Neurale netværk: Opbygning og estimation

Peter N. Bakker

08-06-2023

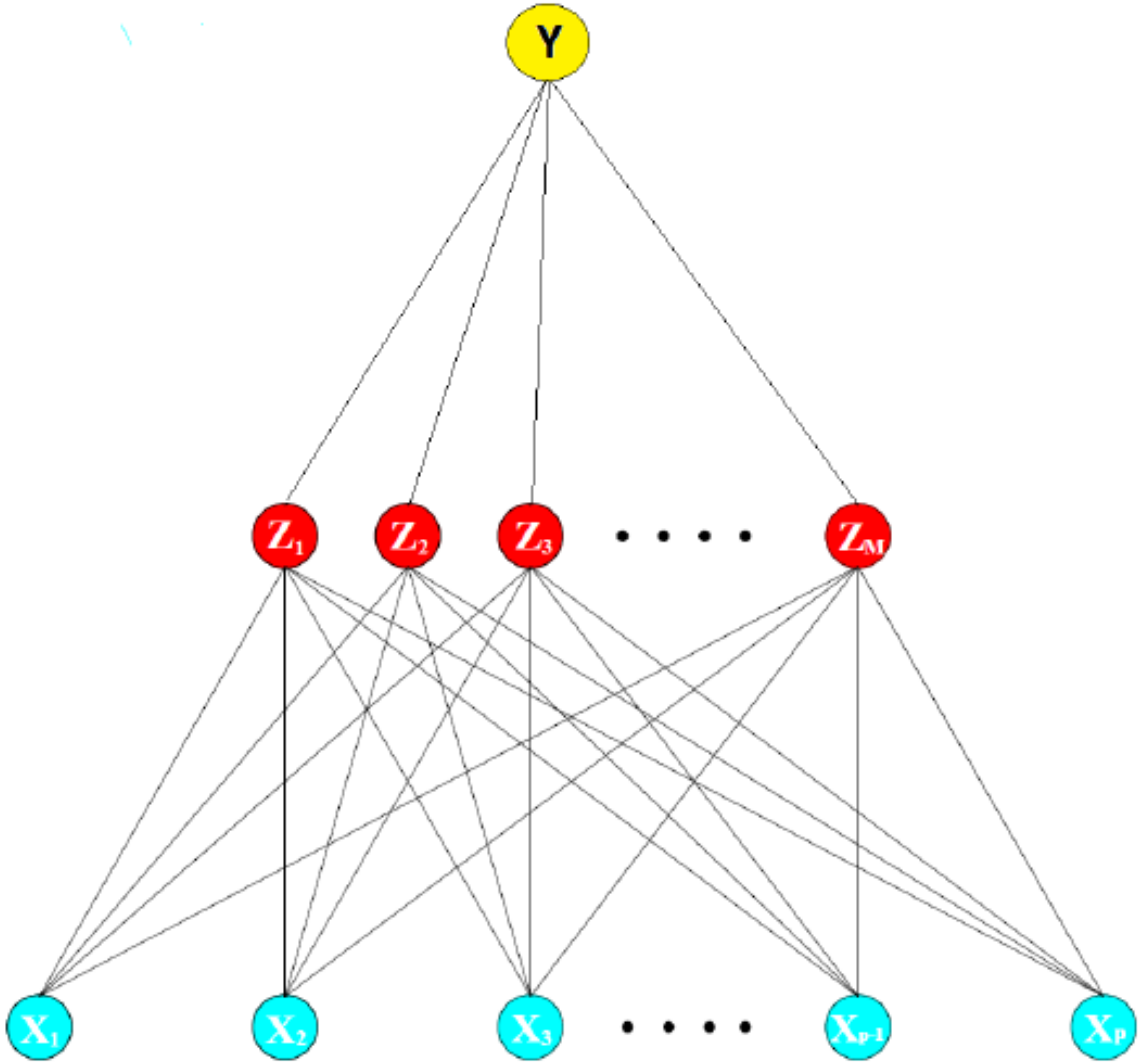
Introduktion til neurale netværk

- Definition: Et neuralt netværk er en sammensætning af forbundne neuroner, der kan modellere komplekse relationer mellem input og output. Ikke-lineær model (generelt).
- Neurale netværk består af lag af neuroner (som en hjerne), der er forbundet med vægte og bias.
- Et typisk neuralt netværk har inputlag, skjulte lag og outputlag.

Opbygning af neurale netværk

- Et neuralt netværk består af flere lag, der behandler input for at generere output.
- Lagstruktur:
  - Inputlag: Modtager og videregiver inputdata, X
  - Skjulte lag: Beregner komplekse funktioner mellem input og output. M "skjulte" Z-variable (hidden units) i hvert hidden layer D,
    $Z_m^1 = \sigma_1(\alpha_{10m} + \alpha_{1m}^T X) \rightarrow Z_m^D = \sigma_D(\alpha_{D0m} + \alpha_{Dm}^T Z^{D-1})$
  - Outputlag: Genererer den endelige output af netværket,
    $f(X) = \beta_0 + \beta^T Z^D$
- Forbindelser mellem lagene:
  - Vægte: Parametre, der styrer styrken af forbindelser mellem neuroner.
  - Bias: Konstante led, der justerer aktiveringerne i neuronerne.

Opbygning af neurale netværk (1 hidden layer)



Activation functions

- Aktiveringsfunktioner bestemmer outputtet fra en neuron baseret på dens input.
- Typer af aktiveringsfunktioner:
  - Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
  - ReLU (Rectified Linear Unit): $f(x) = \max(0, x)$
  - Softmax: $f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$
- Valg af aktiveringsfunktion afhænger af problemets karakteristika og ønsket output. Typisk mindre "bløde" funktioner ved "deep networks" (f.eks. ReLU).

Estimation i neurale netværk

- Estimation i neurale netværk indebærer at beregne output baseret på input og vægte.
- Forward propagation: $z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$, $a^{(l)} = f(z^{(l)})$
- Loss-funktioner: Evaluerer, hvor godt netværket præsterer ved at sammenligne det forudsagte output med det rigtige output.
- Eksempler på loss-funktioner: Mean Squared Error (MSE), Cross-Entropy.
- Backpropagation: Justerer vægte ved at minimere loss-funktionen gennem gradient descent.

Gradient Descent og Backpropagation

- Gradient Descent: En optimeringsalgoritme til at justere vægtene i neurale netværk.
- Backpropagation: En algoritme til at beregne gradienten af loss-funktionen med hensyn til vægtene ved hjælp af kædereglen (partielle afledede).
- Steps i Backpropagation:
  1. Beregn gradienten af loss-funktionen med hensyn til outputlaget.
  2. Propagér gradienten baglæns gennem netværket ved at anvende kædereglen og opdatere vægtene.

$$R(\theta) = \sum_{i=1}^{N} (y_i - f(x_i))^2$$

$$\beta_m^{(r+1)} = \beta_m^{(r)} - \gamma_r \frac{\partial R}{\partial \beta_m^{(r)}}$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} \gamma_R \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}}$$

Regularisering og Dropout

- Backpropagation kan have mange lokale minima. Og globalt minimere kan risikere overfit. Derfor ønskes regularisering.
- Overfitting opstår, når et netværk præsterer godt på træningsdataene, men generaliserer dårligt til nye data.
- Regularisering: Tilføjer en straf til loss-funktionen for at favorisere mere simple og generaliserbare modeller (f.eks. L1 og L2 regularisering).
- Dropout: Deaktiverer tilfældigt en del af neuronerne under træning for at lære mere robuste repræsentationer.
- Inputdata bør normaliseres, så de tilhørende parametre initialiseres på samme niveau.

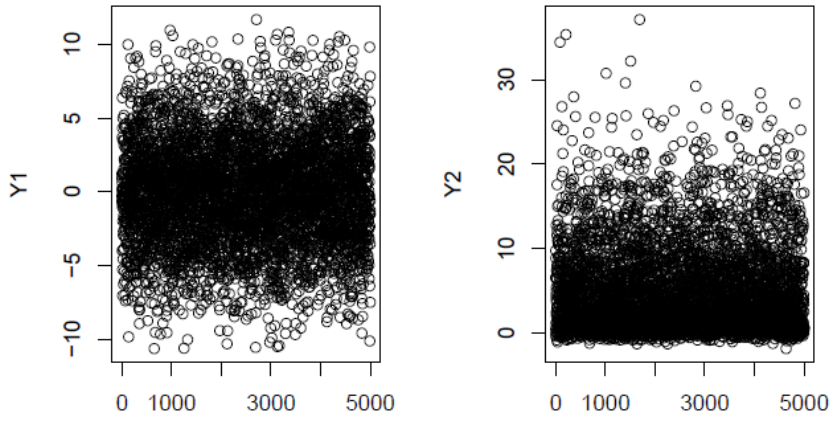
Træning og validering af neurale netværk

- Dataopdeling: Deling af data i træningssæt, valideringssæt og testsæt.
- Træningsproces:
  - Justering af vægte ved gentagen kørsel af forward propagation og backpropagation.
  - Optimering af netværket ved at finde de bedste værdier for vægte og bias.
  - Anvendelse af regulariseringsteknikker som L1, L2 regularisering og dropout. **RMSProp** kan benyttes til læringsrate.
- Validering og overvågning:
  - Evaluer netværkets præstation på valideringssættet under træningen.
  - Overvågning af loss og nøjagtighed for at undgå over- eller underfitting.
- Køres oftest i batches og epochs, hvilket er mindre beregningstungt (og kan minimere overfit).

Dataeksempel: NN til regression (syntetisk datasæt)

Vi kigger nu på syntetisk data til regression, hvor vi kører følgende kodestump for at generere to forskellige datasæt.

```
N <- 5000

X11 <- rnorm(N)
X12 <- rnorm(N)
epsilon1 <- rnorm(N,mean=0,sd=0.2)
Y1 <- 2*X11-3*X12+epsilon1
data1 <- data.frame(Y1,X11,X12)

### opdeling i input og output ###
xData1 <- model.matrix(Y1~.,data1)[,-1]
yData1 <- data1$Y1

X21 <- rnorm(N)
X22 <- rnorm(N)
epsilon2 <- rnorm(N,mean=0,sd=0.5)
Y2 <- (abs(2*X21+X22+2))^1.5+log(abs(X22)+1)*3*sqrt(X21*(X21>0.5))+epsilon2
data2 <- data.frame(Y2,X21,X22)

### opdeling i input og output ###
xData2 <- model.matrix(Y2~.,data2)[,-1]
yData2 <- data2$Y2

par(mfrow=c(1,2))
plot(Y1)
plot(Y2)
```



Dataeksempel: NN til regression (syntetisk datasæt)

- *Data*2 vil have mere gavn af at tilpasse et NN end standard OLS pga. ikke lineær DGP for $Y2$.
- For hvert datasæt fitter jeg et "deep NN" med 4 hidden layers. Jeg benytter dropout og L_2-regularisering. Modellen har i alt 20.451 parametere. 80/20 train-test split med 20% fra train som validering. Der er 500 epochs og batch size 64.
- Jeg sammenligner NN-modellerne med standard OLS og hhv. *eps*1 og *eps*2 for hvert datasæt.

```
##    MSE.NN1  MSE.NN2       eps1      eps2   MSE.lin1 MSE.lin2
## 1 60.66687 0.322563 0.03861044 0.2432734 0.03859981 8.206142
```

- Ikke overaskende performer NN bedre ved ikke lineære data end OLS og omvendt.

Konklusion

- Drøftelse af styrker og svagheder ved neurale netværk:
  - Styrker: Evne til at modellere komplekse sammenhænge, høj fleksibilitet og tilpasningsevne.
  - Svagheder: Behov for store mængder data og computerkraft, risiko for overfitting. Kan ikke give en "fortolkning" af parametre.