Wprowadzenie do sieci neuronowych w R

mgr Adam Mieldzioc

Katedra Metod Matematycznych i Statystycznych,
Uniwersytet Przyrodniczy w Poznaniu

Poznań, 1 czerwca 2017

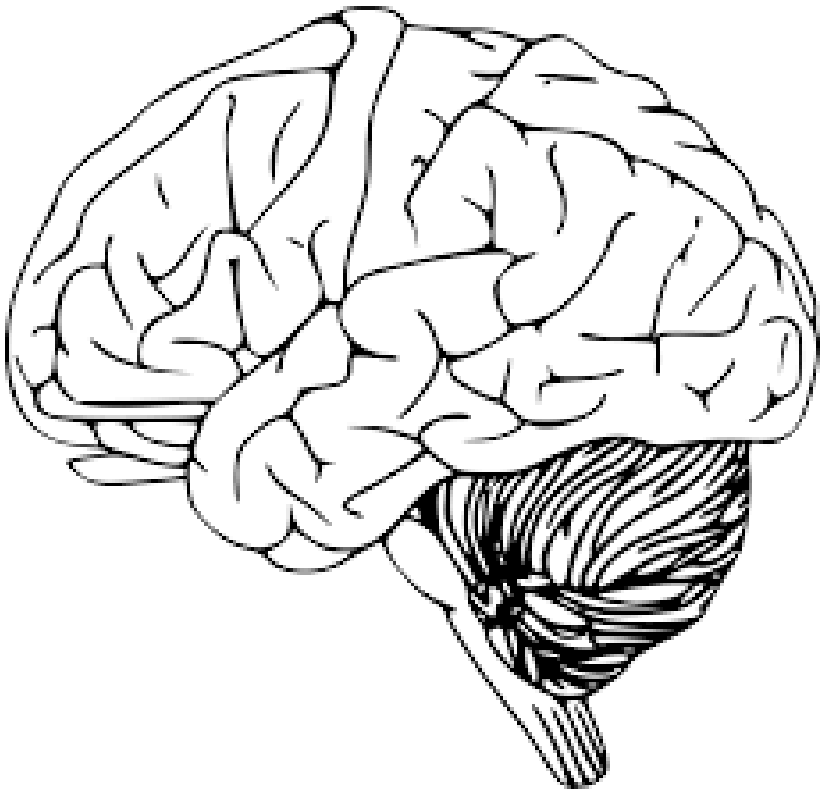
Plan prezentacji

Wstęp

Sieci wielowarstwowe

Przykłady w R

Inspiracja



Najciekawszymi własnościami ludzkiego mózgu są:

- zdolność do rekonstrukcji, odtworzenie sygnału na podstawie niepełnej informacji, która może być obciążona błędem,
- gromadzenie i przetwarzanie informacji,
- bardzo dobre radzenie sobie z rozpoznawaniem, skojarzeniami oraz klasyfikacją,
- korzystając z prostych metod uczenia (np. eksperyment Pawłowa) oraz stosując dużą ilość nieskomplikowanych ogniw przetwarzających informacje (neuronów), jest w stanie wykonywać te wszystkie zawiłe zadania, którym codziennie jest poddawany,
- możliwość przetwarzania 10^{18} operacji logicznych na sekundę.

Jedna z definicji

Sieć neuronowa składa się z grupy lub grup połączonych ze sobą komórek nerwowych. Pojedyncza komórka nerwowa może być połączona z wieloma komórkami nerwowymi, przez co ogólna struktura sieci może być bardzo skomplikowana.

Sztuczna (symulowana) sieć neuronowa to połączona grupa sztucznych komórek nerwowych, która wykorzystuje model matematyczny w celu przetwarzania informacji.

Odrobina historii - ważniejsze daty

- 1943 - McCulloch i Pitts opracowują matematyczny model sztucznego neuronu,
- 1949 - Hebb zauważył, że informacja może być przechowywana w strukturze połączeń między neuronami i jako pierwszy zaproponował metodę uczenia sieci polegającą na zmianach wag połączeń między neuronami,
- 1958 - Rosenblatt buduje perceptron,
- 1975 - Fukushima konstruuje Cognitron,

- 1974, 1982, 1986 - opracowanie niezależnie **algorytmu wstecznej propagacji błędu** (ang. *backpropagation*) przez Werbosa, Parkera oraz Rumelharta i innych,
- (umowny) 2006 - zwiększenie mocy obliczeniowej komputerów, co spowodowało powstanie nowych algorytmów i rodzajów sieci neuronowych.

Przyczyny sukcesu

- Moc
- Prostota

Rodzaje neuronów

- Perceptron
- ADALINE (ADAptive LInear NEuron)
- Neuron sigmoidalny
- Neuron Hebba

Perceptron

Działanie perceptronu można zapisać za pomocą następującego wzoru:

$$y = f(s), \tag{1}$$

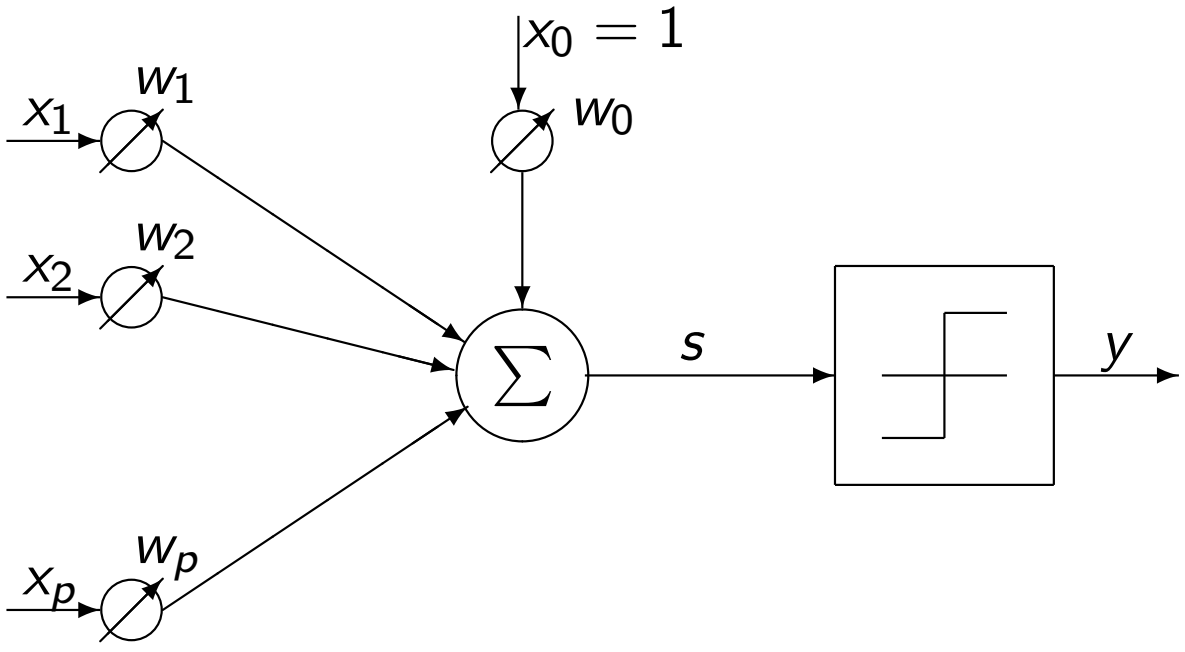
gdzie

$$s = \sum_{i=0}^{p} x_i w_i = \mathbf{x}^{\mathbf{T}}\mathbf{w} + w_0. \tag{2}$$

Przyjmujemy tutaj, że x_0=1, zaś w_0 jest wartością progową. Funkcją aktywacji f jest funkcja bipolarna (funkcja signum) postaci:

$$f(s) = \begin{cases} 1, & dla \quad s \geq 0, \\ -1, & dla \quad s < 0. \end{cases} \tag{3}$$

Model perceptronu



Nazwa funkcji	Opis działania funkcji
liniowa	$f(x) = \begin{cases} 1, & dla \quad x \geq 0 \\ 0, & dla \quad x < 0 \end{cases}$
liniowa dodatnia (relu)	$f(x) = \begin{cases} x, & dla \quad x \geq 0 \\ 0, & dla \quad x < 0 \end{cases}$
logarytmiczno-sigmoidalna	$f(x) = \frac{1}{1+e^{-x}}$
niesymetryczna liniowa z nasyceniem	$f(x) = \begin{cases} 1, & dla \quad x > 1 \\ x, & dla \quad 0 \leq x \leq 1 \\ 0, & dla \quad x < 0 \end{cases}$
symetryczna liniowa z nasyceniem	$f(x) = \begin{cases} 1, & dla \quad x > 1 \\ x, & dla \quad 0 \leq x \leq 1 \\ -1, & dla \quad x < 0 \end{cases}$
tangens hiperboliczny	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
unipolarna	$f(x) = \begin{cases} 1, & dla \quad x \geq 0 \\ 0, & dla \quad x < 0 \end{cases}$

Algorytm uczenia perceptronu

1. Losowo wybieramy początkowe wagi perceptronu $\mathbf{w}(0)$.
2. Uczymy perceptron, za pomocą kolejnych, losowo wybranych elementów $\mathbf{x}_i$ zbioru uczącego $\mathbf{Z}$. Dla każdej obserwacji $\mathbf{x}_i$ obliczamy jej wartość wyjściową y_i zgodnie z formułą (1).
3. Jeżeli wartość y_i różni się od $d(\mathbf{x}_i)$ to dokonujemy modyfikacji wag według wzoru:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + d(\mathbf{x}_i)\mathbf{x}_i. \tag{4}$$

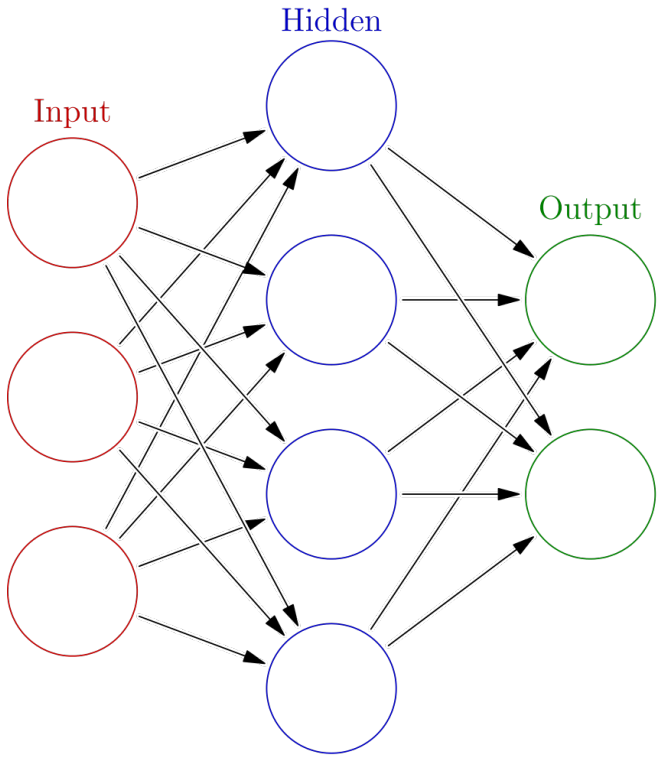
4. Powrót do punktu 2.

Przyczyny powstania i rozwoju sieci wielowarstwowych

- nie wystarczająca moc pojedynczego neuronu,
- prosta i przejrzysta struktura sieci,
- zrozumiały algorytm uczenia sieci neuronowej,
- stabilność sieci.

Struktura sieci wielowarstwowej

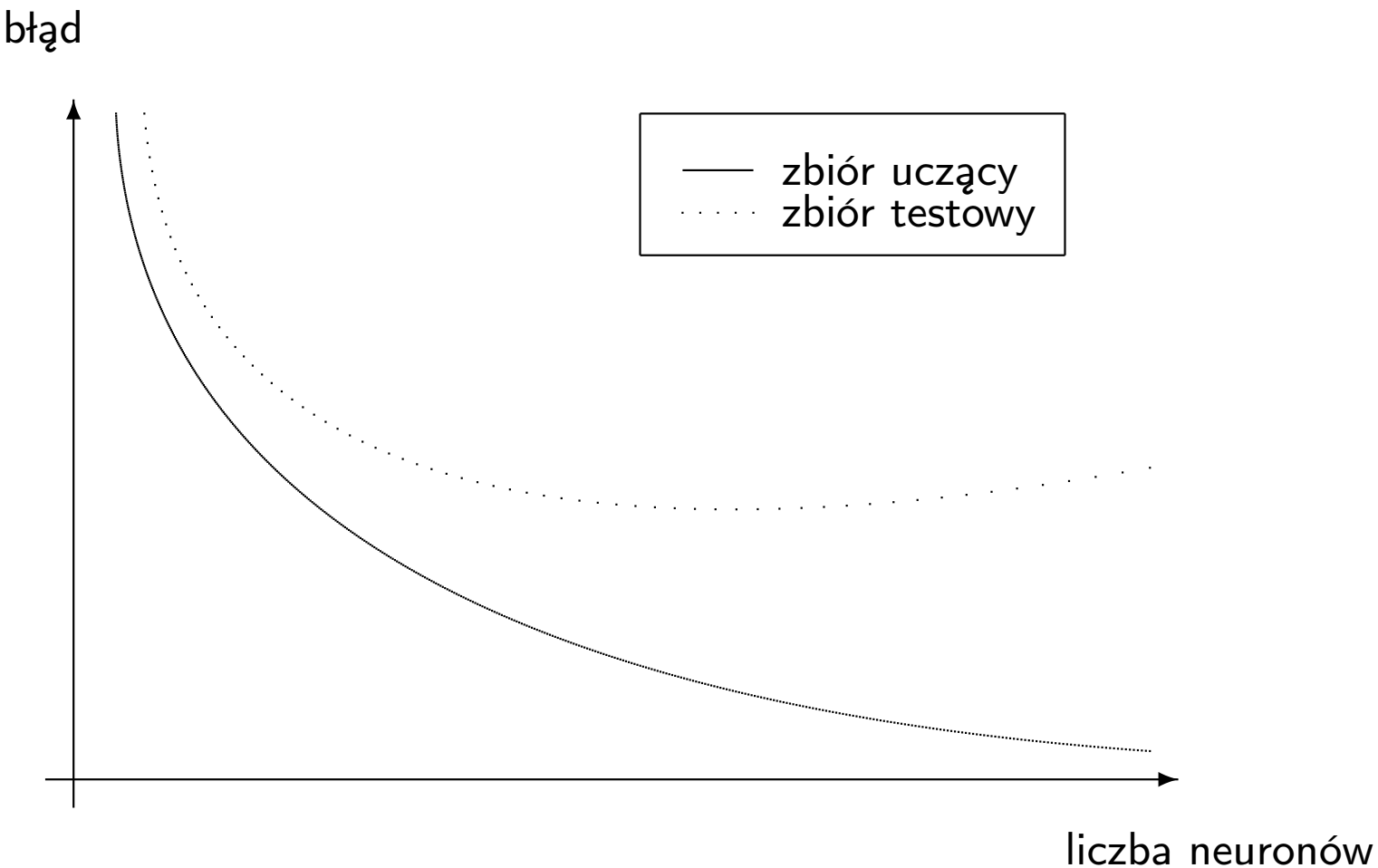
Wielowarstwowe sieci neuronowe, jak sama nazwa wskazuje składają się z wartw. Neurony znajdujące się w tej samej warstwie nie są połączone ze sobą. Każdy neuron z i-tej warstwy jest połączony z każdym neuronem z warstwy $i + 1$. Przepływ sygnału jest od warstwy wejściowej do warstwy wyjściowej. Taką sieć nazywamy siecią jednokierunkową.


Hidden
Input
Output

Funkcja aktywacji nie musi być taka sama dla wszystkich neuronów!

Sygnały wyjściowe ostatniej warstwy są jednocześnie sygnałami wyjściowymi sieci neuronowej. Sygnały te są porównywane z sygnałami wzorcowymi $d_1^L, ..., d_{N_L}^L$.

Efekt szczytu



Funkcja błędu

$$Q = Q(\mathbf{w})$$

$$Q(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}[\mathbf{d}_i - \mathbf{y}(\mathbf{x}_i, \mathbf{w})]^T[\mathbf{d}_i - \mathbf{y}(\mathbf{x}_i, \mathbf{w})]. \quad (5)$$

Podstawy iteracyjnego uczenia sieci neuronowej

Najczęsciej do procesu uczenia wykorzystuje się algorytmy gradientowe. Poprawki wektora wag dokonujemy według wzoru:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t). \tag{6}$$

Najczęściej poprawka $\Delta\mathbf{w}(t)$ polega na dodaniu pewnego wektora $\mathbf{p}(t)$. Mamy:

$$\Delta\mathbf{w}(t) = \eta\mathbf{p}(t),$$

gdzie η jest współczynnikiem uczenia.

W przypadku algorytmu gradientowego skorzystamy z rozwinięcia funkcji błędu w kierunku wektora $\mathbf{p}(t)$:

$$Q(\mathbf{w}+\mathbf{p}) = Q(\mathbf{w}) + [\nabla Q(\mathbf{w})]^T \mathbf{p} + \frac{1}{2}\mathbf{p}^T \mathbf{H}(\mathbf{w})\mathbf{p} + \ldots \qquad (7)$$

Wtedy poprawkę otrzymamy ze wzoru:

$$\Delta \mathbf{w}(t) = -\eta \nabla Q(\mathbf{w}(t)). \qquad (8)$$

Tak działa właśnie metoda największego spadku.

Kryteria stopu

1. $||\nabla Q(\mathbf{w}(t))|| < \epsilon$ - test stacjonarności.
2. $||\mathbf{w}(t) - \mathbf{w}(t-1)|| < \epsilon$ lub $||Q(\mathbf{w}(t)) - Q(\mathbf{w}(t-1))|| < \epsilon$ - test szybkości zbieżności.
3. $t > T_0$ - test liczby iteracji.

Ciekawe twierdzenia

- O uniwersalnych własnościach aproksymujących
- Tw. Kolomogorowa 1953, Lorentza 1976

Algorytm wstecznej propagacji błędu

$$w_{ij}^{(k)}(t+1) = w_{ij}^{(k)}(t) + \eta(t)\frac{\partial Q(\mathbf{w}(t))}{\partial w_{ij}^{(k)}(t)}. \tag{9}$$

Kroki algorytmu

Algorytm wstecznej propagacji błędu

1. Losowo wybieramy początkowe wagi sieci neuronowej.
2. Losowo wybieramy obserwację $\mathbf{x}_i$ ze zbioru uczącego.
3. Wyznaczamy wszystkie wartości wyjściowe.
4. Obliczamy wartość funkcji błędu.
5. Wykorzystując regułę delty modyfikujemy wagi neuronów ostatniej warstwy.
6. Błąd wyjściowy propagujemy od tyłu poprzez obliczenie $\frac{\partial Q(\mathbf{w}(t))}{\partial w_{ij}^{(k)}(t)}$

   i modyfikację wag zgodnie z połączeniami neuronów między warstwami oraz z uwzględnieniem ich funkcji aktywacji.
7. Powtarzamy kroki 2-6 tak długo, aż np. wartość funkcji błędu będzie mniejsza od wyznaczonego poziomu tolerancji.

Pakiety

- neuralnet
- nnet
- deepnet
- darch
- **H2O**
- **MXNet**

H2O

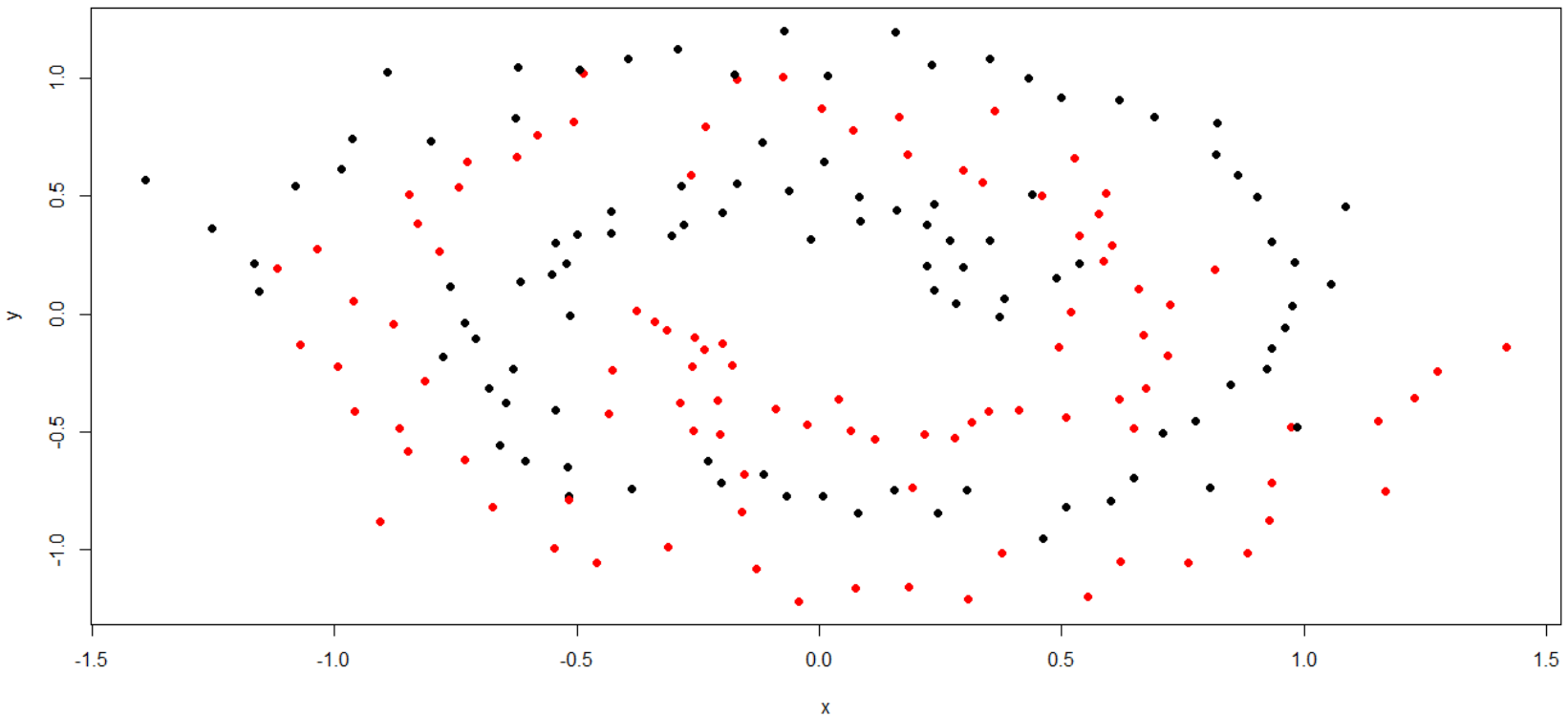
```
library(h2o)
h2o.init(nthreads=-1, max_mem_size="4G")
h2o.removeAll()

setwd("my_path")

data <- h2o.importFile(path =
normalizePath("../another_path"))
```

```
h2o.deeplearning(x, y, training_samples,
    activation = c("Tanh", "TanhWithDropout",
    "Rectifier", "RectifierWithDropout",
    "Maxout", "MaxoutWithDropout"),
    hidden = c(200, 200), epochs = 10,
    seed = -1, adaptive_rate = TRUE,
    rho = 0.99, epsilon = 1e-08, rate = 0.005,
    rate_annealing = 1e-06, rate_decay = 1,
    l1 = 0, l2 = 0, loss = c("Automatic",
    "CrossEntropy", "Quadratic","Huber",
    "Absolute", "Quantile"), stopping_metric
   = c("AUTO","deviance", "logloss", "MSE",
    "RMSE", "MAE","RMSLE", "AUC", "lift_top_group",
    "misclassification","mean_per_class_error"))
```

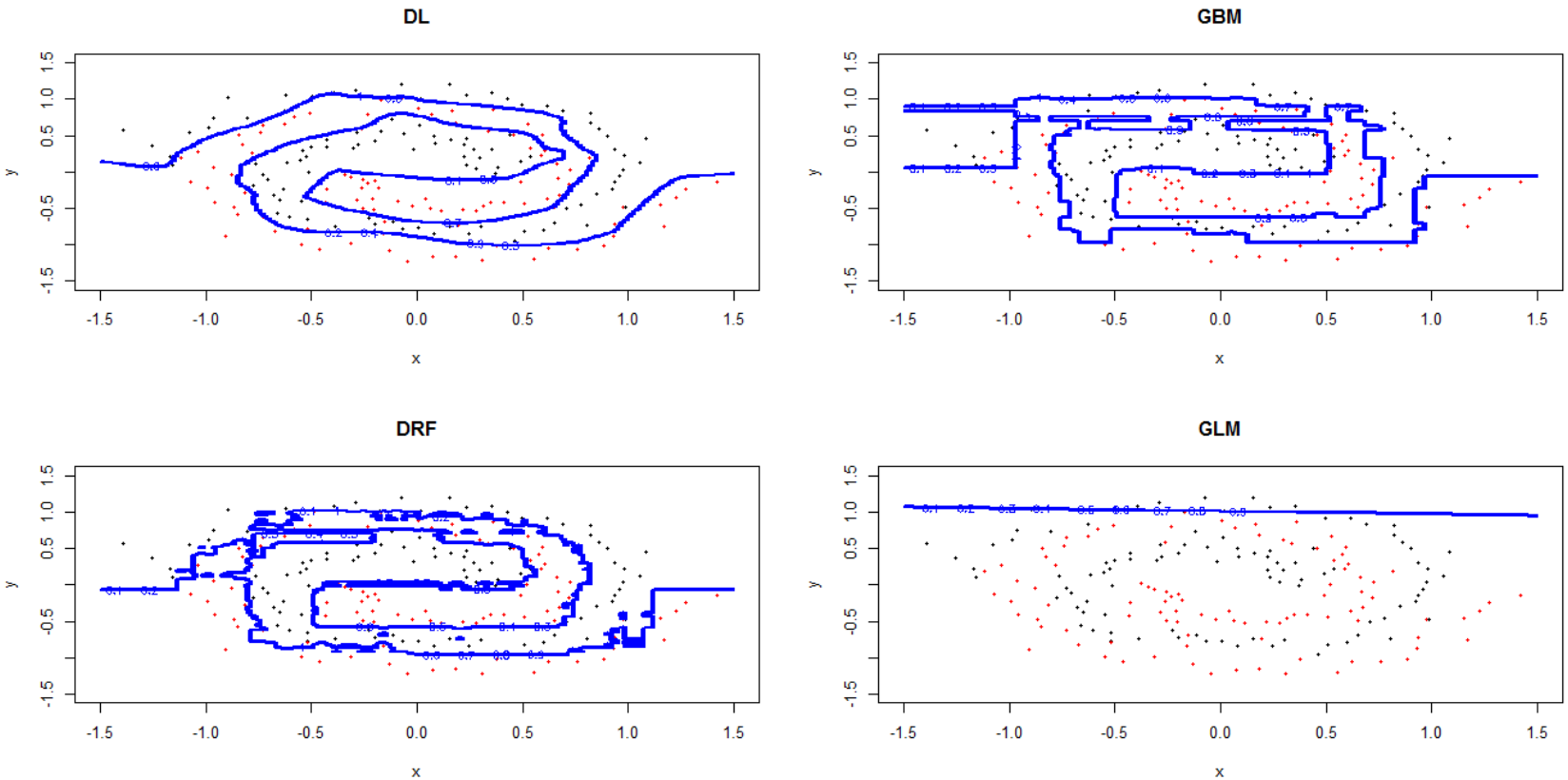
Przykładowe wywołanie

```
h2o.deeplearning(1:2, 3, spiral,
        activation = "Tanh", hidden = c(50, 50, 50)
    epochs = 100)
```

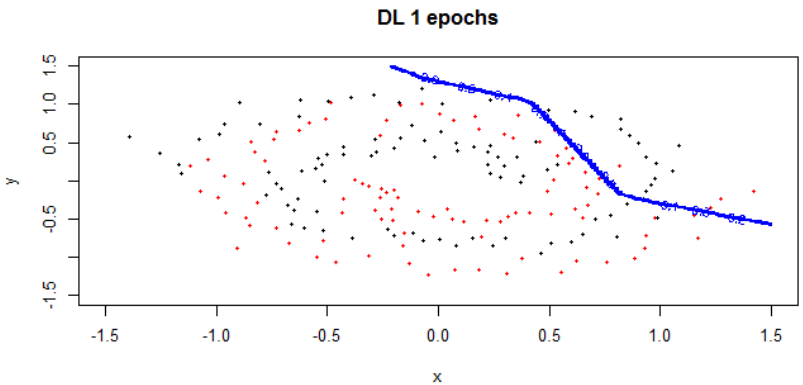
Spiral data

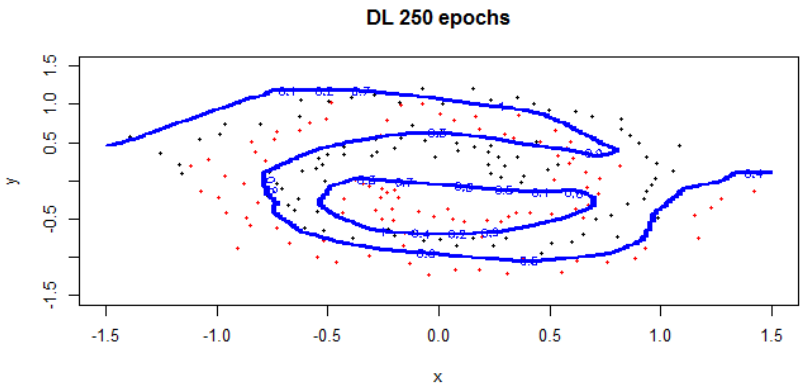


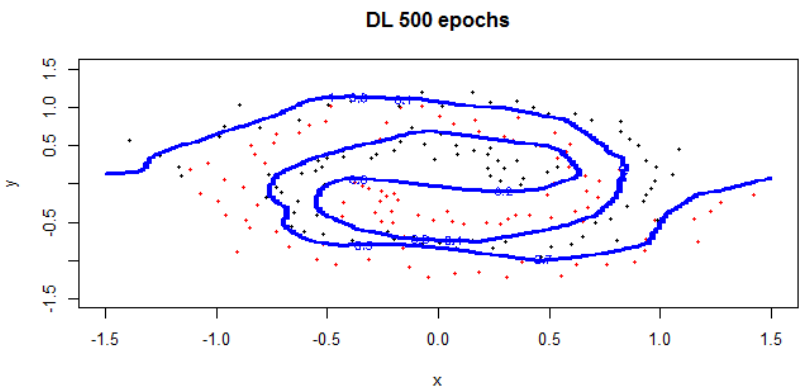
Porównanie z innymi metodami

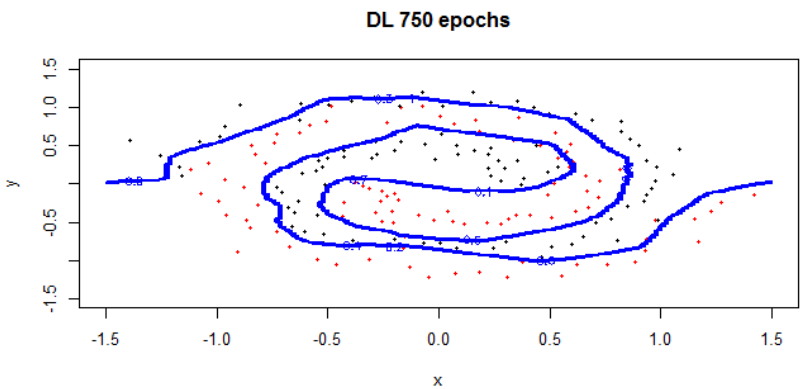


Porównianie działania dla różnej liczby epok

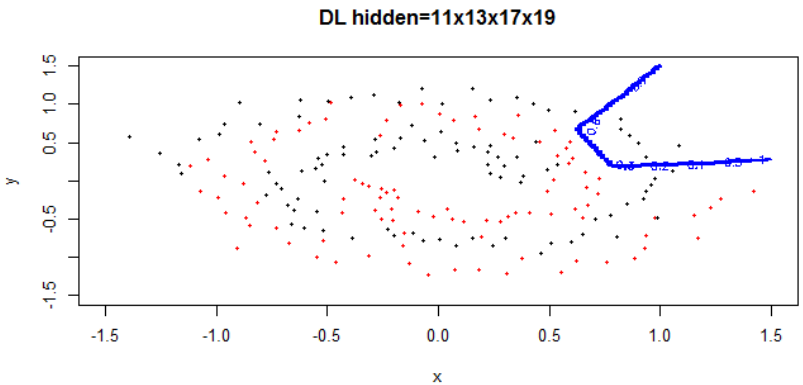


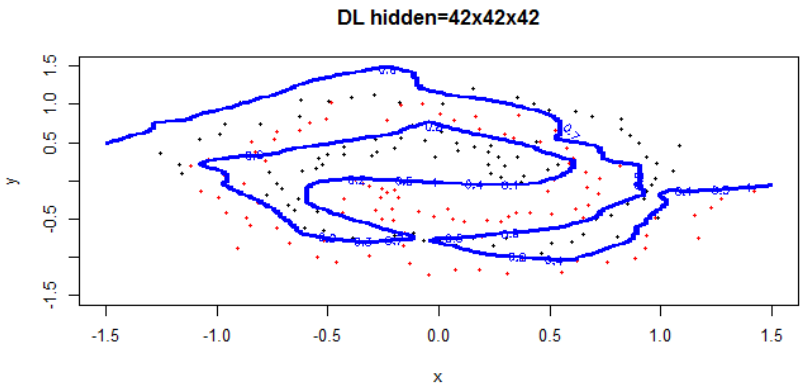


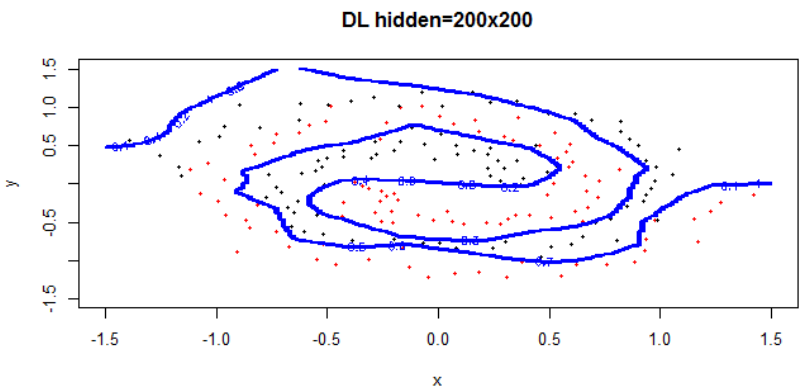


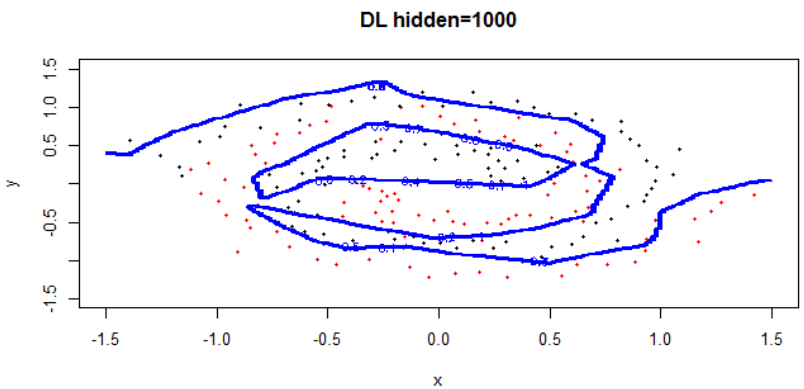


Wybór liczby neuronów









Poszukiwanie najlepszego modelu

```
hyper_params <- list(
  hidden=list(c(32,32,32),c(64,64),
              c(200,200), c(100,100,100)),
  input_dropout_ratio=c(0,0.05),
  activation = c("Tanh", "Rectifier")
)
```

```
search_criteria = list(strategy = "RandomDiscrete",
      max_runtime_secs = 360, max_models = 100,
      seed=1234567, stopping_rounds=5,
      stopping_tolerance=1e-2)
```

```
grid <- h2o.grid(
    algorithm="deeplearning", grid_id="dl_grid",
    training_frame = train1,
    validation_frame = valid,
    x=predictors, y=response, epochs=1000,
    stopping_metric="logloss",
    stopping_tolerance=1e-2,
    stopping_rounds=2, score_validation_samples=200
    score_duty_cycle=0.025, l1=1e-5, l2=1e-5,
    hyper_params=hyper_params,
    search_criteria = search_criteria)
```

```
g = h2o.getGrid("dl_grid",sort_by="err",
          decreasing=FALSE)
best_model <- h2o.getModel(g@model_ids[[1]])
best_model
```

MXNet - v.0.9.4

Bibliografia

1. Krzyśko M. i in., Systemy uczące się, Wydawnictwo WNT, Warszawa 2009
2. Ripley B. D., Pattern recognition and neural networks, Cambridge University Press, Cambridge, 2005
3. Bishop C. M.,Neural networks for pattern recognition, Clarendon Press, Oxford, 1995
4. Dokumentacja pakietu H2O: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/deep-learning.html
5. Dokumentacja pakietu MXNet: http://mxnet.io/api/r/mxnet-r-reference-manual.pdf
6. http://neuralnetworksanddeeplearning.com

Kontakt

Adam Mieldzioc: adam.mieldzioc@mail.up.poznan.pl

Dziękuję za uwagę!