Generalized Linear Models

Mario V. Wüthrich
RiskLab, ETH Zurich

"Deep Learning with Actuarial Applications in R"
Swiss Association of Actuaries SAA/SAV, Zurich
October 14/15, 2021

Programme SAV Block Course

- Refresher: Generalized Linear Models (THU 9:00-10:30)
- Feed-Forward Neural Networks (THU 13:00-15:00)
- Discrimination-Free Insurance Pricing (THU 17:15-17:45)

- LocalGLMnet (FRI 9:00-10:30)
- Convolutional Neural Networks (FRI 13:00-14:30)
- Wrap Up (FRI 16:00-16:30)

Contents: Generalized Linear Models

- Starting with data
- Exponential dispersion family (EDF)
- Generalized linear models (GLMs)
- Maximum likelihood estimation (MLE)
- Canonical link and the balance property
- Covariate pre-processing / feature engineering
- Parameter selection

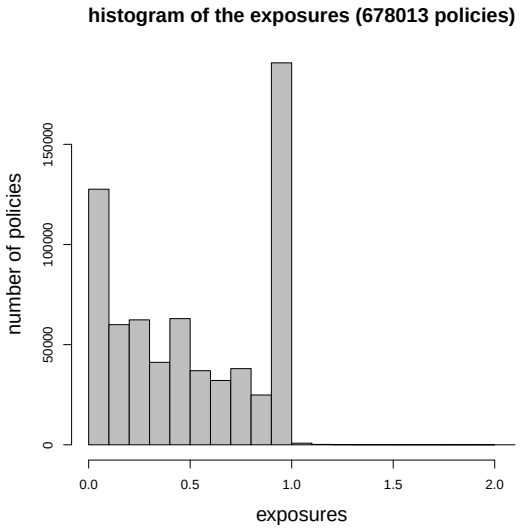
- **Starting with Data**

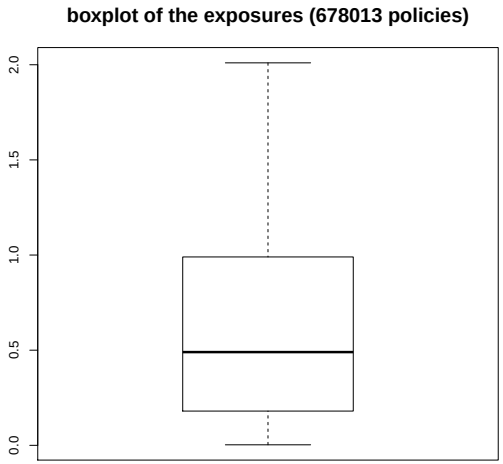
Car Insurance Claims Frequency Data

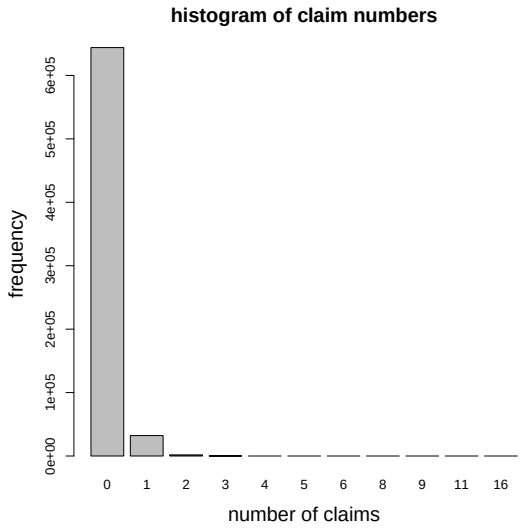
```
'data.frame':   678013 obs. of  12 variables:
 $ IDpol      : num  1 3 5 10 11 13 15 17 18 21 ...
 $ ClaimNb    : num  1 1 1 1 1 1 1 1 1 1 ...
 $ Exposure   : num  0.1 0.77 0.75 0.09 0.84 0.52 0.45 0.27 0.71 0.15 ...
 $ Area       : Factor w/ 6 levels "A","B","C","D",..: 4 4 2 2 2 5 5 3 3 2 ...
 $ VehPower   : int  5 5 6 7 7 6 6 7 7 7 ...
 $ VehAge     : int  0 0 2 0 0 2 2 0 0 0 ...
 $ DrivAge    : int  55 55 52 46 46 38 38 33 33 41 ...
 $ BonusMalus: int  50 50 50 50 50 50 50 68 68 50 ...
 $ VehBrand   : Factor w/ 11 levels "B1","B10","B11",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ VehGas     : Factor w/ 2 levels "Diesel","Regular": 2 2 1 1 1 2 2 1 1 1 ...
 $ Density    : int  1217 1217 54 76 76 3003 3003 137 137 60 ...
 $ Region     : Factor w/ 22 levels "R11","R21","R22",..: 18 18 3 15 15 8 8 20 20 12
```

- 3 categorical covariates, 1 binary covariate and 5 continuous covariates
- Goal: Find systematic effects to explain/predict claim counts `ClaimNb`.

Exposures and Claims

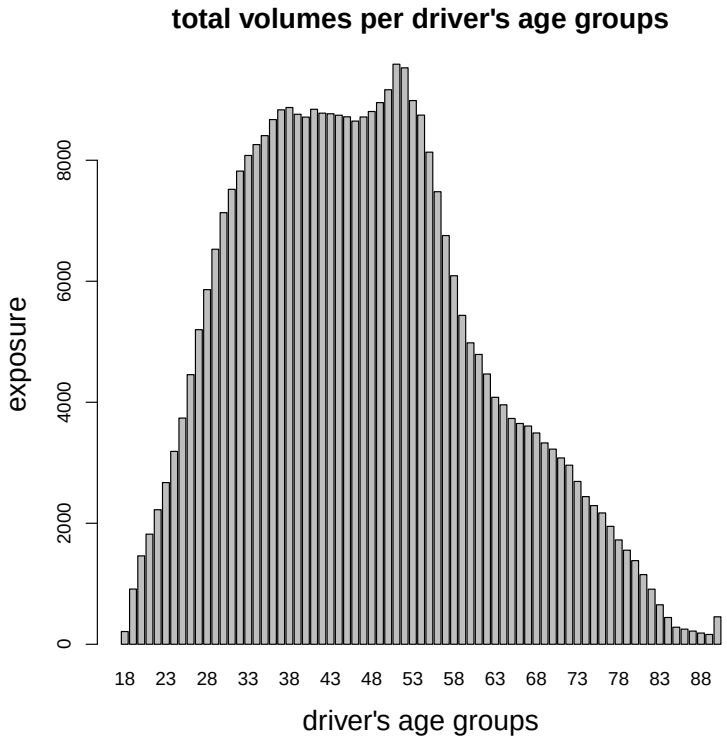


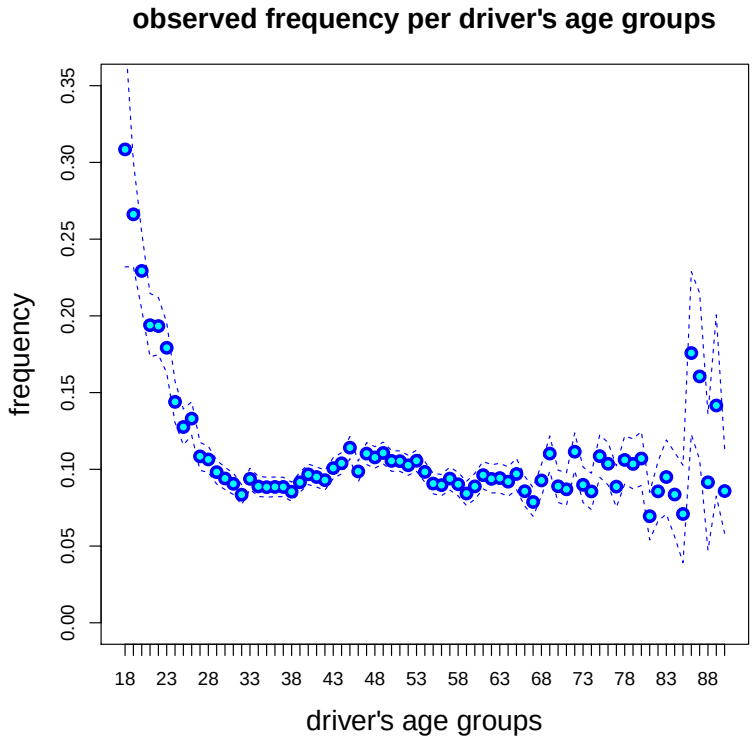




- Most exposures are between 0 and 1 year.
- Exposures bigger than 1 are considered to be data error and are capped at 1.
- Most insurance policies do not suffer any claim (class imbalance problem).

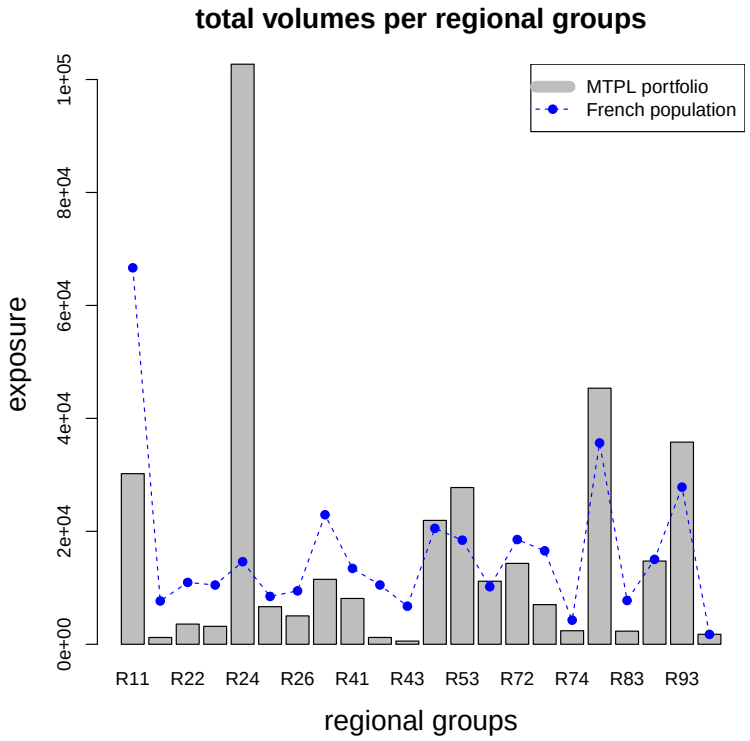
Continuous Covariates: Age of Driver

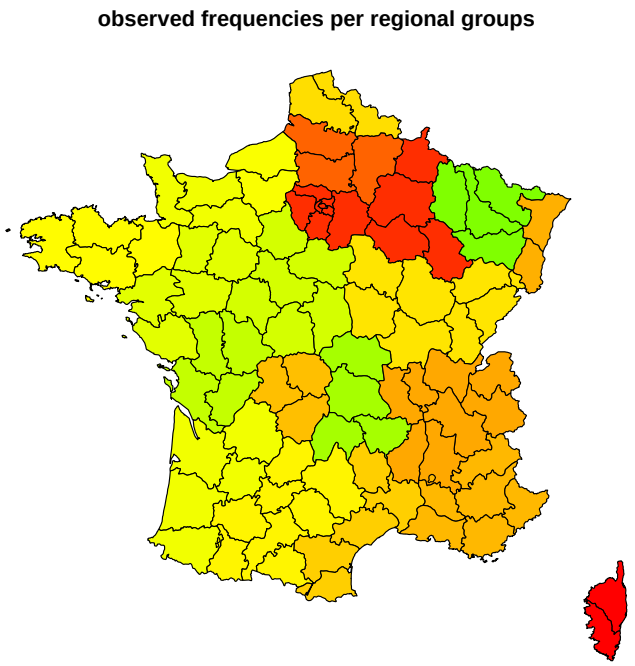




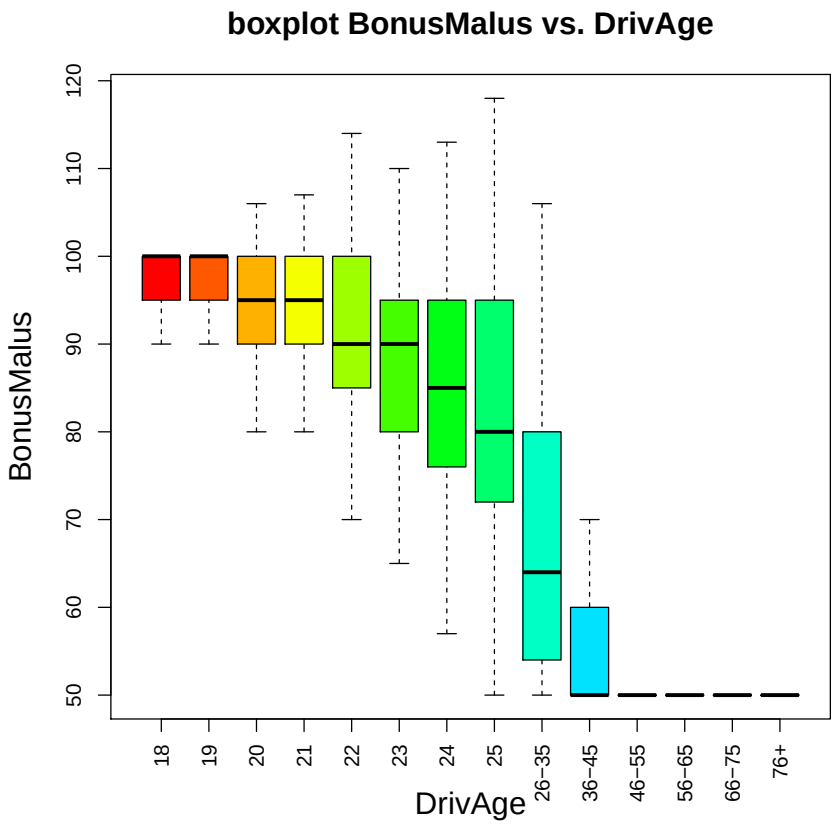
- Systematic effects of continuous covariates are not necessarily monotone.

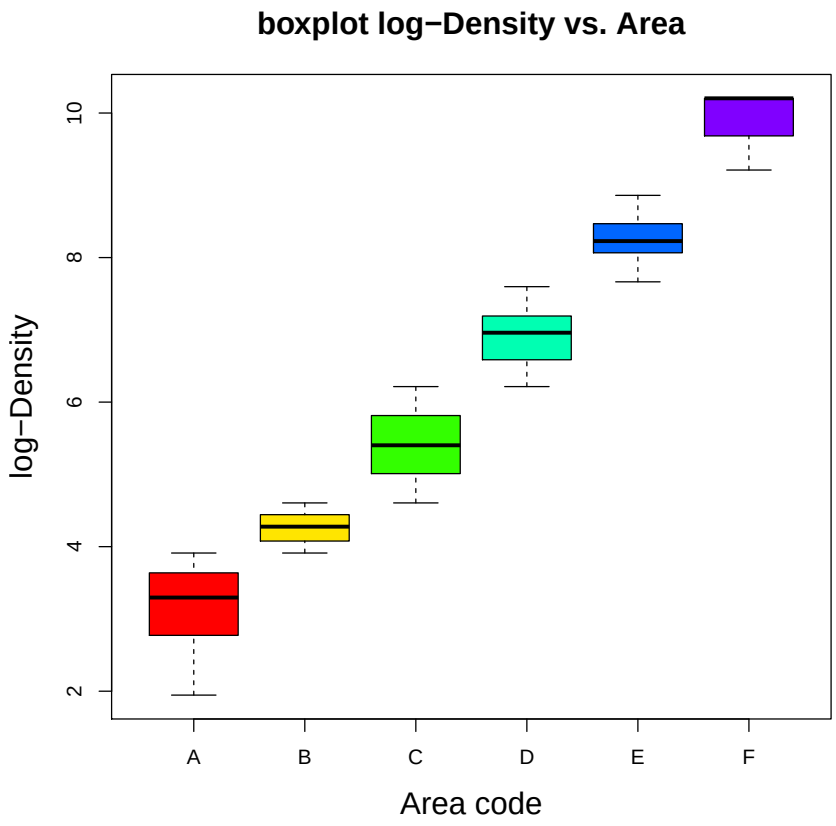
Categorical Covariates: French Region





Covariates: Dependence





- These covariates show strong dependence/collinearity.

Goal: Regression Modeling

- Denote by $\boldsymbol{x}_i$ the covariates of insurance policy $1 \leq i \leq n$.
- **Goal:** Find regression function μ:

$$\boldsymbol{x}_i \mapsto \mu(\boldsymbol{x}_i),$$

such that for all insurance policies $1 \leq i \leq n$ we have

$$\mathbb{E}[N_i] = \mu(\boldsymbol{x}_i) v_i,$$

where N_i denotes the number of claims and $v_i > 0$ is the time exposure of insurance policy $1 \leq i \leq n$ (pro-rata temporis).

- μ extracts the systematic effects from information $\boldsymbol{x}_i$ to explain N_i.

- **Exponential Dispersion Family (EDF)**

Exponential Dispersion Family (EDF)

- Sir Fisher (1934), Barndorff-Nielsen (2014), Jørgensen (1986, 1987).
- Exponential dispersion family (EDF) gives a unified notational framework of a large family of distribution functions.
- The parametrization of this family is chosen such that it is particularly suitable for maximum likelihood estimation (MLE).
- The EDF is the base statistical model for generalized linear modeling (GLM) and for neural network regressions.
- Examples: Gaussian, Poisson, gamma, binomial, categorical, Tweedie's, inverse Gaussian models.

- Remark: This first chapter on GLMs gives us the basic understanding and tools for neural network regression modeling.

Exponential Dispersion Family (EDF)

- Assume $(Y_i)_i$ are independent with density

$$Y_i \sim f(y;\theta_i,v_i/\varphi)=\exp\left\{\frac{y\theta_i-\kappa(\theta_i)}{\varphi/v_i}+a(y;v_i/\varphi)\right\},$$

with

$v_i>0$	(known) exposure of risk i,
$\varphi>0$	dispersion parameter,
$\theta_i\in\boldsymbol{\Theta}$	canonical parameter of risk i in the effective domain $\boldsymbol{\Theta}$,
$\kappa:\boldsymbol{\Theta}\to\mathbb{R}$	cumulant function (type of distribution),
$a(\cdot;\cdot)$	normalization, *not* depending on the canonical parameter θ_i.

Cumulant Function

- Assume $(Y_i)_i$ are independent with density

$$Y_i \sim f(y; \theta_i, v_i/\varphi) = \exp\left\{ \frac{y\theta_i - \kappa(\theta_i)}{\varphi/v_i} + a(y; v_i/\varphi) \right\}.$$

- Cumulant function $\kappa : \boldsymbol{\Theta} \to \mathbb{R}$ is convex and smooth in the interior of $\boldsymbol{\Theta}$.

- Examples:

$$\kappa(\theta) = \begin{cases} \theta^2/2 & \text{Gauss,} \\ \exp(\theta) & \text{Poisson,} \\ -\log(-\theta) & \text{gamma,} \\ \log(1+e^\theta) & \text{Bernoulli/binomial,} \\ -(-2\theta)^{1/2} & \text{inverse Gaussian,} \\ ((1-p)\theta)^{\frac{2-p}{1-p}}/(2-p) & \text{Tweedie with } p > 1,\ p \neq 2. \end{cases}$$

Mean and Variance Function

- The mean is given by

$$\mu_i = \mathbb{E}[Y_i] = \kappa'(\theta_i).$$

- The variance is given by

$$\mathrm{Var}(Y_i) = \frac{\varphi}{v_i}\kappa''(\theta_i) = \frac{\varphi}{v_i}V(\mu_i) > 0,$$

  where $\mu \mapsto V(\mu) = \kappa''((\kappa')^{-1}(\mu))$ is the so-called variance function.

- Examples:

$$V(\mu) = \begin{cases} 1 & \text{Gauss,} \\ \mu & \text{Poisson,} \\ \mu^2 & \text{gamma,} \\ \mu^3 & \text{inverse Gaussian,} \\ \mu^p & \text{Tweedie with } p \geq 1. \end{cases}$$

Maximum Likelihood Estimation (MLE)

- MLE homogeneous θ case: log-likelihood of independent observations $(Y_i)_{i=1}^n$ is

$$\ell_{\boldsymbol{Y}}(\theta) \;=\; \log\left(\prod_{i=1}^n f(Y_i;\theta,v_i/\varphi)\right) \;=\; \sum_{i=1}^n \frac{Y_i\theta-\kappa(\theta)}{\varphi/v_i} + a(Y_i;v_i/\varphi).$$

- This provides score equations

$$\frac{\partial}{\partial\theta}\ell_{\boldsymbol{Y}}(\theta) \;=\; \sum_{i=1}^n \frac{v_i}{\varphi}\left[Y_i-\kappa'(\theta)\right]=0,$$

  and MLE $\widehat{\theta}$

$$\widehat{\theta}=(\kappa')^{-1}\left(\frac{\sum_{i=1}^n v_iY_i}{\sum_{i=1}^n v_i}\right).$$

- MLE is straightforward within the EDF!

Canonical Link and Unbiasedness

- Canonical link $h(\cdot) = (\kappa')^{-1}(\cdot)$

$$\mu = \mathbb{E}[Y] = \kappa'(\theta) \qquad \text{or} \qquad h(\mu) = h(\mathbb{E}[Y]) = \theta.$$

- This provides for the MLE

$$\widehat{\theta} \;=\; (\kappa')^{-1}\left(\frac{\sum_{i=1}^n v_i Y_i}{\sum_{i=1}^n v_i}\right) \;=\; h\left(\frac{\sum_{i=1}^n v_i Y_i}{\sum_{i=1}^n v_i}\right).$$

  The latter gives a sufficient statistics.

- Unbiasedness of estimated means in the homogeneous case

$$\mathbb{E}\left[\widehat{\mathbb{E}}[Y]\right] = \mathbb{E}\left[\kappa'(\widehat{\theta})\right] = \kappa'(\theta).$$

  ▷ Unbiasedness emphasizes that we receive the right price level in pricing.

- **Generalized Linear Models (GLMs)**

Generalized Linear Models (GLMs)

- Nelder–Wedderburn (1972) and McCullagh–Nelder (1983).
- Assume we have heterogeneity between $(Y_i)_{i=1}^n$ which manifests in systematic effects modeled through covariates/features $\boldsymbol{x}_i \in \mathbb{R}^q$.
- Assume for link function choice g and regression parameter $\boldsymbol{\beta} \in \mathbb{R}^{q+1}$

$$\boldsymbol{x}_i \;\mapsto\; g(\mu_i) = g(\mathbb{E}[Y_i]) = g(\kappa'(\theta_i)) = \beta_0 + \sum_{j=1}^{q} \beta_j x_{i,j}.$$

  This gives a GLM with link function g. Parameter β_0 is called intercept/bias.

- Link g should be monotone and smooth.
- The choice $g = h = (\kappa')^{-1}$ is called canonical link.

Design Matrix

- Assume for link function choice g and regression parameter $\boldsymbol{\beta} \in \mathbb{R}^{q+1}$

$$\boldsymbol{x}_i \;\mapsto\; g(\mu_i) = g(\mathbb{E}[Y_i]) = \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle = \beta_0 + \sum_{j=1}^{q} \beta_j x_{i,j}.$$

- The design matrix is

$$\mathfrak{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^\top = \begin{pmatrix} 1 & x_{1,1} & \cdots & x_{1,q} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n,1} & \cdots & x_{n,q} \end{pmatrix} \in \mathbb{R}^{n \times (q+1)}.$$

- The design matrix $\mathfrak{X}$ is assumed to have full rank $q + 1 \leq n$.

- Full rank property is important for uniqueness of MLE of $\boldsymbol{\beta}$.

Maximum Likelihood Estimation of GLMs

- The log-likelihood of independent observations $(Y_i)_{i=1}^n$ is given by

$$\boldsymbol{\beta} \;\mapsto\; \ell_{\boldsymbol{Y}}(\boldsymbol{\beta}) \;=\; \sum_{i=1}^{n} \frac{Y_i h(\mu_i) - \kappa(h(\mu_i))}{\varphi/v_i} + a(Y_i; v_i/\varphi),$$

  with mean $\mu_i = \mu_i(\boldsymbol{\beta}) = g^{-1}\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle$ and canonical parameter $\theta_i = h(\mu_i)$.

- This provides score equations for MLE

$$\nabla_{\boldsymbol{\beta}}\, \ell_{\boldsymbol{Y}}(\boldsymbol{\beta}) \;=\; 0.$$

- Score equations are solved numerically with Fisher's scoring method or the iterated re-weighted least squares (IRLS) algorithm.

MLE and Deviance Loss Functions

- The log-likelihood of independent observations $(Y_i)_{i=1}^n$ is given by

$$\ell_{\boldsymbol{Y}}(\boldsymbol{\beta}) \;=\; \sum_{i=1}^{n} \frac{Y_i h(\mu_i) - \kappa(h(\mu_i))}{\varphi/v_i} + a(Y_i; v_i/\varphi),$$

  with mean $\mu_i = \mu_i(\boldsymbol{\beta}) = g^{-1}\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle$.

- Maximizing log-likelihoods is equivalent to minimizing deviance losses

$$\begin{aligned} D^*(\boldsymbol{Y}, \boldsymbol{\beta}) &= 2\left[\ell_{\boldsymbol{Y}}(\boldsymbol{Y}) - \ell_{\boldsymbol{Y}}(\boldsymbol{\beta})\right] \\ &= 2\sum_{i=1}^{n} \frac{v_i}{\varphi}\Big[Y_i h(Y_i) - \kappa(h(Y_i)) - Y_i h(\mu_i) + \kappa(h(\mu_i))\Big] \;\geq\; 0. \end{aligned}$$

- The deviance loss of the Gaussian model is the square loss function, other examples of the EDF have deviance losses *different* from square losses.

Examples of Deviance Loss Functions

- Gaussian case:

$$D^*(\boldsymbol{Y}, \boldsymbol{\beta}) = \sum_{i=1}^{n} \frac{v_i}{\varphi} (Y_i - \mu_i)^2 \ \geq \ 0.$$

- Gamma case:

$$D^*(\boldsymbol{Y}, \boldsymbol{\beta}) = 2 \sum_{i=1}^{n} \frac{v_i}{\varphi} \left(\frac{Y_i}{\mu_i} - 1 + \log \left(\frac{\mu_i}{Y_i} \right) \right) \ \geq \ 0.$$

- Inverse Gaussian case:

$$D^*(\boldsymbol{Y}, \boldsymbol{\beta}) = \sum_{i=1}^{n} \frac{v_i}{\varphi} \frac{(Y_i - \mu_i)^2}{\mu_i^2 Y_i} \ \geq \ 0.$$

- Poisson case:

$$D^*(\boldsymbol{Y}, \boldsymbol{\beta}) = 2 \sum_{i=1}^{n} \frac{v_i}{\varphi} \left(\mu_i - Y_i - Y_i \log \left(\frac{\mu_i}{Y_i} \right) \right) \ \geq \ 0.$$

Balance Property under Canonical Link

- Under the canonical link $g = h = (\kappa')^{-1}$ we have balance property for the MLE

$$\sum_{i=1}^{n} v_i \widehat{\mathbb{E}}[Y_i] \;=\; \sum_{i=1}^{n} v_i \kappa'\langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x}_i \rangle = \sum_{i=1}^{n} v_i Y_i.$$

  ▷ The estimated model mean over the entire portfolio is unbiased.

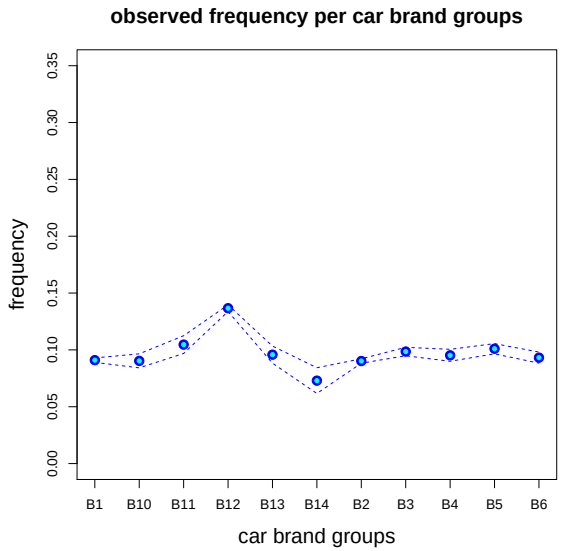
- If one does not work with the canonical link, one should correct in $\widehat{\beta}_0$ for the bias.

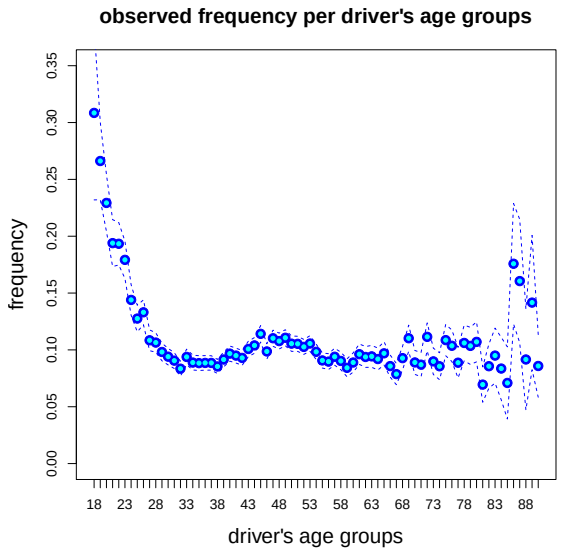
- **Feature Engineering / Covariate Pre-Processing**

Feature Engineering

- Assume monotone link function choice g

$$\boldsymbol{x}_i \;\mapsto\; \mu_i = \mathbb{E}[Y_i] \;=\; g^{-1}\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle = g^{-1}\left(\beta_0 + \sum_{j=1}^{q} \beta_j x_{i,j}\right).$$





- What about categorical covariates and non-monotone covariates?
- What about different interactions?

One-Hot Encoding of Categorical Covariates

B1 $\mapsto \boldsymbol{e}_1 =$	1	0	0	0	0	0	0	0	0	0	0
B10 $\mapsto \boldsymbol{e}_2 =$	0	1	0	0	0	0	0	0	0	0	0
B11 $\mapsto \boldsymbol{e}_3 =$	0	0	1	0	0	0	0	0	0	0	0
B12 $\mapsto \boldsymbol{e}_4 =$	0	0	0	1	0	0	0	0	0	0	0
B13 $\mapsto \boldsymbol{e}_5 =$	0	0	0	0	1	0	0	0	0	0	0
B14 $\mapsto \boldsymbol{e}_6 =$	0	0	0	0	0	1	0	0	0	0	0
B2 $\mapsto \boldsymbol{e}_7 =$	0	0	0	0	0	0	1	0	0	0	0
B3 $\mapsto \boldsymbol{e}_8 =$	0	0	0	0	0	0	0	1	0	0	0
B4 $\mapsto \boldsymbol{e}_9 =$	0	0	0	0	0	0	0	0	1	0	0
B5 $\mapsto \boldsymbol{e}_{10} =$	0	0	0	0	0	0	0	0	0	1	0
B6 $\mapsto \boldsymbol{e}_{11} =$	0	0	0	0	0	0	0	0	0	0	1

- One-hot encoding for the 11 car brands: $\texttt{brand} \mapsto \boldsymbol{e}_{\mathrm{j}} \in \mathbb{R}^{11}$.
- One-hot encoding does not lead to full rank design matrices $\mathfrak{X}$, because we have a redundancy.

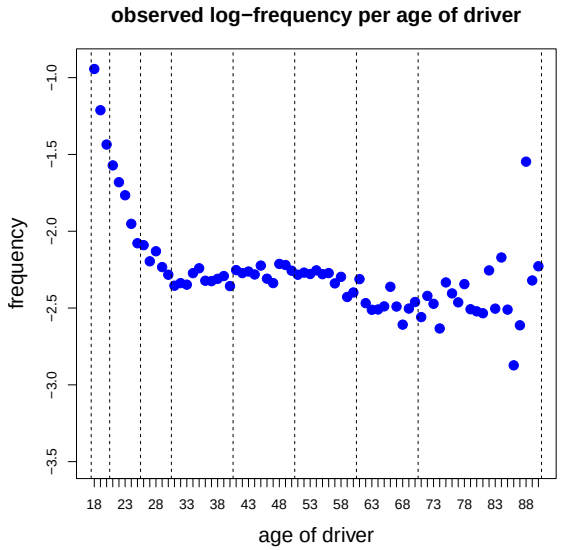
Dummy Coding of Categorical Covariates

B1	0	0	0	0	0	0	0	0	0	0
B10	1	0	0	0	0	0	0	0	0	0
B11	0	1	0	0	0	0	0	0	0	0
B12	0	0	1	0	0	0	0	0	0	0
B13	0	0	0	1	0	0	0	0	0	0
B14	0	0	0	0	1	0	0	0	0	0
B2	0	0	0	0	0	1	0	0	0	0
B3	0	0	0	0	0	0	1	0	0	0
B4	0	0	0	0	0	0	0	1	0	0
B5	0	0	0	0	0	0	0	0	1	0
B6	0	0	0	0	0	0	0	0	0	1

- Declare one label as reference level and drop the corresponding column.
- Dummy coding for the 11 car brands: $\texttt{brand} \mapsto \boldsymbol{x}_j \in \mathbb{R}^{10}$.
- Dummy coding leads to full rank design matrices $\mathfrak{X}$.
- There are other full rank codings like Helmert's contrast coding.

Pre-Processing of Continuous Covariates (1/2)

age class 1:	18-20
age class 2:	21-25
age class 3:	26-30
age class 4:	31-40
age class 5:	41-50
age class 6:	51-60
age class 7:	61-70
age class 8:	71-90



- Continuous features need feature engineering, too, to bring them into the right functional form for GLM. Assume we have log-link for g

$$\boldsymbol{x} \;\mapsto\; \log\left(\mathbb{E}\left[Y\right]\right) = \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle = \beta_0 + \sum_{j=1}^{q} \beta_j x_j.$$

- We build homogeneous categorical classes, and then apply dummy coding.

Pre-Processing of Continuous Covariates (2/2)

- Categorical coding of continuous covariates has some disadvantages.
- By changing continuous features to categorical dummies we lose adjacency relationships between neighboring classes.
- The number of parameters can grow very large if we have many classes.
- Balance property holds true on every categorical level.
  Caution: if we have very rare categorical levels this will lead to over-fitting; and it will also lead to high correlations with the intercept β_0.
- One may also consider other functional forms for continuous covariates, e.g.,

$$\texttt{age} \;\mapsto\; \beta_1 \texttt{age} + \beta_2 \texttt{age}^2 + \beta_3 \log(\texttt{age}).$$

- Similarly, we can model interactions between covariate components

$$(\texttt{age}, \texttt{weight}) \;\mapsto\; \beta_1 \texttt{age} + \beta_2 \texttt{weight} + \beta_3 \texttt{age}/\texttt{weight}.$$

- **Variable Selection**

Variable Selection: Likelihood Ratio Test (LRT)

- **Null hypothesis** H_0: $\beta_1 = \ldots = \beta_p = 0$ for given $1 \leq p \leq q$.
- **Likelihood ratio test (LRT).** Calculate test statistics (nested models)

$$\chi^2_{\boldsymbol{Y}} = D^*(\boldsymbol{Y}, \widehat{\boldsymbol{\beta}}_{H_0}) - D^*(\boldsymbol{Y}, \widehat{\boldsymbol{\beta}}_{\text{full}}) \geq 0.$$

Under H_0, test statistics $\chi^2_{\boldsymbol{Y}}$ is approximately χ^2-distributed with p df.

Variable Selection: Wald Test

- **Null hypothesis** H_0: $\boldsymbol{\beta}_p = (\beta_1, \ldots, \beta_p)^\top = 0$ for given $1 \leq p \leq q$.
- **Wald test.** Choose matrix I_p such that $I_p \boldsymbol{\beta}_{\text{full}} = \boldsymbol{\beta}_p$. Consider Wald statistics

$$W = (I_p \widehat{\boldsymbol{\beta}}_{\text{full}} - 0)^\top \left(I_p \, \mathcal{I}(\widehat{\boldsymbol{\beta}}_{\text{full}})^{-1} \, I_p^\top \right)^{-1} (I_p \widehat{\boldsymbol{\beta}}_{\text{full}} - 0).$$

  Under H_0, test statistics W is approximately χ^2-distributed with p df.

- $\mathcal{I}(\widehat{\boldsymbol{\beta}}_{\text{full}})$ is Fisher's information matrix; the above test is based on asymptotic normality of the MLE $\widehat{\boldsymbol{\beta}}_{\text{full}}$.
- Model only needs to be fitted once.

Model Selection: AIC

- **Akaike's information criterion (AIC)** is useful for non-nested models

$$\text{AIC} = -2\ell_{\boldsymbol{Y}}(\widehat{\boldsymbol{\beta}}) + 2\text{dim}(\boldsymbol{\beta}).$$

- Models do not need to be nested.
- Models can have different distributions.
- AIC considers all terms of the log-likelihood (also normalizing constants).
- Models need to be estimated with MLE.
- Different models need to consider the same data on the same scale (log-normal vs. gamma).

Example: Poisson Frequency GLM

```
Call:
glm(formula = claims ~ powerCAT + area + log(dens) + gas + ageCAT +
                       acCAT + brand + ct, family = poisson(), data = dat, offset =

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.1373  -0.3820  -0.2838  -0.1624   4.3856

Coefficients:
                Estimate Std. Error z value Pr(>!z!)
(Intercept)  -1.903e+00  4.699e-02 -40.509  < 2e-16 ***
powerCAT2     2.681e-01  2.121e-02  12.637  < 2e-16 ***
.
.
powerCAT9    -1.044e-01  4.708e-02  -2.218 0.026564 *
area          4.333e-02  1.927e-02   2.248 0.024561 *
log(dens)     3.224e-02  1.432e-02   2.251 0.024385 *
gasRegular    6.868e-02  1.339e-02   5.129 2.92e-07 ***
.
.
ctZG         -8.123e-02  4.638e-02  -1.751 0.079900 .
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for poisson family taken to be 1)
```

```
26
27     Null deviance: 145532  on 499999  degrees of freedom
28 Residual deviance: 140641  on 499943  degrees of freedom
29 AIC: 191132
```

Forward Parameter Selection: ANOVA

```
Analysis of Deviance Table

Model: poisson, link: log

Response: claims

Terms added sequentially (first to last)


           Df Deviance Resid. Df Resid. Dev
NULL                      499999     145532
acCAT       3  2927.32     499996     142605
ageCAT      7   850.00     499989     141755
ct         25   363.29     499964     141392
brand      10   124.37     499954     141267
powerCAT    8   315.48     499946     140952
gas         1    50.53     499945     140901
area        1   255.20     499944     140646
log(dens)   1     5.07     499943     140641
```

Pay attention: order of covariates inclusion is important.

Backward Parameter Reduction: Drop1

```
Single term deletions

Model:
claims ~ acCAT + ageCAT + ct + brand + powerCAT + gas + area + log(dens)

           Df Deviance    AIC     LRT  Pr(>Chi)
<none>        140641 191132
acCAT       3   142942 193426 2300.61 < 2.2e-16 ***
ageCAT      7   141485 191962  843.91 < 2.2e-16 ***
ct         25   140966 191406  324.86 < 2.2e-16 ***
brand      10   140791 191261  149.70 < 2.2e-16 ***
powerCAT    8   140969 191443  327.68 < 2.2e-16 ***
gas         1   140667 191156   26.32 2.891e-07 ***
area        1   140646 191135    5.06   0.02453 *
log(dens)   1   140646 191135    5.07   0.02434 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

We should keep the full model according to AIC and according to the LRT on a 5% significance level.

- **Car Insurance Frequency Example**

Example: Poisson Frequency Model (1/2)

- The Poisson model has dispersion $\varphi = 1$.
- The Poisson model has cumulant function

$$\theta \;\mapsto\; \kappa(\theta) = \exp(\theta).$$

- Mean and variance of EDFs are given by

$$\mu_i \;=\; \mathbb{E}\left[Y_i\right] \;=\; \kappa'(\theta_i) \;=\; \exp(\theta_i),$$

$$\mathrm{Var}\left(Y_i\right) \;=\; \frac{\varphi}{v_i}\kappa''(\theta_i) \;=\; \frac{1}{v_i}\exp(\theta_i) \;=\; \frac{1}{v_i}\mu_i.$$

  ▷ $N_i = v_i Y_i$ has a Poisson distribution with mean $v_i \mu_i$.

Example: Poisson Frequency Model (2/2)

- Mean of the Poisson model for $N_i = v_i Y_i$

$$v_i \mu_i \;=\; \mathbb{E}\left[N_i\right] \;=\; v_i \kappa'(\theta_i) \;=\; v_i \exp(\theta_i) \;=\; \exp(\log v_i + \theta_i).$$

  The term $\log v_i$ is called offset.

- The Poisson GLM with canonical link $g = h = \log$ is given by

$$\boldsymbol{x}_i \;\mapsto\; \log\left(\mathbb{E}\left[N_i\right]\right) = \log v_i + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle = \log v_i + \beta_0 + \sum_{j=1}^{q} \beta_j x_{i,j}.$$

	run time	# param. $q+1$	AIC	in-sample loss	out-of-sample loss
homogeneous model	–	1	263'143	32.935	33.861
Model GLM1	20s	49	253'062	31.267	32.171

Losses are in 10^{-2}.

Further Points

- To prevent from over-fitting: regularization can be used.
- Ridge regression is based on an L^2-penalization and generally reduces regression parameter components in $\boldsymbol{\beta}$ (exclude the intercept β_0).
- LASSO (least absolute shrinkage and selection operator) regression is based on an L^1-penalization and can set regression parameter components exactly to zero.
- LASSO has difficulties with collinearity in covariate components, therefore, sometimes an elastic net regularization is used which combines ridge and LASSO.
- Regularization has a Bayesian interpretation.
- Generalized additive models (GAMs) allow for more flexibility than GLMs in marginal covariate component modeling. But they often suffer from computational complexity.

References

- Barndorff-Nielsen (2014). Information and Exponential Families: In Statistical Theory. Wiley
- Charpentier (2015). Computational Actuarial Science with R. CRC Press.
- Efron, Hastie (2016). Computer Age Statistical Inference: Algorithms, Evidence, and Data Science. Cambridge UP.
- Fahrmeir, Tutz (1994). Multivariate Statistical Modelling Based on Generalized Linear Models. Springer.
- Fisher (1934). Two new properties of mathematical likelihood. Proceeding of the Royal Society A 144, 285-307.
- Hastie, Tibshirani, Friedman (2009). The Elements of Statistical Learning. Springer.
- Jørgensen (1986). Some properties of exponential dispersion models. Scandinavian Journal of Statistics 13/3, 187-197.
- Jørgensen (1987). Exponential dispersion models. Journal of the Royal Statistical Society. Series B (Methodological) 49/2, 127-145.
- Jørgensen (1997). The Theory of Dispersion Models. Chapman & Hall.
- Lehmann (1983). Theory of Point Estimation. Wiley.
- Lorentzen, Mayer (2020). Peeking into the black box: an actuarial case study for interpretable machine learning. SSRN 3595944.
- McCullagh, Nelder (1983). Generalized Linear Models. Chapman & Hall.
- Nelder, Wedderburn (1972). Generalized linear models. Journal of the Royal Statistical Society. Series A (General) 135/3, 370-384.
- Noll, Salzmann, Wüthrich (2018). Case study: French motor third-party liability claims. SSRN 3164764.
- Ohlsson, Johansson (2010). Non-Life Insurance Pricing with Generalized Linear Models. Springer.
- Wüthrich, Buser (2016). Data Analytics for Non-Life Insurance Pricing. SSRN 2870308, Version September 10, 2020.
- Wüthrich, Merz (2021). Statistical Foundations of Actuarial Learning and its Applications. SSRN 3822407.

Feed-Forward Neural Networks

Mario V. Wüthrich
RiskLab, ETH Zurich

"Deep Learning with Actuarial Applications in R"
Swiss Association of Actuaries SAA/SAV, Zurich
October 14/15, 2021

Programme SAV Block Course

- Refresher: Generalized Linear Models (THU 9:00-10:30)
- Feed-Forward Neural Networks (THU 13:00-15:00)
- Discrimination-Free Insurance Pricing (THU 17:15-17:45)

- LocalGLMnet (FRI 9:00-10:30)
- Convolutional Neural Networks (FRI 13:00-14:30)
- Wrap Up (FRI 16:00-16:30)

Contents: Feed-Forward Neural Network

- The statistical modeling cycle
- Generic feed-forward neural networks (FNNs)
- Universality theorems
- Gradient descent methods for model fitting
- Generalization loss and cross-validation
- Embedding layers

- **The Statistical Modeling Cycle**

The Statistical Modeling Cycle

(1) data collection, data cleaning and data pre-processing ($> 80\%$ of total time)

(2) selection of model class (data or algorithmic modeling culture, Breiman 2001)

(3) choice of objective function

(4) 'solving' a (non-convex) optimization problem

(5) model validation and variable selection

(6) possibly go back to (1)

▷ 'solving' involves:

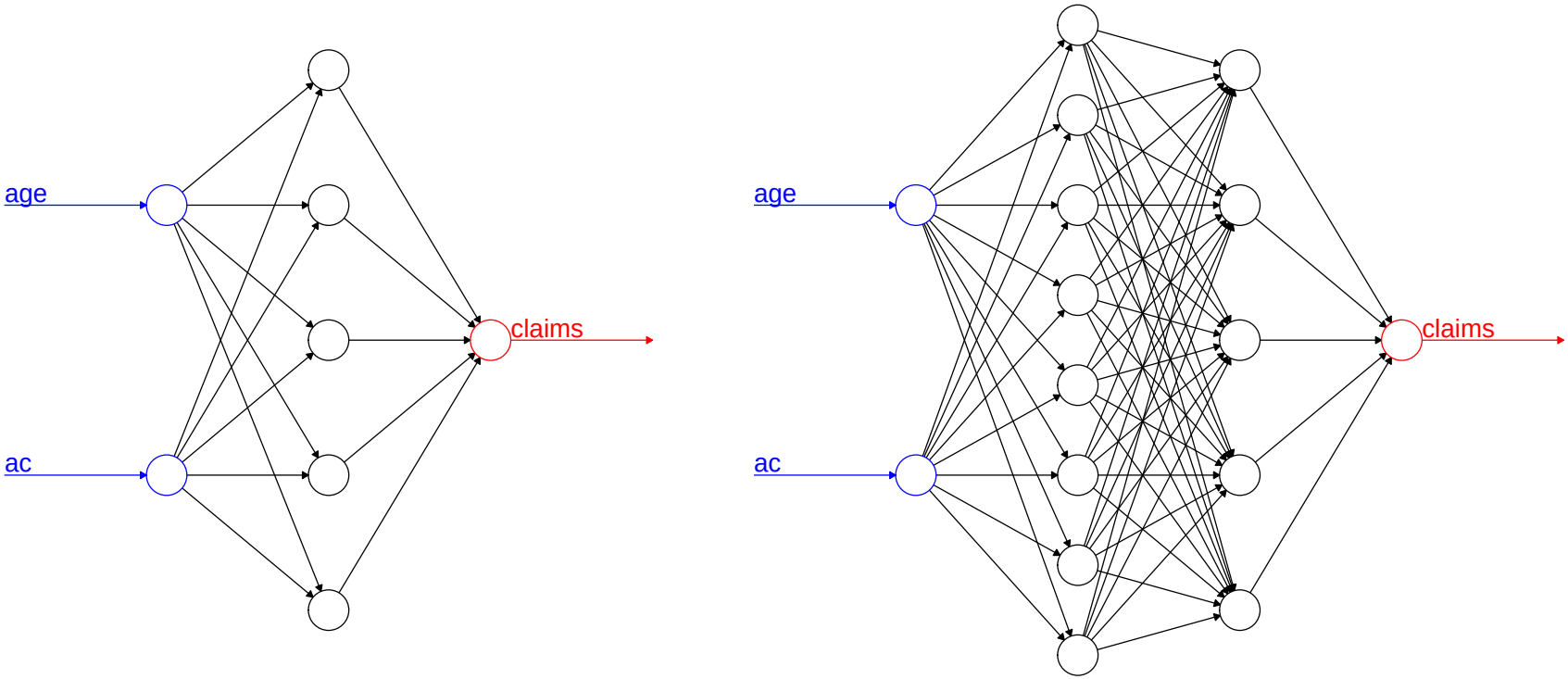
⋆ choice of algorithm
⋆ choice of stopping criterion, step size, etc.
⋆ choice of seed (starting value)

- **Generic Feed-Forward Neural Networks (FNNs)**

Neural Network Architectures

- Neural networks can be understood as an approximation framework.
- Here: neural networks generalize GLMs.
- There are different types of neural networks:
  - ⋆ Feed-forward neural network (FNN): Information propagates in one direction from input to output.
  - ⋆ Recurrent neural network (RNN): This is an extension of FNNs that allows for time series modeling (because it allows for time series (or causal) structures).
  - ⋆ Convolutional neural network (CNN): This is a type of network that allows for modeling temporal and spatial structure, e.g., in image recognition.
- FNNs have stacked hidden layers. If there is exactly one hidden layer, we call the network shallow; if there are multiple hidden layers, we call the network deep.
- There are many special neural network architectures such as generative-adversarial networks (GANs), bottleneck auto-encoder (AE) networks, etc.

Shallow and Deep Fully-Connected FNNs



These two examples are fully-connected FNNs.

Information is processed from the input (in blue) to the output (in red).

Representation Learning

- A GLM with link g has the following structure

$$\boldsymbol{x} \;\mapsto\; \mu(\boldsymbol{x}) = \mathbb{E}\left[Y\right] = g^{-1}\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle.$$

  ▷ This requires manual feature engineering to bring $\boldsymbol{x}$ into the right form.

- Networks perform automated feature engineering.

- A layer is given by a mapping

$$\boldsymbol{z}^{(m)} : \mathbb{R}^{q_{m-1}} \to \mathbb{R}^{q_m}.$$

  ▷ Each layer presents a new representation of the covariates.

- In general, compose layers

$$\boldsymbol{x} \;\mapsto\; \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) \;\stackrel{\text{def.}}{=}\; \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}) \;\in\; \mathbb{R}^{q_d}.$$

Fully-Connected FNN Layer

- Choose dimensions $q_{m-1}, q_m \in \mathbb{N}$ and activation function $\phi : \mathbb{R} \to \mathbb{R}$.

- A (hidden) FNN layer is a mapping

$$\boldsymbol{z}^{(m)} : \mathbb{R}^{q_{m-1}} \to \mathbb{R}^{q_m} \qquad \boldsymbol{x} \mapsto \boldsymbol{z}^{(m)}(\boldsymbol{x}) = \left(z_1^{(m)}(\boldsymbol{x}), \ldots, z_{q_m}^{(m)}(\boldsymbol{x}) \right)^\top,$$

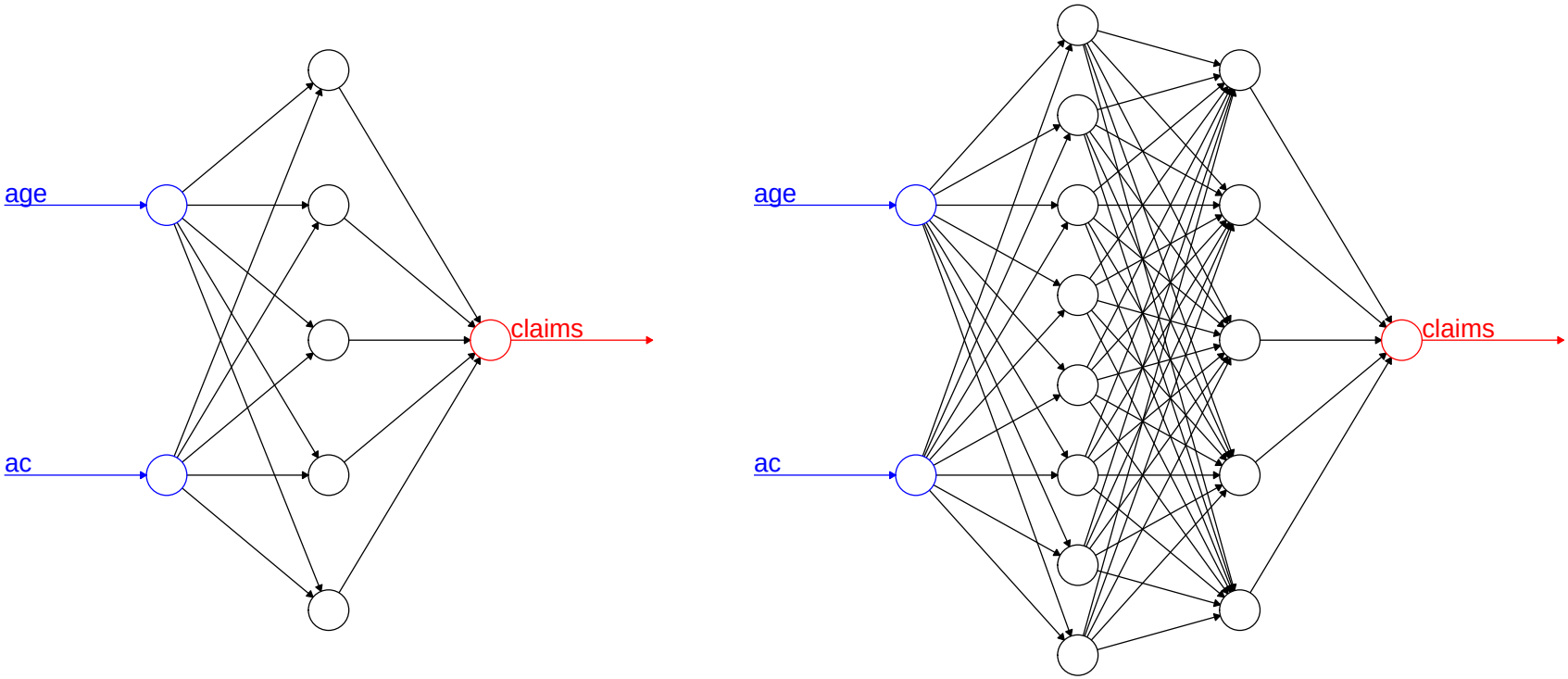
  with (hidden) neurons given by, $1 \leq j \leq q_m$,

$$z_j^{(m)}(\boldsymbol{x}) = \phi \left(w_{j,0}^{(m)} + \sum_{l=1}^{q_{m-1}} w_{j,l}^{(m)} x_l \right) \overset{\text{def.}}{=} \phi\langle \boldsymbol{w}_j^{(m)}, \boldsymbol{x} \rangle,$$

  for given network weights (parameters) $\boldsymbol{w}_j^{(m)} \in \mathbb{R}^{q_{m-1}+1}$.

- Every neuron $z_j^{(m)}(\boldsymbol{x})$ describes a GLM w.r.t. feature $\boldsymbol{x} \in \mathbb{R}^{q_{m-1}}$ and activation ϕ. The resulting function (called ridge function) reflects a compression of information.

Shallow and Deep Fully-Connected FNNs



These two examples are fully-connected FNNs.

Information is processed from the input (in blue) to the output (in red).

Activation Function

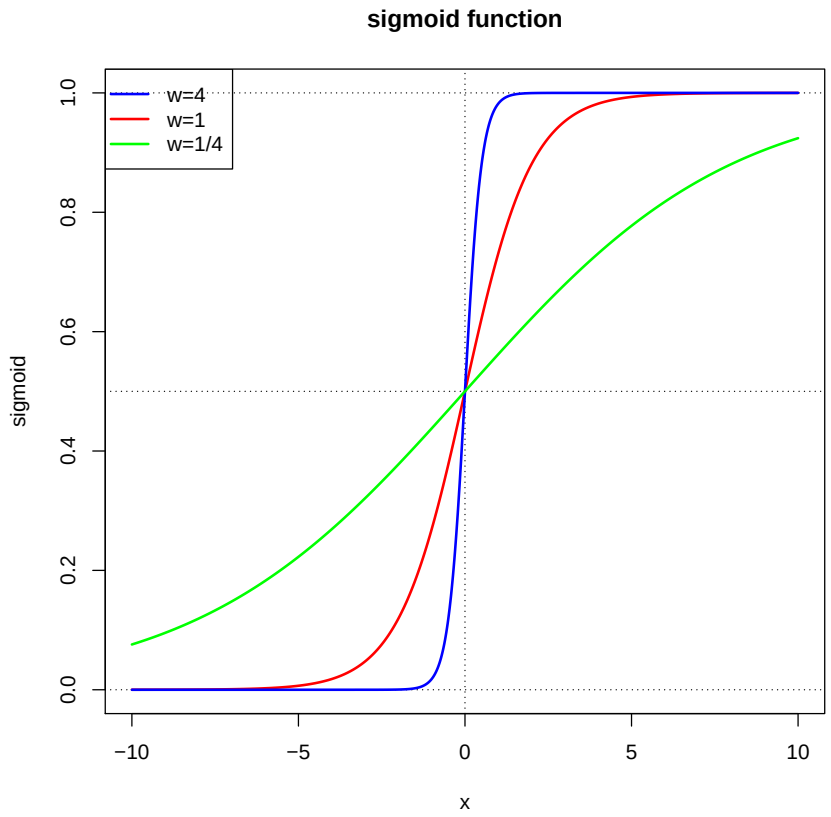
- The activation function $\phi : \mathbb{R} \to \mathbb{R}$ is an inverse link function $\phi = g^{-1}$.
- Since we would like to approximate non-linear regression functions, activation functions should be non-linear, too.
- The most popular choices of activation functions are

sigmoid/logistic function	$\phi(x) = (1 + e^{-x})^{-1} \in (0, 1)$	$\phi' = \phi(1 - \phi)$
hyperbolic tangent function	$\phi(x) = \tanh(x) \in (-1, 1)$	$\phi' = 1 - \phi^2$
exponential function	$\phi(x) = \exp(x) \in (0, \infty)$	$\phi' = \phi$
step function	$\phi(x) = \mathbb{1}_{\{x \geq 0\}} \in \{0, 1\}$	not differentiable in 0
rectified linear unit (ReLU)	$\phi(x) = x\mathbb{1}_{\{x \geq 0\}} \in [0, \infty)$	not differentiable in 0

- We mainly use hyperbolic tangent (with the following relationship to sigmoid)

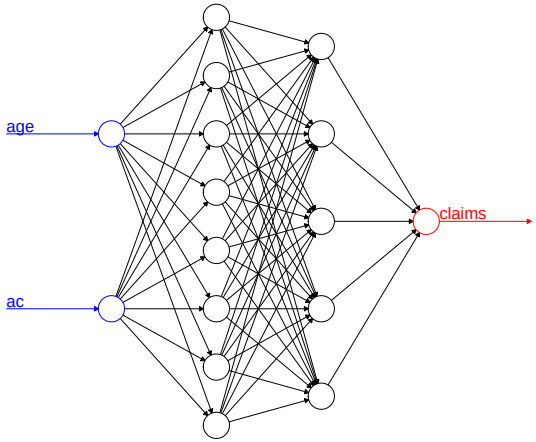
$$x \mapsto \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\left(1 + e^{-2x}\right)^{-1} - 1 = 2\,\mathrm{sigmoid}(2x) - 1.$$

Sigmoid Activation Function $\phi(x) = (1 + e^{-x})^{-1}$



- Sigmoid activation $x \mapsto \phi(wx)$ for weights $w \in \{1/4, 1, 4\}$ and $x \in (-10, 10)$:
  - ⋆ "deactivated" for small values x, i.e. $\phi(wx) \approx 0$ for x small,
  - ⋆ "activated" for big values x, i.e. $\phi(wx) \approx 1$ for x large.

Fully-Connected FNN Architecture



- Choose depth of the network $d \in \mathbb{N}$ and define the FNN layer composition

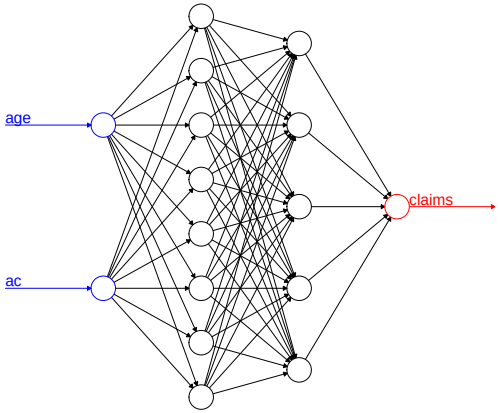
$$\boldsymbol{x} \;\mapsto\; \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) \;\stackrel{\text{def.}}{=}\; \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}) \in \mathbb{R}^{q_d},$$

  with $q_0 = q$ for $\boldsymbol{x} \in \mathbb{R}^q$.

- Define output layer with link function g by

$$\boldsymbol{x}_i \;\mapsto\; \mu_i = \mathbb{E}[Y_i] = g^{-1}\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_i) \right\rangle.$$

FNN Architecture: Interpretations



- Network mapping

$$\boldsymbol{x}_i \;\mapsto\; \mu_i = \mathbb{E}[Y_i] = g^{-1}\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_i) \right\rangle .$$

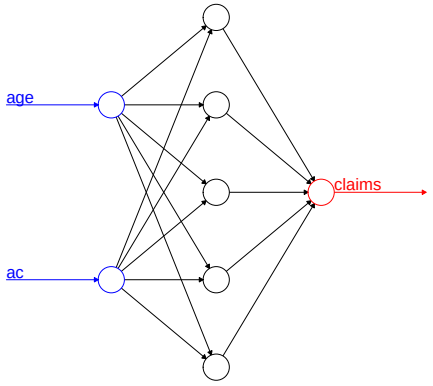
- Mapping $\boldsymbol{x}_i \mapsto \boldsymbol{z}_i = \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_i)$ should be understood as feature engineering or representation learning.

- The linear activation function $\phi(x) = x$ provides a GLM (composition of linear functions is a linear function). Thus, a GLM is a special case of a FNN.

- For depth $d = 0$ we receive a GLM, too.

- **Universality Theorems**

Universality Theorems for FNNs



- Cybenko (1989) and Hornik et al. (1989): Any compactly supported continuous function can be approximated arbitrarily well (in sup- or L^2-norm) by shallow FNNs with sigmoid activation if allowing for arbitrarily many hidden neurons (q_1).

- Leshno et al. (1993): The universality theorem for shallow FNNs holds if and only if the activation function ϕ is non-polynomial.

- Grohs et al. (2019): Shallow FNNs with ReLU activation functions provide polynomial approximation rates, deep FNNs provide exponential rates.

Simple Example Supporting Deep FNNs

- Consider a 2-dimensional example $\mu : [0,1]^2 \to \mathbb{R}_+$

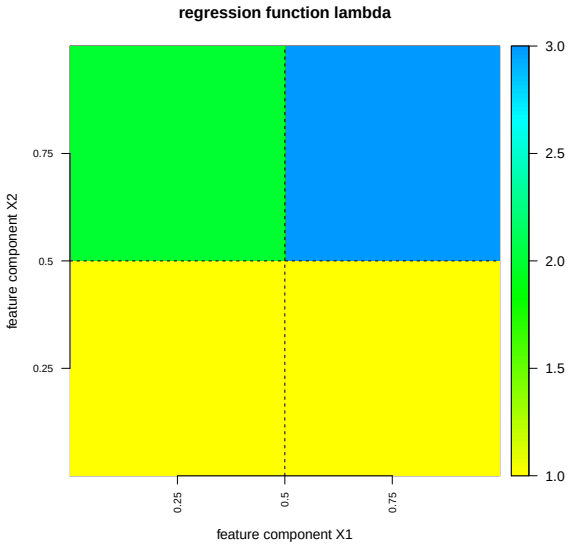
$$\boldsymbol{x} \;\mapsto\; \mu(\boldsymbol{x}) = 1 + \mathbb{1}_{\{x_2 \geq 1/2\}} + \mathbb{1}_{\{x_1 \geq 1/2,\; x_2 \geq 1/2\}} \;\in\; \{1,2,3\}.$$

- Choose step function activation $\phi(x) = \mathbb{1}_{\{x \geq 0\}}$.

- A FNN of depth $d = 2$ with $q_1 = q_2 = 2$

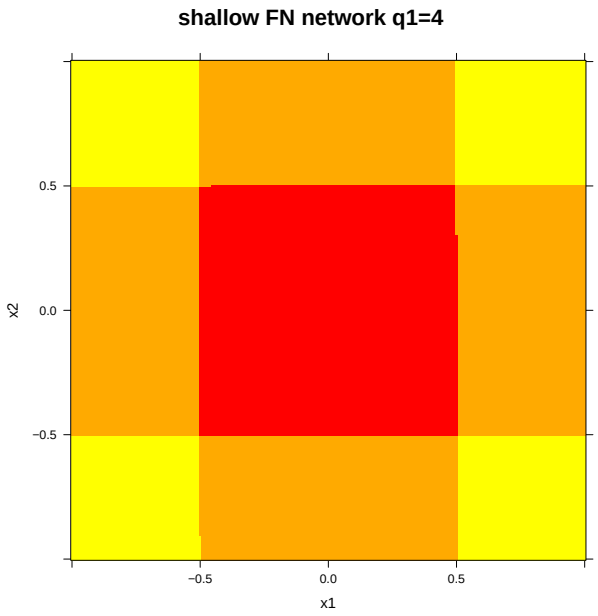
$$\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(2:1)}(\boldsymbol{x}) \right\rangle = \left\langle \boldsymbol{\beta}, (\boldsymbol{z}^{(2)} \circ \boldsymbol{z}^{(1)})(\boldsymbol{x}) \right\rangle,$$

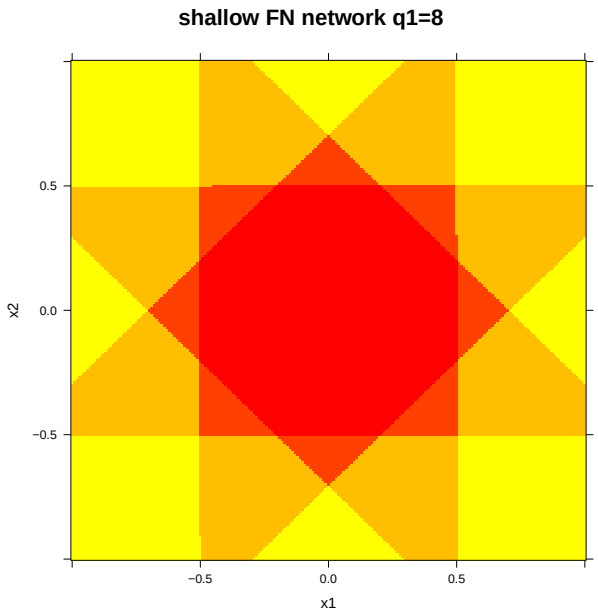
  can perfectly approximate function μ.

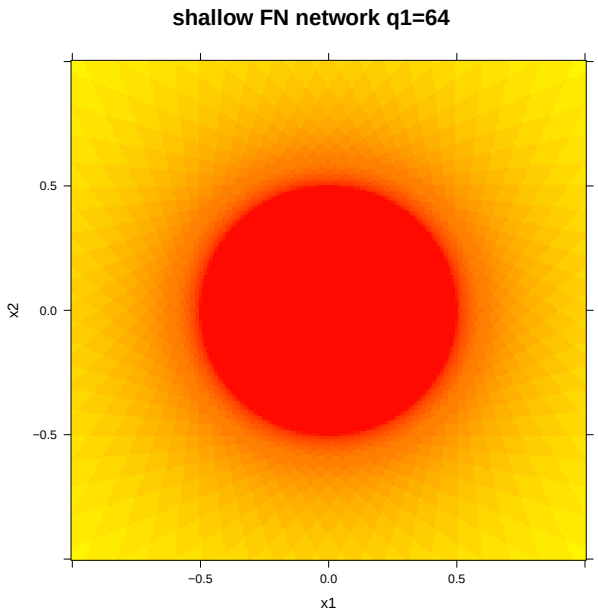


- Deep FNNs allow for more complex interactions of covariates through compositions of layers/functions: wide allows for superposition, and deep allows for composition.

Shallow Neural Networks







- **Gradient Descent Methods for Model Fitting**

Deviance Loss Function

- FNN mapping

$$\boldsymbol{x}_i \;\mapsto\; \mu_i = \mathbb{E}[Y_i] = g^{-1}\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_i)\right\rangle,$$

has network parameter

$$\vartheta = \left(\boldsymbol{w}_1^{(1)}, \ldots, \boldsymbol{w}_{q_d}^{(d)}, \boldsymbol{\beta}\right) \in \mathbb{R}^r,$$

of dimension $r = \sum_{m=1}^{d} q_m(q_{m-1}+1) + (q_d+1)$.

- The deviance loss function under independent observations $(Y_i)_{i=1}^n$

$$\begin{aligned}\vartheta \;\mapsto\; D^*(\boldsymbol{Y},\vartheta) &= 2\left[\ell_{\boldsymbol{Y}}(\boldsymbol{Y}) - \ell_{\boldsymbol{Y}}(\vartheta)\right] \\ &= 2\sum_{i=1}^{n}\frac{v_i}{\varphi}\Big[Y_i h(Y_i) - \kappa(h(Y_i)) - Y_i h(\mu_i) + \kappa(h(\mu_i))\Big] \;\geq\; 0.\end{aligned}$$

- Minimizing deviance loss $D^*(\boldsymbol{Y},\vartheta)$ in network parameter ϑ provides MLE $\widehat{\vartheta}$.

Plain Vanilla Gradient Descent Method (1/2)

- Gradient descent methods (GDMs) stepwise iteratively improve network parameter ϑ by moving into the direction of the maximal (local) decrease of $D^*(\boldsymbol{Y}, \vartheta)$.

- 1st order Taylor expansion of deviance loss in network parameter ϑ

$$D^*(\boldsymbol{Y}, \widetilde{\vartheta}) = D^*(\boldsymbol{Y}, \vartheta) + \nabla_\vartheta D^*(\boldsymbol{Y}, \vartheta)^\top \left(\widetilde{\vartheta} - \vartheta\right) + o\left(\|\widetilde{\vartheta} - \vartheta\|\right),$$

  for $\|\widetilde{\vartheta} - \vartheta\| \to 0$ (we suppose differentiability).

- Calculate the corresponding gradient

$$\nabla_\vartheta D^*(\boldsymbol{Y}, \vartheta) \;=\; \sum_{i=1}^{n} 2\left[\mu_i - Y_i\right] \nabla_\vartheta h(\mu_i).$$

- Back-propagation (Rumelhart et al. 1986) is an efficient way to calculate $\nabla_\vartheta h(\mu_i)$.

Plain Vanilla Gradient Descent Method (2/2)

- Negative gradient $-\nabla_\vartheta D^*(\boldsymbol{Y}, \vartheta)$ gives the direction for ϑ of the maximal local decrease in deviance loss.

- For a given learning rate $\varrho_{t+1} > 0$, the gradient descent algorithm updates network parameter $\vartheta^{(t)}$ iteratively by (adapted locally optimal)

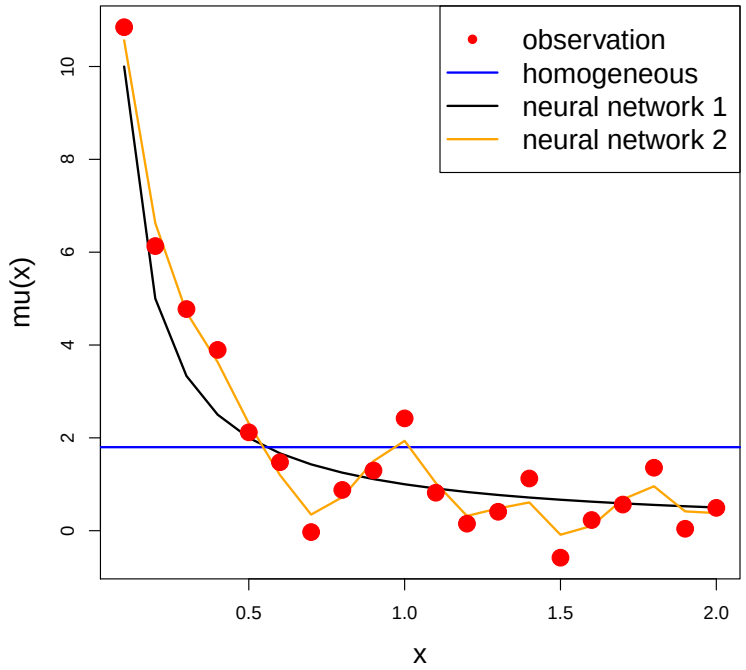
$$\vartheta^{(t)} \;\mapsto\; \vartheta^{(t+1)} = \vartheta^{(t)} - \varrho_{t+1} \nabla_\vartheta D^*(\boldsymbol{Y}, \vartheta^{(t)}).$$

- This update provides new (in-sample) deviance loss for $\varrho_{t+1} \to 0$

$$D^*(\boldsymbol{Y}, \vartheta^{(t+1)}) = D^*(\boldsymbol{Y}, \vartheta^{(t)}) - \varrho_{t+1} \left\| \nabla_\vartheta D^*(\boldsymbol{Y}, \vartheta^{(t)}) \right\|^2 + o\left(\varrho_{t+1}\right).$$

- Using a tempered learning rate $(\varrho_t)_{t \geq 1}$ the network parameter $\vartheta^{(t)}$ converges to a local minimum of $D^*(\boldsymbol{Y}, \cdot)$ for $t \to \infty$.

Over-Fitting in Complex FNNs

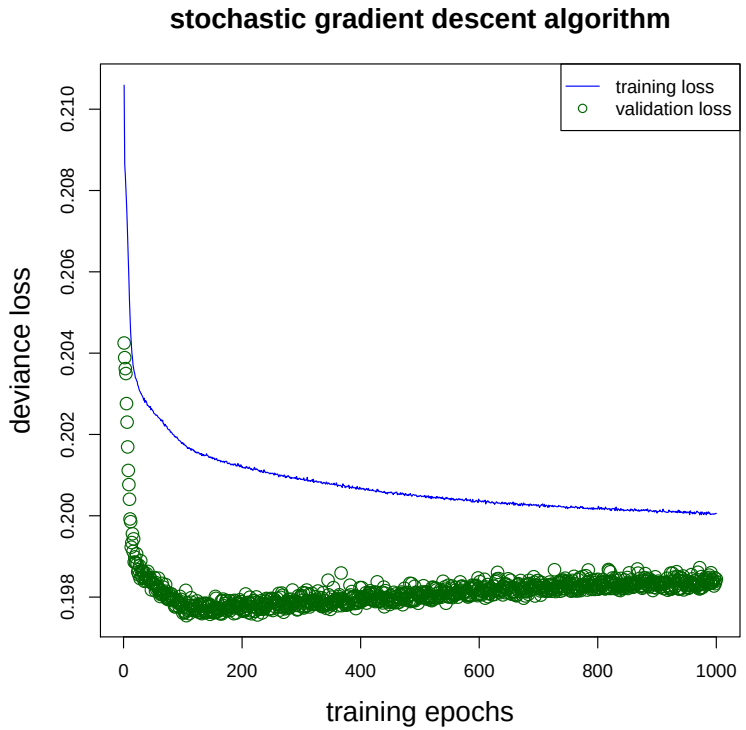


- Convergence to a local minimum of $D^*(\boldsymbol{Y}, \cdot)$ typically means over-fitting.

- Apply early stopping:

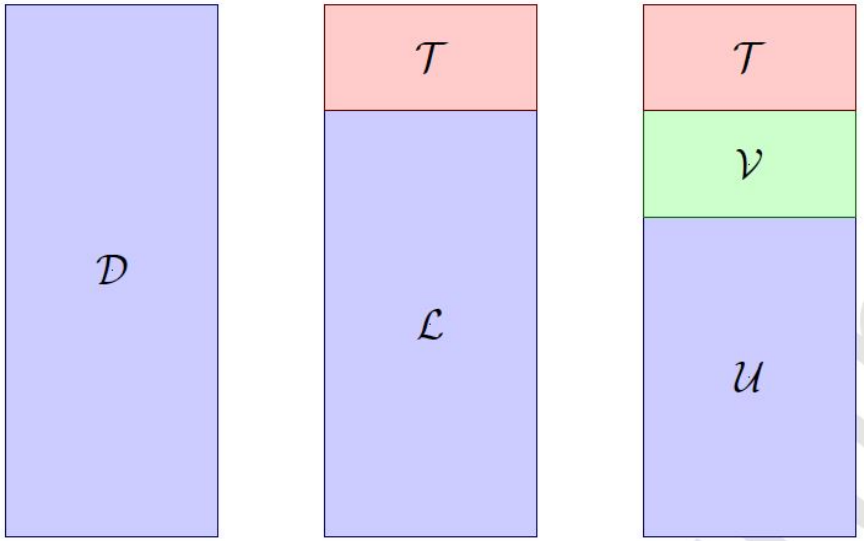
  ⋆ Partition data at random into training data $\mathcal{U}$ and validation data $\mathcal{V}$.
  ⋆ Fit ϑ on $\mathcal{U}$ (in-sample) and track over-fitting on $\mathcal{V}$ (out-of-sample).
  ⋆ The "best" model obviously is non-unique when we use early stopping.

Early Stopping of Gradient Descent Algorithm



- Convergence to a local minimum of $D^*(\boldsymbol{Y}, \cdot)$ typically means over-fitting.
- Apply early stopping:
  - ⋆ Partition data at random into training data $\mathcal{U}$ and validation data $\mathcal{V}$.
  - ⋆ Fit ϑ on $\mathcal{U}$ (in-sample) and track over-fitting on $\mathcal{V}$ (out-of-sample).
  - ⋆ The "best" model obviously is non-unique when we use early stopping.

Use of Data



$\mathcal{D}$	entire data,
$\mathcal{L}$	learning data (in-sample),
$\mathcal{T}$	test data (out-of-sample),
$\mathcal{U}$	training data,
$\mathcal{V}$	validation data

Computational Issues and Stochastic Gradient

- Gradient descent steps

$$\vartheta^{(t)} \;\mapsto\; \vartheta^{(t+1)} = \vartheta^{(t)} - \varrho_{t+1} \nabla_\vartheta D^*(\boldsymbol{Y}, \vartheta^{(t)}),$$

  involve high-dimensional matrix multiplications

$$\nabla_\vartheta D^*(\boldsymbol{Y}, \vartheta) \;=\; \sum_{i=1}^{n} 2\,[\mu_i - Y_i]\,\nabla_\vartheta h(\mu_i),$$

  which are computationally expensive if the size of the training data $\mathcal{U}$ is large.

- Partition training data $\mathcal{U}$ at random in mini batches $\mathcal{U}_k$ of a given size. Use for gradient descent steps one mini batch $\mathcal{U}_k$ at a time. This is called stochastic gradient descent (SGD) algorithm.

- Running through all mini batches $(\mathcal{U}_k)_k$ once is called a training epoch.

- Using the entire training data in each GDM step is called steepest gradient descent.

Size of Mini-Batches

- Partition training data $\mathcal{U}$ at random in mini batches $\mathcal{U}_1, \ldots, \mathcal{U}_K$, and use for each gradient descent step one mini batch $\mathcal{U}_k$ at a time

$$\nabla_\vartheta D^*(\mathcal{U}_k, \vartheta) \;=\; \sum_{i \in \mathcal{U}_k} 2\,[\mu_i - Y_i]\,\nabla_\vartheta h(\mu_i).$$

- Size of mini-batches for Poisson frequencies?

$$\left[\mu(\boldsymbol{x}) - 2\sqrt{\frac{\mu(\boldsymbol{x})}{v}}, \mu(\boldsymbol{x}) + 2\sqrt{\frac{\mu(\boldsymbol{x})}{v}}\right] = \left[5\% - 2\sqrt{\frac{5\%}{2000}}, 5\% + 2\sqrt{\frac{5\%}{2000}}\right] = [4\%, 6\%].$$

Note for Poisson case $\mathbb{E}[N] = \mathrm{Var}(N) = \mu(\boldsymbol{x})v$.

Momentum-Based Gradient Descent Methods

- Plain vanilla GDMs use 1st order Taylor expansions.
- To improve convergence rates we could use 2nd order Taylor expansions.
- 2nd order Taylor expansions involve calculations of Hessians.
- This is computationally not feasible.
- Replace Hessians by momentum methods (inspired by physics/mechanics).
- Choose a momentum coefficient $\nu \in [0, 1)$ and set initial speed $\mathbf{v}^{(0)} = 0 \in \mathbb{R}^r$. Replace plain vanilla GDM update by

$$\begin{aligned} \mathbf{v}^{(t)} &\mapsto \mathbf{v}^{(t+1)} = \nu \mathbf{v}^{(t)} - \varrho_{t+1} \nabla_\vartheta D^*(\boldsymbol{Y}, \vartheta^{(t)}), \\ \vartheta^{(t)} &\mapsto \vartheta^{(t+1)} = \vartheta^{(t)} + \mathbf{v}^{(t+1)}. \end{aligned}$$

Predefined Gradient Descent Methods

- 'rmsprop' chooses learning rates that differ in all directions by consider directional sizes ('rmsprop' stands for root mean square propagation);

- 'adam' stands for adaptive moment estimation, similar to 'rmsprop' it searches for directionally optimal learning rates based on the momentum induced by past gradients measured by an L^2-norm;

- 'nadam' is Nesterov (2007) accelerated version of 'adam' avoiding zig-zag behavior.

- For more details we refer to Chapter 8 of Goodfellow et al. (2016) and Section 7.2.3 in Wüthrich–Merz (2021)

- **Generalization Loss and Cross-Validation**

Empirical Generalization Loss

Typically, for neural network modeling one considers 3 disjoint sets of data.

- Training data $\mathcal{U}$: is used to fit the network parameter ϑ.
- Validation data $\mathcal{V}$: is used to track in-sample over-fitting (early stopping).
- Test data $\mathcal{T}$: is used to study out-of-sample generalization loss.

Assume that $\widehat{\vartheta}^{\mathcal{U},\mathcal{V}}$ is the estimated network parameter based on $\mathcal{U}$ and $\mathcal{V}$. The test data $\mathcal{T}$ is given by $(Y_t, \boldsymbol{x}_t, v_t)_{t=1}^T$. We have (out-of-sample) generalization loss (GL)

$$D^*(\boldsymbol{Y}, \widehat{\vartheta}^{\mathcal{U},\mathcal{V}}) = 2\sum_{t=1}^{T} \frac{v_t}{\varphi}\Big[Y_t h(Y_t) - \kappa(h(Y_t)) - Y_t h(\widehat{\mu}_t^{\mathcal{U},\mathcal{V}}) + \kappa(h(\widehat{\mu}_t^{\mathcal{U},\mathcal{V}}))\Big].$$

- This is an empirical generalization loss based on $\mathcal{T}$ mimicking portfolio distribution.

K-Fold Cross-Validation Loss

- If one cannot afford to partition the data $\mathcal{D}$ into 3 disjoint sets training data $\mathcal{U}$, validation data $\mathcal{V}$ and test data $\mathcal{T}$, one has to use the data more efficiently.
- K-fold cross-validation aims at doing so.
- Partition entire data at random in K subsets $\mathcal{D}_1, \ldots, \mathcal{D}_K$ of roughly equal size.
- Denote by $\widehat{\vartheta}^{(-\mathcal{D}_k)}$ the estimated network parameter based on all data except $\mathcal{D}_k$.
- The K-fold cross-validation loss is given by

$$D^{\mathrm{CV}} = \frac{1}{K} \sum_{k=1}^{K} \left(2 \sum_{t \in \mathcal{D}_k} \frac{v_t}{\varphi} \left[Y_t h(Y_t) - \kappa(h(Y_t)) - Y_t h(\widehat{\mu}_t^{(-\mathcal{D}_k)}) + \kappa(h(\widehat{\mu}_t^{(-\mathcal{D}_k)})) \right] \right).$$

- This mimics K times an out-of-sample generalization loss on $\mathcal{D}_k$, respectively.
- In neural network modeling K-fold cross-validation is computationally too costly.

- **Car Insurance Frequency Example**

Car Insurance Claims Frequency Data

```
'data.frame':   678013 obs. of  12 variables:
 $ IDpol      : num  1 3 5 10 11 13 15 17 18 21 ...
 $ ClaimNb    : num  1 1 1 1 1 1 1 1 1 1 ...
 $ Exposure   : num  0.1 0.77 0.75 0.09 0.84 0.52 0.45 0.27 0.71 0.15 ...
 $ Area       : Factor w/ 6 levels "A","B","C","D",..: 4 4 2 2 2 5 5 3 3 2 ...
 $ VehPower   : int  5 5 6 7 7 6 6 7 7 7 ...
 $ VehAge     : int  0 0 2 0 0 2 2 0 0 0 ...
 $ DrivAge    : int  55 55 52 46 46 38 38 33 33 41 ...
 $ BonusMalus: int  50 50 50 50 50 50 50 68 68 50 ...
 $ VehBrand   : Factor w/ 11 levels "B1","B10","B11",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ VehGas     : Factor w/ 2 levels "Diesel","Regular": 2 2 1 1 1 2 2 1 1 1 ...
 $ Density    : int  1217 1217 54 76 76 3003 3003 137 137 60 ...
 $ Region     : Factor w/ 22 levels "R11","R21","R22",..: 18 18 3 15 15 8 8 20 20 12
```

- 3 categorical covariates, 1 binary covariate and 5 continuous covariates
- Goal: Find systematic effects to explain/predict claim counts.

Feature Engineering

- Categorical features: use either dummy coding or one-hot encoding.
  PS: We come back to this choice below.

- Also continuous features need pro-processing. All feature components should live on a similar scale such that the GDM can be applied efficiently.

▷ Often, the MinMaxScaler is used

$$x_{i,l} \;\mapsto\; x_{i,l}^{\mathrm{MM}} = 2\,\frac{x_{i,l} - x_l^-}{x_l^+ - x_l^-} - 1 \;\in\; [-1,1],$$

where x_l^- and x_l^+ are the minimum and maximum of the domain of $x_{i,l}$.

- Successful application of MinMaxScaler pre-processing requires that the feature distribution is not "too skewed", otherwise pre-processing should be performed with a scaler that accounts for skewness (like the log function).

- Standardization with empirical mean and standard deviation is possible, too.

Deep FNN Coding in R keras

```
library(keras)

q0 <- 12   # dimension of input x
q1 <- 20
q2 <- 15
q3 <- 10

Design <- layer_input(shape = c(q0),  dtype = 'float32', name = 'Design')

Network = Design %>%
          layer_dense(units=q1, activation='tanh', name='hidden1') %>%
          layer_dense(units=q2, activation='tanh', name='hidden2') %>%
          layer_dense(units=q3, activation='tanh', name='hidden3') %>%
          layer_dense(units=1,  activation='exponential', name='Network')

model <- keras_model(inputs = c(Design), outputs = c(Network))
model %>% compile(optimizer = optimizer_nadam(), loss = 'poisson')

summary(model)
```

Deep FNN with $(q_1, q_2, q_3) = (20, 15, 10)$

```
Layer (type)                 Output Shape              Param #
=================================================================
Design (InputLayer)          (None, 12)                0
_________________________________________________________________
hidden1 (Dense)              (None, 20)                260
_________________________________________________________________
hidden2 (Dense)              (None, 15)                315
_________________________________________________________________
hidden3 (Dense)              (None, 10)                160
_________________________________________________________________
Network (Dense)              (None, 1)                 11
=================================================================
Total params: 746
Trainable params: 746
Non-trainable params: 0
```

Poisson FNN Regression with Offset

- Poisson regression with offset and canonical link $g = h = \log$, set $N_i = v_i Y_i$

$$v_i \mu_i \;=\; \mathbb{E}\left[N_i\right] \;=\; v_i \kappa'(\theta_i) \;=\; v_i \exp(\theta_i) \;=\; \exp(\log v_i + \theta_i).$$

- The Poisson FNN regression is given by, set $\mu_i = \mu(\boldsymbol{x}_i)$,

$$\boldsymbol{x}_i \;\mapsto\; \log\left(\mathbb{E}\left[N_i\right]\right) = \log\left(v_i \mu(\boldsymbol{x}_i)\right) = \log v_i + \left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_i) \right\rangle.$$

- The Poisson deviance loss function is given by

$$D^*(\boldsymbol{N}, \vartheta) = \sum_{i=1}^{n} 2\; N_i \left[\frac{v_i \mu(\boldsymbol{x}_i)}{N_i} - 1 - \log\left(\frac{v_i \mu(\boldsymbol{x}_i)}{N_i} \right) \right] \;\geq\; 0,$$

  where the i-th term is set equal to $2 v_i \mu(\boldsymbol{x}_i)$ for $N_i = 0$.

- In `keras` the terms independent of ϑ are dropped in the deviance losses.

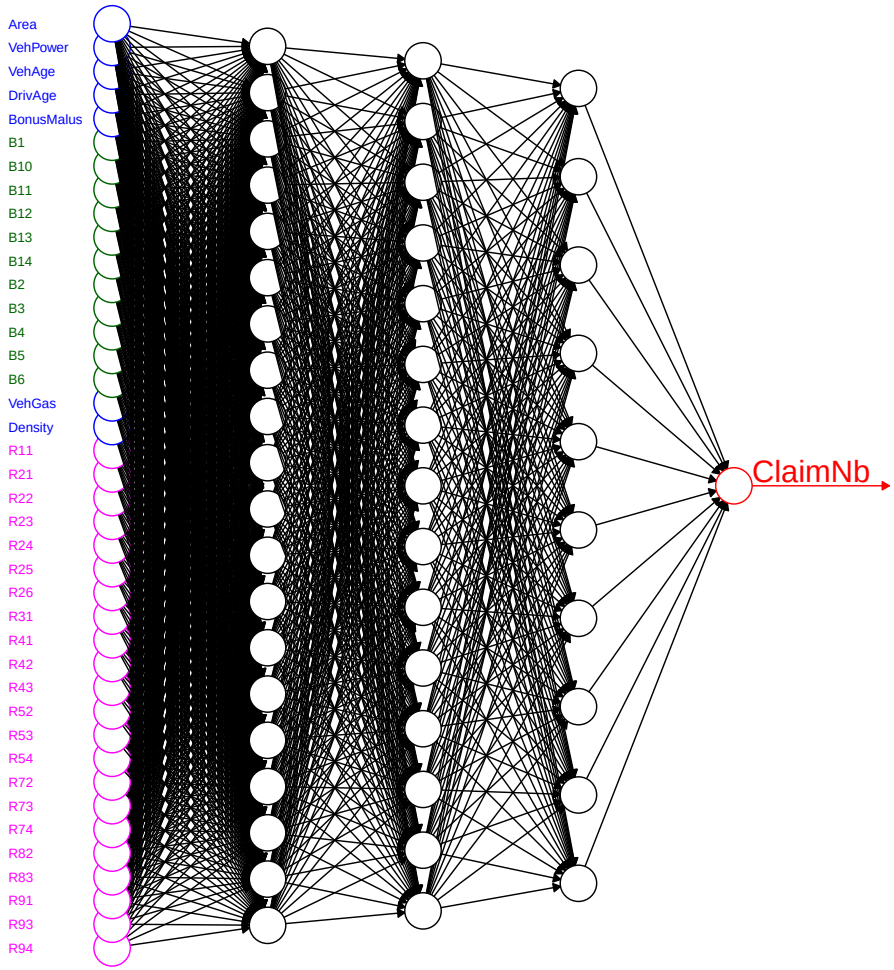
Deep FNN Coding in R keras with Offset

```
q0 <- 12     # dimension of input x
q1 <- 20
q2 <- 15
q3 <- 10
lambda.hom <- 0.05   # initialization of network output (homogeneous model)

Design <- layer_input(shape = c(q0),  dtype = 'float32', name = 'Design')
LogVol <- layer_input(shape = c(1),   dtype = 'float32', name = 'LogVol')

Network = Design %>%
          layer_dense(units=q1, activation='tanh',   name='hidden1') %>%
          layer_dense(units=q2, activation='tanh',   name='hidden2') %>%
          layer_dense(units=q3, activation='tanh',   name='hidden3') %>%
          layer_dense(units=1,  activation='linear', name='Network',
             weights=list(array(0, dim=c(10,1)), array(log(lambda.hom), dim=c(1))))

Response = list(Network, LogVol) %>% layer_add(name='Add') %>%
          layer_dense(units=1, activation= 'exponential', name ='Response',
             trainable=FALSE, weights=list(array(1, dim=c(1,1)), array(0, dim=c(1))))

model <- keras_model(inputs = c(Design, LogVol), outputs = c(Response))
model %>% compile(optimizer = optimizer_nadam(), loss = 'poisson')
```

Deep FNN with $(q_1, q_2, q_3) = (20, 15, 10)$ with Offset

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================
Design (InputLayer)          (None, 12)                0
----------------------------------------------------------------------------------
hidden1 (Dense)              (None, 20)                260       Design[0][0]
----------------------------------------------------------------------------------
hidden2 (Dense)              (None, 15)                315       hidden1[0][0]
----------------------------------------------------------------------------------
hidden3 (Dense)              (None, 10)                160       hidden2[0][0]
----------------------------------------------------------------------------------
Network (Dense)              (None, 1)                 11        hidden3[0][0]
----------------------------------------------------------------------------------
LogVol (InputLayer)          (None, 1)                 0
----------------------------------------------------------------------------------
Add (Add)                    (None, 1)                 0         Network[0][0]
                                                                 LogVol[0][0]
----------------------------------------------------------------------------------
Response (Dense)             (None, 1)                 2         Add[0][0]
==================================================================================
Total params: 748
Trainable params: 746
Non-trainable params: 2
```

Application to French MTPL Data



Input dimension is $q_0 = 40$ (one-hot encoding), this provides $r = 1'306$.

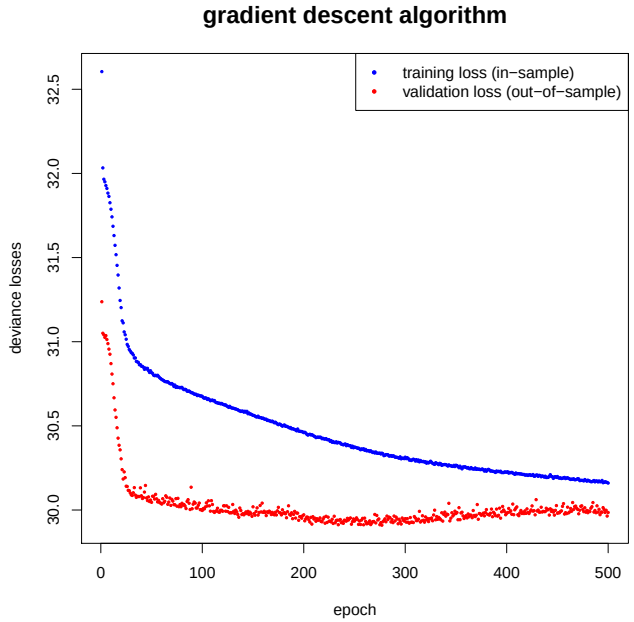
Results of Deep FNN Model

	epochs	run time	# param.	in-sample loss 10^{-2}	out-of-sample loss 10^{-2}	average frequency
homogeneous model		–	1	32.935	33.861	10.02%
Model GLM1		20s	49	31.267	32.171	10.02%
Deep FNN model	250	152s	1'306	30.268	31.673	10.19%

- Network fitting needs quite some run time.
- We perform early stopping.
- The best validation loss model can be retrieved with a callback, see next slide.
- We see a *substantial* improvement in out-of-sample loss on test data $\mathcal{T}$.
- Balance property fails to hold.
- Remark: AIC is not a sensible model selection criterion for FNNs (early stopping).

Callbacks in Gradient Descent Methods

```
path0 <- "./name0"
CB    <- callback_model_checkpoint(path0, monitor = "val_loss",
                        verbose = 0,  save_best_only = TRUE,
                        save_weights_only = TRUE)

model %>% fit(list(X.learn, LogVol.learn), Y.learn,
                       validation_split = 0.1, batch_size = 10000, epochs = 500,
                       verbose = 0, callbacks = CBs)

load_model_weights_hdf5(model, path0)
```



- **Embedding Layers for Categorical Variables**

Categorical Variables and Dummy/One-Hot Encoding

B1	0	0	0	0	0	0	0	0	0	0
B10	1	0	0	0	0	0	0	0	0	0
B11	0	1	0	0	0	0	0	0	0	0
B12	0	0	1	0	0	0	0	0	0	0
B13	0	0	0	1	0	0	0	0	0	0
B14	0	0	0	0	1	0	0	0	0	0
B2	0	0	0	0	0	1	0	0	0	0
B3	0	0	0	0	0	0	1	0	0	0
B4	0	0	0	0	0	0	0	1	0	0
B5	0	0	0	0	0	0	0	0	1	0
B6	0	0	0	0	0	0	0	0	0	1

each row is in $\mathbb{R}^{10}$

B1 $\mapsto$ $\boldsymbol{e}_1$	1	0	0	0	0	0	0	0	0	0	0
B10 $\mapsto$ $\boldsymbol{e}_2$	0	1	0	0	0	0	0	0	0	0	0
B11 $\mapsto$ $\boldsymbol{e}_3$	0	0	1	0	0	0	0	0	0	0	0
B12 $\mapsto$ $\boldsymbol{e}_4$	0	0	0	1	0	0	0	0	0	0	0
B13 $\mapsto$ $\boldsymbol{e}_5$	0	0	0	0	1	0	0	0	0	0	0
B14 $\mapsto$ $\boldsymbol{e}_6$	0	0	0	0	0	1	0	0	0	0	0
B2 $\mapsto$ $\boldsymbol{e}_7$	0	0	0	0	0	0	1	0	0	0	0
B3 $\mapsto$ $\boldsymbol{e}_8$	0	0	0	0	0	0	0	1	0	0	0
B4 $\mapsto$ $\boldsymbol{e}_9$	0	0	0	0	0	0	0	0	1	0	0
B5 $\mapsto$ $\boldsymbol{e}_{10}$	0	0	0	0	0	0	0	0	0	1	0
B6 $\mapsto$ $\boldsymbol{e}_{11}$	0	0	0	0	0	0	0	0	0	0	1

each row is in $\mathbb{R}^{11}$

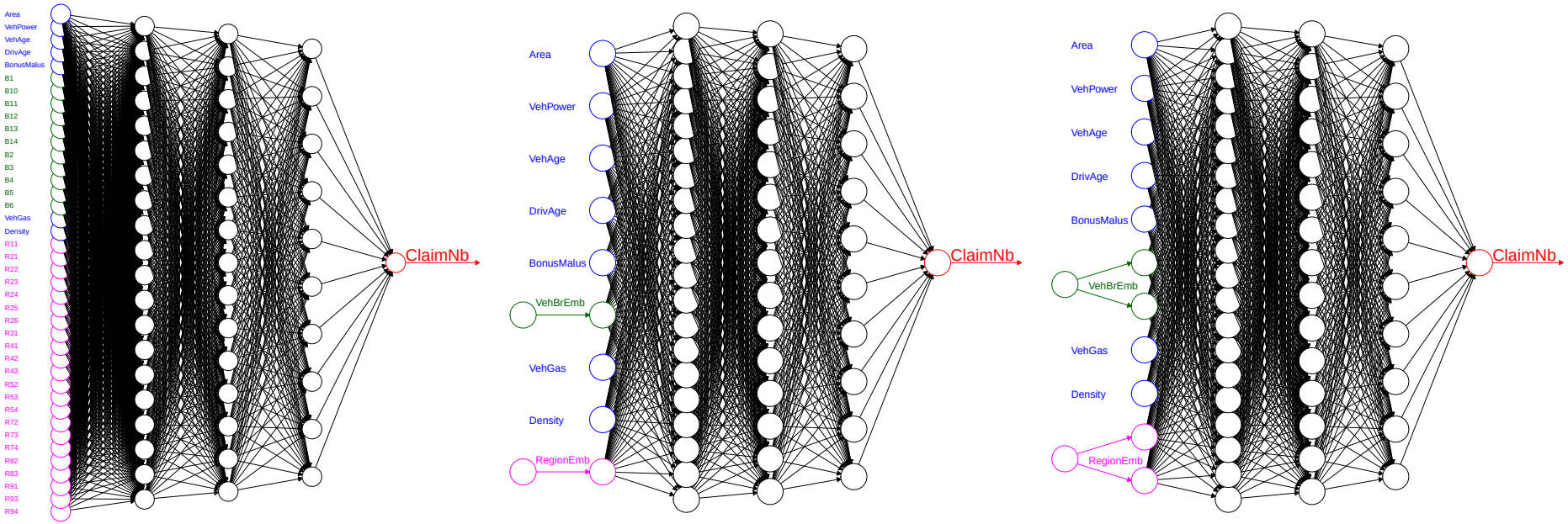
Embeddings for Categorical Variables

- One-hot encoding uses as many dimensions as there are labels (mapping to unit vectors in Euclidean space).
- All labels have the same distance from each other.
- From Natural Language Processing (NLP) we have learned that there are "better" codings in the sense that we should try to map to low-dimensional Euclidean spaces $\mathbb{R}^b$, and similar labels (w.r.t. the regression task) should have some proximity.
- Choose $b \in \mathbb{N}$ and consider an embedding mapping (representation)

$$\boldsymbol{e} : \{\texttt{B1}, \ldots, \texttt{B6}\} \to \mathbb{R}^b, \qquad \texttt{brand} \mapsto \boldsymbol{e}(\texttt{brand}) \stackrel{\text{def.}}{=} \boldsymbol{e}^{\texttt{brand}}.$$

- $\boldsymbol{e}^{\texttt{brand}} \in \mathbb{R}^b$ are called embeddings, and optimal embeddings for the regression task can be learned during GDM training. This amounts in adding an additional (embedding) layer to the FNN.

Deep FNN using Embedding Layers (1/2)



(left) one-hot enc. (middle) $b = 1$-dim. emb's (right) $b = 2$-dim. emb's

- Embedding weights are learned during network training (gradient descent).

Deep FNN using Embedding Layers (2/2)

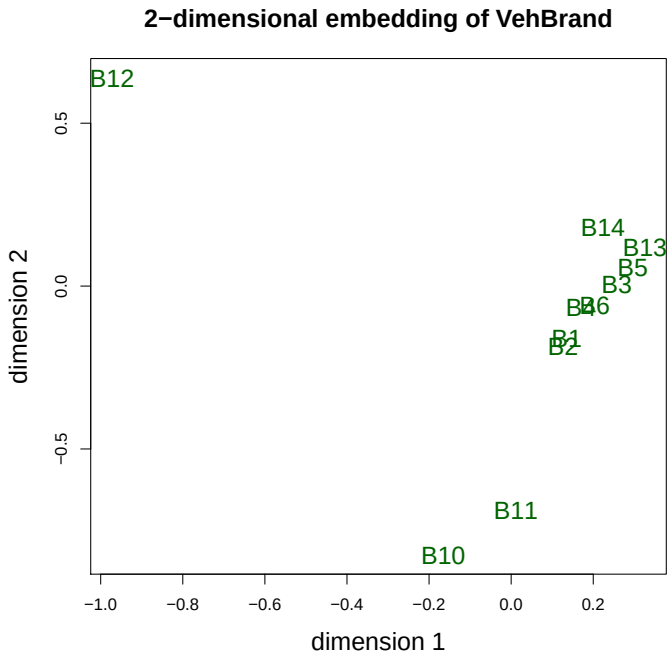
```
Design   <- layer_input(shape = c(7), dtype = 'float32', name = 'Design')
VehBrand <- layer_input(shape = c(1), dtype = 'int32',   name = 'VehBrand')
Region   <- layer_input(shape = c(1), dtype = 'int32',   name = 'Region')
LogVol   <- layer_input(shape = c(1), dtype = 'float32', name = 'LogVol')

BrEmb = VehBrand %>%
    layer_embedding(input_dim=11, output_dim=2, input_length=1, name='BrEmb') %>%
    layer_flatten(name='Br_flat')
ReEmb = Region %>%
    layer_embedding(input_dim=22, output_dim=2, input_length=1, name='ReEmb') %>%
    layer_flatten(name='Re_flat')

Network = list(Design, BrEmb, ReEmb) %>% layer_concatenate(name='concate') %>%
    layer_dense(units=20, activation='tanh',   name='hidden1') %>%
    layer_dense(units=15, activation='tanh',   name='hidden2') %>%
    layer_dense(units=10, activation='tanh',   name='hidden3') %>%
    layer_dense(units=1,  activation='linear', name='Network',
                weights=list(array(0, dim=c(10,1)), array(log(lambda.hom), dim=c(1))))

Response = list(Network, LogVol) %>% layer_add(name='Add') %>%
    layer_dense(units=1, activation='exponential', name='Response', trainable=FALSE,
                weights=list(array(1, dim=c(1,1)), array(0, dim=c(1))))

model <- keras_model(inputs=c(Design,VehBrand,Region,LogVol), outputs=c(Response))
```

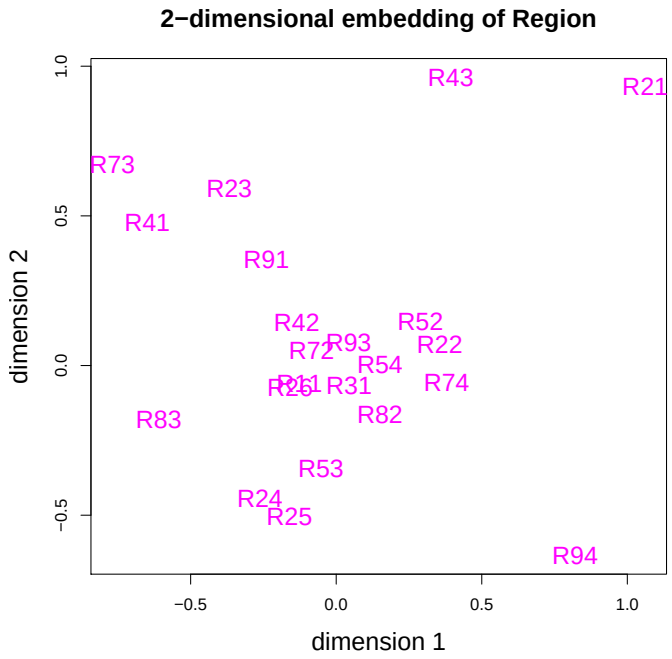
Results of Deep FNN Model with Embeddings

	epochs	run time	# param.	in-sample loss 10^{-2}	out-of-sample loss 10^{-2}	average frequency
homogeneous model		–	1	32.935	33.861	10.02%
Model GLM1		20s	49	31.268	32.171	10.02%
Deep FNN One-Hot	250	152s	1'306	30.268	31.673	10.19%
Deep FNN Emb($b = 1$)	700	419s	719	30.245	31.506	9.90%
Deep FNN Emb($b = 2$)	600	365s	792	30.165	31.453	9.70%

- Network fitting needs quite some run time.
- We perform early stopping using a callback.
- We see a substantial improvement in out-of-sample loss on test data $\mathcal{T}$.
- Balance property fails to hold.
- Remark: AIC is not a sensible model selection criterion for FNNs (early stopping).

Learned Two-Dimensional Embeddings





Two-dimensional embeddings can be nicely plotted and interpreted.

Special Purpose Layers and Other Features

- **Drop-out layers.** A method to prevent from over-training individual neurons to a certain task is to introduce so-called drop-out layers. A drop-out layer, say, after `'hidden2'` of the above listing would remove during a gradient descent step at random any of the 15 neurons in that layer with a given drop-out probability $p \in (0, 1)$, and independently from the other neurons. This random removal will imply that the composite of the remaining neurons needs to sufficiently well cover the dropped-out neurons. Therefore, a single neuron cannot be over-trained to a certain task because it may need to play several different roles at the same time. Drop-out can be interpreted in terms of ridge regression, see Section 18.6 in Efron–Hastie (2016).

- **Normalization layers.** Feature activations $\boldsymbol{z}^{(m:1)}(\boldsymbol{x})$ are scaled back to be centered and have unit standard variance (similar to MinMaxScaler).

- **Skip connections.** Certain layers are skipped in the network architecture, this is going to be used in the LocalGLMnet chapter.

References

- Breiman (2001). Statistical modeling: the two cultures. Statistical Science 16/3, 199-215.
- Cybenko (1989). Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems 2, 303-314.
- Efron (2020). Prediction, estimation, and attribution. Journal American Statistical Association 115/539 , 636-655.
- Efron, Hastie (2016). Computer Age Statistical Inference: Algorithms, Evidence, and Data Science. Cambridge UP.
- Ferrario, Noll, Wüthrich (2018). Insights from inside neural networks. SSRN 3226852.
- Goodfellow, Bengio, Courville (2016). Deep Learning. MIT Press.
- Grohs, Perekrestenko, Elbrächter, Bölcskei (2019). Deep neural network approximation theory. IEEE Transactions on Information Theory.
- Hastie, Tibshirani, Friedman (2009). The Elements of Statistical Learning. Springer.
- Hornik, Stinchcombe, White (1989). Multilayer feedforward networks are universal approximators. Neural Networks 2, 359-366.
- Kingma, Ba (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.
- Leshno, Lin, Pinkus, Schocken (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Networks 6/6, 861-867.
- Nesterov (2007). Gradient methods for minimizing composite objective function. Technical Report 76, Center for Operations Research and Econometrics (CORE), Catholic University of Louvain.
- Noll, Salzmann, Wüthrich (2018). Case study: French motor third-party liability claims. SSRN 3164764.
- Richman (2020a/b). AI in actuarial science – a review of recent advances – part 1/2. Annals of Actuarial Science.
- Rumelhart, Hinton, Williams (1986). Learning representations by back-propagating errors. Nature 323/6088, 533-536.
- Schelldorfer, Wüthrich (2019). Nesting classical actuarial models into neural networks. SSRN 3320525.
- Shmueli (2010). To explain or to predict? Statistical Science 25/3, 289-310.
- Wüthrich, Buser (2016). Data Analytics for Non-Life Insurance Pricing. SSRN 2870308, Version September 10, 2020.
- Wüthrich, Merz (2021). Statistical Foundations of Actuarial Learning and its Applications. SSRN 3822407.

Discrimination-Free Insurance Pricing

Mario V. Wüthrich
RiskLab, ETH Zurich

RiskLab®
Switzerland

"Deep Learning with Actuarial Applications in R"
Swiss Association of Actuaries SAA/SAV, Zurich
October 14/15, 2021

Programme SAV Block Course

- Refresher: Generalized Linear Models (THU 9:00-10:30)
- Feed-Forward Neural Networks (THU 13:00-15:00)
- Discrimination-Free Insurance Pricing (THU 17:15-17:45)

- LocalGLMnet (FRI 9:00-10:30)
- Convolutional Neural Networks (FRI 13:00-14:30)
- Wrap Up (FRI 16:00-16:30)

Contents: Discrimination-Free Insurance Pricing

- Direct discrimination
- Indirect discrimination
- Unawareness price
- Discrimination-free price

- **Direct Discrimination**

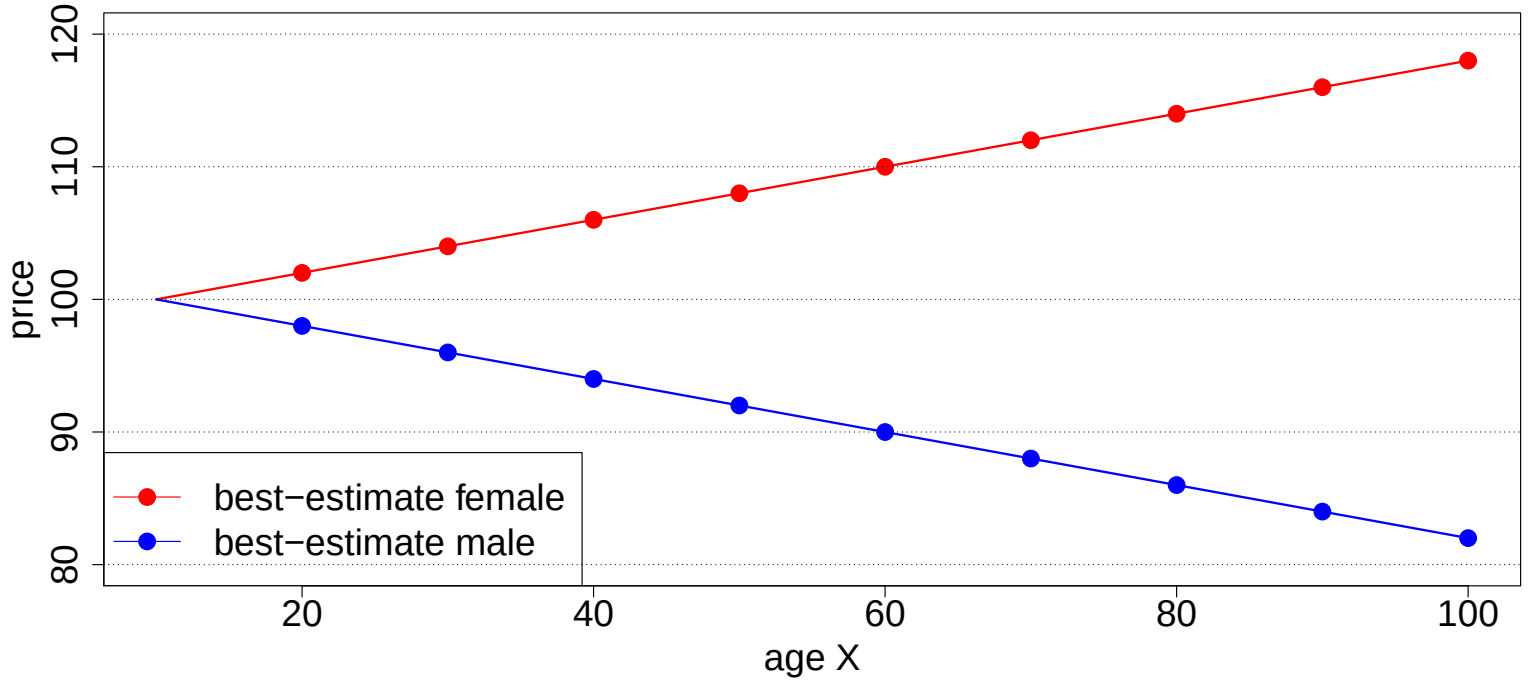
Best-Estimate Pricing

- Basic pricing setup is given by
  - ⋆ Y denotes the claim costs;
  - ⋆ $\boldsymbol{X}$ denotes non-discriminatory covariates;
  - ⋆ $\boldsymbol{D}$ denotes discriminatory covariates.
- Develop regression model for Y using covariates $\boldsymbol{X}$ and $\boldsymbol{D}$ as explanatory variables.
- This motivates best-estimate price for Y

$$\mu(\boldsymbol{X}, \boldsymbol{D}) = \mathbb{E}\left[Y \middle| \boldsymbol{X}, \boldsymbol{D}\right].$$

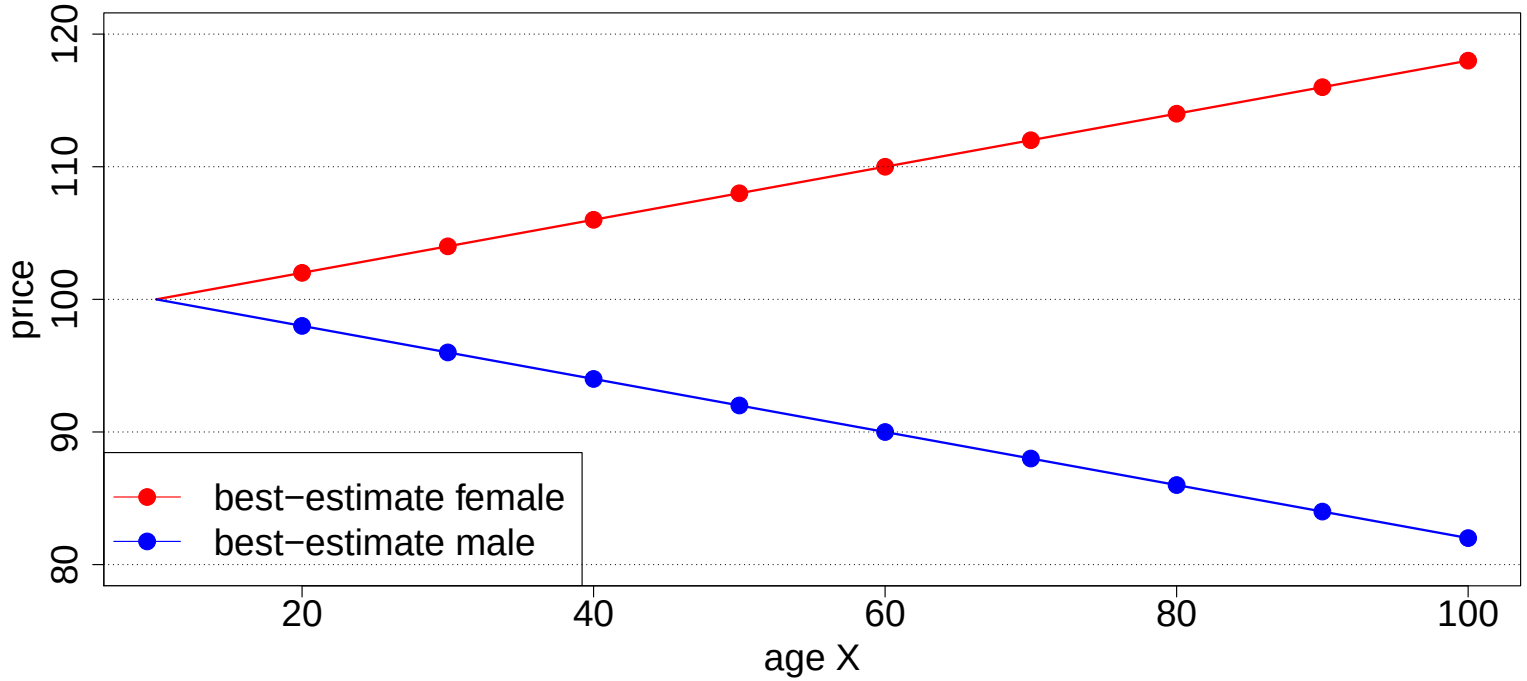
- The best-estimate price
  - ⋆ uses maximal available information $\boldsymbol{X}$ and $\boldsymbol{D}$;
  - ⋆ minimizes prediction uncertainty (in an L^2-sense);
  - ⋆ is discriminatory w.r.t. $\boldsymbol{D}$.

Best-Estimate Price: Example



- Best-estimate prices $\mu(\boldsymbol{X}, \boldsymbol{D})$ using all available information
  - ⋆ with non-discriminatory age information $\boldsymbol{X}$;
  - ⋆ with discriminatory gender information $\boldsymbol{D}$.

Best-Estimate Price: Direct Discrimination

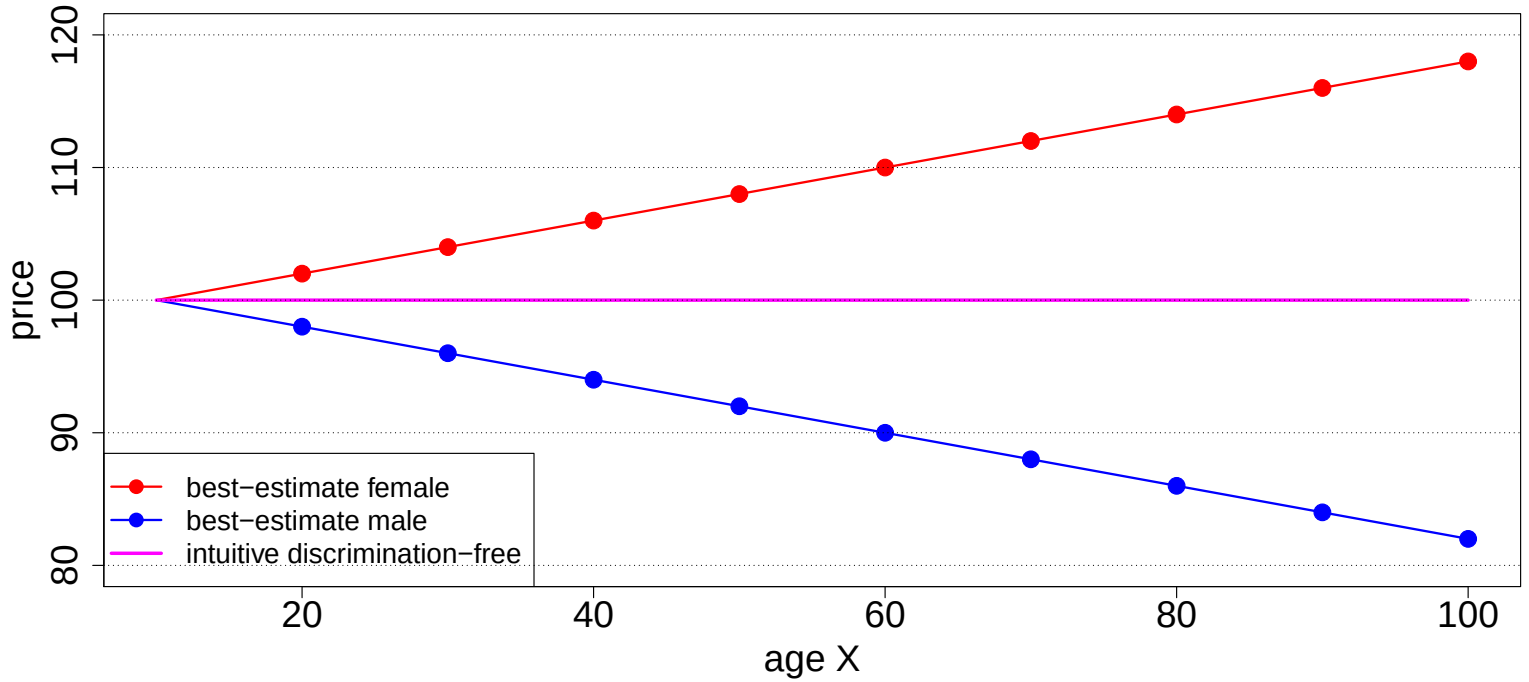


- Article 2(a):[1] "direct discrimination: where one person is treated less favourably, on grounds of sex..."

- Intuitive guess for discrimination-free price?

[1] COUNCIL DIRECTIVE 2004/113/EC of 13 December 2004, Official Journal of the European Union L 373/37

Best-Estimate Price: Direct Discrimination



- Article 2(a): "direct discrimination: where one person is treated less favourably, on grounds of sex..."

- Intuitive guess for discrimination-free price.

- **Unawareness Price and Indirect Discrimination**

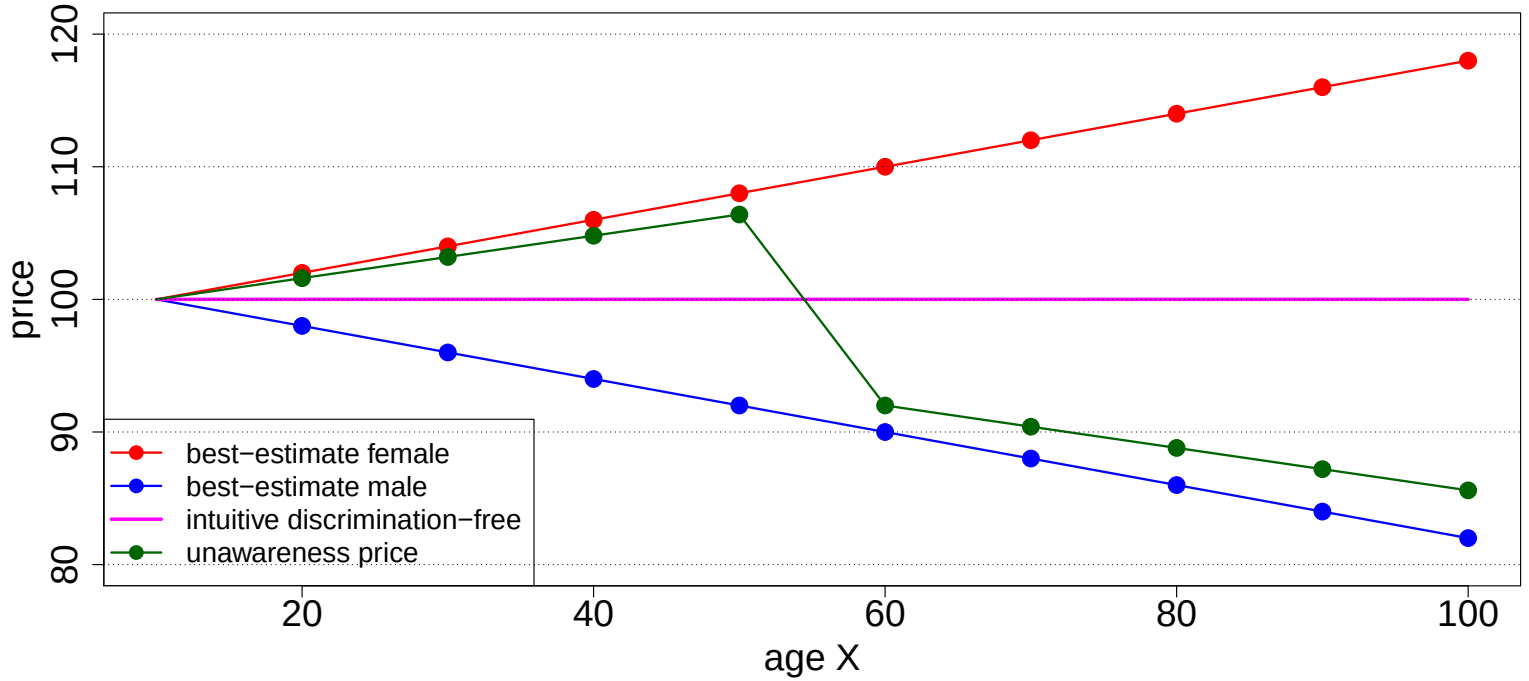
Unawareness Price

- Direct discrimination can be avoided by dropping discriminatory information $\boldsymbol{D}$.
- This provides unawareness price for Y

$$\mu(\boldsymbol{X}) = \mathbb{E}\left[Y \middle| \boldsymbol{X}\right].$$

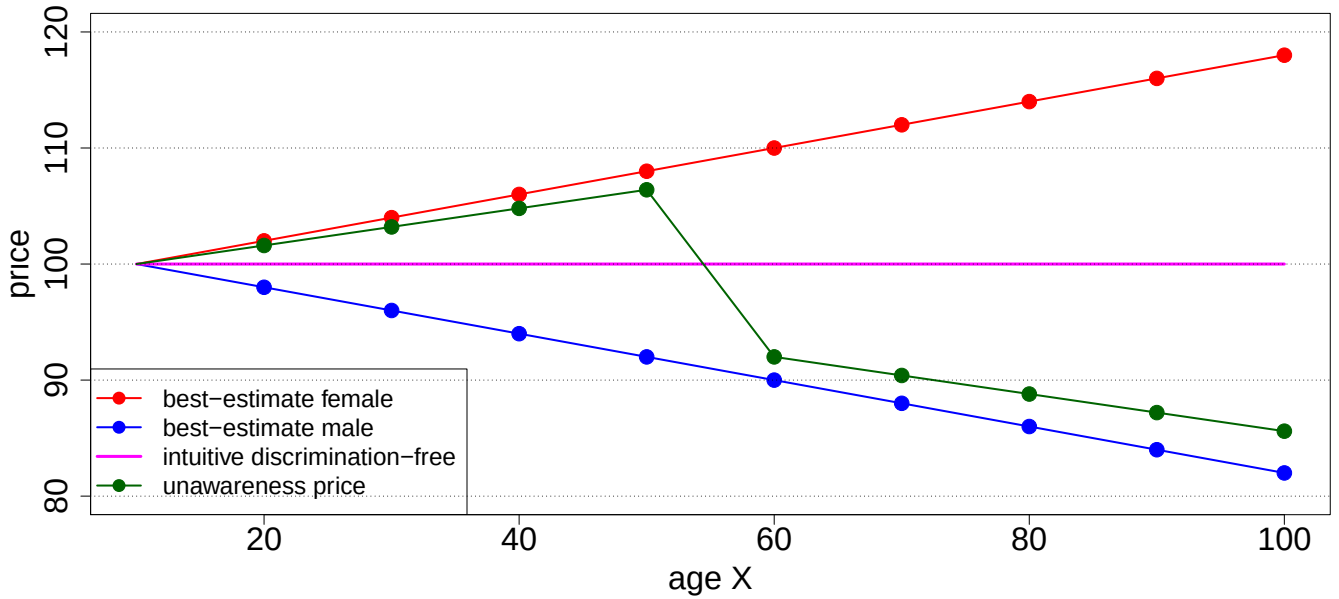
- The unawareness price
  - $\star$ uses maximal available non-discriminatory information $\boldsymbol{X}$;
  - $\star$ minimizes prediction uncertainty (in an L^2-sense w.r.t. $\boldsymbol{X}$);
  - $\star$ is the best approximation to the best-estimate price $\mu(\boldsymbol{X}, \boldsymbol{D})$;
  - $\star$ avoids direct discrimination.

Unawareness Price: Example

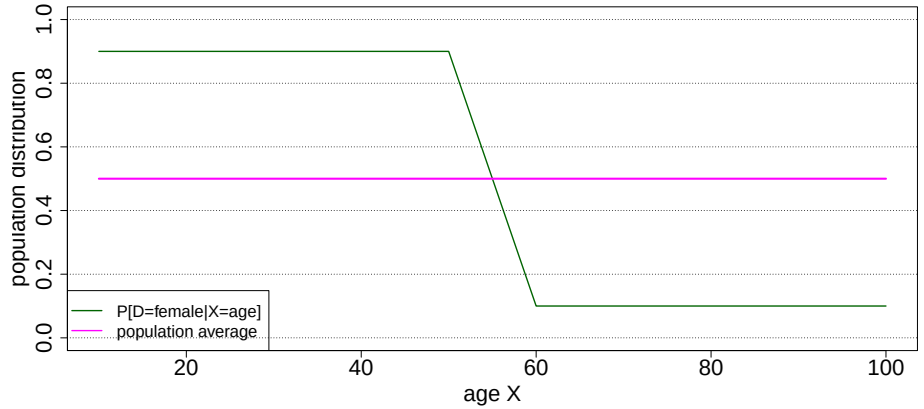


- What goes "wrong" here?

Unawareness Price: Example



- What goes "wrong" here?



What Goes "Wrong" with the Unawareness Price?

- The unawareness prices can be expressed as (tower property)

$$\begin{aligned} \mu(\boldsymbol{X}) &= \mathbb{E}\left[\mu(\boldsymbol{X},\boldsymbol{D})\middle|\,\boldsymbol{X}\right] \\ &= \int_{\boldsymbol{d}} \mu(\boldsymbol{X},\boldsymbol{D}=\boldsymbol{d})\ \mathrm{d}\mathbb{P}\left(\boldsymbol{D}=\boldsymbol{d}|\boldsymbol{X}\right). \end{aligned}$$

- This shows that we infer $\boldsymbol{D}$ from $\boldsymbol{X}$ in the unawareness price.

- Article 2(b):[2] "indirect discrimination: where an apparently neutral provision... would put persons of one sex at a particular disadvantage compared with persons of the other sex, unless that provision... is objectively justified..."

[2] COUNCIL DIRECTIVE 2004/113/EC of 13 December 2004, Official Journal of the European Union L 373/37

- **Discrimination-Free Price**

Discrimination-Free Pricing

- The unawareness prices can be expressed as (tower property)

$$\begin{aligned}\mu(\boldsymbol{X}) &= \mathbb{E}\left[\mu(\boldsymbol{X},\boldsymbol{D})\middle|\,\boldsymbol{X}\right] \\ &= \int_{\boldsymbol{d}} \mu(\boldsymbol{X},\boldsymbol{D}=\boldsymbol{d})\,\mathrm{d}\mathbb{P}\left(\boldsymbol{D}=\boldsymbol{d}|\boldsymbol{X}\right).\end{aligned}$$

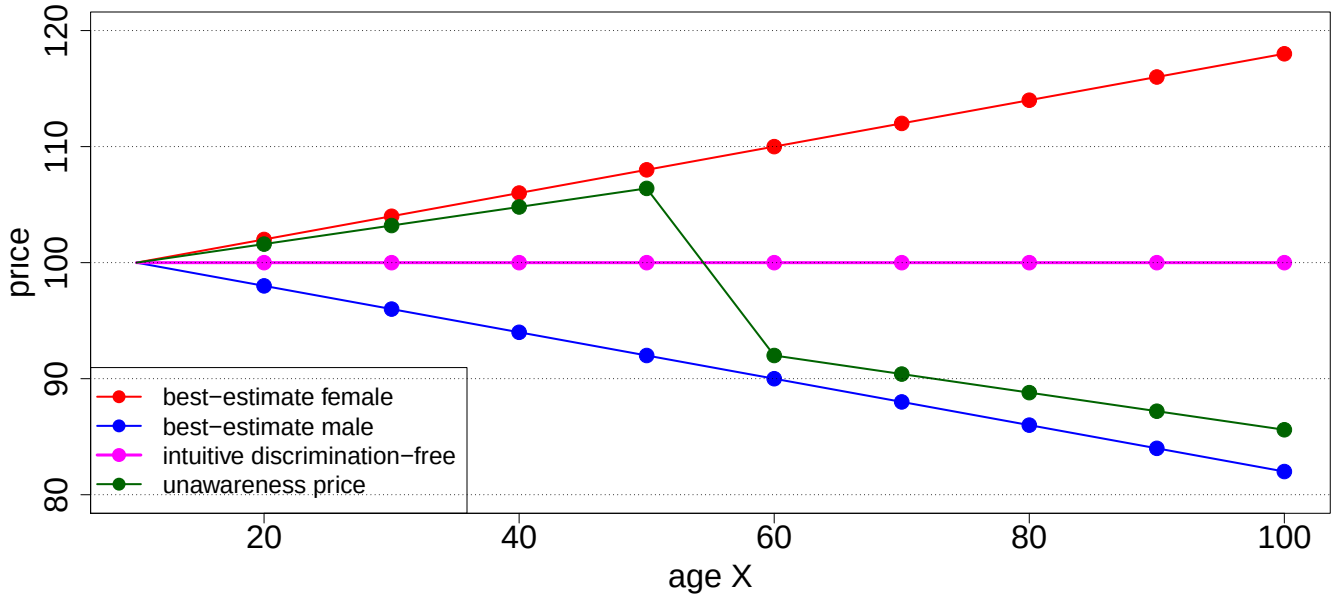
- We need to "break the structure" that allows to infer $\boldsymbol{D}$ from $\boldsymbol{X}$.

- This motivates discrimination-free price

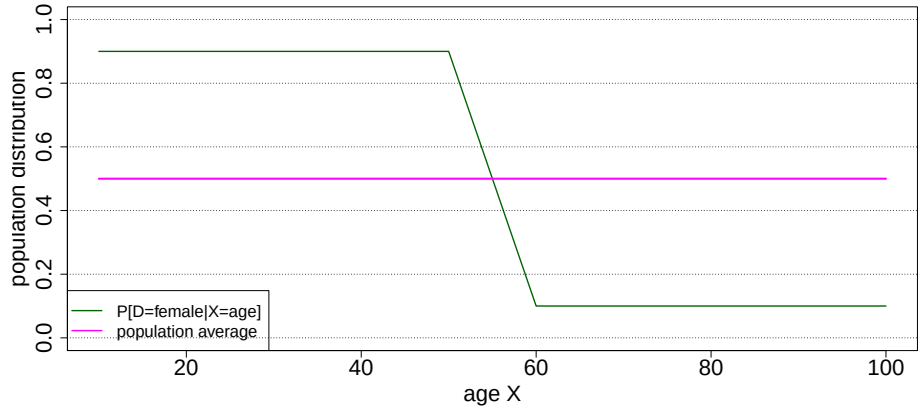
$$\mu^*(\boldsymbol{X}) = \int_{\boldsymbol{d}} \mu(\boldsymbol{X},\boldsymbol{D}=\boldsymbol{d})\,\mathrm{d}\mathbb{P}^*(\boldsymbol{D}=\boldsymbol{d}),$$

  for some choice $\mathbb{P}^*$ (there are infinitely many).

Discrimination-Free Price: Example



- For population distribution $\mathbb{P}^*(\boldsymbol{D}) = \mathbb{P}(\boldsymbol{D})$.



Concluding Remarks

- We need to collect discriminatory information $\boldsymbol{D}$, otherwise we cannot calculate discrimination-free prices, i.e. just knowledge of $\boldsymbol{X}$ is not good enough.
- Lindholm et al. (2020) give a mathematical definition of (in-)direct discrimination.
- For any given problem there are infinitely many choices $\mathbb{P}^*$, and henceforth there are infinitely many discrimination-free prices.
- Discrimination-free prices need to be made unbiased.
- Discrimination-free prices sacrifice predictive power relative to unawareness prices.
- Discrimination-free prices can be motivated by "do-operators" in causal statistics (confounders), see Pearl et al. (2016).
- Discrimination-free prices have same structure as partial dependence plots (PDPs), see Zhao–Hastie (2019) and Lorentzen–Mayer (2020).

- Definition of discrimination-free prices is independent of any model.

- Discrimination-free prices may induce unwanted economic side effects like adverse selection.

- Indirect discrimination can be explained by the fact that non-discriminatory covariates are used to predict discriminatory ones. The better information we have, the more accurately this can be done.

- We did not discuss fairness nor which variables are discriminatory (ethnicity, etc.).

References

- Chen, Guillén, Vigna (2018). Solvency requirement in a unisex mortality model. ASTIN Bulletin 48/3, 1219-1243.
- Chen, Vigna (2017). A unisex stochastic mortality model to comply with EU Gender Directive. Insurance: Mathematics and Economics 73, 124-136.
- Frees, Huang (2020). The discriminating (pricing) actuary. SSRN 3592475.
- Lindholm, Richman, Tsanakas, Wüthrich (2020). Discrimination-free insurance pricing. SSRN 3520676. To appear in ASTIN Bulletin 2022.
- Lorentzen, Mayer (2020). Peeking into the black box: an actuarial case study for interpretable machine learning. SSRN 3595944.
- Pearl, Glymour, Jewell (2016). Causal Inference in Statistics: A Primer. Wiley.
- Zhao, Hastie (2019). Causal interpretations of black-box models. Journal of Business & Economic Statistics.

LocalGLMnet and more

Mario V. Wüthrich
RiskLab, ETH Zurich

"Deep Learning with Actuarial Applications in R"
Swiss Association of Actuaries SAA/SAV, Zurich
October 14/15, 2021

Programme SAV Block Course

- Refresher: Generalized Linear Models (THU 9:00-10:30)
- Feed-Forward Neural Networks (THU 13:00-15:00)
- Discrimination-Free Insurance Pricing (THU 17:15-17:45)

- LocalGLMnet (FRI 9:00-10:30)
- Convolutional Neural Networks (FRI 13:00-14:30)
- Wrap Up (FRI 16:00-16:30)

Contents: LocalGLMnet and more

- Balance property for neural networks
- Multiplicity of equally good FNN models
- The nagging predictor
- LocalGLMnet: interpretable deep learning

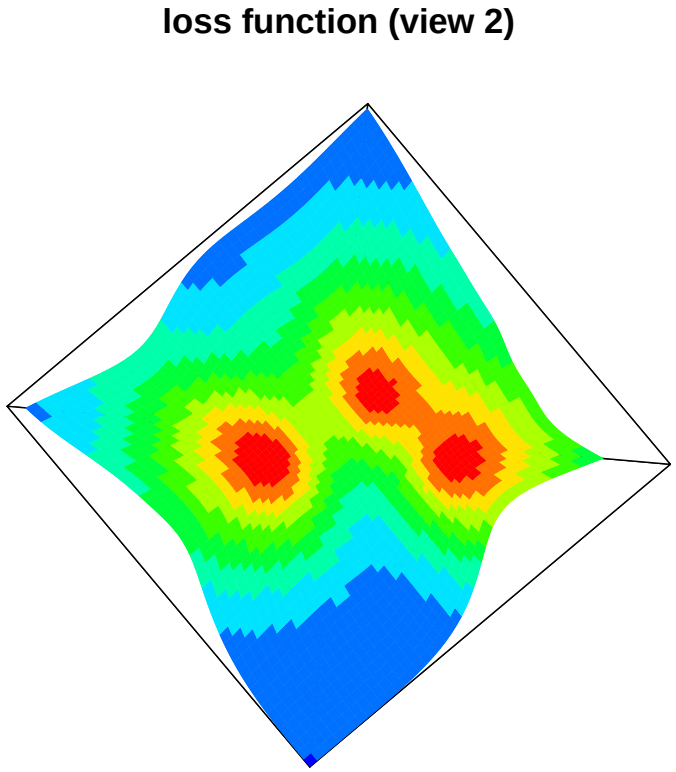
- **Balance Property for Neural Networks**

Balance Property for FNN Models

	epochs	run time	# param.	in-sample loss in 10^{-2}	out-of-sample loss in 10^{-2}	average frequency
homogeneous model		–	1	32.935	33.861	10.02%
Model GLM1		20s	49	31.267	32.171	10.02%
Deep FNN One-Hot	250	152s	1'306	30.268	31.673	10.19%
Deep FNN Emb($b = 1$)	700	419s	719	30.245	31.506	9.90%
Deep FNN Emb($b = 2$)	600	365s	792	30.165	31.453	9.70%
Deep FNN Emb($b = 2$) seed 1		365s	792	30.411	31.503	9.90%
Deep FNN Emb($b = 2$) seed 2		365s	792	30.352	31.418	10.23%
Deep FNN Emb($b = 2$) seed 3		365s	792	30.315	31.500	9.61%

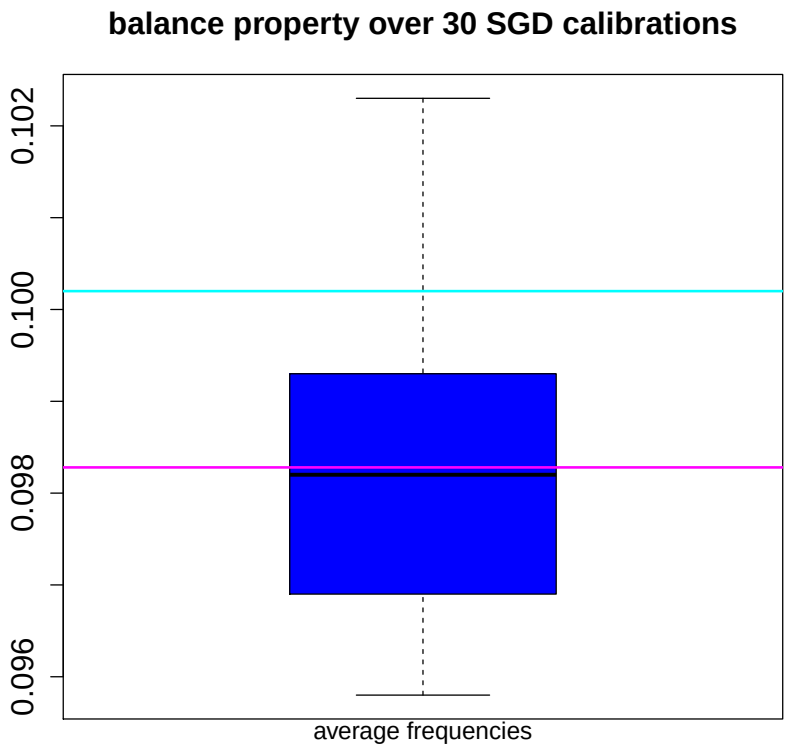
- Balance property fails to hold for FNN regression models.
- The reason is early stopping which prevents from being in a critical point of the deviance loss $D^*(\boldsymbol{Y}, \cdot)$ (under canonical link choice).

Critical Points of Deviance Loss Function



Under the canonical link choice, the balance property is fulfilled in the critical points.

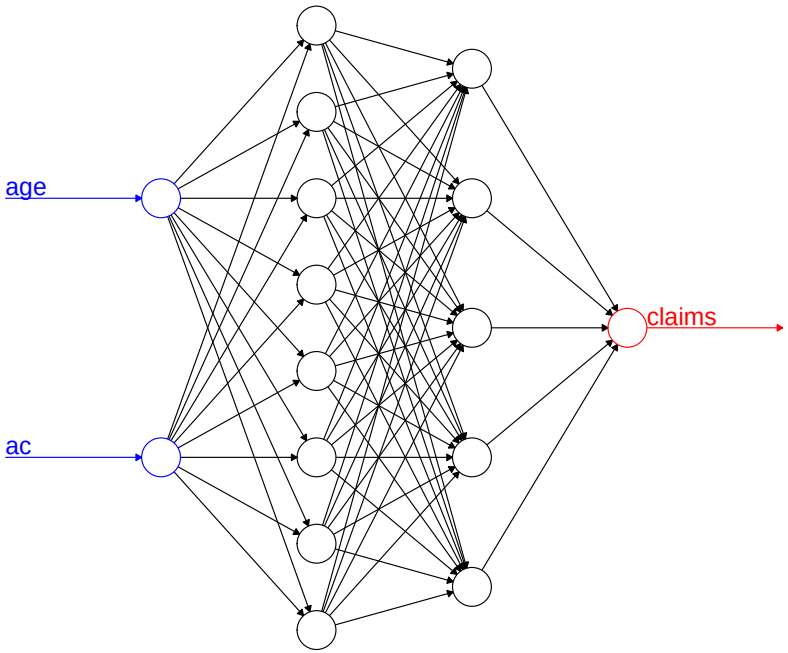
Failure of Balance Property of FNNs

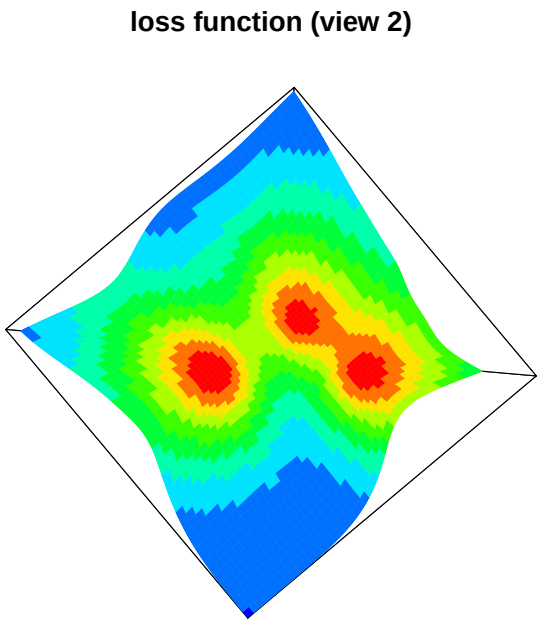


The failure of the balance property is significant.

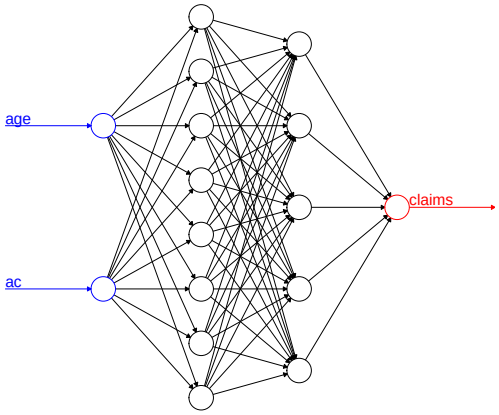
Seeds and Randomness Involved in SGD

```
1 layer_dense(units=q1, activation = "tanh",
2                       kernel_initializer = initializer_glorot_uniform(),
3                       bias_initializer='zeros') %>%
4 layer_dropout(rate=0.05)
5
6 model %>% fit(X, Y, validation_split = 0.2, batch_size = 10000, epochs = 500)
```





Representation Learning: Additional GLM Step



- Network mapping with link g

$$\boldsymbol{x}_i \;\mapsto\; g(\widehat{\mu}_i) = g(\widehat{\mathbb{E}}[Y_i]) = \left\langle \widehat{\boldsymbol{\beta}}, \widehat{\boldsymbol{z}}^{(d:1)}(\boldsymbol{x}_i) \right\rangle,$$

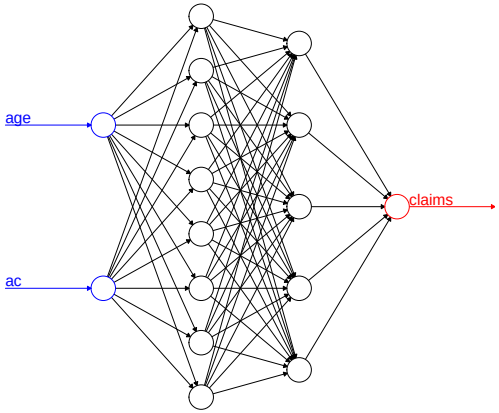
  with GDM fitted network parameter $\widehat{\vartheta} = (\widehat{\boldsymbol{w}}, \widehat{\boldsymbol{\beta}}) \in \mathbb{R}^r$.

- Mapping $\boldsymbol{x}_i \mapsto \widehat{\boldsymbol{z}}_i = \widehat{\boldsymbol{z}}^{(d:1)}(\boldsymbol{x}_i)$ should be understood as learned representation.

- **Idea** (with canonical link choice $g = h$): consider a GLM with new covariates $\widehat{\boldsymbol{z}}_i$.

- If design matrix $\widehat{\mathfrak{Z}} \in \mathbb{R}^{n \times (q_d+1)}$ has full rank $q_d + 1 \leq n$, we have a unique MLE $\widehat{\boldsymbol{\beta}}^{\mathrm{MLE}}$, and the balance property will be fulfilled under canonical link choice.

Rectifying the Balance Property



- Network mapping with link g

$$\boldsymbol{x}_i \;\mapsto\; g(\widehat{\mu}_i) = g(\widehat{\mathbb{E}}[Y_i]) = \left\langle \widehat{\boldsymbol{\beta}}, \widehat{\boldsymbol{z}}^{(d:1)}(\boldsymbol{x}_i) \right\rangle,$$

  with GDM fitted network parameter $\widehat{\vartheta} = (\widehat{\boldsymbol{w}}, \widehat{\boldsymbol{\beta}}) \in \mathbb{R}^r$.

- Choose $\widehat{\boldsymbol{\beta}}^{\mathrm{MLE}}$ for design matrix $\widehat{\mathfrak{Z}} \in \mathbb{R}^{n \times (q_d+1)}$

$$\boldsymbol{x}_i \;\mapsto\; h(\widehat{\mu}_i) = h(\widehat{\mathbb{E}}[Y_i]) = \left\langle \widehat{\boldsymbol{\beta}}^{\mathrm{MLE}}, \widehat{\boldsymbol{z}}^{(d:1)}(\boldsymbol{x}_i) \right\rangle,$$

  for canonical link h.

R Code for Implementing the Balance Property

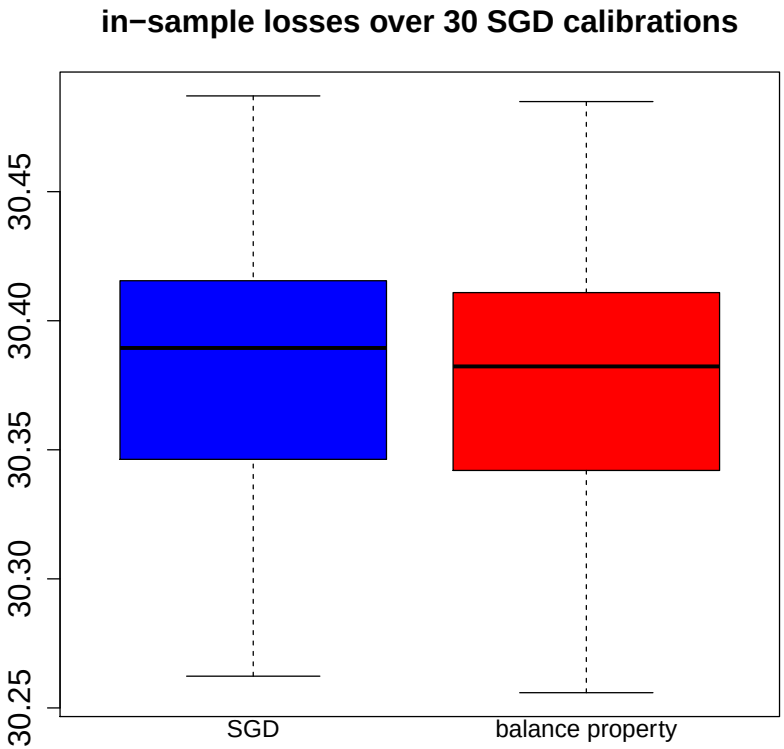
```
z.layer  <- keras_model(inputs=model$input,
                        outputs=get_layer(model,'hidden3')$output)

learn[,c("z1","z2","z3","z4","z5","z6","z7","z8","z9","z10")] <-
            data.frame(z.layer %>% predict(list(Design, VehBrand, Region, LogVol)))

glm(ClaimNb ~ z1 + z2 + z3 + z4 + z5 + z6 + z7 + z8 + z9 + z10,
            data=learn, offset=log(Exposure), family=poisson())
```

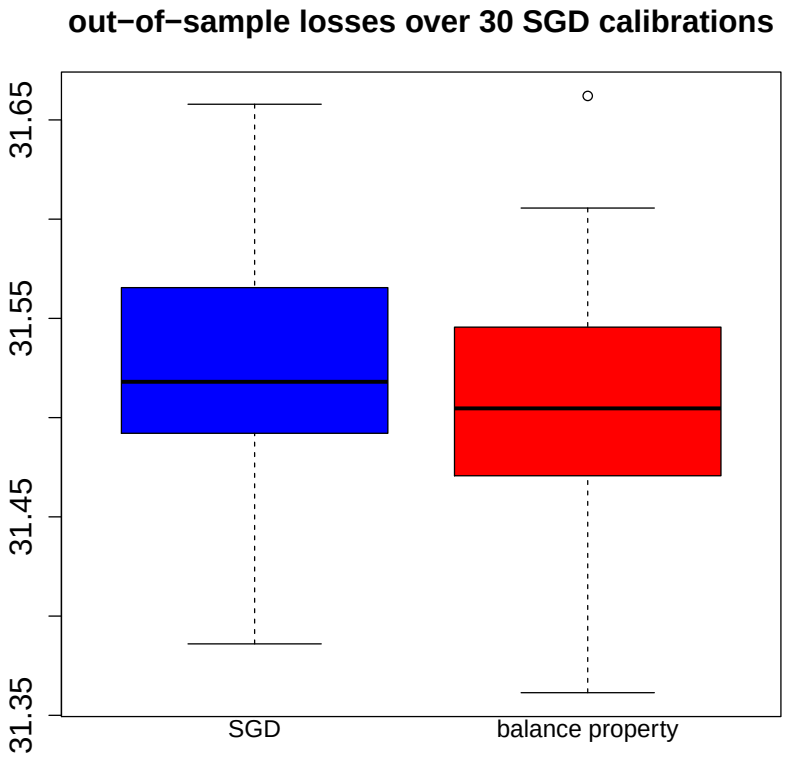
- The R code considers the Poisson model with canonical link $g = h = \log$.
- If we do not have canonical link we still need to adjust the intercept $\widehat{\beta}_0^{\mathrm{MLE}}$.
- The additional GLM step may lead to over-fitting, if the size q_d of the last hidden FNN layer is not too large, there won't be over-fitting.

Balance Property for FNN Models

	epochs	run time	# param.	in-sample loss in 10^{-2}	out-of-sample loss in 10^{-2}	average frequency
homogeneous model		–	1	32.935	33.861	10.02%
Model GLM1		20s	49	31.267	32.171	10.02%
Deep FNN One-Hot	250	152s	1'306	30.268	31.673	10.19%
Deep FNN Emb($b = 1$)	700	419s	719	30.245	31.506	9.90%
Deep FNN Emb($b = 2$)	600	365s	792	30.165	31.453	9.70%
Deep FNN Emb($b = 2$) seed 1		365s	792	30.411	31.503	9.90%
Deep FNN Emb($b = 2$) seed 2		365s	792	30.352	31.418	10.23%
Deep FNN Emb($b = 2$) seed 3		365s	792	30.315	31.500	9.61%
Reg. FNN Emb($b = 2$) seed 1		+7s	792	30.408	31.488	10.02%
Reg. FNN Emb($b = 2$) seed 2		+7s	792	30.346	31.418	10.02%
Reg. FNN Emb($b = 2$) seed 3		+7s	792	30.303	31.462	10.02%

Balance Property for FNN Models

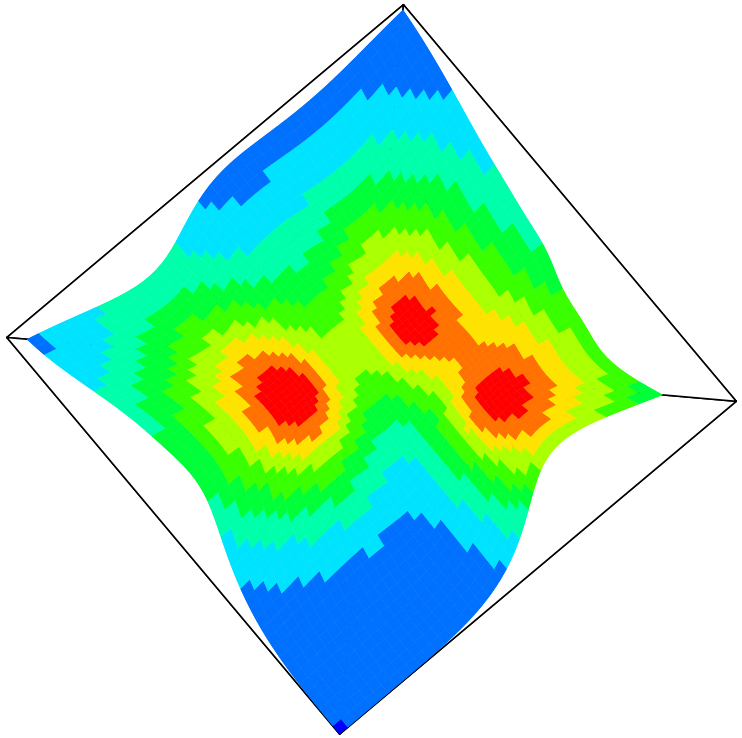




- **The "Best" FNN Regression Model**

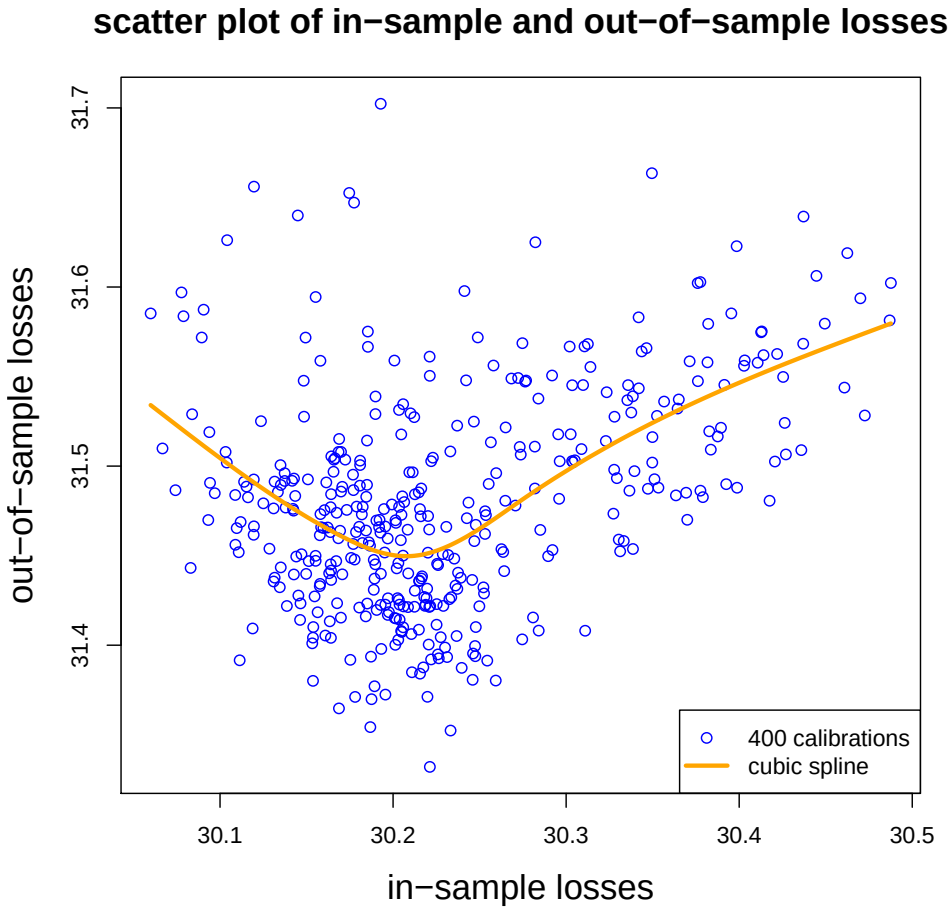
Multiplicity of Equally Good FNN Models

loss function (view 2)



- Many network parameters ϑ produce the same loss figure for a given objective function $\vartheta \mapsto D^*(\boldsymbol{Y}, \vartheta)$, i.e. they are "equally good" (on portfolio level).
- The chosen network solution will depend in the initial seed of the algorithm.
- This is very troublesome for insurance pricing!

Scatter Plot: In-Sample vs. Out-of-Sample Losses



This example is taken from Richman–Wüthrich (2020) and the in-sample losses are smaller than in the table above because we used different data cleaning.

- **The Nagging Predictor**

The Nagging Predictor

- Breiman (1996) uses bootstrap aggregating = bagging to reduce noise in predictors.

- Aggregate over different network predictors of different calibrations $j \geq 1$

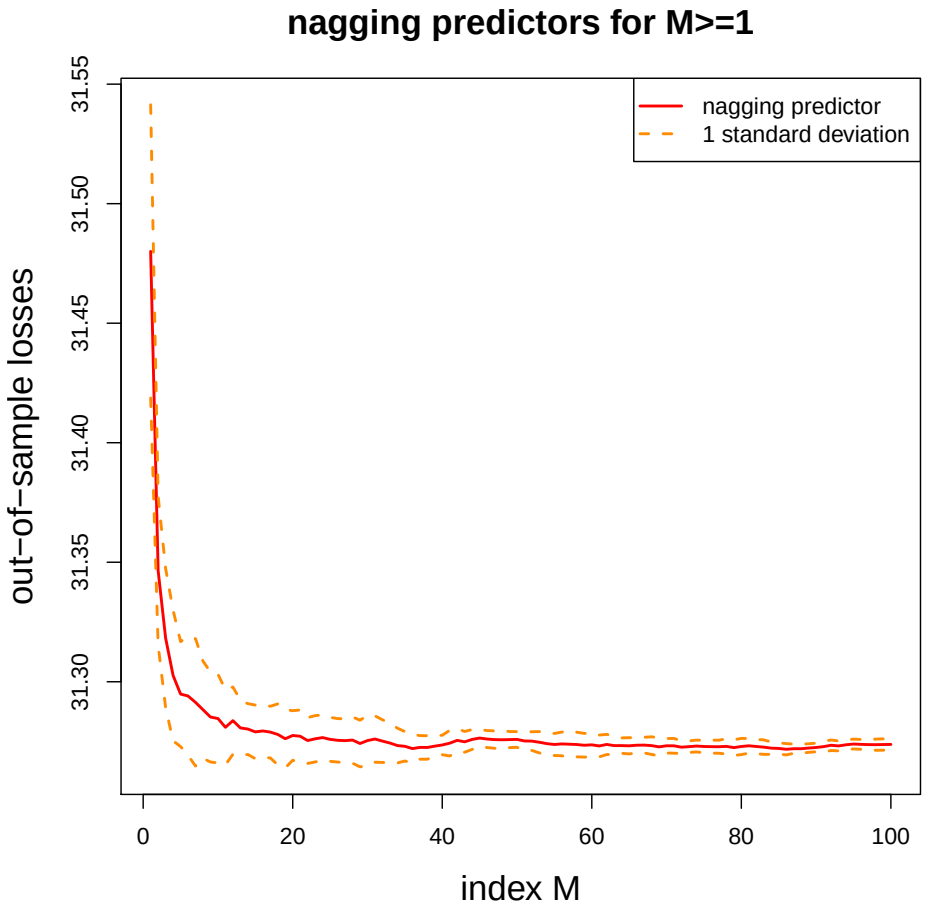
$$\bar{\mu}_i^{(M)} \;=\; \frac{1}{M}\sum_{j=1}^{M} \widehat{\mu}_i^{(j)} \;=\; \frac{1}{M}\sum_{j=1}^{M} \mu_{\widehat{\vartheta}^{(j)}}(\boldsymbol{x}_i).$$

- **Theorem (generalization loss).**
  Under suitable assumptions we have for deviance loss $D^*(Y_i, \cdot)$

$$\mathbb{E}\left[D^*\left(Y_i, \bar{\mu}_i^{(M)}\right)\right] \;\geq\; \mathbb{E}\left[D^*\left(Y_i, \bar{\mu}_i^{(M+1)}\right)\right] \;\geq\; \mathbb{E}\left[D^*(Y_i, \mu_i)\right],$$

  and there is also a corresponding asymptotic normality result (CLT).

Nagging Predictor: Car Insurance Example



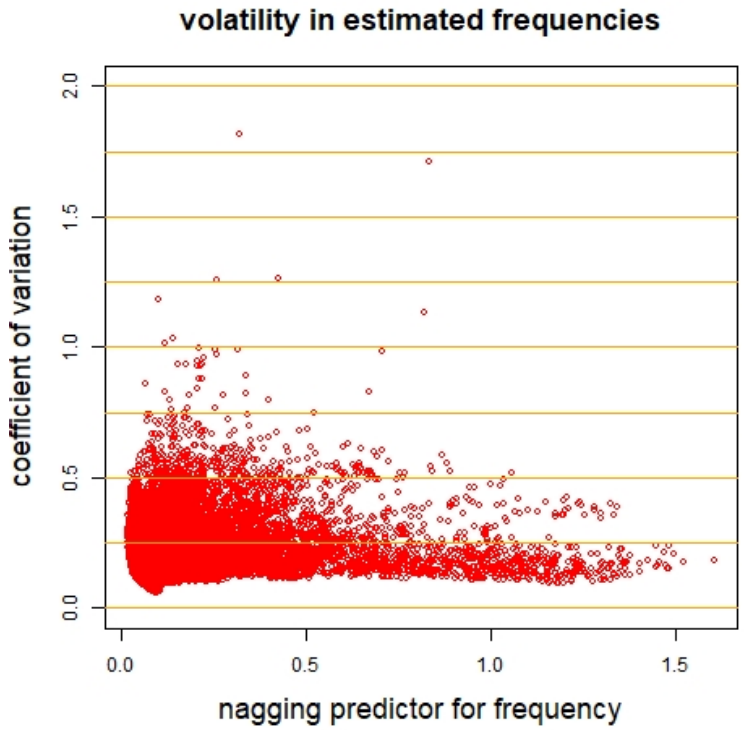
- After $M = 20$ iterations: the out-of-sample loss on portfolio has converged.
- After $M = 40$ iterations: confidence bounds are narrow.

Car Insurance Frequency Poisson Example

	epochs	run time	# param.	in-sample loss in 10^{-2}	out-of-sample loss in 10^{-2}	average frequency
homogeneous model		–	1	32.935	33.861	10.02%
Model GLM1		20s	49	31.267	32.171	10.02%
Deep FNN One-Hot	250	152s	1'306	30.268	31.673	10.19%
Deep FNN Emb($b = 1$)	700	419s	719	30.245	31.506	9.90%
Deep FNN Emb($b = 2$)	600	365s	792	30.165	31.453	9.70%
Deep FNN Emb($b = 2$) seed 1		365s	792	30.411	31.503	9.90%
Deep FNN Emb($b = 2$) seed 2		365s	792	30.352	31.418	10.23%
Deep FNN Emb($b = 2$) seed 3		365s	792	30.315	31.500	9.61%
Reg. FNN Emb($b = 2$) seed 1		+7s	792	30.408	31.488	10.02%
Reg. FNN Emb($b = 2$) seed 2		+7s	792	30.346	31.418	10.02%
Reg. FNN Emb($b = 2$) seed 3		+7s	792	30.303	31.462	10.02%
Nagging predictor for $M = 400$				30.060	31.272	10.02%

The nagging predictor leads to a clear model improvement.

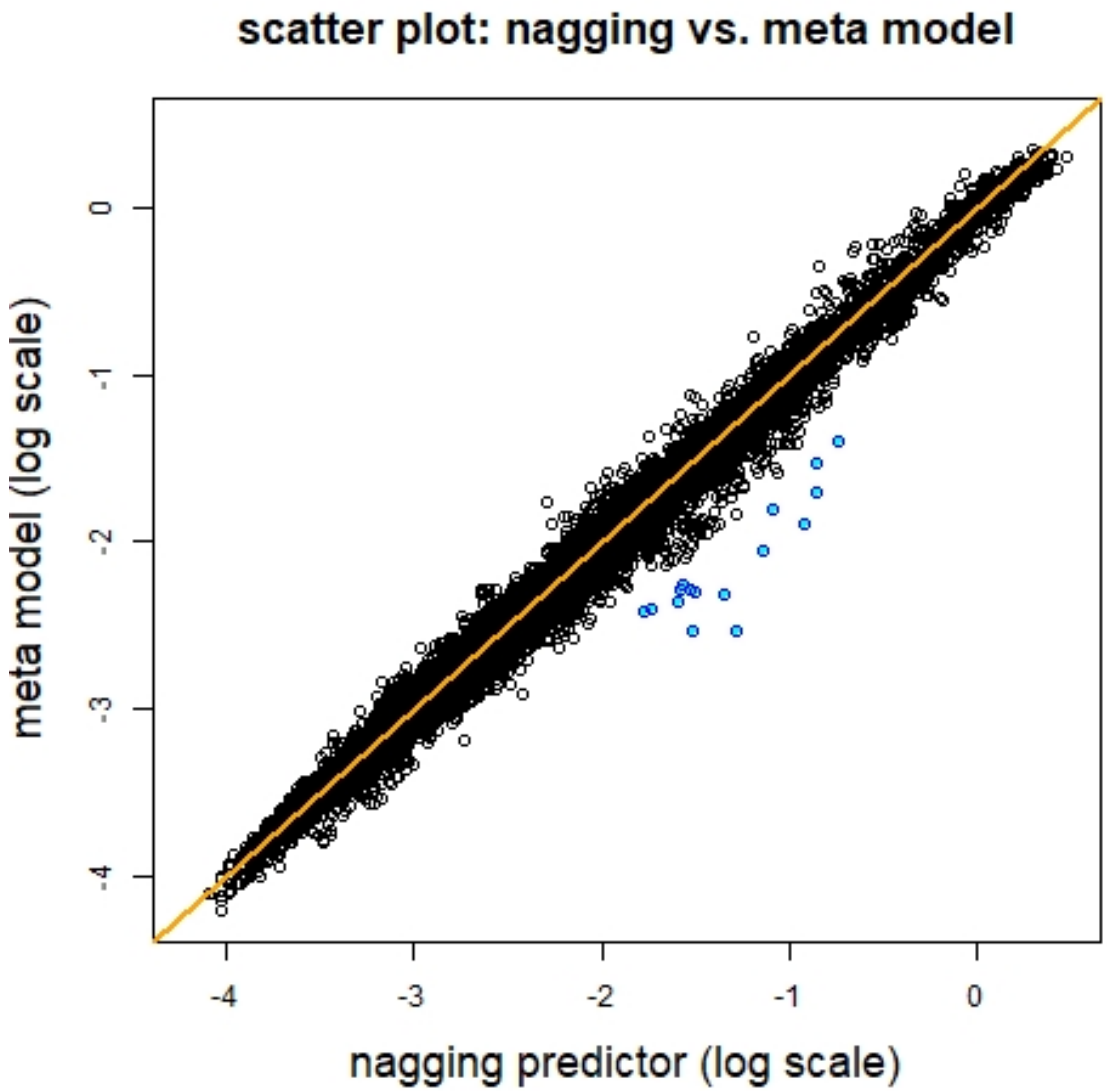
Stability on Individual Policy Level



$$\widehat{\mathrm{CoV}}_i \;=\; \frac{\widehat{\sigma}_i}{\bar{\mu}_i^{(M)}} \;=\; \sqrt{\frac{1}{M-1}\sum_{j=1}^{M}\left(\widehat{\mu}_i^{(j)} - \bar{\mu}_i^{(M)}\right)^2} \Bigg/ \bar{\mu}_i^{(M)}.$$

- Individual policy level: average over 400 networks to get coefficient of variation (CoV) of $1/\sqrt{400} = 5\%$.

Fit Meta Model to Nagging Predictor



- **LocalGLMnet: interpretable deep learning**

Explainability of Deep FNN Predictors

- Network predictors are criticized for not being explainable (black box).
- There are many tools to make network predictors explainable a posteriori:
  - ⋆ partial dependence plots (PDPs)
  - ⋆ accumulated local effects (ALEs)
  - ⋆ locally interpretable model-agnostic explanation (LIME)
  - ⋆ SHapley Additive exPlanation (SHAP)
  - ⋆ marginal attribution by conditioning on quantiles (MACQ)
- Network predictors do not allow for variable selection.
- LocalGLMnet is an architecture that is explainable and allows for variable selection.

LocalGLMnet Architecture

- A GLM has the following regression structure for parameter $\boldsymbol{\beta} \in \mathbb{R}^q$

$$g(\mu(\boldsymbol{x})) = \beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle = \beta_0 + \sum_{j=1}^{q} \beta_j x_j.$$

- **Idea:** Estimate regression parameter $\boldsymbol{\beta} = \boldsymbol{\beta}(\boldsymbol{x})$ with a FNN.

- Define regression attentions through a deep FNN

$$\begin{aligned} \boldsymbol{\beta} : \mathbb{R}^q &\to \mathbb{R}^q \\ \boldsymbol{x} &\mapsto \boldsymbol{\beta}(\boldsymbol{x}) = \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) = \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)} \right)(\boldsymbol{x}). \end{aligned}$$

- The LocalGLMnet is defined by the additive decomposition

$$g(\mu(\boldsymbol{x})) = \beta_0 + \langle \boldsymbol{\beta}(\boldsymbol{x}), \boldsymbol{x} \rangle = \beta_0 + \sum_{j=1}^{q} \beta_j(\boldsymbol{x}) x_j.$$

Interpretation of LocalGLMnet Architecture

- The LocalGLMnet is defined by the additive decomposition

$$g(\mu(\boldsymbol{x})) = \beta_0 + \langle \boldsymbol{\beta}(\boldsymbol{x}), \boldsymbol{x} \rangle = \beta_0 + \sum_{j=1}^{q} \beta_j(\boldsymbol{x}) x_j.$$

- We consider the different cases of regression attentions $\beta_j(\boldsymbol{x})$:
  - ⋆ If $\beta_j(\boldsymbol{x}) \equiv \beta_j$: GLM term $\beta_j x_j$.
  - ⋆ If $\beta_j(\boldsymbol{x}) \equiv 0$: drop term x_j.
  - ⋆ If $\beta_j(\boldsymbol{x}) = \beta_j(x_j)$: no interactions with covariates $x_{j'}$ for $j' \neq j$.
  - ⋆ Test for interactions: Calculate and analyze gradients

$$\nabla \beta_j(\boldsymbol{x}) = \left(\frac{\partial}{\partial x_1} \beta_j(\boldsymbol{x}), \ldots, \frac{\partial}{\partial x_q} \beta_j(\boldsymbol{x}) \right)^\top \in \mathbb{R}^q.$$

  - ⋆ Careful, there is no identifiability: $\beta_j(\boldsymbol{x}) x_j = x_{j'}$.

Implementation of LocalGLMnet

```
1  Design = layer_input(shape = c(q0),  dtype = 'float32', name = 'Design')
2  #
3  Attention = Design %>%
4            layer_dense(units=q1, activation='tanh', name='hidden1') %>%
5            layer_dense(units=q2, activation='tanh', name='hidden2') %>%
6            layer_dense(units=q3, activation='tanh', name='hidden3') %>%
7            layer_dense(units=q0, activation='linear', name='Attention')
8
9  Response  = list(Design, Attention) %>% layer_dot(axes=1) %>%
10           layer_dense(units=1, activation='linear', name='Response')
11 #
12 model <- keras_model(inputs = c(Design), outputs = c(Response))
```

Line 10 is needed to bring in the intercept β_0; but there are other ways to do so, see also next slide for explanation.

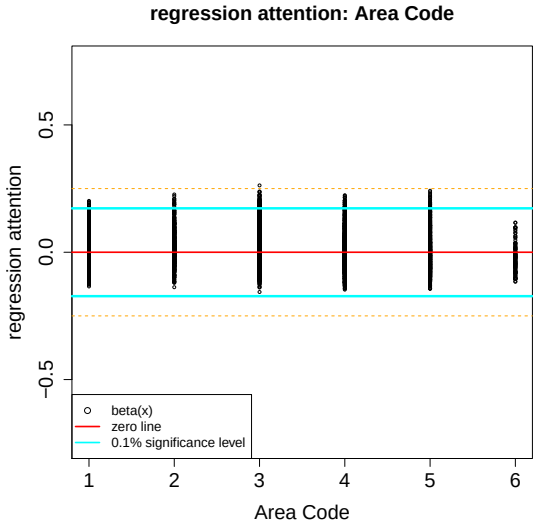
Remarks and Preparation of Example

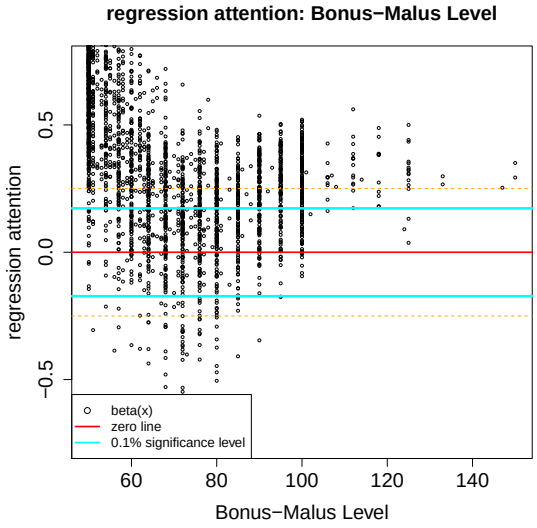
- There are other ways in implementing the intercept. Current solution:

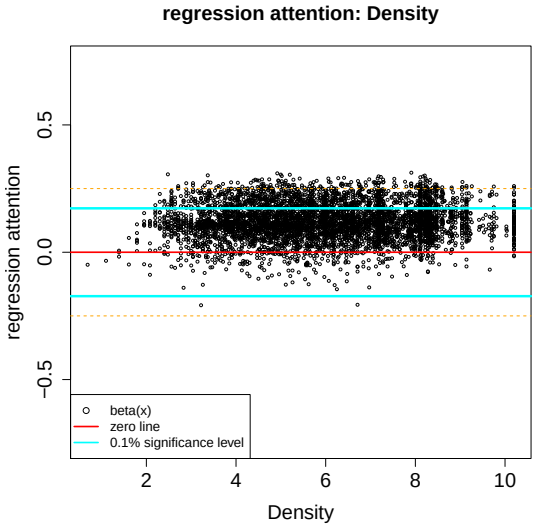
$$g(\mu(\boldsymbol{x})) = \alpha_0 + \alpha_1 \sum_{j=1}^{q} \beta_j(\boldsymbol{x}) x_j.$$

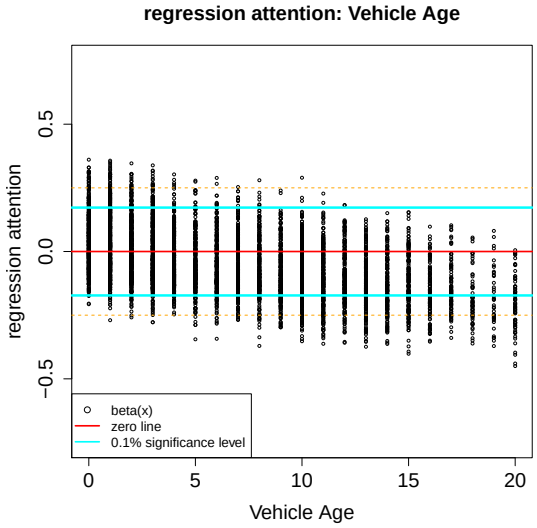
- Categorical variables should either use:
  - ⋆ one-hot encoding
  - ⋆ dummy coding with normalization to centering and unit variance
  - ▷ These codings will allow for interaction of all levels.
- We normalize all covariates to centering and unit variance: comparability!
- Fitting is done completely analogously to a FNN with SGD. Resulting networks are competitive with classical FNNs.

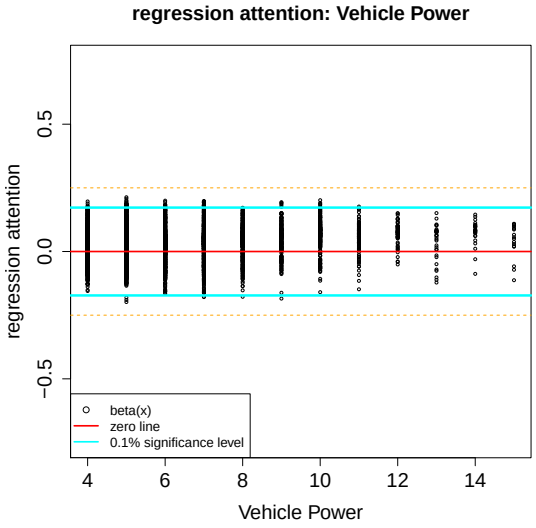
Example: Regression Attentions LocalGLMnet











Confidence Bounds for Variable Selection

- Add (two) purely random covariates (with different distributions)

$$x_{i,q+1} \overset{\text{i.i.d.}}{\sim} U\left[-\sqrt{3}, \sqrt{3}\right] \qquad \text{and} \qquad x_{i,q+2} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1).$$

  These covariates are standardized.

- Consider extended regression function for $\boldsymbol{x}^+ = (x_1, \ldots, x_q, x_{q+1}, x_{q+2})^\top \in \mathbb{R}^{q+2}$

$$g(\mu(\boldsymbol{x}^+)) = \beta_0^+ + \left\langle \boldsymbol{\beta}^+(\boldsymbol{x}^+), \boldsymbol{x}^+ \right\rangle = \beta_0^+ + \sum_{j=1}^{q+2} \beta_j^+(\boldsymbol{x}^+) x_j.$$

- Magnitudes of $\widehat{\beta}_{q+1}^+(\boldsymbol{x}^+)$ and $\widehat{\beta}_{q+2}^+(\boldsymbol{x}^+)$ determine size of insignificant components.

Confidence Bounds for Variable Selection

- Magnitudes of $\widehat{\beta}^+_{q+1}(\boldsymbol{x}^+)$ and $\widehat{\beta}^+_{q+2}(\boldsymbol{x}^+)$ determine size of insignificant components.
- Determine empirical mean and standard deviations for $j = q+1, q+2$

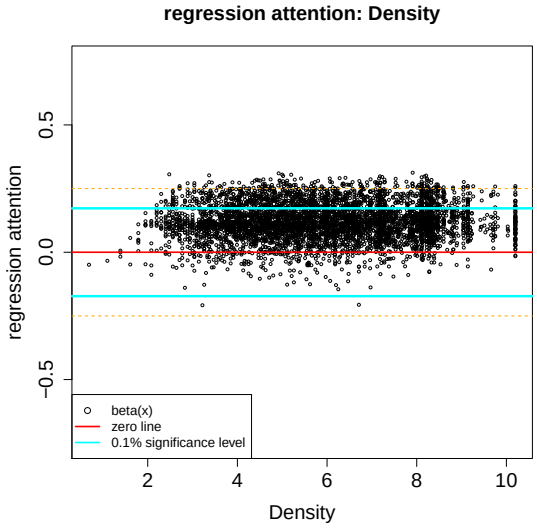
$$\bar{b}_j = \frac{1}{n}\sum_{i=1}^{n} \widehat{\beta}^+_j(\boldsymbol{x}^+_i) \quad \text{and} \quad \widehat{s}_j = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(\widehat{\beta}^+_j(\boldsymbol{x}^+_i) - \bar{b}_j\right)^2}.$$

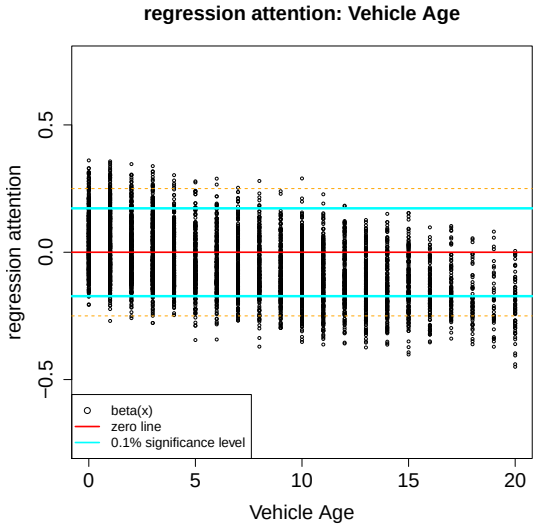
- Null hypothesis $H_0 : \beta_j(\boldsymbol{x}) = 0$ for component j on significance level $\alpha \in (0, 1/2)$ can be rejected if the coverage ratio of the following interval

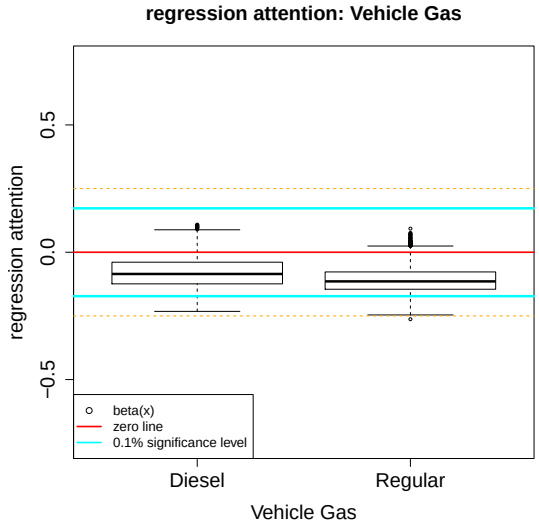
$$I_\alpha = \Big[q_{\mathcal{N}}(\alpha/2) \cdot \widehat{s}_{q+1},\ q_{\mathcal{N}}(1-\alpha/2) \cdot \widehat{s}_{q+1}\Big]$$

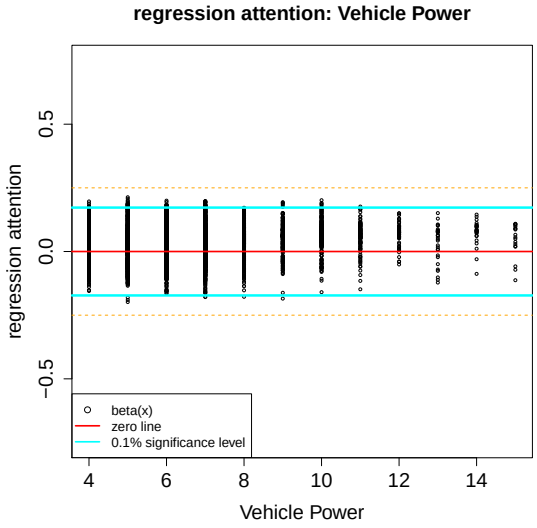
is substantially smaller than $1-\alpha$, where $q_{\mathcal{N}}(p)$ denotes the standard Gaussian quantile on level $p \in (0,1)$.

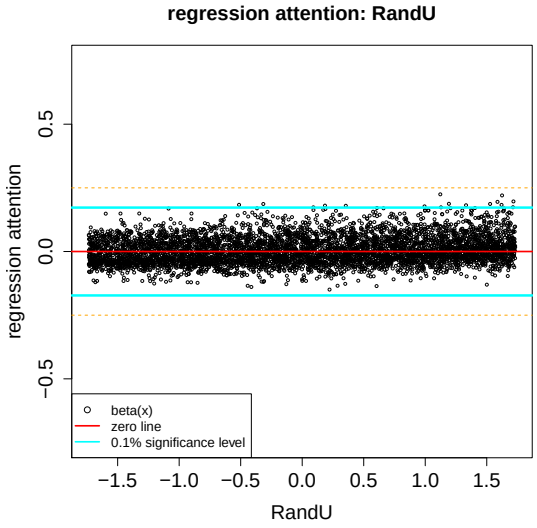
Variable Selection with Cyan Confidence Bounds

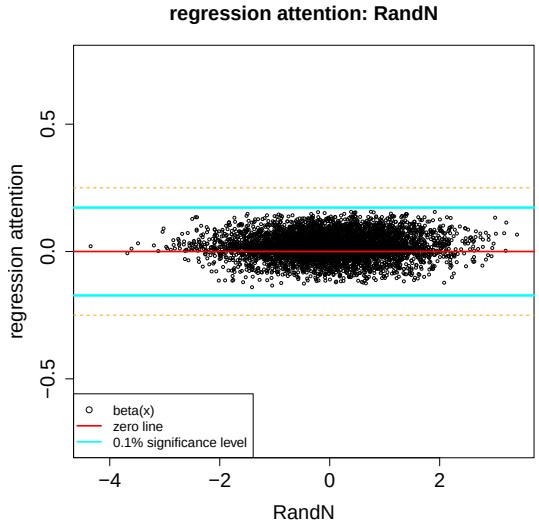




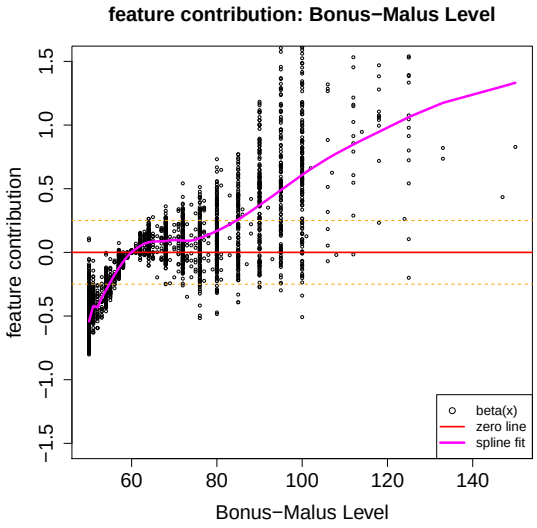


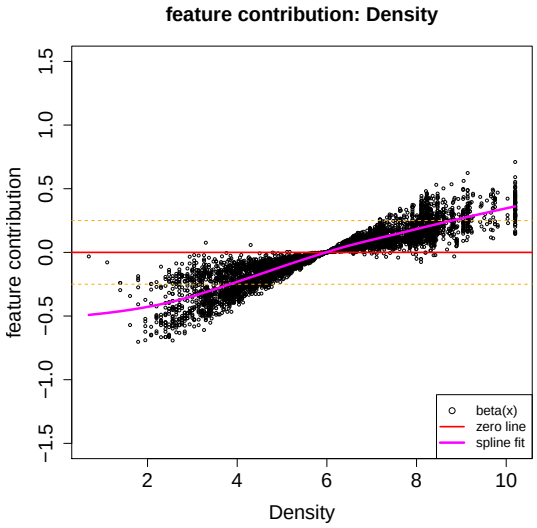


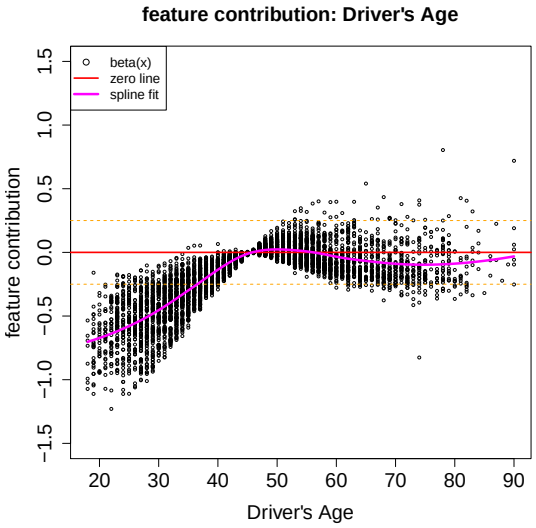


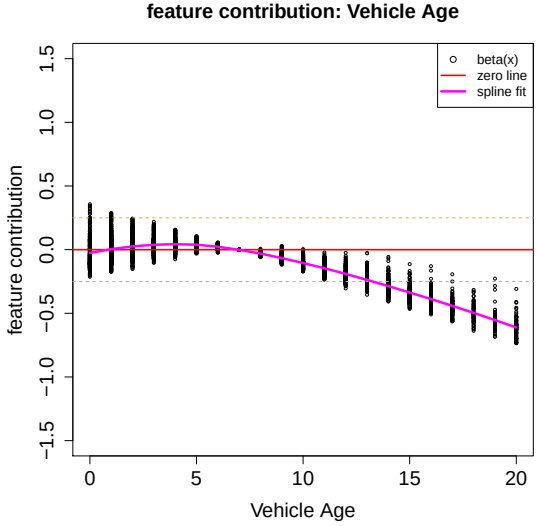


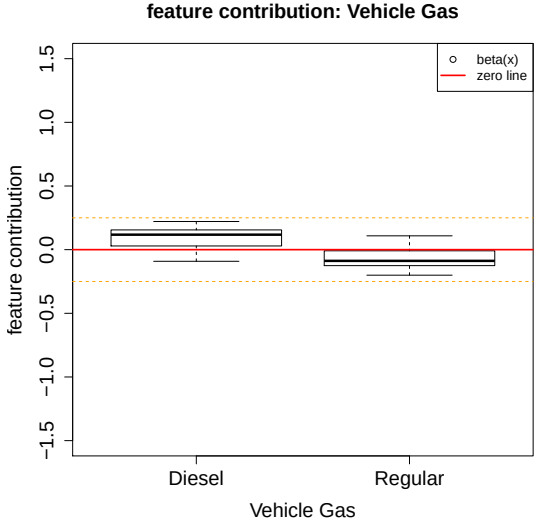
Covariate Contributions $\widehat{\beta}_j(\boldsymbol{x})x_j$



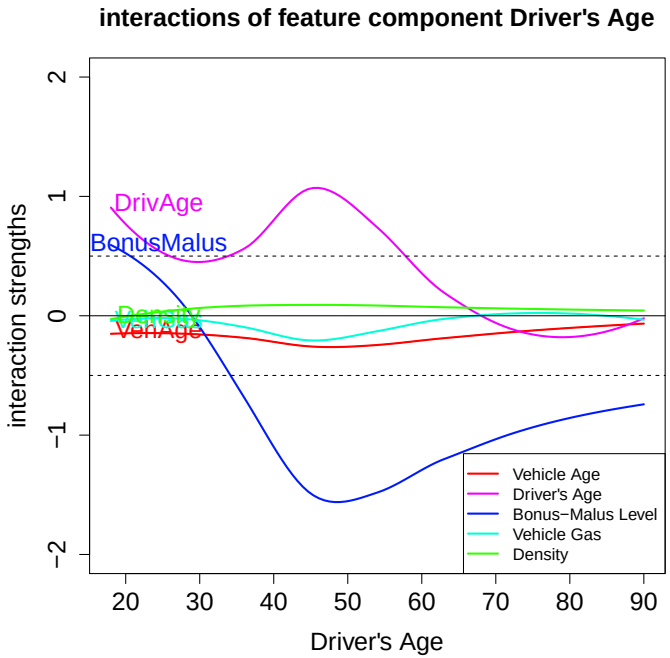


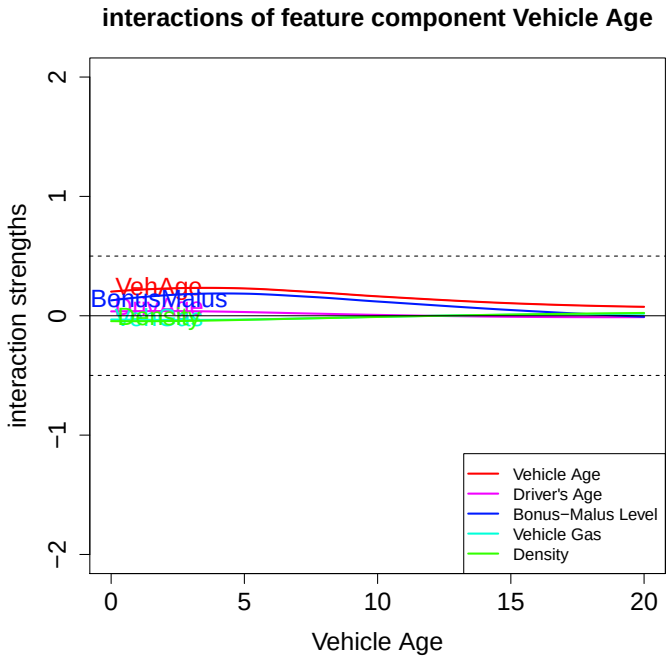






Interactions between Covariate Components





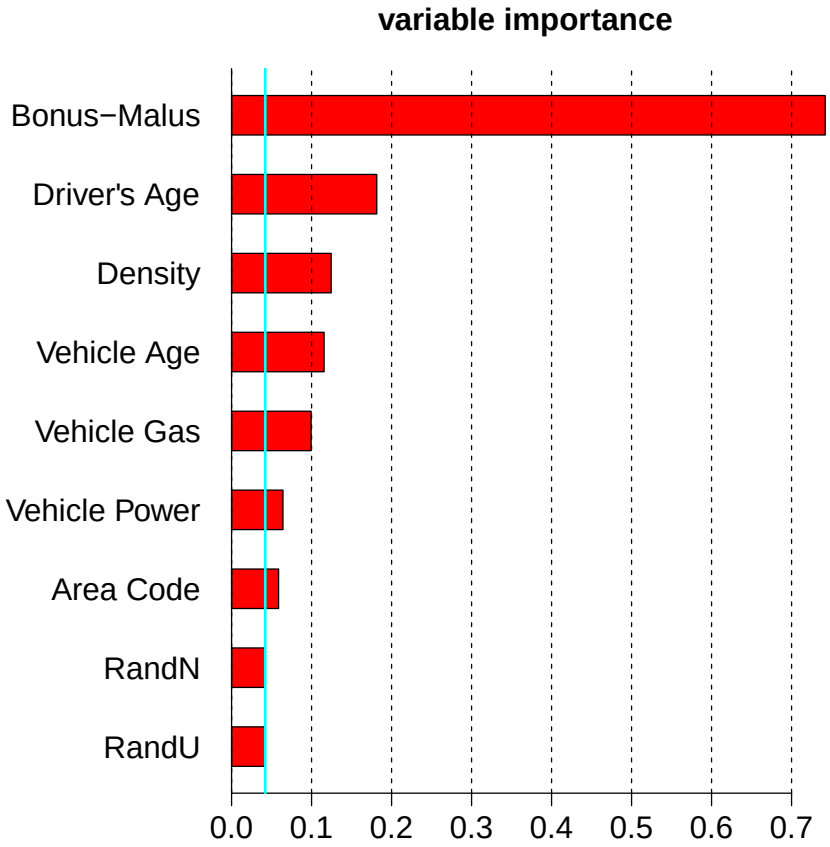
$$\nabla \beta_j(\boldsymbol{x}) = \left(\frac{\partial}{\partial x_1} \beta_j(\boldsymbol{x}), \ldots, \frac{\partial}{\partial x_q} \beta_j(\boldsymbol{x}) \right)^\top \in \mathbb{R}^q.$$

Calculation of Gradients in keras

```
j <- 1  # select the feature component
#
beta.j  <- Attention %>% layer_lambda(function(x) x[,j])
model.j <- keras_model(inputs = c(Design), outputs = c(beta.j))
#
grad <- beta.j %>% layer_lambda(function(x) k_gradients(model.j$outputs,
                                                        model.j$inputs))
model.grad <- keras_model(inputs = c(Design), outputs = c(grad))
#
grad.beta <- data.frame(model.grad %>% predict(as.matrix(XX)))
```

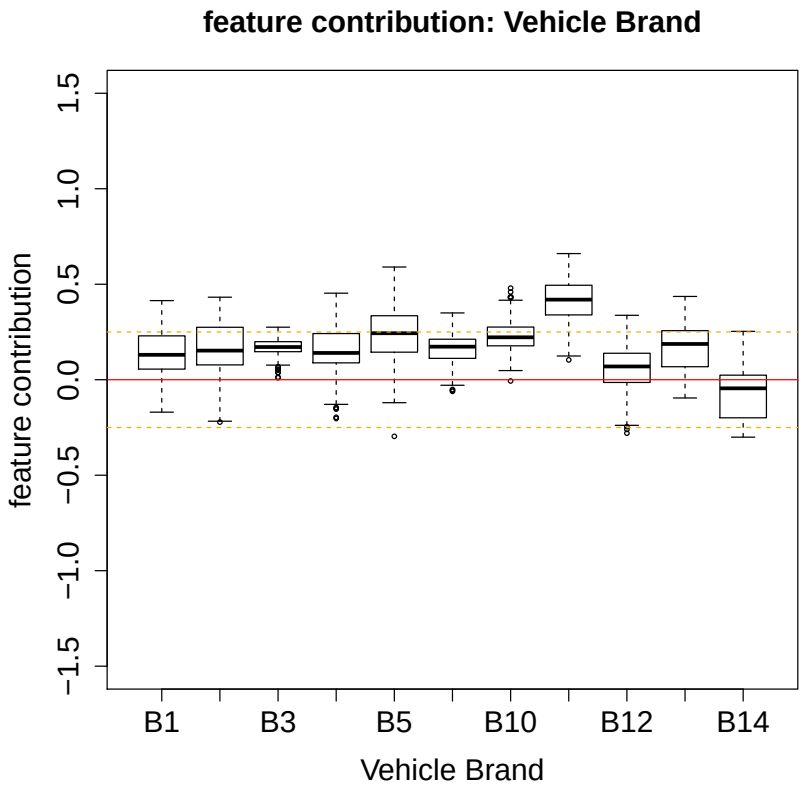
In different TensorFlow/keras versions this may be slightly different.

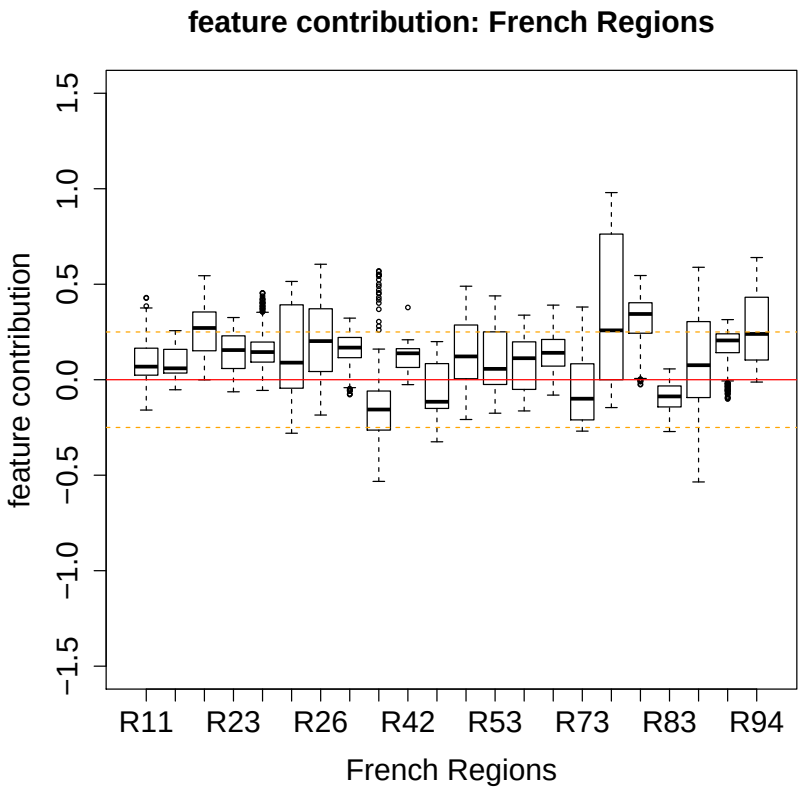
Variable/Term Importance



$$\mathrm{VI}_j = \frac{1}{n}\sum_{i=1}^{n}\left|\widehat{\beta}_j(\boldsymbol{x}_i)\right|.$$

Categorical Covariate Components





LASSO regularization within LocalGLMnets: see Richman–Wüthrich (2021b).

References

- Breiman (1996). Bagging predictors. Machine Learning 24, 123-40.
- Efron, Hastie (2016). Computer Age Statistical Inference: Algorithms, Evidence, and Data Science. Cambridge UP.
- Ferrario, Noll, Wüthrich (2018). Insights from inside neural networks. SSRN 3226852.
- Goodfellow, Bengio, Courville (2016). Deep Learning. MIT Press.
- Hastie, Tibshirani, Friedman (2009). The Elements of Statistical Learning. Springer.
- Lorentzen, Mayer (2020). Peeking into the black box: an actuarial case study for interpretable machine learning. SSRN 3595944.
- Noll, Salzmann, Wüthrich (2018). Case study: French motor third-party liability claims. SSRN 3164764.
- Richman (2020a/b). AI in actuarial science – a review of recent advances – part 1/2. Annals of Actuarial Science.
- Richman, Wüthrich (2020). Nagging predictors. Risks 8/3, 83.
- Richman, Wüthrich (2021a). LocalGLMnet: interpretable deep learning for tabular data. SSRN 3892015.
- Richman, Wüthrich (2021b). LASSO regularization within the LocalGLMnet architecture. SSRN 3927187.
- Schelldorfer, Wüthrich (2019). Nesting classical actuarial models into neural networks. SSRN 3320525.
- Schelldorfer, Wüthrich (2021). LocalGLMnet: a deep learning architecture for actuaries. SSRN 3900350.
- Wüthrich (2020). Bias regularization in neural network models for general insurance pricing. European Actuarial Journal 10/1, 179-202.
- Wüthrich, Buser (2016). Data Analytics for Non-Life Insurance Pricing. SSRN 2870308, Version September 10, 2020.
- Wüthrich, Merz (2019). Editorial: Yes, we CANN! ASTIN Bulletin 49/1, 1-3.
- Wüthrich, Merz (2021). Statistical Foundations of Actuarial Learning and its Applications. SSRN 3822407.

Convolutional Neural Networks

Mario V. Wüthrich
RiskLab, ETH Zurich

"Deep Learning with Actuarial Applications in R"
Swiss Association of Actuaries SAA/SAV, Zurich
October 14/15, 2021

Programme SAV Block Course

- Refresher: Generalized Linear Models (THU 9:00-10:30)
- Feed-Forward Neural Networks (THU 13:00-15:00)
- Discrimination-Free Insurance Pricing (THU 17:15-17:45)

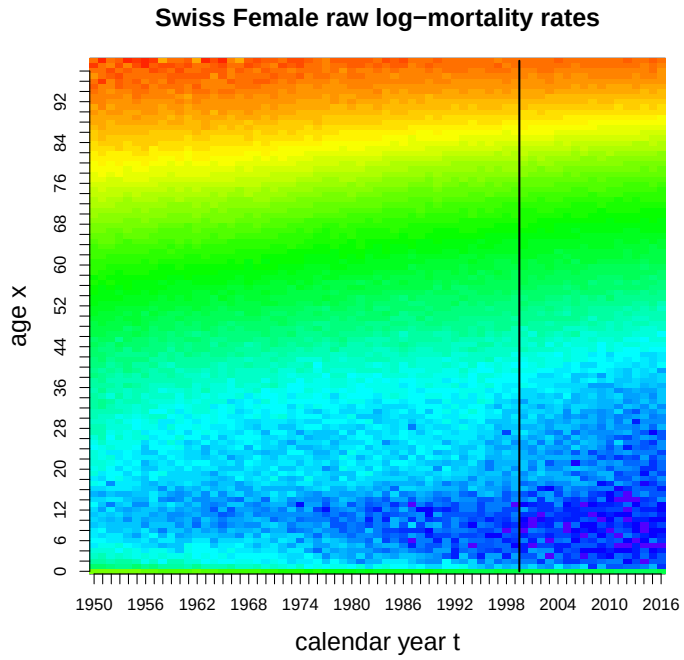
- LocalGLMnet (FRI 9:00-10:30)
- Convolutional Neural Networks (FRI 13:00-14:30)
- Wrap Up (FRI 16:00-16:30)

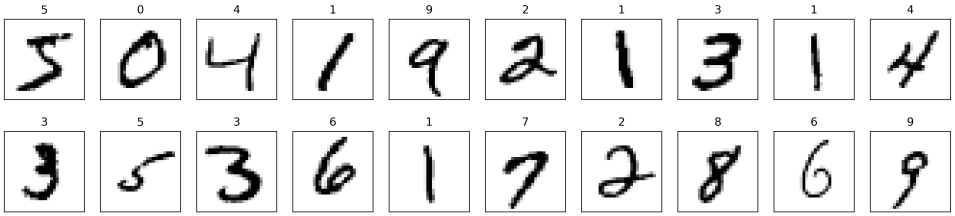
Contents: Convolutional Neural Networks

- Spatial and temporal data
- Convolutional neural networks (CNNs)
- Special purpose tools for CNNs
- CNN examples

- **Spatial and Temporal Data**

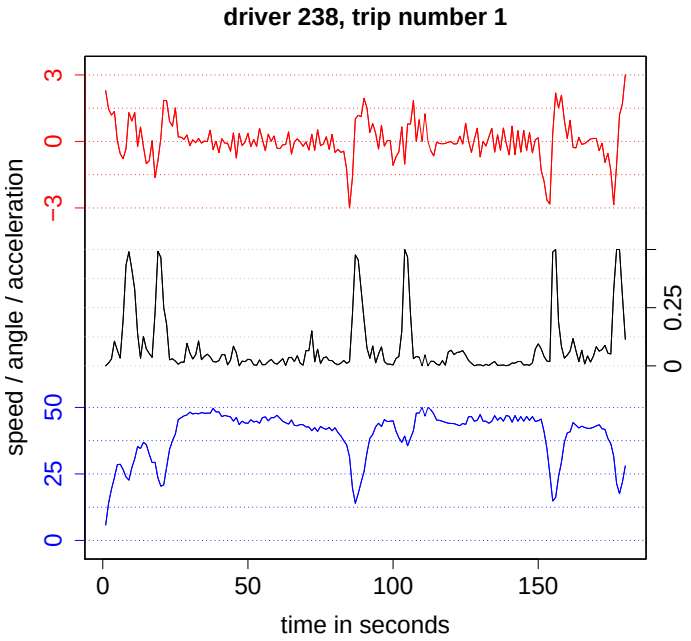
Spatial Objects





- Spatial objects are tensors $\boldsymbol{Z} \in \mathbb{R}^{I \times J \times q}$ of order/mode 3, i.e. 3-dimensional arrays.
- The first two components of $\boldsymbol{Z}$ give the location (i, j) in the picture.
- The 3rd component of $\boldsymbol{Z}$ gives the signals in location (i, j). This 3rd component is called channels. Black-white pictures have $q = 1$ channel (gray scale), color pictures have $q = 3$ channels (RGB channels for red-green-blue).

Temporal and Time Series Objects



‘storm damaged controller at courthouse’

- Temporal objects are matrices $\boldsymbol{Z} \in \mathbb{R}^{T \times q}$ which are tensors of order/mode 2.
- The 1st component of $\boldsymbol{Z}$ gives the location t in the time series.
- The 2nd component of $\boldsymbol{Z}$ gives the signals in location t having q channels.

Using FNNs for Time Series Processing

- Assume we have time series information $\boldsymbol{Z} = \boldsymbol{x}_{0:T} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T)^\top \in \mathbb{R}^{(T+1)\times q}$ to predict response Y_{T+1}

$$\boldsymbol{x}_{0:T} \;\mapsto\; \mu_{T+1}(\boldsymbol{x}_{0:T}) = \mathbb{E}[Y_{T+1}|\boldsymbol{x}_{0:T}] = \mathbb{E}[Y_{T+1}|\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T].$$

- In principle, we could choose a FNN architecture and set

$$\mu_{T+1}(\boldsymbol{x}_{0:T}) = g^{-1}\left(\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_{0:T})\right\rangle\right).$$

- This is not recommended:
  - ⋆ Whenever we collect a new observation $\boldsymbol{x}_{T+1}$ we need to extend the input dimension of the FNN and re-calibrate it.
  - ⋆ The FNN does not recognize any temporal (topological) structure.
  - ⋆ The latter also applies to spatial objects.

RNNs, CNNs and Attention Layers

- There are 3 different ways in network modeling to deal with topological data.

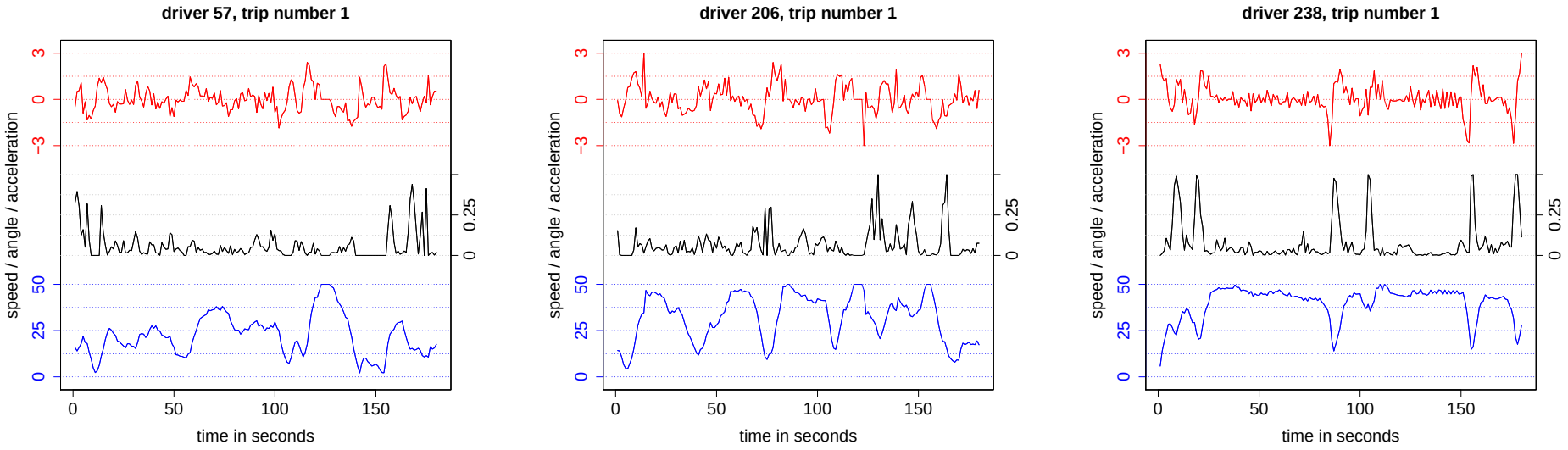
- Recurrent neural networks (RNNs) process information recursively to preserve time series structure. RNNs are most suitable to predict the next response Y_{T+1} based on past information $\boldsymbol{x}_{0:T}$.

- Convolutional neural networks (CNNs) extract local structure from topological objects preserving the topology. This is done by moving small windows, say, across the picture and trying to identify specific structure in this window.
  - ▷ This is similar to rolling windows in financial time series estimation.
  - ▷ CNNs act more locally, whereas RNNs and FNNs act more globally.

- Attention layers move across the time series and try to pay attention to special features in the time series, like giving more or less credibility to them. This is similar to the regression attentions $\boldsymbol{\beta}(\boldsymbol{x})$ in LocalGLMnets.

- **Convolutional Neural Networks (CNNs)**

Functioning of CNNs



- Choose a window (called filter), say, of size $b \times q = 10 \times 3$.
- b is called filter size, kernel size or band width; q is the number of channels.
- Move with this filter across the time series (in time direction t) and try to spot specific structure with this filter (in the rolling window).
- The way of finding structure is with a convolution operation $*$.

CNNs: More Formally

- Start from an input tensor $\boldsymbol{x} \in \mathbb{R}^{I \times J \times q_0}$ of order 3.
- Choose filter sizes $(b_1, b_2, q_0)^\top \in \mathbb{N}^3$ with $b_1 < I$ and $b_2 < J$.
- A CNN operation is a mapping

$$\begin{array}{rcl} \boldsymbol{z}_k : \mathbb{R}^{I \times J \times q_0} & \to & \mathbb{R}^{(I-b_1+1) \times (J-b_2+1)} \\ \boldsymbol{x} & \mapsto & \boldsymbol{z}_k(\boldsymbol{x}) = \left(z_{k;i,j}(\boldsymbol{x})\right)_{1 \le i \le I-b_1+1; 1 \le j \le J-b_2+1}, \end{array}$$

having, for activation function $\phi : \mathbb{R} \to \mathbb{R}$,

$$z_{k;i,j}(\boldsymbol{x}) = \phi\left(w_k + \sum_{l_1=1}^{b_1} \sum_{l_2=1}^{b_2} \sum_{l_3=1}^{q_0} w_{k;l_1,l_2,l_3} \; x_{i+l_1-1,j+l_2-1,l_3}\right),$$

for given intercept $w_k \in \mathbb{R}$ and filter weights $\boldsymbol{W}_k = \left(w_{k;l_1,l_2,l_3}\right)_{l_1,l_2,l_3}$.

CNNs: Convolution Operation

- Choose the corner $(i, j, 1)$ of the tensor as base point. CNN operation considers

$$(i, j, 1) + [0 : b_1 - 1] \times [0 : b_2 - 1] \times [0 : q_0 - 1],$$

with filter weights $\boldsymbol{W}_k$.

- In fact, we perform a sort of convolution which motivates compact notation

$$\begin{array}{rcl} \boldsymbol{z}_k : \mathbb{R}^{I \times J \times q_0} & \to & \mathbb{R}^{(I-b_1+1) \times (J-b_2+1)} \\ \boldsymbol{x} & \mapsto & \boldsymbol{z}_k(\boldsymbol{x}) = \phi\left(\boldsymbol{W}_k * \boldsymbol{x}\right). \end{array}$$

- This convolution operation $*$ reflects one filter with filter weights $\boldsymbol{W}_k$. We can now choose multiple filters (similar to neurons in FNNs):
  $\triangleright$ This explains the meaning of the lower index k (which plays the role of different neurons $1 \leq k \leq q_1$ in FNNs).

CNNs: Multiple Filters

- Choose $q_1 \in \mathbb{N}$ filters, each having filter weights $\boldsymbol{W}_k$, $1 \leq k \leq q_1$.
- A CNN layer is a mapping

$$\begin{aligned} \boldsymbol{z}^{\mathrm{CNN}} : \mathbb{R}^{I \times J \times q_0} &\rightarrow \mathbb{R}^{(I-b_1+1)\times(J-b_2+1)\times q_1} \\ \boldsymbol{x} &\mapsto \boldsymbol{z}^{\mathrm{CNN}}(\boldsymbol{x}) = (\boldsymbol{z}_1(\boldsymbol{x}), \ldots, \boldsymbol{z}_{q_1}(\boldsymbol{x})), \end{aligned}$$

  with filters $\boldsymbol{z}_k(\boldsymbol{x}) = \phi\left(\boldsymbol{W}_k * \boldsymbol{x}\right)$.

- Thus, the (spatial) tensor

$$\boldsymbol{x} \in \mathbb{R}^{I \times J \times q_0},$$

  with q_0 channels is mapped to a (spatial) tensor

$$\boldsymbol{z}^{\mathrm{CNN}}(\boldsymbol{x}) \in \mathbb{R}^{(I-b_1+1)\times(J-b_2+1)\times q_1},$$

  with q_1 filters.

Properties of CNNs

- The convolution operation $*$ respects the local structure.
- FNNs extract global structure, CNNs extract local structure.
- Formally, the global scalar product $\boldsymbol{z}_k(\boldsymbol{x}) = \phi\langle \boldsymbol{w}_k, \boldsymbol{x}\rangle$ of FNNs is replaced by a local convolution $\boldsymbol{z}_k(\boldsymbol{x}) = \phi\left(\boldsymbol{W}_k * \boldsymbol{x}\right)$ for CNNs.
- CNNs have translation invariance properties, see Wiatowski–Bölcskei (2018).
- CNNs generally use less parameters than FNNs and RNNs, because filter weights are re-used/re-located.

- **Special Purpose Tools for CNNs**

CNNs: Padding with Zeros

- A CNN layer reduces the spatial size of the output tensor

$$\begin{aligned} \boldsymbol{z}^{\mathrm{CNN}} : \mathbb{R}^{I\times J\times q_0} &\to \mathbb{R}^{(I-b_1+1)\times(J-b_2+1)\times q_1} \\ \boldsymbol{x} &\mapsto \boldsymbol{z}^{\mathrm{CNN}}(\boldsymbol{x}) = (\boldsymbol{z}_1(\boldsymbol{x}),\ldots,\boldsymbol{z}_{q_1}(\boldsymbol{x})). \end{aligned}$$

- If this is an undesired feature, padding with zeros can be applied at all edges to obtain

$$\begin{aligned} \boldsymbol{z}^{\mathrm{CNN}} : \mathbb{R}^{I\times J\times q_0} &\to \mathbb{R}^{I\times J\times q_1} \\ \boldsymbol{x} &\mapsto \boldsymbol{z}^{\mathrm{CNN}}(\boldsymbol{x}) = (\boldsymbol{z}_1(\boldsymbol{x}),\ldots,\boldsymbol{z}_{q_1}(\boldsymbol{x})). \end{aligned}$$

- Remark that padding does not add any additional parameters, but it is only used to reshape the output tensor.

CNNs: Stride

- Strides are used to skip part of the input tensor $\boldsymbol{x}$ to reduce the size of the output. This may be useful if the input tensor is a very high resolution image.

- Choose stride parameters s_1 and s_2. Consider modified convolution

$$\sum_{l_1}\sum_{l_2}\sum_{l_3} w_{k;l_1,l_2,l_3}\ x_{s_1(i-1)+l_1,s_2(j-1)+l_2,l_3}.$$

- This considers windows

$$(s_1(i-1), s_2(j-1), 1) + [1:b_1] \times [1:b_2] \times [0:q_0-1].$$

CNNs: Dilation

- Dilation is similar to stride, though, different in that it enlarges the filter sizes instead of skipping certain positions in the input tensor.

- Choose dilation parameters e_1 and e_2. Consider modified convolution

$$\sum_{l_1}\sum_{l_2}\sum_{l_3} w_{k;l_1,l_2,l_3}\ x_{i+e_1(l_1-1),j+e_2(l_2-1),l_3}.$$

- This considers

$$(i,j,1) + e_1\,[0:b_1-1] \times e_2\,[0:b_2-1] \times [0:q_0-1].$$

CNNs: Max-Pooling Layers

- Pooling layers help to reduce the sizes of the tensors.
- We choose fixed window sizes b_1 and b_2 and strides $s_1 = b_1$ and $s_2 = b_2$; this gives a partition (disjoint windows).
- The max-pooling layer then considers

$$\begin{array}{rcl} \boldsymbol{z}^{\mathrm{Max}} : \mathbb{R}^{I \times J \times q_0} & \to & \mathbb{R}^{I' \times J' \times q_0} \\ \boldsymbol{x} & \mapsto & \boldsymbol{z}^{\mathrm{Max}}(\boldsymbol{x}) = \mathrm{MaxPool}(\boldsymbol{x}), \end{array}$$

  with $I' = \lfloor I/b_1 \rfloor$ and $J' = \lfloor J/b_2 \rfloor$ (cropping last columns by default), and where the convolution operation $*$ is replaced by a $\max$ operation (modulo channels).
- This extracts the maximums from the (spatially disjoint) windows

$$[b_1(i-1)+1 : b_1 i] \times [b_2(j-1)+1 : b_2 j] \times [k],$$

  for each channel $1 \leq k \leq q_0$, individually.

CNNs: Flatten Layers

- A flatten layer is used to reshape a tensor to a vector.
- Consider mapping

$$\begin{aligned} \boldsymbol{z}^{\text{flatten}} : \mathbb{R}^{I \times J \times q_0} &\to \mathbb{R}^{q_1} \\ \boldsymbol{x} &\mapsto \boldsymbol{z}^{\text{flatten}}(\boldsymbol{x}) = (x_{1,1,1}, \ldots, x_{I,J,q_0})^\top, \end{aligned}$$

  with $q_1 = I \cdot J \cdot q_0$.

- The flattened object $\boldsymbol{z}^{\text{flatten}}(\boldsymbol{x})$ can serve as input to a FNN.
- We have already met this operator with embedding layers for categorical features.

CNNs: Example (1/2)

```
 1 library(keras)
 2 #
 3 shape <- c(180,50,3)
 4 #
 5 model <- keras_model_sequential()
 6 model %>%
 7   layer_conv_2d(filters = 10, kernel_size = c(11,6), activation='tanh',
 8                                                      input_shape = shape) %>%
 9   layer_max_pooling_2d(pool_size = c(10,5)) %>%
10   layer_conv_2d(filters = 5, kernel_size = c(6,4), activation='tanh') %>%
11   layer_max_pooling_2d(pool_size = c(3,2)) %>%
12   layer_flatten()
```

$$180\times50\times3 \overset{\text{CNN1}}{\mapsto} 170\times45\times10 \overset{\text{Max1}}{\mapsto} 17\times9\times10 \overset{\text{CNN2}}{\mapsto} 12\times6\times5 \overset{\text{Max2}}{\mapsto} 4\times3\times5 \overset{\text{flatten}}{\mapsto} 60.$$

CNNs: Example (2/2)

```
1  Layer (type)                 Output Shape              Param #
2  =================================================================
3  conv2d_1 (Conv2D)            (None, 170, 45, 10)       1990
4  _________________________________________________________________
5  max_pooling2d_1 (MaxPooling2D) (None, 17, 9, 10)       0
6  _________________________________________________________________
7  conv2d_2 (Conv2D)            (None, 12, 6, 5)          1205
8  _________________________________________________________________
9  max_pooling2d_2 (MaxPooling2D) (None, 4, 3, 5)         0
10 _________________________________________________________________
11 flatten_1 (Flatten)          (None, 60)                0
12 =================================================================
13 Total params: 3,195
14 Trainable params: 3,195
15 Non-trainable params: 0
```

$$180\times50\times3 \overset{\text{CNN1}}{\mapsto} 170\times45\times10 \overset{\text{Max1}}{\mapsto} 17\times9\times10 \overset{\text{CNN2}}{\mapsto} 12\times6\times5 \overset{\text{Max2}}{\mapsto} 4\times3\times5 \overset{\text{flatten}}{\mapsto} 60.$$

- **Time Series Example: Telematics Data**

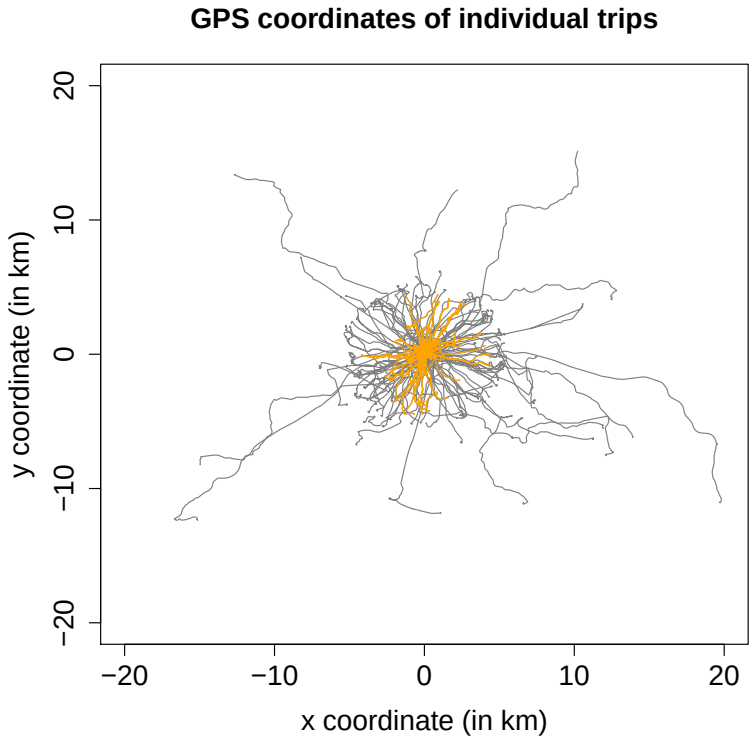
What is Telematics Car Driving Data?

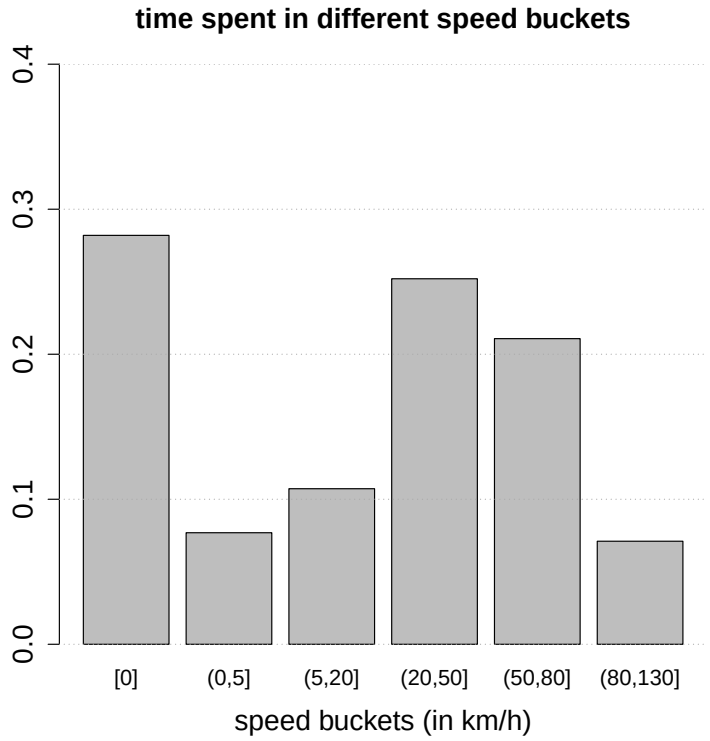
- GPS location data second by second, speed, acceleration, braking, intensity of left and right turns, engine revolutions,
- vehicle sensors and cameras,
- time stamp (day time, rush hour, night, etc.), total distances at different times,
- road type, traffic conditions, weather conditions, etc.,
- traffic rules (e.g. speeding), driving and health conditions, etc.

Volume of telematics car driving data, back of envelope calculation:

- 100KB of telematics data per driver and per day.
- This amounts to 40MB of data per driver per year.
- A small portfolio of 100'000 drivers results every year in 4TB data.

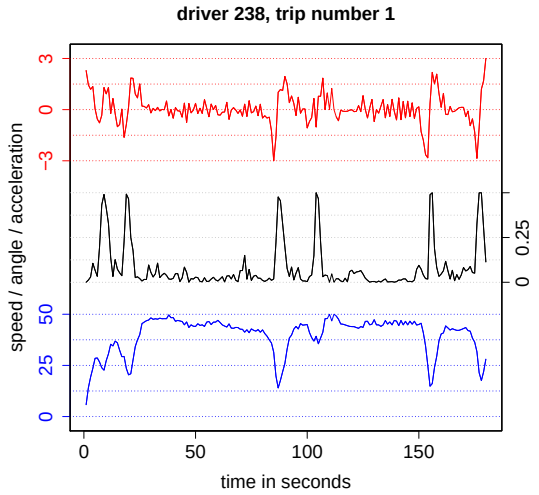
Illustration of GPS Location Data of Selected Driver





Remark that the idling phase is comparably large,
typically, we truncate the idling phase in our analysis.

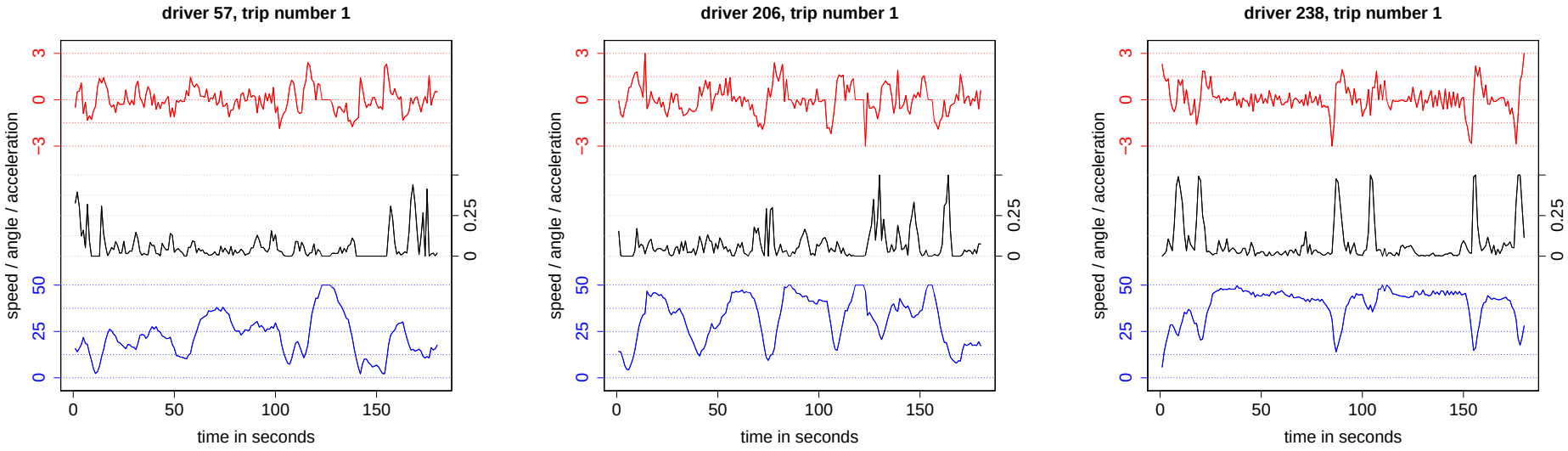
Speed, Acceleration/Braking and Direction



acceleration (m/s^2) / change in direction ($|\sin|$/s)/ speed (km/h)

- We have 3 channels:
  - ⋆ Speed v is concatenated so that $v \in [2, 50]$km/h.
  - ⋆ Acceleration a is censored at ± 3m/s^2 because of scarcity of data and data error. Extreme acceleration $+6$m/s^2, extreme deceleration -8m/s^2.
  - ⋆ Change of direction Δ is censored at $1/2$.

Choose 3 Selected Drivers



- Consider 3 selected drivers, called drivers 57, 206 and 238.
- **Question:** Can we correctly allocate individual trips to the right drivers?
- Assume that of each trip we have 180 seconds of driving experience (at random chosen from the entire trip and pre-processed as described above).

Classification with CNNs

- We choose a CNN because we would like to find similar structure in telematics time series data to discriminate the 3 different drivers.

- Consider (speed-acceleration-change in angle) time series, for $t = 1, \ldots, T = 180$,

$$(v_{s,t}, a_{s,t}, \Delta_{s,t})^\top \in [2, 50]\text{km/h} \times [-3, 3]\text{m/s}^2 \times [0, 1/2],$$

  where $s = 1, \ldots, S$ labels the individual trips of the considered drivers.

- Define 3-dimensional time series feature (covariate with 3 channels)

$$\boldsymbol{x}_s = \Big((v_{s,1}, a_{s,1}, \Delta_{s,1})^\top, \ldots, (v_{s,180}, a_{s,180}, \Delta_{s,180})^\top\Big)^\top \in \mathbb{R}^{180\times 3},$$

  with categorical response $Y_s \in \{57, 206, 238\}$ indicating the drivers.

Logistic Regression for Classification

- Multinomial logistic regression uses linear predictors on the canonical scale

$$\boldsymbol{x} \;\mapsto\; \mathbf{p}^{\mathrm{Logistic}}(\boldsymbol{x}) \;=\; \mathrm{softmax}\Big\langle \mathbf{B}, \boldsymbol{z}^{\mathrm{flatten}}(\boldsymbol{x}) \Big\rangle \;\in\; (0,1)^3,$$

with regression parameters $\mathbf{B} \in \mathbb{R}^{180\cdot 3 \times 3}$:

- ⋆ we need to pre-process time series feature $\boldsymbol{x} \in \mathbb{R}^{180\times 3}$ for suitable shape (flatten to vector) and for suitable functional form (not done here);
- ⋆ the scalar product is understood column-wise in $\mathbf{B}$;
- ⋆ the softmax function is (here) for $j = 1, 2, 3$ given by

$$\mathrm{softmax}\Big\langle \mathbf{B}, \boldsymbol{z} \Big\rangle_j \;=\; \frac{\exp\langle \mathbf{b}_j, \boldsymbol{z}\rangle}{\sum_{k=1}^{3} \exp\langle \mathbf{b}_k, \boldsymbol{z}\rangle} \;\in\; (0,1).$$

where $\mathbf{b}_j$ is the j-th column of $\mathbf{B}$.

CNNs for Logistic Regression

- Multinomial logistic regression uses linear predictors on the canonical scale

$$\boldsymbol{x} \;\mapsto\; \mathbf{p}^{\text{Logistic}}(\boldsymbol{x}) \;=\; \text{softmax}\Big\langle \mathbf{B}, \boldsymbol{z}^{\text{flatten}}(\boldsymbol{x}) \Big\rangle \;\in\; (0,1)^3,$$

with regression parameters $\mathbf{B} \in \mathbb{R}^{180 \cdot 3 \times 3}$.

- We choose a CNN architecture of depth $d \in \mathbb{N}$ for multinomial logistic regression

$$\boldsymbol{x} \;\mapsto\; \mathbf{p}^{\text{CNN}}(\boldsymbol{x}) \;=\; \text{softmax}\left\langle \mathbf{B}, \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)} \right)(\boldsymbol{x}) \right\rangle \;\in\; (0,1)^3,$$

with layers $\boldsymbol{z}^{(m)}$ being described on the next slide.

R Code for CNN Architecture on Time Series Data

```
model <- keras_model_sequential()

#
model %>%

layer_conv_1d(filters=12,kernel_size=5,activation='tanh',input_shape=c(180,3)) %>%
     layer_max_pooling_1d(pool_size = 3) %>%

layer_conv_1d(filters = 10, kernel_size = 5, activation='tanh') %>%
     layer_max_pooling_1d(pool_size = 3) %>%

layer_conv_1d(filters = 8, kernel_size = 5, activation='tanh') %>%
     layer_global_max_pooling_1d() %>%
     layer_dropout(rate = 0.3) %>%

layer_dense(units = 3, activation = 'softmax')
```

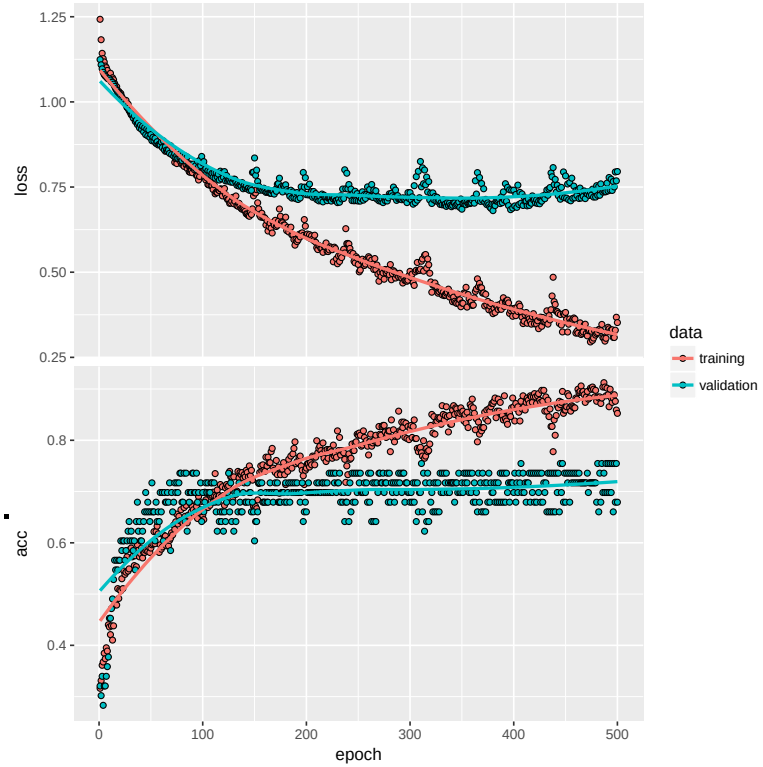
$$180 \times 3 \overset{\text{CNN1}}{\mapsto} 176 \times 12 \overset{\text{Max1}}{\mapsto} 58 \times 12 \overset{\text{CNN2}}{\mapsto} 54 \times 10 \overset{\text{Max2}}{\mapsto} 18 \times 10 \overset{\text{CNN3}}{\mapsto} 14 \times 8 \overset{\text{GlobMax}}{\mapsto} 8 \overset{\text{FNN}}{\mapsto} 3.$$

Explicit CNN Architecture and Network Parameters

```
1
2  Layer (type)                 Output Shape              Param #
3  =================================================================
4  conv1d_1 (Conv1D)            (None, 176, 12)           192
5  _________________________________________________________________
6  max_pooling1d_1 (MaxPoolin (None, 58, 12)            0
7  _________________________________________________________________
8  conv1d_2 (Conv1D)            (None, 54, 10)            610
9  _________________________________________________________________
10 max_pooling1d_2 (MaxPoolin (None, 18, 10)            0
11 _________________________________________________________________
12 conv1d_3 (Conv1D)            (None, 14, 8)             408
13 _________________________________________________________________
14 global_max_pooling1d_1 (Gl (None, 8)                 0
15 _________________________________________________________________
16 dropout_1 (Dropout)          (None, 8)                 0
17 _________________________________________________________________
18 dense_1 (Dense)              (None, 3)                 27
19 =================================================================
20 Total params: 1,237
21 Trainable params: 1,237
22 Non-trainable params: 0
```

Gradient Descent Fitting

- Total data: 521+131 individual trips $\mathcal{L}$ and $\mathcal{T}$
- We use 521 trips $\mathcal{L}$ to learn the network.
- We out-of-sample predict on 131 trips $\mathcal{T}$.
- Split 521 trips 8:2 for train/validation $\mathcal{U}$ and $\mathcal{V}$.
- Callback retrieves best network.
- Gradient descent fitting takes 40 sec.
- The plot shows deviance losses and accuracy/misclassification rate.



Out-of-sample Results

- Out-of-sample confusion matrix on $\mathcal{T}$ (131 trips):

	true labels		
	driver 57	driver 206	driver 238
predicted label 57	33	4	0
predicted label 206	8	38	6
predicted label 238	1	5	36
% correct	78.6%	80.9%	85.7%

- This excellent prediction is based on "minimal information":
  - ⋆ only 180 seconds per trip;
  - ⋆ only very few trips to fit the network (521);
  - ⋆ not optimal data quality;
  - ⋆ not much network fine-tuning has been done.

Other Triples of Drivers

	true labels		
	driver 300	driver 301	driver 302
predicted label 300	61	1	3
predicted label 301	5	42	11
predicted label 302	8	11	25
% correct	82.4%	77.8%	65.8%

	true labels		
	driver 100	driver 150	driver 200
predicted label 100	43	12	2
predicted label 150	5	64	5
predicted label 200	4	2	51
% correct	82.7%	82.1%	87.9%

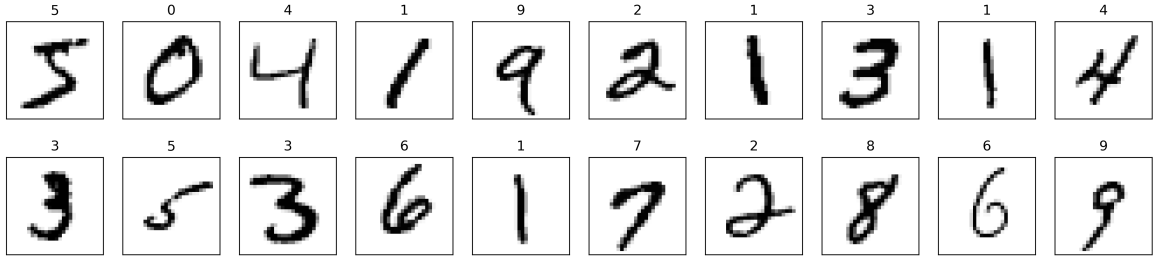
What's Next?

- Do you have any privacy concerns?

- Can we use this data to identify different driving styles?
- How much telematics data is needed to characterize a given driver?
- Can this data be made useful to improve driving behavior and style?

- **Spatial Example: Digits Recognition**

Modified National Institute of Standards and Technology (MINST) Data Set



- We have black-white pictures, i.e., 1 channel.
- Spatial objects are represented by tensors $\boldsymbol{x} \in [0,1]^{28\times 28\times 1}$ of order/mode 3.
- We have a classification problem with categorical response $Y \in \{0,\ldots,9\}$.
- The data basis contains $n = 70'000$ images.

R Code for CNN Architecture on Spatial Data

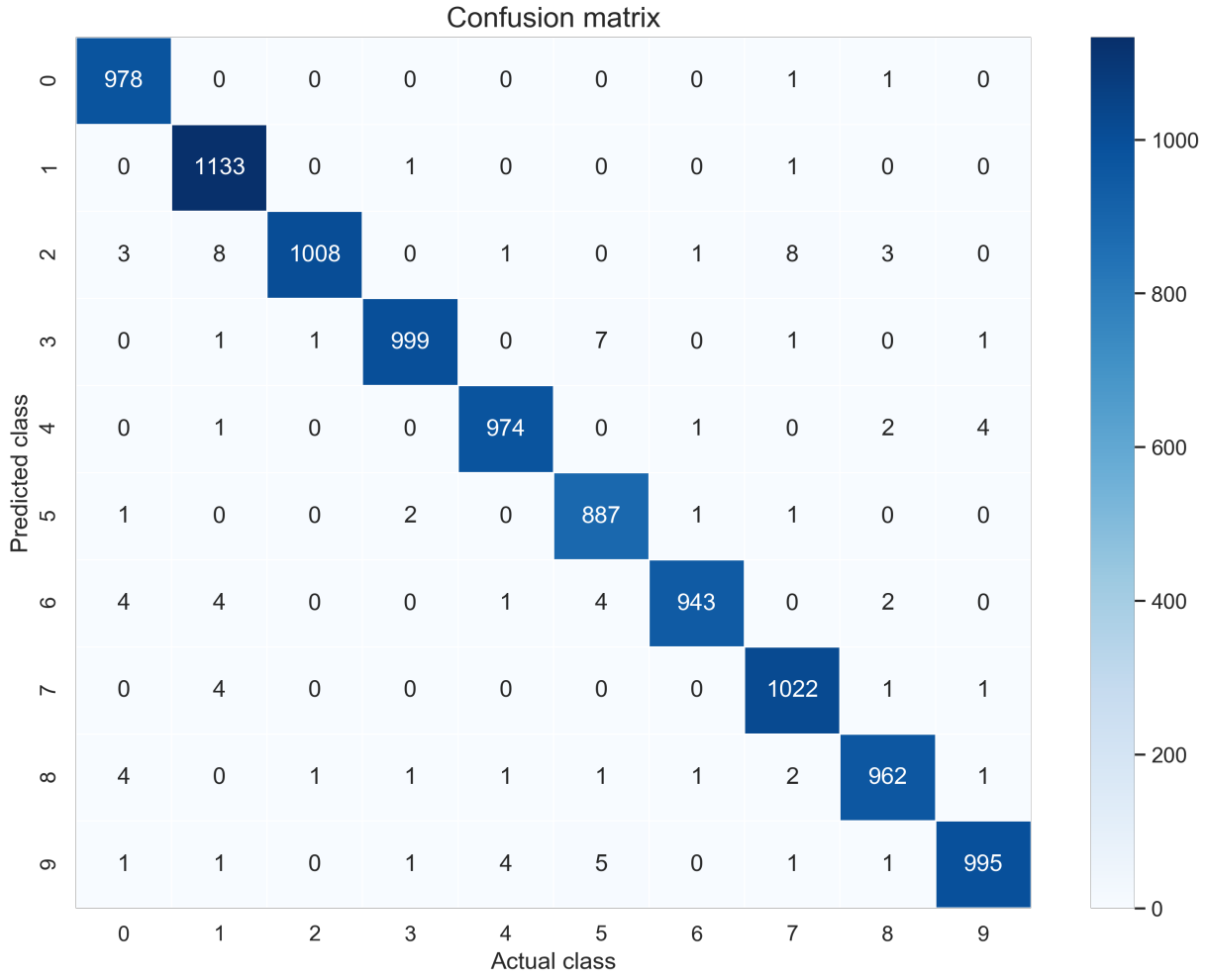
```
model <- keras_model_sequential()

model %>%

layer_conv_2d(filters = 10, kernel_size = c(3,3), padding="valid",
                            activation = "linear", input_shape = c(28,28,1)) %>%
layer_batch_normalization() %>%
layer_activation(activation="relu") %>%
layer_max_pooling_2d(pool_size=c(2,2), strides=c(2,2), padding="valid") %>%

layer_conv_2d(filters = 20, kernel_size = c(3,3), padding="valid") %>%
layer_batch_normalization() %>%
layer_activation(activation="relu") %>%
layer_max_pooling_2d(pool_size=c(2,2), strides=c(1,1), padding="valid") %>%

layer_conv_2d(filters = 40, kernel_size = c(3,3), padding="valid") %>%
layer_batch_normalization() %>%
layer_activation(activation="relu") %>%
layer_max_pooling_2d(pool_size=c(2,2), strides=c(2,2), padding="valid") %>%

layer_flatten() %>%
layer_dense(units=10, activation = 'softmax')
```

Explicit CNN Architecture and Network Parameters

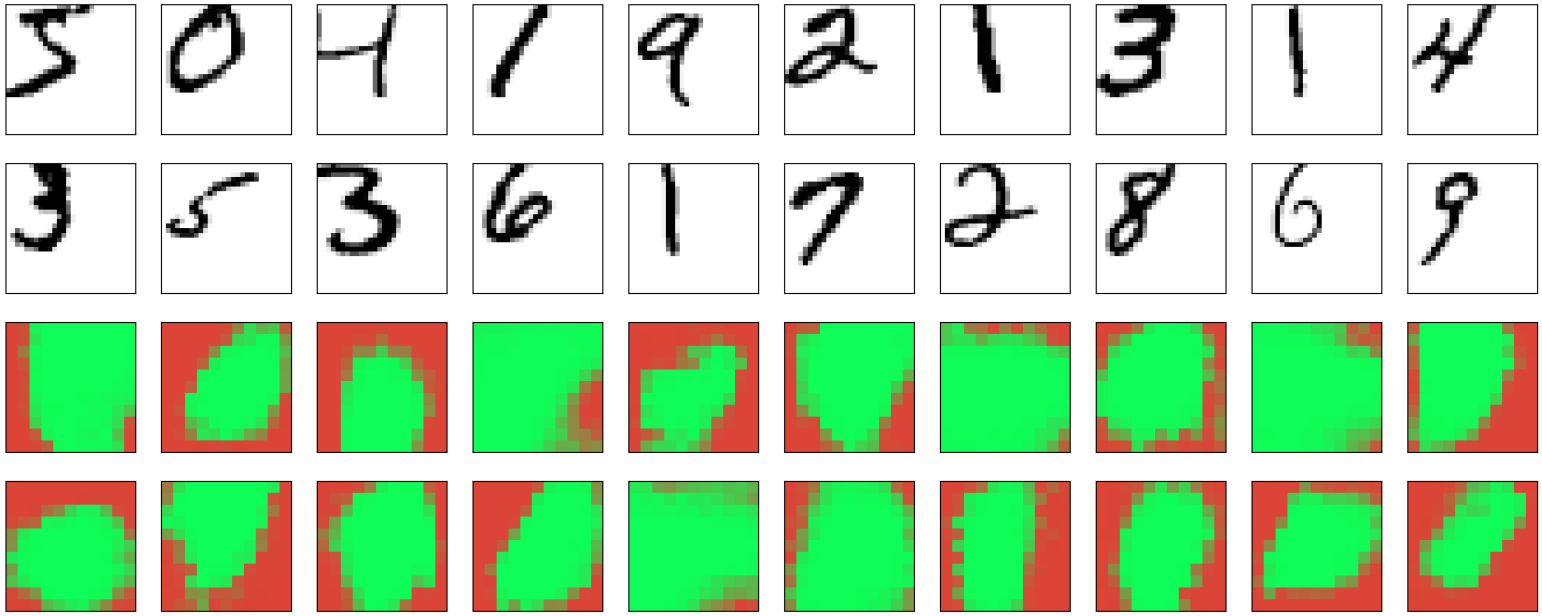
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 26, 26, 10)        100
_________________________________________________________________
batch_normalization_7 (BatchNormalizat (None, 26, 26, 10)        40
_________________________________________________________________
activation_7 (Activation)    (None, 26, 26, 10)        0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2D)  (None, 13, 13, 10)     0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 11, 11, 20)        1820
_________________________________________________________________
batch_normalization_8 (BatchNormalizat (None, 11, 11, 20)        80
_________________________________________________________________
activation_8 (Activation)    (None, 11, 11, 20)        0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2D)  (None, 10, 10, 20)     0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 8, 8, 40)          7240
_________________________________________________________________
batch_normalization_9 (BatchNormalizat (None, 8, 8, 40)          160
_________________________________________________________________
activation_9 (Activation)    (None, 8, 8, 40)          0
_________________________________________________________________
max_pooling2d_9 (MaxPooling2D)  (None, 4, 4, 40)       0
```

```
_________________________________________________________________
flatten_3 (Flatten)          (None, 640)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                6410
=================================================================
Total params: 15,850
Trainable params: 15,710
Non-trainable params: 140 (1/2 of batch normalizations)
```

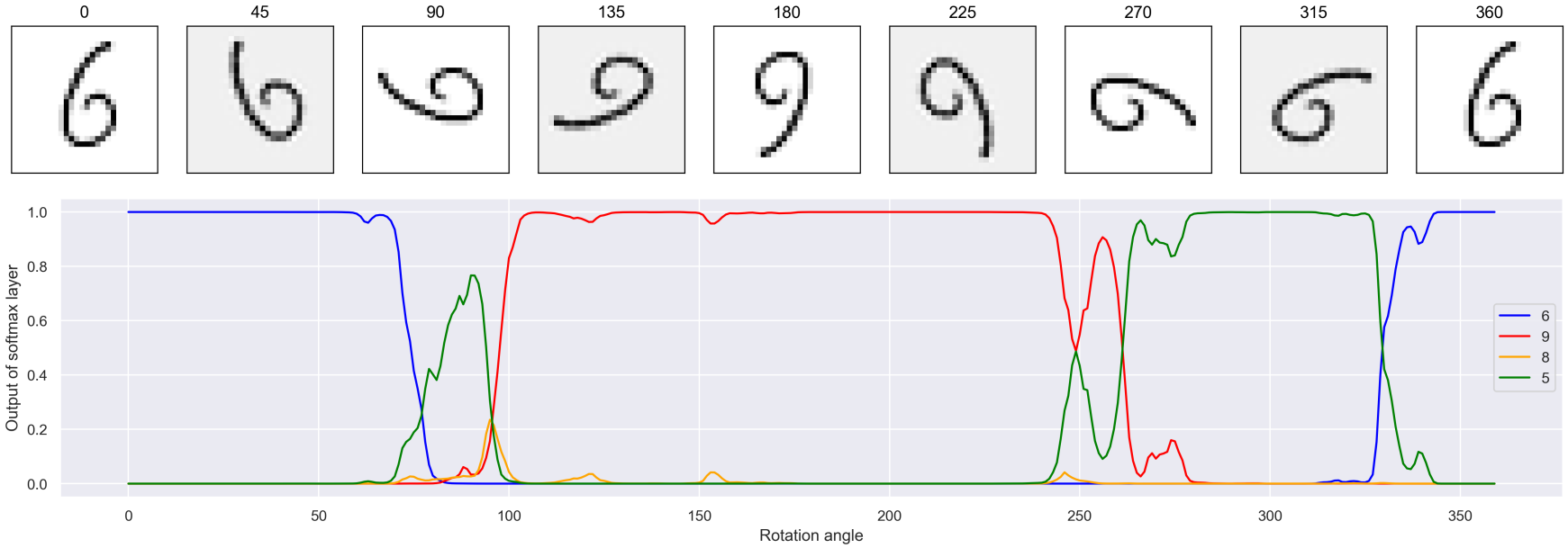
Result: Confusion Matrix



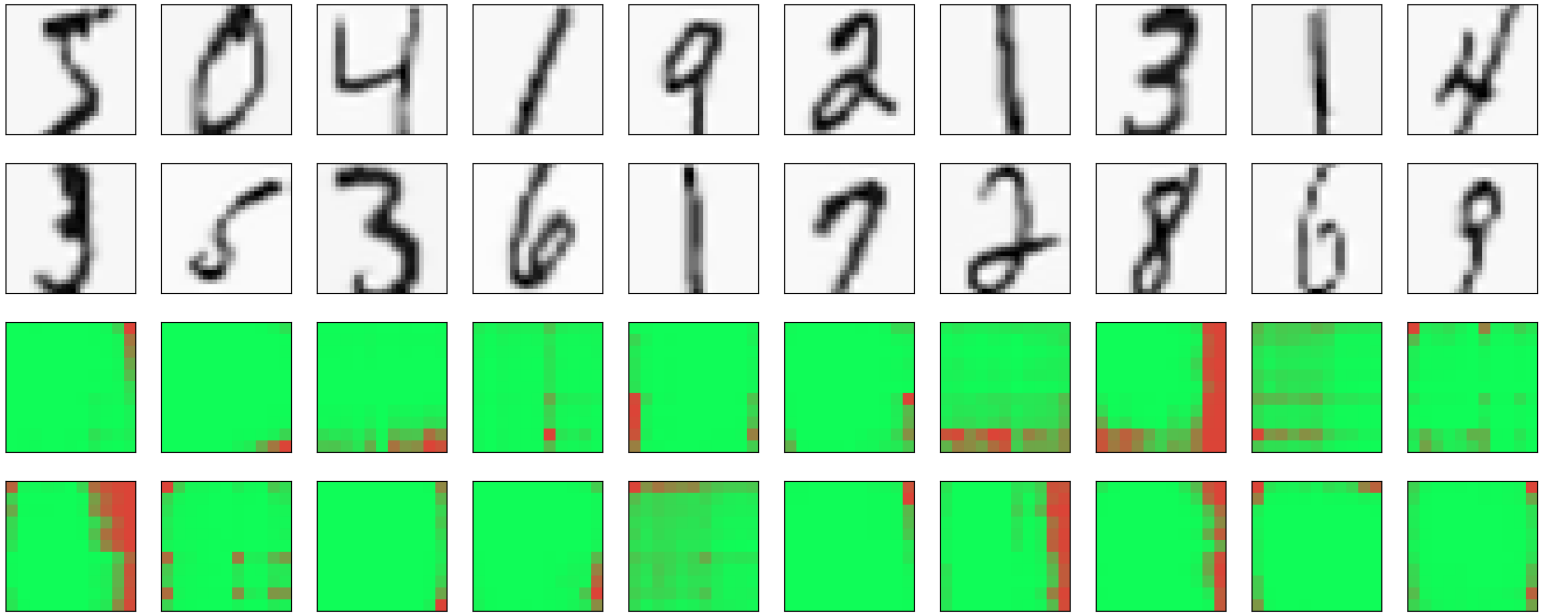
Shift Invariance



Rotation Invariance



Scale Invariance



References

- Efron, Hastie (2016). Computer Age Statistical Inference: Algorithms, Evidence, and Data Science. Cambridge UP.
- Gao, Wüthrich (2019). Convolutional neural network classification of telematics car driving data. Risks 7/1, article 6.
- Goodfellow, Bengio, Courville (2016). Deep Learning. MIT Press.
- Hastie, Tibshirani, Friedman (2009). The Elements of Statistical Learning. Springer.
- Meier, Wüthrich (2020). Convolutional neural network case studies: (1) anomalies in mortality rates (2) image recognition. SSRN 3656210.
- Perla, Richman, Scognamiglio, Wüthrich (2021). Time-series forecasting of mortality rates using deep learning. Scandinavian Actuarial Journal 2021/7, 572-598.
- Wiatowski, Bölcskei (2018). A mathematical theory of deep convolutional neural networks for feature extraction. IEEE Transactions on Information Theory 64/3, 1845-1866.
- Wüthrich, Merz (2021). Statistical Foundations of Actuarial Learning and its Applications. SSRN 3822407.

Recurrent Neural Networks

Mario V. Wüthrich
RiskLab, ETH Zurich

Block Course "Deep Learning with Actuarial Applications in R"
Swiss Association of Actuaries, Zurich
October 15/16, 2020

Programme SAV Block Course

- Refresher: Generalized Linear Models (THU 9:00-10:00)
- Feed-Forward Neural Networks (THU 12:30-14:00)
- Combined Actuarial Neural Network Models (THU 16:30-17:45)

- Recurrent Neural Networks (FRI 10:30-12:00)
- Discrimination-Free Insurance Pricing (FRI 14:30-15:00)
- Unsupervised Learning Methods (FRI 15:30-16:30)

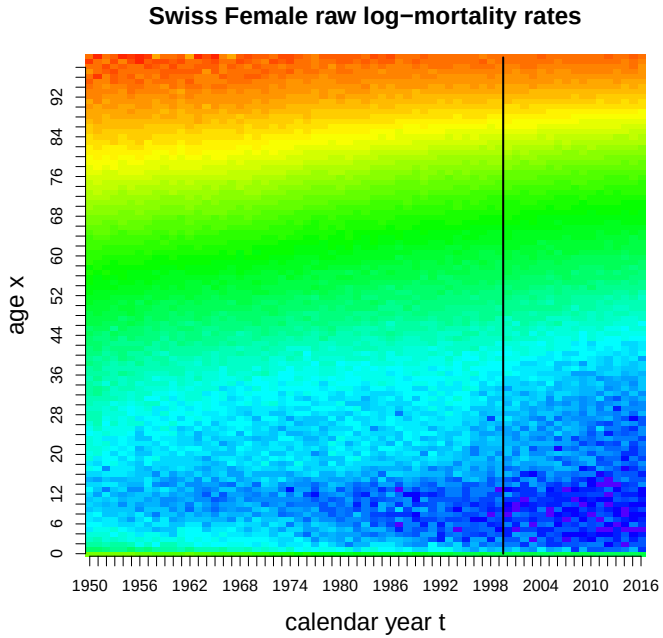
Contents: Recurrent Neural Networks

- Lee–Carter (LC) model
- Recurrent neural networks (RNNs)
- Long short-term memory (LSTM) networks
- Gated recurrent unit (GRU) networks
- Recurrent neural networks (RNNs) vs. convolutional neural networks (CNNs)

- **Lee–Carter (LC) Model and Time-Series**

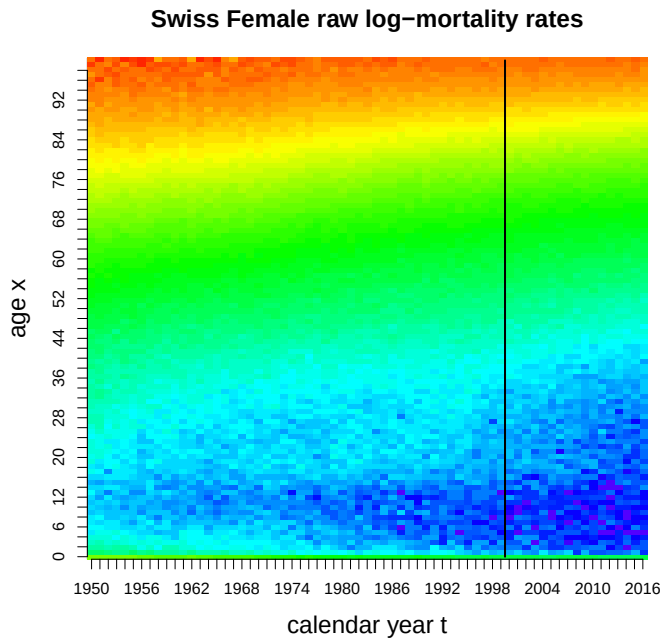
Human Mortality Database (HMD)

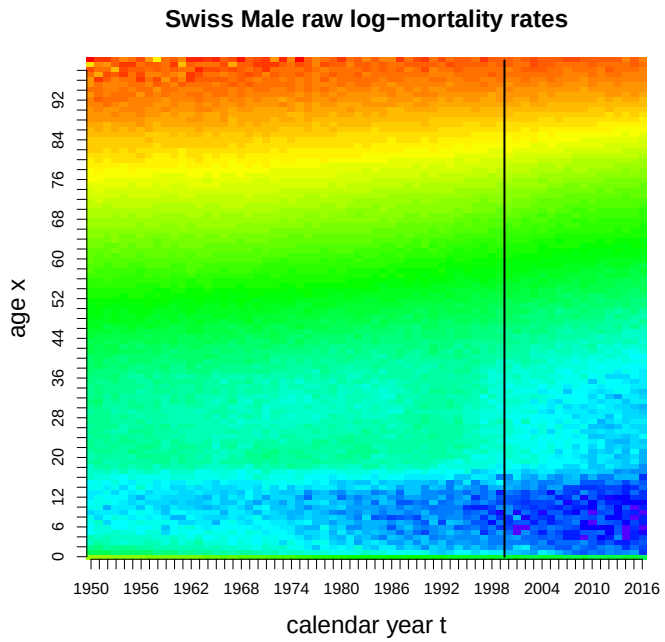
```
1 Classes 'data.table' and 'data.frame':  13400 obs. of  7 variables:
2  $ Gender      : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 1 1 1 1 ...
3  $ Year        : int  1950 1950 1950 1950 1950 1950 1950 1950 1950 1950 ...
4  $ Age         : int  0 1 2 3 4 5 6 7 8 9 ...
5  $ Country     : chr  "CHE" "CHE" "CHE" "CHE" ...
6  $ imputed_flag: chr  "FALSE" "FALSE" "FALSE" "FALSE" ...
7  $ mx          : num  0.02729 0.00305 0.00167 0.00123 0.00101 ...
8  $ logmx       : num  -3.6 -5.79 -6.39 -6.7 -6.9 ...
```



Human Mortality Database (HMD)

- **Aim:** Forecast mortality rates $m_{x,t}^{(i)}$ for ages x, calendar years t and populations i.
- Data available for ages $0 \leq x \leq 99$ and calendar years $1950 \leq t \leq 2016$ of 38 countries and 2 genders, i.e., $i = (r, g) \in \mathcal{I} = \mathcal{R} \times \{\texttt{female}, \texttt{male}\}$.
- Learning data $\mathcal{D} = \{1950 \leq t \leq 1999\}$; test data $\mathcal{T} = \{2000 \leq t \leq 2016\}$.





Lee–Carter (LC) Model (1992)

- Expected log-mortality rate is modeled by a regression function

$$(x,t,i) \;\mapsto\; \log(m_{x,t}^{(i)}) = a_x^{(i)} + b_x^{(i)} k_t^{(i)},$$

  - ⋆ $a_x^{(i)}$ average force of mortality at age x in population i;
  - ⋆ $k_t^{(i)}$ mortality trend in calendar year t of population i;
  - ⋆ $b_x^{(i)}$ mortality trend broken down to ages x of population i.

  - ▷ The inputs (x,i) and (t,i) are treated as categorical variables.
  - ▷ We have log-link, but not a GLM.

- 2-stage estimation and forecasting procedure, for each population i individually:

  1. Estimate $a_x^{(i)}$, $k_t^{(i)}$ and $b_x^{(i)}$ with singular value decomposition (SVD).
  2. Forecast by extrapolating estimated time series $(\widehat{k}_t^{(i)})_{t_0 \le t \le t_1}$ to years $t > t_1$.

Lee–Carter 2-Stage Forecasting

- Center the observed log-mortality rates $\log(M_{x,t}^{(i)})$

$$L_{x,t}^{(i)} = \log(M_{x,t}^{(i)}) - \frac{1}{|\mathcal{D}|} \sum_{s \in \mathcal{D}} \log(M_{x,s}^{(i)}).$$

- Find optimal parameter values with SVD (see also PCA chapter)

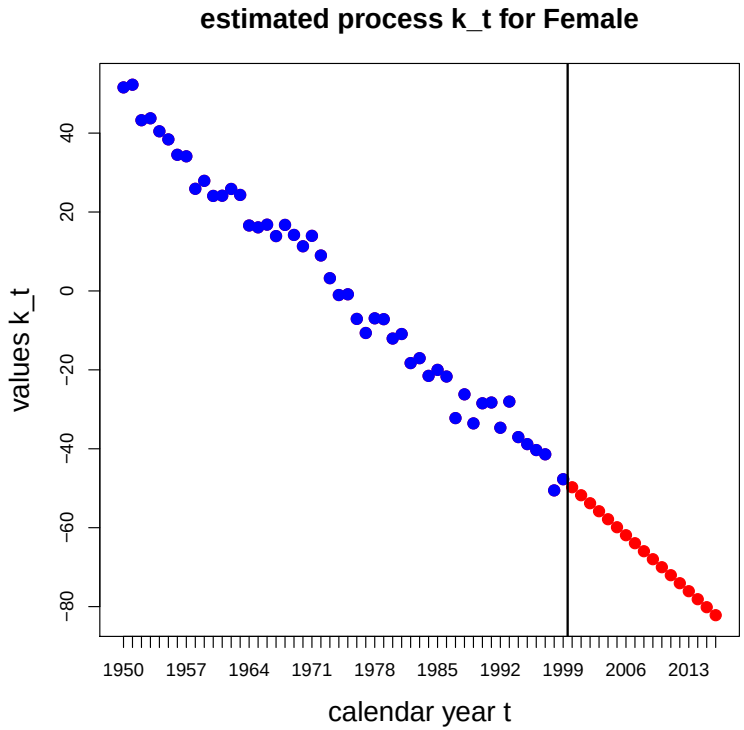
$$\underset{(b_x^{(i)})_x, (k_t^{(i)})_t}{\arg\min} \sum_{t,x} \left(L_{x,t}^{(i)} - b_x^{(i)} k_t^{(i)} \right)^2,$$

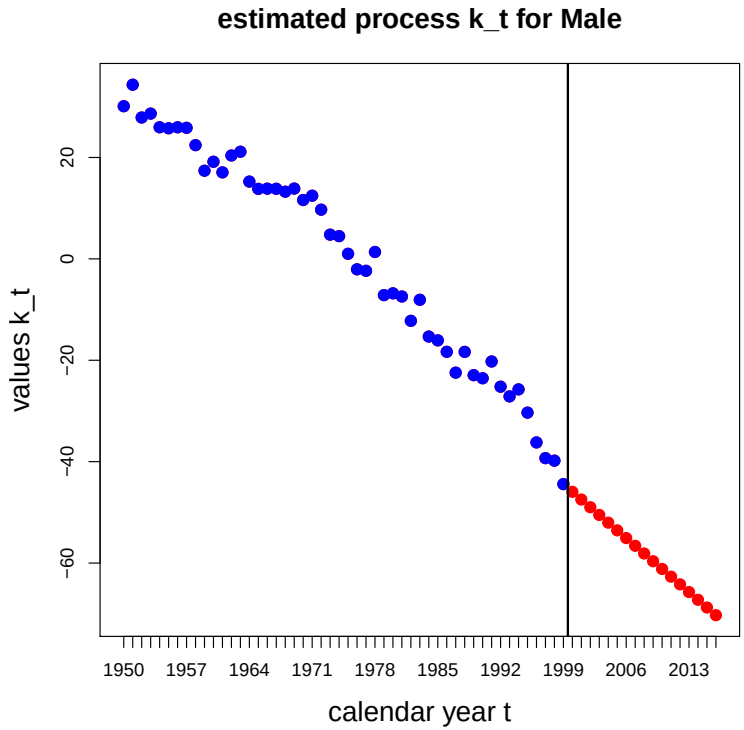
  under side constraint for identifiability $\sum_x \widehat{b}_x^{(i)} = 1$; and $\sum_{t \in \mathcal{D}} \widehat{k}_t^{(i)} = 0$.

- Extrapolate time series $(\widehat{k}_t^{(i)})_{t \in \mathcal{D}}$ using a random walk with drift.

- A random walk with drift often works surprisingly well.

Lee–Carter Forecast for Switzerland

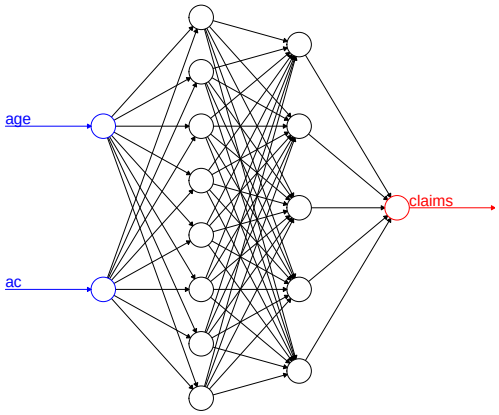




	in-sample MSE		out-of-sample MSE	
	female	male	female	male
LC model with SVD	3.7573	8.8110	0.6045	1.8152

- **Recurrent Neural Networks (RNNs)**

Recap: Feed-Forward Neural Networks (FNNs)



- Deep FNN mapping

$$\boldsymbol{x} \;\mapsto\; \mu = \mathbb{E}[Y] = g^{-1}\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) \right\rangle .$$

- **Goal:** Use time series input $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$ to predict output Y.
- The input of this FNN grows whenever we have a new observation $\boldsymbol{x}_t \in \mathbb{R}^{\tau_0}$.
- This FNN does **not** respect time series (causality) structure.

Plain-Vanilla Recurrent Neural Network (RNN)

- Define a recursive structure using a single RNN layer (upper index$^{(1)}$)

$$\boldsymbol{z}^{(1)} : \mathbb{R}^{\tau_0 \times \tau_1} \to \mathbb{R}^{\tau_1}, \qquad (\boldsymbol{x}_t, \boldsymbol{z}_{t-1}) \;\mapsto\; \boldsymbol{z}_t = \boldsymbol{z}^{(1)}(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}) .$$

- The RNN layer is given by

$$\begin{aligned} \boldsymbol{z}_t &= \boldsymbol{z}^{(1)}(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}) \\ &= \left(\phi\left(\langle \boldsymbol{w}_1^{(1)}, \boldsymbol{x}_t \rangle + \langle \boldsymbol{u}_1^{(1)}, \boldsymbol{z}_{t-1} \rangle \right), \ldots, \phi\left(\langle \boldsymbol{w}_{\tau_1}^{(1)}, \boldsymbol{x}_t \rangle + \langle \boldsymbol{u}_{\tau_1}^{(1)}, \boldsymbol{z}_{t-1} \rangle \right) \right)^\top , \end{aligned}$$

  where the individual neurons $1 \leq j \leq \tau_1$ are modeled by

$$\phi\left(\langle \boldsymbol{w}_j^{(1)}, \boldsymbol{x}_t \rangle + \langle \boldsymbol{u}_j^{(1)}, \boldsymbol{z}_{t-1} \rangle \right) = \phi\left(w_{j,0}^{(1)} + \sum_{l=1}^{\tau_0} w_{j,l}^{(1)} x_{t,l} + \sum_{l=1}^{\tau_1} u_{j,l}^{(1)} z_{t-1,l} \right) .$$

- This RNN has one hidden layer with upper index$^{(1)}$ that is visited T times.

Remarks on RNNs

- Lower index t in $\boldsymbol{z}_t = \boldsymbol{z}^{(1)}(\boldsymbol{x}_t, \boldsymbol{z}_{t-1})$ is time and upper index$^{(1)}$ is the hidden layer.
- This gives time series structure:

$$\cdots \quad \mapsto \quad \boldsymbol{z}_t = \boldsymbol{z}^{(1)}(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}) \quad \mapsto \quad \boldsymbol{z}_{t+1} = \boldsymbol{z}^{(1)}(\boldsymbol{x}_{t+1}, \boldsymbol{z}_t) \quad \mapsto \quad \cdots$$

- Network weights $(\boldsymbol{w}_1^{(1)}, \ldots, \boldsymbol{w}_{\tau_1}^{(1)})^\top \in \mathbb{R}^{\tau_1 \times (\tau_0+1)}$ and $(\boldsymbol{u}_1^{(1)}, \ldots, \boldsymbol{u}_{\tau_1}^{(1)})^\top \in \mathbb{R}^{\tau_1 \times \tau_1}$ are time independent (are shared across every t-loop).
- We have an auto-regressive structure of order 1 in $(\boldsymbol{z}_t)_t$ summarizing the past history; this structure also resembles a state-space model.
- There are different ways in designing RNNs with multiple hidden layers. We give examples of two hidden layers, i.e. depth $d = 2$.

Variants with 2 Hidden RNN Layers

- 1st variant of a two-hidden layer RNN:

$$\begin{aligned} \boldsymbol{z}_t^{(1)} &= \boldsymbol{z}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}^{(1)}\right), \\ \boldsymbol{z}_t^{(2)} &= \boldsymbol{z}^{(2)}\left(\boldsymbol{z}_t^{(1)}, \boldsymbol{z}_{t-1}^{(2)}\right). \end{aligned}$$

- 2nd variant of a two-hidden layer RNN:

$$\begin{aligned} \boldsymbol{z}_t^{(1)} &= \boldsymbol{z}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}^{(1)}, \boldsymbol{z}_{t-1}^{(2)}\right), \\ \boldsymbol{z}_t^{(2)} &= \boldsymbol{z}^{(2)}\left(\boldsymbol{z}_t^{(1)}, \boldsymbol{z}_{t-1}^{(2)}\right). \end{aligned}$$

- 3rd variant of a two-hidden layer RNN:

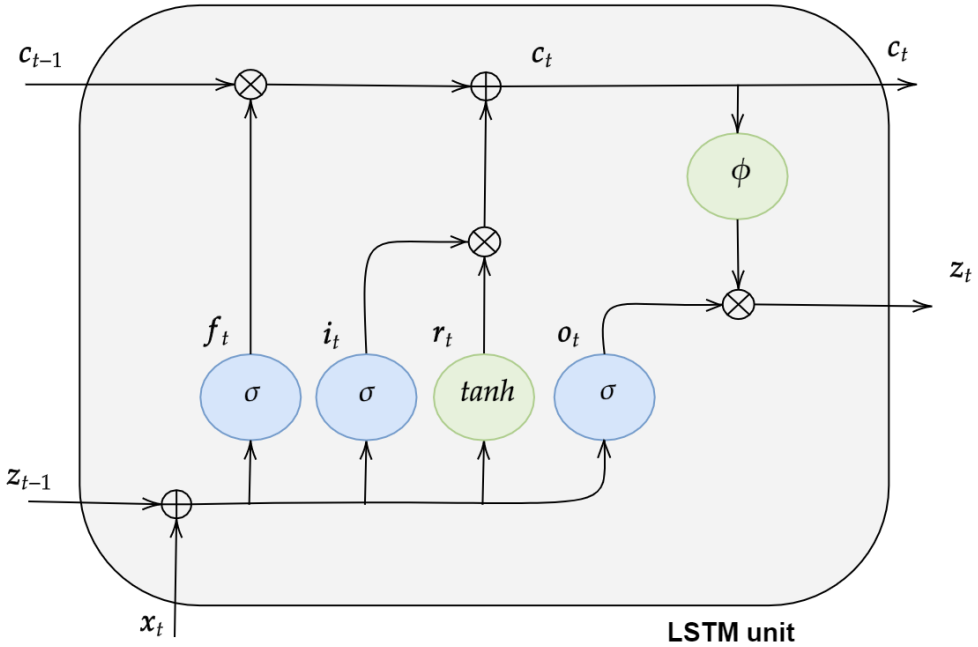
$$\begin{aligned} \boldsymbol{z}_t^{(1)} &= \boldsymbol{z}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}^{(1)}, \boldsymbol{z}_{t-1}^{(2)}\right), \\ \boldsymbol{z}_t^{(2)} &= \boldsymbol{z}^{(2)}\left(\boldsymbol{x}_t, \boldsymbol{z}_t^{(1)}, \boldsymbol{z}_{t-1}^{(2)}\right). \end{aligned}$$

- **Long Short-Term Memory (LSTM) Networks**

Long Short-Term Memory (LSTM) Networks

- The above plain-vanilla RNN architecture is of auto-regressive type of order 1.

- Long short-term memory (LSTM) networks were introduced by Hochreiter–Schmidhuber (1997): design a RNN architecture that can store information for "longer" by using a so-called memory cell c_t.



LSTM Layer: The 3 Gates

- **Forget Gate** (loss of memory rate):

$$\boldsymbol{f}_t = \boldsymbol{f}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}\right) = \phi_\sigma\left(\langle W_f, \boldsymbol{x}_t\rangle + \langle U_f, \boldsymbol{z}_{t-1}\rangle\right) \in (0,1)^{\tau_1}.$$

- **Input Gate** (memory update rate):

$$\boldsymbol{i}_t = \boldsymbol{i}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}\right) = \phi_\sigma\left(\langle W_i, \boldsymbol{x}_t\rangle + \langle U_i, \boldsymbol{z}_{t-1}\rangle\right) \in (0,1)^{\tau_1}.$$

- **Output Gate** (release of memory information rate):

$$\boldsymbol{o}_t = \boldsymbol{o}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}\right) = \phi_\sigma\left(\langle W_o, \boldsymbol{x}_t\rangle + \langle U_o, \boldsymbol{z}_{t-1}\rangle\right) \in (0,1)^{\tau_1}.$$

- Network weights are given by $W_f^\top, W_i^\top, W_o^\top \in \mathbb{R}^{\tau_1 \times (\tau_0+1)}$ (including an intercept), $U_f^\top, U_i^\top, U_o^\top \in \mathbb{R}^{\tau_1 \times \tau_1}$ (excluding an intercept).

LSTM Layer: The Memory Cell

- The above gates determine the release and update of the memory cell $\boldsymbol{c}_t$.
- The memory cell $(\boldsymbol{c}_t)_t$, called *cell state process*, is defined by

$$\begin{aligned} \boldsymbol{c}_t &= \boldsymbol{c}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}, \boldsymbol{c}_{t-1}\right) \\ &= \boldsymbol{f}_t \otimes \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \otimes \phi_{\tanh}\left(\langle W_c, \boldsymbol{x}_t\rangle + \langle U_c, \boldsymbol{z}_{t-1}\rangle\right) \in \mathbb{R}^{\tau_1}, \end{aligned}$$

  for weights $W_c^\top \in \mathbb{R}^{\tau_1 \times (\tau_0+1)}$ (incl. intercept), $U_c^\top \in \mathbb{R}^{\tau_1 \times \tau_1}$ (excl. intercept), and Hadamard product $\otimes$ (element-wise product).
- Finally, define the updated neuron activation, given $\boldsymbol{c}_{t-1}$ and $\boldsymbol{z}_{t-1}$, by

$$\boldsymbol{z}_t = \boldsymbol{z}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}, \boldsymbol{c}_{t-1}\right) = \boldsymbol{o}_t \otimes \phi\left(\boldsymbol{c}_t\right) \in \mathbb{R}^{\tau_1}.$$

- This is one LSTM layer indicated by the upper index$^{(1)}$.

Outputs and Time-Distributed Layers

- The LSTM produces a latent variable $\boldsymbol{z}_T$, based on time series input $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$.
- LSTM prediction: choose link function g and set

$$(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \;\mapsto\; \mu_{T+1} = \mathbb{E}\left[Y_{T+1}\right] = g^{-1}\left\langle \boldsymbol{\beta}, \boldsymbol{z}_T \right\rangle.$$

- Network weights $W_f^\top, W_i^\top, W_o^\top, W_c^\top \in \mathbb{R}^{\tau_1 \times (\tau_0+1)}$, $U_f^\top, U_i^\top, U_o^\top, U_c^\top \in \mathbb{R}^{\tau_1 \times \tau_1}$ and $\boldsymbol{\beta} \in \mathbb{R}^{(\tau_1+1)\times \dim(Y_{T+1})}$. All time t-independent.
- The LSTM produces a latent time series $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T$. A so-called *time-distributed layer* outputs all of them such that we can fit

$$(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t) \;\mapsto\; \mu_{t+1} = \mathbb{E}\left[Y_{t+1}\right] = g^{-1}\left\langle \boldsymbol{\beta}, \boldsymbol{z}_t \right\rangle,$$

using the same output filter $g^{-1}\left\langle \boldsymbol{\beta}, \cdot \right\rangle$ for all $t = 1, \ldots, T$.

- **Code LSTM Layers and Networks**

R Code for Single LSTM Layer Architecture

```
T    <- 10    # length of time series x_1,...,x_T
tau0 <- 3     # dimension of inputs x_t
tau1 <- 5     # dimension of the neurons z_t and cell states c_t

Input <- layer_input(shape=c(T,tau0), dtype='float32', name='Input')

Output = Input %>%
 layer_lstm(units=tau1,activation='tanh',recurrent_activation='tanh',name='LSTM1')%>
 layer_dense(units=1, activation='exponential', name="Output")

model <- keras_model(inputs = list(Input), outputs = c(Output))
```

```
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           (None, 10, 3)             0
_________________________________________________________________
LSTM1 (LSTM)                 (None, 5)                 180
_________________________________________________________________
Output (Dense)               (None, 1)                 6
=================================================================
Total params: 186
Trainable params: 186
Non-trainable params: 0
```

R Code for LSTM Time-Distribution

```
Output = Input %>%
  layer_lstm(units=tau1,activation='tanh',recurrent_activation='tanh',
                                          return_sequences=TRUE, name='LSTM1') %>%
  time_distributed( layer_dense(units=1,activation='exponential',
                                              name="Output"), name='TD')
```

```
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           (None, 10, 3)             0
_________________________________________________________________
LSTM1 (LSTM)                 (None, 10, 5)             180
_________________________________________________________________
TD (TimeDistributed)         (None, 10, 1)             6
=================================================================
Total params: 186
Trainable params: 186
Non-trainable params: 0
```

R Code for Deep LSTMs

```
tau2 <- 4

Output = Input %>%
  layer_lstm(units=tau1,activation='tanh',recurrent_activation='tanh',
                                          return_sequences=TRUE, name='LSTM1') %>%
  layer_lstm(units=tau2,activation='tanh',recurrent_activation='tanh',name='LSTM2')%
  layer_dense(units=1, activation='exponential', name="Output")
```

```
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           (None, 10, 3)             0
_________________________________________________________________
LSTM1 (LSTM)                 (None, 10, 5)             180
_________________________________________________________________
LSTM2 (LSTM)                 (None, 4)                 160
_________________________________________________________________
Output (Dense)               (None, 1)                 5
=================================================================
Total params: 345
Trainable params: 345
Non-trainable params: 0
```

- **Gated Recurrent Unit (GRU) Networks**

Gated Recurrent Unit (GRU) Networks

- A shortcoming of LSTMs is their complexity.
- Gated recurrent unit (GRU) networks were introduced by Cho et al. (2014).
- They should share similar properties as LSTMs but based on less parameters.

GRU Layer

- **Reset gate**:

$$\boldsymbol{r}_t = \boldsymbol{r}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}\right) = \phi_\sigma\left(\langle W_r, \boldsymbol{x}_t\rangle + \langle U_r, \boldsymbol{z}_{t-1}\rangle\right) \in (0,1)^{\tau_1}.$$

- **Update gate**:

$$\boldsymbol{u}_t = \boldsymbol{u}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}\right) = \phi_\sigma\left(\langle W_u, \boldsymbol{x}_t\rangle + \langle U_u, \boldsymbol{z}_{t-1}\rangle\right) \in (0,1)^{\tau_1}.$$

- **Latent time series $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T$:**

$$\boldsymbol{z}_t = \boldsymbol{z}^{(1)}\left(\boldsymbol{x}_t, \boldsymbol{z}_{t-1}\right) = \boldsymbol{r}_t \otimes \boldsymbol{z}_{t-1} + (\mathbf{1} - \boldsymbol{r}_t) \otimes \phi\left(\langle W_z, \boldsymbol{x}_t\rangle + \boldsymbol{u}_t \circ \langle U_z, \boldsymbol{z}_{t-1}\rangle\right) \in \mathbb{R}^{\tau_1},$$

  thus, we consider a credibility weighted average for the update of $\boldsymbol{z}_t$, this can also be understood as a skip connection.

- Network weights are given by $W_r^\top, W_u^\top, W_z^\top \in \mathbb{R}^{\tau_1 \times (\tau_0+1)}$ (including an intercept), $U_r^\top, U_u^\top, U_z^\top \in \mathbb{R}^{\tau_1 \times \tau_1}$ (excluding an intercept).

R Code for Single GRU Layer Architecture

```
T    <- 10    # length of time series x_1,...,x_T
tau0 <- 3     # dimension of inputs x_t
tau1 <- 5     # dimension of the neurons z_t and cell states c_t

Input <- layer_input(shape=c(T,tau0), dtype='float32', name='Input')

Output = Input %>%
 layer_gru(units=tau1,activation='tanh',recurrent_activation='tanh',name='GRU1')%>%
 layer_dense(units=1, activation='exponential', name="Output")

model <- keras_model(inputs = list(Input), outputs = c(Output))
```

```
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           (None, 10, 3)             0
_________________________________________________________________
GRU1 (GRU)                   (None, 5)                 135
_________________________________________________________________
Output (Dense)               (None, 1)                 6
=================================================================
Total params: 141
Trainable params: 141
Non-trainable params: 0
```

- **Example: Mortality Modeling**

Toy Example of LSTMs and GRUs

- Consider raw Swiss female log-mortality rates $\log(M_{x,t})$ for calendar years $1990, \ldots, 2001$ and ages $0 \leq x \leq 99$.

- Set $T = 10$ and $\tau_0 = 3$. Define for ages $1 \leq x \leq 98$ and years $1 \leq t \leq T$ features

$$\boldsymbol{x}_{x,t} = \Big(\log(M_{x-1,1999-(T-t)}), \log(M_{x,1999-(T-t)}), \log(M_{x+1,1999-(T-t)}) \Big)^\top \in \mathbb{R}^{\tau_0},$$

and observations

$$Y_{x,T+1} = \log(M_{x,2000}) \;=\; \log(M_{x,1999-(T-(T+1))}).$$

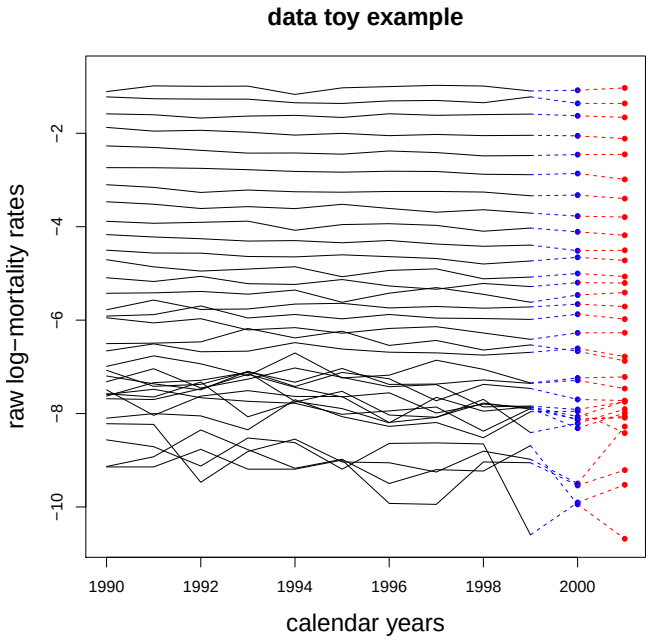
- Based on these definitions, choose training data

$$\mathcal{D} = \{(\boldsymbol{x}_{x,1}, \ldots, \boldsymbol{x}_{x,T}; Y_{x,T+1});\ 1 \leq x \leq 98\}\,.$$

Thus, we have 98 training samples.

Toy Example of LSTMs and GRUs

- Consider ages $(x-1, x, x+1)$ simultaneously in $\boldsymbol{x}_{x,t}$ to smooth inputs over neighboring ages to predict the central mortality rate $Y_{x,T+1}$.

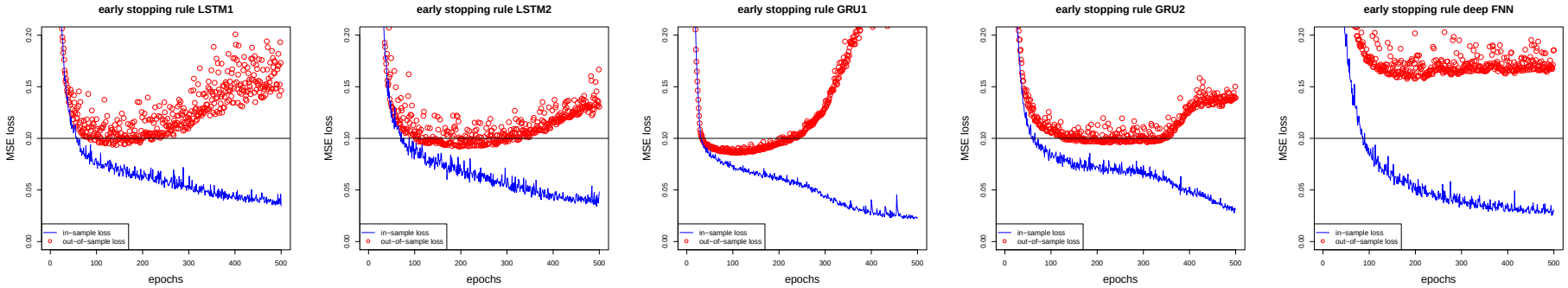


  - ⋆ black lines: explanatory variables $(\boldsymbol{x}_{x,t})_{1\le t\le T}$ (input data)
  - ⋆ blue dots: response variables $Y_{x,T+1}$ (for training)
  - ⋆ test data $\mathcal{T} = \{(\boldsymbol{x}_{x,2}, \ldots, \boldsymbol{x}_{x,T+1}; Y_{x,T+2});\ 1 \le x \le 98\}$ (shifted data); or alternatively $\mathcal{T}_+ = \{(\boldsymbol{x}_{x,1}, \ldots, \boldsymbol{x}_{x,T+1}; Y_{x,T+2});\ 1 \le x \le 98\}$ (expanded data)

Toy Example of LSTMs and GRUs

- Pre-process all variables $x_{x,t}$ with MinMaxScaler to domain $[-1, 1]$.

- Use shallow LSTM1, deep LSTM2 as above, and corresponding GRU1, GRU2, and deep FNN; GDM: blue is in-sample, red is out-of-sample



	# param.	epochs	run time	in-sample loss	out-of-sample loss
LSTM1	186	150	8 sec	0.0655	0.0936
LSTM2	345	200	15 sec	0.0603	0.0918
GRU1	141	100	5 sec	0.0671	0.0860
GRU2	260	200	14 sec	0.0651	0.0958
deep FNN	184	200	5 sec	0.0485	0.1577

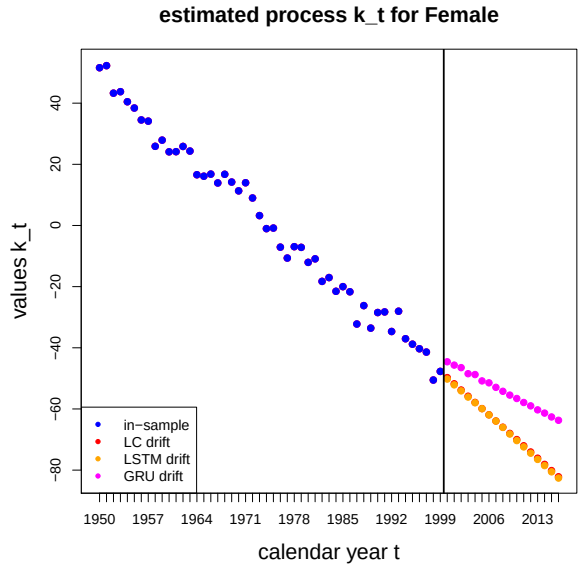
- In general: LSTMs seem more robust than GRUs.

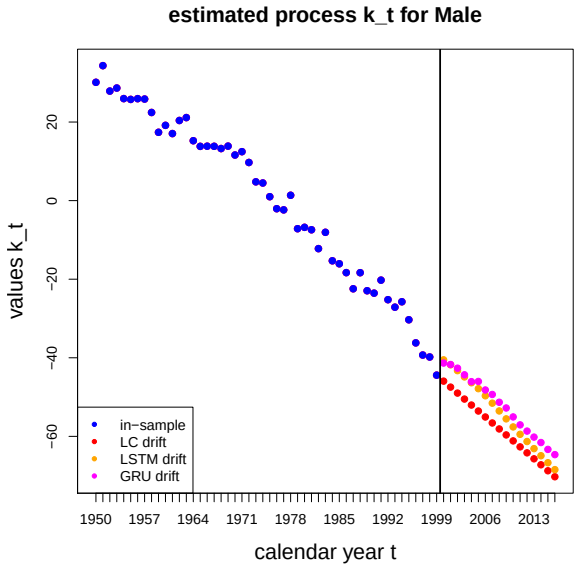
Hyper-Parameters in LSTMs

	# param.	epochs	run time	in-sample	out-of-sample
base case: LSTM1 ($T = 10$, $\tau_0 = 3$, $\tau_1 = 5$)	186	150	8 sec	0.0655	0.0936
LSTM1 ($T = 10$, $\tau_0 = 1$, $\tau_1 = 5$)	146	100	5 sec	0.0681	0.0994
LSTM1 ($T = 10$, $\tau_0 = 5$, $\tau_1 = 5$)	226	150	15 sec	0.0572	0.0795
LSTM1 ($T = 5$, $\tau_0 = 3$, $\tau_1 = 5$)	186	100	4 sec	0.0753	0.1028
LSTM1 ($T = 20$, $\tau_0 = 3$, $\tau_1 = 5$)	186	200	16 sec	0.0678	0.0914
LSTM1 ($T = 10$, $\tau_0 = 3$, $\tau_1 = 3$)	88	200	10 sec	0.0614	0.1077
LSTM1 ($T = 10$, $\tau_0 = 3$, $\tau_1 = 10$)	571	100	5 sec	0.0667	0.0962

Application to Swiss Mortality Data

	in-sample		out-of-sample		run times	
	female	male	female	male	female	male
LSTM3 ($T = 10$, $(\tau_0, \tau_1, \tau_2, \tau_3) = (5, 20, 15, 10)$)	2.5222	6.9458	0.3566	1.3507	225s	203s
GRU3 ($T = 10$, $(\tau_0, \tau_1, \tau_2, \tau_3) = (5, 20, 15, 10)$)	2.8370	7.0907	0.4788	1.2435	185s	198s
LC model with SVD	3.7573	8.8110	0.6045	1.8152	–	–



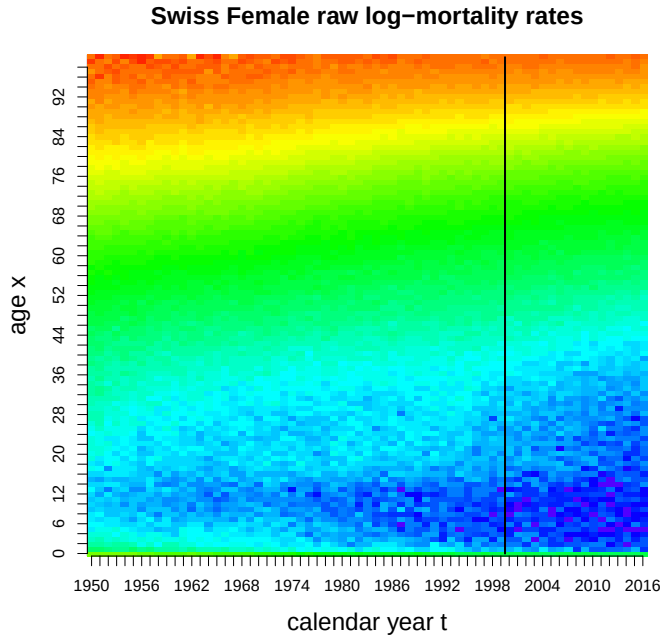


- Main difficulty: Robustness of results.
- More stability by simultaneous multi-population modeling.
- For more sophisticated models see Perla et al. (2020).

- **RNNs vs. Convolutional Neural Networks (CNNs)**

RNNs vs. Convolutional Neural Networks (CNNs)

- RNNs respect time series structures.
- Convolutional neural networks (CNNs) respect local spatial structure.
- Intuitively, for CNNs we move small windows (filters) over the images to discover similar structure at different locations in the images.



Convolutional Neural Networks (CNNs)

- CNNs have been introduced in Fukushima (1980).
- CNNs used for image and speech recognition, natural language processing (NLP), and in many other fields, for references see our tutorial Meier–Wüthrich (2020).
- Main feature of CNNs is translation invariance, see Wiatowski–Bölcskei (2018).

Convolutional Layer: Sketch of Structure

A convolution layer (we consider a two-dimensional image here and a single filter)

$$\boldsymbol{z}^{(m)} \colon \mathbb{R}^{n_1^{(m-1)} \times n_2^{(m-1)}} \to \mathbb{R}^{n_1^{(m)} \times n_2^{(m)}}, \quad \boldsymbol{x} \mapsto \begin{pmatrix} z_{1,1}^{(m)}(\boldsymbol{x}) & \cdots & z_{1,n_2^{(m)}}^{(m)}(\boldsymbol{x}) \\ \vdots & & \vdots \\ z_{n_1^{(m)},1}^{(m)}(\boldsymbol{x}) & \cdots & z_{n_1^{(m)},n_2^{(m)}}^{(m)}(\boldsymbol{x}) \end{pmatrix},$$

with (local) filter/window having filter size $f_1^{(m)}$ and $f_2^{(m)}$

$$\boldsymbol{x} \mapsto z_{i_1,i_2}^{(m)}(\boldsymbol{x}) = \phi\left(w_{0,0}^{(m)} + \sum_{j_1=1}^{f_1^{(m)}} \sum_{j_2=1}^{f_2^{(m)}} w_{j_1,j_2}^{(m)} x_{i_1+j_1-1,i_2+j_2-1} \right).$$

- ⋆ In our tutorial we add activation ϕ only later (after batch normalization).
- ⋆ We illustrate a single filter, multiple filters are used to extract different features.
- ⋆ Multiple filters require three-dimensional inputs in deep CNNs.
- ⋆ Pooling layers, flatten layers and so-called padding is used.

References

- Cho, van Merrienboer, Gulcehre, Bahdanau, Bougares, Schwenk, Bengio (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. $arXiv$:1406.1078.
- Efron, Hastie (2016). Computer Age Statistical Inference: Algorithms, Evidence, and Data Science. Cambridge UP.
- Fukushima (1980). Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36/4, 193-202.
- Goodfellow, Bengio, Courville (2016). Deep Learning. MIT Press.
- Hastie, Tibshirani, Friedman (2009). The Elements of Statistical Learning. Springer.
- Hochreiter, Schmidhuber (1997). Long short-term memory. Neural Computation 9/8, 1735-1780.
- Lee, Carter (1992). Modeling and forecasting US mortality. Journal American Statistical Association 87/419, 659-671.
- Meier, Wüthrich (2020). Convolutional neural network case studies: (1) anomalies in mortality rates (2) image recognition. SSRN 3656210.
- Perla, Richman, Scognamiglio, Wüthrich (2020). Time-series forecasting of mortality rates using deep learning. SSRN 3595426.
- Richman, Wüthrich (2019). Lee and Carter go machine learning. SSRN 3441030.
- Richman, Wüthrich (2020). A neural network extension of the Lee–Carter model to multiple populations. Annals of Actuarial Science.
- Smyl (2019). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. International Journal of Forecasting.
- Wiatowski, Bölcskei (2018). A mathematical theory of deep convolutional neural networks for feature extraction. IEEE Transactions on Information Theory 64/3, 1845-1866.
- Wilmoth, Shkolnikov (2010). Human Mortality Database. University of California.
- Wüthrich, Buser (2016). Data Analytics for Non-Life Insurance Pricing. SSRN 2870308, Version September 10, 2020.

Wrap Up

Mario V. Wüthrich
RiskLab, ETH Zurich

"Deep Learning with Actuarial Applications in R"
Swiss Association of Actuaries SAA/SAV, Zurich
October 14/15, 2021

Programme SAV Block Course

- Refresher: Generalized Linear Models (THU 9:00-10:30)
- Feed-Forward Neural Networks (THU 13:00-15:00)
- Discrimination-Free Insurance Pricing (THU 17:15-17:45)

- LocalGLMnet (FRI 9:00-10:30)
- Convolutional Neural Networks (FRI 13:00-14:30)
- Wrap Up (FRI 16:00-16:30)

Exponential Dispersion Family

- EDF provides a unified notation for a big class of distribution functions.
- EDF has the following structure

$$Y_i \sim f(y;\theta_i, v_i/\varphi) = \exp\left\{\frac{y\theta_i - \kappa(\theta_i)}{\varphi/v_i} + a(y; v_i/\varphi)\right\},$$

with expected mean, and canonical link $h = (\kappa')^{-1}$,

$$\mu_i = \mathbb{E}[Y_i] = \kappa'(\theta_i) \qquad \Longleftrightarrow \qquad h(\mu_i) = \theta_i.$$

- This family contains the Gauss, Poisson, binomial, negative binomial, gamma, inverse Gaussian, and Tweedie's models.
- The cumulant function κ determines the distribution type, and the canonical parameter θ_i is estimated with MLE.

Generalized Linear Models

- GLMs are the basis for regression modeling.
- We start from the EDF

$$Y_i \sim f(y;\theta_i,v_i/\varphi)=\exp\left\{\frac{y\theta_i-\kappa(\theta_i)}{\varphi/v_i}+a(y;v_i/\varphi)\right\},$$

  with mean $\mu_i=\kappa'(\theta_i)$.

- A GLM chooses a link g and assumes linear structure (in covariates)

$$\boldsymbol{x}_i \mapsto g(\mu_i)=g(\mathbb{E}[Y_i])=\langle\boldsymbol{\beta},\boldsymbol{x}_i\rangle=\beta_0+\sum_{j=1}^{q}\beta_j x_{i,j},$$

  for covariate information $\boldsymbol{x}_i\in\mathbb{R}^q$ of insurance policy i.

Generalized Linear Models: Fitting

- GLMs are fit using MLE.
- Maximizing (log-)likelihoods is equivalent to minimizing deviance losses

$$\begin{aligned} D^*(\boldsymbol{Y}, \boldsymbol{\beta}) &= 2\left[\ell_{\boldsymbol{Y}}(\boldsymbol{Y}) - \ell_{\boldsymbol{Y}}(\boldsymbol{\beta})\right] \\ &= 2\sum_{i=1}^{n} \frac{v_i}{\varphi}\Big[Y_i h(Y_i) - \kappa(h(Y_i)) - Y_i h(\mu_i) + \kappa(h(\mu_i))\Big] \;\geq\; 0. \end{aligned}$$

- Deviance losses are distribution-adapted loss functions.
- For canonical link $g = h$, the fitted model fulfills the balance property

$$\sum_{i=1}^{n} v_i \widehat{\mathbb{E}}[Y_i] \;=\; \sum_{i=1}^{n} v_i \kappa'\langle \widehat{\boldsymbol{\beta}}, \boldsymbol{x}_i\rangle = \sum_{i=1}^{n} v_i Y_i.$$

Otherwise, one should adjust the intercept β_0.

(Feed-Forward) Neural Networks

- Neural networks can be seen as extensions of GLMs.
- Neural networks perform representation learning:

$$\boldsymbol{x}_i \;\mapsto\; \mu_i = \mathbb{E}[Y_i] = g^{-1}\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_i) \right\rangle,$$

  with learned representation $\boldsymbol{z}_i = \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_i)$ of $\boldsymbol{x}_i$.

- Neural network

$$\boldsymbol{x} \;\mapsto\; \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) \;=\; \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}),$$

  processes information $\boldsymbol{x}$ into a suitable form.

- The family of networks fulfills the universality property.

(Feed-Forward) Neural Networks

- Neural networks perform representation learning:

$$\boldsymbol{x}_i \;\mapsto\; \mu_i = \mathbb{E}[Y_i] = g^{-1}\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_i) \right\rangle,$$

  with learned representation $\boldsymbol{z}_i = \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_i)$ of $\boldsymbol{x}_i$.

- Deep networks should be preferred to capture interactions more efficiently.

- Categorical variables can be integrated into (so-called) embedding layers.

- Time series, image and text data can be processed (in a similar fashion), using different types of network architectures, but the general philosophy of representation learning is the same.

(Feed-Forward) Neural Networks

- Neural networks perform representation learning:

$$\boldsymbol{x}_i \;\mapsto\; \mu_i = \mathbb{E}[Y_i] = g^{-1}\left\langle \boldsymbol{\beta}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_i) \right\rangle,$$

  with learned representation $\boldsymbol{z}_i = \boldsymbol{z}^{(d:1)}(\boldsymbol{x}_i)$ of $\boldsymbol{x}_i$.

- We have output parameter $\boldsymbol{\beta} \in \mathbb{R}^{q_d+1}$, and each hidden layer $\boldsymbol{z}^{(m)}$ has parameters (weights) $(\boldsymbol{w}_1^{(m)}, \ldots, \boldsymbol{w}_{q_m}^{(m)}) \in \mathbb{R}^{q_m(q_{m-1}+1)}$.

- This network parameter $\vartheta = (\boldsymbol{w}_1^{(1)}, \ldots, \boldsymbol{w}_{q_d}^{(d)}, \boldsymbol{\beta})$ is fit with gradient descent methods, and early stopping is used to prevent from (in-sample) over-fitting.

- Every different seed (starting point) of the gradient descent will provide a different (early stopped) network calibration $\widehat{\vartheta}$.

(Feed-Forward) Neural Networks: Peculiarities

- There is no "unique best" network, but there are infinitely many sufficiently (equally) good networks.
- These sufficiently good networks have equally good predictive power on portfolio level, but they can be quite different on policy level.
- Aggregating/ensembling/nagging helps to reduce noise and, generally, improves predictive models.
- Typically, the balance property fails to hold. This requires an extra fitting (bias regularization) step.
- The LocalGLMnet provides an explainable network architecture, that allows for variable selection, for a variable importance measure, for the study of interactions.
- There is a LASSO version of LocalGLMnet.
- Based on the learned structure one can still try to improve a GLM.

Convolutional Neural Networks

- CNNs process image and time series data.
- FNNs act globally, CNNs act locally.
- CNNs extract local structure via (small size) filters (windows).
- Time series data and text data can also be process through RNNs (not presented here, but there is a SAV tutorial). A RNN is a FNN with loops.
- Often one uses regression trees, random forests and tree boosting as competing data science models to FNNs.
- This works for tabular data, however, time series, image and text data does not have obvious non-network counterparts.

References: www.ActuarialDataScience.org

- Ferrario, Hämmerli (2019). On boosting: theory and applications. SSRN 3402687.
- Ferrario, Nägelin (2020). The art of natural language processing: classical, modern and contemporary approaches to text document classification. SSRN 3547887.
- Ferrario, Noll, Wüthrich (2018). Insights from inside neural networks. SSRN 3226852.
- Lorentzen, Mayer (2020). Peeking into the black box: an actuarial case study for interpretable machine learning. SSRN 3595944.
- Meier, Wüthrich (2020). Convolutional neural network case studies: (1) anomalies in mortality rates (2) image recognition. SSRN 3656210.
- Noll, Salzmann, Wüthrich (2018). Case study: French motor third-party liability claims. SSRN 3164764.
- Rentzmann, Wüthrich (2019). Unsupervised learning: What is a sports car? SSRN 3439358.
- Richman, Wüthrich (2019). Lee and Carter go machine learning: recurrent neural networks. SSRN 3441030.
- Schelldorfer, Wüthrich (2019). Nesting classical actuarial models into neural networks. SSRN 3320525.
- Schelldorfer, Wüthrich (2021). LocalGLMnet: a deep learning architecture for actuaries. SSRN 3900350.

Many thanks for attending!