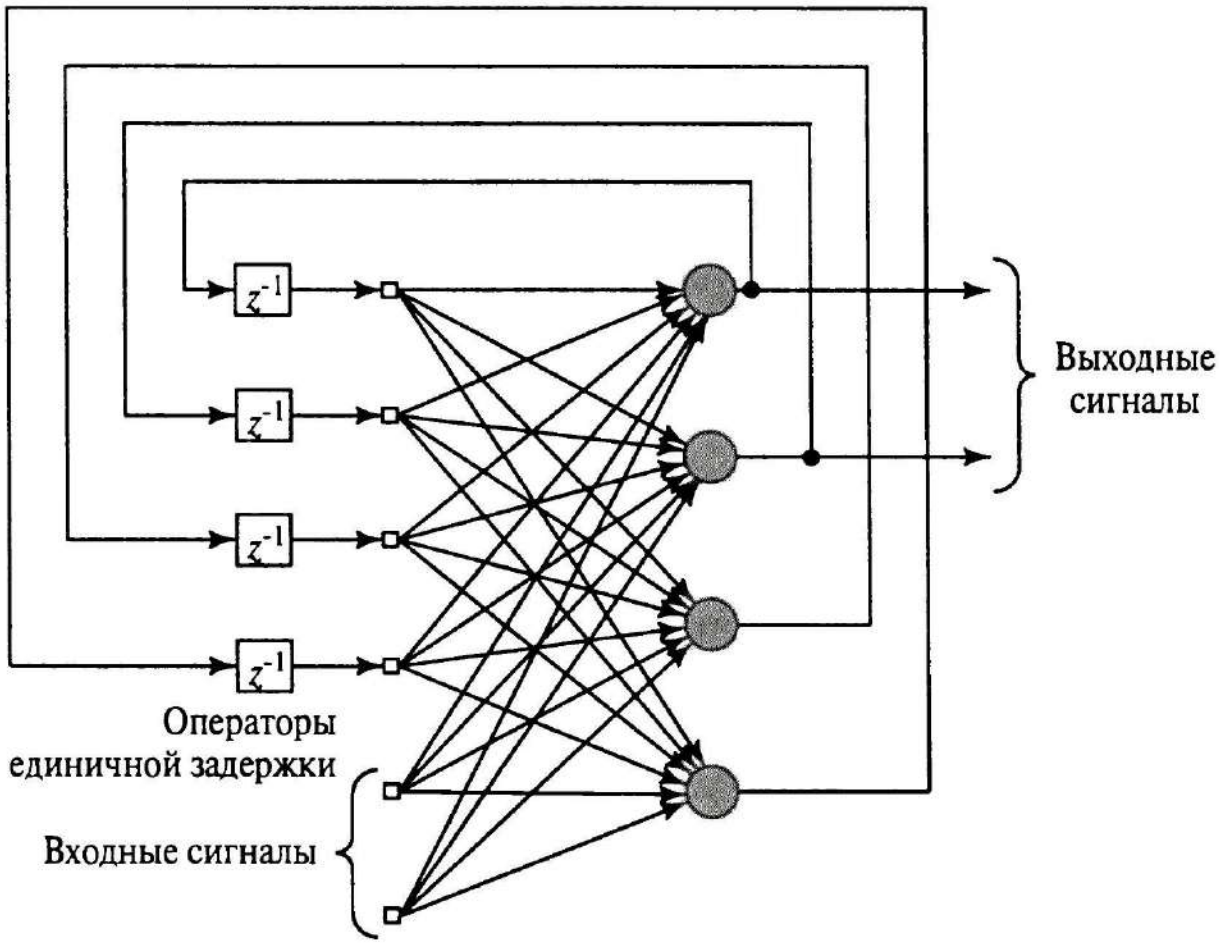


Многослойные сети

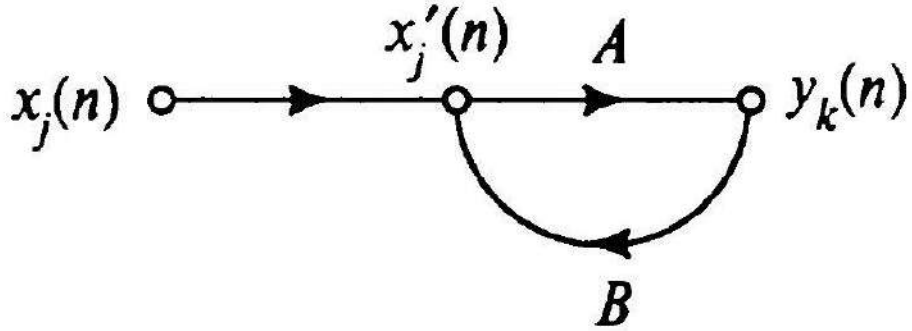
Глобальные свойства данных можно выделить с помощью скрытых слоев
Сеть m-h1-q
10-4-2
Неполносвязная сеть



Рекуррентные сети

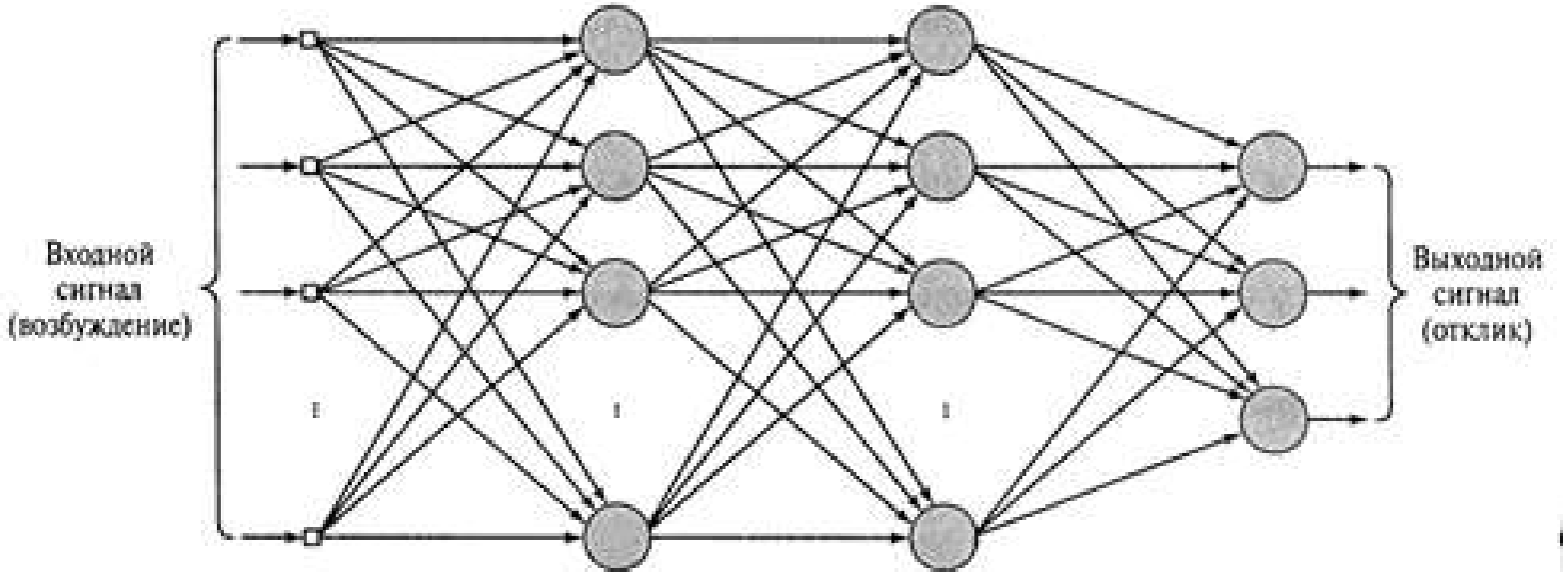


Обозначение обратной связи

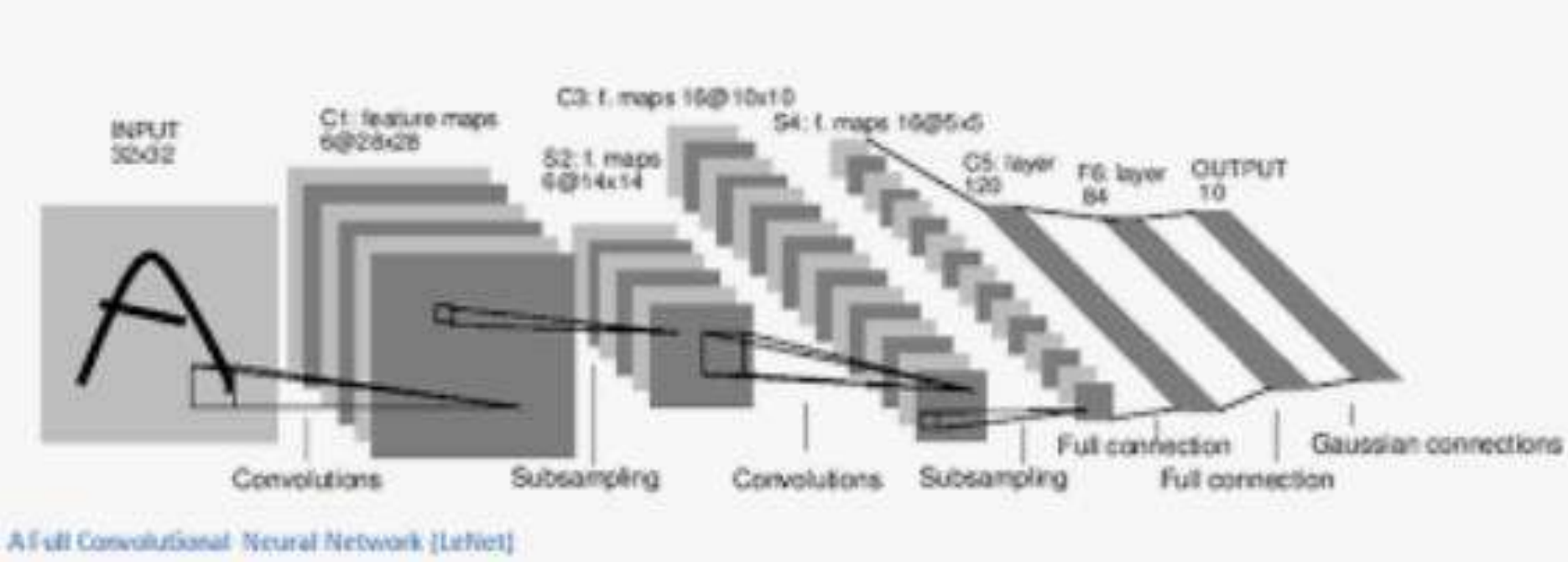


$$y_k(n) = A[x_j'(n)],$$
$$x_j'(n) = x_j(n) + B[y_j(n)],$$

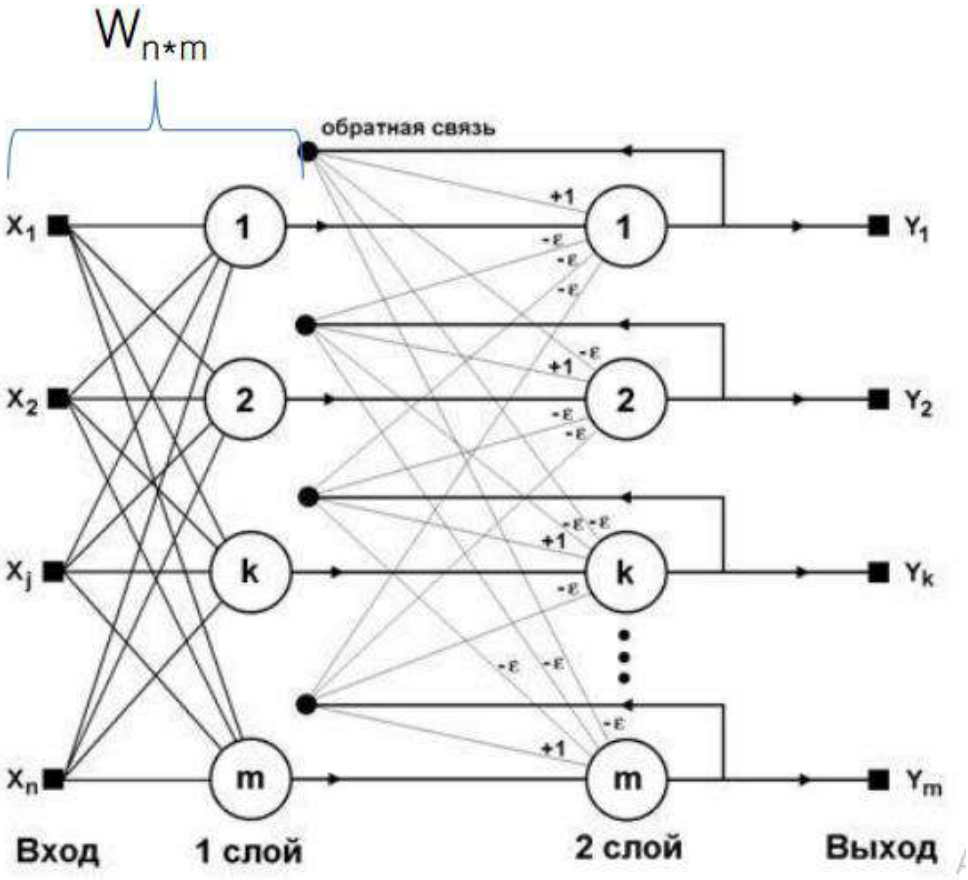
Разные топологии



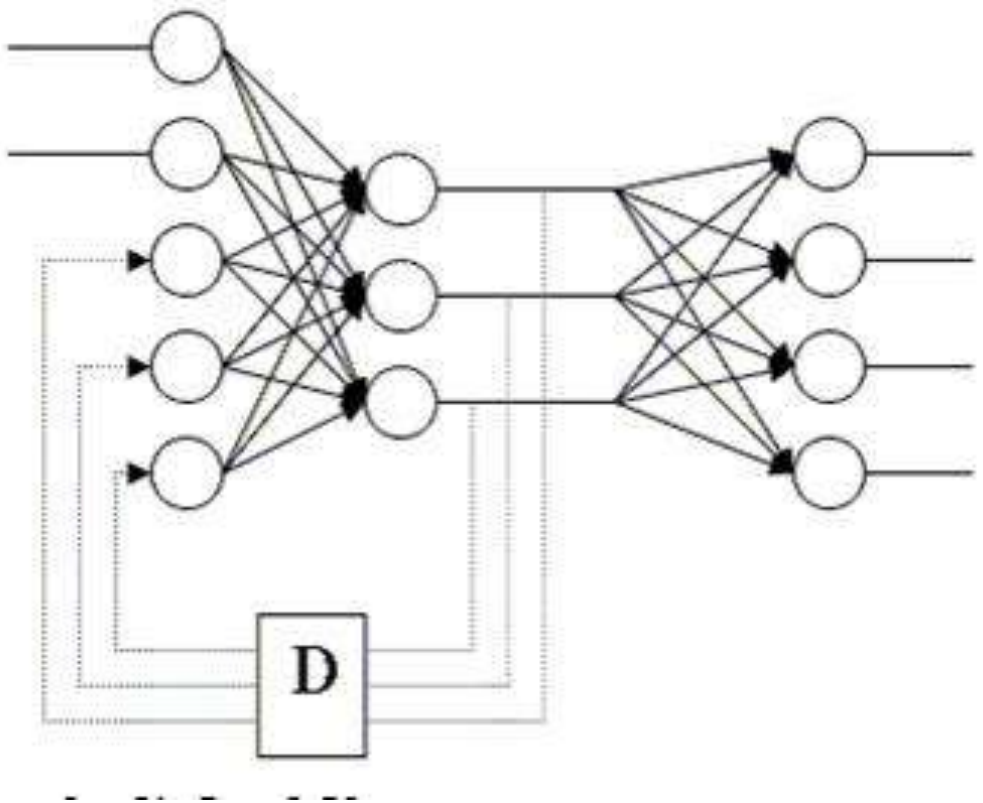
Разные топологии



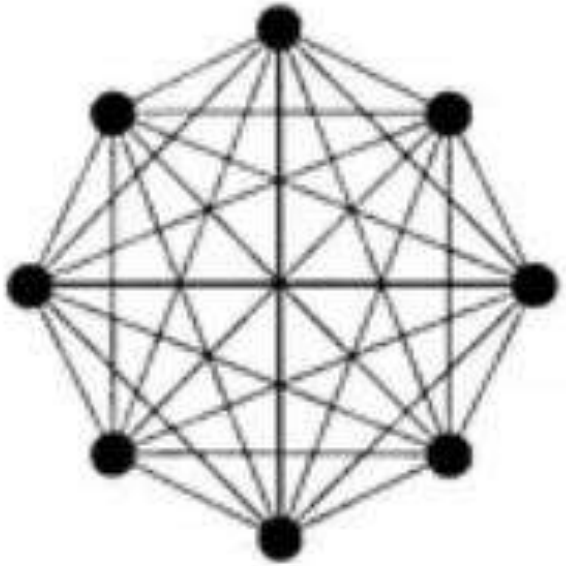
Разные топологии

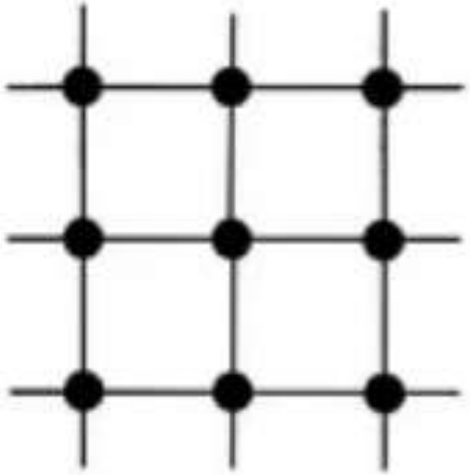


Разные топологии

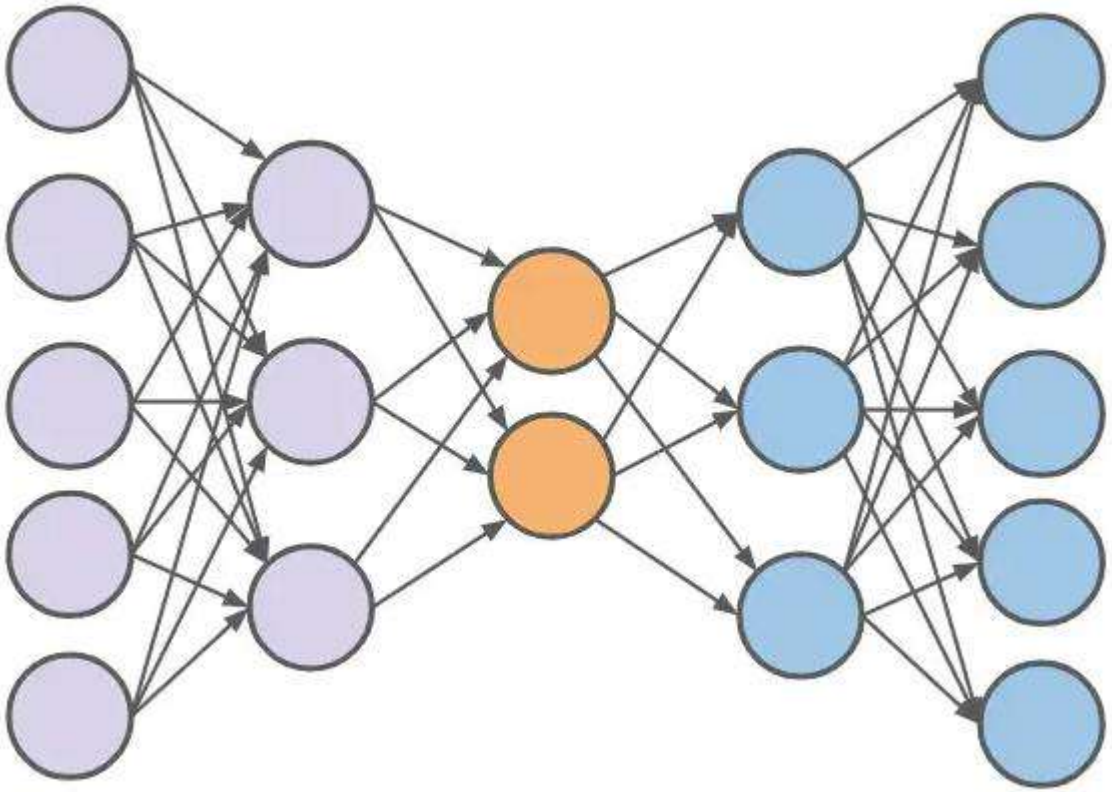


Разные топологии





Разные топологии



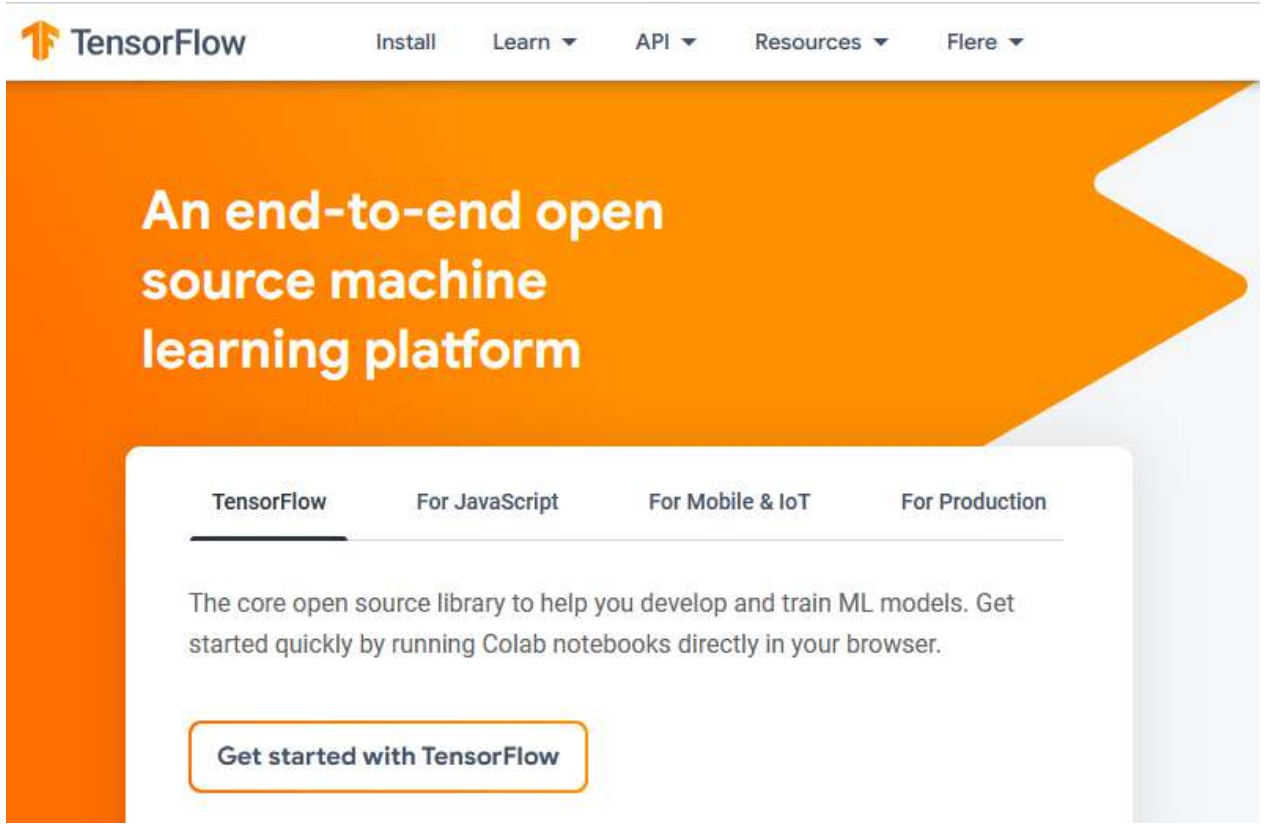
Библиотеки для работы с нейронными сетями (+ МО)

- Tensorflow (https://www.tensorflow.org/)
- Keras (keras.io)
- PyTorch (https://pytorch.org/)
- Scikit-learn (scikit-learn.org)
- OpenCV (https://opencv.org/)

Библиотеки для работы с НС

- Tensorflow (https://www.tensorflow.org/)

Открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. Применяется как для исследований, так и для разработки собственных продуктов Google. Основной API для работы с библиотекой реализован для Python, также существуют реализации для R, C Sharp, C++, Haskell, Java, Go и Swift.
Год выхода: 2015



ı для работы с НС

s://www.tensorflow.org/)

Models & datasets
Pre-trained models and datasets built by Google and the community

Tools
Ecosystem of tools to help you use TensorFlow

Libraries & extensions
Libraries and extensions built on TensorFlow

TensorFlow Certificate program
Differentiate yourself by demonstrating your ML proficiency

Learn ML
Educational resources to learn the fundamentals of ML with TensorFlow

Responsible AI
Resources and tools to

Models & datasets
Pre-trained models and datasets built by Google and the community

1 для работы с НС

TensorFlow Datasets: a collection of ready-to-use datasets.

TensorFlow Datasets is a collection of datasets ready to use, with TensorFlow or other Python ML frameworks, such as Jax. All datasets are exposed as `tf.data.Datasets`, enabling easy-to-use and high-performance input pipelines. To get started see the guide and our list of datasets.

```
import tensorflow as tf
import tensorflow_datasets as tfds

# Construct a tf.data.Dataset
ds = tfds.load('mnist', split='train', shuffle_files=True)

# Build your input pipeline
ds = ds.shuffle(1024).batch(32).prefetch(tf.data.AUTOTUNE)
for example in ds.take(1):
  image, label = example["image"], example["label"]
```

RUN IN A CO NOTEBOOK

Learn ML
Educational resources to learn the fundamentals of ML with TensorFlow

Responsible AI
Resources and tools to integrate Responsible AI

Models & datasets

Pre-trained models and datasets built by Google and the community

1 для работы с НС

tensorflow / tfjs-models Public

Notifications | Fork 3.4k | Star 10.7k

Code | Pull requests 95 | Actions | Security | Insights

master | 65 branches | 73 tags | Go to file | Code

ahmedsabie Add FaceDetection TFJS implementation (#912) ... 70848f5 4 days ago 748 commits

.tslint	Add linting rules for tfjs-models. (#333)	2 years ago
.vscode	Move pose-detection code needed in hand detection to a common folde...	6 months ago
blazeface	move node version to 12 to allow security patches (#924)	last month
body-pix	Bump log4js from 6.3.0 to 6.4.1 in /body-pix (#933)	last month
body-segmentation	Remove images from npm release (#936)	last month
coco-ssd	move node version to 12 to allow security patches (#924)	last month
deeplab	move node version to 12 to allow security patches (#924)	last month
face-landmarks-detection	Add FaceDetection TFJS implementation (#912)	4 days ago
hand-pose-detection	Add FaceDetection TFJS implementation (#912)	4 days ago

About

Pretrained models for TensorFlow.js

js.tensorflow.org

Readme

Apache-2.0 License

10.7k stars

263 watching

3.4k forks

Releases 2

TFJS Task API v0.0.1-alpha.6 Latest

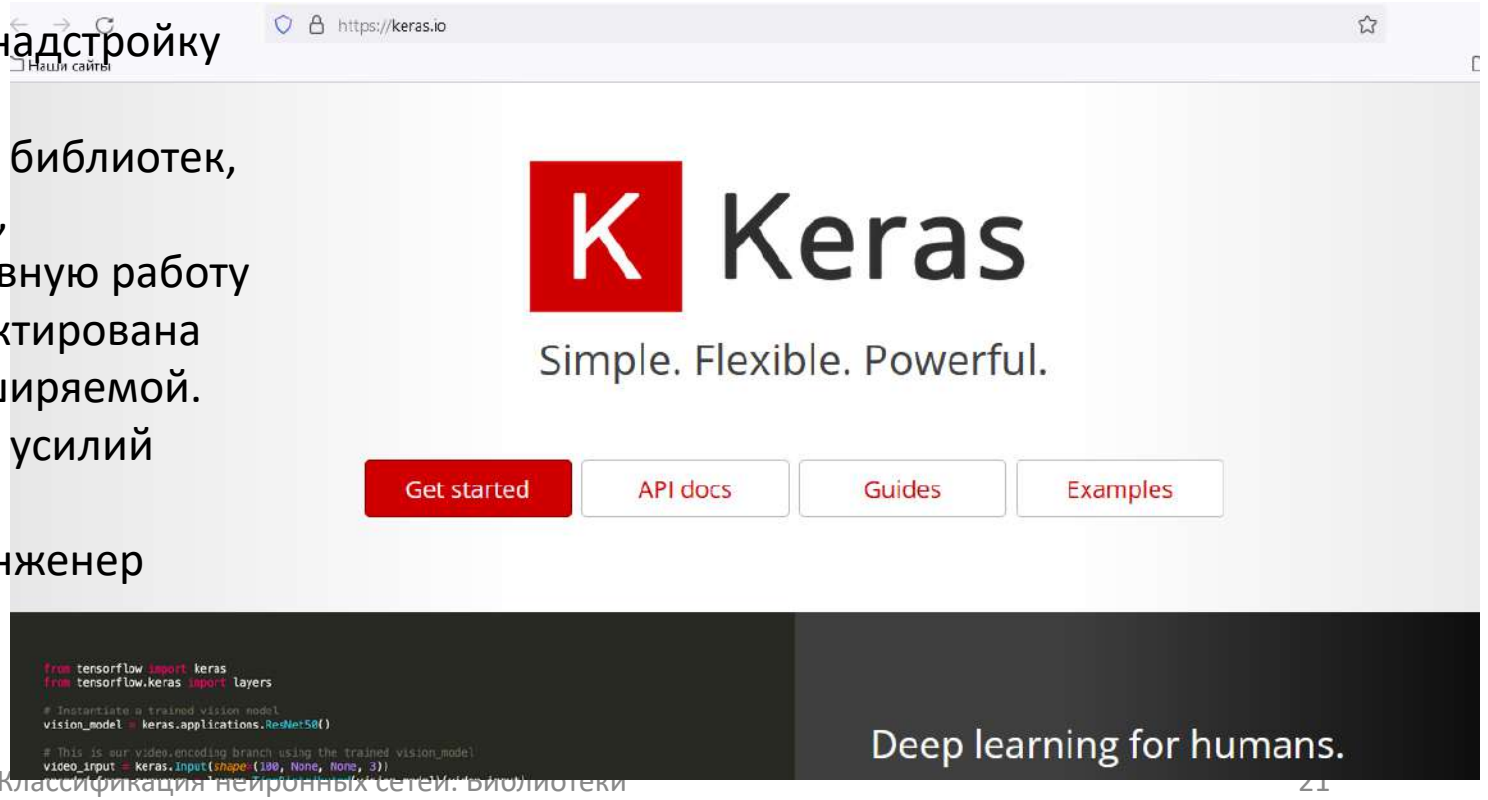
on 13 May 2021

+ 1 release

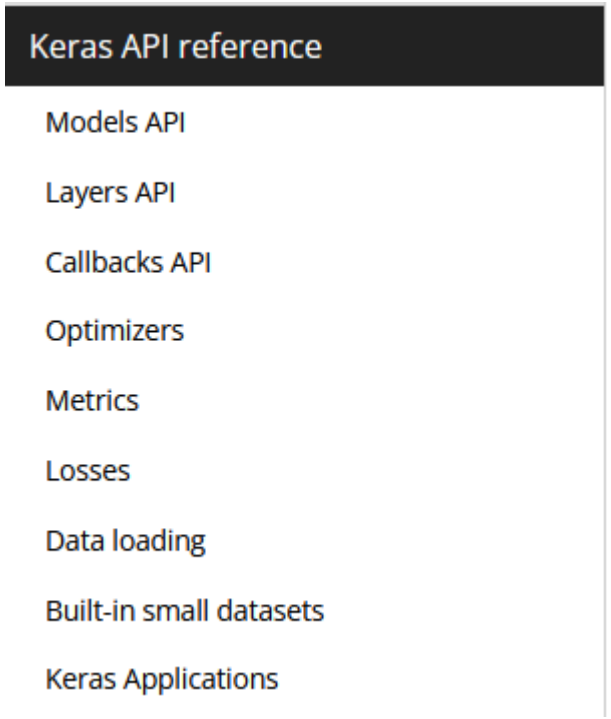
Библиотеки для работы с НС

- Keras (keras.io)

Открытая библиотека, написанная на языке Python и обеспечивающая взаимодействие с искусственными нейронными сетями. Она представляет собой надстройку над фреймворком TensorFlow. До версии 2.3 поддерживались разные версии нейросетевых библиотек, такие как TensorFlow, Microsoft Cognitive Toolkit, Deeplearning4j, и Theano. Нацелена на оперативную работу с сетями глубинного обучения, при этом спроектирована так, чтобы быть компактной, модульной и расширяемой. Она была создана как часть исследовательских усилий проекта ONEIROS, а ее основным автором и поддерживающим является Франсуа Шолле, инженер Google.
Год выхода: 2015



Библиотеки для работы с НС

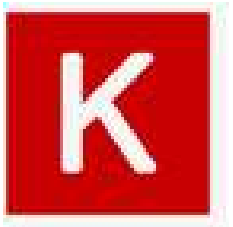


Models API:

The Model class
The Sequential class
Model training APIs

Слои

- The base Layer class
- Layer activations
- Layer weight initializers
- Layer weight regularizers
- Layer weight constraints
- Core layers
- Convolution layers
- Pooling layers
- Recurrent layers
- Preprocessing layers
- Normalization layers
- Regularization layers
- Attention layers
- Reshaping layers
- Merging layers
- Locally-connected layers
- Activation layers



Core layers

Input object
Dense layer
Activation layer
Embedding layer
Masking layer
Lambda layer

Convolution layers

Conv1D layer
Conv2D layer
Conv3D layer
SeparableConv1D layer
SeparableConv2D layer
DepthwiseConv2D layer
Conv2DTranspose layer
Conv3DTranspose layer

Pooling layers

MaxPooling1D layer
MaxPooling2D layer
MaxPooling3D layer
AveragePooling1D layer
AveragePooling2D layer
AveragePooling3D layer
GlobalMaxPooling1D layer
GlobalMaxPooling2D layer
GlobalMaxPooling3D layer
GlobalAveragePooling1D layer
GlobalAveragePooling2D layer
GlobalAveragePooling3D layer

Слои

- The base Layer class
- Layer activations
- Layer weight initializers
- Layer weight regularizers
- Layer weight constraints
- Core layers
- Convolution layers
- Pooling layers
- Recurrent layers
- Preprocessing layers
- Normalization layers
- Regularization layers
- Attention layers
- Reshaping layers
- Merging layers
- Locally-connected layers
- Activation layers

Recurrent layers

LSTM layer
GRU layer
SimpleRNN layer
TimeDistributed layer
Bidirectional layer
ConvLSTM1D layer
ConvLSTM2D layer
ConvLSTM3D layer
Base RNN layer

Preprocessing layers

Text preprocessing
Numerical features preprocessing layers
Categorical features preprocessing layers
Image preprocessing layers
Image augmentation layers

Regularization layers

Dropout layer
SpatialDropout1D layer
SpatialDropout2D layer
SpatialDropout3D layer
GaussianDropout layer
GaussianNoise layer
ActivityRegularization layer
AlphaDropout layer

Normalization layers

BatchNormalization layer
LayerNormalization layer

Слои

- The base Layer class
- Layer activations
- Layer weight initializers
- Layer weight regularizers
- Layer weight constraints
- Core layers
- Convolution layers
- Pooling layers
- Recurrent layers
- Preprocessing layers
- Normalization layers
- Regularization layers
- Attention layers
- Reshaping layers
- Merging layers
- Locally-connected layers
- Activation layers

Reshaping layers

Reshape layer
Flatten layer
RepeatVector layer
Permute layer
Cropping1D layer
Cropping2D layer
Cropping3D layer
UpSampling1D layer
UpSampling2D layer
UpSampling3D layer
ZeroPadding1D layer
ZeroPadding2D layer
ZeroPadding3D layer

Merging layers

Concatenate layer
Average layer
Maximum layer
Minimum layer
Add layer
Subtract layer
Multiply layer
Dot layer

Activation layers

ReLU layer
Softmax layer
LeakyReLU layer
PReLU layer
ELU layer
ThresholdedReLU layer

Методы

- summary method
- get_layer method
- add method …

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(8, input_shape=(16,) ))
model.add(tf.keras.layers.Dense(1))
```

```
Model.summary( line_length=None,
positions=None, print_fn=None,
expand_nested=False,
show_trainable=False, )
```

- compile method
- fit method
- evaluate method
- predict method
- …
- save method
- save_model function
- load_model function
- get_weights method
- …

```
Model.compile(
optimizer="rmsprop",
loss=None, metrics=None,
loss_weights=None,
weighted_metrics=None,
run_eagerly=None,
steps_per_execution=Non
e, jit_compile=None,
**kwargs )
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(le
arning_rate=1e-3),
loss=tf.keras.losses.BinaryCrossentropy(),
metrics=[tf.keras.metrics.BinaryAccuracy(),
tf.keras.metrics.FalseNegatives()])
```

Методы

- summary method
- get_layer method
- add method …

- compile method
- fit method
- evaluate method
- predict method
- …
- save method
- save_model function
- load_model function
- get_weights method
- …

```
Model.fit( x=None, y=None,
batch_size=None, epochs=1,
verbose="auto", 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, )
```

Returns

History object.

Методы

- summary method
- get_layer method
- add method …

- compile method
- fit method
- evaluate method
- predict method
- …
- save method
- save_model function
- load_model function
- get_weights method
- …

```
Model.evaluate(
x=None,
y=None,
batch_size=None,
verbose=1,
sample_weight=None,
steps=None,
callbacks=None,
max_queue_size=10,
workers=1,
use_multiprocessing=False,
return_dict=False,
**kwargs )
```

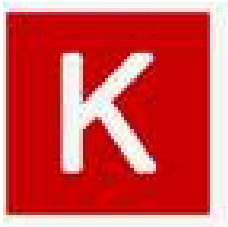
Методы

- summary method
- get_layer method
- add method …

- compile method
- fit method
- evaluate method
- predict method
- …
- save method
- save_model function
- load_model function
- get_weights method
- …

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

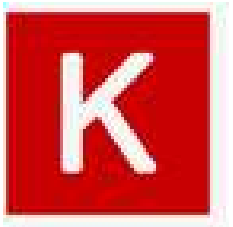
Оптимизаторы



- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam
- Ftrl

```
opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=opt)
```

Метрики



Accuracy metrics

- Accuracy class
- BinaryAccuracy class
- CategoricalAccuracy class
- SparseCategoricalAccuracy class
- TopKCategoricalAccuracy class
- SparseTopKCategoricalAccuracy class

Probabilistic metrics

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- KLDivergence class
- Poisson class

Regression metrics

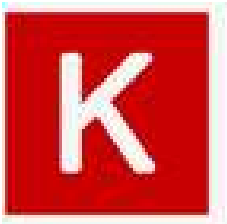
- MeanSquaredError class
- RootMeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- LogCoshError class

Classification metrics based on True/False positives & negatives

- AUC class
- Precision class
- Recall class
- TruePositives class
- TrueNegatives class
- FalsePositives class
- FalseNegatives class
- PrecisionAtRecall class
- SensitivityAtSpecificity class
- SpecificityAtSensitivity class

```
model.compile( optimizer='adam',
loss='mean_squared_error', metrics=[
metrics.MeanSquaredError(), metrics.AUC(), ] )
```

Функции потерь (loss)



Probabilistic losses

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- binary_crossentropy function
- categorical_crossentropy function
- sparse_categorical_crossentropy function
- poisson function
- KLDivergence class
- kl_divergence function

Regression losses

- MeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- mean_squared_error function
- mean_absolute_error function
- mean_absolute_percentage_error function
- mean_squared_logarithmic_error function
- cosine_similarity function
- Huber class
- huber function
- LogCosh class
- log_cosh function

Обучающие выборки

- MNIST digits classification dataset

  load_data function
- CIFAR10 small images classification dataset

  load_data function
- CIFAR100 small images classification dataset

  load_data function
- IMDB movie review sentiment classification dataset

  load_data function

  get_word_index function
- Reuters newswire classification dataset

  load_data function

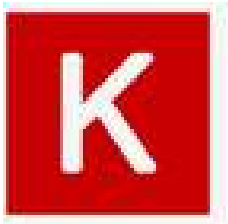
  get_word_index function
- Fashion MNIST dataset, an alternative to MNIST

  load_data function
- Boston Housing price regression dataset

  load_data function

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
assert x_train.shape == (60000, 28, 28)
assert x_test.shape == (10000, 28, 28)
assert y_train.shape == (60000,)
 assert y_test.shape == (10000,)
```

Библиотеки для работы с НС

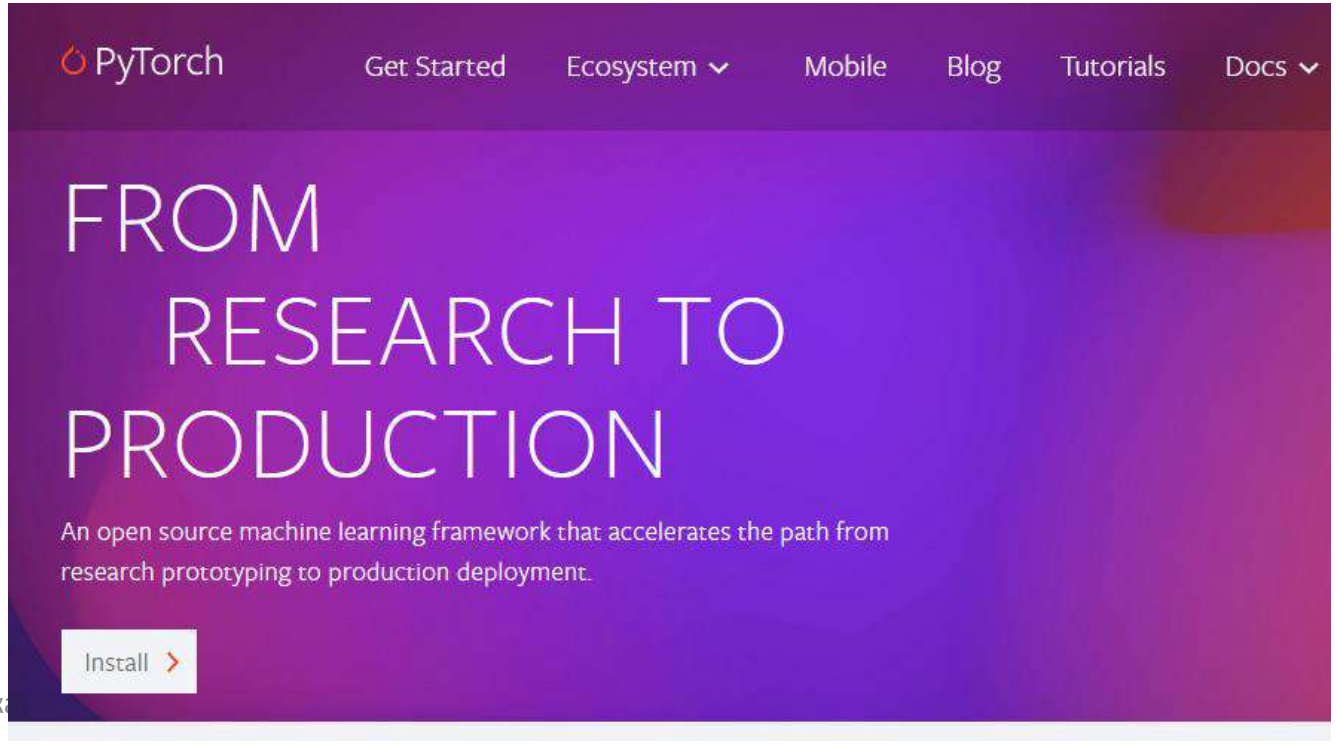


- Примеры работы – практические задания
- Матрица ошибок (confusion matrix)

Библиотеки для работы с НС

- PyTorch (https://pytorch.org/)

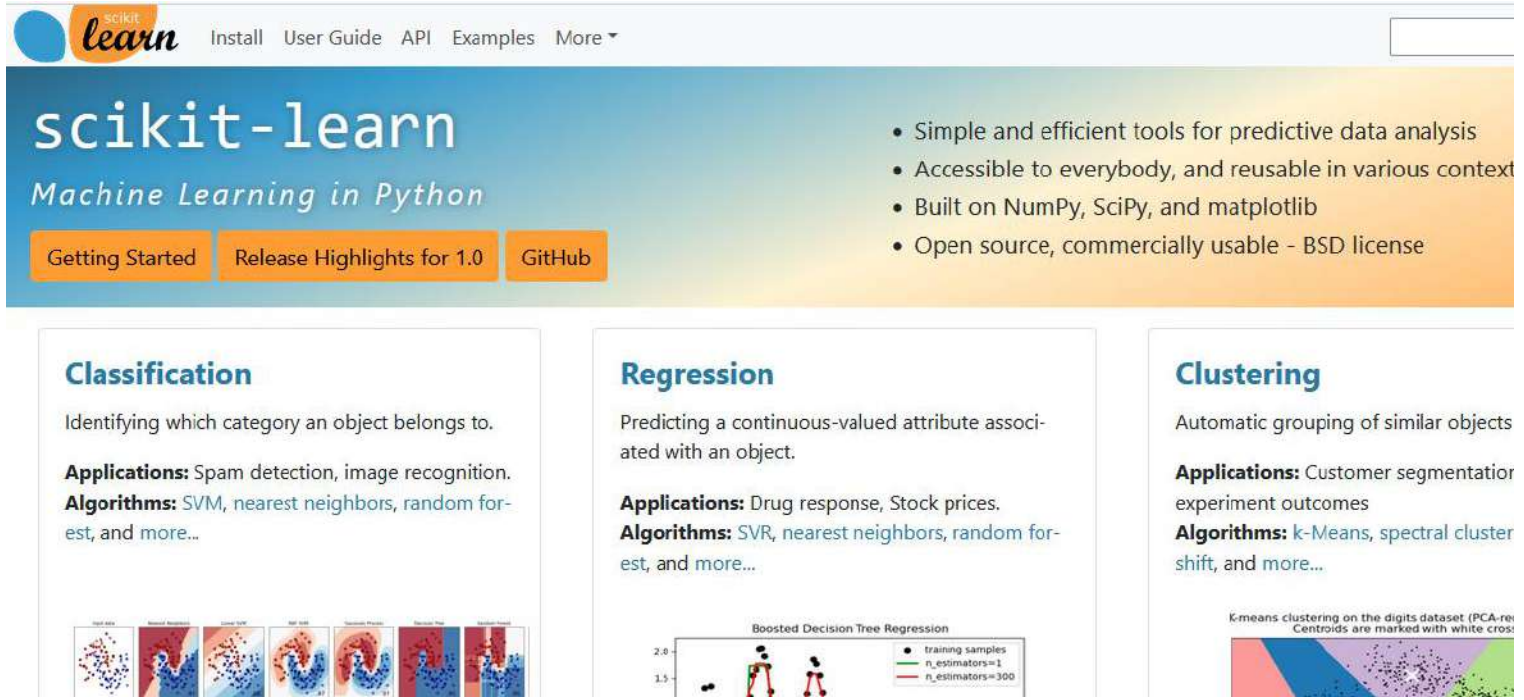
Фреймворк машинного обучения для языка Python с открытым исходным кодом, созданный на базе Torch. Используется для решения различных задач: компьютерное зрение, обработка естественного языка. Разрабатывается преимущественно группой искусственного интеллекта Facebook. Также вокруг этого фреймворка выстроена экосистема, состоящая из различных библиотек, разрабатываемых сторонними командами: PyTorch Lightning и Fast.ai, упрощающие процесс обучения моделей, Pyro, модуль для вероятностного программирования, от Uber, Flair, для обработки естественного языка и Catalyst, для обучения DL и RL моделей.
Год выхода: 2016



Библиотеки для работы с НС

- Scikit-learn (scikit-learn.org)

Бесплатная библиотека программного обеспечения для машинного обучения для языка программирования Python.
Год выхода: 2007



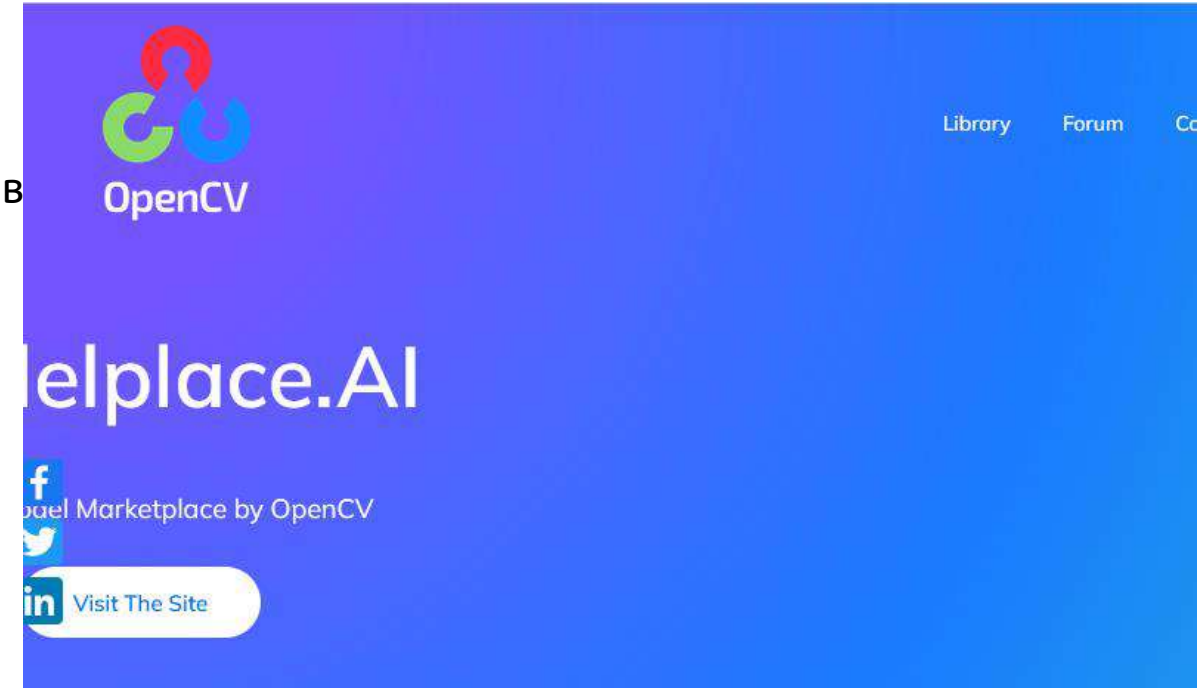
Библиотеки для работы с НС

ensemble.AdaBoostClassifier([...])	An AdaBoost classifier.
ensemble.AdaBoostRegressor([base_estimator, ...])	An AdaBoost regressor.
ensemble.BaggingClassifier([base_estimator, ...])	A Bagging classifier.
ensemble.BaggingRegressor([base_estimator, ...])	A Bagging regressor.
ensemble.ExtraTreesClassifier([...])	An extra-trees classifier.
ensemble.ExtraTreesRegressor([n_estimators, ...])	An extra-trees regressor.
ensemble.GradientBoostingClassifier(*[, ...])	Gradient Boosting for classification.
ensemble.GradientBoostingRegressor(*[, ...])	Gradient Boosting for regression.
ensemble.IsolationForest(*[, n_estimators, ...])	Isolation Forest Algorithm.
ensemble.RandomForestClassifier([...])	A random forest classifier.
ensemble.RandomForestRegressor([...])	A random forest regressor.
ensemble.RandomTreesEmbedding([...])	An ensemble of totally random trees.
ensemble.StackingClassifier(estimators[, ...])	Stack of estimators with a final classifier.
ensemble.StackingRegressor(estimators[, ...])	Stack of estimators with a final regressor.
ensemble.VotingClassifier(estimators, *[, ...])	Soft Voting/Majority Rule classifier for unfitted estimators.
ensemble.VotingRegressor(estimators, *[, ...])	Prediction voting regressor for unfitted estimators.
ensemble.HistGradientBoostingRegressor([...])	Histogram-based Gradient Boosting Regression Tree.
ensemble.HistGradientBoostingClassifier([...])	Histogram-based Gradient Boosting Classification Tree.

Библиотеки для работы с НС

- OpenCV (https://opencv.org/)

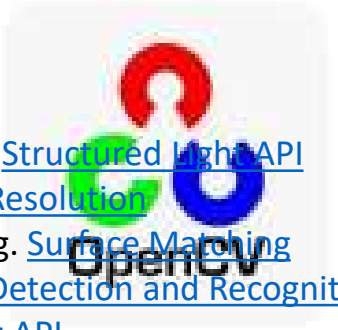
Библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Реализована на C/C++, также разрабатывается для Python, Java, Ruby, Matlab, Lua и других языков. Может свободно использоваться в академических и коммерческих целях - распространяется в условиях лицензии BSD.
Год выхода: 2000



Библиотеки для работы с НС

Main modules:
•core. Core functionality
•imgproc. Image Processing
•imgcodecs. Image file reading and writing
•videoio. Video I/O
•highgui. High-level GUI
•video. Video Analysis
•calib3d. Camera Calibration and 3D Reconstruction
•features2d. 2D Features Framework
•objdetect. Object Detection
•dnn. Deep Neural Network module
•ml. Machine Learning
•flann. Clustering and Search in Multi-Dimensional Spaces
•photo. Computational Photography
•stitching. Images stitching
•gapi. Graph API

Библиотеки для работы с НС



•Extra modules: alphamat. Alpha Matting
•aruco. ArUco Marker Detection
•barcode. Barcode detecting and decoding methods
•bgsegm. Improved Background-Foreground Segmentation Methods
•bioinspired. Biologically inspired vision models and derivated tools
•ccalib. Custom Calibration Pattern for 3D reconstruction
•cudaarithm. Operations on Matrices
•cudabgsegm. Background Segmentation
•cudacodec. Video Encoding/Decoding
•cudafeatures2d. Feature Detection and Description
•cudafilters. Image Filtering
•cudaimgproc. Image Processing
•cudalegacy. Legacy support
•cudaobjdetect. Object Detection
•cudaoptflow. Optical Flow
•cudastereo. Stereo Correspondence
•cudawarping. Image Warping
•cudev. Device layer
•cvv. GUI for Interactive Visual Debugging of Computer Vision Programs
•datasets. Framework for working with different datasets
•dnn_objdetect. DNN used for object detection
•dnn_superres. DNN used for super resolution
•face. Face Analysis
•freetype. Drawing UTF-8 strings with freetype/harfbuzz
•fuzzy. Image processing based on fuzzy mathematics
•hdf. Hierarchical Data Format I/O routines
•hfs. Hierarchical Feature Selection for Efficient Image Segmentation
•img_hash. The module brings implementations of different image hashing algorithms.
•intensity_transform. The module brings implementations of intensity transformation algorithms to adjust image contrast.
•julia. Julia bindings for OpenCV
•line_descriptor. Binary descriptors for lines extracted from an image
•mcc. Macbeth Chart module
•optflow. Optical Flow Algorithms
•ovis. OGRE 3D Visualiser
•phase_unwrapping. Phase Unwrapping API
•plot. Plot function for Mat data
•quality. Image Quality Analysis (IQA) API
•rapid. silhouette based 3D object tracking
•reg. Image Registration
•rgbd. RGB-Depth Processing
•saliency. Saliency API
•sfm. Structure From Motion
•shape. Shape Distance and Matching
•structured_light. Structured Light API
•superres. Super Resolution
•surface_matching. Surface Matching
•text. Scene Text Detection and Recognition
•tracking. Tracking API
•videostab. Video Stabilization
•viz. 3D Visualizer
•wechat_qrcode. WeChat QR code detector for detecting and parsing QR code.
•xfeatures2d. Extra 2D Features Framework
•ximgproc. Extended Image Processing
•xobjdetect. Extended object detection
•xphoto. Additional photo processing algorithms

Библиотеки для работы с нейронными сетями

- Classes
- class cv::ml::ANN_MLP
- Artificial Neural Networks - Multi-Layer Perceptrons.
- class cv::ml::Boost
- Boosted tree classifier derived from DTrees.
- class cv::ml::DTrees
- The class represents a single decision tree or a collection of decision trees.
- class cv::ml::EM
- The class implements the Expectation Maximization algorithm.
- class cv::ml::KNearest
- The class implements K-Nearest Neighbors model.
- class cv::ml::LogisticRegression
- Implements Logistic Regression classifier.
- class cv::ml::NormalBayesClassifier
- Bayes classifier for normally distributed data.
- class cv::ml::ParamGrid
- The structure represents the logarithmic grid range of statmodel parameters.
- class cv::ml::RTrees
- The class implements the random forest predictor.
- struct cv::ml::SimulatedAnnealingSolverSystem
- This class declares example interface for system state used in simulated annealing optimization algorithm.
- class cv::ml::StatModel
- Base class for statistical models in OpenCV ML.
- class cv::ml::SVM
- Support Vector Machines.
- class cv::ml::SVMSGD
- Stochastic Gradient Descent SVM classifier.
- class cv::ml::TrainData
- Class encapsulating training data.

Основы нейронных сетей

Лекция 3

Процессы обучения нейронных сетей

Процессы обучения нейронных сетей

- Парадигмы обучения
- Классификация алгоритмов обучения
- Задачи обучения
- Представление знаний

Парадигмы обучения

Обучение - это процесс, в котором свободные параметры НС настраиваются посредством моделирования среды.
Тип обучения определяется способом подстройки параметров

Парадигмы обучения

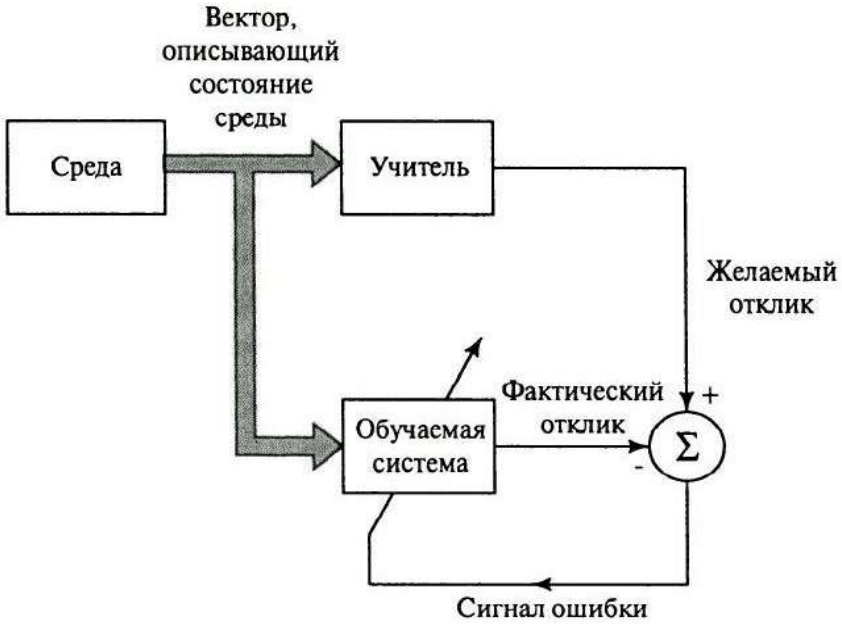
Алгоритм обучения

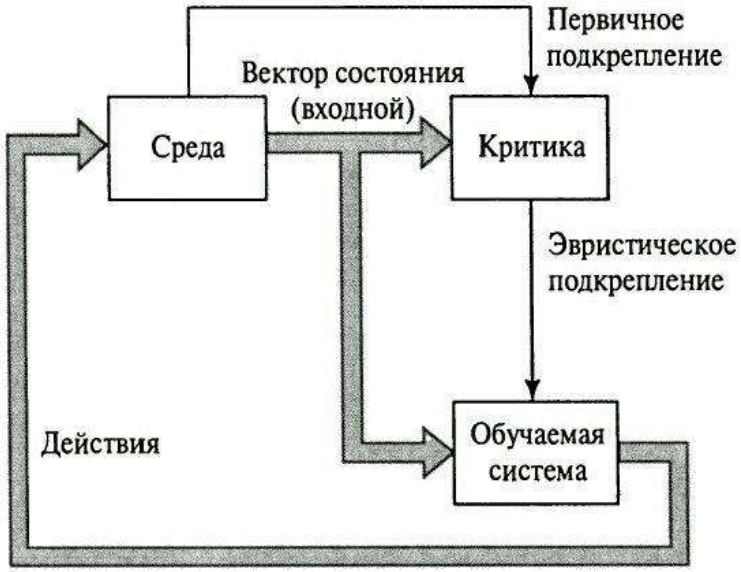
1. В сеть поступают примеры
2. Изменение параметров
3. Сеть отвечает другим образом

Не существует универсального алгоритма обучения

Способ связи с внешним миром

- обучение с учителем
- без учителя
- с подкреплением

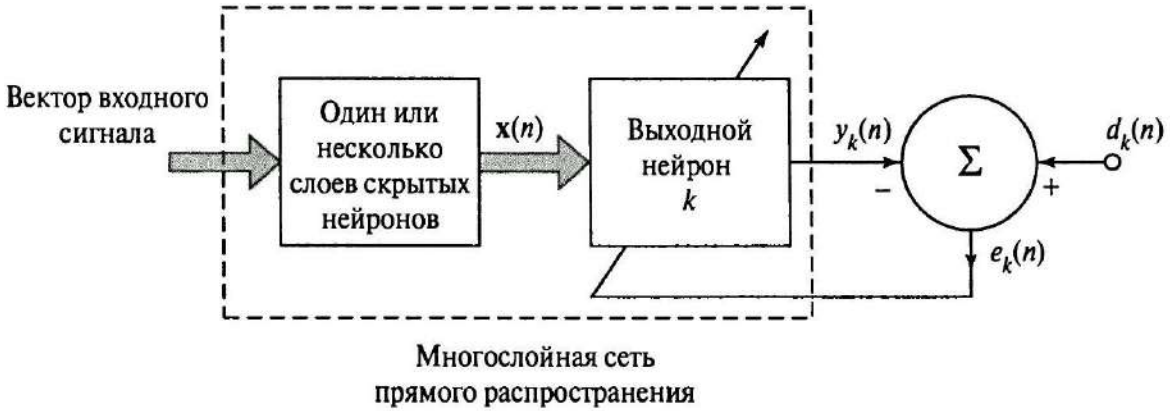




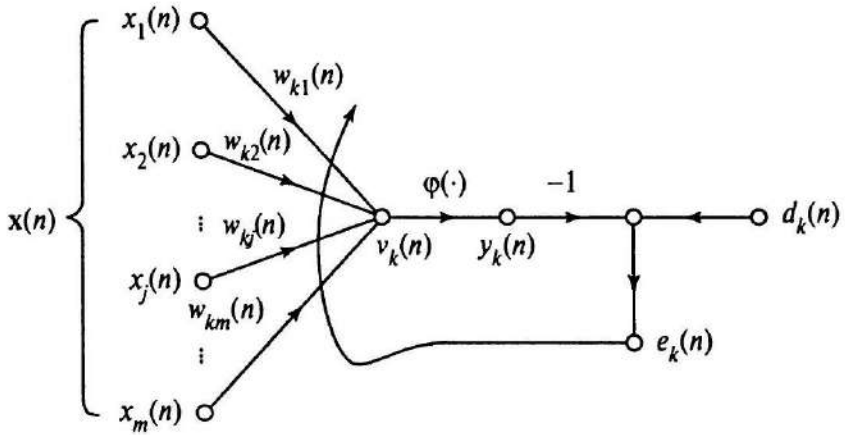
Классификация алгоритмов обучения

- по способу настройки синоптических весов:
  - Обучение, основанное на коррекции ошибок
  - Обучение на основе памяти
  - Хеббовское обучение
  - Конкурентное обучение
  - …

Обучение, основанное на коррекции ошибок



а) Блочная диаграмма нейронной сети;
показаны только нейроны выходного слоя



б) Граф передачи сигнала выходного нейрона

Обучение, основанное на коррекции ошибок

Каждый образец подается на входы сети, затем проходит обработку внутри структуры НС, вычисляется выходной сигнал сети, который сравнивается с соответствующим значением целевого вектора, представляющего собой требуемый выход сети. Затем по определенному правилу вычисляется ошибка, и происходит изменение весовых коэффициентов связей внутри сети в зависимости от выбранного алгоритма. Векторы обучающего множества предъявляются последовательно, вычисляются ошибки и веса подстраиваются для каждого вектора до тех пор, пока ошибка по всему обучающему массиву не достигнет приемлемо низкого уровня.

$$e_k(n) = d_k(n) - y_k(n).$$

Обучение, основанное на коррекции ошибок

Дельта-правило (правило В(У)идроу-Хоффа)
Корректировка, применяемая к синаптическому весу нейрона, пропорциональна произведению сигнала ошибки на входной сигнал, его вызвавший

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n),$$

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n).$$

Модификации!

Обучение, основанное на памяти

При *обучении на основе памяти* (memory-based learning) весь прошлый опыт накапливается в большом хранилище правильно классифицированных примеров вида вход-выход: $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$, где $\mathbf{x}_i$ — входной вектор, а d_i — соответствующий ему желаемый выходной сигнал.

Если требуется классифицировать некоторый неизвестный вектор $\mathbf{x}_{\text{test}}$, из базы данных выбирается выход, соответствующий входному сигналу, близкому к $\mathbf{x}_{\text{test}}$.

Все алгоритмы обучения на основе памяти включают в себя две существенные составляющие.

- Критерий, используемый для определения окрестности вектора $\mathbf{x}_{\text{test}}$.
- Правило обучения, применяемое к примеру из окрестности тестового вектора.

Все алгоритмы отличаются друг от друга способом реализации этих двух составляющих.

Обучение, основанное на памяти

$$\mathbf{x}'_N \in \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$$

считается ближайшим соседом вектора $\mathbf{x}_{\text{test}}$, если выполняется условие

$$\min_i \; d(\mathbf{x}_i, \; \mathbf{x}_{\text{test}}) = \; d(\mathbf{x}'_N, \mathbf{x}_{\text{test}}),$$

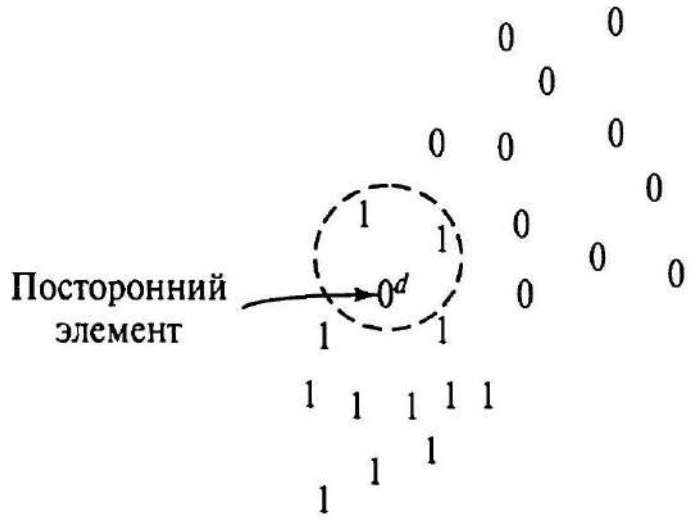
где $d(\mathbf{x}_i, \; \mathbf{x}_{\text{test}})$ — Евклидово расстояние между векторами $\mathbf{x}_i$ и $\mathbf{x}_{\text{test}}$. Класс, к которому относится ближайший сосед, считается также классом тестируемого вектора $\mathbf{x}_{\text{test}}$. Это правило не зависит от распределения, используемого при генерировании примеров обучения.

Обучение, основанное на памяти

Вариацией классификатора на основе ближайшего соседа является *классификатор k-ближайших соседей* (k-nearest neighbor classifier). Он описывается следующим образом.

Находим k классифицированных соседей, ближайших к вектору $\mathbf{x}_{test}$, где k — некоторое целое число.

Вектор $\mathbf{x}_{\text{test}}$ относим к тому классу (гипотезе), который чаще других встречается среди k-ближайших соседей тестируемого вектора.



Обучение Хебба

Цитата

Если аксон клетки A находится на достаточно близком расстоянии от клетки B и постоянно или периодически участвует в ее возбуждении, наблюдается процесс метаболических изменений в одном или обоих нейронах, выражающийся в том, что эффективность нейрона A как одного из возбудителей нейрона B возрастает.

1. *Если два нейрона по обе стороны синапса (соединения) активизируются одновременно (т.е. синхронно), то прочность этого соединения возрастает.*
2. *Если два нейрона по обе стороны синапса активизируются асинхронно, то такой синапс ослабляется или вообще отмирает.*

Обучение Хебба 4 свойства

- Зависимость от времени;
- Локальность;
- Интерактивность (нельзя использовать только один сигнал (пред или пост));
- Корреляция (между пред и постсинаптическим сигналом)

Обучение Хебба. Обобщение

- Хеббовские модели → Синаптическая связь усиливается при положительной корреляции и ослабляется в противном случае
- Антихеббовские → Синаптическая связь ослабляется при положительной корреляции и усиливается в противном случае
- Нехеббовские (не связаны с обработкой по Хеббу)

Обучение Хебба

$$\Delta w_{kj}(n) = F(y_k(n), x_j(n)),$$

Общий вид

Гипотеза Хебба

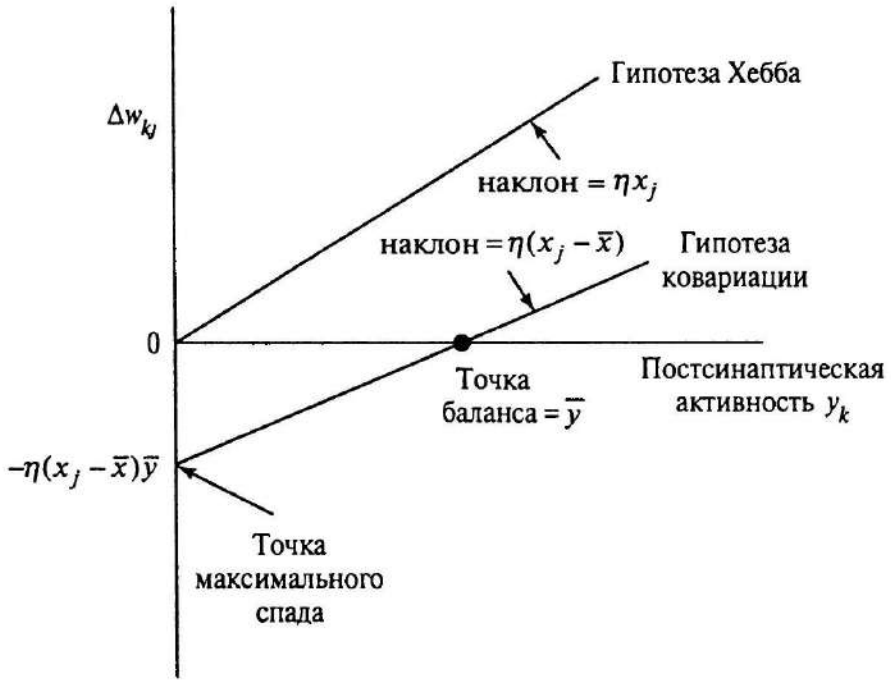
$$\Delta w_{kj}(n) = \eta y_k(n) x_j(n),$$

где η — положительная константа, определяющая *скорость обучения*

Гипотеза ковариации

Обозначим символами $\bar{x}$ и $\bar{y}$ *усредненные по времени значения* предсинаптического x_j и постсинаптического y_k сигналов. Согласно гипотезе ковариации, изменение синаптического веса w_{kj} вычисляется по формуле

$$\Delta w_{kj} = \eta(x_j - \bar{x})(y_k - \bar{y}),$$



чение Хебба

1. Синаптический вес связи усиливается при высоком уровне предсинаптического и постсинаптического сигналов, т.е. в том случае, когда одновременно удовлетворяются следующие условия: $x_j > \bar{x}$ и $y_k > \bar{y}$.
2. Синаптический вес ослабляется в следующих ситуациях.
   - Предсинаптическая активность (т.е. $x_j > \bar{x}$) не вызывает существенной постсинаптической активности $(y_k < \bar{y})$.
   - Постсинаптическая активность $(y_k > \bar{y})$ возникает при отсутствии существенной предсинаптической активности $(x_j < \bar{x})$.

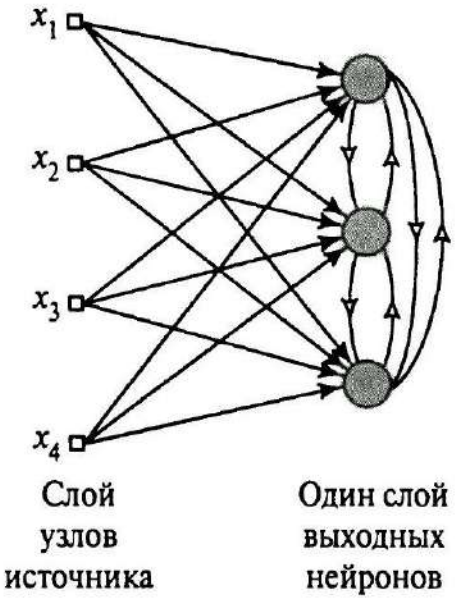
Такое поведение можно рассматривать как форму временной конкуренции между входными моделями.

Конкурентное обучение

WTA

Правило конкурентного обучения основано на использовании трех основных элементов

- Множество одинаковых нейронов со случайно распределенными синаптическими весами, приводящими к *различной* реакции нейронов на один и тот же входной сигнал.
- *Предельное значение* (limit) "силы" каждого нейрона.
- Механизм, позволяющий нейронам *конкурировать* за право отклика на данное подмножество входных сигналов и определяющий единственный активный выходной нейрон (или по одному нейрону на группу). Нейрон, победивший в этом соревновании, называют *нейроном-победителем*, а принцип конкурентного обучения формулируют в виде лозунга "победитель забирает все" (*winner-takes-all*).



WTM

Конкурентное обучение

$$y_k = \begin{cases} 1, & \text{если } v_k > v_j \text{ для всех } \ j, j \neq k, \\ 0 & \text{в остальных случаях,} \end{cases}$$

$$\sum_j w_{kj} = 1 \ \text{ для всех } \ k.$$

Тогда обучение этого нейрона состоит в смещении синаптических весов от неактивных к активным входным узлам. Если нейрон не формирует отклика на конкретный входной образ, то он и не обучается. Если некоторый нейрон выигрывает в конкурентной борьбе, то веса связей этого нейрона равномерно распределяются между его активными входными узлами, а связи с неактивными входными узлами ослабляются. Согласно *правилу конкурентного обучения* (competitive learning rule) изменение Δw_{kj} синаптического весаw_{kj} определяется следующим выражением:

$$\Delta w_{kj} = \begin{cases} \eta(x_j - w_{kj}), & \text{если нейрон } \ k \text{ побеждает в соревновании,} \\ 0, & \text{если нейрон } \ k \text{ побеждает в соревновании,} \end{cases}$$

- Еще один распространенный подход – это обучение нейронной сети генетическим алгоритмом (Genetic Algorithm). По своему принципу он схож с эволюционными процессами природы, которые основываются на комбинировании (скрещивании) результатов. Другими словами, происходит естественный отбор, где новое поколение является продуктом комбинации результатов с самыми лучшими свойствами. Если итог такого скрещивания не подходит по каким-то критериям, то отбор совершается вновь, пока продукт не станет совершенным.

Машина Больцмана

В машине Больцмана все нейроны представляются рекуррентными структурами, работающими с бинарными сигналами. Это значит, что они могут находиться во включенном (соответствующем значению +1) или выключенном (соответствующем значению -1) состоянии. Такая машина характеризуется функцией энергии E, значение которой определяется конкретными состояниями отдельных нейронов, составляющих эту машину. Это можно описать следующим выражением:

$$E = -\frac{1}{2}\sum_{j}\sum_{k(j\neq k)} w_{kj}x_k x_j,$$

где x_j — состояние нейрона j; w_{kj} — синаптический вес связи нейронов j и k. Условие $j \neq k$ подчеркивает тот факт, что в этой сети нейроны не имеют обратных связей с самими собой. Работа этой машины заключается в случайном выборе некоторого нейрона (предположим, k-го) на определенном шаге процесса обучения и переводе этого нейрона из состояния x_k в состояние $-x_k$ при некоторой температуре T с вероятностью

Машина Больцмана

$$P(x_k \to -x_k) = \frac{1}{1 + \exp(-\Delta E_k / T)},$$

Нейроны машины Больцмана можно разбить на две функциональные группы: (visible) и *скрытые* (hidden). Видимые нейроны реализуют интерфейс между сетью и средой ее функционирования, а скрытые работают независимо от внешней среды. Рассмотрим два режима функционирования такой сети.

- *Скованное состояние* (clamped condition), в котором все видимые нейроны находятся в состояниях, предопределенных внешней средой.
- *Свободное состояние* (free-running condition), в котором все нейроны (как видимые, так и скрытые) могут свободно функционировать.

Обозначим ρ_{kj}^{+} *корреляцию* между состояниями нейронов j и k в скованном состоянии. Аналогично, ρ_{kj}^{-} обозначим *корреляцию* (correlation) между состояниями нейронов j и k, когда сеть находится в свободном состоянии. Обе эти корреляции усредняются по всем возможным состояниям машины, находящейся в условиях термального равновесии. Затем, согласно *правилу обучения Больцмана* (Boltzmann learning rule), изменение Δw_{kj} синаптического весаw$_{kj}$ связи между нейронами k и j определяется следующим выражением :

$$\Delta w_{kj} = \eta(\rho_{kj}^{+} - \rho_{kj}^{-}), \qquad j \neq k,$$

где η — параметр скорости обучения. Заметим, что значения ρ_{kj}^{+} и ρ_{kj}^{-} изменяются в диапазоне от -1 до $+1$.

Классификация алгоритмов обучения

Классификация алгоритмов обучения

- Математически процесс обучения можно описать следующим образом. В процессе функционирования нейронная сеть формирует выходной сигнал Y, реализуя некоторую функцию Y = G(X). Если архитектура сети задана, то вид функции G определяется значениями синаптических весов и смещенной сети.
- Пусть решением некоторой задачи является функция Y = F(X), заданная параметрами входных-выходных данных (X^1, Y^1), (X^2, Y^2), …, (X^N, Y^N), для которых $Y^k = F(X^k)$ (k = 1, 2, …, N).
- Обучение состоит в поиске (синтезе) функции G, близкой к F в смысле некторой функции ошибки E.
- Если выбрано множество обучающих примеров – пар (X^N, Y^N) (где k = 1, 2, …, N) и способ вычисления функции ошибки E, то обучение нейронной сети превращается в задачу многомерной оптимизации, имеющую очень большую размерность, при этом, поскольку функция E может иметь произвольный вид обучение в общем случае – многоэкстремальная невыпуклая задача оптимизации.

Классификация алгоритмов обучения

Для решения этой задачи могут использоваться следующие (итерационные) алгоритмы:

- алгоритмы локальной оптимизации с вычислением частных производных первого порядка:
  - градиентный алгоритм (метод наискорейшего спуска),
  - методы с одномерной и двумерной оптимизацией целевой функции в направлении антиградиента,
  - метод сопряженных градиентов,
  - методы, учитывающие направление антиградиента на нескольких шагах алгоритма;
- алгоритмы локальной оптимизации с вычислением частных производных первого и второго порядка:
  - метод Ньютона,
  - методы оптимизации с разреженными матрицами Гессе,
  - квазиньютоновские методы,
  - метод Гаусса-Ньютона,
  - метод Левенберга-Марквардта и др.;
- стохастические алгоритмы оптимизации:
  - поиск в случайном направлении,
  - имитация отжига,
  - метод Монте-Карло (численный метод статистических испытаний);
- алгоритмы глобальной оптимизации (задачи глобальной оптимизации решаются с помощью перебора значений переменных, от которых зависит целевая функция).

Задачи

1. АП
2. Распознавание образов (классификация, извлечение признаков)
3. Апроксимация функции
4. Фильтрация (прогнозирование, восстановление, сглаживание)
5. Управление
6. Генерация

Обучение без учителя применяют для кластеризации, языковых моделей, обнаружения аномалий, статистических моделей.

Представление знаний

Знания - хранимая информация или модели, используемые человеком или компьютером для предсказания, интерпретации и реакции на внешние события

Вопросы:

- какую информацию нужно хранить
- как ее представить для дальнейшего использования

Представление знаний

Знания о мире:
- априорная информация (известное состояние окружающего мира в виде фактов)
- измерения, полученные сенсорами в конкретных условиях, в которых должна функционирование НС, возможно с шумом. Примеры из этого множества используются для обучения

Представление знаний

Примеры:
- маркированные (вход+желаемый отклик) - обучающая выборка
- немаркированные (несколько реализаций одного входного сигнала)

Представление знаний. Этапы создания сети

1. Выбор архитектуры сети (априорная информация + задача + предметная область)
2. Обучение
3. Тестирование (эффективность обучения) - обобщение
4. Работа обученной сети

Представление знаний

- Знания в сети хранятся множеством синаптических весов
- Обучающая выборка должна содержать и отрицательные примеры

Можно выделить 4 правила:

Представление знаний

Пр1. Сходные входные сигналы от схожих классов должны формировать единообразное представление в сети, тогда они должны быть классифицированы к одному классу

Степень схожести

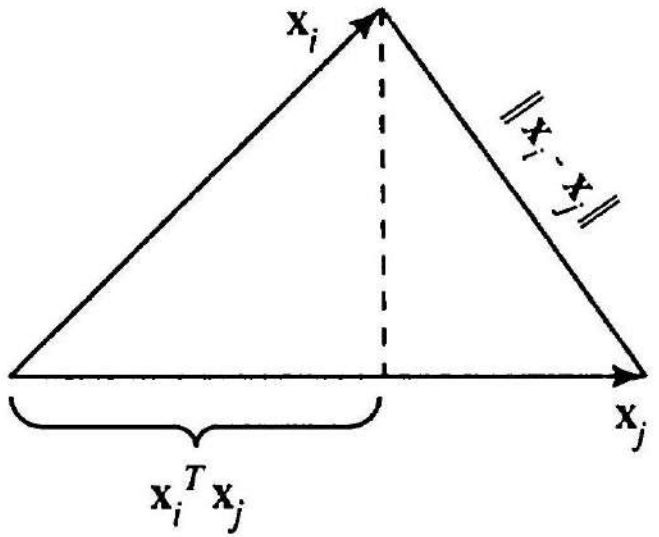
$$d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\| = \left[\sum_{k=1}^{m} (x_{ik} - x_{jk})^2\right]^{1/2}$$

$$(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j = \sum_{k=1}^{m} x_{ik} x_{jk}.$$

Представление знаний

Чтобы формализовать это соотношение, нормализуем векторы $\mathbf{x}_i$ и $\mathbf{x}_j$. длина будет равна единице: $||\mathbf{x}_i|| = ||\mathbf{x}_j|| = 1$.

$$d^2(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j) = 2 - 2\mathbf{x}_i^T\mathbf{x}_j.$$



Представление знаний

Здесь Евклидово расстояние и скалярное произведение описаны в детерминистских терминах. А что будет, если векторы $\mathbf{x}_i$ и $\mathbf{x}_j$ взять из разных множеств данных? Для примера предположим, что различие между двумя множествами данных выражается в различии между векторами их математического ожидания. Пусть векторы $\boldsymbol{\mu}_i$ и $\boldsymbol{\mu}_j$ определяют средние значения векторов $\mathbf{x}_i$ и $\mathbf{x}_j$ соответственно, т.е.

$$\boldsymbol{\mu}_i = E[\mathbf{x}_i],$$

где E — статистический оператор математического ожидания. Вектор $\boldsymbol{\mu}_j$ определяется аналогичным способом. Для измерения расстояния между двумя множествами можно использовать *расстояние Махаланобиса* (Mahalanobis distance), которое обозначается как d_{ij}. Квадрат этой величины определяется следующей формулой

$$d_{ij}^2 = (\mathbf{x}_i - \boldsymbol{\mu}_i)^T \Sigma^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_j),$$

где $\boldsymbol{\Sigma}^{-1}$ — обратная матрица для матрицы ковариации $\boldsymbol{\sigma}$. Предполагается, что матрица ковариации для обоих множеств одна и та же, т.е.

$$\boldsymbol{\Sigma} = E[(\mathbf{x}_i - \boldsymbol{\mu}_i)(\mathbf{x}_i - \boldsymbol{\mu}_i)^T] = E[(\mathbf{x}_j - \boldsymbol{\mu}_j)(\mathbf{x}_j - \boldsymbol{\mu}_j^T)].$$

В частном случае, когда $\mathbf{x}_i = \mathbf{x}_j$, $\boldsymbol{\mu}_i = \boldsymbol{\mu}_j = \boldsymbol{\mu}$ и $\boldsymbol{\Sigma} = \mathbf{I}$, где $\mathbf{I}$ — единичная матрица, расстояние Махаланобиса вырождается в Евклидово расстояние между вектором $\mathbf{x}_i$ и вектором математического ожидания $\boldsymbol{\mu}$.

Представление знаний

Пр2. Элементы, отнесенные к различным классам, должны иметь в сети как можно более отличные представления

Представление знаний

Пр3. Важное свойство должно представлено в сети большим количеством нейронов
Задача локации. Ее эффективность измеряется:
-вероятность обнаружения
-вероятность ложной тревоги
Критерий Неймана-Пирсона

Представление знаний

Пр4. В структуру сети должны быть встроены априорная информация и инварианты, что упрощает архитектуру и обучение.
- учитывает специфику зрения, слуха и др.
- меньше параметров, то есть неполносвязная сеть+ регуляризация

встраивание априорной информации

Нет решения проблемы, комбинация 2-х приемов:
1. Ограничение архитектуры с помощью локальных связей (рецепторные поля)
2. Совместное использование весов

Сокращение параметров сети

Специализированные сети

инвариантность

Минимум три приема:
- структурная инвариантность (ресурсоемкость)
- инвариантность по обучению (нет уверенности в сохранении инвариантности относительно объектов, не используемых в обучении + ресурсы)
- использование инвариантных признаков (знание специфики проблемы)

Основы нейронных сетей

Лекция 4

Однослойный персептрон

Постановка задачи

Множество **T** содержит примеры, одинаково распределенные согласно некоторому вероятностному закону. Размерность входного вектора $\mathbf{x}(i)$ называют *размерностью входного пространства* (dimensionality of the input space).

Входной сигнал $\mathbf{x}(i)$ может быть представлен двумя диаметрально противоположными способами — в пространстве и во времени.

- Если m элементов сигнала $\mathbf{x}(i)$ зарождаются в различных точках пространства, то входной сигнал можно считать *моментальным снимком данных* (snapshot of data).
- Если m элементов сигнала $\mathbf{x}(i)$ представляют собой текущее и $(m-1)$ предыдущее значения возбуждения, *снятые через равномерные промежутки времени* (uniformly spaced in time), говорят, что входной сигнал снимается во временной области.

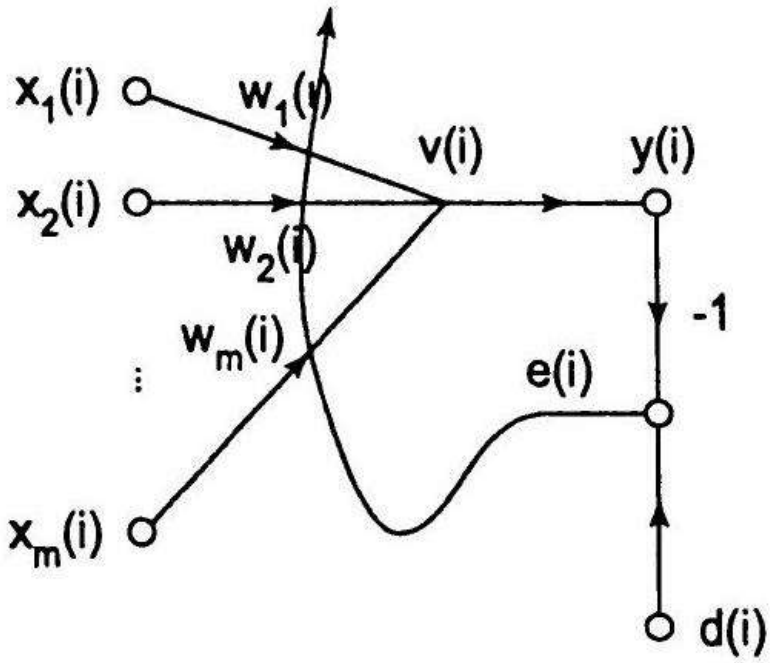
Модель

Требуется построить модель выходного сигнала на основе одного нейрона:
- инициализация - произвольные значения синаптических весов
- настройка весов непрерывно
- вычисление корректирующих значений
- синаптических весов выполняется через равномерные промежутки времени

Такая нейронная модель называется **адаптивным фильтром**

Адаптивный фильтр

Граф прохождения сигнала



В его работе 2 процесса:

Адаптивный фильтр

1. Процесс фильтрации:

- вычисление выходного сигнала на основе входного сигнала

$$y(i) = v(i) = \sum_{k=1}^{m} w_k(i) x_k(i),$$

где $w_1(i), w_2(i), \ldots, w_m(i)$ — m синаптических весов нейрона, измеренных в момент времени i.

- вычисление сигнала ошибки как отклонение выхода системы от целевого ожидаемого сигнала

$$e(i) = y(i) - d(i).$$

Адаптивный фильтр

2. Процесс адаптации - автоматическая настройка синоптических весов на основе сигнала ошибки

Комбинация этих двух процессов называется контуром с обратной связью нейрона
Алгоритм адаптации определяется функцией стоимости, используемой конкретным методом адаптивной фильтрации

Методы безусловной оптимизации

Минимизировать функцию стоимости $\mathbf{E}(\mathbf{w})$ *по отношению к вектору весов* $\mathbf{w}$

$$E(\mathbf{w}) \rightarrow \min .$$

Необходимым условием оптимальности является следующее:

$$\nabla E(\mathbf{w}*) = 0,$$

где ∇ — *оператор градиента*

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots, \frac{\partial}{\partial w_m} \right]^T,$$

а $\nabla \mathbf{E}(\mathbf{w})$ — *вектор градиента* функции стоимости

$$\nabla \mathbf{E}(\mathbf{w}) = \left[\frac{\partial \mathbf{E}}{\partial w_1}, \frac{\partial \mathbf{E}}{\partial w_2}, \ldots, \frac{\partial \mathbf{E}}{\partial w_m} \right]^T.$$

Методы оптимизации

Лучше подходят алгоритмы последовательного спуска:

Начиная с исходного значения $\mathbf{w}(0)$ *генерируется последовательность векторов весовых коэффициентов* $\mathbf{w}(1)$, $\mathbf{w}(2)$, ..., *таких, что с каждой итерацией алгоритма значение функции стоимости уменьшается:*

$$\mathbf{E}(\mathbf{w}(n+1)) < \mathbf{E}(\mathbf{w}(n)),$$

где $\mathbf{w}(n)$ *— предыдущее значение вектора весов;* $\mathbf{w}(n+1)$ *— последующее.*

Метод наискорейшего спуска

Корректировка векторов весов выполняется в направлении макс. уменьшения функции стоимости (противоположном вектору градиента)

$$\Delta \mathbf{w}(n) = \mathbf{w}(n+1) - \mathbf{w}(n) = -\eta \mathbf{g}(n)$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta \mathbf{g}(n),$$

$$\mathbf{g} = \nabla E(\mathbf{w})$$

Метод наискорейшего спуска

$$\mathbf{E}(\mathbf{w}(n+1)) = \mathbf{E}(\mathbf{w}(n)) - \eta \mathbf{g}^T(n)\mathbf{g}(n) = \mathbf{E}(\mathbf{w}(n)) - \eta ||\mathbf{g}(n)||^2.$$

Из этого выражения видно, что для малых значений параметра скорости обучения η значение функции стоимости уменьшается на каждой итерации. Представленное доказательство истинно только для малых значений η.

Метод наискорейшего спуска сходится к оптимальному значению $\mathbf{w}^*$ достаточно медленно. Кроме того, на скорость сходимости влияет значение параметра η.

- Если параметр η мал, алгоритм *замедляется* (overdamped), и траектория изменения $\mathbf{w}(n)$ соответствует гладкой кривой
- Если параметр η велик, алгоритм *ускоряется* (underdamped), и траектория $\mathbf{w}(n)$ принимает зигзагообразный вид
- Если параметр η превосходит некоторое критичное значение, алгоритм становится неустойчивым (т.е. расходящимся).

Метод Ньютона

Идея - минимизации квадратичной аппроксимации E (w)

$$\Delta \mathbf{E}(\mathbf{w}(n)) = \mathbf{E}(\mathbf{w}(n+1)) - \mathbf{E}(\mathbf{w}(n)) =$$
$$= \mathbf{g}^T(n)\Delta\mathbf{w}(n) + 1/2\Delta\mathbf{w}^T(n)\mathbf{H}(n)\Delta\mathbf{w}(n).$$

$$\mathbf{H} = \nabla^2\mathbf{E}(\mathbf{w}) = \begin{bmatrix} \frac{\partial^2\mathbf{E}}{\partial w_1^2} & \frac{\partial^2\mathbf{E}}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2\mathbf{E}}{\partial w_1 \partial w_m} \\ \frac{\partial^2\mathbf{E}}{\partial w_2 \partial w_1} & \frac{\partial^2\mathbf{E}}{\partial w_2^2} & \cdots & \frac{\partial^2\mathbf{E}}{\partial w_2 \partial w_m} \\ \cdots & \cdots & \cdots\cdots & \\ \frac{\partial^2\mathbf{E}}{\partial w_m \partial w_1} & \frac{\partial^2\mathbf{E}}{\partial w_m \partial w_2} & \cdots & \frac{\partial^2\mathbf{E}}{\partial w_m^2} \end{bmatrix}.$$

Метод Ньютона

$$\mathbf{g}(n) + \mathbf{H}(n)\Delta\mathbf{w}(n) = 0.$$

Разрешая это уравнение относительно $\Delta\mathbf{w}(n)$, получим:

$$\Delta\mathbf{w}(n) = -\mathbf{H}^{-1}(n)\mathbf{g}(n).$$

Таким образом,

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta\mathbf{w}(n) = \mathbf{w}(n) - \mathbf{H}^{-1}(n)\boldsymbol{g}(n),$$

где $\mathbf{H}^{-1}(n)$ — матрица, обратная Гессиану.

Метод Ньютона

- быстро сходится
- не приводит к зигзагообразным траектория

Важно:

Матрица Гессе должна быть положительно определенной

$a^T Ha > 0$ для любого вектора a

Метод Гаусса-Ньютона

Метод Гаусса–Ньютона применяется для минимизации функции стоимости, представленной в виде суммы квадратов ошибок. Пусть

$$\mathbf{E}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} e^2(i),$$

Метод Гаусса-Ньютона

$$\mathbf{e}'(n, \mathbf{w}) = \mathbf{e}(n) + \mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n)),$$

где $\mathbf{e}(n)$ — вектор ошибки

$$\mathbf{e}(n) = [e(1), e(2), \ldots, e(m)]^T,$$

$\mathbf{J}(n)$ — *матрица Якобиана* ошибки

$$\mathbf{J}(n) = \left[\begin{array}{cccc} \frac{\partial e(1)}{\partial w_1} & \frac{\partial e(1)}{\partial w_2} & \cdots & \frac{\partial e(1)}{\partial w_m} \\ \frac{\partial e(2)}{\partial w_1} & \frac{\partial e(2)}{\partial w_2} & \cdots & \frac{\partial e(2)}{\partial w_m} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e(n)}{\partial w_1} & \frac{\partial e(n)}{\partial w_2} & \cdots & \frac{\partial e(n)}{\partial w_m} \end{array}\right]_{\mathbf{w}=\mathbf{w}(n)}.$$

Якобиан — это транспонированная матрица градиента $\nabla \mathbf{e}(n)$

$$\nabla E(n) = [\nabla e(1), \nabla e(2), \ldots, \nabla e(n)].$$

Метод Гаусса-Ньютона

Обновленный вектор $\mathbf{w}(n+1)$ можно записать в следующем виде:

$$\mathbf{w}(n+1) = \arg\min_{\mathbf{w}} \left\{ \frac{1}{2} \|\mathbf{e}'(n, \mathbf{w})\|^2 \right\}.$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n)\mathbf{J}(n))^{-1}\mathbf{J}^T(n)\mathbf{e}(n).$$

Метод Гаусса-Ньютона

Исходя из вышесказанного, уравнение метода Гаусса–Ньютона можно записать в несколько видоизмененном виде:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n)\mathbf{J}(n) + \delta\mathbf{I})^{-1}\mathbf{J}^T(n)\mathbf{e}(n).$$

Влияние этой модификации постепенно ослабляется с увеличением количества итераций n. Обратите внимание, что рекурсивное соотношение является решением задачи минимизации *модифицированной* функции стоимости:

$$\mathbf{E}(\mathbf{w}) = \frac{1}{2}\left\{\delta\,\|\mathbf{w} - \mathbf{w}(n)\|^2 + \sum_{i=1}^{n} e^2(i)\right\},$$

где $\mathbf{w}(n)$ — *текущее значение* (current value) вектора весовых коэффициентов $\mathbf{w}(i)$.

ЛФ по методу наименьших квадратов

$$\mathbf{e}(n) = \mathbf{d}(n) - [\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(n)]^T \mathbf{w}(n) = \mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n),$$

$$\mathbf{E}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} e^2(i),$$

Дифференцируя по w (n)

$$\nabla E(n) = -\mathbf{X}^T(n).$$

Следовательно, Якобиан $\mathbf{e}(n)$ можно записать в следующем виде:

$$\mathbf{J}(n) = -\mathbf{X}(n).$$

ЛФ по методу наименьших квадратов

$$\mathbf{w}(n+1) = \mathbf{w}(n) + (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)(\mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n)) =$$
$$= (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)\mathbf{d}(n).$$

Выражение $(\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)$ называют *псевдообратной* матрицей для матрицы данных $\mathbf{X}(n)$ и обозначают следующим образом

$$\mathbf{X}^+(n) = (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n).$$

$$\mathbf{w}(n+1) = \mathbf{X}^+(n)\mathbf{d}(n).$$

Алгоритм минимизации среднеквадратической ошибки LMS

$$\mathbf{E}(\mathbf{w}) = 1/2e^2(n),$$

где $e(n)$ — сигнал ошибки, измеренный в момент времени n. Дифференцируя $\mathbf{E}(\mathbf{w})$ по вектору весов $\mathbf{w}$, получим:

$$\frac{\partial\, \mathbf{E}(\mathbf{w})}{\partial\, \mathbf{w}} = e(n)\frac{\partial\, e(n)}{\partial\, \mathbf{w}}.$$

т. к. линейный нейрон

$$\mathbf{e}(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n).$$

LMS

$$\frac{\partial\, e(n)}{\partial\, \mathbf{w}(n)} = -\mathbf{x}(n),$$

$$\frac{\partial\, \mathbf{E}(\mathbf{w})}{\partial\, \mathbf{w}(n)} = -\mathbf{x}(n)e(n).$$

Используя полученный результат, можно *оценить* вектор градиента

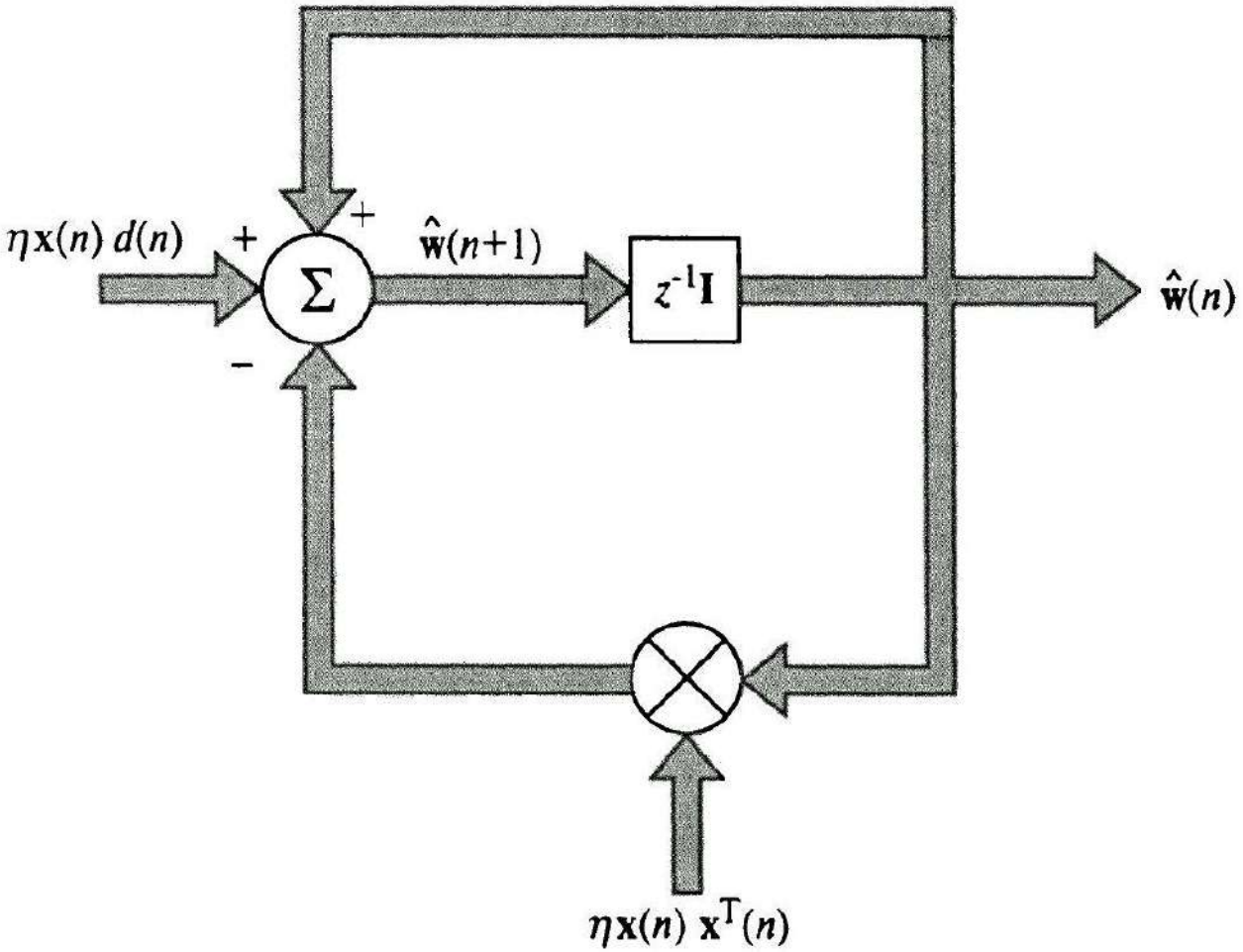
$$\hat{\mathbf{g}}(n) = -\mathbf{x}(n)e(n).$$

И, наконец, используя формулу для вектора градиента в методе наискорейшего спуска можно сформулировать алгоритм минимизации среднеквадратической ошибки в следующем виде:

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\, \mathbf{x}(n)e(n),$$

$$\hat{\mathbf{w}}(0) = 0$$

LMS

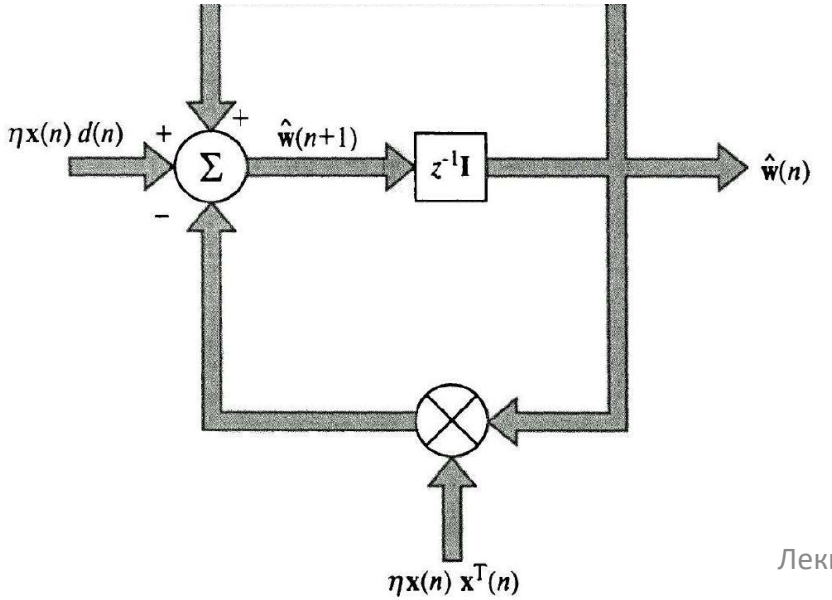


LMS

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\, \mathbf{x}(n)[d(n) - \mathbf{x}^T(n)\hat{\mathbf{w}}(n)] =$$
$$= [\mathbf{I} - \eta\, \mathbf{x}(n)\mathbf{x}^T(n)]\hat{\mathbf{w}}(n) + \eta\, \mathbf{x}(n)d(n),$$

где $\mathbf{I}$ — единичная матрица. При использовании алгоритма минимизации среднеквадратической ошибки

$$\hat{\mathbf{w}}(n) = z^{-1}[\hat{\mathbf{w}}(n+1)],$$



Условия сходимости LMS

С практической точки зрения вопрос устойчивости играет роль именно в смысле *среднеквадратической сходимости* (convergence of the mean square):

$$E[e^2(n)] \to \text{const при } n \to \infty.$$

Для упрощения делаем некоторые упрощения:

1. Векторы входных сигналов $\mathbf{x}(1)$, $\mathbf{x}(2)$, ... являются статистически независимыми друг от друга.
2. В момент времени n вектор входного сигнала $\mathbf{x}(n)$ является статистически независимым от всех предыдущих желаемых откликов, т.е. $d(1)$, $d(2)$, ..., $d(n-1)$.
3. В момент времени n желаемый отклик $d(n)$ зависит от вектора $\mathbf{x}(n)$, но статистически не зависит от всех предыдущих значений желаемого отклика.
4. Вектор входного сигнала $\mathbf{x}(n)$ и желаемый отклик $d(n)$ выбираются из множества, подчиняющегося распределению Гаусса.

Условия сходимости LMS

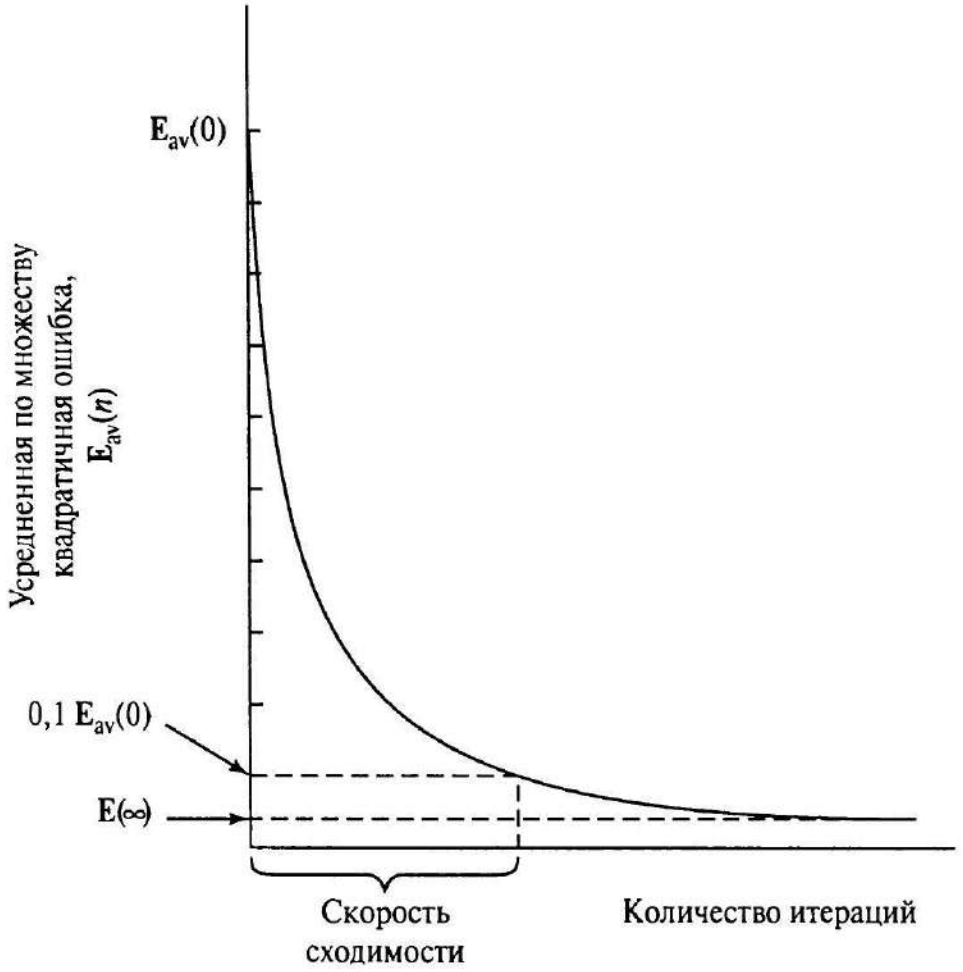
$$0 < \eta < \frac{2}{\lambda_{max}},$$

$$0 < \eta < \frac{2}{\mathrm{tr}[\mathbf{R}_x]},$$

где $\mathrm{tr}[\mathbf{R}_x]$ — след матрицы $\mathbf{R}_x$. По определению след квадратной матрицы определяется как сумма ее диагональных элементов. Так как каждый из диагональных элементов матрицы корреляции $\mathbf{R}_x$ является среднеквадратическим значением соответствующего входного сигнала, условие сходимости алгоритма минимизации среднеквадратической ошибки можно сформулировать в виде

$$0 < \eta < \frac{2}{\text{сумма среднеквадратических значений входных сигналов}}.$$

Преимущества и недостатки LMS



Простота

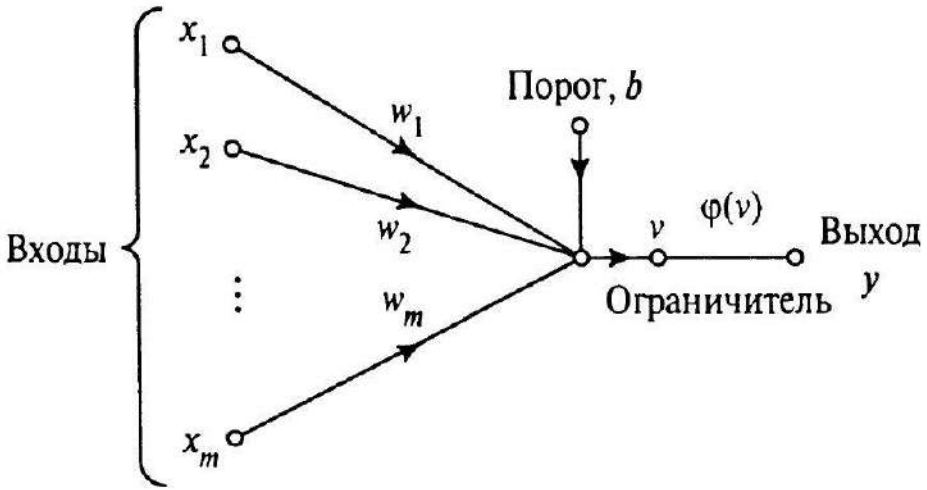
Независимость от модели

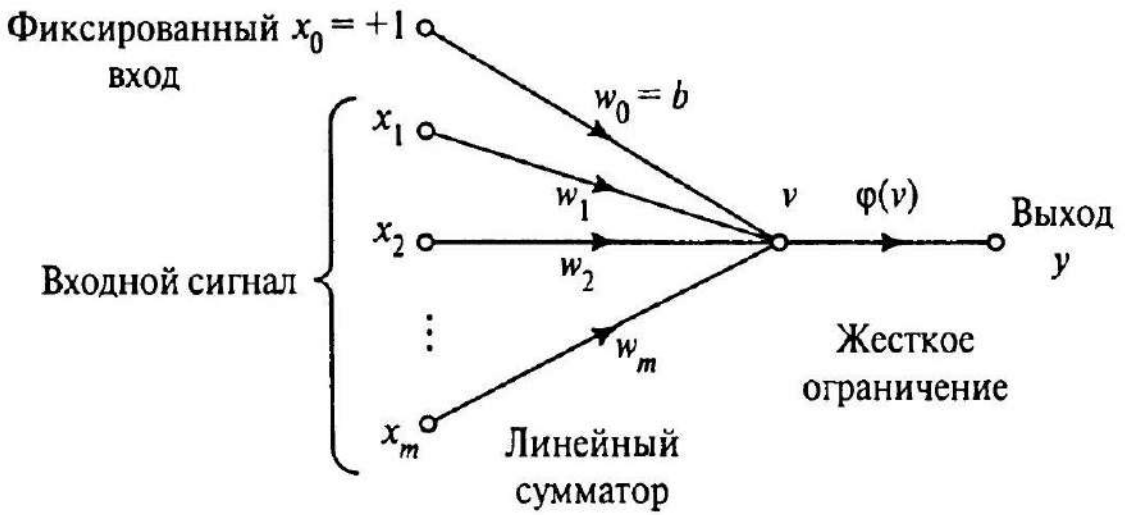
Сходимость

Низкая скорость сходимости

Кувствительность к изменениям собственных чисел матрицы входных сигналов

Персептрон





$$v = \sum_{i=1}^{m} w_i x_i + b.$$

$$v(n) = \sum_{i=0}^{m} w_i(n) x_i(n) = \mathbf{w}^T(n) x(n),$$

Персептрон

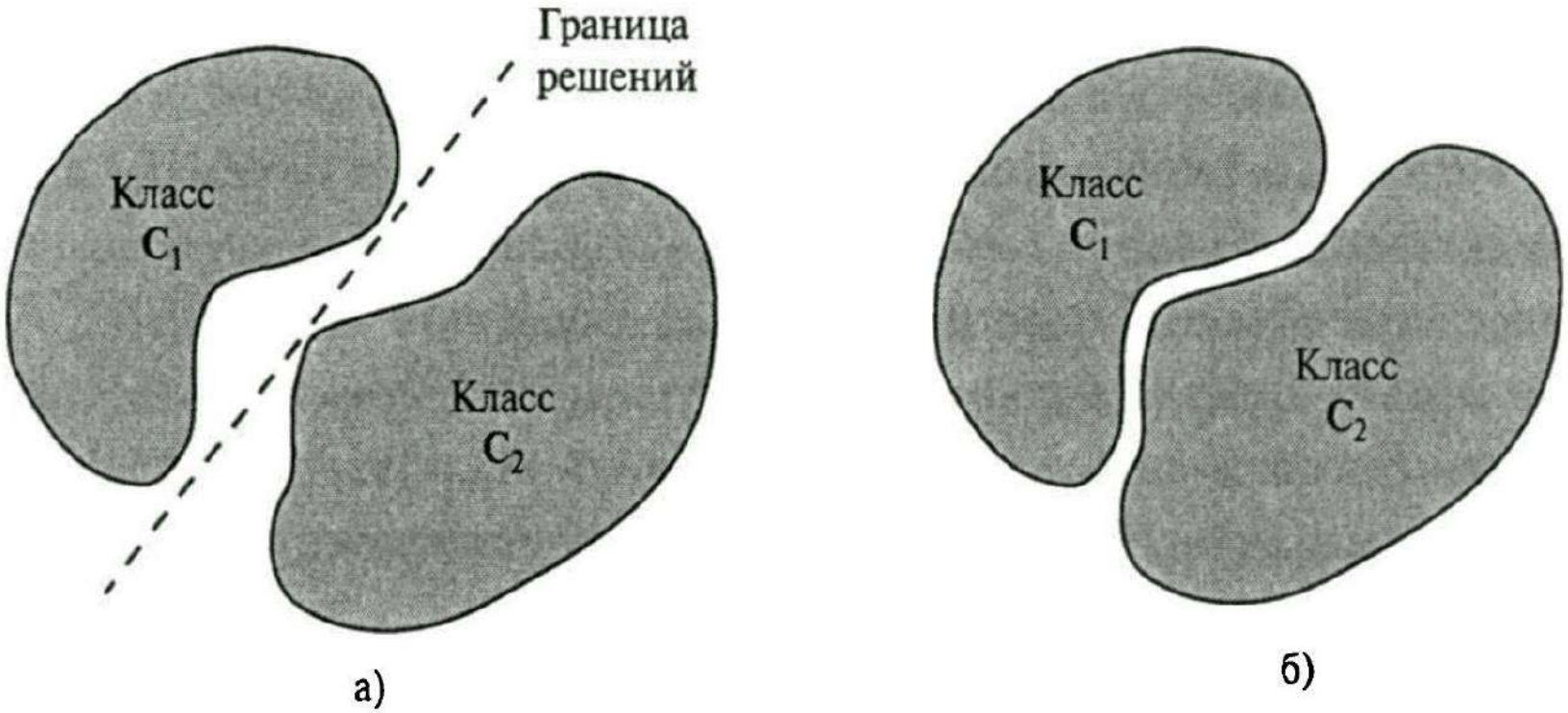
Бинарная классификация:

Целью персептрона является корректное отнесение множества внешних стимулов $x_1, x_2, \ldots, x_m$ к одному из двух классов: $\mathbf{C}_1$ или $\mathbf{C}_2$. Решающее правило такой классификации заключается в следующем: входной сигнал относится к классу $\mathbf{C}_1$, если выход y равен $+1$, и к классу $\mathbf{C}_2$ в противном случае (если выход равен -1).

Чтобы глубже изучить поведение классификатора, целесообразно построить карту областей решения в m-мерном пространстве сигналов, определяемом переменными $x_1, x_2, \ldots, x_m$. В простейшем случае (когда в качестве классификатора выступает персептрон) имеются всего две области решения, разделенные гиперплоскостью, определяемой формулой

$$\sum_{i=1}^{m} w_i x_i + b = 0.$$

Персептрон



Персептрон. Сходимость

Теперь предположим, что входные переменные персептрона принадлежат двум линейно-разделимым классам. Пусть $\mathbf{X}_1$ — подмножество векторов обучения $\mathbf{x}_1(1)$, $\mathbf{x}_1(2)$, ..., которое принадлежит классу $\mathbf{C}_1$, а $\mathbf{X}_2$ — подмножество векторов обучения $\mathbf{x}_2(1)$, $\mathbf{x}_2(2)$, ..., относящееся к классу $\mathbf{C}_2$. Объединение подмножеств $\mathbf{X}_1$ и $\mathbf{X}_2$ составляет все обучающее множество $\mathbf{X}$. Использование подмножеств $\mathbf{X}_1$ и $\mathbf{X}_2$ для обучения классификатора позволит настроить вектор весов $\mathbf{w}$ таким образом, что два класса — $\mathbf{C}_1$ и $\mathbf{C}_2$ — будут линейно-разделимыми. Это значит, что существует такой вектор весовых коэффициентов $\mathbf{w}$, для которого истинно следующее утверждение:

$\mathbf{w}^T\mathbf{x} > 0$ для любого входного вектора $\mathbf{x}$, принадлежащего классу $\mathbf{C}_1$,

$\mathbf{w}^T\mathbf{x} \leq 0$ для любого входного вектора $\mathbf{x}$, принадлежащего классу $\mathbf{C}_2$.

Персептрон. Сходимость

Алгоритм адаптации вектора весовых коэффициентов элементарного персептрона можно сформулировать следующим образом.

Если n-й элемент $\mathbf{x}(n)$ обучающего множества корректно классифицирован с помощью весовых коэффициентов $\mathbf{w}(n)$, вычисленных на n-м шаге алгоритма, то вектор весов не корректируется, т.е. действует следующее правило:

$$\mathbf{w}(n+1) = \mathbf{w}(n) \text{ если } \mathbf{w}^T\mathbf{x}(n) > 0 \text{ и } \mathbf{x}(n) \in \mathbf{C}_1,$$
$$\mathbf{w}(n+1) = \mathbf{w}(n) \text{ если } \mathbf{w}^T\mathbf{x}(n) \leq 0 \text{ и } \mathbf{x}(n) \in \mathbf{C}_2.$$

В противном случае вектор весов персептрона подвергается коррекции в соответствии со следующим правилом:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n)\mathbf{x}(n), \text{ если } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ и } \mathbf{x}(n) \in \mathbf{C}_2,$$
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\mathbf{x}(n), \text{ если } \mathbf{w}^T(n)\mathbf{x}(n) \leq 0 \text{ и } \mathbf{x}(n) \in \mathbf{C}_1,$$

где интенсивность настройки вектора весов на шаге n определяется *параметром скорости обучения* $\eta(n)$.

Если $\eta(n) = \eta > 0$, где η — константа, не зависящая от номера итерации n, вышеописанный алгоритм называется *правилом адаптации с фиксированным приращением* (fixed increment adaptation rule).

Персептрон. Сходимость

Теорема сходимости

Пусть подмножества векторов обучения $\mathbf{X}_1$ *и* $\mathbf{X}_2$ *линейно-разделимы Пусть входные сигналы поступают персептрону только их этих подмножеств. Тогда алгоритм обучения персептрона сходится после некоторого числа* n_0 *итераций в том смысле, что*

$$\mathbf{w}(n_0) = \mathbf{w}(n_0 + 1) = \mathbf{w}(n_0 + 2) = \ldots$$

является вектором решения для $n_0 \leq n_{\max}$.

Персептрон. Алгоритм

$0 < \eta \leq 1$ — параметр скорости обучения.

1. *Инициализация*

Пусть $\mathbf{w}(0)=\mathbf{0}$. Последующие вычисления выполняются для шагов $n = 1, 2, \ldots$.

2. *Активация*

На шаге n активируем персептрон, используя вектор $\mathbf{x}(n)$ с вещественными компонентами и желаемый отклик $d(n)$.

3. *Вычисление фактического ответа*

Вычисляем фактический отклик персептрона:

$$y(n) = \operatorname{sgn}(\mathbf{w}^T(n)\mathbf{x}(n)),$$

где $sgn(\cdot)$ — функция вычисления знака аргумента.

4. *Адаптация вектора весов*

Изменяем вектор весов персептрона:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n),$$

где

$$d(n) = \begin{cases} +1, & \text{если } \mathbf{x}(n) \in \mathbf{C}_1, \\ -1, & \text{если } \mathbf{x}(n) \in \mathbf{C}_2. \end{cases}$$

5. *Продолжение*

Увеличиваем номер итерации n на единицу и возвращаемся к п. 2 алгоритма.

Персептрон

где η — *параметр скорости обучения*, а разность $d(n) - y(n)$ выступает в роли *сигнала ошибки*. Параметр скорости обучения является положительной константой, принадлежащей интервалу $0<\eta \leq 1$. Выбирая значение параметра скорости обучения из этого диапазона, следует учитывать два взаимоисключающих требования

- *Усреднение* (averaging) предыдущих входных сигналов, обеспечивающее устойчивость оценки вектора весов, требует малых значений η.
- *Быстрая адаптация* (fast adaptation) к реальным изменениям распределения процесса, отвечающего за формирование векторов входного сигнала $\mathbf{x}$, требует больших значений η.

Итоги

- АФ на основе LMS и персептрон - реализации однослойного персептрона с обучением коррекцией ошибки

Различия:

1. Линейный нейрон и нелинейный
2. Непрерывность обучения и конечность обучения

Персептрон не способен к обобщению (многослойные сети), но на практике это преодалено

Основы нейронных сетей

Лекция 6

Алгоритм обратного распространения ошибки

Backpropagation

Многослойные сети - обобщение однослойного перестроена
АОРО - обобщение алгоритма минимизации среднеквадртической ошибки LMS
Странное название

Он является одним из основных способов обучения и содержит в своей основе алгоритм вычисления градиентного спуска (двигаясь вдоль градиента, происходит расчет локального максимума и минимума функции).

Признаки многослойного персептрона

1. Нелинейная функция активации каждого нейрона
2. Один или несколько скрытых слоев для извлечения важнейших признаков из входного сигнала

3. Полносвязность сети

+ обучение = вычислительная мощность МП в решении многих задач

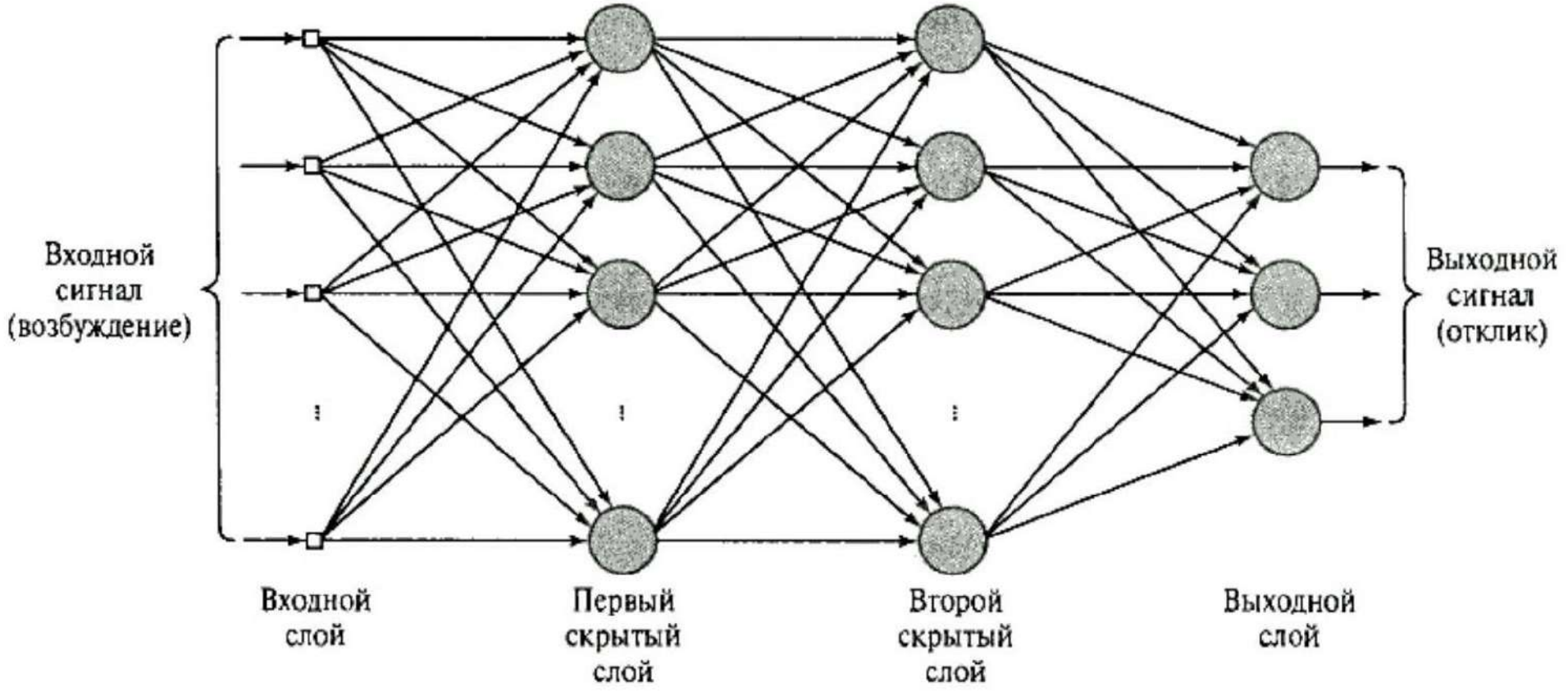
Признаки многослойного персептрона определяют недостатки

1+3 (распределенная форма нелинейности)=> сложность теоретического анализа МП;

2=> визуализация

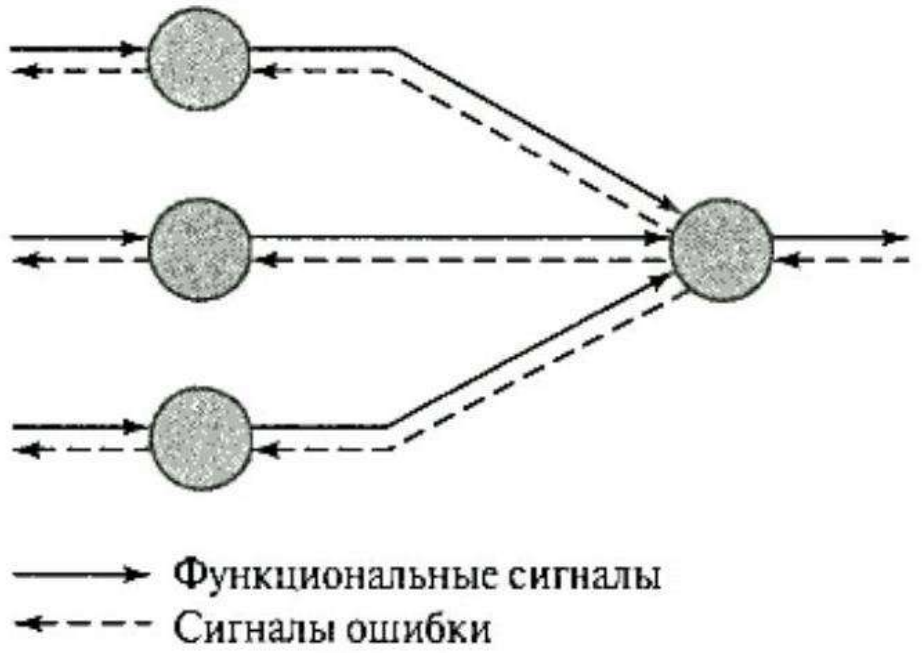
2=> представление входных образов

Структурный граф МП



Два типа сигналов в МП

1. Функциональный сигнал
или входной.
2. Сигнал ошибки.



1 - прямой проход
2 - обратный проход

Любой скрытый или выходной нейрон

1. Вычисление функционального сигнала на выходе;
2. Вычисление оценки вектора градиента;
3. Выход является входом для следующего

Постановка задачи

- Пусть есть обучающая выборка из N примеров.
- Строится сеть МП
- и обучается выбранным способом (2 прохода), пока не достигнет приемлемых результатов.
- Обученная сеть используется для получения выходного сигнала (прямой поход) на основании входного сигнала.

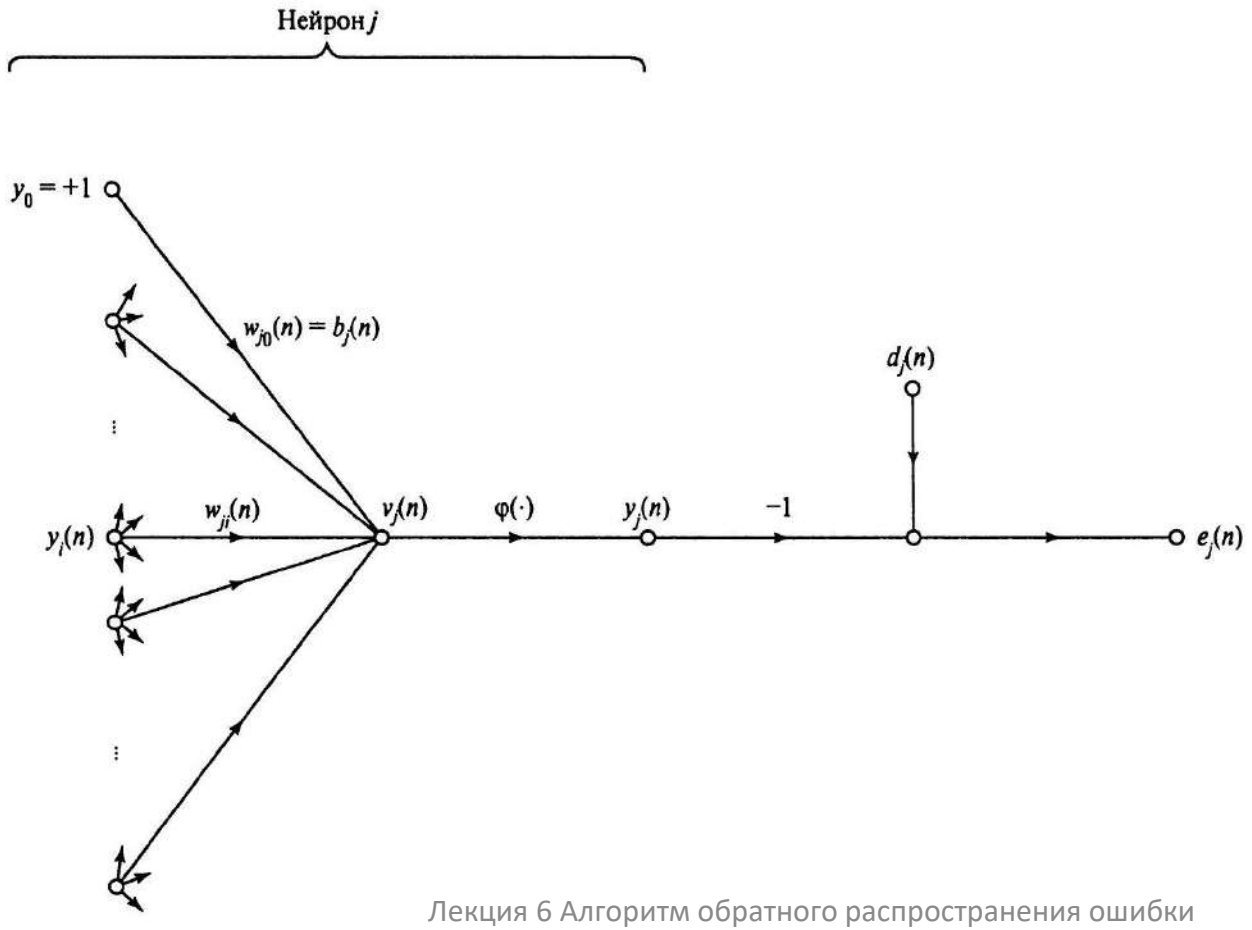
Возможна адаптация.

Обозначения (соглашения)

Индексы i, j, k - разные нейроны;
Итерация n относится к n-ому примеру;
Символы $y_j(n)$, $d_j(n)$, $e_j(n)$, $x_i(n)$, $v_j(n)$, $w_{ji}(n)$;
Размерности слоев ml, $l=0,1,2,...,L$, где L - глубина сети. Для входного слоя - m_0 нейронов, для выходного - m_L или M

Вывод АОРО

Граф передачи сигнала j-ого нейрона



Вывод АОРО

Прямой проход:

$$y_i(n) = x_i(n),$$

Вывод АОРО

Прямой проход:

$$y_i(n) = x_i(n),$$

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n),$$

Вывод АОРО

Прямой проход:

$$y_i(n) = x_i(n),$$

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n),$$

$$y_j(n) = \varphi(v_j(n)),$$

Вывод АОРО

Обратный проход:

$$e_j(n) = d_j(n) - y_j(n).$$

Вывод АОРО

Обратный проход:

$$e_j(n) = d_j(n) - y_j(n).$$

$$\frac{1}{2}e_j^2(n).$$

$$\mathbf{E}(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n),$$

Вывод АОРО

Обратный проход: для выходного слоя

$$e_j(n) = d_j(n) - y_j(n).$$

$$\mathbf{E}(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n),$$

$$\mathbf{E}_{\mathbf{av}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{E}(n).$$

Вывод АОРО

Аналогично LMS, находим частную производную функции стоимости по весам - определяем направление в пространстве весов

$$\frac{\partial \mathbf{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathbf{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}.$$

Вывод АОРО

Найдем все частные производные:

$$\frac{\partial \mathbf{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathbf{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}.$$

$$\frac{\partial \mathbf{E}(n)}{\partial e_j(n)} = e_j(n).$$

$$\mathbf{E}(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n),$$

Вывод АОРО

Найдем все частные производные:

$$\frac{\partial \mathbf{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathbf{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}.$$

$$\frac{\partial \mathbf{E}(n)}{\partial e_j(n)} = e_j(n).$$

$$\mathbf{E}(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n),$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1.$$

$$e_j(n) = d_j(n) - y_j(n).$$

Вывод АОРО

Найдем все частные производные:

$$\frac{\partial \mathbf{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathbf{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}.$$

$$\frac{\partial \mathbf{E}(n)}{\partial e_j(n)} = e_j(n).$$

$$\mathbf{E}(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n),$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1.$$

$$e_j(n) = d_j(n) - y_j(n).$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) \longleftarrow y_j(n) = \varphi(v_j(n)),$$

Вывод АОРО

Найдем все частные производные:

$$\frac{\partial \mathbf{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathbf{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}.$$

$$\frac{\partial \mathbf{E}(n)}{\partial e_j(n)} = e_j(n).$$

$$\mathbf{E}(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n),$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1.$$

$$e_j(n) = d_j(n) - y_j(n).$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n))$$

$$y_j(n) = \varphi(v_j(n)),$$

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n).$$

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n),$$

Вывод АОРО

Найдем все частные производные:

$$\frac{\partial \mathbf{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathbf{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}.$$

$$\frac{\partial \mathbf{E}(n)}{\partial e_j(n)} = e_j(n). \qquad \frac{\partial e_j(n)}{\partial y_j(n)} = -1. \qquad \frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) \qquad \frac{\partial v_j(n)}{\partial w_{ji}(n)} y_i(n). \; {}_{\partial j}(n).$$

$$\frac{\partial \mathbf{E}(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n).$$

Вывод АОРО

$$\frac{\partial \mathbf{E}(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi'_j(v_j(n))y_i(n).$$

Дельта-правило (градиентный спуск)

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathbf{E}(n)}{\partial w_{ji}(n)},$$

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n),$$

где *локальный градиент* (local gradient) $\delta_j(n)$ определяется выражением

$$\delta_j(n) = -\frac{\partial \mathbf{E}(n)}{\partial v_j(n)} = -\frac{\partial \mathbf{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n)\varphi'_j(v_j(n)).$$

Вывод АОРО

Два случая: для выходного j-ого слоя ошибка известна

$$e_j(n) = d_j(n) - y_j(n).$$

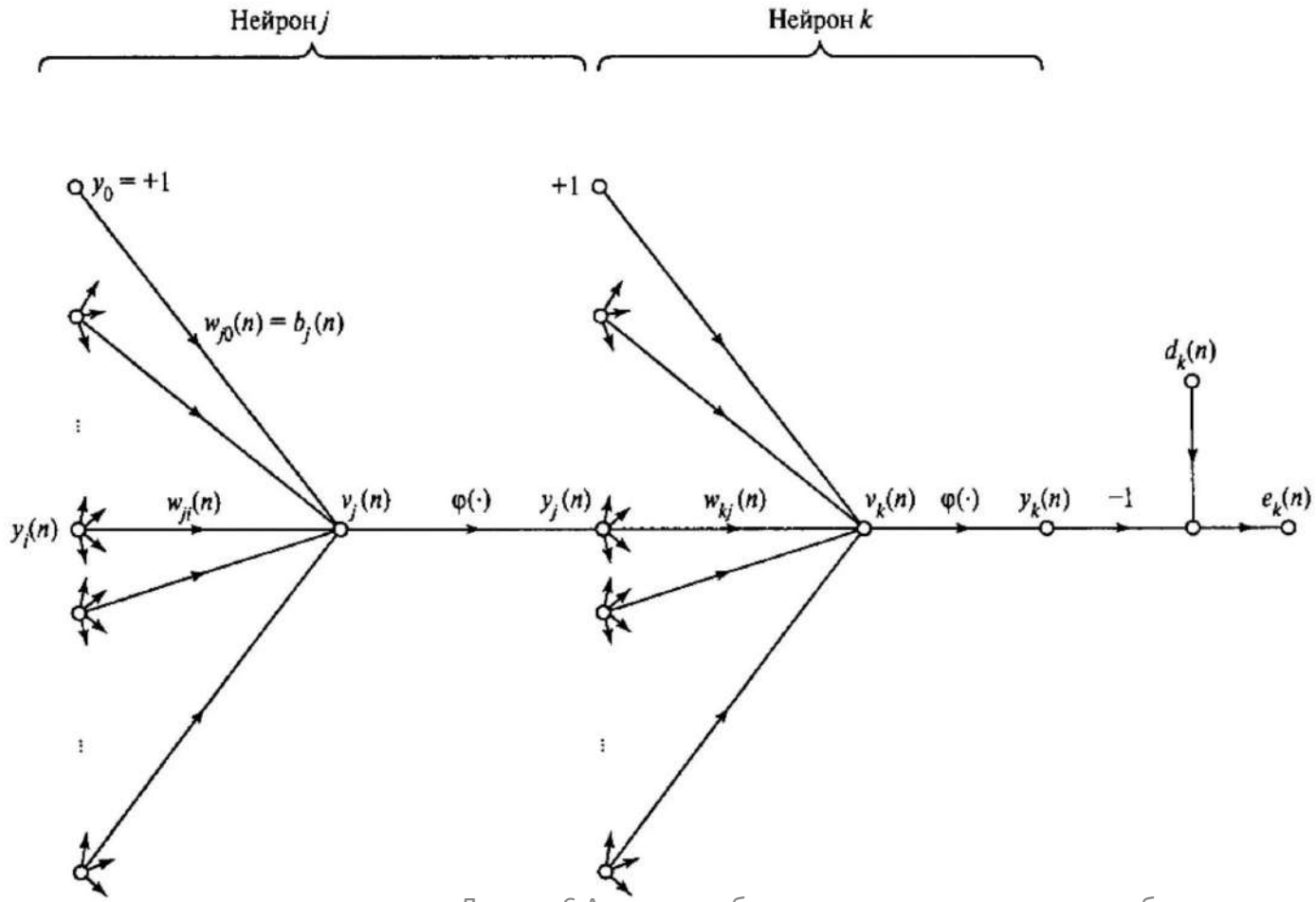
$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n),$$

где *локальный градиент* (local gradient) $\delta_j(n)$ определяется выражением

$$\delta_j(n) = -\frac{\partial \mathbf{E}(n)}{\partial v_j(n)} = -\frac{\partial \mathbf{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) \varphi_j'(v_j(n)).$$

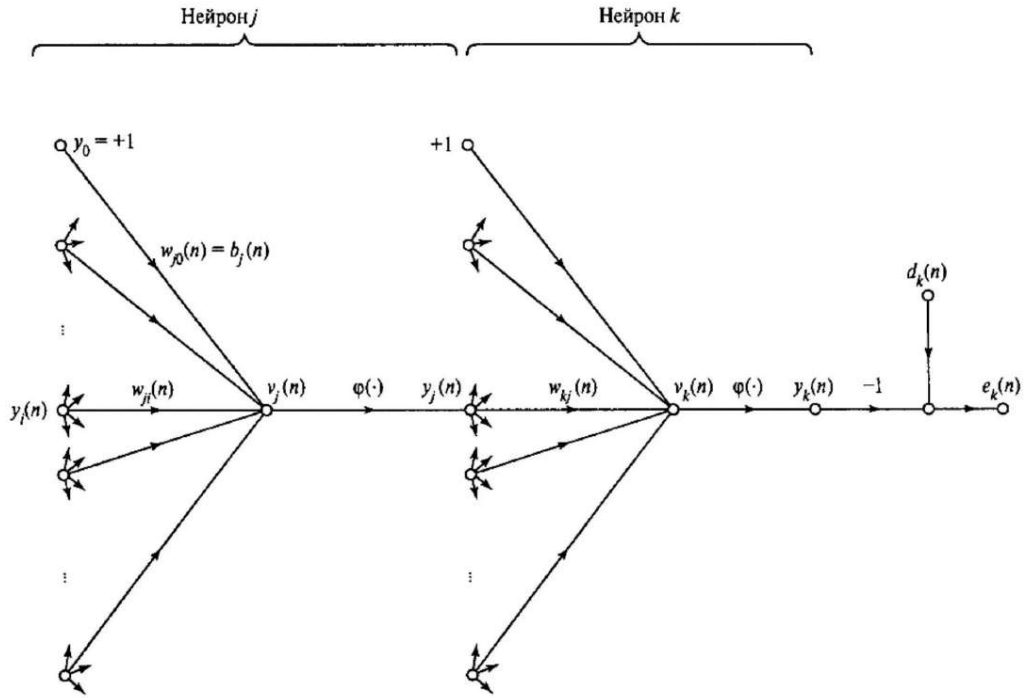
Вывод АОРО

Два случая: для скрытого j-ого слоя ошибка неизвестна



Вывод АОРО, j- скрытый слой

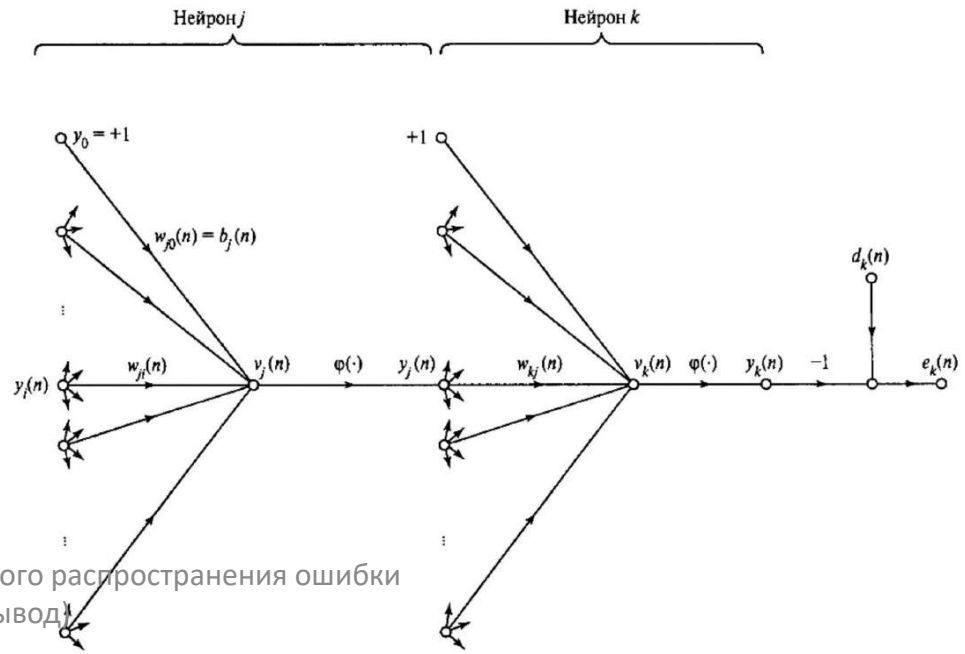
$$\delta_j(n) = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} \varphi_j'(v_j(n)),$$



Вывод АОРО, j- скрытый слой

$$\delta_j(n) = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} \varphi_j'(v_j(n)),$$

$$\mathbf{E}(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n),$$

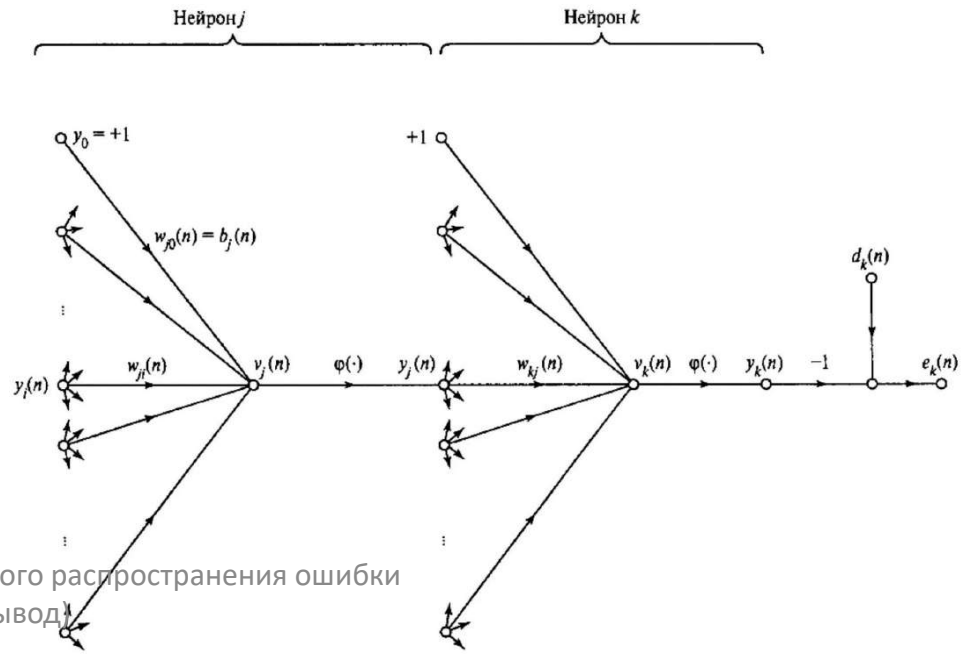


Вывод АОРО, j- скрытый слой

$$\delta_j(n) = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\varphi_j'(v_j(n)),$$

$$\mathbf{E}(n) = \frac{1}{2}\sum_{k\in C} e_k^2(n),$$

$$\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)}.$$

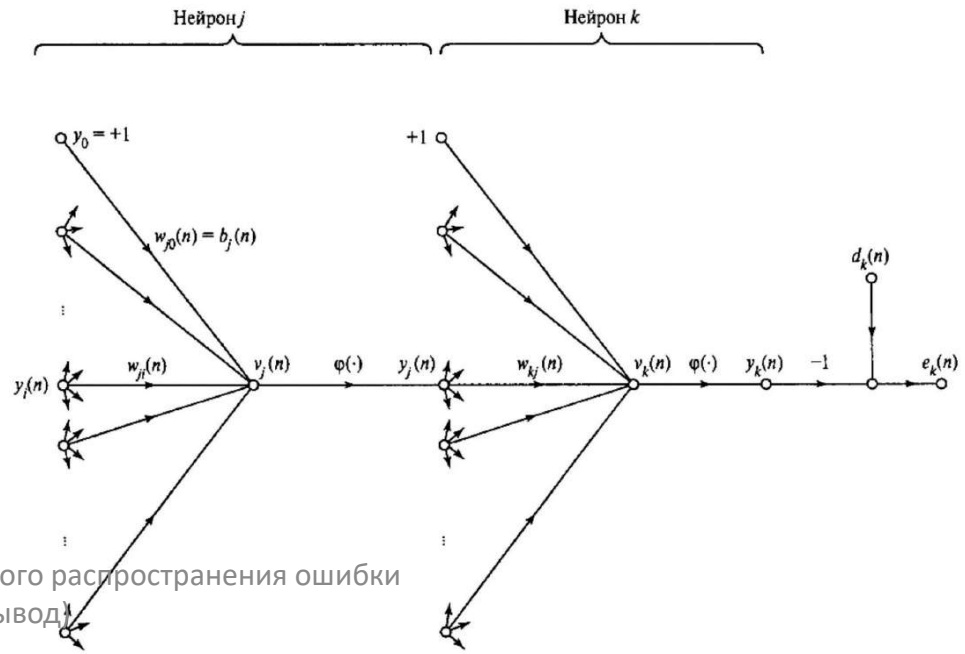


Вывод АОРО, j- скрытый слой

$$\delta_j(n) = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\varphi_j'(v_j(n)),$$

$$\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)}.$$

$$\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} = \sum_k e_k(n)\frac{\partial e_k(n)}{\partial v_k(n)}\frac{\partial v_k(n)}{\partial y_j(n)}.$$



Вывод АОРО, j- скрытый слой

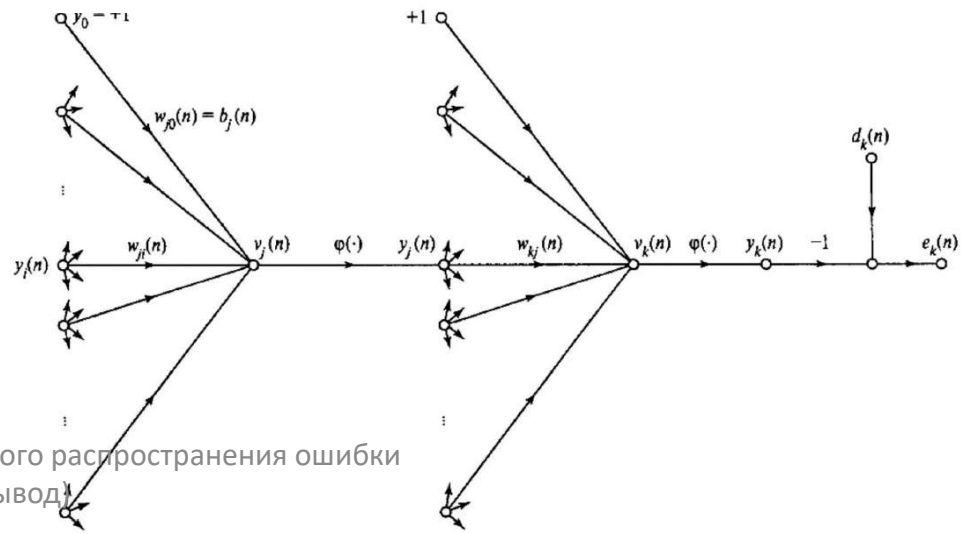
$$\delta_j(n) = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\varphi_j'(v_j(n)),$$

$$\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)}.$$

$$\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} = \sum_k e_k(n)\frac{\partial e_k(n)}{\partial v_k(n)}\frac{\partial v_k(n)}{\partial y_j(n)}.$$

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_j(v_k(n)).$$

Нейрон k



Вывод АОРО, j- скрытый слой

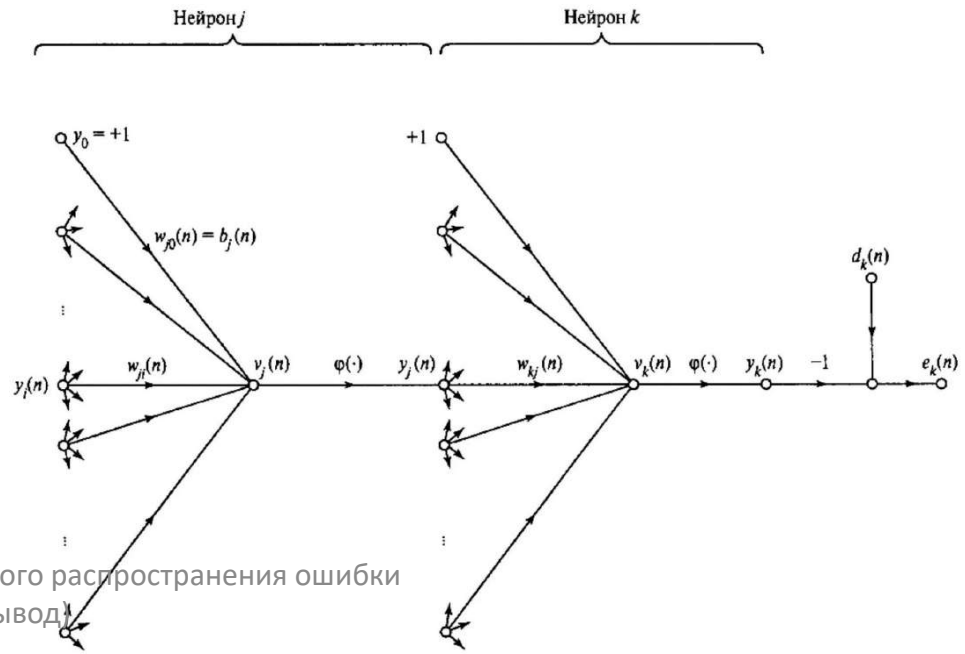
$$\delta_j(n) = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\varphi_j'(v_j(n)),$$

$$\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)}.$$

$$\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} = \sum_k e_k(n)\frac{\partial e_k(n)}{\partial v_k(n)}\frac{\partial v_k(n)}{\partial y_j(n)}.$$

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_j(v_k(n)).$$

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n)).$$



Вывод АОРО, j- скрытый слой

$$\delta_j(n) = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\varphi_j'(v_j(n)),$$

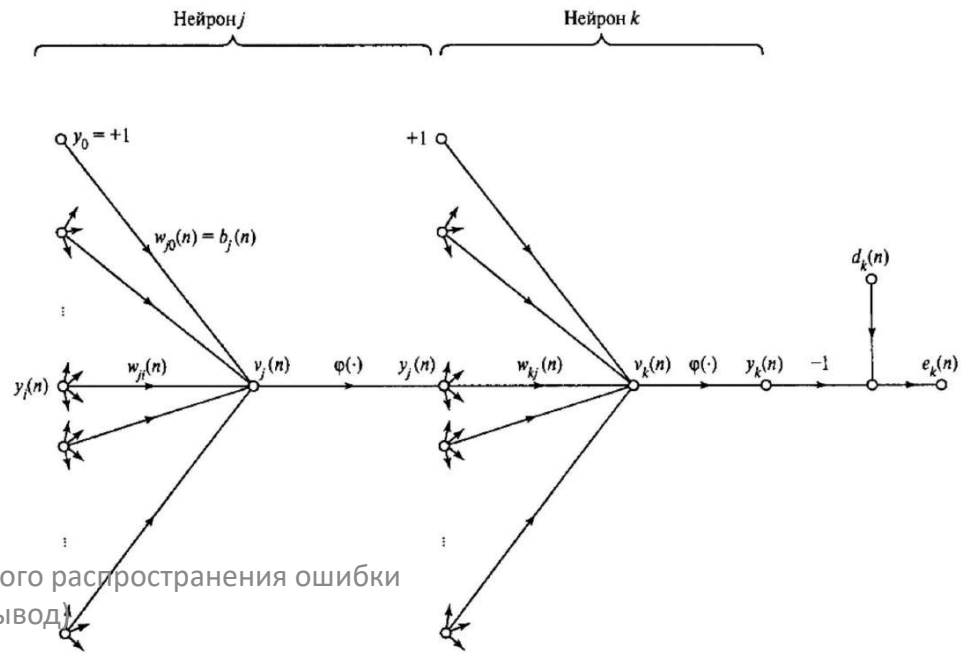
$$\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)}.$$

$$\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} = \sum_k e_k(n)\frac{\partial e_k(n)}{\partial v_k(n)}\frac{\partial v_k(n)}{\partial y_j(n)}.$$

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_j(v_k(n)).$$

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n)).$$

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n).$$

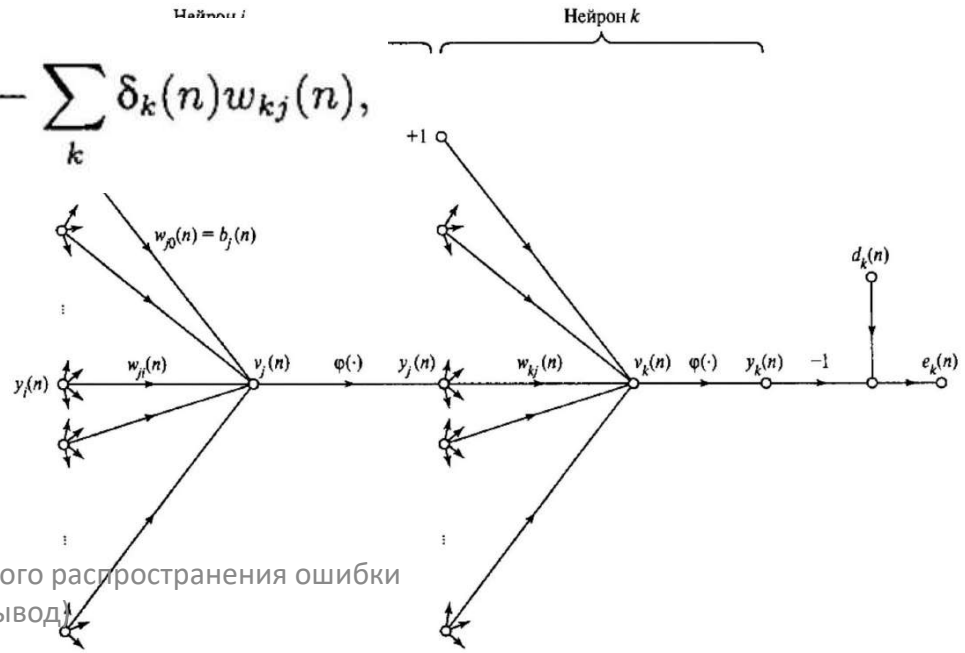


Вывод АОРО, j- скрытый слой

$$\delta_j(n) = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\varphi_j'(v_j(n)),$$

$$\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} = \sum_k e_k(n)\frac{\partial e_k(n)}{\partial v_k(n)}\frac{\partial v_k(n)}{\partial y_j(n)}. \qquad \frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n)). \qquad \frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n).$$

$$\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} = -\sum_k e_k(n)\varphi_k'(v_k(n))w_{kj}(n) = -\sum_k \delta_k(n)w_{kj}(n),$$



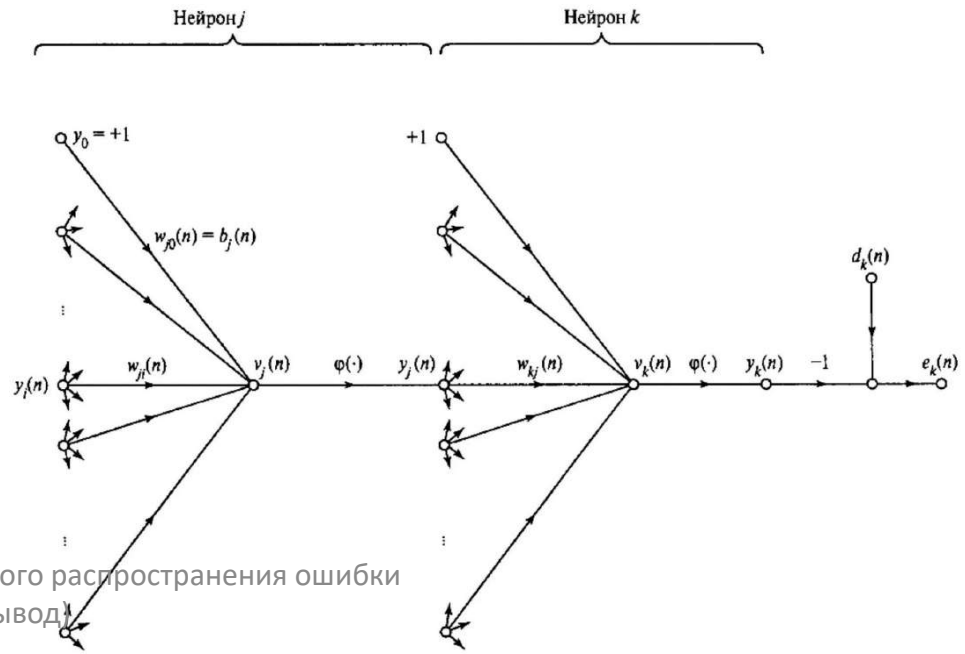
Вывод АОРО, j- скрытый слой

$$\delta_j(n) = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathbf{E}(n)}{\partial y_j(n)}\varphi_j'(v_j(n)),$$

$$\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} = \sum_k e_k(n)\frac{\partial e_k(n)}{\partial v_k(n)}\frac{\partial v_k(n)}{\partial y_j(n)}. \qquad \frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n)). \qquad \frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n).$$

$$\frac{\partial \mathbf{E}(n)}{\partial y_j(n)} = -\sum_k e_k(n)\varphi_k'(v_k(n))w_{kj}(n) = -\sum_k \delta_k(n)w_{kj}(n),$$

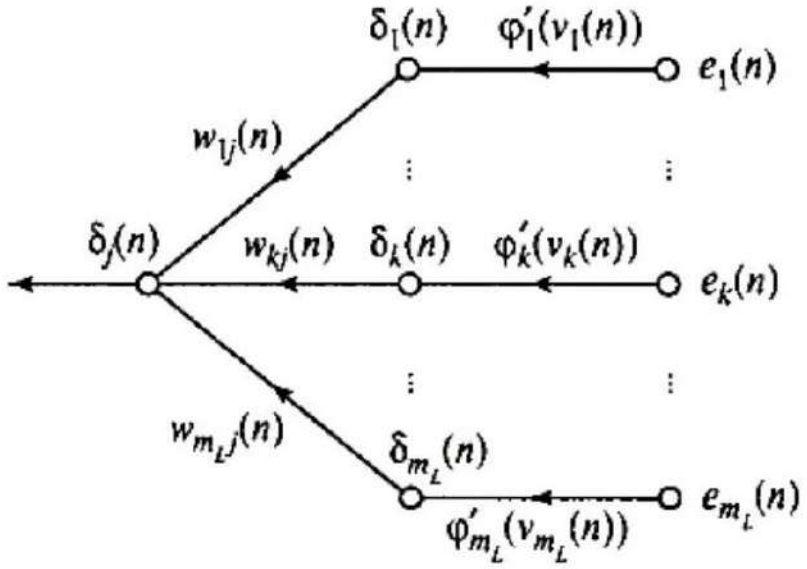
$$\delta_j(n) = \varphi_j'(v_j(n))\sum_k \delta_k(n)w_{kj}(n).$$



Вывод АОРО, j- скрытый слой

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n).$$

Графически формула выглядит:



Вывод АОРО

$$\begin{pmatrix}\text{Коррекция}\\ \text{веса}\\ \Delta w_{ji}(n)\end{pmatrix} = \begin{pmatrix}\text{Параметр ско-}\\ \text{рости обучения}\\ \eta\end{pmatrix} \cdot \begin{pmatrix}\text{Локальный}\\ \text{градиент}\\ \delta_j(n)\end{pmatrix} \cdot \begin{pmatrix}\text{Входной сиг-}\\ \text{нал нейрона } j\\ y_i(n)\end{pmatrix}$$

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n),$$

$$\delta_j(n) = e_j(n)\varphi'_j(v_j(n)).$$

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n).$$

Параметры: функция активации

$$\varphi_j(v_j(n)) = \frac{1}{1+\exp(-av_j(n))}, \quad a > 0 \ , \quad -\infty < v_j(n) < +\infty,$$

$$\varphi_j'(v_j(n)) = \frac{a\exp(-av_j(n))}{[1+\exp(-av_j(n))]^2}.$$

$$y_j(n) = \varphi_j(v_j(n))$$

$$\varphi_j'(v_j(n)) = ay_j(n)[1-y_j(n)].$$

Параметры: функция активации

$$\varphi_j'(v_j(n)) = a y_j(n)[1 - y_j(n)].$$

$$y_j(n) = o_j(n).$$

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) = a[d_j(n) - o_j(n)]o_j(n)[1 - o_j(n)],$$

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) = a y_j(n)[1 - y_j(n)] \sum_k \delta_k(n) w_{kj}(n).$$

Параметры: функция активации

$$\varphi_j(v_j(n)) = a\tanh(bv_j(n)), \quad (a,\ b) > 0,$$

$$\varphi_j'(v_j(n)) = ab\,\mathrm{sech}^2(bv_j(n)) = ab\left(1 - \tanh^2(bv_j(n))\right) =$$
$$= \frac{b}{a}[a - y_j(n)][a + y_j(n)].$$

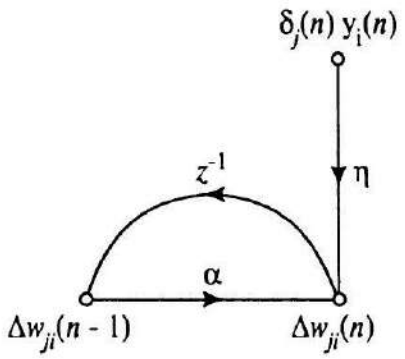
$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) = \frac{b}{a}[d_j(n) - o_j(n)][a - o_j(n)][a + o_j(n)].$$

$$\delta_j(n) = \varphi_j'(v_j(n))\sum_k \delta_k(n)w_{kj}(n) =$$
$$= \frac{b}{a}[a - y_j(n)][a + y_j(n)]\sum_{k} \delta_k(n)w_{kj}(n).$$

Параметры: скорость обучения

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n),$$

где α — как правило, положительное значение, называемое *постоянной момента* (momentum constant). Как показано на рис , она является управляющим воздействием в контуре обратной связи для $\Delta w_{ji}(n)$, где z^{-1} — оператор единичной задержки. Уравнение называется *обобщенным дельта-правилом*[3] (generalized delta rule). При $\alpha = 0$ оно вырождается в обычное дельта-правило.



Параметры: скорость обучения

Для того чтобы оценить влияние последовательности представления обучающих примеров на синаптические веса в зависимости от константы α, перепишем в виде временного ряда с индексом t. Значение индекса t будет изменяться от нуля до текущего значения n. При этом выражение можно рассматривать как разностное уравнение первого порядка для коррекции весов $\Delta w_{ji}(n)$. Решая это уравнение относительно $\Delta w_{ji}(n)$, получим временной ряд длины $n +$

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) y_i(t).$$

$$\delta_j(n) y_i(n) = -\partial \mathbf{E}(n) / \partial w_{ji}(n).$$

Следовательно, соотношение можно переписать в эквивалентной форме:

$$\Delta w_{ji}(n) = -\eta \sum_{t=0}^{n} \alpha^{n-t} \frac{\partial \mathbf{E}(t)}{\partial w_{ji}(t)}.$$

Параметры: скорость обучения

1. Текущее значение коррекции весов $\Delta w_{ji}(n)$ представляет собой сумму экспоненциально взвешенного временно*го* ряда. Для того чтобы этот ряд сходился, постоянная момента должна находиться в диапазоне $0 \leq |\alpha| < 1$. Если константа α равна нулю, алгоритм обратного распространения работает без момента. Следует также заметить, что константа α может быть и отрицательной, хотя эти значения не рекомендуется использовать на практике.
2. Если частная производная $\partial \mathbf{E}(n)/\partial w_{ji}(n)$ имеет один и тот же алгебраический знак на нескольких последовательных итерациях, то экспоненциально взвешенная сумма $\Delta w_{ji}(n)$ возрастает по абсолютному значению, поэтому веса $w_{ji}(n)$ могут изменяться на очень большую величину. Включение момента в алгоритм обратного распространения ведет к *ускорению спуска* (accelerate descent) в некотором постоянном направлении.
3. Если частная производная $\partial \mathbf{E}(n)/\partial w_{ji}(n)$ на нескольких последовательных итерациях меняет знак, экспоненциально взвешенная сумма $\Delta w_{ji}(n)$ уменьшается по абсолютной величине, поэтому веса $w_{ji}(n)$ изменяются на небольшую величину. Таким образом, добавление момента в алгоритм обратного распространения ведет к *стабилизирующему эффекту* (stabilized effect) для направлений, изменяющих знак.

Параметры: обучающая выборка

Эпоха - один цикл представления всех примеров обучения

Порядок представления примеров - случайным образом

Два режима: последовательный (интерактивный)

Параметры: обучающая выборка

- пакетный режим: корректировка весов после подачи примеров эпохи (batch):

$$\mathbf{E}_{\mathbf{av}}(n) = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} e_j^2(n),$$

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathbf{E}_{\mathbf{av}}}{\partial w_{ji}} = -\frac{\eta}{N} \sum_{n=1}^{N} e_j(n) \frac{\partial e_j(n)}{\partial w_{ji}}.$$

Параметры: обучающая выборка

- Размер
- Сбалансированность
- Анализ и предобработка

Параметры: критерий остановки

Нет четко определенного критерия => несколько критериев остановки (сходимости)

1. Норма вектора градиента достигает малых значений - долго сходится
2. Малая абсолютная интенсивность изменений среднеквадратической ошибки за эпоху. 0,1-1%, иногда 0,01, но преждевременная остановка
3. Тестирование (валидация)

Схема алгоритма (общая)

алгоритм циклически обрабатывает примеры из обучающего множества $\{(x(n), d(n))\}_{n=1}^{N}$ следующим образом.

1. *Инициализация* (initialization). Предполагая отсутствие априорной информации, генерируем синаптические веса и пороговые значения с помощью датчика равномерно распределенных чисел со средним значением 0. Дисперсия выбирается таким образом, чтобы стандартное отклонение индуцированного локального поля нейронов приходилось на линейную часть сигмоидальной функции активации (и не достигало области насыщения).

Схема алгоритма

2. *Предъявление примеров обучения* (presentation of training examples). В сеть подаются образы из обучающего множества (эпохи). Для каждого образа последовательно выполняются прямой и обратный проходы, описанные далее в пп. 3 и 4.

3. *Прямой проход* (forward computation). Пусть пример обучения представлен парой $(\mathbf{x}(n), \mathbf{d}(n))$, где $\mathbf{x}(n)$ — входной вектор, предъявляемый входному слою сенсорных узлов; $\mathbf{d}(n)$ — желаемый отклик, предоставляемый выходному слою нейронов для формирования сигнала ошибки. Вычисляем индуцированные локальные поля и функциональные сигналы сети, проходя по ней послойно в прямом направлении. Индуцированное локальное поле нейрона j слоя l вычисляется по формуле

$$v_j^{(l)}(n) = \sum_{i=0}^{m_0} w_{ji}^{(l)}(n) y_i^{(l-1)}(n),$$

где $y_i^{(l-1)}(n)$ — выходной (функциональный) сигнал нейрона i, расположенного в предыдущем слое $l-1$, на итерации n; $w_{ji}^{(l)}(n)$ —синаптический вес связи нейрона j слоя l с нейроном i слоя $l-1$. Для $i = 0$ $y_0^{(l-1)}(n) = +1$, а $w_{j0}^{(l)}(n) = b_j^l(n)$ — порог, применяемый к нейрону j слоя l. Если используется сигмоидальная функция, то выходной сигнал нейрона j слоя l выражается следующим образом:

$$y_j^{(l)}(n) = \varphi_j(v_j(n)).$$

Схема алгоритма

Если нейрон j находится в первом скрытом слое (т.е. $l = 1$), то

$$y_j^{(0)}(n) = x_j(n),$$

где $x_j(n)$ — j-й элемент входного вектора $\mathbf{x}(n)$. Если нейрон j находится в выходном слое (т.е. $l = L$, где L — глубина сети), то

$$y_j^{(L)}(n) = o_j(n).$$

Вычисляем сигнал ошибки

$$e_j(n) = d_j(n) - o_j(n),$$

где $d_j(n)$ — j-й элемент вектора желаемого отклика $\mathbf{d}(n)$.

Схема алгоритма

4. *Обратный проход* (backward computation). Вычисляем локальные градиенты узлов сети по следующей формуле:

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n)\varphi_j'(v_j^{(L)}(n)) & \text{для нейрона } j \text{ выходного слоя } L, \\ \varphi_j'(v_j^{(l)}(n)) \sum_k \delta_k^{(l+1)}(n) w_{kj}^{(l+1)}(n) & \text{для нейрона } j \text{ скрытого слоя } l, \end{cases}$$

где штрих в функции $\varphi_j'(\cdot)$ обозначает дифференцирование по аргументу. Изменение синаптических весов слоя l сети выполняется в соответствии с обобщенным дельта-правилом

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[w_{ji}^{(l)}(n-1)] + \eta\delta_j^{(l)}(n) y_j^{(l-1)}(n),$$

где η — параметр скорости обучения; α — постоянная момента.

Схема алгоритма

5. *Итерации* (iteration). Последовательно выполняем прямой и обратный проходы (согласно пп. 3, 4), предъявляя сети все примеры обучения из эпохи, пока не будет достигнут критерий останова.

Примечание. Порядок представления примеров обучения может случайным образом меняться от эпохи к эпохе. Параметры момента и скорости обучения настраиваются (и обычно уменьшаются) по мере роста количества итераций. Объяснение этого факта будет приведено ниже.

Основы нейронных сетей

Лекция 7

Алгоритм обратного распространения ошибки. Оптимизации и модификации

Постановка задачи

Вопросы:

1. Какие сигналы получит сеть?
2. Как интерпретировать выходные сигналы сети?
3. Как будем оценивать работу сети?

Ответы

В спец устройствах или программах:

- предобработчике
- интерпретаторе ответов
- оценке
+ модификация параметров сети
+ упрощение сети

Улучшение производительности

1. Пакетный или последовательный режим:
- быстрее сходится
- избыточное данные (немаркированные)
- большой размер выборки

Улучшение производительности

2. Состав выборки - max информационной насыщенность в области задачи:
- примеры, вызывающие наибольшие ошибки обучения
- примеры, max отличающиеся от ранее использованных

Улучшение производительности

При обучении
- метод случайного изменения порядка следования примеров
- метод акцентирования, проблемы:
1. неудачный или немаркированные примеры ведут к неэффективности алгоритма, то есть обучаются шуму
2. Искажение распределения примеров в эпохе

Улучшение производительности

3. Функция активации
Антисимметричная Функция может ускорить обучения,
Гиперболический тангенс при a=1,7 b=2/3, тогда
При 0, функция=1,14
При 1, вторая производная максимальна
ReLu + подобные

Улучшение производительности

4. Предобработка желаемых откликов d:
Из области значений функции активации, возможно со сдвигом
Для тангенса

$$d_j = a - \varepsilon.$$

$$d_j = -a + \varepsilon,$$

$$\varepsilon = 0,7159.$$

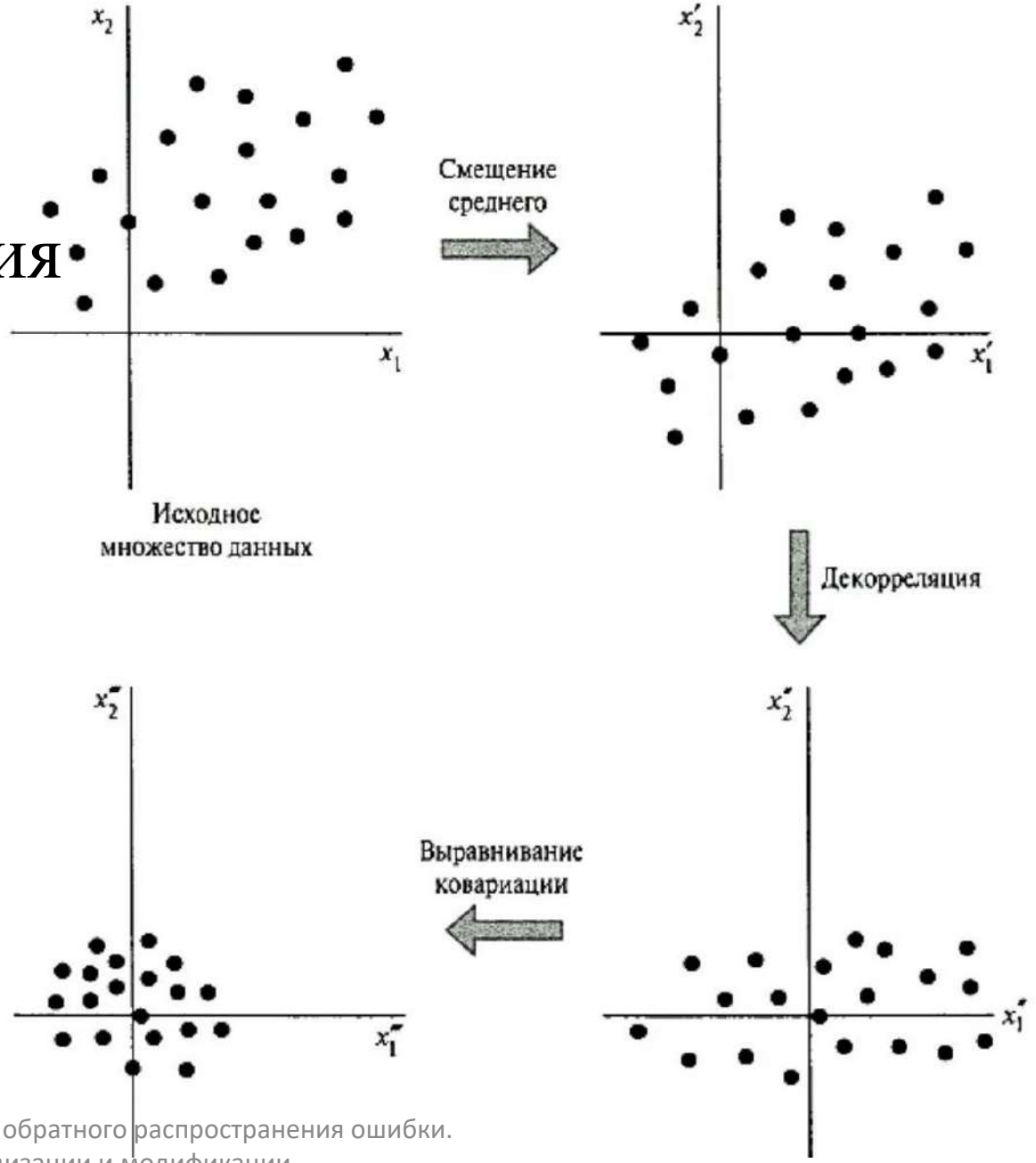
Улучшение производительности

$$\mu_y = E[y_i] = 0 \text{ для всех } i,$$
$$\sigma_y^2 = E[(y_i - \mu_i)^2] = E[y_i^2] = 1 \text{ для всех } i.$$

$$E[y_i y_k] = \begin{cases} 1 \text{ для } k = i, \\ 0 \text{ для } k \neq i, \end{cases}$$

Улучшение

- Смещение среднего значения
- Декорреляция
- Выравнивание ковариации



Улучшение производительности

6. Инициализация

Слишком большие или малые не нужны

$$\mu_y = E[y_i] = 0 \text{ для всех } i,$$
$$\sigma_y^2 = E[(y_i - \mu_i)^2] = E[y_i^2] = 1 \text{ для всех } i.$$

Улучшение производительности

Далее предположим, что входные сигналы некоррелированны:

$$E[y_i y_k] = \begin{cases} 1 \text{ для } k = i, \\ 0 \text{ для } k \neq i, \end{cases}$$

и синаптические веса выбраны из множества равномерно распределенных чисел с нулевым средним

$$\mu_w = E[w_{ji}] = 0 \text{ для всех пар } (j, i)$$

и дисперсией

$$\sigma_w^2 = E[(w_{ji} - \mu_w)^2] = E[w_{ji}^2] \text{ для всех пар } (j, i).$$

Улучшение производительности

Следовательно, математическое ожидание и дисперсию индуцированного локального поля можно выразить так:

$$\mu_v = E[v_j] = E\left[\sum_{i=1}^{m} w_{ji} y_i\right] = \sum_{i=1}^{m} E[w_{ji}] E[y_i] = 0,$$

$$\sigma_v^2 = E[(v_j - \mu_v)^2] = E[v_j^2] = E\left[\sum_{i=1}^{m}\sum_{k=1}^{m} w_{ji} w_{jk} y_i y_k\right] =$$

$$= \sum_{i=1}^{m}\sum_{k=1}^{m} E[w_{ji} w_{jk}] E[y_i y_k] = \sum_{i=1}^{m} E[w_{ji}^2] = m\sigma_w^2,$$

где m — число синаптических связей нейрона.

Улучшение производительности

Для тангенса

$$\sigma_w = m^{-1/2}.$$

Таким образом, желательно, чтобы равномерное распределение, из которого выбираются исходные значения синаптических весов, имело нулевое среднее значение и дисперсию, обратную корню квадратному из количества синаптических связей нейрона.

Улучшение производительности

7. Априорная информация о вычисляемой функции: симметричность, инвариантность

Улучшение производительности

8. Скорость обучения:

Правило 1. Каждый параметр сети (вес) - свое значение скорости обучения

Правило 2. Должна меняться на итерации

Правило 3. Увеличиваем, если на нескольких итерациях локальный градиент имеет один знак, иначе уменьшаем

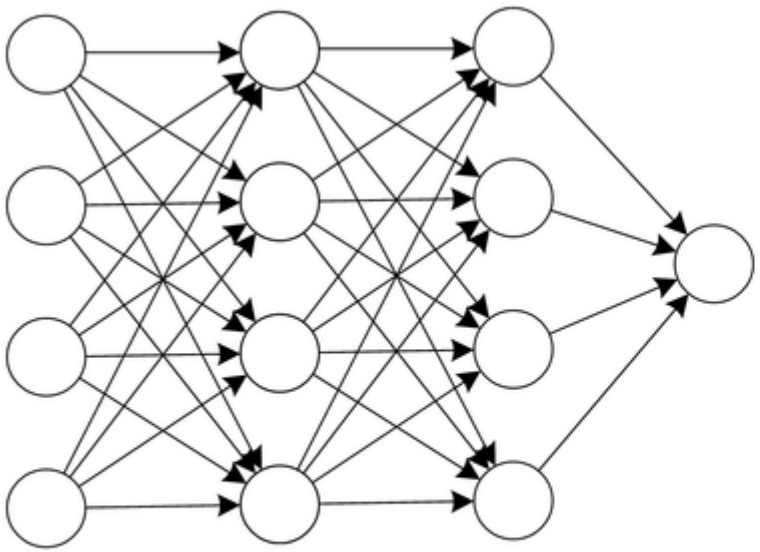
- Верхняя и нижняя граница
- Шаг изменения

Улучшение производительности

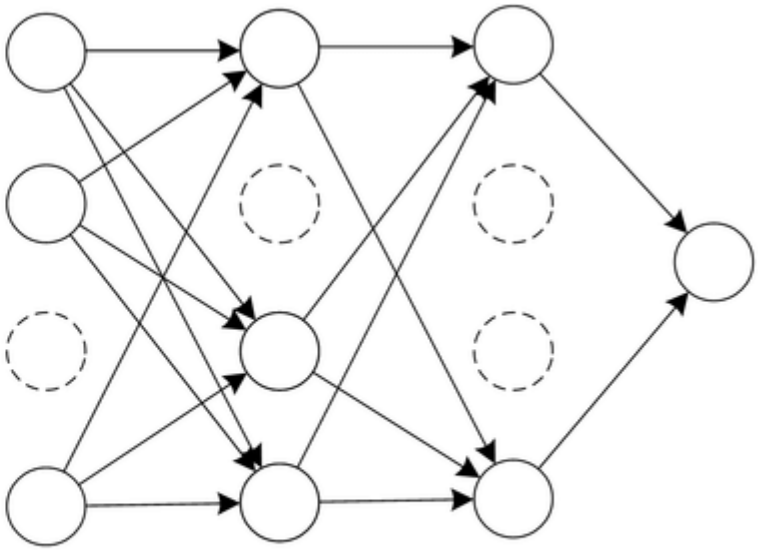
9. Регуляризация:

- Штрафы
- Dropout (прореживание)

Регуляризация – это любая модификация алгоритма обучения, предпринятая с целью уменьшить его ошибку обобщения, не уменьшая ошибку обучения.



(a) Standard Neural Network



(b) Network after Dropout

Улучшение производительности

Интерпретация ответов:

Масштабирование $y=(a+b)/2+(b-a)y/2$

В классификации :

1. Победитель забирает все - m нейронов
2. Знаковая интерпретация ($y>0$, $y=1$, иначе $y=0$.

Последовательность 0,1 - номер класса), $\log_2 m$ нейронов

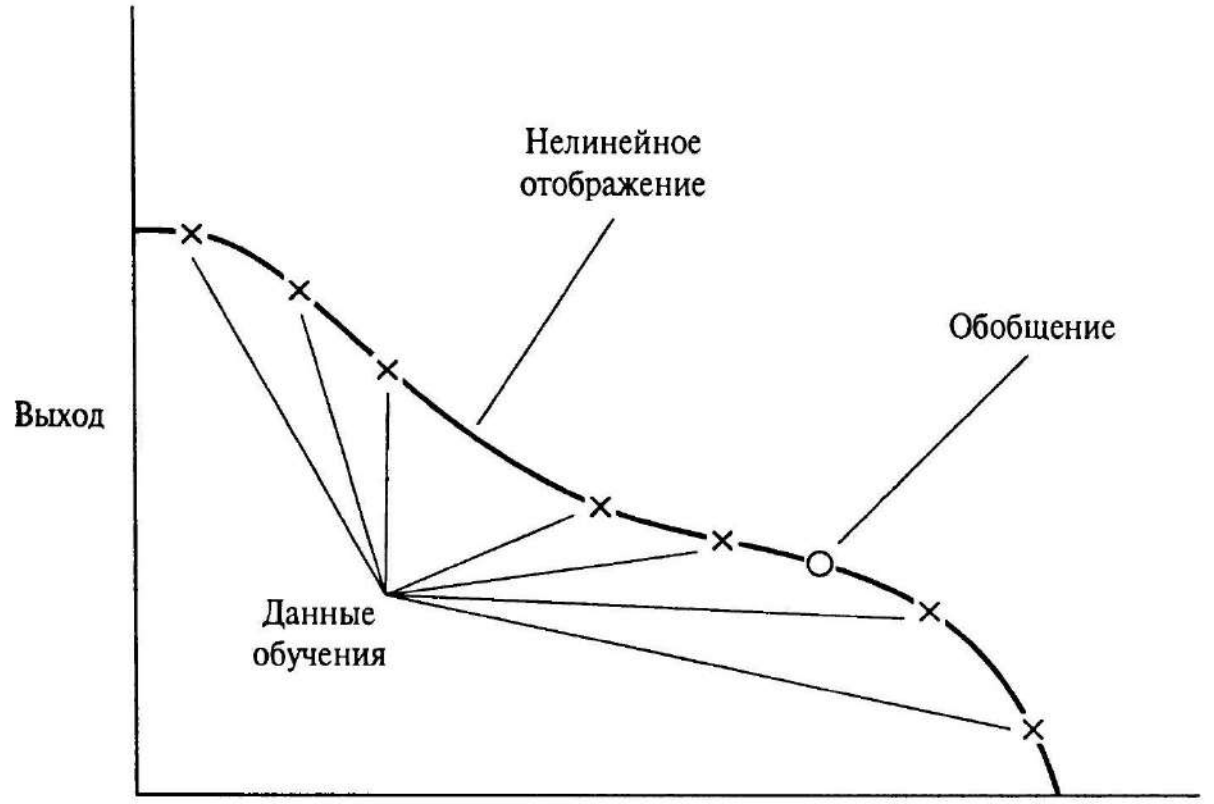
3. Порядковая интерпретация

сортировка y, обозначить n номер (1 - min, k -max),

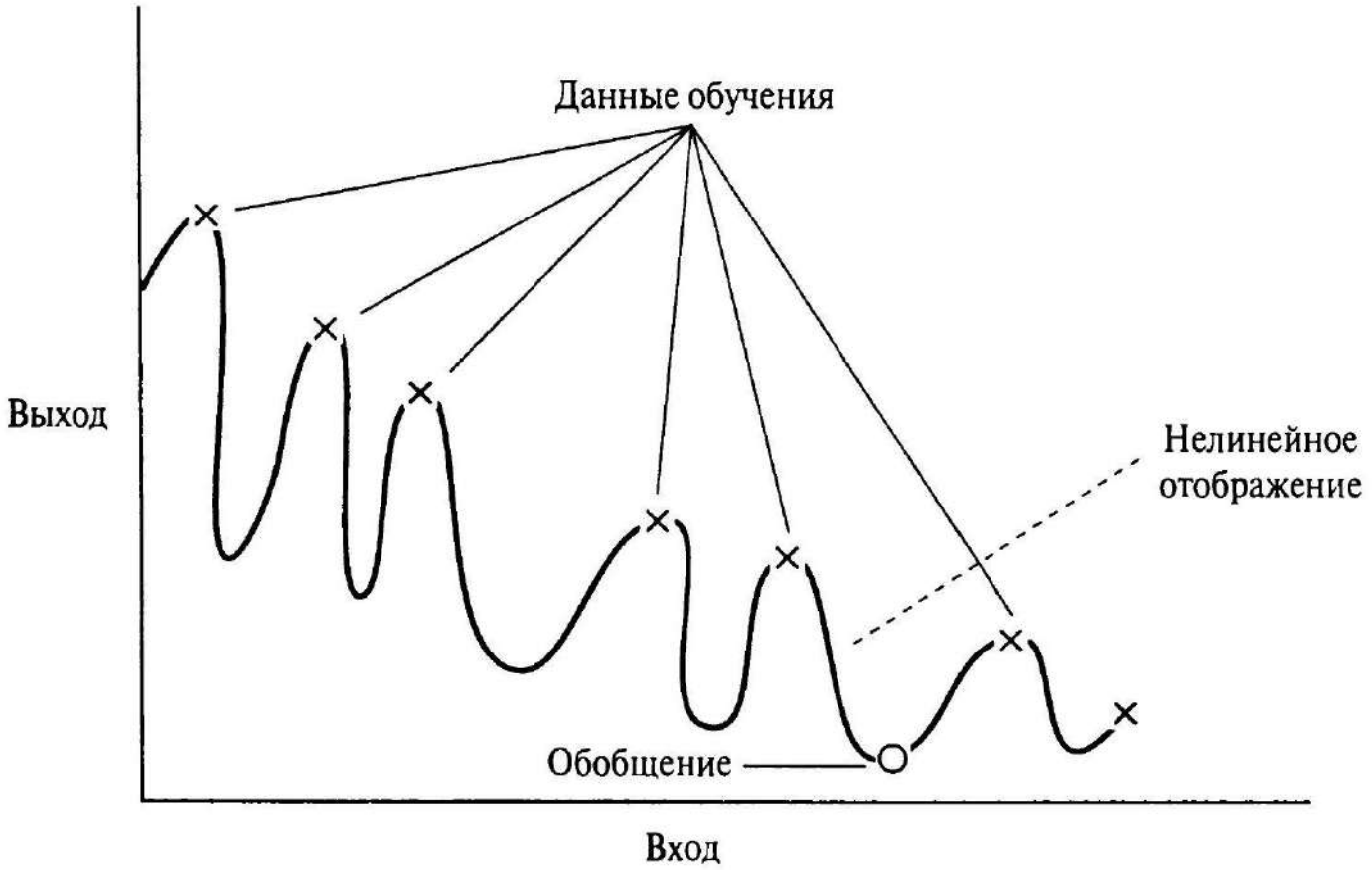
перестановка - номер класса

Обобщение

Сеть с хорошей обобщающей способностью



Обобщение


Данные обучения
Выход
Нелинейное
отображение
Обобщение
Вход

Обобщение

Определяется 3-мя факторами:

1. Архитектурой сети
2. Обучающей выбором
3. Сложностью задачи

Обобщение

Определяется 3-мя факторами:

1. Архитектурой сети
2. Обучающей выбором
3. Сложностью задачи

$$N = O(W/\varepsilon),$$

Обобщение

Каково минимальное количество скрытых слоев многослойного персептрона, обеспечивающего аппроксимацию некоторого непрерывного отображения?

Теорема об универсальной аппроксимации

1. Сеть содержит m_0 входных узлов и один скрытый слой, состоящий из m_1 нейронов. Входы обозначены $x_1, x_2, \ldots, x_{m_0}$.
2. Скрытый нейрон i имеет синаптические веса $w_{i_1}, \ldots, w_{m_0}$ и порог b_i.
3. Выход сети представляет собой линейную комбинацию выходных сигналов скрытых нейронов, взвешенных синаптическими весами выходного нейрона — $\alpha_1, \ldots, \alpha_{m_1}$.

Обобщение

теорема утверждает, что *многослойного персептрона с одним скрытым слоем достаточно для построения равномерной аппроксимации с точностью* ε *для любого обучающего множества, представленного набором входов* $x_1, x_2, \ldots, x_{m_0}$ *и желаемых откликов* $f(x_1, x_2, \ldots, x_{m_0})$. Тем не менее из теоремы не следует, что один скрытый слой является *оптимальным* в смысле времени обучения, простоты реализации и, что более важно, качества обобщения.

Обобщение

1. *Точность наилучшей аппроксимации* (accuracy of the best approximation). В соответствии с теоремой об универсальной аппроксимации для удовлетворения этого требования размер скрытого слоя m_1 должен быть большим.
2. *Точность эмпирического соответствия аппроксимации* (accuracy of empirical fit to the approximation). Для того чтобы удовлетворить этому требованию, отношение m_1/N должно иметь малое значение. Для фиксированного объема N обучающего множества размер скрытого слоя m_1 должен оставаться малым, что противоречит первому требованию.

Обобщение

1. *Локальные признаки* (local feature) извлекаются в первом скрытом слое, т.е. некоторые скрытые нейроны первого слоя можно использовать для разделения входного пространства на отдельные области, а остальные нейроны слоя обучать локальным признакам, характеризующим эти области.
2. *Глобальные признаки* (global feature) извлекаются во втором скрытом слое. В частности, нейрон второго скрытого слоя "обобщает" выходные сигналы нейронов первого скрытого слоя, относящихся к конкретной области входного пространства. Таким образом он обучается глобальным признакам этой области, а в остальных областях его выходной сигнал равен нулю.

Обобщение

Перекрестная проверка –одна из проверок оптимальности модели сети

В рамках этого подхода имеющиеся в наличии данные сначала случайным образом разбиваются на *обучающее множество* (training set) и *тестовое множество* (test set). Обучающее множество, в свою очередь, разбивается на два следующих несвязанных подмножества.

- *Подмножество для оценивания* (estimation subset), используемое для выбора модели.
- *Проверочное подмножество* (validation subset), используемое для тестирования модели.

Обобщение

Пусть параметр r, принадлежащий интервалу от нуля до единицы, определяет разбиение обучающего множества на подмножества оценивания и проверки. Пусть **T** содержит N примеров. Тогда после разбиения $(1 - r)N$ примеров будет принадлежать подмножеству оценивания, а rN примеров — проверочному подмножеству. Подмножество оценивания обозначим $\mathbf{T}'$.

Ключевой вопрос состоит в том, как определить значение параметра r, задающего разбиение обучающего множества $\mathbf{T}$ на подмножества оценивания $\mathbf{T}'$ и проверки $\mathbf{T}''$. при аналитическом исследовании этого вопроса с использованием VC-измерения и компьютерного моделирования были определены некоторые количественные свойства оптимального значения r.

- Если сложность целевой функции, определяющей желаемый отклик d в терминах входного вектора $\mathbf{x}$, мала по сравнению с размером N обучающего множества, то эффективность перекрестной проверки мало зависит от выбора r.
- Если же целевая функция становится более сложной по сравнению с размером N обучающего множества, то выбор оптимального значения r оказывает существенное влияние на эффективность перекрестной проверки, причем это значение уменьшается.

- Одно и то же *фиксированное* значение r *почти оптимально* подходит для широкого диапазона сложностей целевой функции.

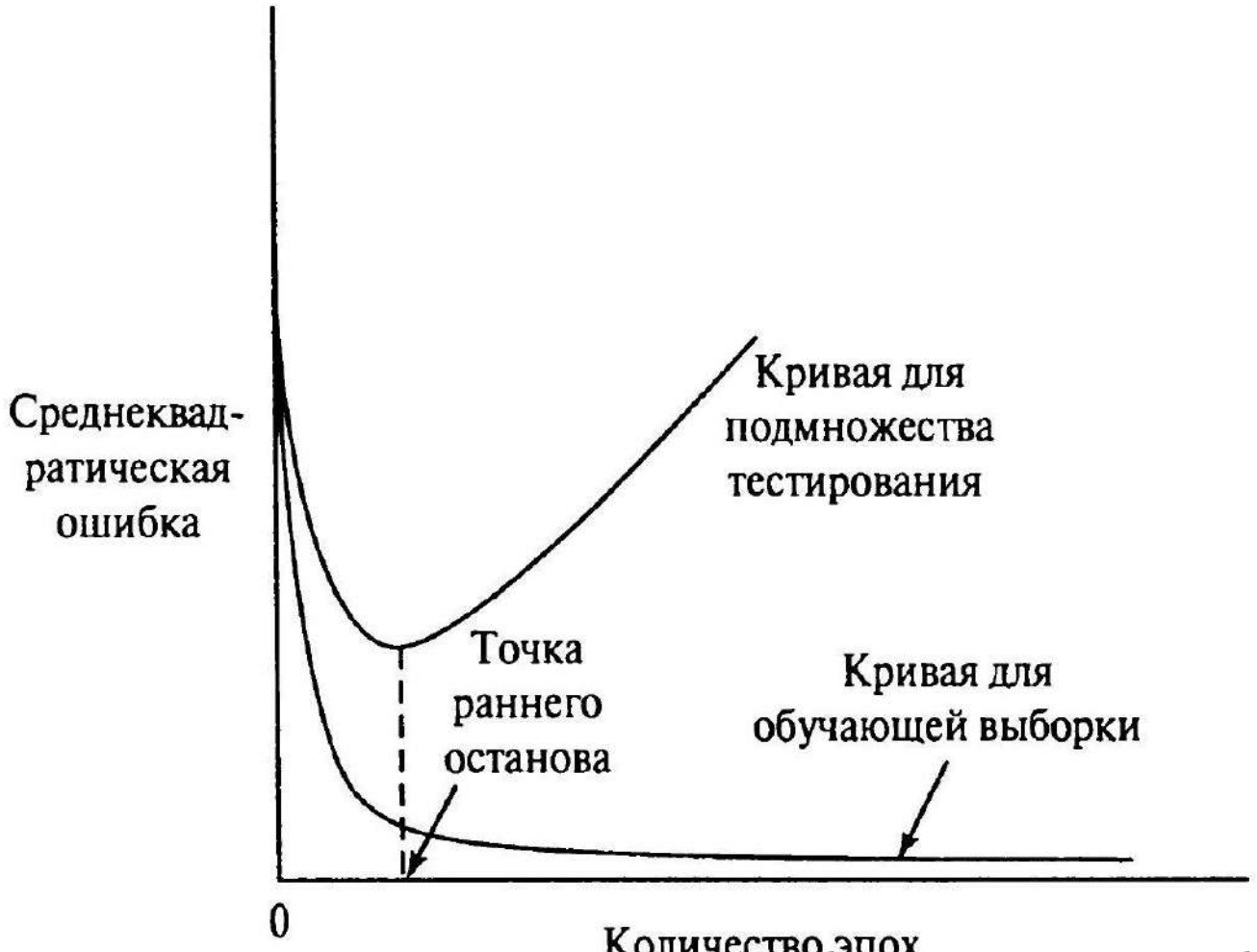
r=0,2

Обобщение

Наступление стадии излишнего переобучения можно определить с помощью перекрестной проверки, в которой данные разбиты на два подмножества — оценивания и проверки. Множество оценивания используется для обычного обучения сети с небольшой модификацией: сеанс обучения периодически останавливается (через каждые несколько эпох), после чего сеть тестируется на проверочном подмножестве. Более точно, периодический процесс оценивания-тестирования выполняется следующим образом.

- После завершения этапа оценивания (обучения) синаптические веса и уровни порогов многослойного персептрона фиксируются, и сеть переключается в режим прямого прохода. Ошибка сети вычисляется для каждого примера из проверочного подмножества.
- После завершения тестирования наступает следующий этап процесса обучения, и все повторяется.

Обобщение


Среднеквад-
ратическая
ошибка
Кривая для
подмножества
тестирования
Точка
раннего
останова
Кривая для
обучающей выборки
0
Количество эпох

Обобщение

А что если данные обучения не содержат шума? Как в этом случае определить точку раннего останова для детерминированного сценария? Частично на этот вопрос можно ответить следующим образом. Если ошибки на подмножествах оценивания и проверки не одинаково стремятся к нулю, то емкости сети не достаточно для точного функционирования модели. Лучшее, что можно сделать в таком случае, — это постараться минимизировать *интегрированную квадратичную ошибку* (integrated squared error), что (приблизительно) эквивалентно минимизации обычной глобальной среднеквадратической ошибки при равномерном распределении входного сигнала.

Обобщение

Предупреждение

Первый — это *неассимптотический режим* (nonasymptotic mode), при котором $N < 30W$, где N — размер обучающего множества; W — количество свободных параметров сети. В этом режиме метод раннего останова повышает эффективность обобщения по сравнению с полным обучением (при котором используется все множество примеров без останова сеанса). Согласно полученным результатам, переучивание может наступить при $N < 30W$, и только в этом случае существует практическая польза от использования перекрестной проверки для останова сеанса обучения. Оптимальное значение параметра r, определяющее разбиение множества примеров на подмножества для оценивания и проверки, должно выбираться из соотношения

$$r_{\text{opt}} = 1 - \frac{\sqrt{2W - 1} - 1}{2(W - 1)}.$$

При больших значениях W эта формула сводится к следующей:

$$r_{\text{opt}} \simeq 1 - \frac{1}{\sqrt{2W}}, \quad W >> 0.$$

Например, при $W = 100$, $r = 0,07$. Это значит, что 93% данных обучения составляет подмножество оценивания и только 7% приходится на проверочное подмножество.

Обобщение

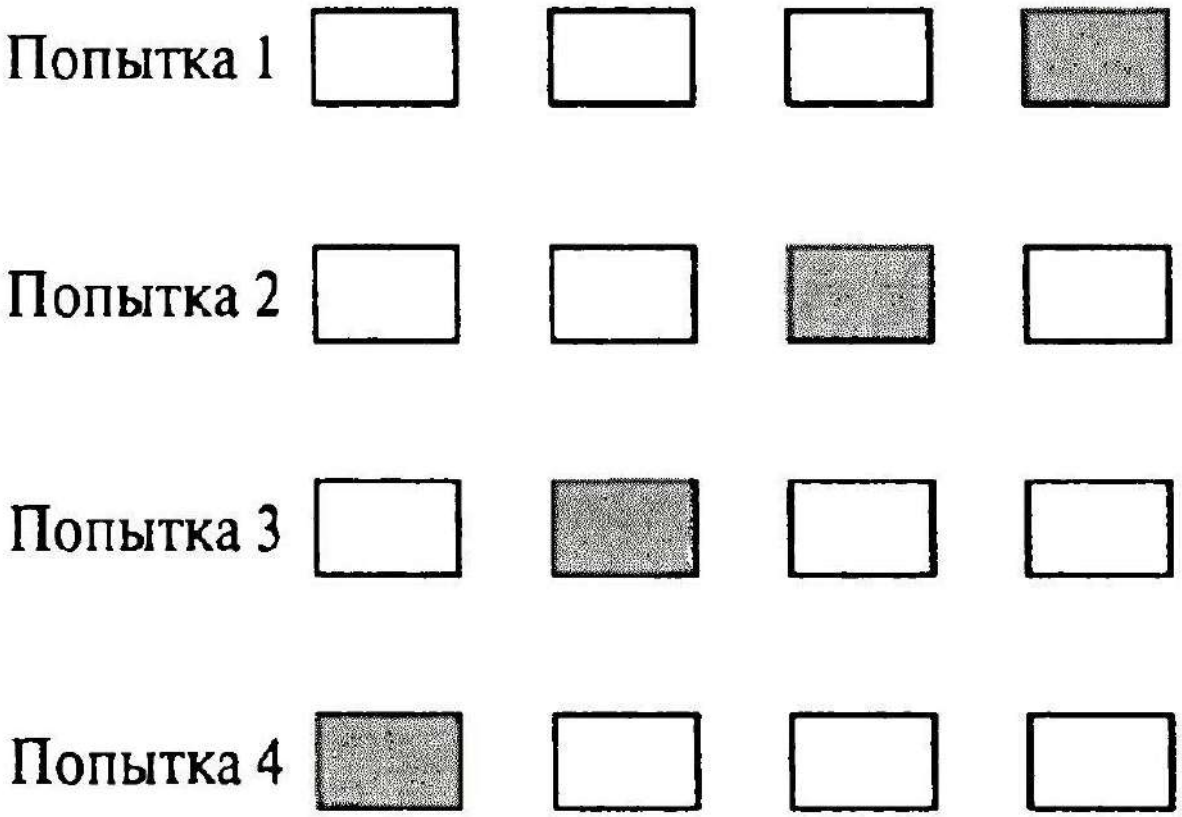
Вторым типом поведения является *асимптотический режим* (asymptotic mode), при котором $N > 30W$. В этом режиме при использовании метода раннего останова наблюдается лишь небольшое улучшение эффективности обобщения по сравнению с полным обучением. Другими словами, *полное обучение* (exhaustive learning) приводит к удовлетворительному результату, если размер множества обучения велик по сравнению с количеством параметров сети.

Обобщение

Варианты перекресток проверки

Предложенный подход к реализации перекрестной проверки получил название *метода удержания* (hold out method). Существуют и другие варианты перекрестной проверки, которые также нашли свое применение на практике, особенно при недостатке маркированных примеров. В такой ситуации можно использовать *многократную перекрестную проверку* (multifold cross-validation), разделив имеющееся в наличие множество из N примеров на K подмножеств ($K > 1$). При этом подразумевается, что N кратно K. Обучение модели проводится на всех подмножествах, кроме одного. На этом оставшемся подмножестве выполняется тестирование и определяется ошибка оценивания. Эта процедура повторяется K раз, при этом каждый раз для тестирования используются разные подмножества (случай для $K = 4$ показан на рис.). Эффективность такой модели вычисляется путем усреднения квадратичной ошибки по всем попыткам. Метод многократной перекрестной проверки имеет один недостаток: он требует очень большого объема вычислений, так как обучение

Обобщение



Обобщение

Если количество доступных примеров N ограничено, можно использовать экстремальную форму многократной перекрестной проверки — метод исключения одного примера. В этом случае для обучения модели используется $N - 1$ примеров, а тестирование выполняется на оставшемся. Эксперимент повторяется N раз, причем каждый раз для проверки используются разные примеры. После этого квадратичная ошибка оценки усредняется по всем N экспериментам.

Модификации – Rprop (OpenCV - явно)

- **Метод упругого распространения (**М. Ридмиллер (M.Riedmiller) и Г. Браун (H.Braun)**)**
- Этот метод называют также Resilient propagation (сокращенно Rprop). Он был предложен как альтернатива классическому способу обучения, который требует слишком много времени. Для увеличения скорости операций было разработано много вспомогательных алгоритмов.
- Этот метод является основным при обучении по принципу epoch (один полный проход датасета через НС). Для подгонки весовых коэффициентов он использует лишь знаки производных частного случая. При этом обязательно выдерживать правило, позволяющее определить значение коррекции коэффициента веса.

Модификации - Rprop

- Если на этой стадии вычислений производная меняет свой знак на противоположный, то это говорит о чересчур большом изменении и об упущении локального минимума. Следовательно, нужно возвратить весу предыдущее значение и уменьшить величину изменения. Если же знак остался прежним, то следует поднять величину изменения веса для максимальной сходимости.
- Если закрепить ключевые показатели подстройки весов, то можно не настраивать глобальные параметры – это является дополнительным плюсом использования метода. Причем существуют готовые значения таких показателей. Их применение рекомендовано, но жестких рамок по выбору значений нет.

Модификации - Rprop

- Чтобы величина веса не была чрезмерно большой или, наоборот, маленькой, следует оперировать значением коррекции с установленными пределами. При расчете этого значения необходимо придерживаться правила.
- Если в определенной точке производная меняет свой знак с «+» на «−», то это говорит о росте ошибки (пропуск лок. минимума). Поэтому вес требуется изменить в меньшую сторону. В противоположной ситуации – вес нужно увеличить.

Модификации - Rprop

В этом случае порядок операций будет таковым:

- определение значения коррекции;

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ \Delta_{ij}^{(t)}, & \frac{\partial E^{(t)}}{\partial w_{ij}} \frac{\partial E^{(t-1)}}{\partial w_{ij}} > 0 \\ \eta^- \Delta_{ij}^{(t)}, & \frac{\partial E^{(t)}}{\partial w_{ij}} \frac{\partial E^{(t-1)}}{\partial w_{ij}} < 0 \end{cases}$$

$$0 < \eta^- < 1 < \eta^+$$

Модификации - Rprop

В этом случае порядок операций будет таковым:

- определение значения коррекции;

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ \Delta_{ij}^{(t)}, & \frac{\partial E^{(t)}}{\partial w_{ij}} \frac{\partial E^{(t-1)}}{\partial w_{ij}} > 0 \\ \eta^- \Delta_{ij}^{(t)}, & \frac{\partial E^{(t)}}{\partial w_{ij}} \frac{\partial E^{(t-1)}}{\partial w_{ij}} < 0 \end{cases}$$

$$0 < \eta^- < 1 < \eta^+$$

Если на текущем шаге частная производная по соответствующему весу w_{ij} поменяла свой знак, то это говорит о том, что последнее изменение было большим, и алгоритм проскочил локальный минимум (соответствующую теорему можно найти в любом учебнике по математическому анализу), и, следовательно, величину изменения необходимо уменьшить на η и вернуть предыдущее значение весового коэффициента: другими словами необходимо произвести 'откат'.

$$\Delta w_{ij}(t) = \Delta w_{ij}(t) - \Delta_{ij}^{(t-1)}$$

Модификации - Rprop

Если знак частной производной не изменился, то нужно увеличить величину коррекции на $\eta+$ для достижения более быстрой сходимости.
Зафиксировав множители $\eta-$ и $\eta+$, можно отказаться от глобальных параметров настройки нейронной сети, что также можно рассматривать как преимущество рассматриваемого алгоритма перед стандартным алгоритмом Backprop.
Рекомендованные значения для
$\eta-= 0.5$, $\eta+= 1.2$,
но нет никаких ограничений на использование других значений для этих параметров.
Для того, чтобы не допустить слишком больших или малых значений весов, величину коррекции ограничивают сверху максимальным Δmax и снизу минимальным Δmin значениями величины коррекции, которые по умолчанию, соответственно, устанавливаются равными 50 и 1.0E-6.

начальные значения для всех Δij устанавливаются равными 0.1.

Модификации - Rprop

В этом случае порядок операций будет таковым:

- определение значения коррекции;
- расчет частных производных;

Модификации - Rprop

В этом случае порядок операций будет таковым:

- определение значения коррекции;
- расчет частных производных;
- расчет новой величины коррекции весовых значений;

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}^{(t)}, & \frac{\partial E^{(t)}}{\partial w_{ij}} > 0 \\ +\Delta_{ij}^{(t)}, & \frac{\partial E^{(t)}}{\partial w_{ij}} < 0 \\ 0, & \frac{\partial E^{(t)}}{\partial w_{ij}} = 0 \end{cases}$$

Модификации - Rprop

В этом случае порядок операций будет таковым:

- определение значения коррекции;
- расчет частных производных;
- расчет новой величины коррекции весовых значений;
- корректировка весов.

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t),$$

Если условие остановки алгоритма не исполняется, то происходит возврат к расчету производных, и цикл запускается по новому кругу.

Основы нейронных сетей

Лекция 8

Алгоритм обратного распространения ошибки. Оптимизации и модификации

Структура сети

Минимизация размера сети без потери производительности

Два подхода:

- наращивание сети
- упрощение сети

Структура сети. Наращивание

Идея: простая структура, изначально недостаточная для решения задача. Добавляется нейронов или слой, если ошибка перестает уменьшаться при обучении

Примеры реализации:

1. Метод каскадный корреляции:

- входной и выходной слой
- нет скрытых слоев

Структура сети. Наращивание

- добавление скрытых нейронов по одному, веса входного слоя не изменяются, только выходные и добавленные.

Пока не достигается нужной производительности сети

Структура сети. Наращивание

2. Адаптация аоро:
- 2 прохода
- 3-ий проход - адаптация структурного уровня: если ошибка > заданного значения, то добавляется нейрон, причем в нужное место по принципу:
- если веса колеблются, то этому нейрону не хватает вычислительной мощности и добавляется еще нейрон
- нейрон может удаляться,если если он избыточен
Ресурсоемкость

Структура сети

Второй подход - упрощение структуры сети. Два метода:

1. Регуляризация сложности
2. Удаление синаптических весов (dropout – на практике)

Структура сети. Регуляризация сложности

1. Достоверность данных обучения
2. Качество модели

=> минимизация общего риска - теория регуляризации Тихонова:

$$R(\mathbf{w}) = \mathbf{E}_s(\mathbf{W}) + \lambda \mathbf{E}_c(\mathbf{w}).$$

$\mathbf{E}_c(\mathbf{w})$. - штраф за сложность

Структура сети. Регуляризация сложности

- λ - параметр регуляризации

Если =0, => архитектура сети определяется примерами обучения
Если достаточно большое, то играет роль для определения архитектуры

Структура сети. Регуляризация сложности

Один из вариантов

$$\mathbf{E}_c(\mathbf{w}, k) = \frac{1}{2} \int \left\| \frac{\partial^k}{\partial \mathbf{x}^k} F(\mathbf{x}, \mathbf{w}) \right\|^2 \mu(\mathbf{x}) d\mathbf{x},$$

где $F(\mathbf{x}, \mathbf{w})$ — выполняемое моделью отображение входа на выход; $\mu(\mathbf{x})$ — некоторая весовая функция, определяющая область входного пространства, на которой функция $F(\mathbf{x}, \mathbf{w})$ должна быть гладкой. Это делается для того, чтобы k-я производная функции $F(\mathbf{x}, \mathbf{w})$ по входному вектору $\mathbf{x}$ принимала малое значение. Чем больше величина k, тем более гладкой (т.е. менее сложной) будет функция $F(\mathbf{x}, \mathbf{w})$.

3 процедуры регуляризации

Структура сети. Регуляризация сложности

1. Снижение весов

$$\mathbf{E}_c(\mathbf{w}) = \|\mathbf{w}\|^2 = \sum_{i \in \mathbf{C}_{\text{total}}} w_i^2$$

Все веса на 2 группы:

- важные
- избыточные (имеют малое влияние) --> 0

Простота, иногда хорошо работает

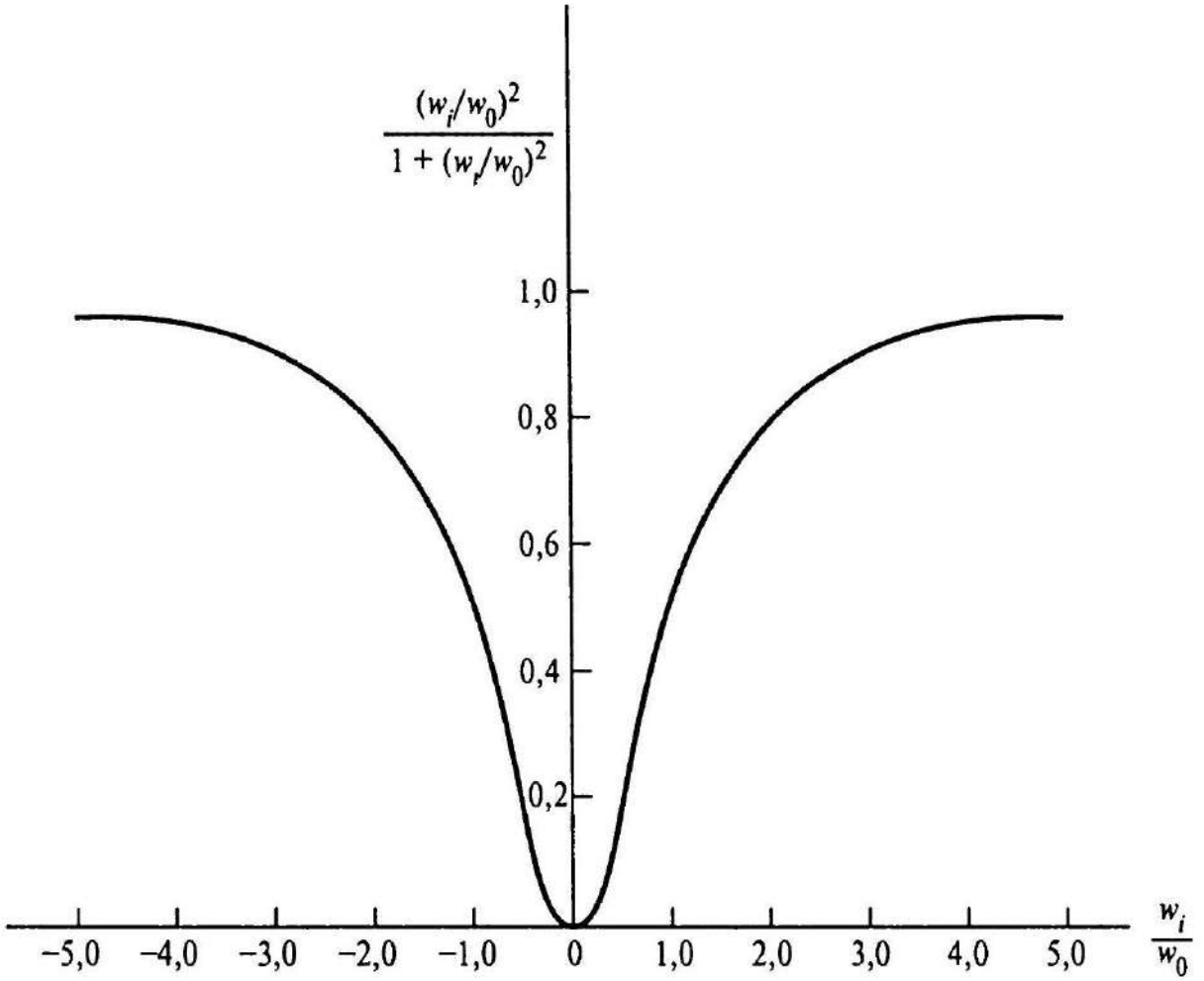
Структура сети. Регуляризация сложности

2. Исключение весов

$$\mathbf{E}_c(\mathbf{w}) = \sum_{i \in \mathbf{C}_{\text{total}}} \frac{(w_i/w_0)^2}{1 + (w_i/w_0)^2}$$

где w_0 — некоторый предопределенный параметр; w_i — вес i-го синапса сети. Под $\mathbf{C}_{total}$ понимается множество всех синаптических связей сети. Отдельные слагаемые штрафа за сложность симметричны относительно частного w_i/w_0, что наглядно показано на рис. Если $|w_i| >> w_0$, штраф за сложность (стоимость) для этого веса достигает максимального значения — единицы. Это значит, что вес w_i имеет высокую ценность для процесса обучения методом обратного распространения.

Структура сети. Регуляризация сложности



Структура сети. Регуляризация сложности

3-ий способ:

$$\mathbf{E}_c(\mathbf{w}) = \sum_{j=1}^{M} w_{oj}^2 \, \|\mathbf{w}_j\|^p,$$

где w_{oj} — веса выходного слоя; $\mathbf{w}_j$ — весовой вектор j-го нейрона скрытого слоя, а показатель степени p определяется выражением

$$p = \begin{cases} 2k - 1 \text{ для глобального сглаживания,} \\ 2k \text{ для локального сглаживания,} \end{cases}$$

где k — порядок дифференцирования функции $F(\mathbf{x}, \mathbf{w})$ по $\mathbf{x}$.

Структура сети. Регуляризация сложности

Выполняет 2 функции:

1. Разделяет роли весов нейронов скрытого и выходного слоев
2. Отслеживает взаимодействие этих двух множеств весов

Ресурсоемкость

Пример

Структура сети. Упрощение на основе Гессиана

для аналитического прогнозирования эффекта изменения синаптических весов строится локальная модель поверхности ошибок. Создание такой модели начинается с локальной аппроксимации функции стоимости $\mathbf{E}_{\text{av}}$ с помощью *ряда Тейлора* (Taylor series) в окрестности выбранной точки:

$$\mathbf{E}_{\text{av}}(\mathbf{w}+\Delta\mathbf{w}) = \mathbf{E}_{\text{av}}(\mathbf{w}) + \mathbf{g}^T(\mathbf{w})\Delta\mathbf{w} + \frac{1}{2}\Delta\mathbf{w}^T\mathbf{H}\Delta\mathbf{w} + O(\|\Delta\mathbf{w}\|^3),$$

где $\Delta\mathbf{w}$ — возмущение, применяемое к данной точке $\mathbf{w}$; $\mathbf{g}(\mathbf{w})$ — вектор градиента,

Структура сети. Упрощение на основе Гессиана

Требуется определить множество параметров, исключение которых из многослойного персептрона вызовет минимальное увеличение функции стоимости $\mathbf{E}_{av}$. Для того чтобы решить эту задачу на практике, необходимо выполнить следующее.

1. *Внешняя аппроксимация* (external approximation). Предполагается, что параметры удаляются из сети только после сходимости процесса обучения (т.е. после *полного обучения сети* (fully trained)). Следствие этого допущения состоит в том, что некоторые значения параметров соответствуют локальным или глобальным минимумам поверхности ошибки. В таком случае вектор градиента $\mathbf{g}$ можно считать равным нулю и слагаемое $\mathbf{g}^T \Delta \mathbf{w}$ в правой части можно не учитывать. В противном случае мера выпуклости (которая будет определена позже) для данной задачи может быть определена неверно.
2. *Квадратичная аппроксимация* (quadratic approximation). Предполагается, что поверхность ошибок в окрестности глобального или локального минимума является приблизительно "квадратичной". Исходя из этого, в выражении можно также не учитывать все слагаемые более высокого порядка.

Структура сети. Упрощение на основе Гессиана

Тогда упрощается до вида:

$$\Delta \mathbf{E}_{\mathbf{av}} = \mathbf{E}(\mathbf{w} + \Delta \mathbf{w}) - \mathbf{E}(\mathbf{w}) = \frac{1}{2} \Delta \mathbf{w}^T \mathbf{H} \Delta \mathbf{w}.$$

Структура сети. Упрощение на основе Гессиана

Целью OBS является обнуление одного из синаптических весов для минимизации приращения функции стоимости $\mathbf{E}_{\mathrm{av}}$ Пусть $w_i(n)$ — некоторый синаптический вес связи. Удаление этого веса эквивалентно выполнению условия

$$\Delta w_i + w_i = 0$$

или

$$\mathbf{1}_i^T \Delta \mathbf{w} + w_i = 0,$$

где $\mathbf{1}_i$ — единичный вектор, все элементы которого равны нулю за исключением i-го, который равен единице. Теперь цель OBS можно определить в следующем виде

Структура сети. Упрощение на основе Гессиана

$$\Delta \mathbf{E}_{\mathrm{av}} = \mathbf{E}(\mathbf{w} + \Delta \mathbf{w}) - \mathbf{E}(\mathbf{w}) = \frac{1}{2} \Delta \mathbf{w}^T \mathbf{H} \Delta \mathbf{w}.$$

Необходимо минимизировать квадратичную форму $\frac{1}{2} \Delta \mathbf{w}^T \mathbf{H} \Delta \mathbf{w}$ *по отношению к приращению вектора весов* $\Delta \mathbf{w}$*, принимая в качестве ограничения равенство нулю выражения* $\mathbf{1}_i^T \Delta \mathbf{w} + w_i$*. После этого требуется минимизировать результат относительно индекса* i*.*

Из этого определения вытекает наличие двух уровней минимизации. Одна процедура минимизации выполняется по векторам синаптических весов, оставшихся после обнуления i-го вектора весов. Вторая операция минимизации определяет, какой из векторов можно исключить.

Структура сети. Упрощение на основе Гессиана

Для решения этой *задачи условной оптимизации* (constrained optimization problem) в первую очередь нужно построить *Лагранжиан* (Lagrangian):

$$S = \frac{1}{2}\Delta \mathbf{w}^T \mathbf{H} \Delta \mathbf{w} - \lambda(\mathbf{1}_i^T \Delta \mathbf{w} + w_i),$$

где λ — *множитель Лагранжа*

Дифференцируя S по $\triangle$w, учитывая условие

$$\mathbf{1}_i^T \Delta \mathbf{w} + w_i = 0,$$

Структура сети. Упрощение на основе Гессиана

оптимальное приращение вектора весов $\mathbf{w}$ вычисляется по следующей формуле:

$$\Delta \mathbf{w} = -\frac{w_i}{[\mathbf{H}^{-1}]_{i,i}} \mathbf{H}^{-1} \mathbf{1}_i,$$

а соответствующее ему оптимальное значение Лагранжиана для элемента w_i — по формуле

$$S_i = \frac{w_i^2}{2\,[\mathbf{H}^{-1}]_{i,i}},$$

где $\mathbf{H}^{-1}$ — матрица, обратная Гессиану $\mathbf{H}$; $[\mathbf{H}^{-1}]_{i,i}$ — элемент с индексом i, i этой обратной матрицы. Лагранжиан S_i, оптимизированный по $\Delta \mathbf{w}$, подлежащий ограничению при исключении i-го синаптического веса w_i, называется *степенью выпуклости* w_i. Выпуклость S_i описывает рост среднеквадратической ошибки (как меры эффективности), вызываемый удалением синаптического веса w_i.

Структура сети. Упрощение на основе Гессиана

- веса с малыми значениями оказывает слабое влияние на ошибку, но

величина S_i также обратно пропорциональна диагональным элементам матрицы, обратной Гессиану. Таким образом, если величина $[\mathbf{H}^{-1}]_{i,i}$ мала, то даже небольшие веса могут оказывать существенное влияние на среднеквадратическую ошибку.

Для удаления выбираются веса, соответствующие наименьшей степени выпуклости

Структура сети. Упрощение на основе Гессиана

Предположение: сеть обучена, для упрощения 1 выходной нейрон

$$\mathbf{E}_{\mathbf{av}}(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}(d(n) - o(n))^2$$

$$o(n) = F(\mathbf{w}, \mathbf{x}),$$

Структура сети. обратная матрица

$$\frac{\partial \mathbf{E}_{\mathbf{av}}}{\partial \mathbf{w}} = -\frac{1}{N}\sum_{n=1}^{N}\frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}}(d(n) - o(n)),$$

$$\mathbf{H}(N) = \frac{\partial^2 \mathbf{E}_{\mathbf{av}}}{\partial \mathbf{w}^2} = \frac{1}{N}\sum_{n=1}^{N}\left\{\left(\frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}}\right)\left(\frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}}\right)^T - \right.$$
$$\left. - \left(\frac{\partial^2 F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}^2}\right)(d(n) - o(n))\right\}.$$

Структура сети. обратная матрица

$$\mathbf{H}(N) = \frac{\partial^2 \mathbf{E}_{\text{av}}}{\partial \mathbf{w}^2} = \frac{1}{N} \sum_{n=1}^{N} \left\{ \left(\frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}} \right) \left(\frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}} \right)^T - \right.$$
$$\left. - \left(\frac{\partial^2 F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}^2} \right) (d(n) - o(n)) \right\}.$$

- зависимость от N
- можно не учитывать второе слагаемое, так как сеть обучена

Структура сети. обратная матрица

$$\mathbf{H}(N) \simeq \frac{1}{N} \sum_{n=1}^{N} \left(\frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}} \right) \left(\frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}} \right)^{T}$$

Структура сети. обратная матрица

$$\mathbf{H}(N) \simeq \frac{1}{N} \sum_{n=1}^{N} \left(\frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}} \right) \left(\frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}} \right)^{T}$$

Для упрощения записи введем вектор размерности $W \times 1$:

$$\boldsymbol{\xi}(n) = \frac{1}{\sqrt{N}} \frac{\partial F(\mathbf{w}, x(n))}{\partial \mathbf{w}},$$

Структура сети. обратная матрица

$$\mathbf{H}(N) \simeq \frac{1}{N} \sum_{n=1}^{N} \left(\frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}} \right) \left(\frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}} \right)^T$$

Для упрощения записи введем вектор размерности $W \times 1$:

$$\boldsymbol{\xi}(n) = \frac{1}{\sqrt{N}} \frac{\partial F(\mathbf{w}, x(n))}{\partial \mathbf{w}},$$

$$\mathbf{H}(n) = \sum_{k=1}^{n} \boldsymbol{\xi}(k) \boldsymbol{\xi}^T(k) = H(n-1) + \boldsymbol{\xi}(n) \boldsymbol{\xi}^T(n), \quad n = 1, 2, \ldots, N.$$

Структура сети. обратная матрица

Лемма об инвертировании матриц или равенство Вудбурри

Пусть **A** и **B** — две положительно определенные матрицы, удовлетворяющие соотношению

$$\mathbf{A} = \mathbf{B}^{-1} + \mathbf{C}\mathbf{D}\mathbf{C}^{T},$$

где **C** и **D** — две другие матрицы. Согласно лемме об инвертировании матриц, матрица, обратная **A**, определяется соотношением

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{B}\mathbf{C}(\mathbf{D} + \mathbf{C}^{T}\mathbf{B}\mathbf{C})^{-1}\mathbf{C}^{T}\mathbf{B}.$$

Структура сети. обратная матрица

$$\mathbf{A} = \mathbf{H}(n),$$
$$\mathbf{B}^{-1} = \mathbf{H}(n-1),$$
$$\mathbf{C} = \boldsymbol{\xi}(n),$$
$$\mathbf{D} = 1.$$

Структура сети. обратная матрица

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{BC}(\mathbf{D} + \mathbf{C}^T\mathbf{BC})^{-1}\mathbf{C}^T\mathbf{B}.$$

$$\begin{aligned}
\mathbf{A} &= \mathbf{H}(n), \\
\mathbf{B}^{-1} &= \mathbf{H}(n-1), \\
\mathbf{C} &= \boldsymbol{\xi}(n), \\
\mathbf{D} &= 1.
\end{aligned}$$

$$\mathbf{H}^{-1}(n) = \mathbf{H}^{-1}(n-1) - \frac{\mathbf{H}^{-1}(n-1)\boldsymbol{\xi}(n)\boldsymbol{\xi}^T(n)\mathbf{H}^{-1}(n-1)}{1 + \boldsymbol{\xi}^T(n)\mathbf{H}^{-1}(n-1)\boldsymbol{\xi}(n)}.$$

Структура сети. обратная матрица

Рекурсивные вычисления продолжаются до тех пор, пока не будет обработано все множество из N примеров. Для инициализации этого алгоритма необходимо задать достаточно большое начальное приближение $\mathbf{H}^{-1}(0)$, так как на каждом шаге рекурсии элементы матрицы будут только убывать. Это требование можно удовлетворить с помощью следующего значения матрицы:

$$\mathbf{H}^{-1}(0) = \delta^{-1}\mathbf{I},$$

где δ — достаточно малое положительное число; $\mathbf{I}$ — единичная матрица. Такая форма исходного значения гарантирует, что матрица $\mathbf{H}^{-1}(n)$ всегда будет положительно определенной. Влияние δ прогрессивно уменьшается по мере подачи все большего количества примеров.

1. Минимизируем среднеквадратическую ошибку в процессе обучения данного многослойного персептрона.
2. $$\boldsymbol{\xi}(n) = \frac{1}{\sqrt{N}} \frac{\partial F(\mathbf{w},\mathbf{x}(n))}{\partial \mathbf{w}},$$
   где $F(\mathbf{x},\mathbf{w})$ — отображение "вход-выход", реализуемое многослойным персептроном с вектором синаптических весов $\mathbf{w}$ и вектором входного сигнала $\mathbf{x}$.
3. С помощью рекурсивного выражения вычисляем матрицу, обратную Гессиану ($\mathbf{H}^{-1}$).
4. Находим некоторое i, соответствующее наименьшей степени выпуклости
   $$S_i = \frac{w_i^2}{2[\mathbf{H}^{-1}]_{i,i}},$$
   где $[\mathbf{H}^{-1}]_{i,i}$ — i,i-й элемент этой матрицы $\mathbf{H}^{-1}$. Если степень выпуклости S_i гораздо меньше, чем среднеквадратическая ошибка $\mathbf{E}_{\text{av}}$, то данный синаптический вес w_i удаляется, и мы снова переходим к шагу 4. В противном случае переходим к шагу 5.
5. Изменяем все синаптические веса сети, применяя приращение
   $$\Delta \mathbf{w} = -\frac{w_i}{[\mathbf{H}^{-1}]_{i,i}} \mathbf{H}^{-1} \mathbf{1}_i,$$
   переходим к шагу 2.
6. Процесс вычисления прекращается, если больше не существует весов, которые можно удалить из сети, существенно не увеличив среднеквадратическую ошибку (в этот момент сеть желательно обучить заново).

Преимущества и ограничения АОРО

1. Связность

Вычисления в каждом нейрона выполняются обособленно

Плюсы: подходит для параллельной архитектуры

Плавное снижение производительности при сбоях, отказоустойчивость +-

Преимущества и ограничения АОРО

2. Извлечение признаков

Скрытые слоя - детекторы признаков

Можно использовать в картах идентичности, сжатие данных

Преимущества и ограничения АОРО

3. Универсальный аппроксиматор

Даже для кусочно-дифференцируемых функций

4. Вычислительно эффективный алгоритм

O (W)

Преимущества и ограничения АОРО

5. Медленная сходимость

Если поверхность ошибок - плоская, то изменения малы, нужно много итераций

Если поверхность искривлена, то изменения велики и можно пропустить локальный минимум

Неправильное направление градиента

+: другие методы не всегда лучше в реальных условиях

Преимущества и ограничения АОРО

6. Локальные минимумы

7. Лучше не использовать полносвязность на основе априорной информации

8. Скорость обучения

Скорость <= min (1/количество входов нейрона в слое)

Некоторые методы оптимизации:

- Annealing
- Stochastic Gradient Descent
- AW-SGD
- Momentum (SGD или RMSprop)
- Nesterov Momentum (SGD)
- AdaGrad
- AdaDelta
- ADAM
- BFGS
- LBFGS

Основы нейронных сетей

Лекция 10

Рекуррентные сети

Классификация

- с учителем
- без учителя
- их сочетание - веса настраивают однократно на основе информации извне, то есть запоминает образцы до поступления реальных данных и не меняется в процессе!

Сети Хопфилда и Хемминга для организации ассоциативной памяти

Рекуррентные сети, обратная связь

Ассоциативная память (АП)

Задача. Известен набор из m двоичных сигналов - образцы (изображения, оцифрованный звук, данные описывающие характеристики объектов или процессов) $X^1,...X^m$.

На вход подается неидеальный зашумленный сигнал. Требуется либо восстановить его, либо определить класс принадлежности.

К какому из образцов ближе поданный сигнал?

АП- система, определяющая взаимную зависимость векторов

- если компоненты одного и того же вектора, то говорят об **автоассоциативной памяти (**сеть Хопфилда **)**
- 2 различных вектора, то **память гетероассоциативного типа** (сеть Хемминга и ДАП [BAM-Bidirectional Associative Memory])

Мера близости отдельных множеств

Расстояние Хемминга - число несовпавших компонент двух векторов $y=(y_1,...,y_n)$ и $d=(d_1,...,d_n)$

$$d_H(y,d)=\sum_{i=1}^{n}(d_i(1-y_i)+y_i(1-d_i))$$

Если используются двоичные значения 0 и 1

$$d_H(y,d)=\frac{1}{2}(n-\sum_{i=1}^{n}d_i y_i)$$

Если используются биполярные значения -1 и 1

Если y=d, то мера Хемминга =0

Сеть Хопфилда

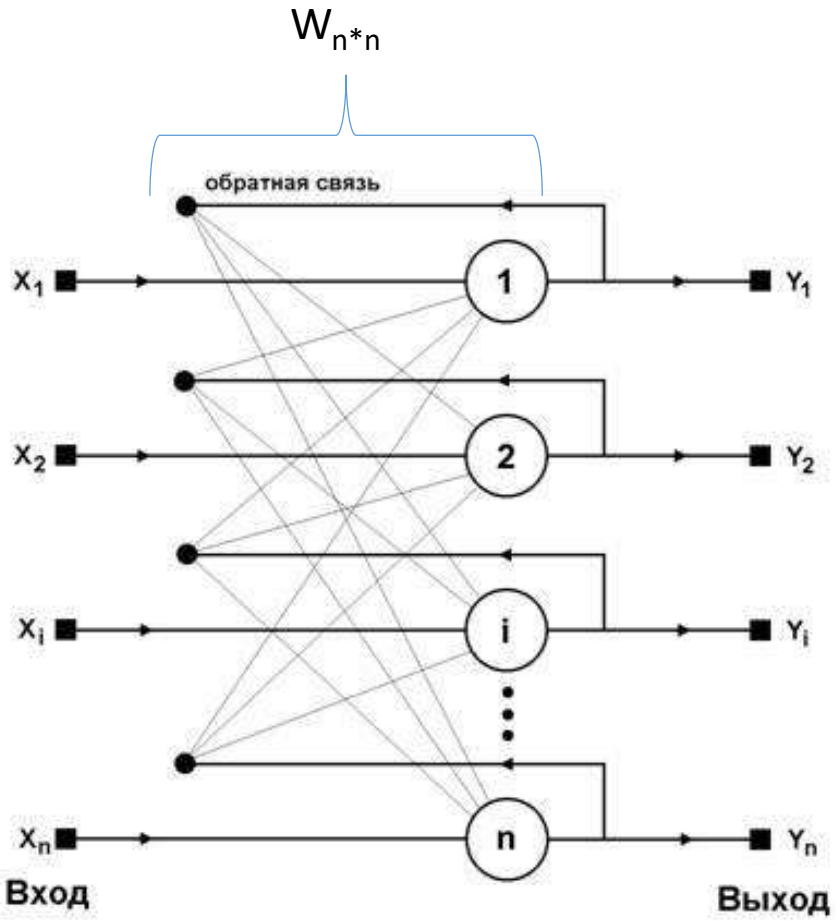
Задача. Дан набор из m образцов

при этом $x_i^k = \begin{cases} 1 \\ -1 \end{cases}, i = 1..n, k = 1..m$

$$X = \begin{matrix} X^1 \\ \dots \\ X^m \end{matrix} = \begin{pmatrix} x_1^1 & x_2^1 & \dots & x_n^1 \\ x_1^2 & & \dots & x_n^2 \\ \dots & & & \dots \\ x_1^m & x_2^m & \dots & x_n^m \end{pmatrix}$$

Подается искаженный сигнал X^* , требуется восстановить сигнал, т.е. получить $X^к = X^*$. В случае неудачи – новый сигнал(не образец)

Сеть Хопфилда. Структура



Отсутствует автосвязь

Симметричная матрица весов $W=W^T$

На достижение устойчивого состояния влияет ещё режим работы сети

- Синхронный режим
- Асинхронный режим

только асинхронный режим работы сети гарантирует достижение устойчивого состояния сети, в синхронном случае возможно бесконечное переключение между двумя разными состояниями (такая ситуация называется динамическим аттрактором, в то время как устойчивое состояние принято называть статическим аттрактором)

Сеть Хопфилда

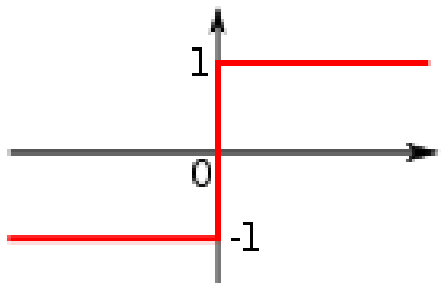
2 режима:

- Обучение – определяется W
- Классификация – подается искаженный сигнал X^* и вычисляется выходной сигнал Y

Сеть Хопфилда. Классификация

1. Y(0)=X*
2. Рассчитывается новое состояние $$y_j(t) = f(\sum_{i=1}^{n} w_{ij} y_i(t-1)), j = 1..n$$
3. Сравнивается Y(t) с Y(t-1) – мера Хемминга, если равны с точностью e, то стоп, иначе t+1, переход к ш2

f – функция активации



Остановка работы

Продолжаем, пока состояния Y(t) и Y(t+1) не совпадут (или, в случае синхронного режима работы, не совпадут Y(t-1) и Y(t+1)
и одновременно Y(t-2) и Y(t)) .
Именно этот процесс называется конвергенцией сети. Полученное устойчивое состояние Y(t) (статический аттрактор), или, возможно, в синхронном случае пара Y(t) и Y(t+1) (динамический аттрактор), является ответом сети на данный входной образ.

Остановка работы

Во время работы сети Хопфилда признаком нахождения решения является момент, когда достигается аттрактор, статический (когда на каждом следующем шаге повторяется устойчивое состояние Y(t)) или, возможно, динамический (когда до бесконечности чередуются два разных состояния Y(t) и Y(t+1)).

Это конечное состояние сети и является её реакцией на данный образ.

Остановка работы

Нормальным ответом является такое устойчивое состояние, которое совпадает с одним из запомненных при обучении векторов. Но при некоторых условиях (в частности, при слишком большом количестве запомненных образов) результатом работы может стать так называемый ложный аттрактор («химера»), состоящий из нескольких частей разных запомненных образов.

В синхронном режиме сеть может к тому же прийти к динамическому аттрактору.

Обе эти ситуации в общем случае являются нежелательными, поскольку не соответствуют ни одному запомненному вектору — а соответственно, не определяют класс, к которому сеть отнесла входной образ.

Сеть Хопфилда. Обучение Хебба

$$w_{ij} = \begin{cases} \frac{1}{n}\sum_{k=1}^{m} x_i^k x_j^k, i \neq j \\ 0, i = j \end{cases}$$

или

$$w_{ij} = \begin{cases} \sum_{k=1}^{m} x_i^k x_j^k, i \neq j \\ 0, i = j \end{cases}$$

Недостатки: емкость m=0,13n при e=0.01 до 0.15n

$$m \approx \frac{n}{2\ln n}$$

Хорошо запоминает взаимно ортогональные вектора или близкие к ним

Свойства

Важной характеристикой нейронной сети является отношение числа ключевых образов M, которые могут быть запомнены, к числу нейронов сети N:

$$a = M/N$$

Для сети Хопфилда значение a не больше 0.14.

Синхронный режим работы сети (редко)

Если работа сети моделируется на одном процессоре, то при синхронном режиме последовательно просматриваются нейроны, однако их состояния запоминаются отдельно и не меняются до тех пор, пока не будут пройдены все нейроны сети.

Когда все нейроны просмотрены, их состояния одновременно (то есть синхронно, отсюда и название) меняются на новые. Таким образом, достигается моделирование параллельной работы последовательным алгоритмом. (время передачи сигнала – одинаковое)

Асинхронный режим работы сети

Если моделировать работу сети как последовательный алгоритм, то в асинхронном режиме работы состояния нейронов в следующий момент времени меняются последовательно:

- вычисляется локальное поле для первого нейрона в момент t,
- определяется его реакция,
- нейрон устанавливается в новое состояние (которое соответствует его выходу в момент t + 1),
- Потом вычисляется локальное поле для второго нейрона с учётом нового состояния первого, меняется состояние второго нейрона, и так далее — состояние каждого следующего нейрона вычисляется с учетом всех изменений состояний рассмотренных ранее нейронов.

Асинхронный режим работы сети

вне зависимости от количества запомненных образов и начального состояния сеть непременно придёт к устойчивому состоянию

Сеть Хопфилда

Слишком быстро переходим в устойчивое состояние («химеры»)

- Периодически случайным образом добавляем полосы исходного изображения
- Добавляем ограничение на количество итераций

Сеть Хопфилда

Другой способ:

- Добавляем связи (анализ ПО+меняем формулу обучения)
- Добавляем слои (сеть Хемминга)
- Непрерывная функция активации (сигмоид)

Сеть Хопфилда (модификации). Обучение проекцией

Подобрать W, чтобы WX=X, тогда

$W=XX^{+}$ или, если образцы – линейно независимы, то упрощается до

$W=X(X^{T}X)^{-1}X^{T}$ или в итерационной форме для образцов X^{k}, k=1..m :

$W^{0}=0$

$Y^{k}=(W^{k-1}-E)X^{k}$

$Wk=W^{k-1}-(Y^{k}Y^{kT})/(Y^{kT}Y^{k})$

Емкость m увеличивается до n-1

Сеть Хопфилда

Модификация - Δ-проекция –градиентная форма алгоритма минимизации.

Многократно применяется на множестве образцов вплоть до стабилизации:

$W=W+(h/n)(X^k-WX^k)X^{kT}$, $h\in(0.7,0.9)$

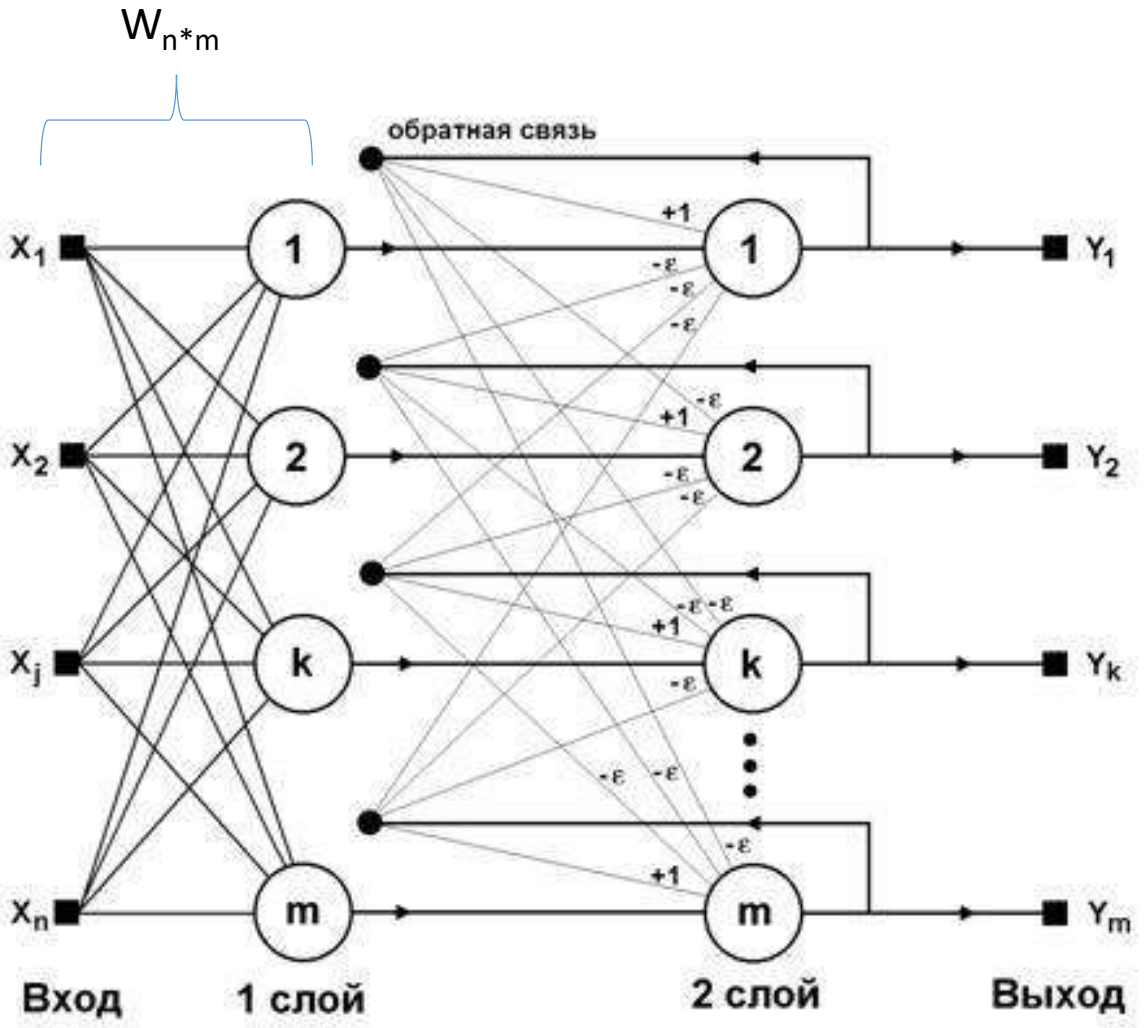
Реализация сети

- Предобработка образцов
- Создание сети (узлы или вся сеть?)
- Тестирование

Сеть Хемминга

- Бинарные сигналы
- Результат – номер класса, т.е. $Y=(y_1,\ldots,y_m)$, $y_j=1$, то j – номер класса сигнала X^*
- Меньше памяти

Сеть Хемминга. Структура



$$0 < \varepsilon \leq \frac{1}{m}$$

Сеть Хемминга.

- Обучение

$$w_{ij} = x_i^j, i = 1..n,\ j = 1..m$$

или

$$w_{ij} = \frac{x_i^j}{2}, i = 1..n,\ j = 1..m$$

Сеть Хемминга.

- Обучение

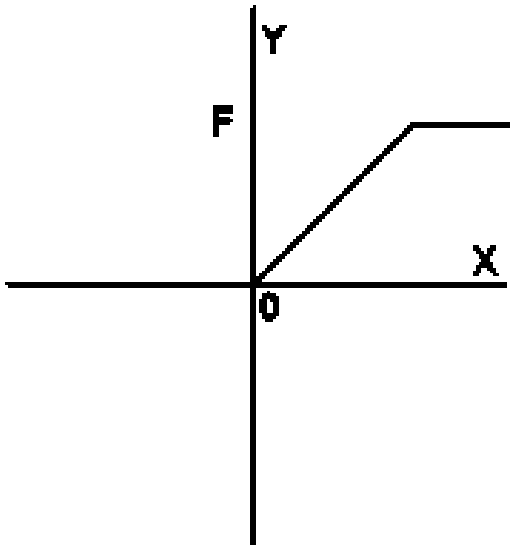
$$w_{ij} = x_i^j, i = 1..n,\ j = 1..m \text{ или } \qquad w_{ij} = \frac{x_i^j}{2}, i = 1..n,\ j = 1..m$$

- Классификация:
  первый слой

$$y_j^1 = f\left(\sum_{i=1}^{n} w_{ij} x_i^* + \frac{n}{2}\right), j = 1..m$$

S1

T



Сеть Хемминга.

- Обучение

$$w_{ij} = x_i^j, i = 1..n, j = 1..m \text{ или } w_{ij} = \frac{x_i^j}{2}, i = 1..n, j = 1..m$$

- Классификация:
  первый слой
  второй слой

$$y_j^2(0) = y_j^1$$

S^1

$$y_j^1 = f(\sum_{i=1}^{n} w_{ij} x_i^* + \frac{n}{2}), j = 1..m$$

S^2

$$y_j^2(t) = f(y_j^2(t-1) - \varepsilon \sum_{i=1, i \neq j}^{m} y_i^2(t-1)), j = 1..m$$

Сравнивается

$$y^2(t), y^2(t-1) \qquad \|y^2(t) - y^2(t-1)\| \leq E_{max}$$

Сеть Хемминга.

- Обучение

$$w_{ij} = x_i^j, i = 1..n, j = 1..m \text{ или } w_{ij} = \frac{x_i^j}{2}, i = 1..n, j = 1..m$$

- Классификация:
  первый слой
  второй слой

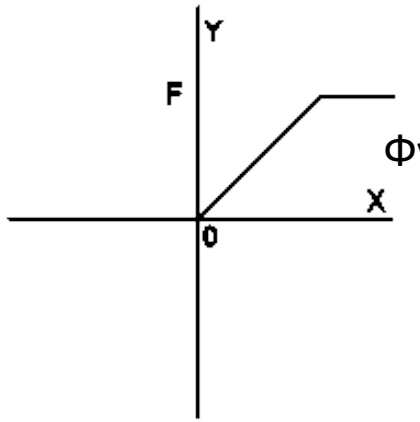
$$y_j^2(0) = y_j^1$$

S^1

$$y_j^1 = f(\sum_{i=1}^{n} w_{ij} x_i^* + \frac{n}{2}), j = 1..m$$

S^2

$$y_j^2(t) = f(y_j^2(t-1) - \varepsilon \sum_{i=1, i \neq j}^{m} y_i^2(t-1)), j = 1..m$$



Функция активации f

Сравнивается $\|y^2(t) - y^2(t-1)\| \leq E_{max}$

величина F=n/2 (большие значения и могут быть разными в слоях)

Сеть Хемминга.

- Идеальный выход – одно положительное значение
- Выходной сигнал – WTA «победитель забирает все»

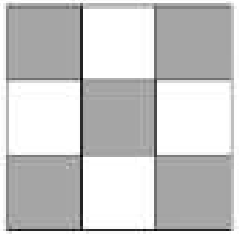
$$y_j = \begin{cases} 1, & y_j = \max_i(y_i) \\ 0, & в\ противном\ случае \end{cases}$$

Сеть Хемминга

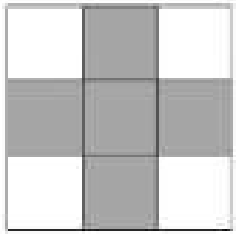
- Если данные входного образа сильно зашумлены или в обучающей выборке отсутствовал подходящий эталон, в результате остановки цикла могут быть получены несколько положительных выходов, причем значение любого из них окажется меньше, чем Emax.
- В этом случае делается заключение о невозможности отнесения входного образа к определенному классу, однако индексы положительных выходов указывают на наиболее схожие с ним эталоны.

Сеть Хемминга. Пример

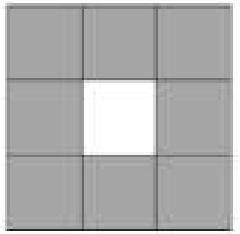
Образ 1



Образ 2



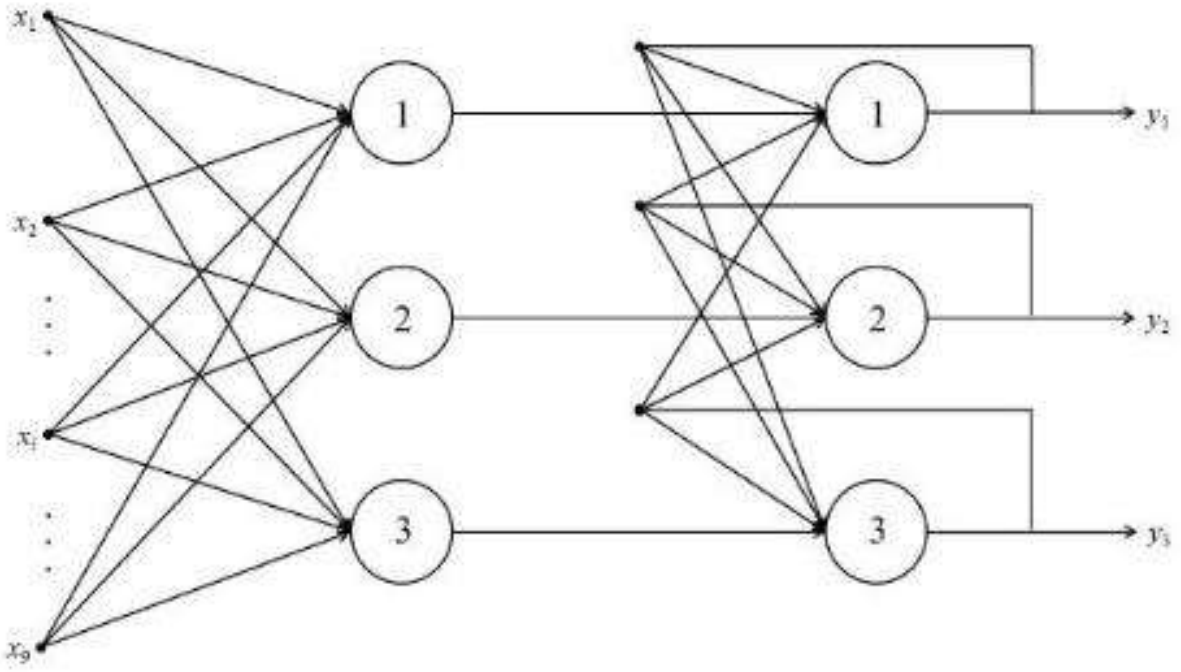
Образ 3



X=

№ образа	№ входной переменной								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	1	–1	1	–1	1	–1	1	–1	1
2	–1	1	–1	1	1	1	–1	1	–1
3	1	1	1	1	–1	1	1	1	1

Сеть Хемминга



Сеть Хемминга

W=

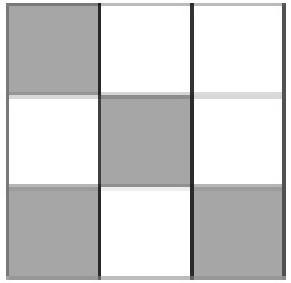
№ нейрона первого слоя	№ входной переменной								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	0,5	–0,5	0,5	–0,5	0,5	–0,5	0,5	–0,5	0,5
2	–0,5	0,5	–0,5	0,5	0,5	0,5	–0,5	0,5	–0,5
3	0,5	0,5	0,5	0,5	–0,5	0,5	0,5	0,5	0,5

$T = 4,5$ $\varepsilon = 0,3$ $E_{max} = 0,1$

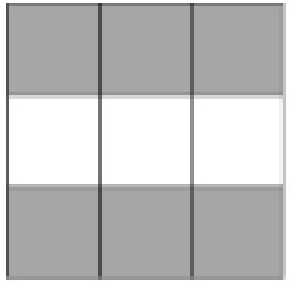
Сеть Хемминга

- Тестирование

Образ 4



Образ 5



Сеть Хемминга

$$\vec{x}^T = [1, -1, -1, -1, 1, -1, 1, -1, 1]$$

$$\vec{s}_1 = \begin{bmatrix} 8{,}00 \\ 2{,}00 \\ 3{,}00 \end{bmatrix}, \qquad \vec{y}_1 = \begin{bmatrix} 4{,}50 \\ 2{,}00 \\ 3{,}00 \end{bmatrix}.$$

- Первый слой
- Изменение сигналов в сети Хэмминга при подаче на входы образа 4

Номер итерации	Вектор состояний			Вектор выходов			$\|\vec{y}^{(q-1)} - \vec{y}^{(q)}\|$
	$s_{21}^{(q)}$	$s_{22}^{(q)}$	$s_{23}^{(q)}$	$y_{21}^{(q)}$	$y_{22}^{(q)}$	$y_{23}^{(q)}$	
1	8,00	2,00	3,00	4,50	2,00	3,00	–
2	3,00	–0,25	1,05	3,00	0,00	1,05	10,05
3	2,69	–1,22	0,15	2,69	0,00	0,15	0,91
4	2,64	–0,85	–0,66	2,64	0,00	0,00	0,02

Класс 1

Сеть Хемминга

- Образ 5 $\vec{x}^T = [1, 1, 1, -1, -1, -1, 1, 1, 1]$

Номер итерации	Вектор состояний			Вектор выходов			$\|\vec{y}^{(q-1)} - \vec{y}^{(q)}\|$
	$s_{21}^{(q)}$	$s_{22}^{(q)}$	$s_{23}^{(q)}$	$y_{21}^{(q)}$	$y_{22}^{(q)}$	$y_{23}^{(q)}$	
1	6,00	2,00	7,00	4,50	2,00	4,50	–
2	2,55	–0,70	2,55	2,55	0,00	2,55	11,61
3	1,79	–1,53	1,79	1,79	0,00	1,79	1,17
4	1,25	–1,07	1,25	1,25	0,00	1,25	0,57
5	0,87	–0,75	0,87	0,87	0,00	0,87	0,28
6	0,61	–0,52	0,61	0,61	0,00	0,61	0,14
7	0,43	–0,37	0,43	0,43	0,00	0,43	0,07

Основы нейронных сетей

Лекция 11

Рекуррентные сети

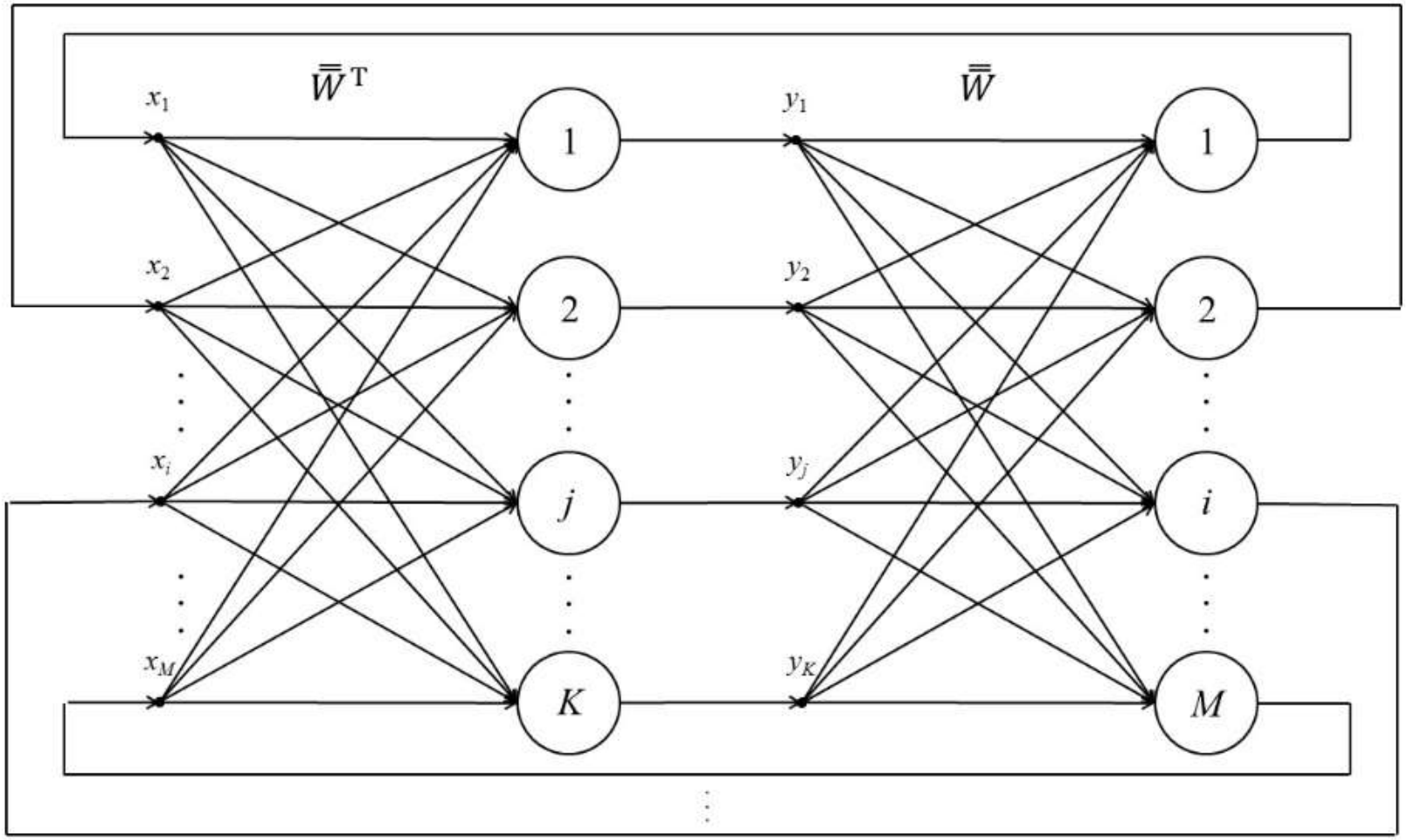
ДАП (Коско)

- НС гетероассоциативной памяти – сети Коско
- **Задача.** Известен набор из *N* эталонных входных образов X и соответствующих им *N* идеальных выходных образов Y.
- После обучения сеть Коско должна уметь из поданного на ее вход зашумленного входного вектора выделить один из заложенных в нее выходных идеальных образов или дать заключение, что входные данные не соответствуют ни одному из них.
- Таким образом, в отличие от нейронной сети Хопфилда, сеть двунаправленной ассоциативной памяти должна не только избавиться от шума во входном векторе, но и сопоставить соответствующий ему выходной вектор.

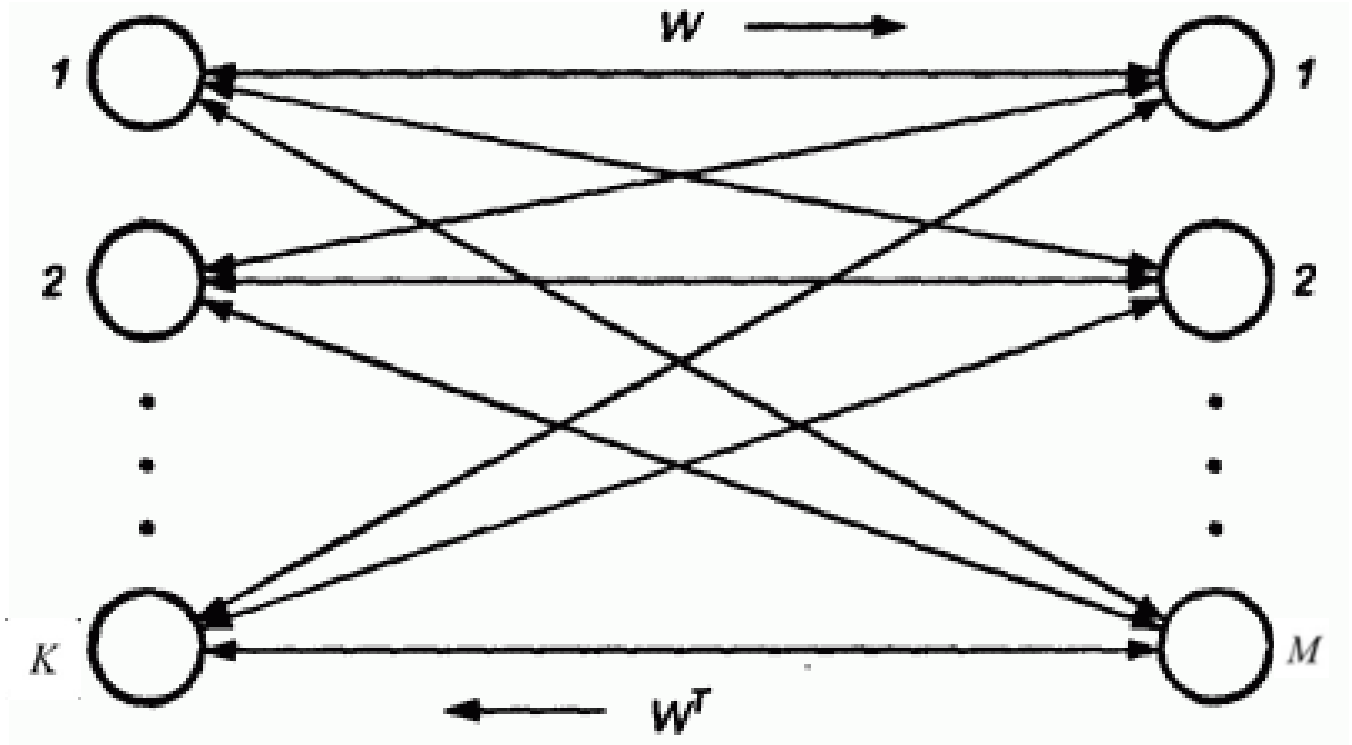
ДАП (Коско)

- Структура сети:
- 2 слоя
- Количество нейронов первого слоя *K* равно количеству бинарных выходных переменных.
- Число нейронов второго слоя *M* равно числу бинарных переменных, которые кодируют входной образ, соответствующий выходному.
- Сами значения входных и выходных переменных принадлежат бинарному множеству {–1; 1}

ДАП (Коско)



ДАП (Коско) – другая форма записи



ДАП (Коско)

- Обучение:
  - Формируется матрица ассоциированных входных образов X (N×M) – как сеть Хопфильда
  - Формируется матрица ассоциированных выходных образов Y (N×K)

ДАП (Коско)

- Обучение:
  - Формируется матрица ассоциированных входных образов X (N×M) – как сеть Хопфильда
  - Формируется матрица ассоциированных выходных образов Y (N×K)

№ образа	№ выходной бинарной переменной					
	1	2	…	*j*	…	*K*
1	y_{11}	y_{12}	…	y_{1j}	…	y_{1K}
2	y_{21}	y_{22}	…	y_{2j}	…	y_{2K}
…	…	…	…	…	…	…
k	y_{k1}	y_{k2}	…	y_{kj}	…	y_{kK}
…	…	…	…	…	…	…
N	y_{N1}	y_{N2}	…	y_{Nj}	…	y_{NK}

ДАП (Коско)

- Обучение:
  - Формируется матрица ассоциированных входных образов X (N×M) – как сеть Хопфильда
  - Формируется матрица ассоциированных выходных образов Y (N×K)
  - Вычисляется матрица $W=X^TY$ (M×K)

$$w_{ij} = \sum_{k=1}^{N} x_{ik} y_{kj}$$

ДАП (Коско)

Работа сети:

1. $x^{(0)}=x^{*}$- входной зашумленный сигнал

ДАП (Коско)

Работа сети:

1. $x^{(0)}=x^{*}$- входной зашумленный сигнал
2. Итерации по q
3. $s_{1j}^{(q+1)} = \sum_{i=1}^{M} w_{ji} x_i^{(q)}$

ДАП (Коско)

Работа сети:

1. $x^{(0)}=x^{*}$- входной зашумленный сигнал
2. Итерации по q
3. $s_{1j}^{(q+1)} = \sum_{i=1}^{M} w_{ji} x_i^{(q)}$
4. $y_j^{(q+1)}$=f($s_{1j}^{(q+1)}$+T) T – порог, может T=0

ДАП (Коско)

Работа сети:

1. $x^{(0)}=x^{*}$- входной зашумленный сигнал
2. Итерации по q
3. $s_{1j}^{(q+1)} = \sum_{i=1}^{M} w_{ji} x_i^{(q)}$
4. $y_j^{(q+1)}$=f($s_{1j}^{(q+1)}$+T) T – порог, может T=0

Если T – одинаково для всех – называется **гомогенной сетью Коско,** иначе негомогенной (когда входные или выходные вектора содержат различные по своему происхождению элементы)

ДАП (Коско)

Работа сети:

1. $x^{(0)}=x^{*}$- входной зашумленный сигнал
2. Итерации по q
3. $s_{1j}^{(q+1)} = \sum_{i=1}^{M} w_{ji} x_i^{(q)}$
4. $y_j^{(q+1)}$=f($s_{1j}^{(q+1)}$+T) T – порог, может T=0

f – пороговая функция

$$f(x) = \begin{cases} 1, x > 0 \\ 0, x < 0 \\ f(x) \text{предыдущей итерации}, x = 0 \end{cases}$$

ДАП (Коско)

Работа сети:

1. $x^{(0)}=x^{*}$- входной зашумленный сигнал
2. Итерации по q
3. $s_{1j}^{(q+1)} = \sum_{i=1}^{M} w_{ji} x_i^{(q)}$
4. $y_j^{(q+1)}$=f($s_{1j}^{(q+1)}$+T) T – порог, может T=0

f – пороговая функция

Или сигмоид $f(x) = \frac{1}{1+\exp(-\alpha x)}$если a – большое, тогда это ≈ пороговая

ДАП (Коско)

Работа сети:

1. $x^{(0)}=x^{*}$- входной зашумленный сигнал
2. Итерации по q
3. $s_{1j}^{(q+1)} = \sum_{i=1}^{M} w_{ji} x_i^{(q)}$
4. $y_j^{(q+1)}$=f($s_{1j}^{(q+1)}$+T) T – порог
5. Для всех итераций, кроме первой ($q \neq 1$), проверяется условие окончания – стабилизация элементов выходного вектора. Если оно не выполняется, цикл расчетов продолжается с п. 6, иначе – переход к п. 8.

ДАП (Коско)

Работа сети:

1. $x^{(0)}=x^{*}$- входной зашумленный сигнал
2. Итерации по q
3. $s_{1j}^{(q+1)} = \sum_{i=1}^{M} w_{ji} x_i^{(q)}$
4. $y_j^{(q+1)}$=f($s_{1j}^{(q+1)}$+T) T – порог
5. Для всех итераций, кроме первой ($q \neq 1$), проверяется условие окончания – стабилизация элементов выходного вектора. Если оно не выполняется, цикл расчетов продолжается с п. 6, иначе – переход к п. 8.
6. Полученные значения элементов выходного образа подаются на входы нейронов второго слоя

$s_{2i}^{(q+1)} = \sum_{j=1}^{K} w_{ij} y_j^{(q+1)}$ или $s_2^{(q+1)} = w \quad y^{(q+1)}$ - векторная форма

ДАП (Коско)

Работа сети:

1. $x^{(0)}=x^{*}$- входной зашумленный сигнал
2. Итерации по q
3. $s_{1j}^{(q+1)} = \sum_{i=1}^{M} w_{ji} x_i^{(q)}$
4. $y_j^{(q+1)}=f(s_{1j}^{(q+1)}+T)$ T – порог
5. Для всех итераций, кроме первой ($q \neq 1$), проверяется условие окончания – стабилизация элементов выходного вектора. Если оно не выполняется, цикл расчетов продолжается с п. 6, иначе – переход к п. 8.
6. Полученные значения элементов выходного образа подаются на входы нейронов второго слоя

$s_{2i}^{(q+1)} = \sum_{j=1}^{K} w_{ij} y_j^{(q+1)}$ или $s_2^{(q+1)} = w \quad y^{(q+1)}$ - векторная форма

7. $x_i^{(q+1)}=f(s_{2i}^{(q+1)})$

ДАП (Коско)

Работа сети:

1. $x^{(0)}=x^{*}$- входной зашумленный сигнал
2. Итерации по q
3. $s_{1j}^{(q+1)} = \sum_{i=1}^{M} w_{ji} x_i^{(q)}$
4. $y_j^{(q+1)}$=f($s_{1j}^{(q+1)}$+T) T – порог
5. Для всех итераций, кроме первой ($q \neq 1$), проверяется условие окончания – стабилизация элементов выходного вектора. Если оно не выполняется, цикл расчетов продолжается с п. 6, иначе – переход к п. 8.
6. Полученные значения элементов выходного образа подаются на входы нейронов второго слоя

$s_{2i}^{(q+1)} = \sum_{j=1}^{K} w_{ij} y_j^{(q+1)}$ или $s_2^{(q+1)} = w \quad y^{(q+1)}$ - векторная форма

7. $x_i^{(q+1)}$=f($s_{2i}^{(q+1)}$)

8. Проверяется соответствие стабилизировавшегося вектора выходных значений одному из идеальных образов матрицы Y . Если соответствие установлено, можно сделать вывод, что нейронной сети удалось найти ассоциацию между зашумленным входом и одним из идеальных выходов. В противном случае можно сказать, что входной вектор был слишком сильно зашумлен и соответствие не может быть установлено.

ДАП (Коско)

Работа сети (развёртка) до стабилизации

$$f(x_0\mathbf{W}) = y_1 \to f(y_1\mathbf{W}^T) = x_1 \to f(x_1\mathbf{W}) = y_2 \to$$
$$\to f(y_2\mathbf{W}^T) = x_2 \to f(x_2\mathbf{W}) = y_3 \to$$
$$\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots$$
$$\to f(y_f\mathbf{W}^T) = x_f \to f(x_f\mathbf{W}) = y_f \ ,$$

ДАП (Коско)

$$f(x_0W) = y_1 \to f(y_1W^T) = x_1 \to f(x_1W) = y_2 \to$$
$$\to f(y_2W^T) = x_2 \to f(x_2W) = y_3 \to$$
$$\dots \quad \dots \quad \dots \quad \dots \quad \dots$$
$$\to f(y_fW^T) = x_f \to f(x_fW) = y_f \ ,$$

- Работа сети (кратко)
- Каждой промежуточной точке процесса (x_q,y_q) можно сопоставить энергетическую функцию , определяемую в виде
- $E_q=-x_qWy_q^t \to min$

ДАП (Коско)

- Синхронный режим
- Искусственная нейронная сеть Коско имеет ограничения, связанные с количеством запоминаемых образов. Если все они достаточно сильно друг от друга отличаются, в первом приближении достаточно выполнения условия: $N \leq n$, где n – минимальное из двух слоев количество нейронов. Предельно строгое условие аналогично неравенству для сети Хопфилда.
- В отличие от искусственной нейронной сети Хопфилда, в сетях Коско весьма сложно столкнуться с ситуацией зацикленного чередования двух различных образов в выходном векторе.

ДАП (Коско)

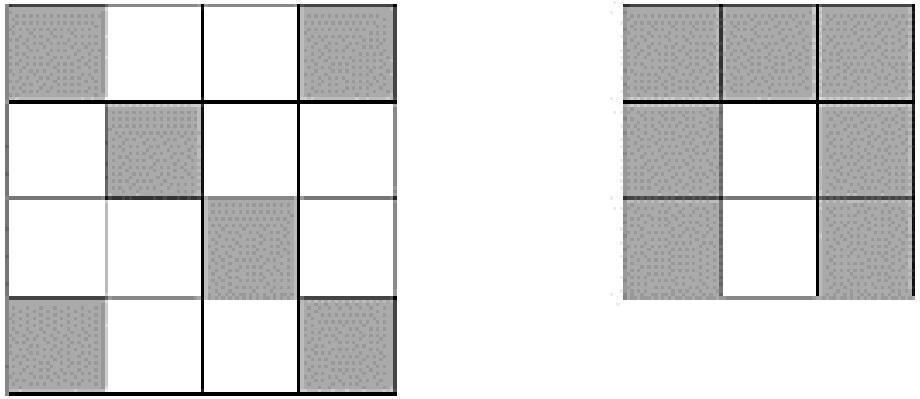
- Задачу ассоциации неоднородных образов с помощью нейронной сети Коско можно свести к задаче их классификации. В этом случае количество переменных выходного образа должно соответствовать количеству возможных классов.
- Для каждого эталонного образа класс кодируется так: в выходном векторе все значения элементов, кроме одного, принимаются равными –1.
- Единственное положительное значение элемента выходного вектора располагается в позиции, соответствующей нужному классу.

ДАП (Коско) Пример

- Входной вектор содержит 16 элементов, выходной – 9 элементов. Соответственно, структура нейронной сети будет включать 9 нейронов в первом слое и 16 – во втором.
- Значение элемента входного вектора, равное +1, соответствует закрашенному фрагменту изображения входного образа, –1 – незакрашенному.

ДАП (Коско) Пример

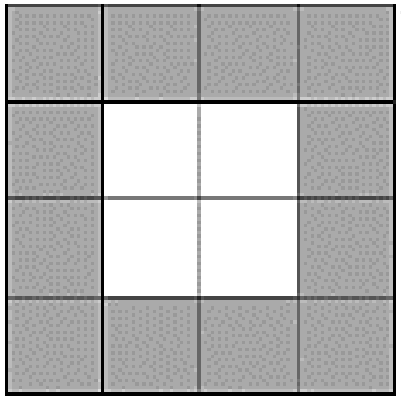
Образ 1



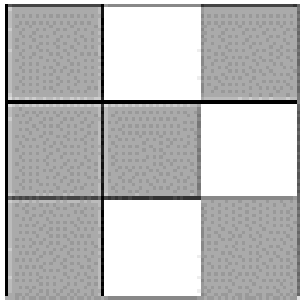
входной

выходной

Образ 2



входной



выходной

ДАП (Коско) Пример

- составляется матрица входных эталонных образов

№ образа	№ входной переменной							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	1	–1	–1	1	–1	1	–1	–1
2	1	1	1	1	1	–1	–1	1

№ образа	№ входной переменной							
	9	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	–1	–1	1	–1	1	–1	–1	1
2	1	–1	–1	1	1	1	1	1

ДАП (Коско) Пример

- составляется матрица выходных эталонных образов

№ образа	№ выходной переменной								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	1	1	1	1	–1	1	1	–1	1
2	1	–1	1	1	1	–1	1	–1	1

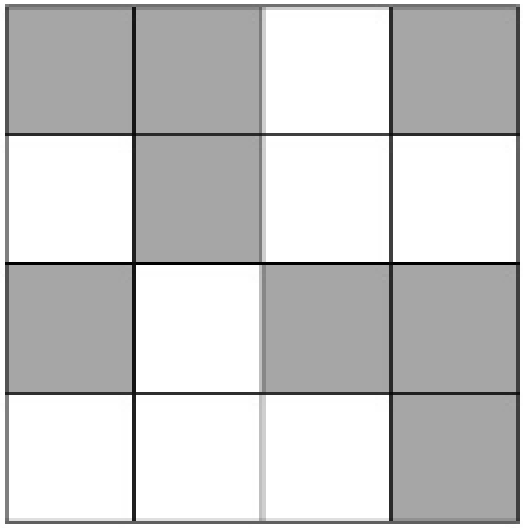
ДАП (Коско) Пример

- определяются весовые коэффициенты сети Коско

№ входного нейрона	№ выходного нейрона								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	2	0	2	2	0	0	2	–2	2
2	0	–2	0	0	2	–2	0	0	0
3	0	–2	0	0	2	–2	0	0	0
4	2	0	2	2	0	0	2	–2	2
5	0	–2	0	0	2	–2	0	0	0
6	0	2	0	0	–2	2	0	0	0
7	–2	0	–2	–2	0	0	–2	2	–2
8	0	–2	0	0	2	–2	0	0	0
9	0	–2	0	0	2	–2	0	0	0
10	–2	0	–2	–2	0	0	–2	2	–2
11	0	2	0	0	–2	2	0	0	0
12	0	–2	0	0	2	–2	0	0	0
13	2	0	2	2	0	0	2	–2	2
14	0	–2	0	0	2	–2	0	0	0
15	0	–2	0	0	2	–2	0	0	0
16	2	0	2	2	0	0	2	–2	2

ДАП (Коско) Пример

- Для тестирования сети Коско



№ входа или выхода	$x_i^{(0)}$	Итерация 1				Итерация 2			
		расчет выходов		расчет входов		расчет выходов		расчет входов	
		$s_{1j}^{(1)}$	$y_j^{(1)}$	$s_{2i}^{(1)}$	$x_i^{(1)}$	$s_{1j}^{(2)}$	$y_j^{(2)}$	$s_{2i}^{(2)}$	$x_i^{(2)}$
1	1	8	1	12	1	12	1	12	1
2	1	8	1	–6	–1	20	1	–6	–1
3	–1	8	1	–6	–1	12	1	–6	–1
4	1	8	1	12	1	12	1	12	1
5	–1	–8	–1	–6	–1	–20	–1	–6	–1
6	1	8	1	6	1	20	1	6	1
7	–1	8	1	–12	–1	12	1	–12	–1
8	–1	–8	–1	–6	–1	–12	–1	–6	–1
9	1	8	1	–6	–1	12	1	–6	–1
10	–1	–	–	–12	–1	–	–	–12	–1
11	1	–	–	6	1	–	–	6	1
12	1	–	–	–6	–1	–	–	–6	–1
13	–1	–	–	12	1	–	–	12	1
14	–1	–	–	–6	–1	–	–	–6	–1
15	–1	–	–	–6	–1	–	–	–6	–1
16	1	–	–	12	1	–	–	12	1
$\|\vec{y}^{(q)} - \vec{y}^{(q-1)}\|$	–	–	–	–	–	–	0	–	–

Как видно из таблицы, стабилизация выходного вектора наступила уже на второй итерации, и она оказалась идентичной первому выходному эталонному образу.
Следовательно, нейронная сеть, моделирующая работу гетероассоциативной памяти, успешно решила задачу для зашумленного набора входных значений.

№ образа	№ выходной переменной								
	1	2	3	4	5	6	7	8	9
1	1	1	1	1	–1	1	1	–1	1
2	1	–1	1	1	1	–1	1	–1	1

Проблемы ДАП (Коско)

Неверные результаты:

- лок. минимум для E,
- Разные вектора x и y

Модификации ДАП (Коско)

W= $\sum_{i=1}^{m} x_i^T y_i + (r-1) x_j^T y_j$ -

увеличиваем количество участия пары с j для каждой пары, тогда

$$W^q = W^{q-1} + (r-1) x_j^T y_j$$

ДАП (Коско)

Разработано много разновидностей двунаправленной ассоциативной памяти, основными из которых являются:

- непрерывная ДАП (с сигмоидами в качестве функций активации нейронов),
- адаптивная ДАП (с изменением весов в процессе функционирования сети),
- конкурирующая ДАП (с конкуренцией нейронов внутри каждого слоя) и др.

Основы нейронных сетей

Лекция 12

Рекуррентные сети

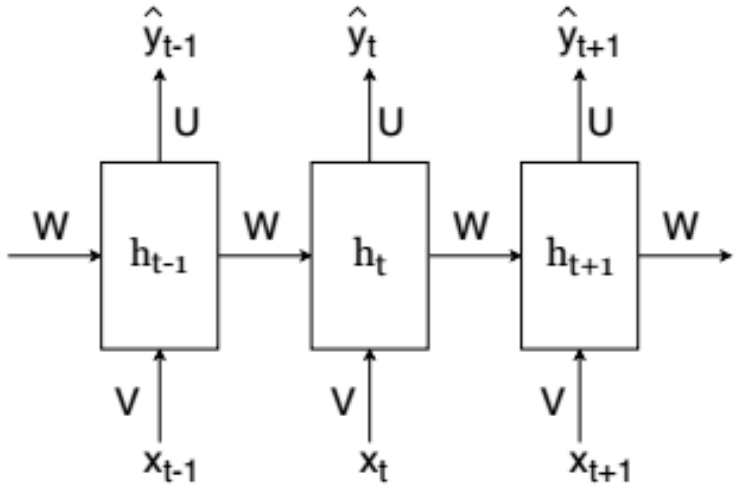
RNN

- Рекуррентные нейронные сети (РНС, англ. Recurrent neural network; RNN) — вид нейронных сетей, где связи между элементами образуют направленную последовательность. Благодаря этому появляется возможность обрабатывать серии событий во времени или последовательные пространственные цепочки.
- В отличие от многослойных перцептронов, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины. Поэтому сети RNN применимы в таких задачах, где нечто целостное разбито на части, например: распознавание рукописного текста или распознавание речи.
- Было предложено много различных архитектурных решений для рекуррентных сетей от простых до сложных. В последнее время наибольшее распространение получили сеть с долговременной и кратковременной памятью (LSTM) и управляемый рекуррентный блок (GRU).

RNN

- Существует много разновидностей, решений и конструктивных элементов рекуррентных нейронных сетей.
- Трудность рекуррентной сети заключается в том, что если учитывать каждый шаг времени, то становится необходимым для каждого шага времени создавать свой слой нейронов, что вызывает серьёзные вычислительные сложности. Кроме того, многослойные реализации оказываются вычислительно неустойчивыми, так как в них как правило исчезают или зашкаливают веса. Если ограничить расчёт фиксированным временным окном, то полученные модели не будут отражать долгосрочных трендов. Различные подходы пытаются усовершенствовать модель исторической памяти и механизм запоминания и забывания.

Модель рекуррентной нейронной сети (RNN)



h_t — скрытое состояние в момент t

$$h_t = f(Vx_t + Wh_{t-1} + b)$$

$$\hat{y}_t = g(Uh_t + \hat{b})$$

Обучение сети — минимизация суммарных потерь:

$$\sum_{t=1}^{n} \mathcal{L}_t(y_t, \hat{y}_t) \to \min_{V,U,W,b,\hat{b}}$$

Сеть обучается с помощью алгоритма Bakpropagation[1]

Детали обучения RNN: производные по U и W

Градиент по U зависит только от величин в момент t:

$$\frac{d\mathcal{L}_t}{dU} = \frac{\partial \mathcal{L}_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial U}$$

Градиент по W зависит от всех предыдущих величин:

$$\frac{d\mathcal{L}_t}{dW} = \frac{\partial \mathcal{L}_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{dh_t}{dW}$$

$$\frac{dh_t}{dW} = \frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}}\frac{dh_{t-1}}{dW} =$$

$$= \frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\frac{dh_{t-2}}{dW} =$$

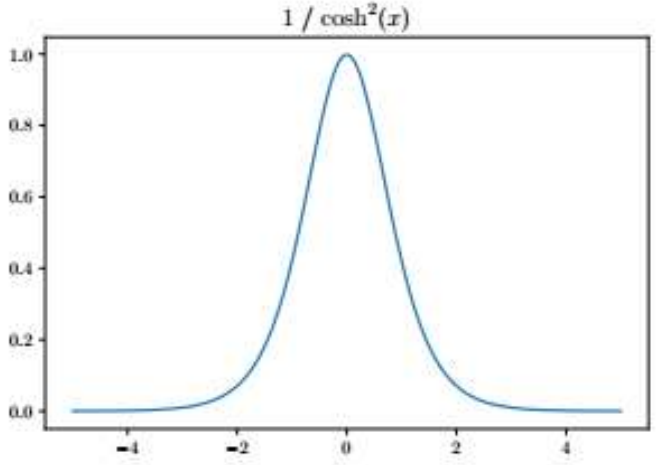
$$= \ \ldots \ = \sum_{k=1}^{t}\left(\prod_{i=k+1}^{t}\frac{\partial h_i}{\partial h_{i-1}}\right)\frac{\partial h_k}{\partial W}$$

Градиент по V считается аналогично градиенту по W

Детали обучения RNN: взрыв и затухание градиентов

Взрыв градиента:

$$\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \to \infty$$



Затухание градиента:

$$\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \to 0$$

$$\frac{\partial h_i}{\partial h_{i-1}} = diag\left(\frac{1}{\mathrm{ch}^2(z_i)}\right) W$$

$$z_i = Vx_i + Wh_{i-1} + b$$

если $f = \tanh$

Популярные способы борьбы с взрывом/затуханием:

- Gradient clipping (против взрыва)
- Модели LSTM и GRU (против затухания)

Надо каждый раз умножать на одну и ту же W, и норма градиента может расти или убывать экспоненциально.

• Взрывающиеся градиенты: надо каждый раз умножать на W, и норма градиента может расти экспоненциально.

Gradient clipping

Ограничение нормы градиентов:

Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

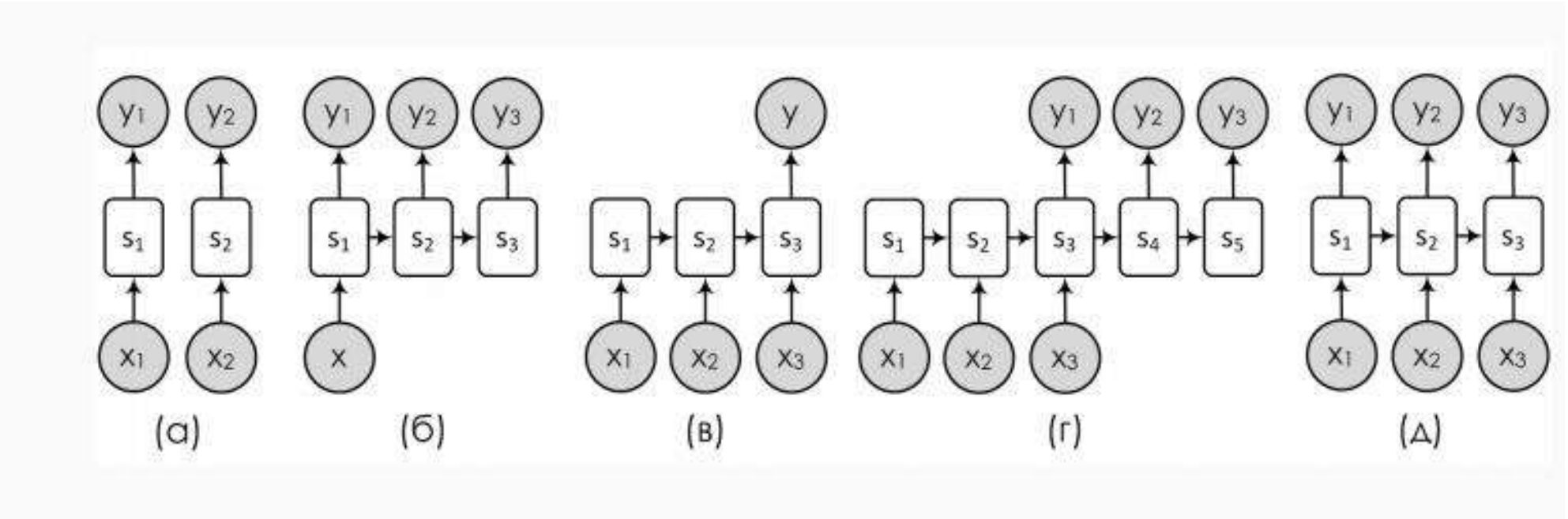
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

Как выбрать порог?
Например, брать среднюю норму градиента для весов по запускам без gradient clipping

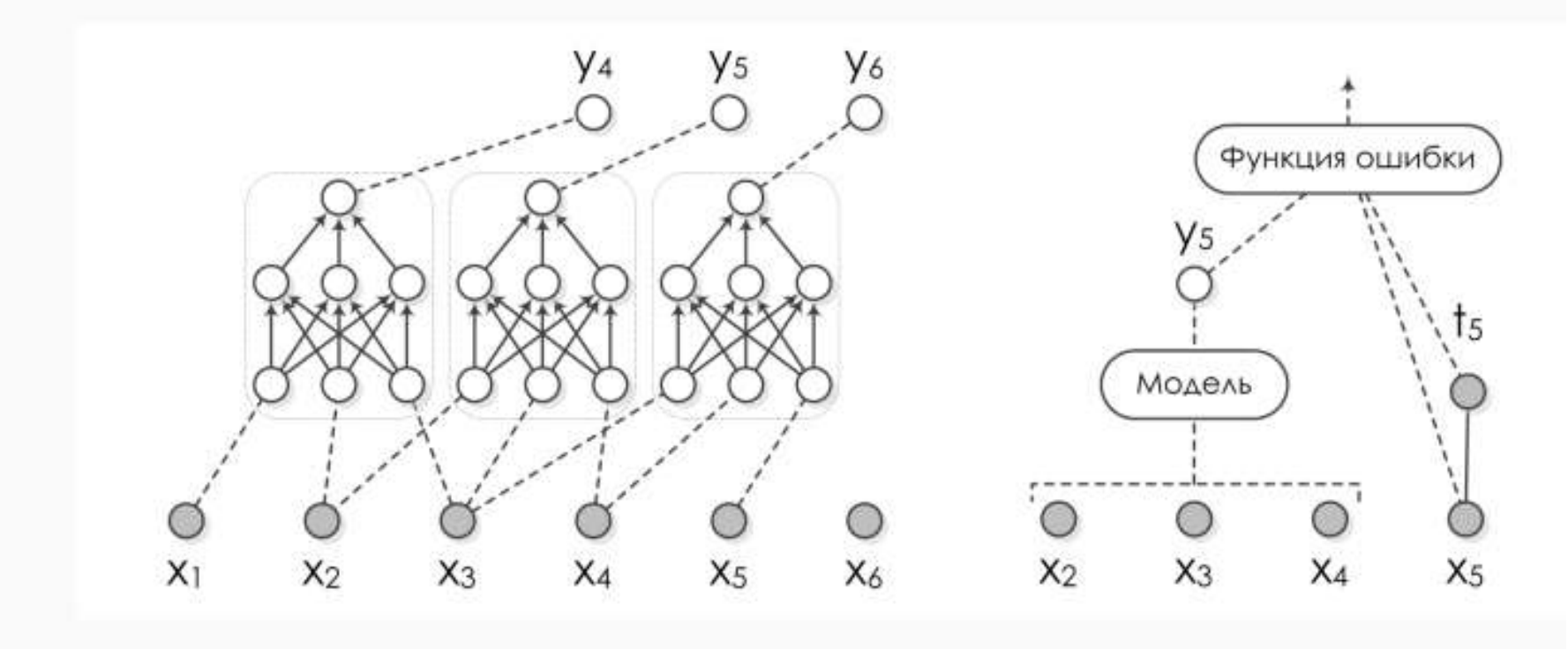
- Два варианта – ограничить общую норму или каждое значение:
  - `sgd = optimizers.SGD(lr=0.01, clipnorm=1.)`
  - `sgd = optimizers.SGD(lr=0.01, clipvalue=.05)`

Задачи применения

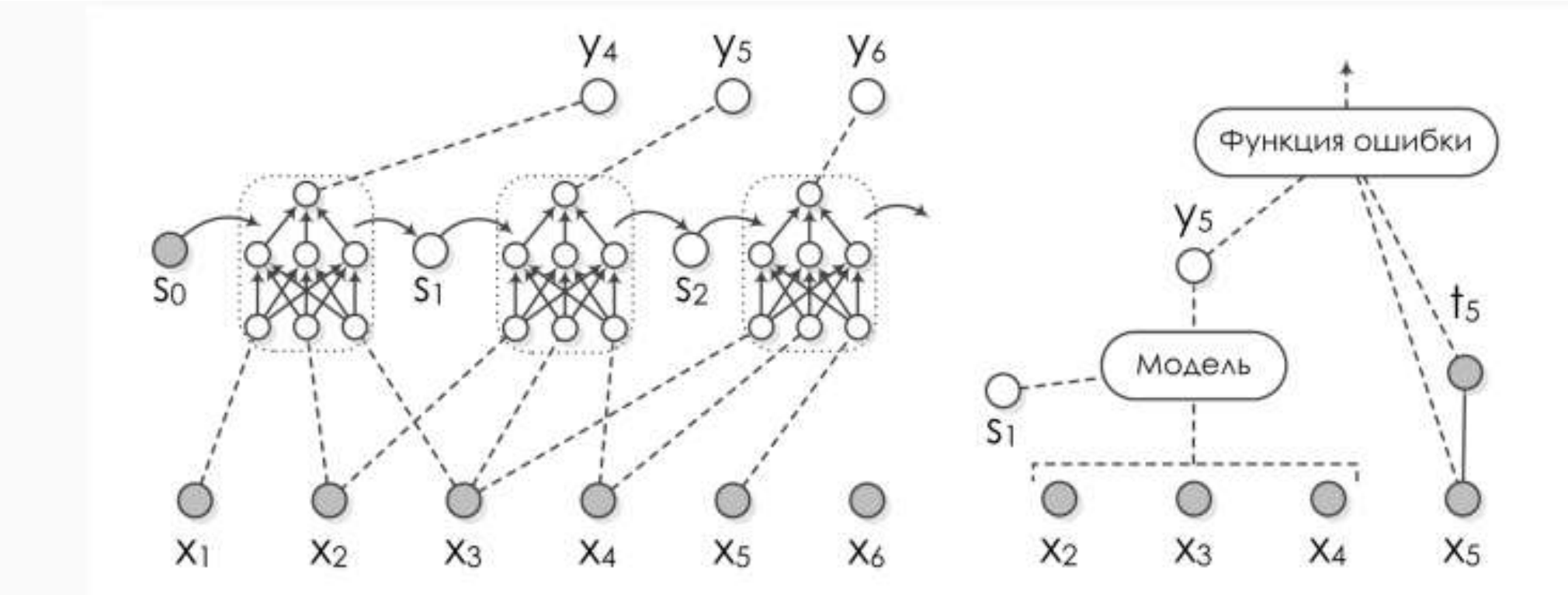
- Последовательности: текст, временные ряды, речь, музыка,…



Метод скользящего окна для применения к последовательности

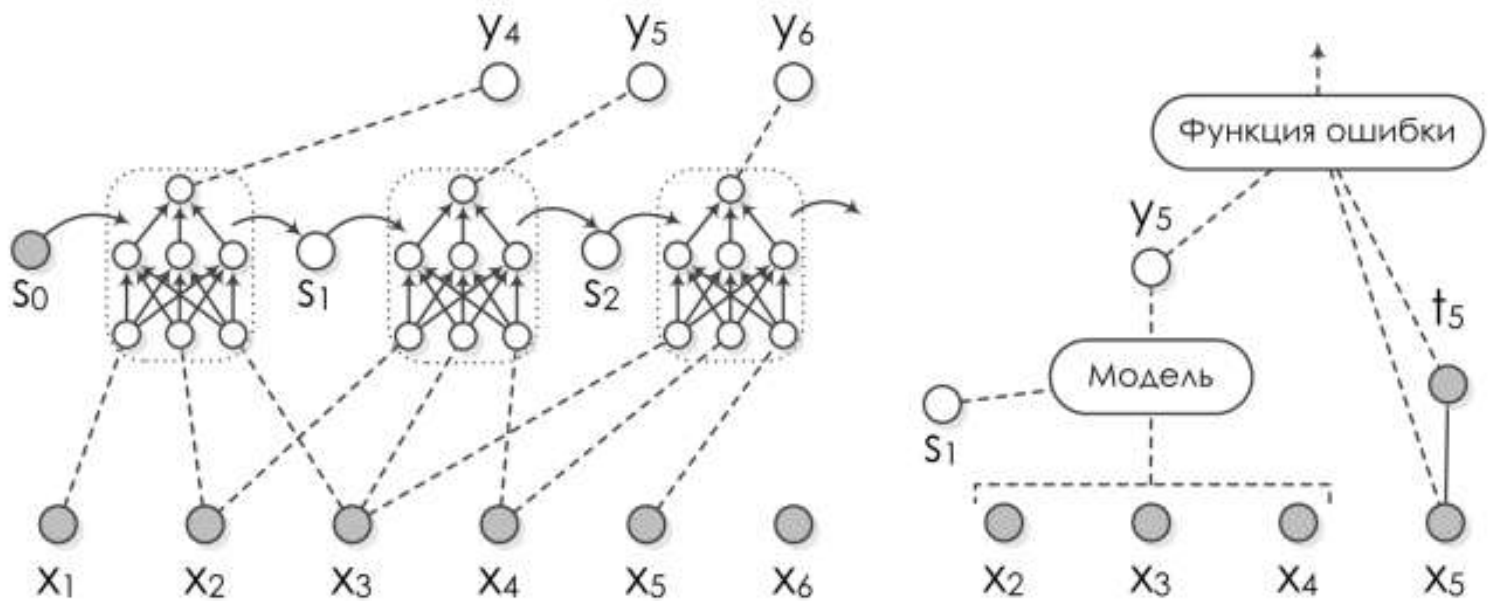


RNN : сохранение скрытого состояния и обновление



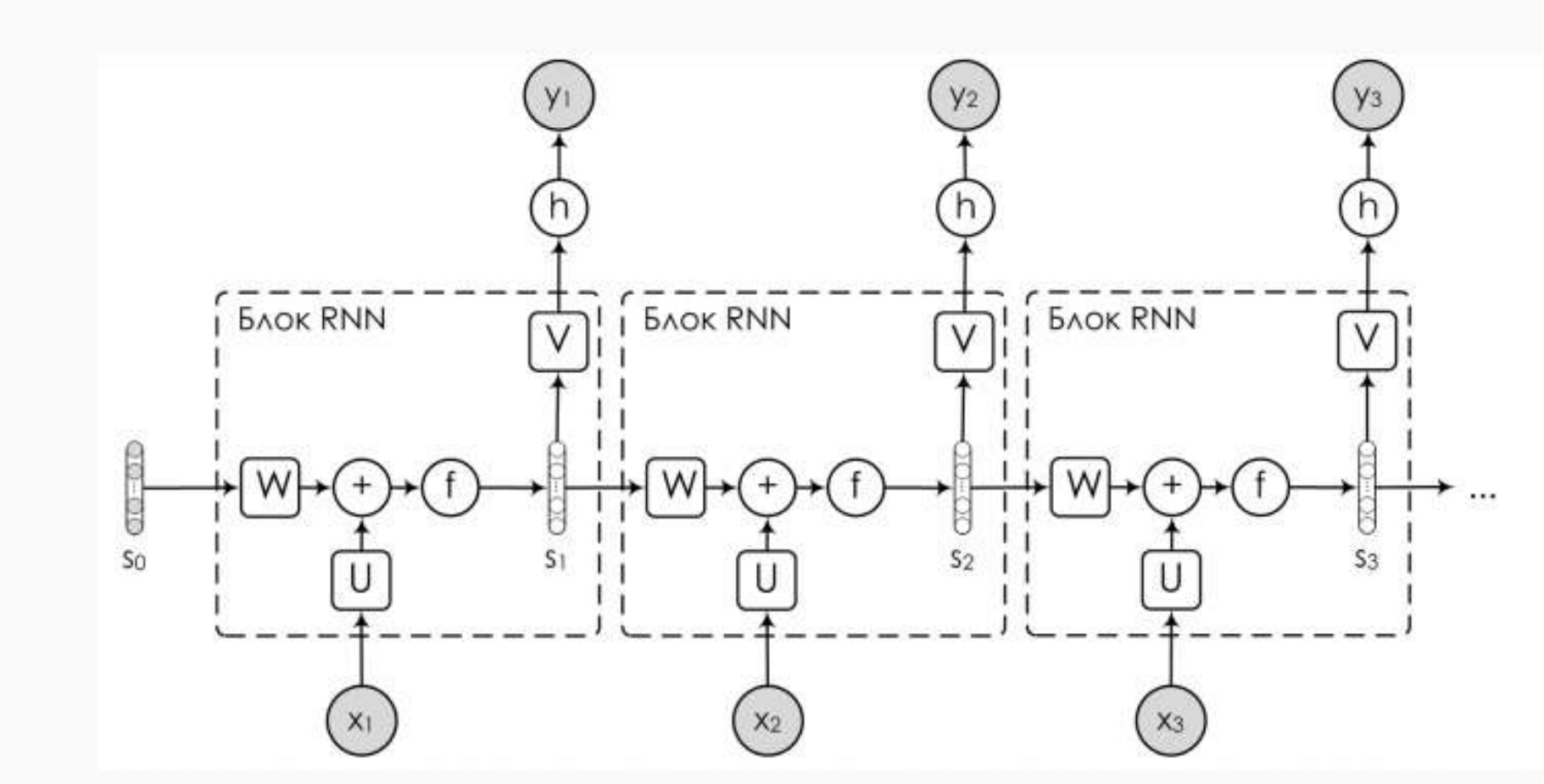
- Но как теперь делать backpropagation? Получается, что в графе вычислений теперь циклы:

$$s_i = h(x_i, x_{i+1}, x_{i+2}, s_{i-1}).$$



$$y_6 = f(x_3, x_4, x_5, s_2) = f(x_3, x_4, x_5, h(x_2, x_3, x_4, s_1)) =$$
$$= f(x_3, x_4, x_5, h(x_2, x_3, x_4, h(x_1, x_2, x_3, s_0))).$$

RNN


y1
y2
y3
h
h
h
Блок RNN
Блок RNN
Блок RNN
V
V
V
W
+
f
W
+
f
W
+
f
U
U
U
s0
s1
s2
s3
...
x1
x2
x3

RNN

$$\mathbf{a}_t = \mathbf{b} + W\mathbf{s}_{t-1} + U\mathbf{x}_t,$$
$$\mathbf{s}_t = f(\mathbf{a}_t),$$
$$\mathbf{o}_t = \mathbf{c} + V\mathbf{s}_t,$$
$$\mathbf{y}_t = h(\mathbf{o}_t),$$

где f – рекуррентная нелинейность, h – функция выхода.

Некоторые RNN

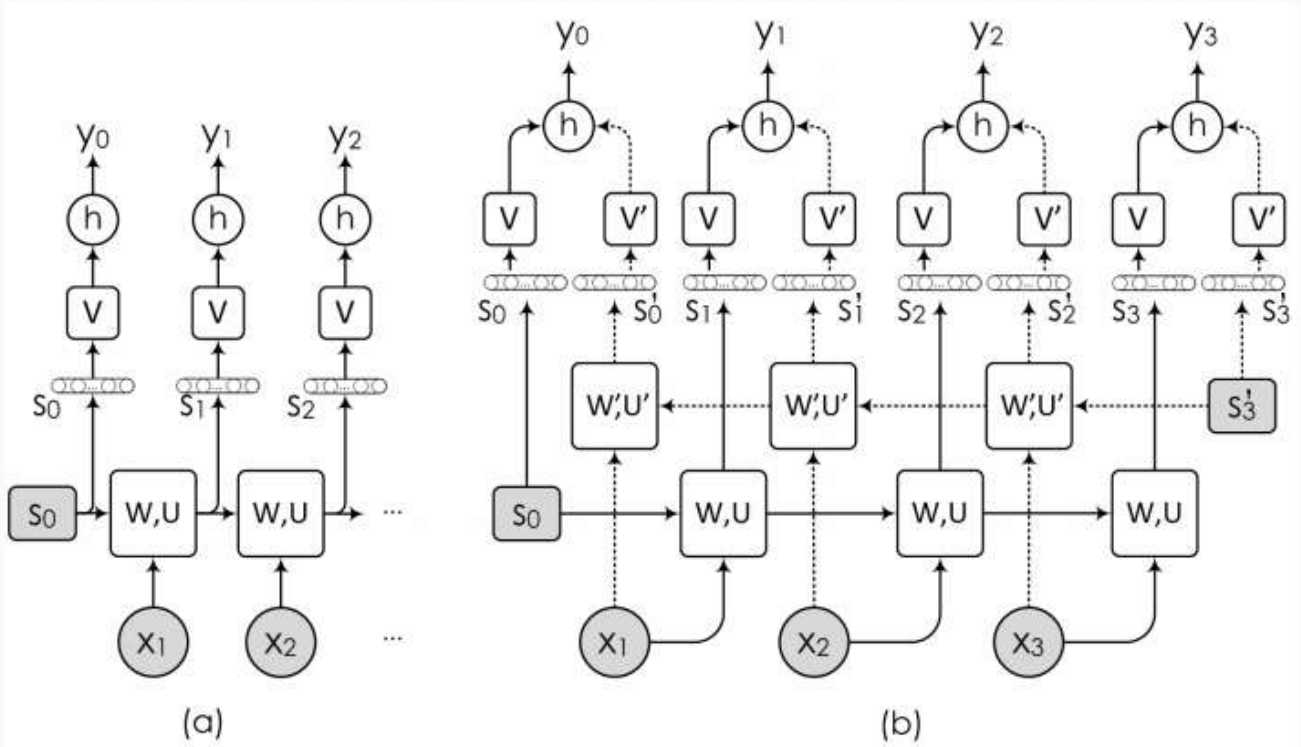
- Сеть Хопфилда — это такой тип рекуррентной сети, когда все соединения симметричны. Изобретена Джоном Хопфилдом в 1982 году и гарантируется, что динамика такой сети сходится к одному из положений равновесия. Если при создании соединений используют обучение Хебба, то сеть Хопфилда может работать как надежная ассоциативная память, устойчивая к изменению подключений.
- Сеть Хемминга – 2 слоя

Некоторые RNN

- Двунаправленная ассоциативная память (BAM)- Нейронная сеть Коско
- Вариацией сети Хопфилда является двунаправленная ассоциативная память (BAM).
- BAM имеет два слоя, каждый из которых может выступать в качестве входного, находить (вспоминать) ассоциацию и генерировать результат для другого слоя

Двунаправленная RNN

- Иногда нужен контекст с обеих сторон:



Двунаправленная RNN

- Формально:

$$\begin{aligned}
\mathbf{s}_t &= \sigma\left(\mathbf{b} + W\mathbf{s}_{t-1} + U\mathbf{x}_t\right), \\
\mathbf{s}'_t &= \sigma\left(\mathbf{b}' + W'\mathbf{s}'_{t+1} + U'\mathbf{x}_t\right), \\
\mathbf{o}_t &= \mathbf{c} + V\mathbf{s}_t + V'\mathbf{s}'_t, \\
\mathbf{y}_t &= h\left(\mathbf{o}_t\right).
\end{aligned}$$

- И это, конечно, обобщается на любой другой тип конструкций.

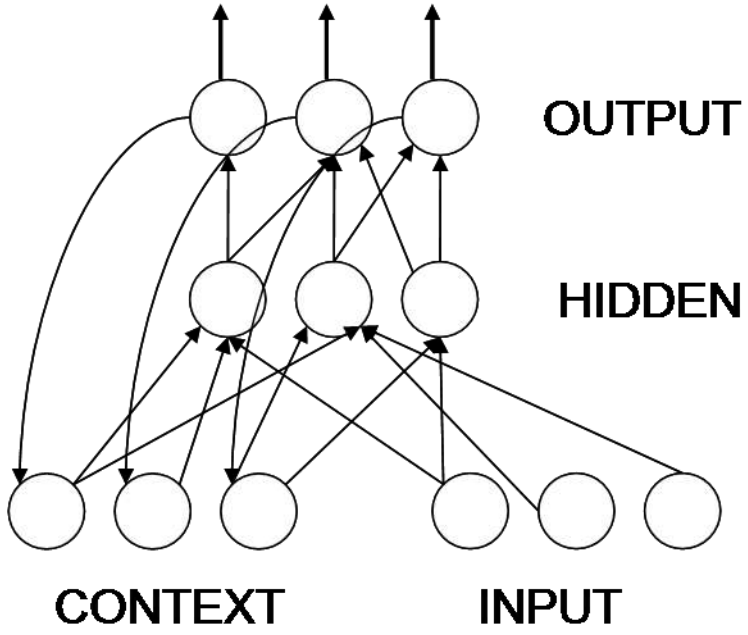
Некоторые RNN

- Сеть Джордана — вид нейронных сетей, который получается из многослойного перцептрона, если на его вход подать, помимо входного вектора, выходной с задержкой на один или несколько тактов.

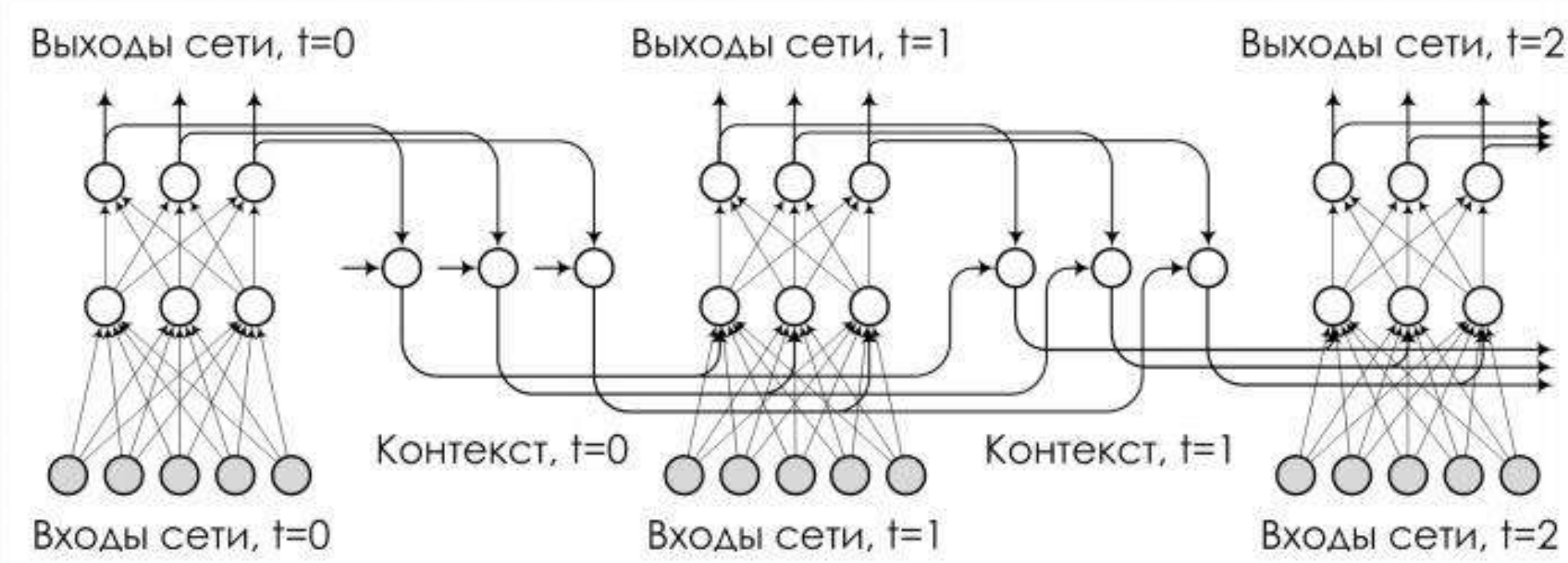
- В первых рекуррентных сетях главной идеей было дать сети видеть свой выходной образ на предыдущем шаге. У такой сети только часть рецепторов принимает сигналы из окружающего мира, на другие рецепторы приходит выходной образ из предыдущего момента времени. Рассмотрим прохождение последовательности сигналов через сеть. Сигнал поступает на группу рецепторов соединенных с внешним миром (INPUT) и проходит в скрытый слой (HIDDEN). Преобразованный скрытым слоем сигнал пойдет на выходной слой (OUTPUT) и выйдет из сети, а его копия попадет на задержку. Далее в сеть, на рецепторы, воспринимающие внешние сигналы, поступает второй образ, а на контекстную группу рецепторов (CONTEXT) — выходной образ с предыдущего шага из задержки. Далее со всех рецепторов сигнал пойдет в скрытый слой, затем на выходной.

Сеть Джордана

- Сеть Джордана — вид нейронных сетей, который получается из многослойного перцептрона, если на его вход подать, помимо входного вектора, выходной с задержкой на один или несколько тактов.
- В первых рекуррентных сетях главной идеей было дать сети видеть свой выходной образ на предыдущем шаге. У такой сети только часть рецепторов принимает сигналы из окружающего мира, на другие рецепторы приходит выходной образ из предыдущего момента времени. Рассмотрим прохождение последовательности сигналов через сеть. Сигнал поступает на группу рецепторов соединенных с внешним миром (INPUT) и проходит в скрытый слой (HIDDEN). Преобразованный скрытым слоем сигнал пойдет на выходной слой (OUTPUT) и выйдет из сети, а его копия попадет на задержку. Далее в сеть, на рецепторы, воспринимающие внешние сигналы, поступает второй образ, а на контекстную группу рецепторов (CONTEXT) — выходной образ с предыдущего шага из задержки. Далее со всех рецепторов сигнал пойдет в скрытый слой, затем на выходной.



Сеть Джордана



сеть Элмана

- Нейронная сеть Элмана представляет из себя трёхслойную нейронную сеть. Дополнительно к сети добавлен набор «контекстных блоков» (u на иллюстрации). Средний (скрытый) слой соединён с контекстными блоками с фиксированным весом, равным единице. С каждым шагом времени на вход поступает информация, которая проходит прямой ход к выходному слою в соответствии с правилами обучения. Фиксированные обратные связи сохраняют предыдущие значения скрытого слоя в контекстных блоках (до того как скрытый слой поменяет значение в процессе обучения). Таким способом сеть сохраняет своё состояние, что может использоваться в предсказании последовательностей, выходя за пределы мощности многослойного перцептрона.

Сеть Джордана

Сети Элмана и Джордана называют также «простыми рекуррентными сетями» (SRN).

Сеть Элмана[24]

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

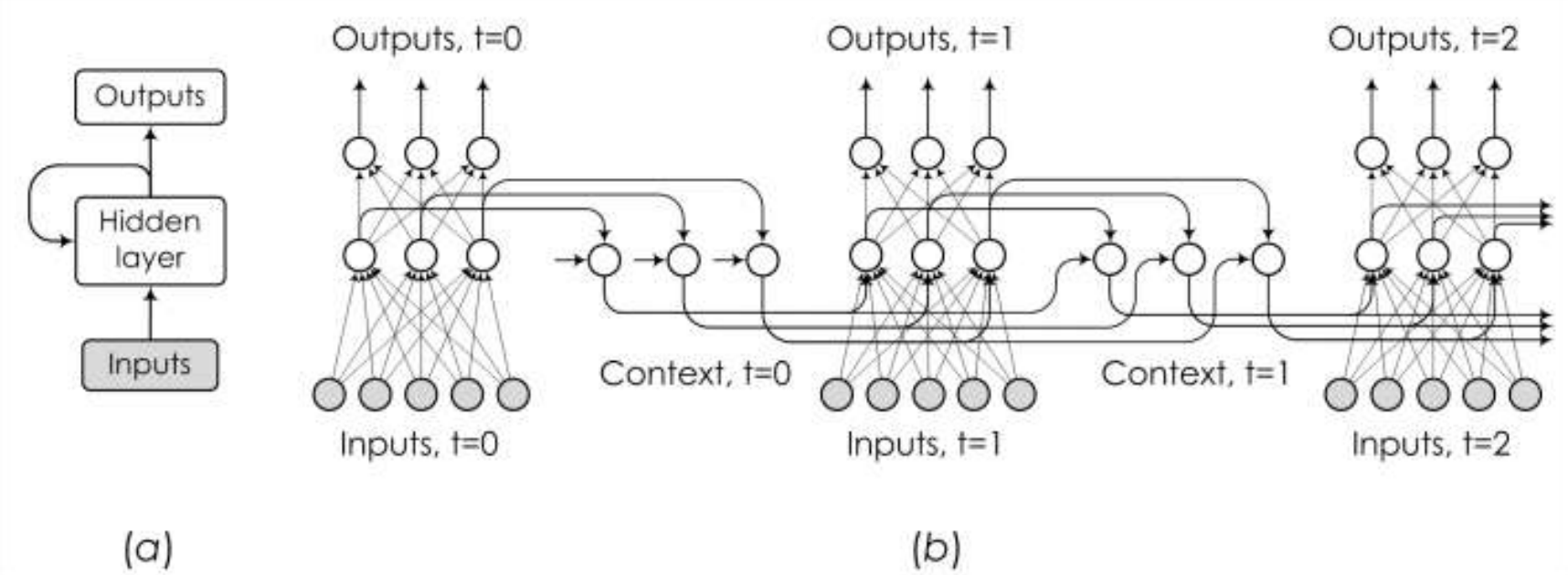
Сеть Джордана[25]

$$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

Обозначения переменных и функций:

- x_t: вектор входного слоя
- h_t: вектор скрытого слоя
- y_t: вектор выходного слоя
- W, U и b: Матрица и вектор параметров
- σ_h и σ_y: Функция активации

- Сеть Элмана (Elman; конец 1980-х):



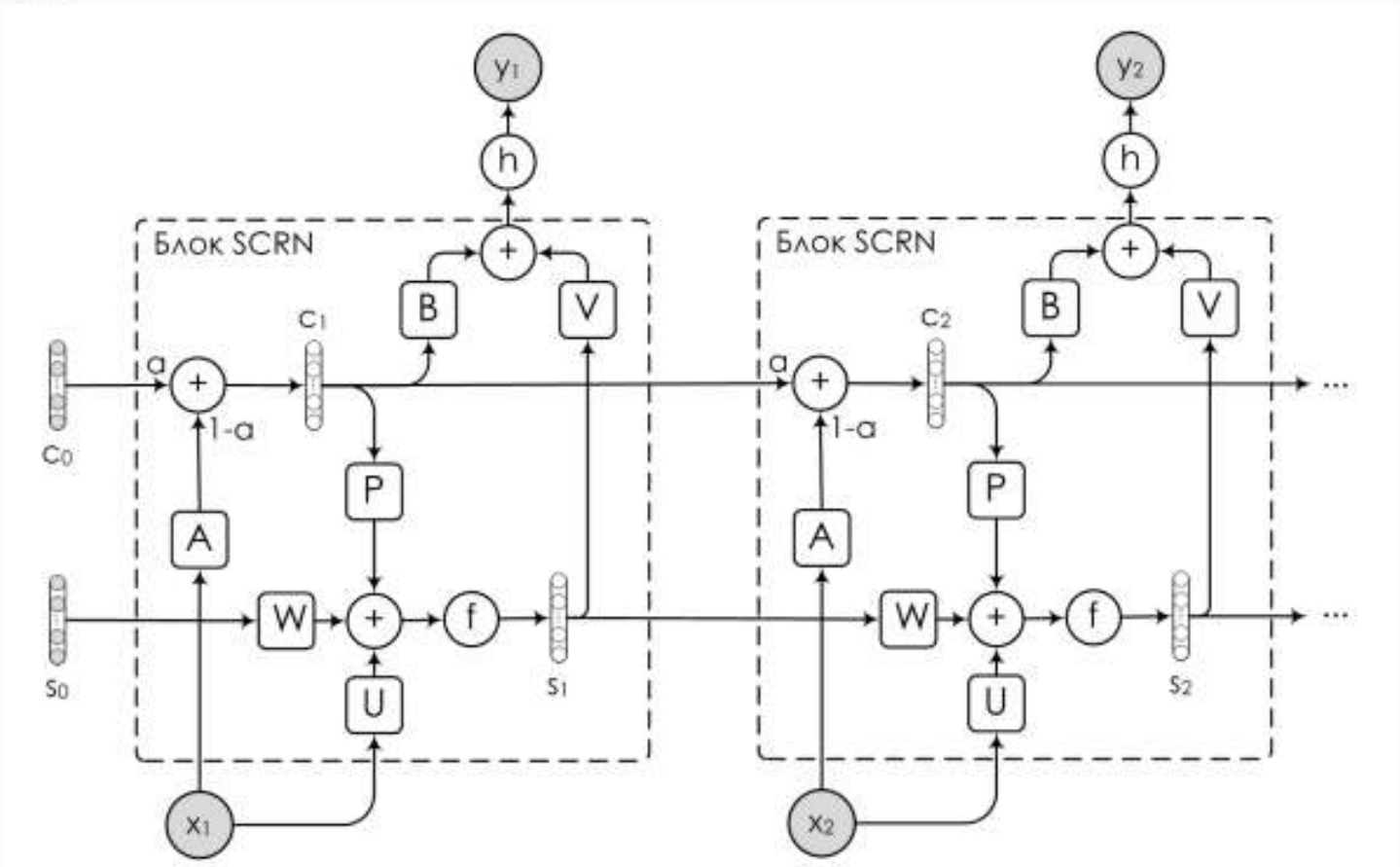
- Разница в том, что нейроны контекста c_t получают входы со скрытого уровня, а не выходов.
- И нет никаких весов от предыдущих c_{t-1}! То есть веса фиксированы и равны 1.

- Это приводит к хорошим долгосрочным эффектам, потому что нет нелинейности между последовательными шагами, и карусель константной ошибки получается по определению:

$$c_t = c_{t-1} + U x_t.$$

- Идея: можно зафиксировать градиенты, использовав единичную матрицу весов вместо обучаемой W.
- Долгосрочная память тут есть... но обучать очень трудно, потому что градиенты надо возвращать к началу последовательности.

- (Mikolov et al., 2014): Structurally Constrained Recurrent Network (SCRN).
- Сочетание двух идей – $\mathbf{s}_t$ с W и $\boldsymbol{c}_t$ с диагональной матрицей рекуррентных весов.



- Формально:

$$\begin{aligned}
\mathbf{c}_t &= (1-\alpha)A\mathbf{x}_t + \alpha\mathbf{c}_{t-1}, \\
\mathbf{s}_t &= f(P\mathbf{c}_t + U\mathbf{x}_t + W\mathbf{s}_{t-1}), \\
\mathbf{y}_t &= h(V\mathbf{s}_t + B\mathbf{s}_t).
\end{aligned}$$

- SCRN – это просто обычный RNN, где $\mathbf{s}_t$ и $\mathbf{c}_t$ в одном векторе, и матрица рекуррентных весов имеет вид

$$W = \begin{pmatrix} R & P \\ \mathbf{0} & \alpha\mathbf{I} \end{pmatrix},$$

  - (Le et al., 2015): как правильно инициализировать рекуррентные веса
  - IRNN – составим рекуррентные веса с ReLU-активациями и инициализируем единичной матрицей; похоже на SCRN, но ещё проще

LSTM (long short-term memory) - история

Начиная с 2007 года LSTM приобрела популярность и смогла вывести на новый уровень распознавание речи, показав существенное улучшение по сравнению с традиционными моделями. В 2009 году появился подход классификации по рейтингу (Connectionist Temporal Classification, (CTC)). Этот метод позволил рекуррентным сетям подключить анализ контекста при распознавании рукописного текста. В 2014 году китайская энциклопедия и поисковая система Baidu, используя рекуррентные сети с обучением по CTC смогли поднять на новый уровень показатели Switchboard Hub5’00, опередив традиционные методы.

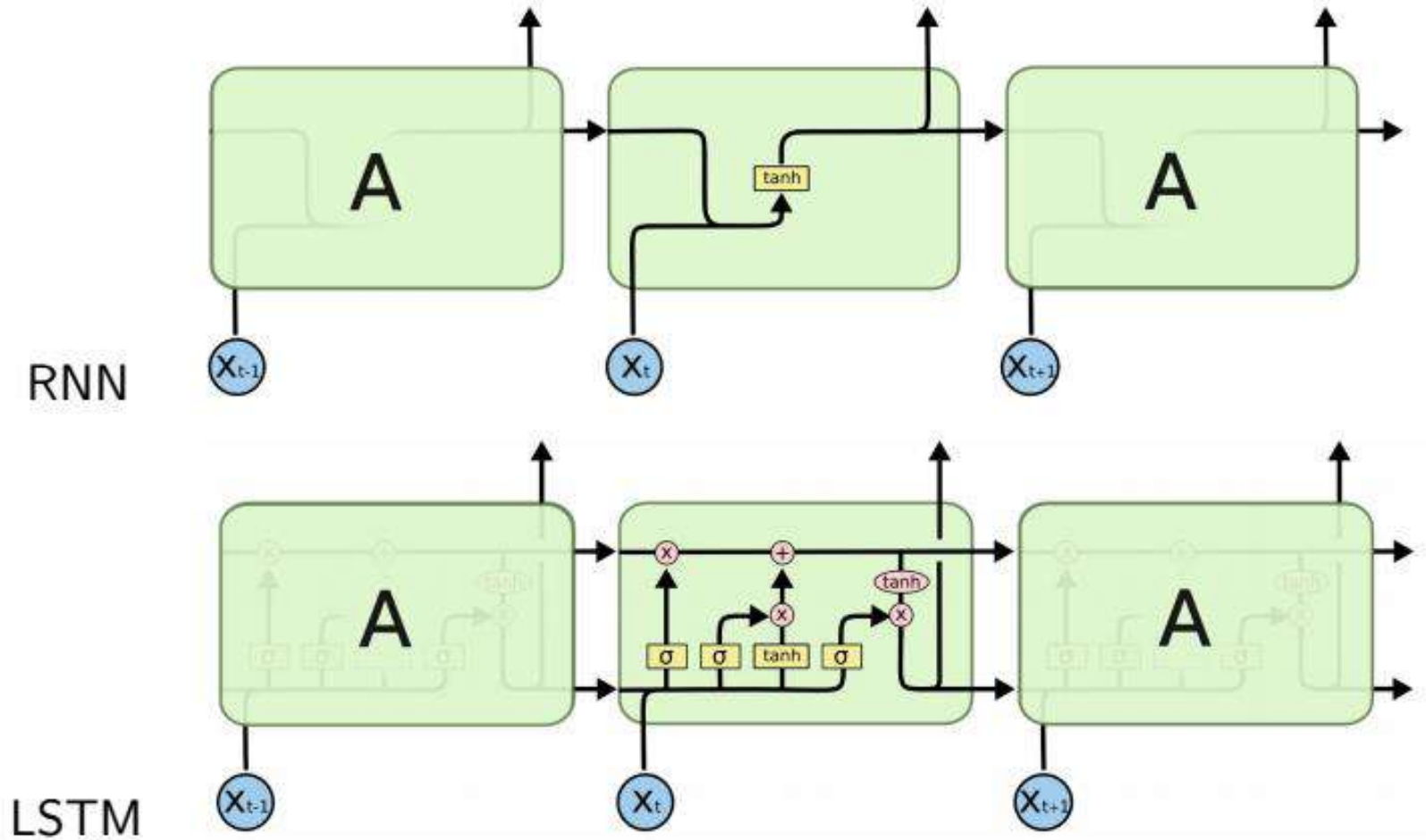
LSTM - история

LSTM привела также к улучшению распознавания речи с большими словарями и улучшения синтеза речи по тексту, и нашла также применение в операционной системе Google Android. В 2015 году распознавание речи у Google значительно повысило показатели вплоть до 49 %, причиной того стало использование специальной системы обучения LSTM на базе CTC в системе Google voice search.

LSTM вывело на новый уровень качество машинного перевода, построения языковых моделей и обработки многоязычного текста. Сочетание LSTM со свёрточными нейронными сетями (CNN) позволило усовершенствовать автоматическое описание изображений.

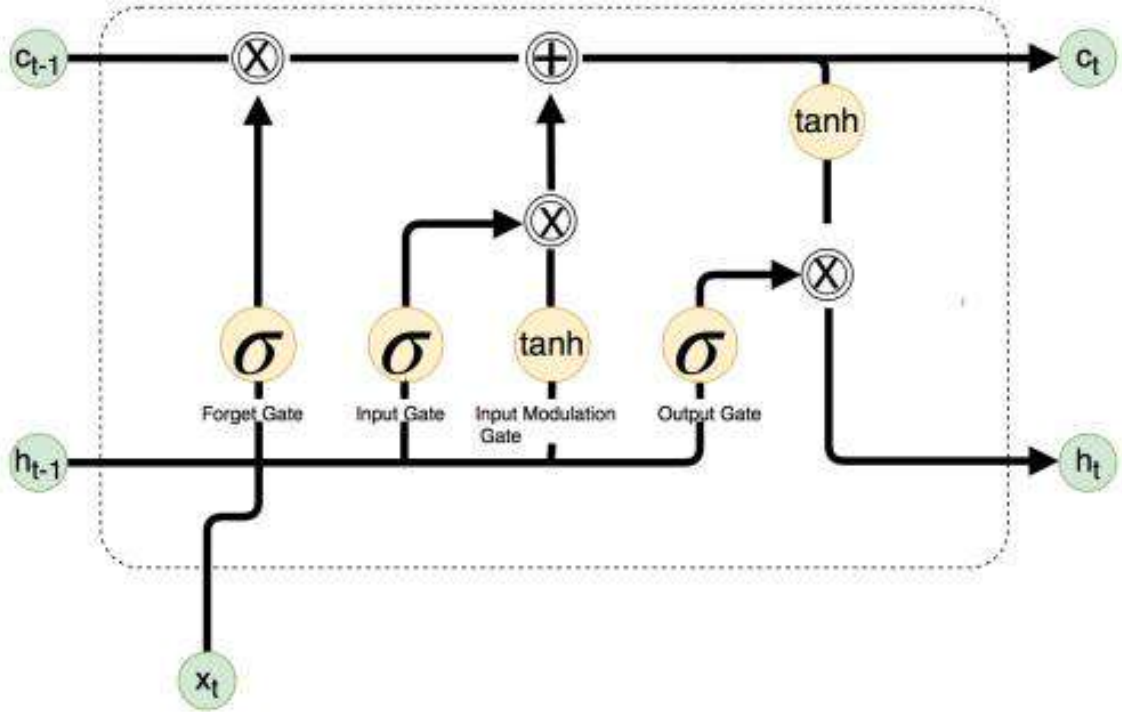
LSTM сеть

Используем более сложную структуру ячейки:



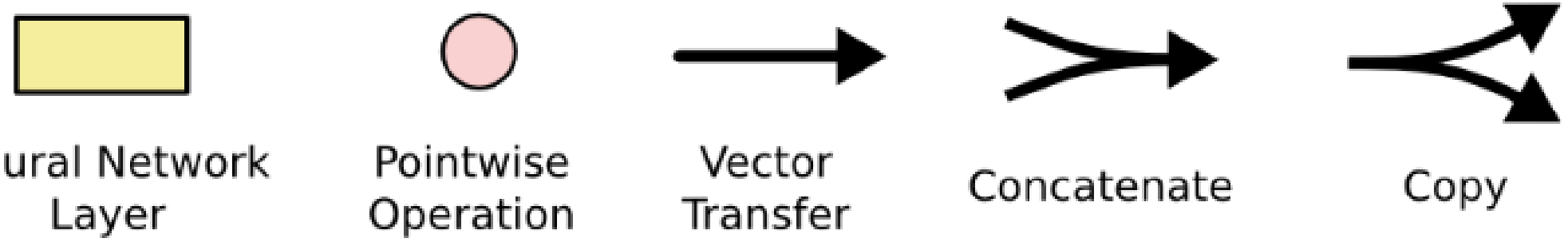
LSTM ячейка

$$z_t = [h_{t-1}, x_t]$$
$$f_t = \sigma(W_f \cdot z_t + b_f)$$
$$i_t = \sigma(W_i \cdot z_t + b_i)$$
$$\hat{C}_t = \text{th}(W_c \cdot z_t + b_c)$$
$$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{C}_t$$
$$o_t = \sigma(W_o \cdot z_t + b_o)$$
$$h_t = o_t \cdot \tanh(C_t)$$



Обучается с помощью алгоритма Backpropagation

Почему решает проблему затухающих градиентов? Частично потому, что C_t зависит от C_{t-1} линейно, т.е $\frac{\partial C_t}{\partial C_{t-1}} = f_t$



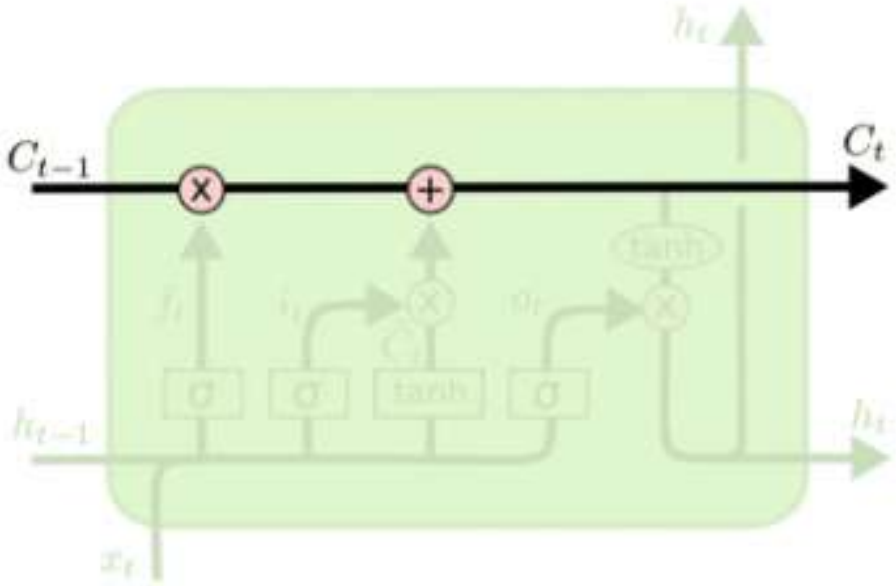
Слой нейронной сети; поточечная операция; векторный перенос; объединение; копирование.

На схеме выше каждая линия переносит целый вектор от выхода одного узла ко входу другого. Розовыми кружочками обозначены поточечные операции, такие, как сложение векторов, а желтые прямоугольники – это обученные слои нейронной сети. Сливающиеся линии означают объединение, а разветвляющиеся стрелки говорят о том, что данные копируются и копии уходят в разные компоненты сети.

Основная идея LSTM

Ключевой компонент LSTM – это состояние ячейки (cell state) – горизонтальная линия, проходящая по верхней части схемы.

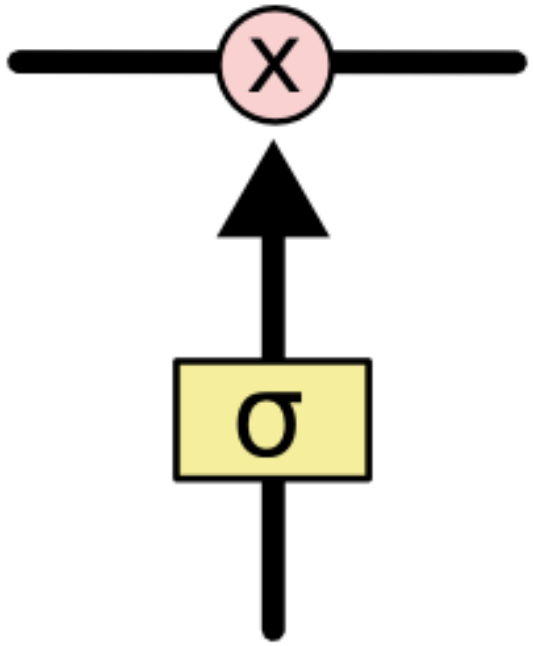
Состояние ячейки напоминает конвейерную ленту. Она проходит напрямую через всю цепочку, участвуя лишь в нескольких линейных преобразованиях. Информация может легко течь по ней, не подвергаясь изменениям.



Основная идея LSTM

Тем не менее, LSTM может удалять информацию из состояния ячейки; этот процесс регулируется структурами, называемыми фильтрами (gates).

Фильтры позволяют пропускать информацию на основании некоторых условий. Они состоят из слоя сигмоидальной нейронной сети и операции поточечного умножения.

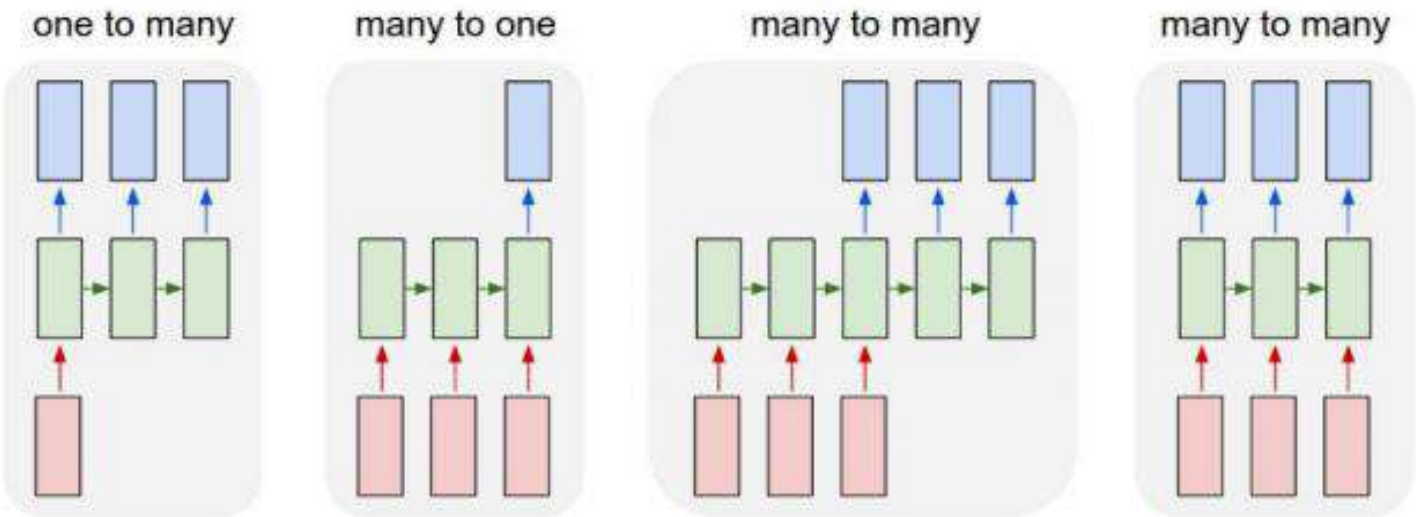


Основная идея LSTM

Сигмоидальный слой возвращает числа от нуля до единицы, которые обозначают, какую долю каждого блока информации следует пропустить дальше по сети. Ноль в данном случае означает “не пропускать ничего”, единица – “пропустить все”.

В LSTM три таких фильтра, позволяющих защищать и контролировать состояние ячейки.

Разные архитектуры рекуррентных сетей



Примеры задач:

one to many Генерация описания изображения

many to one Классификация предложений

many to many(1) Перевод с одного языка на другой

many to many(2) Определение частей речи

Пример с LSTM – Определить рейтинг фильма

- Loads the IMDB dataset.
- Это набор данных из 25 000 обзоров фильмов из IMDB, помеченных настроениями (положительные/отрицательные). Обзоры были предварительно обработаны, и каждый обзор закодирован в виде списка индексов слов (целых чисел). Для удобства слова индексируются по общей частоте в наборе данных, так что, например, целое число "3" кодирует 3-е наиболее частое слово в данных. Это позволяет выполнять быстрые операции фильтрации, такие как: "учитывайте только 10 000 наиболее распространенных слов, но исключите 20 наиболее распространенных слов".Как правило, "0" не обозначает конкретное слово, а вместо этого используется для кодирования любого неизвестного слова.

Пример с LSTM – Определить рейтинг фильма

```
import numpy as np
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras import models
from keras import layers
```

Пример с LSTM – Определить рейтинг фильма

```
number_of_f=1000
(data_train,target_train),(data_test,target_test)=imdb.load_data(num_words=number_of_f)
```

Пример с LSTM – Определить рейтинг фильма

data_train[0]

[1, 14, 22, 16, 43, 530, 973, 2, 2, 65, 458, 2, 66, 2, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2,
336, 385, 39, 4, 172, 2, 2, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2, 19, 14, 22, 4, 2, 2, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 2, 4, 22, 17, 515, 17, 12, 16, 626,
18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2, 2, 16, 480, 66, 2, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 2, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407,
16, 82, 2, 8, 4, 107, 117, 2, 15, 256, 4, 2, 7, 2, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 2, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2, 56, 26, 141, 6, 194, 2, 18,
4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 2, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 2, 88, 12, 16, 283, 5, 16, 2, 113, 103, 32, 15, 16, 2, 19, 178, 32]

218

```
[11] len(data_train[0])

     218

[12] len(data_train[1])

     189

[7]  target_train[0]

     1
```

Пример с LSTM – Определить рейтинг фильма

```
f_train=sequence.pad_sequences(data_train,maxlen=400)
f_test=sequence.pad_sequences(data_test,maxlen=400)
```

рейтинг

xlen=400)

en=400)

```
f_train[0]
```

```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         1,  14,  22,  16,  43, 530, 973,   2,   2,  65, 458,   2,  66,
         2,   4, 173,  36, 256,   5,  25, 100,  43, 838, 112,  50, 670,
         2,   9,  35, 480, 284,   5, 150,   4, 172, 112, 167,   2, 336,
       385,  39,   4, 172,   2,   2,  17, 546,  38,  13, 447,   4, 192,
        50,  16,   6, 147,   2,  19,  14,  22,   4,   2,   2, 469,   4,
        22,  71,  87,  12,  16,  43, 530,  38,  76,  15,  13,   2,   4,
        22,  17, 515,  17,  12,  16, 626,  18,   2,   5,  62, 386,  12,
         8, 316,   8, 106,   5,   4,   2,   2,  16, 480,  66,   2,  33,
         4, 130,  12,  16,  38, 619,   5,  25, 124,  51,  36, 135,  48,
        25,   2,  33,   6,  22,  12, 215,  28,  77,  52,   5,  14, 407,
        16,  82,   2,   8,   4, 107, 117,   2,  15, 256,   4,   2,   7,
         2,   5, 723,  36,  71,  43, 530, 476,  26, 400, 317,  46,   7,
         4,   2,   2,  13, 104,  88,   4, 381,  15, 297,  98,  32,   2,
        56,  26, 141,   6, 194,   2,  18,   4, 226,  22,  21, 134, 476,
        26, 480,   5, 144,  30,   2,  18,  51,  36,  28, 224,  92,  25,
       104,   4, 226,  65,  16,  38,   2,  88,  12,  16, 283,   5,  16,
         2, 113, 103,  32,  15,  16,   2,  19, 178,  32], dtype=int32)
```

Пример с LSTM – Определить рейтинг фильма

```
net1=models.Sequential()
```

```
net1.add(layers.Embedding(input_dim=number_of_f,output_dim=128))
```

```
net1.add(layers.LSTM(units=128))
```

```
net1.add(layers.Dense(units=1,activation="sigmoid"))
```

```
net1.compile(loss="binary_crossentropy",optimizer="adam",metrics=["accuracy"])
```

Пример с LSTM – Определить рейтинг фильма

```
[20] net1.compile(loss="binary_crossentropy",optimizer="adam",metrics=["accuracy"])
```

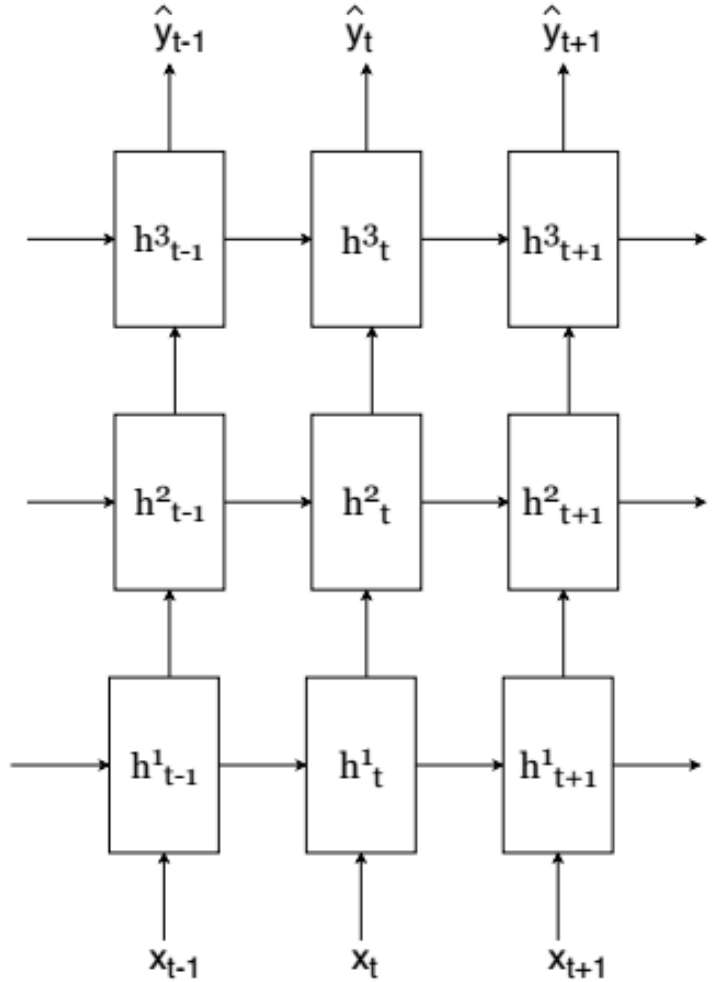
```
[21] hist1=net1.fit(f_train,target_train,epochs=3,verbose=1,batch_size=1000,validation_data=(f_test,target_test))
```

```
Epoch 1/3
25/25 [==============================] - 364s 15s/step - loss: 0.6894 - accuracy: 0.6217 - val_loss: 0.6634 - val_accuracy: 0.6865
Epoch 2/3
25/25 [==============================] - 356s 14s/step - loss: 0.6517 - accuracy: 0.6979 - val_loss: 0.6319 - val_accuracy: 0.7010
Epoch 3/3
25/25 [==============================] - 348s 14s/step - loss: 0.5294 - accuracy: 0.7560 - val_loss: 0.4231 - val_accuracy: 0.8150
```

Глубокие рекуррентные сети (deep RNN, layers stacking)

Выходы одной рекуррентной сети подаются на вход другой:

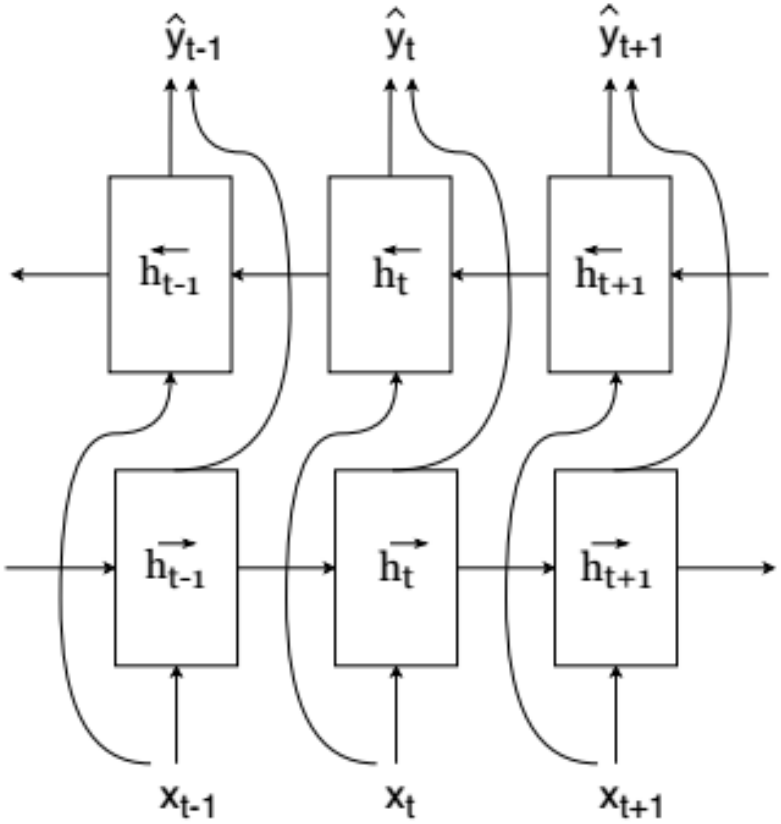
$$h_t^1, C_t^1 = LSTM(h_{t-1}^1, C_{t-1}^1, x_t)$$
$$h_t^2, C_t^2 = LSTM(h_{t-1}^2, C_{t-1}^2, h_t^1)$$
$$h_t^3, C_t^3 = LSTM(h_{t-1}^1, C_{t-1}^1, h_t^2)$$
$$y_t = g(Uh_t^2 + \hat{b})$$



Двунаправленные сети (biderectional)

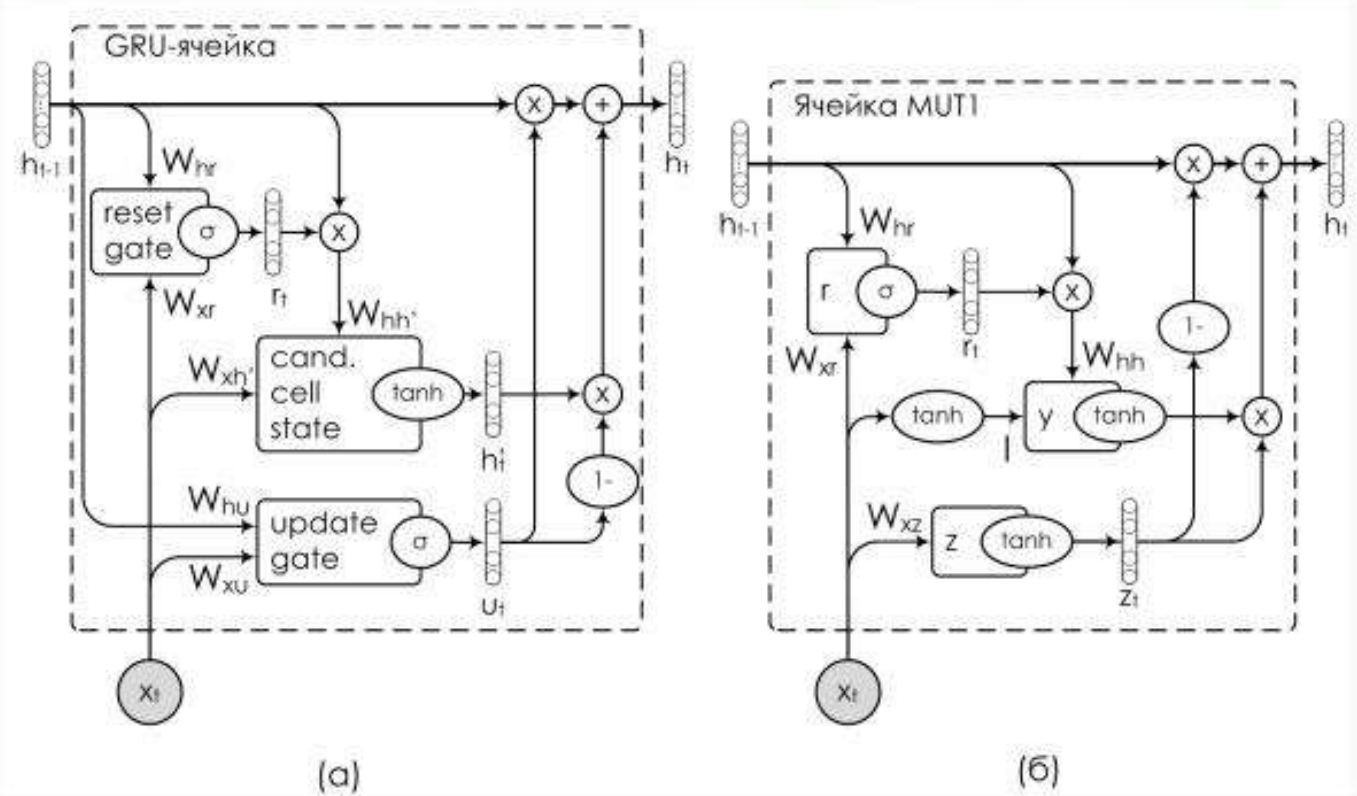
Конкатенация выходов двух сетей, одна идёт слева направо, другая справа налево:

$$\overrightarrow{h_t}, \overrightarrow{C_t} = \overrightarrow{LSTM}(\overrightarrow{h_{t-1}}, \overrightarrow{C_{t-1}}, x_t)$$
$$\overleftarrow{h_t}, \overleftarrow{C_t} = \overleftarrow{LSTM}(\overleftarrow{h_{t-1}}, \overleftarrow{C_{t-1}}, x_t)$$
$$y_t = g(U[\overrightarrow{h_t}, \overleftarrow{h_t}] + \hat{b})$$



На практике часто работают лучше чем однонаправленные!

- ...Gated Recurrent Units (GRU; Cho et al., 2014).
- В GRU тоже есть прямой путь для градиентов, но проще.

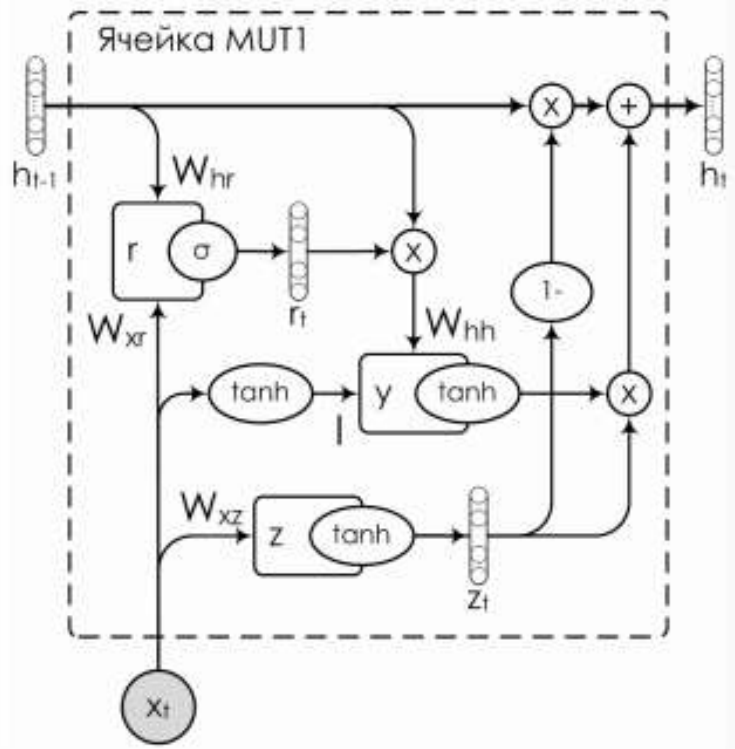


- Формально:

$$
\begin{aligned}
\boldsymbol{u}_t &= \sigma(W_{xu}\mathbf{x}_t + W_{hu}\boldsymbol{h}_{t-1} + \mathbf{b}_u) \\
\boldsymbol{r}_t &= \sigma(W_{xr}\mathbf{x}_t + W_{hr}\boldsymbol{h}_{t-1} + \mathbf{b}_r) \\
\boldsymbol{h}'_t &= \tanh(W_{xh'}\mathbf{x}_t + W_{hh'}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})) \\
\boldsymbol{h}_t &= (1 - \boldsymbol{u}_t) \odot \boldsymbol{h}'_t + \boldsymbol{u}_t \odot \boldsymbol{h}_{t-1}
\end{aligned}
$$

- Теперь есть update gate и reset gate, нет разницы между $\boldsymbol{c}_t$ и $\boldsymbol{h}_t$.
- Меньше матриц (6, а не 8 или 11 с замочными скважинами), меньше весов, но только чуть хуже LSTM работает.
- Так что можно больше GRU поместить, и сеть станет лучше.

- Другие варианты тоже есть.
- (Józefowicz, Zaremba, Sutskever, 2015): огромное сравнение, выращивали архитектуры эволюционными методами.
- Три новых интересных архитектуры; например:

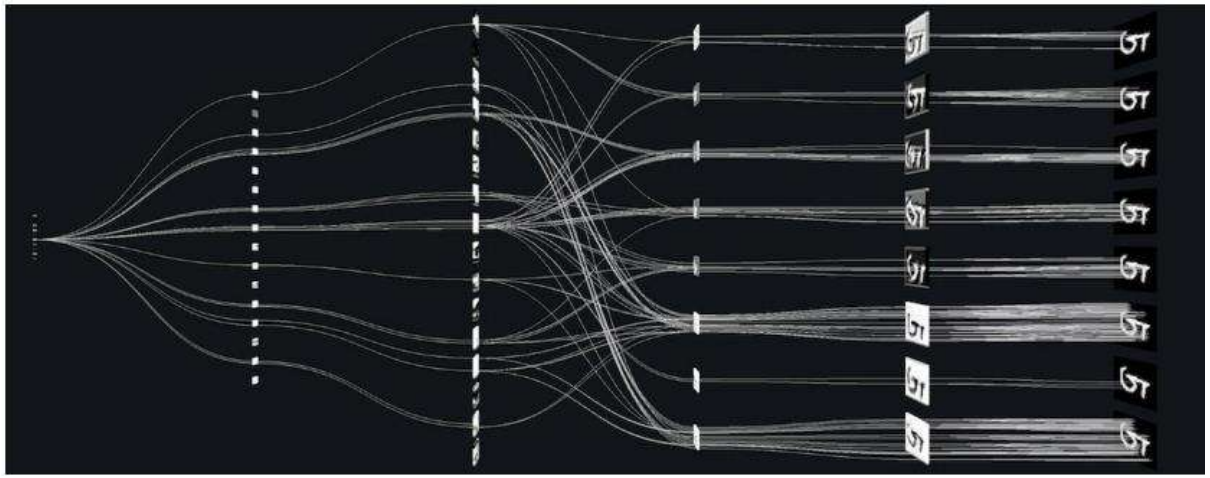


Основы нейронных сетей

Лекция 13

Свёрточные сети

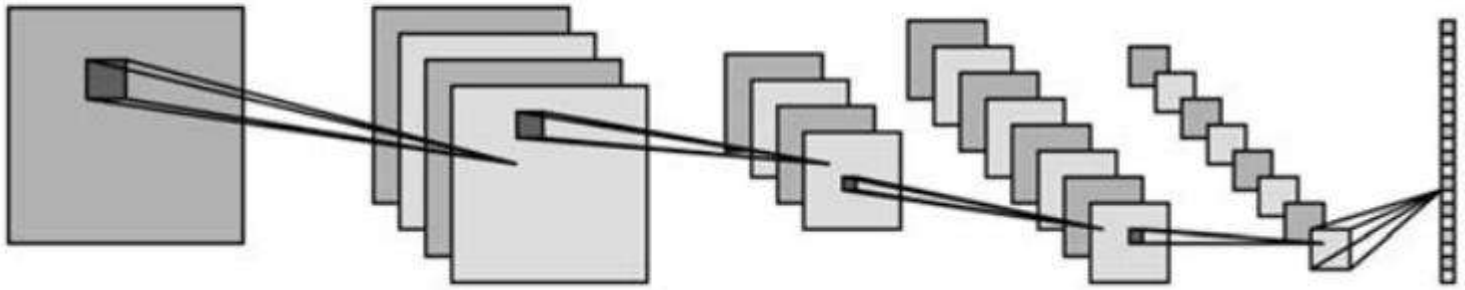
CNN



- эффективное распознавание изображений.
- В основе сверточной нейронной сети заложены некоторые особенности зрительной коры головного мозга, где существует набор простых клеток, реагирующих на наличие прямых линий, расположенных под разными углами, и сложных клеток, реагирующих на активацию определенного набора простых клеток.
- Архитектура была предложена Яном Ле Куном в 1998

Архитектура CNN

- Сверточный слой (Convolution layer)
- Субдискретизирующий слой (Subsampling layer)
- Полносвязный слой (Стандартный слой персептрона)



Архитектура CNN. Свёртка

- фильтр для каждого канала изображения, ядро которого представляет собой матрицу.
- Механизм свертки заключается в последовательном преобразовании фрагментов предыдущего слоя скользящим окном.
- Каждый фрагмент следующего слоя является результатом матричного умножения соответствующего фрагмента предыдущего слоя на ядро свертки.

Архитектура CNN. Свёртка

- Такой подход к архитектуре принципиально отличает сверточную нейронную сеть от полносвязной нейронной сети, позволяя не хранить веса связей для каждого из выходных нейронов, вместо этого возможно хранить лишь значения матрицы ядра свертки.
- локальная связность элементов изображения сохраняется, в то время как для многослойного перцептрона локальная связность не учитывается.
- Таким образом, значительно сокращается количество настраиваемых параметров, а это, в свою очередь, упрощает процесс обучения классификатора.

Архитектура CNN. Свёртка

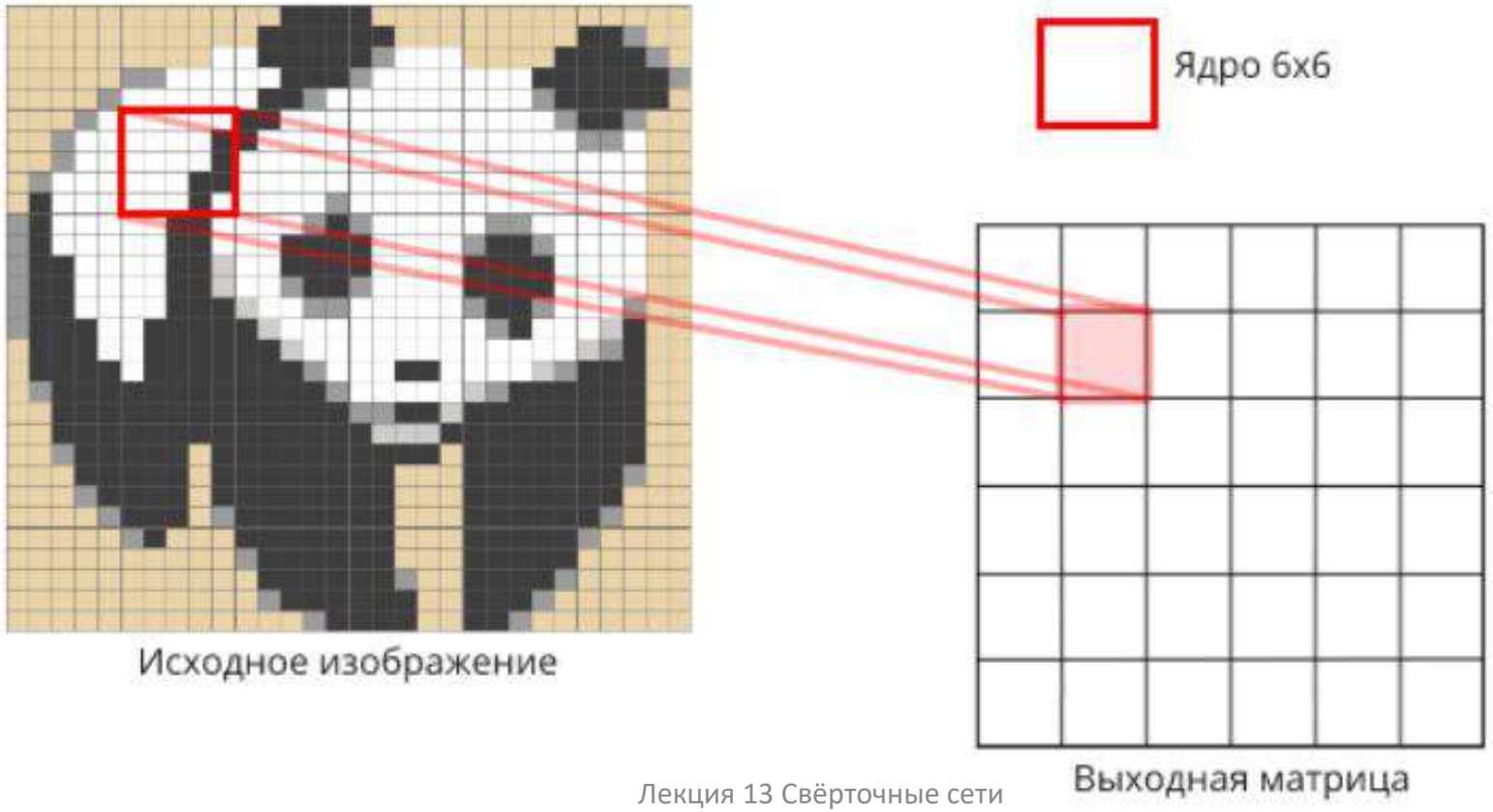
— операция часто использующаяся для обработки изображений и может быть описана формулой:

$$(f * g)[m,n] = \sum_{k,l} f[m-k, n-l] * g[k,l]$$

где *f* – исходная матрица изображения размера *[m,n], g* – ядро (матрица) свертки размера *[k,l]*

Архитектура CNN. Свёртка

- Визуализация операции свертки (фильтры)



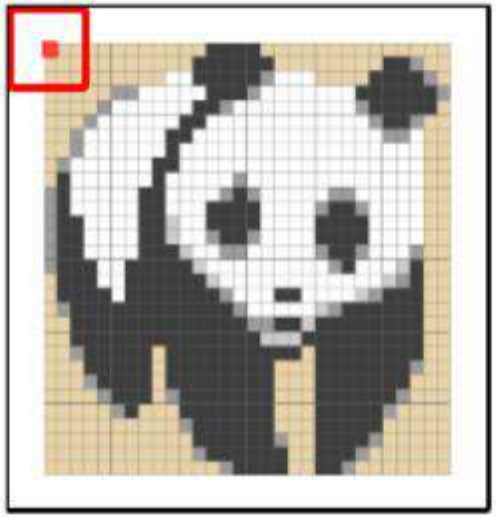
Архитектура CNN. Свёртка

При этом в зависимости от метода обработки краев исходной матрицы, результат может быть

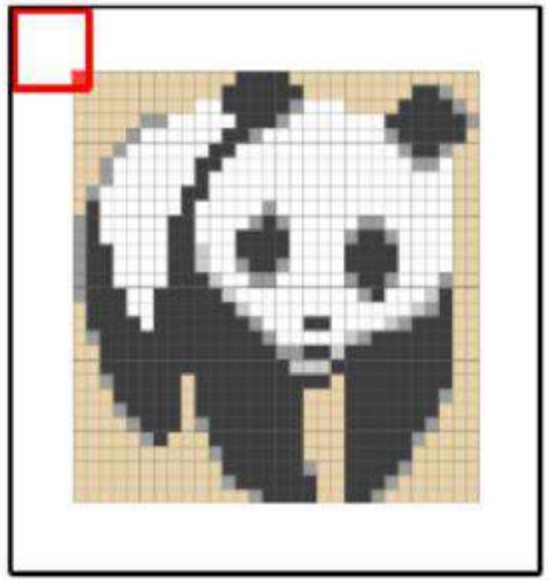
- меньше исходного изображения(valid),
- такого же размера (same)
- или большего размера (full).

При этом элементы вне матрицы изображения считаются нулевыми и не вносят вклад в результат воздействия оператора свертки.

valid



same



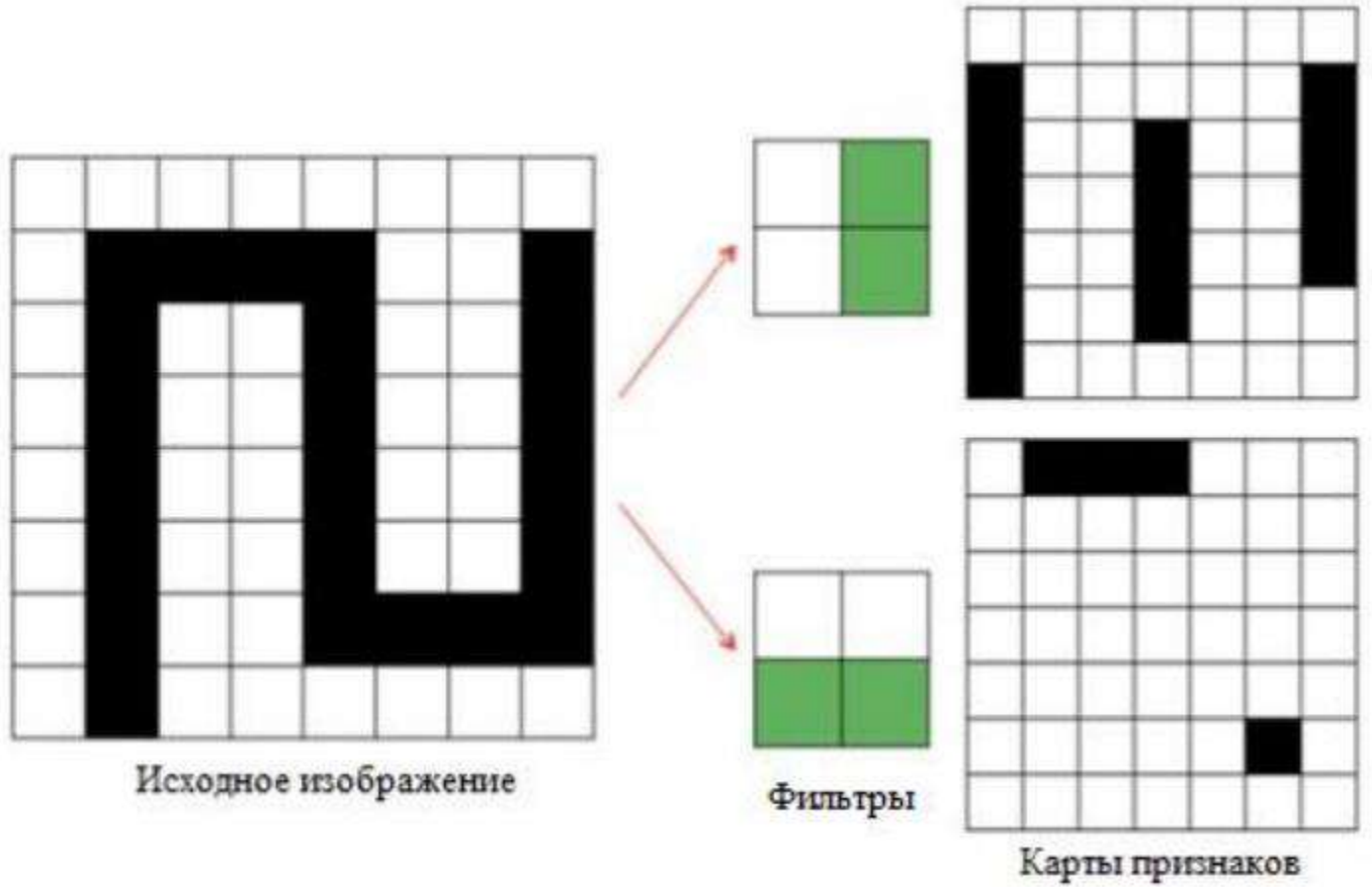
full

Архитектура CNN. Свёртка

- Фильтры позволяют находить определенные признаки на исходном изображении
- на выходе получаем так называемую «карту признаков», которая показывает какие части исходного изображения содержат описанный в фильтре признак.

Архитектура CNN. Свёртка

- Фильтры позволяют находить определенные признаки на исходном изображении
- на выходе получаем так называемую «карту признаков», которая показывает какие части исходного изображения содержат описанный в фильтре признак.



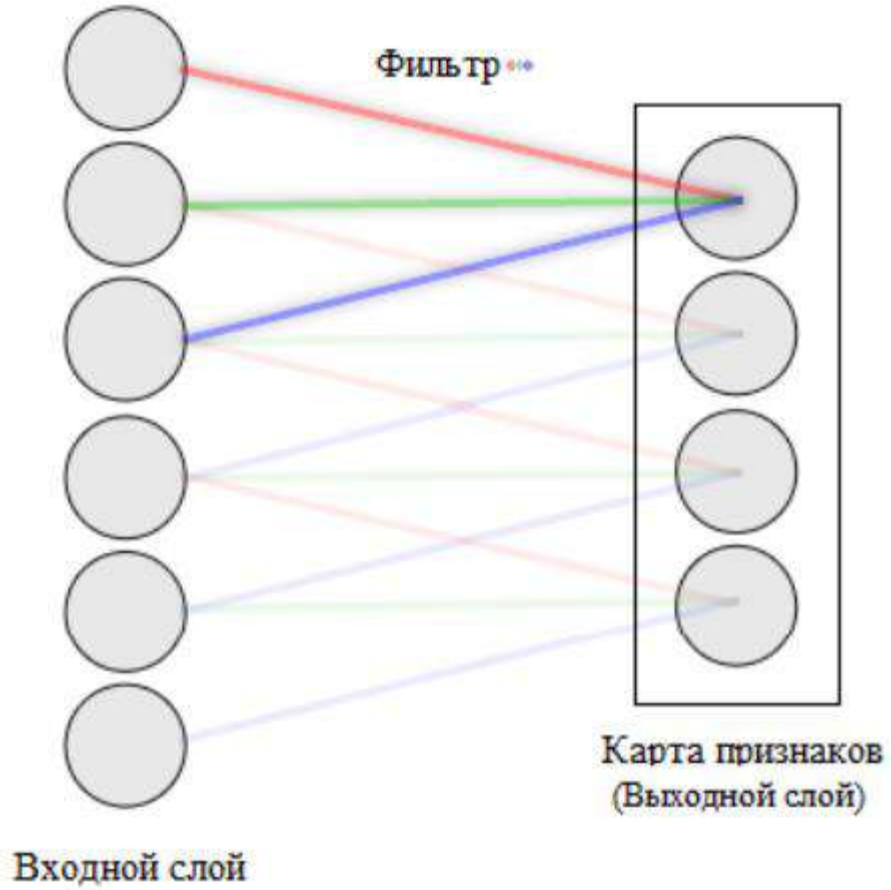
Архитектура CNN. Свёртка в НС

- Набор весов связей одинаков для каждого нейрона выходного слоя и по сути своей является фильтром.

- Каждый нейрон данного слоя соединен лишь с частью входных данных и выполняет операцию свертки, составляя карту признаков.

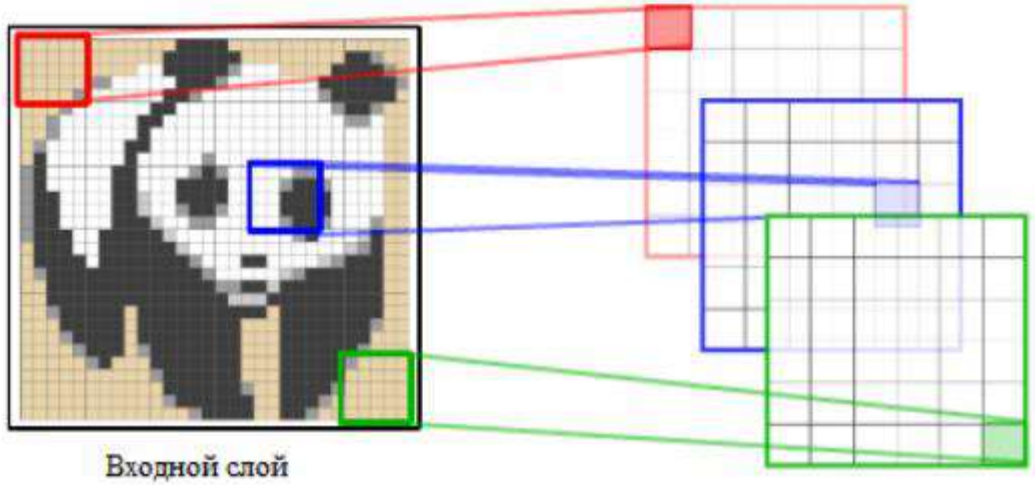
$$x^l = f\left(x^{l-1} * k^l + b^l\right)$$

где x^l— выход слоя l,
f — функция активации,
b^l — коэффициент сдвига для карты признаков l,
k^l — l-ое ядро свертки.



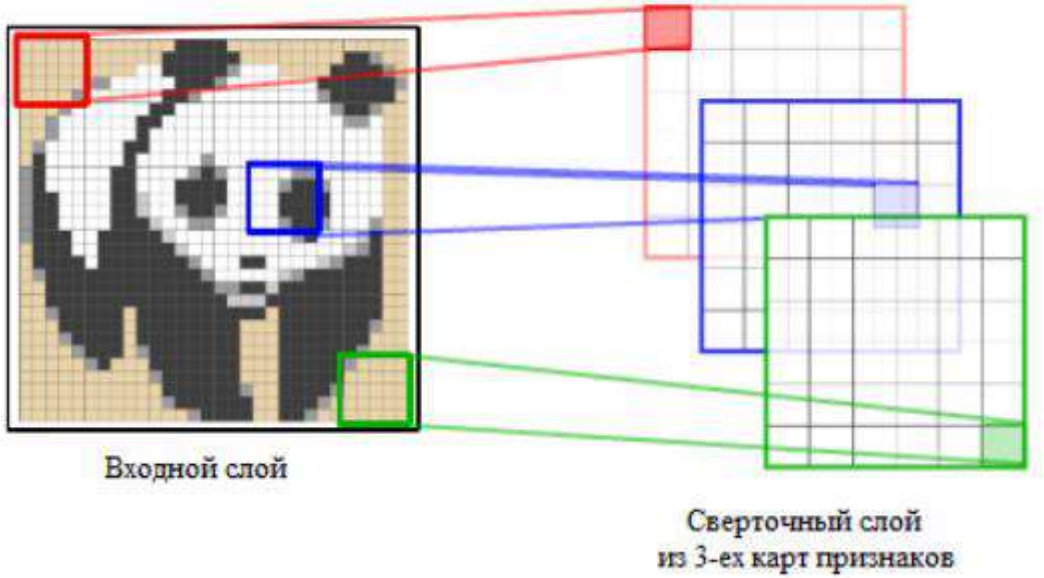
Архитектура CNN. Свёртка в НС

- Таких фильтров, выявляющих множество различных признаков, может быть несколько. В таком случае получается несколько карт признаков, такой набор карт и называется сверточным слоем.
-



Архитектура CNN. Свё

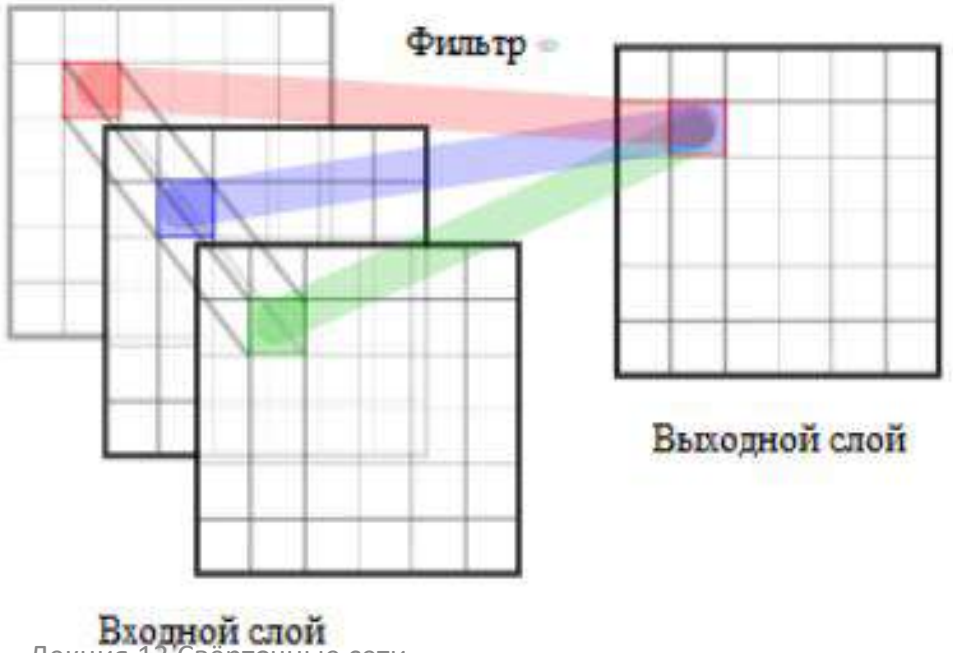
- Таких фильтров, выявляющих множе
может быть несколько. В таком случа
признаков, такой набор карт и назыв



- эти фильтры не закладываются исследователем заранее, а формируются путем обучения сети.

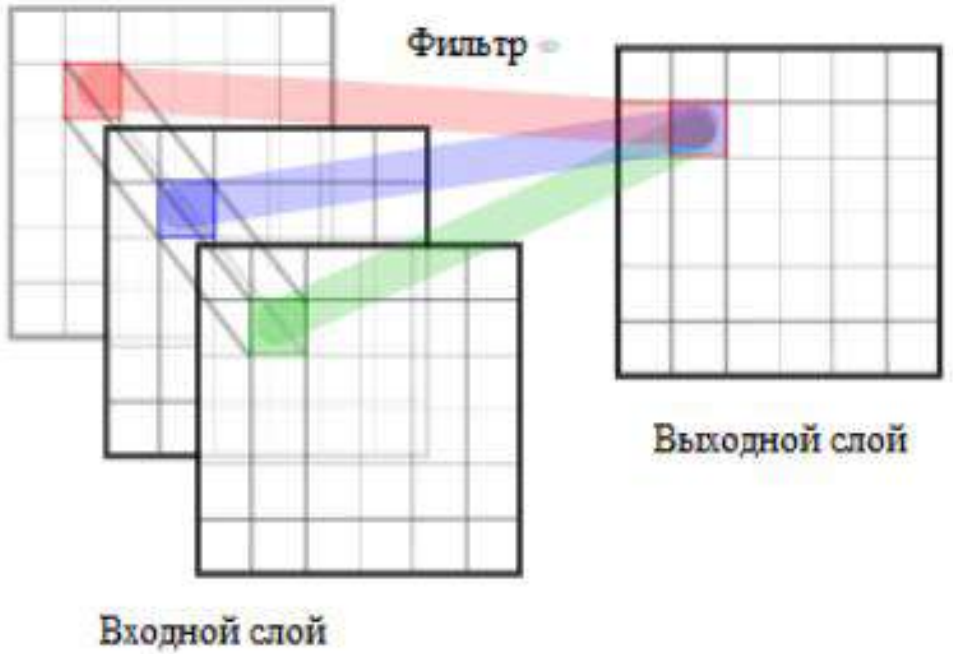
Архитектура CNN. Свёртка в НС

Так как требуется учесть информацию с нескольких карт признаков, то следует создать фильтры, зависящие от всего объема информации и пересекающие несколько карт признаков в пределах слоя.

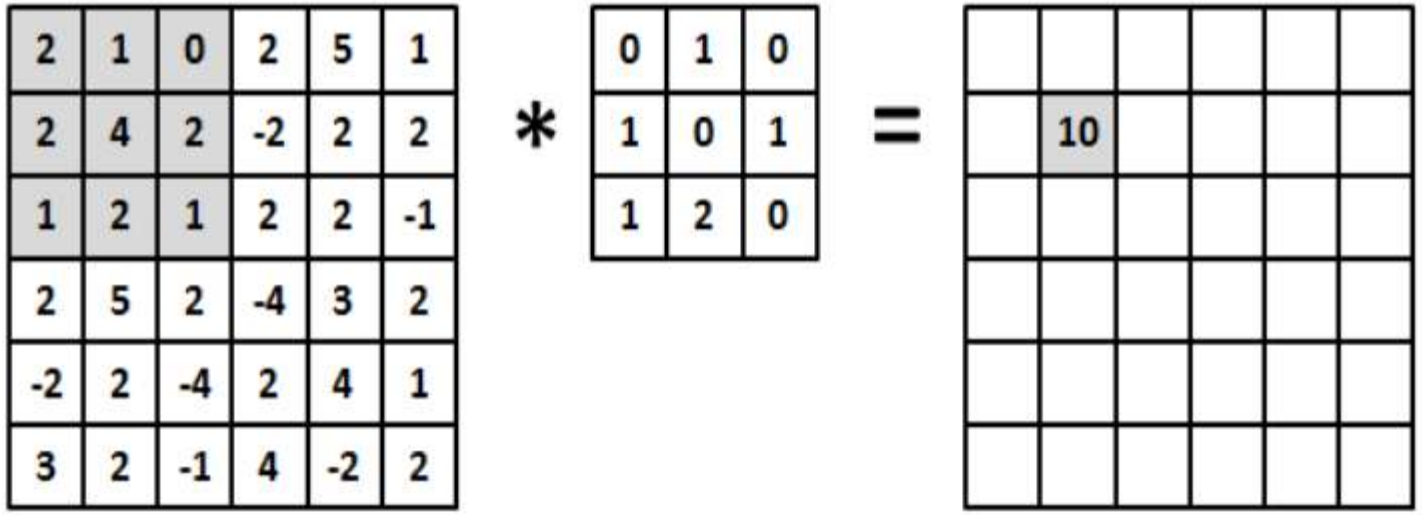


Архитектура CNN. Свёртка в НС

Входной слой учитывает двумерную топологию изображений и состоит из нескольких карт (матриц), карта может быть одна, в том случае, если изображение представлено в оттенках серого, иначе их 3, где каждая карта соответствует изображению с конкретным каналом (красным, синим и зеленым).



Архитектура CNN. Свёртка в НС

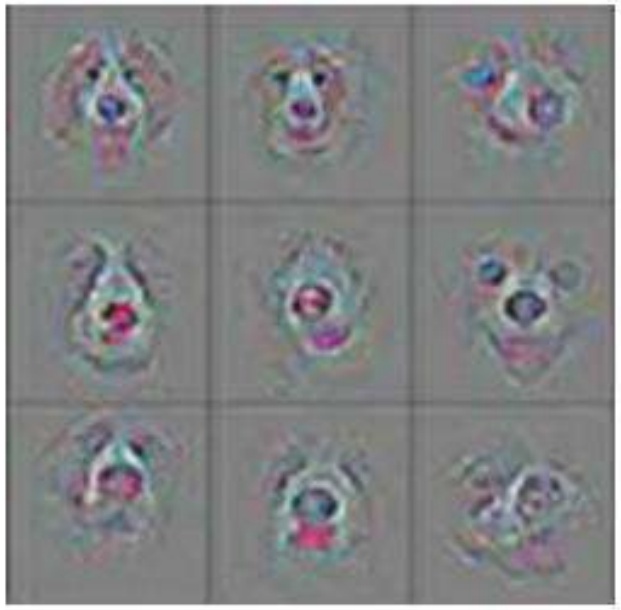


- Ограничений на выбор функции активации для сверточного слоя нет
- функция ReLU:

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

Архитектура CNN. Свёртка в НС

- Веса сверточных слоев (фильтры) не закладываются исследователем заранее, а формируются путем обучения сети. В этом и заключается основная идея CNN. Во время обучения сеть сама подбирает признаки, по которым она будет различать входные данные максимально эффективно. Другими словами, сеть запоминает конкретные визуальные образы, схожие с классами, которой ей требуется распознавать



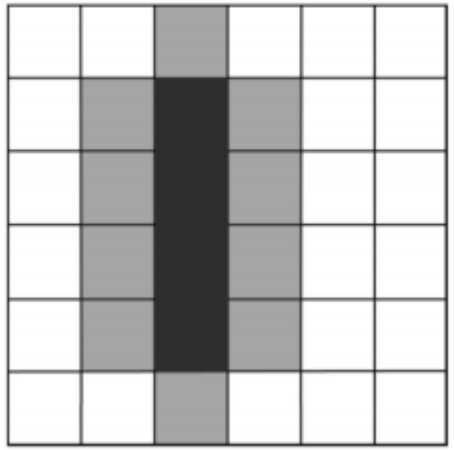
Фильтры (запомненные признаки) полученные при обучении CNN распознавать собак

Параметры свёрточного слоя

- **Число признаков (filters count, fc)** – это количество фильтров, которые есть в слое.
- **Размер фильтров (filter size, fs)** – это высота и ширина тензора фильтров. Обычно является нечётным числом, наиболее часто используются фильтры размером 3 или 5.
- **Шаг свёртки (stride, S)** – это количество пикселей, на которое перемещается матрица фильтра по входному изображению. Когда шаг равен 1, фильтры перемещаются по одному пикселю за раз. Когда шаг равен 2, тогда фильтры перескакивают на 2 пикселя за раз. Чем больше шаг, тем меньшего размера карты признаков получаются на выходе.
- **Дополнения нулями (padding, P)**

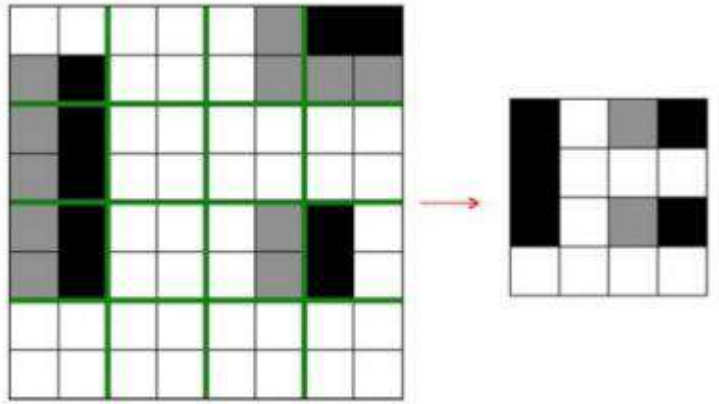
Архитектура CNN. Субдискретизирующий слой

- чтобы уточнить информацию на полученных картах признаков, поскольку результата применения фильтра может быть недостаточно. Ключевые пиксели карты признаков могут быть окружены «ореолом» из-за которого в дальнейшем признак может быть определен неправильно.
- Чтобы избавиться от этой проблемы субдикретизирующий слой выполняет уменьшение размера входной карты признаков.
- Основной его функцией является нелинейное уплотнение карты признаков.



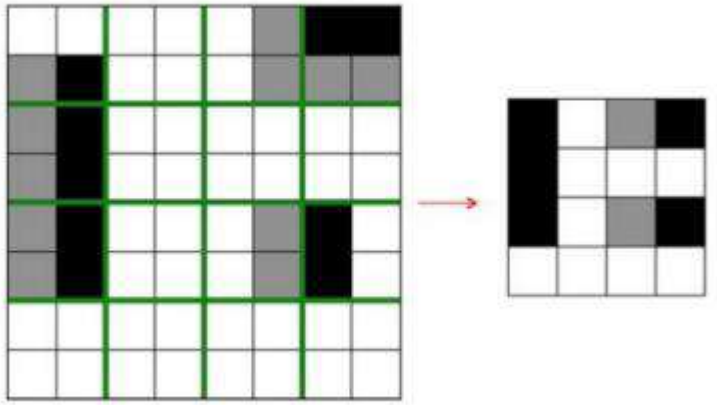
Архитектура CNN. Субдискретизирующий слой

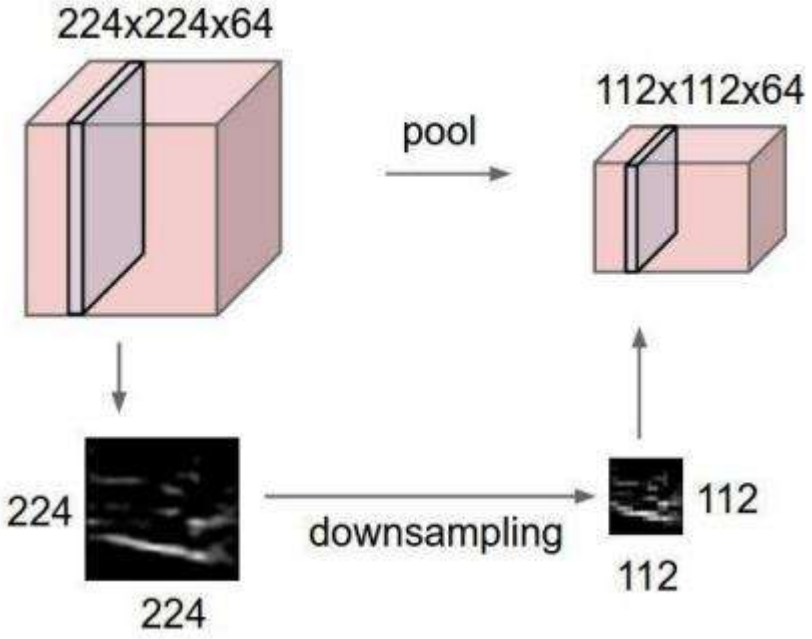
- Это можно делать разными способами, но самым популярным является метод «max-pooling» — вся карта признаков разделяется на ячейки 2x2 элемента, из которых выбираются максимальные по значению.

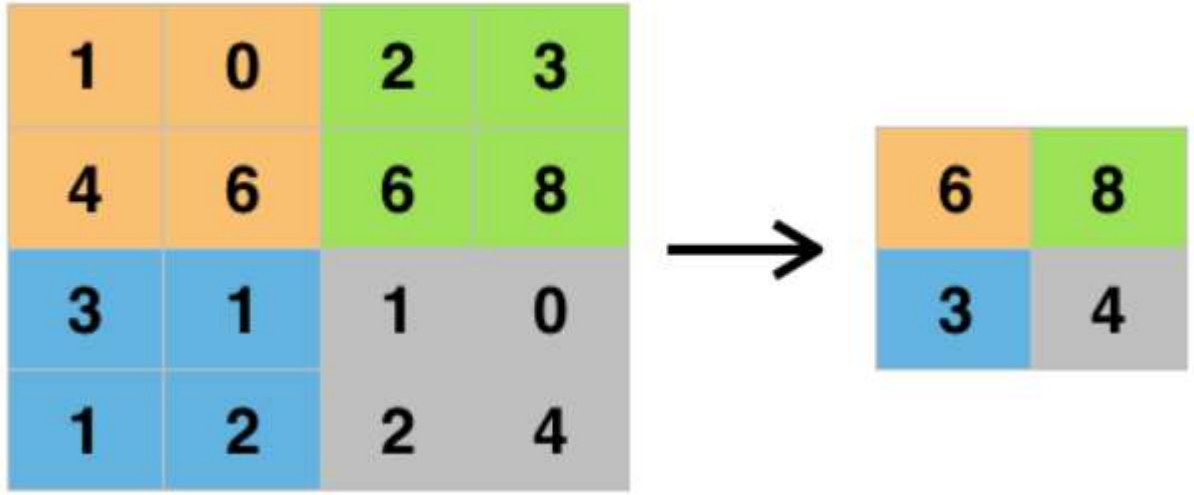


Архитектура CNN. Субдискретизирующий слой

- Это можно делать разными способами, но самым популярным является метод «max-pooling» — вся карта признаков разделяется на ячейки 2x2 элемента, из которых выбираются максимальные по значению.







Архитектура CNN. Субдискретизирующий слой

- Каждая операция субдискретизации позволяет значительно уменьшить размерность изображения, фильтруя ненужные детали, позволяя избежать переобучения.
- В основе лежит идея о том, что после получения набора признаков на каждом шаге, на следующем шаге настолько подробное изображение уже не требуется.
- Использование этого слоя позволяет сократить количество параметров НС и устранить ненужные признаки, а также создать некоторую инвариантность относительно сдвига изображения.

Архитектура CNN. Субдискретизирующий слой

- Данный слой может быть описан формулой:

$$x^l = f\left(a^l \cdot subsample\left(x^{l-1}\right) + b^l\right)$$

- где x^l— выход l-го слоя, f — функция активации, a,b— коэффициенты, $subsample$— операция выборки локальных максимальных значений.

Архитектура CNN. Полносвязный слой

- Данные после объединения подаются на вход полносвязной нейронной сети, которая, в свою очередь, также может состоять из нескольких слоёв

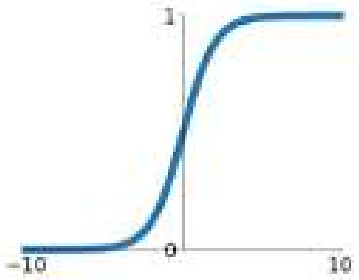
$$x_j^l = f\left(\sum_i x_i^{l-1} * k_j^l + b_j^l\right)$$

- наиболее популярным методом обучения CNN является метод обратного распространения ошибки (и его модификации).
- Но существует и множество других способов обучения, как с учителем, так и без.

Функции активации

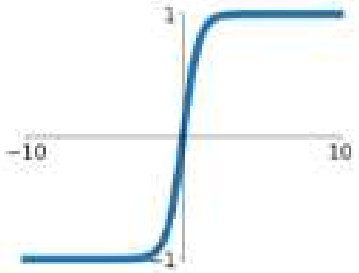
Sigmoid

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



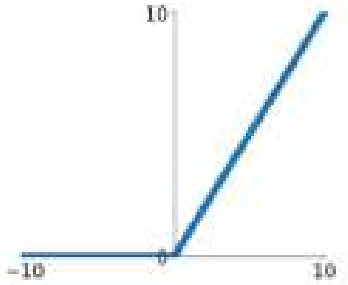
tanh

$$\tanh(x)$$



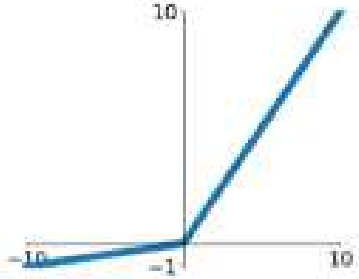
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

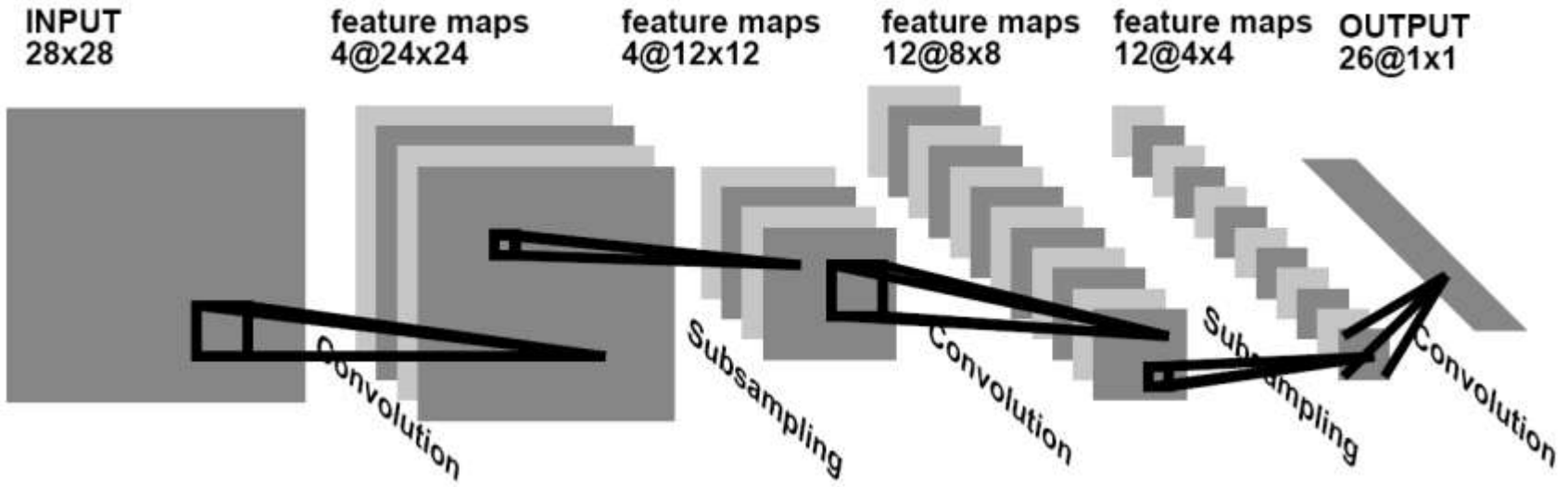


Архитектура CNN. Полносвязный слой

- Полносвязная нейронная сеть является завершающим уровнем при построении сети этой архитектуры.
- После прохождения всех предыдущих слоев данные преобразуются от исходного изображения к более абстрактным картам признаков, объем данных гораздо меньше исходного, эти данные интерпретируются как абстрактные признаки, выявленные в изображении.

Архитектура CNN

- Архитектура CNN состоит из использовании слоев свертки, слоев субдискретизации.



Архитектура CNN

- Это позволяет составлять карты признаков из карт признаков, что означает способность распознавания сложных иерархий признаков.
- После прохождения множества слоев CNN карта признаков вырождается в вектор или даже скаляр, но таких карт признаков становится сотни, поэтому на выходе в сети устанавливают несколько слоев полносвязной НС (персептрона) (или любого другого классификатора) на вход которым и подаются конечные признаки в виде вектора.

Задания для самостоятельного решения по разделам классификация и кластеризация.

1. Точки {(4, –1), (8, –2), (1, 1), (3, 6)} принадлежат классу A, а точки {(–8, 4), (–2, –3), (–1, –1), (2, –9)} — классу B. Постройте минимальную сеть, правильно классифицирующую эти точки.

2. Определите сеть с радиальными базисными функциями, решающую проблему XOR, в предположении, что функции активности скрытых элементов имеют вид $\varphi(r) \approx \sqrt{c^2 + r^2}$.

Задачи по теме «Кластеризация»

3. Векторы $\mathbf{x}$, $\mathbf{p}_1$ и $\mathbf{p}_2$ являются следующими:

$$\mathbf{x} = [0.2 \quad -1.4 \quad 2.3],$$

$$\mathbf{p}_1 = [0.6 \quad -4.0 \quad 7.0],$$

$$\mathbf{p}_2 = [0.1 \quad -1.0 \quad 2.2].$$

(а) К какому из прототипов оказывается ближе всего вектор $\mathbf{x}$ в смысле евклидова расстояния?

(б) К какому из прототипов оказывается ближе всего вектор $\mathbf{x}$ в смысле скалярного произведения?

(в) Скорректируйте весовой вектор прототипа-победителя из п. (а) в соответствии с алгоритмом обучения SOFM при норме обучения 0.8.

(г) Скорректируйте весовой вектор прототипа-победителя из п. (а) в соответствии с алгоритмом обучения SOFM для случая использования скалярного произведения при норме обучения 0.8.

4. Норма обучения в сети SOFM уменьшается в течение первых 1000 итераций по закону

$$\eta(n) = 0.15\left(1 - \frac{n}{1000}\right),$$

где n обозначает номер итерации.

(а) Сколько итераций будет выполнено прежде, чем норма обучения уменьшится до значения 0.003?

(а) Почему указанный закон не является хорошим выбором?

Задания по разделу классификация.

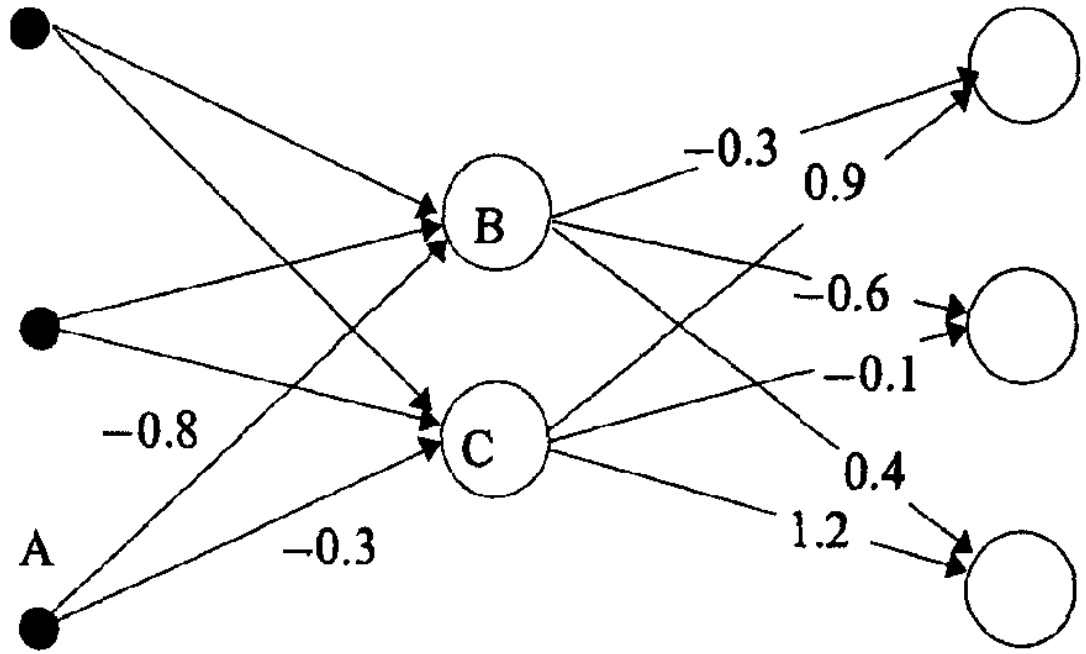
1. На рис. показана сеть с обратным распространением ошибок во время обработки учебного вектора [1 0 0.9 0.9], для которого целевым выходным вектором является [0.1 0.9 0.1]. Пусть выходом элемента B является значение 0 6, а выходом элемента C — значение 0 8. Предположим, что функцией активности является сигмоид.

(a) Вычислите фактический выходной вектор.

(b) Вычислите значения ошибок для каждого выходного элемента.

(в) Вычислите значения ошибок для каждого скрытого элемента.

(г) Вычислите изменения весовых значений для связей, идущих от элемента A. Норма обучения предполагается равной 0 25



2. Предположим, что точки {(–1, 1), (–1, –1). (1, –1)} принадлежат классу A, а точки {(–2, –2), (1, 1), (2, 2), (4, 1)} — классу B

(а) Докажите, что эти классы не являются линейно отделимыми.

(б) Предположив, что выход элементов сети задается условием

$$\text{выход} = \begin{cases} 1, & \text{если комбинированный ввод} \geq 0, \\ 0, & \text{если комбинированный ввод} < 0, \end{cases}$$

покажите, что определенный ниже матрицей $\mathbf{W}_1$ первый слой весовых значений в сети с тремя слоями преобразует проблему в линейную (первая строка матрицы $\mathbf{W}_1$ определяет весовые коэффициенты смещения):

$$\mathbf{W}_1 = \begin{bmatrix} 1 & -6 \\ -2 & -2 \\ -1 & -3 \end{bmatrix}.$$

3. Сеть типа 2-2-1 с радиальными базисными функциями используется для решения проблемы XOR Первый слой весов задан матрицей

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Покажите, что данная сеть решает задачу XOR.