Neurale Netværk: Rekurrente og Convolutional Neurale Netværk

Peter N. Bakker

08-06-2023

Rekurrente Neurale Netværk (RNN)

- RNN er en type af neurale netværk, der er designet til at håndtere sekvensdata og problemer, hvor tidligere information er vigtig.
- Struktur af RNN:
  - Rekurrente forbindelser mellem neuroner, der tillader feedback af information gennem tid.
  - Behandler input og internt gemt tilstand ved hvert tidsskridt.
- Anvendelser af RNN:
  - Sprogmodellering, maskinoversættelse, talegenkendelse og tidsrækkeanalyse. Tidsrækker for nu :)
    Vi står i tidspunkt t_0 og har al information frem til t_0: $U_1, \ldots, U_{t_0}$. Vi ønsker at prædiktere næste periode U_{t_0+1}. I praksis vil man bruge den seneste information og dermed bruge $U_{t_0-T+1}, \ldots, U_{t_0}$, hvor T er passende stort til at estimere. Som et slags "rolling window" estimation.

Opbygning af RNN - 1 hidden layer

- Opbygning af et RNN baseret på den angivne notation:
  - For hvert tidspunkt $t = 1, ..., T$ defineres de M hidden units $Z_t = (Z_t^1, ..., Z_t^M)$ ved:

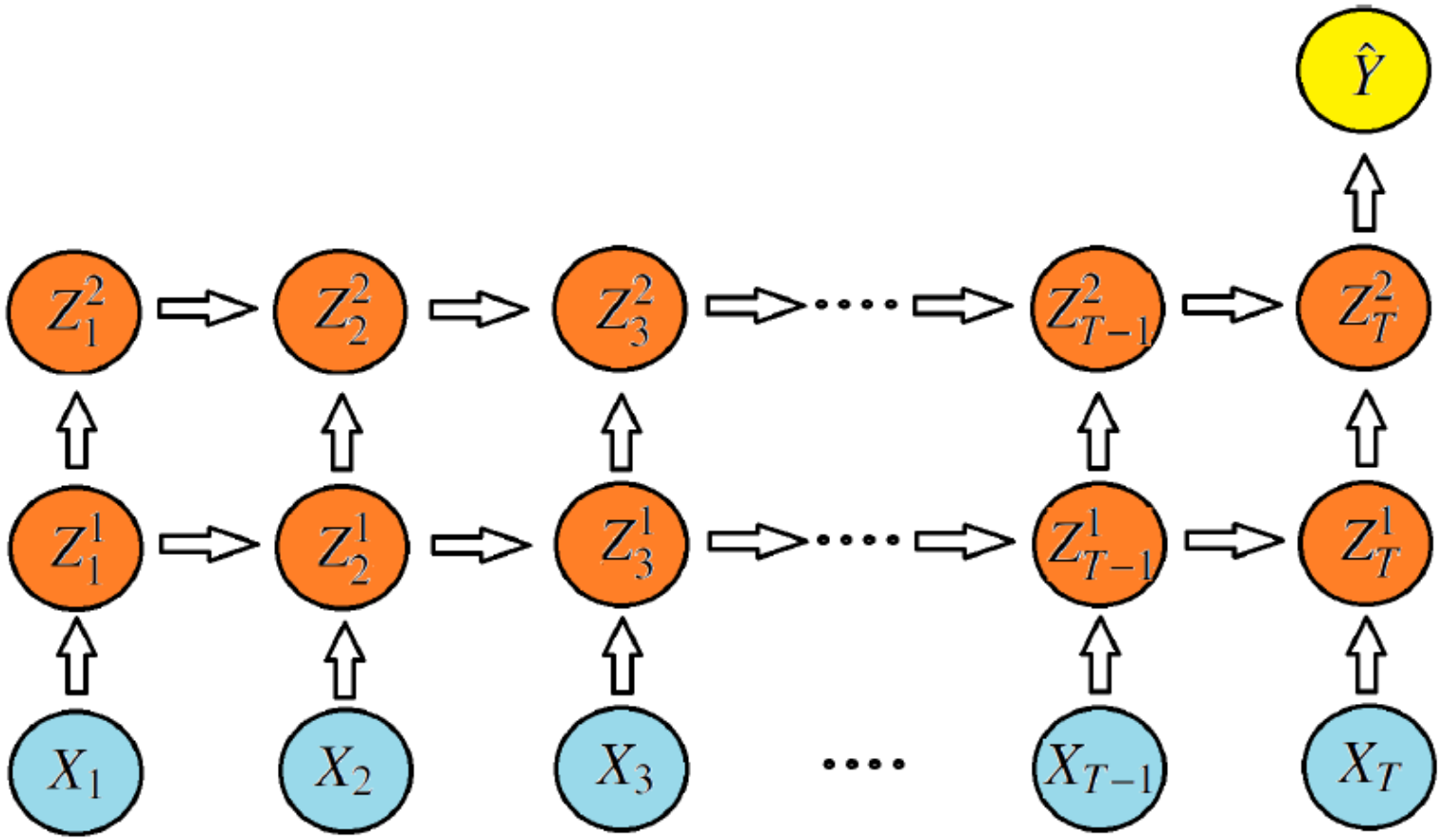
$$Z_t = \sigma(\alpha_0 + A_{xz} X_t + A_{zz} Z_{t-1})$$

  - α_0: En p-dimensional vektor.
  - A_{xz}: En $M \times p$-dimensional matrix. Ens på alle tidspunkter
  - A_{zz}: En $M \times M$-dimensional matrix. Ens på alle tidspunkter
  - σ: En aktiveringsfunktion (f.eks. sigmoid eller tanh).
  - Output $\hat{Y}$ udregnes som:

$$\hat{Y} = \beta_0 + \beta^T Z_T$$

  - Her er $f(X_1, ..., X_T) = \hat{Y}$ den ønskede funktion.

Illustration af RNN - 2 hidden layers



Estimering og problemer med RNN

- Estimering i RNN: Bruger backpropagation gennem tid (BPTT) til at estimere gradienten og opdatere vægtene.
- Problemer med BPTT:
  - Gradienteksplodering: Gradientværdierne bliver ekstremt store, hvilket kan føre til ustabilitet i træningen.
  - Gradientforsvindelse: Gradientværdierne bliver ekstremt små og svækkes, hvilket gør det vanskeligt for RNN at lære på langsigtede afhængigheder.
  - Simpelt RNN kan altså have god "short term" memory, men dårlig "long term". Jo større T, desto mere klart bliver det her problem.
- Behov for en løsning, der adresserer disse problemer og muliggør træning af RNN på længere sekvenser.

Long Short-Term Memory (LSTM)

- LSTM er en speciel type RNN-arkitektur, der adresserer problemerne med gradienteksplodering og gradientforsvindelse.
- Struktur af LSTM:
  - Hukommelsesceller C_t^k (memory cells, ekstra sæt af variable sammen med Z_t^k), der kan gemme information over tid og styre informationsflowet.
  - Gennemfører beregninger ved hjælp af gates, der regulerer informationsflowet i netværket.
- Løsning af problemerne:
  - Forget gate: Regulerer informationsflowet og styrer, hvilke oplysninger der skal glemmes.
  - Input gate: Regulerer informationsflowet og bestemmer, hvilke nye oplysninger der skal tilføjes til hukommelsescellen.
  - Output gate: Regulerer informationsflowet og bestemmer, hvilke oplysninger der skal sendes som output fra hukommelsescellen ($C_t^k \rightarrow Z_t^k$).
  - New C gate: Bestemmer de nye kandidatværdier til hukommelsescellen.

LSTM Gater og notation

- Opskriften på opdateringer:
  - Input gate (I):

$$I = \sigma(\alpha_I \cdot [h_{t-1}, x_t] + A_k^I \cdot [Z_{k-1,t}, Z_{k,t-1}])$$

  - Forget gate (F):

$$F = \sigma(\alpha_F + A_k^F \cdot [Z_{k-1,t}, Z_{k,t-1}])$$

  - Output gate (O):

$$O = \sigma(\alpha_O + A_k^O \cdot [Z_{k-1,t}, Z_{k,t-1}])$$

  - New C gate (C):

$$C = \tanh(\alpha_C + A_k^C \cdot [Z_{k-1,t}, Z_{k,t-1}])$$

- Ovenstående notation:
  - α_I, α_F, α_O, α_C: M-dimensionelle vektorer.
  - A_k^I, A_k^F, A_k^O, A_k^C: M$\times$(p+M)-dimensionale matricer.
  - σ: Sigmoid-funktionen.

Opdatering i LSTM

- Bemærkninger om notationen:
  - I, F og O har indgange i intervallet $(0, 1)$.
  - C har indgange i intervallet $(-1, 1)$.
- Opdatering i LSTM:

$$C_k^t = F \odot C_k^{t-1} + I \odot C$$

$$Z_k^t = O \odot \tanh(C_k^t)$$

- $\odot$ repræsenterer elementvis multiplikation.
- Der sker hukommelsesbevaring fra C_k^{t-1} til C_k^t:
  - C_k^{t-1} indgår i C_k^t i en nedskaleret udgave (på grund af værdierne i F (vi glemmer noget)).
  - Ny information tilføjes i form af en nedskaleret udgave af C (igennem I).
  - I sidste ende er det Z_t vi bruger til at danne vores prædiktion.

$$\hat{Y} = \beta_0 + \beta^T Z_T$$

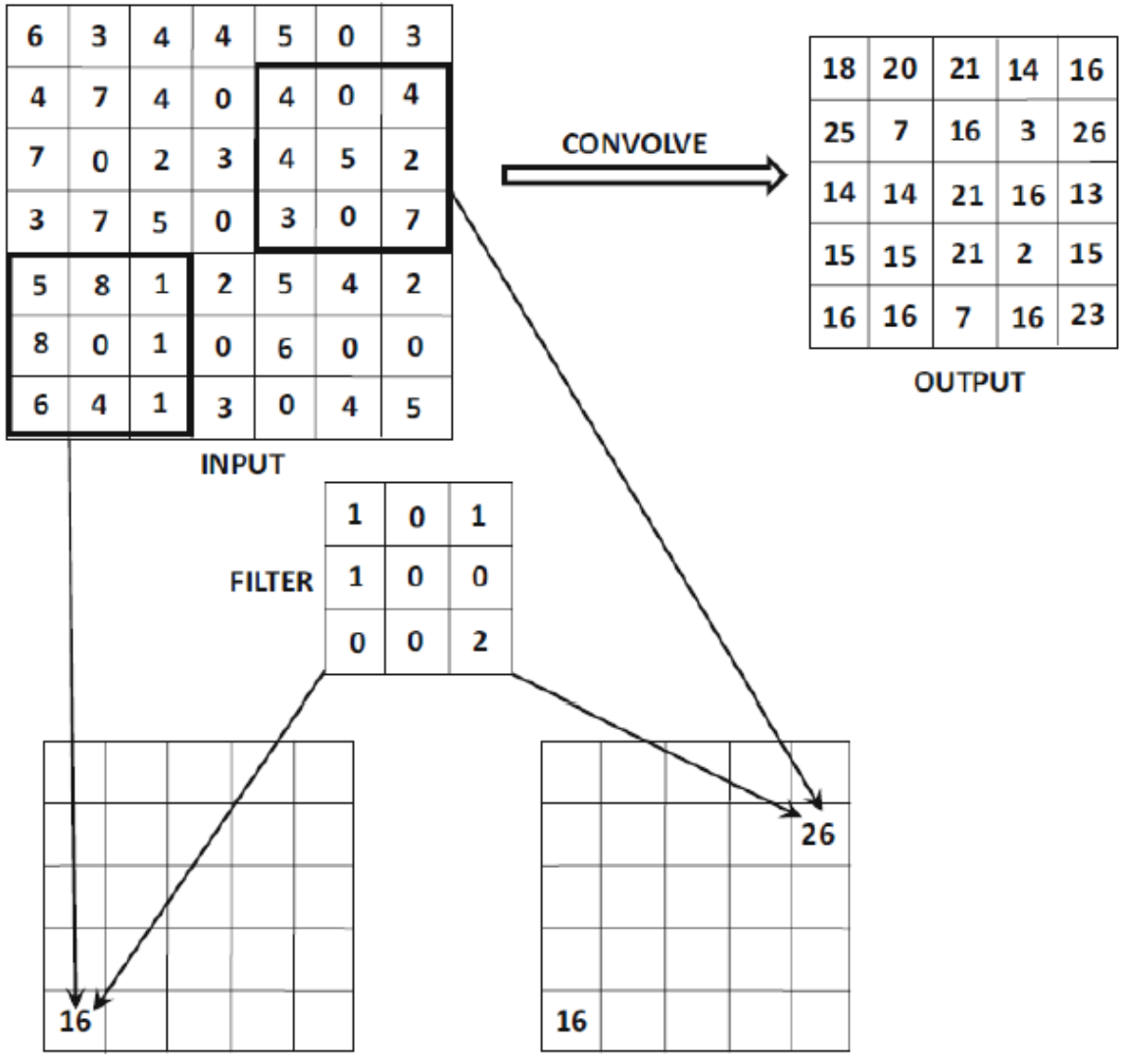
Convolutional Neurale Netværk (CNN)

- CNN er en type af neurale netværk, der er designet til at håndtere billed- og lyddata samt problemer med lokal struktur.
- Struktur af CNN:
  - Konvolutionelle lag, der udtrækker lokale features ved hjælp af filtre.
  - Pooling-lag, der reducerer dimensionerne og bevare de vigtigste informationer.
- Anvendelser af CNN:
  - Billedgenkendelse, objektdetektion, mønstergenkendelse og lyddataanalyse.

Convolution i CNN

- Convolution er en vigtig operation i CNN, der bruges til at udtrække lokale features fra inputbilledet.
- I et sort-hvidt billede baseret på en 28x28 matrix kan vi bruge en 3x3 filterkerne til convolution.
- Convolution udføres ved at placere filterkernen på forskellige positioner af inputbilledet og udføre elementvis multiplikation og summering.
- Outputet fra convolutionen kaldes et feature map, der indeholder information om forskellige features i inputbilledet. I det her eksempel benytter vi 16 filtre, som i alt skaber 16 26x26 matricer, som er vores feature map Z_1. Inden man går videre til max pooling vil man typisk benytte en ReLU activation function på feature mappet (Z_1 i det her tilfælde).

Convolution



Max Pooling i CNN

- Max pooling er en pooling-teknik, der bruges til at reducere dimensionerne af feature map og bevare de vigtigste informationer.
- I max pooling vælges den maksimale værdi fra et område af feature map og gemmes i det resulterende poolingslag.
- En typisk størrelse for max pooling er 2x2-matrix.
- Max pooling bidrager til at reducere varians i CNN og reducerer antallet af parametre i netværket.

Overgang til almindelig NN

- Man samler resultaterne for max pooling i en ny vektor kaldet Z_2. Dernæst flades Z_2 ud ien vektor af længde $13 \cdot 13 \cdot 16 = 2704$.
- Nu laves en 10-dimensional vektor Z_3 som omdannes til et "almindelig" NN for at kunne prædiktere: $Z_3 = \sigma(\alpha_3 + A_3 Z_2)$
- σ er f.eks. ReLU. α_3 er en 10-dimensional vektor, A_3 er en 10x2704 matrix.
- Til slut laver vi funktionerne $f_k(X)$ ved at bruge softmax-funktioner:

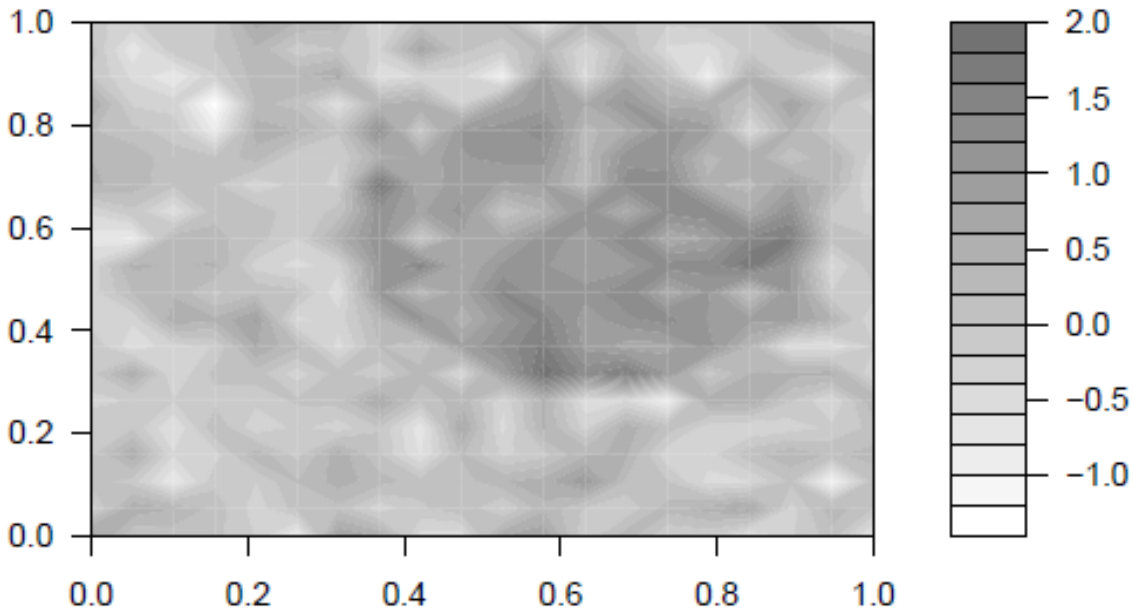
$$f_k(X) = g_k(\beta_{00} + \beta_0 Z_3, \beta_{10} + \beta_1 Z_3, \ldots, \beta_{90} + \beta_9 Z_3)$$

for $k = 0, \ldots, 9$, hvor altså

$$g_k(x_0, \ldots, x_9) = \frac{e^{x_k}}{\sum_{\ell=0}^{9} e^{x_\ell}}$$

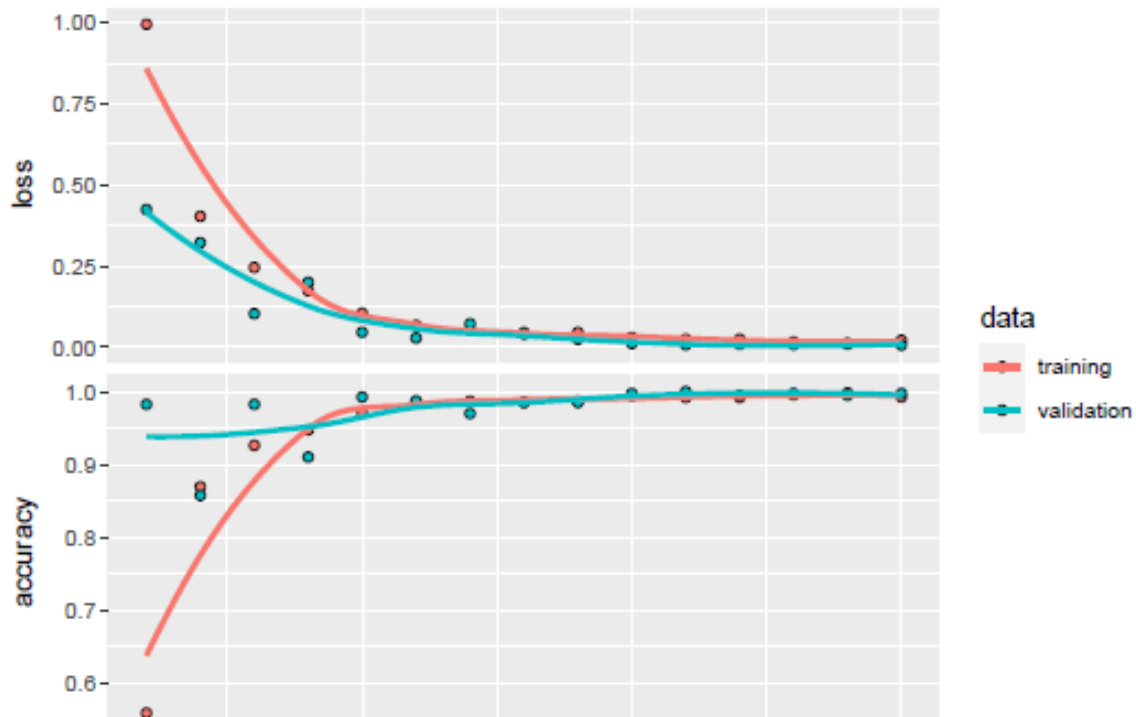
Dataeksempel CNN (billede klassifikation)

- Eksempel: Billede genkendelse af hhv. rektangel, cirkel og linjer. Hvert billede er sort hvid som 20x20 pixels. I alt 3000 billeder. Der er tilføjet "støj" til billederne.



Dataeksempel CNN (billede klassifikation)

- Jeg sætter 2000 billeder i train og 1000 i test og standardiserer billederne. Jeg fitter et CNN bestående af et convolutional lag med 32 filtre og 3x3 filtre (benytter ReLU). Et max pooling lag med 2x2 filtre. "Flader" det ud til en vektor med 64 units og benytter igen ReLU. Jeg benytter også dropout for at undgå overfit. Til sidst evalueres hver kategori med softmax. Jeg tilbageholder 20% af train til validation.
- I alt 166.467 parametre.



Dataeksempel CNN (billede klassifikation)

- Jeg får en fejlrate på sølle 0,4 pct. Hvor det kun er cirkler klasse 1 der fejl prædikteres.
- Hvis jeg øger støjen ved f.eks. at sætte "std" til 1 får jeg en fejlrate på 7,1 pct. Og gætter forkert på alle slags billedtyper.

Opsummering: Fordele og ulemper

- RNN (Rekurrente Neurale Netværk)
  - Fordele: Sekvensdata, langsigtede afhængigheder
  - Ulemper: Gradientproblemer, begrænset hukommelse
- LSTM (Long Short-Term Memory)
  - Fordele: Løser gradientproblemer, lærer langsigtede afhængigheder
  - Ulemper: Øget kompleksitet, kræver mere data og tid
- CNN (Convolutional Neurale Netværk)
  - Fordele: Billed- og lyddata, lokal struktur
  - Ulemper: Kræver mange data, kan ikke håndtere langsigtede afhængigheder (kan ikke det samme som RNN)