UMU_54907 DeepLearning

2019VT

Lab2 v0.0

Weather prediction

sygr0003 ==Sylvie Grafeuille

UMEÅ UNIVERSITET

Applied Physics and Electronics

Umeå universitet

Postadress: 901 87 Umeå

Content

Figures

1. Intro

1.1. About this document

This document is the second laboration rapport for the course UMU54907 deep learning,

tow subjects are presented:

1) training of several Artificial Neural networks for weather prediction.

2) Some reflections about Deep Learning

I wrote this document in order to not mix up code with comments in the notebooks and python code .

The notebooks have almost no comments and can be therefore more easily edited when testing the models

The document written in English except for the “uppgift” given by the teacher written in Swedish.

V 0.0 , 20190502

Intended audience.

The reader is expected to be a beginner in machine learning with some basic knowledge about python , jupyter , colab and machine learning .

Document overview

Part1:

Chapter 1 Intro , info about this document and the lab

Chapter 2 present the program for this lab and the implemented models

Chapter 3 presents loss curves related to the implemented models

Chapter 4 present some predictions made by the models

Chapter 5 Appendix : other related information

Part2:

Chapter 6 : Some reflections about Deep Learning

1.2. Uppgift Laboration 2

Labbspec, Deep Learning , Kalle Prorok

Som laboration 2 gör man ett eget litet projekt, helst med egna data som man analyserar med Deep Learning och dokumenterar hyggligt bra. Tänker också att man spelar in en 4-5 min filmsnutt där man beskriver projektet och resultaten så vi kan lägga upp/länka till dem för alla att lära och njuta av? (man kan t ex använda Windows 10 inbyggda screen recorder eller Jing).

Om projektet är hemligt/känsligt är det ok om man spelar in men bara visar det för läraren (som inte sprider det vidare), ev via icke-spridningsavtal (NDA).

Det får inte vara ett trivialprojekt av typen i lab 1 eller andra allmänt förekommande standardproblem (som filmklassificering, huspriser etc) och ska innehålla några lite mer avancerade former av Deep Learning.

Du ska även diskutera effekterna av Deep Learning på samhället, miljön och arbetsmarknaden.

betygsättning

Betygsättningen görs av läraren och baseras på följande lista med egenskaper som ger poäng och beroende på summan så får man betyg, +2 = G(3), +4 =

VG(4) och +6 = MVG(5). Viss reservation för korrigeringar och även andra trix kan ge poäng.

Egna data +1p,

Avancerat nät (LSTM/GRU/RNN) +1p,

Analys av hyperparametrarnas betydelse/värden +1p,

GAN +2p

TPU-användning +1p

Analys med TensorBoard +1p

Snygg visualisering +1p

Hantering av sekvenser +1p

Lite mer avancerad hantering av text +1p

Hantering av rörlig video (typ dataspel under spelande) +1p

Implementation i ett inbyggt system (mikrodator typ Raspberry Pi) +1p

Användning av funktionellt API i Keras +1p

Användandet av Deep Reinforcement Learning +2p

Motsvarande pluspoäng om man valt att köra Matlab istället.

Skrivit ihop en mycket bra dokumentation +1p

Extra ”proffsig” presentationsfilm (med uppsnabbade körexempel, framtidsvisioner etc) +1p

Väkommen med fler förslag och Lycka till!

1.3. About the subject and guidelines

The subject

The subject of this lab is weather forecasting problem introduced in the course book (Cholet chap6) and choosen as an an typical example of using RNN (Recurent Neural Networks)

The programs implemented in this lab are inspired by the weather prediction problem presented in the book from Cholet : 6.3-advanced-usage-of-recurrent-neural-networks.py

1.4. About the data_Set and the problem to solve

Short intro

In the weather forecasting problem, where we have access to a timeseries of data points coming from sensors installed on the roof of a building, such as temperature, air pressure, and humidity, which we use to predict what the temperature will be 24 hours after the last data point collected.

About the assumptions

The assumptions are that 10 days of recorded features collected once per hour can be used for temperature predictions on day ahead. If this assumption was true there would be no need of meteorologists

But is can be interesting anyway to see what you can predict with RNN models and if you can honestly conclude that RNN models give better results than the naive model that will be presented soon.

Features and predictions

14 different quantities (such air temperature, atmospheric pressure, humidity, wind direction, and so on) were recorded every 10 minutes, over several years (2009–2016)

The data to consider comes from a file called jena_climate_2009_2016.csv

The datafile count of 420,551 lines of data (each line is a timestep: a record of a date and 14 weather-related values),

You'll use it to build a model that takes as input some data from the recent past (a few days' worth of data points) and predicts the air temperature 24 hours in the future.

record each 10 mn <-> 6 records per hour <-> 24 record per day

The recorded features are:

["Date Time", "p (mbar)", "T (degC)", "Tpot (K)", "Tdew (degC)", "rh (%)", "VPmax (mbar)", "VPact (mbar)", "VPdef (mbar)", "sh (g/kg)", "H2OC (mmol/mol)", "rho (g/m**3)", "wv (m/s)", "max. wv (m/s)", "wd (deg)"]

Guidelines for the model implementations

This lab follows the tips from cholet 6.3 Going even further , things you could try in order to improve performance on our weather forecasting problem:

The Recommendations

Adjust the number of units in each recurrent layer in the stacked setup. Our current choices are largely arbitrary and thus likely suboptimal.

Adjust the learning rate used by our `RMSprop` optimizer.

Try using `LSTM` layers instead of `GRU` layers.

Try using a bigger densely-connected regressor on top of the recurrent layers, i.e. a bigger `Dense` layer or even a stack of `Dense` layers.

Don't forget to eventually run the best performing models (in terms of validation MAE) on the test set! Least you start developing architectures that are overfitting to the validation set.

As usual: deep learning is more an art than a science, and while we can provide guidelines as to what is likely to work or not work on a given problem, ultimately every problem is unique and you will have to try and evaluate different strategies empirically. There is currently no theory that will tell you in advance precisely what you should do to optimally solve a problem. You must try and iterate.

Run on TPU

As it can be seen in the appendix the computation time when running RNN on GPU is annoying big (2 minutes per step) if you want to run a serie of test. Therefore the training was mostly run on TPU but the models are written so they can also be run on GPU . More information about this is given in the rubric presenting the notebook: lab2SaveW

1.5. Environment for programming and test

Local host and colab

My localhost is a desktop without GPU running Ubuntu 18.04

The program were run both run on localhost and colab ,

software

The code use for this lab is written in python .

Keras and tensor flow as backend where used. The versions used in local host seems to be the same as the one in colab.

```
syl1>  python3  --version
Python 3.6.7
syl1>  jupyter --version
4.4.0
syl1>  python3 localW0.py
Using TensorFlow backend.
keras.__version__:  2.2.4
```

and in colab:

```
! python –version
! jupyter --version
import keras
print (keras.__version__)
Python 3.6.7
4.4.0
Using TensorFlow backend.
2.2.4
```

2. Notebooks and python files on gitHub

As Colab is in a virtual environment without easy procedure to read and write files I decided to put all information about this lab on Github as well as the notebooks meant to be run in colab.

2.1. The Repository

The repository in which the code files python and notebooks , the json files , h5 files and this pdf is kept is

https://github.com/SylGrafe/RepoDl04

2.2. Brief presentation of the programs

3 Notebooks and 3 python files imported by the notebooks in colab (or by other python files when running on local host) were implemented for this lab

The notebooks meant to be run in colab are

lab2SaveW run fit_generator on several models and save the history in a json file

and the weights in a h5 file

lab2LoadW load weights and run some predictions allows also to plot predicted temp vs real temp and nasive model predictions

lab2ReadJSON read json files containing several history and allow the user to print information about the run and plot the loss curves

the python files are:

lab2U00 contain code for data preparation and the data generators

lab2MakeModels contain the code to produces models

lab2JsonUtils contains the code to save the retrieve the information from the json file

I did also use python version of the notebook to run on local host because I found it more convinient these python files : loadW_lab2Loc.py , saveW_lab2Loc.py , readJson_lab2Loc.py

have been uses to produce the outputs presented here but are are not included in the repo because they are just copies of the notebooks already there.

2.3. python code

Once more I wanted to be able to share code between programs an wrote therefore

3 python pgm that can be imported either in colab or when running the code on local host

lab2MakeModels.py

About TPU

As the code written in lab2saveW is meant to be run on TPU , functional models must be used instead of sequential models so any model build in lab2MakeModels is a functional model , the first line of code to build it is always

```
source = Input(shape=(TSLen, nbOfFeat) ,    batch_size=batch_size,    dtype=tf.float32, name='Input')
```

About optimizer and loss

Following the example given in the course book the loss functions mae and the optimizer is RMS

about mae

the loss function for this lab is mae " mean absolut error" meaning the average absolute

difference between yi and xi.

about RMS

https://en.wikipedia.org/wiki/Stochastic_gradient_descent#RMSProp

RMSProp (for Root Mean Square Propagation) is also a method in which the learning rate is adapted for each of the parameters. … RMSProp has shown excellent adaptation of learning rate in different applications.

list of models

The list of the models that have been implemented are presented now. The models contains with GRU or LSTM and Dense layers , for a complete description see lab2MakeModels.py

```
########################### function makeGGDDModel ()
# 2 gru layers   with dropout=0.1 ,  and recurrent_dropout = 0.5  , 2 dense layers
def makeGGDDModel(TSLen, nbOfFeat,  batch_size=None , lrPar=0.001 , u1=32 , u2=64 ,
            d1=32,d2=64 ):


########################### function makeGGGDDModel ()
# 3 gru layers   with dropout=0.1 ,  and recurrent_dropout = 0.5 ,
#  2 dense layers  more than the dense layer for predicted_var
def makeGGGDDModel(TSLen, nbOfFeat,  batch_size=None , lrPar=0.001 , u1=32 ,
u2=64 ,u3=64 ,
            d1=32,d2=64 ):


########################### function  makeLLDDModel ()
# 2 lstm  layers  with dropout=0.1 ,  and recurrent_dropout = 0.5 ,
#  2 dense layers more than the dense layer for predicted_var
def makeLLDDModel(TSLen, nbOfFeat,  batch_size=None , lrPar=0.001 , u1=32 , u2=64 ,
            d1=32,d2=64 ):


########################### function makeGRUGRUModel ()
def makeGRUGRUModel(TSLen, nbOfFeat,  batch_size=None , lrPar=0.001 , u1=32, u2=64,
d1=1 ):
# 2 gru layers   with dropout=0.1 ,  and recurrent_dropout = 0.5  ,
#  max one extra dense layers  more than the dense layer for predicted_var


########################### function makeGRU2Model ()
```

```
def makeGRU2Model(TSLen, nbOfFeat,  batch_size=None , lrPar=0.001):
# 3 gru layers with dropout=0.1 ,  and recurrent_dropout = 0.5 ,
# the first gru has always  32 unit and one second  gru has always  64 units

########################## function makeGRU1Model ()
# 1 gru layer with 32 units
# This is the  first recurrent baseline from  the book
# 6.3-advanced-usage-of-recurrent-neural-networks.py
def makeGRU1Model(TSLen, nbOfFeat,  batch_size=None , lrPar=0.001):

########################## function makeDenseModel ()
# simple dense model one layers with 32 units
def makeDenseModel(TSLen, nbOfFeat,  batch_size=None , lrPar=0.001):

########################## function makeLSTMLSTMModel ()
def makeLSTMLSTMModel(TSLen, nbOfFeat,  batch_size=None , lrPar=0.001 , u1=32 , u2=64 ):
# 2 lstm layers  with dropout=0.1 ,  and recurrent_dropout = 0.5
```

Notice that lr and the nb of units are parameters to the unctions.

One Example makeGGDDModel()

The model most is use was ggdd and the definition is given now:

```
def makeGGDDModel(TSLen, nbOfFeat,  batch_size=None , lrPar=0.001 , u1=32 , u2=64 ,
                d1=32,d2=64 ):
   source = Input(shape=(TSLen, nbOfFeat),          batch_size=batch_size,
                dtype=tf.float32, name='Input')
   gru1 = GRU(u1, name='GRU1' ,         dropout=0.1,  recurrent_dropout=0.5,
            return_sequences=True )(source)
   gru2 = GRU(u2, name='GRU2',       dropout=0.1,  recurrent_dropout=0.5  )(gru1)
   dense1 = Dense(d1, name='Dense1')(gru2)
   dense2 = Dense(d2, name='Dense2')(dense1)
   predicted_var = Dense(1, name='Output')(dense2)
   model = tf.keras.Model(inputs=[source], outputs=[predicted_var])
   model.compile(         optimizer=tf.train.RMSPropOptimizer(learning_rate=lrPar),
      loss='mae' )
   retrun model
```

example of use

In this snippet you see how to create the model “ggdd” . Notice the parameters:

learning rate : myLr , nb of units for gru1: myUnits1 Nb of units for gru2: myUnits2 , for the dense layers 1 : myD1 and and for the dense layer2 :myD2

```
elif modelStruct == "ggdd":
   model=lab2MakeModels.makeGGDDModel (tSLen1 , nbOfFeat1, batch_size = inputBS,
      lrPar=myLr , u1=myUnits1 , u2=myUnits2  , d2=myD2 ,  d1=myD1)
   infoStr="u:%d:%d , d:%d,%d" %    (myUnits1 , myUnits2 , myD1 , myD2)
```

Lab2U00

Lab2U00 contains methods related to the data preparation and the generators

The init method (which must be called prio to any other methods) prepare the

Data , define the generators and also retrieve the TPU address (if any) so that the model can be run on TPU

About the generators

The aim of a generator is to produce batches of timesets and labels containing information extracted from the datafile (in this case jena_climate_2009_2016.csv.)

I will now try to use my own words to describe the how the generators in lab2U00 yield data

By reading the datafile line by line in init() we build an array that I call rawDataList in this document and float_data in the code . Lets call uRef one element of this array .

each uRef has nbOfFeaturePerUnit features (in this case nbOfFeaturePerUnit= 14) and was recorded each 10 mn in the file (So 6 uRef were recorded per hour)

The time serie (we will call previousUnitsList) related to uRef contains the data preceding (back in time) uRef , back to a certain time limit called lookback but also filtered so as only one unit is given per hour instead of 6 units.

Lookback is expressed in the code as a number of units in rawDataList
so the number of units in previousUnitsList is given by lookback/step

Numerical application :
keep 10 days of data so lookback =1440 and select one unit per hour so step=6
len (previousUnitsList) = lookback//step = 240
Conclusion there are 240 units in each timeserie

The aim of the model is to predict the tempAHead , the temperature timeAhead of time
of the unit uRef
Lets call uAhead the unit in rawDataList corresponding to the unit that is timeAHead
of uRef

in the reference code 6.3-advanced-usage-of-recurrent-neural-networks.py
timeAhead is refers as 'delay' and is expressed as nb of units (in rawDataList) between uRef
and uAHead

Numerical application: to calculate delay.
1 day ahead gives delay 24*6 = 144 units

shapes and values related to the generator

The generators will produces batches containing samples and targets were an element of a batch can be describe as tulpe (sample , target) with sample = previousUnitsList (notice that sample is a 2 dim array) and target = uHourAHead['temp'] (notice that target is a float)

So if you take into account the dimension of the batch you may say that samples are tensors with the shape (batch_size , len(previousUnitsList) , nbOfFeaturePerUnit)
targets are tensors of shape (batch_size,)

As wee need 3 sets of data (for training , validation and test) the generator have the parameters min_index and max_index to select part of the data for each purpose

delay is the parameter used to retrieve the targets temperatures at the right position in the array (one day ahead)

Numeric application

for lookback=1440 step=6 and batch_size=128 and delay = 144

((python code))

```
print ("float_data.shape" , float_data.shape)
# never forget to break out of a train_generator
for data_batch, labels_batch in train_gen:
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
    break
ind=1
ind1=0
print ("one unit in one previousUnitsList" , data_batch[ind][ind1])
print ("target for one previousUnitsList" , labels_batch[ind])
```

((output))

```
float_data.shape (420551, 14)
data batch shape: (128, 240, 14)
labels batch shape: (128,)
one unit in one previousUnitsList
[ 1.18904779  0.93111007  0.82916185  0.80680629 -0.73616464  0.83455888   0.75821408  0.66815138  0.72582276
0.72783246 -0.59202122 -0.18286601  0.66525457  0.94769548]
target for one previousUnitsList 0.401316172643444
```

abstract generator in lab2U00

A code snippet of the generator is given now

```
def generator(data, lookback, delay, min_index, max_index,      shuffle=False, batch_size=128, step=6):
    if max_index is None:
        max_index = len(data) - delay - 1
    i = min_index + lookback
    while 1:
        if shuffle:
            rows = np.random.randint(
                min_index + lookback, max_index, size=batch_size)
        else:
            if i + batch_size >= max_index:
                i = min_index + lookback
            rows = np.arange(i, min(i + batch_size, max_index))
```

```
        i += len(rows)
    samples = np.zeros((len(rows),     lookback // step,       data.shape[-1]))
    targets = np.zeros((len(rows),))
    for j, row in enumerate(rows):
        indices = range(rows[j] - lookback, rows[j], step)
        samples[j] = data[indices]
        targets[j] = data[rows[j] + delay][1]
    yield samples, targets
```

Some work the is presented now was necessary to understand this code

demoNumpy10.py

I wrote also demoNumpy10.py , a program similar to generator (in lab2U00) in order to illustrate which data are produced by generator () . The parameters are the same as for test_gen in lab2U00.py : batch_size:128 , min_index=300001 , delay=144 lookback=1440 step=6

The code

```
print (infoStr )
print ("batch_size:%d , min_index=%d , delay=%d lookback=%d step=%d \n" %
          (batch_size ,min_index ,  delay , lookback , step))
if True:
  i = min_index + lookback
  for batchNb in range (2):
      # print ("************ batchNb %d"  % (batchNb) )
      if shuffle:
          rows = np.random.randint(
              min_index + lookback, max_index, size=batch_size)
      else:
          if i + batch_size >= max_index:
              i = min_index + lookback
          rows = np.arange(i, min(i + batch_size, max_index))
          i += len(rows)
      targets = np.zeros((len(rows),))
      for j, row in enumerate(rows):
          indices = range(rows[j] - lookback, rows[j], step)
          if (j== 0 or j == len(rows) -1 ):
            print ("row %d ->indices [ %d" % (j , indices[0]) , end="")
            # print ("indices" , indices)
            for num in indices:
```

```
            #print (num , end=" , ")
            pass
          theTargetInd= rows[j] + delay
          # just print the last
          print ("... %d ] targetInd =%d" %( num ,  theTargetInd) )
          if j==0:
            print ("...")
      rawInd = rows[-1]
      calculatedRefInd= min_index + lookback + ( batchNb * batch_size)
      calculatedFirstTargetInd= calculatedRefInd + delay

      print ("batch %d rawInd=:%d  ,FirstTargetInd:%d   \n" %
      (batchNb , rawInd  , calculatedFirstTargetInd ) )
 print (" the nb of targets per batch is len(targets)= " , len(targets) )
 print ("for one timeStep in one batch len(indices)=" , len(indices))
```

output from demoNumpy10.py

```
syl1>  python3    demoNumpy10.py
demoNumpy10.py  print the data indices  and target indices generated by the generators
the refered generator is defined as generator () in the file  lab2U00.py

batch_size:128 , min_index=300001 , delay=144 lookback=1440 step=6

row 0 ->indices [ 300001... 301435 ] targetInd =301585
...
row 127 ->indices [ 300128... 301562 ] targetInd =301712
batch 0 rawInd=:301568  ,FirstTargetInd:301585

row 0 ->indices [ 300129... 301563 ] targetInd =301713
...
row 127 ->indices [ 300256... 301690 ] targetInd =301840
batch 1 rawInd=:301696  ,FirstTargetInd:301713

 the nb of targets per batch is len(targets)=  128
for one timeStep in one batch len(indices)= 240
```

conclusion about the generators

In general (**) the labels generated by each batch are kept in an array of length batch_size

This array contains the normalized temperatures found in

float_data[targetInd : targetInd+batch_size]

with targetInd= min_index + lookback + (batchNb * batch_size) + delay

(**) This concern batchNb when the condition

if i + batch_size >= max_index:

found in generator() is false

Example

the labels generated by the first batch in testGen () are the normalized temperatures found in

float_data[301457:301585]

Another think important to notice is that the timeseries are shuffled for the train_generator but not for the others

Retrieving dates

The generator generate batch of float_data without any references to the date at which the data was recorded because it is not relevant for the training. But I wanted to be able to relate predictions

to the date for this predictions and wrote some methods to do this.

It is possible the retrieve the date of the target temperature if you know the corresponding indices in float_data of the target temperature.

As explain earlier targetInd, the indice of the first target temp in a batch is given by

targetInd= min_index + lookback + (batchNb * batch_size) + delay

where batchNb is the counter at which to break out of the generator

Example of retrieving a given batch refered by batchBn is given here

```
batchNb =0
for rawInd in   getGenXYZ (theMinIndex=myMinIndex):
  batchCounter+=1
  if (batchCounter >= batchNb):
    break
```

List of some utility methods

Some methods related to the display of predicted or real temp or yesterday temp are:

def getGenXYZ (theMinIndex=300001):

define another generator to retrieve batches that do not always starts at min_index=300001

theMinIndex is a parameter to the generator

usefull for getting specific predictions

def getIndiceXYZ (min_index , batchNb):

retruns the indice of the first target yield by getGenXYZ () for batchNb

calculate real temperatures using std and mean

def realTemp (temp):

return theStd[1]*temp + theMean[1]

def getOneFloatData (rawInd):

return the floatData coresponding to the line rawInd in the cvs file

but remember that a only ONE of selInterval line is kept for the data generator

def readValuesFromFile (indArr , offset):

read some selected lines from the csv file

retruns the dates and temp extracted from theses lines

Using this methods it is possible the relate the predictions generated by predict_generator and the data generated from test_gen

lab2JsonUtils

Lab2JsonUtils.py contains the methods to save information about the models and the fit history in json file. It contain also methods to retrieve this information and to plot loss_curves.

This pgm is similar to lab1Utils.py (written for lab1)

The main difference with lab1Utils.py is

1)No acc , only loss val_loss and test_loss are to be plotted.

2) Each time a plot is shown dthe figure is saved under the name modelStruct_ timeStamp.png

for example: lstmlstm_1504_2013.png

about results.json

Reesults.json is a file containing of the results produced but training a given model implementation

To create results.json :

run lab2saveW several time changing model and or parameters between each run .

Each run produces a json file (and a h5 file for the weights) related to the notebook

If the pgm was run in colab the json file and h5 file were downloaded file on localhost

When all test were done most json files were then concatenated into a single file: results.json who was hen push back into the Repo

example of information dumped for one test

Here is given the kind of information dumped for one test in a json file

{"modelStruct": "ggdd", "compInfo": "rmsprop , lr=0.001 , mae", "histDict": {"loss": [0.3389, 0.3083, 0.2976, 0.2912, 0.2852, 0.2822, 0.2793, 0.2751, 0.2721, 0.2704, 0.2669, 0.265, 0.2643, 0.2625, 0.2609, 0.2593, 0.2587, 0.2572, 0.255, 0.2542, 0.2535, 0.2514, 0.2514, 0.2503, 0.2485, 0.2483, 0.2473, 0.2467, 0.246, 0.2444, 0.2439, 0.243, 0.2436, 0.2433, 0.2418, 0.2408, 0.2415, 0.2402, 0.2397, 0.2394, 0.2388, 0.2386, 0.2387, 0.2381, 0.2367, 0.2368, 0.2358, 0.2365, 0.2359, 0.2356], "val_loss": [0.2757, 0.2694, 0.2697, 0.2638, 0.2703, 0.2694, 0.2673, 0.269, 0.2716, 0.2655, 0.2761, 0.2712, 0.2726, 0.2736, 0.2764, 0.2738, 0.2736, 0.2718, 0.2753, 0.2763, 0.2888, 0.277, 0.2829, 0.2853, 0.2808, 0.2792, 0.2824, 0.2796, 0.2883, 0.2834, 0.2853, 0.2814, 0.2906, 0.2835, 0.2793, 0.2871, 0.2825, 0.2844, 0.2866, 0.2869, 0.2887, 0.2861, 0.2785, 0.2836, 0.287, 0.3012, 0.2828, 0.2826, 0.2823, 0.2832]}, "histParams": {"batch_size": null, "epochs": 50, "steps": 1500, "samples": 1500, "verbose": 1, "do_validation": true, "metrics": ["loss", "val_loss"]}, "timeStamp": "1804_1653", "info": "u:32:64 , d:64,128", "h5": "ggdd_1804_1653.h5", "testRes": 0.2521, "codeRef": "lab2SaveW onTPU"}

explanations

codeRef : the reference to the note book from which the results where produced

"codeRef": "lab2SaveW"

modelStruct a reference to a if block in the code related to the structure of the model

{"modelStruct": "ggdd"

timestamp : a time stamp at which the pgm was run

"timeStamp": "1804_1653"

info : some more nformation that the user wanted to dump

"info": "u:32:64 , d:64,128",

h5 name of the file containing the weights that were saved

"h5": "ggdd_1804_1653.h5",

compInfo compilering info inclusing the learning rate

"compInfo": "rmsprop , lr=0.001 , mae"

2.4. notebooks

Lab2M00 : Code from cholet

This notebook contains code and explanations directly copied from the course book

It includes the naive method (the temp tomorrow at this time of the day will be the same temp a now) . The mae from the naive model is the reference value that we hope to beat with the models implemented in this lab.

The naive method gives mae =0.29

```
def evaluate_naive_method():
    batch_maes = []
    for step in range(val_steps):
        samples, targets = next(val_gen)
        preds = samples[:, -1, 1]
        mae = np.mean(np.abs(preds - targets))
        batch_maes.append(mae)
    print(np.mean(batch_maes))
```

evaluate_naive_method()

```
.2897359729905486
```

Lab2SaveW

This notebooks was used to train models and save info in json file and weights in h5 files

running on TPU

Lab2saveW is meant to be run on TPU but may also be run on CPU or GPU.

To run on TPU function API must be used to build the mode and also the use of tf.contrib.tpu.keras_to_tpu_model

Notice that tensor flow warns that this procedure will not be valid on tensor 2.0

The code to modify the model is given here:

```
theModel = tf.contrib.tpu.keras_to_tpu_model(    model,
  strategy=tf.contrib.tpu.TPUDistributionStrategy(
     tf.contrib.cluster_resolver.TPUClusterResolver(TpuAddress)))
```

Lab2LoadW

Lab2LoadW is meant to read weights file , run evaluate and predict and plot some predictions

Results from a naive prediction using data directly extracted from the CVS file were also printed from this notebook.

Note that it is not possible to run this code on TPU(I did not understood why.) but it does not matter because the code does not require too much computation .

about loading weights

h5 files are read by the pgm lab2loadW and used to make predictions

'modelStruct' and number of units (u1 , u2 , d1 , d2) in each recurrent layer or Dense layers must fit the one that was used when the h5 file was created

Example , you want to use the weights for the model refered by the time stample 1204_1011 .

Using lab2readJson you can list the characteristics of all test and get time stample , model struct and number of units as shown here

```
23 1204_1011  ggdd  rmsprop  , lr=0.01 , mae u1:32:64 , d:64,128
```

this teaches you that the weights for the test with the time stample 1204_1011 were saved in the file ggdd_1204_1011.h5

Remeber that to load the weights from the file ggdd_1204_1011.h5 a model corresponding to ggdd rmsprop u1:32:64 , d:64,128 must be created first.

The plot

lab2eadJSON

The notebook ColabReadJson access theDumpFileName , the file containing dumped results.

Via a simple interface the user may choose and print :

a summary of all results

a more details presentation for any result

to plot the fit history for any result

by default theDumpFileName in colab is RepoDl04//results.json

Notice that by changing the parameter theDumpfilename in the notebook you may access another file created in the virtual environment

commands for user interface

In this extract from output for readJson pgm (run on local host) you get the list of user commands

to get this list of command press return when running lab2ReadJson

```
syl1> python3 readJson_lab2Loc.py res1504NEW.json
Will read dump file res1504NEW.json
indStr [moreStr] >
    explore the content of the dumpfile res1504NEW.json
    if indStr== s     summary : print list of all records  in the file
    if indStr== o     ordered summary , highest  min (val_loss)  first
    if indStr ==e     exit the Pgm
    if indStr== valid record indice       print part of the record
    if indStr==validindice and moreStr==a   print the record
    if indStr==validindice and moreStr==p   print part of the record and plot the history
```

example get ordered list of records

In this example run on local host the user ask to get an ordered list of results

extracted from a Json file : res1504NEW.json

Because the user used the command ‘o’ the list is ordered, the results with highest values for min (val_loss) are presented first.

```
syl1> python3 readJson_lab2Loc.py res1504NEW.json
will read dump file res1504NEW.json

indStr [moreStr] > o
 indice <codeRef> timeStample, modelStruct, info
min (val_loss) at i/nb epochs ,..... test_loss

 0 <1304_1333>, dense, rmsprop , lr=0.002 , mae no more info
 0.3033 at 5/25 ,..... test: 0.3251
 1 <1504_2013>, lstmlstm, rmsprop , lr=0.001 , mae u:32,64
 0.261 at 6/25 ,..... test: 0.2677
 2 <1504_1604>, grugru, rmsprop , lr=0.002 , mae u1:32, u2:64 , d1:64
 0.265 at 10/25 ,..... test: 0.2544
 3 <1504_1730>, lldd, rmsprop , lr=0.001 , mae u1:32:64 , d:64,128
 0.2632 at 7/25 ,..... test: 0.2491
 4 <1504_1705>, grugru, rmsprop , lr=0.001 , mae u1:32, u2:64 , d1:64
 0.2598 at 17/25 ,..... test: 0.2352
```

example: plot history

In this example run on local host , the user ask to see the history plot of the result with indice 1 in the preceding ordered list , the output seen in the terminal is:

```
indStr [moreStr] > 1 P
63_m0_colab onTPU, lstmlstm, rmsprop , lr=0.001 , mae, 1504_2013
u:32,64
test Results 0.2677
best val loss : at epochs 6 /25 value 0.261

history Params: {'batch_size': None, 'epochs': 25, 'steps': 500, 'samples': 500, 'verbose': 1,
'do_validation': True, 'metrics': ['loss', 'val_loss']}
```

and the corresponding plot is:

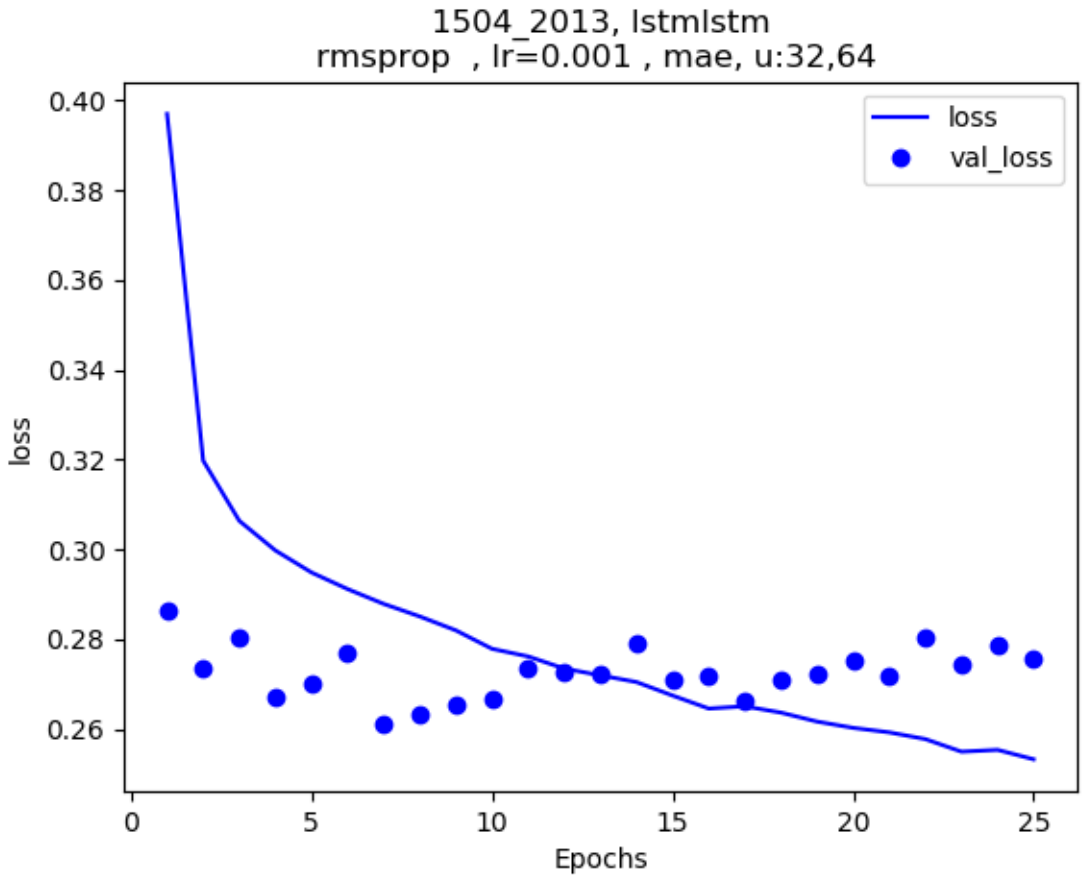


Illustration 1: example of plot lstmlstm_1504_2013

3. Results

3.1. About

Strange val_loss curves

By results I mean the presentation of the val_loss and loss curves representing the training

of the model.

The study of the validation curve should inform about how well the model is training
The loss curves is decreasing normally but I had difficulties to understand the val_loss curves

I do not understand why the validation loss is always much better than the loss at the first epochs but it was the same phenomena (unexplained) in the book

I wonder why are the val_loss fluctuating so much is it realted to noise in the data ?

About the runs

The model training were made to see the evolution of the val_loss not get "the best model" so I the training run more than the minimal number of epochs to get the best val_loss .

Try onceearly training stop

See test made with gggdd (3 gru and to dense) with 40 and 9 epochs

3.2. List of test done

A first list of the run made for this lab is given now , sorted by timestample :
This first list presents only the main information model name and main parameters used .
More details about the run made will be given later on.

As you can see, the guidelines presented at the beginning of this report have been implemented: testing several leraning rate and several models ans several nb of units for the layers.

```
syl1>  python3 lab2JsonUtils.py
..
0 1004_0826  D32 D1  rmsprop lr=0.001, mae GPU or CPU inputBatchSize 128
1 1004_0858  dense  rmsprop 0.001 , lr=, mae GPU or CPU inputBatchSize 128
2 1004_0913  gru1  rmsprop  , lr=0.001 , mae GPU or CPU inputBatchSize 128
3 1004_1109  gru1  rmsprop  , lr=0.001 , mae  TPU , inputBatchSize 128
4 1004_1122  dense  rmsprop  , lr=0.001 , mae  TPU , inputBatchSize 128
5 1004_1139  gru2  rmsprop  , lr=0.001 , mae  TPU , inputBatchSize 128
```

```
6 1004_1238  gru2  rmsprop  , lr=0.001 , mae no info yet
7 1004_1317  gru2  rmsprop  , lr=0.001 , mae no info yet
8 1004_1404  grugru  rmsprop  , lr=0.001 , mae u1:64, u2:64
9 1004_1454  lstmlstm  rmsprop  , lr=0.001 , mae u1:32, u2:64
10 1004_1533  lstmlstm  rmsprop  , lr=0.001 , mae u1:64, u2:64
11 1004_1645  lstmlstm  rmsprop  , lr=0.001 , mae u1:64, u2:128
12 1004_1717  grugru  rmsprop  , lr=0.001 , mae u1:64, u2:128 , d1:12
13 1004_1758  grugru  rmsprop  , lr=0.001 , mae u1:32, u2:64 , d1:12
14 1004_1825  grugru  rmsprop  , lr=0.001 , mae u1:6, u2:12 , d1:1
15 1004_1858  grugru  rmsprop  , lr=0.001 , mae u1:12, u2:24 , d1:1
16 1004_1959  grugru  rmsprop  , lr=0.001 , mae u1:32, u2:32 , d1:1
17 1104_1612  grugru  rmsprop  , lr=0.001 , mae u1:32, u2:64 , d1:12
18 1104_1748  grugru  rmsprop  , lr=0.001 , mae u1:32, u2:64 , d1:24
19 1104_1833  grugru  rmsprop  , lr=0.001 , mae u1:32, u2:64 , d1:128
20 1204_0803  ggdd  rmsprop  , lr=0.001 , mae u1:32:64 , d:32,64
21 1204_0858  ggdd  rmsprop  , lr=0.001 , mae u1:32:64 , d:64,128
22 1204_0937  ggdd  rmsprop  , lr=0.0001 , mae u1:32:64 , d:64,128
23 1204_1011  ggdd  rmsprop  , lr=0.01 , mae u1:32:64 , d:64,128
24 1204_1058  ggdd  rmsprop  , lr=0.002 , mae u1:32:64 , d:64,128
25 1204_1153  ggdd  rmsprop  , lr=0.0015 , mae u1:32:64 , d:64,128
26 1204_1233  lldd  rmsprop  , lr=0.0015 , mae u1:32:64 , d:64,128
27 1204_1314  lldd  rmsprop  , lr=0.001 , mae u1:32:64 , d:64,128
28 1204_1346  lldd  rmsprop  , lr=0.002 , mae u1:32:64 , d:64,128
29 1304_1333  dense  rmsprop  , lr=0.002 , mae no more info
30 1504_1604  grugru  rmsprop  , lr=0.002 , mae u1:32, u2:64 , d1:64
31 1504_1705  grugru  rmsprop  , lr=0.001 , mae u1:32, u2:64 , d1:64
32 1504_1730  lldd  rmsprop  , lr=0.001 , mae u1:32:64 , d:64,128
33 1504_2013  lstmlstm  rmsprop  , lr=0.001 , mae u:32,64
```

The corresponding weights files (not all test have been saved) are :

```
syl1>  ls *.h5
dense_1304_1333.h5  ggdd_1204_1153.h5    grugru_1504_1705.h5
ggdd_1204_0858.h5   grugru1004_2058.h5   lldd_1204_1233.h5
```

ggdd_1204_0937.h5 grugru_1104_1947.h5 lldd_1204_1314.h5
ggdd_1204_1002.h5 grugru_1104_2032.h5 lldd_1204_1346.h5
ggdd_1204_1011.h5 grugru_1104_2136.h5 lldd_1504_1730.h5
ggdd_1204_1058.h5 grugru_1504_1604.h5 lstmlstm_1504_2013.h5

(note that the name of the h5 is build from model_struct and timeStample)

3.3. How the results are presented

The results are presented ordered by the bast values of test_loss even if the model

has most of the time being trained too much and the validation_loss has already deteriorate.

The reason that the models are trained "too much" is that I wanted to see the evolution of the validation_loss.

Some models were anyway trained with (What I believe is) the optimal nb of epochs

and I did not notice that the test_loss got better!

The model used for theses test is refered by the model_struct which can easly be related to the model definition imlemented in lab2MakeModels.py

Example: lstmlst is related to makeLstmLstmModel()

3.4. Influence of the learning rate ?

I made some run with a given model changing learning rate between the run and it seems that lr=0.01 is the best but I'm not sure of this conclusion. Anyway I present some curves now

related to these test.

changing lr with dense model

Even if a fully connected model model_struc='dense' is not suppose to be a good model for time series I made some test with it changing the nb of epochs or the learning rate but the results are all similar.

As expected the test loss (mae around 0.32) is worst that the naive mae. (0.289)

Here is en extract out of the ordered list of results were the worst result hade indice 0

indice <codeRef> timeStample, modelStruct, info
min (val_loss) at i/nb epochs ,..... test_loss

1 <1004_1122>, dense, rmsprop , lr=0.001 , mae TPU , inputBatchSize 128

0.3039 at 5/15 ,..... test: 0.328

2 <1704_1817>, dense, rmsprop , lr=0.001 , mae no more info

0.3013 at 3/50 ,..... test: 0.3277

3 <1304_1333>, dense, rmsprop , lr=0.002 , mae no more info

0.3033 at 5/25 ,..... test: 0.3251

Once more the tre curves are quite similar and loss curve corresponding to lr=0.01 is presented now

Dense_ 1704_1817 test_loss=0.328

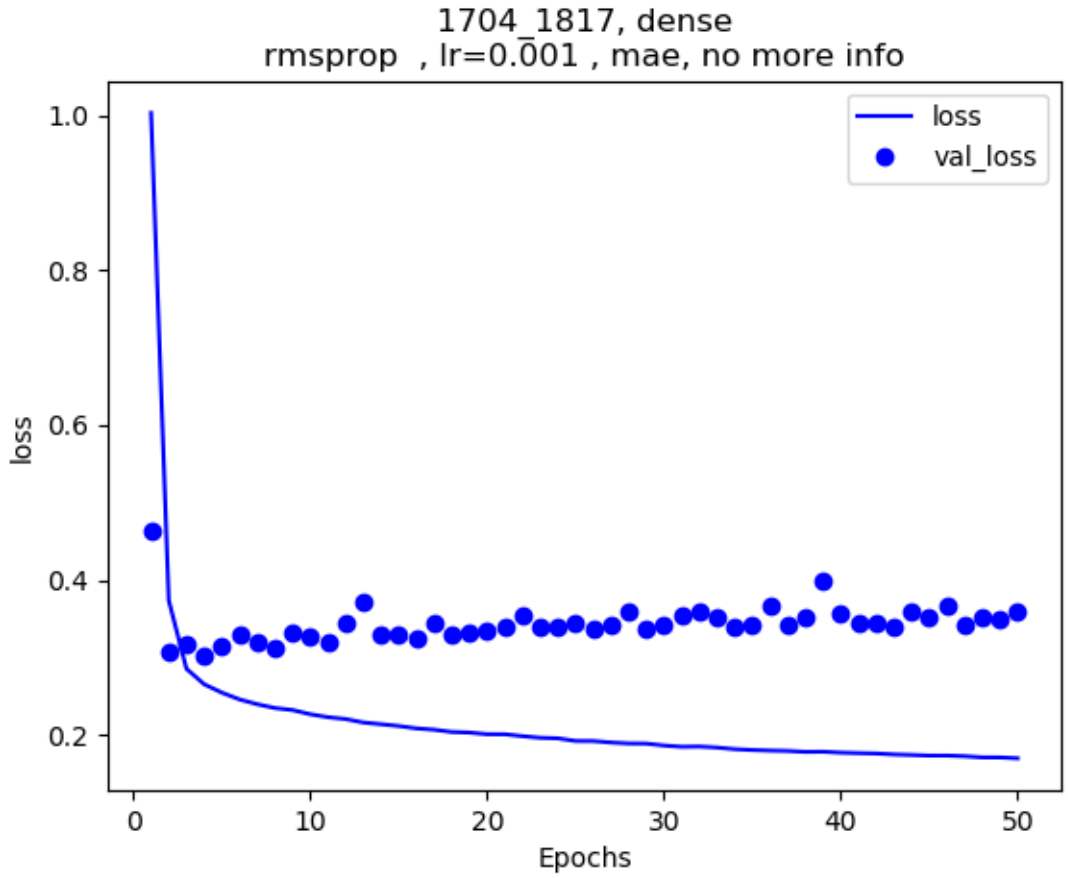


Illustration 2: dense_1704_1817

Validation loss is slowly getting worse and the best val_loss came at epoch 3

This example is the best example of overfitting . It is also interssant to notice that the validation loss is not so noisy to the contrary of other results.

Changing lr with grugru

grugru was run changing the learning rate (with lr=0.002 and 0.001) the results are

indice <codeRef> timeStample, modelStruct, info
min (val_loss) at i/nb epochs ,..... test_loss
18 <1504_1604>, grugru, rmsprop , **lr=0.002** , mae u1:32, u2:64 , d1:64
0.265 at 10/25 ,..... test: 0.2544
36 <1504_1705>, grugru, rmsprop , **lr=0.001** , mae u1:32, u2:64 , d1:64
0.2598 at 17/25 ,..... test: 0.2352

seems that lr=0.01 gets better results than lr=0.02

As expected the best validation loss is obtain more rapidelly on curve with lr=0.02

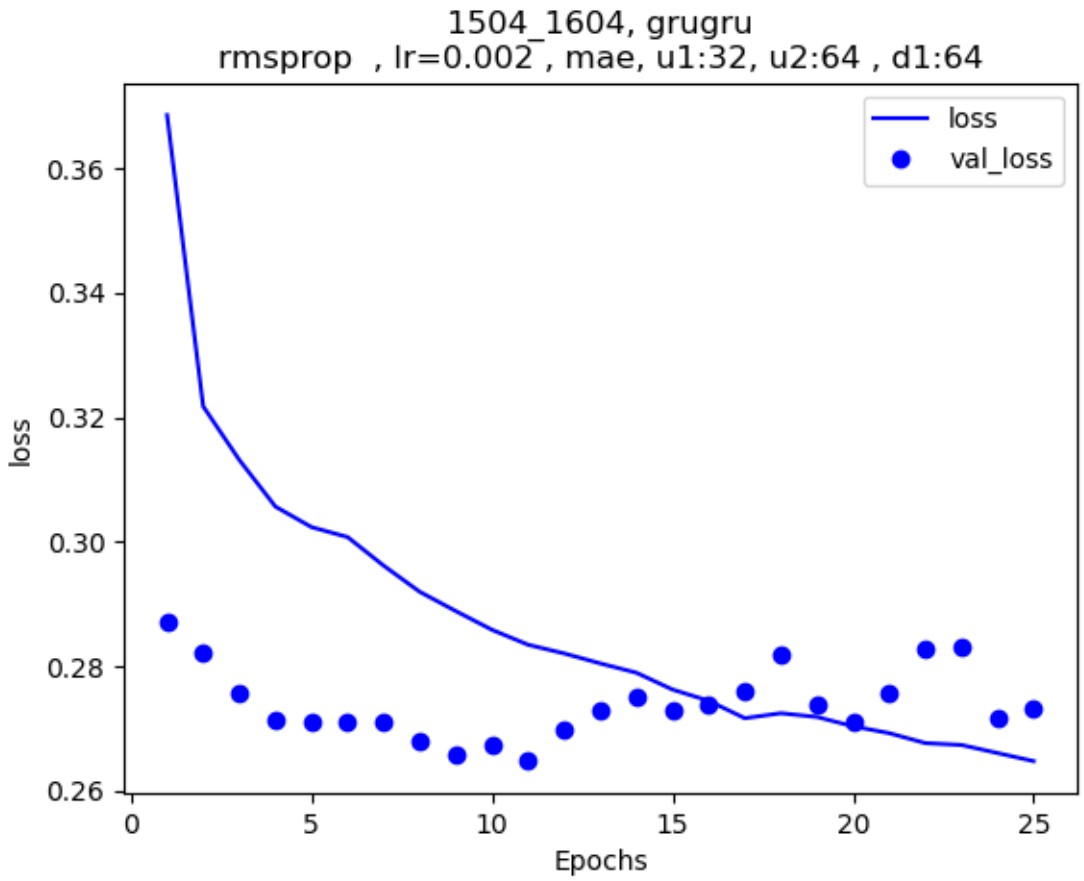


Illustration 3: influence of learning rate grugru lr=0.002

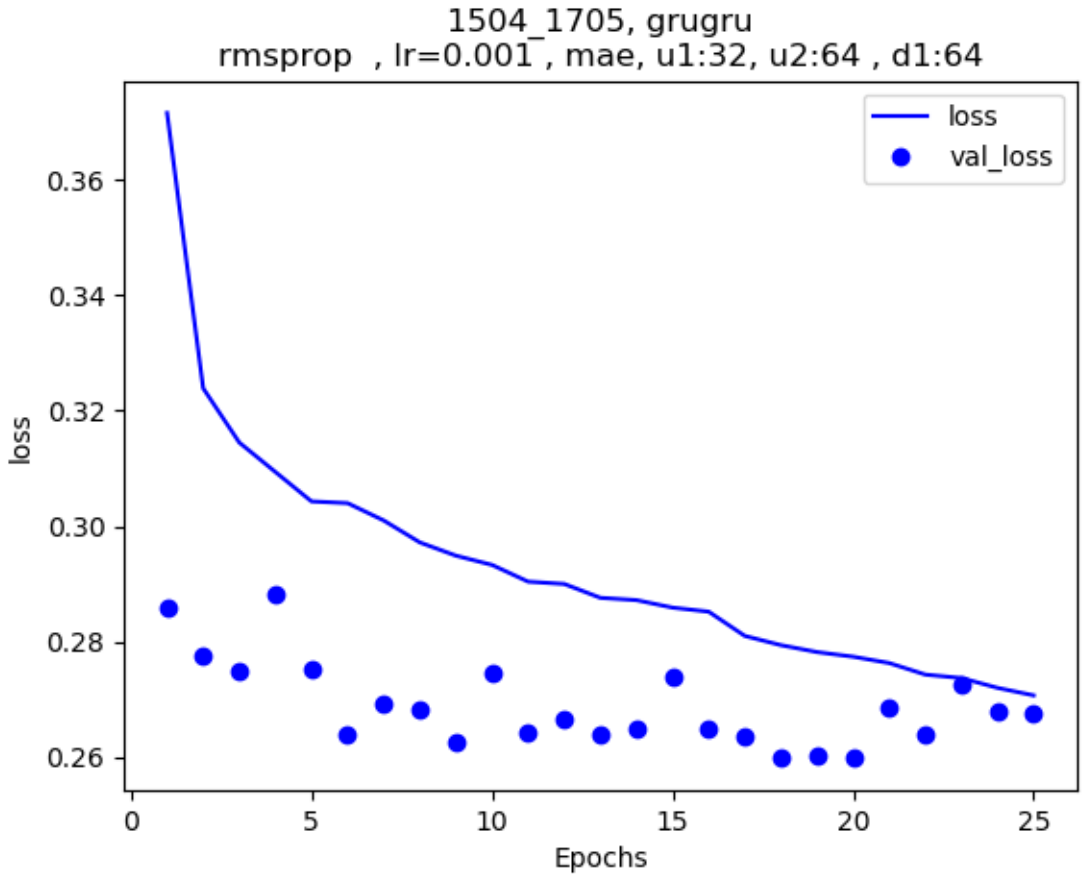


Illustration 4 influenve of learning rate grugru lr=0.001

the validation _loss for curve with lr=0.02 deteriorate more than the one with lr=0.01

changing lr with ggdd

Here is en extract out of the ordered list of results for learning rates

0.001 , 0.015 , 0.01 and 0.02 for the the model ggdd

indice <codeRef> timeStample, modelStruct, info
min (val_loss) at i/nb epochs ,..... test_loss
9 <1204_0937>, ggdd, rmsprop , lr=0.0001 , mae u1:32:64 , d:64,128
0.2682 at 24/25 ,..... test: 0.2751
19 <1204_1058>, ggdd, rmsprop , lr=0.002 , mae u1:32:64 , d:64,128
0.2694 at 4/25 ,..... test: 0.2542
20 <1204_1153>, ggdd, rmsprop , lr=0.0015 , mae u1:32:64 , d:64,128

0.2657 at 16/25 ,..... test: 0.2528
26 <2204_1643>, ggdd, rmsprop , lr=0.001 , mae u:32:64 , d:64,128
0.2625 at 21/25 ,..... test: 0.246

the corresponding curves are presented now close to each other to make easier comparison

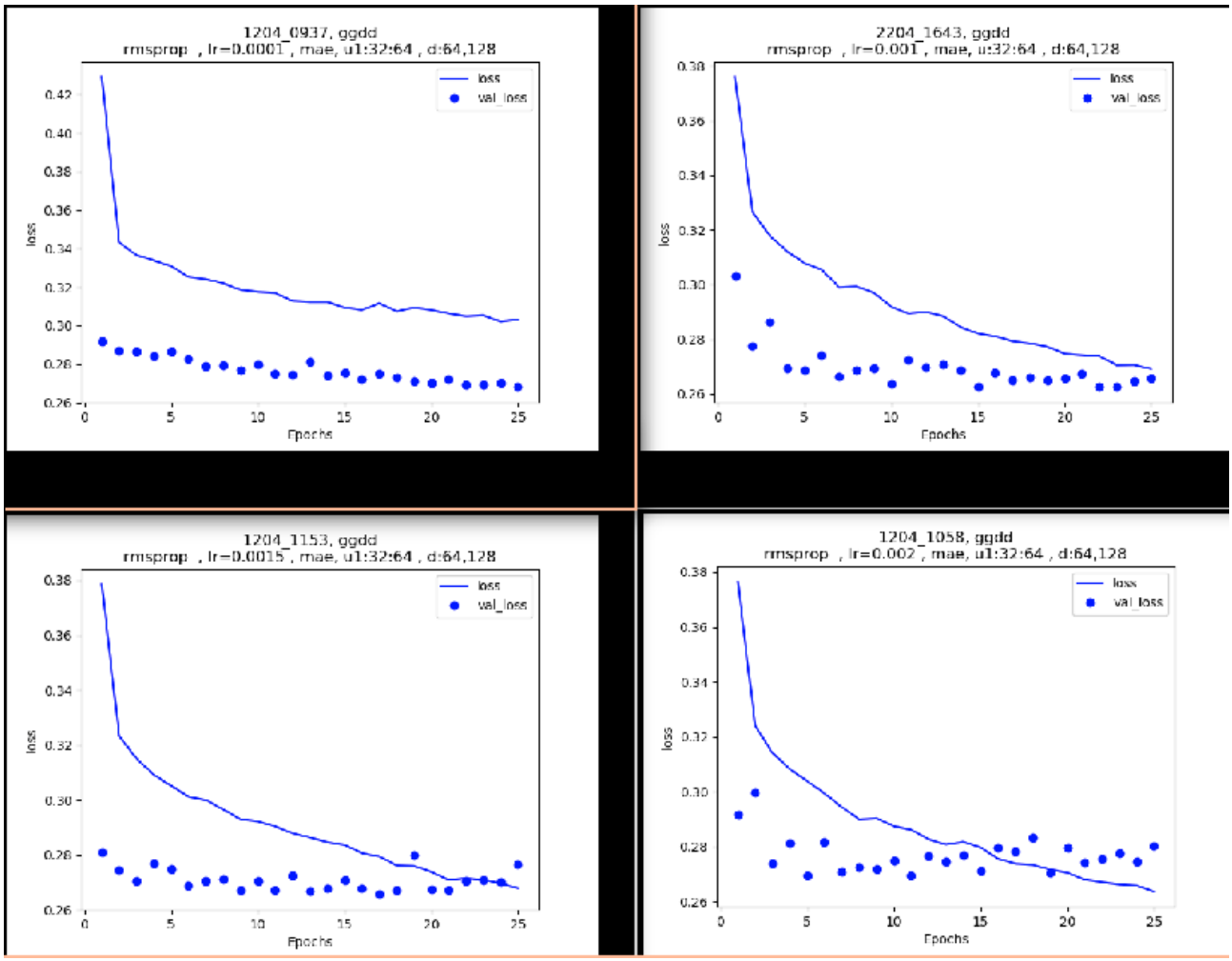


Illustration 5: changing learning rate with ggdd

the obvious observation is that the loss curve lr=0.0001 goes down more slowly and the final training loss (.3032) is much worst than for the others curve (around 0.2691).

and gives also worst test_loss .

All other validation loss curves are as usual noisy and around 0.265 .

3.5. Run with gru layers

Gru1 and gru2

Some test were done with makegru1

(1 gru layer with 32 units , This is the first recurrent baseline from the book)

and other with makegru2(2 gru layers with dropout=0.1 , and recurrent_dropout = 0.5 ,
the first gru has always 32 unit and one second gru has always 64 units)

Here is en extract out of the ordered list of results for test made with lr=0.01 , the worst result have indice 0

```
 6 <1004_1109>, gru1, rmsprop , lr=0.001 , mae TPU , inputBatchSize 128
0.2627 at 4/15 ,..... test: 0.2878
 7 <1004_0913>, gru1, rmsprop , lr=0.001 , mae GPU or CPU inputBatchSize 128
0.2643 at 3/20 ,..... test: 0.2807
22 <1004_1238>, gru2, rmsprop , lr=0.001 , mae no info yet
0.2641 at 13/35 ,..... test: 0.25
31 <1004_1139>, gru2, rmsprop , lr=0.001 , mae TPU , inputBatchSize 128
0.2657 at 8/30 ,..... test: 0.243
34 <1004_1317>, gru2, rmsprop , lr=0.001 , mae no info yet
0.2674 at 20/35 ,..... test: 0.2384
```

conclusion 2 layers gave better test_loss than one layer

grugru

2 gru layers with dropout=0.1 , and recurrent_dropout = 0.5 ,

and if d1 > 1 then the model have another a dense layer before the last Dense (that define predicted var.)

changing u1 and u2 d1=1

indice <codeRef> timeStample, modelStruct, info
min (val_loss) at i/nb epochs ,..... test_loss
15 <1004_1825>, grugru, rmsprop , lr=0.001 , mae u1:6, u2:12 , d1:1
0.2681 at 18/20 ,..... test: 0.2573
25 <1004_1858>, grugru, rmsprop , lr=0.001 , mae u1:12, u2:24 , d1:1
0.2634 at 18/20 ,..... test: 0.2476
29 <1004_1959>, grugru, rmsprop , lr=0.001 , mae u1:32, u2:32 , d1:1
0.2643 at 6/20 ,..... test: 0.245

It is clear that when there are not enough units the loss_curve does not get better than 0.32 to compare to other run with grugru or ggdd were it goes down to 0.27 at the last epoch.

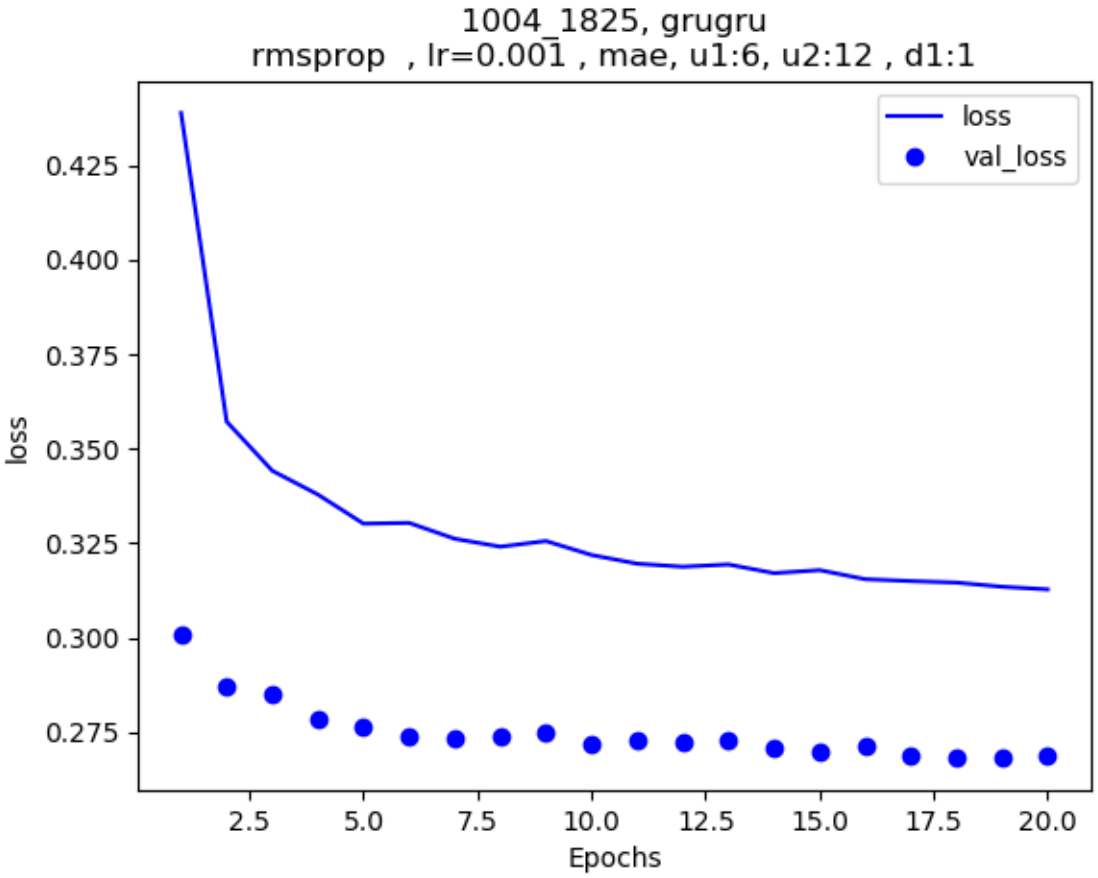


Illustration 6: grugru not enough unit

Changing d1 in grugru

Test made with grugru u1=32 , u2=64 , lr=0.001 changing d1 the size of the dense layer

Seems we got best results with d1=64

indice <codeRef> timeStample, modelStruct, info
min (val_loss) at i/nb epochs ,..... test_loss
17 <1104_1612>, grugru, rmsprop , lr=0.001 , mae u1:32, u2:64 , d1:12
0.2615 at 21/25 ,..... test: 0.255
34 <1104_1748>, grugru, rmsprop , lr=0.001 , mae u1:32, u2:64 , d1:24
0.2628 at 15/25 ,..... test: 0.2397
28 <1104_1833>, grugru, rmsprop , lr=0.001 , mae u1:32, u2:64 , d1:128
0.2608 at 11/25 ,..... test: 0.2454
37 <1504_1705>, grugru, rmsprop , lr=0.001 , mae u1:32, u2:64 , d1:64
0.2598 at 17/25 ,..... test: 0.2352

Other wise 4 curves look quite similar I my eyes , as D1=64 got the best test_loss I will present it

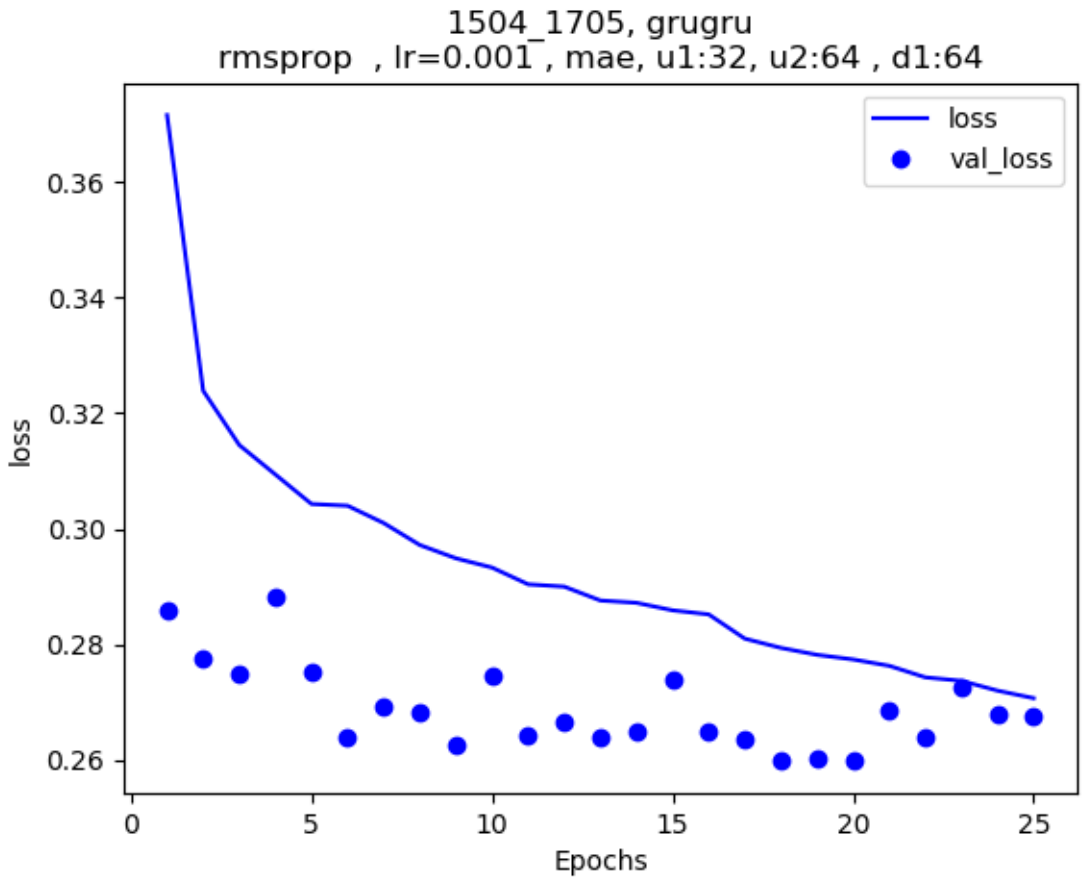


Illustration 7: grugru u1=32 u2=64 d1=64

ggdd

2 gru layers with dropout=0.1 , and recurrent_dropout = 0.5

2 dense layers more than the dense layer for predicted_var

the run with indice 26 and 39 are the same (same model , same hyperparameters …)

and get

 . The best validation loss are

indice <codeRef> timeStample, modelStruct, info
min (val_loss) at i/nb epochs ,..... test_loss
26 <2204_1643>, ggdd, rmsprop , lr=0.001 , mae u:32:64 , d:64,128
0.2625 at 21/25 ,..... test: 0.246
30 <1204_0803>, ggdd, rmsprop , lr=0.001 , mae u1:32:64 , d:32,64
0.2658 at 19/25 ,..... test: 0.2436
38 <1204_0858>, ggdd, rmsprop , lr=0.001 , mae u1:32:64 , d:64,128
0.2629 at 21/25 ,..... test: 0.2315

ggdd_1204_0858

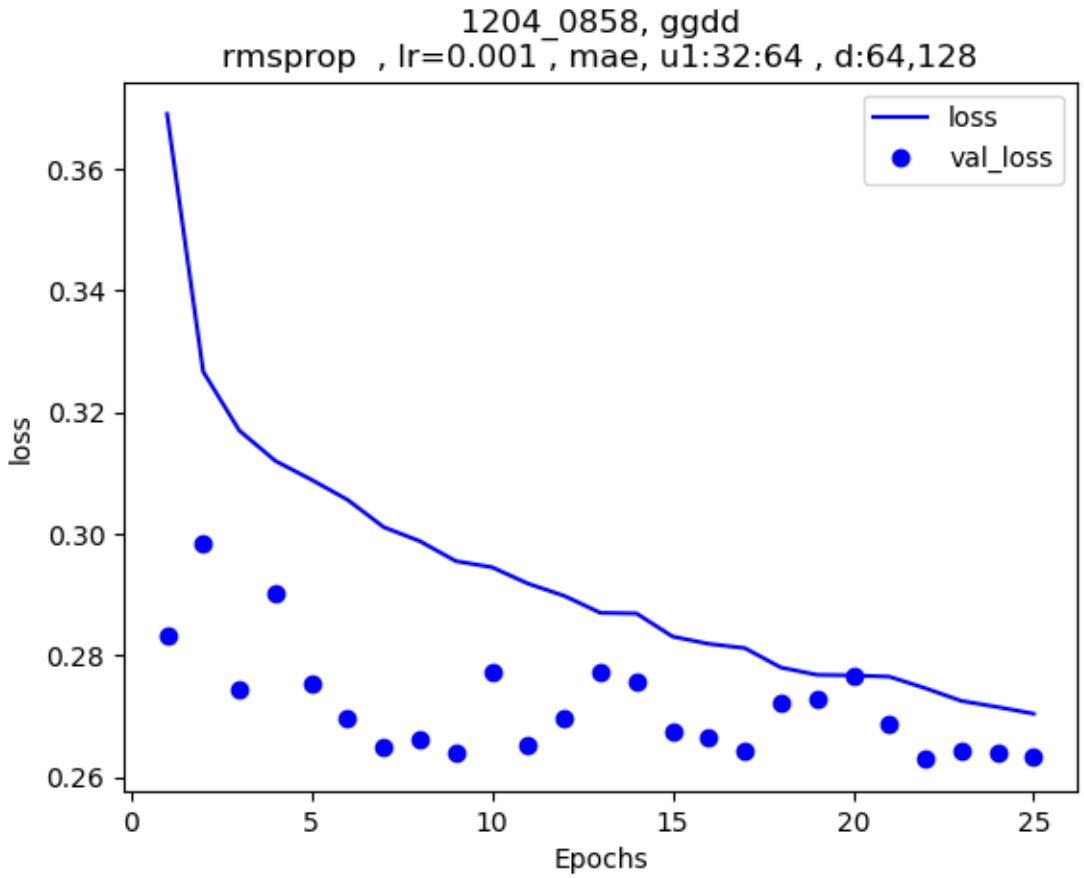


Illustration 8: loss curve ggdd_1204_0858

Looking more closely at the related curves (only one ot them is presented in the document but can be reproduced using the notebook lab2ReadJson) I conclude that

bigger dense layer gives the better optimization.

Test with Steps-per-epochs=1500

All runs were made with steps_per_epochs= 500 in fit_generator except the one presented now.

Note: Nb of steps per epochs is part of the history Params and can be seen by entering the indice of the run in lab2ReandJSON as seen here in this output from readJSonlab2 (run on local host))

syl1> python3 readJson_lab2Loc.py 1804_1653.json

indStr [moreStr] > 0

lab2SaveW onTPU, ggdd, rmsprop , lr=0.001 , mae, 1804_1653

u:32:64 , d:64,128

test Results 0.2521

best val loss : at epochs 3 /50 value 0.2638

history Params: {'batch_size': None, 'epochs': 50, **'steps': 1500,** 'samples': 1500, 'verbose': 1, 'do_validation': True, 'metrics': ['loss', 'val_loss']}

'metrics': ['loss', 'val_loss']}

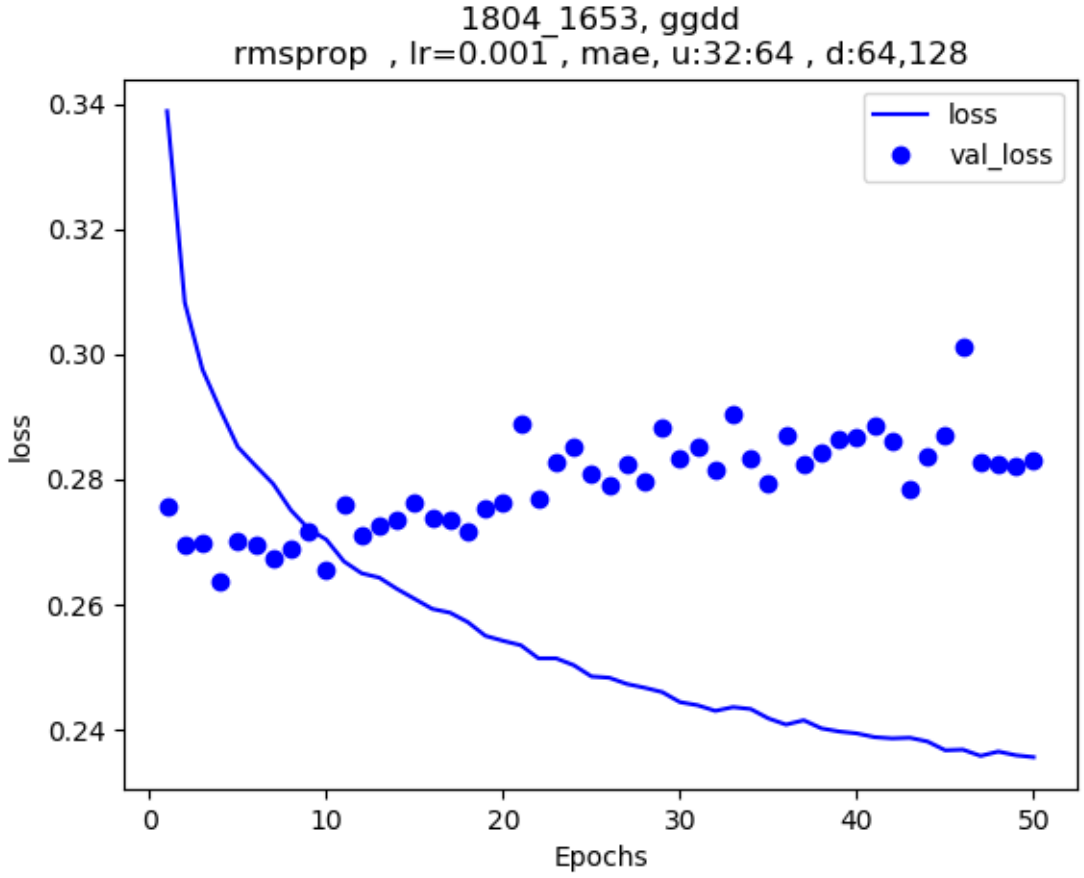


Illustration 9: plot : 1500 steps instead for 500

a clear overfitting I would say !!

gggdd

3 gru layers with dropout=0.1 , and recurrent_dropout = 0.5 ,

2 dense layers more than the dense layer for predicted_var

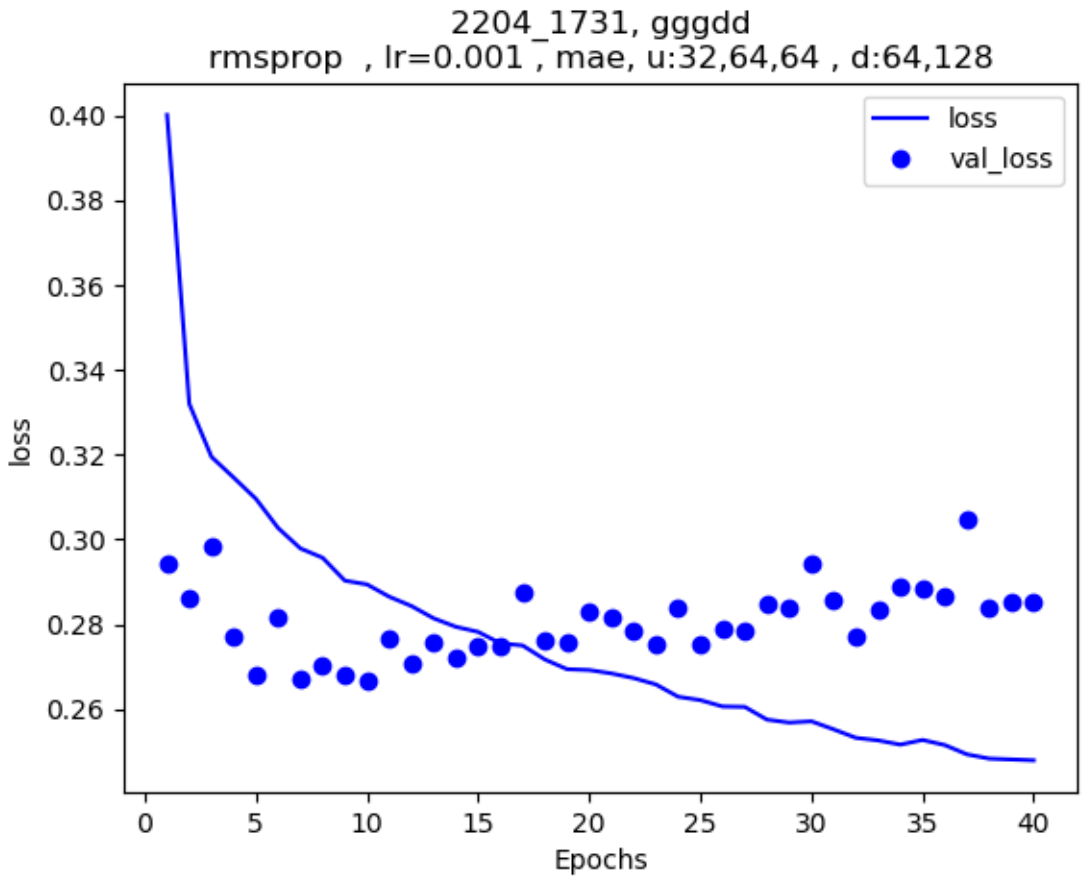
This time 2 test were made

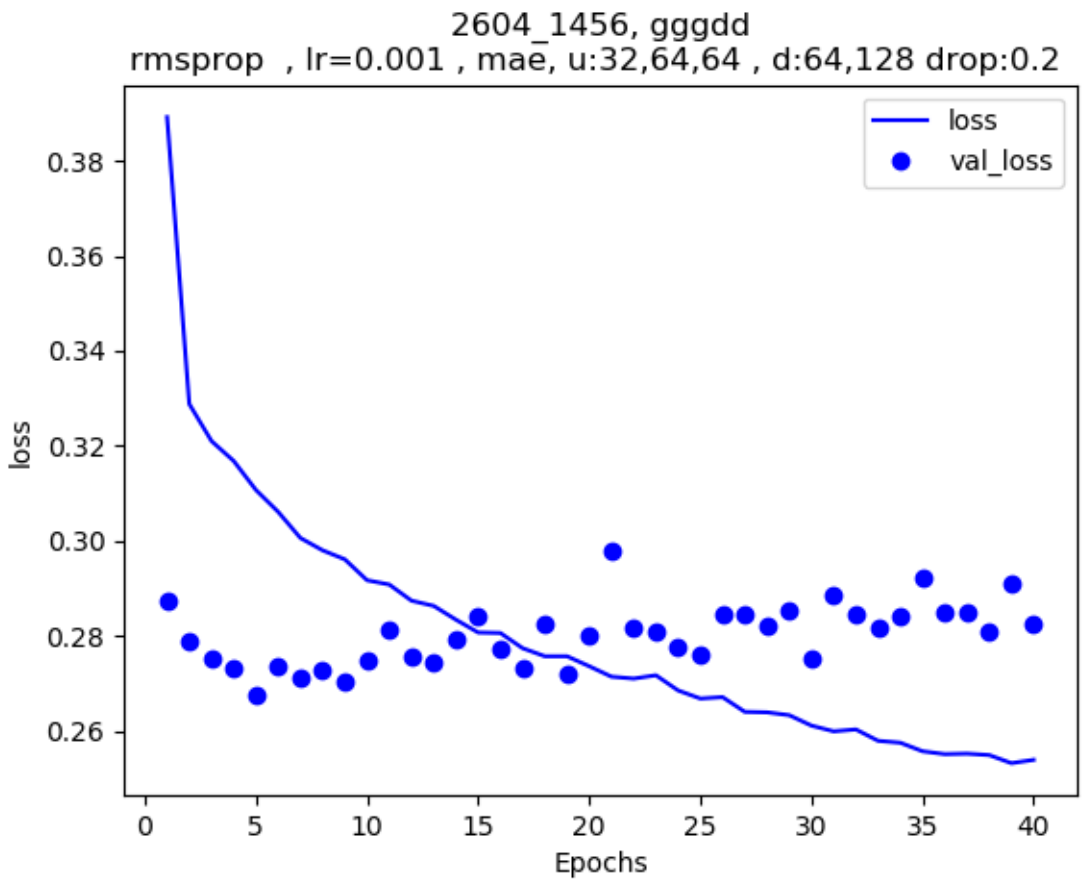
1) to stop the model at what the epochs that gave the best val_loss
2) to use dropout 0.2 instead of 0.1

indice <codeRef> timeStample, modelStruct, info
min (val_loss) at i/nb epochs ,..... test_loss
30 <2504_1622>, gggdd, rmsprop , lr=0.001 , mae u:32,64,64 , d:64,128
0.2685 at 5/9 ,..... test: 0.2446
xx <2604_1456>, gggdd, rmsprop , lr=0.001 , mae u:32,64,64 , d:64,128 drop:0.2
0.2676 at 4/40 ,..... test: 0.2337
38 <2204_1731>, gggdd, rmsprop , lr=0.001 , mae u:32,64,64 , d:64,128
0.2667 at 9/40 ,..... test: 0.2326

gggdd changing dropout

the two curves presented now differs only by the dropout which is 0.1 in the first case and 0.2 in the second case


2204_1731, gggdd
rmsprop , lr=0.001 , mae, u:32,64,64 , d:64,128
loss
val_loss
loss
Epochs
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0.26
0
5
10
15
20
25
30
35
40


2604_1456, gggdd
rmsprop , lr=0.001 , mae, u:32,64,64 , d:64,128 drop:0.2
loss
val_loss
loss
Epochs
0.38
0.36
0.34
0.32
0.30
0.28
0.26
0
5
10
15
20
25
30
35
40

Seems that there is less over fitting with dropout=0,2 !

3.6. Run with lstm layers

Lstm should be more adapted to time serie but did not get better results then with Gru layers

lstmlstm

(lstmlstm== 2 lstm layers with dropout=0.1 , and recurrent_dropout = 0.5 no dense layers at the end) change u1 u2 between the run

The test with indice 11 and 16 differs by nb of epochs for the training.

Here is en extract out of the ordered list of results were the worst result have indice 0

```
indice <codeRef> timeStample, modelStruct, info
min (val_loss) at i/nb epochs ,..... test_loss
 8  <1004_1645>,  lstmlstm, rmsprop  , lr=0.001 , mae u1:64, u2:128
0.2607 at 7/20  ,..... test: 0.2783
10  <1004_1533>,  lstmlstm, rmsprop  , lr=0.001 , mae u1:64, u2:64
0.2623 at 6/25  ,..... test: 0.2745
11  <1504_2013>,  lstmlstm, rmsprop  , lr=0.001 , mae u:32,64
0.261 at 6/25  ,..... test: 0.2677
16  <1004_1454>,  lstmlstm, rmsprop  , lr=0.001 , mae u1:32, u2:64
0.2609 at 12/35  ,..... test: 0.2559
```

I choose to show the 2 best lstm lstm curves

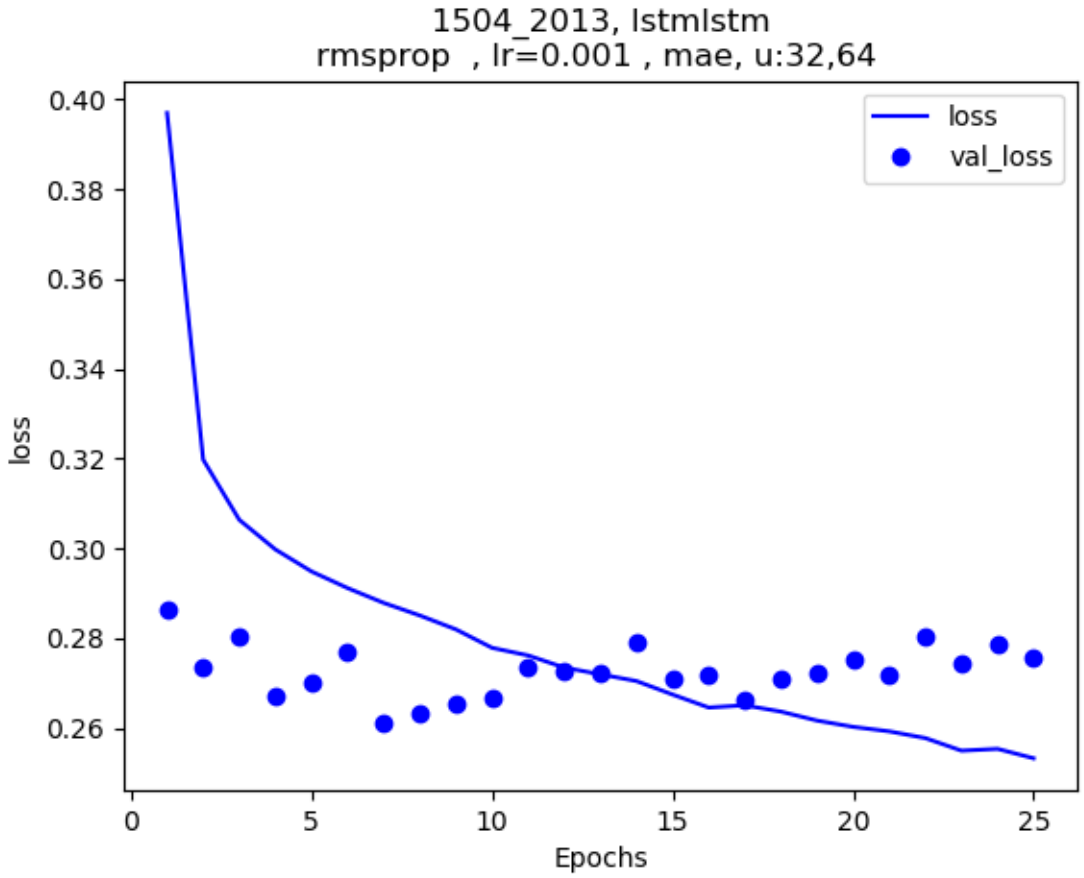


Illustration 10: loss curve lstmlstm_1504_2030

Validation loss is deteriorating after epoch 16 and also fluctuating , meaning ?

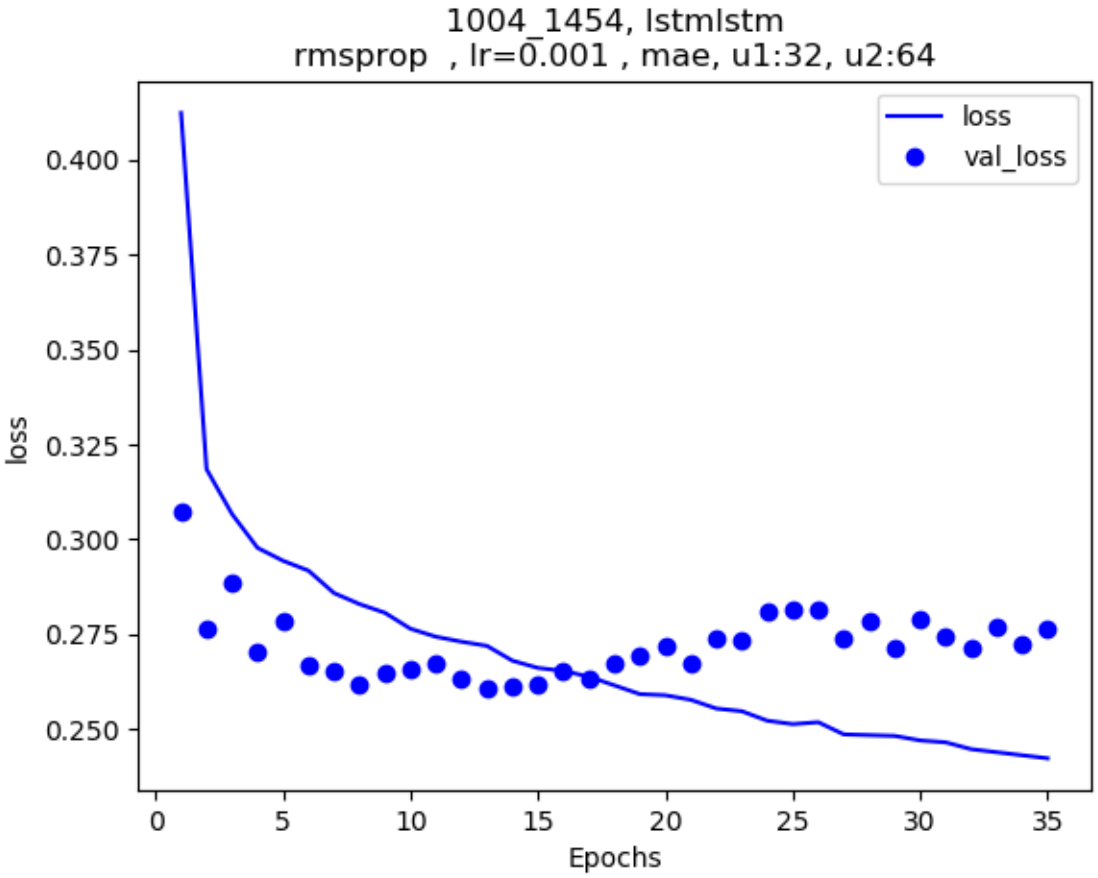


Illustration 11: lstmlstm 1004_1454

Differ from the previous curve by the nb of epochs

You could believe that the validation_loss is getting a bit better after epoch 25 , meaning ?

Lldd

Then I made some run with the lldd model

(2 lstm layers with dropout=0.1 , and recurrent_dropout = 0.5 ,

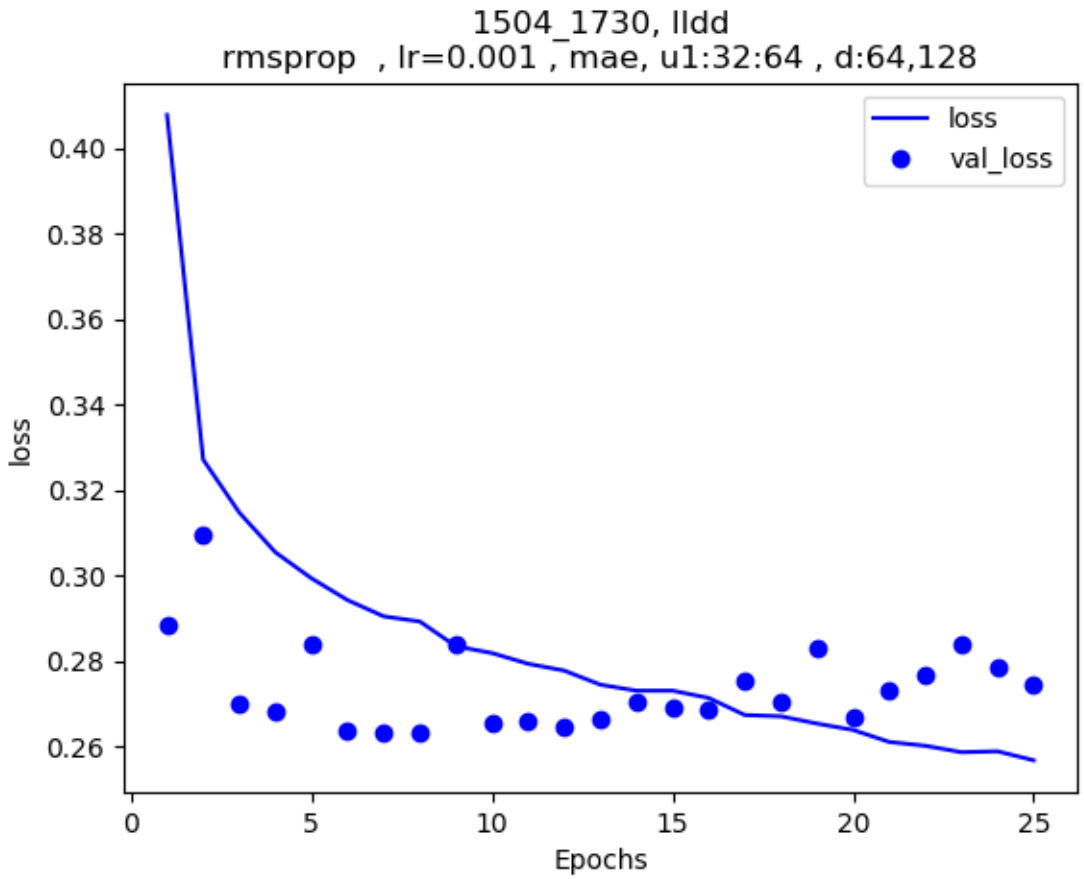
2 dense layers more than the dense layer for predicted_var)

One more I tested several nb of units and changes the leatning rate

Here is en extract out of the ordered list of results were the worst result have indice 0

indice <codeRef> timeStample, modelStruct, info
min (val_loss) at i/nb epochs ,..... test_loss
12 <1204_1346>, lldd, rmsprop , lr=0.002 , mae u1:32:64 , d:64,128
0.2617 at 4/25 ,..... test: 0.265
13 <1704_1639>, lldd, rmsprop , lr=0.001 , mae u1:32:64 , d:32,64
0.2645 at 11/25 ,..... test: 0.2588
14 <1204_1314>, lldd, rmsprop , lr=0.001 , mae u1:32:64 , d:64,128
0.2641 at 10/25 ,..... test: 0.2587
21 <1204_1233>, lldd, rmsprop , lr=0.0015 , mae u1:32:64 , d:64,128
0.2632 at 5/25 ,..... test: 0.2516
24 <1504_1730>, lldd, rmsprop , lr=0.001 , mae u1:32:64 , d:64,128
0.2632 at 7/25 ,..... test: 0.2491

once more the curves are similar and I choose to present the last one



4. some predictions

I really wanted to see example of predictions to see if I could relate them to the val_loss and test_loss that were presented I the previous chapter.

It is quite fun to look at the predictions from models and to compare to the naive model

so for the first plots predictions of the the naive model are also presented

Some predictions are presented now ,

the mae from eval_generator (normalized mae) as ell as the mae direclty caculted from the data are printed in the title.

Remember you may plot others prediction from the notebook labLoadW

4.1. About

About the data_sets and the dates

	Training	Val	test
Indexs in float_data	0:200000	200001:300000	300001:420550
start_date - stop_date :	01.01.2009- 19.10.2012	19.10.2012- 13.09.2014	14.09.2014 - 31.12.2016
Nb of days In the set	3years 10 months	1 years 11 months	Years 2 months

about the plots

Each plot represents temperatures evolution under 21h and 30 mn (because the batch_size is 128)

Let refer this plots by firstTargetDate in format (dd.mm.yyyy) the timeStample of the first point

one or two plot are presented for each firstTargetDate

plot1) true temp vs predicted temperature

plot2) yesterday temp vs true temp (the data are directly extracted from the csv file (also as a way to verify that I did not mixup all the indices)

selected firstTargetDate

predictions are presented now at 4 dates evenly distributed in one year.

As explained before to run a prediction for a given date you must select the corresponding min_index (see lab2U00) in the range 300000: 450000

the selected date were for test_data

Min index	Fist targetdate
324000	10.03.2015
337000	08.06.2015
350000	06.09.2015
370000	23.01.2016

4.2. ggdd_1204_1153

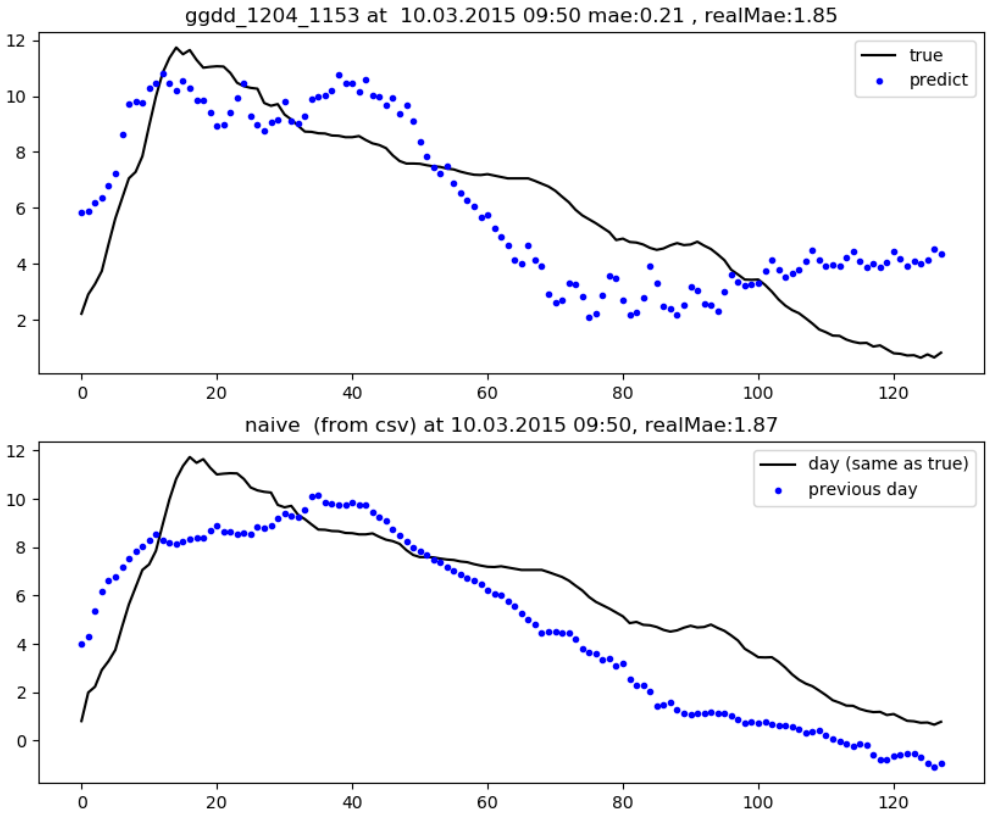


Illustration 12: ggdd_1204_1153 pred 324000

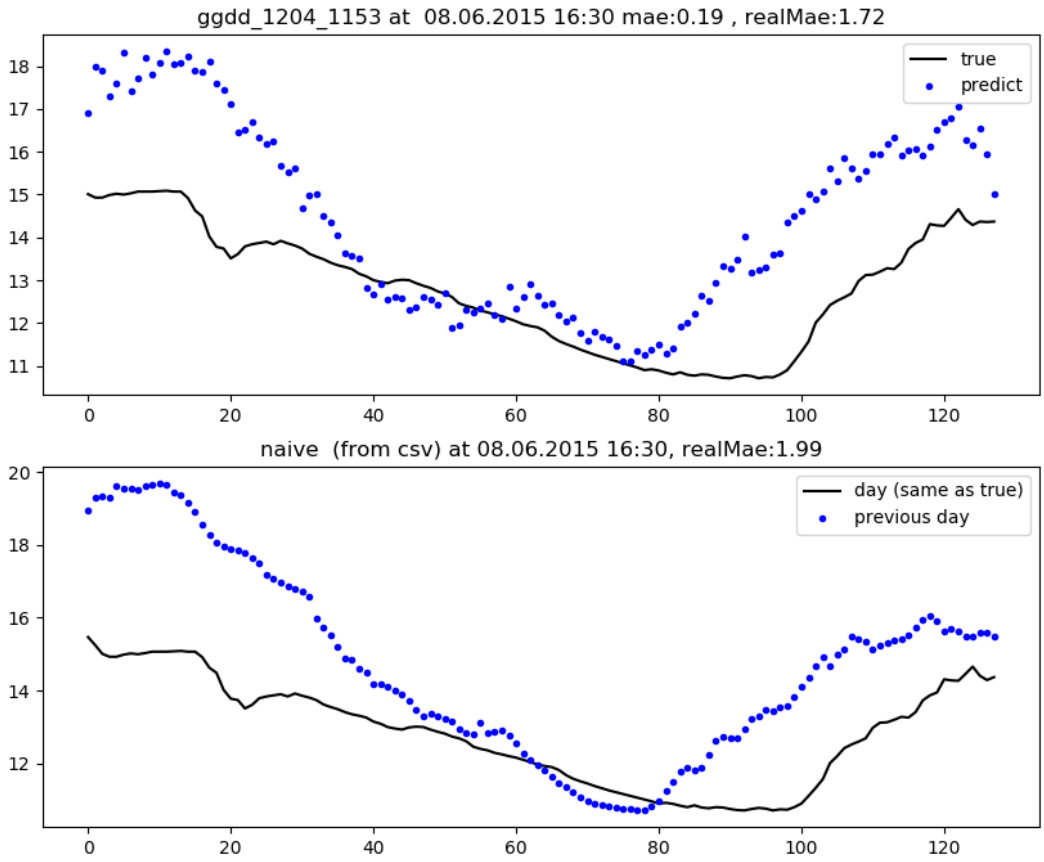


Illustration 13: ggdd_1204_1153 pred 337000

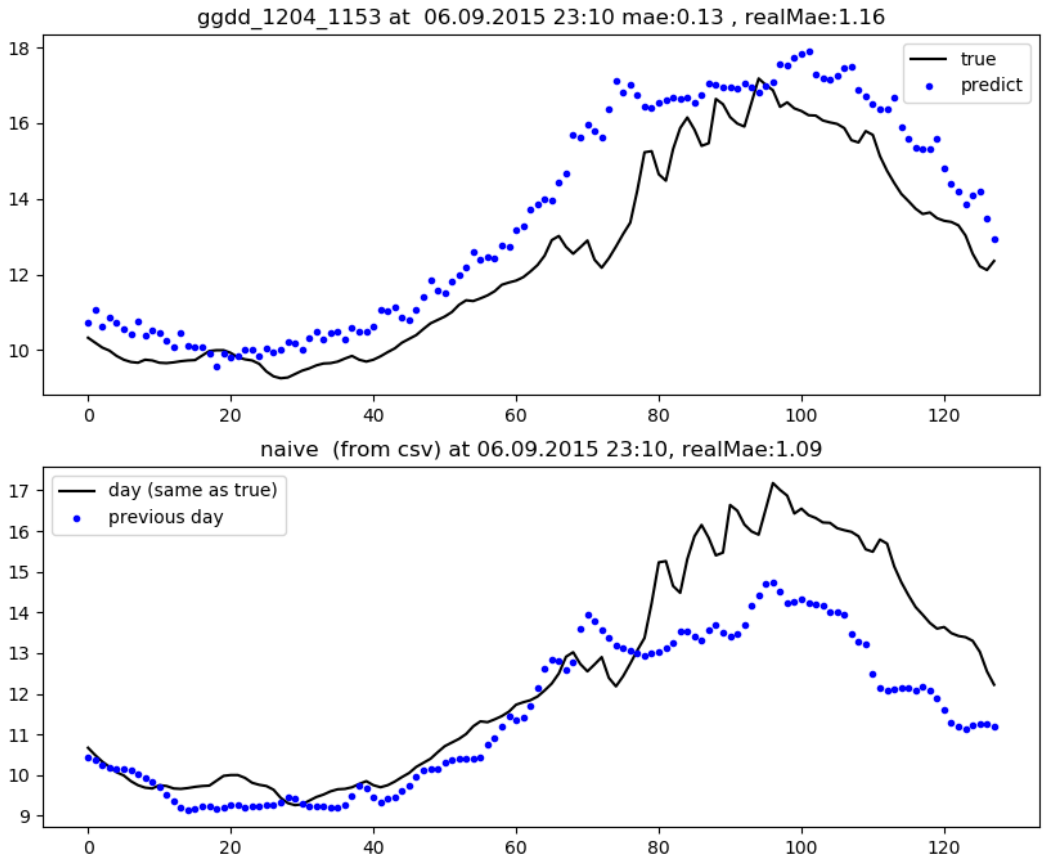


Illustration 14: ggdd_1204_1154 pred 350000

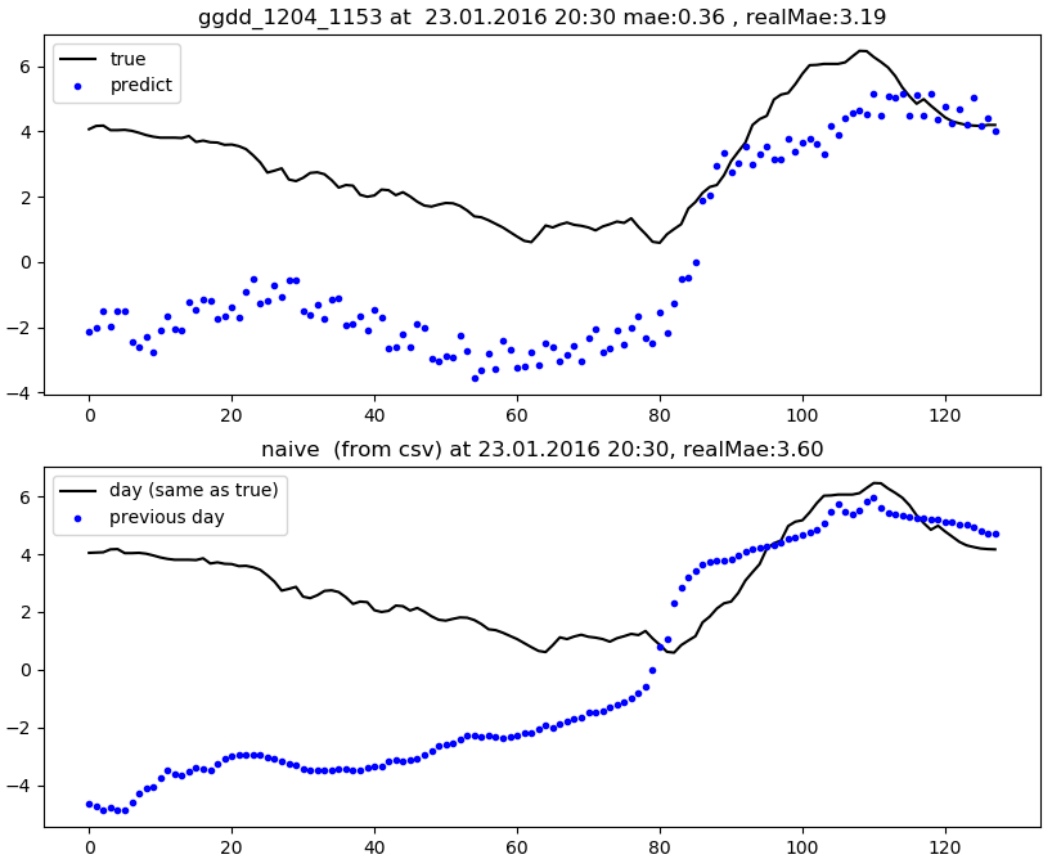


Illustration 15: ggdd_1204_1153 pred 350000

4.3. lldd

The naive model is not shown anymore because it has already been shown previoulsy

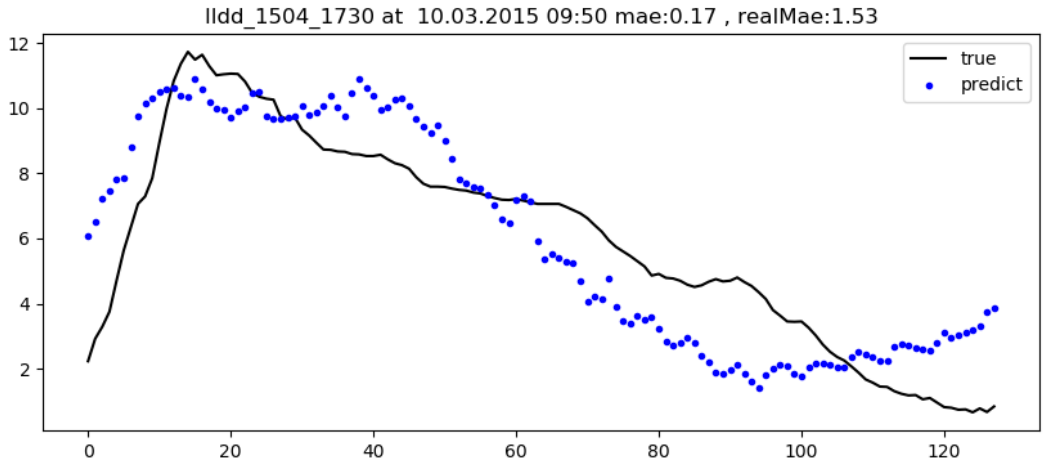


Illustration 16: lldd_1504_1730 temp 324000

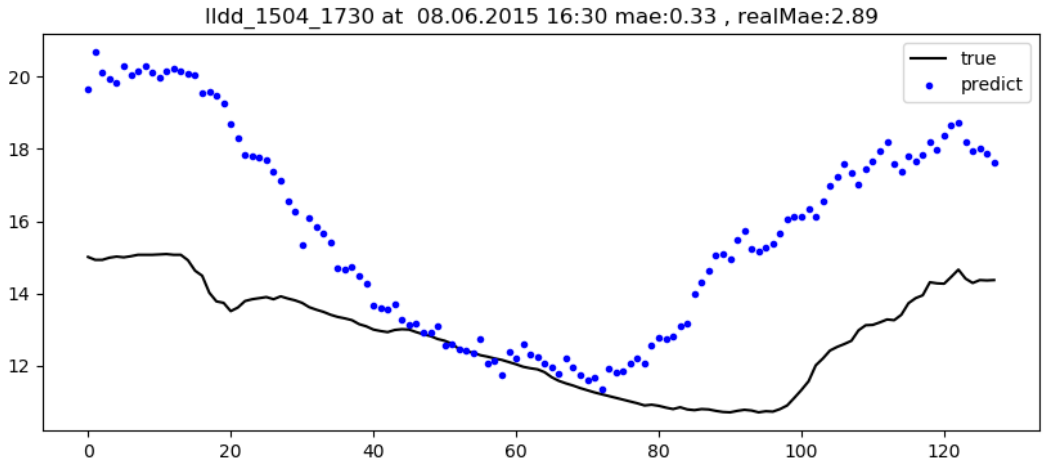


Illustration 17: lldd_1504_1730 temp 337000

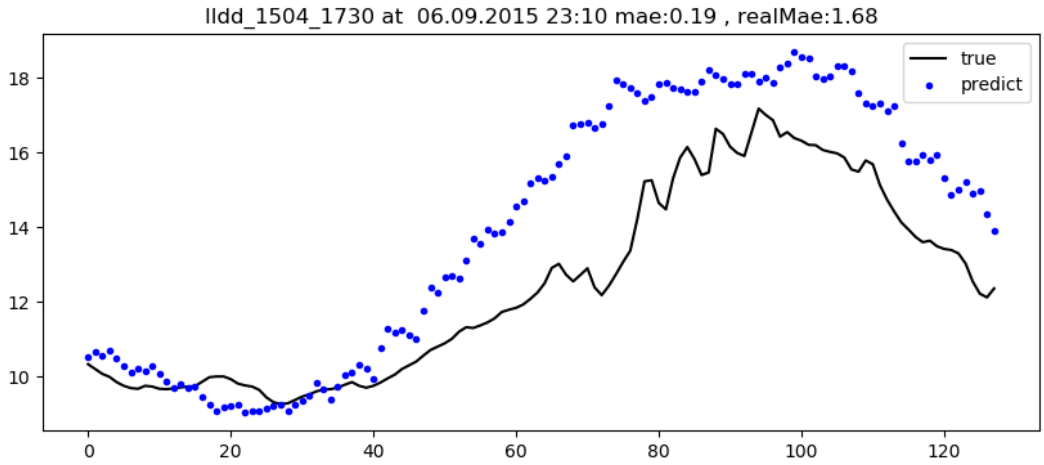


Illustration 18: lldd_1504_1730 temp 350000

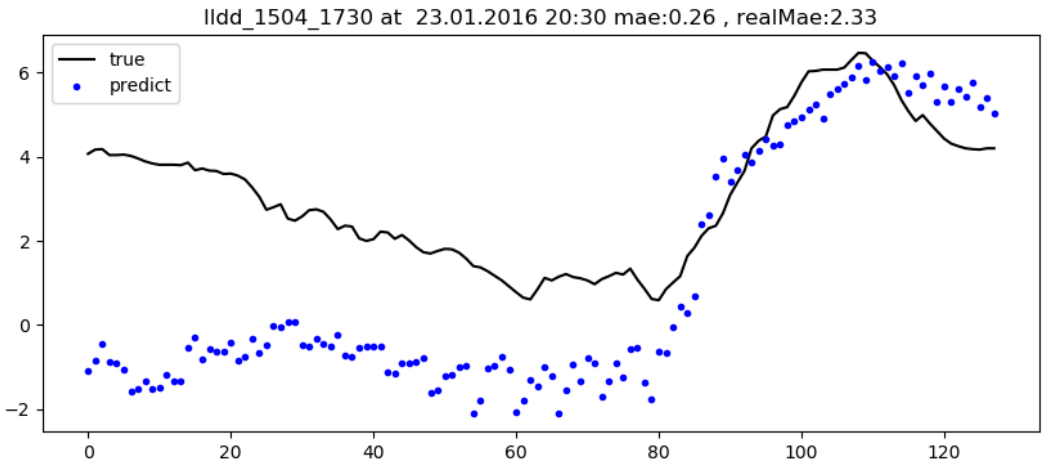


Illustration 19: lldd_1504_1730 temp 370000

4.4. dense

Even if Dense do not care about timeseries it stil manage to predict reasonables values ,

and I could not notice that it was so much worst that the RNN predictions

The naive model is not shown because it has already been shown previously

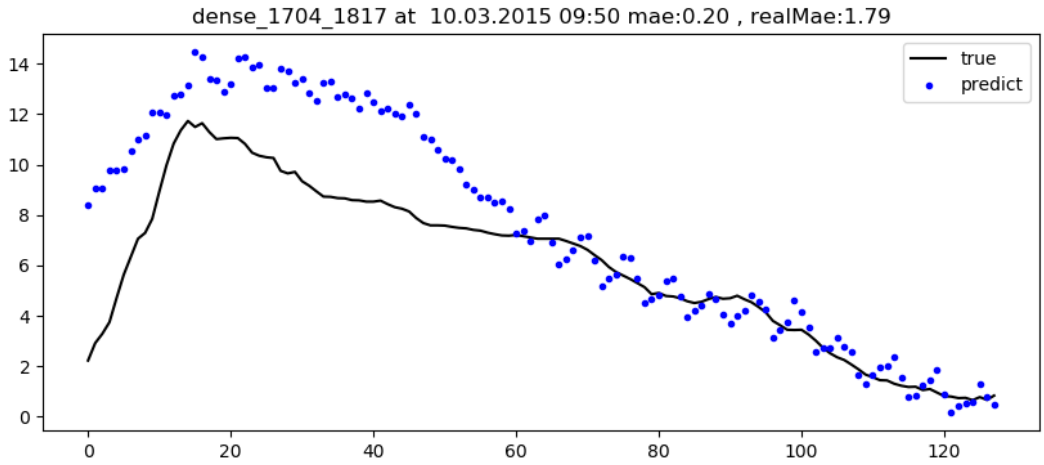


Illustration 20: dense temp 324000

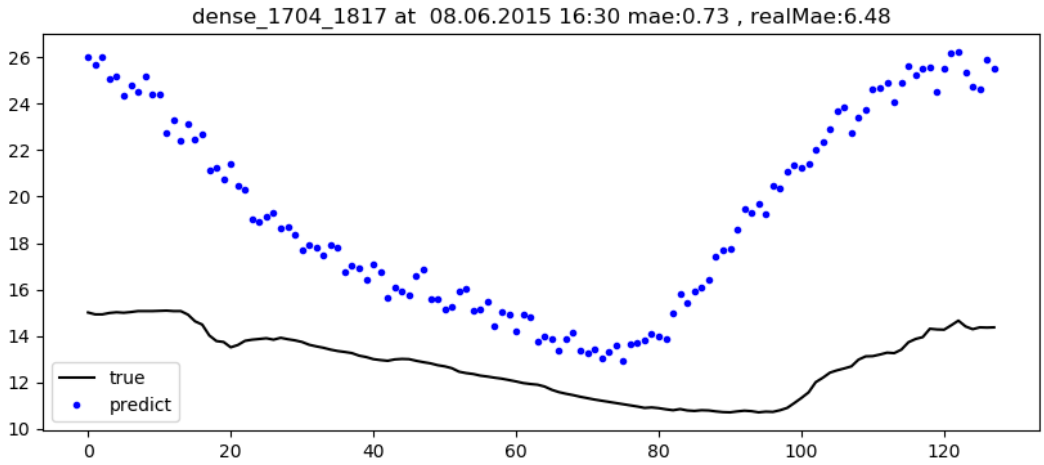


Illustration 21: dense temp 337000

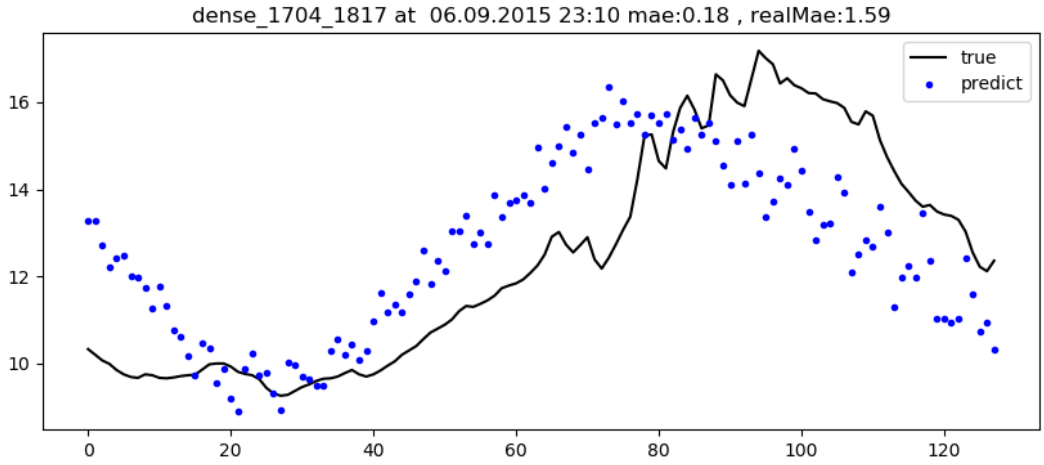


Illustration 22: dense temp 350000

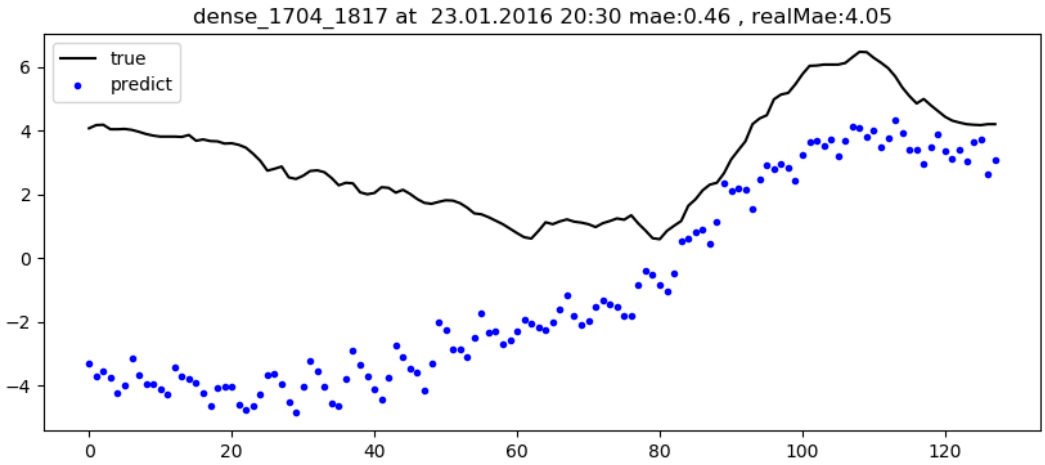


Illustration 23: dense temp 370000

4.5. for fun make predictions on training data

It is always said that the predictions should not be made on training data .

The training data covers float_data [0:200000]

I did it anyways and was expecting the see a better fit between the true label and the predicted labels on the training data which is not true on this examples

the selected date for this test were:

Min index	Fist targetdate
24000	27.06.2009
37000	25.09.2009
50000	25.12.2009
70000	13.05.2010

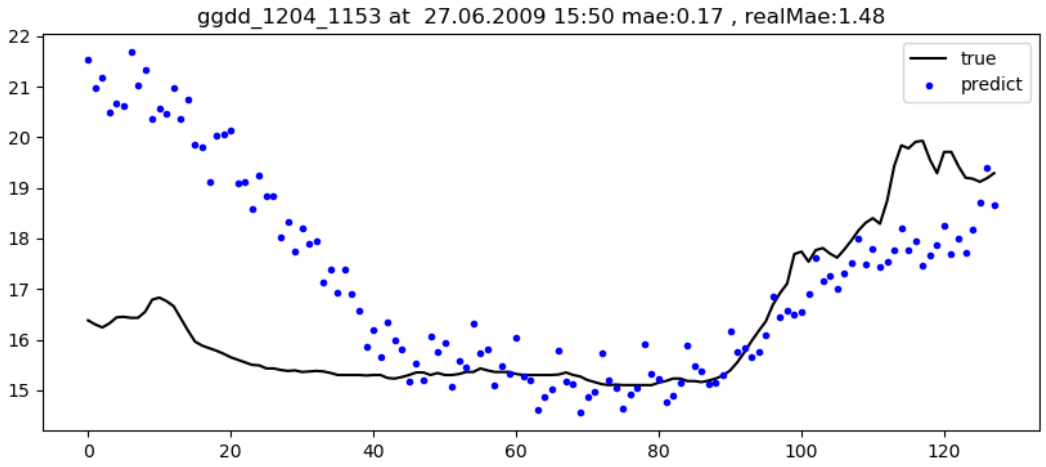


Illustration 24: ggdd_1204_1153 temp 24000

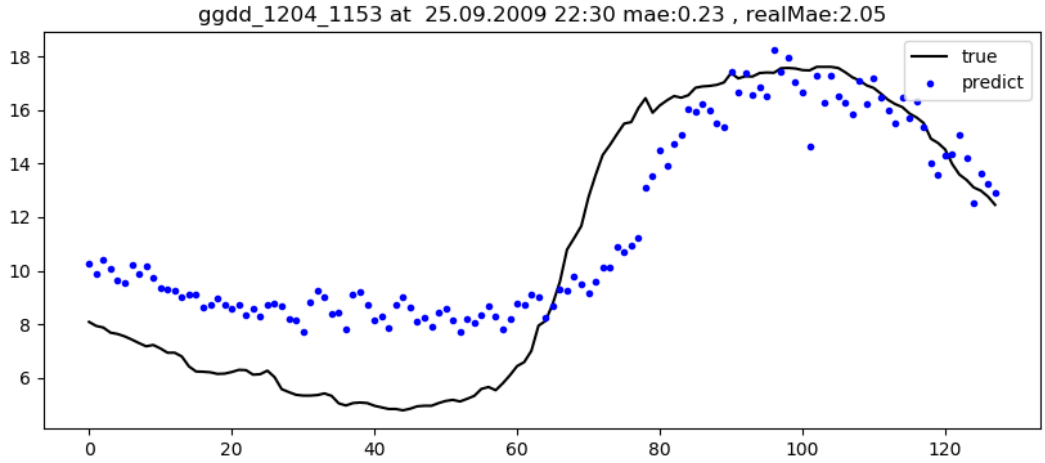


Illustration 25: ggdd_1204_1153 temp 37000

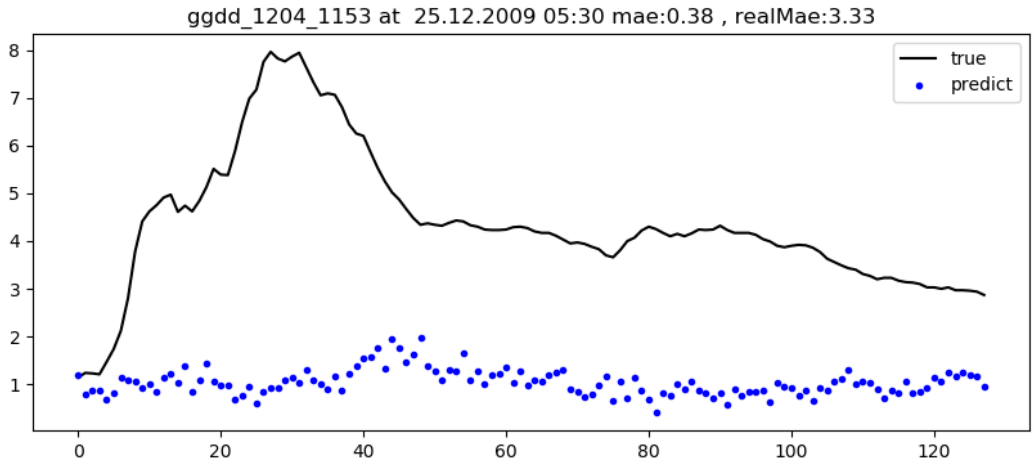
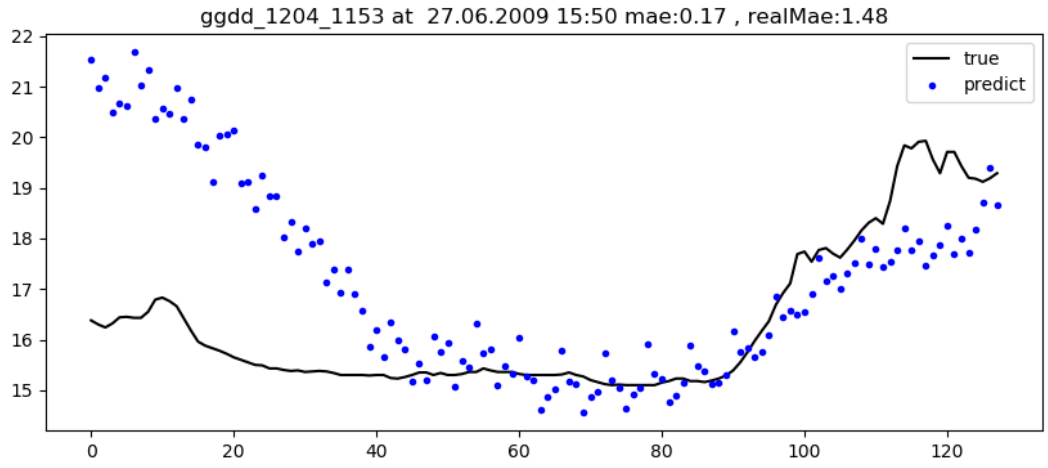


Illustration 26: ggdd_1204_1153 temp 50000



5. Conclusions

Quoting Cholet ch 6.3 , about Weather predictions with jena

<<This is a fairly challenging problem that exemplifies many common difficulties encountered when working with timeseries.>>

My own main conclcusion is : “the challenge is still open “

5.1. About the models

Several model were test from a simple dense layer to a model with 3 gru layers and 2 dense layers.

Total about 37 training were made (see list in the appendix) with the models presented I nthe beginning (see ch2 . Pythoncode - lab2MakeModels)

The recurent models tested in this labs gave validation loss similar to the one presented in the book (cholet cchap 6.3) : around 0.265 !

No metrics

Notice that there was no metrics used (so was the case in the book) .

mae was the loss function and was therefore also to consider as a a kind of metric.

5.2. about training loss and validation loss

To evaluate the model you must look at the training loss curve and the validation loss curve .

Training loss give you hints about the optimization .

to detect overfitting you must look at both curves at once

The optimisation worked as expected and in most of the training the curse was close to horizontal

Validation curves were noisy and quite horizontals .

making is most of the time hard to distinguish a clear over- fitting from noise.

5.3. Ggdd u:32, 64 , d:64,128 best model ?

It is really hard to conclude about which model seems best because the results are quite similar.

In particular I did not notice a big difference between using lstm and gru layers .

So I choose the model that gave the best test_results and it is

39 <1204_0858>, ggdd, rmsprop , lr=0.001 , mae u1:32:64 , d:64,128

0.2629 at 21/25 ,..... test: 0.2315

5.4. About the predictions

On the few prediction presented here I did not notice much differences between the models and not either between using test data och training data for making the predictions

Of course it does not mean much

to plot 4 batch (21:30 hours per batch) out of 2 years and 2 months possible data (test dataset)

and plot 4 batch (21:30 hours per batch) out of 3 years and 10 months for the (training set)

Note : It is easy to produce more predictings curves using the notebook lab2loadW if you are interested.

But still you may say that the predictions are quite similar to the prediction of the naive model so it seems that all the implemented models manage to take into account the data from previous days

6. Appendix

6.1. Some definitions

Recurrent neural network

https://en.wikipedia.org/wiki/Recurrent_neural_network

A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to

process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition[1] or speech recognition.

6.2. About running TPU and about functional models

It takes much more time to run functional model than sequential models

but the problem is that you cannot use sequential models with TPU

Using TPU is much faster than to run on GPU.

Outputs from fit_generator will be presented now to illustrate these facts.

The code for fit_generator is

```
theFit = theModel.fit_generator(
   lab2U00.getTrainGen(), steps_per_epoch = 500,   validation_data=lab2U00.getValGen(),
 validation_steps=lab2U00.getValSteps(),   epochs=myEpochs,   verbose=1)
```

sequential, dense on GPU 11s per epoch

The model

```
model = Sequential()
model.add(layers.Flatten(input_shape=(lookback // step, float_data.shape[-1])))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1))
```

computation time

```
Extract
500/500 [==============================] - 13s 26ms/step - loss: 1.2508 - val_loss: 0.6043
Epoch 2/20
500/500 [==============================] - 11s 22ms/step - loss: 0.4372 - val_loss: 0.3868
Epoch 3/20
500/500 [==============================] - 12s 24ms/step - loss: 0.2989 - val_loss: 0.3491
```

sequential , one GRU, on GPU per epoch :195s

Notice that sequential model on GPU is much faster than functional model on GPU

the model

```
model = Sequential()
model.add(layers.GRU(32, input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))
```

computation time

```
Epoch 1/20
500/500 [==============================] - 195s 391ms/step - loss: 0.3070 - val_loss: 0.2728
Epoch 2/20
500/500 [==============================] - 194s 389ms/step - loss: 0.2854 - val_loss: 0.2673
Epoch 3/20
500/500 [==============================] - 194s 389ms/step - loss: 0.2780 - val_loss: 0.2757
Epoch 4/20
500/500 [==============================] - 194s 387ms/step - loss: 0.2739 - val_loss: 0.2635
Epoch 5/20
500/500 [==============================] - 193s 386ms/step - loss: 0.2668 - val_loss: 0.2634
Epoch 6/20
500/500 [==============================] - 195s 390ms/step - loss: 0.2640 - val_loss: 0.2704
Epoch 7/20
500/500 [==============================] - 195s 390ms/step - loss: 0.2586 - val_loss: 0.2733
```

functional :GRU1 , on GPU per epoch : 151s + 350s

Using makeGRU1Model , the Model:

```
source = Input(shape=(TSLen, nbOfFeat),
               batch_size=batch_size,
               dtype=tf.float32, name='Input')
gru1 = GRU(32, name='GRU')(source)
predicted_var = Dense(1, name='Output')(gru1)
model = tf.keras.Model(inputs=[source], outputs=[predicted_var])
```

computation time:

```
TPU not found
run on GPU or CPU
fit_generator(lab2U00.getTestGen() , steps=50)
Epoch 1/20
769/769 [==============================] - 151s 196ms/step - loss: 0.2725
500/500 [==============================] - 350s 700ms/step - loss: 0.3522 - val_loss: 0.2725
Epoch 2/20
769/769 [==============================] - 148s 192ms/step - loss: 0.2653
500/500 [==============================] - 341s 683ms/step - loss: 0.2894 - val_loss: 0.2653
...
Epoch 8/20
769/769 [==============================] - 141s 184ms/step - loss: 0.2728
500/500 [==============================] - 328s 656ms/step - loss: 0.2574 - val_loss: 0.2728
```

functional , dense on TPU per epoch 16s + 28 s

Using makeDenseModel () , the Model:

```
source = Input(shape=(TSLen, nbOfFeat) ,
            batch_size=batch_size,
            dtype=tf.float32, name='Input')
flat1 = tf.layers.flatten (source)
dense1 = Dense (32, activation='relu') (flat1)
predicted_var = Dense(1, name='Output')(dense1)
model = tf.keras.Model(inputs=[source], outputs=[predicted_var])
```

a More complete were you can also more output connected to TPU and tensor flow
to illustrate the computation time

```
Found TPU at: grpc://10.84.200.162:8470
WARNING:tensorflow:From <ipython-input-3-78e0039efe2c>:86: flatten (from tensorflow.python.layers.core) is deprecated and will be removed in a future version.
Instructions for updating:
Use keras.layers.flatten instead.
INFO:tensorflow:Querying Tensorflow master (grpc://10.84.200.162:8470) for TPU system metadata.
INFO:tensorflow:Found TPU system:
INFO:tensorflow:*** Num TPU Cores: 8
INFO:tensorflow:*** Num TPU Workers: 1
INFO:tensorflow:*** Num TPU Cores Per Worker: 8
```

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:CPU:0, CPU, -1, 1333935762917645281)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:XLA_CPU:0, XLA_CPU, 17179869184, 2506085700897269717)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:0, TPU, 17179869184, 7199165218210040068)

...

TPU, 17179869184, 8493044712714401640)

INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU_SYSTEM:0, TPU_SYSTEM, 17179869184, 9661584545094885187)

WARNING:tensorflow:tpu_model (from tensorflow.contrib.tpu.python.tpu.keras_support) is experimental and may change or be removed at any time, and without warning.

fit_generator(lab2U00.getTestGen() , steps=50)

Epoch 1/15

INFO:tensorflow:New input shapes; (re-)compiling: mode=train (# of cores 8), [TensorSpec(shape=(16,), dtype=tf.int32, name='core_id_20'), TensorSpec(shape=(16, 240, 14), dtype=tf.float32, name='Input_30'), TensorSpec(shape=(16, 1), dtype=tf.float32, name='Output_target_70')]

INFO:tensorflow:Overriding default placeholder.

INFO:tensorflow:Remapping placeholder for Input

INFO:tensorflow:Started compiling

INFO:tensorflow:Finished compiling. Time elapsed: 2.45865797996521 secs

INFO:tensorflow:Setting weights on TPU model.

499/500 [==============================>.] - ETA: 0s - loss: 1.3548INFO:tensorflow:New input shapes; (re-)compiling: mode=eval (# of cores 8), [TensorSpec(shape=(16,), dtype=tf.int32, name='core_id_30'), TensorSpec(shape=(16, 240, 14), dtype=tf.float32, name='Input_30'), TensorSpec(shape=(16, 1), dtype=tf.float32, name='Output_target_70')]

INFO:tensorflow:Overriding default placeholder.

INFO:tensorflow:Remapping placeholder for Input

INFO:tensorflow:Started compiling

INFO:tensorflow:Finished compiling. Time elapsed: 2.73699688911438 secs

769/769 [==============================] - 16s 20ms/step - loss: 0.6801

500/500 [==============================] - 37s 74ms/step - loss: 1.3536 - val_loss: 0.6801

Epoch 2/15

769/769 [==============================] - 12s 16ms/step - loss: 0.3296

500/500 [==============================] - 29s 58ms/step - loss: 0.4956 - val_loss: 0.3296

...

Epoch 6/15

769/769 [==============================] - 12s 16ms/step - loss: 0.3039

500/500 [==============================] - 28s 57ms/step - loss: 0.2486 - val_loss: 0.3039

….

functional GRUGRU + TPU per epoch 17s+54s

Grugru is defined in lab2MakeModels.py consiste of 2 gru layers

some output to illustrate the computation time , Notice also the compilation time

```
Found TPU at: grpc://10.84.200.162:8470
INFO:tensorflow:Querying Tensorflow master (grpc://10.84.200.162:8470) for TPU system metadata.
….
INFO:tensorflow:Overriding default placeholder.
INFO:tensorflow:Remapping placeholder for Input
INFO:tensorflow:Started compiling
INFO:tensorflow:Finished compiling. Time elapsed: 13.618455648422241 secs
....
Epoch 3/35
769/769 [==============================] - 15s 20ms/step - loss: 0.2708
500/500 [==============================] - 53s 106ms/step - loss: 0.3118 - val_loss: 0.2708
```

functional GRU2 on TPU per epoch 17s+54s

Gru2 can be the same as grugru also defined in lab2MakeModels.py

some output to illustrate the computation time :

```
Found TPU at: grpc://10.84.200.162:8470
INFO:tensorflow:Querying Tensorflow master (grpc://10.84.200.162:8470) for TPU system metadata.
INFO:tensorflow:Found TPU system:
INFO:tensorflow:*** Num TPU Cores: 8
INFO:tensorflow:*** Num TPU Workers: 1
INFO:tensorflow:*** Num TPU Cores Per Worker: 8
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:CPU:0, CPU, -1,
1333935762917645281)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:XLA_CPU:0,
XLA_CPU, 17179869184, 2506085700897269717)
....
Epoch 1/30
INFO:tensorflow:New input shapes; (re-)compiling: mode=train (# of cores 8), [TensorSpec(shape=(16,),
dtype=tf.int32, name='core_id_40'), TensorSpec(shape=(16, 240, 14), dtype=tf.float32, name='Input_50'),
```

```
INFO:tensorflow:Started compiling
INFO:tensorflow:Finished compiling. Time elapsed: 7.076465368270874 secs
Epoch 2/30
769/769 [==============================] - 16s 21ms/step - loss: 0.2769
500/500 [==============================] - 53s 107ms/step - loss: 0.3208 - val_loss: 0.2769
```

6.3. Complete list of run

Here is the list of the run made at this date (20190426) and I believe it will be the final list

As before the list is ordered by the value test_loss where the first result (indice 0)

got the worst test_loss. Notice that the best val_loss seen under training is also presented as well as the epoch at which this value was achieved versus total nb of epochs for the run.

```
 indice <codeRef> timeStample, modelStruct, info
min (val_loss) at i/nb epochs ,..... test_loss
 0  <1004_0858>,  dense, rmsprop 0.001 , lr=, mae GPU or CPU inputBatchSize 128
0.3765 at 1/2  ,..... test: 0.3447
 1  <1004_1122>,  dense, rmsprop  , lr=0.001 , mae  TPU , inputBatchSize 128
0.3039 at 5/15  ,..... test: 0.328
 2  <1704_1817>,  dense, rmsprop  , lr=0.001 , mae no more info
0.3013 at 3/50  ,..... test: 0.3277
 3  <1304_1333>,  dense, rmsprop  , lr=0.002 , mae no more info
0.3033 at 5/25  ,..... test: 0.3251
 4  <1204_1011>,  ggdd, rmsprop  , lr=0.01 , mae u1:32:64 , d:64,128
0.2972 at 0/25  ,..... test: 0.3136
 5  <1004_0826>,  D32 D1, rmsprop lr=0.001, mae GPU or CPU inputBatchSize 128
0.3062 at 3/5  ,..... test: 0.2919
 6  <1004_1109>,  gru1, rmsprop  , lr=0.001 , mae  TPU , inputBatchSize 128
0.2627 at 4/15  ,..... test: 0.2878
 7  <1004_0913>,  gru1, rmsprop  , lr=0.001 , mae GPU or CPU inputBatchSize 128
0.2643 at 3/20  ,..... test: 0.2807
 8  <1004_1645>,  lstmlstm, rmsprop  , lr=0.001 , mae u1:64, u2:128
0.2607 at 7/20  ,..... test: 0.2783
 9  <1204_0937>,  ggdd, rmsprop  , lr=0.0001 , mae u1:32:64 , d:64,128
```

0.2682 at 24/25 ,..... test: 0.2751

10 <1004_1533>, lstmlstm, rmsprop , lr=0.001 , mae u1:64, u2:64

0.2623 at 6/25 ,..... test: 0.2745

11 <1504_2013>, lstmlstm, rmsprop , lr=0.001 , mae u:32,64

0.261 at 6/25 ,..... test: 0.2677

12 <1204_1346>, lldd, rmsprop , lr=0.002 , mae u1:32:64 , d:64,128

0.2617 at 4/25 ,..... test: 0.265

13 <1704_1639>, lldd, rmsprop , lr=0.001 , mae u1:32:64 , d:32,64

0.2645 at 11/25 ,..... test: 0.2588

14 <1204_1314>, lldd, rmsprop , lr=0.001 , mae u1:32:64 , d:64,128

0.2641 at 10/25 ,..... test: 0.2587

15 <1004_1825>, grugru, rmsprop , lr=0.001 , mae u1:6, u2:12 , d1:1

0.2681 at 18/20 ,..... test: 0.2573

16 <1004_1454>, lstmlstm, rmsprop , lr=0.001 , mae u1:32, u2:64

0.2609 at 12/35 ,..... test: 0.2559

17 <1104_1612>, grugru, rmsprop , lr=0.001 , mae u1:32, u2:64 , d1:12

0.2615 at 21/25 ,..... test: 0.255

18 <1504_1604>, grugru, rmsprop , lr=0.002 , mae u1:32, u2:64 , d1:64

0.265 at 10/25 ,..... test: 0.2544

19 <1204_1058>, ggdd, rmsprop , lr=0.002 , mae u1:32:64 , d:64,128

0.2694 at 4/25 ,..... test: 0.2542

20 <1204_1153>, ggdd, rmsprop , lr=0.0015 , mae u1:32:64 , d:64,128

0.2657 at 16/25 ,..... test: 0.2528

21 <1204_1233>, lldd, rmsprop , lr=0.0015 , mae u1:32:64 , d:64,128

0.2632 at 5/25 ,..... test: 0.2516

22 <1004_1238>, gru2, rmsprop , lr=0.001 , mae no info yet

0.2641 at 13/35 ,..... test: 0.25

23 <1004_1404>, grugru, rmsprop , lr=0.001 , mae u1:64, u2:64

0.2632 at 9/35 ,..... test: 0.2496

24 <1504_1730>, lldd, rmsprop , lr=0.001 , mae u1:32:64 , d:64,128

0.2632 at 7/25 ,..... test: 0.2491

25 <1004_1858>, grugru, rmsprop , lr=0.001 , mae u1:12, u2:24 , d1:1

0.2634 at 18/20 ,..... test: 0.2476

26 <2204_1643>, ggdd, rmsprop , lr=0.001 , mae u:32:64 , d:64,128

0.2625 at 21/25 ,..... test: 0.246

27 <1004_1758>, grugru, rmsprop , lr=0.001 , mae u1:32, u2:64 , d1:12

0.2614 at 17/20 ,..... test: 0.2456

28 <1104_1833>, grugru, rmsprop , lr=0.001 , mae u1:32, u2:64 , d1:128

0.2608 at 11/25 ,..... test: 0.2454

29 <1004_1959>, grugru, rmsprop , lr=0.001 , mae u1:32, u2:32 , d1:1

0.2643 at 6/20 ,..... test: 0.245

30 <2504_1622>, gggdd, rmsprop , lr=0.001 , mae u:32,64,64 , d:64,128

0.2685 at 5/9 ,..... test: 0.2446

31 <1204_0803>, ggdd, rmsprop , lr=0.001 , mae u1:32:64 , d:32,64

0.2658 at 19/25 ,..... test: 0.2436

32 <1004_1139>, gru2, rmsprop , lr=0.001 , mae TPU , inputBatchSize 128

0.2657 at 8/30 ,..... test: 0.243

33 <1004_1717>, grugru, rmsprop , lr=0.001 , mae u1:64, u2:128 , d1:12

0.2627 at 9/20 ,..... test: 0.2427

34 <1104_1748>, grugru, rmsprop , lr=0.001 , mae u1:32, u2:64 , d1:24

0.2628 at 15/25 ,..... test: 0.2397

35 <1004_1317>, gru2, rmsprop , lr=0.001 , mae no info yet

0.2674 at 20/35 ,..... test: 0.2384

36 <2204_1845>, gggdd, rmsprop , lr=0.001 , mae u:32,64,64 , d:64,128, drop 0.2

0.2681 at 13/20 ,..... test: 0.238

37 <1504_1705>, grugru, rmsprop , lr=0.001 , mae u1:32, u2:64 , d1:64

0.2598 at 17/25 ,..... test: 0.2352

38 <2204_1731>, gggdd, rmsprop , lr=0.001 , mae u:32,64,64 , d:64,128

0.2667 at 9/40 ,..... test: 0.2326

39 <1204_0858>, ggdd, rmsprop , lr=0.001 , mae u1:32:64 , d:64,128

0.2629 at 21/25 ,..... test: 0.2315

7. Reflections about deep Learning

This section contains some information about the effect of deep learning on society , environment and employment.

7.1. Starting with Cholets list of Deep Learning achievements:

Some extracts from the course book (Cholet 2017 , ch1)

Deep Learning gets remarkable results on perceptual problems such as seeing and hearing—problems , In particular,
Near-human-level image classification
Near-human-level speech recognition
Near-human-level handwriting transcription
Improved machine translation
Improved text-to-speech conversion
Digital assistants such as Google Now and Amazon Alexa
Near-human-level autonomous driving
Improved ad targeting, as used by Google, Baidu, and Bing
Improved search results on the web
Ability to answer natural-language questions
Superhuman Go playing

7.2. My own experience

I did not reflect about machine learning or deep learning before taking this course and have just few own reflections about the subject:

google search is extremely good to find the relevant site looking for technical information (computing tips …)

google translate is quite useful even if it does not match a good old fashion dictionary (Websters , Larousse)

you tube has a list up next that I guess is the result of deep learning algo

So I used google search to find information about the influence of deep learning in society , jobs and environment an will give now some links

7.3. general reflections

Machine learning

https://royalsociety.org/topics-policy/projects/machine-learning/

The Royal Society's programme of work on machine learning has been investigating the potential of this technology over the next 5-10 years, and the barriers to realising that potential.

I look at some of interactive infographic in particular exaplore developmentns in machine learning AI machine learnign in the world around you:

this I did not know about:

Further developments in retail could include highly automated shopping experiences, such as those being developed by Amazon, in which shoppers and their product selections are automatically detected as they move through the store, and charges made automatically. This uses a combination of sensor technology and machine learning, which supports the computer vision needed to follow what shoppers are buying.

I also look partly at the that video "You and AI – The Practical Applications of AI"

https://www.youtube.com/watch?v=7L7bywQSTig

some subjects presented by the speakers are:

VPA virtual personal assistant , ASR automatic speech recognition ,

machine learning in astronomy , AI and the changing landscape of weather and climate risk

7.4. big brothers and co

Godfather of deep learning: Chinas 'AI tech is 1984 bigbrother

https://futurism.com/the-byte/godfather-deep-learning-china-1984-big-brother

Yoshua Bengio, a computer scientist who's widely considered one of the three "godfathers" of deep learning, opened up in a new interview with Bloomberg about his **anxiety about how China is using the technology he helped develop to surveil and manipulate large populations.**

find you in CCTV footage without using face recognition

https://www.rt.com/news/442123-ai-cctv-search-engine/

At the moment, surveillance footage must be searched manually to find a person based on common descriptors like clothing and gender. These identifying factors are known as soft biometrics.

...

The team used deep learning - which goes beyond machine learning (where patterns are set into algorithms and require supervision) by incorporating 'self-learning'- **to train a convolutional neural network (CNN) to recognize soft biometrics using computer vision.**

...

Piglet's Big Brother: How AI Changes Agriculture

https://u.today/piglets-big-brother-how-ai-changes-agriculture

Throughout the history of time, scientists and farmers have been working together, searching for ways to advance agricultural practices by utilizing cutting-edge technology to increase yields and profit, while ensuring crops and animals are happy and healthy. Now, we're seeing the use of deep learning and specifically computer vision emerging in agriculture, revolutionizing the industry that feeds billions.

….

7.5. The end of the age of humans

https://becominghuman.ai/the-end-of-the-age-of-humans-384a1756ee5d

This article name some main fears related to machine learning ,robotisation and deep learning :

We will be watched constantly , Robots will take our jobs , Algorithm errors will spark chaos

We will become half-robot, half-human

7.6. Computing and jobs

Robotic process automation

https://en.wikipedia.org/wiki/Robotic_process_automation

Robotic process automation (or RPA) is an emerging form of business process automation technology based on the notion of software robots or artificial intelligence (AI) workers.[1]

In traditional workflow automation tools, a software developer produces a list of actions to automate a task and interface to the back-end system using internal application programming interfaces (APIs) or dedicated scripting language. In contrast, RPA systems develop the action list by watching the user perform that task in the application's graphical user interface (GUI), and then perform the automation by repeating those tasks directly in the GUI. This can lower the barrier to

use of automation in products that might not otherwise feature APIs for this purpose.

Ai interns: Software already taking jobs from humans

https://www.newscientist.com/article/mg22630151.700-ai-interns-software-already-taking-jobs-from-humans/#.VY2CxPlViko

TECHNOLOGY NEWS 31 March 2015

People have talked about robots taking our jobs for ages. Problem is, they already have – we just didn't notice

I like this article , it presents

ROSS , automate a whole chunk of the legal research normally carried out by entry-level paralegals.

Blue Prism, a start-up which developed O2's artificial workers.

and soem short reflection about the future

7.7. Environment and co

Forecasting Air Pollution with Recurrent Neural Networks

https://towardsdatascience.com/forecasting-air-pollution-with-recurrent-neural-networks-ffb095763a5c

This study is similar to my own lab (weather prediction) and therefore it can be fun to compare with .

Data science competitions to build a better world

https://www.drivendata.org/

DrivenData works on projects at the intersection of data science and social impact, in areas like international development, health, education, research and conservation, and public services. We want to give more organizations access to the capabilities of data science, and engage more data scientists with social challenges where their skills can make a difference.

We've worked with more than 35 organizations across 50+ projects, many of these made possible by the amazing efforts of the DrivenData community. Check out some examples below!

7.8. other interesting sites

Big Brother is Biased

https://www.cmdagency.com/blog-big-brother-is-biased/

Even the most logically constructed, objective algorithm is learning from the biases—from ideology to zip code to race to gender—built into our world and hence into the data being generated. This perpetuates the biases, now codified as facts. **These skewed results then have the potential to unintentionally become the "gold insight" driving decision-making in business and industry.**

...

So why do we care as an agency? We want to help our technology sector clients to address the ethical and societal questions that technology innovation can unleash. This is territory newly resonant for the tech industry and we want to face it head on.

...

xxxxxxxxxxxxx end of document