IB031 Mice Protein Expression

Vladimír Bača

May 23, 2019

Dataset

Vybraný dataset Mice Protein Expression (https://archive.ics.uci.edu/ml/datasets/Mice+Protein+Expression) obsahuje úrovne expresie proteínov v mozgoch myší, ktoré sú z 8 rôznych kontrolných skupín. Kontrolná skupina je kombinácia 3 binárnych atribútov: Genotyp (S/Bez Downovho syndrómu), Liek (Memantine/Bez lieku) a Stimulácia (S/Bez). Úlohou bude naučiť model určiť kontrolnú skupinu na základe expresií proteínov.

Exploračná analýza

Prvý stĺpec datasetu (MouseID) obsahuje ID myší a nebude pre nás relevantný. Stĺpce 2 - 78 obsahujú expresie jednotlivých proteínov a budú predstavovať vstupné dáta. Stĺpce 79 - 81 (Genotype, Treatment, Behavior) sú binárne atribúty, ktoré určujú danú kontrolnú skupinu a budú pre nás výstupom. Posledný stĺpec (class) obsahuje už len reťazec kombinujúci predchádzajúce tri stĺpce. Spolu teda máme 77 vstupných atribútov (kladné reálne čísla) a 3 výstupné binárne atribúty. Dataset obsahuje 1080 inštancií.

```
data <- read.csv(file="dataset.csv", header=TRUE, sep=",")
str(data)
```

```
## 'data.frame':    1080 obs. of  82 variables:
##  $ MouseID        : Factor w/ 1080 levels "18899_1","18899_10",..: 46 53 54 55 56 57 58 59 60 47 ...
##  $ DYRK1A_N       : num  0.504 0.515 0.509 0.442 0.435 ...
##  $ ITSN1_N        : num  0.747 0.689 0.73 0.617 0.617 ...
##  $ BDNF_N         : num  0.43 0.412 0.418 0.359 0.359 ...
##  $ NR1_N          : num  2.82 2.79 2.69 2.47 2.37 ...
##  $ NR2A_N         : num  5.99 5.69 5.62 4.98 4.72 ...
##  $ pAKT_N         : num  0.219 0.212 0.209 0.223 0.213 ...
##  $ pBRAF_N        : num  0.178 0.173 0.176 0.176 0.174 ...
##  $ pCAMKII_N      : num  2.37 2.29 2.28 2.15 2.13 ...
##  $ pCREB_N        : num  0.232 0.227 0.23 0.207 0.192 ...
##  $ pELK_N         : num  1.75 1.6 1.56 1.6 1.5 ...
##  $ pERK_N         : num  0.688 0.695 0.677 0.583 0.551 ...
##  $ pJNK_N         : num  0.306 0.299 0.291 0.297 0.287 ...
##  $ PKCA_N         : num  0.403 0.386 0.381 0.377 0.364 ...
##  $ pMEK_N         : num  0.297 0.281 0.282 0.314 0.278 ...
##  $ pNR1_N         : num  1.022 0.957 1.004 0.875 0.865 ...
##  $ pNR2A_N        : num  0.606 0.588 0.602 0.52 0.508 ...
##  $ pNR2B_N        : num  1.88 1.73 1.73 1.57 1.48 ...
##  $ pPKCAB_N       : num  2.31 2.04 2.02 2.13 2.01 ...
##  $ pRSK_N         : num  0.442 0.445 0.468 0.478 0.483 ...
##  $ AKT_N          : num  0.859 0.835 0.814 0.728 0.688 ...
##  $ BRAF_N         : num  0.416 0.4 0.4 0.386 0.368 ...
##  $ CAMKII_N       : num  0.37 0.356 0.368 0.363 0.355 ...
##  $ CREB_N         : num  0.179 0.174 0.174 0.179 0.175 ...
##  $ ELK_N          : num  1.87 1.76 1.77 1.29 1.32 ...
##  $ ERK_N          : num  3.69 3.49 3.57 2.97 2.9 ...
```

```
##  $ GSK3B_N        : num  1.54 1.51 1.5 1.42 1.36 ...
##  $ JNK_N          : num  0.265 0.256 0.26 0.26 0.251 ...
##  $ MEK_N          : num  0.32 0.304 0.312 0.279 0.274 ...
##  $ TRKA_N         : num  0.814 0.781 0.785 0.734 0.703 ...
##  $ RSK_N          : num  0.166 0.157 0.161 0.162 0.155 ...
##  $ APP_N          : num  0.454 0.431 0.423 0.411 0.399 ...
##  $ Bcatenin_N     : num  3.04 2.92 2.94 2.5 2.46 ...
##  $ SOD1_N         : num  0.37 0.342 0.344 0.345 0.329 ...
##  $ MTOR_N         : num  0.459 0.424 0.425 0.429 0.409 ...
##  $ P38_N          : num  0.335 0.325 0.325 0.33 0.313 ...
##  $ pMTOR_N        : num  0.825 0.762 0.757 0.747 0.692 ...
##  $ DSCR1_N        : num  0.577 0.545 0.544 0.547 0.537 ...
##  $ AMPKA_N        : num  0.448 0.421 0.405 0.387 0.361 ...
##  $ NR2B_N         : num  0.586 0.545 0.553 0.548 0.513 ...
##  $ pNUMB_N        : num  0.395 0.368 0.364 0.367 0.352 ...
##  $ RAPTOR_N       : num  0.34 0.322 0.313 0.328 0.312 ...
##  $ TIAM1_N        : num  0.483 0.455 0.447 0.443 0.419 ...
##  $ pP70S6_N       : num  0.294 0.276 0.257 0.399 0.393 ...
##  $ NUMB_N         : num  0.182 0.182 0.184 0.162 0.16 ...
##  $ P70S6_N        : num  0.843 0.848 0.856 0.76 0.768 ...
##  $ pGSK3B_N       : num  0.193 0.195 0.201 0.184 0.186 ...
##  $ pPKCG_N        : num  1.44 1.44 1.52 1.61 1.65 ...
##  $ CDK5_N         : num  0.295 0.294 0.302 0.296 0.297 ...
##  $ S6_N           : num  0.355 0.355 0.386 0.291 0.309 ...
##  $ ADARB1_N       : num  1.34 1.31 1.28 1.2 1.21 ...
##  $ AcetylH3K9_N   : num  0.17 0.171 0.185 0.16 0.165 ...
##  $ RRP1_N         : num  0.159 0.158 0.149 0.166 0.161 ...
##  $ BAX_N          : num  0.189 0.185 0.191 0.185 0.188 ...
##  $ ARC_N          : num  0.106 0.107 0.108 0.103 0.105 ...
##  $ ERBB4_N        : num  0.145 0.15 0.145 0.141 0.142 ...
##  $ nNOS_N         : num  0.177 0.178 0.176 0.164 0.168 ...
##  $ Tau_N          : num  0.125 0.134 0.133 0.123 0.137 ...
##  $ GFAP_N         : num  0.115 0.118 0.118 0.117 0.116 ...
##  $ GluR3_N        : num  0.228 0.238 0.245 0.235 0.256 ...
##  $ GluR4_N        : num  0.143 0.142 0.142 0.145 0.141 ...
##  $ IL1B_N         : num  0.431 0.457 0.51 0.431 0.481 ...
##  $ P3525_N        : num  0.248 0.258 0.255 0.251 0.252 ...
##  $ pCASP9_N       : num  1.6 1.67 1.66 1.48 1.53 ...
##  $ PSD95_N        : num  2.01 2 2.02 1.96 2.01 ...
##  $ SNCA_N         : num  0.108 0.11 0.108 0.12 0.12 ...
##  $ Ubiquitin_N    : num  1.045 1.01 0.997 0.99 0.998 ...
##  $ pGSK3B_Tyr216_N: num  0.832 0.849 0.847 0.833 0.879 ...
##  $ SHH_N          : num  0.189 0.2 0.194 0.192 0.206 ...
##  $ BAD_N          : num  0.123 0.117 0.119 0.133 0.13 ...
##  $ BCL2_N         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ pS6_N          : num  0.106 0.107 0.108 0.103 0.105 ...
##  $ pCFOS_N        : num  0.108 0.104 0.106 0.111 0.111 ...
##  $ SYP_N          : num  0.427 0.442 0.436 0.392 0.434 ...
##  $ H3AcK18_N      : num  0.115 0.112 0.112 0.13 0.118 ...
##  $ EGR1_N         : num  0.132 0.135 0.133 0.147 0.14 ...
##  $ H3MeK4_N       : num  0.128 0.131 0.127 0.147 0.148 ...
##  $ CaNA_N         : num  1.68 1.74 1.93 1.7 1.84 ...
##  $ Genotype       : Factor w/ 2 levels "Control","Ts65Dn": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Treatment      : Factor w/ 2 levels "Memantine","Saline": 1 1 1 1 1 1 1 1 1 1 ...
```

```
##  $ Behavior       : Factor w/ 2 levels "C/S","S/C": 1 1 1 1 1 1 1 1 1 1 ...
##  $ class          : Factor w/ 8 levels "c-CS-m","c-CS-s",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Niektoré vstupné hodnoty chýbajú (1396, t.j. 1,7% chýbajúcich hodnôt).

```
missing_values_count <- length(data[is.na(data)])
missing_values_ratio <- missing_values_count / (77*nrow(data))
missing_values_count
```

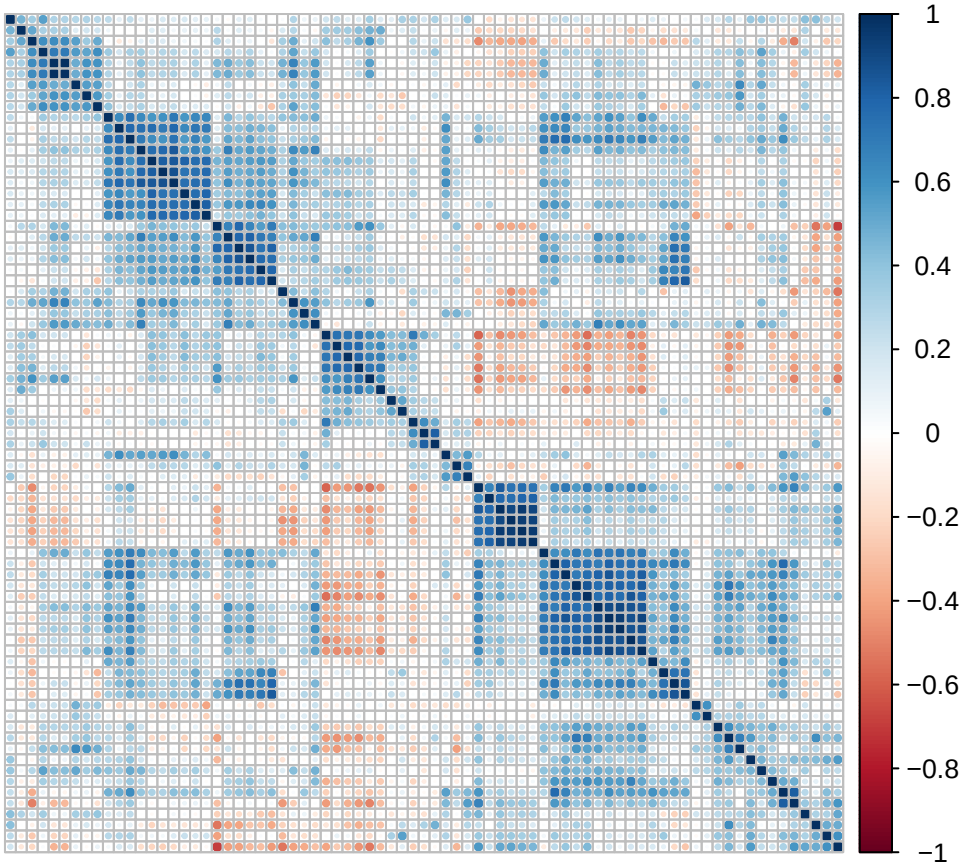
```
## [1] 1396
```

```
missing_values_ratio
```

```
## [1] 0.01678692
```

Vykreslíme si korelačnú maticu vstupných atribútov (zgrupené sú atribúty so silnou koreláciou, chýbajúce dáta sú imputované).

```
imputedData <- kNN(data[, 2:78], k=5, imp_var = FALSE)
corMat <- cor(imputedData)
corrplot(corMat, order = "hclust", tl.pos = FALSE)
```



Ako vidíme, medzi proteínmi existuje niekoľko skupín, ktorých expresia spolu výrazne koreluje. Mohlo by teda byť zaujímavé vyskúšať feature extraction.

Pozrieme sa tiež na distribúciu výstupných tried.

```r
table(data[, 79:81])
```

```
## , , Behavior = C/S
##
##          Treatment
## Genotype  Memantine Saline
##   Control       150    135
##   Ts65Dn        135    105
##
## , , Behavior = S/C
##
##          Treatment
## Genotype  Memantine Saline
##   Control       150    135
##   Ts65Dn        135    135
```

Ako vidíme, triedy sú pomerne rovnomerne distribuované. Z distribúcie tiež možno vyčítať naivný baseline odhad. Tým by bol klasifikátor vyberajúci najpočetnejšiu triedu, t.j. napr. Control-Memantine-C/S. Jeho accuracy by bola $150/1080 = 13{,}9\%$.

Predspracovanie

Nakoľko dáta sú zoradené podľa kontrolných skupín, na začiatok ich náhodne zamiešame. Potom ich rozdelíme na 80% trénovaciu a 20% testovaciu množinu.

```r
data <- data[sample(nrow(data)),]
data_train <- data[1:round(nrow(data)*0.8),]
data_test <- data[(round(nrow(data)*0.8)+1):nrow(data),]
```

Chýbajúce dáta v trénovacej množine imputujeme pomocou metódy kNN. V testovacej množine chýbajúce hodnoty len nahradíme nulou.

```r
data_train <- kNN(data_train, k=5, imp_var = FALSE)

data_test[is.na(data_test)] <- 0
```

Pre spracovanie neurónovou sieťou potrebujeme dáta rozdeliť na maticu vstupov a maticu výstupov. Vstupné hodnoty sú kladné reálne čísla, preto ich len vložíme do matice. Výstupná trieda je zložená z troch binárnych atribútov, transformujeme ju teda na vektor troch čísel 0/1. Neurónová sieť potom bude mať vo výstupnej vrstve tri neuróny, ktoré budeme trénovať na hodnoty 0 alebo 1 podľa konkrétnej triedy.

```r
train_x <- as.matrix(data_train[, 2:78])

genotype <- ifelse(data_train$Genotype == "Control", 0, 1)
treatment <- ifelse(data_train$Treatment == "Memantine", 1, 0)
behaviour <- ifelse(data_train$Behavior == "C/S", 1, 0)

train_y <- matrix(c(genotype, treatment, behaviour), ncol = 3, byrow = FALSE)

test_x <- as.matrix(data_test[, 2:78])
```

```r
genotype <- ifelse(data_test$Genotype == "Control", 0, 1)
treatment <- ifelse(data_test$Treatment == "Memantine", 1, 0)
behaviour <- ifelse(data_test$Behavior == "C/S", 1, 0)

test_y <- matrix(c(genotype, treatment, behaviour), ncol = 3, byrow = FALSE)
```

Multi-layer perceptron

ML model našej skupiny je multi-layer perceptron, teda viacvrstvová neurónová sieť. Táto je zložená z jednotlivých neurónov. Neurón je jednoduchá výpočtová jednotka, inšpirovaná biologickými neurónmi, ktorá má vektor reálnych vstupov a jeden reálny výstup. Výpočet prebieha tak, že jednotlivé vstupy sú najprv vynásobené váhami a potom sčítané. Na výsledok sa aplikuje aktivačná funkcia, typicky sigmoida a jej výstup sa pošle ďalej. Vo viacvrstvovej sieti je niekoľko takýchto neurónov usporiadaných do vrstiev. Spodná vrstva pritom tvorí vstup celej siete a vrchná jej výstup. Každý neurón (okrem tých vo vstupnej vrstve) má za vstup výstupy všetkých neurónov vo vrstve pod ním.

Učenie v tomto modeli má za cieľ nastaviť váhy jednotlivých spojení tak, aby výstup siete na vzorových vstupoch bol čo najbližšie cieľovým výstupom. Prebieha tak, že sieť spracuje danú dávku vstupov a jej výstupy sa porovnajú s požadovanými výstupmi. Rozdiel medzi požadovaným a reálnym výstupom tvorí chybovú funkciu. Na túto sa môžeme pozerať ako na funkciu vektora všetkých váh, ktorú vieme derivovať. Preto ju môžeme optimalizovať pomocou gradientného zostupu. Tento postup iteratívne opakujeme, kým sa sieť zlepšuje (a nepreučuje).

Výber modelu

V našom prípade pre neurónové siete použijeme R balíček keras.

Neurónová sieť bude mať 77 vstupov (77 vstupných atribútov z datasetu). Výstupná vrstva bude mať 3 neuróny, každý bude predstavovať jeden binárny výstupný atribút (0 pre jednu možnosť, 1 pre druhú). Po niekoľkých experimentoch sa ako vhodné ukázali 2 skryté vrstvy. Pre 1 vrstvu bola accuracy výrazne nižšia, 3 vrstvy naopak už na accuracy nepridávali. Ako aktivačná funkcia bola použitá sigmoida, s výnimkou výstupnej vrstvy, kde bola použitá hard_sigmoid. Dôvodom je, že cieľovým výstupom je vždy buď 0 alebo 1, hodnoty medzi nimi len zaokrúhlime. Tento výber mal lepšie výsledky než sigmoida.

Najlepšou nájdenou architektúrou bola sieť so 77-20-10-3 neurónmi. Stratová funkcia bola mse, metrika accuracy.

```r
model <- keras_model_sequential()
model %>%
  layer_dense(units = 20, activation = "sigmoid", input_shape = c(77)) %>%
  layer_dense(units = 10, activation = "sigmoid") %>%
  layer_dense(units = 3, activation = "hard_sigmoid")

model %>% compile(
  loss = 'mse',
  metrics = 'accuracy',
  optimizer = 'rmsprop'
)

summary(model)
```

```
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
```

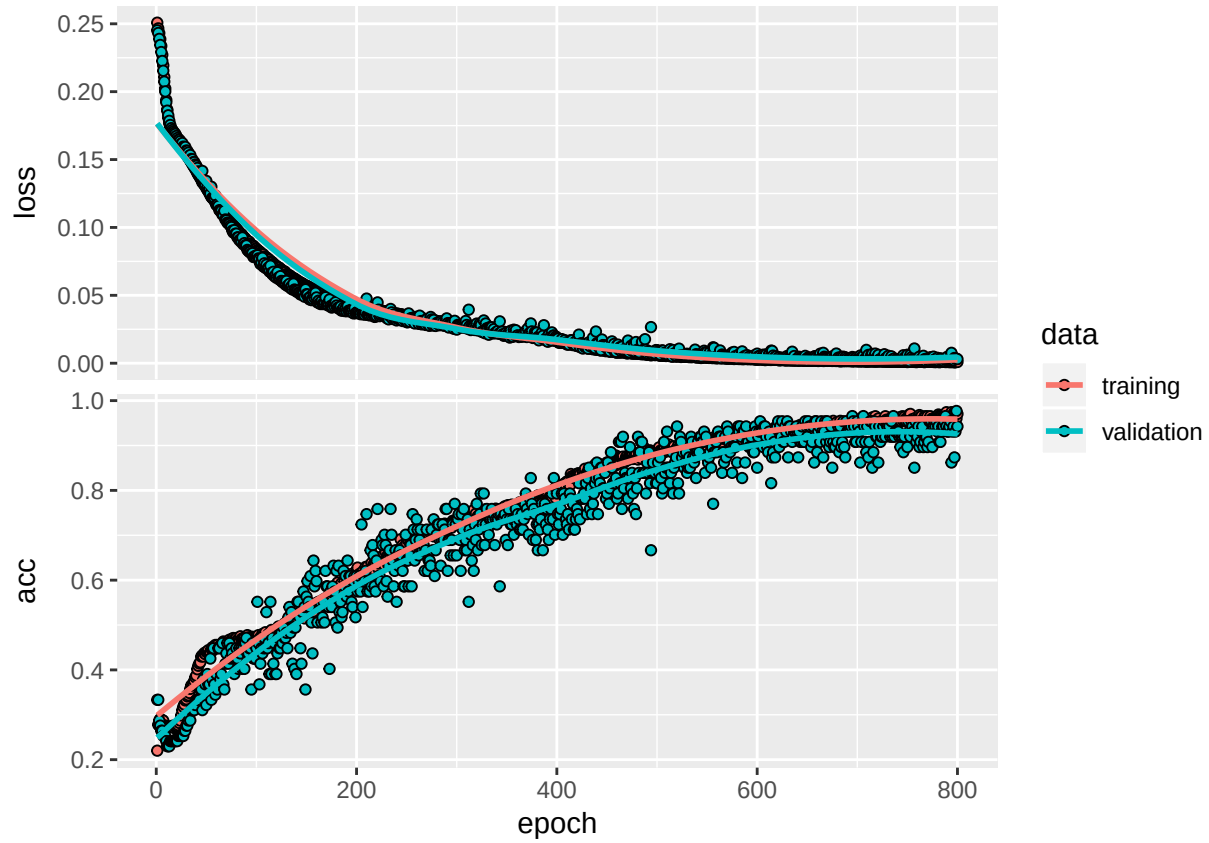
```
## =================================================================================
## dense (Dense)                   (None, 20)                    1560
## _________________________________________________________________________________
## dense_1 (Dense)                 (None, 10)                    210
## _________________________________________________________________________________
## dense_2 (Dense)                 (None, 3)                     33
## =================================================================================
## Total params: 1,803
## Trainable params: 1,803
## Non-trainable params: 0
## _________________________________________________________________________________
```

Tréning

Túto sieť natrénujeme na trénovacej množine.

```
history <- model %>% fit(
  epochs = 800, batch_size = 10,
  x = train_x, y = train_y, validation_split = 0.1
)

plot(history)
```



Výslednú sieť otestujeme na testovacej množine. Keďže ide o klasifikačnú úlohu, ako metriku použijeme accuracy, t.j. podiel správne klasifikovaných inštancií (pre všetky tri atribúty) a všetkých testovaných inš-

tancií. Nakoľko výsledný vektor z neurónovej siete treba zaokrúhliť, počítanie výslednej accuracy je mierne nemotorné.

```
prediction <- predict(model, x = test_x)
prediction <- round(prediction)

diffs <- rowSums(abs(test_y - prediction))

test_accuracy <- length(diffs[diffs == 0])/length(diffs)
test_accuracy
```

```
## [1] 0.9675926
```

Výsledná accuracy sa pohybuje (v závislosti od náhodných faktorov) medzi 95 - 98%, čo je veľmi dobrý výsledok (neporovnateľný s naivným odhadom 13,9%).

Morphological analysis on FastText word embeddings

Nasleduje použitie uvedeného modelu na datasete slov a ich morfologických štítkov od kolegu Adama Bajgera. Potrebné sú súbory final_labels_for_R a word_vectors_for_R.

Aby sme mohli použiť moju neurónovú sieť na týchto dátach, je nutné upraviť vstupnú a výstupnú vrstvu. Výsledkom je pomerne bizarná kombinácia našich dvoch sietí.

```
nn <- keras_model_sequential()
nn %>%
  layer_dense(units = 20, activation = "sigmoid", input_shape = c(100)) %>%
  layer_dense(units = 10, activation = "sigmoid") %>%
  layer_dense(units = 12, activation = 'hard_sigmoid')

summary(nn)
```

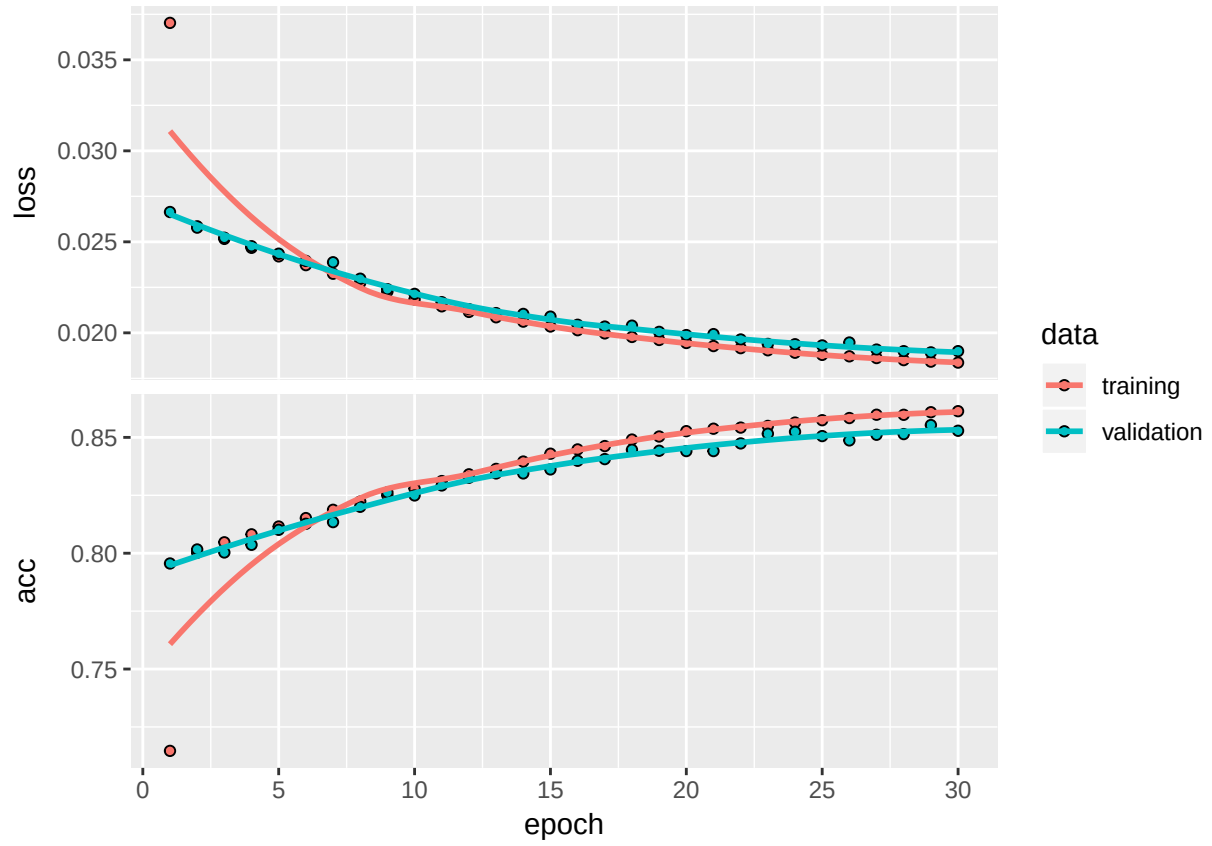
```
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## dense_3 (Dense)              (None, 20)                2020
## _________________________________________________________________________
## dense_4 (Dense)              (None, 10)                210
## _________________________________________________________________________
## dense_5 (Dense)              (None, 12)                132
## =========================================================================
## Total params: 2,362
## Trainable params: 2,362
## Non-trainable params: 0
## _________________________________________________________________________
```

```
nn %>%
  compile(
    loss = 'mse',
    metrics = 'accuracy',
    optimizer = 'rmsprop'
  )
```

Sieť natrénujeme.

```
f_history <- nn %>% fit(
  train_word_vectors, train_word_labels,
  epochs = 30, batch_size = 10, validation_split = 0.1
)

plot(f_history)
```



Vyhodnotíme accuracy.

```
nn %>% evaluate(test_word_vectors, test_word_labels)
```

```
## $loss
## [1] 0.2760796
##
## $acc
## [1] 0.2140204
```

Výsledná accuracy je sa pohybuje medzi 82 - 85%. To je prekvapivo dobrý výsledok, vzhľadom k vyššej rozmernosti druhého datasetu a k tomu, že sieť je približne 10x menšia, než pôvodná sieť pre druhý dataset.