Boston

Beate and Oliver

1/16/2020

Goal of this script

We want to implement linear transformation models in NN and compare the achieved NLL and estimated ceofficients with the MLT results.

We fit a transformation function $h : (y|x) \to (z|x)$ with the property $(z|x) = h(y|x) \sim N(0,1)$

In a **linear** transformation model the transformation function has the special form: $h_Y(y) - \sum_i \beta_i x_i$

Then we know, that.

- $F_{Y|X=x}(y) = F_z(h_Y(y) - \sum_i \beta_i x_i)$

Importing the required packages

```
library(MASS)
library(ggplot2)
```

```
## Warning: As of rlang 0.4.0, dplyr must be at least version 0.8.0.
## x dplyr 0.7.6 is too old for rlang 0.4.2.
## i Please update dplyr with `install.packages("dplyr")`.
```

```
library(mlt)
```

```
## Warning: package 'mlt' was built under R version 3.5.2

## Loading required package: basefun

## Warning: package 'basefun' was built under R version 3.5.2

## Loading required package: variables

## Warning: package 'variables' was built under R version 3.5.2

##
## Attaching package: 'variables'

## The following object is masked from 'package:ggplot2':
##
##     unit
```

```
library(basefun)
library(keras)
library(tensorflow)
```

```
## Warning: package 'tensorflow' was built under R version 3.5.2
```

```
library(tfprobability)
```

```
## Warning: package 'tfprobability' was built under R version 3.5.2
```

```
T_STEPS = 25000
T_OUT = 5000
```

Source functions h and h_dash in w and w/o batch magic

```
# source("mlt_utils.R")  # eg scaling fct
# # preparing eval_h an eval_h_dash, fct implemented in tfp
# source("mlt_utils_keras_v2.R")  # causes error when knittering
#source('https://raw.githubusercontent.com/tensorchiefs/dl_playr/master/mlt/bern_utils.R')
#source('~/Documents/workspace/dl_playr/mlt/bern_utils.R')
source('bern_utils.R')
source('data.R')
```

Loading the data

We scale the y-varible to [0,1]

```
xy_dat = get_data_boston()
```

```
##  [1] "Names in X : crim"    "Names in X : zn"      "Names in X : indus"
##  [4] "Names in X : chas"    "Names in X : nox"     "Names in X : rm"
##  [7] "Names in X : age"     "Names in X : dis"     "Names in X : rad"
## [10] "Names in X : tax"     "Names in X : ptratio" "Names in X : b"
## [13] "Names in X : lstat"
```

```
dat = xy_dat$dat
dat$y_obs = dat$y
dat$y = NULL
y_range = xy_dat$scale
dat$y_scale = dat$y_obs
dat$y_obs = NULL

x = xy_dat$x
y = xy_dat$y

x = x[,'rm',drop=FALSE]
```

Defining the model

We set up the formula for the model:

```
#fm_large = (y_scale ~ crim + zn + indus + chas + nox + rm + age + dis + rad + tax + ptratio + b + lsta
#fm_small = (y_scale ~  rm + lstat) #lm log lik 346
fm_uni = (y_scale ~  rm)
(fm = fm_uni)
```

```
## y_scale ~ rm
```

```
#is_univariate = TRUE
sum(dat$rm**2)  # 20234.6 to compare with BH data in paper
```

```
## [1] 20234.6
```

Baseline Linear Model

```

fit_lm = lm(fm, data=dat)
fit_lm$coef
```

```
## (Intercept)          rm
##  -0.8815694   0.2022691
```

```
NLL_LM = logLik(fit_lm) /nrow(dat) + log(y_range)# the smaller the better
NLL_LM
```

```
## 'log Lik.' 4.306852 (df=3)
```

MLT fit and results

Variable and Model definition and fit

```
nb = 1  # order defining the Number of Bernstein fct in polynom
len_theta = nb+1
# specify a numeric variable with data in [0,1] and principle bounds [0,Inf]
var_y <- numeric_var("y_scale", support = c(0, 1), bounds = c(-Inf, Inf), add = c(0,0))
# what is done with the bound information (default bounds c(-INF, INF)

# set up monoton increasing polynomial of order nb with Bernstein basis function
bb <- Bernstein_basis(var_y, order=nb, ui="increasing")

# set up grid in interval supp+add -> gives data.frame with col y_scale
y_grid <- as.data.frame(mkgrid(bb, n = 500))

# set up model for mlt
ctm = ctm(bb, shift=fm[-2L], data=dat, todistr="Normal")
#~-1 + crim
#ctm = ctm(bb, shift = ~ b + crim - 1, data=dat, todistr="Normal")
# fm[-2L] defnes the basis function for the shift term h_y(y) in h(y|x)=h_y(y)+h_x(x)
# the intercept is included in the baseline-trafo h_y(y) (not in linear predictor h_x(x))
```

Fit of the model:

```
# fit the mlt model
mlt_fit <- mlt(ctm, data = dat, verbose=TRUE)
```

logLik with MLT

```
(logLik_mlt = logLik(mlt_fit))  #  df = nr-theta + nr-beta
```

```
## 'log Lik.' 253.0957 (df=3)
```

```
# compare to logLik of the baseline model - the larger the better
NLL_MLT = -logLik_mlt / nrow(dat) + log(y_range)
NLL_MLT
```

```
## 'log Lik.' 3.306473 (df=3)
```

Estimated coefficients with MLT

Get the coefficients of the trafo h from the mlt fit:

```r
( mlt_fit$coef )
```

```
## Bs1(y_scale) Bs2(y_scale)           rm
##     6.007884    12.822891    -1.378463
```

```r
( theta = mlt_fit$coef[1:(nb+1)] )
```

```
## Bs1(y_scale) Bs2(y_scale)
##     6.007884    12.822891
```

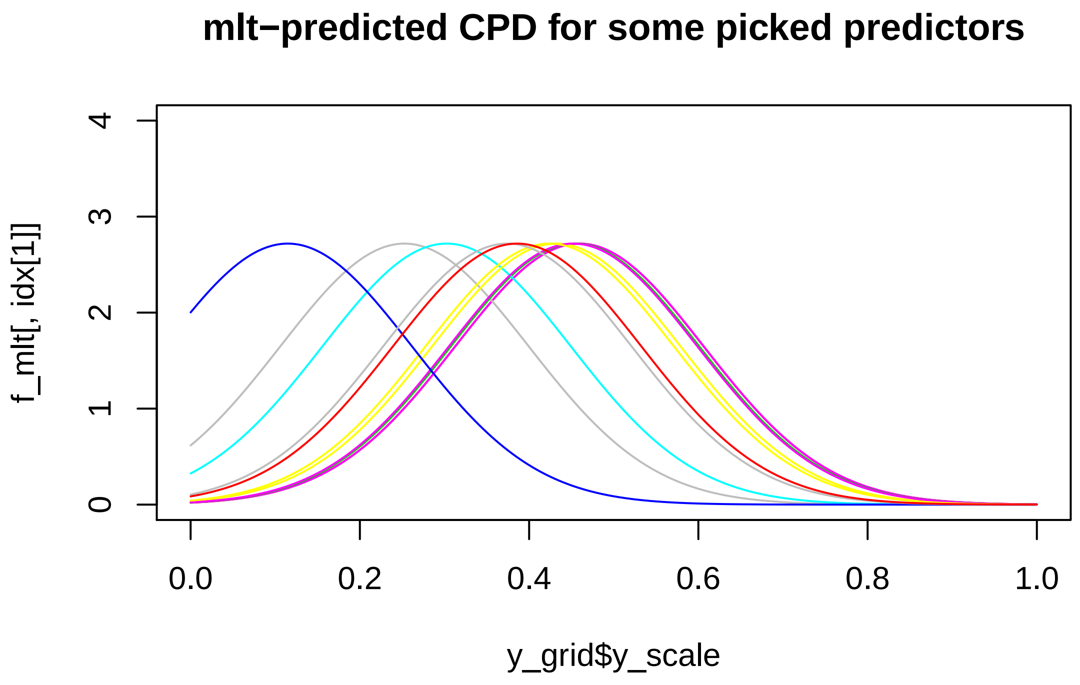
```r
( beta = mlt_fit$coef[(nb+2):length(mlt_fit$coef)] )
```

```
##        rm
## -1.378463
```

The conditional PDF for some observations

```r
  f_mlt = predict(mlt_fit, newdata=dat,  q=y_grid$y_scale, type='density')

  q_mlt = predict(mlt_fit, newdata=dat,
                  prob=c(0.025,0.25,0.5, 0.75,0.975), type='quantile')
  q_mlt = t(q_mlt)
  #q_mlt = matrix(q_mlt$exact, ncol = 5, byrow = TRUE)
  set.seed(3)
  idx = sample(1:ncol(f_mlt))[1:10]
  m = max(f_mlt[,idx])
  plot(y_grid$y_scale, f_mlt[,idx[1]], type='l',col='red', ylim=c(0,4),
       main="mlt-predicted CPD for some picked predictors")
  for (i in idx){
    lines(y_grid$y_scale, f_mlt[,i], col=i)
  }
```



NN

NN approach for a linear shift model, modeled with NN

Fitting means to find the *nb* coefficients *theta* for the Bernsteinpolynom which approximaties the transformation function with *nb* being set to:

```
nb
```

```
## [1] 1
```

Preparing input and output

```
y = tf$Variable(as.matrix(dat$y_scale)[,drop=FALSE], dtype='float32')
y$shape # has to be (#y,1)
```

```
## (506, 1)
```

```
# conditional - we give the rm-variables as input to the NN
#x = tf$Variable(as.matrix(dat$rm)[,drop=FALSE], dtype='float32')

#x = tf$Variable(as.matrix(dat[,c('rm','lstat'),drop=FALSE]), dtype='float32')

#dat$chas = as.numeric(as.character(dat$chas))
x = tf$Variable(x, dtype='float32') #all
x$shape  # has to be (#y,1) for a univariate model
```

```
## (506, 1)
```

```
source('model_3.R')
source('bern_utils.R')
source("model_utils.R")
x_dim = as.integer(dim(x)[2])
model_3 = new_model_3(len_theta = as.integer(len_theta), x_dim = x_dim, y_range=y_range)
run = 1
history = model_train(model_3, make_hist(), x_train = x, y_train = y,
                      x_test = x, y_test = y, T_STEPS=T_STEPS)
```

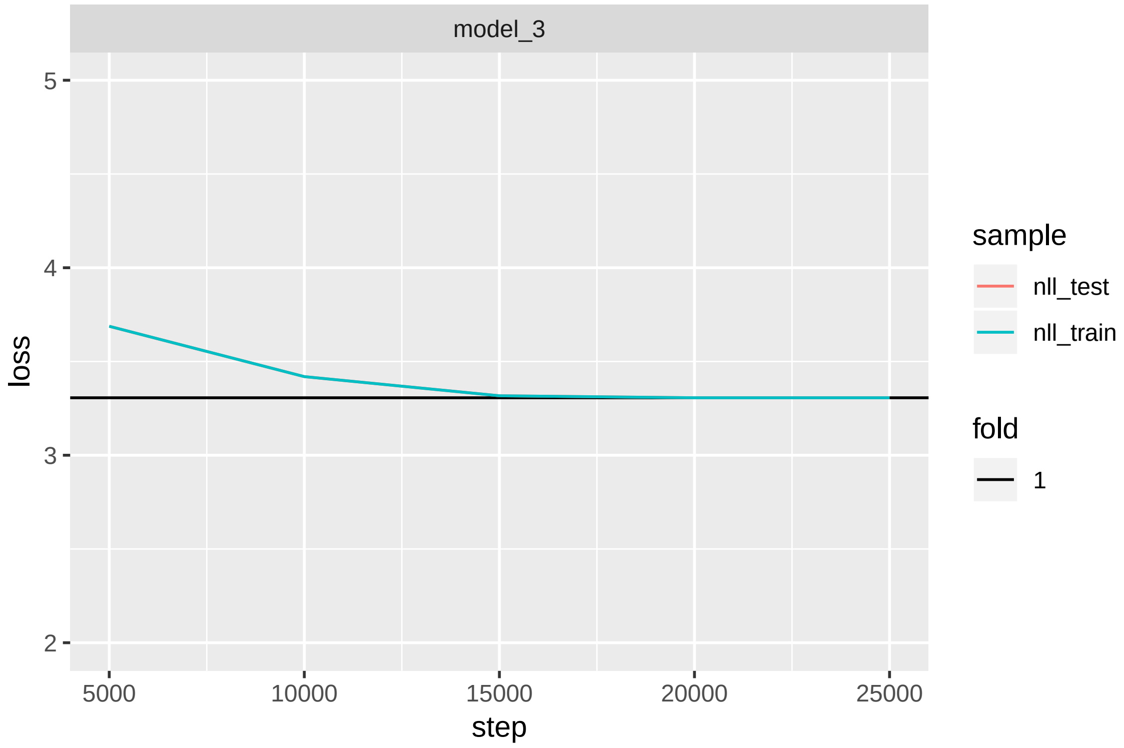
```
## [1] "5000 model_3: likelihood (in optimize)  3.68786811828613 likelihood (in test)  3.68778157234192
## [1] "10000 model_3: likelihood (in optimize)  3.41925096511841 likelihood (in test)  3.4192166328430
## [1] "15000 model_3: likelihood (in optimize)  3.31706285476685 likelihood (in test)  3.3170549869537
## [1] "20000 model_3: likelihood (in optimize)  3.30646777153015 likelihood (in test)  3.3064675331115
## [1] "25000 model_3: likelihood (in optimize)  3.30646753311157 likelihood (in test)  3.3064675331115
```

```
history$step = as.integer(history$step)
history$fold = as.integer(history$fold)
history$nll_train = as.numeric(history$nll_train)
history$nll_test = as.numeric(history$nll_test)
history$OK = NULL# = as.numeric(history$OK)

library(tidyr)
h = gather(history, 'sample', 'loss', nll_train:nll_test)
h$loss = as.numeric(h$loss)
h$sample = as.factor(h$sample)
h$fold = as.factor(h$fold)
```

```
hh =h[!is.na(h$loss),]

ggplot(hh, aes(x=step,y=loss, color=sample, linetype=fold)) +
ylim(2,5) + geom_hline(yintercept=NLL_MLT)+  geom_line() + facet_grid(. ~ method)
```



Compare NN model to MLT model

Get beta coefficients

```
( beta_nn = as.numeric( model_3$model_beta$get_weights()[[1]]) )
```

```
## [1] 0.9685355
```

```
 ( beta_mlt = mlt_fit$coef[(len_theta+1):(len_theta+ncol(x))] )
```

```
##        rm
## -1.378463
```

```
  beta_nn/beta_mlt
```

```
##         rm
## -0.7026201
```

Get theta coefficients

```
  one = tf$ones(shape = c(1,1))
  ( theta_nn = to_theta(model_3$model_hy(one)) )
```

```
## tf.Tensor([[-2.655225  4.159855]], shape=(1, 2), dtype=float32)
```

```
  ( theta_mlt = mlt_fit$coef[1:len_theta] )
```

```
## Bs1(y_scale) Bs2(y_scale)
##     6.007884    12.822891
```

```
  theta_nn$numpy()/theta_mlt
```

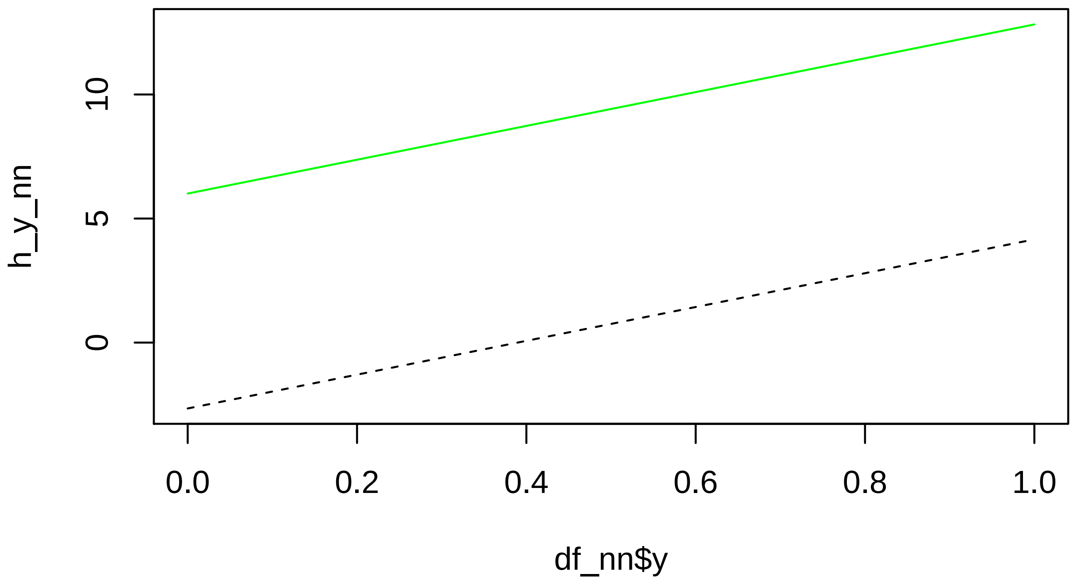
```
##            [,1]      [,2]
## [1,] -0.4419568 0.3244085
```

predict baseline trafo (first part of trafo w/o shift)

```
#nn
out_row = model_3$model_hy(one) #Pick row and compute CPD
df_nn = bernp.p_y_h(model_3$bernp, out_row, from = 0, to = 1, length.out = length(y_grid$y_scale))
h_y_nn = df_nn$h

# mlt
h_y_mlt = predict(bb, newdata = y_grid, coef = theta_mlt, type='trafo')


plot(df_nn$y, h_y_nn,  type='l', lty=2, ylim=c(min(h_y_nn,h_y_mlt), max(h_y_nn,h_y_mlt)))
lines(y_grid$y_scale, h_y_mlt, type='l',col='green')
```

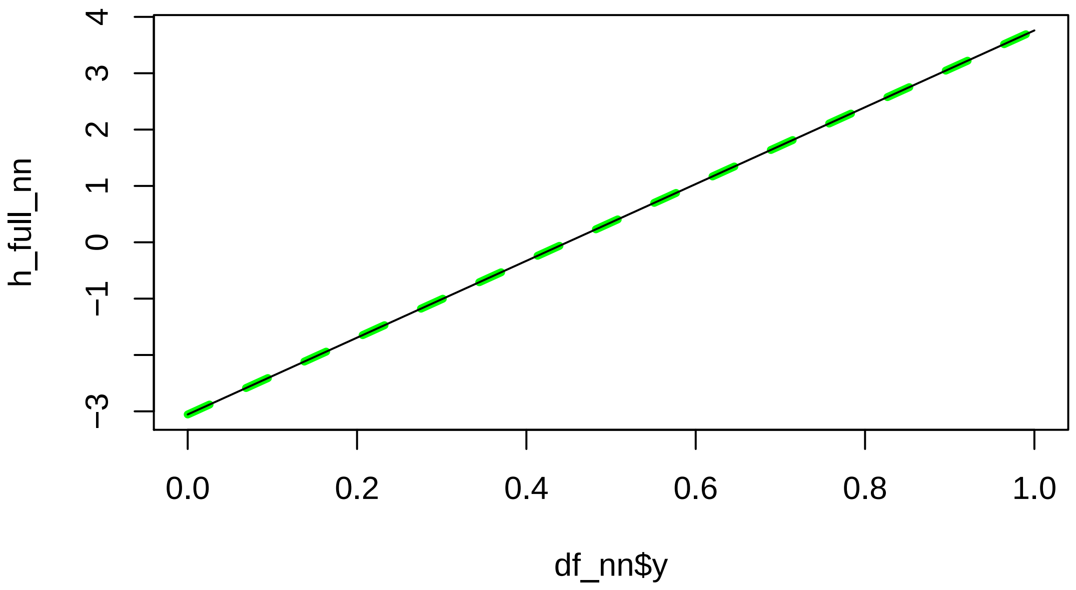


Predict full trafo (all parts of trafo inclusive shift) for picked observation

```
# nn
pick_idx = 1
shift = beta_nn %*% x[pick_idx,1:length(beta_nn)]$numpy()
out_row = model_3$model_hy(one) #Pick row and compute CPD
df_nn = bernp.p_y_h(model_3$bernp, out_row, from = 0, to = 1, length.out = 100, out_eta = shift)
h_full_nn = df_nn$h
```

```r
# mlt:
h_full_mlt = predict(mlt_fit, newdata = dat[pick_idx,], q=y_grid$y_scale, type='trafo')

plot(df_nn$y, h_full_nn, type='l',lty=2,lwd=4, col='green')
lines(y_grid$y_scale, h_full_mlt)
```



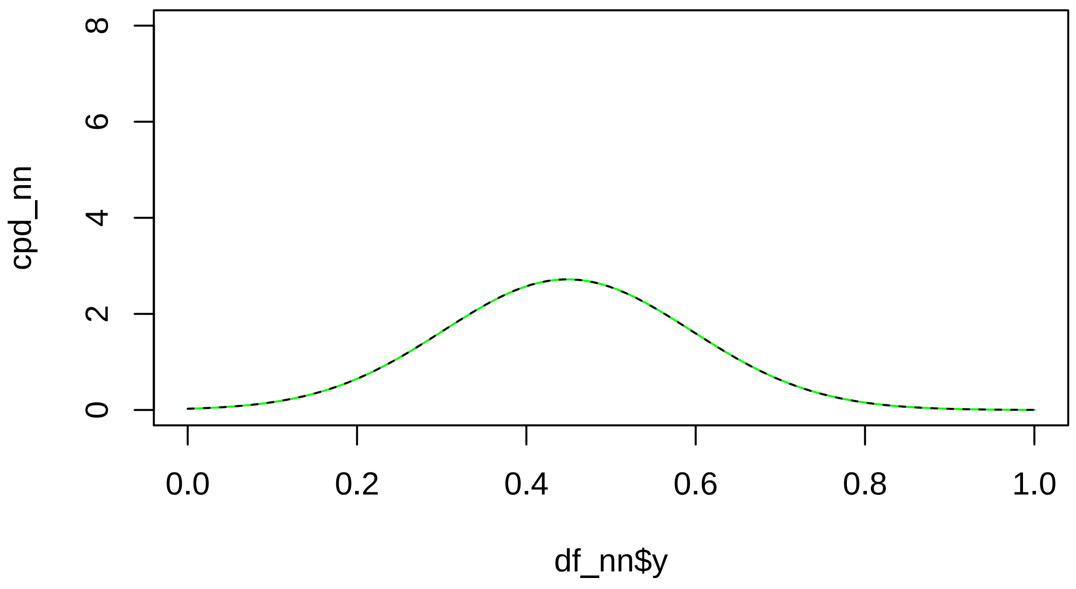
Predict CPD for picked observations

```r
# NN

cpd_nn = df_nn$p_y

# mlt:
cpd_mlt = predict(mlt_fit, newdata = dat[pick_idx,], q=y_grid$y_scale, type='density')


plot(df_nn$y, cpd_nn, ylim=c(0,8), type="l", col='green')
lines(y_grid$y_scale, cpd_mlt, lty=2)
```


cpd_nn
df_nn$y
0
2
4
6
8
0.0
0.2
0.4
0.6
0.8
1.0