Projection Pursuit Model i Sieci Neuronowe

Agata Rogowska, Pola Lubińska

March 6, 2022

Wstęp

- Projection Pursuit Model
- Neural Network Model (model sieci neuronowych)

Projection Pursuit Regression

- macierz $X_{n\times d}$ - n osób, d cech
- $Y_{n\times 1}$ - 1 - osoba w klasie A, 0 - osoba w klasie B

Chcemy estymować $p_i = P(Y_i = 1|X_i) = E(Y_i|X_i)$, $i = 1, \ldots, n$.
Klasyfikacja:

$$\widehat{p_i} \geqslant 0.5 \Rightarrow \widehat{Y_i} = 1, \text{ czyli, } i\text{-ta osoba należy do klasy A,}$$
$$\widehat{p_i} < 0.5 \Rightarrow \widehat{Y_i} = 0, \text{ czyli, } i\text{-ta osoba należy do klasy B.}$$

Projection Pursuit Regression

Najprostszym sposobem estymacji p_i będzie użycie modelu regresji logistycznej, która zakłada poniższy związek między prawdopodobieństwem sukcesu p_i, a wektorem cech X_i

Regresja Logistyczna

$$\text{logit}(p_i) = \log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \sum_{j=1}^{d} \beta_j X_{ij} \tag{1}$$

Projection Pursuit Regression

Najprostszym sposobem estymacji p_i będzie użycie modelu regresji logistycznej, która zakłada poniższy związek między prawdopodobieństwem sukcesu p_i, a wektorem cech X_i

Regresja Logistyczna

$$\text{logit}(p_i) = \log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \sum_{j=1}^{d} \beta_j X_{ij} \tag{1}$$

Może się jednak okazać, że zależność między prawdopodobieństwem sukcesu p_i, a wektorem cech X_i nie jest dobrze modelowana przez (1). Wtedy zależność tę chcielibyśmy określić w bardziej ogólny sposób:

$$\text{logit}(p_i) = \log\left(\frac{p_i}{1-p_i}\right) = h(X_i) \tag{2}$$

$h(X_i)$ - nieliniowa funkcja X_i - problem z określeniem

Projection Pursuit Regression

Uogólnione modele addytywne (GAM) Hastiego i Tibshiraniego

$$h(X_i) = \beta_0 + \sum_{j=1}^{d} f_j(X_{ij}), \tag{3}$$

f_j estymujemy używając metod wygładzania wykresu rozrzutu (scatterplot smoothers) i ograniczamy do średniej zero.

Jednak nie wszystkie funkcje h można modelować jako suma funkcji gładkich jak w (3).

Projection Pursuit Regression

Projection Pursuit Regression

$$h(X_i) = \beta_0 + \sum_{j=1}^{M} \beta_j f_j(\alpha_j^T X_i), \tag{4}$$

gdzie

- $\sum_{k=1}^{d} \alpha_{kj}^2 = 1,$
- $(f_j) = 0,$
- $(f_j^2) = 1.$

Jest to model addytywny, ale nie danych wejściowych, tylko pochodnych cech $V_j = \alpha_j^T X_i$. Zmienna skalarna $V_j = \alpha_j^T X_i$ jest rzutem wektora X_i na wektor α_j, a ponieważ szukamy α_j takiego aby model dobrze dopasowywał się do danych, stąd nazwa „projection pursuit".

Projection Pursuit Regression

Model postaci $\widehat{p}_i = h(X_i)$ dopasowujemy do danych treningowych minimalizując residualną sumę kwadratów (RSS).

$$\mathrm{RSS} = \sum_{i=1}^{n}(Y_i - \widehat{p}_i)^2 = \sum_{i=1}^{n}\left(Y_i - \beta_0 - \sum_{j=1}^{M}\beta_j f_j(\alpha_j^T X_i)\right)^2 \tag{5}$$

Musimy jednakże nałożyć ograniczenie na na f_j, aby uniknąć przeuczenia.

Projection Pursuit Regression

Rozważmy przypadek:

- $M = 1$
- $\beta_0 = 0$, $\beta_1 = 1$

Wtedy:

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(Y_i - f(\alpha^T X_i) \right)^2$$

Estymacja f - algorytm Gaussa–Newtona
Estymacja α - metoda quasi–Newtona

Projection Pursuit Regression

Szereg Taylora dla $f(x)$:

$$f(x_k + \triangle x) \approx f(x_k) + \bigtriangledown f(x_k)^T \triangle x$$

W naszym przypadku:

- $x_k + \triangle x = \alpha^T x_i$
- $x_k = \alpha_{OLD}^T x_i$
- $\bigtriangledown f(x_k) = f'(\alpha_{OLD}^T x_i)$
- $\triangle x = \alpha^T x_i - \alpha_{OLD}^T x_i = (\alpha - \alpha_{OLD})^T x_i$

więc możemy zapisać:

$$f(\alpha^T x_i) \approx f(\alpha_{OLD}^T x_i) + f'(\alpha_{OLD}^T x_i)(\alpha - \alpha_{OLD})^T x_i \tag{6}$$

Projection Pursuit Regression

$$
\begin{aligned}
\mathrm{RSS} &= \sum_{i=1}^{n} \left(Y_i - f(\alpha^T X_i) \right)^2 \approx \\
&\approx \sum_{i=1}^{n} \left(Y_i - f(\alpha_{OLD}^T X_i) - f'(\alpha_{OLD}^T X_i)(\alpha - \alpha_{OLD})^T X_i) \right)^2 = \\
&= \sum_{i=1}^{n} \left(Y_i - f(\alpha_{OLD}^T X_i) - f'(\alpha_{OLD}^T X_i)\alpha^T X_i + f'(\alpha_{OLD}^T X_i)\alpha_{OLD}^T X_i) \right)^2 = \\
&= \sum_{i=1}^{n} \left(f'(\alpha_{OLD}^T X_i) \left(\frac{Y_i - f(\alpha_{OLD}^T X_i)}{f'(\alpha_{OLD}^T X_i)} - \alpha^T X_i + \alpha_{OLD}^T X_i) \right) \right)^2 = \\
&= \sum_{i=1}^{n} \left(f'(\alpha_{OLD}^T X_i) \left(\alpha_{OLD}^T X_i + \frac{Y_i - f(\alpha_{OLD}^T X_i)}{f'(\alpha_{OLD}^T X_i)} - \alpha^T X_i) \right) \right)^2 = \\
&= \sum_{i=1}^{n} f'(\alpha_{OLD}^T X_i)^2 \left(\left(\alpha_{OLD}^T X_i + \frac{Y_i - f(\alpha_{OLD}^T X_i)}{f'(\alpha_{OLD}^T X_i)} \right) - \alpha^T X_i) \right)^2
\end{aligned}
$$

Projection Pursuit Regression

$$RSS \approx \sum_{i=1}^{n} f'(\alpha_{OLD}^T X_i)^2 \left(\left(\alpha_{OLD}^T X_i + \frac{Y_i - f(\alpha_{OLD}^T X_i)}{f'(\alpha_{OLD}^T X_i)} \right) - \alpha^T X_i) \right)^2$$

Aby zminimalizować prawą stronę, przeprowadzamy regresję liniową bez intercepta z wagami $f'(\alpha_{OLD}^T X_i)^2$, w której za zmienną objaśnianą przyjmujemy $\alpha_{OLD}^T X_i + \frac{Y_i - f(\alpha_{OLD}^T X_i)}{f'(\alpha_{OLD}^T X_i)}$, zmiennymi objaśniającymi są X_i, a współczynniki zapisane są w wektorze α. Wyznaczenie $\widehat{\alpha}$ będzie aktualizowało wektor α_{NEW}.
Zarówno estymacja g, jak i estymacja α powtarzane są tak długo aż osiągniemy zbieżność.
W przypadku kiedy $M > 1$ model budowany jest wykorzystując algorytm forward stage-wise dodając w każdym etapie parę (α_m, f_m), $m = 1, \ldots, M$.

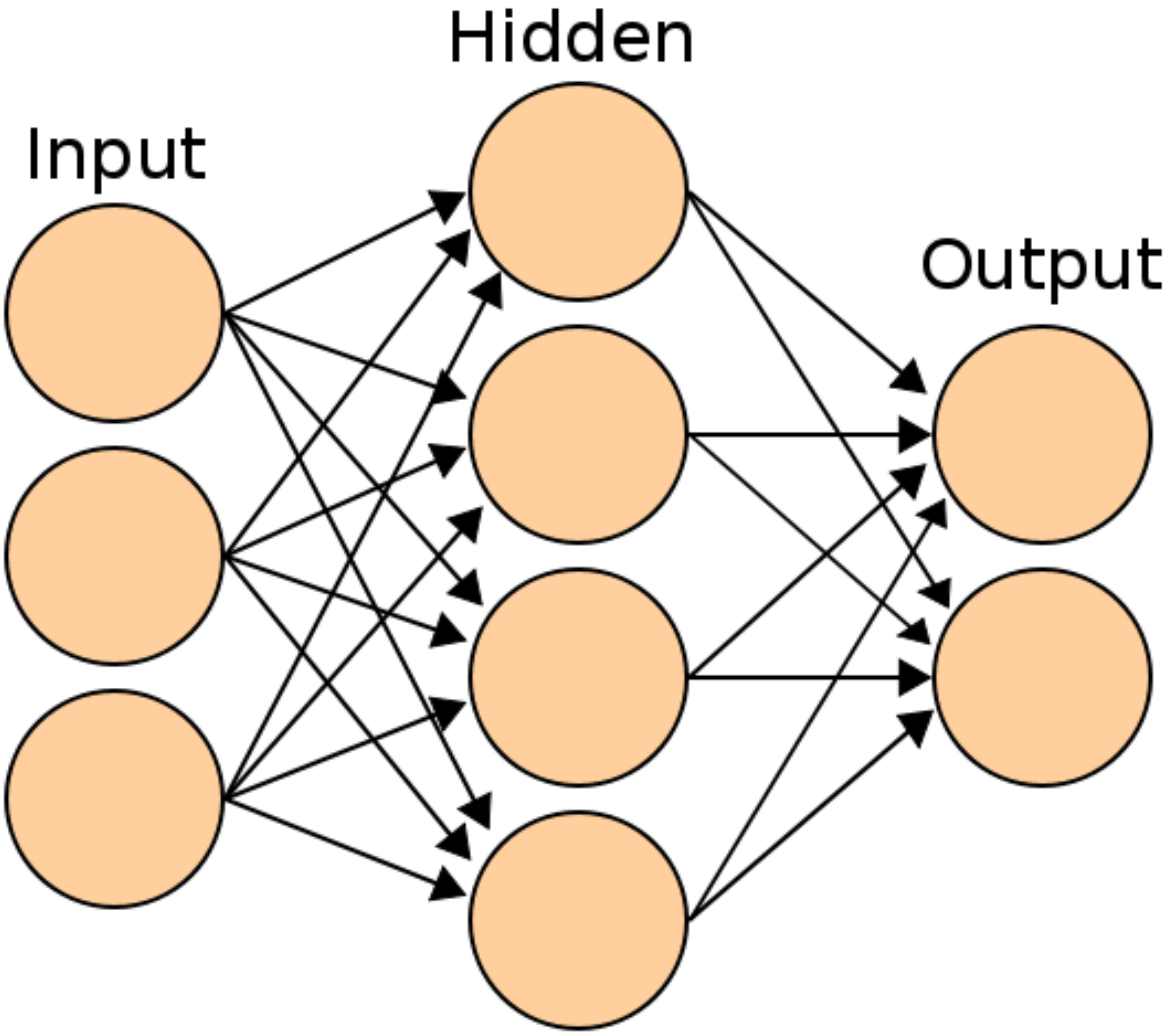
Projection Pursuit Regression

Uwagi:

- Można zastosować dowolną metodę wygładzenia. Jednakże stosowanie regresji lokalnie i tworzenie smoothing splines jest wygodne.
- Po każdym kroku, f_m z poprzedniego kroku możemy ponownie dopasować używając procedury *backfitting* (rodział 9). Prowadzi to do mniejszej liczby M, lecz nie jest jasne czy poprawia to predykcję.
- Zwykle nie dopasowujemy po raz drugi α_m, ponieważ może to prowadzić do nadmiernych obliczeń.
- Zwykle liczba M szacowana jest podczas działania algorytmu forward stage-wise. Kończymy budowę modelu, gdy kolejny m nie poprawia dopasowania modelu. Możemy też użyć walidacji krzyżowej.

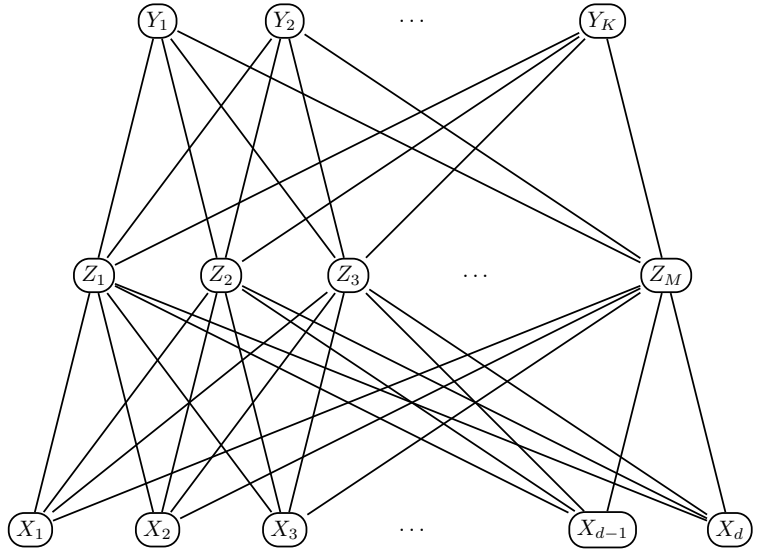
Model nie był szeroko stosowany w statystyce, ponieważ w momencie wprowadzenia, jego wymagania obliczeniowe przekroczyły możliwości większości łatwo dostępnych komputerów.

Sieci Neuronowe



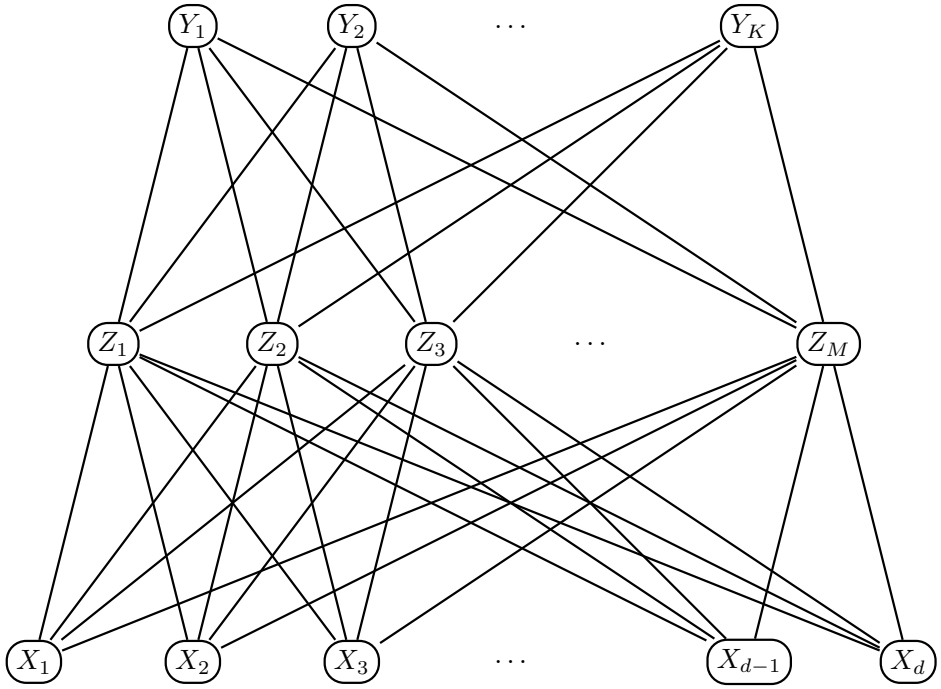
Sieci Neuronowe

„Waniliowy” (podstawowy) model sieci neuronowej, zwany również single hidden layer back-propagation network lub perceptronem jednowarstwowym. Dwuetapowy model regresji lub klasyfikacji.



W przypadku regresji, zwykle, $K = 1$ i wtedy mamy tylko jedną zmienną wyjściową (Y_1). Jeżeli klasyfikujemy obiekty do K klas, to Y_k modeluje prawdopodobieństwo k-tej klasy.

Sieci Neuronowe



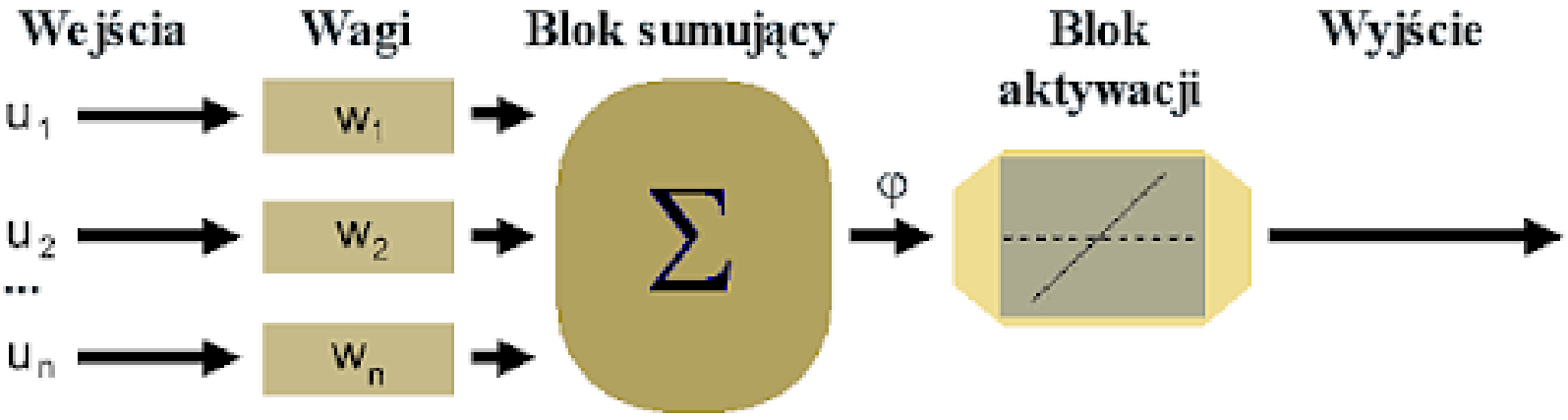
$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X),\ m = 1, \ldots, M,$$

$$T_k = \beta_{0k} + \beta_k^T Z,\ k = 1, \ldots, K,$$

$$h_k(X) = g_k(T),\ k = 1, \ldots, K,$$

gdzie $Z = (Z_1, \ldots, Z_M)$, $T = (T_1, \ldots, T_K)$, a funkcję σ nazywamy funkcją aktywacji.

Sieci Neuronowe



Przejście pomiędzy danymi wejściowymi, a warstwą ukrytą, a później z warstwy ukrytej do danych wyjściowych.

Sieci Neuronowe

Funkcja aktywacji może być na przykład:

- funkcją liniową

$$y = a\varphi + b,$$

- funkcją skoku jednostkowego

$$y = \begin{cases} 1 & \text{jeśli } \varphi > \varphi_h \\ 0 & \text{dla pozostałych,} \end{cases}$$

  gdzie φ_h jest zadaną stałą wartością progową.
- funkcją sigmoidalną (dokładniej opisuje nieliniową charakterystykę funkcji aktywacji neuronu biologicznego)

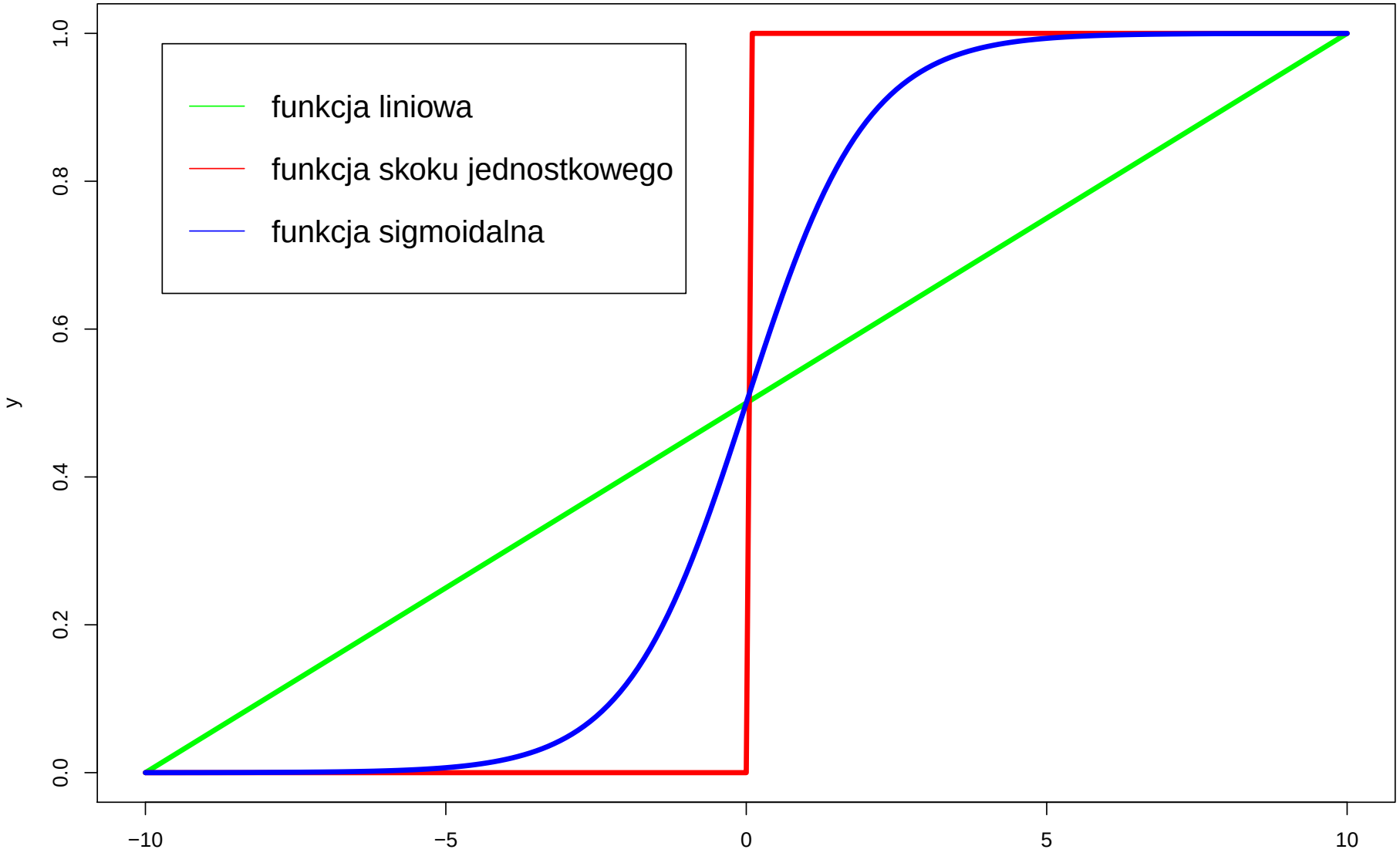
$$y = \frac{1}{1 + e^{-\varphi}},$$

- funkcją tangesoidalną

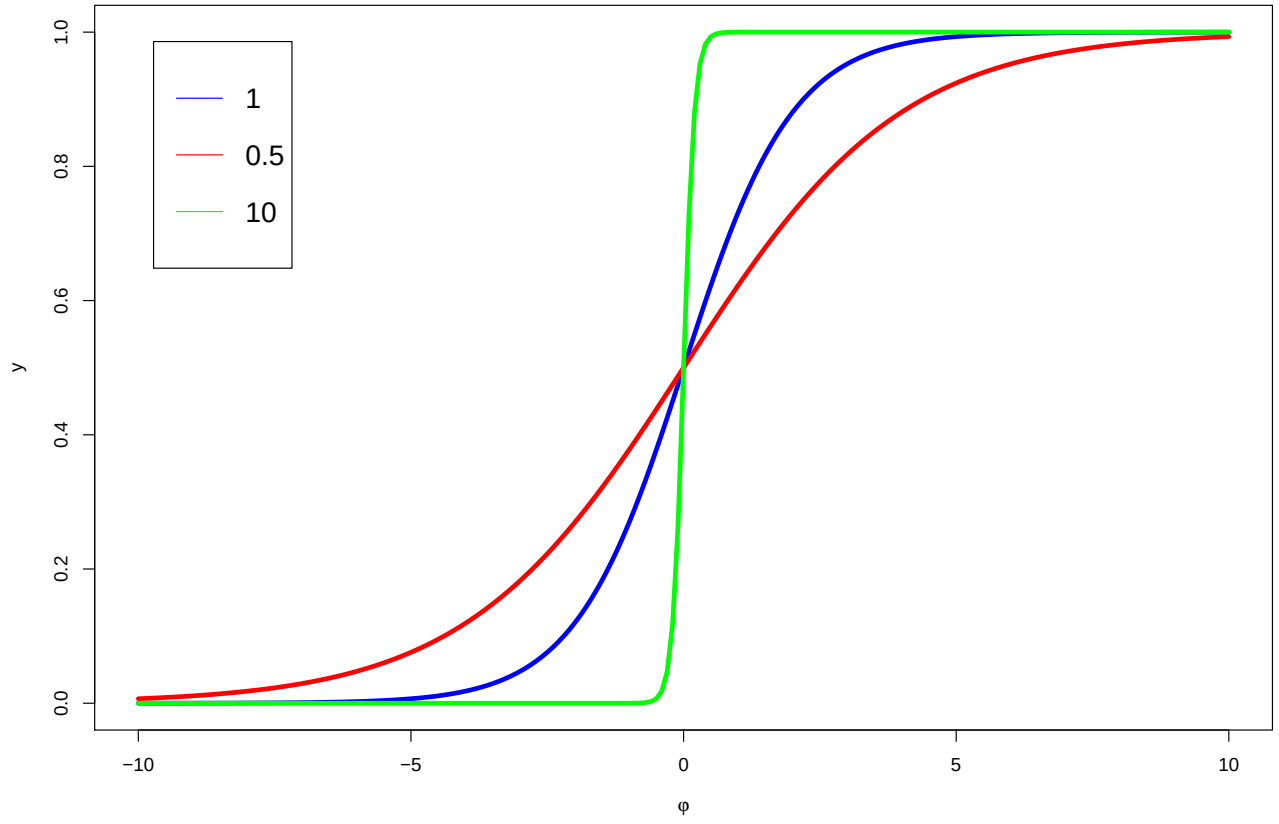
$$y = tgh(\frac{\varphi}{2}) = \frac{1 - e^{-\varphi}}{1 + e^{-\varphi}},$$

  gdzie *thg* to tangens hiperboliczny.

Sieci Neuronowe

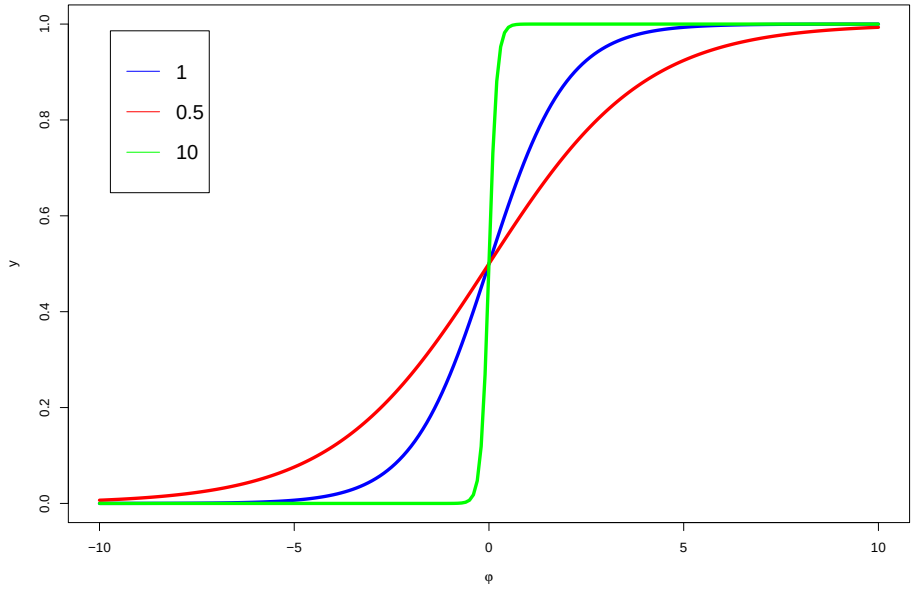


Sieci Neuronowe



Powyższy rysunek przedstawia wykres funkcji $sigmoid(\varphi)$ (niebieska), $sigmoid(\frac{1}{2}\varphi)$ (czerwona) oraz $sigmoid(10\varphi)$ (zielona).

Sieci Neuronowe



Na powyższym wykresie obserwujemy, że wartości funkcji sigmoidalnej zależą od normy wektora α_m oraz, że:

- jeżeli wartości $||\alpha_m||$ są małe (czerwona), to wartości funkcji sigmoidalnej zaczynają przypominać wartości funkcji liniowej;
- jeżeli wartości $||\alpha_m||$ są duże (zielona), to wartości funkcji sigmoidalnej zaczynają przypominać wartości funkcji skoku jednostkowego.

Sieci Neuronowe

Zauważmy, że w przypadku gdy σ jest funkcją tożsamościową, to model jest modelem liniowych zmiennych wejściowych. Stąd możemy myśleć o sieci neuronowej jak o nieliniowym uogólnieniu modelu liniowego, zarówno dla regresji, jak i dla klasyfikacji.
Funkcja $g_k(T)$ umożliwia ostateczną transformację wektora T. W przypadku regresji zazwyczaj wybieramy funkcję tożsamościową $g_k(T) = T_k$. Natomiast w przypadku klasyfikacji wybieramy funkcję softmax

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}$$

Funkcja ta daje wyniki, których suma jest równa jeden (sumowanie od 1 do K).

Sieci Neuronowe

Zauważmy, że model sieci neuronowej z jedną warstwą ukrytą ma tę samą formę co model PPR. Różnica polega na tym, że model PPR wykorzystuje funkcje nieparametryczne $g_m(v)$. Natomiast sieci neuronowe znacznie prostszą, trzyparametrową funkcję $\sigma(v)$. Zależność pomiędzy modelem PPR, a siecią neuronową możemy przedstawić jako

$$f_m(\omega_m^T X) = \sigma(\alpha_{0m} + \alpha_m^T X) = \sigma(\alpha_{0m} + ||\alpha_m||(\omega_m^T X))$$

gdzie $\omega_m = \frac{\alpha}{||\alpha_m||}$.

Sieci Neuronowe - dopasowywanie parametrów (*wag*)

Wagi:

$$\theta = \{\alpha_{0m}, \alpha_m : m = 1, \ldots, M\} \cup \{\beta_{0k}, \beta_k : k = 1, \ldots, K\}$$

Sieci Neuronowe - dopasowywanie parametrów (*wag*)

Dla regresji:

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2$$

Dla klasyfikacji:

$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i)$$

z klasyfikatorem $G(x) = argmax_k f_k(x_i)$

Sieci Neuronowe - dopasowywanie parametrów (*wag*)

Niech $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$ oraz $z_i = (z_{1i}, \ldots, z_{Mi})$.
Wtedy:

$$R(\theta) = \sum_{i=1}^{N} R_i = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2$$

a pochodne są postaci:

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) z_{mi}$$

$$\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i) x_{il}$$

Sieci Neuronowe - dopasowywanie parametrów (*wag*)

(r+1) iteracja Gradient descent

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}$$

$$\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}}$$

gdzie γ_r jest szybkością uczenia (ang. *learning rate*).
W omawianym przypadku γ_r jest zazwyczaj stałą i może być także optymalizowana tak, aby w każdym kroku minimalizować funkcję błędu.

Sieci Neuronowe - dopasowywanie parametrów (*wag*)

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi}$$

$$\frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} x_{il}$$

δ, s - błędy

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}$$

Powyższe równanie nazywane jest równaniem propagacji wstecznej (ang. *back propagation equation*).

Sieci Neuronowe - dopasowywanie parametrów (*wag*)

Algorytm propagacji wstecznej:

- Krok wprzód: dla ustalonych wag obliczamy $\hat{f}_k(x_i)$
- Krok w tył: obliczamy błędy δ oraz s. Następnie obliczamy parametry w kolejnym kroku opadania gradientu.

Sieci Neuronowe - dopasowywanie parametrów (*wag*)

Jak uniknąć przeuczenia?

Sieci Neuronowe - dopasowywanie parametrów (*wag*)

Jak uniknąć przeuczenia?
Metoda *weight decay*:

$$R(\theta) + \lambda J(\theta)$$

gdzie $\lambda \geqslant 0$ oraz:

$$J(\theta) = \sum_{k,m} \beta_{km}^2 + \sum_{m,l} \alpha_{ml}^2$$

Sieci Neuronowe - dopasowywanie parametrów (*wag*)

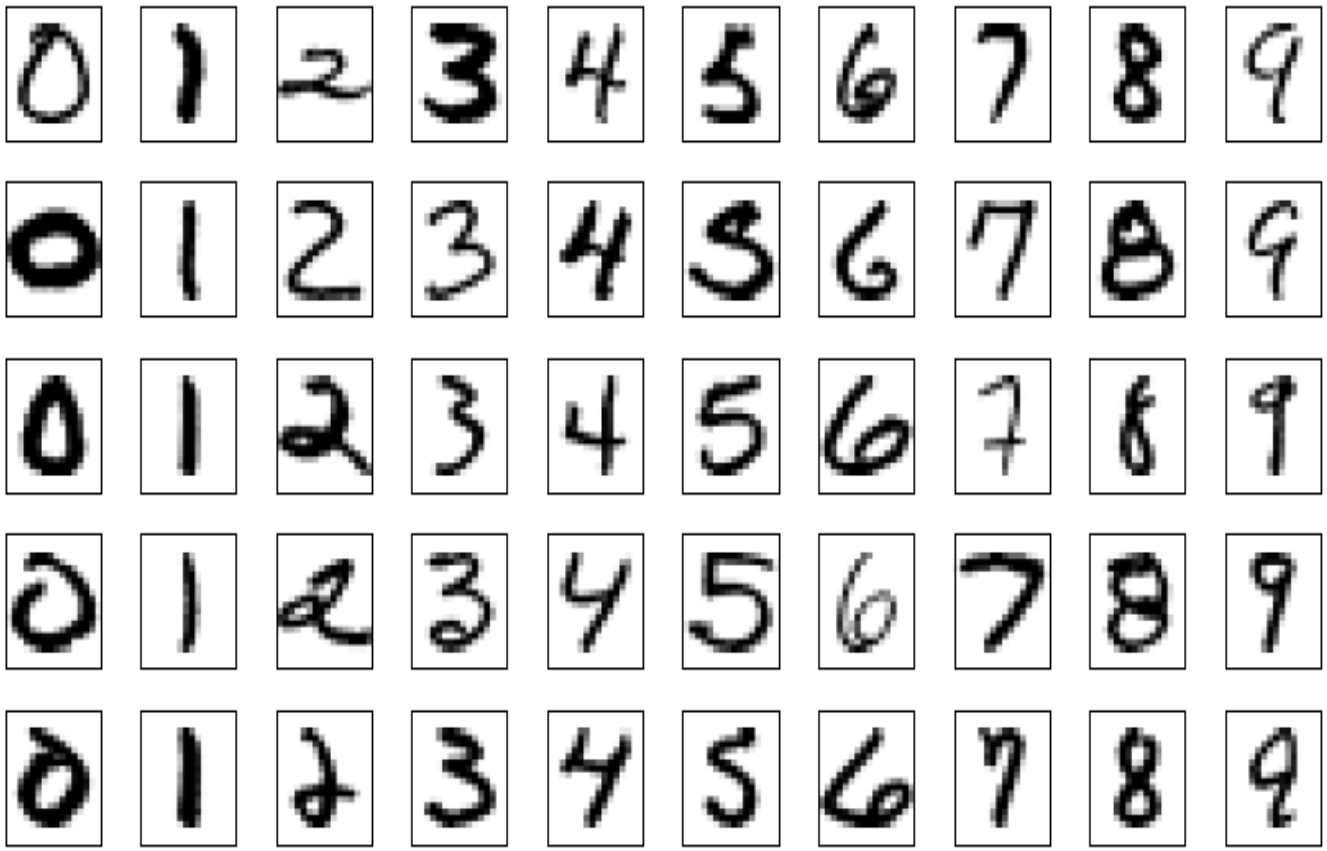
Jak uniknąć przeuczenia? (druga metoda)
Early stopping - ustalamy, że jeżeli model nie poprawi się po ustalonych r iteracjach o ustalony ϵ, to przerywamy gradient descent.

Sieci Neuronowe - dopasowywanie parametrów (*wag*)

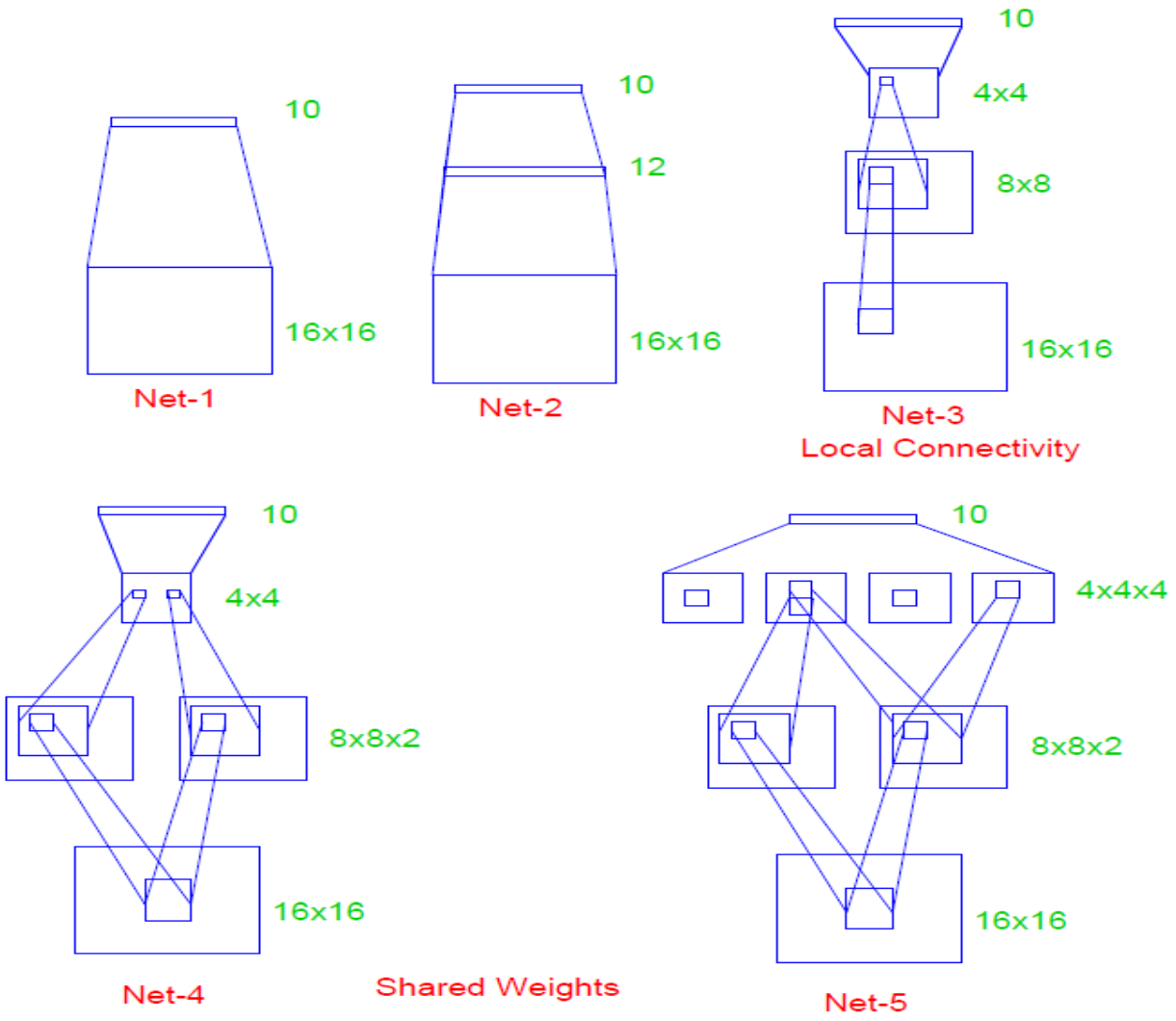
Ważne kwestie w dopasowywaniu wag:

- Wartości początkowe - losowe, z okolic zera
- Standaryzacja danych
- Liczba elementów warstwy ukrytej (M) - od 5 do 100, lepiej zacząć od większej

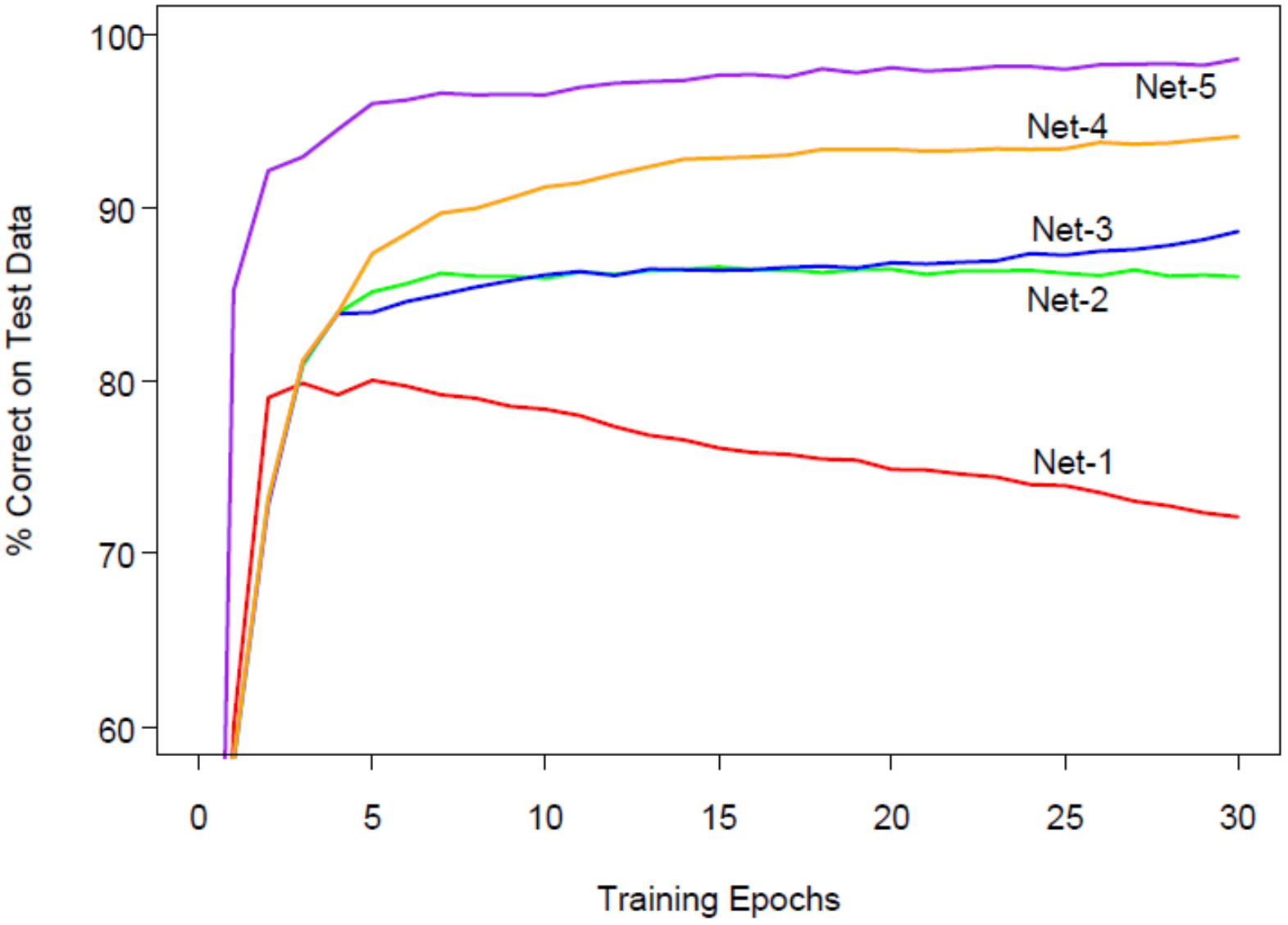
Przykład - ZIP Code Data



Przykład - ZIP Code Data



Przykład - ZIP Code Data



Dziękujemy za uwagę :)