Predictive Modelling og Machine Learning: Opgavesæt 5

2023-04-11

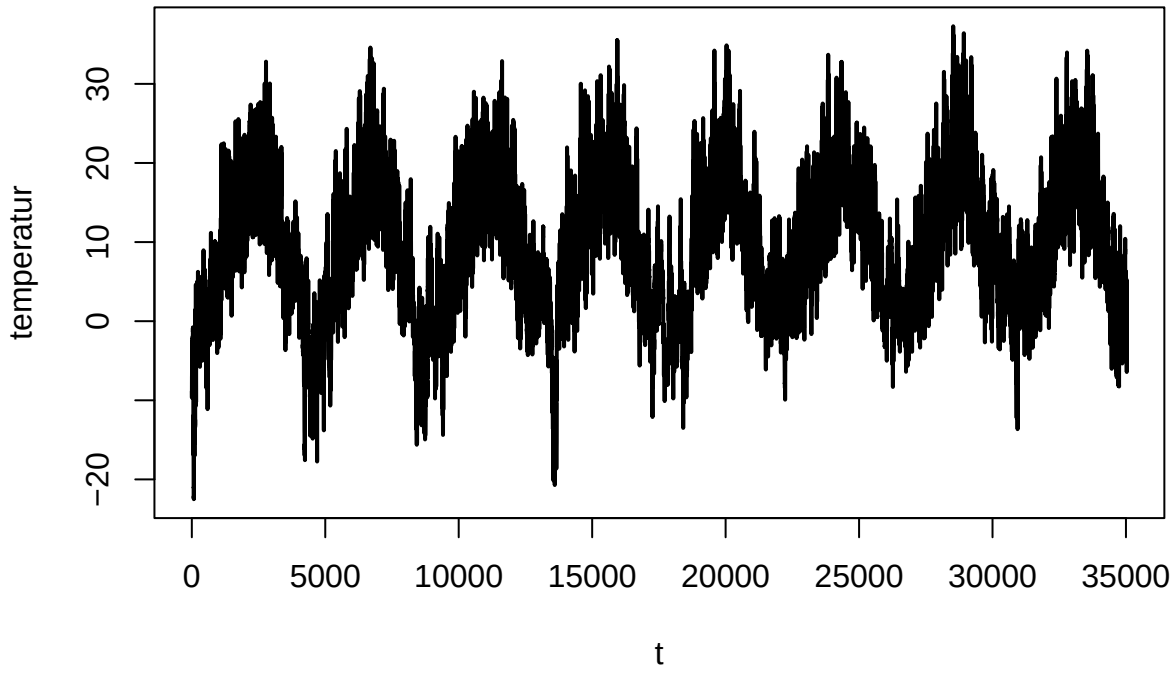
Opgave 1

Denne opgave tager udgangspunkt i datasættet vejrdata.txt, som indeholder vejrmålinger foretaget ved Max–Plack–Instituttet, der ligger i byen Jena i Tyskland. Der er målinger for hver anden time mellem 1. januar 2009 og 31. december 2016 – i alt målinger til 35037 forskellige tidspunkter. Til hvert tidspunkt er der målt:

- `lufttryk`: Luftrykket i mbar
- `temperatur`: Temperaturen i grader Celsius
- `luftfugtighed`: Luftfugtigheden i procent.

a)

Vi indlæsser datasættet i R. Og plotter kurven for temperaturudviklingen over tid.



b)

Jeg standardiserer `temperatur`-variablen.

c)

Jeg fitter et almindeligt rekurrent neural netværk frem til og med tidspunkt t_0 og sætter $T = 20$ (hukkomelse bagud, som bruges til information til at fitte på). Jeg benytter to hidden layers med hhv. 10 og 8 units i hver. Jeg benytter "tanh" som activation function i de to hidden layers. Jeg tilbage 10% af data til validering. Jeg inkluderer også drouput på 0.001

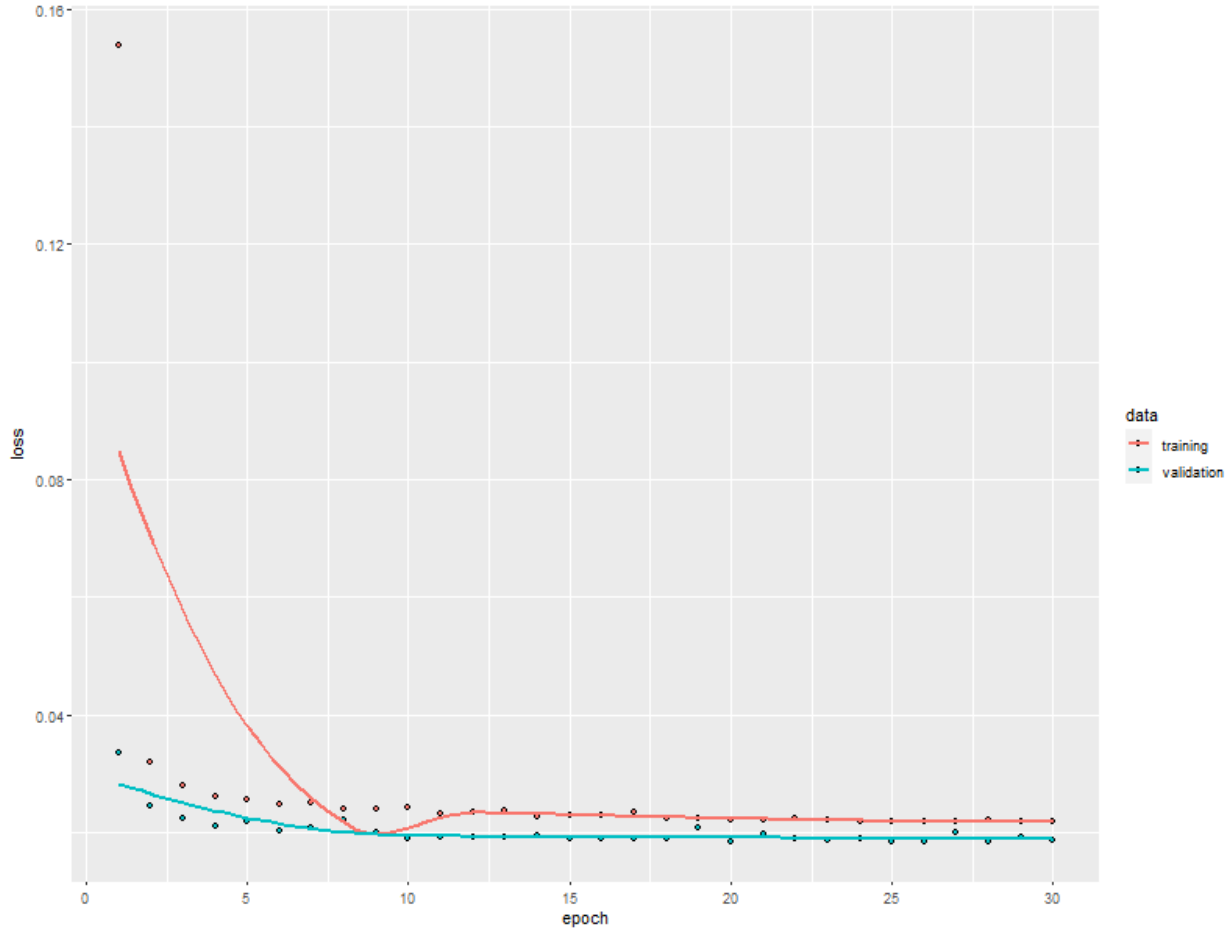


Figure 1: Træning og Val MSE af almindelig RNN

Det ser ikke ud til at være problematisk med fit af modellen, da val og train falder i takt med at epoch stiger.

d)

Jeg fitter modellen igen men uden valideringssplit, da de 10% valideringsdata er taget som de sidte 10% af information med i fittet, når værdier efter tid 32000 skal prædikteres.

```
## Model: "sequential"
## _________________________________________________________________________
```

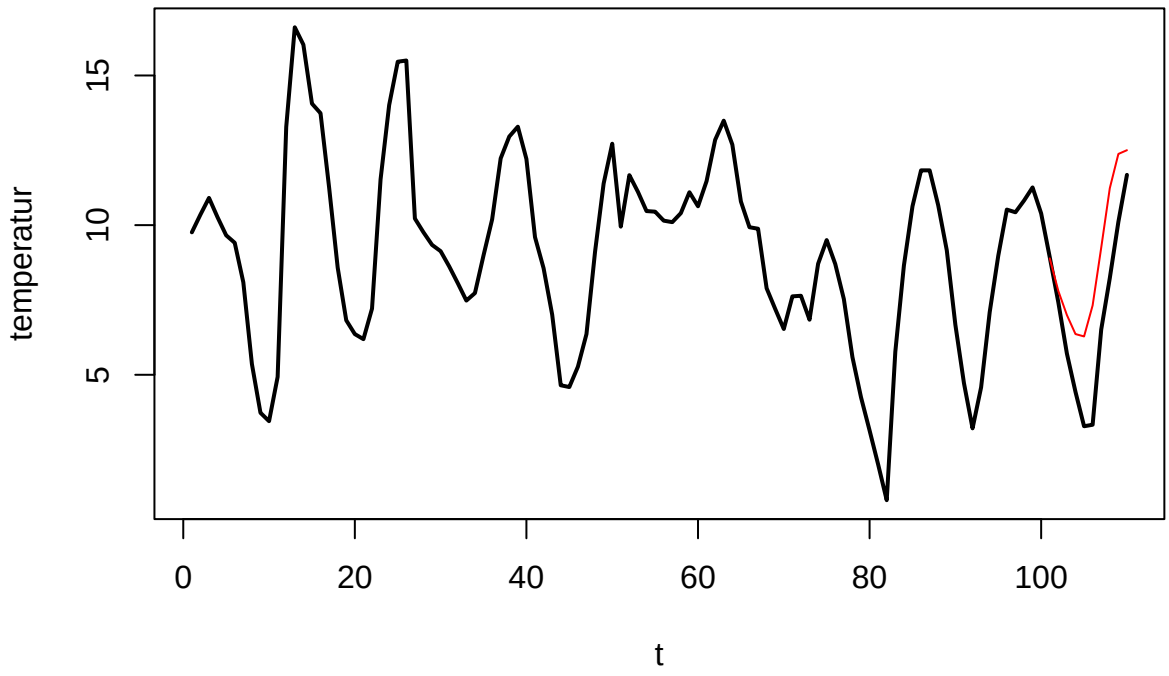
```
##  Layer (type)                 Output Shape              Param #
## =================================================================
##  simple_rnn_1 (SimpleRNN)     (None, 20, 10)            120
##  simple_rnn (SimpleRNN)       (None, 8)                 152
##  dense (Dense)                (None, 1)                 9
## =================================================================
## Total params: 281
## Trainable params: 281
## Non-trainable params: 0
## _________________________________________________________________

##
## Final epoch (plot to see history):
## loss: 0.02165
```

e)

Jeg bruger nu den fittede model til prædiktere variablen `temperatur` til tidspunkterne:

$$32001, 32002, \ldots, 32010$$

på baggrund af `temperatur` i tidsintervallet $31981, 31982, \ldots, 32010$




```
## MSE: 5.276894
```

Det virker umiddelbart til at modellen er rigtig god til at prædiktere temperaturerne. Hvilket heller ikke virker meget usædvanligt, da temperaturer jo pr. definition har regelmæssige trend ift. årstiderne.

f)

Jeg gentager nu hele analysen igen med med en LSTM-model. Jeg benytter samme T, hidden layers og units igen som før. Jeg bruger igen 30 epochs med batch size på 64.

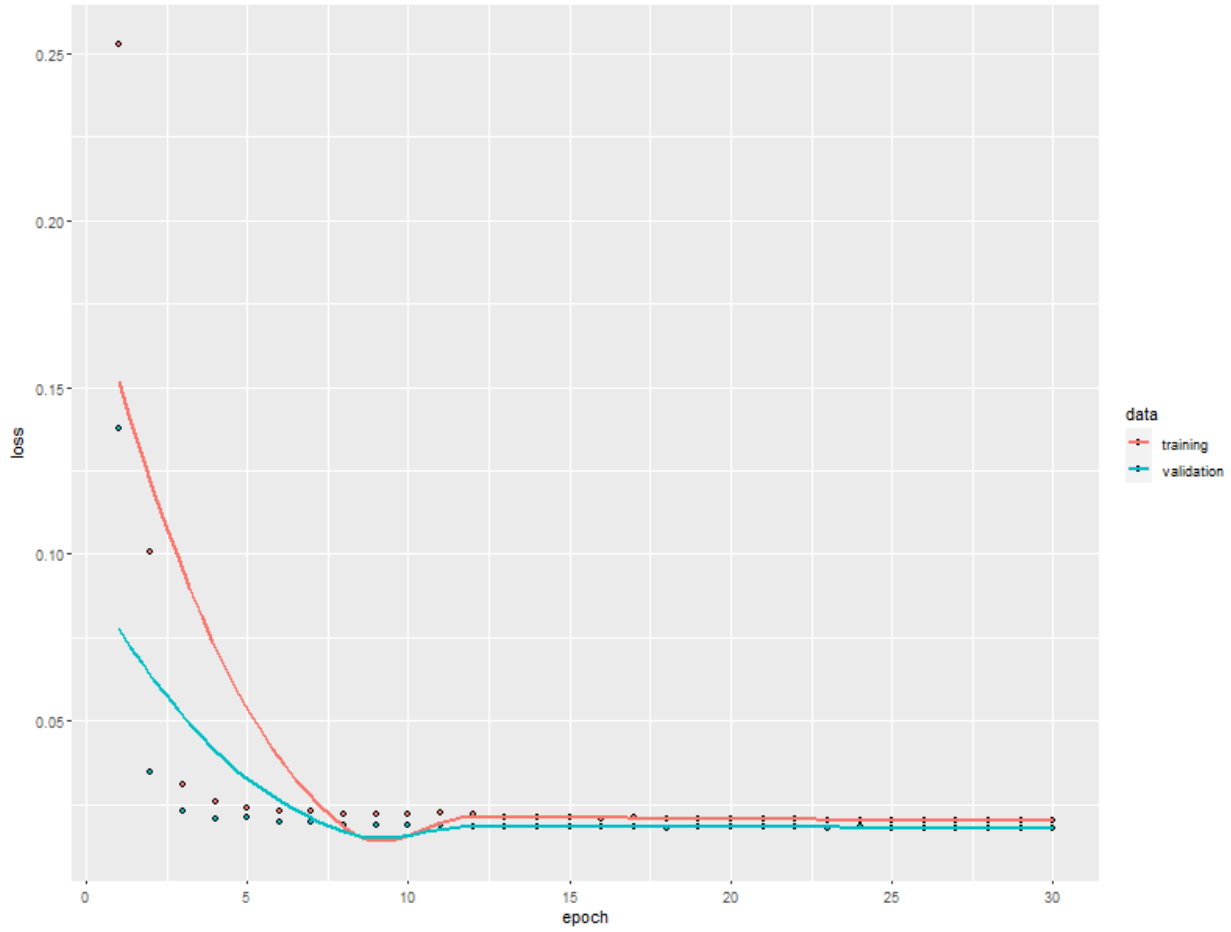
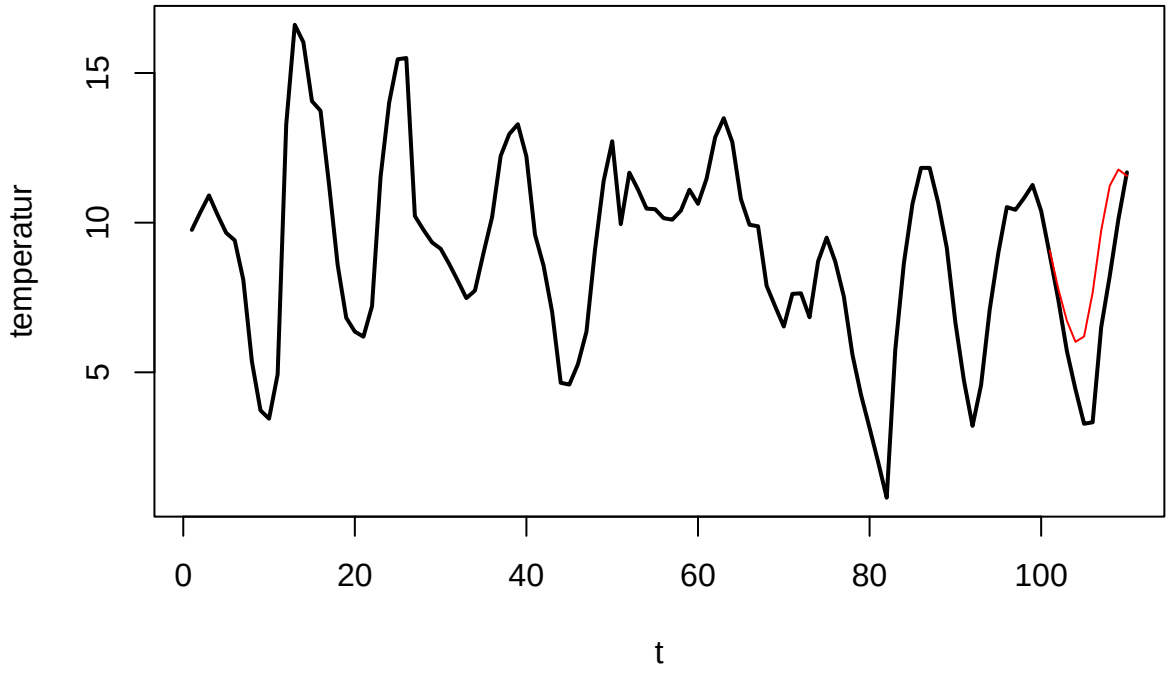


Figure 2: Træning og Val MSE af LSTM RNN

```
## Model: "sequential_1"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  lstm_1 (LSTM)                      (None, 20, 10)                  480
##  lstm (LSTM)                        (None, 8)                       608
##  dense_1 (Dense)                    (None, 1)                       9
## =========================================================================
## Total params: 1,097
## Trainable params: 1,097
## Non-trainable params: 0
## _________________________________________________________________________
```

```
##
## Final epoch (plot to see history):
## loss: 0.01942
```

Igen ser tænnings og validerings loss fint ud. Så jeg træner modellen igen uden validerings-split.




```
## MSE: 5.318852
```

Interessant nok virker det til at LSTM-modellen prædikterer dårlige med en MSE på ca. 4,19 kontra 1,48 ved den almindelige RNN. Dette kan bla. skyldes overfit, forkert hyperparametrer valg så som antal lag, units, epochs etc.

g)

Jf. summaries fra tidligere vides det at den simple RNN har 281 parametere og LSTM har 1097. Årsagen til, at LSTM-modellen har flere parametre end den simple RNN-model, skyldes forskellen i designet af de to modeller.

I den simple RNN-model bestemmes antallet af parametre af antallet af enheder i hver skjult lag og inputstørrelsen (længden af inputvektoren). I vores eksempel har den simple RNN-model 2 skjulte lag med henholdsvis 10 og 8 units og en inputlængde på 20. Dermed er det samlede antal parametre i den simple RNN-model:

$$(10 \times 20) + 10 + (8 \times 10) + 8 + 1 = 281$$

På den anden side har hver LSTM-celle i LSTM-modellen 4 sæt vægte: inputvægte, rekursive vægte, bias-værdier og cellestatvægte. Inputvægtene og de rekursive vægte ganges med henholdsvis inputdata og tidligere output, og de resulterende værdier lægges sammen med bias-værdierne for at producere cellens input.

Antallet af parametre i hver LSTM-celle bestemmes af antallet af enheder i cellen og størrelsen på input og output. I vores eksempel har LSTM-modellen 2 skjulte lag med henholdsvis 10 og 8 enheder og en inputvektorlængde på 20. Dermed er det samlede antal parametre i LSTM-modellen:

$$(4 \times (20 \times 10 \times 10)) + (4 \times (8 \times 10 \times 10)) + (10 \times 4) + (8 \times 4) + 1 = 1,097$$

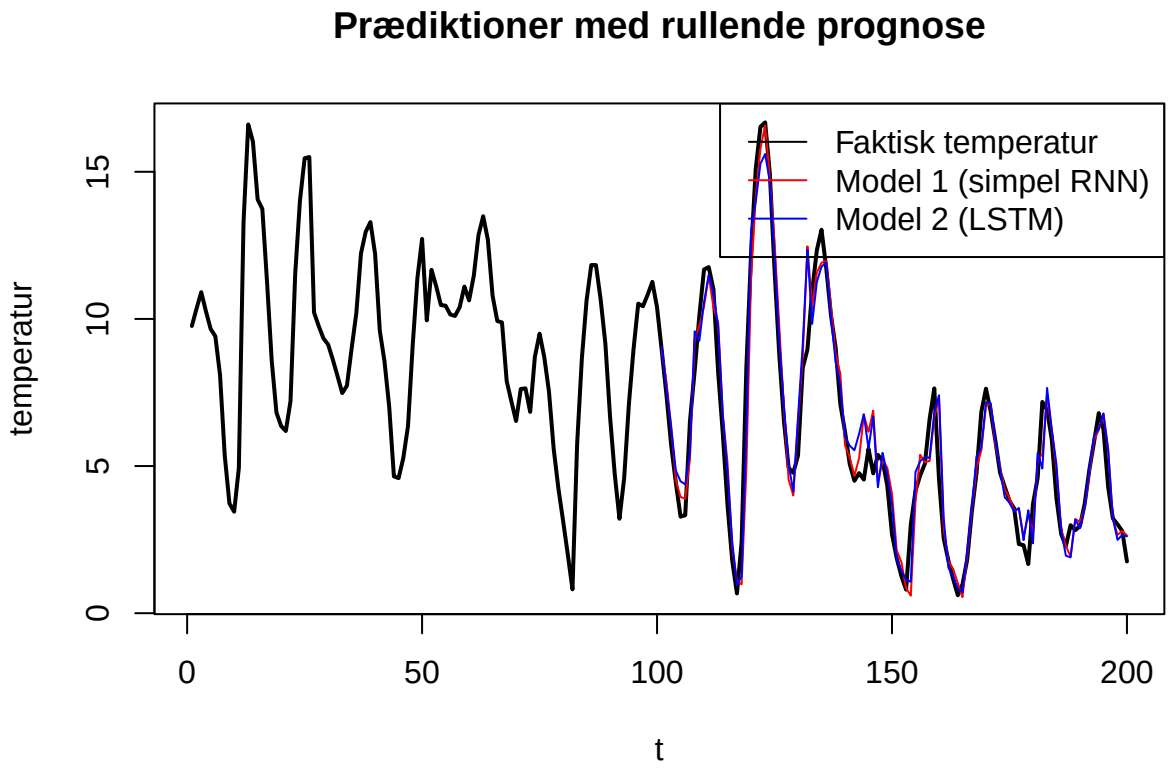
Derfor har LSTM-modellen flere parametre end den simple RNN-model, fordi hver LSTM-celle har flere sæt vægte og bias end den simple RNN-celle.

h)

Denne tilgang til prædiktionen er baseret på at bruge en rullende prognose, hvor de første T datapunkter bruges til at lave den første prædiktion, og derefter tilføjes et yderligere datapunkt til inputsekvensen, så der kan laves en ny prædiktion, osv. Det betyder, at prædiktionerne bliver lavet i realtid baseret på de mest aktuelle data.

Sammenlignet med den forrige prædiktionsmetode, hvor der blev lavet en prædiktion for en fast periode (10 eller 100 perioder) fremad, giver den rullende prognose en mere dynamisk prædiktion, der tager hensyn til den seneste udvikling i inputdataene. Dette kan gøre prædiktionerne mere nøjagtige og pålidelige.

I dette tilfælde bruges de seneste T datapunkter til at lave prædiktionen, og derefter tilføjes et nyt datapunkt fra den virkelige tidsserie for at lave den næste prædiktion. Dette betyder, at prædiktionen for hvert tidspunkt er baseret på en længere inputsekvens end tidligere prædiktioner, hvilket kan forbedre modellens evne til at fange komplekse sammenhænge i dataene. Sagt med andre ord har vi flere datapunkter i vores estimation.



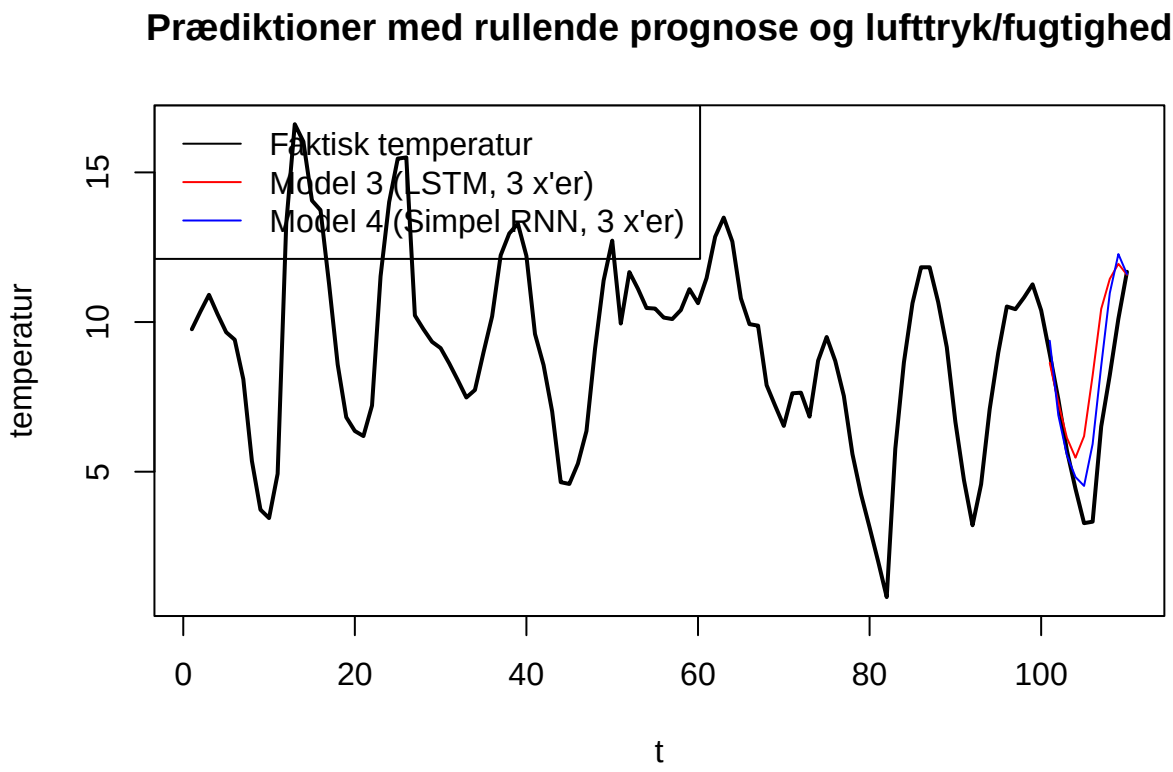
```
## Simpel MSE: 0.9655648
##  LSTM MSE: 0.9455636
```

Som det fremgår af plottet og MSE ovenfor prædikterer begge modeller markant bedre end før. Hvilket skyldes at der nu prædikteres på længere datafrekvenser end før. I den oprindelige tilgang blev prædiktionerne lavet ud fra de sidste T datapunkter uden at tage hensyn til tidligere data. Dette kan føre til mindre nøjagtige prædiktioner, især hvis der er stærke langsigtede afhængigheder i dataene. Ved at bruge en rullende tilgang kan modellen tage hensyn til disse langsigtede afhængigheder og generelt bedre fange de komplekse mønstre i dataene. Det er derfor, vi ser en forbedring i prædiktionerne ved brug af den rullende tilgang.

Ulemper ved den rullende metode er at det er mere beregningstungt. Samt kan det være svært at bedømme hvilken størrelse T skal have. Hvis T er for lille, kan modellen ikke fange de langsigtede afhængigheder i dataene, mens hvis T er for stor, kan det føre til overfitting af modellen. Endelig kan den rullende tilgang være mere følsom overfor outliers eller pludselige ændringer i dataene, da tidligere data også påvirker prædiktionerne. Hvis der sker pludselige ændringer i dataene, kan det tage tid for modellen at "justere" sig og lave nøjagtige prædiktioner igen.

i)

Jeg inkluderer nu også `lufttryk` og `luftfugtighed` som prædiktorer. Bemærk, at det nu kun giver mening at prædiktere med metoden fra spørgsmål (h):




```
## MSE_3: 6.277797
##  MSE_4: 2.521344
```

Det virker til at inkluderingen af `lufttryk` og `luftfugtighed` som prædiktorer gør modellerne bedre til at prædiktere end tilfældet med kun `temperatur` som prædiktorer. Tilføjelsen af lufttryk og luftfugtighed som ekstra lag i `Xarray` giver modellerne mulighed for at lære afhængighederne mellem temperatur, lufttryk og luftfugtighed, og på den måde prædiktere temperatur mere præcist. Lufttryk og luftfugtighed kan påvirke temperatur og derfor er deres tilstedeværelse i modellen vigtig for at tage højde for den effekt, de har på temperaturen.

Opgave 2

Convolutional neuralt netværk til klassifikation. I denne delopgave skal vi benytte et kunstigt fremstillet datasæt. Det vil sige, at du først skal køre den medfølgende kodestump, der ligger i filen opgavesat5.R. I koden laves i alt 3000 billeder, som hver har størrelsen 20×20 pixels. Alle billederne ligger i et 3–dimensionalt array kaldet *xdata*.

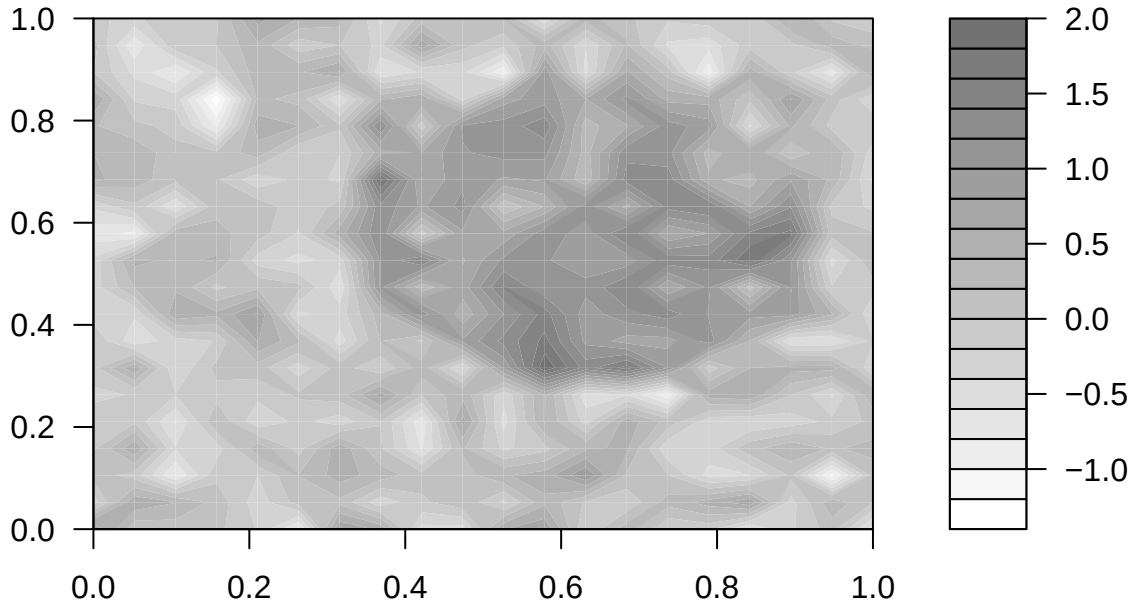
Der er tre typer af billeder:

- Rektangler – disse har fået label–værdien 0 i den tilhørende vektor *ydata*. Bemærk, at rektanglerne varierer i størrelse (både længde og bredde) og placering
- Cirkler – disse har fået label–værdien 1 i den tilhørende vektor *ydata*. Bemærk, at cirklerne varierer i både størrelse og placering
- Linjer – disse har fået label–værdien 2 i den tilhørende vektor *ydata*. Bemærk, at linjerne varierer i både hældning og skæring.

Bemærk, at der er tilsat støj til alle billederne i kraft af normalfordelte variable.

Vi indlæser data her:

Billederne kan inspiceres ved f.eks. at køre:



Her vises f.eks. en cirkel.

a)

Vi deler datasættet op i en træningsdel og testdel.

Så ligger datasættet opdelt i træning (2000 billeder) og test (1000 billeder) på samme måde, som datasættet ved forelæsningen.

b)

Vi standardisere observationsværdierne i billederne og omdanne y-variablene til 0-1-kategorier ved hjælp af keras-pakken i R.

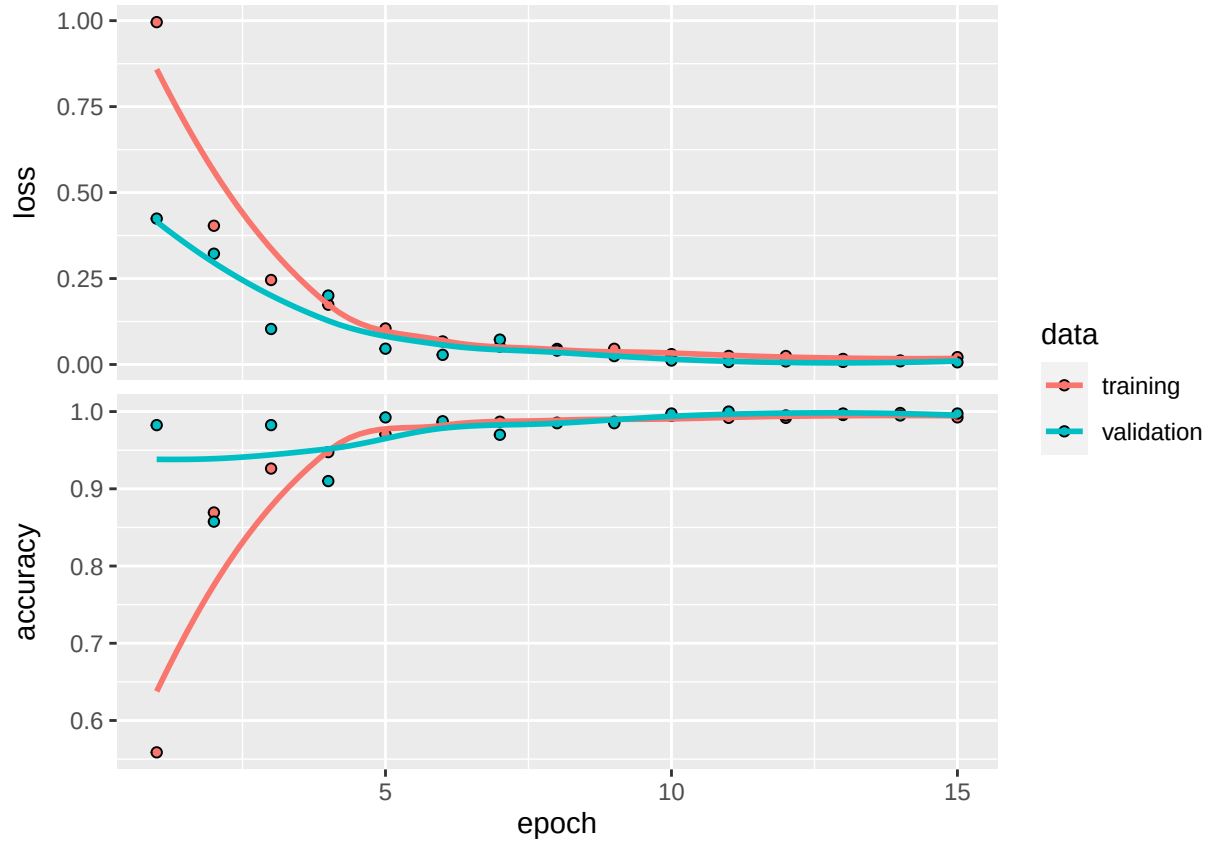
c)

Vi kan opbygge det convolutional neurale netværk ved hjælp af keras-pakken i R. Her bruger vi en arkitektur bestående af et convolutional lag med 32 filtre, et max-pooling lag, et almindeligt skjult neuralt netværkslag med 64 neuroner og et outputlag med 3 neuroner (en for hver klasse). Vi bruger også dropout-lag for at undgå overfitting:

d)

Vi kan bede om at få udskrevet et summary af modellen ved at køre `summary()`-funktionen på *model*-objektet:

```
## Model: "sequential_4"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  conv2d (Conv2D)                    (None, 18, 18, 32)              320
##  max_pooling2d (MaxPooling2D)       (None, 9, 9, 32)                0
##  flatten (Flatten)                  (None, 2592)                    0
##  dense_5 (Dense)                    (None, 64)                      165952
##  dropout (Dropout)                  (None, 64)                      0
##  dense_4 (Dense)                    (None, 3)                       195
## =========================================================================
## Total params: 166,467
## Trainable params: 166,467
## Non-trainable params: 0
## _________________________________________________________________________
```




```
##
## Final epoch (plot to see history):
##         loss: 0.02135
##     accuracy: 0.9925
##     val_loss: 0.006269
## val_accuracy: 0.9975
```

Vi kan se, at modellen har 166,467 trænbare parametre, og at inputtet forventes at have formen (None, 20, 20, 1) (hvor "None" står for batch-størrelsen). Som også er det forventede.

e)

Vi fitter netværket på testdatasættet og finder fejlraten, og finder ud af hvilke billeder den har sværest ved at identificere.

```
## Error rate: 0.004
```

```
## Number of wrong predictions: 4
```

```
##
##     0
##   1 4
```

Det viser sig, at det kun er cirklerne der er vanskelige at prædiktere. Selvom netværket umiddelbart performer rigtig godt. Højere standardafvigelse gør modellen dårlige i sin prædiktionsevne og vice versa (hvilket også er meget intuitivt, da det angiver hvor meget støj der er i data).