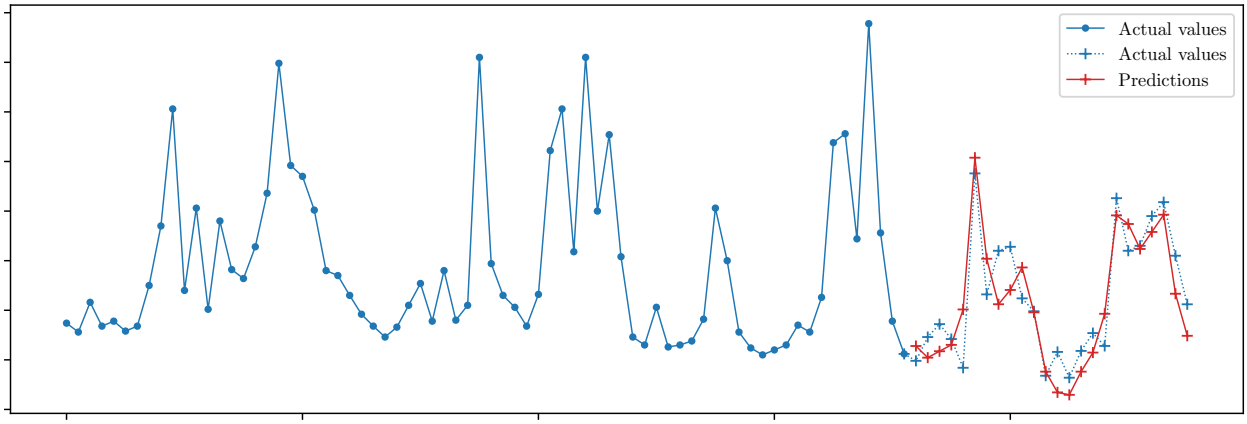
Prognos av ett dygns elförbrukning i en villa med djupinlärning

Jonatan Haraldsson
jonatan.haraldsson@icloud.com



Prognos av elförbrukning under ett dygn givet tre dygns tidigare data med träningsdata

Juni 2024
Kurs: *Deep Learning - metoder och tillämpningar, 7,5 hp*
Kurskod: *5TF078*

Umeå Universitet

1 Introduktion

I en värld med allt kraftfullare datorer har djupinlärning — maskininlärning med djupa neurala nätverk (deep learning) — blivit ett mer lättillgängligt verktyg för att kunna göra träffsäkra prognoser inom områden som meteorologi, finans och klimatvetenskap. På grund av elektrifiering av industri och transporter förväntas världens energibehov öka och en större efterfrågan kan leda till högre elpriser. Således kan det vara viktigt och intressant för hushåll att kunna förutspå sin egen elproduktion över dygnets timmar och exempelvis matcha det med spotpriset på energi. Historiskt sett har förutsägelser av framtida elförbrukning dessutom varit svåra med traditionella metoder [1], vilket gör att detta problem lämpar sig väl för djupinlärning.

I denna rapport presenteras metod och resultat, med tillhörande diskussion, då djupinlärning med olika typer av neurala nätverk och parametrar användes för att förutspå ett dygns elförbrukning i en villa givet tre dygns indata.

2 Teori

Arkitekturen för ett neural nätverk kan skilja sig mycket beroende på vilket problem nätverket avser att lösa, men för att förutspå en framtida serie av värden är återkopplade neurala nätverk (recurrent neural networks, RNN) vanligt förekommande. För att träna återkopplade nät krävs att data delas in i sekvenser bestående av indata, X, och målvärden, y, enligt tabell 1. Indata här består av värdena a_0 till a_{n-1} och måldata består av a_n.

Tabell 1: Sekvenser eller fönstring av indata X och måldata y.

Indata	**Måldata**
$X_1 = \begin{bmatrix} a_0 & a_1 & \dots & a_{n-1} \end{bmatrix}$	$y_1 = \begin{bmatrix} a_n \end{bmatrix}$
$X_2 = \begin{bmatrix} a_n & a_{n+1} & \dots & a_{2n-1} \end{bmatrix}$	$y_2 = \begin{bmatrix} a_{2n} \end{bmatrix}$

En viktig egenskap hos de återkopplade näten är att de låter utvärdet från tidigare sekvenser av data ”hänga med” till nästa sekvens och modellen får således ett minne. I maskininlärningsbiblioteket `Keras` i `Python` finns bland annat lagren Long Short–Term Memory (`LSTM`), Gated Recurrent Unit (`GRU`) och `Simple RNN`, vilka alla är återkopplade lager.

Ytterligare en viktig komponent vid djupinlärning är en modells förlustfunktion, loss function, vilken modellen ser till att minimera under träningsförfarandet. Några vanliga förlustfunktioner är medelvärdet av det absoluta felet, MAE, vilket ges av

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i| \tag{1}$$

och medelvärdet av det kvadrerade felet, MSE, vilket ges av

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2; \tag{2}$$

där y_i är det faktiska värdet och $\hat{y}_i$ är modellens predikterade värde. Eftersom felet $e = (\hat{y}_i - y_i)$ för MSE växer enligt $\mathcal{O}(n^2)$ kommer stora fel värderas högre, vilket gör att MSE

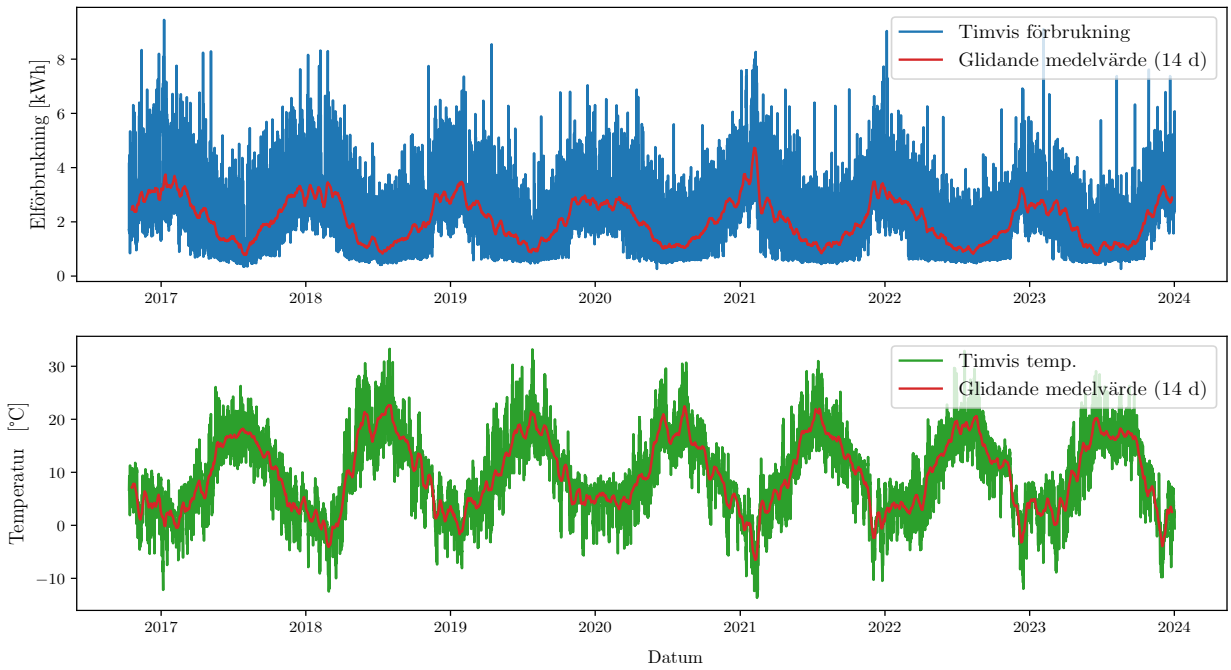
lämpar sig väl för data med mycket extremvärden. För MAE värderas i stället felen linjärt och stora och små fel värderas mer likt. En annan förlustfunktion som kombinerar MAE och MSE är Huber, vilken ges av

$$\text{Huber} = \begin{cases} \frac{1}{2}(\hat{y}_i - y_i)^2, \text{ då } |\hat{y}_i - y_i| \leq \delta \\ \delta \cdot (|\hat{y}_i - y_i| - \delta/2), \text{ då } |\hat{y}_i - y_i| > \delta, \end{cases} \tag{3}$$

där δ är en godtycklig parameter. Sammantaget ger Huber ett beteende likt MSE vid låga fel och likt MAE vid större fel.

3 Metod

Inledningsvis skaffades data för elförbrukning varje timme i en villa* sedan 2016-10-12, vilket sammantaget gav c:a 63 300 datapunkter. För att ge nätverken mer data laddades timvärden för utomhustemperaturen ner från SMHI:s mätstation *Göteborg A* [2] och dagslängden i timmar för varje dag beräknades enligt [3]. Då elförbrukning har dygnperiodicitet multiplicerades dagslängden med $\sin\left(\frac{\pi}{12}h\right)$, där h är timmar. För vissa timmar i dataserien för utomhustemperatur saknades mätvärden och för att få fullständig data sattes de värdena till medelvärdet av temperaturen vid samma tid föregående samt nästkommande dygn. Likadant gjordes för dataserien med elförbrukningen fast där förbrukningen var 0 kWh, alltså då det var strömavbrott. I figur 2 visas data för elförbrukning och temperatur, vilka användes för att träna nätverken, och det följer att elförbrukning och temperatur helt klart samvarierar öer årstiderna.



Figur 2: Data över elförbrukning och temperatur från 2016-10-12 till 2024-01-01 i en svensk villa. Därtill beräknades ett glidande medelvärde på 14 dygn.

Jag hade från början tänkt att använda min egen elförbrukning, men mina föräldrar hade data som sträckte sig nästan fyra år längre bakåt och inom ML känns det som att "kvantitet ⇒ kvalitet".

Innan träning av nätverken inleddes normerades all data, X, enligt

$$X_{\text{norm}} = (X - \mu_X)/\sigma_X,$$

där μ_X är medelvärde och σ_X är standardavvikelse. Vidare delades data upp i 70 % tränings-, 20 % validerings- och 10 % testdata och sedan genererades sekvenser enligt de som beskrivs i tabell 1. I samtliga försök sattes indatasekvensen till 72 h:s data och utdatasekvensen, alltså antalet timmar som önskas förutspås, till 24 h. Därefter författades lite olika modeller enligt följande mall:

```
exemple_model = keras.models.Sequential([

    '''Different types of layers with different units'''

    keras.layers.Dense(units=hours_to_predict) # End layer with 24 units same as
    ↪  number of hours to predict
])
```

För hitta en optimal modell för att förutspå framtida elförbrukning gjordes experiment med olika modeller och lager av typerna `LSTM`, `SimpleRNN`, `GRU` och slutligen inkluderades även ett faltnings lager `Conv1D`. En lista över samtliga modeller som testades återfinns i bilaga B. För de olika modellerna justerades hyperparametrarna (främst noder och regulariseringsinställningar) och det absoluta felet, körtiden, mm. noterades. Även modellernas träningskurvor följdes för att upptäcka eventuell över/underträning. Det maxiamala antalet epoker sattes godtyckligt till 25 och en `EarlyStopping` etablerades också för att se till att modellen inte körde för många epoker. För att kunna jämföra modellerna sinsemellan valdes mätvärdet (metric) MAE. Modellernas prestanda utvärderades också grafiskt, då prediktionerna och de faktiska värdena plottades i en figur. Ytterligare en jämförelse mellan prediktioner och faktiska värden för några utvalda modeller gjordes i ett histogram. I histogrammet plottades differensen mellan modellens prediktioner och de faktiska värdena e $= (\hat{y} - y)$ och därefter beräknades antalet datapunkter inom ett symmetriskt intervall från noll enligt $|e| \leq \varepsilon$, där ε sattes till ett godtyckligt värde.

Vad gäller förlustfunktionen prövades tre olika, MSE, MAE och Huber (ekvation 1, 2 och 3). På grund av att indata var tämligen spretig och att elförbrukningen skiljer sig mycket från timme till timme användes förlustfunktionen MSE till en början.

För den högst presterande modellen genomfördes även tre omgångar autotuning med `BayesianOptimizat` där inlärningstakt och noder varierades. Efter tuningen genomfördes ytterligare träning på modellen. Under de första två optimeringarna med tunern användes MSE som förlustfunktion och i det sista fallet användes Huber som förlustfunktion.

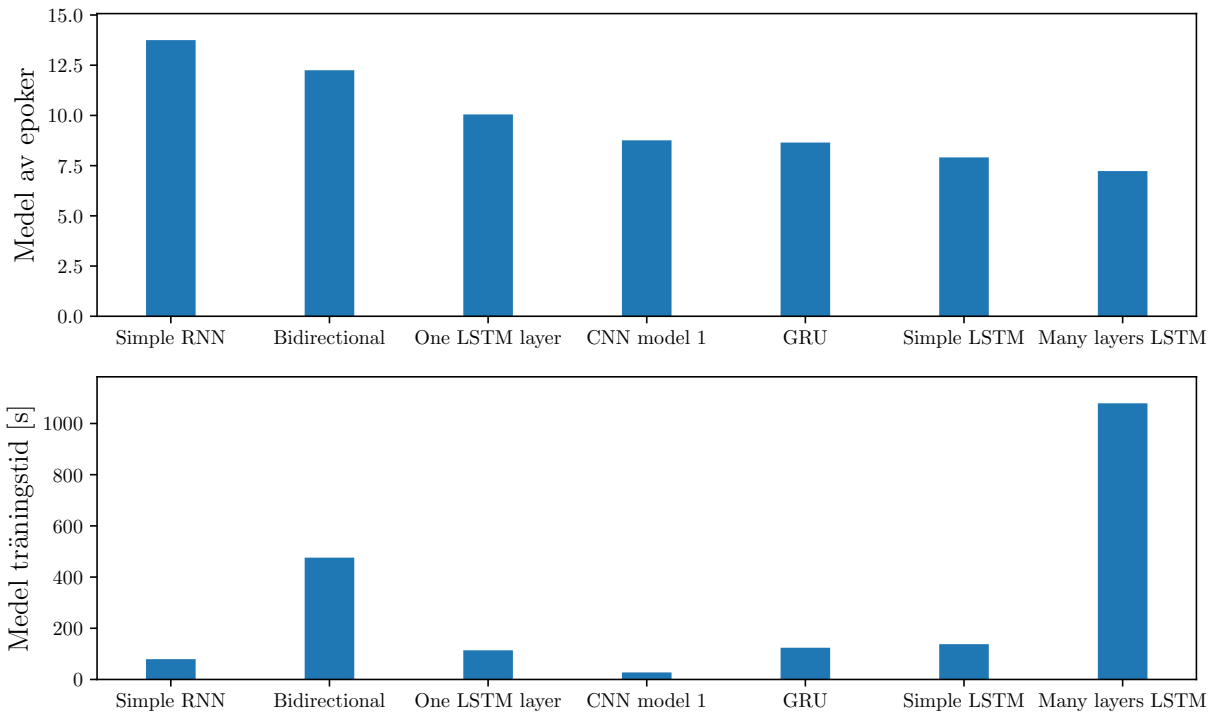
Jupyter-filen och all tillhörande data finns tillgängliga här: `Till Colab` och `Data energi`, `Data temperatur`. För vissa lösningar i koden togs inspiration från källor och forum på nätet. Dessa är refererade till i kod-filen.

4 Resultat för experiment och den bästa modellen

I följande avsnitt presenteras resultatet av experimenten och data från träningsförloppet och sedan presenteras den bästa modellen som experimenterats fram samt resultatet från tuningen.

4.1 Resultat från experiment

Sammantaget testades 46 olika uppsättningar parametrar för modellerna i bilaga B och en lista över resultatet i sin helhet återfinns i bilaga A. I figur 3 visas medelvärdet av antalet epoker varje modell körde samt medelvärdet över varje modells totala träningstid. Från diagrammet följer att *Many layers LSTM* hade avsevärt längst träningstid samt färst antal epoker. Kortast träningstid hade *CNN model 1* och *Simple RNN* körde i snitt flest epoker. Det lägsta absoluta felet för varje modell, vilket presenteras i tabell 2, visar på att *Many layers LSTM* bäst lyckades förutspå elförbrukning. I övrigt säger resultatet i tabell 2 att `LSTM`-modellerna generellt gav ett lägre MAE jämfört med övriga modeller. Vidare är hade det återkopplade nätet `Simple RNN` lägst prestanda.



Figur 3: Stapeldiagram över medelvärdet av antalet epoker och träningstid under träningsförloppet. Det följer att *Simple RNN* i snitt körde flest epoker och *Many layers LSTM* körde färst. I träningstid tog *Many layers LSTM* längst tid, medan *CNN model 1* tog kortast tid att träna.

Tabell 2: Lista över modellernas lägsta uppmätta absoluta fel. Det följer att *Many layers LSTM* presterade bäst.

Modell	**Absolut fel (MAE)**
Many layers LSTM	0,370
Simple LSTM	0,394
One LSTM layer	0,432
GRU	0,434
CNN model 1	0,434
Bidirectional	0,442
Simple RNN	0,451

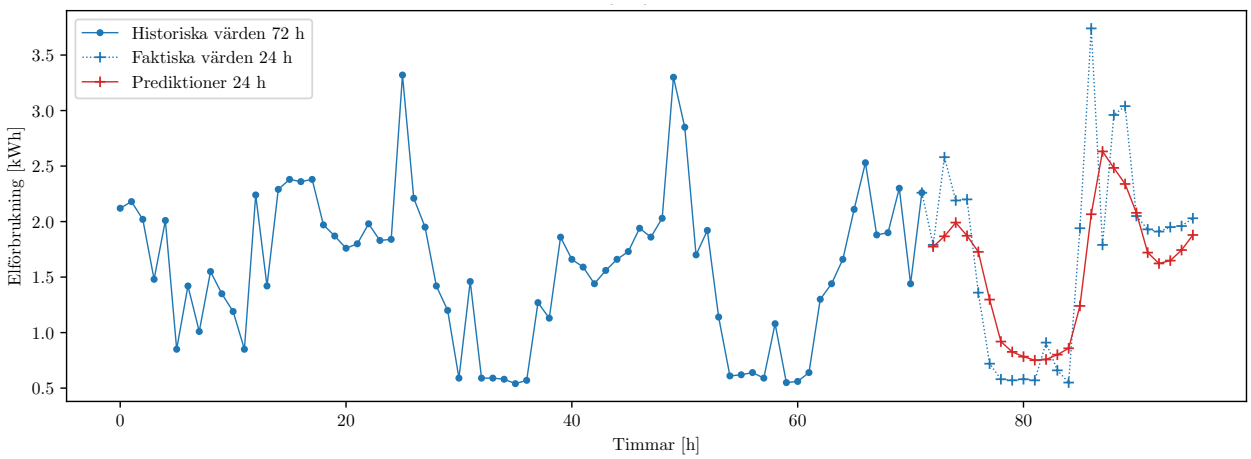
4.2 Resultat och prediktioner för den bästa modellen och tuner

Efter experimenterandet av olika modeller erhölls ett absolut fel 0,370 för modellen *Many layers LSTM*, vilken hade följande utseende:

```
lstm_model1 = keras.models.Sequential([
    keras.layers.LSTM(256, return_sequences=True,recurrent_regularizer=L2(0.1)),
    keras.layers.LSTM(64,  return_sequences=True),
    keras.layers.LSTM(32,  return_sequences=False),

    keras.layers.Dense(units=hours_to_predict*2, activation='relu'),
    keras.layers.Dense(units=hours_to_predict)
])
```

Det bästa MAE-värdet erhölls med inlärningstakten 0,005 och lossfunktionen Huber. Hur väl denna modell lyckades förutspå framtida värden från testdata visas i figur 4. Även om prediktionerna inte riktigt helt lyckas följa alla toppar och dalar i elförbrukningen, följer det från grafen att modellen tämligen väl lyckas följa trenderna i det kommande dygnets elproduktion.



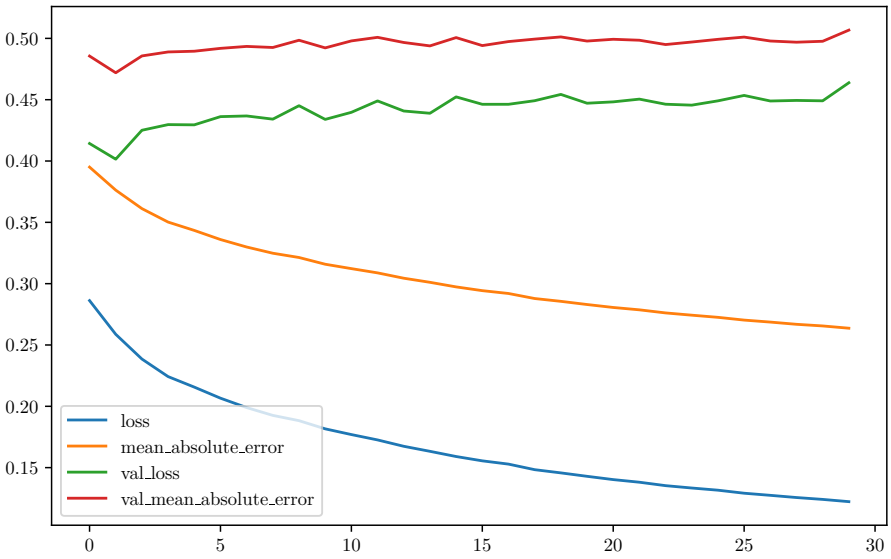
Figur 4: *Icke-tunad modell.* Elförbrukningsdata för 72 h, faktiska data följande 24 h samt modellens prediktioner följande 24 h. Från grafen följer att modellen tämligen väl lyckas förutspå framtida data.

På grund av långa körtider genomfördes tuning på en modell med ett `Dense`-lager och två `LSTM`-lager i stället för tre, som i den bästa modellen ovan. Det är ingen skillnad mellan tune 1 och tune 2 i hur tuner ställdes in, dock observeras ändå en liten skillnad i MAE. För tune 3 användes förlustfunktionen Huber i stället för MSE, vilket visade sig ge något lägre testförlust (test loss). De tunade modellernas parametrar presenteras i tabell 3 och träningsförloppet för en tunad modell presenteras i figur 5. Från träningskurvorna (figur 5) observeras att kurvorna för träningsdata avviker från valideringsdata bara efter några epoker och att MAE för valideringsdata tycks öka, vilket tyder på att modellen övertränar. Ytterligare ett bevis på att den tunade modellen övertränar kan observeras i figur 6, där spridningen för tune3-modellen och en icke-tunad modell jämförs. Från histogrammet följer att den icke-tunade modellens fel har mindre spridning, eftersom $P(|e| \leq \varepsilon)$ är större för båda felgränserna ε_1 och ε_2. Detta kan också verifieras med standardavvikelserna, vilka blev 0,71 och 0,53 för den tunade respektive icke-tunade modellen.

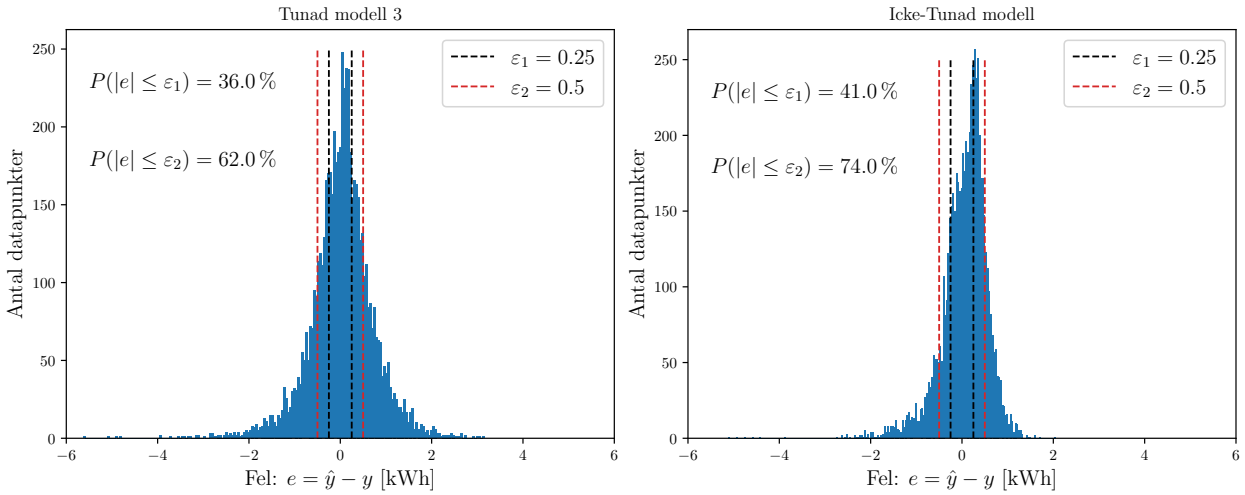
Hur väl den bästa modellen från tunern, tune 3, lyckades förutspå framtida data visas i figur 7 och även här observeras att prestandan hos den tunade modellen är något sämre än den i figur 4.

Tabell 3: Hyperparametrar och MAE för modeller efter tuning.

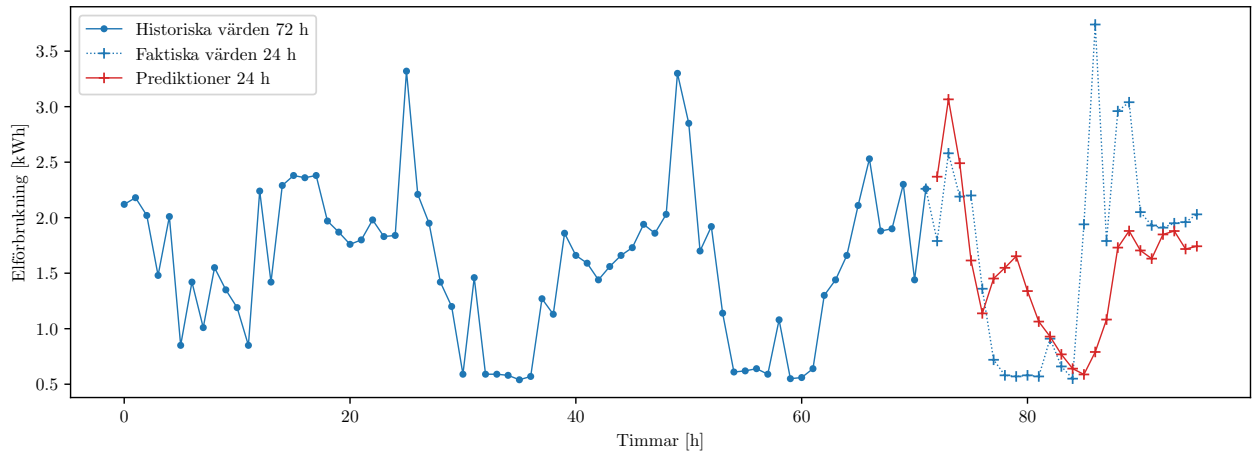
Hyperparameter	**Tune 1**	**Tune 2**	**Tune 3**
Noder i 1:a LSTM-lager	160	160	196
Noder i 2:a LSTM-lager	96	96	32
Noder i 1:a Dense-lager	72	72	48
Learning rate	0,001	0,001	0,001
Resultat	-	-	-
MAE test	0,468	0,471	0,442
Loss	0,410	0,422	0,165



Figur 5: Träningsförlopp för den tunade modellen. I figuren observeras att modellen är övertränad, eftersom validerings- och träningskurvorna inte följer varandra efter några få epoker.



Figur 6: Två histogram där felet, $e = (\hat{y} - y)$, mellan prediktion och faktiskt värde plottas för en tunad modell och för en icke-tunad modell samt sannolikheter att hitta data inom felintervallen ε_1 och ε_2. Från histogrammen följer att den icke-tunade modellen har en mindre spridning.



Figur 7: *Tunad modell*. Elförbrukningsdata för 72 h, faktiska data följande 24 h samt modellens prediktioner följande 24 h.

5 Diskussion och slutsatser

Att den bästa modellen från experimenten bestod av flera `LSTM`-lager efter varandra är inte oväntat, eftersom `LSTM`-modellen just är anpassad för tidsserier. Dock hade den modellen betydligt längst träningstider, vilket heller inte är helt oväntat, då modellen är mer komplex jämfört med de övriga. Trots sin långa körttid tenderade modellen att köra ganska få epoker, vilket kan tyda på att modellen snabbt konvergerar till ett minima. De sista epokerna innan träningen bröts var även MAE-värdena relativt konstanta.

Vad gäller lossfunktionen verkar det som att MAE ger bättre värden än MSE, något som är lite oväntat. Data för elförbrukningen, se figur 2, var tämligen spretig och från timme till timme kan den variera mycket, vilket tyder på att MSE borde givit modellen bättre

möjlighet att korrigera sig efter de större svängningarna. Att Huber gav bäst resultat kan förklaras av att den kombinerar MAE och MSE.

Vid vissa körningar av modellen *Many layers LSTM*, i synnerhet de med ett litet värde på inlärningstakt, erhölls i princip platta prediktionskurvor. Även om dessa hade ett relativt lågt MAE var modellen tämligen dålig på att förutspå värden. Härav följer att MAE inte ensamt ska användas för att bedömma huruvida en modell löst ett problem väl.

Om prediktionskurvorna för den bästa modellen i figur 4 och 7 jämförs kan slutsatsen dras att tuningen gör att prediktionerna sämre följer de faktiska värdena. En förklaring till detta är att den tunade modellen överträner, vilket bekräftas av träningskurvorna i 5 samt felets spridning i figur 6. En `EarlyStopping` användes under träningsförloppet efter tuning, men med facit i hand skulle en finare tolerans på `Patience` lagts till. Alternativt skulle en regularisering lagts in som ett lager i modellen och graden av regularisering skulle exempelvis kunnat lagts in som en parameter i tunern.

Samtlig programmering och körning av kod har skett lokalt på en *MacBook Pro M1 Pro* och på grund av stundtals långa körtider (upp till 3 h för tuning) genomfördes endast tre tuningkörningar. Det hade varit mer optimalt att ha tillgång till en GPU, vilken förmodligen hade snabbat upp testandet mycket. Exempelvis hade det varit intressant att sätta δ i huber (se ekvation 3) som en parameter att låta tunern variera. Vidare hade kortare körtider också öppnat upp möjligheten att testa fler och mer komplexa modeller med fler lager.

Ett exempel på sådana modeller, vilka dessutom framgångsrikt lyckats förutspå framtida elförbrukning upptill 500 h framåt, är modeller som kombinerar `CNN`- och `LSTM`-lager [4]. Att kombinera dessa två lagertyper för att förutspå framtida elförbrukning gjordes redan år 2019 och då erhölls, för sin tid, överlägsna resultat [5].

Sammanfattningsvis lyckades djupinlärning användas för att förutspå framtida elförbrukning med ett lägsta absolut fel på 0,37 och modellens prediktioner lyckas följa de faktiska värdena okej. Även om modellen träffar fel följer den ofta de trender som finns i elförbrukning, vilket får betraktas som bra. Den bästa modellen bestod av tre `LSTM`-lager och två `Dense`-lager, och dess parameterinställningar provades fram. Framöver rekommenderas att fler modeller testas, förslagsvis kombinationer av `CNN` och `LSTM` och även fler nätverk med det återkopplade nätet `GRU`. Om tuning genomförs bör regularisering e.d. användas för att motverka överträning hos modellen. Avslutningsvis ligger det något i att det är svårt att göra träffsäkra prediktioner av den snabbt varierande elförbrukningen, precis som det hävdas i [1].

Referenser

[1] M. Ahmad, "Seasonal decomposition of electricity consumption data", *Review of Integrative Business and Economics Research*, årg. 6, nr 4, s. 271, 2017. [Online]. Tillgänglig: https://www.sibresearch.org/uploads/3/4/0/9/34097180/riber_6-4_20b17-079_271-275.pdf.

[2] SMHI. "Ladda ner meteorologiska observationer, Göteborg A". (10 maj 2024), [Online]. Tillgänglig: https://www.smhi.se/data/meteorologi/ladda-ner-meteorologiska-observationer/#param=airtemperatureInstant,stations=core,stationid=71420 (hämtad 2024-05-26).

[3] A. Champneys. "The length of days 2", University of Bristol. (), [Online]. Tillgänglig: https://www.bristol.ac.uk/media-library/sites/engineering/engineering-mathematics/documents/modelling/teacher/daylength_t.pdf (hämtad 2024-05-12).

[4] "Predicting Household Electric Power Consumption Using Multi-step Time Series with Convolutional LSTM", *Big Data Research*, årg. 31, s. 100 360, 2023, ISSN: 2214-5796. DOI: https://doi.org/10.1016/j.bdr.2022.100360. [Online]. Tillgänglig: https://www.sciencedirect.com/science/article/pii/S2214579622000545.

[5] T.-Y. Kim och S.-B. Cho, "Predicting residential energy consumption using CNN-LSTM neural networks", *Energy*, årg. 182, s. 72–81, 2019, ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2019.05.230. [Online]. Tillgänglig: https://www.sciencedirect.com/science/article/pii/S0360544219311223.

Bilagor

A Utprövning av modeller resultat

Model	Metric	Metric Value	Loss	Loss Value	Time [s]	L. Rate	Epochs
Many layers LSTM	MAE	0.370	Huber	0.121	842	0.001	6
Many layers LSTM	MAE	0.373	Huber	0.125	817	0.005	6
Many layers LSTM	MAE	0.383	Huber	0.129	828	0.001	6
Many layers LSTM	MAE	0.383	MAE	0.389	834	0.005	6
Many layers LSTM	MAE	0.385	MAE	0.388	1220	0.001	6
Simple LSTM	MAE	0.394	MAE	0.394	101	0.001	6
Simple LSTM	MAE	0.397	Huber	0.137	106	0.001	6
Simple LSTM	MAE	0.399	MAE	0.399	169	0.005	10
Many layers LSTM	MAE	0.400	MSE	0.299	1040	0.005	7
Many layers LSTM	MAE	0.418	MSE	0.326	875	0.005	6
Simple LSTM	MAE	0.423	Huber	0.154	101	0.01	6
Simple LSTM	MAE	0.426	MSE	0.334	138	0.005	8
One LSTM layer	MAE	0.432	Huber	0.158	109	0.005	10
GRU	MAE	0.434	Huber	0.161	82.2	0.001	6
CNN model 1	MAE	0.434	MAE	0.434	23.0	0.001	9
GRU	MAE	0.434	MAE	0.434	98.8	0.005	7
Many layers LSTM	MAE	0.435	MSE	0.363	1130	0.005	7
GRU	MAE	0.437	Huber	0.162	82.1	0.005	6
CNN model 1	MAE	0.438	Huber	0.164	24.2	0.001	9
Simple LSTM	MAE	0.438	MSE	0.355	170	0.001	10
One LSTM layer	MAE	0.438	MAE	0.438	131	0.001	12
GRU	MAE	0.439	MAE	0.439	176	0.001	13
One LSTM layer	MAE	0.441	Huber	0.165	99.5	0.001	9
CNN model 1	MAE	0.441	Huber	0.165	20.6	0.0005	8
Many layers LSTM	MAE	0.442	MSE	0.366	999	0.001	7
Bidirectional	MAE	0.442	Huber	0.168	379	0.005	10
Bidirectional	MAE	0.445	MAE	0.445	576	0.001	15
One LSTM layer	MAE	0.446	MSE	0.367	103	0.005	9
Simple RNN	MAE	0.451	MAE	0.451	97.5	0.001	18
Simple LSTM	MAE	0.452	MSE	0.379	155	0.0005	9
Simple RNN	MAE	0.452	Huber	0.172	32.0	0.001	6
Bidirectional	MAE	0.452	MSE	0.383	994	0.001	25
GRU	MAE	0.455	MSE	0.386	164	0.001	11
CNN model 1	MAE	0.456	MAE	0.456	15.8	0.005	6
CNN model 1	MAE	0.457	Huber	0.175	27.1	0.005	11
Bidirectional	MAE	0.458	MSE	0.388	231	0.01	6
Simple RNN	MAE	0.461	MAE	0.461	69.7	0.005	13
Many layers LSTM	MAE	0.461	MSE	0.396	2230	0.0001	15
Simple RNN	MAE	0.461	MSE	0.396	118	0.001	20
Bidirectional	MAE	0.463	MSE	0.397	232	0.001	6
Simple RNN	MAE	0.465	MSE	0.403	76.6	0.001	14
Simple RNN	MAE	0.470	Huber	0.184	58.6	0.005	11
Bidirectional	MAE	0.485	MSE	0.434	422	0.1	11
CNN model 1	MAE	0.490	MSE	0.441	36.7	0.01	12
Many layers LSTM	MAE	0.813	MSE	1.22	1007	0.1	7
CNN model 1	MAE	0.813	MSE	1.06	17.3	0.1	6

B Modellerna

```
# First LSTM model. A simple LSTM model with one more Dense layer
lstm_model0 = keras.models.Sequential([

    keras.layers.LSTM(64, return_sequences=False),

    keras.layers.Dense(units=hours_to_predict*2, activation='relu'),
    keras.layers.Dense(units=hours_to_predict)
])

# Second model RNN
lstm_model1 = keras.models.Sequential([

    keras.layers.LSTM(256, return_sequences=True, recurrent_regularizer=L2(0.1)),
    keras.layers.LSTM(64, return_sequences=True),
    keras.layers.LSTM(32, return_sequences=False),

    keras.layers.Dense(units=hours_to_predict*2, activation='relu'),
    keras.layers.Dense(units=hours_to_predict)
])

# Thrid model BiDirectional LSTM
lstm_model2 = keras.models.Sequential([
    keras.layers.Bidirectional(keras.layers.LSTM(32,
    ↪  recurrent_regularizer=L2(0.01), return_sequences=True)),
    keras.layers.LSTM(32,recurrent_regularizer=L2(0.01)),
    keras.layers.Dense(units=hours_to_predict)
])

# Fourth model LSTM
lstm_model3 = keras.models.Sequential([

    keras.layers.LSTM(32, return_sequences=False),

    keras.layers.Dense(units=hours_to_predict)
])

# Simple RNN model
simple_RNN_1 = keras.models.Sequential([
    keras.layers.SimpleRNN(32,return_sequences=False),
    keras.layers.Dense(units=hours_to_predict)
])

# GRU model
GRU_model_1 = keras.models.Sequential([
    keras.layers.GRU(32),
    keras.layers.Dense(units=hours_to_predict)
])

# CNN model
CNN_model_1 = keras.models.Sequential([
    keras.layers.Conv1D(filters = 72, kernel_size=3, padding='same',
    ↪  activation='relu'),
    keras.layers.Flatten(),
    keras.layers.Dense(units=hours_to_predict)
])
```