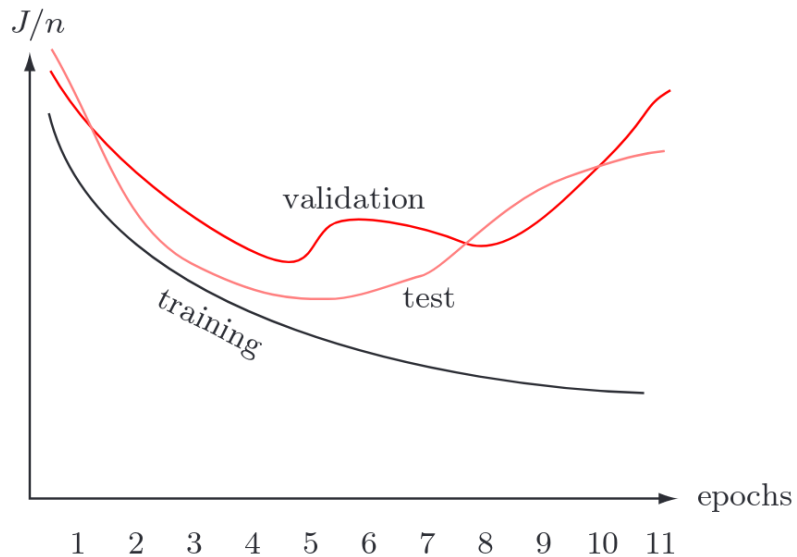
Generalizacja i regularyzacja sieci MLP

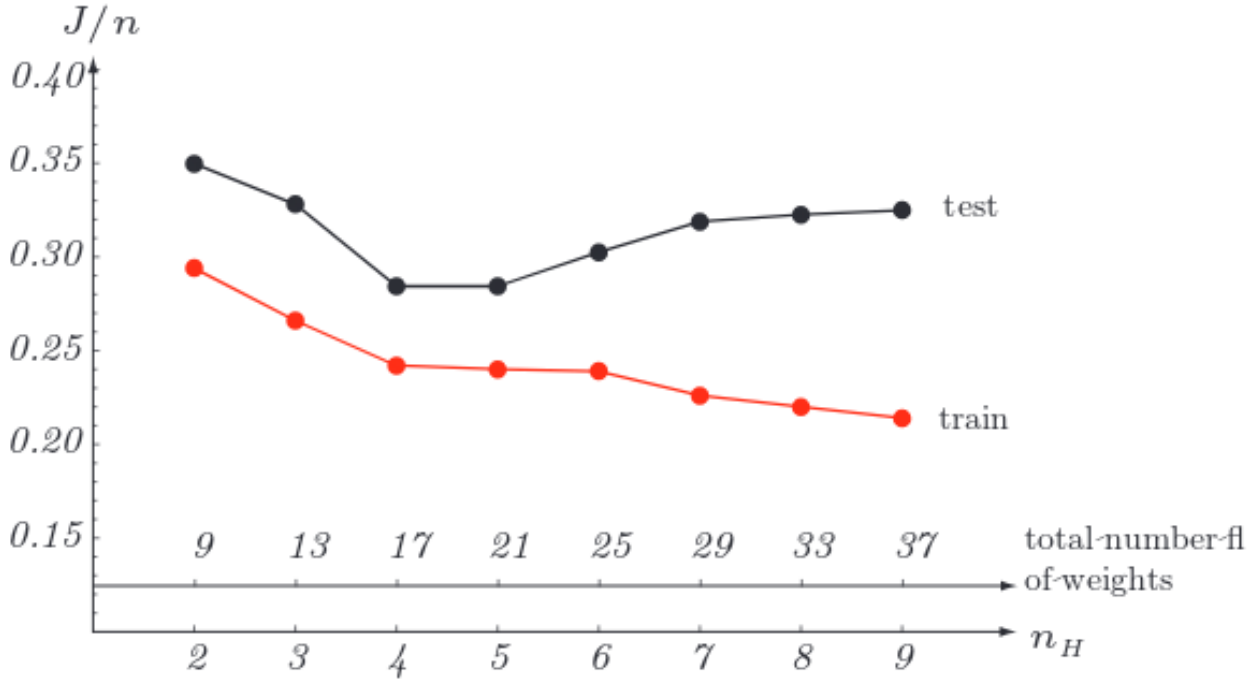
Generalizacja

- Wartość funkcji błędu dla zbioru treningowego może osiągnąć 0. Czy to znaczy, że mamy najlepszy model?
- **Generalizacja** jest celem uczenia - uogólnienie reguły, która wytworzyła dane. Dążymy do minimalizacji błędu na danych, które nie zostały użyte w treningu.
- Zbiór **walidacyjny** - wyodrębniony fragment danych (np. ze zbioru uczącego) pozwala na ocenę błędu generalizacji - przydatny przy określaniu punktu zatrzymania treningu



Duda, Hart, Pattern Recognition

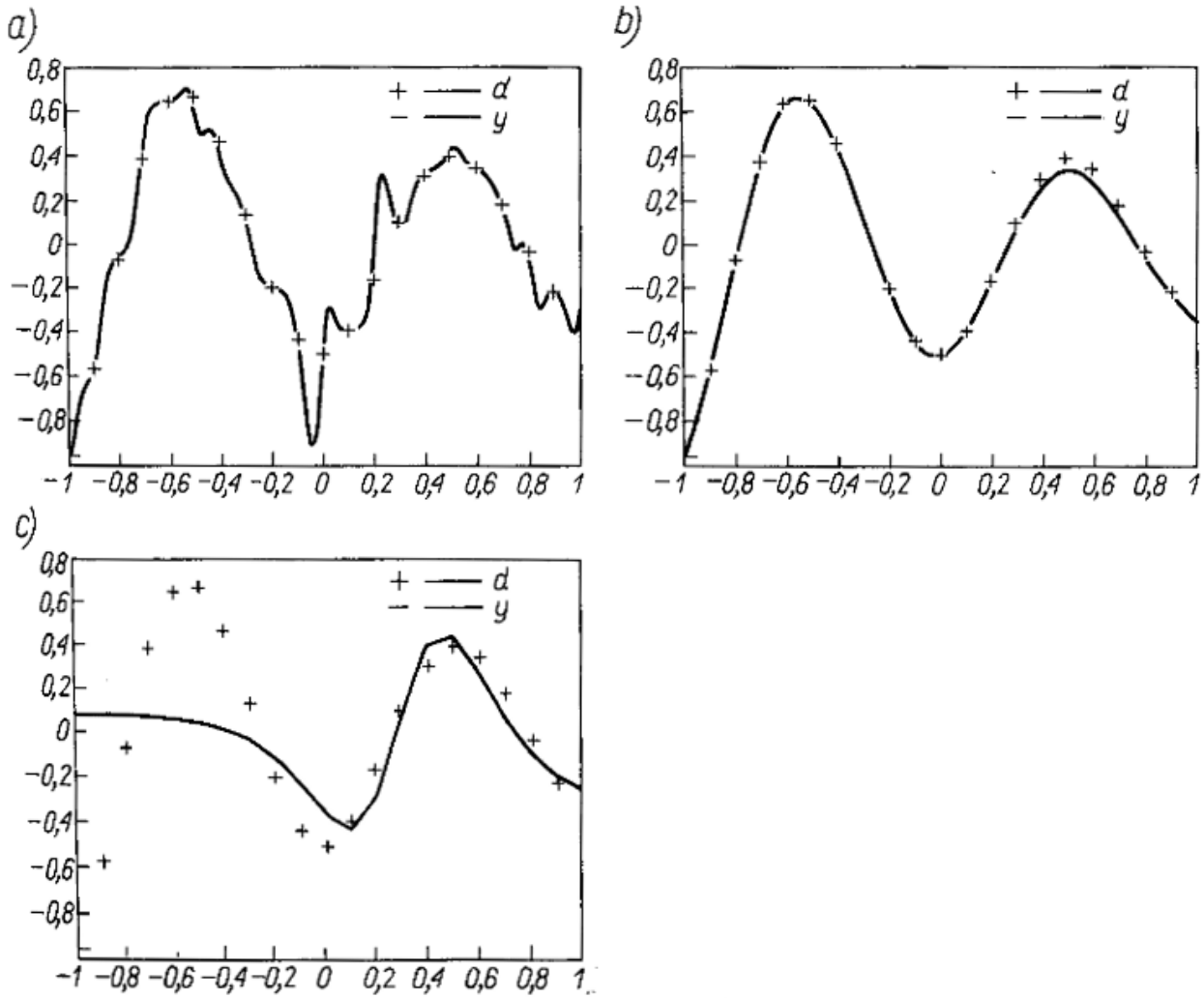
Rozmiar sieci



- ilość wag określa ilość stopni swobody - nie powinna przekraczać liczby przypadków uczących (np. $n/10$)
- za dużo wag - model uczy się na pamięć (przeuczenie)
- za mało wag - model zbyt prosty (niedouczony)

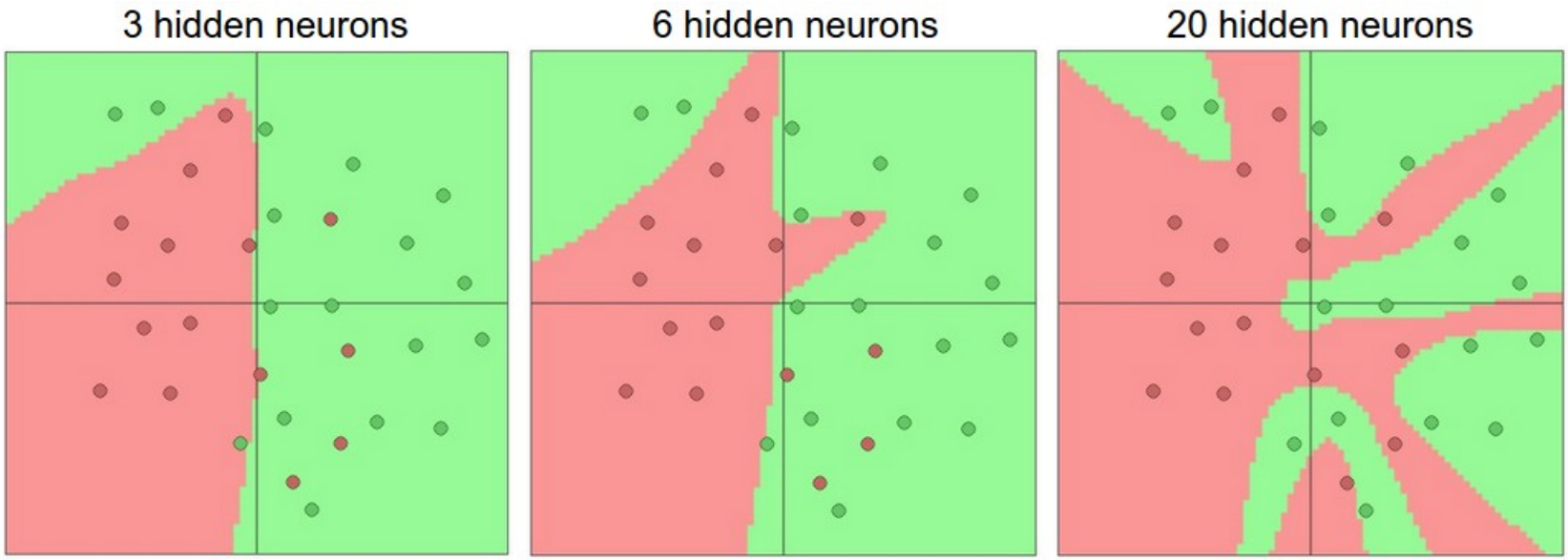
Rozmiar warstwy ukrytej

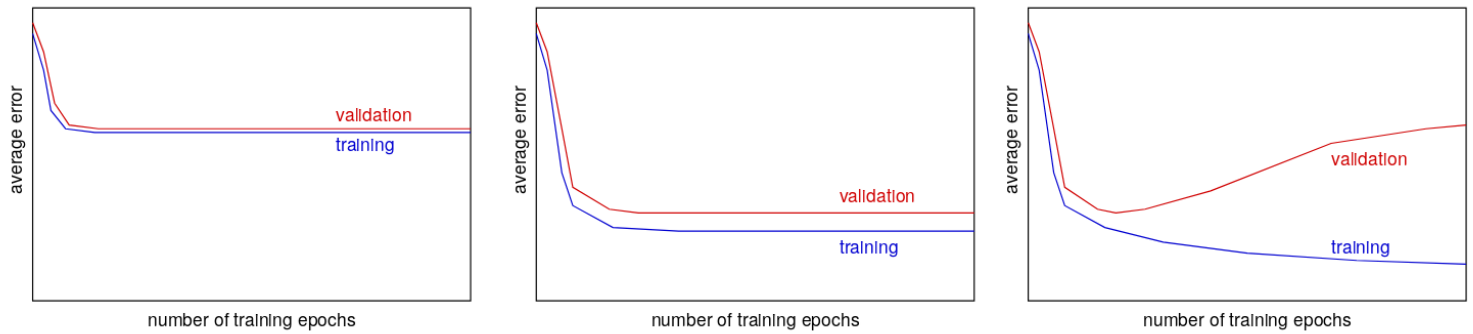
Zdolność uogólniania w problemie aproksymacji



Rozmiar warstwy ukrytej

Zdolność uogólniania w problemie klasyfikacji





- niedouczenie - błąd treningowy i walidacyjny pozostają duże
- przeuczenie - błąd walidacyjny rośnie a błąd treningowy maleje

Rys: Riedmiller, Machine Learning

Metody regularyzacji sieci neuronowych

Regularyzacja to metody, których celem jest poprawienie generalizacji

- dobór wielkości modelu (ilości neuronów i wag), preferowane są najprostsze rozwiązania (brzytwa Ockhama)
  - spośród wielu modeli wybierz ten o najmniejszym błędzie walidacyjnym i najmniejszej liczbie parametrów (neuronów)
  - **pruning** - usuwanie nadmiarowych połączeń lub neuronów z wytrenowanej sieci
  - metody **konstrukcyjne** (rosnące) - zacznij od najprostszego modelu i zwiększaj jego złożoność w kolejnych iteracjach
  - sieci **ontogeniczne** - rozrastające się i kurczące się w czasie treningu
- małe wartości wag są preferowane
  - wagi zerowe oznaczają brak połączenia
  - neurony sigmoidalne z małymi wagami są w przybliżeniu liniowe

Regularyzacja

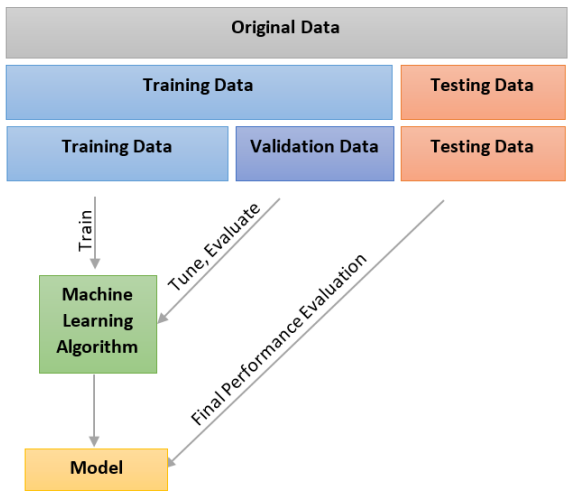
- modyfikacje funkcji kosztu wymuszające odpowiednią strukturę sieci, np. wymuszające minimalizację wartości wag

$$\bar{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda\Omega(\mathbf{w})$$

  gdzie $\lambda > 0$ steruje wielkością regularyzacji
- **wczesne zatrzymanie** treningu, zanim błąd walidacyjny wzrośnie
- zwiększenie liczby danych
- przetransformowanie danych do takiej postaci, która zwiększy szansę znalezienie optymalnego rozwiązania
- dodanie szumu do wag lub danych treningowych
- selekcja cech - wybór tylko istotnych zmiennych do treningu
- uśrednianie wyników z wielu modeli, np.: boosting, bagging

Trening i walidacja

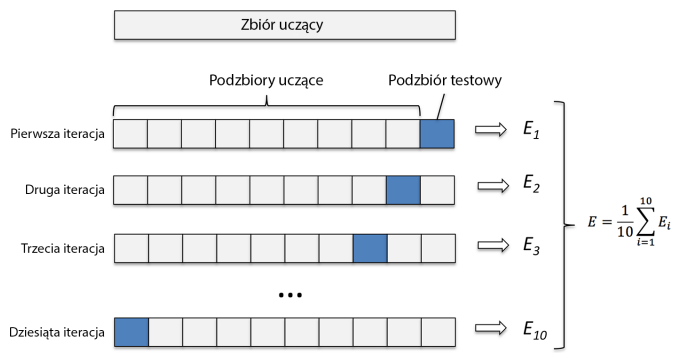
- uczenie sieci minimalizuje błąd treningowy a nie błąd generalizacji
- **zbiór walidacyjny** pozwala oszacować błąd generalizacji w trakcie treningu. Wydzielany ze zbioru treningowego, więc powoduje zmniejszenie liczby próbek uczących.
- **zbiór testowy** używany wyłącznie do ewaluacji nauczonego modelu



Rys. S. Mostafa Eissa, ☞ Introduction to Machine Learning , 2016

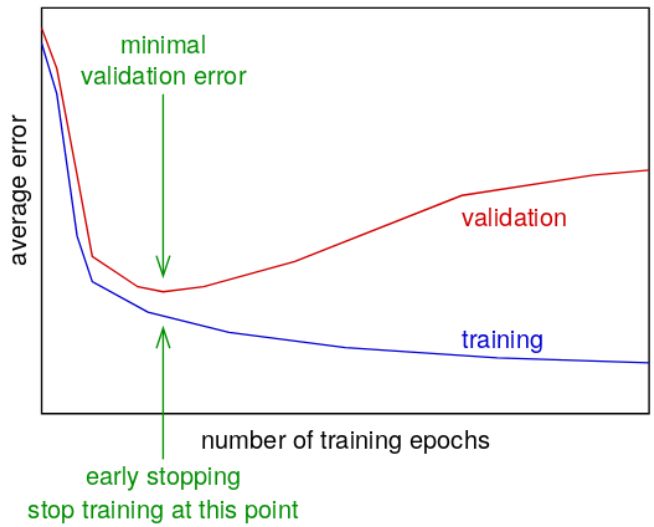
Walidacja krzyżowa (kroswalidacja)

- **Kroswalidacja** - metoda pozwalająca na oszacowanie błędu generalizacji i wariancji modelu poprzez powtarzanie treningu i ewaluacji na kolejnych podziałach zbioru uczącego
- k krotna kroswalidacja - dzieli zbiór na k części, gdzie $k-1$ jest używanych do treningu a reszta do testu
- kroswalidacja **leave-one-out** (LOO) - gdy N równe liczbie wektorów uczących, tj. trening odbywa się na $N-1$ przypadkach. Stosowane dla bardzo małych danych.
- średni błąd wyznaczany jest z k treningów



Rys. S. Raschka, Python Machine Learning, MIT, 2019

Wczesne zatrzymanie



Strategie:

- przerwij trening, gdy błąd walidacyjny zacznie rosnąć. Często wystarczy tylko kilka iteracji.
- trenuj przez N epok i zapamiętaj model o najmniejszym błędzie walidacyjnym

Rys: Riedmiller, Machine Learning

Dobór liczby neuronów

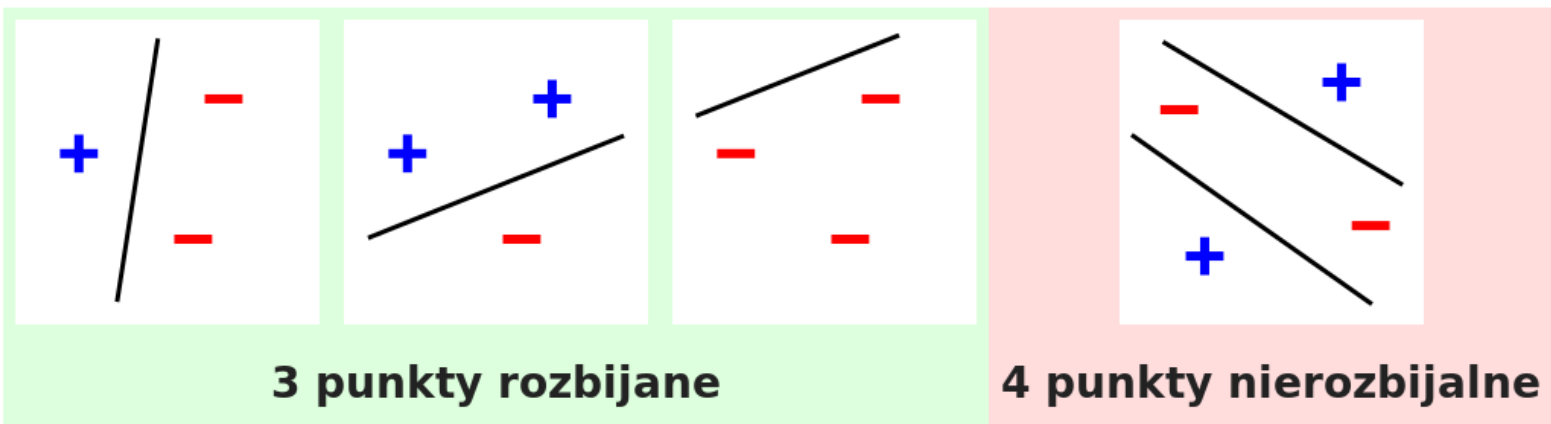
Probabilistyczne oszacowanie błędu generalizacji E_G sieci MLP klasyfikującej

$$E_G(\mathbf{w}) \leq E_T(\mathbf{w}) + \epsilon\left(\frac{N}{D}, E_T\right)$$

gdzie
E_T błąd treningowy
N - liczba wzorców uczących,
D - miara złożoności Vapnika-Chervonenkisa (VC dimension), zależy od złożoności sieci (ilości wag)
ϵ - przedział ufności, jego wartość maleje ze wzrostem $\frac{N}{D}$
Przeuczenie może zajść gdy $D > N$

Wymiar VC

Wymiar VC klasyfikatora to największa liczba punktów, którą klasyfikator jest w stanie rozbić, tj. podzielić przy dowolnej kombinacji etykiet (2^N możliwości w przypadku 2 klas)



Dla perceptronu (podział hiperpłaszczyzną) $D = 3$

Wymiar VC dla MLP

Oszacowanie wartości wymiary VC dla sieci MLP z binarna funkcja aktywacji

$$2\left\lceil \frac{N_h}{2} \right\rceil d \leq D \leq 2N_w(1 + \log N_n)$$

N_h - ilość neuronów ukrytych
d - ilość cech (wymiar przestrzeni wejściowej)
N_w - liczba wag
N_n - liczba wszystkich neuronów

- w praktyce $D \approx N_w$, dla funkcji sigmoidalnych $D \approx 2N_w$
- oszacowanie wymiaru VC pozwala określić minimalny wymiar zbioru uczącego
- liczba próbek powinna być kilkukrotnie (np. 10 razy) większa od wymiaru VC

Regularyzacja L_2

Regularyzacja L_2 (weight decay, regularyzacja Tichonova) - dąży do zmniejszenia wartości wag

$$\hat{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\lambda \sum_{ij} w_{ij}^2$$

gdzie $\lambda > 0$ decyduje o sile regularyzacji (typowo $\lambda = 0.01$ lub 0.001)
Zmniejszane są wszystkie wagi w trakcie uczenia. Korekta wag $\hat{w}_{ij}$ w stosunku do wersji nieregularyzowanej w_{ij}

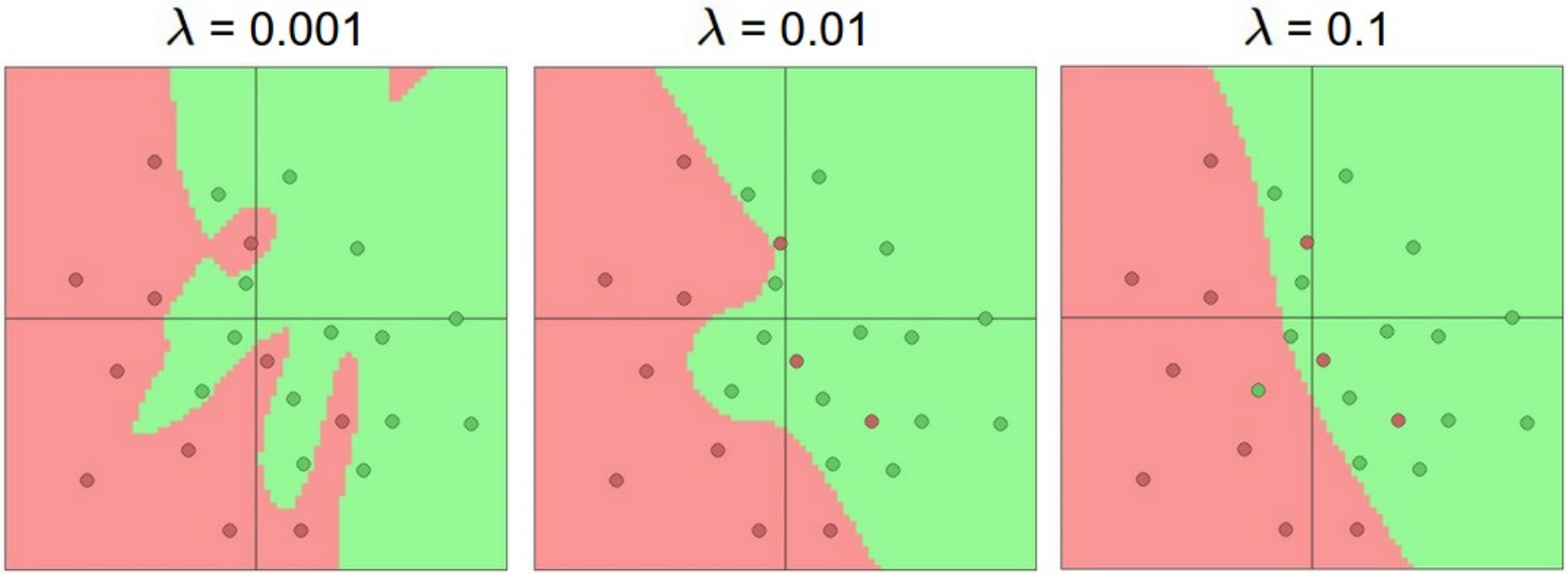
$$\hat{w}_{ij} = (1 - \lambda\eta) w_{ij}$$

$$\mathbf{w}(n+1) = \hat{\mathbf{w}}(n) - \eta \nabla E(\mathbf{w})$$

Możliwe modyfikacje:

- wagi, które zmalały poniżej pewnego progu mogą być usunięte
- usuwamy neurony dla których $\sum |w_i|$ jest bliskie zeru

Regularyzacja



Rys: http://cs231n.github.io/neural-networks-1/

Zmodyfikowany człon kary:

$$\hat{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\lambda \sum_{ij} \frac{w_{ij}^2}{1 + \sum_k w_{ik}^2}$$

Minimalizacja E powoduje:

- zmniejszenie wartości wag w_{ij}
- eliminację neuronów dla których $\sum_k |w_{ik}|$ jest bliskie zera

Regularyzowana zmiana wag zależna jest od wartości w_{ij}

$$\hat{w}_{ij} = \left(1 - \lambda\eta \frac{1 + 2\sum_{k \neq j} w_{ik}^2}{\left(1 + \sum_k w_{ik}^2\right)^2} \right) w_{ij}$$

- małe wartości wag w_{ik} prowadzące do i-tego neuronu zanikają
- duże wartości wag $\sum_k |w_{ik}|$ neuronu niwelują wpływ czynnika korekcyjnego

Regularyzacja L_1

$$\hat{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda \sum_{ij} |w_{ij}|$$

$$\nabla \hat{E}(\mathbf{w}) = \nabla E(\mathbf{w}) + \lambda \, \mathrm{sgn}(\mathbf{w})$$

regularyzacja wpływa na gradient ze stałą wartością (zależy tylko od znaku w_{ij})

$$\hat{w}_{ij} = \begin{cases} w_{ij} - \lambda\eta & \text{dla} \quad w_{ij} > 0 \\ w_{ij} + \lambda\eta & \text{dla} \quad w_{ij} < 0 \end{cases}$$

Regularyzacja L_1 daje rzadsze rozwiązanie od L_2, tzn. dąży do rozwiązania o mniejszej liczbie niezerowych wag

Metody wrażliwościowe redukcji sieci

- Eliminacja wag na podstawie jej wpływu na wartość błędu
- Wagi o małej wartości $|w_i|$ są mniej istotne od dużych wag
- Wrażliwość połączenia

$$S_{ij} = E - E(w_{ij} = 0)$$

- Połączenia o najmniejszej wrażliwości usuwa się po czym sieć jest douczana
- Usuwamy neuron dla którego wszystkie dochodzące (lub wychodzące) połączenia są wyzerowane

Optimal Brain Damage (LeCun)

Założenie: hesjan $\mathbf{H}$ jest diagonalnie dominujący, więc uwzględniamy tylko składową diagonalną.

Z rozwinięcia funkcji błędu E w szereg Tylora i przyjmując $\nabla E = 0$ (punkt zbieżności dla wytrenowanej sieci)

$$\Delta E = \frac{1}{2}\Delta \mathbf{w}^T \mathbf{H} \Delta \mathbf{w} = \frac{1}{2}\sum_i H_{ii} \Delta w_i^2$$

Miara ważności wagi

$$S_i = \frac{1}{2}\frac{\partial^2 E}{\partial^2 w_i} w_i^2$$

Wagi o najmniejszym S_i mogą być usunięte z wytrenowanej sieci.

Optimal brain Damage

1. Wytrenuj sieć dowolnym algorytmem uczenia
2. Wyznacz diagonalne elementy hesjanu oraz wartości wrażliwości S_i
3. Posortuj wagi zgodnie z rosnącymi wartościami S_i i obetnij te o najmniejszych wartościach.
4. Wróć do punktu 1

Redukcja neuronów o małej aktywności

- Neurony o niewielkiej aktywności (których sygnał wyjściowy nie zmienia się dla całego zbioru treningowego) można usunąć bez szkody dla generalizacji
- duża aktywność neuronu świadczy o jego przydatności
- modyfikacja funkcji kosztu z czynnikiem kary za zbyt małą aktywność neuronów (Y. Chaurin)

$$\hat{E} = E + \lambda \sum_{i=1}^{k} \sum_{j=1}^{n} e(\Delta_{ij}^2)$$

gdzie Δ_{ij} oznacza zmianę sygnału wyjściowego i-tego neuronu dla j-tego wektora treningowego

- przykładowy czynnik korelacyjny

$$e = \frac{1}{1 + \Delta_{ij}^2}$$

Architektury konstruktywne (rozrastające się)

- automatycznie dobierają złożoność modelu do złożoności problemu, rozrastając się od prostych do złożonych modeli
- strategia zachłanna - dodanie neuronu maksymalizuje zysk w pojedynczym kroku rozbudowy
- przedwczesne przerwanie uczenia nie musi być katastrofą
- (zazwyczaj) niski koszt obliczeniowy, np. w każdym cyklu douczamy tylko dodatkowy neuron, reszta połączeń jest „zamrożona”
- możliwość budowania sieci o różnorodnych funkcjach aktywacji
- przykład: algorytm wieżowy i piramidalny, korelacja kaskadowa

Korelacja kaskadowa (Fahlman i Labiere, 1989)

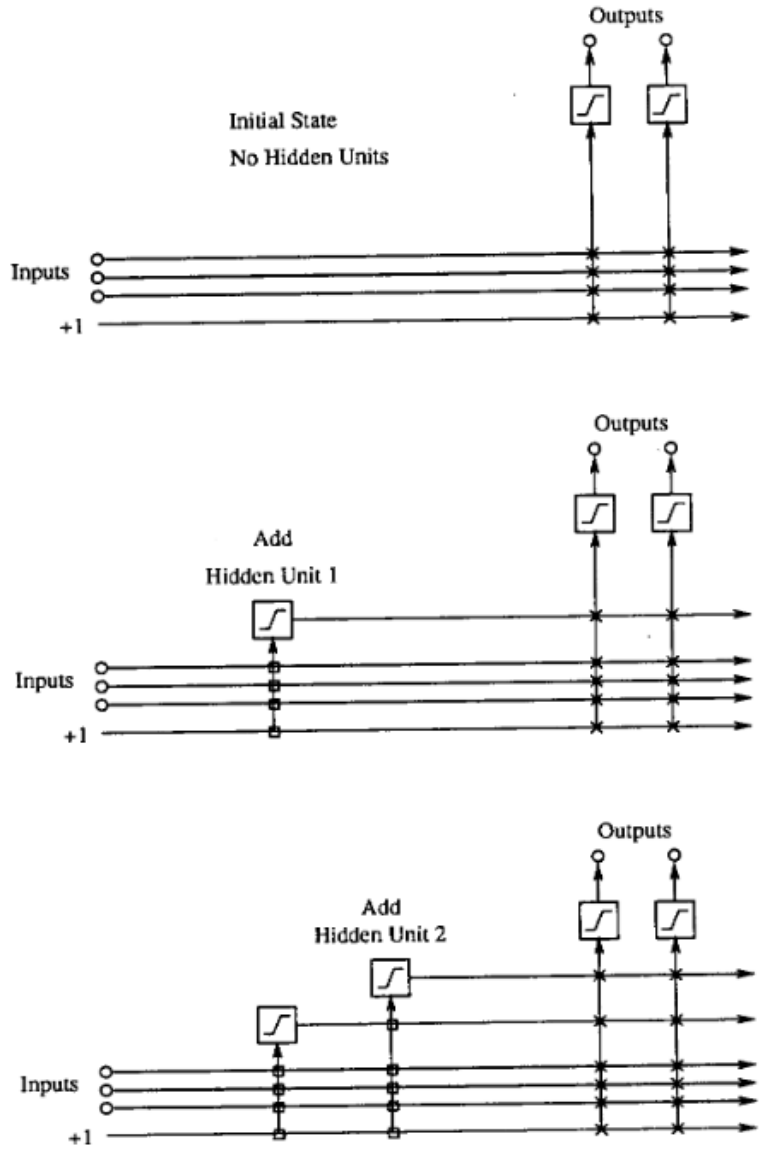
Sieć rozrastająca się:

- zaczynamy od treningu sieci bez warstwy ukrytej
- w kolejnych krokach dodawany jest neuron do warstwy ukrytej, wejścia neuronu połączone są ze wszystkimi wejściami sieci i wyjściami poprzednio dodanych neuronów ukrytych
- wagi neuronu dobierane są w procesie maksymalizacji korelacji nowego neuronu k z błędem wykazywanym przez neurony wyjściowe

$$C = \sum_{j=1}^{M} \left| \sum_{i=1}^{N} (o(\mathbf{x}_i) - \overline{o}) \left((y_{ij} - f_j(\mathbf{x}_i)) - (\overline{y}_{ij} - \overline{f}_j(\mathbf{x}_i)) \right) \right|$$

gdzie N - liczba wzorców, M - liczba wyjść sieci

Korelacja kaskadowa



Kaskadowa korelacja

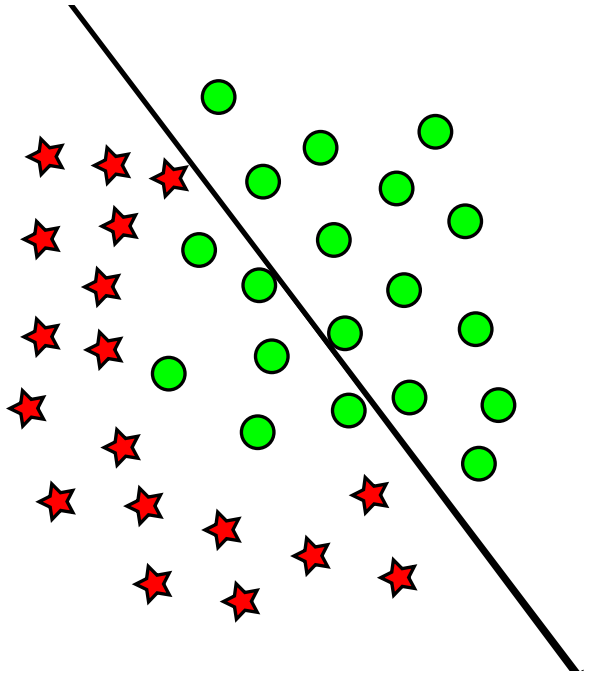
- wagi kandydata maksymalizujące korelację są „zamrażane” - douczaniu podlegają wyłącznie wagi wyjściowe sieci
- jeśli wyjścia neuronu kandydata są skorelowane dodatnio z błędem to w wyniku treningu wykształci wagi, które mają szansę zniwelować ten błąd
- w każdym kroku rozpatruje się zbiór kandydatów (od 5 do 10), mogą posiadać różne funkcje aktywacji (sigmoidalna, gaussowaka, itp.),
- kandydaci trenowani są równolegle konkurując ze sobą, wybiera się najlepszego (o największej korelacji)

Klasyfikator cząstkowy

Niech $\mathbb{Q}^+ \cup \mathbb{Q}^- \subset \mathbb{X} \quad \wedge \quad \mathbb{Q}^+ \cap \mathbb{Q}^- = \emptyset$

Klasyfikator cząstkowy f_c

daje odpowiedź $+1$ dla co najmniej jednego obiektu ze zbioru $\mathbb{Q}^+$ oraz odpowiedź -1 dla wszystkich elementów ze zbioru $\mathbb{Q}^-$



Niech $\mathbb{R} = \{(\mathbf{x}_i, y_i) : f_c(\mathbf{x}_i) = +1\} \subset \mathbb{Q}^+$

Dla funkcji logicznych (wejścia binarne) zawsze istnieje klasyfikator cząstkowy w postaci perceptronu

Ogólna konstruktywistyczna metoda sekwencyjna

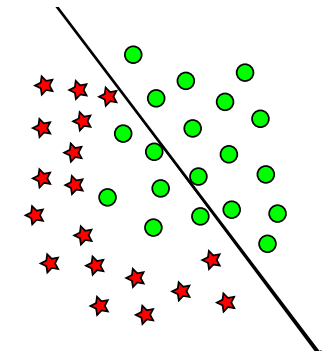
General Sequential Constructive Method (GSCM, Musseli, 1998)

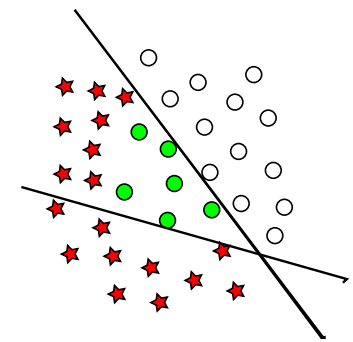
Algorytm 1 GSCM problem dwuklasowy

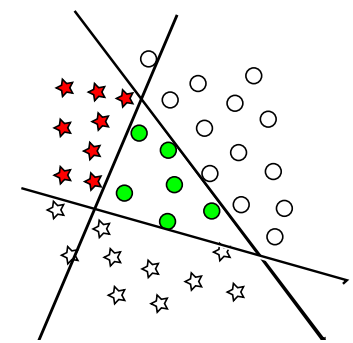
Input: Zbiór treningowy $\mathbb{D}$

Output: Sieć jednowarstwowa

1: $h \leftarrow 0$ ▷ pusta warstwa ukryta
2: **while** zbiór $\mathbb{D}$ zawiera przypadki z przeciwnych klas **do**
3: $h \leftarrow h + 1$
4: wybierz etykietę $d_h = \{-1, +1\}$
5: wytrenuj klasyfikator cząstkowy f_h dla klasy d_h
6: $\mathbb{D} \leftarrow \mathbb{D} \setminus \mathbb{R}_h$
7: Utwórz wynikową sieć zawierającą klasyfikatory cząstkowe w warstwie ukrytej







Przykładowe wagi połączeń do warstwy wyjściowej

$$u_0 = \sum_{j=1}^{h} u_j + d_{h+1}, \qquad u_j = d_j 2^{h-j} \quad \text{dla} \quad j = 1, \ldots, h$$

Uczenie klasyfikatora cząstkowego

Realizacja klasyfikatora cząstkowego za pomocą perceptronu, przykłady:

- algorytm nieregularnego podziału (Irregular Partitioning IPA, Marchand, Golea, 1993)
- algorytm zamiany etykiet (Target Switch TSA, Campbell, Vicente, 1995)

Algorytm nieregularnego podziału

1. startuje z pustego zbioru odseparowanych $\mathbb{R} = \emptyset$
2. dla kolejnych wektorów $\mathbf{x} \in \mathbb{Q}^+ \setminus \mathbb{R}$
   trenuje perceptron w poszukiwaniu hiperpłaszczyzny oddzielającej $\{\mathbf{x}\} \cup \mathbb{R}$ od $\mathbb{Q}^-$
   jeżeli istnieje separacja to $\mathbb{R} \leftarrow \mathbb{R} \cup \{\mathbf{x}\}$
3. zwróć ostatnio uzyskany perceptron

Uczenie klasyfikatora cząstkowego

Algorytm zamiany etykiet

1. wytrenuj perceptron dowolną metodą
2. jeżeli nie realizuje on klasyfikatora cząstkowego to znajdź najdalej oddalony od płaszczyzny decyzyjnej, błędnie klasyfikowany wektor z $\mathbb{Q}^+$ i zmień jego etykietę (przenieś do $\mathbb{Q}^-$)
3. powtarzaj aż do uzyskania perceptronu realizującego klasyfikator cząstkowy

Algorytmy rozrastające się - podsumowanie

- Problem z przeuczeniem, kiedy zaprzestać rozrost?
- Za dużą sieć można przyciąć ograniczając wpływ przeuczenia
- Uczenie zachłanne - dokładany pojedynczy neuron, koszt obliczeń zależy od czasu uczenia nowego neuronu, złożone struktury danych mogą nie zostać wykryte przez dodanie pojedynczego neuronu, zachłanna strategia nie zawsze jest najlepsza
- Algorytmy rosnące nie gwarantują najprostszych sieci
- Niektóre tworzą specyficzne architektury
- Sieci ontogeniczne - rosną i kurczą się w trakcie treningu
  - wyszukiwanie najlepszej architektury, stosuje się np. alg. genetyczne, ewolucyjne
  - zazwyczaj wymagające obliczeniowe

Rozmiar zbioru treningowego

- większa liczba danych zmniejsza ryzyko przeuczenia
- „There's is no data like more data"
- dodanie szumu do danych uodparnia sieć na drobne zmiany w przestrzeni wejściowej
- sztuczne zwielokrotnianie danych (*data augmentation*), często stosowane z dobrymi rezultatami
  - w klasyfikacji obrazów: zmiany kontrastów i nasycenia barw, przesunięcia, obroty, itp.
  - w analizie mowy: pogłos, szum otoczenia, zmiana głośności, przyspieszenie i zwolnienie mowy, itp..

Selekcja i ekstrakcja cech

- zmienne, które nie zawierają istotnych informacji mogą wpływać negatywnie na trening
- metody selekcji cech usuwają nieistotne zmienne zmniejszając tym samym rozmiar zbioru treningowego
- dodanie cech, które niosą wartościową informację poprawia generalizację
- istnieje wiele metod, najczęściej dobierane są do specyfiki problemu
- niezbalansowane klasy - częsty problem w klasyfikacji, standardowe funkcje błędu identycznie traktują błędy popełniane dla różnych klas. W ekstremalnym przypadku wszystkie przypadki z klasy mniejszościowej zostaną błędnie przypisane do klasy większościowej.