



Polsko-Japońska Akademia Technik Komputerowych

Wydział Informatyki

Zastosowanie filtrów Kalmana do poprawy predykcji cen giełdowych przy użyciu sieci LSTM

Praca Dyplomowa

Autor:	Mikołaj Warda (s28034)
Kierunek studiów:	Informatyka
Specjalizacja:	Data Science
Promotor:	dr Sinh Hoa Nguyen Thi

8 grudnia 2025

Spis treści

1	Wstęp	3
2	Podstawy teoretyczne	4
2.1	Przewidywanie szeregów czasowych na rynkach finansowych	4
2.1.1	Wskaźnik siły względnej (Relative Strength Index, RSI)	5
2.1.2	Wstęgi Bollingera (Bollinger Bands)	5
2.2	Sieci neuronowe w prognozowaniu rynków finansowych	6
2.2.1	Podstawy działania sztucznego neuronu	6
2.2.2	Architektura sieci neuronowej	9
2.2.3	Uczenie sieci neuronowej	11
2.2.4	Sieci rekurencyjne	12
2.2.5	Sieci Long Short-Term Memory	14
2.2.6	Filtr Kalmana	16
3	Metodyka badań	18
3.1	Środowisko badawcze	18
3.2	Przygotowanie danych	19
3.2.1	Pobranie danych	19
3.2.2	Przetwarzanie wstępne i inżynieria cech	19
3.2.3	Skalowanie danych	20
3.2.4	Podział danych	21
3.3	Architektura modeli	21
3.3.1	Model bazowy (Base Model)	21
3.3.2	Model wzbogacony (Enriched Model)	21
3.3.3	Model z filtrem Kalmana (Kalman Model)	21
3.4	Proces uczenia	22
3.4.1	Strojenie hiperparametrów	22
3.4.2	Przebieg procesu uczenia	22
3.4.3	Metryki oceny	23

Streszczenie

1. Wstęp

Prognozowanie cen akcji odgrywa kluczową rolę w finansach, wspierając inwestorów w podejmowaniu świadomych decyzji zarządzania swoim portfolio. Chaotyczny charakter rynków sprawia jednak, że trafne przewidywanie notowań pozostaje trudnym zadaniem. Tradycyjna analiza techniczna bywa niewystarczająca - jest wrażliwa na szum, a wnioski często zawierają element subiektywności [?].

W ostatnich latach dynamiczny rozwój uczenia maszynowego, zwłaszcza sieci neuronowych, znacząco zmienił podejście do modelowania danych czasowych (w tym giełdowych). Architektury takie jak LSTM (Long Short-Term Memory) potrafią uchwycić złożone zależności i długookresowe relacje w danych, co czyni je obiecującymi narzędziami do prognozowania cen [?]. Niemniej jednak ich skuteczność nadal zależy od jakości danych wejściowych. Głównym problemem, jest wysoka wrażliwość na losowe wahania, które są nieodłącznym elementem danych finansowych. Te zniekształcenia wynikają z nieprzewidywalnych zdarzeń rynkowych. Model, zamiast uczyć się rzeczywistych trendów, modeluje wspomniane zakłócenia, obniżając swoją zdolność do generalizacji na nowych danych. W rezultacie, predykcje mogą być obciążone znacznym błędem, co podważa ich użyteczność w podejmowaniu decyzji inwestycyjnych [?].

Można temu zapobiegać stosując techniki filtracji na danych wejściowych. Jednym z klasycznych narzędzi tego typu jest filtr Kalmana, który może pełnić rolę modułu wygładzania i korekty obserwacji, podnosząc stabilność i dokładność predykcji. Trenowanie sieci neuronowej na przefiltrowanych danych przy użyciu filtra Kalmana może zredukować wpływ szumu, pozwalając na lepsze uchwycenie istotnych wzorców i trendów w danych [?, ?].

W ramach niniejszej pracy dyplomowej, podjęto się zbadania skuteczności zastosowania filtra Kalmana jako etapu przetwarzania danych dla modeli LSTM w kontekście prognozowania cen akcji spółki Amazon.com Inc (AMZN) . Wybór tematu pracy wyniknął z chęci zdobycia wiedzy na temat działania architektury LSTM oraz zbadania wpływu filtracji danych na jakość prognoz. Dodatkową motywacją była możliwość wszechstronnego rozwoju ze względu na złożony charakter problemu, łączącego zagadnienia z dziedziny uczenia maszynowego, analizy szeregów czasowych oraz teorii filtrów.

W dalszej części pracy, w sekcji *"Podstawy teoretyczne"*, przedstawiono wszelkie niezbędne pojęcia związane z tematem pracy i przebiegiem badań ze szczególnym naciskiem na architekturę sieci LSTM oraz filtr Kalmana. Następnie, w sekcji *"Metodyka badań"*, opisano podejście badawcze, w tym przygotowanie danych, implementację modelu, metryki oceny oraz wykorzystane narzędzia. Kolejna sekcja *"Wyniki i dyskusja"* prezentuje uzyskane wyniki eksperymentów wraz z ich analizą i interpretacją. Na zakończenie, w sekcji *"Podsumowanie i wnioski"*, podsumowano przeprowadzone badania, przedstawiono kluczowe wnioski oraz zasugerowano kierunki dalszych badań w tym obszarze.

2. Podstawy teoretyczne

2.1. Przewidywanie szeregów czasowych na rynkach finansowych

Dane finansowe, takie jak np. ceny akcji, zawierają obserwacje tworzące szeregi czasowe. Innymi słowy, są to dane, które reprezentują zmiany wartości w określonych odstępach czasu. Formalnie szeregiem czasowym określa się zbiór uporządkowanych obserwacji w czasie, gdzie każda obserwacja jest powiązana z określoną chwilą czasową [?]:

$$X = \{x_1, x_2, x_3, \dots, x_n\} \quad (1)$$

gdzie x_i reprezentuje wartość obserwacji w czasie t_i , a n to liczba obserwacji w szeregu czasowym.

Abstrahując od rynków finansowych, przedstawiono kilka przykładów szeregów czasowych z innych dziedzin, celem lepszego zobrazowania tej koncepcji:

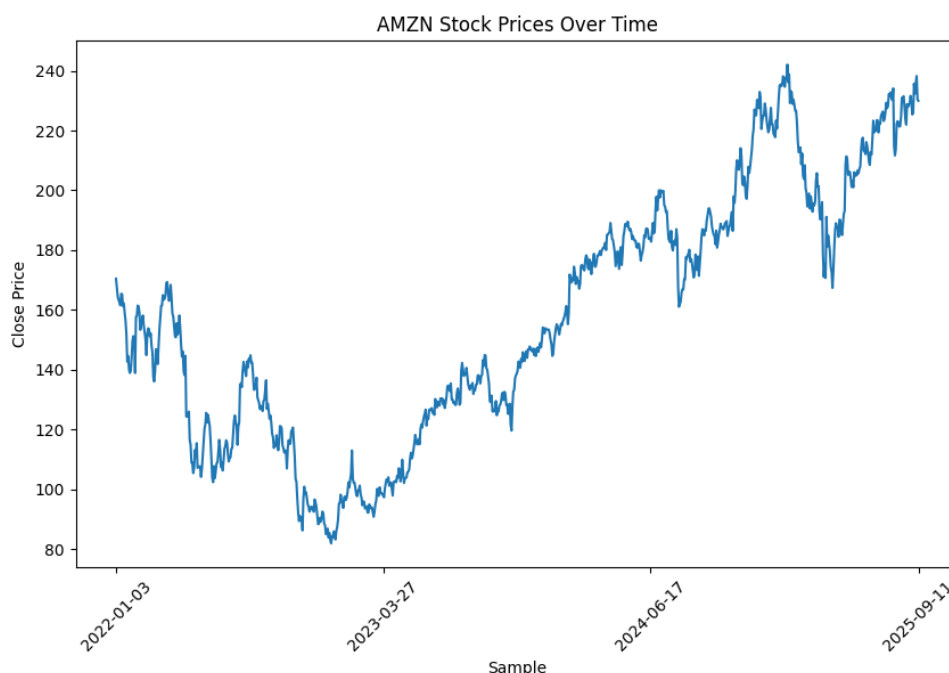
- **Energetyka:** Godzinowe zużycie energii elektrycznej
- **Meteorologia:** Codzienny pomiar temperatury
- **Sport:** Wyniki meczów drużyny na przestrzeni sezonu

Najczęściej spotykanym formatem danych finansowych są tzw. dane OHLC. Przedstawione w takiej postaci informacje tworzą szereg czasowy, w którym każda obserwacja składa się z czterech części [?]:

- **Open (O):** Cena otwarcia - cena, po której dany instrument finansowy rozpoczął notowania w danym okresie.
- **High (H):** Cena najwyższa - najwyższa cena osiągnięta przez instrument finansowy w danym okresie.
- **Low (L):** Cena najniższa - najniższa cena osiągnięta przez instrument finansowy w danym okresie.
- **Close (C):** Cena zamknięcia - cena, po której instrument finansowy zakończył notowania w danym okresie.

Na potrzeby tej pracy, wykorzystano składnik *Close*, ze względu na jego powszechne wykorzystanie w analizie i prognozie cen akcji [?].

Rysunek 1 przedstawia przykładowy wykres cenowy względem składnika *Close* spółki Amazon.com Inc (AMZN) na przestrzeni kilku lat w odstępach dziennych. Dane przedstawione na wykresie są nieregularne i zawierają liczne fluktuacje, co jest charakterystyczne dla danych finansowych. Ta wysoka zmienność stanowi główne wyzwanie dla modeli predykcyjnych, co motywuje do poszukiwania skutecznych metod filtrujących [?].



Rysunek 1: Wykres cenowy względem składnika *Close* Amazon.com Inc (AMZN) w latach 2022-2025. Źródło danych: Yahoo Finance [?].

Tradycyjnie analiza danych rynkowych opierała się o wskaźniki techniczne. Można je określić mianem narzędzi statystycznych reprezentujących różne aspekty zachowań cen [?]. W niniejszej pracy wykorzystano trzy wskaźniki techniczne, które opisano w następnych podsekcjach.

2.1.1 Wskaźnik siły względnej (Relative Strength Index, RSI)

Wskaźnik siły względnej (RSI) to popularny wskaźnik techniczny używany do oceny siły i prędkości zmian cen aktywów finansowych [?]. Wyraża się go wzorem:

$$RSI = 100 - \frac{100}{1 + RS} \quad (2)$$

gdzie *RS* (Relative Strength) to stosunek średnich wzrostów do średnich spadków cen w określonym czasie. Według badań, optymalny okres do obliczania RSI wynosi 14 dni [?].

Produktem końcowym RSI jest liczba z zakresu od 0 do 100, która interpretowana jest następująco:

- Wartości powyżej 70 sugerują, że akcja jest wykupiona i może nastąpić spadek cen.
- Wartości poniżej 30 sugerują, że akcja jest wyprzedana i może nastąpić wzrost cen.
- Wartości pomiędzy 30 a 70 wskazują na neutralny stan rynku.

2.1.2 Wstęgi Bollingera (Bollinger Bands)

Wstęgi Bollingera to narzędzie analizy technicznej, dostarczające informacji o zmienności rynku [?]. Składają się z trzech linii na wykresie cenowym:

- **Środkowa linia:** Prosta średnia krocząca (SMA) obliczona na podstawie cen zamknięcia w określonym czasie. Najczęściej używanym okresem jest 20 dni.
- **Górna wstęga:** Obliczana jako suma wartości środkowej linii i dwukrotności odchylenia standardowego cen w tym okresie.
- **Dolna wstęga:** Obliczana jako różnica wartości środkowej linii i dwukrotności odchylenia standardowego cen w tym okresie.

Na podstawie ww. składowych można zinterpretować dwie nowe miary, które zostały wykorzystane w niniejszej pracy:

- **Bollinger Bandwidth (BBW):** Miara szerokości wstęg Bollingera, obliczana jako stosunek różnicy między górną a dolną wstęgą do środkowej linii [?]:

$$BBW = \frac{UpperBand - LowerBand}{MiddleLine} \quad (3)$$

- **Bollinger %B (BB%):** Miara położenia ceny względem wstęg Bollingera, obliczana jako stosunek różnicy między ceną zamknięcia a dolną wstęgą do różnicy między górną a dolną wstęgą [?]:

$$BB\% = \frac{Close - LowerBand}{UpperBand - LowerBand} \quad (4)$$

Opisane powyżej wskaźniki techniczne - RSI, BBW oraz BB% - dostarczają cennych informacji o dynamice rynku. W tradycyjnej analizie mogą posłużyć do subiektywnej interpretacji przez analityków. W nowoczesnych podejściach, opartych na modelach uczenia maszynowego, można je wykorzystać do wzbogacenia danych o dodatkowe cechy, co potencjalnie może poprawić jakość prognoz [?].

2.2. Sieci neuronowe w prognozowaniu rynków finansowych

Wraz z rozwojem mocy obliczeniowej i technik uczenia maszynowego, pojawiły się bardziej zaawansowane metody analizy i prognozowania w dziedzinie finansów [?]. Do najpopularniejszych podejść należą sieci rekurencyjne (RNN), w tym ich zaawansowane warianty, takie jak *LSTM* (Long Short-Term Memory). Sieci te są zdolne do uchwycenia złożonych wzorców i zależności w szeregach czasowych [?, ?].

W niniejszej sekcji zostały omówione podstawy działania sieci neuronowych. Przedstawiona została budowa pojedynczego neuronu wchodzącego w skład sieci, kluczowe algorytmy uczenia oraz architektura LSTM.

2.2.1 Podstawy działania sztucznego neuronu

Ważnym kamieniem milowym w dziedzinie uczenia maszynowego było wprowadzenie matematycznego modelu neuronu przez McCullocha i Pittsa w 1943 roku [?]. Współcześnie można mówić o różnych architekturach takiego neuronu, jednak ich podstawowa zasada działania pozostaje podobna.

Neuron otrzymuje na wejściu sygnały x_1, x_2, \dots, x_n , które są ważone przez odpowiednie wagi w_1, w_2, \dots, w_n . Następnie, sumuje te ważone sygnały i dodaje do nich wartość obciążenia (bias) b [?]:

$$z = \sum_{i=1}^n w_i x_i + b \quad (5)$$

Otrzymana wartość z , zwana logitem (surowym wyjściem) neuronu, jest następnie przekształcana przez funkcję aktywacji $f(z)$ [?]:

$$\hat{y} = f(z) = f\left(\sum_{i=1}^n w_i x_i + b\right) \quad (6)$$

Funkcja aktywacji modyfikuje logit z , produkując finalne wyjście neuronu \hat{y} – stopień aktywacji. W zależności od zastosowanej funkcji aktywacji, wyjście może przyjmować różne formy [?]. Tą różnicę najprościej zilustrować na przykładzie dwóch popularnych funkcji aktywacji:

- **Funkcja skokowa:**

$$f(z) = \begin{cases} 1 & \text{jeśli } z \geq 0 \\ 0 & \text{jeśli } z < 0 \end{cases} \quad (7)$$

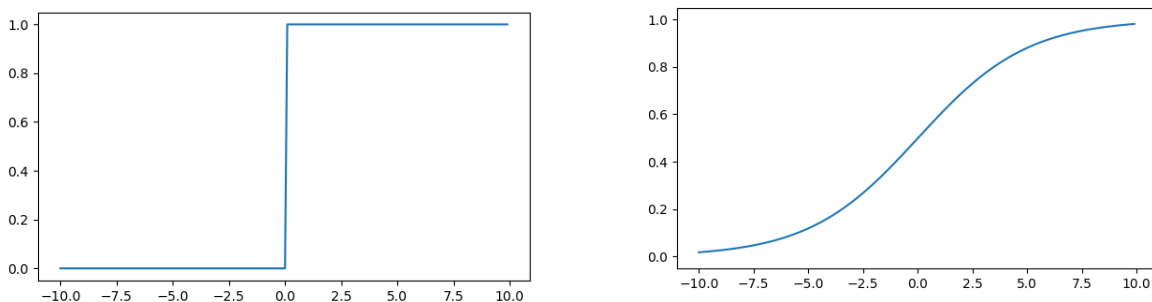
W tym przypadku, wyjście neuronu jest binarne. Innymi słowy, neuron zostanie albo aktywowany (1), albo nieaktywowany (0), w zależności od tego, czy logit z przekracza pewien próg (w tym przypadku 0) [?].

- **Funkcja sigmoidalna:**

$$f(z) = \frac{1}{1 + e^{-z}} \quad (8)$$

W tym przypadku, wyjście neuronu jest ciągłe i mieści się w zakresie od 0 do 1. Oznacza to, że neuron może przyjmować różne poziomy aktywacji, co pozwala na wyrażenie stopnia pewności co do wyniku [?].

Rysunek 2 przedstawia porównanie ww. funkcji aktywacji. Po lewej stronie znajduje się wykres funkcji skokowej, gdzie wyjście nagle zmienia się z 0 na 1 w punkcie $z = 0$. Po prawej stronie znajduje się wykres funkcji sigmoidalnej, gdzie wyjście zmienia się stopniowo od 0 do 1 wraz ze wzrostem wartości z .



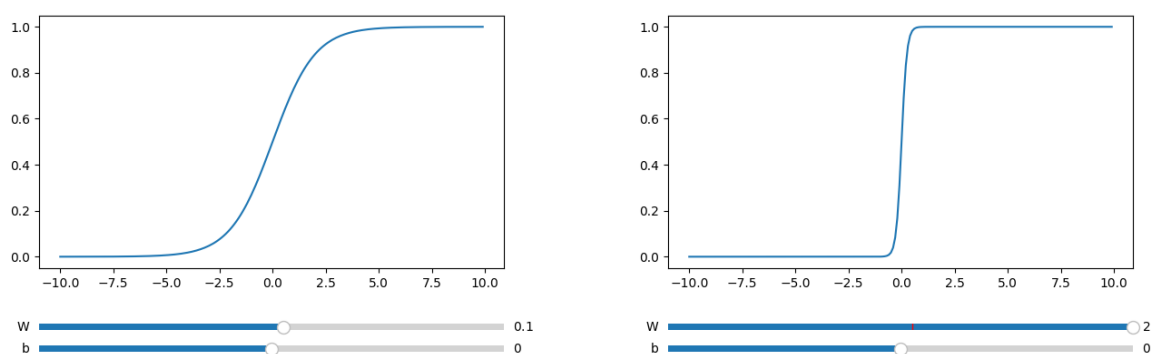
Rysunek 2: Porównanie funkcji aktywacji: (po lewej) funkcja skokowa, (po prawej) funkcja sigmoidalna. Opracowanie własne przy użyciu Matplotlib.

Istnieje wiele innych funkcji aktywacji (np. ReLU, tanh, softmax), a każda z nich ma swoje unikalne właściwości i zastosowania [?]. W przypadku funkcji skokowej, neuron podejmuje twardą decyzję, aktywując się w pełni lub wcale. Przez brak elastyczności, funkcja ta nie pozwala na wyrażenie przekonania co do wyniku [?]. W przeciwieństwie do niej, funkcja sigmoidalna umożliwia płynną aktywację, pozwalając uwzględnić stopień pewności [?].

Znając rolę funkcji aktywacji, można lepiej wyjaśnić działanie parametrów uczonych neuronu, czyli wag w_i oraz obciążenia b .

Wagi określają, jak duży wpływ ma każdy sygnał wejściowy x_i na logit z . Poprzez modelowanie wag, w trakcie procesu uczenia, neuron może nauczyć się, które cechy wejściowe są bardziej istotne dla danego zadania [?].

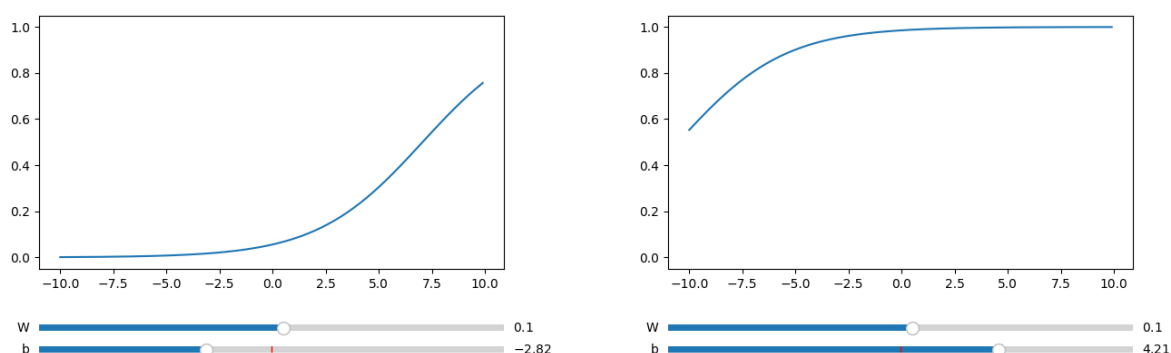
Rysunek 3 przedstawia wpływ różnych wartości wagi w na funkcję sigmoidalną w neuronie z jednym wejściem i bez obciążenia. Po lewej stronie znajduje się wykres dla małej wagi $w = 0.1$, gdzie funkcja zmienia się powoli i ma łagodne nachylenie. Po prawej stronie znajduje się wykres dla dużej wagi $w = 2.0$, gdzie funkcja zmienia się gwałtownie i ma strome nachylenie. Można zaobserwować, że wraz ze wzrostem wartości wagi, funkcja sigmoidalna zaczyna przypominać funkcję skokową. Oznacza to, że neuron staje się pewniejszy swoich decyzji, aktywując się niemal natychmiast po przekroczeniu progu.



Rysunek 3: Wpływ wartości wagi (w) na kształt funkcji sigmoidalnej: (po lewej) $w = 0.1$, (po prawej) $w = 2.0$. Opracowanie własne przy użyciu Matplotlib.

Drugim kluczowym parametrem jest obciążenie, czyli bias b , który przesuwa próg funkcji aktywacji wzdłuż osi poziomej. W istocie umożliwia to opóźnienie lub przyspieszenie momentu, w którym neuron zostaje aktywowany [?].

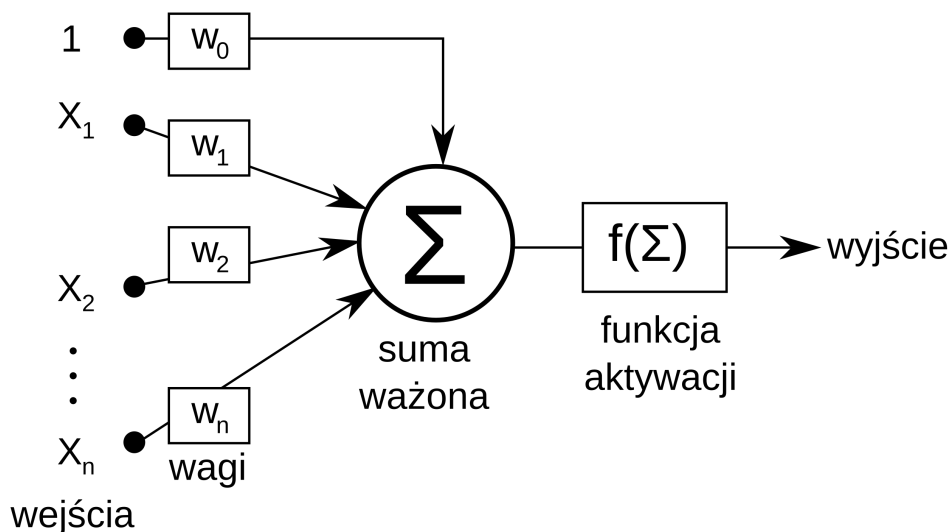
Rysunek 4 przedstawia wpływ różnych wartości biasu b na kształt funkcji sigmoidalnej w neuronie z jednym wejściem i wagą $w = 0.1$.



Rysunek 4: Wpływ wartości bias (b) na kształt funkcji sigmoidalnej: (po lewej) $b = -2.82$, (po prawej) $b = 4.21$.

Opisane powyżej parametry – wagi w_i , obciążenie b oraz funkcja aktywacji $f(z)$ – stanowią podstawę działania sztucznego neuronu [?].

Rysunek 5 podsumowuje budowę sztucznego neuronu, ilustrując jak sygnały wejściowe są przetwarzane przez wagę, sumowane z obciążeniem i przekształcane przez funkcję aktywacji.



Rysunek 5: Schemat budowy sztucznego neuronu. Źródło: [?]

Niestety pojedynczy neuron jest ograniczony w swoich możliwościach – może nauczyć się jedynie prostych zależności liniowo-separowalnych. Inaczej mówiąc, jest w stanie rozróżnić tylko te wzorce, które można oddzielić prostą linią w przestrzeni cech. Aby radzić sobie z bardziej złożonymi problemami, konieczne jest łączenie wielu neuronów w wielowarstwowe struktury [?].

2.2.2 Architektura sieci neuronowej

W celu modelowania bardziej złożonych zależności, łączy się wiele neuronów w struktury zwane sieciami neuronowymi. Składają się one z warstw, gdzie każda warstwa zawiera wiele neuronów. Typowa sieć neuronowa składa się z trzech głównych typów warstw [?]:

1. **Warstwa wejściowa:** Odpowiada za przyjmowanie danych wejściowych.
2. **Warstwy ukryte:** Przetwarzają dane poprzez zestaw neuronów, ucząc się złożonych wzorców.
3. **Warstwa wyjściowa:** Generuje ostateczne prognozy lub klasyfikacje na podstawie przetworzonych danych.

Warstwy można łączyć na różne sposoby, np. poprzez pełne połączenie, konwolucje czy rekurencje [?]. W przypadku pełnego połączenia tworzy się połączenia każdego neuronu z jednej warstwy do każdego neuronu w następnej warstwie [?]. Przepływ sygnału w takiej sieci odbywa się na zasadzie propagacji w przód (ang. forward propagation) – od warstwy wejściowej, przez warstwy ukryte, aż do warstwy wyjściowej.

Formalnie ten proces można opisać krokami obliczeniowymi dla każdej warstwy [?]:

1. **Warstwa wejściowa:**

$$\mathbf{a}^{(0)} = \mathbf{x} \quad (9)$$

2. **Warstwy ukryte:**

$$\mathbf{a}^{(l)} = f(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}) \quad \text{dla } l = 1, 2, \dots, L - 1 \quad (10)$$

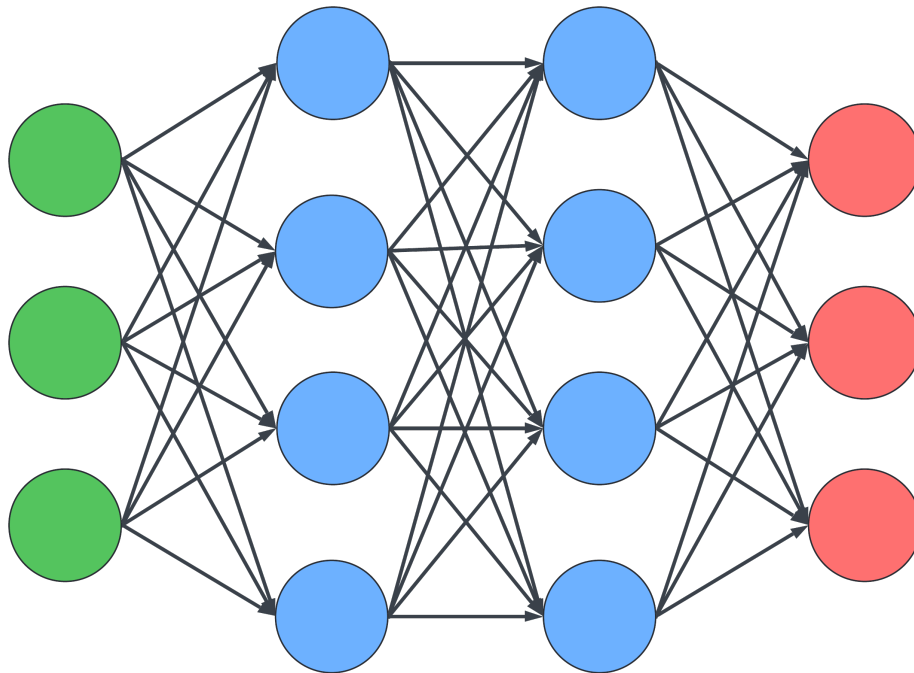
3. **Warstwa wyjściowa:**

$$\hat{\mathbf{y}} = g(\mathbf{W}^{(L)}\mathbf{a}^{(L-1)} + \mathbf{b}^{(L)}) \quad (11)$$

gdzie:

- \mathbf{x} to wektor danych wejściowych,
- $\hat{\mathbf{y}}$ to wektor danych wyjściowych (prognoz),
- $\mathbf{a}^{(l)}$ to aktywacje (wyjścia) warstwy l ,
- $\mathbf{W}^{(l)}$ to macierz wag warstwy l ,
- $\mathbf{b}^{(l)}$ to wektor obciążeń warstwy l ,
- f to funkcja aktywacji dla warstw ukrytych,
- g to funkcja aktywacji dla warstwy wyjściowej,
- L to liczba warstw w sieci.

Rysunek 6 przedstawia schematyczną budowę prostej sieci neuronowej z jedną warstwą wejściową, dwiema warstwami ukrytymi i jedną warstwą wyjściową.



Rysunek 6: Budowa prostej sieci neuronowej z jedną warstwą wejściową, dwiema warstwami ukrytymi i jedną warstwą wyjściową. Opracowanie własne na podstawie GeeksforGeeks.

Przedstawiona architektura oraz proces propagacji w przód definiują przepływ informacji w sieci neuronowej w celu generowania prognoz [?, ?]. Jednakże aby prognoza była trafna, parametry sieci – wartości wag $\mathbf{W}^{(l)}$ oraz obciążeń $\mathbf{b}^{(l)}$ – muszą zostać odpowiednio dobrane [?]. Ten proces nazywany jest uczeniem sieci i zostanie omówiony w następnej podsekcji.

2.2.3 Uczenie sieci neuronowej

Uczenie sieci neuronowej polega na algorytmicznym dostosowywaniu jej parametrów (wag i obciążeń) w celu minimalizacji błędu prognoz. Działa ono w oparciu o zestaw danych zawierających zarówno dane wejściowe \mathbf{x} , jak i odpowiadające im rzeczywiste wartości wyjściowe \mathbf{y} . Innymi słowy, sieć uczy się na podstawie przykładów, aby poprawić swoje prognozy [?].

Aby proces uczenia zwracał rzetelne wyniki, zestaw danych dzieli się na trzy niezależne podzbiory [?]:

- **Zbiór treningowy:** Służy do uczenia sieci poprzez dostosowywanie jej parametrów [?].
- **Zbiór walidacyjny:** Używany do monitorowania wydajności sieci podczas treningu i dostrajania hiperparametrów [?].
- **Zbiór testowy:** Służy do oceny ostatecznej wydajności sieci na niewidzianych wcześniej danych [?].

Sam proces dostosowywania parametrów sieci przeprowadzany jest na zbiorze treningowym i można go opisać w kilku kluczowych krokach [?]:

1. **Inicjalizacja:** Na początku, wagi $\mathbf{W}^{(l)}$ oraz obciążenia $\mathbf{b}^{(l)}$ są inicjalizowane małymi, losowymi wartościami [?].
2. **Propagacja w przód:** Dane wejściowe \mathbf{x} są przekazywane przez sieć, generując prognozy $\hat{\mathbf{y}}$ [?].
3. **Obliczanie błędu:** Prognoza modelu $\hat{\mathbf{y}}$ jest porównywana z rzeczywistą wartością \mathbf{y} za pomocą funkcji kosztu (np. MSE), co pozwala na zmierzenie błędu prognozy [?]:

$$L = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad (12)$$

gdzie L to wartość funkcji kosztu MSE, a n to liczba próbek w zbiorze treningowym.

4. **Propagacja wsteczna błędu:** Obliczony błąd jest propagowany wstecz sieci, od warstwy wyjściowej do wejściowej. Celem tego kroku jest obliczenie gradientów funkcji kosztu w każdym neuronie sieci. Gradienty te wskazują kierunek i wielkość zmian, które należy wprowadzić w parametrach sieci, aby skutecznie minimalizować błąd prognozy. Formalnie gradient jest pochodną funkcji kosztu względem wag lub obciążeń [?]:

$$\frac{\partial L}{\partial \mathbf{W}^{(l)}}, \quad \frac{\partial L}{\partial \mathbf{b}^{(l)}} \quad (13)$$

5. **Aktualizacja parametrów:** Wagi i obciążenia są aktualizowane za pomocą algorytmu optymalizacji (np. Stochastic Gradient Descent, Adam), wykorzystując obliczone gradienty [?]:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial L}{\partial \mathbf{W}^{(l)}}, \quad \mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta \frac{\partial L}{\partial \mathbf{b}^{(l)}} \quad (14)$$

gdzie η to współczynnik nauki (learning rate), określający wielkość kroków aktualizacji.

Proces ten jest powtarzany przez wiele iteracji (epok) na całym zbiorze treningowym, aż do osiągnięcia zadowalającej dokładności prognoz lub spełnienia kryteriów zatrzymania (np. minimalny błąd na zbiorze walidacyjnym). Wynikowo sieć neuronowa uczy się optymalnych wartości wag i obciążeń, co pozwala jej na generowanie trafnych prognoz na niewidzianych wcześniej danych [?].

2.2.4 Sieci rekurencyjne

Tradycyjna architektura sieci neuronowej, opisana w poprzednich podsekcjach, zakłada, że dane wejściowe są niezależne od siebie [?]. W przypadku danych finansowych (szeregi czasowe) takie założenie jest błędne, ponieważ obserwacje są ze sobą powiązane w czasie. Można sobie to wyobrazić na przykładzie cen akcji, gdzie ceny z kilku poprzednich dni wpływają na cenę w dniu dzisiejszym. Wówczas pojawia się potrzeba "pamiętania" wcześniejszych informacji podczas przetwarzania bieżących danych. Sieci rekurencyjne (RNN) zostały zaprojektowane właśnie w tym celu [?].

Podstawowym budulcem sieci RNN jest komórka rekurencyjna, która posiada zdolność do przechowywania stanu ukrytego h_t [?]. Stan ukryty h_t jest nośnikiem informacji z poprzednich kroków czasowych [?]. Aby skutecznie taką informację utrwalić, sieci RNN wykorzystują mechanizm rekurencji, gdzie wyjście z poprzedniego kroku czasowego jest wykorzystywane jako dodatkowe wejście do bieżącego kroku [?].

Formalnie, stan ukryty h_t w czasie t jest wektorem obliczanym na podstawie bieżącego wejścia x_t oraz poprzedniego stanu ukrytego h_{t-1} [?]:

$$h_t = f(W_h h_{t-1} + W_x x_t + b) \quad (15)$$

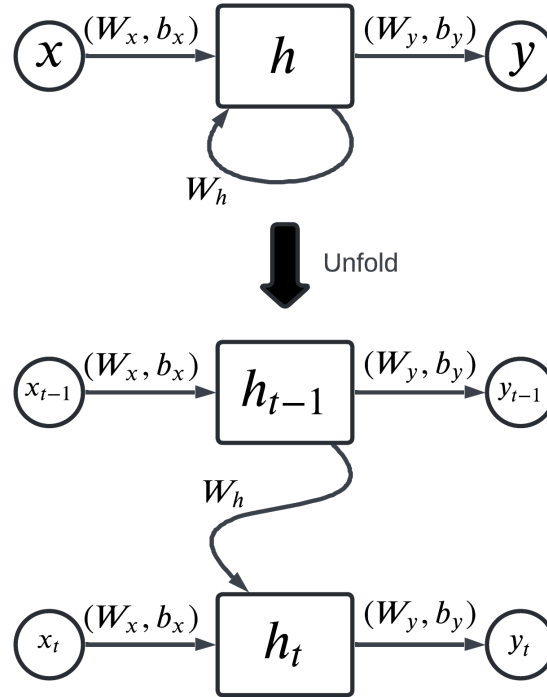
gdzie W_h to macierz wag dla stanu ukrytego, W_x to macierz wag dla wejścia, b to wektor obciążeń, a f to funkcja aktywacji.

Gdy obliczony zostanie ostatni stan ukryty h_t , sieć może wygenerować prognozę \hat{y}_t na podstawie tego stanu [?]:

$$\hat{y}_t = g(W_y h_t + b_y) \quad (16)$$

gdzie W_y to macierz wag dla wyjścia, b_y to wektor obciążeń wyjścia, a g to funkcja aktywacji wyjścia.

Rysunek 7 przedstawia schematyczną budowę komórki rekurencyjnej w sieci RNN. Aby lepiej zobrazować działanie komórki RNN, na rysunku pokazano jej rozwinięcie w czasie. Stan ukryty h_t jest przekazywany z jednego kroku czasowego do następnego.



Rysunek 7: Budowa komórki rekurencyjnej w sieci RNN. Opracowanie własne na podstawie [?].

Taka architektura stanowi solidny fundament do modelowania szeregów czasowych [?]. Niemniej jednak, tradycyjne sieci RNN mają pewne ograniczenia. Są nimi m.in. problem zanikającego i eksplodującego gradientu podczas procesu uczenia, co utrudnia naukę długoterminowych zależności w danych [?, ?]. Aby wyjaśnić na czym polegają te problemy, należy wrócić do procesu opartego na wstecznej propagacji błędów.

W przypadku sieci RNN, odpowiednikiem algorytmu wstecznej propagacji jest algorytm zwany propagacją wsteczną w czasie (ang. *Backpropagation Through Time, BPTT*). Tak jak w przypadku klasycznej propagacji wstecznej, BPTT polega na obliczaniu gradientów funkcji kosztu względem wag i obciążeń sieci. Ze względu na współdzielenie wag w czasie, gradienty są sumowane przez wszystkie kroki czasowe, co daje ostateczny gradient dla każdej wagi [?]:

$$\frac{\partial L}{\partial \mathbf{W}_h} = \sum_{t=1}^T \frac{\partial L}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_h} \quad (17)$$

W praktyce, propagacja błędów w czasie wiąże się z wielokrotnym mnożeniem macierzy wag \mathbf{W}_h odpowiadających za stan ukryty [?]:

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{h}_T} \prod_{k=t+1}^T \frac{\partial \mathbf{h}_k}{\partial \mathbf{h}_{k-1}} \quad (18)$$

To właśnie ten mechanizm prowadzi do dwóch problemów:

- **Problem zanikającego gradientu:** Jeśli wartości w macierzy wag \mathbf{W}_h są małe (norma macierzy mniejsza od 1), to przy wielokrotnym mnożeniu sygnał błędów (gradient) staje się wykładniczo coraz mniejszy. W konsekwencji sieć nie jest w stanie nauczyć się zależności między danymi, które są od siebie odległe w czasie [?].

- **Problem eksplodującego gradientu:** Jest to sytuacja odwrotna. Jeśli wartości w macierzy wag W_h są duże (norma macierzy większa od 1), sygnał błędu przy wielokrotnym mnożeniu rośnie wykładniczo, stając się ogromną liczbą. Na skutek tego aktualizacje wag stają się tak duże, że proces uczenia staje się niestabilny [?].

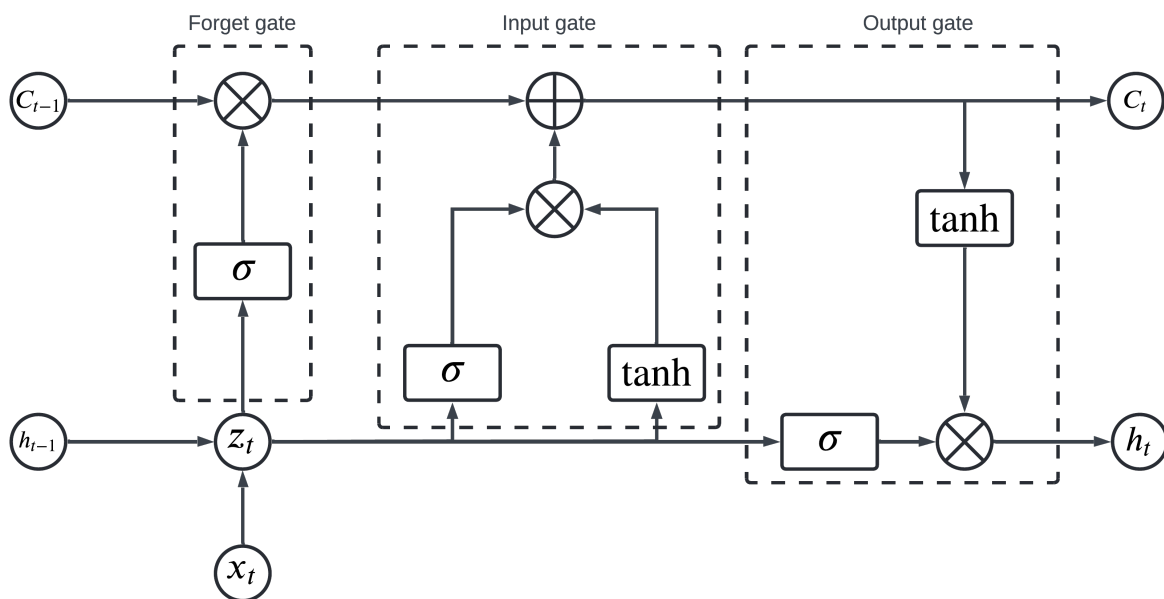
Aby skutecznie radzić sobie z tymi problemami, opracowano zaawansowane architektury sieci rekurencyjnych. Jedną z najpopularniejszych jest Long Short-Term Memory (LSTM), która zostanie omówiona w następnej podsekcji [?].

2.2.5 Sieci Long Short-Term Memory

Sieci Long Short-Term Memory (LSTM) zostały zaprojektowane, aby niwelować problemy zanikającego i eksplodującego gradientu występujące w tradycyjnych sieciach RNN [?]. Cechę tę osiągnięto poprzez wprowadzenie mechanizmów zwanych bramkami (ang. *gates*) oraz dodatkowego stanu komórki C_t [?]. Stan komórki C_t , w odróżnieniu od stanu ukrytego h_t , jest nośnikiem długoterminowych informacji. Dzięki temu LSTM mogą efektywnie przechowywać i wykorzystywać informacje z odległych kroków czasowych [?]. Komórka LSTM składa się z trzech głównych bramek [?]:

1. **Bramka zapominania (Forget Gate):** Decyduje, które informacje z poprzedniego stanu komórki C_{t-1} należy zachować, a które odrzucić.
2. **Bramka wejścia (Input Gate):** Określa, które nowe informacje z bieżącego wejścia x_t powinny zostać dodane do stanu komórki C_t .
3. **Bramka wyjścia (Output Gate):** Kontroluje, które informacje ze stanu komórki C_t zostaną wykorzystane do wygenerowania bieżącego stanu ukrytego h_t .

Rysunek 8 przedstawia schematyczną budowę komórki LSTM wraz z jej trzema bramkami.



Rysunek 8: Budowa komórki LSTM z trzema bramkami: bramką zapominania, bramką wejścia i bramką wyjścia. Opracowanie własne na podstawie [?].

Każda z bramek w komórce LSTM działa na podobnej zasadzie jak tradycyjny neuron, wykorzystując funkcję aktywacji sigmoidalnej do generowania wartości między 0 a 1 [?]. Służy to określeniu stopnia przepuszczania informacji przez daną bramkę. Przepływ informacji w komórce LSTM można opisać następującymi krokami [?]:

1. **Bramka zapominania:** Oblicza się wartość bramki zapominania f_t na podstawie bieżącego wejścia x_t oraz poprzedniego stanu ukrytego h_{t-1} . Otrzymana wartość decyduje, które informacje z poprzedniego stanu komórki C_{t-1} należy zachować [?]:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad (19)$$

$$C_t = f_t * C_{t-1} \quad (20)$$

2. **Bramka wejścia:** Oblicza się wartość bramki wejścia i_t oraz tworzy się kandydat na nowe informacje \tilde{C}_t na podstawie bieżącego wejścia x_t oraz poprzedniego stanu ukrytego h_{t-1} . Następnie, przy pomocy nowych informacji aktualizuje się stan komórki C_t [?]:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (21)$$

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \quad (22)$$

$$C_t = C_t + i_t * \tilde{C}_t \quad (23)$$

3. **Bramka wyjścia:** Oblicza się wartość bramki wyjścia o_t na podstawie bieżącego wejścia x_t oraz poprzedniego stanu ukrytego h_{t-1} . Następnie, generuje się bieżący stan ukryty h_t na podstawie zaktualizowanego stanu komórki C_t [?]:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (24)$$

$$h_t = o_t * \tanh(C_t) \quad (25)$$

gdzie:

- σ to funkcja aktywacji sigmoidalnej,
- \tanh to funkcja aktywacji tangens hiperboliczny,
- W_f, W_i, W_C, W_o to macierze wag dla poszczególnych bramek,
- b_f, b_i, b_C, b_o to wektory obciążeń dla poszczególnych bramek.

Kluczem do skutecznej minimalizacji problemu zanikającego gradientu w LSTM jest mechanizm bramek, który umożliwia selektywne przepuszczanie informacji przez stan komórki C_t [?]. Zamiast wielokrotnego mnożenia wag, jak ma to miejsce w tradycyjnych RNN, LSTM wykorzystują operacje dodawania do aktualizacji stanu komórki oraz mnożenia do określenia stopnia zachowania informacji. Dzięki takiej architekturze, gradienty mogą przepływać przez sieć w sposób bardziej kontrolowany, co znacząco ogranicza ryzyko ich zanikania lub eksplodowania i poprawia stabilność procesu uczenia [?].

2.2.6 Filtr Kalmana

Sieci neuronowe (w tym LSTM) są potężnymi narzędziami do modelowania danych. Jednakże w praktyce, dane często zawierają szumy i fluktuacje, które wprowadzają niepożądane błędy w prognozach [?]. W przypadku cen akcji, takie zakłócenia mogą wynikać z nagłych wydarzeń rynkowo-politycznych, np.: niespodziewanych decyzji regulacyjnych czy kryzysów gospodarczych [?]. Aby skutecznie minimalizować wpływ takich szumów, stosuje się różne techniki filtracji danych.

Ogólne pojęcie filtracji odnosi się do procesu usuwania niepożądanych części sygnału, w celu uzyskania czystszy sygnału [?]. Jedną z zaawansowanych metod filtracji jest filtr Kalmana. Filtr Kalmana to algorytm rekurencyjny służący do estymacji stanu dynamicznego systemu (np. poruszającego się pojazdu, cen akcji) [?]. Działa on poprzez łączenie dwóch źródeł informacji:

- **Modelu predykcyjnego:** Opisuje, jak stan systemu zmienia się w czasie.
- **Pomiarów obserwacyjnych:** Dostarcza informacji o aktualnym stanie systemu.

Obydwie z ww. informacji są obarczone niepewnością – model predykcyjny może nie uwzględniać wszystkich czynników wpływających na system, a pomiary mogą być zanieczyszczone szumem [?]. Z tego powodu filtr Kalmana wykorzystuje podejście probabilistyczne, aby w sposób optymalny łączyć te dwie informacje i uzyskać jak najlepszą estymację stanu systemu [?]. Proces działania filtru Kalmana można podzielić na trzy główne kroki [?]:

1. **Inicjalizacja stanu:** Na początku definiuje się początkowy stan systemu x oraz macierz kowariancji błędu estymacji P [?]. Na początku można założyć, że stan jest nieznany, więc macierz P jest ustawiona na dużą wartość, co odzwierciedla wysoką niepewność.

Kluczowe jest również zdefiniowanie macierzy opisujących model systemu:

- **F – macierz przejścia stanu**, która opisuje, zmianę stanu systemu w czasie.
- **H – macierz obserwacji**, która mapuje prawdziwy stan systemu x na przestrzeń pomiarową z .
- **Q – macierz kowariancji szumu procesu**, która reprezentuje niepewność związaną z modelem predykcyjnym (np. czynniki nieujęte w modelu).
- **R – macierz kowariancji szumu pomiarowego**, która reprezentuje niepewność związaną z samymi pomiarami (np. niedokładność czujnika).
- **B – macierz sterowania** (opcjonalna), która opisuje wpływ zewnętrznego sterowania u na stan systemu.

2. **Predykcja:** Wyznacza się przewidywany stan systemu \hat{x} oraz przewidywaną macierz kowariancji błędu estymacji \hat{P} [?]:

$$\hat{x} = Fx + Bu \quad (26)$$

$$\hat{P} = FPF^T + Q \quad (27)$$

3. **Aktualizacja:** Gdy dostępny jest nowy pomiar z , wyznacza się resztę pomiarową (ang. residual) y , oraz zysk Kalmana K (ang. Kalman gain) [?]:

$$y = z - H\hat{x} \quad (28)$$

$$\mathbf{K} = \hat{\mathbf{P}}\mathbf{H}^T(\mathbf{H}\hat{\mathbf{P}}\mathbf{H}^T + \mathbf{R})^{-1} \quad (29)$$

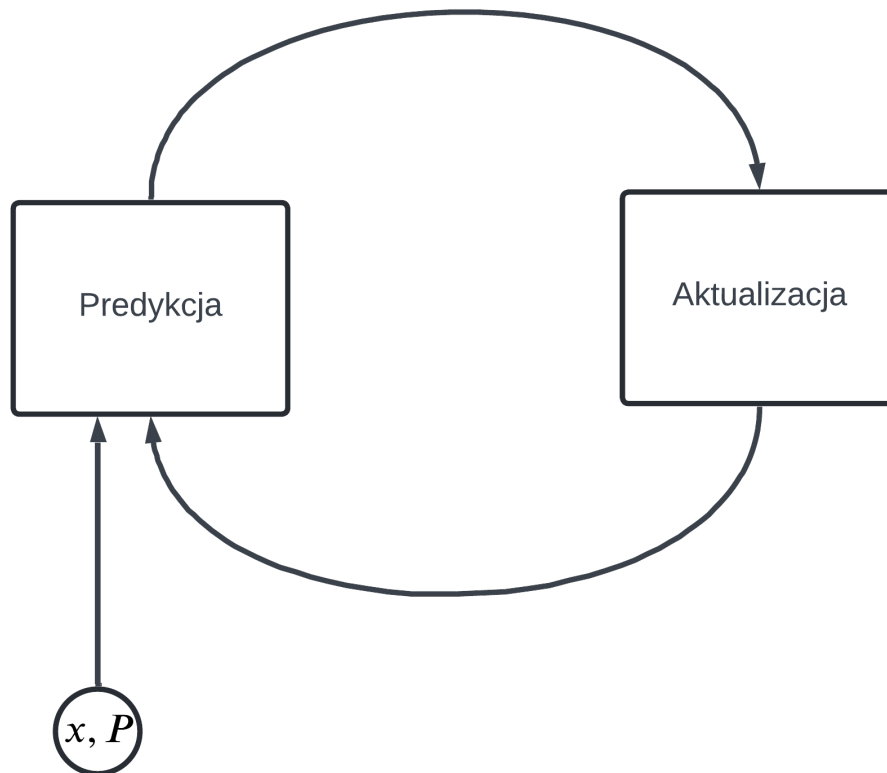
Następnie, aktualizuje się estymowany stan \mathbf{x} oraz macierz kowariancji błędu estymacji \mathbf{P} :

$$\mathbf{x} = \hat{\mathbf{x}} + \mathbf{K}\mathbf{y} \quad (30)$$

$$\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{H})\hat{\mathbf{P}} \quad (31)$$

Zaktualizowane wartości stanu \mathbf{x} oraz macierzy \mathbf{P} są następnie wykorzystywane w kolejnym kroku predykcji.

Rysunek 9 przedstawia schematyczny cykl działania filtra Kalmana, ilustrując kroki predykcji i aktualizacji.



Rysunek 9: Cykl działania filtra Kalmana, ilustrujący kroki predykcji i aktualizacji. Opracowanie własne na podstawie [?].

W kontekście prognozowania cen akcji, filtr Kalmana może być użyty do odszumienia danych w fazie ich wstępnego przetwarzania [?]. Dzięki temu sieć LSTM otrzymuje czystsze dane wejściowe, co może prowadzić do bardziej trafnych prognoz. Integracja filtra Kalmana z siecią LSTM pozwala na skuteczne łączenie zalet obu metod: zdolności LSTM do modelowania złożonych zależności czasowych oraz umiejętności filtra Kalmana do redukcji szumów w danych [?].

3. Metodyka badań

Celem przeprowadzonych badań było porównanie skuteczności predykcji cen akcji przy użyciu tradycyjnej sieci LSTM oraz sieci LSTM wspomaganej filtrem Kalmana. W tym celu zaprojektowano i zaimplementowano trzy odrębne modele, z których każdy służył weryfikacji innej hipotezy badawczej:

- **Model bazowy (Base Model):** Stanowił punkt odniesienia (ang. *baseline*). Jego zadaniem było prognozowanie cen wyłącznie na podstawie ich historycznych wartości, przy użyciu prostej architektury LSTM.
- **Model wzbogacony (Enriched Model):** Miał na celu weryfikację, czy dostarczenie modelowi dodatkowego kontekstu rynkowego w postaci wskaźników analizy technicznej poprawi jakość predykcji.
- **Model z filtrem Kalmana (Kalman Model):** Służył do zbadania głównej hipotezy pracy – czy wstępna filtracja i odsumienie danych wejściowych przy użyciu filtru Kalmana pozytywnie wpływa na skuteczność prognoz sieci LSTM.

W niniejszym rozdziale przedstawiono szczegółowy opis metodologii badawczej zastosowanej w pracy, obejmujący proces przygotowania danych, implementację modeli, procedurę uczenia oraz metryki oceny wydajności.

3.1. Środowisko badawcze

Do przeprowadzenia eksperymentów wykorzystano język programowania Python 3.12 oraz następujące biblioteki:

- **Pandas:** Biblioteka służąca do manipulacji i analizy danych, w szczególności do pracy z danymi tabelarycznymi i szeregami czasowymi. Została wykorzystana do wczytywania, czyszczenia oraz transformacji danych giełdowych.
- **NumPy:** Podstawowy pakiet do obliczeń naukowych w Pythonie, zapewniający wsparcie dla wielowymiarowych tablic i macierzy oraz szerokiej gamy funkcji matematycznych.
- **PyTorch:** Jedna z wiodących platform do uczenia głębokiego, wykorzystana do zdefiniowania architektury sieci LSTM, implementacji pętli treningowej oraz przeprowadzenia procesu uczenia modeli.
- **Optuna:** Biblioteka do automatycznej optymalizacji hiperparametrów modeli uczenia maszynowego, która została użyta do znalezienia optymalnych wartości takich parametrów jak liczba warstw LSTM, liczba neuronów w warstwach, współczynnik nauki oraz wielkość partii treningowej (batch size).
- **Scikit-learn:** Kompleksowa biblioteka do uczenia maszynowego, z której wykorzystano m.in. narzędzia do skalowania danych (MinMaxScaler) oraz obliczania metryk oceny modeli (MSE, RMSE).
- **FilterPy:** Specjalistyczna biblioteka dostarczająca implementacje różnych algorytmów filtrujących, w tym filtru Kalmana, który został użyty do odsumiania danych wejściowych.

- **Technical Analysis (ta):** Biblioteka umożliwiająca obliczanie wskaźników analizy technicznej, które zostały wykorzystane do wzbogacenia zestawu cech wejściowych modeli.
- **Matplotlib & Seaborn:** Biblioteki do wizualizacji danych, które posłużyły do generowania wszystkich wykresów przedstawionych w pracy, m.in. wykresów cen, krzywych uczenia i porównań predykcji z wartościami rzeczywistymi.
- **yfinance:** Biblioteka umożliwiająca pobieranie historycznych danych rynkowych z serwisu Yahoo! Finance.

Całość kodu została zorganizowana w ramach projektu Kedro, co zapewniło modularność, reużywalność oraz łatwość zarządzania eksperymentami.

3.2. Przygotowanie danych

Proces przygotowania danych stanowił pierwszy etap badań. Jego celem było uzyskanie oczyszczonego i odpowiednio sformatowanego zestawu danych giełdowych spółki Amazon (AMZN), który mógł zostać użyty do trenowania i testowania modeli LSTM. Proces ten obejmował następujące kroki odpowiednio: pobrania danych, przetwarzania wstępnego, inżynierii cech oraz podziału na zbiory treningowy, walidacyjny i testowy [?].

3.2.1 Pobranie danych

Dane historyczne dotyczące cen akcji spółki Amazon zostały pobrane z serwisu Yahoo! Finance przy użyciu biblioteki yfinance. Zbiór danych obejmował dzienne notowania z okresu od 3 stycznia 2022 roku do 11 września 2025 roku. Każdy rekord w surowym zbiorze danych zawierał informacje OHLC (Open, High, Low, Close), wolumen obrotu oraz datę notowania [?]. Tabela 1 przedstawia przykładowe rekordy z pobranego zbioru danych.

Tabela 1: Przykładowe rekordy z surowego zbioru danych dla spółki AMZN. Źródło: [?]

Date	Open	High	Low	Close	Volume
2022-01-03	170.40	170.70	166.16	167.55	63520000
2022-01-04	167.52	171.40	166.35	170.44	70726000
2022-01-05	164.36	167.13	164.36	166.88	64302000
2022-01-06	163.25	164.80	161.94	163.45	51958000
2022-01-07	162.55	165.24	162.03	163.84	46606000
...

3.2.2 Przetwarzanie wstępne i inżynieria cech

Surowe dane w formie szeregu czasowego nie są bezpośrednio kompatybilne z architekturą sieci LSTM, która oczekuje danych w formacie nadzorowanym, tj. par (sekwencja wejściowa, wartość docelowa). W związku z tym, pierwszym krokiem było przekształcenie danych do odpowiedniego formatu, który zawierałby sekwencje historycznych cen jako wejścia oraz odpowiadające im przyszłe ceny jako wartości docelowe [?]. Na potrzeby przeprowadzenia eksperymentów podjęto decyzję uwzględnieniu cen zamknięcia (Close) z trzech poprzednich dni jako cech wejściowych do przewidywania ceny zamknięcia w dniu następnym. Tabela 2 przedstawia przykładowe rekordy z przetworzonego zbioru danych.

Tabela 2: Przykładowa struktura danych po przekształceniu do formatu nadzorowanego.

Close	Close (t-1)	Close (t-2)	Close (t-3)
163.25	164.36	167.52	170.40
162.55	163.25	164.36	167.52
161.49	162.55	163.25	164.36
165.36	161.49	162.55	163.25
165.21	165.36	161.49	162.55
...

Po tym fundamentalnym etapie, który był wspólny dla wszystkich eksperymentów, dalsze kroki przetwarzania i inżynierii cech zostały zróżnicowane. Powodem tego była potrzeba przygotowania trzech odrębnych wariantów zbioru danych, z których każdy był dedykowany jednemu z porównywanych modeli.

1. **Zbiór dla modelu bazowego (Base Model):** W tym wariantcie nie stosowano zaawansowanej inżynierii cech. Jediną cechą wejściową była historyczna cena zamknięcia (Close).
2. **Zbiór dla modelu wzbogaconego (Enriched Model):** Ten zbiór został rozszerzony o dodatkowe cechy uzyskane w procesie inżynierii cech. Do ceny zamknięcia dołączono trzy wskaźniki analizy technicznej: wskaźnik siły względnej (RSI), szerokość wstęg Bollingera (BBW) oraz procentowe położenie ceny względem wstęg (%B).
3. **Zbiór dla modelu z filtrem Kalmana (Kalman Model):** W tym podejściu cena zamknięcia została poddana procesowi filtracji przy użyciu filtra Kalmana.

Po zdefiniowaniu cech dla każdego z trzech wariantów, wszystkie zbiory zostały poddane tym samym, końcowym etapom przetwarzania, aby zapewnić spójność i rzetelność porównania.

3.2.3 Skalowanie danych

Skalowanie cech wejściowych jest kluczowym krokiem w przygotowaniu danych do trenowania sieci neuronowych – przyspiesza oraz stabilizuje proces uczenia sieci neuronowej [?]. W ramach niniejszej pracy zbadano i porównano dwie strategie skalowania danych:

- **Normalizacja (Min-Max Scaling):** Przekształca wartości cech do z góry określonego zakresu, w tym przypadku [0, 1].
- **Standaryzacja (Z-score Standardization):** Przekształca dane tak, aby miały średnią równą 0 i odchylenie standardowe równe 1.

Ważnym aspektem metodologicznym było dopasowanie skalera wyłącznie na danych treningowych. Następnie ten sam, dopasowany skaler został użyty do transformacji zarówno zbioru walidacyjnego, jak i testowego. Takie podejście zapobiega "przeciekowi danych"(ang. data leakage) z przyszłości do modelu, co jest niezbędne dla uzyskania wiarygodnych i rzetelnych wyników [?].

3.2.4 Podział danych

Po zakończeniu wszystkich etapów przetwarzania i inżynierii cech, zbiory danych zostały podzielone na trzy odrębne części:

- **Zbiór treningowy (Training Set):** Obejmuje 72% danych i służy do trenowania modeli LSTM.
- **Zbiór walidacyjny (Validation Set):** Obejmuje 18% danych i jest wykorzystywany do monitorowania wydajności modeli podczas treningu oraz do strojenia hiperparametrów.
- **Zbiór testowy (Test Set):** Obejmuje pozostałe 10% danych i służy do ostatecznej oceny wydajności wytrenowanych modeli.

Podział danych został przeprowadzony w sposób chronologiczny, aby zachować naturalną kolejność czasową szeregów danych giełdowych i uniknąć potencjalnych problemów związanych z przeciekami danych [?].

3.3. Architektura modeli

W niniejszej pracy porównano trzy różne modele LSTM, z których każdy charakteryzował się odmiennym podejściem do przetwarzania danych wejściowych. Wszystkie modele zostały zaimplementowane przy użyciu biblioteki PyTorch, co zapewniło elastyczność i wydajność niezbędną do przeprowadzenia eksperymentów [?].

Chociaż ogólny schemat tworzenia modeli był wspólny, to jednak poszczególne modele różniły się kluczowymi hiperparametrami architektury, takimi jak wymiar danych wejściowych, liczba warstw LSTM oraz liczba neuronów w warstwach ukrytych. Różnice te wynikały z różnicy w charakterystyce danych wejściowych, których dany model miał używać do predykcji cen akcji. Poniżej przedstawiono szczegółowy opis każdego z trzech modeli.

3.3.1 Model bazowy (Base Model)

Model bazowy stanowił najprostszą implementację, zaprojektowaną w celu ustanowienia punktu odniesienia (ang. *baseline*). Architektura modelu bazowego składała się z jednej warstwy LSTM z 27 neuronami w warstwie ukrytej, co zapewniało wystarczającą zdolność modelowania prostych zależności czasowych w danych wejściowych.

3.3.2 Model wzbogacony (Enriched Model)

Model wzbogacony rozszerzał architekturę modelu bazowego poprzez dodanie trzech wskaźników analizy technicznej (RSI, BBW, %B) jako dodatkowych cech wejściowych. Z tego względu zastosowano bardziej złożoną architekturę, obejmującą dwie warstwy LSTM oraz większą liczbę neuronów w warstwach ukrytych.

3.3.3 Model z filtrem Kalmana (Kalman Model)

Model ten operował na takich samych danych jak model bazowy, jednak przed podaniem ich na wejście sieci LSTM, dane te były poddawane procesowi filtracji przy użyciu filtru Kalmana. Jego architektura była tożsama z architekturą modelu bazowego, co pozwalało na bezpośrednie porównanie wpływu samej filtracji danych na wyniki predykcji.

Tabela 3 przedstawia zestawienie kluczowych hiperparametrów architektury każdego z trzech modeli LSTM użytych w eksperymentach.

Tabela 3: Zestawienie kluczowych hiperparametrów architektury modeli.

Hiperparametr	Base Model	Enriched Model	Kalman Model
Wymiar wejścia	1	4	1
Liczba warstw LSTM	1	2	1
Liczba neuronów ukrytych	27	64	27
Wymiar wyjścia	1	1	1

Podsumowując, architektury trzech modeli zostały świadomie zróżnicowane, aby umożliwić rzetelne zbadanie postawionych hipotez. Model bazowy i model z filtrem Kalmana posiadały identyczną, prostą architekturę, co pozwoliło na wyizolowanie i ocenę wpływu samej filtracji danych. Z kolei model wzbogacony, ze względu na wielowymiarowy charakter danych wejściowych, został wyposażony w bardziej złożoną strukturę, zdolną do przetwarzania bogatszego zestawu informacji. W kolejnej sekcji opisano proces uczenia i ewaluacji tych modeli.

3.4. Proces uczenia

Proces uczenia obejmował strojenie hiperparametrów, wybór funkcji straty oraz algorytmu optymalizacji, właściwy trening na danych treningowych oraz monitorowanie wydajności na zbiorze walidacyjnym. Każdy z trzech modeli był trenowany niezależnie, z zastosowaniem tych samych procedur, aby zapewnić odpowiednią porównywalność wyników.

3.4.1 Strojenie hiperparametrów

W celu znalezienia optymalnych wartości hiperparametrów dla każdego z modeli, zastosowano bibliotekę Optuna do automatyzacji tego procesu [?]. Przeszukiwana przestrzeń hiperparametrów była wspólna dla wszystkich modeli i obejmowała:

- **Liczba warstw LSTM:** wartości całkowite z zakresu [1, 3].
- **Liczba neuronów w warstwach ukrytych:** wartości całkowite z zakresu [16, 128].
- **Współczynnik nauki (learning rate):** wartości z rozkładu logarytmicznego w zakresie [1e-4, 1e-2].
- **Wielkość partii treningowej (batch size):** wybór spośród wartości [16, 32, 64, 128].
- **Liczba epok treningowych:** wartości całkowite z zakresu [10, 100].

Dla każdego modelu przeprowadzono 50 prób optymalizacji (ang. trials), a najlepsze zestawy hiperparametrów zostały następnie w ostatecznym treningu modeli.

3.4.2 Przebieg procesu uczenia

Finalny trening modeli przeprowadzono na zbiorze treningowym, wykorzystując zoptymalizowane zestawy hiperparametrów. Jako funkcję kosztu, której wartość była minimalizowana w trakcie uczenia, wybrano błąd średniokwadratowy (MSE). Do aktualizacji

wag modelu posłużył optymalizator Adam, będący standardem w zadaniach uczenia głębokiego ze względu na jego efektywność i adaptacyjny dobór współczynnika uczenia [?].

Liczba epok, przez które trenowano każdy model, została również ustalona jako jeden z hiperparametrów w procesie automatycznego strojenia. Wydajność modelu w trakcie uczenia była na bieżąco monitorowana na zbiorze walidacyjnym. Pozwoliło to na generowanie krzywych uczenia i weryfikację, czy model nie wykazuje oznak znacznego przeuczenia (ang. *overfitting*).

3.4.3 Metryki oceny

Do oceny wydajności modeli w trakcie procesu uczenia zastosowano trzy metryki:

- **Mean Squared Error (MSE):** Miara średniego kwadratu różnic między wartościami przewidywanymi a rzeczywistymi.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (32)$$

gdzie y_i to rzeczywista wartość, \hat{y}_i to wartość przewidywana przez model, a n to liczba próbek.

- **Root Mean Squared Error (RMSE):** Pierwiastek kwadratowy z MSE, który daje wynik w tych samych jednostkach co dane wejściowe.

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (33)$$

gdzie y_i to rzeczywista wartość, \hat{y}_i to wartość przewidywana przez model, a n to liczba próbek.

- **R-squared (R^2):** Miara dopasowania modelu, wskazująca, jaka część wariancji danych jest wyjaśniona przez model.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (34)$$

gdzie y_i to rzeczywista wartość, \hat{y}_i to wartość przewidywana przez model, \bar{y} to średnia, a n to liczba próbek.

Cały proces uczenia został zaimplementowany jako pipeline w ramach projektu Kedro, co pozwoliło na ustrukturyzowanie i zautomatyzowanie wszystkich opisanych kroków. Dzięki modularnej naturze Kedro, każdy etap – od strojenia hiperparametrów za pomocą Optuna, po finalny trening modeli – został zdefiniowany jako oddzielny, reużywalny węzeł (ang. *node*). Parametry procesu, takie jak przeszukiwane zakresy hiperparametrów czy liczba prób optymalizacji, zostały odseparowane od logiki kodu i umieszczone w plikach konfiguracyjnych. Takie podejście zapewniło nie tylko przejrzystość i łatwość zarządzania eksperymentami, ale przede wszystkim gwarantowało powtarzalność (ang. *reproducibility*) procesu uczenia dla każdego z trzech modeli [?]. Po zakończeniu treningu, wytrenowane modele zostały poddane ewaluacji na zbiorze testowym.