Uniwersytet Rzeszowski

Sztuczna inteligencja

Model sieci neuronowej rozpoznającej klasy zawarte w zbiorze ACS-F1

Prowadzący:

dr inż. Jacek Bartman

Autor:

Michał Komsa

Informatyka- I rok, studia magisterskie

Numer albumu: 123 633

Grupa laboratoryjna: Lab2

Rzeszów r.a. 2024/2025

Spis treści

1. Charakterystyka problemu i zbioru danych

ACS-F1 to baza danych opracowana przez Instytut iCoSys (Institute of Artificial Intelligence and Complex Systems), zawierająca sygnatury zużycia energii elektrycznej przez urządzenia domowe, przeznaczona do zadań rozpoznawania urządzeń.

Zbiór danych został stworzony w 2013 roku i obejmuje 100 urządzeń podzielonych na 10 kategorii:

- Lodówki i zamrażarki,
- Telewizory (LCD lub LED),
- Kuchenki mikrofalowe,
- Systemy Hi-Fi,
- Laptopy (podłączone do ładowania),
- Automaty do kawy,
- Stacje komputerowe,
- Telefony komórkowe (podłączone do ładowania),
- Lampki,
- Drukarki.

Każde urządzenie było monitorowane przez dwie sesje trwające po godzinie, z częstotliwością próbkowania wynoszącą 0,1 Hz (co 10 sekund), co pozwoliło na uchwycenie pełnych profili zużycia energii w różnych stanach pracy. Pomiar obejmował sześć parametrów: moc czynną, moc bierną, prąd RMS, napięcie RMS, częstotliwość oraz fazę napięcia względem prądu.

Ze względu na to, iż dane te reprezentują pomiary wykonywane w czasie to postanowiono zachować sekwencyjne ich przetwarzanie oraz pomiary (360 pomiarów na obiekt) zapakować w jedno cykl czasu. Tak przetworzone dane składały się z 200 dwuwymiarowych cykli czasowych (360x6). Każda klasa była reprezentowana przez 20 takich cykli. Wśród tych danych nie było brakujących danych ani cech kategorialnych.

2. Preprocessing danych

Pierwszą przeszkodą na drodze załadowania danych do modelu była postać danych, jaka była udostępniana na stronie źródłowej, tj. https://icosys.ch/acs-f1. W pobranym archiwum znajdowało się wiele plików takich jak wykresy w różnych formatach i pliki źródłowe danych. Pliki te posiadały rozszerzenie .mat, mimo iż nic nie miały wspólnego z tym językiem programowania. Pomiary tam znajdujące się były wypisane w sposób płaski „po spacji”. Aby usunąć zbędne pliki wykorzystano skrypt znajdujący się na Rysunek 1.

```python
from pathlib import Path

datasets = Path('datasets')

for dir in datasets.iterdir():
    label = datasets / dir.name

    for file in label.iterdir():
        if '.mat' not in file.name:
            file.unlink()
```

Rysunek 1. Kod odpowiedzialny za usunięcie zbędnych plików.

Tak załadowane dane automatycznie przy odczytywaniu pakowane w cykle czasowe posiadające 360 rekordów. Niektóre pomiary nie posiadały pełnej ilości 360 rekordów, więc w celu uzupełnienia danych do jednolitego kształtu uzupełniano brakujące dane 0 na samym początku cyklu. Rekurencyjne sieci neuronowe są dużo czulsze na dane znajdujące się na końcu cyklu niż na początku. Posłużył do tego skrypt znajdujący się na Rysunek 2.

```python
colnames = ['phAngle', 'freq', 'reacPower', 'power', 'rmsVolt', 'rmsCur']

data = []
labels = []

for dir in datasets.iterdir():
    label = datasets / dir.name

    for file in label.iterdir():

        with open(file, 'r') as f:
            file_content = f.readlines()

        file_content = [line[1:-2] for line in file_content if line[0] != '#']
        columns = np.array([list(map(float, row.split(' '))) for row in file_content])
        padded = tf.keras.preprocessing.sequence.pad_sequences(columns, maxlen=360, dtype='float32', padding='pre').T

        data.append(padded)
        labels.append(data_labels[file.parent.name])

data, labels = np.array(data), np.array(labels)
```

Rysunek 2. Kod odpowiedzialny za załadowanie danych z plików.

Kolejnym krokiem było przetworzenie danych do takiej postaci, w której sieć mogłaby z nimi jak najlepiej pracować. W tym celu wykorzystano interfejs potoków udostępniamy przez bibliotekę scikt-learn. Interfejs ten jednak nie jest przystosowany do pracy z danymi trójwymiarowymi (ilość obiektów, ilość rekordów, ilość cech), więc aby klasy mechanizmy scikt-learn mogły poprawnie przetwarzać te dane własnoręcznie zaimplementowano własne transformery danych, które zarządzały kształtem danych na początku i na końcu potoku. Transformery te znajdują się na Rysunek 3.

```python
from sklearn.base import BaseEstimator, TransformerMixin


class Reshape3DTo2D(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X.reshape(-1, 6)


class Reshape2DTo3D(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X.reshape(-1, 360, 6)
```

Rysunek 3. Implementacja niestandardowych transformerów danych.

Na ich podstawie skomponowano 4-etapowy potok, który kolejno zmienia dane na dane tabelaryczne, uzupełnia braki danych (jakby takowe się pojawiły w nowych danych wchodzących do modelu), skaluje dane do wartości z przedziału od 0 do 1 i na koniec ponownie zmienia je w cykle czasowe. Ostateczny potok znajduje się na Rysunek 4.

```python
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler


preprocess = Pipeline([
    ('to2D', Reshape3DTo2D()),
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', MinMaxScaler()),
    ('to3D', Reshape2DTo3D())
])
```

Rysunek 4. Implementacja potoku przetwarzania danych.

Kolejnym krokiem przed przekazaniem danych do sieci neuronowej było wydzielenie 3 zbiorów danych:

- Treningowego,
- Walidacyjnego,
- Testowego.

Podział przeprowadzono w proporcji 70%-20%-10% za pomocą losowania stratyfikowanego, aby w każdym zbiorze był taki sam procent obiektów każdej klasy.

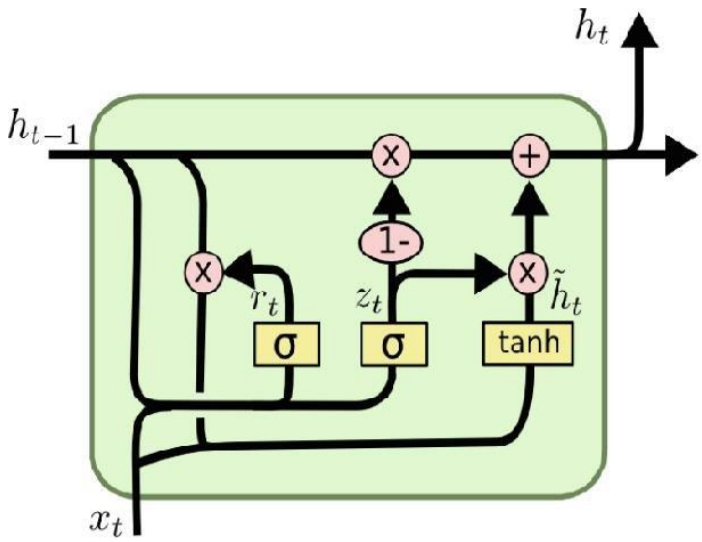
Jednak to nie koniec- ze względu na małą ilość danych wykonano również augmentację danych treningowych. Na podstawie zbioru treningowego wygenerowano dodatkowo po 2 próbki na każdy cykl dodając do nich nieznaczny szum wygenerowany na bazie klasycznego rozkładu normalnego. Generowanie danych zostało ukazane na Rysunek 5.

```
# Augmentation new data with noise
X_train_aug = []
Y_train_aug = []

for i, arr in enumerate(X_train):
    for j in range(2):
        if j == 0: np.random.seed(SEED)
        else: np.random.seed(SEED+1)
        X_train_aug.append(arr + np.random.normal(0, 0.001, size=arr.shape))
        Y_train_aug.append(Y_train[i])

X_train_aug = np.array(X_train_aug)
Y_train_aug = np.array(Y_train_aug)

X_train = np.concatenate([X_train, X_train_aug], axis=0)
Y_train = np.concatenate([Y_train, Y_train_aug], axis=0)
```

Rysunek 5. Kod odpowiedzialny za augmentację danych.

3. Projekt sieci neuronowej

Ze względu na przetwarzania danych w sposób sekwencyjny model nie mógł być prostą siecią feed-forward. Aby skutecznie przetwarzać takie dane, należy wykorzystać sieci rekurencyjnego. Podstawowa jednostka rekurencyjna jest jednak bardzo prosta i w praktyce rzadka daje dobre rezultaty. W ostatecznym modelu wykorzystano warstwę GRU (*Gated Recurrent Unit*), czyli warstwę rekurencyjną działającą na bazie dwóch bramek- bramki wejściowej oraz zapominającej. Schemat komórki GRU znajduje się na Rysunek 6.



Rysunek 6. Komórka GRU. Źródło: SBAG: A Hybrid Deep Learning Model for Large Scale Traffic Speed Prediction.

Po więcej informacji odsyłam do pracy *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. Rysunek 7 przedstawia kod, który został wykorzystany do utworzenia modelu sieci neuronowej.

```
reset_seeds(SEED)

def make_model():
    return tf.keras.Sequential([
        tf.keras.layers.Input(shape=(360, 6)),
        tf.keras.layers.GRU(units=256, return_sequences=True, reset_after=True),
        tf.keras.layers.GlobalMaxPooling1D(),
        tf.keras.layers.Dense(units=10, activation='softmax'),
    ])

model = make_model()
model.load_weights('started.weights.h5')

model.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.Nadam(),
              metrics=['accuracy']
)

model.summary()

mc1 = tf.keras.callbacks.ModelCheckpoint('best_model_1.keras', monitor='val_accuracy', mode='max', save_best_only=True)
mc2 = tf.keras.callbacks.ModelCheckpoint('best_model_2.keras', monitor='val_loss', mode='min', save_best_only=True)

preprocess.fit(X_train)
history = model.fit(
    preprocess.transform(X_train),
    tf.keras.utils.to_categorical(Y_train),
    validation_data=(preprocess.transform(X_validate), tf.keras.utils.to_categorical(Y_validate)),
    batch_size=8,
    epochs=100,
    callbacks=[mc1, mc2],
)
```

Rysunek 7. Kod odpowiedzialny za stworzenie i wytrenowanie modelu.

Strukturę wynikowego modelu ukazano na Rysunek 8.

Model: "sequential"

Layer (type)	Output Shape	Param #
gru (GRU)	(None, 360, 256)	202,752
global_max_pooling1d (GlobalMaxPooling1D)	(None, 256)	0
dense (Dense)	(None, 10)	2,570

Total params: 205,322 (802.04 KB)

Trainable params: 205,322 (802.04 KB)

Non-trainable params: 0 (0.00 B)

Rysunek 8. Schemat wykorzystanego modelu.

Jak można zauważyć na Rysunek 8, model ten składa się z 3 warstw, nie licząc warstwy wejściowej, która tylko wprowadza dane do modelu:

- **GRU**: główna rekurencyjna warstwa modelu posiadająca 256 neuronów. Wykorzystanie jej z flagą *return_sequences* powoduje, że warstwa nie zwraca jedynie końcowego stanu dla cyklu, dla wektor stanów każdego z rekordów cyklu.
- **GlobalMaxPooling1D**: warstwa wyciągająca maksymalną wartość z wektora zwróconego przez warstwę GRU.
- **Dense**: wyjściowa warstwa gęsto połączona posiadająca 10 neuronów. Wykorzystanie jej z funkcją aktywacji powoduje, że jej neurony symbolizują prawdopodobieństwo przynależności cyklu do każdej z 10 etykiet. Prawdopodobieństwa te sumują się do wartości 1.

Na podstawie przytoczonego podsumowania można również odczytać, że całkowity ciężar wag, które sieć będzie optymalizować wynosi ponad 800KB, co przekłada się na ponad 200 000 wag do uczenia.

Po skomponowaniu modelu nadchodzi czas na jego kompilację, czyli konfiguracji wszystkich wymaganych elementów:

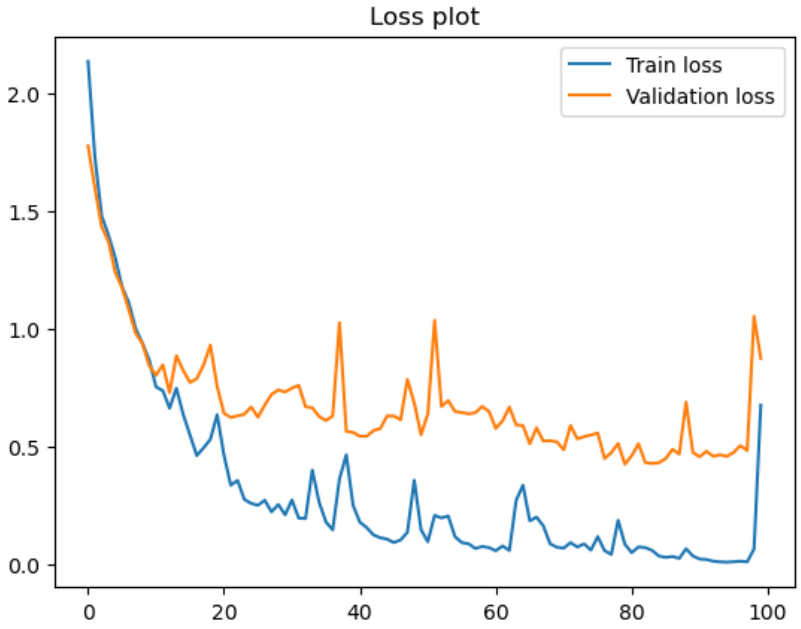
- **Funkcja straty**: kategorialna entropia krzyżowa (*categorical crossentropy*). Funkcja straty wykorzystywana w problemach klasyfikacji wieloklasowej, w której etykiety są przekazywane w postaci wektora zakodowanego w sposób one-hot (*one-hot encoding*).
- **Optymalizator**: NADAM (*Adaptative Moment Estimation with Nesterov momentum*). Obecnie najsilniejszy z optymalizatorów (ADAM) wzbogacony o sztuczkę z momentum Nesterova.
- **Metryki**: klasyczna dokładność jako jedyna metryka wykorzystywana w uczeniu modelu.

Przed uczeniem, stworzono również wywołania zwrotne, których zadaniem było zapisywanie modelu do pliku za każdym razem, kiedy poprawi się ogólna dokładność lub funkcja straty na zbiorze walidacyjnym.

Po tych operacjach przeprowadzono uczenie modelu korzystając z wcześniej opisanego potoku przetwarzania danych. W jednym cyklu wykorzystano wsad o wielkości 8 obiektów. Cały proces uczenia przeprowadzono w wymiarze 100 epok.

4. Analiza uzyskanych wyników

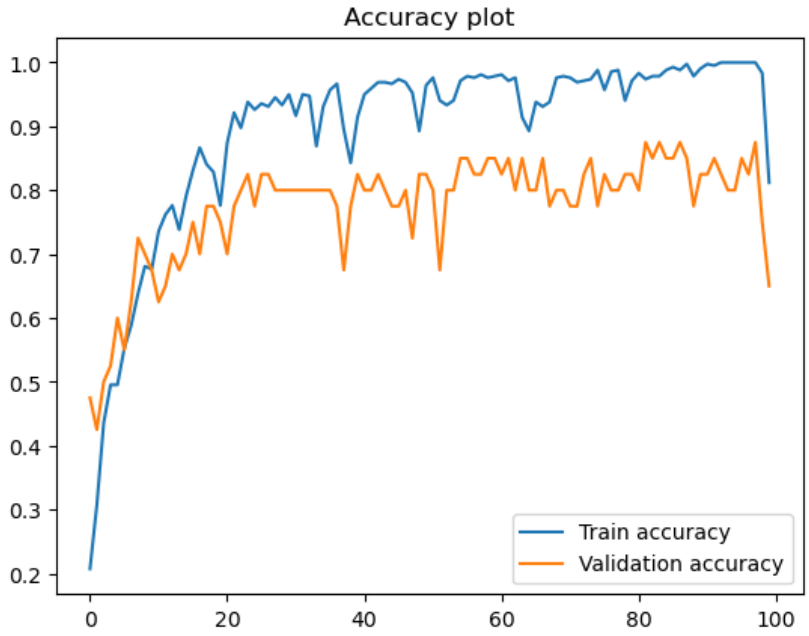
Na początku pragnę przedstawić krzywe uczenia. Rysunek 9 przedstawia wartość funkcji straty na zbiorach walidacyjnym i treningowym.



Rysunek 9. Wykres zmiany wartości funkcji straty w procesie uczenia.

Jak można zauważyć na Rysunek 9 proces uczenia przebiegał zgodnie z intuicją. Model stale starał się dopasowywać do danych treningowych, toteż na tym zbiorze wartość funkcji straty jest mniejsza. Modele rekurencyjne dla dużych cykli czasowych korzystające z silnych optymalizatorów bywają niestabilne. Na tym wykresie również to widać- funkcja straty, która powinna stale maleć posiada bifurkacje.

Rysunek 10 przedstawia wykres zmiany metryki dokładności, który można skomentować podobnie jak wykres na Rysunek 9.

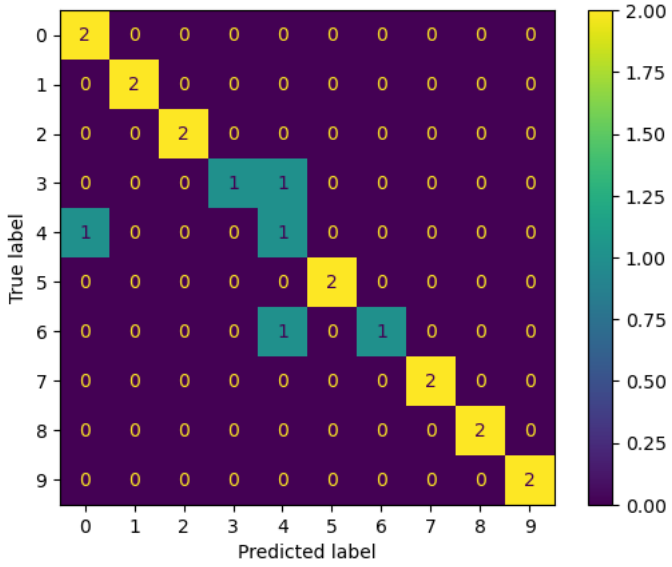


Rysunek 10. Wykres zmiany metryki dokładności w procesie uczenia.

Niestabilność modelu przezwyciężono za pomocą wywołań zwrotnych, które zapisywały najlepszą wartość modelu do wskazanego pliku. Po zakończeniu procesu uczenia załadowano te modelu i sprawdzono jego zdolność do przewidywania klas na obcych do tej pory danych (na zbiorze testowym, który posiada 10% danych, tj. 20 obiektów). Rysunek 11 przedstawia wyniki tego podsumowania.

```
Test accuracy: 0.85
Test recall: 0.85
Test precision: 0.9
Test f1: 0.8533333333333333
2025-05-19 17:22:24.209590: E tensorflow/core/framework/node_def_util.cc:680] NodeDef
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f38a17f6600>
```



Rysunek 11. Ewaluacja modelu na danych testowych.

Na samym początku można zauważyć podstawowe metryki:

- **Accuracy/Dokładność: 0.85**
- **Recall/Czułość: 0.85**
- **Precision/Precyzja: 0.9**
- **F1: 0.85(3)**

Oznaczają one całkiem dobry model, który ogólną dokładność predykcji posiada na poziomie 85%. Jak na 10 klas jest wynik całkiem dobry.

Poza podstawowymi metrykami wygenerowano również macierz konfuzji, która konkretnie pokazuje, jak się rozkładały błędy klasyfikacji. Na jej podstawie można stwierdzić, że model zbyt często przewiduje klasę 4 (stacje komputerowe) kosztem klas 3 (systemy Hi-Fi) i 6 (laptopy podłączone do ładowania).

Niezależnie od głównej sieci neuronowej wykonano również na identycznym modelu ewaluację za pomocą walidacji krzyżowej ze stratyfikowanym podziałem na 5 foldów, która dała średnią dokładność na poziomie **74.5%**.

5. Wnioski

W przeprowadzonym projekcie udało się skutecznie zaimplementować model rekurencyjnej sieci neuronowej typu GRU do klasyfikacji urządzeń na podstawie ich sygnatur energetycznych z bazy danych ACS-F1.

Dzięki odpowiedniemu przygotowaniu danych, uwzględniającemu zarówno przetwarzanie sekwencyjne, jak i odpowiedni preprocessing, model osiągnął wysoką skuteczność predykcji na poziomie 85% dokładności. Proces augmentacji danych okazał się mieć duże znaczenie dla procesów uczenia korzystających walidacji krzyżowej- modele bez augmentacji osiągały niejednokrotnie dokładność na poziomie 20-50%.

Mimo niewielkiej liczby dostępnych próbek, struktura modelu oraz zastosowane techniki przetwarzania danych wykazały się dużą skutecznością. Ogólnie osiągnięte rezultaty potwierdzają zasadność zastosowania architektury GRU w zadaniach klasyfikacji danych czasowych i stanowią dobrą podstawę do dalszych prac nad rozbudową i doskonaleniem tego typu modeli.