AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Wykorzystanie głębokiego uczenia w prognozowaniu zapotrzebowania i produkcji energii elektrycznej z odnawialnych źródeł energii

mgr inż. Michał Pikus
Katedra Informatyki Stosowanej

Kraków 06.12.2023

Agenda

- **Omówienie współpracy w kontekście projektów badawczych**
- **Wprowadzenie do problemu wykorzystania głębokiego uczenia w prognozowaniu zapotrzebowania i produkcji energii elektrycznej z odnawialnych źródeł energii**
- **Aspekty techniczne realizacji projektu**
- **Omówienie wyników**
- **Wykaz publikacji**

Zrealizowane projekty

- **Projekt B+R nr: POIR.01.01.01-00-1924/20 realizowany wspólnie z firmą Aigorithmics**

Inteligentna platforma sprzętowa o funkcjonalności pomiarowo-diagnostyczno-sterującej, umożliwiająca efektywne bilansowanie zapotrzebowania z potencjałem wytwórczym w zakresie energii elektrycznej w ujęciu lokalnym.

- **Projekt B+R nr: POIR.01.01.01-00-0506/21 realizowany wspólnie z firmą Aigorithmics**

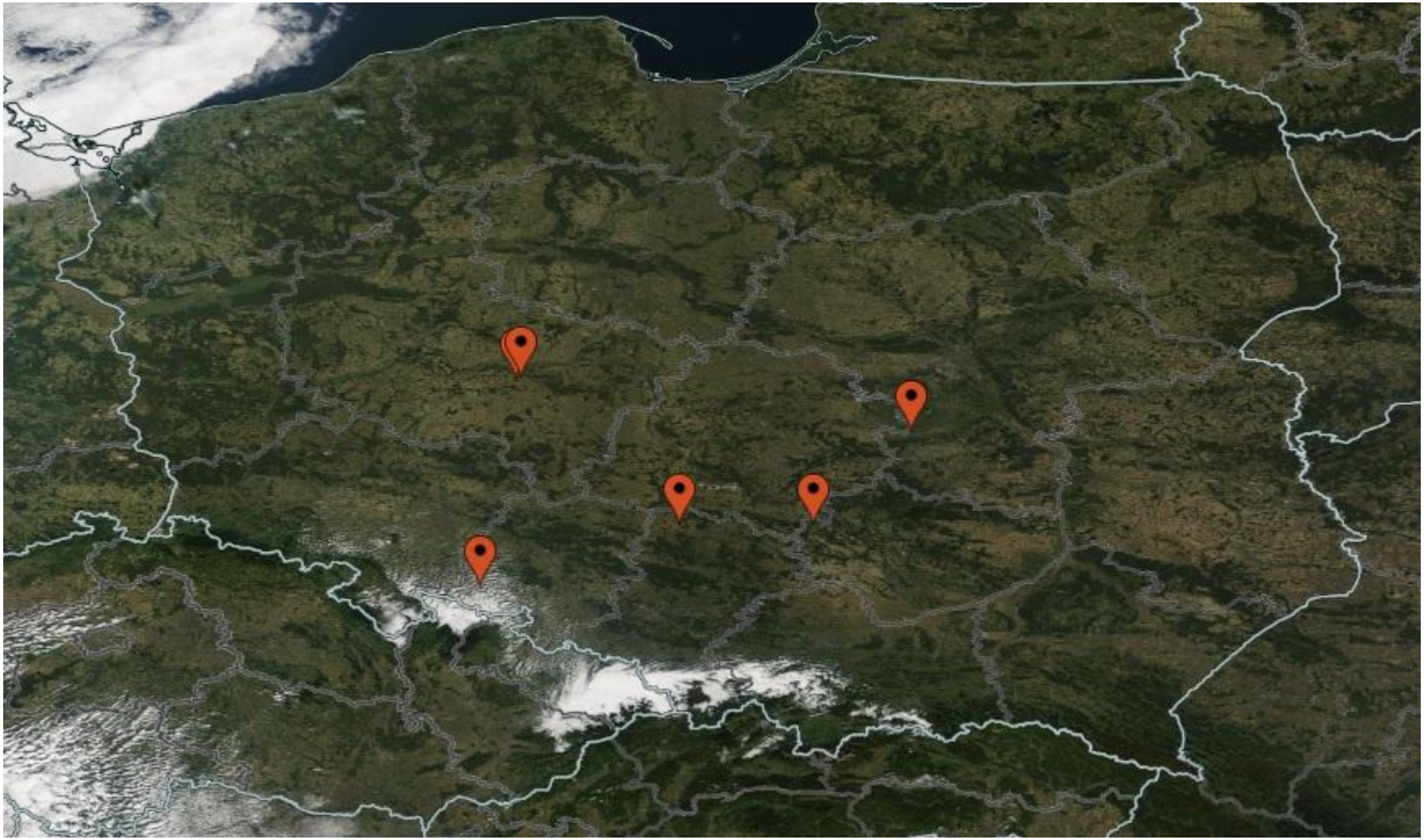
Opracowanie wysokowydajnej platformy cyfrowej samo uczących się mechanizmów inteligentnej analizy i predykcji stanu do efektywnej współpracy uczestników zdecentralizowanego rynku wytwarzania energii odnawialnej (OZE)

Analiza eksploracyjna EDA (PV)

Dane dotyczące produkcji energii z paneli fotowoltaicznych.

- **Dane pochodziły z sześciu gospodarstw.**
- **Pobierano je w odstępach 15-minutowych.**
- **W zależności od wariantu do danych z każdej farmy dodawano dane pogodowe oraz dane ze zdarzeń (SCADA) (scalone pomiary z falowników).**
- **Podano również dane prognozy produkcji energii na podstawie prognozy pogody oraz dane z liczników energii dostarczonej do sieci.**

Lokalizacja farm fotowoltaicznych



Parametry wykorzystane w uczeniu sieci DNN dla farm fotowoltaicznych

- całkowita wartość z czujnika nasłonecznienia
- wartość dzienna z czujnika nasłonecznienia
- temperatura modułu fotowoltaicznego (średnia)
- temperatura zewnętrzna
- pogoda (promieniowanie słoneczne, zachmurzenie)
- napięcie wejściowe DC temperatura wewnątrz falownika (wartość średnia)
- napięcie wejściowe AC z sieci energetycznej
- współczynnik mocy (power factor)
- moc czynna
- moc bierna
- wartość wyprodukowanej energii przez cały okres użytkowania
- wydajność falownika
- moc wytwarzana przez falownik (średnia)
- dzienna wartość wyprodukowanej energii

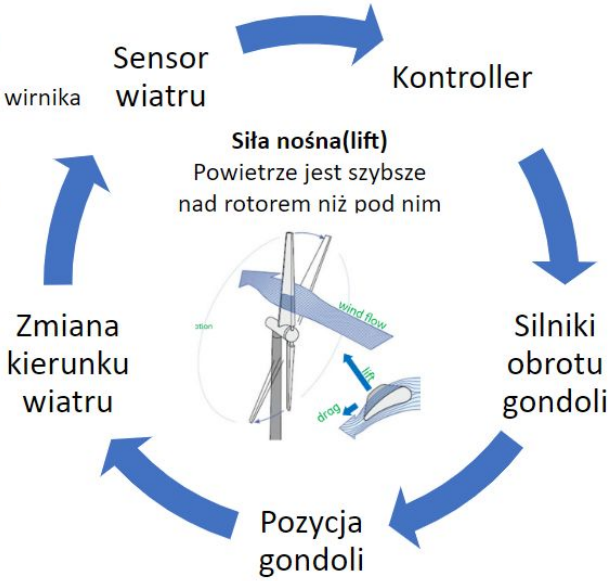
Analiza eksploracyjna EDA (WIND)

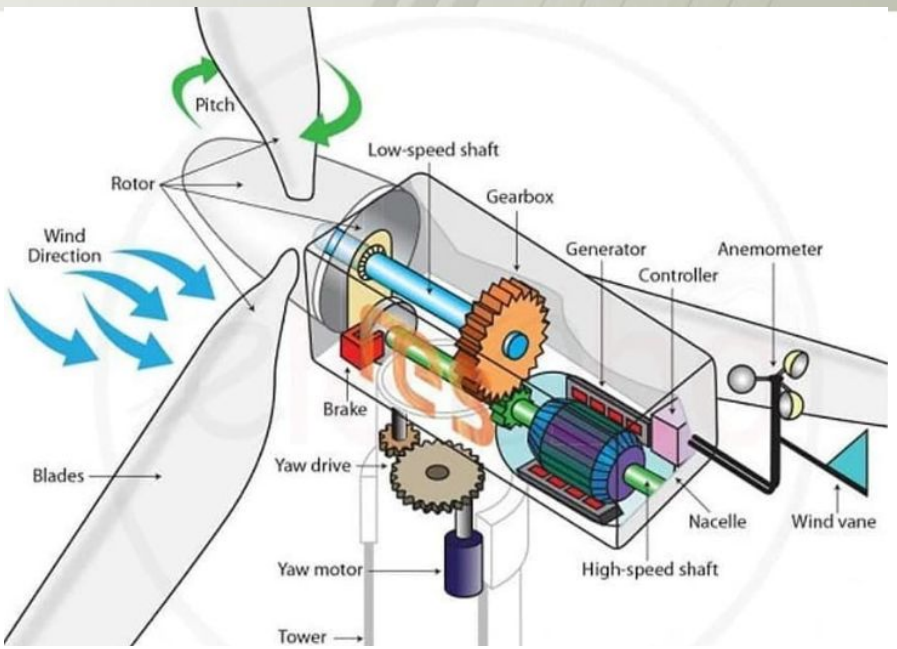
Dane dotyczące produkcji energii z farm wiatrowych.

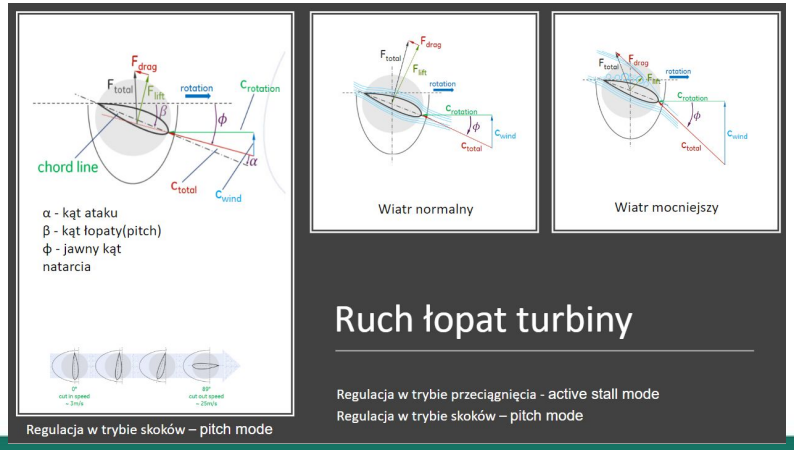
- **Dane pochodziły z 4 gospodarstw.**
- **Pobierano je w odstępach 10 minutowych uwzględnione zostały również dane o prognozach pogody**
- **Rozdzielczość danych pogodowych wynosi 1h**
- **Ze względu na specyfikę (różna liczba generatorów) stworzone zostały modele dla każdej z farm wiatrowych**

Zasada działania turbiny wiatrowej

- Siła nośna łopat rotora
- Wiatr od strony przodu wirnika
- Kierunek obrotu
zgodnie ze wskazówkami zegaru

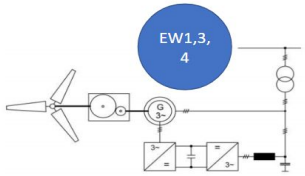






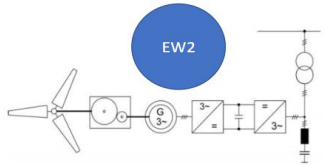
Generatory

Asynchroniczny, podwójnie zasilany(stator i rotor), czteropolowy - DFIG, chłodzony



- Stator przyłączony bezpośrednio do sieci
- Prędkość rotora jest zmienna
- Konwerter zasila rotor w AC
- Modulacja magnetycznego pola f2 by odpowiadała sieci f1

Synchroniczny generator z wirnikem ze stałymi magnesami - PMG, chłodzony



- Stator przyłączony do konwertera
- Prędkość rotora jest zmienna
- Konwerter zasila sieć w AC
- Modulacja energii z statora by odpowiadała częstotliwości sieci f1

Parametry wykorzystane w uczeniu sieci DNN dla farm wiatrowych

- temperatura łożyska generatora
- temperatura łożysk przekładni
- temperatura gondoli
- RPM wirnika
- prędkość wiatru
- kierunek wiatru
- temperatura otoczenia
- moc bierna
- moc czynna
- napięcie L1, L2, L3
- natężenie L1, L2, L3
- kierunek gondoli
- przyspieszenie wieży
- temperatura przekładni
- średni kąt nachylenia łopatek.

Architektury DNN

Sieci RNN:

- **Vanilla LSTM**
- **Wielowarstwowe LSTM**
- **Enkoder - Dekoder LSTM**
- **CNN**

Architektury hybrydowe (zaproponowane w pracy):

- **CNN + LSTM**
- **Conv + LSTM**

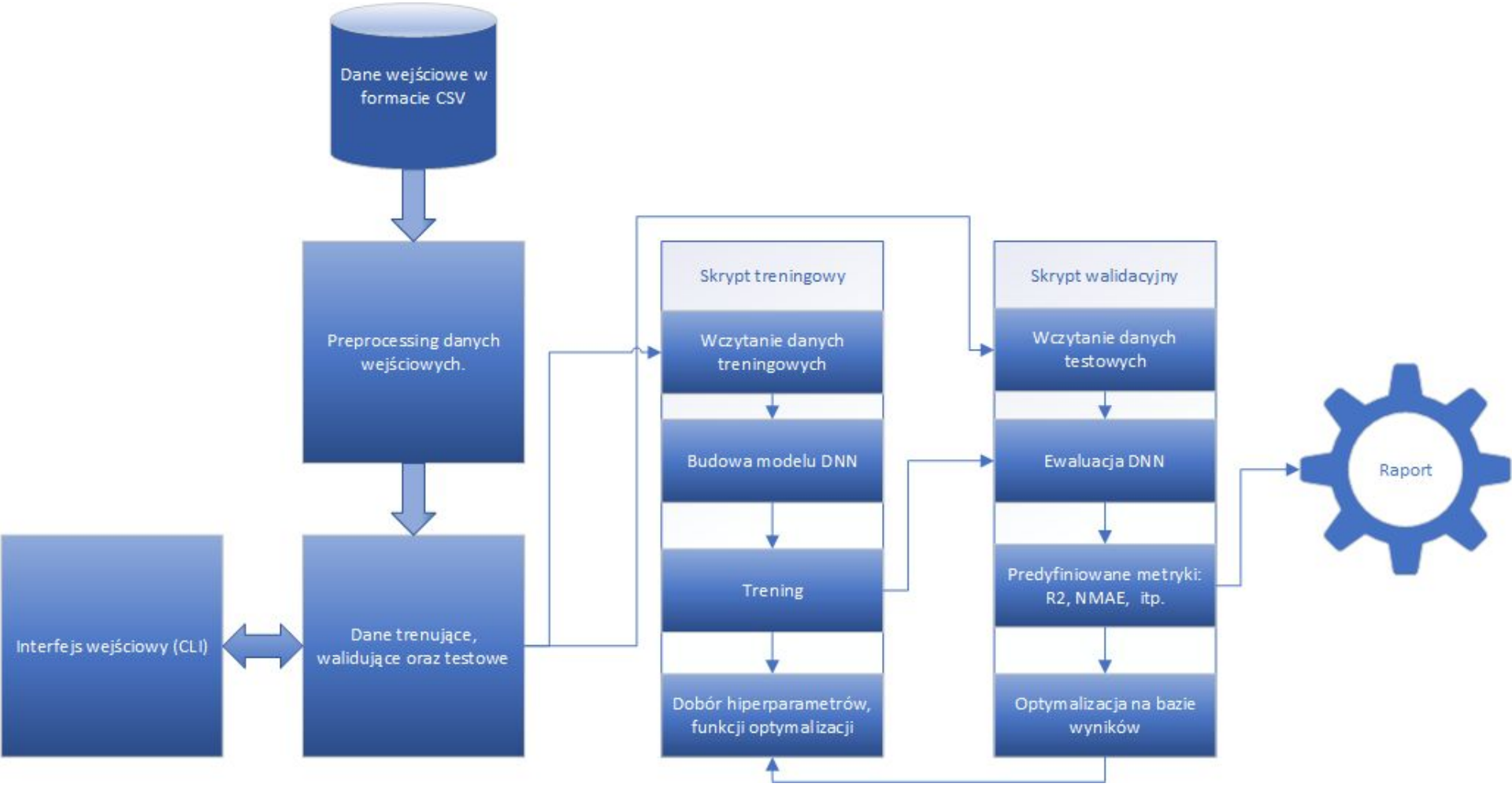
Architekura typu transformer:

- **TFT**

Automatyzacja, benchmark

- **Ręczny wybór najlepszego modelu jest również trudny, jeśli opiera się wyłącznie na metrykach i wynikach graficznych.**
- **Dobór odpowiednich hiperparametrów oraz zoptymalizowany model w trzech wymiarach (m.in. koszt obliczeń, waga i wydajność) został osiągnięty za pomocą naszych benchmarków oraz przy użyciu frameworka Nvidia TSPP.**
- **Główną zaletą korzystania ze zautomatyzowanego środowiska do wybierania wielu modeli jest łatwy sposób porównania każdego wyniku.**
- **Istnieje możliwość zmierzenia każdej metryki dla każdej metody na każdym etapie nauki.**
- **Benchmarki mogą pokazać zalety lub wady każdej architektury. Możemy też skupić się głównie na aspektach optymalizacyjnych.**
- **Czas wszystkich ewaluacji opisanych metod podejścia wynosił co najmniej półtora miesiąca.**

Diagram blokowy rozwiązania



Fragment implementacji (część treningowa)

```
from keras.models import Sequential, load_model
from keras.layers import Dense, RepeatVector, TimeDistributed, LSTM
from matplotlib import pyplot
import tensorflow as tf
from Common import *
import pandas as pd

def prepare_data_for_dnn(train, input_vector_length, n_out=1):
    data = train
    X, y = list(), list()
    input_begin = 0
    for _ in range(len(data))
        input_tail = input_begin + input_vector_length
        output_tail = input_tail + n_out
        if output_tail < len(data):
            x_input = data[input_begin:input_tail, :]
            X.append(x_input)
            y.append(data[input_tail:output_tail, 0])
        input_begin += 1
    return array(X), array(y)

def build_train_dnn(train, input_vector_length):
    train_x, train_y = prepare_data_for_dnn(train,
    input_vector_length)
    epochs, batch_size, verbose  =  100, 1024, 0
    number_of_strides, number_of_attributes, n_outputs = train_x.
    shape[1], train_x.shape[2], train_y.shape[1]
    dnn = Sequential()
    dnn.add(Conv1D(filters=64, kernel_size=2, activation='relu',
    input_shape=(number_of_strides,number_of_attributes)))
    dnn.add(Conv1D(filters=64, kernel_size=2, activation='relu'))
    dnn.add(MaxPooling1D(pool_size=2))
    dnn.add(Flatten())
    dnn.add(RepeatVector(n_outputs))
    dnn.add(LSTM(200, activation='relu', return_sequences=True))
    dnn.add(TimeDistributed(Dense(100, activation='relu')))
    dnn.add(TimeDistributed(Dense(1)))
    dnn.compile(loss='mse', optimizer='adam')
    train_x = tf.convert_to_tensor(train_x, dtype=tf.float32)
    train_y = tf.convert_to_tensor(train_y, dtype=tf.float32)
    dnn.fit(train_x, train_y, epochs=epochs, batch_size=batch_size,
     verbose=verbose, shuffle=False, use_multiprocessing=True)
    dnn.save('cnnlstm.h5')
    return dnn

```

Fragment implementacji (część walidacyjna)

```python
from datetime import datetime
from math import sqrt
import numpy as np
from numpy import split
from numpy import array
from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_pinball_loss, \
    mean_squared_log_error, median_absolute_error, explained_variance_score, r2_score
from sklearn.metrics import  mean_absolute_percentage_error as skmape
from sklearn.model_selection import train_test_split
from matplotlib import pyplot
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler

def evaluate_model(train, test, input_vector_length):
    model = build_train_dnn(train, input_vector_length)
    history = [x for x in train]
    predictions = list()
    for i in range(len(test)):
        yhat_sequence = forecast(model, history, input_vector_length)
        predictions.append(yhat_sequence)
        history.append(test[i])
    predictions = array(predictions)
    score, scores = evaluate_dnn(test[:], predictions)

    return score, scores

def evaluate_dnn(actual, predicted):
    scores = list()
    actual = abs(actual)
    predicted = abs(predicted)
    maxi = max(actual[:, 0])
    mae = mean_absolute_error(actual[:,0], predicted[:,0])
    mae = mean_absolute_error(actual[:, 0], predicted[:, 0]) / maxi
    mpl = mean_pinball_loss(actual[:,0], predicted[:,0])
    mape = mean_absolute_percentage_error(actual[:,0], predicted[:,0])
    wmape = weighted_mape_tf(actual[:, 0], predicted[:, 0])
    mse = mean_squared_error(actual[:,0], predicted[:,0])
    meae = median_absolute_error(actual[:,0], predicted[:,0])
    evs = explained_variance_score(actual[:,0], predicted[:,0])
    r2 = r2_score(predicted[:,0], actual[:,0])
    rmse = sqrt(mse)

    scores.append(rmse)
    scores.append(mae)
    scores.append(mpl)
    scores.append(mape)
    scores.append(mse)
    scores.append(meae)
    scores.append(evs)
    scores.append(r2)
    print('REAL:', np.mean(actual), 'PREDICTED:', np.mean(predicted))
    np.random.seed(19680801)

    dt = 0.01
    t = np.arange(0, 2005, dt)

    fig, axs = pyplot.subplots(2, 1)
    axs[0].plot(actual[:,0])
    axs[0].set_xlabel('item')
    axs[0].set_ylabel('actual')
    axs[0].grid(True)

    axs[1].plot(predicted[:,0])
    axs[1].set_xlabel('item')
    axs[1].set_ylabel('predicted')
    axs[1].grid(True)

    now = datetime.now()
    date_time = now.strftime("%d%m%Y%H%M%S")
    fig.tight_layout()
    pyplot.savefig('MultiLSTMAVGAMOUNT'+date_time+'.png')

    predict_plot = predicted[:, 0]
    true_plot = actual[:, 0]

    pyplot.figure(figsize=(16, 6))
    pyplot.plot(true_plot, label='True', linewidth=1)
    pyplot.plot(predict_plot, label=' MultiLSTMAVGAMOUNT', color='y', linewidth=1)

    pyplot.ylabel('AVG amount')
    pyplot.xlabel('Test items')
    pyplot.legend()
    now = datetime.now()
    date_time = now.strftime("%d%m%Y%H%M%S")
    pyplot.savefig('MultiLSTMAVGAMOUNT' + date_time + '.png')
    print(predicted[:,0])
```

Zastosowane metryki R^2

- **Współczynnik determinacji, oznaczony jako R2 lub r2 i wymawiany jako „R do kwadratu”, jest proporcją zmienności zmiennej zależnej, która jest przewidywalna na podstawie zmiennych niezależnych. Nazywana jest ona często również miarą dopasowania modelu.**
- **Jest to metryka stosowana w kontekście modeli statystycznych, której głównym celem jest albo przewidywanie przyszłych wyników, albo testowanie hipotez na podstawie innych powiązanych informacji. Stanowi miarę, jak dobrze zaobserwowane wyniki są replikowane przez model, w oparciu o proporcję całkowitej zmienności wyników wyjaśnionej przez model**

$$R^2 := \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \geqslant 0,$$

gdzie:

y_i – i-ta obserwacja zmiennej y,

$\hat{y}_i$ – wartość teoretyczna zmiennej objaśnianej (na podstawie modelu),

$\bar{y}$ – średnia arytmetyczna empirycznych wartości zmiennej objaśnianej.

Zastosowane metryki MAPE

- **Średni bezwzględny błąd procentowy (MAPE), znany również jako średnie bezwzględne odchylenie procentowe (MAPD), jest miarą dokładności przewidywania metody prognozowania w statystyce. Zwykle wyraża dokładność jako stosunek określony wzorem:**

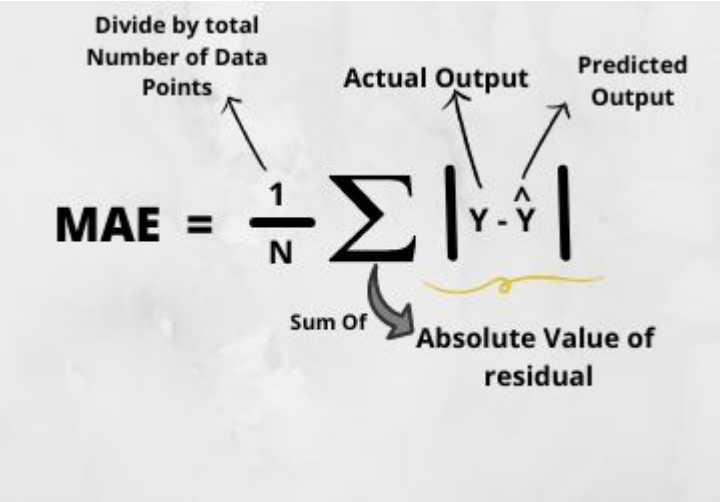
$$\mathrm{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

MAPE ma jednak pewne ograniczenia, które czynią go mniej odpowiednim w określonych scenariuszach.

- Dzielenie przez zero: MAPE nie nadaje się dobrze do sytuacji, w których rzeczywiste wartości są bliskie zera. Kiedy mianownik w obliczeniu procentowym jest mały lub zerowy, może to prowadzić do nieskończonych lub bardzo wysokich wartości, czyniąc MAPE bezsensownym. Problem ten pojawia się często podczas szacowania danych, które mogą zbliżać się do zera lub nawet stać się ujemne.
- Asymetryczny wpływ błędów: MAPE traktuje błędy przeszacowania i niedoszacowania symetrycznie. Jednakże w niektórych dziedzinach, takich jak prognozowanie energii, błędy zawyżenia i niedoszacowania mogą mieć znacząco różne konsekwencje. Ta nierównowaga wpływu błędów sprawia, że MAPE jest mniej odpowiedni do takich scenariuszy.

Zastosowane metryki MAE, NMAE

- **Średni błąd bezwzględny (MAE)**
- **MAE to bardzo prosta metryka, która oblicza bezwzględną różnicę między wartościami rzeczywistymi i przewidywanymi.**



Znormalizowany średni błąd bezwzględny (NMAE) to alternatywna metryka, która pozwala pokonać ograniczenia MAPE w sytuacjach obejmujących dane, które mogą być ujemne lub bliskie zeru. NMAE normalizuje błąd, dzieląc go przez zakres rzeczywistych wartości, zapewniając bardziej zrównoważoną miarę dokładności. Przyjrzyjmy się, dlaczego NMAE jest szczególnie przydatny w prognozowaniu zapotrzebowania na energię elektryczną, gdy wykorzystanie energii słonecznej na dachu może spowodować popyt bliski zeru lub ujemny.

Zastosowane metryki NMAE

Stosując NMAE zamiast MAPE, można skuteczniej ocenić dokładność prognozowania zapotrzebowania na obciążenie. NMAE normalizuje błąd bezwzględny według zakresu wartości rzeczywistych, uwzględniając sytuacje, w których popyt zbliża się do zera lub staje się ujemny. Ta normalizacja zapewnia lepsze odzwierciedlenie wydajności modelu prognostycznego, nawet w okresach niskiego lub ujemnego popytu.

$$NMAE_1 = \frac{MAE}{mean(actual\ values)}$$

Wyniki (metryki)

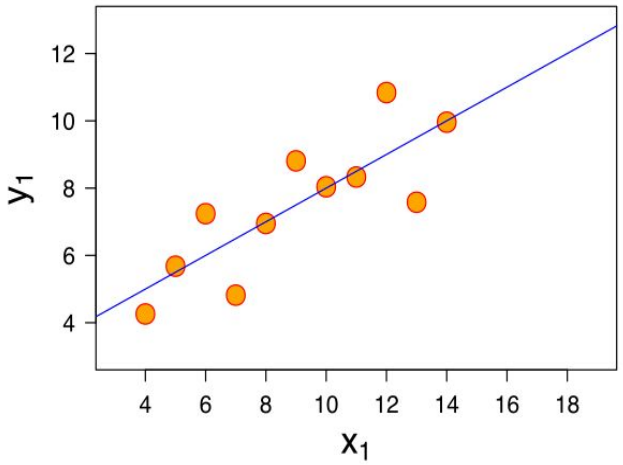
Farmy fotowoltaiczne

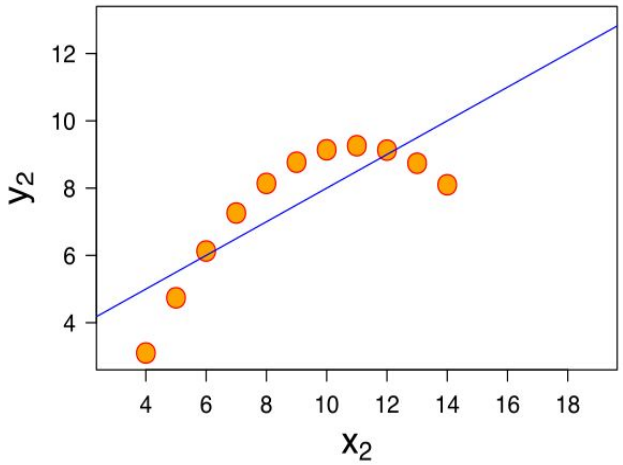
model	r2_train	r2_val	r2_test	nmae_train	nmae_val	nmae_test	mape_train	mape_val	mape_test
CNN	0.8926	0.8748	0.7332	0.0270	0.0323	0.0129	0.7874	0.6151	1.5140
GRU	0.9127	0.8966	0.6814	0.0253	0.0291	0.0115	0.4711	0.4384	0.7673
CNNEncoderDecoderLSTM	0.9254	0.9056	0.8359	0.0216	0.0270	0.0086	0.5621	0.4646	0.7854
ConvEncoderDecoderLSTM	0.8917	0.8946	0.4418	0.0288	0.0304	0.0214	0.8560	0.5642	2.2483
EncoderDecoderLSTM	0.9188	0.9023	0.8160	0.0221	0.0278	0.0100	0.5180	0.4633	0.9545
MultiStepLSTM	0.9342	0.9100	0.8661	0.0180	0.0246	0.0064	0.3277	0.3274	0.4546

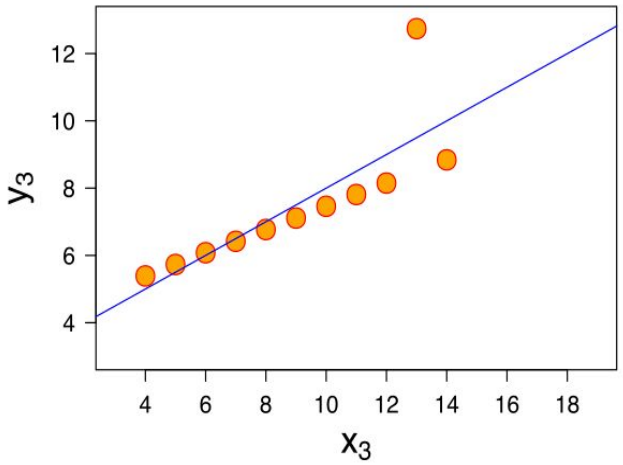
Farmy wiatrowe

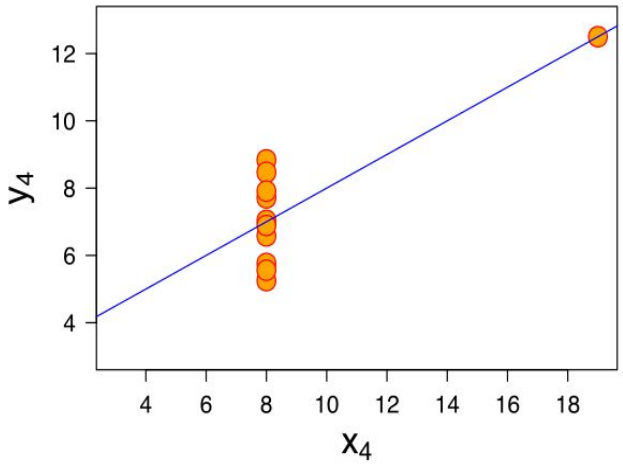
Model	tr_r2	val_r2	test_r2	tr_nmae	val_nmae	test_nmae	tr_mape	val_mape	test_mape
Conv	0.860555	0.831844	0.893344	0.133273	0.136442	0.12814	17.87283	21.40018	11.54999
CNN	0.961488273	0.939942183	0.972699748	0.039265582	0.043214156	0.036746415	9.947282032	2.064749699	4.845566254
GRU	0.875660296	0.92280927	0.837244492	0.064636894	0.048723355	0.088102071	5.903229809	6.282241227	3.163177828
CNNLSTM	0.972113654	0.957232844	0.976446107	0.033375186	0.035767942	0.037136579	4.729748767	8.496661982	1.938968401
ConvLSTM	0.973430461	0.959852958	0.983159757	0.037380265	0.041978844	0.031499257	1.508929922	1.304960646	2.030106146
EncDecLSTM	0.97079854	0.963857743	0.982859827	0.037771842	0.037116146	0.030474626	4.178814161	9.743417476	2.023657286
MultiLSTM	**0.974199023**	**0.966429507**	**0.983649928**	**0.034035485**	**0.033559874**	**0.028191401**	**7.257572021**	**1.105741192**	**5.420490873**

Miara determinacji a uzyskane wykresy





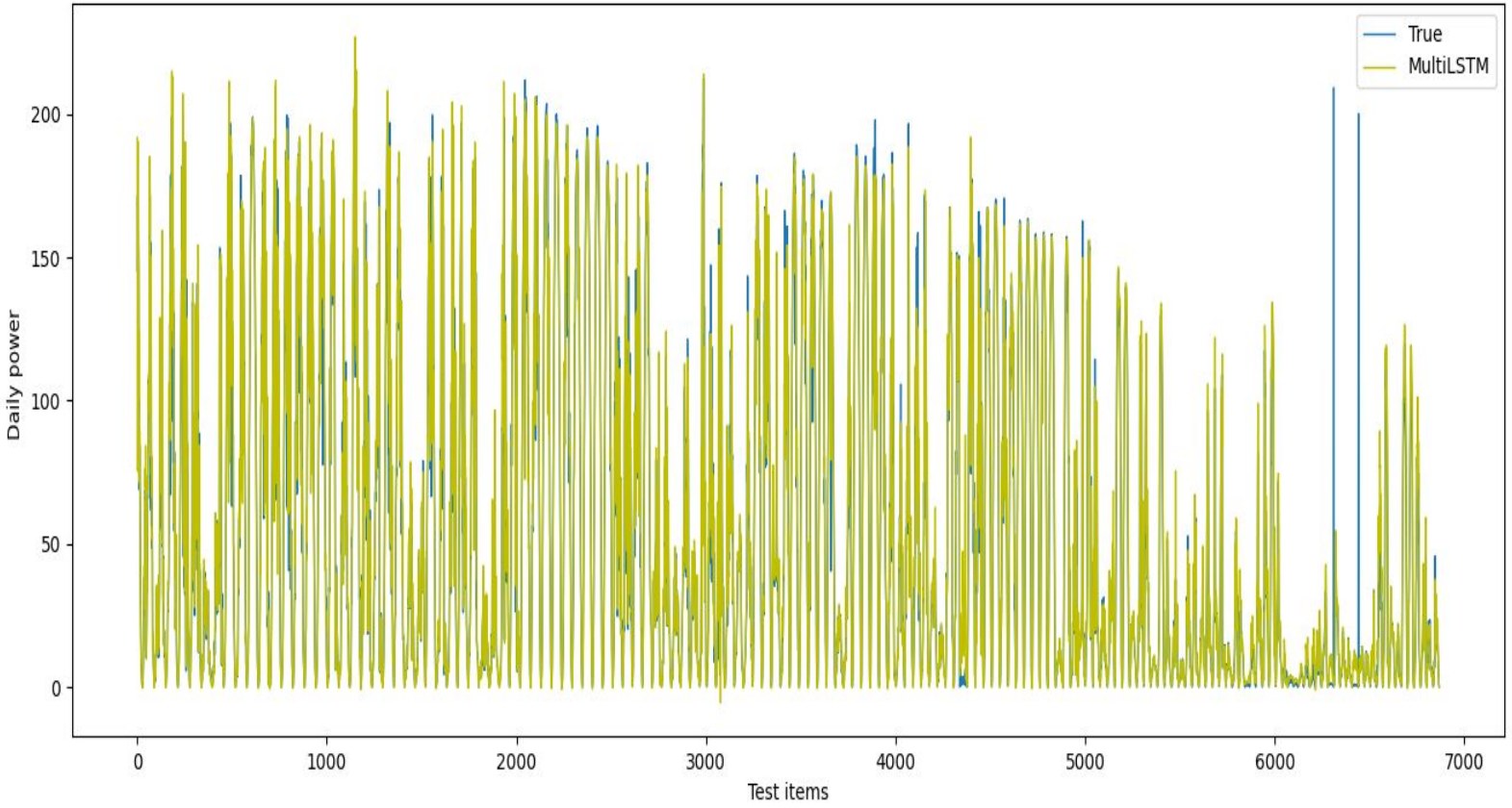




Wszystkie cztery zestawy danych wydają się być identyczne, jeżeli weźmiemy pod uwagę ich charakterystykę statystyczną, ale znacznie różnią się od siebie w ujęciu graficznym. Kwartet Anscombe'a Współczynnik determinacji R^2 0,666 (różnica pojawia się dopiero na czwartym miejscu po przecinku)

Wizualizacja działania sieci DNN w przypadku farmy fotowoltaicznej

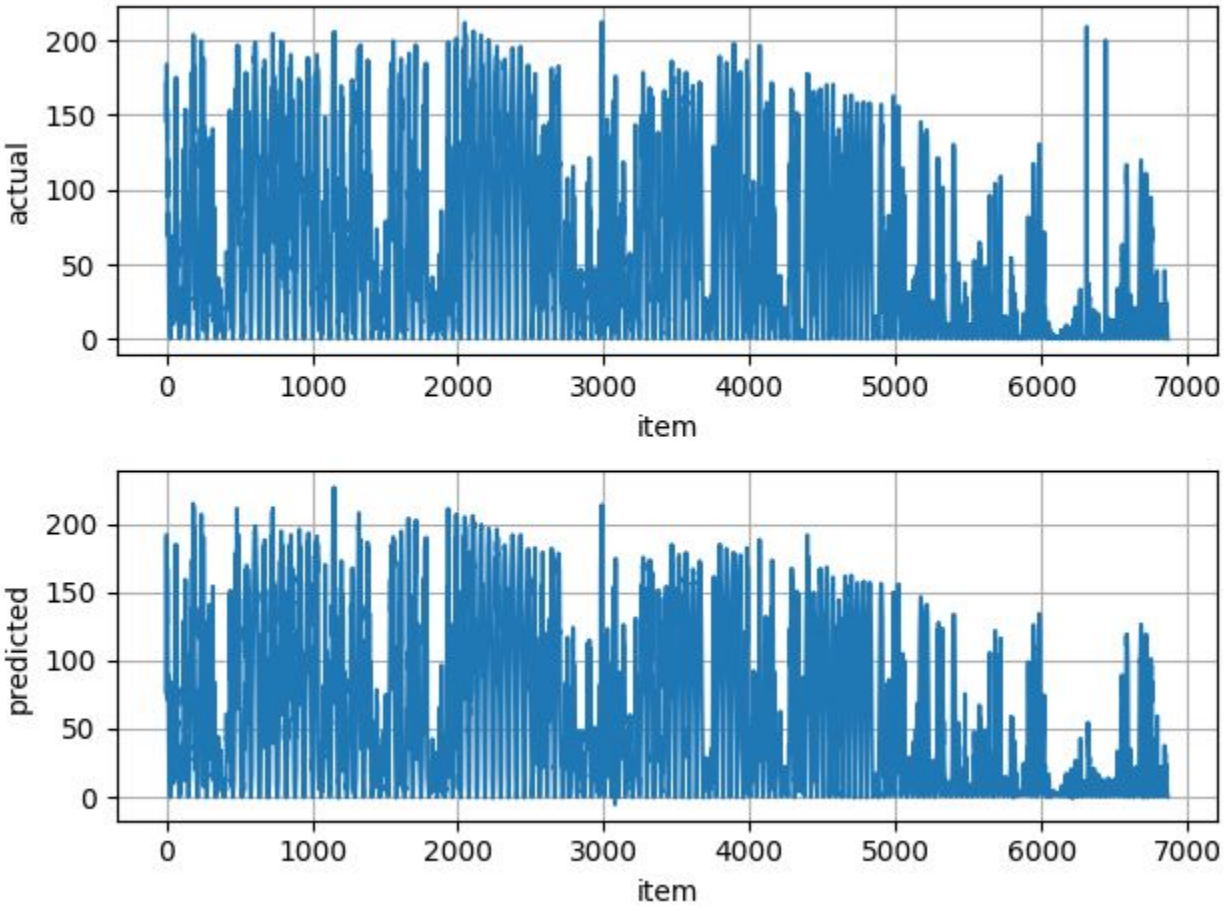
Wygenerowana moc w Kwh



Próbka testowa

Wizualizacja działania sieci DNN w przypadku farmy fotowoltaicznej

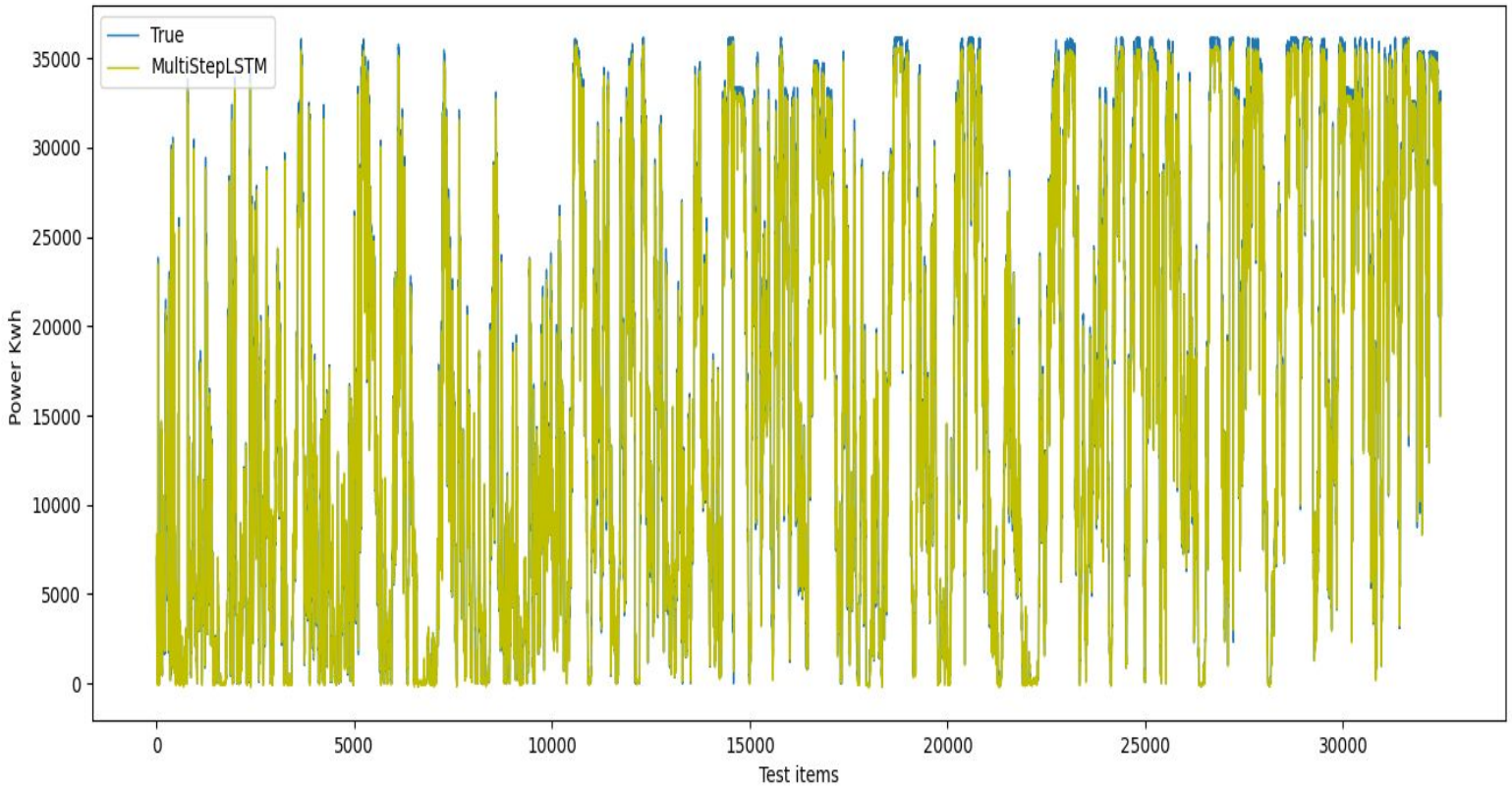
Wygenerowana moc w Kwh



Próbka testowa

Wizualizacja działania sieci DNN w przypadku farmy wiatrowej

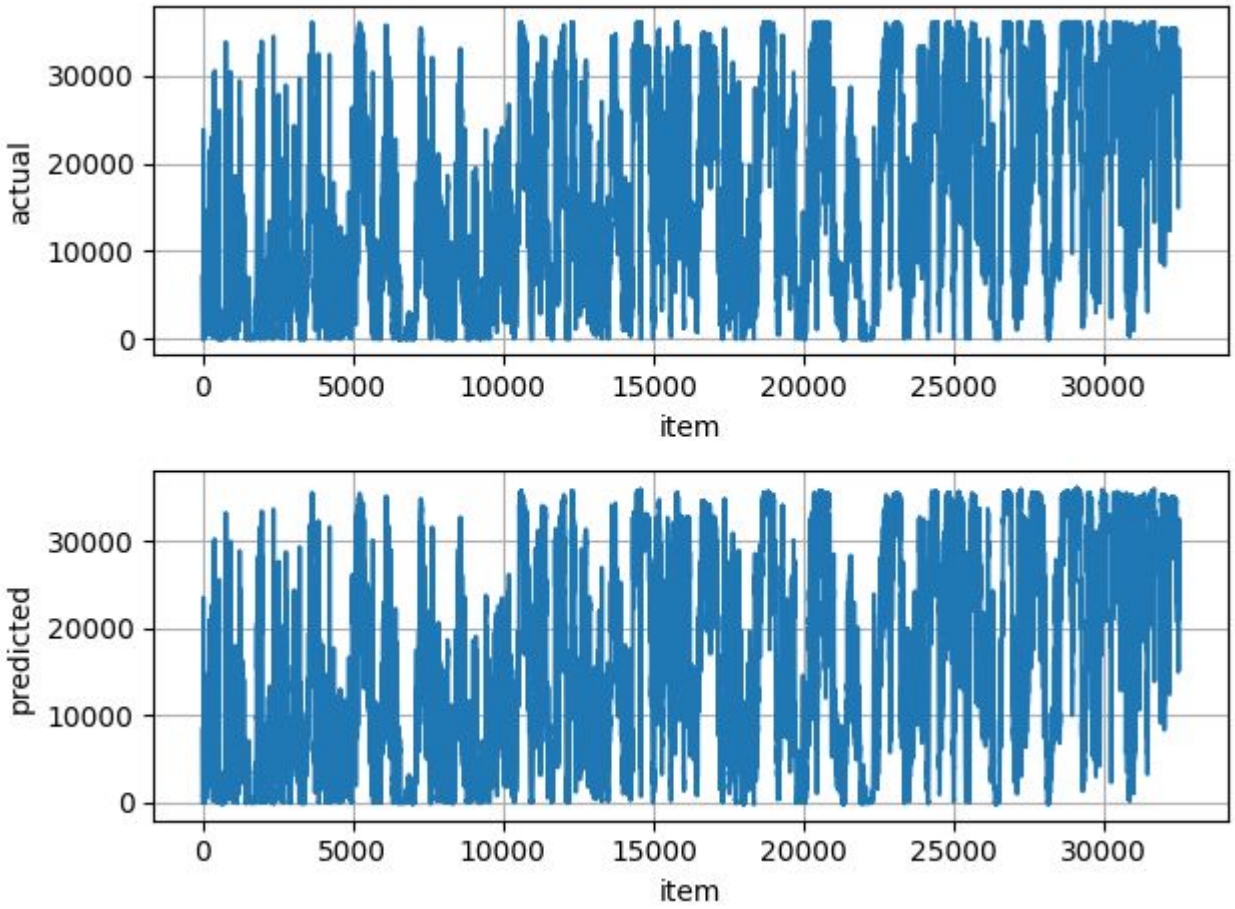
Wygenerowana moc w Kwh



Próbka testowa

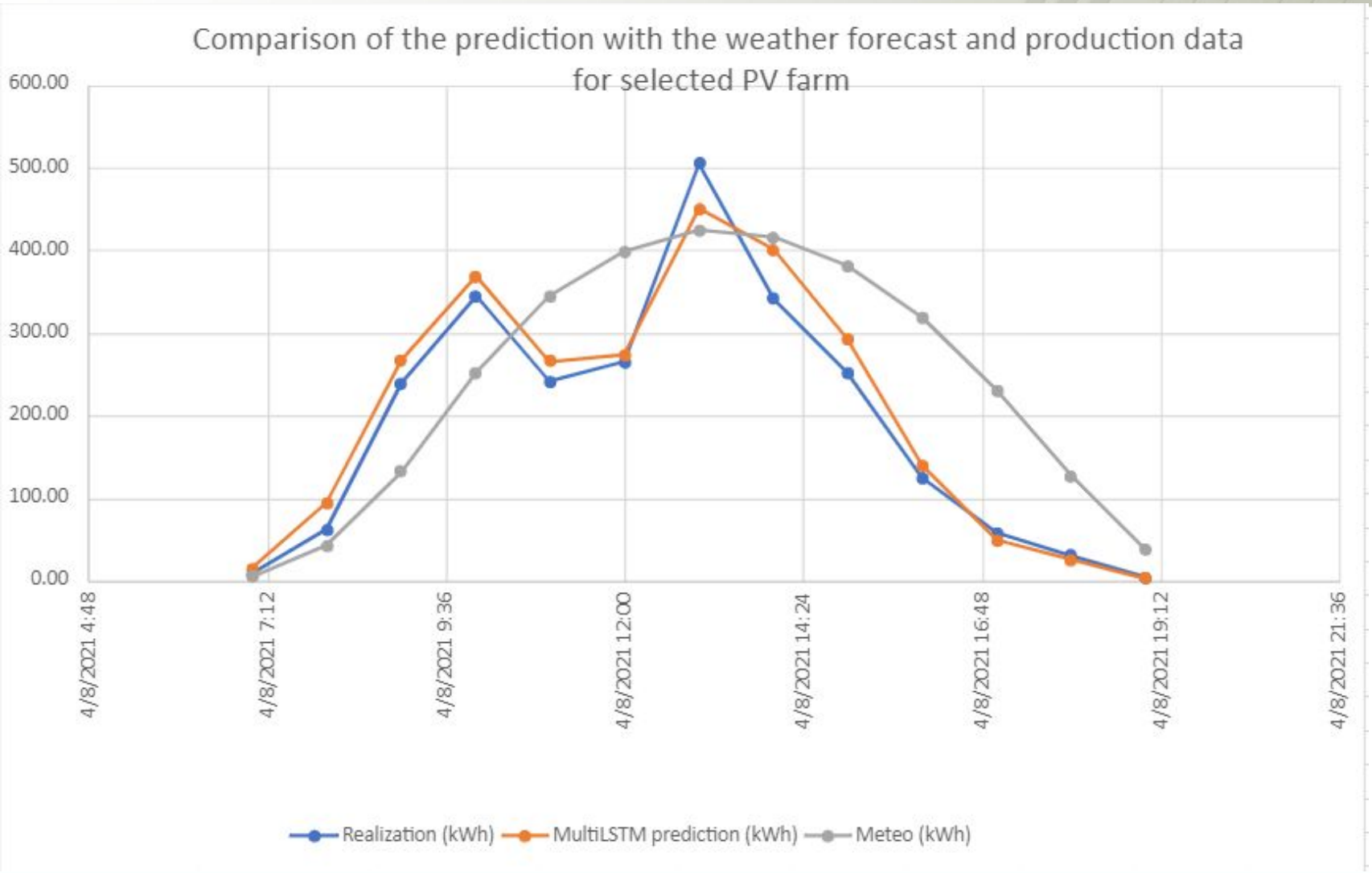
Wyniki WIND

Wygenerowana moc w Kwh



Próbka testowa

Porównanie z predykcją od dostawcy pogody



Zastosowanie ww. modeli w środowisku produkcyjnym

- **Współpraca z firmą Asseco - projekt wybrany do dalszego rozwoju**
- **Nawiązanie współpracy z PGE**
- **Przygotowywanie rozwiązania opartego o docker aby umożliwić integrację na serwer obliczeniowy**

Plany badawcze (future works)

- **Potencjalna integracja z systemami smart home (predykcja energii umożliwiająca sterowanie z wyprzedzeniem)**
- **Stworzenie taniego modułu elektronicznego umożliwiającego predykcję energii w czasie rzeczywistym dla mniejszych mikroinstalacji. Wykorzystanie mikrokontrolera STM32 + MLTiny**
- **Algorytmy sterowania dla systemów offgrid**
- **Optymalizacja ładowania / rozładowywania akumulatorów używanych w bankach energii**

Dziękuję za uwagę