УДК 519.68:007.5

# Лабораторная работа 1. Персептрон Румельхарта

**Цели работы**: реализация персептрона Румельхарта и радиально базисной сети. Анализ возможностей и ограничений метода обратного распространения ошибки.

**Ключевые слова:** метод обратного распространения ошибки, переобучение сети, паралич коэффициентов.

## § 1. Обзор теории

### 1.1. Описание базовых моделей.

Определение 1. *Персептрон Румельхарта* — это многослойная нейронная сеть, состоящая из однотипных слоев $R_1, ..., R_n$, для которых используются:

- сигмоида в качестве функции активации $\sigma(x) = 1/(1 + e^{-x})$.
- для всех нейронов $r_{ij} \in R_j$ значения $y^{r_{ij}}(t) = \sigma\left(\sum \omega_k^{r_{ij}} \cdot x_k^{r_{ij}}(t) - \theta^{r_{ij}}\right)$.
- сквозное подключение слоев $x_k^{r_{ij}}(t) = y^{r_{kj-1}}(t)$ для всех $j > 1$.
- общий параметр $\nu < 1$, задающий скорость обучения слоев.
- для выходного слоя $R_n$ оценка ошибки ($e_i$ — эталоны для выходов):

$$\delta^{r_{in}}(t) = y^{r_{in}}(t)\,(1 - y^{r_{in}}(t))\,(y^{r_{in}}(t) - e_i(t))$$

- для внутренних слоев $R_j$ $(j \neq n)$ суммарная оценка ошибки:

$$\delta^{r_{ij}}(t) = y^{r_{ij}}(t)\,(1 - y^{r_{ij}}(t)) \sum_p \delta^{r_{pj+1}}(t)\,\omega_i^{r_{pj+1}}(t)$$

- общие формулы для коррекции весов и порогов для всех слоев:

$$\omega_k^{r_{ij}}(t+1) = \omega_k^{r_{ij}}(t) - \nu\,\nabla J_{\omega_k^{r_{ij}}}(t); \quad \nabla J_{\omega_k^{r_{ij}}}(t) = \delta^{r_{ij}}(t)\,x_k^{r_{ij}}(t)$$

$$\theta^{r_{ij}}(t+1) = \theta^{r_{ij}}(t) - \nu\,\nabla J_{\theta^{r_{ij}}}(t); \quad \nabla J_{\theta^{r_{ij}}}(t) = \delta^{r_{ij}}(t)$$

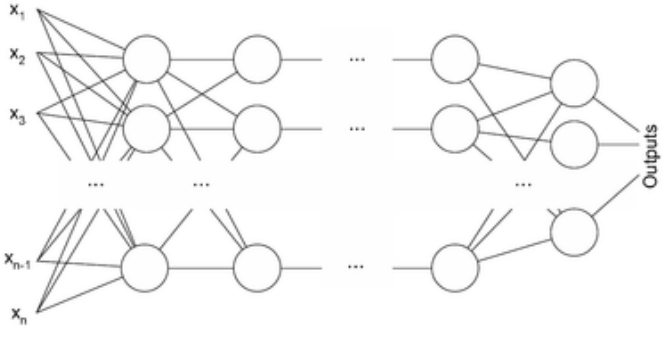


Рис. 1. Персептрон Румельхарта общего вида.

Определение 2. Сетью *радиально базисных функций* будем называть двухслойную нейронную сеть, в которой для нейронов первого слоя используется:

- Функция агрегации $\psi(\overline{x^i}, \overline{\omega^i}) = \sqrt{\sum (x_k^i - \omega_k^i)^2}$, задающая расстояние от входных значений $\overline{x^i}$ до эталонной точки, представленной весами $\overline{\omega^i}$.
- Функция активации $y^i(t) = \phi(z_i) = e^{-\beta_i z_i^2}$, где $z_i = \psi(\overline{x^i}, \overline{\omega^i})$.

Выходной слой сети как правило состоит из одного нейрона с линейной функцией активации и взвешенным суммированием в качестве функции агрегации:

$$y(t) = \sum \omega_k \cdot x_k(t), \quad \text{где} \quad x_k(t) = y^k(t).$$

Обучение сети производится методом обратного распространения ошибки:

$$\omega_k(t+1) = \omega_k(t) - \nu \nabla J_{\omega_k}(t); \quad \nabla J_{\omega_k}(t) = (y(t) - e(t))\, x_k(t)$$

$$\beta_k(t+1) = \beta_k(t) - \nu \nabla J_{\beta_k}(t); \quad \nabla J_{\beta_k}(t) = (e(t) - y(t))\, x_k(t)\, \omega_k(t)\, \beta_k(t)$$

$$\omega_k^i(t+1) = \omega_k^i(t) - \nu \nabla J_{\omega_k^i}(t);$$

$$\nabla J_{\omega_k^i}(t) = (e(t) - y(t))\, x_i(t)\, \omega_i(t)\, 2\sqrt{\beta_i\, ln(1/x_i(t))} \left( \frac{x_k^i(t) - \omega_k^i(t)}{z^i(t)} \right)$$

### 1.2. Градиентный спуск с переменным шагом.

Определение 3. Градиентный спуск *momentum* использует схему вида:

$$v(t+1) = \gamma\, v(t) + \nu \nabla J_{\theta(t)}, \quad \theta(t+1) = \theta(t) - v(t+1)$$

В этом выражении $\nabla J_\theta$ задает градиент для оптимизируемого параметра $\theta$, а параметр инерционности устанавливается в $\gamma \sim 0.9$.

Определение 4. Ускоренный спуск *Нестерова* использует схему вида:

$$v(t+1) = \gamma\, v(t) + \nu \nabla J_{\theta(t) - v(t)}, \quad \theta(t+1) = \theta(t) - v(t+1)$$

Также как и для метода momentum параметр $\gamma$ можно взять $\gamma \sim 0.9$.

Определение 5. Метод *Adagrad* использует схему вида:

$$G_\theta(t) = \sum_{\tau \leqslant t} \nabla J_{\theta(\tau)}, \quad \theta(t+1) = \theta(t) - \frac{\nu}{\sqrt{G_\theta(t) + \epsilon}} \cdot \nabla J_{\theta(t)}$$

В этом выражении суммирование проводится по всем вычисленным ранее значениям градиентов для параметра $\theta$, а параметр $\epsilon \sim 10^{-8}$ вводится для предотвращения деления на ноль.

Определение 6. Метод *Adam* использует схему вида:

$$m(t+1) = \beta_1 m(t) + (1 - \beta_1) \nabla J_{\theta(t)}, \quad v(t+1) = \beta_2 v(t) + (1 - \beta_2) \left( \nabla J_{\theta(t)} \right)^2$$

$$\hat{m}(t) = \frac{m(t)}{1 - \beta_1^t}, \quad \hat{v}(t) = \frac{v(t)}{1 - \beta_2^t}, \quad \theta(t+1) = \theta(t) - \frac{\nu}{\sqrt{\hat{v}(t+1) + \epsilon}} \cdot \hat{m}(t+1)$$

Значения по умолчанию для $\beta_1 \sim 0.9$ и $\beta_2 \sim 0.999$, а $\epsilon \sim 10^{-8}$.

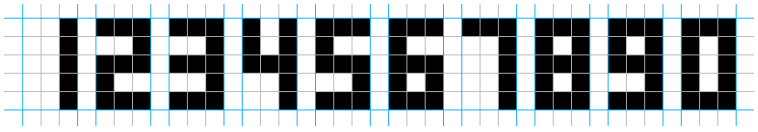

Рис. 2. Схема кодирования цифр для входов персептрона.

## § 2. Практические задания

**Задание № 1**: Классификация цифр.

1. Реализовать персептрон Румельхарта, для классификации цифр на рисунке 2. Входные данные — матрицы 5x3 из нулей и единиц, кодирование классов позиционное (один выходной нейрон = один класс для цифр).
2. Имеет ли значение в каком порядке подавать примеры для обучения нашему персептрону? Что будет, если в примерах отдельные цифры будут подаваться чаще?
3. Что представляет из себя эффект паралича коэффициентов сети? Проиллюстрируйте это на примере, поясните механизм возникновения.
4. Реализуйте альтернативные методы стохастического градиентного спуска для данного примера и сравните их скорости сходимости.
5. Что представляет из себя эффект переобучения сети? Поясните на конкретном примере, постаравшись добиться переобучения данной модели, увеличивая число промежуточных слоев.

**Задание № 2**: Апроксимация хаотических последовательностей.

1. В соотвествии с вариантом выбрать последовательность:
   - *Вариант 1*: $x(t+1) = x(t)(1 - 2(x(t)))$.
   - *Вариант 2*: $x(t+1) = 4x(t)(1 - x(t))$.
   - *Вариант 3*: $x(t+1) = x(t)(1 - x(t)^2)$.
2. Реализовать радиально базисную сеть для предсказания членов последовательности. Входные данные — $x(t)$, а выходные — $x(t+1)$.
3. Обучающую выборку следует собрать из 10-15 значений первых членов последовательности, задав начальное значение $x(0)$ произвольно.
4. Проверить, может ли обученная сеть распознавать и предсказывать новые члены за пределами обучающей выборки.
5. Реализуйте альтернативные методы стохастического градиентного спуска для данного примера и сравните их скорости сходимости и качество обучения (предсказание за пределами обучающей выборки).
6. Возможны ли в данной моделе паралич коэффициентов или переобучение? Если да, то как бы они себя проявили?

## Список литературы

[1] Scikit learn, “Stochastic Gradient Descent”.
[2] Scikit learn, “Multilayer perceptron training methods comparison”.
[3] Dive Into Deep learning, “Stochastic Gradient Descent”.
[4] Dive Into Deep learning, “Multilayer perceptron”.
[5] J. Moody, C. J. Darken, “Fast learning in networks of locally tuned processing units”, *Neural Computation*, **1**:2 (1989), 281–294.

[6] Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J., "Learning representations by back-propagating errors", *Nature*, **6088**:323 (1986), 533–536.