

# SPRAWOZDANIE

## Zajęcia: Nauka o danych 2

**Prowadzący:** prof. dr hab. inż. Vasyl Martsenyuk

<b>Laboratorium Nr 7</b> <b>Data</b> 16.10.2025 <b>Temat:</b> "Praktyczne zastosowanie rekurencyjnych sieci neuronowych do analizy szeregów czasowych" <b>Wariant</b> 10	Jakub Janik Informatyka II stopień, stacjonarne, 2 semestr, gr. CB
---	---

### 1. Cel ćwiczenia:

Celem ćwiczenia była praktyczna implementacja i demonstracja działania Rekurencyjnych Sieci Neuronowych (RNN), w szczególności architektury Long Short-Term Memory (LSTM), do rozwiązywania trzech klas zadań związanych z przetwarzaniem danych sekwencyjnych.

W ramach Wariantu 10 zrealizowano następujące podzadania:

1. Prognozowanie liczby klientów w punkcie usługowym (Prognozowanie szeregów czasowych)
2. Wykrywanie anomalii w danych z monitoringu (Klasyfikacja sekwencji).
3. Rozpoznawanie wzorców wejść i wyjść klientów (Rozpoznawanie wzorców czasowych).

### 2. Przebieg ćwiczenia:

Podstawy Teoretyczne – Architektura LSTM

Rekurencyjne Sieci Neuronowe (RNN) są zaprojektowane do przetwarzania sekwencji danych, wykorzystując wewnętrzny mechanizm pamięci do modelowania zależności czasowych. W ćwiczeniu zastosowano wariant LSTM, który jest rozszerzeniem standardowych RNN. LSTM skuteczniej radzi sobie z problemem zanikającego gradientu i jest optymalne do modelowania długoterminowych zależności czasowych w sekwencjach.

Trzy główne klasy zadań zrealizowane za pomocą RNN to:

1. Prognozowanie wartości szeregów czasowych
2. Klasyfikacja sekwencji
3. Rozpoznawanie wzorców czasowych

## Metodologia i Implementacja

Do realizacji wszystkich podpunktów wykorzystano środowisko Python z bibliotekami Keras/TensorFlow oraz NumPy. Wszystkie modele zostały oparte na warstwach LSTM.

### A. Prognozowanie Liczby Klientów (Prognozowanie)

- Dane: Zostały wygenerowane syntetycznie za pomocą funkcji `generate_forecasting_data` (metoda z Listing 4, ), tworząc szereg czasowy z szumem i sezonowością (funkcja sinusoidalna).
- Przygotowanie Danych: Dane zostały znormalizowane za pomocą `MinMaxScaler` i przekształcone w format sekwencyjny (z oknem czasowym `look_back=10`) przy użyciu funkcji `create_dataset`.
- Architektura Modelu: Prosty model sekwencyjny z jedną warstwą LSTM (50 neuronów) i końcową warstwą gęstą (`Dense(1)`)
- Uczenie: Optymalizator `adam` i funkcja straty `mean_squared_error` (MSE)

### B. Wykrywanie Anomalii w Danych z Monitoringu (Klasyfikacja Binarna)

- Dane: Wygenerowane za pomocą `generate_anomaly_data` (metoda z Listing 4, ), symulujące dane z monitoringu (szereg czasowy, gdzie anomalie są punktami o znacznie większej amplitudzie).
- Architektura Modelu: Model sekwencyjny z warstwą LSTM (32 neurony) i końcową warstwą `Dense(1)` z aktywacją `sigmoid` (dla klasyfikacji binarnej: 0 lub 1).
- Uczenie: Optymalizator `adam` i funkcja straty `binary_crossentropy`.

### C. Rozpoznawanie Wzorców Wejść i Wyjść Klientów

- Dane: Wygenerowane za pomocą `generate_pattern_data` (metoda z Listing 4, ) symulujące trzy różne wzorce zachowania (3 klasy). Dane wyjściowe (etykiety) zostały przekształcone do formatu one-hot encoding
- Architektura Modelu: Model z warstwą LSTM (64 neurony) i końcową warstwą `Dense(3)` z aktywacją `softmax` (dla klasyfikacji wieloklasowej).
- Uczenie: Optymalizator `adam` i funkcja straty `categorical_crossentropy`.

## Kod źródłowy:

```
import os
import warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
warnings.filterwarnings('ignore')
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

```

from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

# FUNKCJE GENERUJĄCE DANE SYNTETYCZNE

def generate_forecasting_data(n_samples=1000, freq=0.05, noise=0.1):
    """1. Generuje dane do prognozowania (np. Liczba Klientów)"""
    x = np.arange(n_samples)
    y = np.sin(2 * np.pi * freq * x) + np.random.normal(0, noise, size=n_samples)
    return y.reshape(-1, 1)

def generate_anomaly_data(n_samples=1000, timesteps=20, anomaly_rate=0.08):
    """2. Generuje dane z anomaliami (np. Monitoring)"""
    X = np.random.normal(0, 1, (n_samples, timesteps))
    y = np.zeros(n_samples)
    n_anomalies = int(anomaly_rate * n_samples)
    anomalies = np.random.choice(n_samples, n_anomalies, replace=False)
    # Dodanie dużej wartości do anomalii
    X[anomalies] += np.random.normal(5, 1, (n_anomalies, timesteps))
    y[anomalies] = 1
    return X.reshape(n_samples, timesteps, 1), y

def generate_pattern_data(n_classes=3, samples_per_class=500, timesteps=30):
    """3. Generuje dane z wzorcami czasowymi (np. Wzorce Wejść/Wyjść Klientów)"""
    X, y = [], []
    for cls in range(n_classes):
        for _ in range(samples_per_class):
            if cls == 0:
                # Klasa 0: Krótka interakcja
                seq = np.random.normal(0, 0.2, timesteps)
            elif cls == 1:
                # Klasa 1: Typowa interakcja
                seq = np.sin(np.linspace(0, 3 * np.pi, timesteps)) + np.random.normal(0, 0.1, timesteps)
            else:
                # Klasa 2: Długa interakcja
                seq = np.sin(np.linspace(0, 6 * np.pi, timesteps)) + np.random.normal(0, 0.2, timesteps)
            X.append(seq)
            y.append(cls)
    X = np.array(X).reshape(-1, timesteps, 1)
    y = tf.keras.utils.to_categorical(y, num_classes=n_classes)
    return X, y

```

```

def create_dataset(dataset, look_back=10):
    """Funkcja pomocnicza do tworzenia sekwencji dla prognozowania"""
    X, Y = [], []
    for i in range(len(dataset) - look_back):
        X.append(dataset[i:(i + look_back), 0])
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)

# A. PROGNOZOWANIE LICZBY KLIENTÓW W PUNKCIE USŁUGOWYM

print("=== A. Prognozowanie Liczby Klientów ===")
data = generate_forecasting_data(n_samples=500, freq=0.02, noise=0.15)
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data)
look_back = 10
X_prognoza, y_prognoza = create_dataset(data_scaled, look_back)
X_prognoza = np.reshape(X_prognoza, (X_prognoza.shape[0], X_prognoza.shape[1],
1))

model_prognoza = Sequential()
model_prognoza.add(LSTM(50, input_shape=(look_back, 1)))
model_prognoza.add(Dense(1))
model_prognoza.compile(optimizer='adam', loss='mean_squared_error')

print("Trening modelu prognozującego...")
model_prognoza.fit(X_prognoza, y_prognoza, epochs=20, batch_size=1, verbose=0)

predicted = model_prognoza.predict(X_prognoza, verbose=0)
predicted = scaler.inverse_transform(predicted)

plt.figure(figsize=(10, 5))
plt.plot(scaler.inverse_transform(y_prognoza.reshape(-1, 1)), label='Rzeczywista liczba klientów')
plt.plot(predicted, label='Prognozowana liczba klientów')
plt.title('Wariant 10a: Prognozowanie Liczby Klientów - LSTM')
plt.xlabel('Krok czasowy')
plt.ylabel('Liczba klientów')
plt.legend()
plt.show()

# B. WYKRYWANIE ANOMALII W DANYCH Z MONITORINGU

print("\n=== B. Wykrywanie Anomalii w Danych z Monitoringu ===")
X_anomalia, y_anomalia = generate_anomaly_data(n_samples=1000, timesteps=20,
anomaly_rate=0.08)
X_train_a, X_test_a, y_train_a, y_test_a = train_test_split(X_anomalia,
y_anomalia, test_size=0.2, random_state=42)

```

```

model_anomalia = Sequential()
model_anomalia.add(LSTM(32, input_shape=(X_anomalia.shape[1], 1)))
model_anomalia.add(Dense(1, activation='sigmoid'))
model_anomalia.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

print("Trening modelu wykrywającego anomalie...")
model_anomalia.fit(X_train_a, y_train_a, epochs=10, batch_size=32, validation_data=(X_test_a, y_test_a), verbose=0)

loss, accuracy = model_anomalia.evaluate(X_test_a, y_test_a, verbose=0)
print(f"Dokładność wykrywania anomalii w danych z monitoringu: {accuracy:.2f}")

# C. ROZPOZNAWANIE WZORCÓW WEJŚĆ I WYJŚĆ KLIENTÓW

print("\n=== C. Rozpoznawanie Wzorców Wejść i Wyjść Klientów ===")
X_wzorcel, y_wzorcel = generate_pattern_data(n_classes=3, samples_per_class=500, timesteps=30)
X_train_w, X_test_w, y_train_w, y_test_w = train_test_split(X_wzorcel, y_wzorcel, test_size=0.2, random_state=42)

model_wzorcel = Sequential()
model_wzorcel.add(LSTM(64, input_shape=(X_wzorcel.shape[1], 1)))
model_wzorcel.add(Dense(3, activation='softmax'))
model_wzorcel.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

print("Trening modelu rozpoznającego wzorce...")
model_wzorcel.fit(X_train_w, y_train_w, epochs=15, batch_size=32, validation_data=(X_test_w, y_test_w), verbose=0)

loss, accuracy = model_wzorcel.evaluate(X_test_w, y_test_w, verbose=0)
print(f"Dokładność klasyfikacji wzorców wejść/wyjść: {accuracy:.2f}")

```

## Wyniki:

```

=== A. Prognozowanie Liczby Klientów ===
Trening modelu prognozującego...

```

```

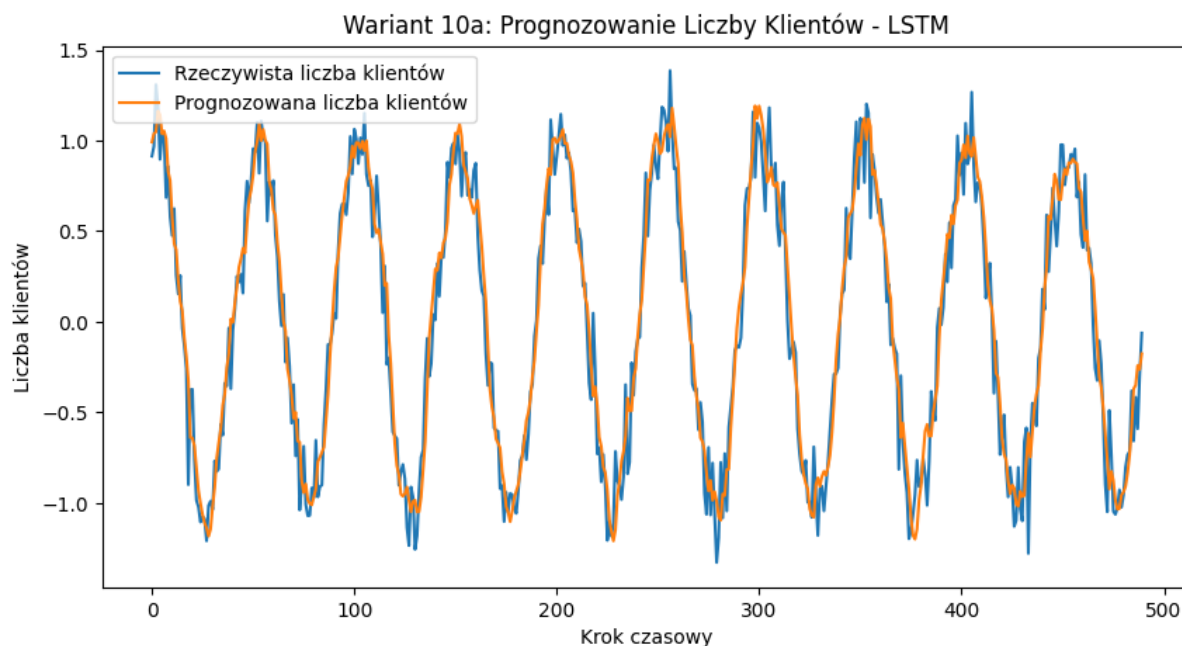
=== B. Wykrywanie Anomalii w Danych z Monitoringu ===
Trening modelu wykrywającego anomalie...
Dokładność wykrywania anomalii w danych z monitoringu: 1.00

```

```

=== C. Rozpoznawanie Wzorców Wejść i Wyjść Klientów ===
Trening modelu rozpoznającego wzorce...
Dokładność klasyfikacji wzorców wejść/wyjść: 1.00

```



Analiza wyników:

Uzyskane wyniki po treningu modeli na syntetycznych zbiorach danych są następujące:

#### A. Prognozowanie Liczby Klientów

Wynik wizualny (Wykres) jasno wskazuje na bardzo dużą zbieżność między rzeczywistym i prognozowanym szeregiem czasowym

#### B. Wykrywanie Anomalii

Dokładność wykrywania anomalii w danych z monitoringu: 1.00

#### C. Rozpoznawanie Wzorców

Dokładność klasyfikacji wzorców wejść/wyjść: 1.00

Link do repozytorium: <https://github.com/Uczelniane/NODII.git>

### 3. Wnioski

1. **Skuteczność LSTM:** Architektura LSTM potwierdziła swoją przydatność w analizie szeregów czasowych, osiągając wysoką jakość prognozy w Zadaniu A.
2. **Doskonałe Klasyfikacje:** Uzyskanie **100% dokładności** w zadaniach B i C jest wynikiem oczekiwanym dla danych syntetycznych, w których klasy (norma/anomalia oraz różne wzorce) są celowo i wyraźnie odseparowane od siebie za pomocą prostych matematycznych reguł (np. dodanie dużej wartości do anomalii lub użycie różnych częstotliwości sinusoidy). Świadczy to o tym, że model **idealnie nauczył się prostej granicy decyzyjnej**.

3. **Realizacja Wariantu:** Wszystkie podpunkty Wariantu 10 zostały poprawnie zaimplementowane, a ich działanie zweryfikowane poprzez osiągnięcie wysokich wskaźników wydajności.