# Искусственные нейронные сети: основы практического применения

## Лекция 2

Крощенко А.А.

Брестский государственный технический университет

24.05.2017

# 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.

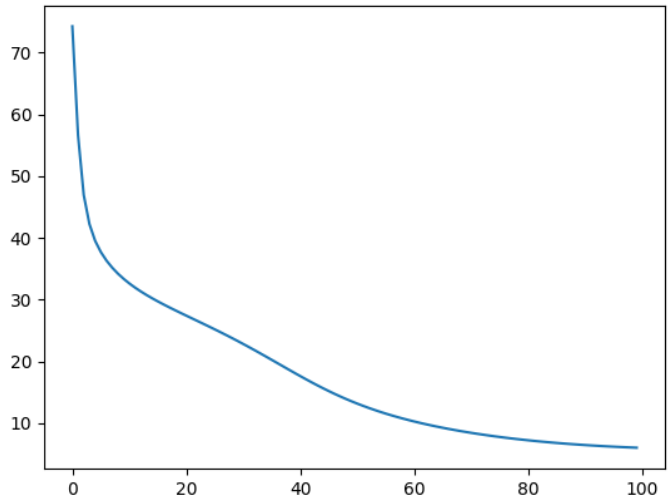



Figure 1: Кривая ошибок

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)
```

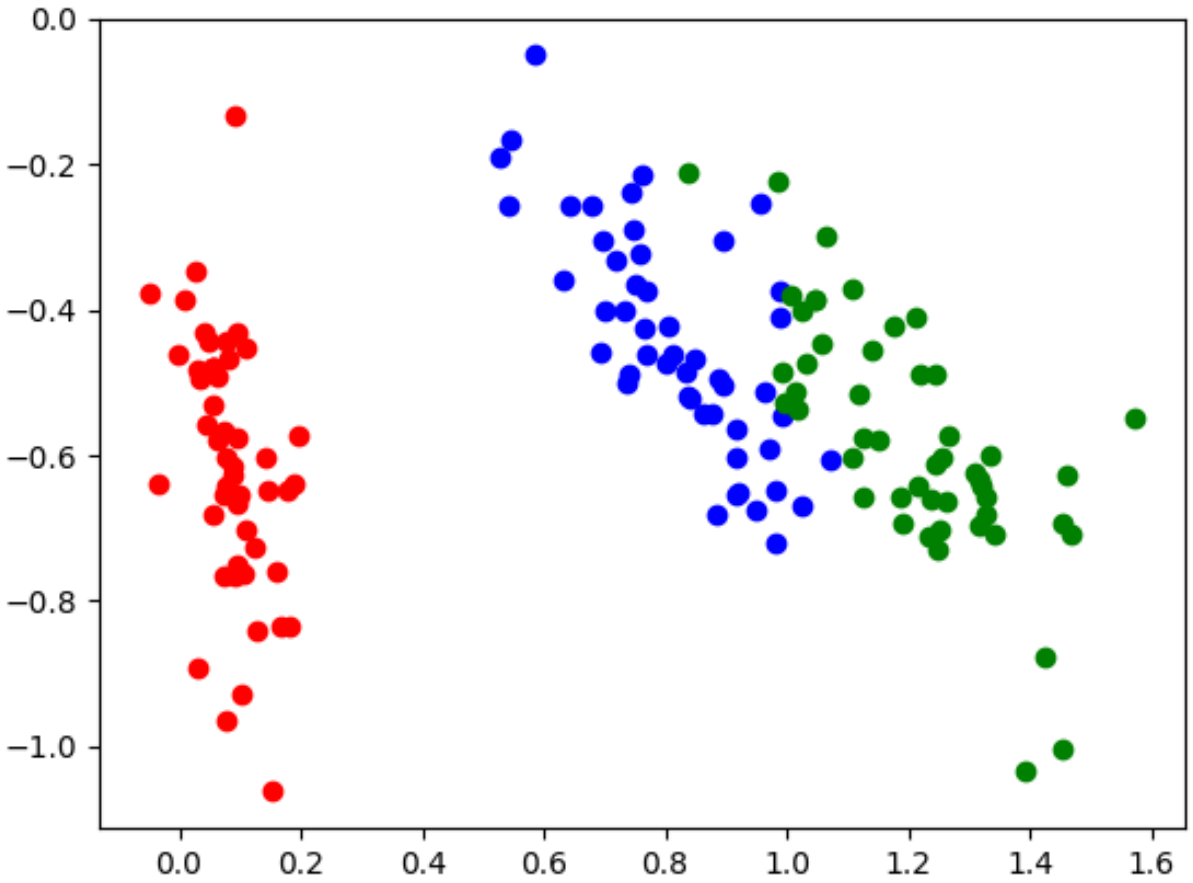
# Тоже самое, но с использованием scikit-learn

```
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

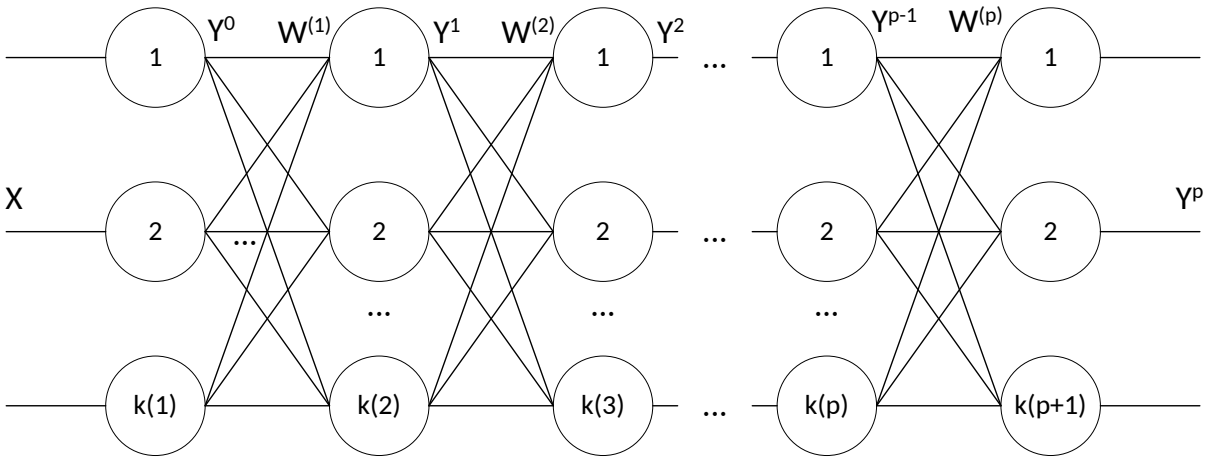



Figure 2: Пример нейронной сети с произвольно большим количеством слоев

Глубокой можно считать сеть с более чем 4 обрабатывающими слоями.

# Почему глубокие нейронные сети работают?

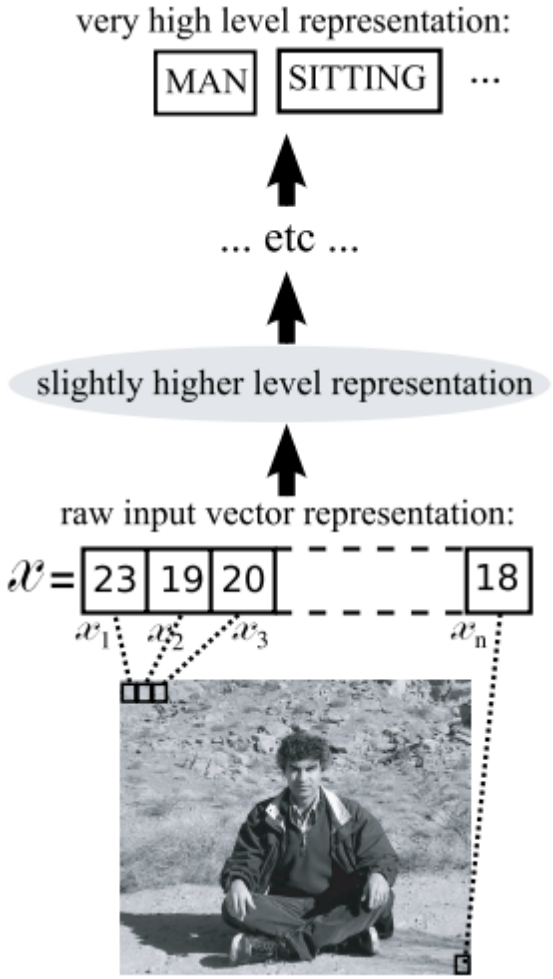
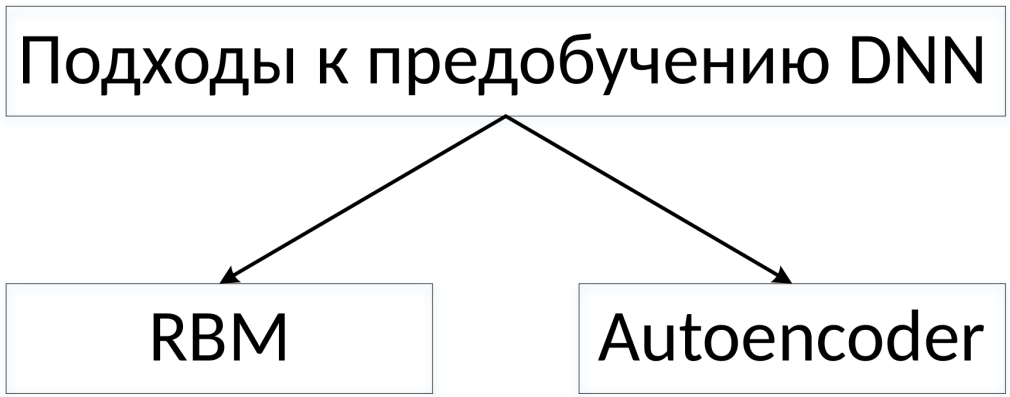


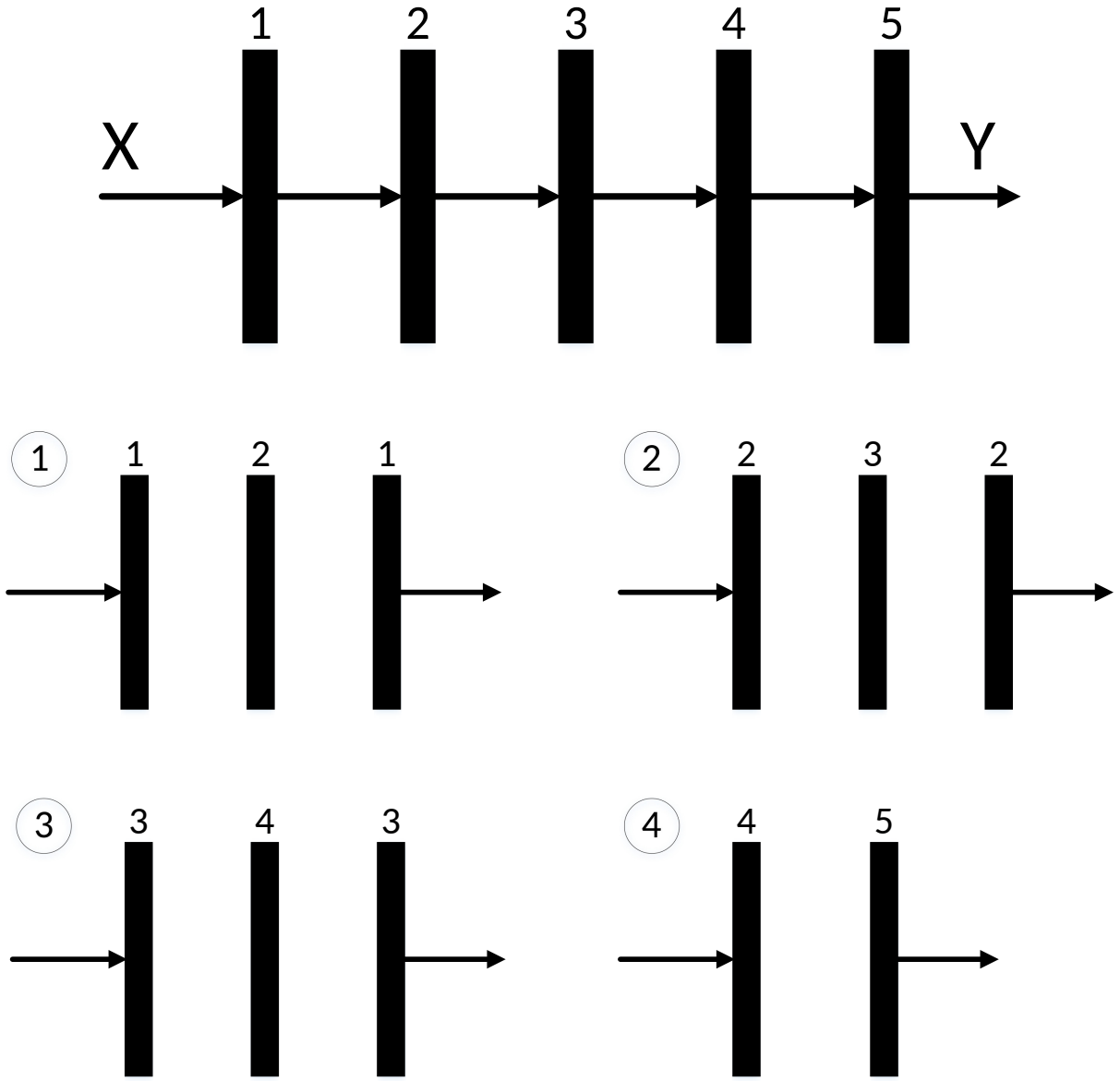

Figure 3: Иерархия признаков. Фото взято из статьи Y. Bengio. Learning Deep Architectures for AI

# 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

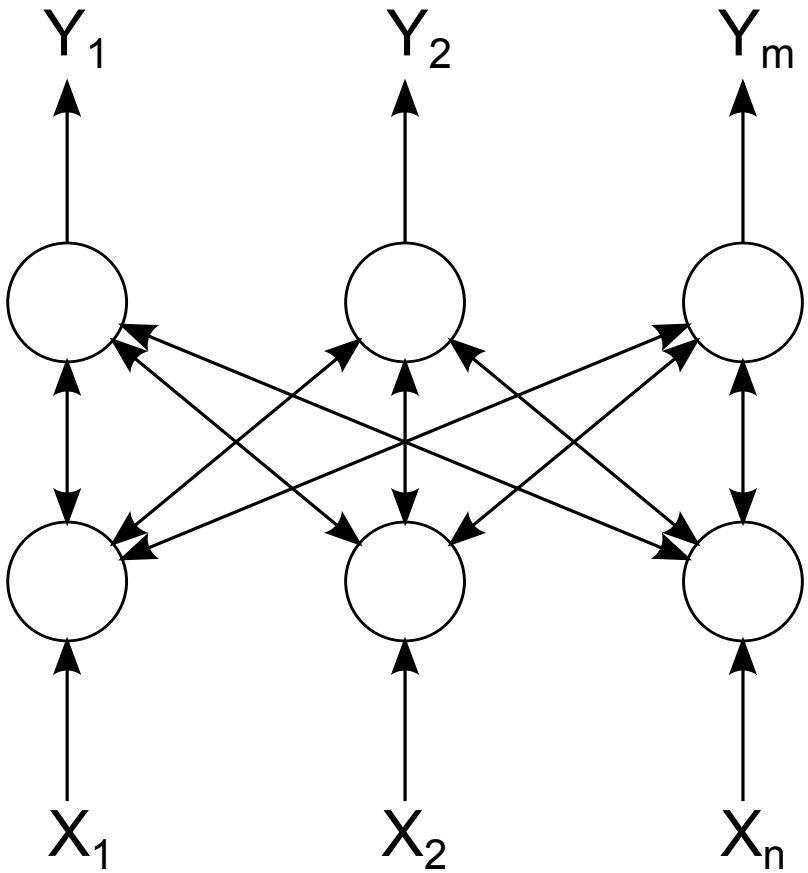


# Автоэнкодерный подход

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.

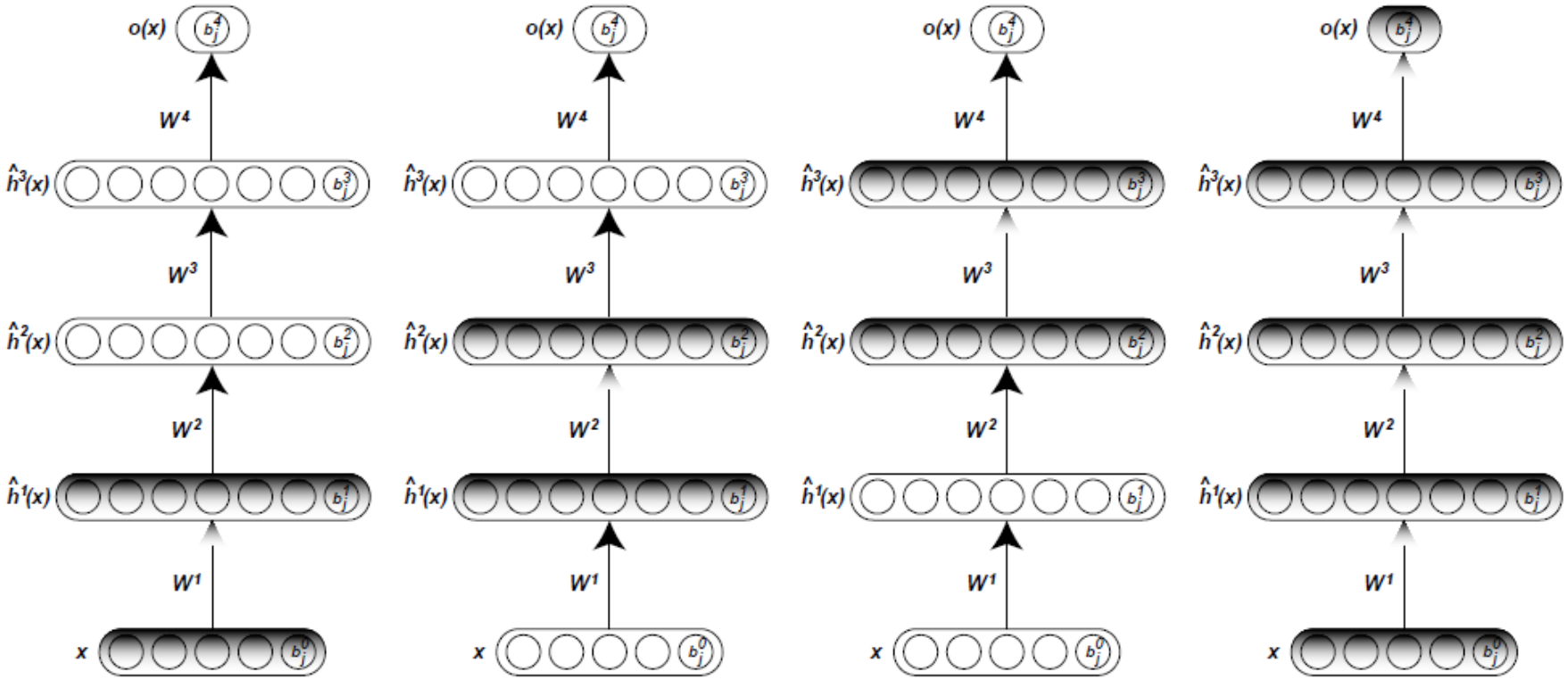
# Подход на основе RBM



**Вход:** $x_i(0)$ – образ из обучающей выборки
$\alpha$ - 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 1: 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)
```

# Основная программа: конфигурация сети

```
#output results
print "Train efficiency: " + str(testing(net, train_data,
    train_labels))
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

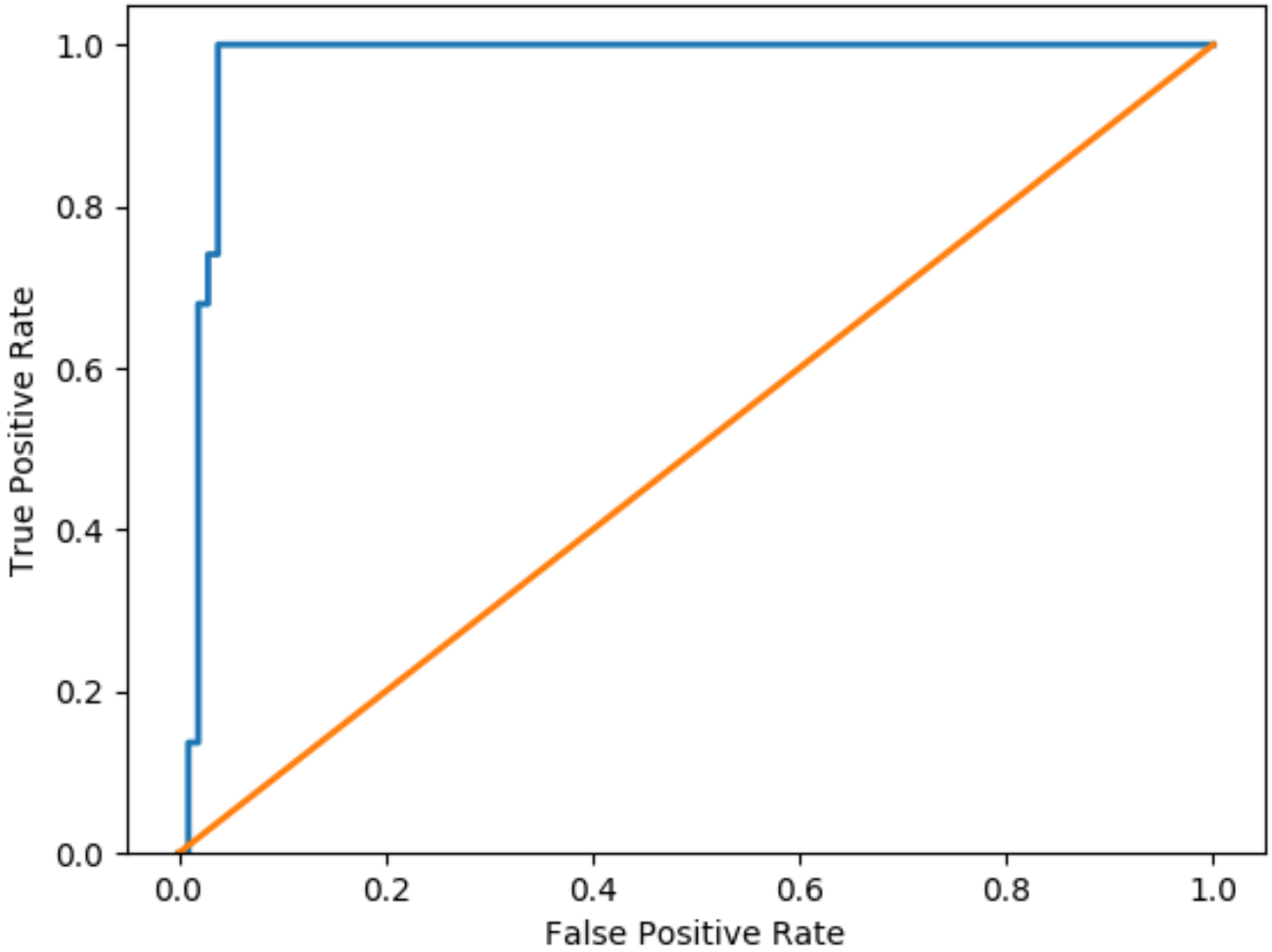
| | | Действительные значения | |
|---|---|---|---|
| | | 1 | 0 |
| Классификатор | 1 | 212 | 4 |
| | 0 | 0 | 101 |

Таблица 2: 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`