Искусственные нейронные сети: основы практического применения

Лекция 1

Крощенко А.А.

Брестский государственный технический университет

23.05.2017

Где используются нейронные сети сегодня?

- Обработка естественного языка
- Автоматический машинный перевод (в том числе текста на изображениях)
- Распознавание изображений
- Сегментация изображений (выделение объектов и их последующее распознавание)
- Генерация рукописного текста
- Синтез художественных изображений (картин)
- Как ключевая часть игровых ботов
- Прогнозирование курса валют и котировок акций
- Составная часть робототехнических систем разных уровней
- Прогнозирование погодных аномалий
- Компонент различных медицинских систем

Известные технологии и приложения, использующие нейронные сети

- Навигационная система Neurala марсохода Curiosity
- Персональный ассистент Siri – способна предугадывать и понимать естественно-языковые вопросы и запросы (Apple).
- Alexa – технология умного дома от Amazon. Способна искать информацию в интернете, делать покупки, планировать расписание, управлять освещением в доме, выполнять полив, регулировать термостат и многое другое. Управление голосовыми командами.
- Prisma App. - приложение для создания картин по фотографиям, использующее различные художественные стили.

Ключевые темы курса

1. Обучение и функционирование линейного нейрона
2. Многослойный персептрон. Глубокий многослойный персептрон
3. Задачи классификации и регрессии
4. Задача кластеризации
5. Радиально-базисная НС и ее приложение
6. Автоассоциативная НС, автоэнкодер. Задача семантического кодирования
7. Лингвистический анализ и НС
8. Рекуррентные НС
9. Сверточные НС. Распознавание рукописных цифр
10. Глубокие сверточные НС. Сегментация изображений

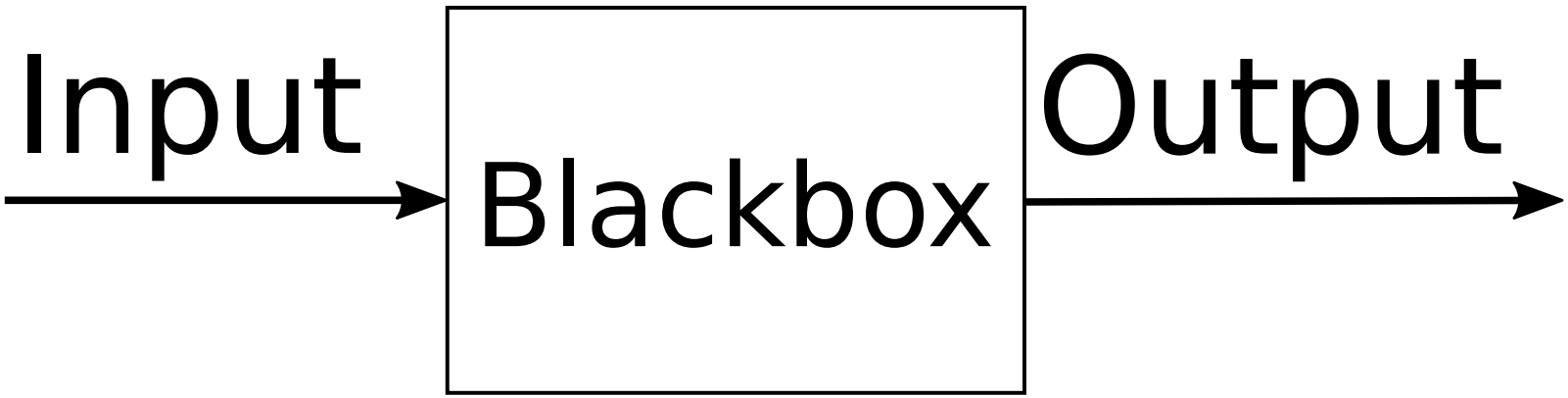
Литература

1. Головко В.А. Нейроинтеллект: теория и применение. Книга 1: Организация и обучение нейронных сетей с прямыми и обратными связями. Брест Изд. БПИ, 1999 – 264 с.
2. Хайкин С. Нейронные сети: полный курс, 2-е издание. – Москва, ИД "Вильямс", 2016 – 1104 с.
3. Флах, П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. – Москва, ДМК Пресс, 2015 – 400 с.
4. Введение в статистическое обучение с примерами на языке R. – Москва, ДМК Пресс, 2016 – 460 с. (https://github.com/ranalytics/islr-ru)h

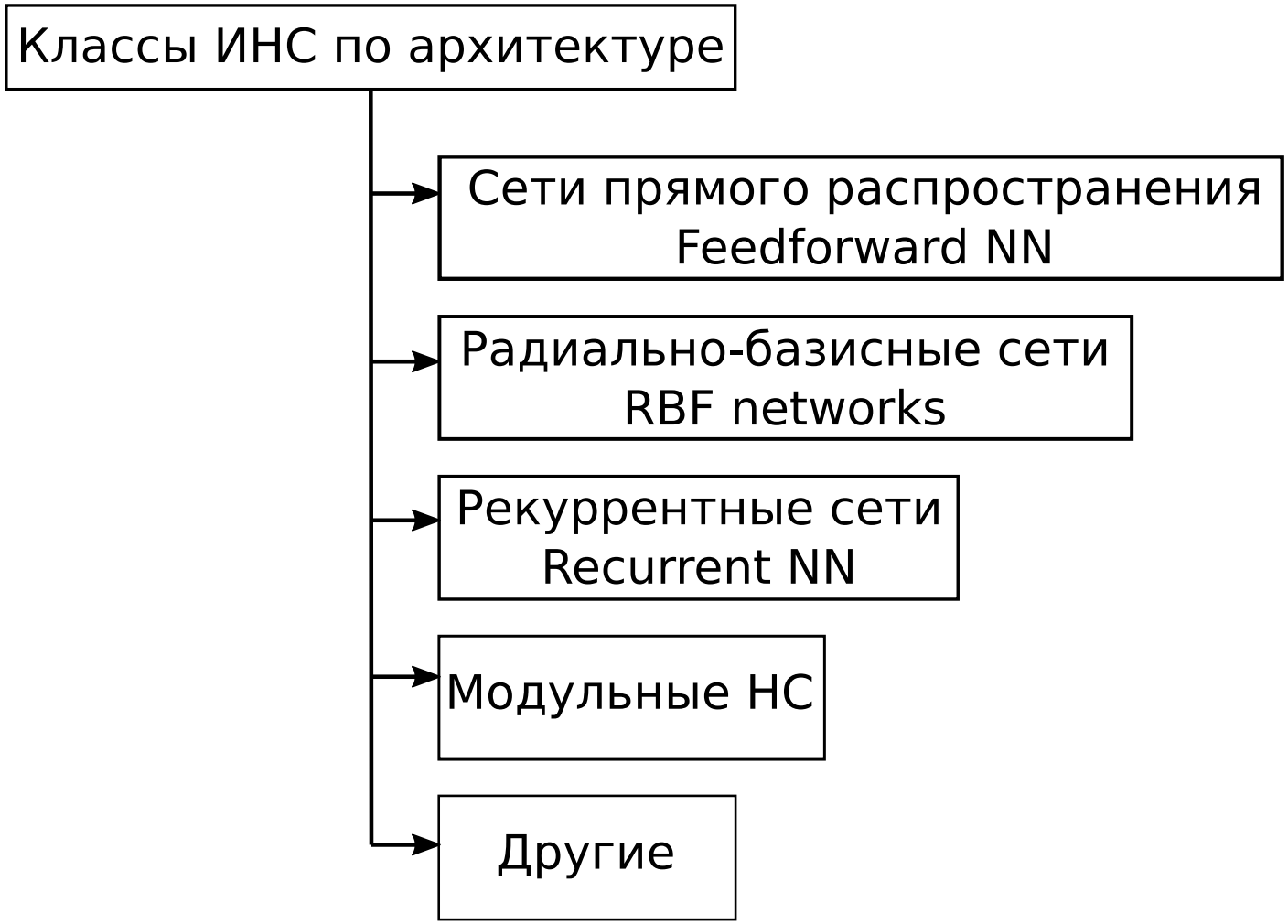
Программные средства, используемые в курсе

- Интерпретатор языка программирования Python с установленными пакетами matplotlib, numpy, scikit-learn и др.
- Фреймворк Tensorflow
- Фреймворк Caffe/Caffe2
- IDE PyCharm Community Edition
- github (`https://github.com/kroschenko/IHSMarkit_NN_course`)

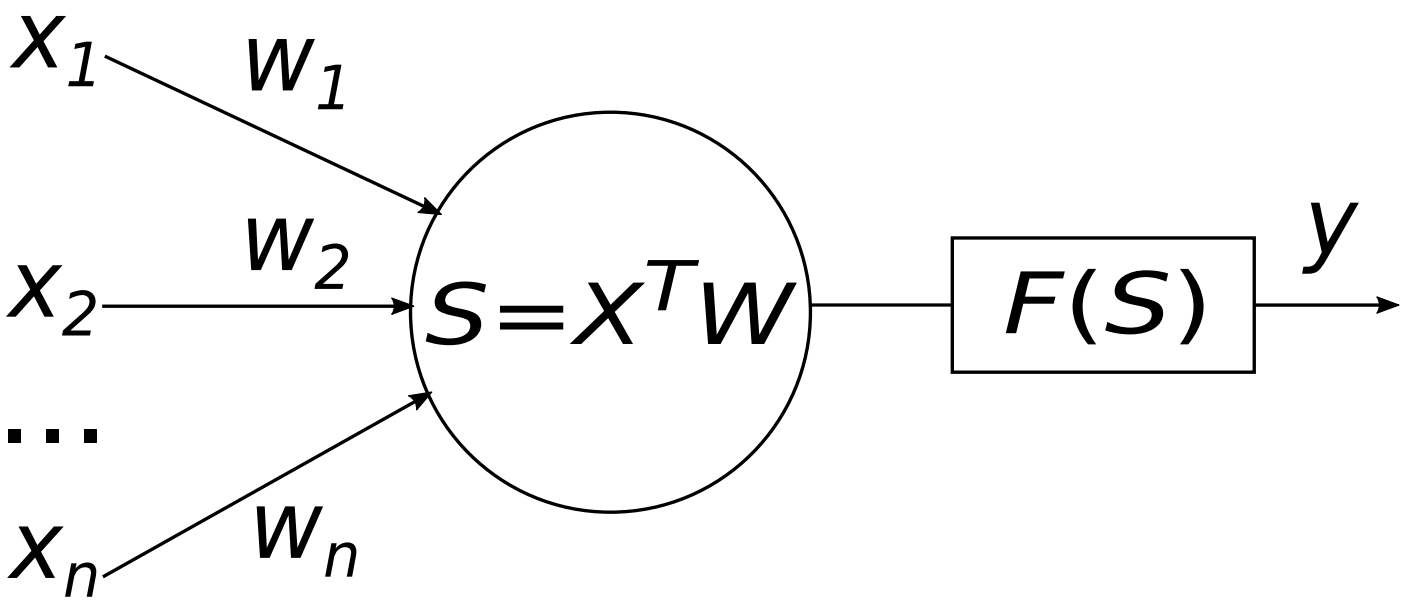
Что такое ИНС?



Классификация ИНС



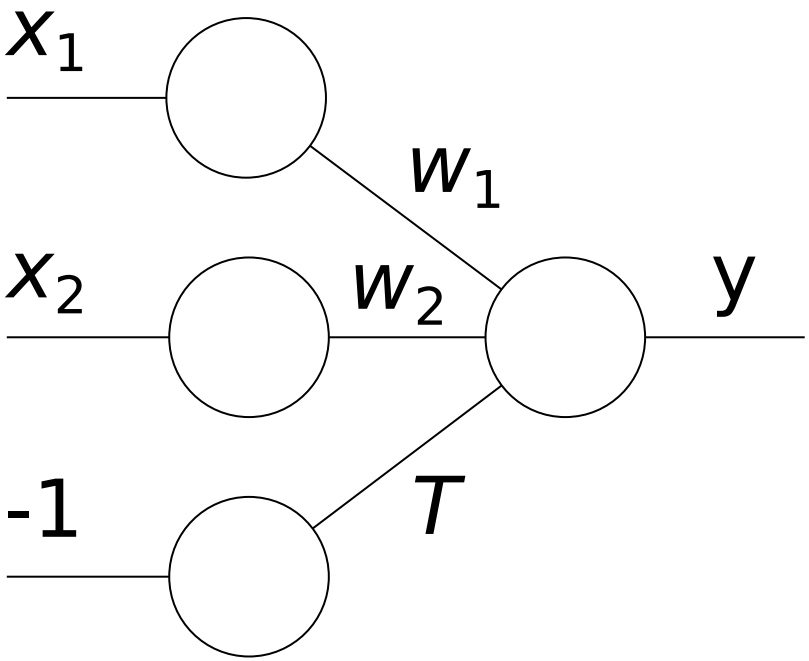
Структура искусственного нейрона



$$X = (x_1, x_2, ..., x_n)^T, W = (w_1, w_2, ..., w_n)^T$$

$$y = F\Big(\sum_{i=0}^{n} x_i w_i\Big) \tag{1}$$

Искусственный нейрон с двумя входами



$$y = F(w_1 x_1 + w_2 x_2 + T) \tag{2}$$

Функции активации

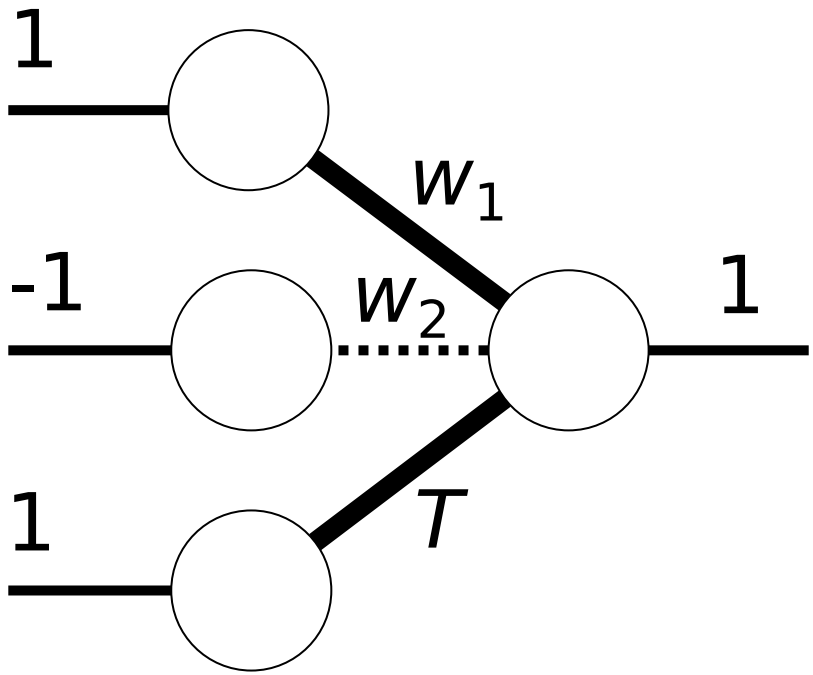
1. Линейная:
$$f(x) = ax + b$$
2. Пороговая:
$$f(x) = \begin{cases} 1, S > 0 \\ 0, S \leq 0 \end{cases}$$
3. Сигмоидная:
$$\frac{1}{1 + e^{-ax}}$$
4. Гиперболический тангенс:
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
5. ReLU-функция:
$$f(x) = max(0, x)$$
6. Softmax-функция:
$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}, j = 1, ..., N.$$

Основные определения

Обучающая выборка (training set) – выборка X_{train}, используемая для корректировки параметров нейронной сети в процессе ее обучения. В случае реализации обучения с учителем дополнительно содержит эталонные значения.
Тестовая (контрольная) выборка (test set) – выборка X_{test}, которая применяется для проверки эффективности обученной нейронной сети. Элементы контрольной выборки не используются в процессе обучения.
Обучение с учителем (learning with a teacher) – процесс подгонки параметров модели (нейронной сети), целью которого является минимизация разницы между выходом модели и эталонным значением для элементов обучающей выборки.
Обучение без учителя (learning without a teacher) – процесс подгонки параметров модели (нейронной сети), выполняемый без эталонных значений (нет зависимых переменных, «руководящих» процессом обучения).
Обобщающая способность – способность сети выдавать корректные данные для примеров, не входящих в обучающую выборку.

Правило обучения Хебба

$$\begin{cases} w_j(t=0) = 0, \forall j \\ w_j(t+1) = w_j(t) + x_j t \end{cases} \tag{3}$$



Алгоритм обучения

Вход: X – данные, G – желаемые отклики сети
Результат: обученный нейрон Neuron
инициализация весов W и порога T
foreach $x_i \in X$ **and** $g_i \in G$ **do**
  foreach $w_j \in W$ **do**
    $w_j(t+1) = w_j(t) + x_{ij} g_i$
  end
  $T(t+1) = T(t) + g_i$
end

Алгоритм 1: Обучение по правилу Хебба

Задача: логическая операция «ИЛИ»

x_1	x_2	**OR**
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

Таблица 1: Исходные данные

Решение

```
import numpy as np
import itertools as it

class HebbNeuron:
  def __init__(self):
    self.w1 = 0
    self.w2 = 0
    self.T = 0

  def test(self, samples):
    for sample in samples:
      weightedSum = self.w1*sample[0] + self.w2*sample[1] + self.T
      if weightedSum > 0:
        y = 1
      else:
        y = -1
    print '('+str(sample[0])+', ' + str(sample[1]) +'): '+ str(y)
```

Решение: продолжение

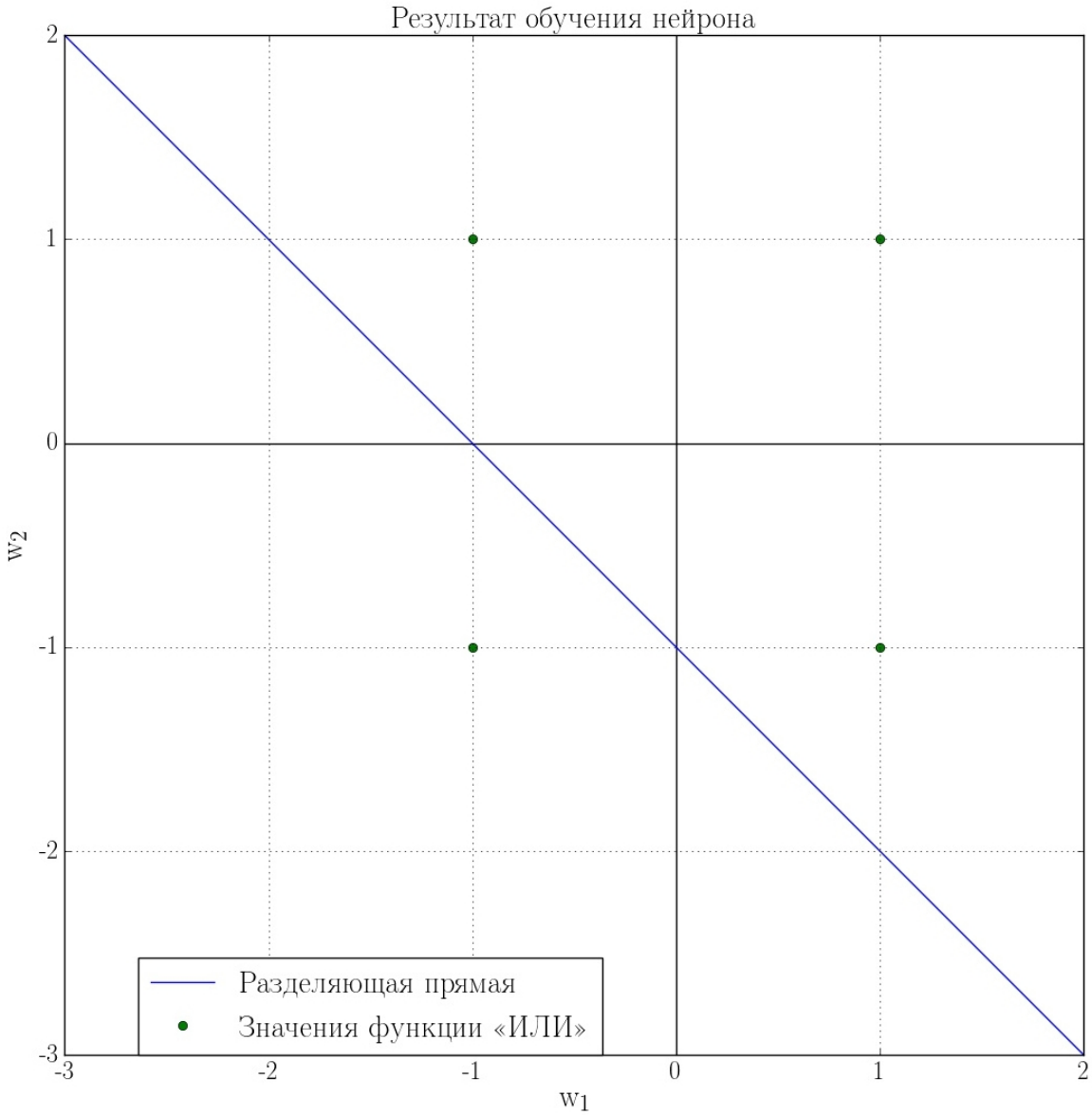
```
  def train(self, samples, targets):
    for sample, target in it.izip(samples, targets):
      self.w1 += sample[0] * target
      self.w2 += sample[1] * target
      self.T += target

if __name__ == "__main__":
  samples = np.array([[-1, -1], [-1, 1], [1, -1], [1, 1]])
  targets = np.array([-1, 1, 1, 1])
  neuron = HebbNeuron()
  neuron.test(samples)
  neuron.train(samples, targets)
  neuron.test(samples)
```

Результат обучения

Epoch	Weights
0	(0, 0, 0)
1	(1, 1, -1)
2	(0, 2, 0)
3	(1, 1, 1)
4	(2, 2, 2)

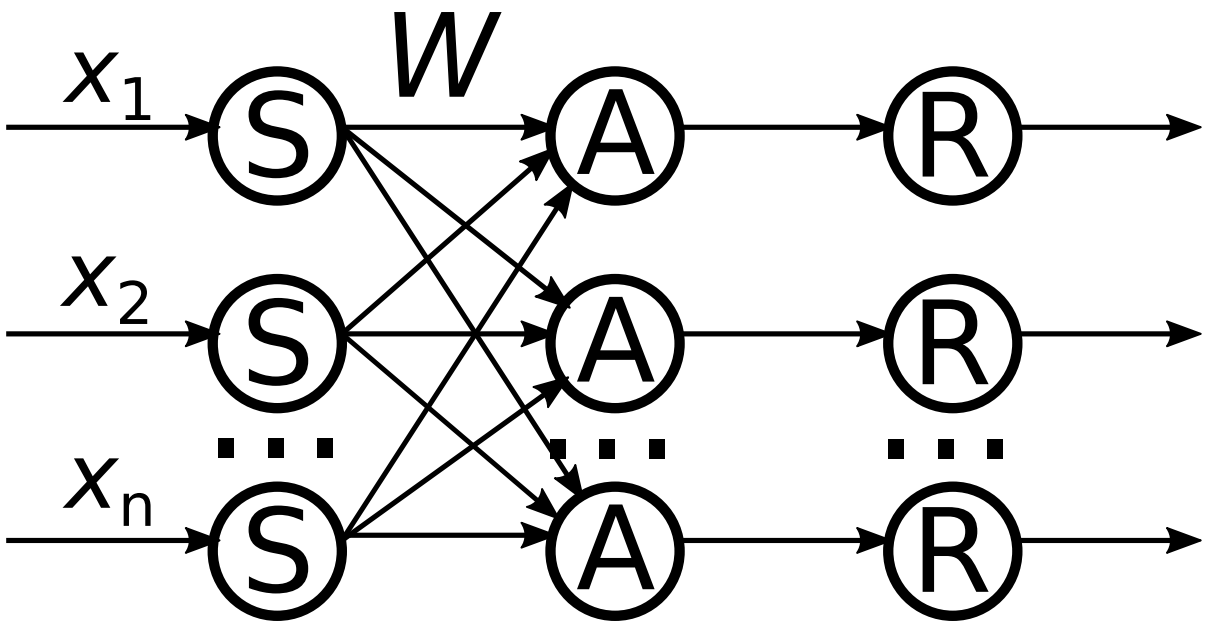
Таблица 2: Эволюция весовых коэффициентов



Правило Хебба: выводы

- Просто программируется
- Не гарантирует сходимости процедуры обучения (при $n >= 5$)
- Может использоваться при построении различного рода нейросетевой памяти

Персептрон Розенблатта



S – сенсорные, A – ассоциативные, R – эффекторные

Один обрабатывающий слой

Процедура обучения Розенблатта

Вход: X – данные, G – желаемый отклик сети
Результат: обученный нейрон Neuron
инициализация весов W и порога T
while $\exists y_i | y_i \neq g_i$ **do**
  foreach $x_i \in X$ **and** $g_i \in G$ **do**
    $y_i = Neuron(x_i)$
    if $y_i \neq g_i$ **then**
      foreach $w_j \in W$ **do**
        $w_j(t+1) = w_j(t) + \alpha x_{ij} g_i$
      end
      $T(t+1) = T(t) + \alpha g_i$
    end
  end
end

Алгоритм 2: Обучение Розенблатта

Задача: логическая операция «И»

x_1	x_2	**AND**
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

Таблица 3: Исходные данные

Решение

```
import numpy as np
import itertools as it
import random

class RosenblattNeuron:
  def __init__(self, rate):
    self.w1 = random.random()
    self.w2 = random.random()
    self.T = random.random()
    self.rate = rate

  def activate(self, sample):
    weightedSum = self.w1*sample[0] + self.w2 * sample[1] + self.T
    y = self.thresActivateFunction(weightedSum)
    return y

  def thresActivateFunction(self, x):
    if x < 0:
      return -1
    else:
      return 1
```

Решение: продолжение

```
def test(self, samples):
  for sample in samples:
    weightedSum = self.w1*sample[0] + self.w2*sample[1] + self.T
    y = self.thresActivateFunction(weightedSum)
  print '('+str(sample[0])+', ' + str(sample[1]) + '): '+str(y)

def train(self, samples, targets):
  isFinish = False
  epochsCount = 0
  while not isFinish:
    isFinish = True
    for sample, target in it.izip(samples, targets):
      y = self.activate(sample)
      if y != target:
        isFinish = False
        self.w1 += self.rate * sample[0] * target
        self.w2 += self.rate * sample[1] * target
        self.T += self.rate * target
    epochsCount += 1
  return epochsCount
```

Решение: продолжение

```
if __name__ == "__main__":
    samples = np.array([[-1, -1], [-1, 1], [1, -1], [1, 1]])
    targets = np.array([-1, -1, -1, 1])
    neuron = RosenblattNeuron(0.1)
    neuron.test(samples)
    epochsCount = neuron.train(samples, targets)
    print 'Epochs count = ' + str(epochsCount)
    neuron.test(samples)
```

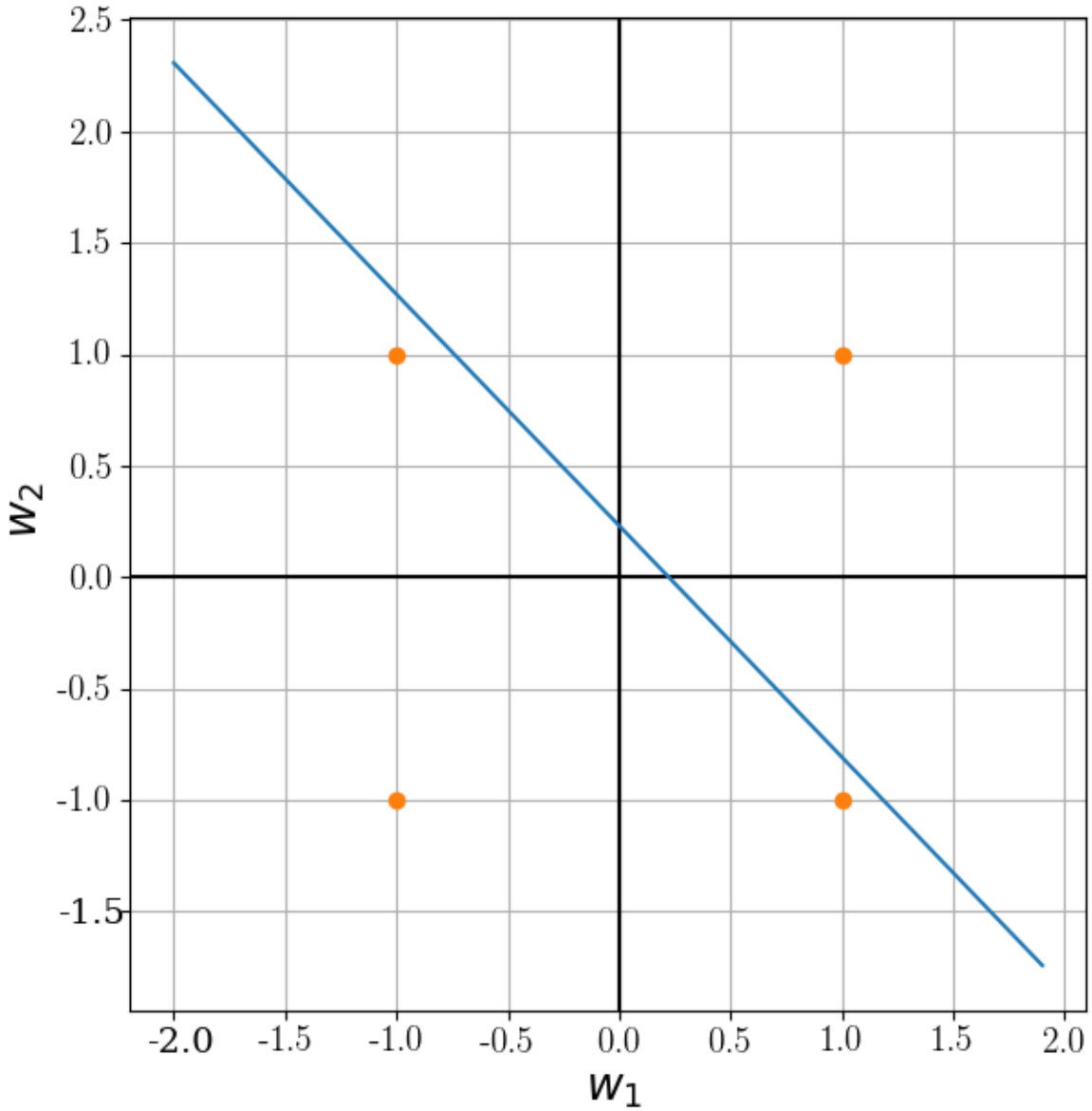
Результат обучения: изменение весовых коэффициентов

Сеть обучилась за 5 эпох

Epoch	Weights
0	(0.0168, 0.8379, 0.4684)
1	(0.1168, 0.7379, 0.3684)
2	(0.2168, 0.6379, 0.2684)
3	(0.3168, 0.5379, 0.1684)
4	(0.3168, 0.5379, -0.0316)
5	(0.4168, 0.4379, -0.1316)

Таблица 4: Эволюция весовых коэффициентов

Результат обучения: график

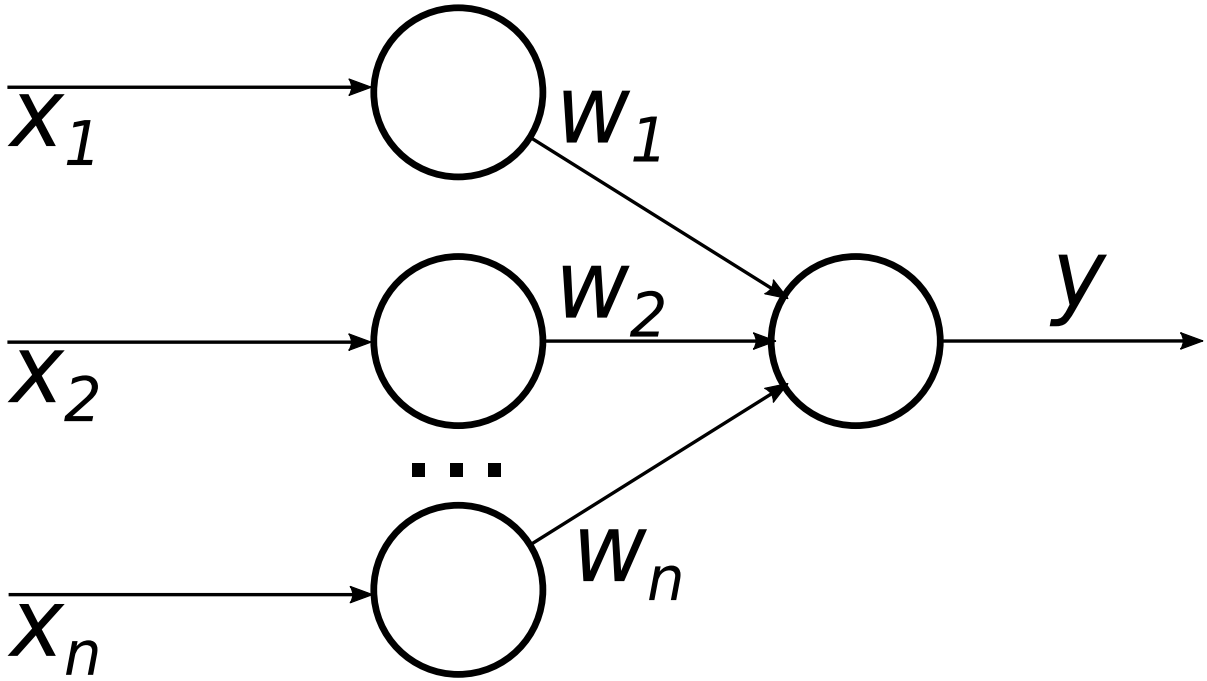


Процедура обучения Розенблатта: выводы

- Присутствует параметр скорости обучения
- Не изменяются весовые коэффициенты, если выход совпадает с эталоном
- Входные образы подаются до тех пор, пока не произойдет обучение
- Если существует решение задачи, сеть обучается за конечное число шагов (теорема о сходимости персептрона)
- Персептрон Розенблатта, формирующий линейную разделяющую поверхность, не способен решить задачу приближения логической функции «исключающее ИЛИ». По этой причине когда-то нейронные сети остановились в своем развитии почти на 10 лет.

Линейный нейрон: сеть типа Adaline

Adaline – Adaptive Linear Element



Правило обучения Видроу-Хоффа (метод наименьших средних квадратов)

$E = \frac{1}{2}\sum_{i=1}^{n}(y_i - g_i)^2$

Вход: X – данные, G – желаемый отклик, E_m – минимальная желаемая LSE-ошибка, α - скорость обучения

Результат: обученный нейрон *Neuron*

инициализация весов W и порога T

while $E > E_m$ **do**

  foreach $x_i \in X$ **and** $g_i \in G$ **do**

    $y_i = Neuron(x_i)$

    foreach $w_j \in W$ **do**

      $w_j(t+1) = w_j(t) - \alpha(y_i - g_i)x_{ij}$

    end

    $T(t+1) = T(t) - \alpha(y_i - g_i)$

  end

  Вычисляется ошибка E для всей выборки X

end

Алгоритм 3: Обучение Видроу-Хоффа

Задача: простейшая регрессия

x_1	x_2	**Target**
0,1	0,2	0,3
0,4	0,5	0,6
0,7	0,8	0,9

Таблица 5: Исходные данные

Решение

```
import numpy as np
import itertools as it
import random

class WidrowHoffNeuron:
  def __init__(self, rate, Em):
    self.w1 = random.random()
    self.w2 = random.random()
    self.T = random.random()
    self.rate = rate
    self.Em = Em

  def activate(self, sample):
    weightedSum = self.w1*sample[0] + self.w2 * sample[1] + self.T
    return weightedSum
```

Решение: продолжение

```
def train(self, samples, targets):
  epochs_count = 0
  isFinish = False
  error_curve = []
  while not isFinish:
    E = 0
    for sample, target in it.izip(samples, targets):
      y = self.activate(sample)
      E += (y - target) * (y - target)
      self.w1 -= self.rate * (y - target) * sample[0]
      self.w2 -= self.rate * (y - target) * sample[1]
      self.T -= self.rate * (y - target)
    epochs_count += 1
    isFinish = E < self.Em
    error_curve.append(E)
  return error_curve, epochs_count
```

Решение: продолжение

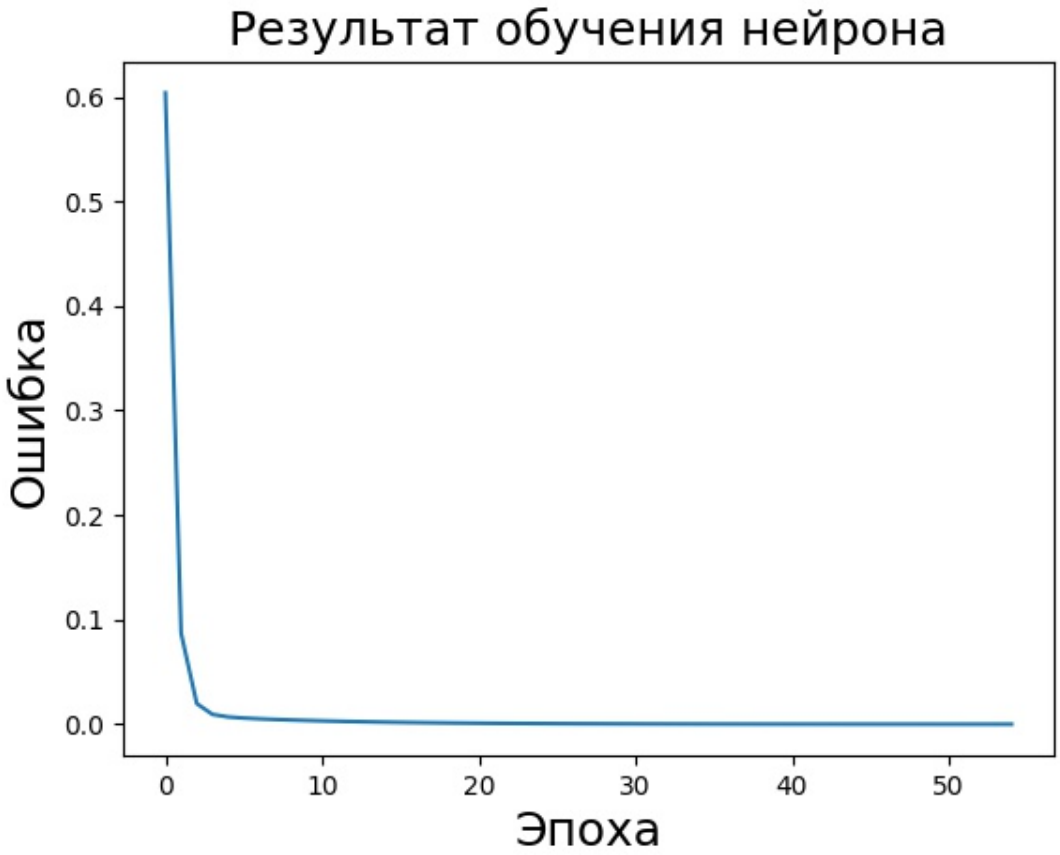
```
if __name__ == "__main__":
    samples = np.array([[0.1, 0.2], [0.4, 0.5], [0.7, 0.8]])
    targets = np.array([0.3, 0.6, 0.9])
    neuron = WidrowHoffNeuron(0.2, 1e-5)
    error_curve, epochs_count = neuron.train(samples, targets)
    test_samples = np.array([[2.3, 2.4], [2.6, 2.7]])
    neuron.test(samples)
    neuron.test(test_samples)
    print 'Epochs count = ' + str(epochs_count)
```

Результат обучения

Сеть достигла желаемой ошибки 1e-5 за 55 эпох обучения

x_1	x_2	**Output**
0,1	0,2	0,303
0,4	0,5	0,601
0,7	0,8	0,899
2,3	2,4	2,486
2,6	2,7	2,783

Таблица 6: Тестирование



Правило обучения Видроу-Хоффа: выводы

1. Больший выбор в представлении выходных данных
2. Может использоваться для решения задач прогнозирования
3. Формулы обучения схожи с используемыми в многослойных сетях при применении метода обратного распространения ошибки. Общая основа - дельта-правило.
4. Используются при построении линейных фильтров (важнейшее приложение – в интерконтинентальных телефонных системах для подавления шума)

Многослойные нейронные сети

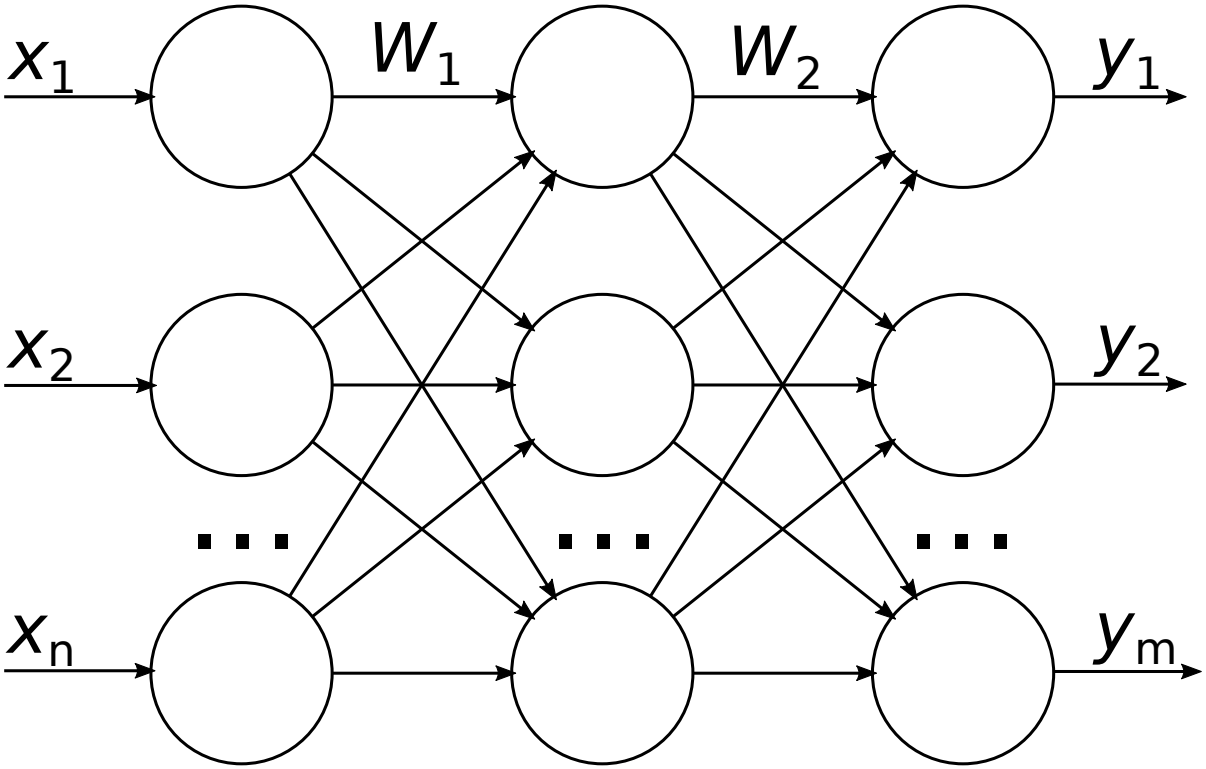


Figure 1: Трехслойная нейронная сеть

Многослойные нейронные сети

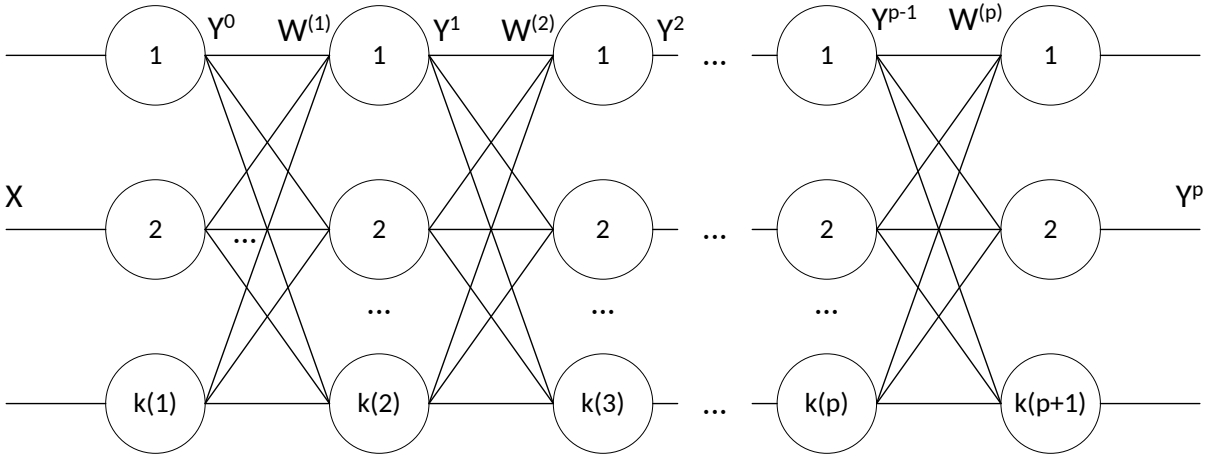


Figure 2: Пример нейронной сети с произвольно большим количеством слоев

Алгоритм обратного распространения ошибки

Вход: X – данные, G – желаемый отклик, E_m – MSE-ошибка, α - скорость
Результат: обученная нейронная сеть *Net*
инициализация весов W и порогов T
while $E > E_m$ **do**
  foreach $x \in X$ **and** $g \in G$ **do**
    Вычисляются активации $y_i, i = 1, ..., LastLayerIndex$
    Вычисляются ошибки:

$$\gamma_j = \begin{cases} y_j - g_j, j = LastLayerIndex \\ \sum_i \gamma_i F'(S_i) w_{ji}, j = 1, ..., LastLayerIndex - 1 \end{cases}$$

    for *(i=0; i <= LastLayerIndex-1; ++i)* **do**
      $w_{i(i+1)}(t+1) = w_{i(i+1)}(t) - \alpha\gamma_{i+1}F'(S_{i+1})y_i,$
      $T_{i+1}(t+1) = T_{i+1}(t) - \alpha\gamma_{i+1}F'(S_{i+1}).$
    end
  end
  Вычисляется ошибка $E = \frac{1}{L}\sum_{k=1}^{L}(y^k - g^k)^2$
end

Задача: исключающее «ИЛИ»

x_1	x_2	**XOR**
0	0	0
0	1	1
1	0	1
1	1	0

Таблица 7: Исходные данные

Решение

```
import numpy as np
import network
import layer
from activate_functions import Logistic
import backpropagation as bpr

def prepareData():
  data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
  labels = np.array([0, 1, 1, 0])
  return data, labels

def test(net, data):
  output = net.activate(data)
  print output
```

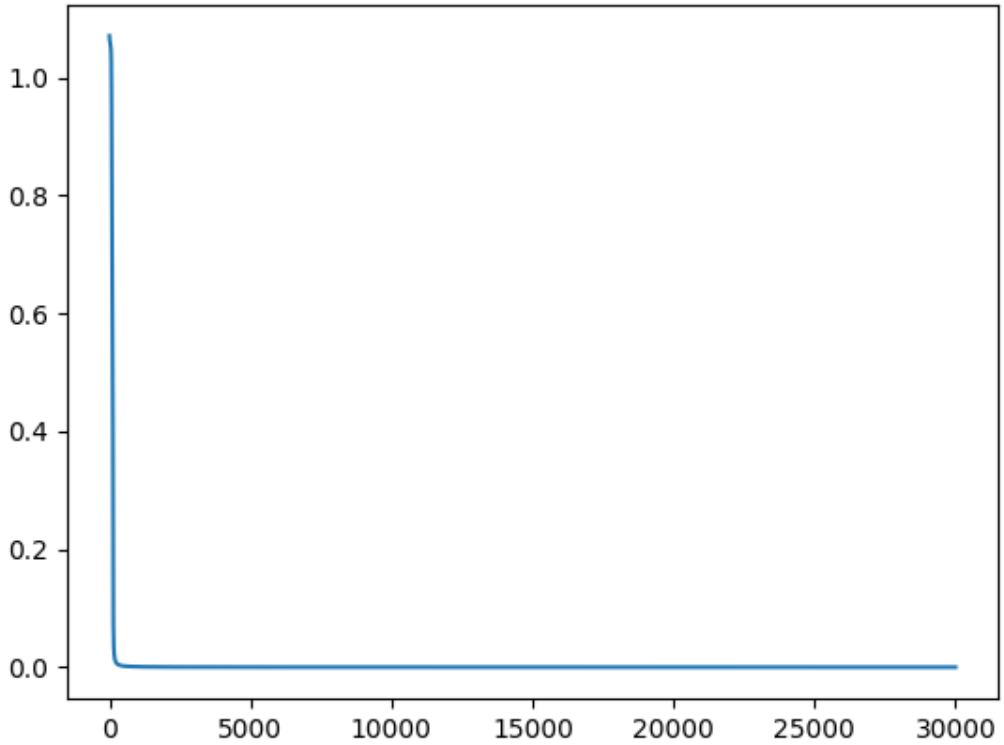
Решение: продолжение

```
net = network.Network()
layer_1 = layer.FullyConnectedLayer(Logistic(), 2, 2)
layer_2 = layer.FullyConnectedLayer(Logistic(), 2, 1)
net.append_layer(layer_1)
net.append_layer(layer_2)
params = bpr.Backprop_params(30000, 1e-5, 1, 0.9, 0, [0.7, 0.7], 0)
method = bpr.Backpropagation(params, net)
data, labels = prepareData()
method.train(data, labels)
test(net, data)
```

Результаты

x_1	x_2	**Output**
0	0	0,002
0	1	0,998
1	0	0,998
1	1	0,003

Таблица 8: Тестирование XOR



Задача: классификация образов из выборки IRIS

Выборка содержит 150 записей, описывающих ирисы трех разных видов. Каждая запись включает 4 значения, представляющих собой характеристики конкретного цветка. Выборка была предложена Фишером в 1936 для демонстрации работы разработанного им линейного дискриминантного анализа.

Scikit-learn: машинное обучение в Python

Scikit-learn позволяет решать множество задач машинного обучения:

- Регрессия
- Классификация
- Предобработка данных
- Кластеризация
- Понижение размерности...
  Кроме этого, содержит встроенные средства для загрузки некоторых выборок, на которых можно проводить собственные исследования.
  Официальный сайт: `http://scikit-learn.org`

Загрузка и подготовка данных

```
import sklearn.datasets as datasets
from sklearn.model_selection import train_test_split
import numpy as np
RANDOM_SEED=42

def prepareData():
  irises_dataset = datasets.load_iris()
  data = irises_dataset['data']
  labels = irises_dataset['target']
  return train_test_split(data, labels, test_size=0.33,
      random_state=RANDOM_SEED)
```

Результаты

Обучающая выборка – 97% правильно распознанных изображений, тестовая - 100 % правильно распознанных изображений.

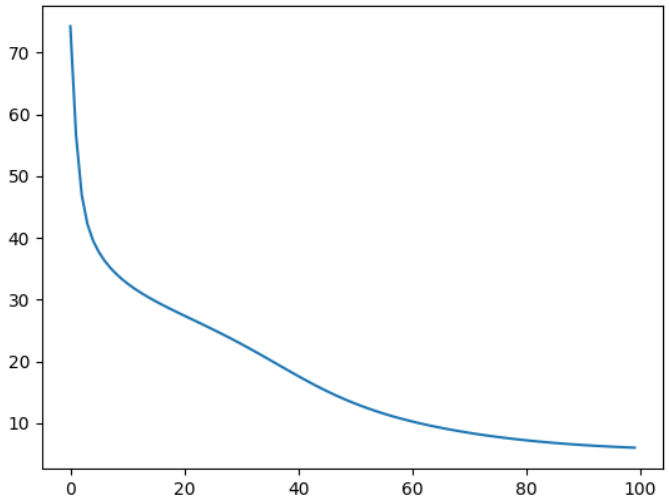


Figure 3: Кривая ошибок

Условия проведения эксперимента: сеть 4-256-3 с сигмоидными функциями активации, скорость обучения – 0.01, моментный параметр – 0.9, онлайн-обучение.

Отступление: метод PCA

Метод PCA (principal component analysis) предназначен для понижения размерности исходных данных. Полезен при визуализации многомерных данных, а также при выполнении выделения признаков (features extraction) для последующего обучения с использованием нейронной сети (`http://www.visiondummy.com/2014/05/feature-extraction-using-pca/`).
Алгоритм PCA будет иметь вид:
Вход: X – данные
Результат: Данные $\tilde{X}$ с уменьшенной размерностью
1. Вычисляется ковариационная матрица $cov_X = cov(X)$
2. Находятся собственные векторы и значения cov_X
3. Выбирается вектор(ы), соответствующий максимальному собственному значению(ям)
4. Матрицу, составленную из этих векторов, используем для понижения размерности

Реализация алгоритма PCA

```
import numpy as np

def prepareData():
  irises_dataset = datasets.load_iris()
  return irises_dataset['data'], irises_dataset['target']

def pca_method(data):
  cov_matrix = np.cov(data.T)
  V, PC = np.linalg.eig(cov_matrix)
  sort_index = np.argsort(-1 * V)
  PC = PC[:, sort_index]
  data = np.dot((PC.T)[0:2], data.T)
  return data.T

irises_data, irises_target = prepareData()
irises_reduction = pca_method(irises_data)
```

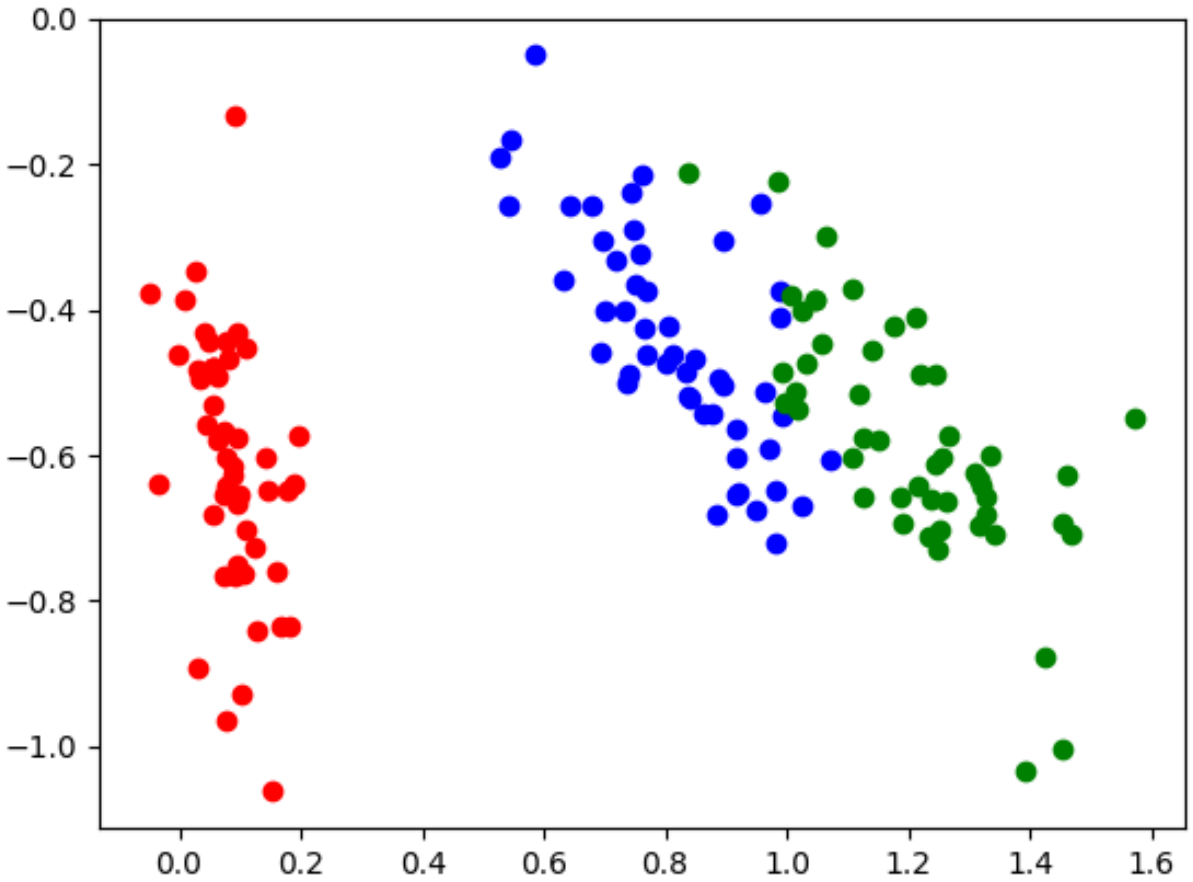
Тоже самое, но с использованием scikit-learn

```
from sklearn.decomposition import PCA

def prepareData():
  irises_dataset = datasets.load_iris()
  return irises_dataset['data'], irises_dataset['target']

irises_data, irises_target = prepareData()
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(irises_data)
```

Визуализация выборки (PCA) и объяснение полученных результатов



Подсчет потерь в информативности

```
full_info = V.sum()
V = V[sort_index][0:2]
reduce_info = V.sum()
print (100 - reduce_info / full_info * 100)
```

Потери после применения PCA составят около 4%.

Основные проблемы обратного распространения и пути их решения

- Медленная сходимость градиентного метода с постоянным шагом обучения
- Проблема выбора подходящей скорости обучения
- Градиентный метод не различает точек локального и глобального минимума
- Влияние случайной инициализации на процесс поиска решения
- Сложность программной реализации

Глубокие нейронные сети

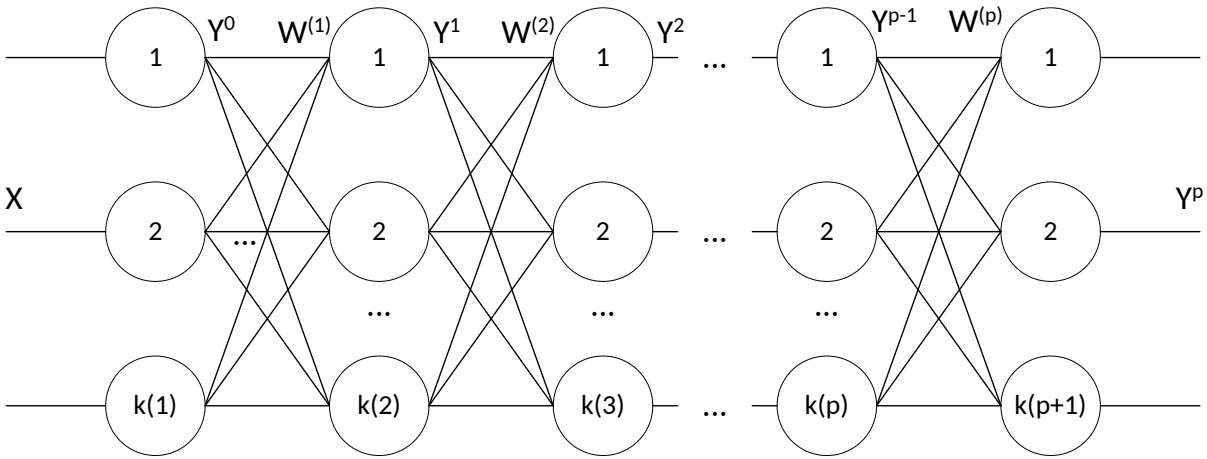


Figure 4: Пример нейронной сети с произвольно большим количеством слоев

Глубокой можно считать сеть с более чем 4 обрабатывающими слоями.

Почему глубокие нейронные сети работают?

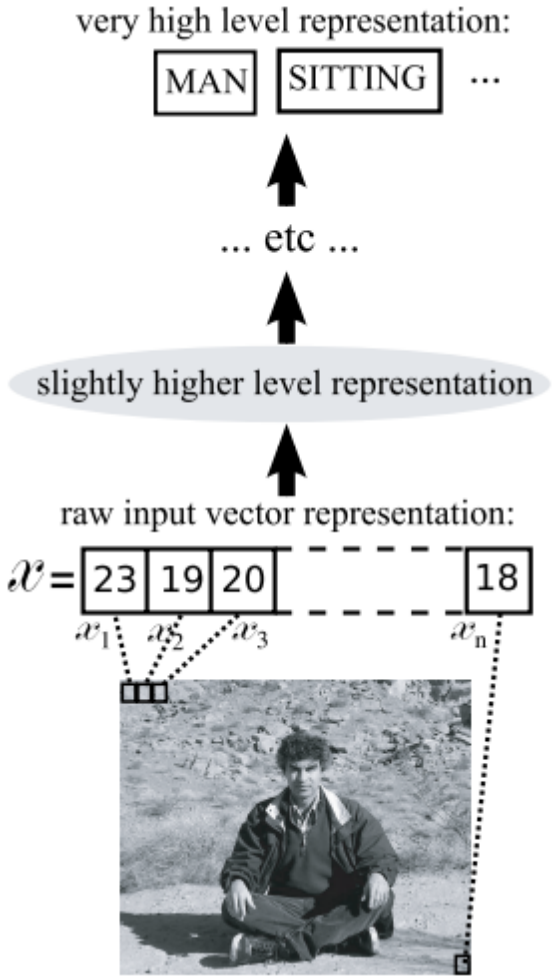
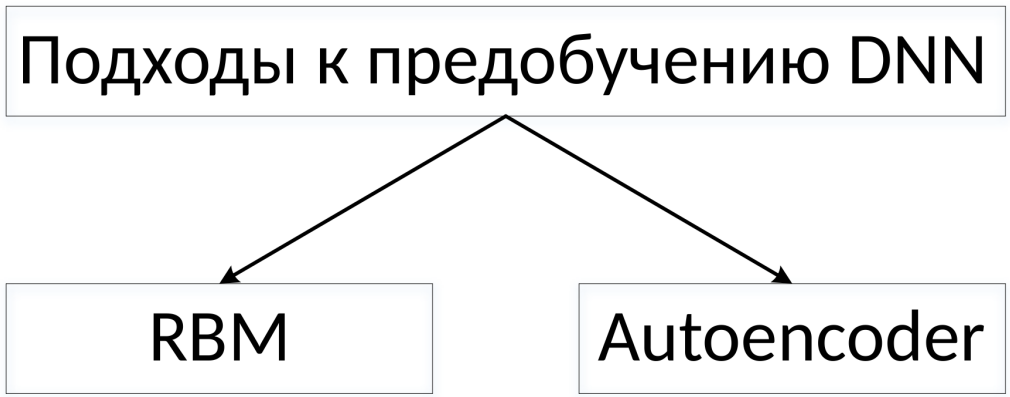


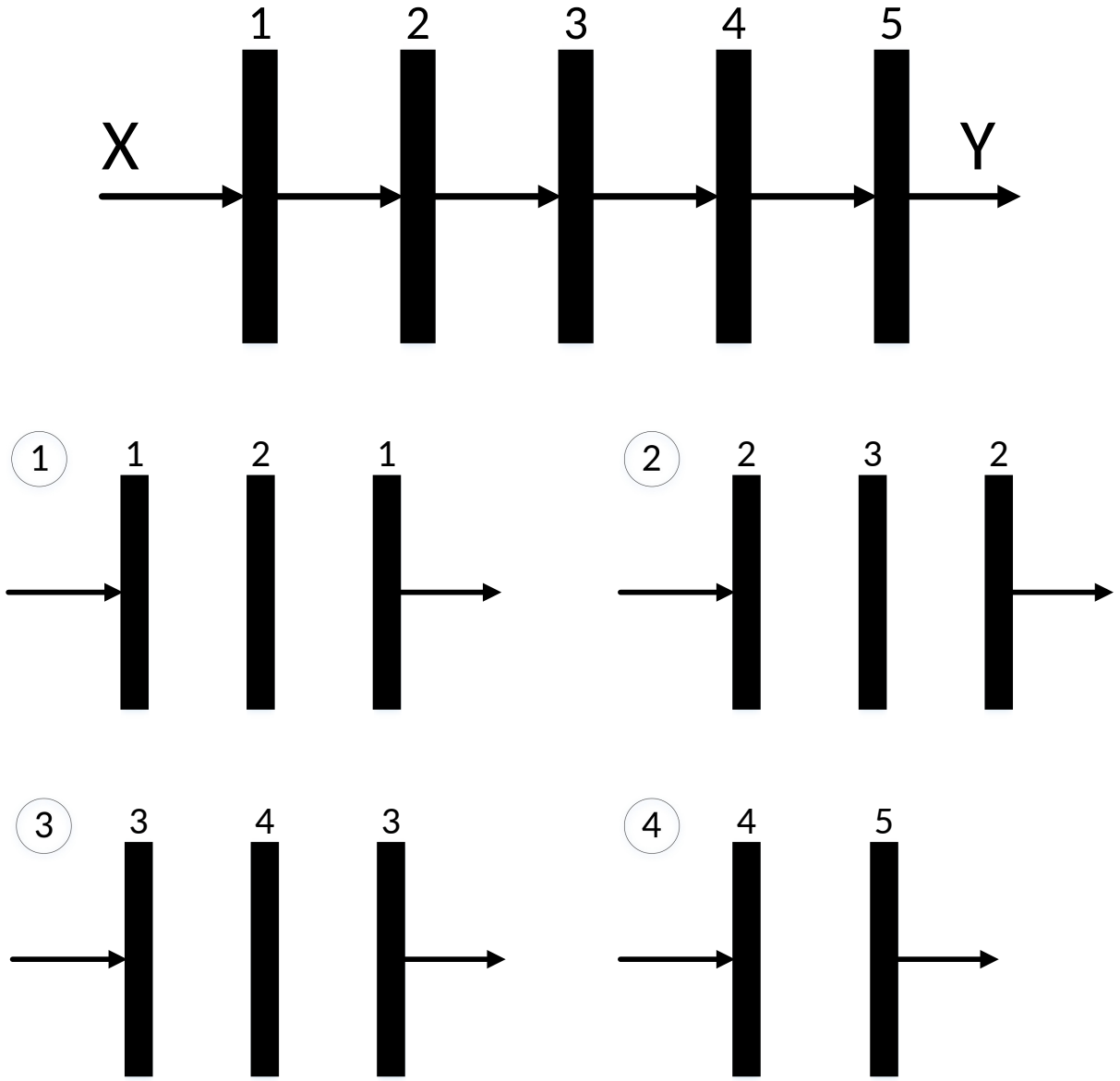
Figure 5: Иерархия признаков. Фото взято из статьи Y. Bengio. Learning Deep Architectures for AI

Основные методы, применяемые для обучения ГНС

- Метод обратного распространения ошибки с функцией активации ReLU (большая обучающая выборка)
- Предобучение НС (при малых выборках, позволяет преодолеть переобучение)



Автоэнкодерный подход

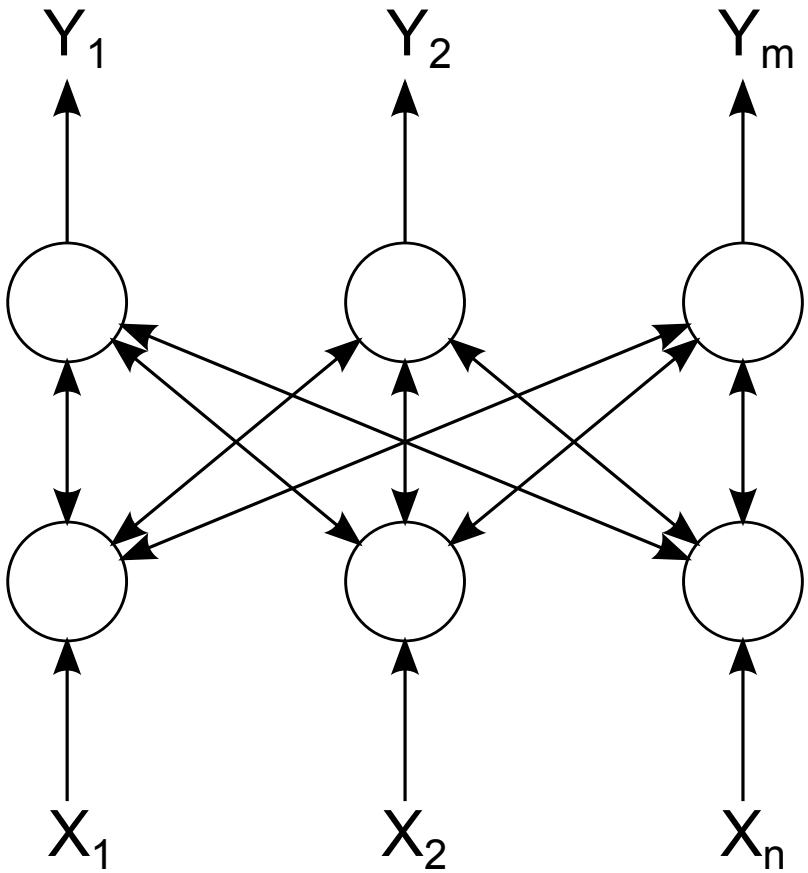


Автоэнкодерный подход

Данный процесс можно представить в виде следующего алгоритма:

1. Конструируется автоассоциативная сеть с входным слоем X, скрытым Y и выходным слоем X.
2. Обучается автоассоциативная сеть, например при помощи алгоритма обратного распространения ошибки (как правило не более 100 эпох) и фиксируются синаптические связи первого слоя W_1 .
3. Берется следующий слой и формируется автоассоциативная сеть аналогичным образом.
4. Используя настроенные синаптические связи предыдущего слоя W_1, подаем входные данные на вторую автоассоциативную сеть и обучаем ее аналогичным образом. В результате получаются весовые коэффициенты второго слоя W_2.
5. Процесс продолжается до последнего слоя нейронной сети.
6. Берется последний слой нейронной сети и обучается с учителем.
7. Обучается вся сеть для точной настройки параметров при помощи алгоритма обратного распространения ошибки.

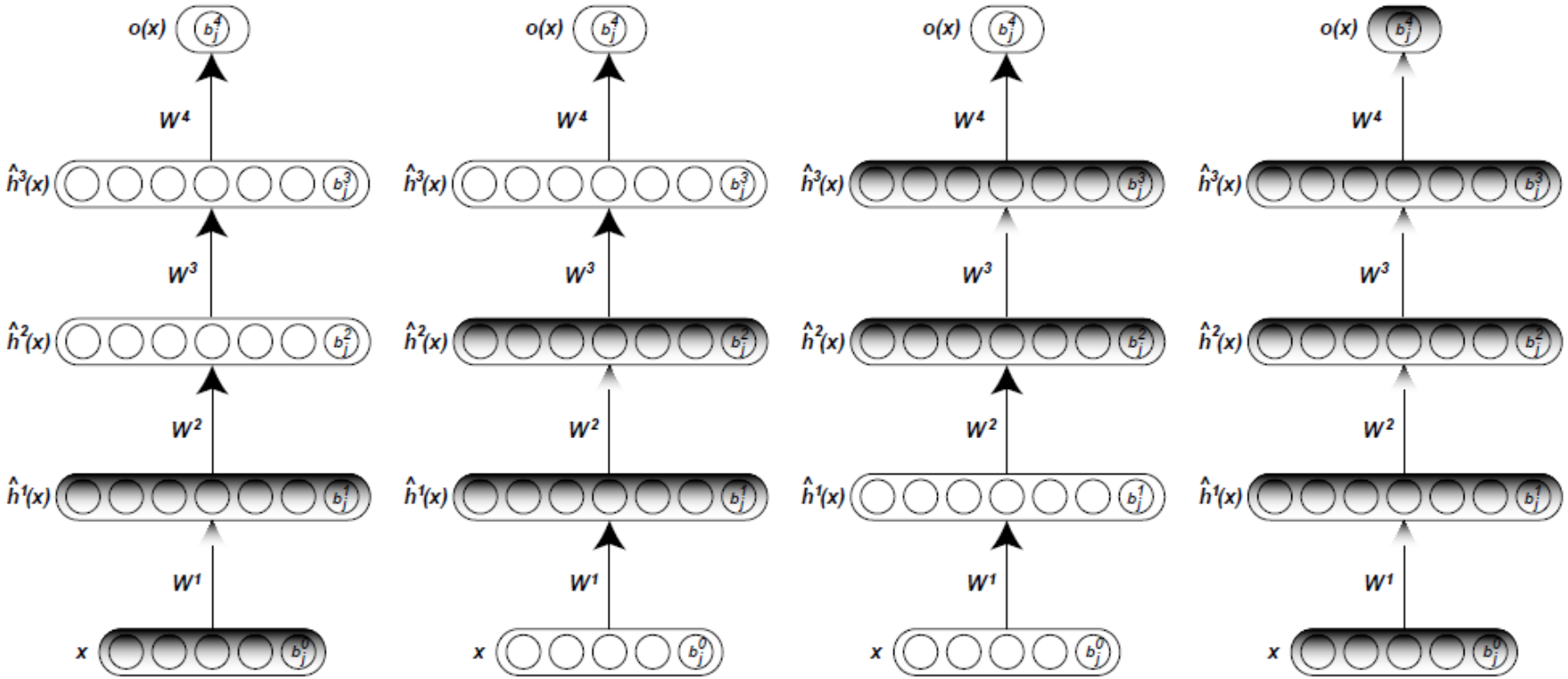
Подход на основе RBM



Вход: $x_i(0)$ – образ из обучающей выборки
α - скорость обучения
Результат: матрица весовых коэффициентов W, вектор порогов видимых элементов b, вектор порогов скрытых нейронов c

foreach *скрытого нейрона* j **do**
  Вычислить $P(y_j(0) = 1|x_i(0))$ (для биномиальных нейронов $\text{sigm}(\sum_i w_{ij}x_i(0) + T_j)$)
  Генерировать $y_j(0) \in \{0, 1\}$ из $P(y_j(0)|x_i(0))$
end
foreach *видимого нейрона* i **do**
  Вычислить $P(x_i(1) = 1|y_j(0))$ (для биномиальных нейронов $\text{sigm}(\sum_j w_{ij}y_j(0) + T_i)$)
  Генерировать $x_i(1) \in \{0, 1\}$ из $P(x_i(1)|y_j(0)$
end
foreach *скрытых нейронов* j **do**
  Вычислить $P(y_j(1) = 1|x_i(1))$ (для биномиальных нейронов $\text{sigm}(\sum_i w_{ij}x_i(1) + T_j)$)
end
$W \leftarrow W + \alpha(x(0)y(0)' - x(1)P(y(1) = 1|x(1))')$
$T_i \leftarrow T_i + \alpha(x(0) - x(1))$
$T_j \leftarrow T_j + \alpha(y(0) - P(y(1) = 1|x(1)))$

Алгоритм предобучения на основе RBM



(a) First hidden layer pre-training

(b) Second hidden layer pre-training

(c) Third hidden layer pre-training

(d) Fine-tuning of whole network

Взято из Hinton G. Greedy layer-wise algorithm (Journal of Machine Learning Research), 2009

Задачи классификации и регрессии

Оценочной функцией называется отображение $\tilde{f} : X \to \mathbb{R}$.
Проблема обучения регрессии заключается в построении оценочной функции по примерам $(x_i, f(x_i))$, где $f(x)$ – неизвестная функция.
Задача классификации состоит в построении классификатора, т.е. отображения $\tilde{c} : X \to C$, где $C = \{C_1, C_2, ...C_k\}$ - конечное и обычно небольшое множество меток классов.
Под обучением классификатора будем понимать построение функции $\tilde{c}$, которая как можно лучше аппроксимирует $c(x)$ –неизвестную функцию.

Оценка качества классификатора

Существует несколько подходов к оценке качества классификатора

- По итоговому значению ошибки (менее презентабельная оценка)
- По обобщающей способности в процентах (более показательный)
- ROC-анализ (более надежный, чем первые два)

ROC-анализ

Предназначен для объективной оценки бинарного классификатора. В принципе может использоваться и для многомерного классификатора, но это требует применения специальных предположений (например, «один против всех»).
Предполагает вычисление специальных показателей (**точность**, **специфичность**, **полнота**, **f1-мера**) и построение т.н. ROC-кривой, площадь под которой служит для сравнительной характеристики классификатора.
Для расчета показателей ROC-анализа нужно составить следующую таблицу:

		Действительные значения	
		1	0
Классификатор	1	TP	FP
	0	FN	TN

Таблица 9: Вспомогательная таблица

TP, FP, FN, TN задают соответственно количество истинноположительных, ложноположительных, ложноотрицательных и истинноположительных значений из общего числа элементов исследуемого множества.

ROC-анализ: продолжение

На основании полученной таблицы могут быть вычислены показатели:

- Полнота (чувствительность – sensitivity): $TP/(TP + FN)$
- Специфичность (specificity): $TN/(TN + FP)$
- Точность (precision): $TP/(TP + FP)$
- F1-мера:

$$F = 2\frac{Precision * Sensitivity}{Precision + Sensitivity}$$

ROC-анализ: продолжение

Далее может быть построен следующий график, показывающий соотношение истинно-положительных и ложно-положительных ответов в зависимости от заданного порога t. Это кривая называется **ROC-кривой**.
Показатель AUC (Area Under Curve – площадь под ROC-кривой) определяет эффективность работы бинарного классификатора и может использоваться для сравнительной оценки.

Оценка работы БК на примере выборки Tic-Tac-Toe

Данная выборка составлена из возможных вариантов игры «Крестики-Нолики» и может использоваться для обучения классификатора определению одного из двух возможных исходов.
Эта выборка взята с сайта:
`https://archive.ics.uci.edu/ml/datasets.html`

Загрузка данных и подготовка модели

```
def loadDataFromFile(path):
  f = open(path, "rb")
  data = []
  labels = []
  for str in f:
    substr = str.split(",")
    tmp = []
    for i in range(0, 9):
      if substr[i] == "o":
        tmp.append(1)
      if substr[i] == "x":
        tmp.append(-1)
      if substr[i] == "b":
        tmp.append(0)
      data.append(tmp)
    if substr[9][:len(substr[9])-1] == "negative":
      labels.append(0)
    else:
      labels.append(1)
  data = np.array(data)
  labels = np.array(labels)
  return train_test_split(data, labels, test_size=0.33)
```

Вычисление ROC-показателей

```
def calcROC(net, data, labels):
  output = net.activate(data)
  answer = output > 0.5
  answer = answer.reshape(len(answer))
  TP = TN = FP = FN = 0
  for i in range(0, len(answer)):
    if answer[i] == labels[i] == 1:
      TP += 1
    if answer[i] == labels[i] == 0:
      TN += 1
    if answer[i] == 1 and labels[i] == 0:
      FP += 1
    if answer[i] == 0 and labels[i] == 1:
      FN += 1
  Precision = float(TP) / (TP + FP)
  Sensitivity = float(TP) / (TP + FN)
  print 'Sensitivity = ' + str(float(TP) / (TP + FN))
  print 'Specificity = ' + str(float(TN) / (TN + FP))
  print 'Precision = ' + str(float(TP) / (TP + FP))
  print 'F-score = ' + str(2 * (Precision * Sensitivity)/(Precision
      + Sensitivity))
```

Построение ROC-кривой и нахождение AUC

```
def drawROCCurve(net, data, labels):
  output = net.activate(data)
  answer = output.reshape(len(output))
  P = (labels == 1).sum()
  N = (labels == 0).sum()
  t = 0
  tmax = 1
  dx = 0.0001
  points = []
  while t <= tmax:
    FP = TP = 0
    for i in range(0, len(answer)):
      if answer[i] >= t:
        if labels[i] == 1:
          TP += 1
        else:
          FP += 1
    SE = TP / float(P)
    m_Sp = FP / float(N)
    points.append([m_Sp, SE])
    t += dx
  print points
  points.reverse()
```

Построение ROC-кривой и нахождение AUC

```
points = np.array(points)
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(points[:, 0], points[:, 1], lw=2, label='ROC curve')
plt.plot([0.0, 1.0], [0.0, 1.0], lw=2)
plt.show()
auc = 0
for i in xrange(1, len(points)):
  auc += (points[i, 0] - points[i - 1, 0]) * points[i, 1]
print 'auc = ' + str(auc)
```

Основная программа: конфигурация сети

```
#load data from file
data = loadDataFromFile("Datasets/tic-tac-toe.data.txt")
#network configure
net = Network()
layer_1 = FullyConnectedLayer(Logistic(), 9, 9)
layer_3 = FullyConnectedLayer(Logistic(), 9, 1)
net.append_layer(layer_1)
net.append_layer(layer_3)
params = Backprop_params(500, 1e-5, 10, 0.9, False, [0.01, 0.01],
    0)
method = Backpropagation(params, net)
train_data = data[0]
test_data = data[1]
train_labels = data[2]
test_labels = data[3]

#learning
error_curve = method.train(train_data, train_labels)
plot(error_curve)
#output results
print "Train efficiency: " + str(testing(net, train_data,
    train_labels))
```

Основная программа: конфигурация сети

```
print "Test efficiency: " +  str(testing(net, test_data,
    test_labels))

#calc ROC-characteristics
calcROC(net, test_data, test_labels)
drawROCCurve(net, test_data, test_labels)
```

Результаты

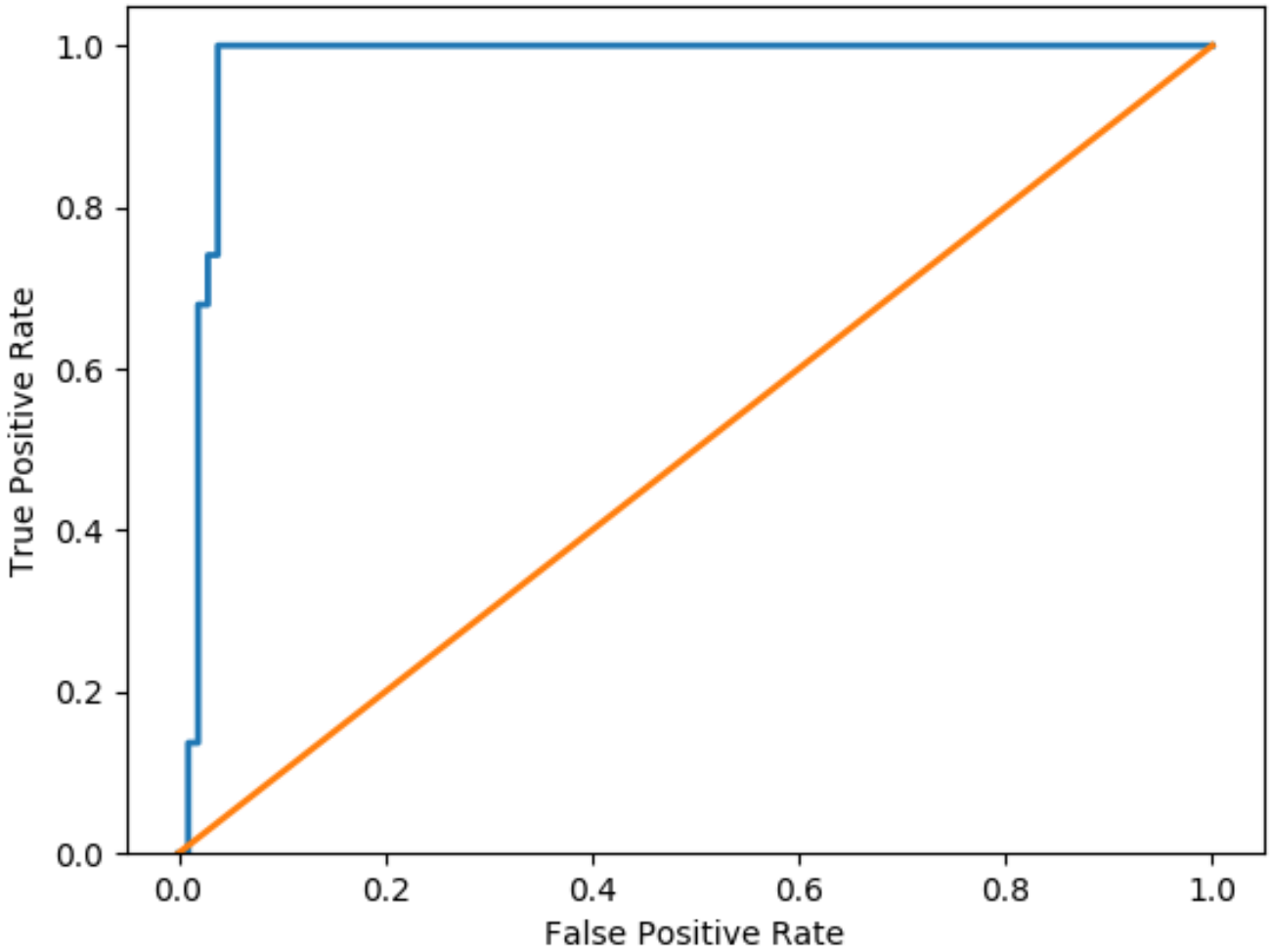
		Действительные значения	
		1	0
Классификатор	1	212	4
	0	0	101

Таблица 10: Вспомогательная таблица

T.o. *Sensitivity* $= 1.0$, *Specificity* $= 0.96$, *Precision* $= 0.981$, F-score $= 0.99$.

ROC-кривая

auc = 0.976729559748



Домашнее задание

Взять какую-нибудь из выборок для тестирования алгоритмов машинного обучения (регрессионная или классификационная задача), обучить сеть с наиболее подходящей на Ваш взгляд архитектурой (используя либо предложенный код, либо возможности соответствующих фреймворков) и продемонстрировать результаты в следующий раз. Объяснить их.
Поиск подходящей выборки рекомендую начать отсюда:
`https://archive.ics.uci.edu/ml/datasets.html`