METODY INŻYNIERII WIEDZY
METHODS OF KNOWLEDGE ENGINEERING

Sztuczne Sieci Neuronowe

Adrian Horzyk

horzyk@agh.edu.pl

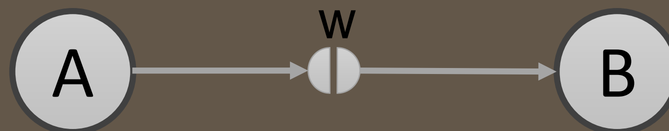
Akademia Górniczo-Hutnicza w Krakowie

Mózg i Neurony

Mózgi i biologiczne neurony zainspirowały naukowców do opracowania różnych modeli sztucznych neuronów i ich sieci.

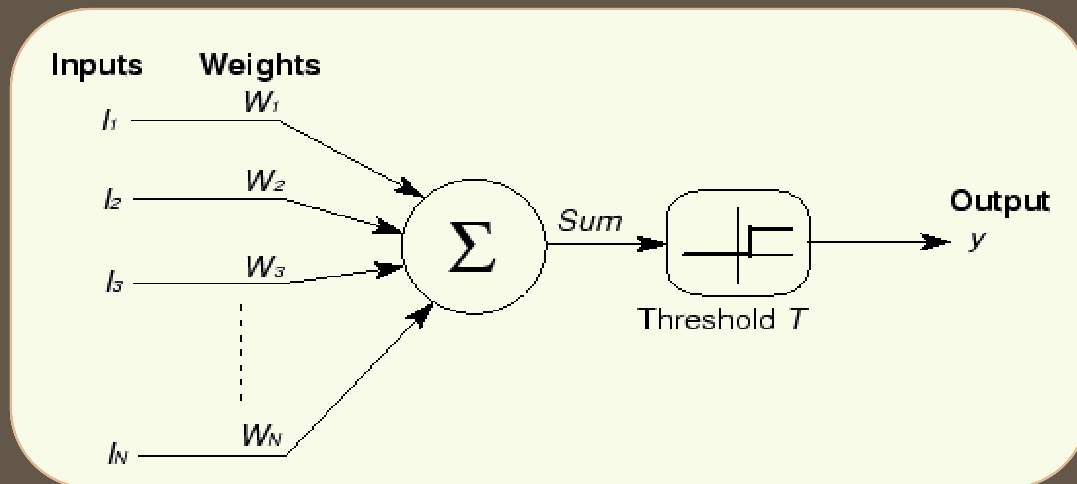
Reguła uczenia Hebba stwierdza, że *„gdy akson komórki (neuronu) A jest wystarczająco blisko, aby pobudzić komórkę (neuron) B i wielokrotnie lub uporczywie bierze udział w jej aktywacji, zachodzi pewien proces wzrostu lub przemiana metaboliczna w jednej lub w obu komórkach tak, że wydajność komórki A [w], jako jednej z komórek biorących udział w aktywacji komórki B, jest zwiększona"*.

[D. O. Hebb, 1949.]



Zasada ta zakłada, że połączenia między neuronami są ważone, a wartość wagi [w] jest funkcją liczby pobudzeń aktywowanego neuronu postsynaptycznego za pośrednictwem połączeń i synaps, przez które przechodzi pobudzenie i które biorą udział w aktywacji neuronu postsynaptycznego.

Model neuronów McCulloch-Pittsa (1. generacji) implementuje tylko najbardziej podstawowy mechanizm ważonej integracji (sumowania) bodźców wejściowych i funkcji aktywacji progowej, pomijając kwestie czasu, plastyczności i innych ważnych czynników. Ten prosty model umożliwia adaptację prostych sieci neuronowych.



Reguła uczenia Hebba i Oja

Reguła uczenia Hebba definiuje wagę połączenia od neuronu j do neuronu i:

$$w_{ij} = x_i \cdot x_j$$

gdzie x_i i x_j to wejścia równe 0 lub 1 dla neuronów i oraz j (i≠j), aktualizowane po każdej prezentacji wzorca treningowego lub po zaprezentowaniu wszystkich (p) wzorców:

$$w_{ij} = \frac{1}{p} \sum_{k=1}^{p} x_i^k \cdot x_j^k$$

gdzie x_i^k jest k-tym wejściem i-tego neuronu.
Uogólniona reguła uczenia Hebba jest zdefiniowana dla odpowiedzi postsynaptycznej y_n:

$$\Delta \boldsymbol{w} = \boldsymbol{w_{n+1}} - \boldsymbol{w_n} = \eta \cdot \boldsymbol{x_n} \cdot y_n$$

Reguła uczenia Oja jest szczególnym przypadkiem uogólnionego algorytmu Hebba z jednym neuronem, który jest wyraźnie stabilny, w przeciwieństwie do zasady Hebba. Zmiana wag presynaptycznych w dla danej odpowiedzi wyjściowej neuronu y_n w reakcji na jego wejście x_n jest dokonywana według:

$$\Delta \boldsymbol{w} = \boldsymbol{w_{n+1}} - \boldsymbol{w_n} = \eta \cdot \boldsymbol{w_n}(\boldsymbol{x_n} - y_n \cdot \boldsymbol{w_n})$$

gdzie η jest współczynnikiem uczenia, który może się zmieniać w czasie procesu adaptacji sieci, a n definiuje numer iteracji czasu upływającego w sposób dyskretny: 1, 2, 3, ...

Model Neuronu McCulloch-Pitts'a

Model ten jest również znany jako **liniowa bramka progowa** (***linear threshold gate***) wykorzystująca **liniową funkcję skokową** (***linear step function***), ponieważ klasyfikuje zestaw wejść jedynie do dwóch różnych klas.

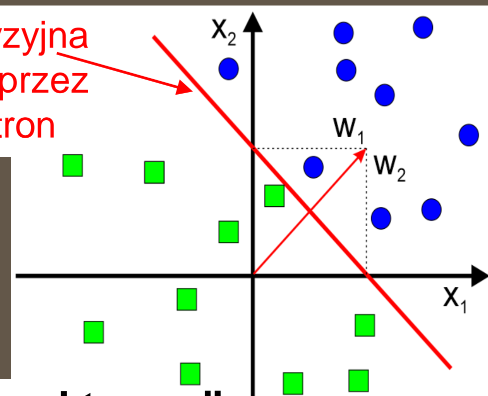
Model ten wykorzystuje skokową **funkcję aktywacji**, zwaną też **twardym przełączeniem** (***hard-switch***) **f**, która powoduje, że neuron ulega aktywacji, gdy ważona suma S bodźca wejściowego X osiąga **próg aktywacji** (***threshold***) θ.

$$y = f(S) = \begin{cases} 0 & S < \theta \\ 1 & S \geq \theta \end{cases}$$

$$S = \sum_{k=1}^{K} \mathbf{x}_k \cdot \mathbf{w}_k$$

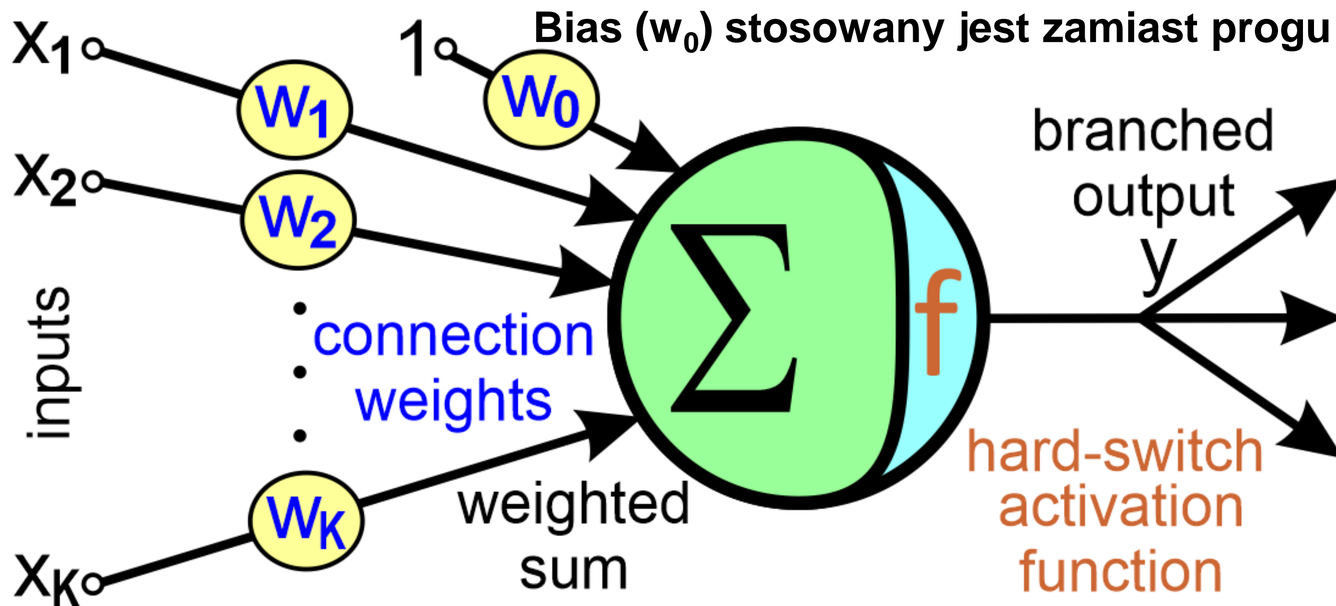
$$X = [\mathbf{x}_1, \dots, \mathbf{x}_k]$$

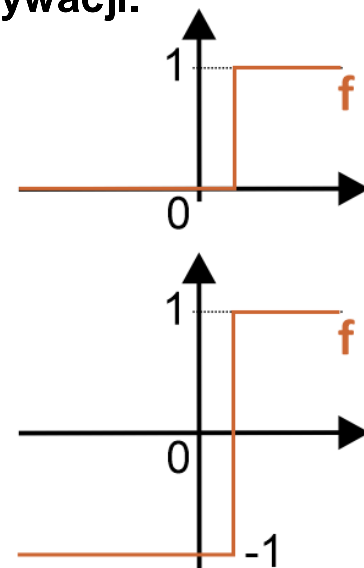
Perceptron z przełączeniem skokowym



Model ten wykorzystuje **skokową funkcję aktywacji**, która służy jako twardy przełącznik między dwoma stanami: {0, 1} lub {-1, 1} zgodnie z używaną funkcją aktywacji f:

Bias (w_0) stosowany jest zamiast progu aktywacji.





$$y = f(S) = \begin{cases} 0 & S < 0 \\ 1 & S \geq 0 \end{cases} \qquad S = \sum_{k=0}^{K} \mathbf{x}_k \cdot \mathbf{w}_k \qquad X = [\mathbf{x}_1, \ldots, \mathbf{x}_k] \qquad \mathbf{x}_0 = 1$$

Uczenie Perceptronu Skokowego

Uczenie nadzorowane perceptronu skokowego dla danego zestawu danych treningowych składającego się z wzorców treningowych $\{(X_1, d_1), \ldots, (X_N, d_N)\}$, gdzie d_n jest pożądaną (uczoną) wartością wyjściową dla wejściowego wektora treningowego X_n, jest zdefiniowane następująco:

1. Losowo wybierz małe początkowe wagi w zakresie [-0,1; 0,1].
2. Stymuluj perceptron kolejnym wektorem treningowym X_n, gdzie $n = 1, \ldots, N$.
3. Oblicz sumę ważoną S i wartość wyjściową $y_n = f(S)$.
4. Porównaj obliczoną wartość wyjściową y_k z pożądaną (uczoną) wartością wyjściową d_n.
5. Jeżeli $y_n \neq d_n$ wtedy $\Delta w_k = (d_n - y_n) \cdot x_k$ w odwrotnym przypadku nie rób nic w przypadku uczenia online, kiedy dokonujemy aktualizacji wag po każdym zaprezentowanym wzorcu uczącym, albo oblicz $\Delta w_k \mathrel{+}= 1/N \cdot \sum_{n=1,\ldots,N} (d_n - y_n) \cdot x_k$ po prezentacji każdego wzorca w przypadku uczenia offline (wsadowego), lecz aktualizację wag $w_k \mathrel{+}= \Delta w_k$ dokonujemy po prezentacji wszystkich wzorców uczących $k=0,\ldots,K$.
6. Jeśli średni błąd iteracji $E = 1/N \cdot \sum_{n=1,\ldots,N} |d_n - y_n|$ jest większy niż maksymalny błąd określony przez użytkownika wtedy rozpocznij następną iterację, przechodząc do kroku 2. Algorytm może się też zatrzymać się po określonej maksymalnej liczbie iteracji, po której rezygnujemy z dalszej adaptacji.

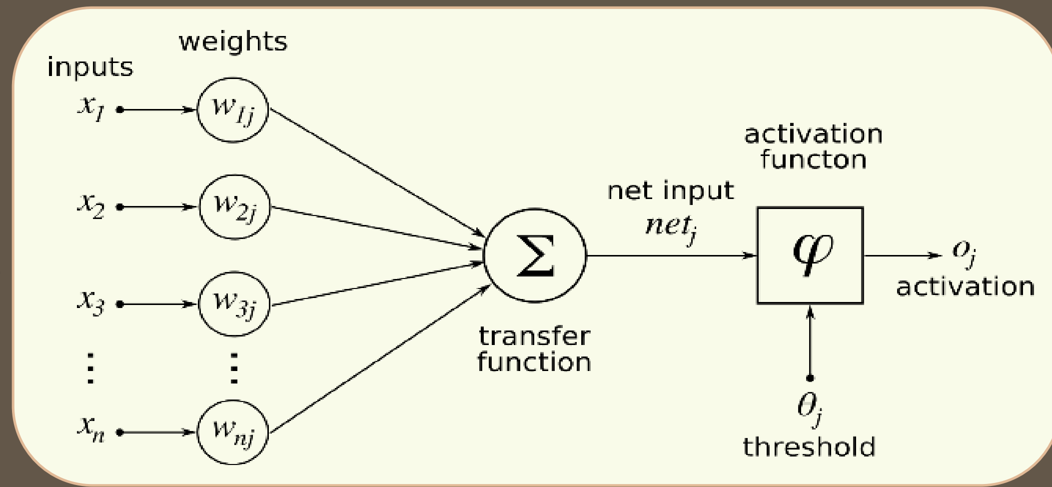
Perceptron jedno i wielowarstwowy

Grupa perceptronów zorganizowanych w pojedynczą warstwę może być użyta do wielo-klasyfikacji (*multi-classification*), co oznacza klasyfikację wektorów wejściowych do kilku klas jednocześnie. Taka grupa perceptronów nazywana jest jednowarstwową siecią perceptronową (*single-layer perceptron network*), która ma pewne ograniczenia swoich zdolności adaptacyjnych. Z tego powodu zwykle używamy perceptronu wielowarstwowego (*MLP - multi-layer perceptron*), tj. sieci składającej się z kilku warstw zawierających różną liczbę perceptronów. Pierwsza warstwa nazywana jest warstwą wejściową (*input layer*), ostatnia nazywana jest warstwą wyjściową (*output layer*), a wszystkie warstwy pomiędzy nimi nazywane są warstwami ukrytymi (*hidden layers*), jak pokazano na rysunku.

Mózg i Neurony

Mózgi i biologiczne neurony zainspirowały naukowców do opracowania różnych modeli sztucznych neuronów i ich sieci.

Modele neuronów wykorzystujące nieliniowe ciągłe funkcje aktywacji (2. generacji) umożliwiają budowanie wielowarstwowych sieci neuronowych (np. MLP) i dostosowanie takich sieci do bardziej złożonych (nieliniowych) zadań obliczeniowych.



Wykorzystanie skokowych funkcji aktywacji (*hard-switch*) ograniczało możliwości pierwszych neuronów, więc matematycy zaproponowali użycie nieliniowych funkcji aktywacji z miękkim przełączeniem (*soft-switch*) pomiędzy stanami {0, 1} lub {-1, 1}, które były różniczkowalne, co pozwoliło to na zastosowanie metod gradientowych do adaptacji (treningu) takich neuronów.

Ten typ modeli neuronów jest obecnie najczęściej stosowany, jednak ma poważne ograniczenia i niezgodności w stosunku do biologicznych pierwowzorów, co nie umożliwia zachowanie pewnych cech biologicznych sieci neuronowych.

Perceptron z miękkim przełączeniem

Model ten wykorzystuje ciągłą funkcję aktywacji (np. sigmoidalną lub tangens hiperboliczny), która służy jako miękki przełącznik (*soft-switch*) między dwoma stanami: (0, 1) lub (-1, 1) zgodnie z używaną funkcją f:

inputs: x_1, x_2, …, x_K, 1

connection weights: w_1, w_2, …, w_K, w_0

weighted sum: Σ

soft-switch activation function: f

branched output: y

$$y = f(S) = \frac{1}{1+e^{-\beta \cdot S}} \in (0,1) \quad OR \quad y = f(S) = \frac{2}{1+e^{-\beta \cdot S}} - 1 \in (-1,1) \quad OR \quad y = f(S) = \tanh(\beta \cdot S) \in (-1,1)$$

Reguła Delta do Adatapcji

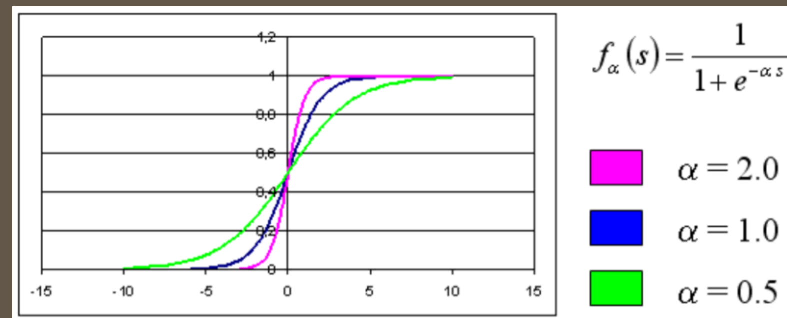
Reguła delta (*delta rule*) wykorzystuje neurony z miękkim przełączeniem, których funkcje aktywacji są ciągłe, aby umożliwić ich różniczkowanie. Delta zdefiniowana jest jako różnica między pożądanymi d_n i obliczonymi y_n wyjściami sieci: $\delta_n = d_n - y_n$. Ta reguła może wynikać z minimalizacji funkcji błędu średnio-kwadratowej (*mean square error function*):

$$Q = \frac{1}{2}\sum_{n=1}^{N}(d_n - y_n)^2 \qquad gdzie \qquad y_n = f(S) \qquad S = \sum_{k=0}^{K} \mathrm{x}_k \cdot \mathrm{w}_k$$

Korekta wagi dla różniczkowalnej funkcji aktywacji f jest obliczana według:

$$\Delta \mathrm{w}_k = \eta \cdot \delta_n \cdot f'(S) \cdot \mathrm{x}_k \qquad where \quad \delta_n = d_n - y_n$$

gdzie f' jest pochodną funkcji f.



Gdy funkcja aktywacji $f(S) = \frac{1}{1+e^{-\alpha S}}$ jest sigmoidalna, wtedy otrzymamy następujące wyrażenie do aktualizacji wartości wag:

$$\Delta \mathrm{w}_k = \eta \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot \mathrm{x}_k \qquad gdzie \quad \delta_n = d_n - y_n$$

Wprowadzenie do algorytmu wstecznej propagacji błędów wielowarstwowych perceptronów

Ciągły i miękki charakter funkcji sigmoidalnej pozwala osiągnąć różniczkowalność w całym zakresie zmienności. Jest to konieczne dla popularnych algorytmów uczenia się, takich jak wsteczna propagacja błędów (*backpropagation*) lub uczenie konwolucyjne (*convolution learning*).

Ze względu na ograniczone zdolności adaptacyjne jednowarstwowych sieci perceptronowych używamy wielowarstwowych sieci perceptronowych (*MLP - multi-layer perceptron network*), które składają się z kilku warstw neuronów mogących zawierać różną liczbę neuronów.

Perceptron wielowarstwowy nie może korzystać z liniowego funkcji aktywacji, ponieważ zawsze można go uprościć do jednowarstwowej liniowej sieci perceptronowej.

Sieci MLP mogą być szkolone przy użyciu algorytmu propagacji wstecznej błędów (*BP -backpropagation algorithm*), który radzi sobie z ograniczeniami uczenia sieci jednowarstwowych wskazanymi przez Minsky'ego i Paperta w 1969 roku.

Algorytm BP jest jednak często zbyt wolny, aby zaspokoić potrzeby związane z uczeniem maszynowym, lecz został zrehabilitowany w 1989 r., kiedy stał się silnikiem uczenia się znacznie szybszych i obecnie bardzo popularnych głębokich sieci konwolucyjnych (*CNN - Convolutional Neural Networks*). Dlatego ten algorytm ma kluczowe znaczenie dla różnych architektur neuronalnych!

Algorytm Wstecznej Propagacji Błędów

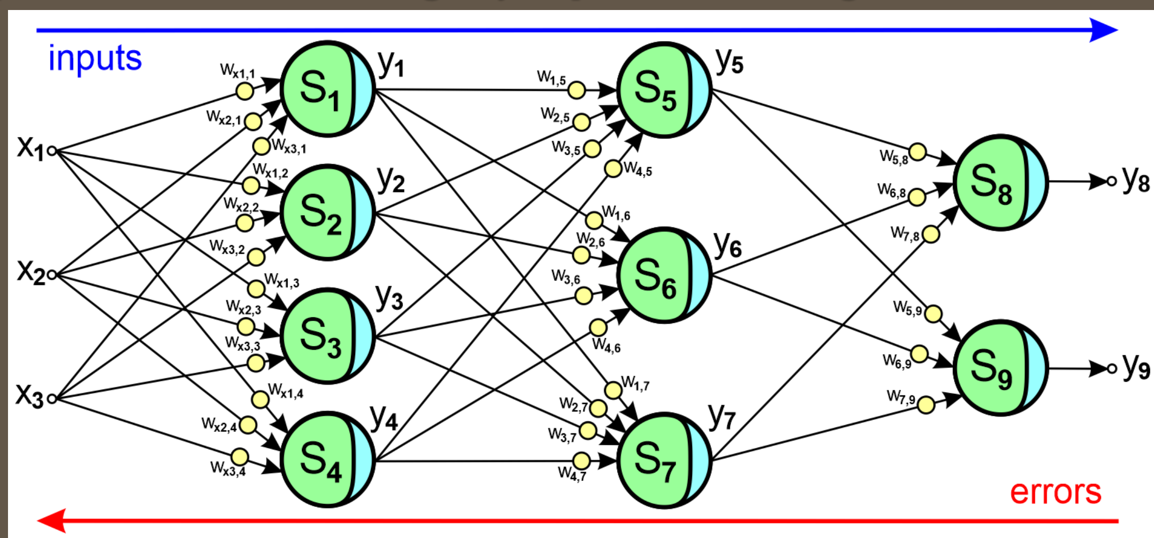
Algorytm propagacji wstecznej (BP) obejmuje dwie główne fazy:

1. Faza propagacji wejść propaguje sygnały wejściowe poprzez wszystkie ukryte warstwy do neuronów warstwy wyjściowej. W tej fazie neurony sumują ważone sygnały wejściowe dochodzące z neuronów w poprzedniej warstwy.
2. Faza propagacji błędu propaguje błędy (wartości delta) obliczone na wyjściach sieci neuronowej. W tej fazie neurony sumują ważone błędy (wartości delta) pobrane z neuronów w następnej warstwy.

Obliczone korekty wag są używane do aktualizacji wag następująco:

- obliczone korekty są wprowadzane natychmiast po ich obliczeniu podczas treningu online,
- średnia wartość wszystkich obliczonych korekt każdej wagi jest wprowadzana po zakończeniu całego cyklu treningowego dla wszystkich wzorców treningowych podczas treningu offline.

Algorytm ten jest wykonywany do momentu, gdy średni błąd kwadratowy $\boldsymbol{Q}$, obliczony dla wszystkich próbek treningowych będzie mniejszy niż pożądana wartość lub algorytm jest wykonywany do określonej maksymalnej liczby cykli uczących.



Algorytm Wstecznej Propagacji Błędów

Najpierw, wejścia x_1, x_2, x_3 stymulują neurony w pierwszej warstwy ukrytej. Neurony obliczają ważone sumy wejść S_1, S_2, S_3, S_4, i obliczają wartości wyjściowe y_1, y_2, y_3, y_4, które stają się wejściami dla neuronów kolejnej warstwy ukrytej:

$$S_n = \sum_{k=1}^{3} x_k \cdot w_{x_k,n} \qquad y_n = f(S_n)$$

Algorytm Wstecznej Propagacji Błędów

Następnie, wyjścia y_1, y_2, y_3 ,y_4 stymulują neurony z drugiej warstwy ukrytej. Neurony obliczają sumy ważone wejść S_5, S_6, S_7, i obliczają wartości wyjściowe y_5, y_6, y_7, które stają się wejściami dla neuronów warstwy wyjściowej:

$$S_n = \sum_{k=1}^{4} y_k \cdot \mathbf{w}_{k,n} \qquad y_n = f(S_n)$$

Algorytm Wstecznej Propagacji Błędów

W końcu, wyjścia y_5, y_6, y_7 stymulują neurony warstwy wyjściowej.
Neurony obliczają sumy ważone wejść S_8 i S_9 , i obliczają wartości wyjściowe y_8, y_9 które są wyjściami sieci neuronowej i odpowiedzią sieci:

$$S_n = \sum_{k=5}^{7} y_k \cdot \mathbf{w}_{k,n} \qquad y_n = f(S_n)$$

Algorytm Wstecznej Propagacji Błędów

Następnie, wyjścia neuronów wyjściowych y_8, y_9 są porównywane do pożądanych (uczonych) wyjść sieci d_8, d_9 i błędy δ_8, δ_9 są wyznaczane. Te błędy są propagowane wstecz przez sieć w celu obliczenia korekty wag połączonych neuronów:

$$\delta_n = d_n - y_n$$

Algorytm Wstecznej Propagacji Błędów

Błędy δ_8 i δ_9 są następnie stosowane do obliczenia korekty wag dla sygnałów wejściowych y_5, y_6, y_7, i propagowane wstecz poprzez połączenia do neuronów warstwy poprzedniej w celu obliczenia błędów δ_5, δ_6, δ_7:

$$\Delta \mathbf{w}_{k,n} = -\eta \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot y_k \qquad \delta_k = \sum_{n=8}^{9} \delta_n \cdot \mathbf{w}_{k,n} \cdot (1 - y_n) \cdot y_n$$

Algorytm Wstecznej Propagacji Błędów

Następnie, błędy δ_5, δ_6 i δ_7 są następnie stosowane do obliczenia korekty wag dla sygnałów wejściowych y_1, y_2, y_3, y_4, i propagowane wstecz poprzez połączenia do neuronów warstwy poprzedniej w celu obliczenia błędów δ_1, δ_2, δ_3, δ_4:

$$\Delta \mathbf{w}_{k,n} = -\eta \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot y_k \qquad \delta_k = \sum_{n=5}^{7} \delta_n \cdot \mathbf{w}_{k,n} \cdot (1 - y_n) \cdot y_n$$

Algorytm Wstecznej Propagacji Błędów

W końcu, błędy δ_1, δ_2, δ_3, δ_4 są następnie stosowane do obliczenia korekty wag dla sygnałów wejściowych x_1, x_2, x_3:

$$\Delta \mathbf{w}_{k,n} = -\eta \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot y_k$$

Inicjalizacja sieci i parametry uczenia

Liczba neuronów warstwy ukrytej powinna być większa niż w warstwie wejściowej, aby umożliwić sieci tworzenie reprezentacji różnych kombinacji i funkcji. W przypadku prostych problemów jedna lub dwie warstwy ukryte są zwykle wystarczające.

Liczba neuronów w kolejnych warstwach zwykle maleje. Mogą być również określane eksperymentalnie lub przy użyciu metod ewolucyjnych czy genetycznych.

Inicjalizacja wag odbywa się poprzez ustawienie każdej wagi na wartość losową o niewielkiej wartości wybranej z puli liczb losowych, np. z zakresu od -0,1 do +0,1.

Współczynnik (szybkości) uczenia η powinna być stopniowo dostosowywana ($\eta < 1$), biorąc pod uwagę wymagania stabilności i zbieżności (zwykle zaczynamy od $\eta = 0.1$). Jednakże, ponieważ zbieżność jest zazwyczaj dość szybka, gdy błąd staje się bardzo mały, zaleca się przywrócenie początkowej wartości tego współczynnika przed kontynuowaniem. Istnieje wiele strategii i metod zmiany tego kluczowego parametru w procesie szkolenia.

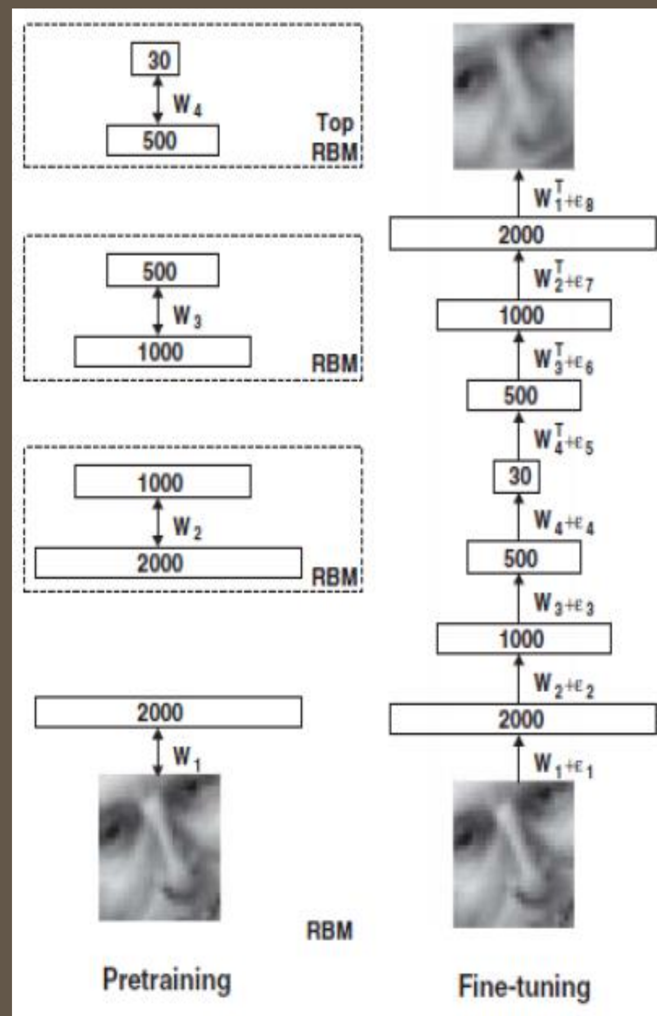
Aby uniknąć utknięcia algorytmu BP w minimum lokalnym (paraliż uczenia) lub oscylacji wokół minimum, należy zastosować modyfikację szybkości uczenia się poprzez odpowiednie zmiany tego współczynnika lub w innym możliwe sposoby, omówione później.

Problem Znikających Gradientów

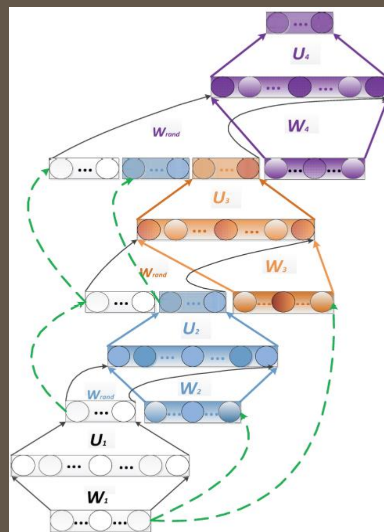
Przy stosowaniu gradientowych strategii uczenia się dla wielu warstw (np. MLP) zazwyczaj natrafiamy na problem zanikających gradientów, ponieważ pochodne są zawsze w zakresie [0, 1], więc ich wielokrotne przemnożenia prowadzą do bardzo małych liczb powodujących nieznaczące zmiany wag w warstwach neuronów bliskich wejściu sieci MLP.

Problem ten można rozwiązać za pomocą strategii transformacji przestrzeni wejściowej (PCA, ICA) i dostrajania, która najpierw trenuje warstwę modelu po warstwie w sposób nienadzorowany (np. przy użyciu głębokiego auto-kodera), a następnie używamy algorytmu wstecznej propagacji w celu dostrojenia sieci.

Hinton, Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. Science 2006



Dlatego, jeśli chcemy stworzyć głęboką wielowarstwową topologię MLP, musimy rozwiązać problem znikającego gradientu.



Aby rozwiązać ten problem, powinniśmy stopniowo budować głęboką strukturę. Będzie to jeden z celów naszych zajęć laboratoryjnych.

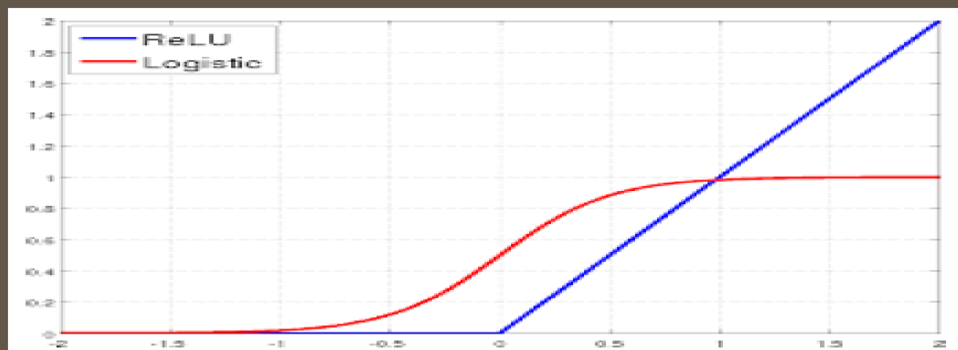
Rectified Linear Units (ReLU)

Poprawiona Funkcja Liniowa (ReLU - *Rectified Linear Unit*) eliminuje problem zanikających gradientów (tj. Pochodne zawsze znajdują się w zakresie [0, 1], więc ich wielokrotne mnożenia prowadzą do bardzo małych liczb powodujących znikome zmiany wag w warstwach neuronów, które są daleko od wyjścia sieci MLP), gdy używamy wielu ukrytych warstw (np. w głębokich sieciach neuronowych).

Jednostki ReLU są zdefiniowane jako: $f(S) = \max(0, S)$ zamiast korzystać z funkcji logistycznej.

Strategia wykorzystująca jednostki ReLU opiera się na szkoleniu solidnych cech dzięki rzadkim (rzadszym) aktywacjom tych jednostek, ponieważ gdy wartość funkcji jest równa 0, nie propagujemy sygnału do połączonych neuronów i nie musimy propagować delta z powrotem w takich neuronach, jak również podczas propagacji wstecznej.

Innym rezultatem używania ReLU jest to, że proces uczenia jest zazwyczaj szybszy.



Nair, Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. ICML 2010

Przezwyciężenie problemów uczenia

W celu przezwyciężenia trudności uczenia sieci algorytmem propagacji wstecznej błędów możemy wykorzystać:

- Bias - dodatkowy stały sygnał wejściowy (say $x_0=1$), który jest ważony ($w_{0,n}$) w jakiś sposób przypomina próg stosowany w modelach neuronów z twardym przełączeniem
- Momentum – które zazwyczaj zmniejsza skłonność do niestabilności i pozwala uniknąć szybkich wahań ($0 < \alpha < 1$), ale może nie zawsze działać lub może zaszkodzić zbieżności:

$$\Delta \mathbf{w}_{k,n}^{p} = \alpha \cdot \Delta \mathbf{w}_{k,n}^{p-1} + \eta \cdot \delta_n \cdot f'\left(\sum_{k=0}^{K} \mathbf{x}_k \cdot \mathbf{w}_k\right) \cdot \mathbf{x}_k = \alpha \cdot \Delta \mathbf{w}_{k,n}^{p-1} + \eta \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot \mathbf{x}_k$$

- Wygładzanie - nie zawsze jest wskazane z tego samego powodu:

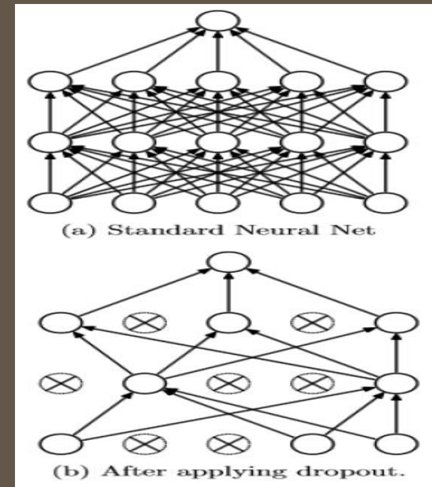
$$\Delta \mathbf{w}_{k,n}^{p} = \alpha \cdot \Delta \mathbf{w}_{k,n}^{p-1} + (1 - \alpha) \cdot \delta_n \cdot f'\left(\sum_{k=0}^{K} \mathbf{x}_k \cdot \mathbf{w}_k\right) \cdot \mathbf{x}_k$$
$$= \alpha \cdot \Delta \mathbf{w}_{k,n}^{p-1} + (1 - \alpha) \cdot \delta_n \cdot (1 - y_n) \cdot y_n \cdot \mathbf{x}_k$$

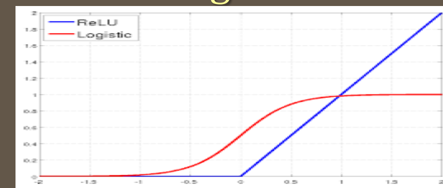
gdzie p jest okresem treningowym (cyklem) wzorców treningowych.

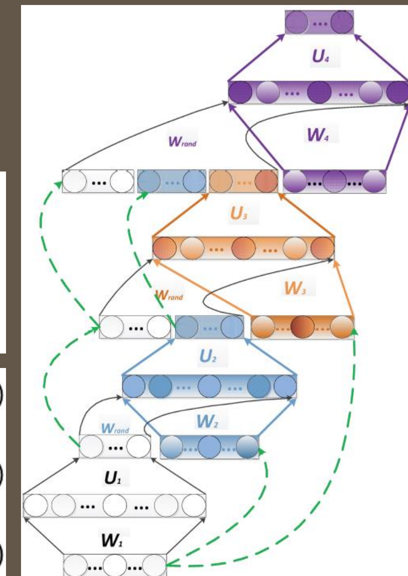
Przezwyciężenie problemów uczenia

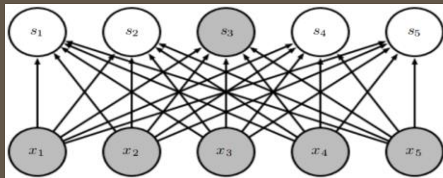
Aby przezwyciężyć problemy zbieżności algorytmu wstecznej propagacji, możemy:

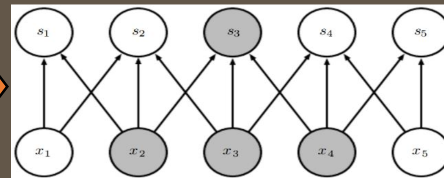
- Modyfikować wielkość parametru uczenia η w trakcie procesu uczenia.
- Rozpoczynać proces uczenia wielokrotnie z różnymi wagami inicjalnymi.
- Wykorzystać różne architektury sieci, np. zmienić liczbę warstw lub liczbę neuronów w tych warstwach.
- Wykorzystać algorytm genetyczny lub podejście ewolucyjne aby znaleźć bardziej odpowiednią architekturę sieci neuronowej.
- Zmniejszyć liczbę wejść by pokonać problem przekleństwa wymiarowości danych wejściowych.
- Zmiana zakresu funkcji sigmoidalnej z [0, 1] na [-1, 1].
- Użycie rzadkich połączeń, a nie każdy-z-każdym pomiędzy kolejnymi warstwami.
- Przełączanie między strategiami szkolenia off-line i on-line, ponieważ strategia off-line jest szybsza i bardziej stabilna, ale strategia online lepiej unika lokalnych minimów.
- Zamrozić wagi w uprzednio przeszkolonej warstwie lub podsieci.
- Użycie *rectified linear units* (ReLU), $f(S) = \max(0, S)$.
- Użycie walidacji krzyżowej, aby uniknąć problemu nadmiernego dopasowania (*overfitting*).
- Użycie techniki regulacji dropout.
- Użycie uczenia głębokiego i głębokich architektur sieciowych i strategii.

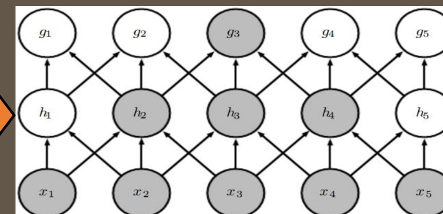












K-krotna Walidacja Krzyżowa

Strategia walidacji krzyżowej pozwala nam wykorzystywać wszystkie dostępne wzorce do szkolenia i walidacji na przemian podczas procesu szkolenia.

„K-fold" oznacza, że dzielimy wszystkie wzorce treningowe na K rozłączając je na mniej lub bardziej równe podzbiory. Następnie trenujemy wybrany model w K-razy podzbiorów K-1, a także testujemy ten model na boku K-razy.

Podzbiór walidacji zmienia się w trakcie kolejnych kroków uczenia/treningu:

5-FOLD	SUBSETS OF TRAINING PATTERNS				
STEPS	1	2	3	4	5
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TEST	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TEST	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TEST

K-fold Cross-Validation

Stosujemy różne parametry K w zależności od liczby wzorców treningowych:

K jest zwykle małe ($3 \leq K \leq 10$) dla dużych zbiorów uczących:

- Pozwala nam to lepiej zweryfikować model, jeśli jest testowany na większej liczbie schematów treningowych.
- Zmniejsza to również liczbę kroków szkoleniowych, które należy wykonać.

K jest zwykle duże ($10 \leq K \leq N$) dla mniejszych zbiorów uczących, gdzie N jest całkowitą liczbą wszystkich wzorców uczących.

- Pozwala nam to na wykorzystanie większej liczby wzorów do treningu i uzyskanie lepiej dopasowanego modelu.

5-FOLD	SUBSETS OF TRAINING PATTERNS				
STEPS	1	2	3	4	5
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TEST	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TEST	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TEST

10-FOLD	SUBSETS OF TRAINING PATTERNS									
STEPS	1	2	3	4	5	6	7	8	9	10
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
6	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN
7	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN
8	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN
9	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN
10	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST

K-krotna Walidacja Krzyżowa

N-krotna walidacja krzyżowa (strategia jednego opuszczenia - one-leave-out strategy) jest rzadko stosowana, ponieważ zbiór danych elementu N musi zostać przeszkolony N razy. Następną wadą jest to, że używamy tylko jednego wzorca w każdym kroku do walidacji całego modelu, co zwykle nie jest wystarczające do uzyskania wiarygodnych wyników i generalizacji modelu. Taki wynik nie jest reprezentatywny dla całej kolekcji i modelu CI. To rozwiązanie jest czasami używane w przypadku bardzo małych zestawów danych dla problemów regresji.

N-FOLD	One-element subsets of the training patter set consisting of 20 patterns																			
STEPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
6	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
7	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
8	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
9	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
10	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
11	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
12	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
13	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
14	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
15	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
16	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN
17	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN
18	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN
19	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN
20	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST

Strategie Selekcji Podziałów K-krotnej Walidacji Krzyżowej

Sposób wyboru wzorców testowych w każdym kroku treningowym powinien być proporcjonalny i reprezentatywny z punktu widzenia każdej klasy, tj. niezależny od liczebności klas! Musimy też rozważyć, w jaki sposób dane treningowe są zorganizowane (uporządkowane czy nie) w zestawie danych treningowych:

- Losowo
- Pogrupowane według kategorii (klas)
- Uporządkowane według wartości ich atrybutów
- Pogrupowane równocześnie według klas i uporządkowane według wartości ich atrybutów
- W nieznany sposób

5-FOLD	SUBSETS OF TRAINING PATTERNS THAT ARE RANDOMLY ORDERED IN THE DATA SET																																							
STEPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TEST	TEST	TEST	TEST	TEST	TEST	TEST

5-FOLD	SUBSETS OF TRAINING PATTERNS																																							
STEPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST

5-FOLD	SUBSETS OF TRAINING PATTERNS														
STEPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN
3	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN
4	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST

Nieuporządkowane wzorce (losowa kolejność) K-krotnej Walidacji Krzyżowej

Wzory testowe można również wybierać losowo z powtarzaniem lub bez:

5-FOLD	SUBSETS OF TRAINING PATTERNS																			
STEPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST
3	TEST	TRAIN	TRAIN	TRAIN	TEST	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN
4	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN

STEPS	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN
2	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST
3	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN
4	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN
5	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN

5-FOLD	SUBSETS OF TRAINING PATTERNS THAT ARE RANDOMLY ORDERED IN THE DATA SET																			
STEPS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN
2	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST
3	TEST	TRAIN	TRAIN	TRAIN	TEST	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN
4	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN
5	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN

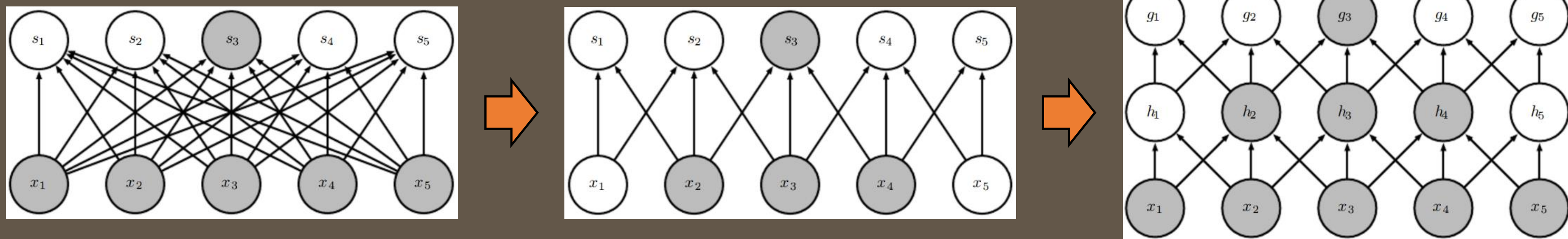
STEPS	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN
2	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST
3	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN
4	TRAIN	TEST	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN
5	TEST	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TEST	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN	TRAIN

Wybór pomiędzy różnymi opcjami powinien być dokonany na podstawie początkowej kolejności lub braku porządku wzorców wszystkich klas w zbiorze danych, aby uzyskać reprezentatywny wybór wzorców testowych stosowanych dla walidowanego modelu. Wzorce stosowane do testowania nie powinny się powtarzać w kolejnych grupach testowych, jedynie, że stosujemy mniej wiarygodne i prostsze podejście swobodnego losowania wzorców walidacyjnych.

Sieci Głębokie i Strategie Uczenia Głębokiego

Strategie uczenia głębokiego (deep learning strategies) zakładają zdolność do:

- aktualizację tylko wybranej części neuronów (drop-out), które najlepiej odpowiadają danym danych wejściowych, więc inne neurony i ich parametry (np. wagi, progi) nie są aktualizowane;
- unikanie łączenia wszystkich neuronów między kolejnymi warstwami, więc nie stosujemy strategii połączenia każdy-z-każdym znanej i powszechnie używanej w MLP i innych sieciach, ale staramy się pozwolić neuronom na specjalizację w rozpoznawaniu subwzorców, które można wyodrębnić z ograniczonych podzbiorów wejść;

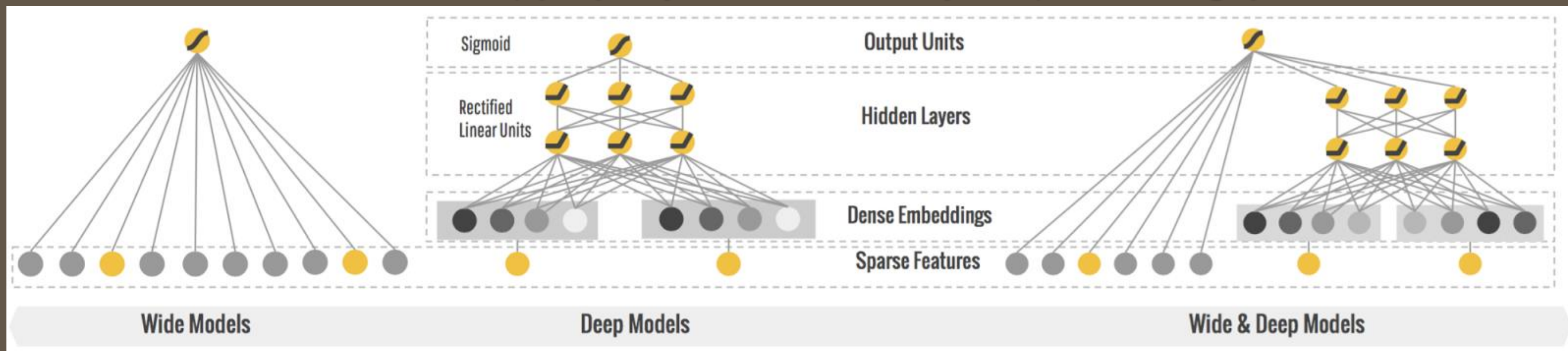


- tworzenie połączeń między różnymi warstwami i podsieciami, nie tylko między kolejnymi warstwami;
- stosowanie wielu podsieci, które można połączyć na różne sposoby, aby umożliwić neuronom z tych podsieci wyspecjalizowanie się w definiowaniu lub rozpoznawaniu ograniczonych podzbiorów funkcji lub subwzorców;
- pozwolenie neuronom na specjalizację i nie pokrywanie reprezentowanych regionów i reprezentację tych same cech lub subwzorców.

Szerokie i Głębokie Sieci Neuronowe

Szerokie (broad / wide) sieci neuronowe zakładają:

- Dodawanie neuronów w jednej ukrytej warstwie, aż wyniki treningu będą zadowalające.
- Podczas tego rodzaju treningu czasami zamrażamy wagi dodanych wcześniej neuronów i dostosowujemy tylko wagi aktualnie dodanego neuronu. Dzięki temu proces szkolenia jest szybszy, ale niekoniecznie lepszy.
- Szerokie sieci dostosowują się zwykle znacznie szybciej niż sieci głębokie.

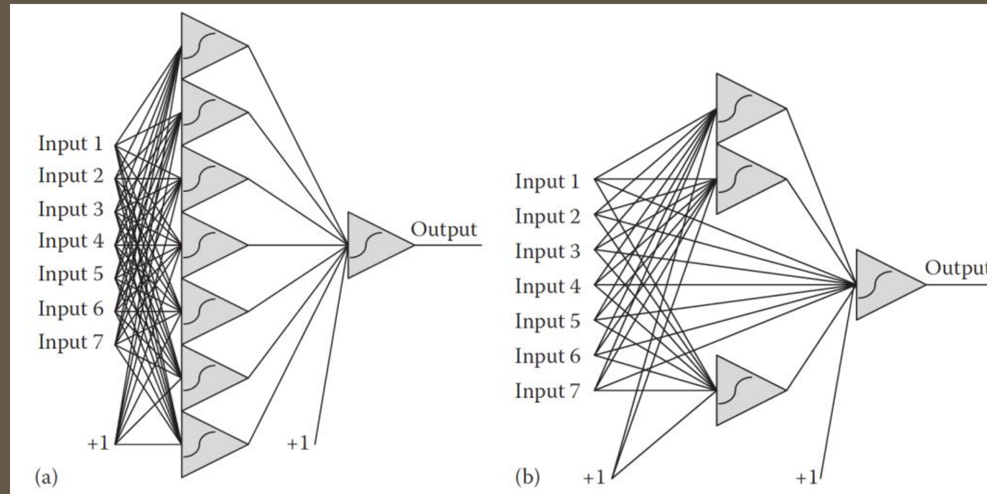


- Możemy się rozwijać:
  - Szerokie modele sieci neuronowych (dodawanie nowych neuronów w tej samej warstwie),
  - Modele głębokich sieci neuronowych (dodawanie nowych warstw neuronów w różnych warstwach),
  - Szerokie i głębokie modele sieciowe (łączące te dwa podejścia),
  - Podsieci, które specjalizują się w reprezentacji ograniczonego podzbioru funkcji, a następnie łączyły te podsieci w jedną dużą sieć neuronową.

Sieci neuronowe BMLP i FCC

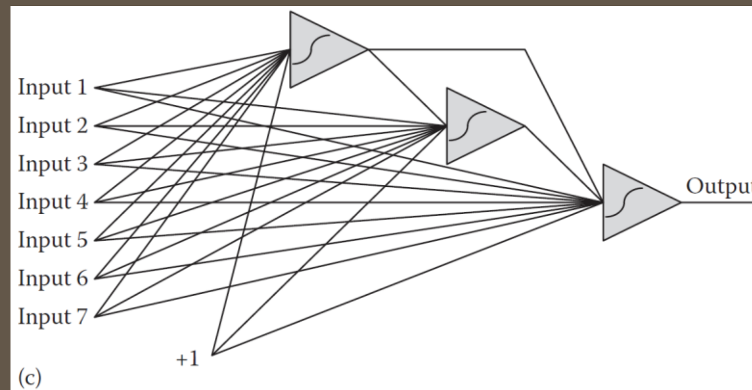
Sieci neuronowe BMLP zakładają:

- podłączenie neuronów nie tylko do neuronów poprzedniej warstwy, ale dodatkowo do wszystkich wejść sieci:



W pełni połączone kaskadowe sieci neuronowe (FCC) zakładają:

- dodawanie nowych neuronów w kolejnych ukrytych warstwach (każda ukryta warstwa składa się tylko z jednego neuronu) i połączenie ich ze wszystkimi wejściami i wszystkimi neuronami z poprzednich warstw.



Łączenie podsieci w głęboką strukturę neuronową

Stopniowo rozwijane głębokie sieci neuronowe zakładają:

- **Trenuj małą architekturę sieci neuronowej (zwykle z jedną warstwą ukrytą), aż ta sieć zmniejszy swój błąd treningowy.**
- **Utwórz następną podsieć i podłącz ją ponownie do wszystkich wejść sieci i dodatkowo do wyjść z wcześniej utworzonych i wyszkolonych podsieci, które można zamrozić lub pozostawić do następnej adaptacji wraz z nowo utworzoną podsiecią.**
- **Powtarzaj krok 2, aż sieć osiągnie zadowalający niski poziom funkcji błędu.**

Opcjonalnie można zamrozić wagi w uprzednio przeszkolonej warstwie lub podsieci.

ORDER of creating and training the MLP subnetworks creating the final DeepMLP

1st subnetwork outputs/results

2nd subnetwork outputs/results

3rd subnetwork outputs/results

FINAL CLASSIFICATION

FINAL OUTPUTS OF DeepMLP

MLP4

MLP3

MLP2

MLP1

Each green neuron should be connected to every yellow neuron in the following hidden layers as the cyan neuron

Ourput Layer of the 4th subnetwork

4th Hidden Layer

Ourput Layer of the 3rd subnetwork

3rd Hidden Layer

Ourput Layer of the 2nd subnetwork

2nd Hidden Layer

Ourput Layer of the 1st subnetwork

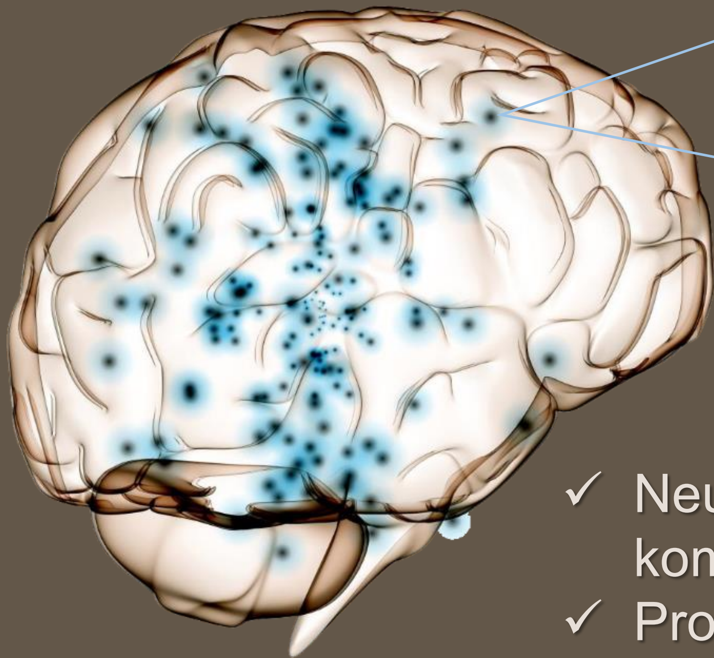
1st Hidden Layer

Input Layer

INPUTS

Strategie skojarzeniowe dostosowujących się biologicznych neuronów

Możemy znaleźć rozwiązanie w strukturach mózgu, w których dane są przechowywane razem z ich relacjami.

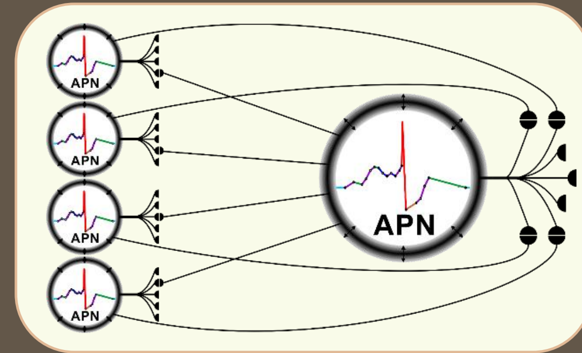


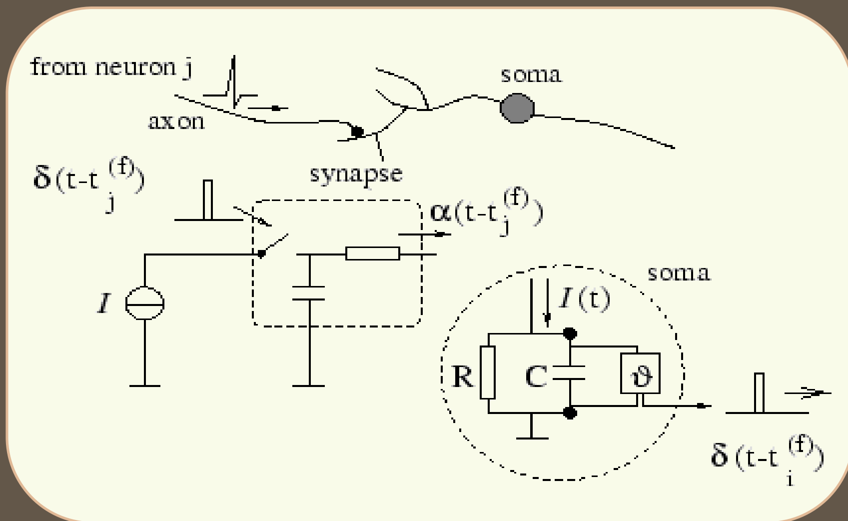
- ✓ Neurony mogą reprezentować dowolny podzbiór kombinacji danych wejściowych, które je aktywują.
- ✓ Procesy plastyczności neuronów automatycznie łączą neurony i wzmacniają połączenia, które reprezentują powiązane dane i obiekty.

Wykorzystajmy zoptymalizowane biologicznie rozwiązania!

Modele Neuronów trzeciej i czwartej generacji

3. **Impulsowe modele neuronów** wzbogaciły ten model o wprowadzenie czasu, który jest bardzo ważny podczas integracji bodźców i późniejszego modelowania procesów.





4. **Asocjacyjne modele pulsacyjne neuronów (APN)** wytwarzają serię impulsów (skoków) w czasie, którego częstotliwość określa poziom skojarzenia (asocjacji). Ponadto wzbogacają model automatycznym mechanizmem plastyczności neuronalnej, który pozwala neuronom warunkowo łączyć się i konfigurować skojarzeniowe struktury neuronowe reprezentujące dane, obiekty i ich sekwencje oraz relacje pomiędzy nimi.

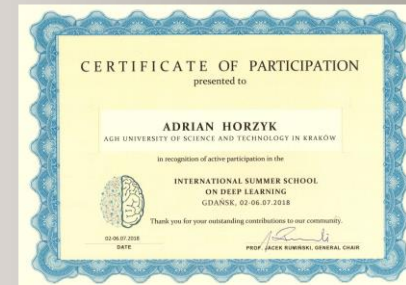
Bibliografia i Literatura

1. Nikola K. Kasabov, ***Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence***, In Springer Series on Bio- and Neurosystems, Vol 7., Springer, 2019.
2. Ian Goodfellow, Yoshua Bengio, Aaron Courville, ***Deep Learning***, MIT Press, 2016, ISBN 978-1-59327-741-3 or PWN 2018.
3. Holk Cruse, ***Neural Networks as Cybernetic Systems***, 2nd and revised edition
4. R. Rojas, ***Neural Networks***, Springer-Verlag, Berlin, 1996.
5. ***Convolutional Neural Network*** (Stanford)
6. ***Visualizing and Understanding Convolutional Networks***, Zeiler, Fergus, ECCV 2014
7. **IBM: https://www.ibm.com/developerworks/library/ba-data-becomes-knowledge-1/index.html**
8. **NVIDIA: https://developer.nvidia.com/discover/convolutional-neural-network**
9. **A. Horzyk**, J. A. Starzyk, J. Graham, ***Integration of Semantic and Episodic Memories***, IEEE Transactions on Neural Networks and Learning Systems, Vol. 28, Issue 12, Dec. 2017, pp. 3084 - 3095, 2017, DOI: 10.1109/TNNLS.2017.2728203.
10. **A. Horzyk**, J.A. Starzyk, ***Multi-Class and Multi-Label Classification Using Associative Pulsing Neural Networks***, IEEE Xplore, In: 2018 IEEE World Congress on Computational Intelligence (WCCI IJCNN 2018), 2018, (in print).
11. **A. Horzyk**, J.A. Starzyk, ***Fast Neural Network Adaptation with Associative Pulsing Neurons***, IEEE Xplore, In: 2017 IEEE Symposium Series on Computational Intelligence, pp. 339 -346, 2017, DOI: 10.1109/SSCI.2017.8285369.
12. **A. Horzyk**, K. Gołdon, ***Associative Graph Data Structures Used for Acceleration of K Nearest Neighbor Classifiers***, LNCS, In: 27th International Conference on Artificial Neural Networks (ICANN 2018), 2018, (in print).
13. **A. Horzyk**, ***Deep Associative Semantic Neural Graphs for Knowledge Representation and Fast Data Exploration***, Proc. of KEOD 2017, SCITEPRESS Digital Library, pp. 67 - 79, 2017, DOI: 10.13140/RG.2.2.30881.92005.
14. **A. Horzyk**, ***Neurons Can Sort Data Efficiently***, Proc. of ICAISC 2017, Springer-Verlag, LNAI, 2017, pp. 64 - 74, ICAISC BEST PAPER AWARD 2017 sponsored by Springer.
15. **Horzyk, A.**, ***How Does Generalization and Creativity Come into Being in Neural Associative Systems and How Does It Form Human-Like Knowledge?***, Elsevier, Neurocomputing, Vol. 144, 2014, pp. 238 - 257, DOI: 10.1016/j.neucom.2014.04.046.

Adrian Horzyk

horzyk@agh.edu.pl

Google: Horzyk



University of Science and Technology in Krakow, Poland