3g SRP - Kunstig intellegens

Jeppe Møldrup

Nørresundby gymnasium og HF

21/12-2018

Abstract

abstract

Indholdsfortegnelse

1 Redegørelse af et ANN netværk

1.1 ANN's baggrund

Et ANN(Artificial Neural Network) er en form for kunstig intellegens, hvor man prøver at efterligne den menneskelige hjerne ved at lave simple versioner af det netværk af de mange milliarder neuroner der findes i den menneskelige hjerne. Ligesom hvor en biologisk menneskehjerne har neuroner og synapser med forskellige styrker, så har et ANN også det, dog lidt simplere så man kan tillægge værdier til de forskellige neuroner og synapser.

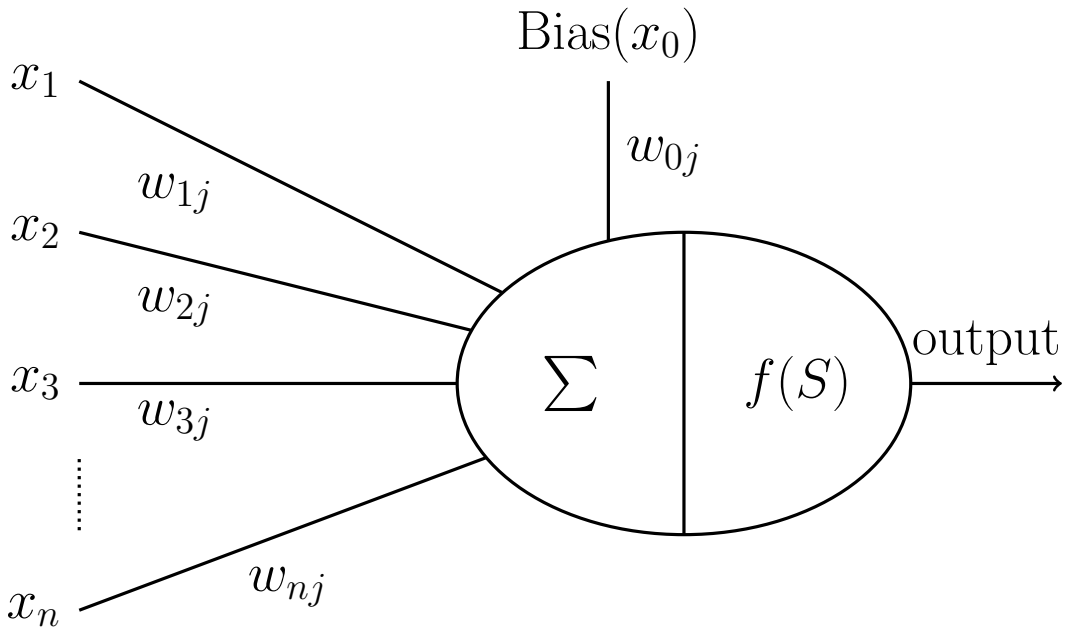
1.2 Strukturen bag neuronerne i et ANN

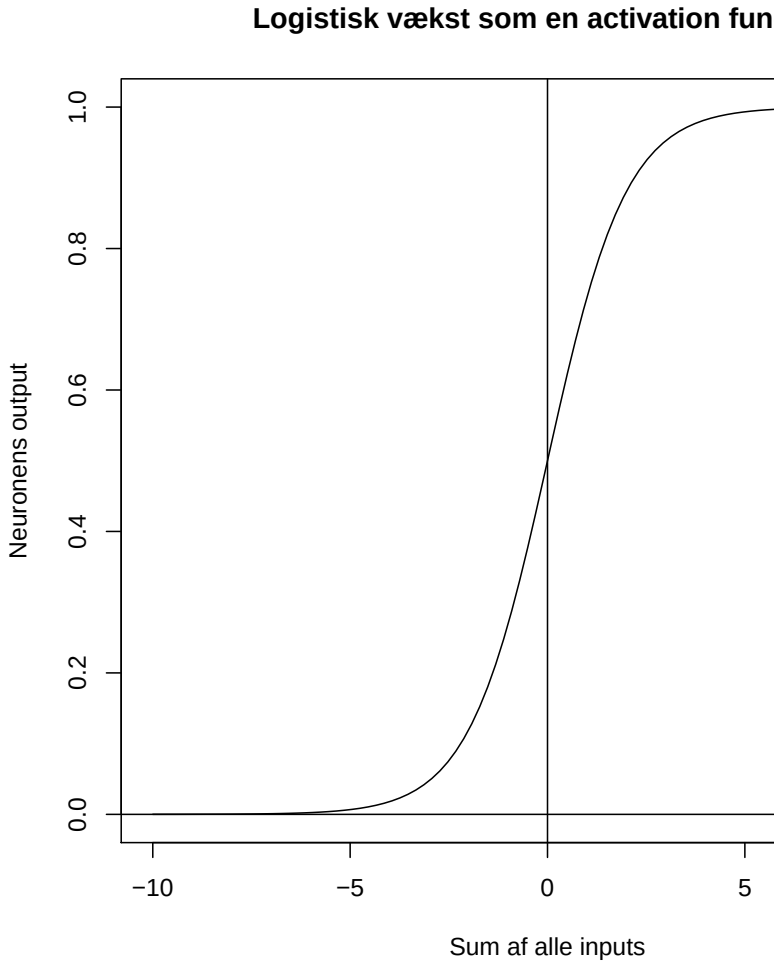
I den menneskellige hjerne er der som sagt milliarder af forskellige neuroner[1]. I et ANN laver man ikke ligeså mange neuroner som i en rigtig hjerne, da det er umuligt med den teknologi vi har i dag, men det er stadigvæk i fokus at lave et netværk med så mange neuroner som muligt. Derfor er det vigtigt at disse neuroner er matematisk set meget simple, så det er nemt for en moderne computer bare at blæse gennem matematikken i et ANN. Dermen er den strukter man har valgt bag én neuron i et givet ANN vist i figur 1.

Hvor alle x_n er outputs fra andre neuroner der er tilsluttet denne neuron via synapser, w_{nj} er såkaldte vægte der angiver hvor forstærket signalet er fra outputtet af sidste neuron og bias er en værdi der der enten vil øge eller sænke outputtet af en given neuron. Det der så sker inde i neuronen er to ting.
Først bliver alle inputs ganget sammen med deres vægte og summeret op.
Derefter bliver der ført en såkaldt "Activation function" der typisk bare normaliserer summationen mellem f.eks 0 og 1[2]. Her betragter jeg logistisk vækst med et maksimum på 1 som min activation function, hvor funktionen vil så tage et vilkårligt tal og så spytte et andet tal ud der ligger mellem 0 og 1, jeg

[1] Jørgen Lützen og Morten Møller 2018.
[2] Ali Kattan, Rosni Abdullah og Zong Woo Geem 2011.

Figure 1: Model af en neuron



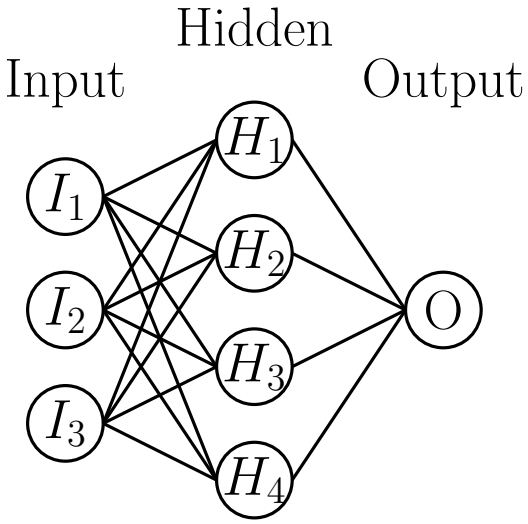


betegner funktionen med symbolet *sigma*

Et ANN består af en hel masse neuroner fra figur 1 der er sammensat som på figur 2

Figur 2 er et eksempel på et meget simpelt netværk der er kendt som et Feed Forward Artificial Neural Network, eller FFANN. Her bliver alle neuroner delt op i forskellige lag, i det her eksempel er der 3, men der kunne også være flere. Det første lag kaldes for "Input layer", det sidste kaldes for "Output layer" og alle imellem kaldes for "Hidden layers". Så er inputtet til hver neuron outputtet fra alle neuroner i sidste lag. Grunden til at alle neuroner i et givent lag så ikke har den samme værdi idet deres værdi afhænger af samme neuroner, er fordi de alle sammen har forskellige vægte.

Figure 2: Model af et neuralt netværk



Så f.eks. har første neuron 4 forskellige vægte, én til hver af neuronerne i næste lag[3]. Allerede i dette meget simple netværk er det ret rodet i forhold til de små tilslutninger mellem neuronerne. Så derfor er det smart at have en god systematisk navngivning til de forskellige elementer der opgør et neuralt netværk.
Denne opgave bruger følgende navngivning.
Neuroner(Hvor L er hvilket lag neuronen er i, ikke en eksponent): n_i^L
Vægte har i indekset 2 numre, et nummer for hvilken neuron de er fra, og et nummer for hvilken neuron de skal til, derudover også en indeks der viser hvilket lag de er fra w_{ij}^L, så det kunne f.eks. være w_{11}^1, hvor vægten er fra input laget, og den går fra første neuron til første neuron af hidden laget.
Biasser har samme navngivning som en neuron bare med et b i stedet.
Da alle neuroner bliver representeret af et tal, kan et lag skrives som en

[3]Ali Kattan, Rosni Abdullah og Zong Woo Geem 2011.

vektor, hvor hvert koordinat af vektoren er en neuron i laget, f.eks.

$$\vec{H} = \begin{pmatrix} n_1^2 \\ n_2^2 \\ n_3^2 \\ \cdot \\ \cdot \\ \cdot \\ n_n^2 \end{pmatrix}$$

Her er det hidden lag opskrevet som en vektor.

1.3 Matematikken bag synapser i et ANN og fejlfunktion

Ideen om at træne et ANN kommer fra at man gerne vil finde de helt rigtige værdier til de forskellige vægte i netværket der gør at man får et så præcist resultat som muligt. For at kunne vide hvordan man skal ændre på vægtene skal man vide hvor langt fra det korrekte svar man er, og man skal have nogle værdier for vægtene allerede. For at finde ud af hvor langt man er fra et korrekt svar bruger man en såkaldt fejlfunktion. Man har nogle datasæt med kendte inputs og outputs, hvor man så tester inputsne og ser hvordan netværkets outputs er i forhold til de korrekte outputs. Her bruger man så fejlfunktionen til at vurderer hvor langt fra det korrekte svar man er. Der er flere forskellige fejlfunktioner, men i denne opgave vil jeg bruge følgende fejlfunktion

$$f = \sum_{p=1}^{P} \sum_{i=1}^{S} (t_i^p - o_i^p)^2$$

hvor
f er en værdi der angiver fejl
P Antallet af datasæt til træning af netværket
S Antallet af output-neuroner
t Er "target" værdien for en neuron, dvs. den korrekte værdi
o Er værdien fra netværket[4]

[4]Ali Kattan, Rosni Abdullah og Zong Woo Geem 2011.

1.4 Backpropagation og Gradient descent

For at træne et netværk kigger man på outputtet og arbejder tilbage i netværket for at se hvad man kan gøre bedre for at få et bedre resultat. Det er hele ideen med Backpropagation og den underlæggende algoritme, Gradient descent. Fejlfunktionen er en vigtig del af puslespillet når det kommer til at træne netværket, da man prøver at minimere netværkets fejl, dvs. finde et minimum i fejlfunktionen hvor værdien der angiver fejl er så lille som mulig. Fejlfunktionen er en funktion der tager alle neuroner, vægte og biasser som input, og spytter et tal ud. Det er derfor ikke praktisk at prøve at finde fejlfunktionens monotoniforhold, i stedet finder man fejlfunktionens "gradient". Gradienten er en vektor der viser hvilken retning hældningen er højest i et givet punkt på en funktion. Gradienten for en flervariabelfunktion findes ved at kombinere alle partielle afledte fra funktionen. f.eks. funktionen $f(x, y)$ til have de to partielle afledte

$$\frac{\partial f}{\partial x} \quad \text{og} \quad \frac{\partial f}{\partial y}$$

For at kombinere dem til en vektor ganges de ind på deres enhedsvektorer($\vec{i}$ og $\vec{j}$) og så lægger man dem sammen.[5]

$$\nabla f(x, y) = \frac{\partial f}{\partial x}\vec{i} + \frac{\partial f}{\partial y}\vec{j} = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

For at minimere fejlfunktionen tager man et skridt i den negative retning af gradienten, dvs. den vej hvor hældningen er mindst. Hvis man bliver ved med at gøre det kommer man tættere og tættere på et minimum. Man kommer højst sandsyndeligt til at finde et lokalt minimum, da man ikke søger hele funktionen, men man bare tager skridt nedad i det lokale område man nu befinder sig i i funktionen. Dette kaldes gradient descent, fordi man stiger ned af gradienten.

Fejlfunktionen og dermed gradienten er defineret ud fra hele sættet af træningsdata. Det er ikke en smart måde at træne netværket ved at gå alt dataen igennem, da det bare er for intensivt for computeren. I stedet deler man træningsdata op i forskellige minisæt og træner netværket ud fra et minisæt af gangen. Det betyder at den gradient man finder ikke er den

[5] Wikipedia 2013.

helt præcise, da man ikke har udregnet den ud fra alt træningsdataet, men den peger nogenlunde i den rette retning. I modsætning til gradient descent, kaldes dette for stochastic gradient descent når man deler træningsdataet op inden man træner netværket.[6]

1.5 Udledning af træningsformler

Hvis vi betragter det neurale netværk i figur 2, ser vi at værdien af outputtet er givet ved

$$z_1^3 = (n_1^2 \cdot w_{11}^2 + n_2^2 \cdot w_{21}^2 + n_3^2 \cdot w_{31}^2 + n_4^2 \cdot w_{41}^2 + b_1^2)$$
$$n_1^3 = \sigma(z_1^3)$$

(Jeg deler det op i to, hvor z_1^3 er summen af alle neuroner ganget med deres vægte) Og fejlfunktionen(Hvor y er der korrekte svar)

$$f = (n_1^3 - y)^2$$

Hvis vi så f.eks. vil finde vægten w_{11}^2's indflydelse på fejlfunktionen, skal vi finde dens partielle afledte, dvs.

$$\frac{\partial f}{\partial w_{11}^2}$$

Jeg har 3 funktioner der afhænger af hinanden, dvs. en kombineret funktion, så jeg skal bruge kædereglen. Kædereglen siger at

$$(f(g(x)))' = f'(g(x)) \cdot g'(x) \Leftrightarrow \frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}$$

Jeg benytter kædereglen og finder

$$\frac{\partial f}{\partial w_{11}^2} = \frac{\partial f}{\partial n_1^3} \cdot \frac{\partial n_1^3}{\partial z_1^3} \cdot \frac{\partial z_1^3}{\partial w_{11}^2}$$

Referencer

Bøger

[2] Ali Kattan, Rosni Abdullah og Zong Woo Geem, *Artificial neural network training and software implementation techniques*. 2011.

[6] 3Blue1Brown 2017.

Artikler og andre online kilder

[1] 3Blue1Brown. (2017). What is backpropagation really doing? — Deep learning, chapter 3, side: https://www.youtube.com/watch?v=Ilg3gGewQ5U (sidst set 12. dec. 2018).

[3] Jørgen Lützen og Morten Møller. (2018). Hjerne, side: http://denstoredanske.dk/Krop,_psyke_og_sundhed/Sundhedsvidenskab/Sammenlignende_anatomi_og_fysiologi/hjerne (sidst set 8. dec. 2018).

[4] Wikipedia. (2013). Gradient, side: https://da.wikipedia.org/wiki/Gradient (sidst set 12. dec. 2018).