Politechnika Warszawska
Wydział Mechaniczny Energetyki i Lotnictwa

Wykorzystanie sztucznych sieci neuronowych przy planowaniu zabiegów konserwacyjnych

Przewidywanie współczynnika zużycia gazowej turbiny napędowej przy użyciu prostej sieci neuronowej oraz Amazon Web Service.

Karol Roliński

21.01.2021 r.

1. Wstęp

Aby pozostać konkurencyjnym na rynku przedsiębiorstwa starają się ograniczać ponoszone koszty. Ważnym aspektem tej strategii jest właściwe planowanie konserwacji maszyn i urządzeń. Konserwacja jest istotnym czynnikiem zapewniającym jakość, w niektórych przypadkach decydującym o długoterminowym sukcesie firmy. Źle utrzymywane zasoby mogą powodować niestabilność i częściowo lub całkowicie wstrzymać produkcję. Wadliwe działanie maszyn lub nawet całkowite awarie mogą być źródłem znaczących kosztów.

Obecnie stosowana konserwacja polega na wymianie elementów układu, gdy ich żywotność dobiega końca. Odpowiedni moment na przeprowadzenie remontu można oszacować na wiele sposobów. Głównie dokonuje się tego na podstawie wcześniej zdobytego doświadczenia oraz znajomości średniej żywotności maszyny. Niestety ta metoda nie gwarantuje bezawaryjnego działania systemu. Ponadto może dojść do zbyt wczesnej wymiany komponentów oraz do zbyt częstych przeglądów, które mogą być bardzo kosztowne. Ustalenie właściwych interwałów między przeglądami z pewnością nie jest łatwym zadaniem, zwłaszcza przy zróżnicowanym wykorzystaniu maszyny np. w przemyśle morskim. Dlatego najlepszym rozwiązaniem dla redukcji kosztów wydaje się konserwacja CBM (ang. *Condition-Based Maintenance*). Polega ona na aktywnym monitorowaniu parametrów pracy maszyn i urządzeń w celu podjęcia decyzji, kiedy należy wykonać przegląd. W większości przypadków nie jest możliwe pobieranie informacji na temat stanu zużycia każdego elementu z osobna. CBM wymaga stworzenia dokładnego modelu zdolnego do przewidzenia poziomu zużycia poszczególnych elementów składowych na podstawie parametrów pracy całego systemu.

Zastosowanie technik uczenia maszynowego może wspomóc dokonywanie optymalnych decyzji dotyczących konserwacji. W dzisiejszych czasach większość zakładów przemysłowych posiada zaawansowaną sieć czujników zbierającą dane na temat każdego procesu w czasie rzeczywistym. Połączenie danych pomiarowych z informacjami na temat awarii pozwala na utworzenie przydatnych zestawów danych do celów konserwacji predykcyjnej.

2. Model

Aby móc przedstawić przydatność stosowania sztucznych sieci neuronowych przy planowaniu zabiegów konserwacyjnych potrzebny jest zbiór danych. W tym celu skorzystano z repozytorium danych UCI (*University of California Irvine*) [4]. Jednym z udostępnionych tam datasetów jest zestaw składający się z 11934 punktów pomiarowych. Został on otrzymany w wyniku zasymulowania pracy układu napędowego statku dalekomorskiego. W eksperymencie fregata napędzana była przez turbinę gazową. Symulator pracy tego układu został stworzony oraz udoskonalany przez rok na podstawie pomiarów pracy jednostek podobnego typu. Potwierdza to rzetelność otrzymanych wyników.

Dla każdego punktu pomiarowego dostępne są wartości 16 parametrów pracy układu napędowego, tj.:

- Położenie dźwigni,
- Prędkość statku [węzły],
- Moment obrotowy turbiny gazowej [kNm],

- Prędkość obrotu wału turbiny [obr/min],
- Prędkość obrotu wału sprężarki [obr/min],
- Napęd śruby prawej [kN],
- Napęd śruby lewej [kN],
- Temperatura spalin za turbiną [°C],
- Temperatura wlotowa powietrza do sprężarki [°C],
- Temperatura wylotowa powietrza ze sprężarki [°C],
- Ciśnienie wylotowe spalin [bar],
- Ciśnienie powietrza przed sprężarką [bar],
- Ciśnienie powietrza za sprężarką [bar],
- Ciśnienie spalin za turbiną [bar],
- Kontrola wtrysku paliwa (TIC) [%],
- Zużycie paliwa [kg/s][4].

Ponadto dla każdego punktu pomiarowego podane zostały następujące współczynniki:
- Współczynnik zużycia sprężarki,
- Współczynnik zużycia turbiny [4].

Utworzony model predykcyjny będzie na podstawie wartości parametrów pracy układu przewidywał wartości współczynników zużycia sprężarki oraz turbiny. Dla wszystkich punktów pomiarowych znajdują się one odpowiednio w przedziale [1; 0,95] oraz [1; 0,975] [4]. W następującej części pracy zostaną przedstawione kolejne kroki podejmowane przy tworzeniu modelu predykcyjnego z wykorzystaniem sieci neuronowych. Neurony sieci tworzą trzy rodzaje warstw:

- ✓ wejściową, która pobiera oraz przekazuje dane do kolejnej warstwy;
- ✓ ukrytą, w której zachodzi proces uczenia się, czyli szukania powiązań między poszczególnymi neuronami;
- ✓ wyjściową, która zwraca wyniki przeprowadzonej analizy [5].

Każda sieć składa się z jednej warstwy wejściowej, jednej wyjściowej oraz dowolnej liczby warstw ukrytych Zdecydowano się utworzyć model składający się z jednej warstwy ukrytej. Warstwa wejściowa będzie składać się z liczby neuronów odpowiadającej liczbie wykorzystywanych atrybutów. Model posiadać będzie jedną warstwę ukrytą z ośmioma neuronami oraz warstwę wyjściową z jednym neuronem. Celem modelu będzie przewidzenie wartości współczynnika zużycia turbiny gazowej.

2.1 Wstępna obróbka danych

Pierwszym krokiem w tworzeniu modelu sztucznej sieci neuronowej jest przygotowanie zbioru danych. Po załadowaniu bardzo ważne jest, aby modelowany proces był w pełni zrozumiały. Następnie wyeliminowane zostają atrybuty, które nie mają wpływu na daną etykietę. Wyboru można dokonać na podstawie znajomości rzeczywistych zależności lub wtórnie obserwując wstępnie uzyskany model. Elementy o wagach bliskich zeru są odrzucane.

```
import io
ds = pd.read_fwf(io.BytesIO(uploaded['data.txt']))
ds.columns = ["Lever_position", "Ship_Speed", "Turb_Shaft_Torque", "Turb_Rate", "Gen_Rate",
              "Star_Prop_Torque", "Port_Prop_Torque", "HP_Turb_Temp", "Comp_In_Temp", "Comp_Out_Temp",
              "HP_Turb_Exit_Pres", "Comp_In_Pres", "Comp_Out_Pres", "Turb_Exh_Pres", "Turb_Inj_Ctrl",
              "Fuel", "Comp_decay_state", "Turb_decay_state"]

ds = ds.drop(columns = ["Lever_position", "Ship_Speed", "Comp_In_Temp", "Comp_In_Pres", "Star_Prop_Torque"])

y_comp = ds.pop("Comp_decay_state")
y_turb = ds.pop("Turb_decay_state")
ds_train, ds_test, yt_train, yt_test = train_test_split( ds, y_turb, test_size = 0.3, random_state = 0)

nc = ds_train.shape[1]
```

Rys. 1. Obróbka wstępna danych

W przypadku tego modelu wyeliminowane zostały następujące atrybuty:

- Położenie dźwigni,
- Prędkość statku [węzły],
- Temperatura wlotowa powietrza do sprężarki [°C],
- Ciśnienie powietrza przed sprężarką [bar],
- Napęd śruby prawej [kN],

Następnie z datasetu wydzielono etykiety w wyniku czego otrzymane zostały 3 niezależne obiekty. Aby móc trenować oraz ewaluować model dane muszą zostać podzielone na zestaw szkoleniowy oraz testowy. Zdecydowano się przeznaczyć 70% dostępnych danych do szkolenia modelu. Funkcja dokonuje podziału w sposób losowy przy okazji mieszając kolejność poszczególnych instancji.

2.2 Budowa modelu

Jedną z najprostszych technik uczenia maszynowego są estymatory regresji liniowej. Doskonale nadają się do przewidywania wartości liczbowych. Bardziej zaawansowanym narzędziem są sztuczne sieci neuronowe, które oferują dużo szerszy zakres możliwości. Dlatego w tym przypadku postanowiono skorzystać z bardzo prostego rozwiązania, sieci neuronowej z jedną warstwą ukrytą. Podobnie, jak w przypadku estymatorów liniowych model można wykonać na wiele sposobów. Dla wygody skorzystano z biblioteki TensorFlow, a szczególnie wchodzącego w nią skład modułu o nazwie keras. Umożliwia on w prosty sposób tworzenie sztucznych sieci neuronowych. Ponadto bardzo elastyczna architektura biblioteki TensorFlow umożliwia łatwe wdrażanie obliczeń na różnych platformach (CPU, GPU, TPU), począwszy od komputerów stacjonarnych, przez klastry serwerów, skończywszy na urządzenia mobilnych [6].

Zgodnie z wcześniej wspomnianymi założeniami model będzie się składał z trzech warstw oznaczonych l_0, l_1 oraz l_2. Pierwsza warstwa służy wprowadzeniu danych do modelu, dlatego liczba neuronów równa jest liczbie wykorzystywanych atrybutów. W tym przypadku warstwa wejściowa składa się z 11 neuronów.

W pojedynczej warstwie ukrytej składającej się z 8 neuronów zachodzi główna część szkolenia układu. Najważniejszą częścią tworzenia modelu jest odpowiedni dobór funkcji aktywacji, który uzależniony jest od rodzaju zadania jakie ma pełnić sieć. Istnieją jednak pewne cechy, które musi posiadać każda z nich. Od funkcji aktywacji wymaga się, aby przejście między wartością minimalną i maksymalną było ciągłe, pochodna była ciągła i łatwa do obliczenia. Zazwyczaj w danej warstwie sieci stosuje się jeden wybrany typ funkcji aktywacji, ale funkcje te mogą być różne

```
l_0 = tf.keras.layers.InputLayer(input_shape=(nc,))
l_1 = tf.keras.layers.Dense(units=8, kernel_regularizer=l2(0.001), bias_regularizer=l2(0.001), activation='relu')
l_2 = tf.keras.layers.Dense(units=1)


model = tf.keras.Sequential([l_0, l_1, l_2])
model.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(learning_rate=0.0008))
model.fit(ds_train, yt_train, epochs=3000, batch_size= 12, verbose=0, validation_split=0.25, callbacks = [
                        keras.callbacks.EarlyStopping(monitor='val_loss', patience=200),
                        keras.callbacks.ModelCheckpoint(filepath='path', monitor='val_loss',
                                                        save_best_only=True, mode='min', verbose=1)])

y_pred = list(model.predict(ds_test))
yt_por = yt_test.reset_index(drop=True)
```

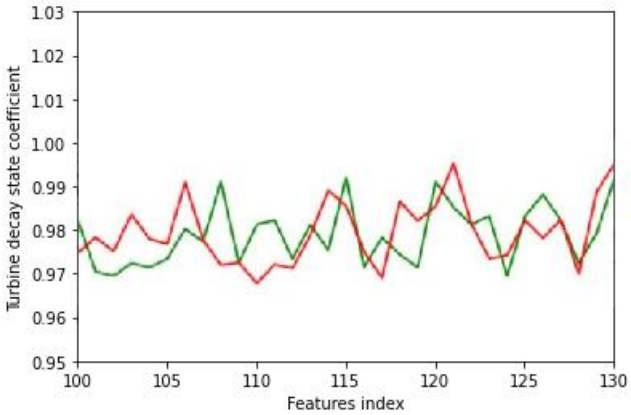
Rys. 2. Model sieci neuronowej

dla każdej z warstw. W przypadku tej sieci skorzystano z funkcji rampy ‘relu’ (ang. *Rectified Linear Unit*). Ponadto zastosowano dwa regulatory, tj. wag oraz przesunięcia. W modelu zastosowano optymalizator wykorzystujący algorytm Adam. Jest to jest iteracyjna metoda do optymalizacji, która opiera się na adaptacyjnej estymacji momentów pierwszego i drugiego rzędu [5].

Z datasetu treningowego wydzielony został podzbiór w celu sprawdzania przydatności modelu podczas każdej iteracji. Poświęcone 25% zbioru szkoleniowego służy bieżącemu monitorowaniu postępów modelu. Ponadto zostały zastosowane dwa wywołania zwrotne. Jedno z nich przeznaczone do przedwczesnego przerwania szkolenia, gdy błąd względem zbioru sprawdzającego nie ulegnie zmniejszeniu przez 200 epok. Drugi natomiast zapisuje takie wagi modelu dla których osiągnięto najniższy błąd.

2.3 Wyniki

W celu śledzenia efektywności modelu podczas jego tworzenia na bieżąco monitorowano średniokwadratowy błąd zestawu testowego względem zbioru sprawdzającego. Po dokonaniu kilku drobnych poprawek ostateczna wersja modelu miała stratę 5,88e-5. Na poniższym rysunku przedstawiono fragment porównania predykcji (kolor czerwony) i rzeczywistych danych (kolor zielony).



3. Obliczenia w chmurze

Cloud computing to usługa, która umożliwia gromadzenie oraz przetwarzanie danych bez konieczności korzystania z oprogramowania zainstalowanego na sprzęcie lokalnym. Korzyści płynące z takiego rozwiązania to przede wszystkim ograniczenie kosztów oraz wygoda. Serwisy udostępniające obliczanie w chmurze gwarantują mobilność, duże moce obliczeniowe oraz pamięci do przechowywania danych. Usługa jest bardzo kuszącym rozwiązaniem, zwłaszcza jeśli sporadycznie wymagane jest korzystanie z dużych mocy obliczeniowych, ponieważ zwykle ma charakter pay-per-use [7].

Amazon Web Service oferuje swoim użytkownikom różnorodne narzędzia. Jedno z nich jest specjalnie dedykowane do technik uczenia maszynowego. Amazon SageMaker umożliwia użytkownikom tworzenie, szkolenie i wdrażanie modeli uczenia maszynowego w chmurze. Jest to idealne narzędzie na potrzeby tej pracy, ponieważ pozwala na uruchomienie Jupyter notebook na wybranej przez użytkownika instancji bez konieczności ręcznej konfiguracji. Drugim użytecznym narzędziem udostępnianym przez AWS jest Amazon Simple Storage Service (S3). Jest to usługa pamięci masowej oferująca wysoką wydajność i bezpieczeństwo przechowywanych danych [7].

Aby móc skorzystać z usługi SageMaker należy najpierw zapisać uzyskany wcześniej model. W tym przypadku zapisano model oraz wagi w dwóch oddzielnych plikach.

```
!mkdir -p dnn
save_path='./dnn/'
model.save_weights(os.path.join(save_path,"uci3_model_v2_weights.h5"), "w")
model_json=model.to_json()
with open(os.path.join(save_path,"uci3_model_v2.json"), "w") as json_file:
  json_file.write(model_json)
```

Rys. 3. Zapisywanie modelu w formacie .json oraz .h5

Następnie po zalogowaniu na portalu AWS SageMaker oraz wybraniu instancji, z której w późniejszym etapie będziemy korzystać, uruchamiamy Jupyter Notebook. W nim na wstępie importujemy szereg bibliotek umożliwiających korzystanie z interjesu Pythona, późniejsze wykonywanie poleceń w środowisku SageMaker oraz tworzenie modeli sieci neuronowych.

```
import boto3, re
from sagemaker import get_execution_role

role = get_execution_role()

import tensorflow.keras
from tensorflow.keras.models import model_from_json

!mkdir keras_model
```

Rys. 4. Rozpoczęcie pracy w AWS SageMaker

Po stworzeniu folderu keras_model przesyłamy wcześniej zapisany model do tego folderu. Następnie ładujemy model oraz jego parametry do Jupyter notebook. Należy zwrócić uwagę, aby załadować pliki w odpowiednim formacie danych.

```
import tensorflow as tf

json_file = open('./keras_model/'+'uci3_model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json, custom_objects={"Adam": tf.keras.optimizers.Adam(learning_rate=0.0008),
                                                                  "L2": tf.keras.regularizers.l2(0.001)})
loaded_model.load_weights('./keras_model/uci3_model_weights.h5')
```

Rys. 6. Ładowanie modelu do środowiska SageMaker

Po załadowaniu danych do Jupyter notebook należy przygotować przestrzeń na serwerze AWS. W dedykowanym miejscu zostanie nasz model zapisany, żeby w łatwy sposób mógłby być przez użytkownika wywołany do wykonania predykcji. Po skompilowaniu wdrożenia zostaje ono zapisane.

```
from tensorflow.python.saved_model import builder
from tensorflow.python.saved_model.signature_def_utils import predict_signature_def
from tensorflow.python.saved_model import tag_constants

if tf.executing_eagerly():
   tf.compat.v1.disable_eager_execution()

model_version = '1'
export_dir = 'export/Servo/' + model_version

build = builder.SavedModelBuilder(export_dir)
signature = predict_signature_def(inputs={"inputs": loaded_model.input}, outputs={"score": loaded_model.output})

from keras import backend as K

import tensorflow.compat.v1.keras.backend as K

with K.get_session() as sess:
    build.add_meta_graph_and_variables(
        sess=sess, tags=[tag_constants.SERVING], signature_def_map={"serving_default": signature})
    build.save()
```

Rys. 5. Przygotowanie wdrożenia do zapisu modelu w przestrzeni AWS

Następnie utworzony zostaje plik tar pełniący rolę archiwum. Zostanie on zapisany w chmurze AWS, tj. w S3. Po zdefiniowaniu skąd model ma zaczytywać dane, zostaje stworzony plik train.py, który zostanie użyty do rozpoczęcia szkolenia modelu. Aby móc uruchomić model sztucznej sieci neuronowej w środowisku AWS SageMaker należy wywołać go w sposób przedstawiony poniżej. Zostaje zdefiniowane miejsce, z którego ma być zaczytywany model, wersja TensorFlow oraz nazwa instancji, która ma zostać do tego użyta. W tym przypadku użyta do tego zostaje jedna instancja standardowego przeznaczenia o nazwie ‘ml.t2.medium’.

```
import tarfile
with tarfile.open('model.tar.gz', mode='w:gz') as archive:
    archive.add('export', recursive=True)

import sagemaker
sagemaker_session = sagemaker.Session()
inputs = sagemaker_session.upload_data(path='model.tar.gz', key_prefix='model')

!touch train.py

from sagemaker.tensorflow.model import TensorFlowModel
sagemaker_model = TensorFlowModel(model_data = 's3://' + sagemaker_session.default_bucket() + '/model/model.tar.gz',
                                  role = role,
                                  framework_version = '2.2.0',
                                  entry_point = 'train.py')

predictor = sagemaker_model.deploy(initial_instance_count=1,
                                  instance_type='ml.t2.medium')
```

Rys. 7. Definiowanie pracy modelu w środowisku SageMaker

Po wyszkoleniu modelu zostaje utworzony tzw. ‘endpoint’. Aby móc w pełni wykorzystać możliwości AWS potrzebny nam jest ‘endpoint’, żeby móc uruchomić ten konkretnie model z dowolnej platformy posiadającej dostęp do sieci. Poniżej widoczny jest sposób wywołania modelu. Po podaniu dowolnego zestawu atrybutów model podaje wynik zgodnie z oczekiwaniami.

```
predictor.endpoint
```

```
'tensorflow-inference-2021-01-14-13-59-27-071'
```

```
endpoint_name = 'tensorflow-inference-2021-01-14-13-59-27-071'


import sagemaker
from sagemaker.tensorflow.model import TensorFlowModel
predictor=sagemaker.tensorflow.model.TensorFlowPredictor(endpoint_name, sagemaker_session)


import json
import boto3
import numpy as np
import io

client = boto3.client('runtime.sagemaker')

ds_test = [[1309.492,1372.592,6649.217,8.741,471.177,547.352,1.14,6.11,1.019,2.172,0.096]]


result = predictor.predict(ds_test)
print(result)
```

```
{'predictions': [[0.981211364]]}
```

```
sagemaker.Session().delete_endpoint(predictor.endpoint_name)
print(f"deleted {predictor.endpoint_name} successfully!")
```

Rys. 8. Finalizacja modelu

Aby nie generować niepotrzebnych kosztów należy po skończeniu sesji upewnić się, że instancja została zwolniona. Dokonuje się tego poprzez usunięcie endpoint’u.

4. Wnioski

Stworzony model sztucznej sieci neuronowej w skuteczny sposób pozwala określić współczynnik zużycia turbiny. Można tego dokonać zarówno w środowisku python 3 na lokalnym komputerze oraz z wykorzystaniem serwisu AWS w chmurze. W obu przypadkach otrzymany zostaje ten sam wynik.

Dostępność takich narzędzi jak Amazon SageMaker czy Amazon S3 pozwala użytkownikom mniej doświadczonym w kilku prostych krokach uruchomić zdalną instancję w chmurze. Duża prostota oraz niskie koszty sprawiają, że każdy może skorzystać z możliwości jakie oferuje chmura. Połączenie uczenie maszynowe z szeroką gamą usług oferowanych przez AWS, stwarza nieskończenie wiele możliwości rozwoju dla zarówno dla użytkownika prywatnego jak również przedsiębiorstwa.

Bibliografia

1. Coraddu A., Oneto L., *Machine Learning Approaches for Improving Condition-Based Maintenance of Naval Propulsion Plants*, Journal of Engineering for the Maritime Environment, 2014, DOI: 10.1177/1475090214540874,
2. Kothamasu R., Huang, S. H., *Adaptive mamdani fuzzy model for condition-based maintenance,* Fuzzy Sets and Systems, 158, 2715-2733,

3. Cipollini F., Oneto L., *Condition-Based Maintenance of Naval Propulsion Systems with Supervised Data Analysis*, Preprint submitted to Ocean Engineering May 25, 2018,
4. http://archive.ics.uci.edu/ml/datasets/condition+based+maintenance+of+naval+propulsion+plants,
5. https://www.sztucznainteligencja.org.pl/definicja/sieci-neuronowe,
6. https://www.tensorflow.org,
7. https://aws.amazon.com.