METODY SZTUCZNEJ INTELIGENCJI

MLP – multi layer perceptron

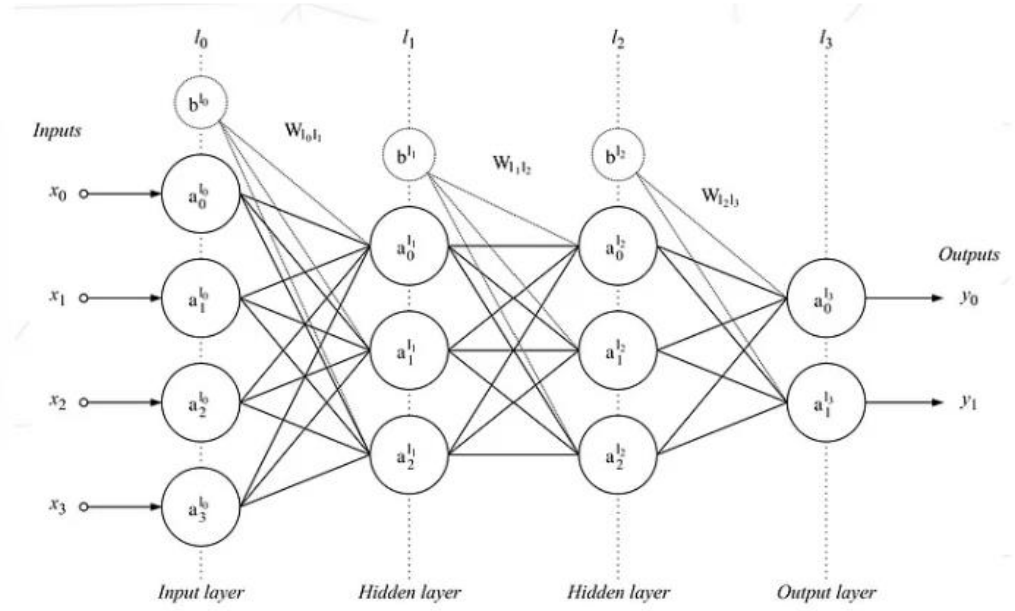
1. Wstęp teoretyczny

1.1 Definicja MLP

MLP (Multi-Layer Perceptron) to klasyczna architektura **sztucznej sieci neuronowej**, która znajduje zastosowanie w bardzo wielu dziedzinach **uczenia maszynowego** i **sztucznej inteligencji**. Dzięki swojej zdolności do modelowania złożonych, nieliniowych zależności między danymi wejściowymi a wyjściowymi, MLP sprawdza się w różnorodnych zadaniach.

MLP wyróżnia się kilkoma cechami:

- Składa się z co najmniej **trzech warstw** neuronów: warstwy wejściowej (input layer), co najmniej **jednej warstwy ukrytej** (hidden layer) oraz warstwy wyjściowej (output layer).
- Każda warstwa (z wyjątkiem wejściowej) jest w pełni połączona z warstwą poprzednią – to znaczy, że **każdy neuron** w danej warstwie odbiera sygnał (wartość) od **każdego neuronu** w warstwie poprzedniej i dodatkowo posiada **bias**.
- W warstwach ukrytych stosuje się **nieliniowe funkcje aktywacji** (najczęściej ReLU, Sigmoid, Tanh itp.), co pozwala sieci uczyć się nieliniowych zależności w danych.



Rys. 1 Przykładowy schemat wielowarstwowej sieci z perceptronami

Proces uczenia można podzielić na trzy fazy – trening, walidacja i test:

- **Trening:**
  To etap, podczas którego model "uczy się" na podstawie danych treningowych. Wagi sieci są aktualizowane na podstawie obliczanego błędu (funkcji kosztu) przy użyciu algorytmów optymalizacji. Model jest „dostosowywany" do danych, co może prowadzić do dopasowania zarówno istotnych, jak i nieistotnych cech, jeśli nie stosujemy technik regularyzacji.
- **Walidacja:**
  W tej fazie używa się oddzielnego zbioru danych walidacyjnych, który nie jest wykorzystywany do aktualizacji wag. Służy do monitorowania postępów uczenia i do dostrajania hiperparametrów (np. szybkości uczenia, liczby neuronów, itp.). Dzięki walidacji można również stosować techniki takie jak early stopping, aby zapobiec przeuczeniu – model przestaje trenować, gdy wyniki na zbiorze walidacyjnym przestają się poprawiać.
- **Test:**
  Po zakończeniu treningu (i dopasowaniu modelu przy pomocy walidacji) ocenia się ostateczną jakość modelu na zbiorze testowym. Testowanie odbywa się na danych, których model wcześniej nie widział, co pozwala uzyskać miarę zdolności modelu do generalizacji na nowe, niewidziane dane.

Poniżej znajduje się przegląd głównych **zastosowań MLP**:

1. Klasyfikacja danych

MLP jest powszechnie używany do klasyfikacji, gdy dane mają postać **wektorów cech.**

- Klasyfikacja kwiatów IRIS (jak w przykładzie),
- Diagnoza chorób na podstawie wyników badań (np. cukrzyca, nowotwory),
- Klasyfikacja e-maili jako spam/niespam,
- Rozpoznawanie typu klienta (np. lojalny / okazjonalny / ryzykowny).

2. Regresja

MLP może również być użyty do **regresji**, czyli przewidywania wartości ciągłych

- Przewidywanie ceny mieszkania na podstawie lokalizacji i parametrów,
- Prognozowanie popytu lub sprzedaży,
- Przewidywanie zużycia energii,
- Modelowanie funkcji fizycznych lub ekonomicznych.

3. Rozpoznawanie wzorców i obrazów

Choć w nowoczesnych aplikacjach używa się CNN, to MLP może być stosowany do prostszych zadań:

- Rozpoznawanie cyfr (np. MNIST),
- Klasyfikacja binarnych obrazów (np. czy jest twarz na obrazie?),
- Rozpoznawanie znaków pisma maszynowego.

4. Przetwarzanie sygnałów

MLP może klasyfikować sygnały lub przewidywać ich przyszłą wartość.

- Klasyfikacja rodzaju dźwięku (np. muzyka vs mowa),
- Analiza EKG (czy jest arytmia?),
- Detekcja uszkodzeń w maszynach na podstawie drgań (diagnostyka).

5. Przetwarzanie tekstu (NLP) – w prostych zastosowaniach

MLP może być stosowany do analizy tekstu, jeśli tekst został przetworzony na wektory cech (np. TF-IDF, word embeddings):

- Klasyfikacja nastroju recenzji (pozytywna/negatywna),
- Detekcja tematów wiadomości,
- Rozpoznawanie języka.

6. Uczenie ze wzmocnieniem (Reinforcement Learning)

MLP może być używany jako **funkcja aproksymująca** (np. Q-value approximator) w algorytmach takich jak Q-learning, DQN.

7. Zastosowania przemysłowe i inżynierskie

- Sterowanie procesami (np. sieci MLP sterujące robotami, przewidywaniem trajektorii),
- Rozpoznawanie defektów w produkcji,
- Modelowanie systemów dynamicznych.

Kiedy stosować MLP

- Dane mają **strukturę wektorową** (tablice, tabelaryczne dane, dane z czujników)
- Problem nie wymaga uwzględniania lokalnych struktur (np. obrazów – tam lepsze CNN)
- Zależy nam na **szybkim prototypowaniu** i prostocie
- Chcemy przetestować, czy prosta sieć neuronowa wystarczy przed użyciem głębszych modeli

Kiedy nie stosować MLP

- Jeśli dane mają **strukturę sekwencyjną** (np. tekst, audio) – lepiej użyć RNN/LSTM/Transformer,
- Jeśli dane to **obrazy 2D/3D** – lepsze będą CNN (konwolucyjne sieci neuronowe),
- Jeśli problem wymaga uwzględnienia **zależności czasowych** – potrzebne są sieci rekurencyjne.

1.2 Zasada działania

MLP w regresji

Cel: przewidzenie wartości rzeczywistej (np. liczby, pomiaru) na podstawie wejściowego wektora cech.

Zasada działania:

- Dane wejściowe przechodzą przez warstwy ukryte, jak w klasyfikacji.
- Ostatnia warstwa wyjściowa składa się zazwyczaj z jednego neuronu, bez funkcji aktywacji (czyli liniowe wyjście).
- Sieć zwraca wartość liczbową jako predykcję.

Uczenie:

- Używa się funkcji kosztu typu MSE (Mean Squared Error) lub MAE (Mean Absolute Error).
- Proces optymalizacji minimalizuje różnicę między przewidywaną a rzeczywistą wartością.

MLP w klasyfikacji

Cel: przypisanie każdej próbce wejściowej odpowiedniej klasy spośród dostępnych.

Zasada działania:

- Dane wejściowe (wektor cech) trafiają do warstwy wejściowej.
- Każda warstwa ukryta wykonuje transformację: oblicza sumę ważoną wejść, dodaje bias, a następnie stosuje funkcję aktywacji (np. ReLU, sigmoid, tanh).
- Ostatnia warstwa wyjściowa zawiera tyle neuronów, ile jest klas w problemie.
- Zastosowanie funkcji aktywacji softmax w ostatniej warstwie powoduje, że wyjście sieci to wektor prawdopodobieństw, których suma wynosi 1.

- Sieć wybiera klasę z najwyższym prawdopodobieństwem jako wynik klasyfikacji.

Uczenie:

- Używa się funkcji kosztu typu cross-entropy, która mierzy różnicę między rozkładem przewidywanym przez sieć a rzeczywistą etykietą w formie one-hot.
- Uczenie odbywa się za pomocą metody gradientu prostego (backpropagation) i optymalizacji (np. SGD, Adam).

Co to jest softmax?

Funkcja **softmax** przekształca **wektor dowolnych liczb rzeczywistych** (tzw. logitów) w **rozkład prawdopodobieństwa**. Jest stosowana w **ostatniej warstwie sieci neuronowej** w zadaniach klasyfikacji wieloklasowej.

Jak działa softmax?

Dla wektora $\mathbf{z} = [z_1, z_2, \ldots, z_K]$, funkcja softmax zwraca nowy wektor:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- z to "logit", czyli surowy wynik neuronu (np. z fullyConnectedLayer).
- K to liczba klas.
- Softmax przypisuje każdemu elementowi wartość z przedziału (0,1), tak że **suma wszystkich elementów wynosi 1**.

Softmax zamienia wyniki sieci na **prawdopodobieństwa klas**. Klasa z największym prawdopodobieństwem jest uznawana za **przewidywaną przez sieć**.

Przykład:

Logity (surowe wyjście sieci):

$$z = [2.0,\ 1.0,\ 0.1]$$

Softmax przekształci je w:

$$[0.659,\ 0.242,\ 0.099]$$

Sieć "uważa", że najbardziej prawdopodobna jest klasa 1 (65.9%).

Co to jest cross-entropy (entropia krzyżowa)?

Cross-entropy to **funkcja kosztu** (loss function), która mierzy różnicę między:

- Rozkładem prawdopodobieństwa **przewidywanym przez model** (czyli wyjściem softmaxa),
- A **rzeczywistą etykietą klasy** (najczęściej w formie one-hot).

Jak działa cross-entropy?

Dla rzeczywistej klasy zapisanej jako zero-jeden:

$$\mathbf{y} = [0, 1, 0] \quad \text{(czyli prawdziwa klasa to klasa 2)}$$

I przewidywanego rozkładu z softmaxa:

$$\hat{\mathbf{y}} = [0.2, 0.7, 0.1]$$

Funkcja cross-entropy:

$$\text{Loss} = -\sum_{i=1}^{K} y_i \cdot \log(\hat{y}_i)$$

W tym przypadku:

$$\text{Loss} = -\log(0.7) \approx 0.357$$

- Im wyższe prawdopodobieństwo przypisane **właściwej klasie**, tym **niższy błąd**.
- Jeśli sieć się myli (przypisuje niskie prawdopodobieństwo prawidłowej klasie), błąd jest wysoki.
- Funkcja ta **mocno karze** pewne błędne klasyfikacje.

Softmax + cross-entropy = standardowy mechanizm klasyfikacji

W klasyfikacji wieloklasowej:

1. **Sieć wylicza logity** (np. `[2.0, 1.0, 0.1]`).
2. **Softmax** zamienia je w prawdopodobieństwa klas.
3. **Cross-entropy** porównuje wynik z prawdziwą klasą i oblicza błąd.

4. **Wsteczna propagacja** (backpropagation) aktualizuje wagi, aby zmniejszyć ten błąd.

Dlaczego softmax + cross-entropy działa dobrze razem?

- Softmax generuje **probabilistyczne wyjście**.
- Cross-entropy mierzy, jak dobrze to prawdopodobieństwo **pasuje do rzeczywistej etykiety**.
- Wspólnie dają **silny sygnał gradientu**, co przyspiesza i stabilizuje uczenie.

Element	Softmax	Cross-entropy
Typ	Funkcja aktywacji (na wyjściu sieci)	Funkcja kosztu (loss function)
Zastosowanie	Klasyfikacja wieloklasowa	Porównanie przewidywań z rzeczywistością
Wynik	Rozkład prawdopodobieństwa	Liczba reprezentująca błąd
Idealny przypadek	1 dla prawdziwej klasy	0 (brak błędu)

Porównanie RMSE i Cross-Entropy

Cecha	RMSE	Cross-Entropy
Typ problemu	Regresja (ciągłe wartości)	Klasyfikacja (kategorie, etykiety)
Typ wyjścia	Liczba rzeczywista	Rozkład prawdopodobieństwa
Oczekiwana aktywacja	brak (lub liniowa)	softmax
Jednostka błędu	Ta sama co jednostka danych (np. °C)	Wartość bez jednostki (logarytmiczna)
Zachowanie przy błędach	Równomierna kara	Duża kara za błędne klasyfikacje
Czułość na klasy	Traktuje klasy jako liczby	Traktuje klasy jako rozłączne zdarzenia
Zalecana funkcja kosztu	`regressionLayer` w MATLAB	`classificationLayer` w MATLAB

Porównanie klasyfikacji i regresji w MLP

Cecha	Klasyfikacja	Regresja
Typ wyjścia	Etykieta klasy	Liczba rzeczywista
Warstwa wyjściowa	FullyConnected + softmax	FullyConnected (1 neuron, liniowy)

Funkcja aktywacji	softmax	brak (lub liniowa)
Funkcja kosztu	cross-entropy	RMSE
Zastosowanie	rozpoznawanie klas, etykietowanie	przewidywanie wartości liczbowych

1.3 Trenowanie (uczenie) MLP

Niezależnie od typu zadania, trenowanie sieci MLP składa się z kilku standardowych etapów:

1. **Inicjalizacja wag** (losowa).
2. **Forward propagation (propagacja w przód):**
    a. Dane wejściowe przechodzą przez kolejne warstwy (fully connected + funkcje aktywacji).
    b. Otrzymujemy wynik na wyjściu sieci.
3. **Obliczenie funkcji kosztu (loss function):**
    a. Mierzymy różnicę między wynikiem sieci a wartością oczekiwaną (etykietą lub wartością).
4. **Backpropagation (propagacja wsteczna):**
    a. Obliczamy gradienty błędu względem wag.
    b. Wagi są aktualizowane w kierunku zmniejszającym błąd (gradient descent).
5. **Powtarzanie (epoki):**
    a. Proces jest powtarzany przez wiele epok (iteracji), aż do zbieżności lub spełnienia warunku stopu.

Trenowanie MLP w regresji

Sieć uczy się przewidywać **wartość liczbową** (np. cenę, temperaturę, wynik pomiaru).

Charakterystyka:

- **Wyjście:** 1 neuron (dla jednej wartości do przewidzenia)
- **Aktywacja na wyjściu:** brak lub liniowa (tzn. bez ograniczenia zakresu)
- **Etykiety:** liczby rzeczywiste
- **Funkcja kosztu:** `mean squared error (MSE)` lub `mean absolute error (MAE)`
- **Warstwa końcowa w MATLAB:** `regressionLayer`

Trening w MATLABie:

```
layers = [
    featureInputLayer(nFeatures)
    fullyConnectedLayer(10)
    reluLayer
    fullyConnectedLayer(1)          % jedna wartość na wyjściu
    regressionLayer
];

options = trainingOptions('adam', ...);
net = trainNetwork(XTrain, YTrain, layers, options);
```

Trenowanie MLP w klasyfikacji

Sieć uczy się przypisywać każdemu wejściu **klasę** spośród kilku możliwych.

Charakterystyka:

- **Wyjście:** liczba neuronów = liczba klas (np. 3 dla IRIS)
- **Aktywacja na wyjściu:** `softmax` (zamienia wektor na prawdopodobieństwa klas)
- **Etykiety:** one-hot encoding lub `categorical` (np. "A", "B", "C")
- **Funkcja kosztu:** `cross-entropy`
- **Warstwa końcowa w MATLAB:** `classificationLayer`

Trening w MATLABie:

```
layers = [
    featureInputLayer(nFeatures)
    fullyConnectedLayer(10)
    reluLayer
    fullyConnectedLayer(nClasses)
    softmaxLayer
    classificationLayer
];

options = trainingOptions('adam', ...);
net = trainNetwork(XTrain, YTrain, layers, options);
```

Różnice w treningu: klasyfikacja vs regresja

Etap / Cecha	Klasyfikacja	Regresja
Typ wyjścia	Wektor prawdopodobieństw (softmax)	Liczba rzeczywista (bez aktywacji)

Liczba neuronów wyj.	= liczba klas	Zwykle 1
Funkcja aktywacji	softmax	liniowa (brak)
Funkcja kosztu	cross-entropy	mean squared error (MSE)
Etykiety (Y)	klasy (np. 'A', 'B', 'C')	wartości ciągłe (np. 27.5)
Warstwa końcowa	`classificationLayer`	`regressionLayer`

Aby „automatycznie” zmieniać hiperparametry (np. szybkości uczenia, liczby neuronów, itp.) w trakcie walidacji, czyli dostosowywać je dynamicznie w zależności od wyników walidacyjnych, można rozważyć kilka podejść:

1. **Harmonogram zmiany współczynnika uczenia**
   MATLAB pozwala ustawić w opcjach treningu harmonogram zmiany (schedule) współczynnika uczenia. Przykładowo, używając opcji `'LearnRateSchedule','piecewise'` wraz z `'LearnRateDropPeriod'` i `'LearnRateDropFactor'`, możesz zaprogramować automatyczne obniżanie współczynnika uczenia co określoną liczbę epok.
2. **Optymalizacja hiperparametrów z wykorzystaniem bayesopt**
   Jeśli chcemy przeszukać przestrzeń hiperparametrów (np. współczynnik uczenia, wielkość batcha itp.) w celu znalezienia optymalnych ustawień na podstawie wyników walidacji, można wykorzystać funkcję `bayesopt`. W tym podejściu definiujemy funkcję, która dla danego zestawu hiperparametrów trenuje sieć i zwraca miarę błędu (np. 1 - accuracy) na zbiorze walidacyjnym.
3. **Własny, niestandardowy trening (custom training loop)**
   Jeśli zależy nam na dynamicznej adaptacji hiperparametrów (np. zmiana współczynnika uczenia „w locie” na podstawie trendu błędu walidacyjnego), trzeba stworzyć własną pętlę treningową z wykorzystaniem obiektów `dlnetwork` i `dlarray`. Dzięki takiej pętli mamy pełną kontrolę nad procesem – można monitorować błąd walidacyjny po każdej epoce i w zależności od jego wartości modyfikować hiperparametry.

1.4 Metody optymalizacji uczenia i normalizacji danych

Metody optymalizacji uczenia

MATLAB, korzystając z Deep Learning Toolbox, udostępnia kilka głównych optymalizatorów, które można wykorzystać przy trenowaniu sieci MLP:

1. **SGDM – Stochastic Gradient Descent with Momentum**

a. **Jak działa:**
SGDM aktualizuje wagi sieci na podstawie gradientu obliczonego dla mini-batcha danych. Do standardowej metody stochastycznego gradientu dodaje się składnik momentum, który uwzględnia wcześniejsze zmiany wag. Dzięki temu optymalizator potrafi wygładzić aktualizacje, przyspieszyć zbieżność i pomóc uniknąć utknięcia w lokalnych minimach.
b. **Konfiguracja w MATLABie:**
Używając funkcji `trainingOptions` ustawiasz `'sgdm'` jako metodę optymalizacji. Dodatkowo możesz ustawić parametry takie jak `'InitialLearnRate'` (początkowa szybkość uczenia) oraz `'Momentum'`.

2. **Adam – Adaptive Moment Estimation**
a. **Jak działa:**
Adam wykorzystuje zarówno pierwszy moment (średnia gradientu), jak i drugi moment (średnia kwadratów gradientu) do dynamicznego dostosowywania szybkości uczenia dla każdej wagi oddzielnie. Dzięki temu algorytm jest odporny na problemy związane z różnicami w skali parametrów oraz szumem w danych, co często skutkuje szybszą konwergencją.
b. **Konfiguracja w MATLABie:**
W `trainingOptions` wybierasz `'adam'` jako metodę. Parametry, które możesz ustawić, to m.in. `'InitialLearnRate'` czy inne opcje, które pozwalają kontrolować adaptacyjne aspekty algorytmu.
3. **RMSprop – Root Mean Squared Propagation**
a. **Jak działa:**
RMSprop utrzymuje wykładniczy średni ruchomy kwadratów gradientów, co pozwala na normalizację aktualizacji wag. Dzięki temu zmiany w wagach są dostosowywane do aktualnej skali gradientu, co pomaga radzić sobie z problemem oscylacji i zanikania gradientów.
b. **Konfiguracja w MATLABie:**
W nowszych wersjach MATLABa opcja `'rmsprop'` może być dostępna jako metoda optymalizacji. Ustawiasz ją podobnie jak pozostałe, a kluczowe parametry to szybkość uczenia oraz ewentualne ustawienia związane z wykładniczym średnim ruchomym.

Wybór optymalizatora i jego parametrów

- Wybór konkretnego algorytmu zależy od specyfiki problemu, charakterystyki zbioru danych oraz architektury sieci.

- **Adam** jest często wybierany jako domyślny optymalizator, gdyż adaptuje szybkość uczenia dla każdej wagi, co może przyspieszyć proces uczenia.
- **SGDM** bywa korzystny przy bardzo dużych zbiorach danych lub głębszych sieciach, gdzie momentum pomaga w płynnej aktualizacji wag.
- **RMSprop** może być przydatny w sytuacjach, gdy gradienty są bardzo zmienne lub występują problemy z oscylacjami.

Normalizacja Danych

Dropout to metoda regularizacji, która polega na losowym "wyłączaniu" (dezaktywacji) części neuronów podczas treningu sieci. Przy każdym przebiegu (epoce) treningowej dla każdej próbki pewna część neuronów (określana przez współczynnik dropout, np. 0.5) jest pomijana w procesie propagacji sygnału.
Losowo wyłączając neurony, dropout zapobiega współadaptacji cech przez poszczególne neurony, co zmusza model do uczenia się bardziej rozproszonych i niezależnych reprezentacji. Dzięki temu sieć jest mniej podatna na przeuczenie, ponieważ nie może polegać na zestawie specyficznych neuronów przy każdej iteracji. W fazie testowania, gdy dropout nie jest stosowany, aktywacje neuronów są skalowane, aby zachować spójność z treningiem.

Warstwę dropout dodaje się do sieci, która losowo dezaktywuje określony procent neuronów podczas treningu. Przykład:

```
% Przykładowa warstwa dropout z prawdopodobieństwem 0.5
dropoutLayer(0.5, 'Name', 'drop1')
```

W architekturze sieci umiezcza się tę warstwę np. po warstwie aktywacji:

```
layers = [
    featureInputLayer(inputSize, 'Name', 'input')
    fullyConnectedLayer(10, 'Name', 'fc1')
    reluLayer('Name', 'relu1')
    dropoutLayer(0.5, 'Name', 'drop1')  % Dropout po aktywacji
    fullyConnectedLayer(numClasses, 'Name', 'fc2')
    softmaxLayer('Name', 'softmax')
    classificationLayer('Name', 'output')
];
```

Batch Normalization (normalizacja partii) to technika normalizacji danych, która standaryzuje wejścia do warstw sieci neuronowej. Wykonuje się to poprzez przekształcenie danych tak, aby miały średnią 0 i wariancję 1 w obrębie każdej mini partii (batcha) treningowej.
Dla każdej mini partii obliczana jest średnia i wariancja, a następnie dane są normalizowane. Po tej operacji zwykle dodaje się skalowanie i przesunięcie (przez parametry uczone podczas treningu), aby sieć mogła zachować elastyczność w reprezentacji. Batch Normalization przyspiesza proces uczenia, zmniejsza tzw. problem wewnętrznego przesunięcia danych (internal covariate shift) oraz umożliwia stosowanie wyższych współczynników uczenia, co często przekłada się na lepszą stabilność modelu.

Przykład:
```
batchNormalizationLayer('Name', 'bn1')
```

Można umieścić ją między warstwą w pełni połączoną a funkcją aktywacji:

```
layers = [
    featureInputLayer(inputSize, 'Name', 'input')
    fullyConnectedLayer(10, 'Name', 'fc1')
    batchNormalizationLayer('Name', 'bn1')  % Normalizacja przed
aktywacją
    reluLayer('Name', 'relu1')
    fullyConnectedLayer(numClasses, 'Name', 'fc2')
    softmaxLayer('Name', 'softmax')
    classificationLayer('Name', 'output')
];
```

L1 / L2 Regularization

- **L1 Regularization:**
  Polega na dodaniu do funkcji kosztu dodatkowego składnika, który jest proporcjonalny do sumy wartości bezwzględnych wag w sieci.
  Ta forma regularyzacji zachęca model do ustawienia niektórych wag dokładnie na zero, co prowadzi do rzadkich (sparse) reprezentacji. Dzięki temu L1 może działać jako metoda selekcji cech, eliminując te mniej istotne.
- **L2 Regularization:**
  Dodaje do funkcji kosztu składnik proporcjonalny do sumy kwadratów wag.
  L2 karze duże wartości wag, co skutkuje ich bardziej równomiernym rozkładem i mniejszą wartością. W efekcie model jest mniej podatny na przeuczenie,

ponieważ nie polega na bardzo dużych wartościach wag, co pozwala na lepszą generalizację na nowych danych.

W MATLABie L2 regularization (czasami nazywana też weight decay) ustawiasz w opcjach treningu poprzez parametr `'L2Regularization'`. Przykład:

```
options = trainingOptions('adam', ...
    'MaxEpochs', 50, ...
    'MiniBatchSize', 16, ...
    'InitialLearnRate', 0.01, ...
    'L2Regularization', 0.001, ...  % Przykładowa wartość karania L2
    'Plots', 'training-progress', ...
    'Verbose', false);
```

MATLAB nie posiada bezpośredniej opcji do L1 regularization w funkcji `trainingOptions`. Aby osiągnąć efekt L1, trzeba:

- **Ręcznie zmodyfikować funkcję kosztu**, dodając składnik proporcjonalny do sumy wartości bezwzględnych wag,
- **Użyć niestandardowych warstw** lub własnych pętli treningowych, które implementują dodatkowy składnik karzący duże wartości wag.

1.5 Sposób doboru architektury MLP

Dobór architektury sieci MLP to proces iteracyjny:

1. **Liczba warstw i neuronów**
    a. **Początkowo zacznij od prostego modelu:** Dla wielu problemów warto wypróbować jedną lub dwie warstwy ukryte.
    b. **Złożoność problemu:** Im bardziej złożone zależności w danych, tym potencjalnie więcej neuronów i/lub warstw może być potrzebnych. Jednak większa liczba parametrów zwiększa ryzyko przeuczenia.
    c. **Eksperymentowanie:** Testuj różne konfiguracje (np. od 10 do 100 neuronów w warstwie) i sprawdzaj, która architektura daje lepsze wyniki na zbiorze walidacyjnym.
2. **Metoda uczenia (optymalizator)**
    a. **Popularne opcje:** Adam jest często dobrym punktem wyjścia, ponieważ automatycznie dostosowuje tempo uczenia, ale możesz również rozważyć SGD, RMSprop czy Adagrad.
    b. **Wielkość zbioru:** Dla dużych zbiorów danych i bardziej skomplikowanych modeli niekiedy warto eksperymentować z optymalizatorami, które lepiej radzą sobie w takich warunkach.
3. **Funkcje aktywacji**

   a. **ReLU (Rectified Linear Unit):** Jest najczęściej używana w warstwach ukrytych, ze względu na prostotę i efektywność.
   b. **Alternatywy:** W niektórych przypadkach możesz spróbować LeakyReLU, ELU lub tanh, zwłaszcza gdy problem z "zanikającymi gradientami" staje się widoczny.
   c. **Warstwa wyjściowa:** Dla klasyfikacji wieloklasowej stosuj softmax, a dla problemów regresyjnych – funkcję liniową lub odpowiednią inną aktywację.
4. **Regularizacja i normalizacja**
   a. **Dropout:** Pomaga zapobiegać przeuczeniu, wyłączając losowo część neuronów podczas treningu.
   b. **Batch Normalization:** Może przyspieszyć uczenie i poprawić stabilność modelu.
   c. **L1/L2 Regularization:** Dodanie kar za duże wartości wag również może być pomocne.
5. **Optymalizacja hiperparametrów**
   a. **Metody automatyczne:** Grid search, random search czy bayesopt mogą pomóc w systematycznym poszukiwaniu najlepszych wartości hiperparametrów (takich jak liczba neuronów, współczynnik uczenia, wielkość batcha).
   b. **Walidacja krzyżowa:** Używaj zbioru walidacyjnego, aby ocenić uogólnianie modelu i uniknąć przeuczenia.
6. **Znajomość problemu**
   a. **Analiza danych:** Wstępna analiza statystyczna i wizualizacja danych mogą dostarczyć wskazówek, jak złożony model jest potrzebny.
   b. **Ekspercka wiedza:** W niektórych dziedzinach istnieją rekomendacje co do rozmiaru modelu, wynikające z doświadczenia praktycznego.

2. Przykłady MLP

Regresja

Model MLP ma za zadanie przewidywać wartość funkcji y = 3*sin(x)*exp(-x*2) % na podstawie x, przy dodanym szumie losowym.

Klasyfikacja

1. Klasyfikacja płci pacjentów (Patients Dataset) na podstawie znajomości wieku, wzrostu i wagi

2. MLP Z PODZIAŁEM NA ZBIORY (TRAIN, VALIDATION, TEST) - W tym skrypcie trenujemy sieć neuronową typu MLP (Multi-Layer Perceptron) na zbiorze danych IRIS. Sieć będzie dokonywać klasyfikacji odmian irysów.
3. MLP Z PODZIAŁEM NA ZBIORY (TRAIN, VALIDATION, TEST) – ZBIÓR IONOSPHERE. W tym skrypcie trenujemy sieć neuronową typu MLP (Multi-Layer Perceptron) na zbiorze danych IONOSPHERE. Zbiór zawiera pomiary sygnałów radarowych (X) i etykietę (Y) określającą, czy sygnał jest "good" (g) czy "bad" (b). Sieć dokona binarnej klasyfikacji. Dodatkowo wykorzystujemy walidację (20%)
4. MLP Z PODZIAŁEM NA ZBIORY (TRAIN, VALIDATION, TEST) – ZBIÓR IONOSPHERE, Optymalizacja hiperparametrów MLP na zbiorze Ionosphere z użyciem bayesopt
5. MLP Z PODZIAŁEM NA ZBIORY (TRAIN, VALIDATION, TEST) – ZBIÓR IONOSPHERE, Optymalizacja hiperparametrów MLP na zbiorze Ionosphere z użyciem własnych funkcji

Poniżej znajduje się 5 przykładów zastosowań inżynierskich MLP do własnego opracowania:

1. **Klasyfikacja defektów w produkcji**
    a. Możliwe dane:
        i. Obrazy produktów (np. części elektronicznych, mechanicznych elementów) pobrane z kamer inspekcyjnych,
        ii. Cechy ekstrakcji obrazu, takie jak kształt, tekstura, kolory, krawędzie.
    b. Zadanie: rozpoznawanie, czy produkt jest bez wad, czy zawiera defekty (np. pęknięcia, zarysowania).
2. **Diagnostyka drgań maszyn**
    a. Możliwe dane:
        i. Sygnały akustyczne lub drgania z czujników przyspieszeniowych,
        ii. Cechy takie jak częstotliwości, amplitudy, analiza FFT, energia sygnału w określonych pasmach.
    b. Zadanie: klasyfikacja stanu pracy maszyny – normalna praca vs. awaria lub nieprawidłowości (np. niewyważenie, zużycie łożysk).
3. **Monitorowanie stanu urządzeń przemysłowych (predykcja awarii)**
    a. Możliwe dane:
        i. Dane z czujników: temperatura, ciśnienie, przepływ, prędkość obrotowa, zużycie energii, drgania, itp.,
        ii. Dane historyczne z systemów SCADA lub PLC.

   b. Zadanie: klasyfikacja stanu urządzenia na kategorie, np. „normalny”, „ostrzeżenie” i „awaria”, co pozwala zaplanować konserwację zapobiegawczą.
4. **Analiza jakości materiałów**
   a. Możliwe dane:
      i. Wyniki pomiarów materiałowych, takie jak twardość, gęstość, przewodność, skład chemiczny, struktura mikro, itp.,
      ii. Obrazy mikroskopowe (po odpowiedniej ekstrakcji cech).
   b. Zadanie: klasyfikacja materiału jako wysokiej lub niskiej jakości, albo przypisanie do jednej z kilku kategorii (np. stal niskowęglowa, wysokowęglowa, stopowa).
5. **Rozpoznawanie typów zanieczyszczeń w procesach przemysłowych**
   a. Możliwe dane:
      i. Pomiar właściwości chemicznych i fizycznych, takie jak pH, przewodność, skład chemiczny, zawartość cząstek, itp.,
      ii. Dane z czujników optycznych lub spektroskopowych.
   b. Zadanie: klasyfikacja substancji lub materiałów, które pojawiają się w procesie (np. rodzaj zanieczyszczenia, które może wpływać na jakość produktu), co pozwala na automatyczną kontrolę jakości i reagowanie na nieprawidłowości.