Rețele neuronale LSTM pentru serii de timp

Lumînăraru Ionuț - 342

Duzi Mihai-Nicolae - 352

Rezumat

- Această lucrare analizează diverse modele de inteligență artificială pentru serii de timp, cu scopul prezicerii consumului de curent electric al unei locuințe. Pentru testarea modelelor, am folosit setul de date **Individual Household Electric Power Consumption**. Sunt prezentate și comparate modele auto-regresive, printre care și modele din sfera învățării adânci precum rețelele neuronale LSTM și transformes.
- Motivația acestui studiu este de a găsi o metoda de regresie care sa prezică cu o precizie rezonabilă consumul cu electricitate dintr-o gospodărie pe o anumită perioadă având in vedere volumul de date si resursele fizice
- Principalele metode de a rezolva acest task sunt: modele ARMA, retele Long Short-Term Memory, Transformers.
- Soluția pe care am ales-o este LSTM deoarece LSTM-urile sunt avantajoase față de Transformers în cazul seturilor de date de dimensiune medie, deoarece necesită mai puțini parametri, sunt mai stabile la antrenare și modelează natural dependențele secvențiale, având un consum redus de memorie.

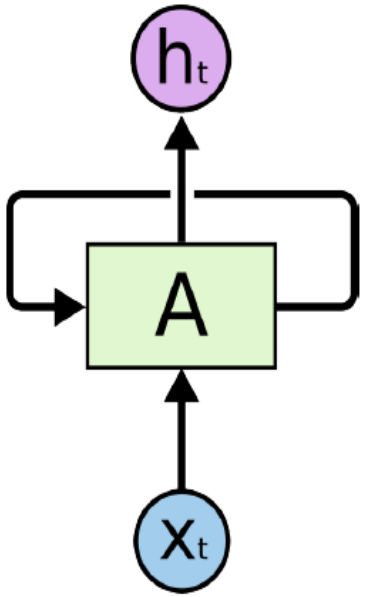
Setul de date

- In acest dataset avem 9 features-uri
- primele 2 nu vor fi folosite deoarece se subînțeleg din logica seriei de timp.
- Feature-ul Global_active_power este cel pe care vrem sa îl prezicem iar restul sunt date exogene(date care au o corelație mare cu feature-ul pe care vrem să îl prezicem) care vor fi folosite pentru a imbunătății prezicerea.
- Deoarece setul inițial conținea peste două milioane de eșantioane, am decis să facem o operație de downsampling, luând măsurătorile din oră în oră, astfel reducând setul de date la aproximativ 35000 de eșantioane. Pentru fiecare oră am luat media celor 60 de minute, cu excepția intensității, unde am luat valoarea maximă.

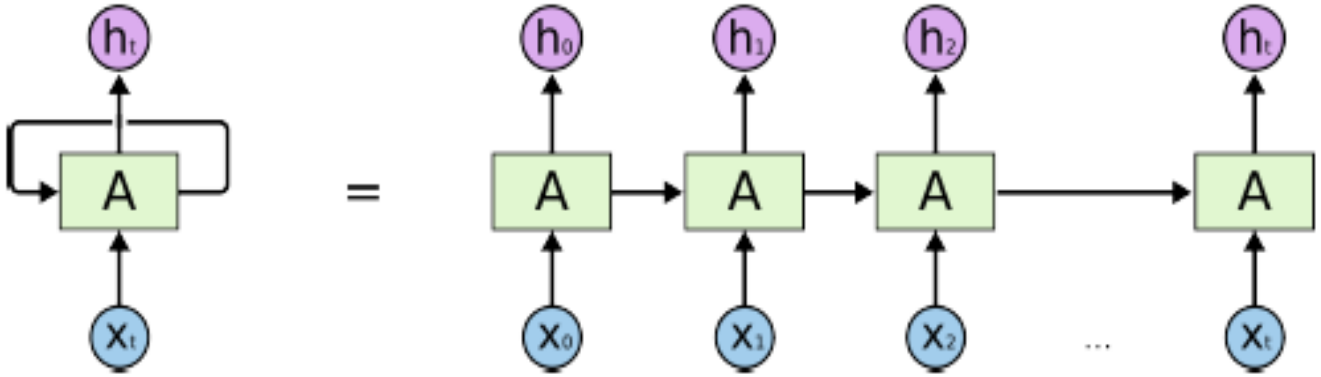
Date	Feature	Date			no
Time	Feature	Categorical			no
Global_active_power	Feature	Continuous			no
Global_reactive_power	Feature	Continuous			no
Voltage	Feature	Continuous			no
Global_intensity	Feature	Continuous			no
Sub_metering_1	Feature	Continuous			no
Sub_metering_2	Feature	Continuous			no
Sub_metering_3	Feature	Continuous			no

RNN(Recurent Neural Network)

- Rețelele neuronale sunt printre cele mai puternice modele din învățarea automată. Însă, rețelele neuronale clasice au o problemă majoră, anume lipsa de memorie. Cu o rețea clasică nu te poți, folosi de date din trecut pentru a prezice datele din viitor, cum ar fi consumul de curent al unei case.
- Nu putem spune același lucru și despre rețelele neuronale recurente, care pot ține în memorie informații și despre valorile eșantioanelor trecute, pentru a prezice valorile din viitor. Mai jos avem schema unei rețele neuronale recurente, respectiv a unei rețele neuronale recurente „desfăcute”.



Rețeaua neuronala recurentă scrisă desfășurat



Cum funcționează modelul

- **w_1 = ponderea elementului current**
- **b_1 = bias-ul elementului curent**
- **w_2 = ponderea memoriei anterioare**
- **w_3 = ponderea finală**
- **b_3 = bias-ul final**

Fie şirul k, unde:

$k_1 = ReLU(x_1w_1 + b_1)$

$k_{i+1} = ReLU(x_{i+1}w_1 + k_iw_2 + b_1), 1 < i \leq n.$

Predict, ia finală pentru următorul element va fi $w_3k_n + b_3$. Mai jos avem un exemplu pentru seria de timp [1, 0.8, 0.6], $w_1 = 0.5$, $w_2 = -1$, $w_3 = 0.5$, $b_1 = 0.5$, $b_3 = 0$

$k_1 = ReLU(1 \cdot 0.5 + 0.5) = 1$

$k_2 = ReLU(0.8 \cdot 0.5 + 1 \cdot (-1) + 0.5) = 0$

$k_3 = ReLU(0.6 \cdot 0.5 + 0 \cdot (-1) + 0.5) = 0.8$

$\widehat{x4} = 0.5 \cdot 0.8 + 0 = 0.4$

Vanishing gradient și Exploding Gradient

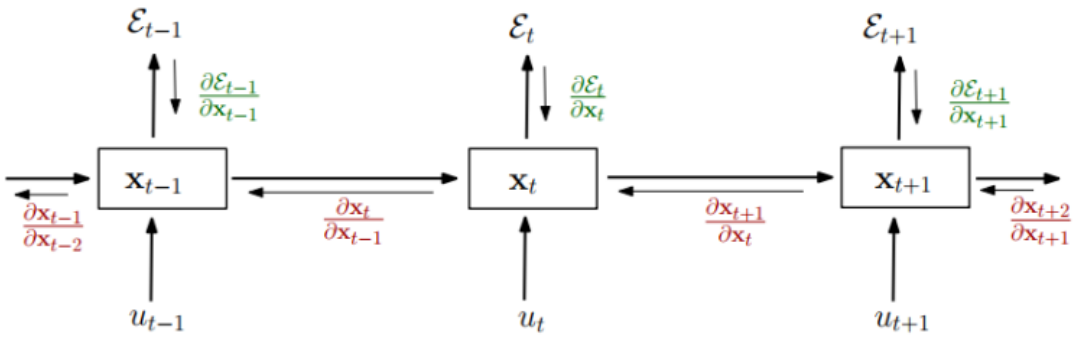
Întrucât la fiecare termen nou calculat folosim aceleași ponderi, după mai multe operat, ii vom ajunge la una din cele 2 probleme:

a. Dispariția gradientului, care are loc atunci când w1 este în intervalul (-1, 1). Din cauza înmulţirii repetate cu valori subunitare, gradientul va ajunge să fie 0.

b. Explozia gradientului, care are loc atunci când w1 este în afara intervalului (-1, 1). Din cauza înmulţirii repetate cu valori supraunitare, gradientul crește exponențial, iar la back-propagare se fac schimbări foarte mari care fac imposibilă convergența modelului.

Problema apare atunci cand $\frac{\partial Xi}{\partial Xi-1}$ <1 $\Rightarrow$ Vanishing

Sau cand $\frac{\partial Xi}{\partial Xi-1} > 1 \Rightarrow Exploding$



$$\mathbf{x}_t = F(\mathbf{x}_{t-1}, \mathbf{u}_t, \theta) \quad (1)$$

$$\mathbf{x}_t = \mathbf{W}_{rec}\sigma(\mathbf{x}_{t-1}) + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b} \quad (2)$$

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \le t \le T} \frac{\partial \mathcal{E}_t}{\partial \theta} \quad (3)$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \le k \le t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right) \quad (4)$$

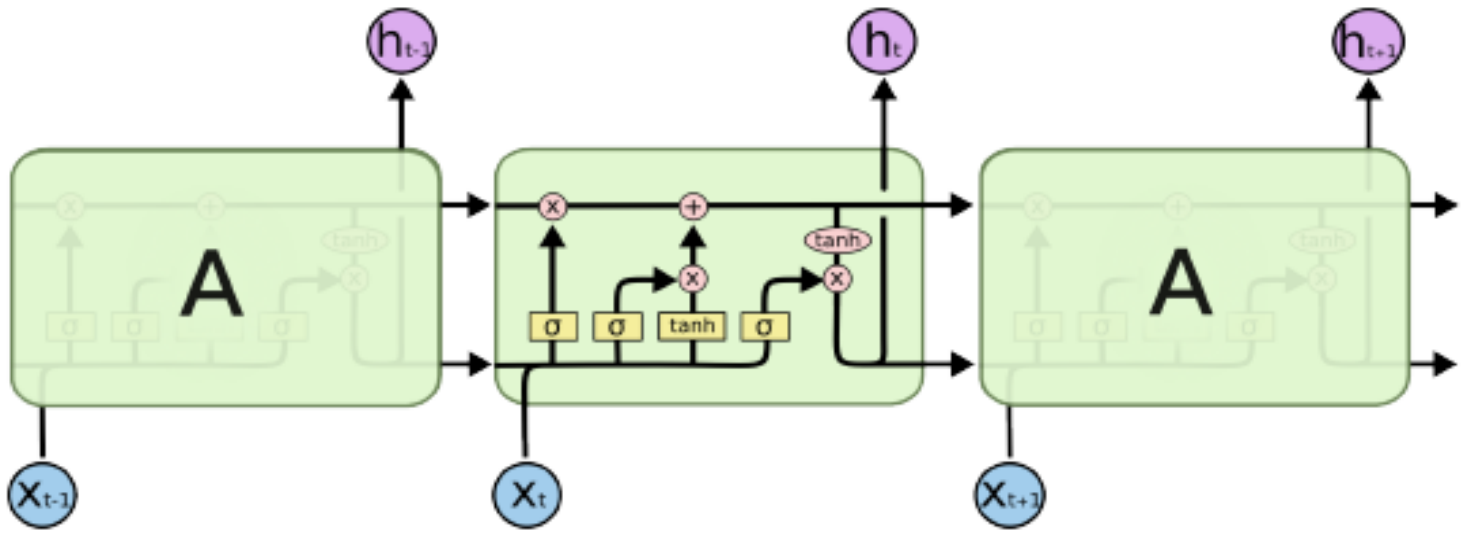
$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \ge i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \ge i > k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1})) \quad (5)$$

LSTM(Long Short-Term-Memory) networks

După cum am menționat anterior, rețelele neuronale nu reușesc să țină în memorie prea multe eșantioane din cauza problemei gradientului. Această problemă nu există însă și la rețelele neuronale LSTM care folosesc două tipuri de memorie:

- a. Memoria pe termen scurt (Short-Term) – aceeași cu cea din RNN
- b. Memoria pe termen lung (Long-Term) - poate reține informații pe termen lung pe care RNN-ul nu le putea reține

Mai jos avem schema unei rețele neuronale LSTM:


h_{t-1}
h_t
h_{t+1}
A
A
tanh
σ
σ
tanh
σ
X_{t-1}
X_t
X_{t+1}

Notăm cu C și h memoria pe termen lung, respectiv scurt și le inițializăm cu 0. De asemenea, avem seria de timp x cu n elemente, iar noi vrem să prezicem următorul element. De asemenea, avem următoarele ponderi:

Wf-Forget Gate,

Wi - Input Gate

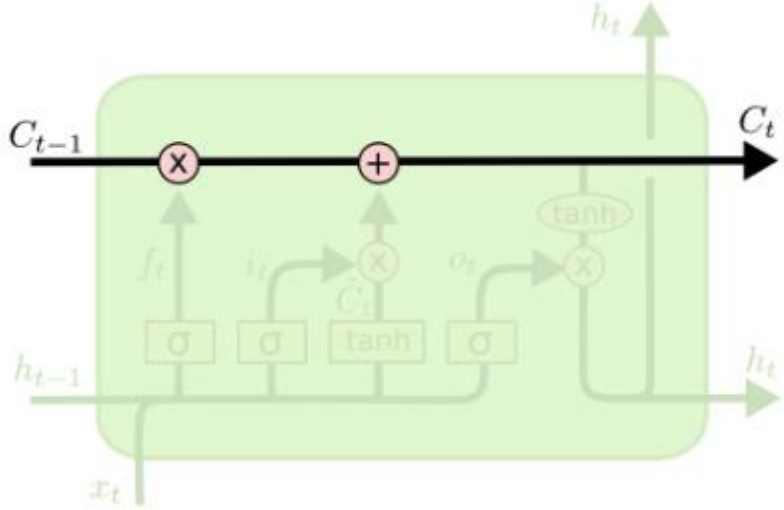
Wc Candidate Gate

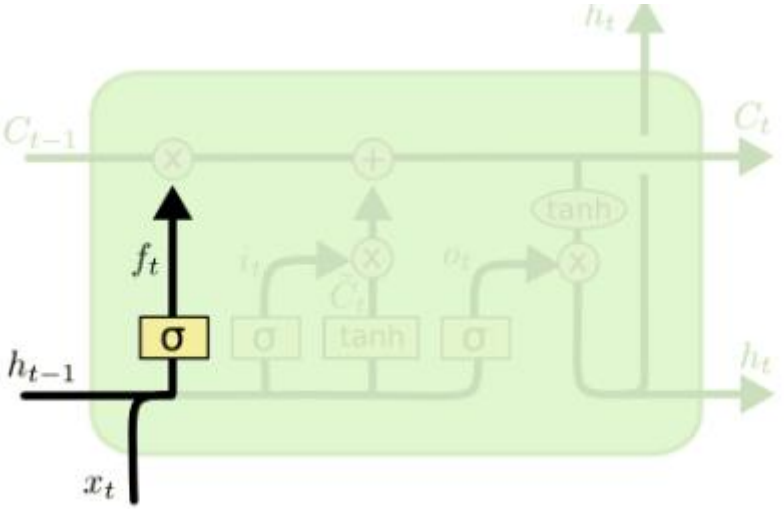
Wo - Output Gate.

Respectiv, bias-urilelor: bf , bi, bc, bo.

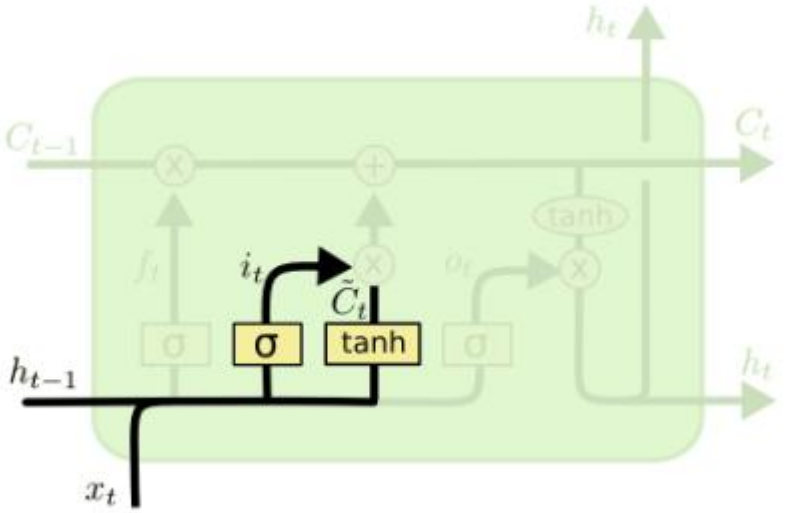
Primul pas: Forget Gate În acest pas decidem câte procente din L păstrăm folosind formula:

f = sigmoid(Wf ▪ [x, h] + bf).

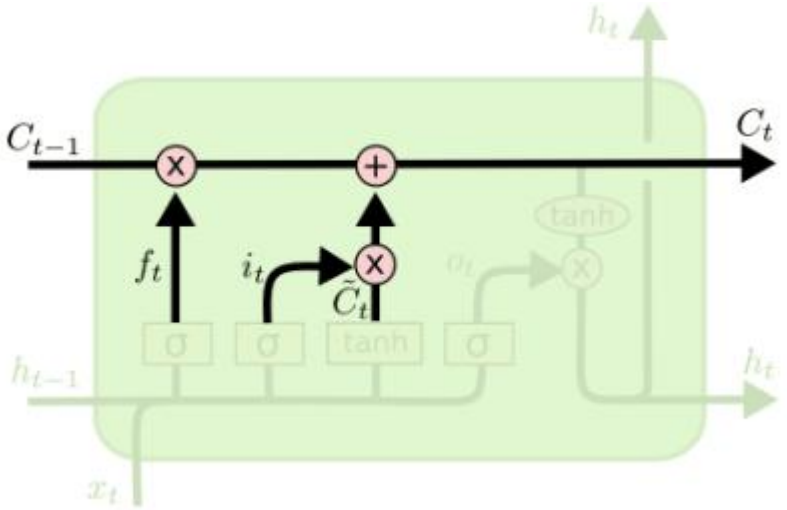




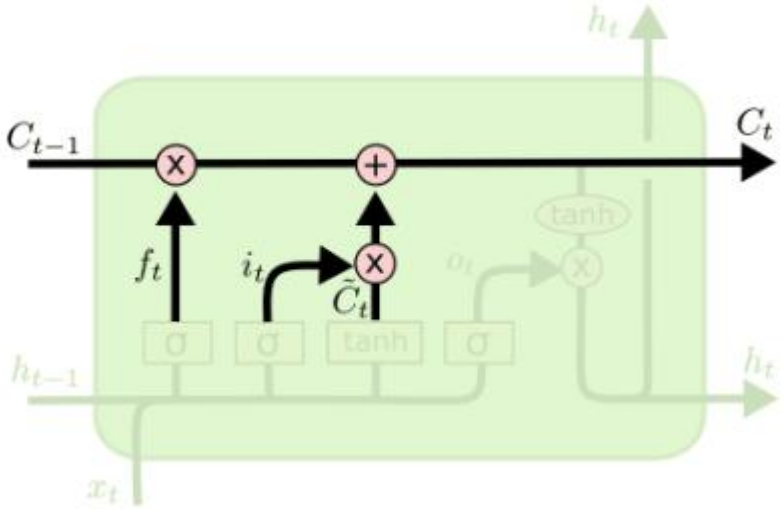
i = sigmoid(Wi ▪ [x, h] + bi).



Cnou = tanh(Wc ▪ [x, h] + bc).

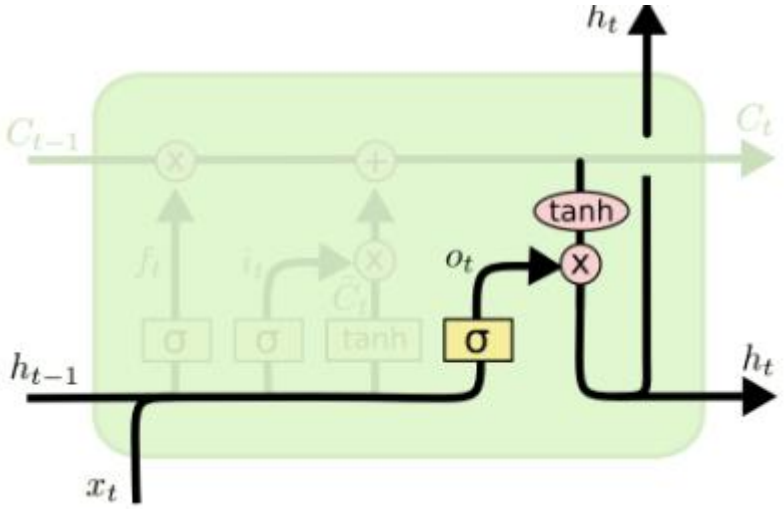


C = Cvechi ・ f + Cnou ・ i.



o = sigmoid(Wo ・ [x, h] + bo).

hnou = o ・ tanh(C)



Repetăm procedeul folosind noua memorie și parcurgând seria de timp. Când ajungem la finalul ei, predict, ia va fi dată de valoarea sau de valorile din hnou.

Observăm că $\frac{\partial C}{\partial Cvechi}$ = f, și deoarece f este o valoare între 0 şi 1 riscul de a apărea problema gradientului este mult mai scăzut.

$$\frac{\partial s_{t'}}{\partial s_t} = \prod_{k=1}^{t'-t} \sigma(v_{t+k})$$

Functia de forward pentru o unitate LSTM in python folosind pytorch

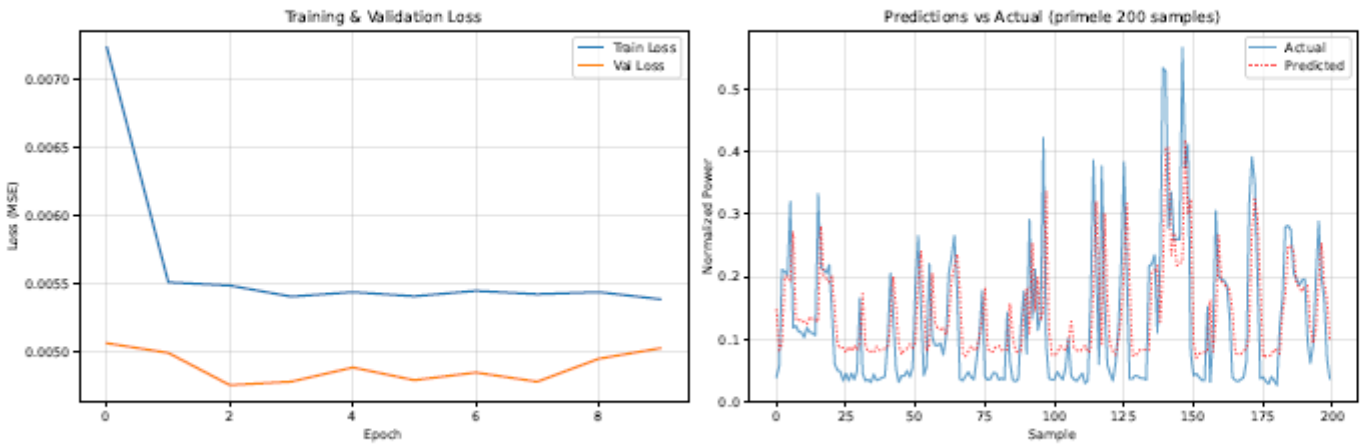
```
    self.U_o = nn.Linear(hidden_size, hidden_size, bias=False)

def forward(self, x_t, h_prev, C_prev):
    f_t = torch.sigmoid(self.W_f(x_t) + self.U_f(h_prev))
    i_t = torch.sigmoid(self.W_i(x_t) + self.U_i(h_prev))
    C_tilde = torch.tanh(self.W_c(x_t) + self.U_c(h_prev))

    C_t = f_t * C_prev + i_t * C_tilde

    o_t = torch.sigmoid(self.W_o(x_t) + self.U_o(h_prev))
    h_t = o_t * torch.tanh(C_t)

    h_t = self.dropout(h_t)

    return h_t, C_t
```

Rezultate

- Inițial, am antrenat o rețea neuronală LSTM doar pe seria de timp a coloanei ,,Global_active_power” din setul de date menționat de mai sus.
- Pentru antrenament am folosit:
- MinMaxScale pentru date
- Secvente de lungime 48 si un batch de 64
- 3 straturi si un dropout de 20%
- Funcția de pierdere optimizată este Huber Loss(un hibrid intre MSE si MAE)
- Optimizatorul folosit este Adam

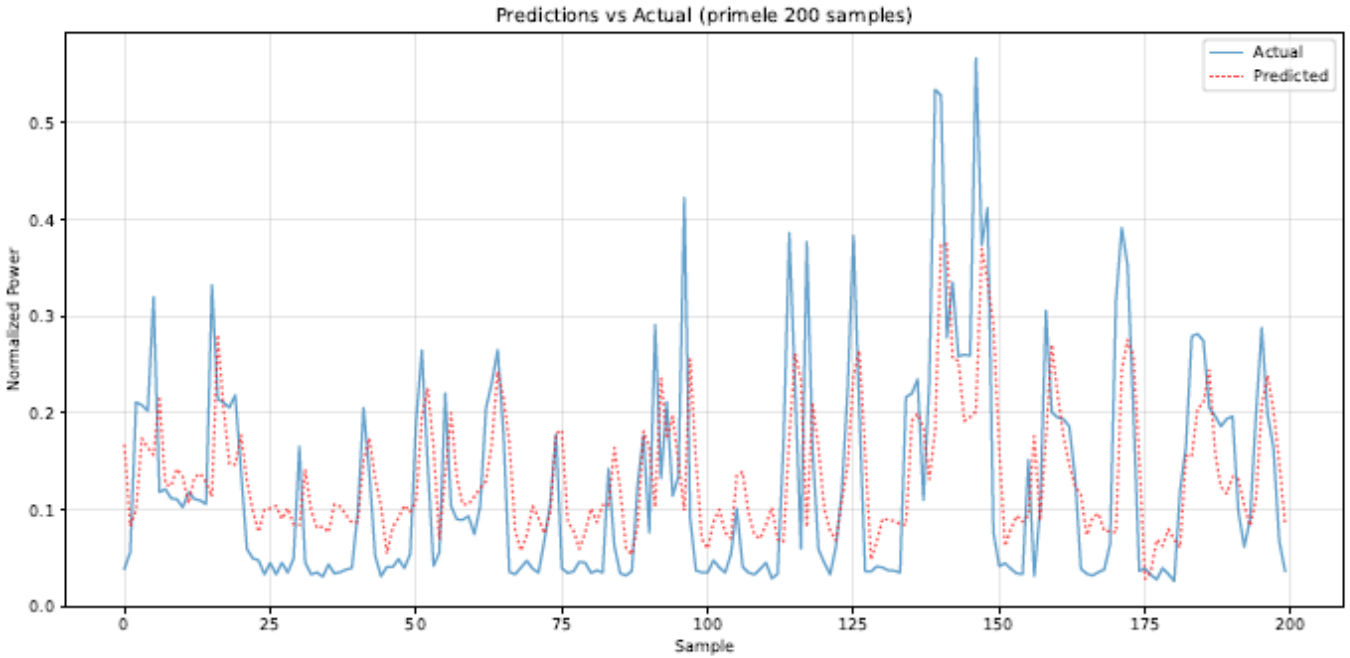
Rezultate fără exogene

Test Loss (MSE): 0.003395, Test RMSE: 0.058268.



Rezultate cu date exogene

Test Loss (MSE): 0.003182, Test RMSE: 0.056405.

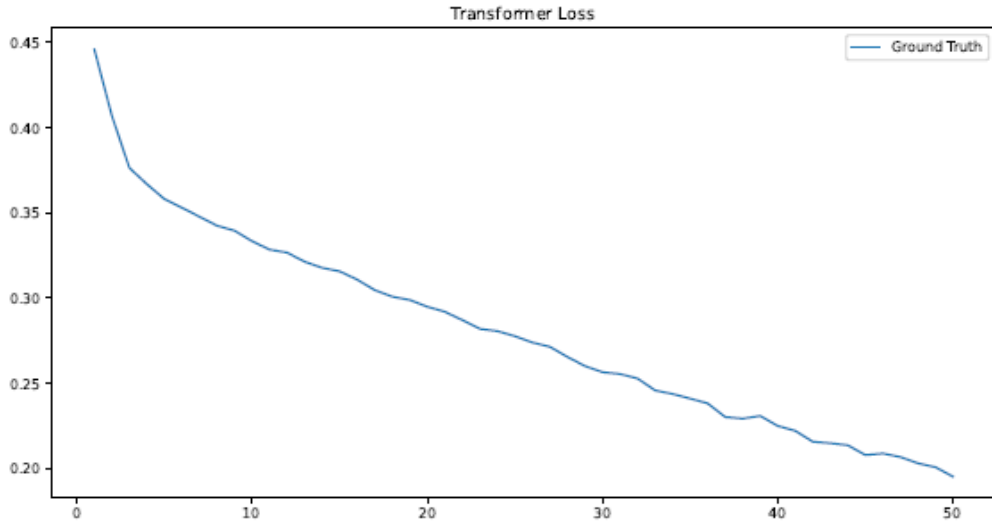


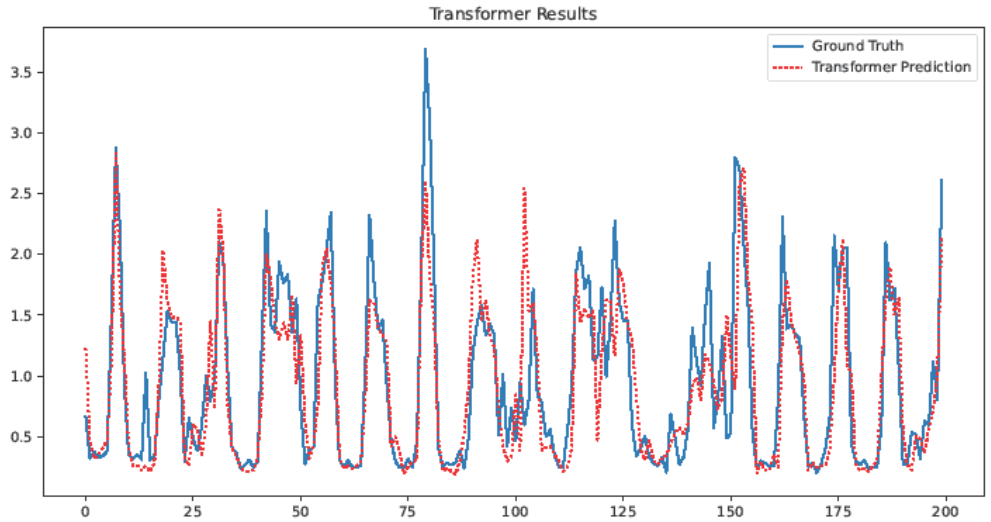
Alte modele de predicție
Transformer

- Chiar dacă rețelele neuronale LSTM sunt mult mai bune in ceea ce privește memoria pe termen lung acestea sunt tot supuse riscului riscului problemei gradientului.
- La transformeri în schimb aceasta problemă dispare de tot deoarece între fiecare două poziții din secvență, fie ele cuvinte, tokeni sau numere dintr-o serie de timp, ,,distanța” este 1. Cu alte cuvinte un transformer se poate uita oricât de mult in spate, fără ca acesta să piardă context.
- Un alt avantaj important al transformerilor față de RNN sau LSTM este paralelizarea.

Rezultat Tranformer

- Am antrenat un transformer cu următorii hiperparametri: Lungimea secvenței: 24, Stare ascunsă: 64, Straturi: 2, Capete: 4, Epoci: 50, Normalizare: StandardScaler, Rata de învățare: 5 $*10^{-4}$, Funcție de optimizat: Mean Squared Error, Optimizator: Adam.



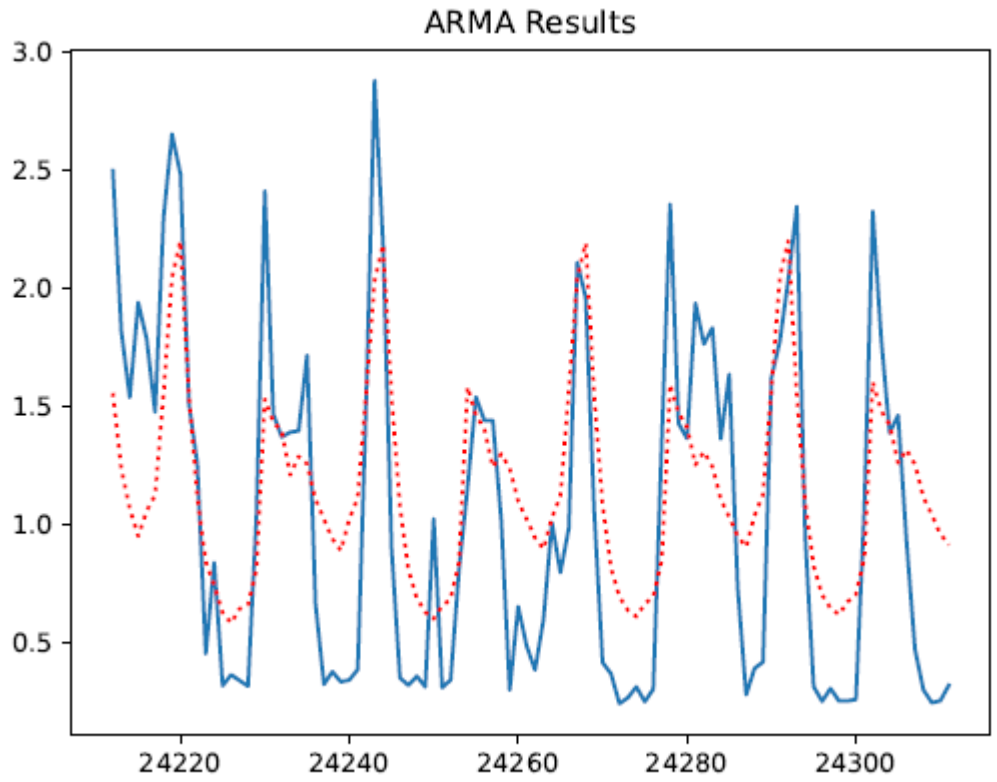


ARMA

- AR - Modelul AR, sau Autoregresiv, este după cum îi spune și numele un model de regresie adaptat la serii de timp. Ideea de bază este că trecutul influențează viitorul, iar acest lucru este modelat matematic de o combinație liniară a ultimelor p valori din trecut, unde p este dimensiunea modelului. Formula matematică este următoarea:
- $\widehat{y\_i} = X^T$ y, unde X reprezintă cele p ponderi ale modelului, iar y reprezintă ultimele p valori înainte de cea de pe poziția i
- ARMA - Modelul ARMA este combinația dintre AR și MA. Vom avea atât parametri autoregresivi, cât și parametri de mediere (Moving Average). Din acest motiv, minimizarea erorii acestor parametri este mult mai complicată.”

Rezultat ARMA

- În urma antrenării modelului sezonal ARMA, care se uită la secvențe de câte 24 de eșantioane, sau 24 de ore, am folosit un parametru autoregresiv, un ordin de diferențiere și un parametru de mediere, obținând media pătratelor erorilor de 23.8666.



Concluzii

- Modelul ARMA este de departe cel mai slab dintre cele prezentate
- Datele exogene în LSTM au scăzut loss-ul cu ~6.28%
- Transformer-ul are un loss mai mare din cauza scalării standard, dar în grafice se vede că a dat cele mai bune rezultate
- De asemenea, transformer-ul este mult mai rapid, am antrenat 50 de epoci în 3-4 minute față de LSTM unde 10 epoci au durat 15 minute.

Posibile îmbunătățiri

- Încercarea a mai multor combinații de hiperparametri, în special la ARMA pentru o performanță mai bună.
- Antrenarea unui RNN pentru a vedea cum pe termen lung acesta nu face față.
- Încercarea unor modele pre-antrenate pe secvențe

Mulțumim pentru atenție!