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

18 listopada 2025

Spis treści

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 obarczone 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 filtru 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 [**?**]:

$$X = \{x_1, x_2, x_3, \ldots, x_n\} \tag{1}$$

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:

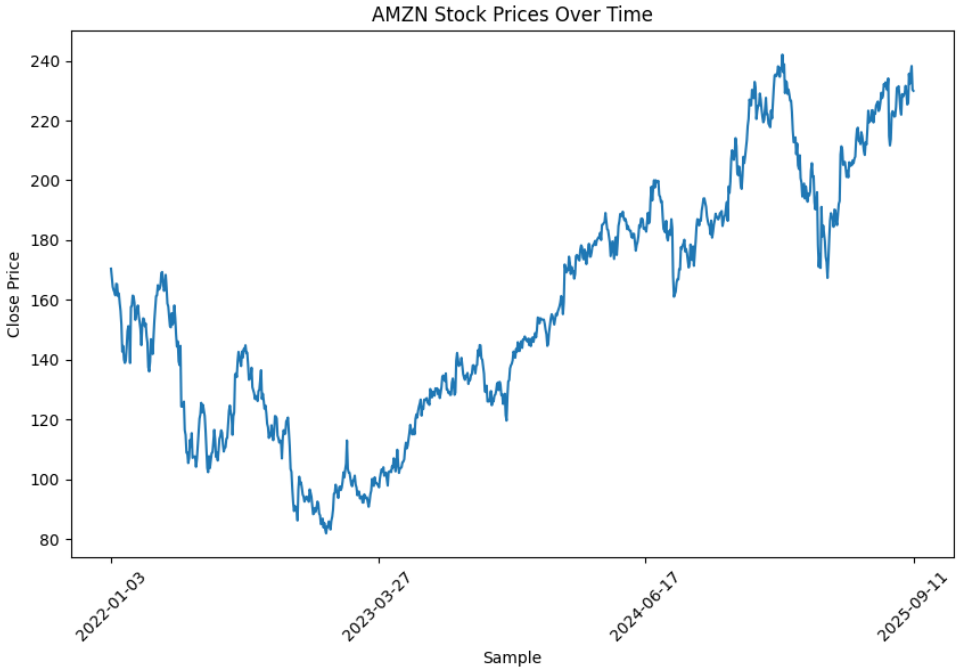
- **Energetyka:** Godzinowe zużycie energii elektrycznej
- **Meteorologia:** Codzienny pomiar temperatury
- **Sport:** Wyniki meczów drużyny na przestrzeni sezonu

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:

$$RSI = 100 - \frac{100}{1 + RS} \tag{2}$$

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 [**?**]. Składaja 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 [**?**]:

$$BBW = \frac{UpperBand - LowerBand}{MiddleLine} \tag{3}$$

- **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 [**?**]:

$$BB\% = \frac{Close - LowerBand}{UpperBand - LowerBand} \tag{4}$$

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 Pitts'a w 1943 roku [**?**]. Współcześnie można mówić o różnych architekturach takiego neuronu, jednak ich podstawowa zasada działania pozostaje podobna.

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

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

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

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

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:**

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

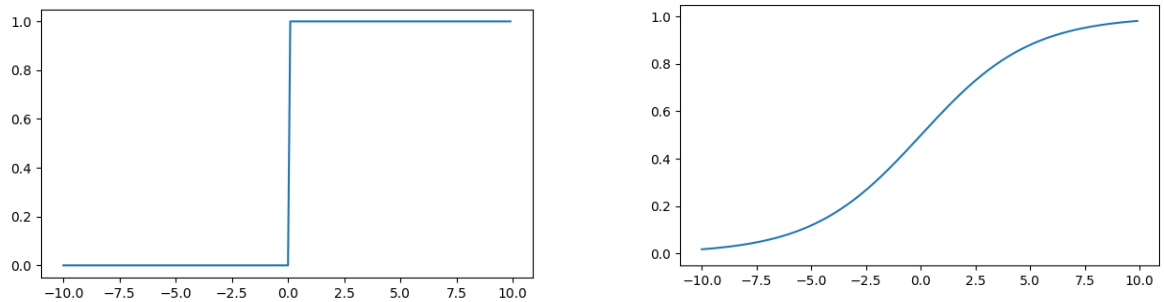
  W tym przypadku, wyjście neuronu jest binarne. Innymi słowy, neuron zostanie albo aktywowany (1), albo nieaktywowany (0), w zależności od tego, czy logit z przekracza pewien próg (w tym przypadku 0) [**?**].

- **Funkcja sigmoidalna:**

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

  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.



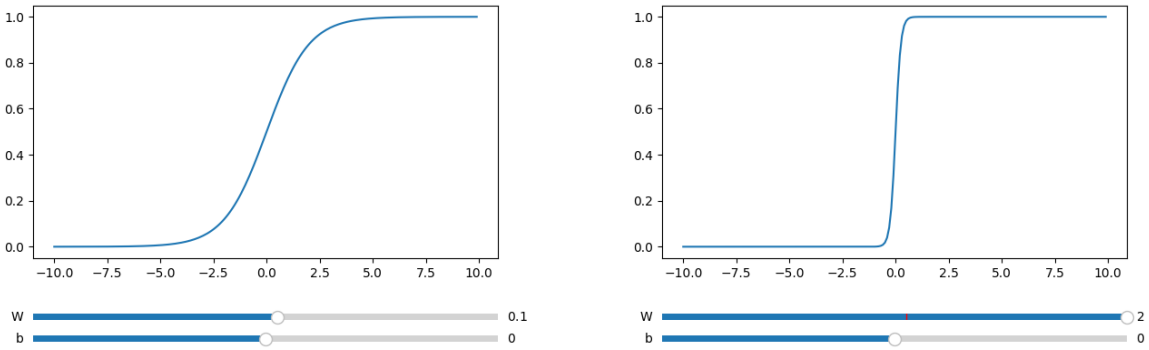
Rysunek 2: Porównanie funkcji aktywacji: (po lewej) funkcja skokowa, (po prawej) funkcja sigmoidalna. Opracownanie własne przy użyciu Matplotlib.

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 [**?**].

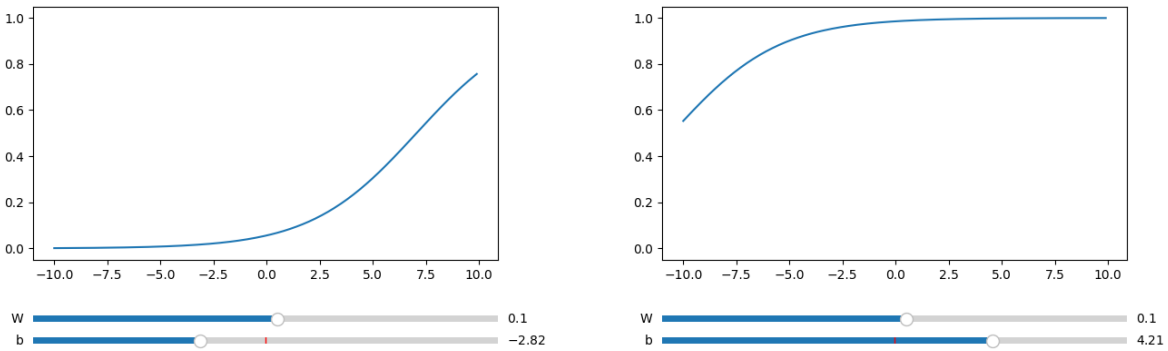
Rysunek 3 przedstawia wpływ różnych wartości wagi w na funkcję sigmoidalną w neuronie z jednym wejściem i bez obciążenia. Po lewej stronie znajduje się wykres dla małej wagi $w = 0.1$, gdzie funkcja zmienia się powoli i ma łagodne nachylenie. Po prawej stronie znajduje się wykres dla dużej wagi $w = 2.0$, 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 [**?**].

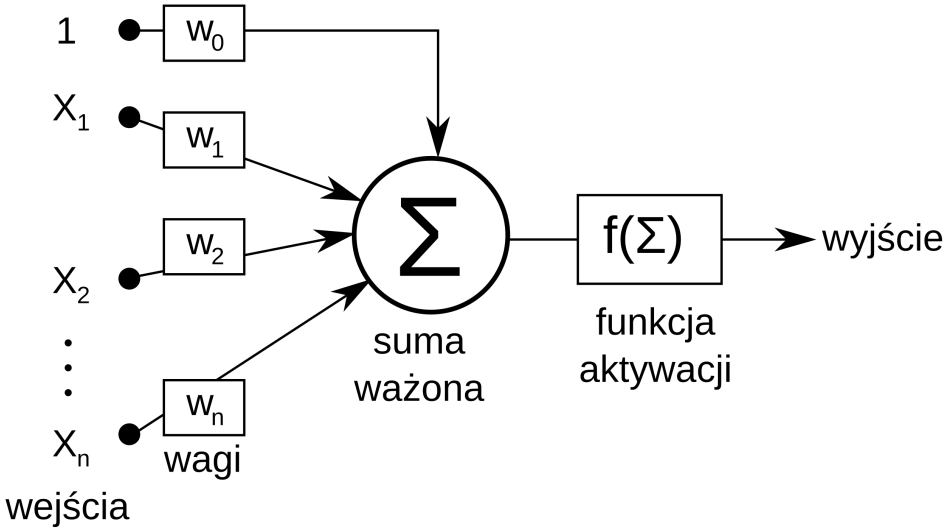
Rysunek 4 przedstawia wpływ różnych wartości biasu b na kształt funkcji sigmoidalnej w neuronie z jednym wejściem i wagą $w = 0.1$.



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 pojedyczny 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 łączenie, konwolucje czy rekurencje [**?**]. W przypadku pełnego łączenia 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:**
$$\mathbf{a}^{(0)} = \mathbf{x} \tag{9}$$

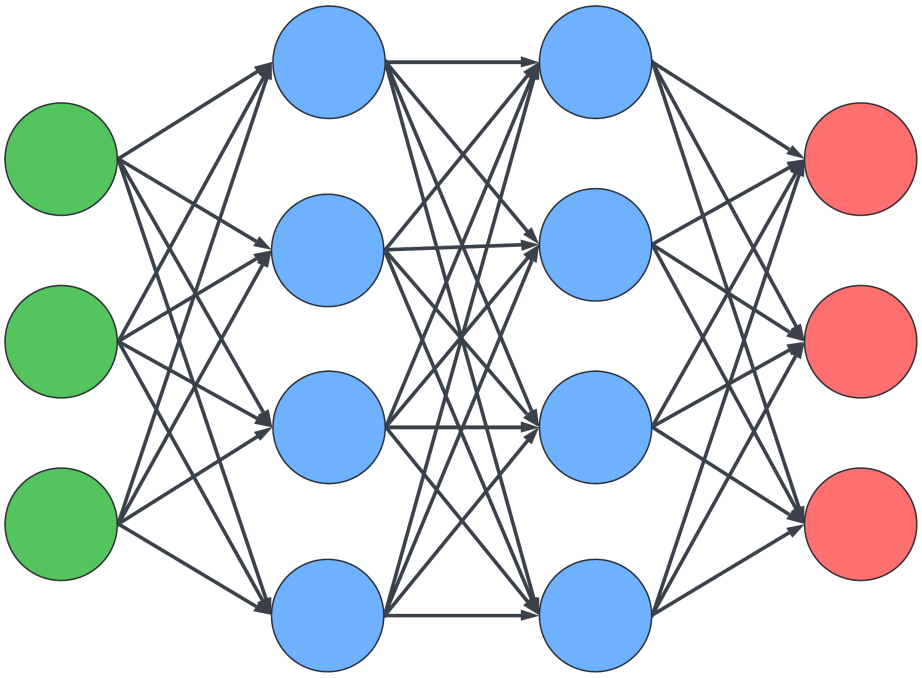
2. **Warstwy ukryte:**
$$\mathbf{a}^{(l)} = f(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}) \quad \text{dla } l = 1, 2, \ldots, L-1 \tag{10}$$

3. **Warstwa wyjściowa:**
$$\hat{\mathbf{y}} = g(\mathbf{W}^{(L)}\mathbf{a}^{(L-1)} + \mathbf{b}^{(L)}) \tag{11}$$

   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 [**?**]:
$$L = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{12}$$
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 [**?**]:
$$\frac{\partial L}{\partial \mathbf{W}^{(l)}}, \quad \frac{\partial L}{\partial \mathbf{b}^{(l)}} \tag{13}$$
5. **Aktualizacja parametrów:** Wagi i obciążenia są aktualizowane za pomocą algorytmu optymalizacji (np. Stochastic Gradient Descent, Adam), wykorzystując obliczone gradienty [**?**]:
$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta\frac{\partial L}{\partial \mathbf{W}^{(l)}}, \quad \mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta\frac{\partial L}{\partial \mathbf{b}^{(l)}} \tag{14}$$
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 architekura 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 $\mathbf{h}_t$ [**?**]. Stan ukryty $\mathbf{h}_t$ 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 $\mathbf{h}_t$ w czasie t jest wektorem obliczanym na podstawie bieżącego wejścia $\mathbf{x}_t$ oraz poprzedniego stanu ukrytego $\mathbf{h}_{t-1}$ [**?**]:

$$\mathbf{h}_t = f(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}) \tag{15}$$

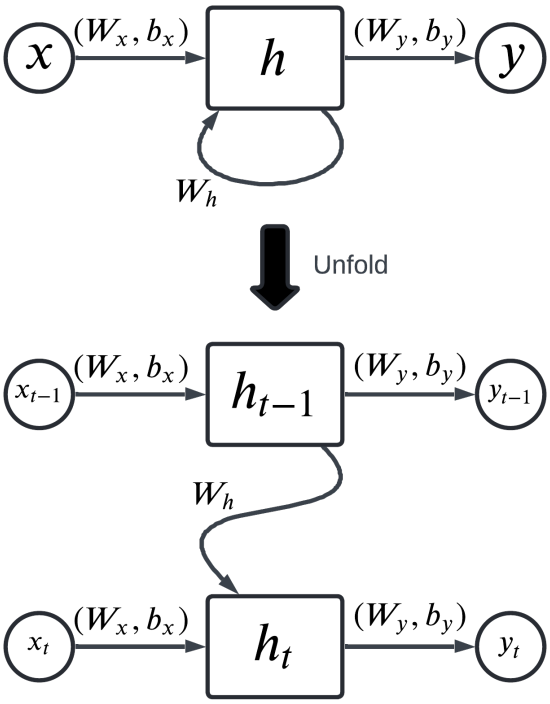
gdzie $\mathbf{W}_h$ to macierz wag dla stanu ukrytego, $\mathbf{W}_x$ to macierz wag dla wejścia, $\mathbf{b}$ to wektor obciążeń, a f to funkcja aktywacji.

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

$$\hat{\mathbf{y}}_t = g(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y) \tag{16}$$

gdzie $\mathbf{W}_y$ to macierz wag dla wyjścia, $\mathbf{b}_y$ 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 $\mathbf{h}_t$ 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 błędu.

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 [**?**]:

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

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

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

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 błędu (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 $\mathbf{W}_h$ 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

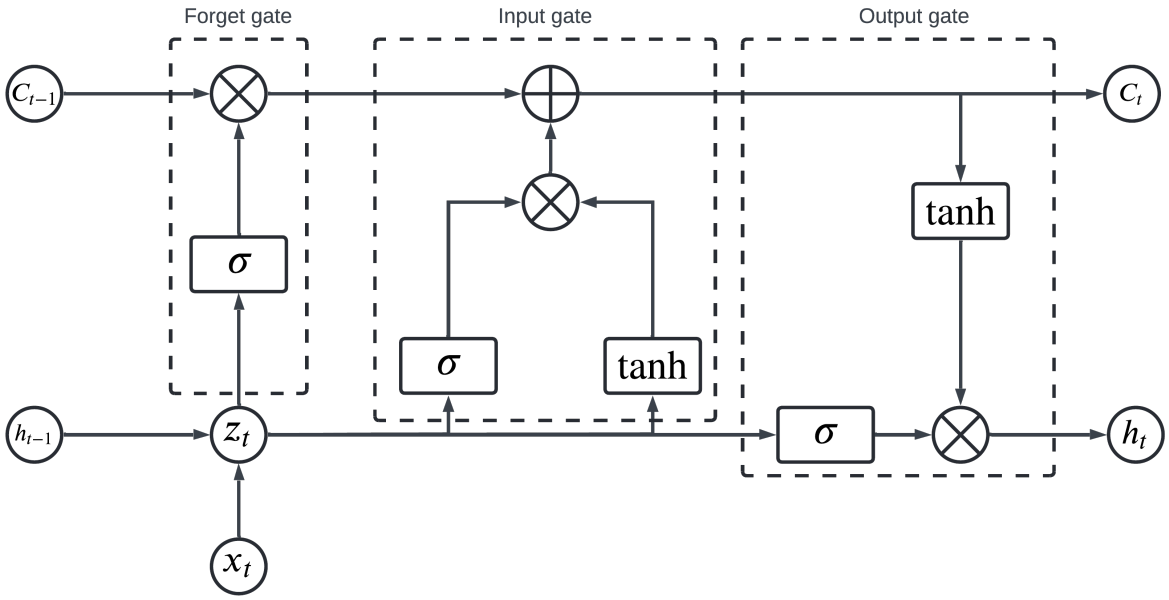
Sieci Long Short-Term Memory (LSTM) zostały zaprojektowane, aby niwelować problemy zanikającego i eksplodującego gradientu występujące w tradycyjnych sieciach RNN [**?**]. Cechę tę osiągnięto poprzez wprowadzenie mechanizmów zwanych bramkami (ang. *gates*) oraz dodatkowego stanu komórki $\mathbf{C}_t$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

1. **Bramka zapominania (Forget Gate):** Decyduje, które informacje z poprzedniego stanu komórki $\mathbf{C}_{t-1}$ należy zachować, a które odrzucić.

2. **Bramka wejścia (Input Gate):** Określa, które nowe informacje z bieżącego wejścia $\mathbf{x}_t$ powinny zostać dodane do stanu komórki $\mathbf{C}_t$.

3. **Bramka wyjścia (Output Gate):** Kontroluje, które informacje ze stanu komórki $\mathbf{C}_t$ zostaną wykorzystane do wygenerowania bieżącego stanu ukrytego $\mathbf{h}_t$.

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 $\mathbf{f}_t$ na podstawie bieżącego wejścia $\mathbf{x}_t$ oraz poprzedniego stanu ukrytego $\mathbf{h}_{t-1}$. Otrzymana wartość decyduje, które informacje z poprzedniego stanu komórki $\mathbf{C}_{t-1}$ należy zachować [**?**]:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{19}$$

$$\mathbf{C}_t = \mathbf{f}_t * \mathbf{C}_{t-1} \tag{20}$$

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

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \tag{21}$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \tag{22}$$

$$\mathbf{C}_t = \mathbf{C}_t + \mathbf{i}_t * \tilde{\mathbf{C}}_t \tag{23}$$

3. **Bramka wyjścia:** Oblicza się wartość bramki wyjścia $\mathbf{o}_t$ na podstawie bieżącego wejścia $\mathbf{x}_t$ oraz poprzedniego stanu ukrytego $\mathbf{h}_{t-1}$. Następnie, generuje się bieżący stan ukryty $\mathbf{h}_t$ na podstawie zaktualizowanego stanu komórki $\mathbf{C}_t$ [**?**]:

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \tag{24}$$

$$\mathbf{h}_t = \mathbf{o}_t * \tanh(\mathbf{C}_t) \tag{25}$$

gdzie:

- σ to funkcja aktywacji sigmoidalnej,
- $\tanh$ to funkcja aktywacji tangens hiperboliczny,
- $\mathbf{W}_f, \mathbf{W}_i, \mathbf{W}_C, \mathbf{W}_o$ to macierze wag dla poszczególnych bramek,
- $\mathbf{b}_f, \mathbf{b}_i, \mathbf{b}_C, \mathbf{b}_o$ 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 $\mathbf{C}_t$ [**?**]. 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 [**?**].