www.math-stockholm.se/cirkel

24 februari 2022

Välkomna!

Idag:

- Vad är maskininlärning?
- Vad är djupinlärning?
- Hur tränar man modellen?

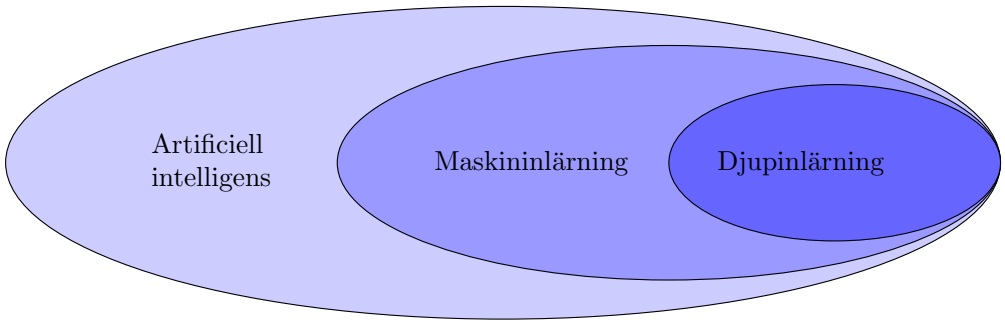
Figur: Artificiell intelligens

Vad är maskininlärning? (1)

- Med data kan man lära datorer att själva upptäcka och lära sig regler för att lösa en uppgift, utan att programmera datorerna med regler för just den specifika uppgiften i fråga.
- Detta kallas för maskininlärning.
- Maskininlärning är ett område inom artificiell intelligens och används inom
  - Robotik
  - Programvaruutveckling
  - Självkörande bilar
  - Medicinsk diagnostik
  - Strömningstjänster på internet osv.

Vad är maskininlärning? (2)

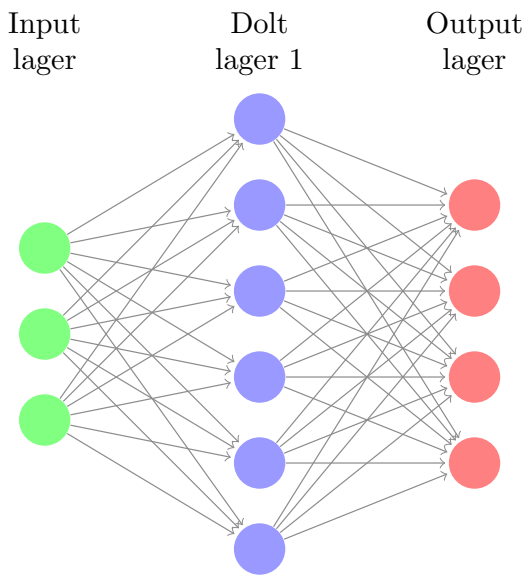
- Samhället blir mer och mer digitaliserat
- Vi har samlat mer och mer data
  - Svårt att hantera all data
  - Maskininlärning kan hjälpa oss att bättre analysera och förstå vår data
- Vi ska diskutera djupinlärning som är en del av området maskininlärning



Figur: Artificiell intelligens, maskininlärning och djupinlärning

Vad är djupinlärning? (1)

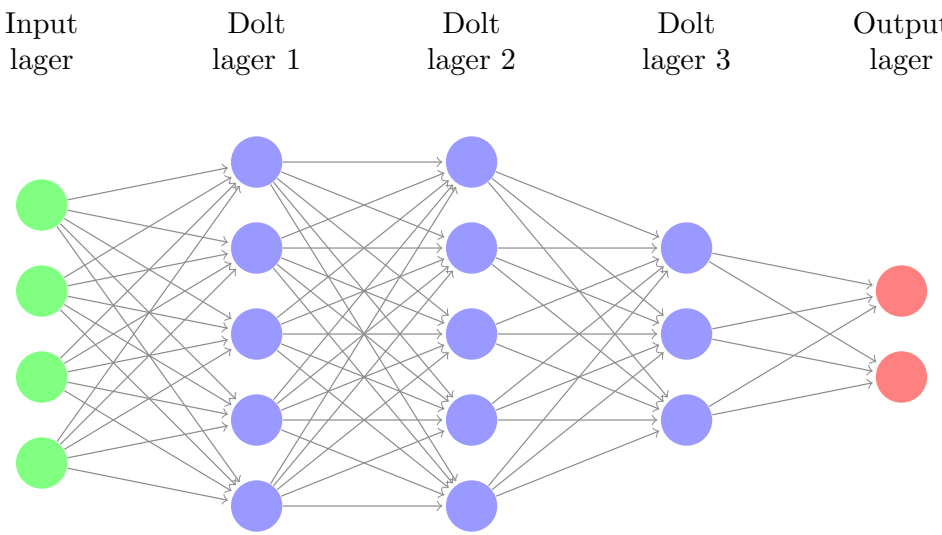
- Inspirerad av hjärnans struktur och skapar representationer i flera steg som kallas **lager**
- Ett djupt neuralt nätverk med ett **inputlager**, ett **dolt lager** och ett **outputlager**



Figur: Ett djupt neuralt nätverk med ett dolt lager

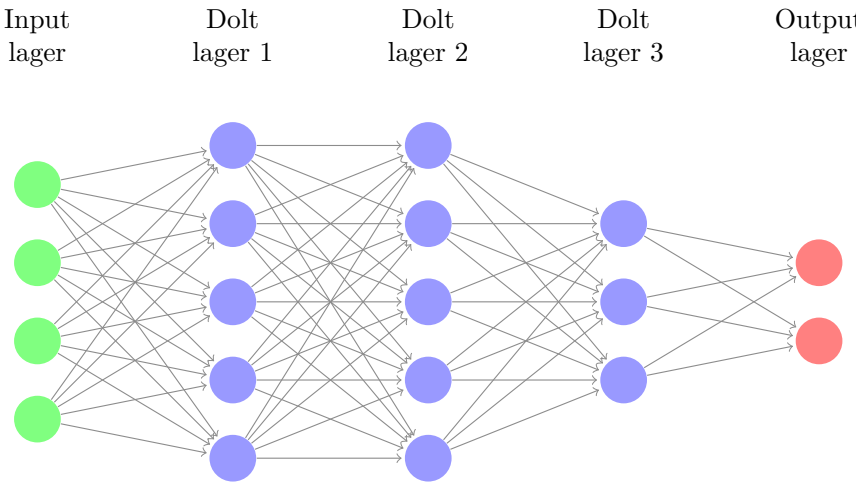
Vad är djupinlärning? (1)

- Inspirerad av hjärnans struktur och skapar representationer i flera steg som kallas **lager**
- Ett djupt neuralt nätverk med ett inputlager, tre dolda lager och ett outputlager



Figur: Ett djupt neuralt nätverk med tre dolda lager

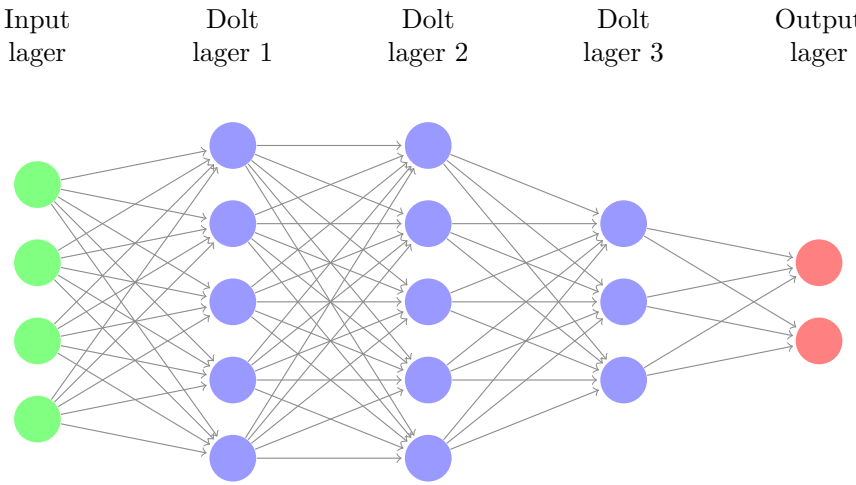
Noder



Figur: Ett djupt neuralt nätverk med tre dolda lager

- Var och en av cirklarna kallas en **nod**
- Noder representerar informationen som flyter in i nätverket
- Poängen är att skapa bättre representationer för varje lager

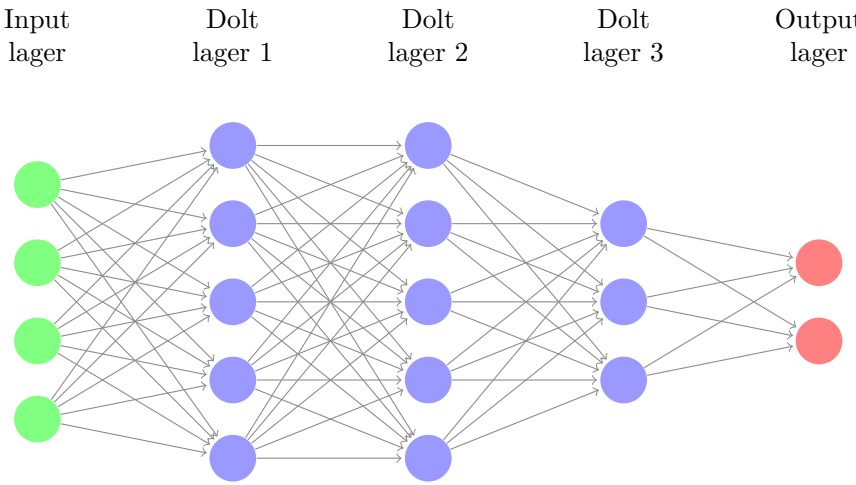
Noder



Figur: Ett djupt neuralt nätverk med tre dolda lager

- Djupa neurala nätverket förvandlar data som matas in (**input**) till andra representationer som blir mer informativa (**output**)

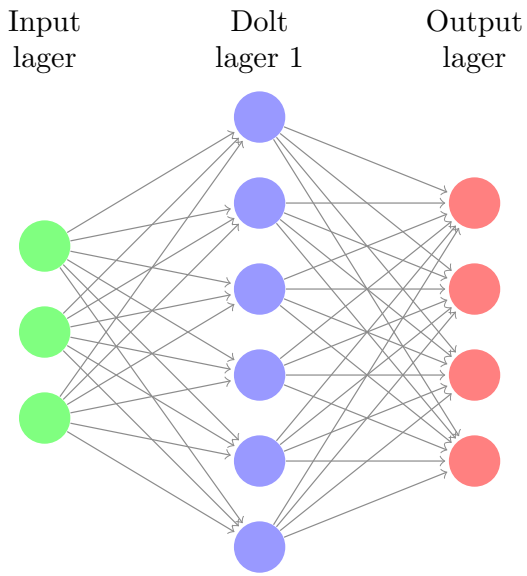
Noder



Figur: Ett djupt neuralt nätverk med tre dolda lager

- Generellt kan man säga att nätverket filtrerar informationen så att endast de **användbara** och **karaktäristiska dragen** är kvar i det sista lagret
- Man kan använda **flera hundra lager** med **hundratals noder** i varje lager om man har en dator som kan klara av det

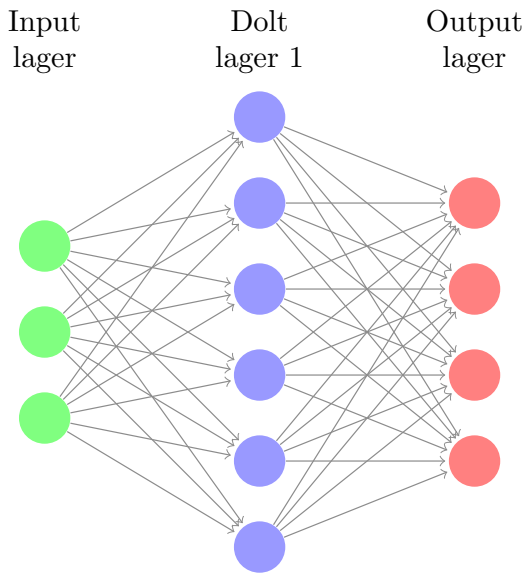
Ett djupt neuralt nätverk



Figur: Ett djupt neuralt nätverk med ett dolt lager

- Varje lager tar emot information från föregående lager i form av flera tal
- Därefter gör beräkningar med talen

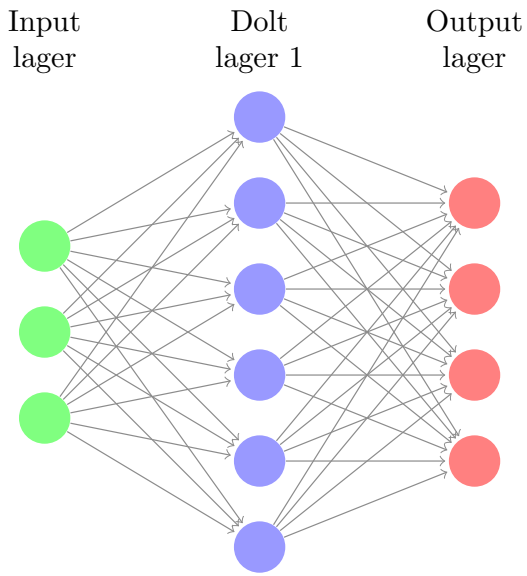
Ett djupt neuralt nätverk



Figur: Ett djupt neuralt nätverk med ett dolt lager

- Talen representerar någon typ av information
- De nya talen skickas vidare till nästa lager
- Vi säger att **ingångsnoderna** tar **ingångsvärden**

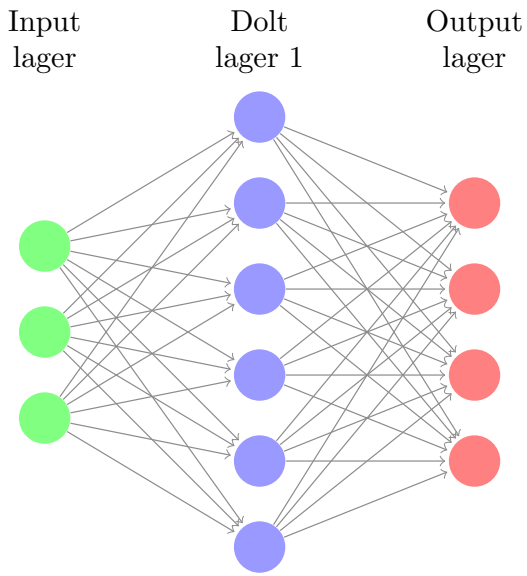
Ett djupt neuralt nätverk



Figur: Ett djupt neuralt nätverk med ett dolt lager

- Ingångsnoderna är de gröna noderna
- De kan t.ex. vara en binär 1 eller 0, en del av ett RGB-färgvärde m.m.

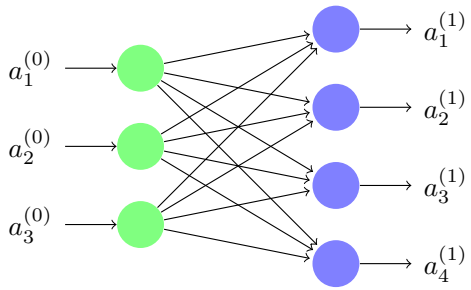
Ett djupt neuralt nätverk



Figur: Ett djupt neuralt nätverk med ett dolt lager

- Talen som kommer ut från modellen heter **utgångsnoderna**
- Utgångsnoderna är de röda noderna

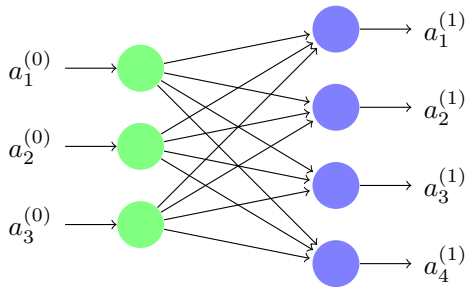
Att beräkna det andra lagret



Figur: Beräkna andra lagret

- Vi ska gå igenom ett exempel där vi beräknar en nod i det andra lagret
- Noterar att det finns andra sätt att arrangera modellen än som vi har
- Poängen är att parametrarna är kopplade till varandra, som kan ses i en mängd olika möjliga modeller

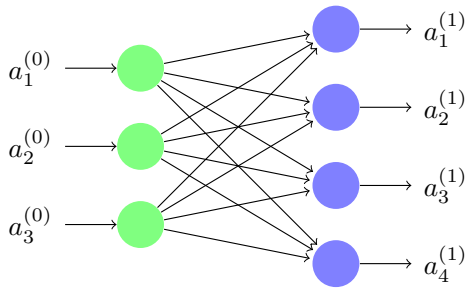
Att beräkna det andra lagret



Figur: Beräkna andra lagret

- Har valt ett enkelt sätt att börja med
- Ska titta på en mer komplicerad modell senare
- Notera: varje nod (cirkel) håller ett tal
- Vi ska beräkna nod $a_1^{(1)}$

Att beräkna det andra lagret



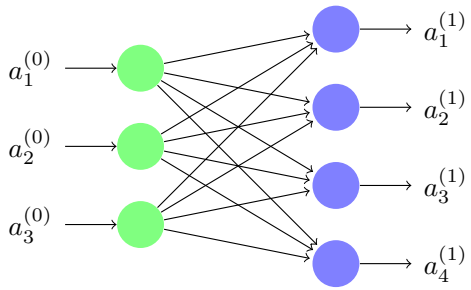
Figur: Beräkna andra lagret

- Noden $a_1^{(1)}$ är första noden (subscript) i det första dolda lagret (superscript)
- Noder i inputlagret har superscript noll, d.v.s.

$$a_j^{(0)},\ j = 1, \ldots, n_0$$

där $n_0 = 3$ är det totala antalet noder i inputlagret

Att beräkna det andra lagret



Figur: Beräkna andra lagret

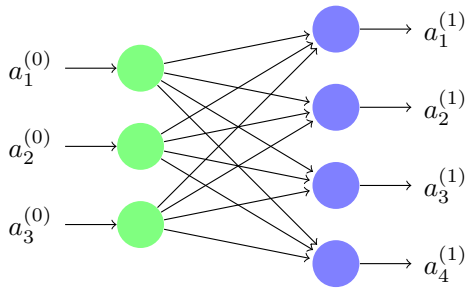
- Vi beräknar $a_1^{(1)}$ som

$$a_1^{(1)} = \sigma(w_{1,1}^{(0)} a_1^{(0)} + w_{1,2}^{(0)} a_2^{(0)} + w_{1,3}^{(0)} a_3^{(0)} + b_1^{(0)})$$

med **vikten** $w_{1,j}^{(0)}$, $j = 1, 2, 3$, **bias** $b_1^{(0)}$ och där

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Att beräkna det andra lagret



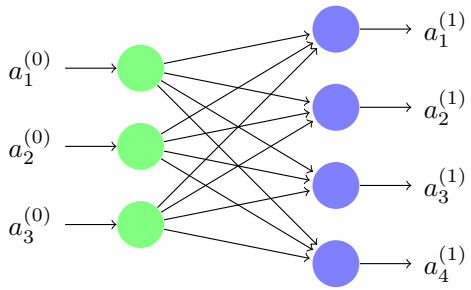
Figur: Beräkna andra lagret

- I allmänhet har vi $w_{i,j}^{(k)}$, $a_j^{(k)}$ och $b_i^{(k)}$, där

$$i = 1, \ldots, n_{k+1}, \quad j = 1, \ldots, n_k \text{ och } k = 0, 1, \ldots, \ell, \ell + 1$$

- k representerar vilket lager vi började på
- ℓ är det totala antalet dolda lager i det djupa neurala nätverket
- n_k är antalet noder i det k:te lagret av nätverket

Att beräkna det andra lagret



Figur: Beräkna andra lagret

- Notera: genom att **träna modellen** får vi $w_{i,j}^{(k)}$ och $b_i^{(k)}$, där

$$i = 1, \ldots, n_{k+1}, \quad j = 1, \ldots, n_k \text{ och } k = 0, 1, \ldots, \ell, \ell + 1$$

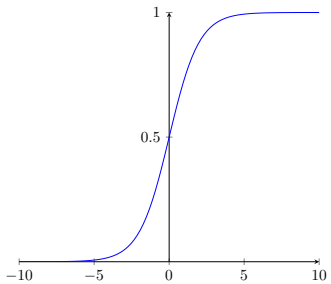
  som ger det bästa resultatet
- Mer om detta lite senare

Aktiveringsfunktion (1)

- Vår **aktiveringsfunktion** σ är definieras som

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Syftet med aktiveringsfunktionen är att omvandla slutresultatet till ett tal som är lättare att hantera
- En sigmoidal aktiveringsfunktionen omvandlar alla inkommande värden till ett tal mellan 0 och 1
- Tar stora positiva tal till ett tal nära 1 och stora negativa tal till ett tal nära 0

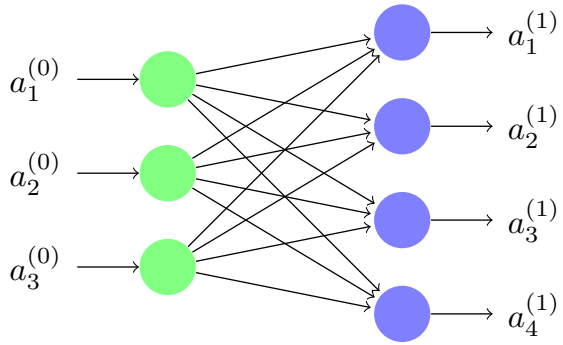


Figur: Aktiveringsfunktion σ, s.k. sigmoidal

Mer om noder, vikter och bias

- Vikter är tal som är koefficienter och bias är tal
- Med andra ord multipliceras noderna med vikter och därefter summeras alla input och ett bias adderas där var och en av linjerna representerar en vikt
- Bias är, precis som vikterna, ytterligare en parameter som vi beräknar
- Genom att välja bias kan man öka eller minska en nods värde
- Vikter och bias är de parametrar vi justerar när vi **tränar** modellen
- Vi tar ett annat exempel . . .

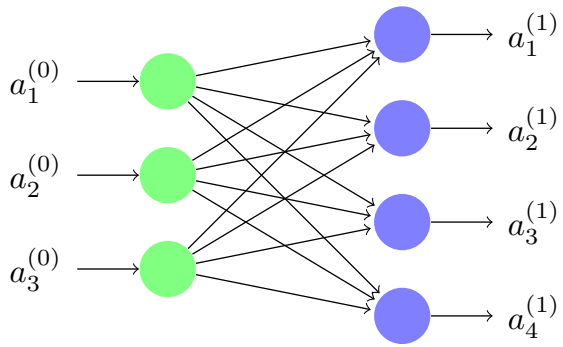
Mer om noder, vikter och bias



Figur: Beräkna andra lagret

- Nu beräknar vi alla noder i det första dolda lagret
- Vi skriver beräkningar med en matris och vektorer som gör det lättare att organiserar alla vikter och bias
- Det blir också lättare att analysera hur många parametrar vi måste träna i ett visst problem om vi skriver så här

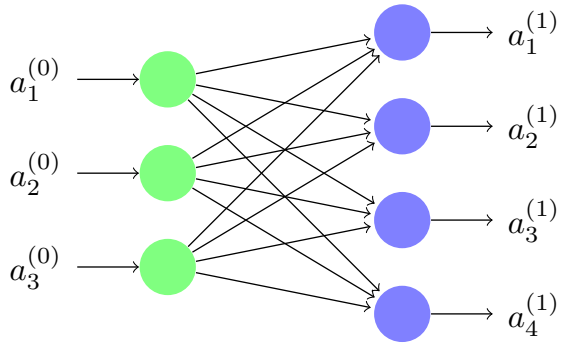
Mer om noder, vikter och bias



Figur: Beräkna andra lagret

$$\begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \\ a_4^{(1)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{1,1}^{(0)} & w_{1,2}^{(0)} & w_{1,3}^{(0)} \\ w_{2,1}^{(0)} & w_{2,2}^{(0)} & w_{2,3}^{(0)} \\ w_{3,1}^{(0)} & w_{3,2}^{(0)} & w_{3,3}^{(0)} \\ w_{4,1}^{(0)} & w_{4,2}^{(0)} & w_{4,3}^{(0)} \end{bmatrix} \begin{bmatrix} a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \end{bmatrix} + \begin{bmatrix} b_1^{(0)} \\ b_2^{(0)} \\ b_3^{(0)} \\ b_4^{(0)} \end{bmatrix} \right)$$

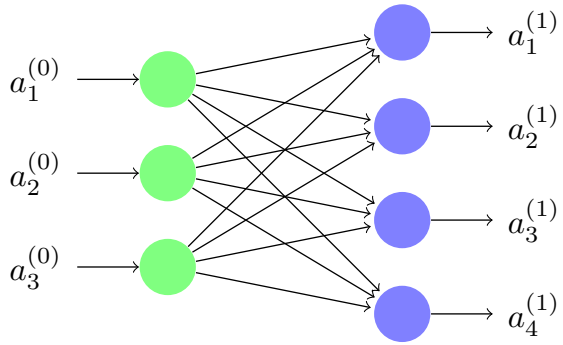
Mer om noder, vikter och bias



Figur: Beräkna andra lagret

$$\Longleftrightarrow \quad \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \\ a_4^{(1)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{1,1}^{(0)} a_1^{(0)} + w_{1,2}^{(0)} a_2^{(0)} + w_{1,3}^{(0)} a_3^{(0)} + b_1^{(0)} \\ w_{2,1}^{(0)} a_1^{(0)} + w_{2,2}^{(0)} a_2^{(0)} + w_{2,3}^{(0)} a_3^{(0)} + b_2^{(0)} \\ w_{3,1}^{(0)} a_1^{(0)} + w_{3,2}^{(0)} a_2^{(0)} + w_{3,3}^{(0)} a_3^{(0)} + b_3^{(0)} \\ w_{4,1}^{(0)} a_1^{(0)} + w_{4,2}^{(0)} a_2^{(0)} + w_{4,3}^{(0)} a_3^{(0)} + b_4^{(0)} \end{bmatrix} \right)$$

Mer om noder, vikter och bias



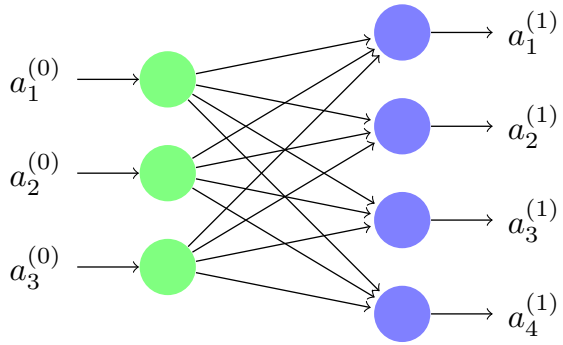
Figur: Beräkna andra lagret

- Multiplikation av två matriser A och B är möjlig då A är av typ $m \times n$ och B är av typ $n \times p$
- Då är

$$A \cdot B = C = (c_{ij})_{m \times p}$$

där $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$, $i = 1, 2, \ldots, m$ och $j = 1, 2, \ldots, p$

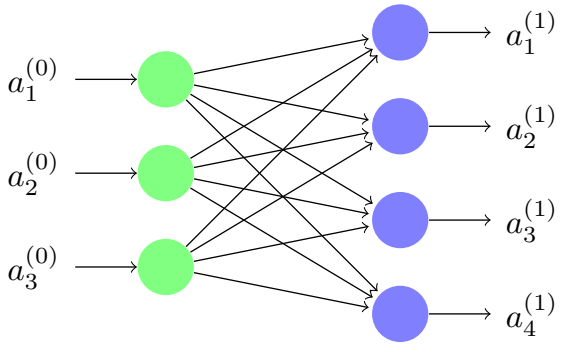
Mer om noder, vikter och bias



Figur: Beräkna andra lagret

- Addition av två matriser A och B är möjlig då A och B är av samma dimension, och beräknas genom att addera elementen parvis

Mer om noder, vikter och bias

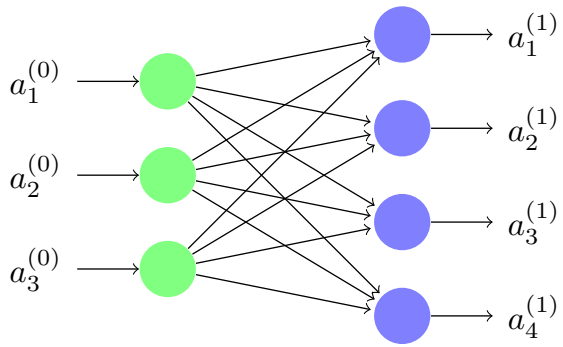


Figur: Beräkna andra lagret

- Vi evaluerar aktiveringsfunktionen elementvis, d.v.s.

$$\sigma\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\right) = \begin{bmatrix} \sigma(x_1) \\ \sigma(x_2) \\ \vdots \\ \sigma(x_n) \end{bmatrix}$$

Mer om noder, vikter och bias



Figur: Beräkna andra lagret

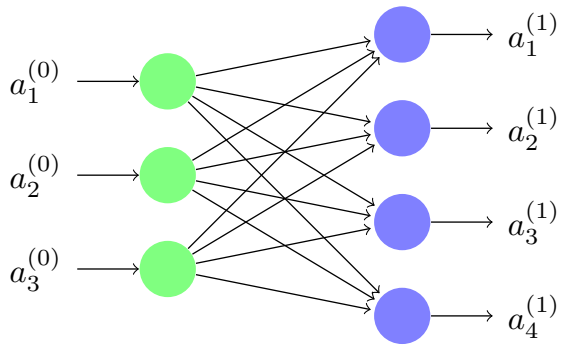
- För att beräkna alla noder i det första dola lagret, d.v.s.

$$a_i^{(1)}, \ i = 1, \ldots, n_1,$$

där $n_1 = 4$, behöver vi $n_0 \times n_1 = 3 \times 4 = 12$ vikter och $n_1 = 4$ bias

- Vi använder den sigmoidala aktiveringsfunktionen σ

Mer om noder, vikter och bias



Figur: Beräkna andra lagret

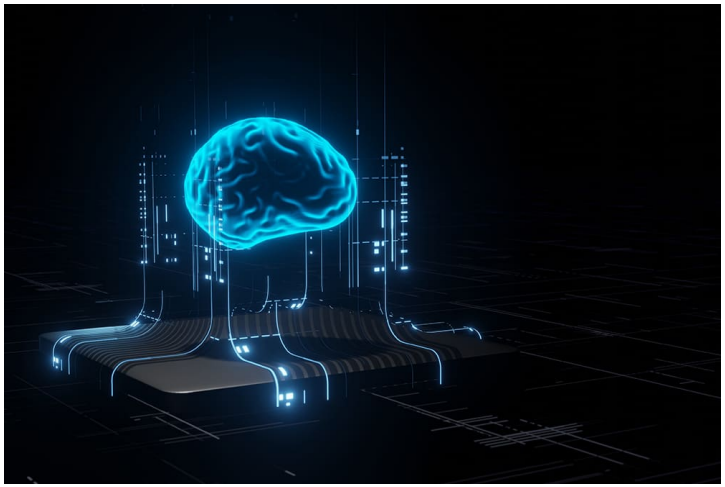
- Ännu mer generellt kan vi beräkna $a_1^{(1)}, \ldots, a_{n_1}^{(1)}$ som

$$\begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_{n_1}^{(1)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{1,1}^{(0)} & w_{1,2}^{(0)} & \cdots & w_{1,n_0}^{(0)} \\ w_{2,1}^{(0)} & w_{2,1}^{(0)} & \cdots & w_{2,n_0}^{(0)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_1,1}^{(0)} & w_{n_1,2}^{(0)} & \cdots & w_{n_1,n_0}^{(0)} \end{bmatrix} \begin{bmatrix} a_1^{(0)} \\ a_2^{(0)} \\ \vdots \\ a_{n_0}^{(0)} \end{bmatrix} + \begin{bmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_{n_1}^{(0)} \end{bmatrix} \right)$$

där n_0 är det **totala antalet noder i inputlagret** och n_1 är det **totala antalet noder i det första dolda lagret**

Mer om noder, vikter och bias

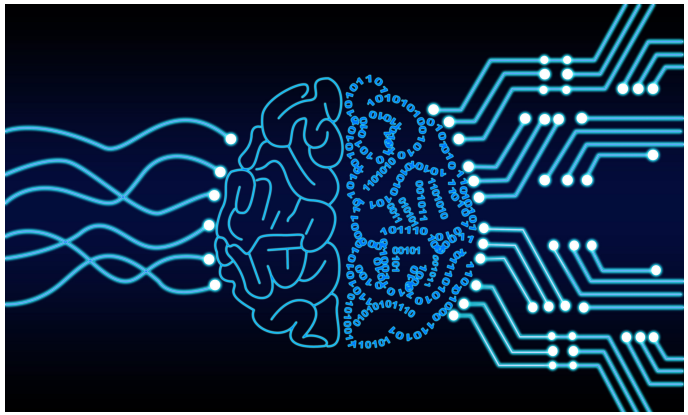
- Det är viktigt att välja bra parametrar (vikter och bias)
- Att ha tränat modellen väl betyder att vi har valt vikter och bias så att det neurala nätverket klarar av uppgiften den utvecklats för
- Senare kommer vi att ge en mer rigorös definition av en väl fungerande modell och diskuterar vi hur träningen går till



Figur: Artificiell intelligens

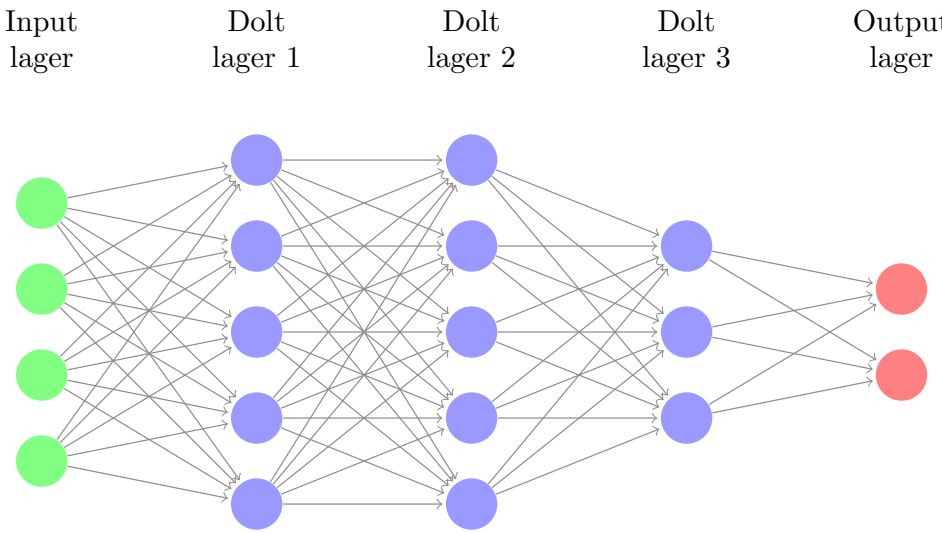
Mer om noder, vikter och bias

- Det krävs mycket jobb för att hitta de rätta vikterna och bias eftersom neurala nätverk kan innehålla miljoner vikter och bias
- Det är ännu svårare att hitta dem eftersom en justering i en enda vikt kan påverka hela neuronnätet
- Den stora utmaningen är alltså att justera alla dessa vikter så att resultatet blir optimalt



Figur: Artificiell intelligens

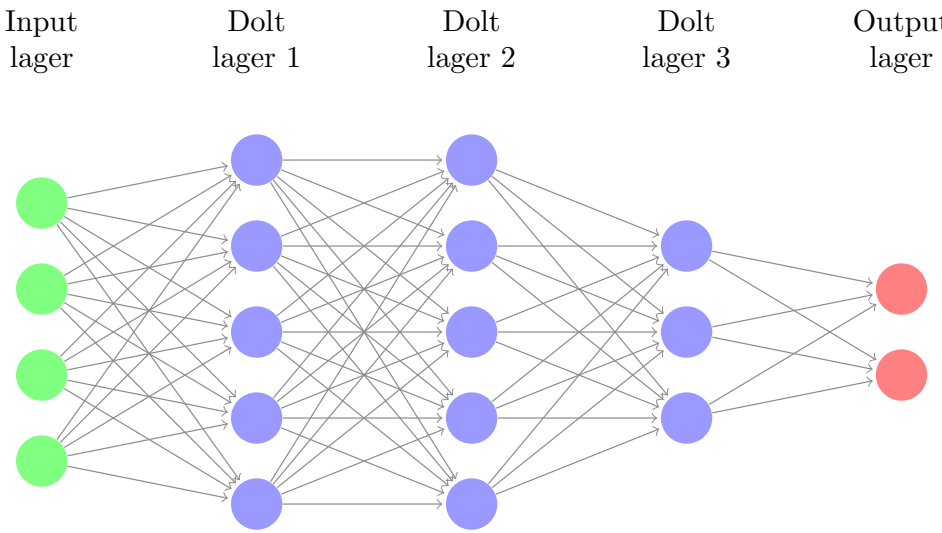
Ett djupt neuralt nätverk med tre dolda lager



Figur: Ett djupt neuralt nätverk med tre dolda lager

- Kan beräkna alla vikter och bias i det djupa neurala nätverket med tre dolda lager på samma sätt
- Delar upp det här i 4 olika steg som följer

Ett djupt neuralt nätverk med tre dolda lager



Figur: Ett djupt neuralt nätverk med tre dolda lager

- Varje steg representerar anslutningen mellan två lager
- Beräknar från vänster till höger
- Börjar med ingångsnoderna och varje steg efter det första börjar med resultatet från det föregående lagret

Steg 1: Inputlager till det första dolda lagret

$$\begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \\ a_4^{(1)} \\ a_5^{(1)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{1,1}^{(0)} & w_{1,2}^{(0)} & w_{1,3}^{(0)} & w_{1,4}^{(0)} \\ w_{2,1}^{(0)} & w_{2,2}^{(0)} & w_{2,3}^{(0)} & w_{2,4}^{(0)} \\ w_{3,1}^{(0)} & w_{3,2}^{(0)} & w_{3,3}^{(0)} & w_{3,4}^{(0)} \\ w_{4,1}^{(0)} & w_{4,2}^{(0)} & w_{4,3}^{(0)} & w_{4,4}^{(0)} \\ w_{5,1}^{(0)} & w_{5,2}^{(0)} & w_{5,3}^{(0)} & w_{5,4}^{(0)} \end{bmatrix} \begin{bmatrix} a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ a_4^{(0)} \end{bmatrix} + \begin{bmatrix} b_1^{(0)} \\ b_2^{(0)} \\ b_3^{(0)} \\ b_4^{(0)} \\ b_5^{(0)} \end{bmatrix} \right)$$

Totalt antal parametrar att välja i Steg 1		
antal vikter	antal bias	totalt
$4 \times 5 = 20$	5	$20 + 5 = 25$

Steg 2: Det första dolda lagret till det andra dolda lagret

$$\begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \\ a_4^{(2)} \\ a_5^{(2)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} & w_{1,4}^{(1)} & w_{1,5}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} & w_{2,4}^{(1)} & w_{2,5}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} & w_{3,3}^{(1)} & w_{3,4}^{(1)} & w_{3,5}^{(1)} \\ w_{4,1}^{(1)} & w_{4,2}^{(1)} & w_{4,3}^{(1)} & w_{4,4}^{(1)} & w_{4,5}^{(1)} \\ w_{5,1}^{(1)} & w_{5,2}^{(1)} & w_{5,3}^{(1)} & w_{5,4}^{(1)} & w_{5,5}^{(1)} \end{bmatrix} \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \\ a_4^{(1)} \\ a_5^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \\ b_5^{(1)} \end{bmatrix} \right)$$

Totalt antal parametrar att välja i Steg 2		
antal vikter	antal bias	totalt
$5 \times 5 = 25$	5	$25 + 5 = 30$

Steg 3: Det andra dolda lagret till det tredje dolda lagret

$$\begin{bmatrix} a_1^{(3)} \\ a_2^{(3)} \\ a_3^{(3)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} & w_{1,3}^{(2)} & w_{1,4}^{(2)} & w_{1,5}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} & w_{2,3}^{(2)} & w_{2,4}^{(2)} & w_{2,5}^{(2)} \\ w_{3,1}^{(2)} & w_{3,2}^{(2)} & w_{3,3}^{(2)} & w_{3,4}^{(2)} & w_{3,5}^{(2)} \end{bmatrix} \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \\ a_4^{(2)} \\ a_4^{(2)} \\ a_5^{(2)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \end{bmatrix} \right)$$

Totalt antal parametrar att välja i Steg 3		
antal vikter	antal bias	totalt
$5 \times 3 = 15$	3	$15 + 3 = 18$

Steg 4: Det tredje dolda lagret till outputlagret

$$\begin{bmatrix} a_1^{(4)} \\ a_2^{(4)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{1,1}^{(3)} & w_{1,2}^{(3)} & w_{1,3}^{(3)} \\ w_{2,1}^{(3)} & w_{2,2}^{(3)} & w_{2,3}^{(3)} \end{bmatrix} \begin{bmatrix} a_1^{(3)} \\ a_2^{(3)} \\ a_3^{(3)} \end{bmatrix} + \begin{bmatrix} b_1^{(3)} \\ b_2^{(3)} \end{bmatrix} \right)$$

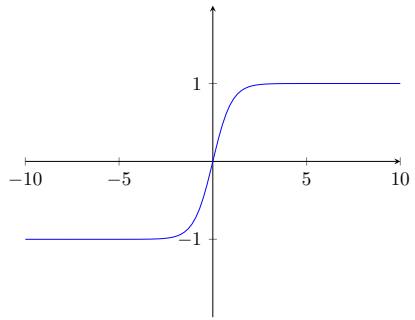
Totalt antal parametrar att välja i Steg 4		
antal vikter	antal bias	totalt
$3 \times 2 = 6$	2	$6 + 2 = 8$

- Då har vi i totalt: $25 + 30 + 18 + 8 = 81$ parametrar att välja
- Vi ska senare diskutera vad det betyder att välja parametrar
- Kort sagt: att välja alla parametrar är samma sak som att träna ett djupt neuralt nätverk

Mer om aktiveringsfunktioner

- En sigmoidal aktiveringsfunktion är inte det enda möjliga valet för en aktiveringsfunktion
- Tangens hyperbolicus sådant att

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

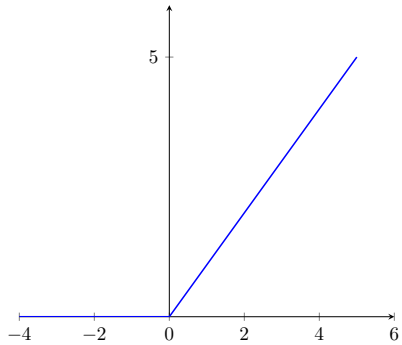


Figur: Aktiveringsfunktion tanh, tangens hyperbolicus

Mer om aktiveringsfunktioner

- En sigmoidal aktiveringsfunktion är inte det enda möjliga valet för en aktiveringsfunktion
- Den mest använda aktiveringsfunktionen idag heter ReLU (eng: *Rectified Linear Unit*)

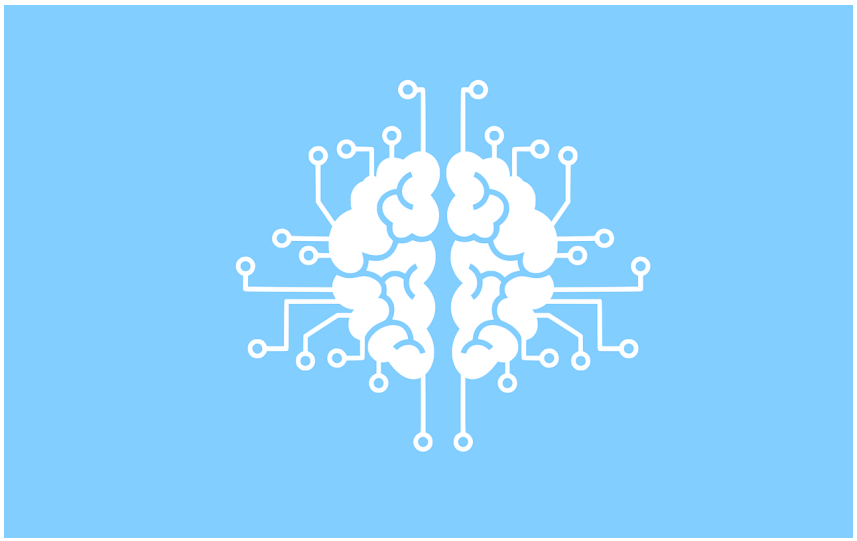
$$R(x) = \max(0, x)$$



Figur: Aktiveringsfunktion R, s.k. ReLU

Förlustfunktionen (1)

- Vikterna justeras med hjälp av **förlustfunktionen** (eng: *loss function*)
- Förlustfunktionen beräknar hur stor skillnaden är mellan nätverkets output och sanningen genom att testa modellen på exempel där vi redan vet sanningen



Figur: Artificiell intelligens

Förlustfunktionen (1)

- Vikterna justeras med hjälp av **förlustfunktionen** (eng: *loss function*)
- Förlustfunktionen beräknar hur stor skillnaden är mellan nätverkets output och sanningen genom att testa modellen på exempel där vi redan vet sanningen

- Testar nätverket på ett exempel och definierar förlustfunktionen L som

$$L(X) = |Y(X) - \hat{Y}(X)|^2$$

  - X är ingångsnoderna på ett exampel
  - $Y(X)$ är nätverkets gissning på X (utgångsnoderna)
  - $\hat{Y}(X)$ är etiketten på exemplet (sanningen)
  - $|z|^2 = z_1^2 + \ldots + z_n^2$ är längd i kvadrat av vektorn $z \in \mathbb{R}^n$

Förlustfunktionen (2)

- Inom maskininlärning tittar vi på många exempel när vi tränar modellen
- Tittar på medelvärdet av $L(X) = |Y(X) - \hat{Y}(X)|^2$ över N exempel, d.v.s.,

$$\frac{1}{N}\sum_{l=1}^{N}|Y(X_l) - \hat{Y}(X_l)|^2,\ X = \{X_l\}_{l=1}^{N}$$

där varje X_l i mängden $\{X_l\}_{l=1}^{N}$ representerar ett exempel

Förlustfunktionen (3)

- Den här förlustfunktionen, d.v.s. medelvärdet av $L(X) = |Y(X) - \hat{Y}(X)|^2$ över N exempel

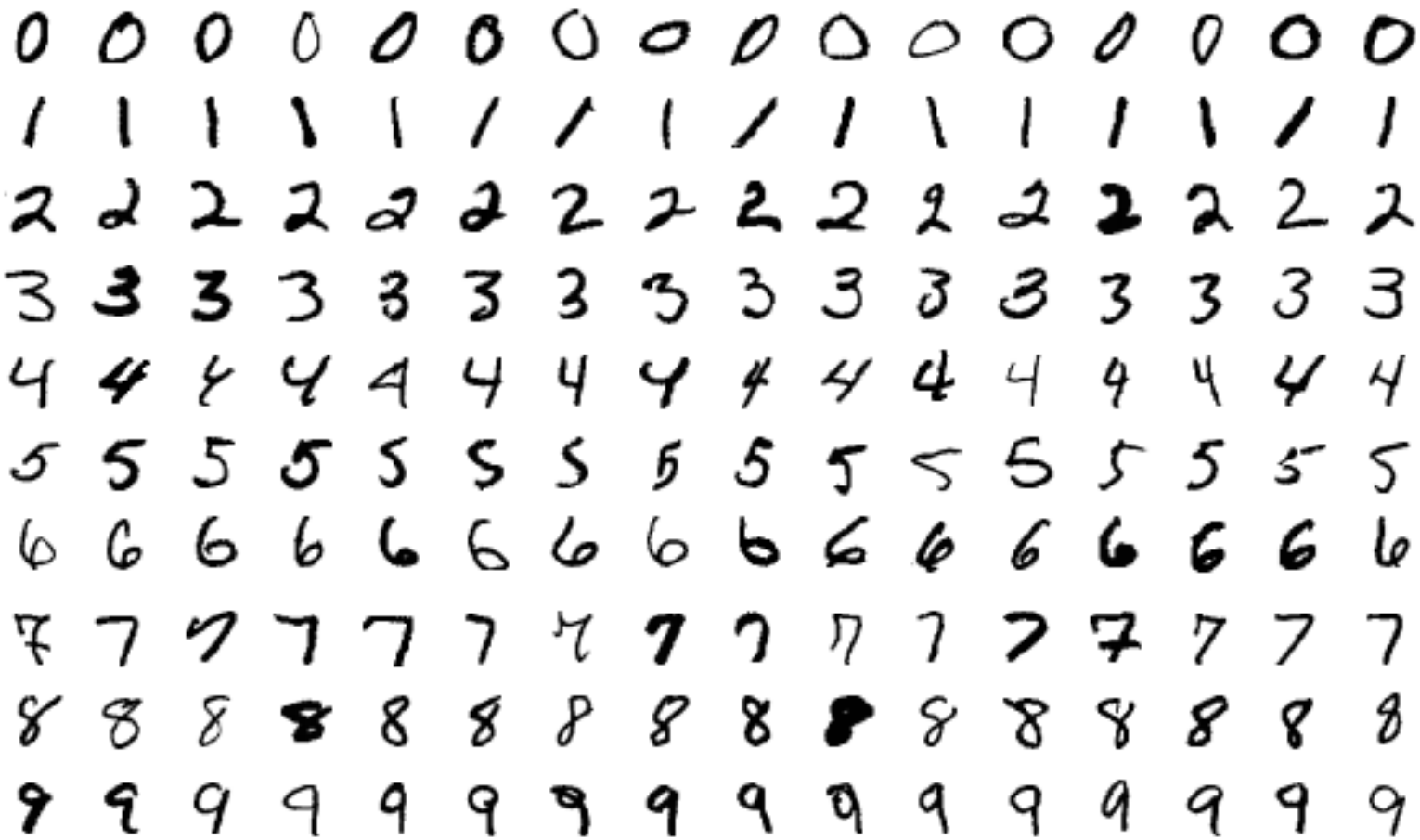
$$\frac{1}{N}\sum_{l=1}^{N}|Y(X_l) - \hat{Y}(X_l)|^2, \; X = \{X_l\}_{l=1}^{N}$$

  heter **medelkvadratfel** (eng: *mean square error*)
- Inte den enda möjliga förlustfunktionen
- Ska använda en förlustfunktion som heter **korsentropifunktionen** (eng: *cross entropy function*) när vi progammerar lite senare

Förlustfunktionen (4)

- Om värdet på förlustfunktionen $L(X) = |Y(X) - \hat{Y}(X)|^2$ är stort på ett exempel X så är skillnaden mellan nätverkets gissning och sanningen stor
- Om värdet på förlustfunktionen $\frac{1}{N}\sum_{l=1}^{N}|Y(X_l) - \hat{Y}(X_l)|^2$ är stort på en mängd $X = \{X_l\}_{l=1}^{N}$ fungerar neurala nätverket dåligt
- Om värdet på $\frac{1}{N}\sum_{l=1}^{N}|Y(X_l) - \hat{Y}(X_l)|^2$ litet
  - Neurala nätverket gjort många bra gissningar
  - Modellen fungerar bra
- Vi tar ett exempel för att gör detta konkret

Handskrivna siffror från MNIST-databasen (1)



Figur: Handskrivna siffror från MNIST-databasen, bild från Wikipedia

Handskrivna siffror från MNIST-databasen (2)

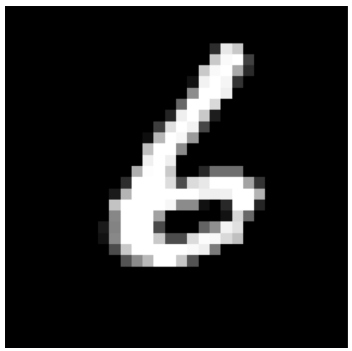
- Uppgift är att klassifiera handskriva siffror med ett djupt neuralt nätverk
- Ska använda MNIST-databasen (eng: *Modified National Institute of Standards and Technology*) för att träna modellen
- MNIST-databasen är en samling av sextio tusen små, kvadratiska gråskalebilder
- Varje bild har 28×28 pixlar, med en enda handskriven siffra mellan 0 och 9
- Bilderna kommer med en etikett med den rätta siffran som en **enhetsvektor**

Handskrivna siffror från MNIST-databasen (3)

- Enhetsvektorn är sådan att ett element är 1 och alla andra element är 0
- Elementet som är 1 motsvarar siffran som är på bilden
- Låt X representerar en bild av siffran 6
- Motsvarande etiketten är då

$$\hat{Y}(X) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T \in \mathbb{R}^{10},$$

där T betecknar transponat



Figur: En handskriven siffra från MNIST-databasen

Handskrivna siffror från MNIST-databasen (4)

- Notera: transponatet av en vektor är sådan att

$$\begin{bmatrix} a & b & c \end{bmatrix}^T = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

- Siffran 0 har etiketten

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \in \mathbb{R}^{10}$$

Handskrivna siffror från MNIST-databasen (5)

- MNIST-databasen är utmärkt för att träna neurala nätverk eftersom det finns så många märkta exempel
- Först tar vi de $28 \times 28 = 784$ pixlarna i en bild
- Varje pixel är ett värde som bestämmer pixelns färg
- 0 representerar svart och 1 är vit och alla tal i intervallet $(0, 1)$ är nyanser av grått

Handskrivna siffror från MNIST-databasen (6)

- Skapar en lång vektor med 784 element från varje bild
- Vektorn går in till det neurala nätverket som

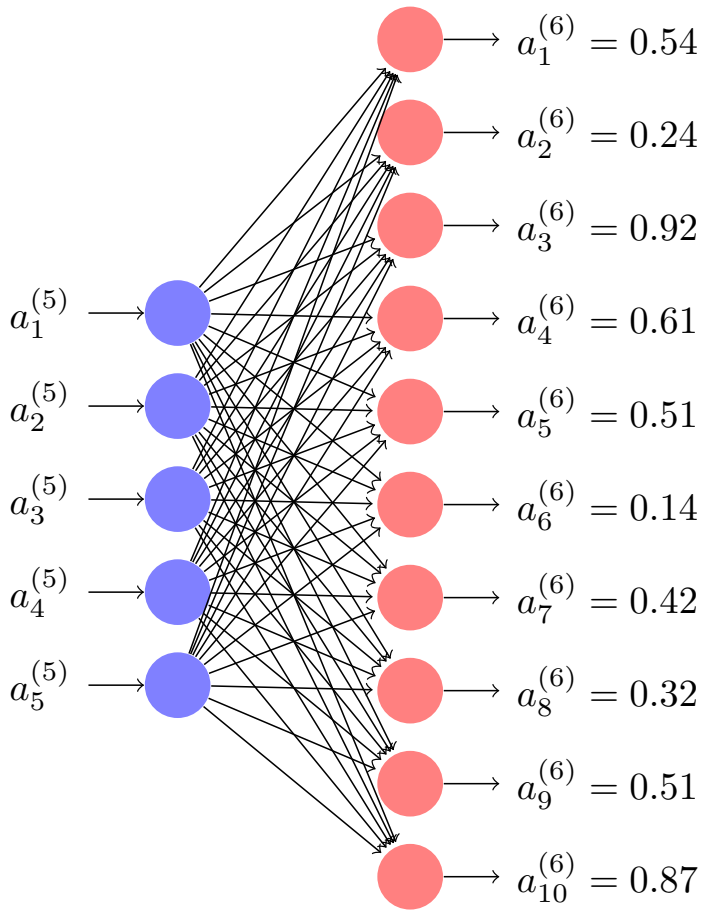
$$X = \begin{bmatrix} a_1^{(0)} & a_2^{(0)} & \cdots & a_{784}^{(0)} \end{bmatrix}^T \in \mathbb{R}^{784},$$

där bilden ges av matrisen

$$\begin{bmatrix} a_1^{(0)} & a_{29}^{(0)} & \cdots & a_{757}^{(0)} \\ a_2^{(0)} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{28}^{(0)} & \cdots & \cdots & a_{784}^{(0)} \end{bmatrix} \in \mathbb{R}^{28\times 28}, \; a_i^{(0)} \in [0,1], i = 1, \ldots, 784$$

- Vektorn X består av ingångsnoderna
- Vektorn har all information vi behöver från bilden

Handskrivna siffror från MNIST-databasen (7)

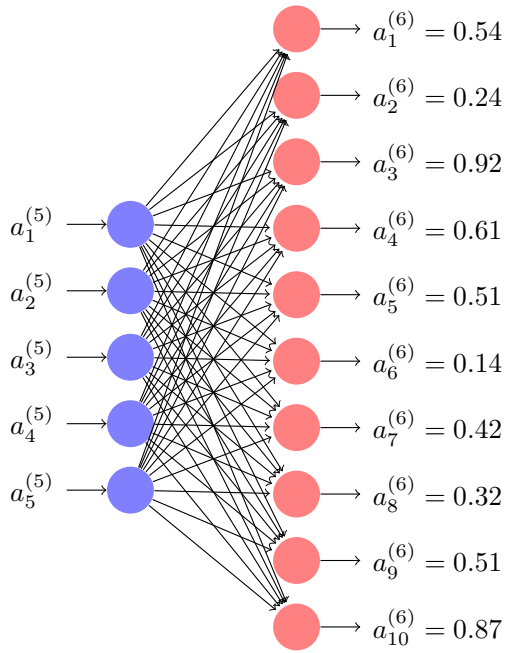


Figur: Det sista steget i klassificering av handskrivna siffror i ett djupt neuralt nätverk med 5 dolda lager

Handskrivna siffror från MNIST-databasen (8)

- Från ingångsnoderna beräknar vi utgångsnoderna genom samma process som beskrevs innan
- 10 utgångsnoder
- Varje utgångsnod motsvarar en siffra mellan 0 och 9
- Siffran som motsvarar utgångsnoden med det största värdet är modellens klassificering

Handskrivna siffror från MNIST-databasen (9)



Figur: Det sista steget i klassificering av handskrivna siffror i ett djupt neuralt nätverk med 5 dolda lager

- Sista steget i klassificering av handskrivna siffror i ett djupt neuralt nätverk med 5 dolda lager
- Modellen gissar att exemplet är en tvåa eftersom utgångsnoden $a_3^{(6)} = 0.92$ har det största värdet

Handskrivna siffror från MNIST-databasen (10)

- Anta först att nätverket klassificerar exempelet X rätt, d.v.s. att etiketten $\hat{Y}(X)$ ges av enhetsvektorn

$$\hat{Y}(X) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \in \mathbb{R}^{10}$$

- Beräknar förlustfunktionens värdet på exemplet i som längden i kvadrat av vektorn

$$\begin{bmatrix} (0.54 - 0.00) \\ (0.24 - 0.00) \\ (0.92 - 1.00) \\ (0.61 - 0.00) \\ (0.51 - 0.00) \\ (0.14 - 0.00) \\ (0.42 - 0.00) \\ (0.32 - 0.00) \\ (0.51 - 0.00) \\ (0.87 - 0.00) \end{bmatrix} \in \mathbb{R}^{10}$$

Handskrivna siffror från MNIST-databasen (11)

- Anta nu att nätverket klassificerar exempelet X fel, d.v.s. att etiketten $\hat{Y}(X)$ ges av enhetsvektorn

$$\hat{Y}(X) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T \in \mathbb{R}^{10}$$

  d.v.s. siffra är en 9a i verkligheten men nätverket gissar att det är en 2a

- Beräknar vi förlustfunktionens värdet som längden i kvadrat av

$$\begin{bmatrix} (0.54 - 0.00) \\ (0.24 - 0.00) \\ (0.92 - 0.00) \\ (0.61 - 0.00) \\ (0.51 - 0.00) \\ (0.14 - 0.00) \\ (0.42 - 0.00) \\ (0.32 - 0.00) \\ (0.51 - 0.00) \\ (0.87 - 1.00) \end{bmatrix} \in \mathbb{R}^{10}$$

Handskrivna siffror från MNIST-databasen (12)

- Förlustfunktionens värde av den första (rätta) vektorn beräknas som

$$\begin{aligned} L(X) = \quad & (0.54 - 0.00)^2 + (0.24 - 0.00)^2 + (0.92 - 1.00)^2 \\ + \quad & (0.61 - 0.00)^2 + (0.51 - 0.00)^2 + (0.14 - 0.00)^2 \\ + \quad & (0.42 - 0.00)^2 + (0.32 - 0.00)^2 + (0.51 - 0.00)^2 \\ + \quad & (0.87 - 0.00)^2 \end{aligned}$$

d.v.s.

$$L(X) = 2.3032$$

Handskrivna siffror från MNIST-databasen (13)

- Förlustfunktionens värde av den andra (fela) vektorn beräknas som

$$\begin{aligned} L(X) = & \quad (0.54-0.00)^2+(0.24-0.00)^2+(0.92-0.00)^2 \\ & + \quad (0.61-0.00)^2+(0.51-0.00)^2+(0.14-0.00)^2 \\ & + \quad (0.42-0.00)^2+(0.32-0.00)^2+(0.51-0.00)^2 \\ & + \quad (0.87-1.00)^2 \end{aligned}$$

d.v.s.

$$L(X) = 2.4032$$

- Det första värdet var mindre än värdet här eftersom nätverket klassificerade exemplet rätt och det här fel

Att träna modellen (1)

- Förlustfunktionen visar hur mycket modellens beräkningar skiljer sig från verklighetens
- Söker det **globala minimumet** i förlustfunktionen, d.v.s. vikter och bias sådan att förlustfunktion minimeras
- Ett djupt neuralt nätverk kan ha tusentals vikter och bias
- I praktiken hittar vi en approximation till vikterna och bias som minimerar förlustfunktionen
- Ska använda **iterativa metoder** för att approximera vikterna och bias
- Iterativa metoder börjar med en startgissning och genom successiva förändringar av densamma åstadkommer en successivt förbättrad approximation av lösningen till problemet

Att träna modellen (2)

- Först: för enkelhetens skull, minimerar vi en funktion

$$f : \mathbb{R}^2 \to \mathbb{R}$$

- Senare: minimerar vi en förlustfunktion för att träna ett neuralt nätverk

Att träna modellen (3)

- Metoden **gradientnedstigning** är en iterativ metod för att lösa ett minimeringsproblem
- Med varje iteration av metoden får vi en bättre approximation av **minimipunkten**
- Gradientnedstigning tar *steg* i motsatt riktning till gradienten av funktionen
- Notera: ∇f betyder samma sak som $\text{grad} f$ eftersom funktionen minskar snabbast i den riktningen

Att träna modellen (4)

Algoritm 1: Gradientnedstigning

input : Startgissning $(x_1^{(0)}, \ldots, x_k^{(0)}) \in \mathbb{R}^k$
Gradient av funktionen $f : \mathbb{R}^k \to \mathbb{R}$ sådan att $\nabla f \in \mathbb{R}^k$
$\gamma \in \mathbb{R}$
Parameter $m \in \mathbb{Z}$ (antal steg)

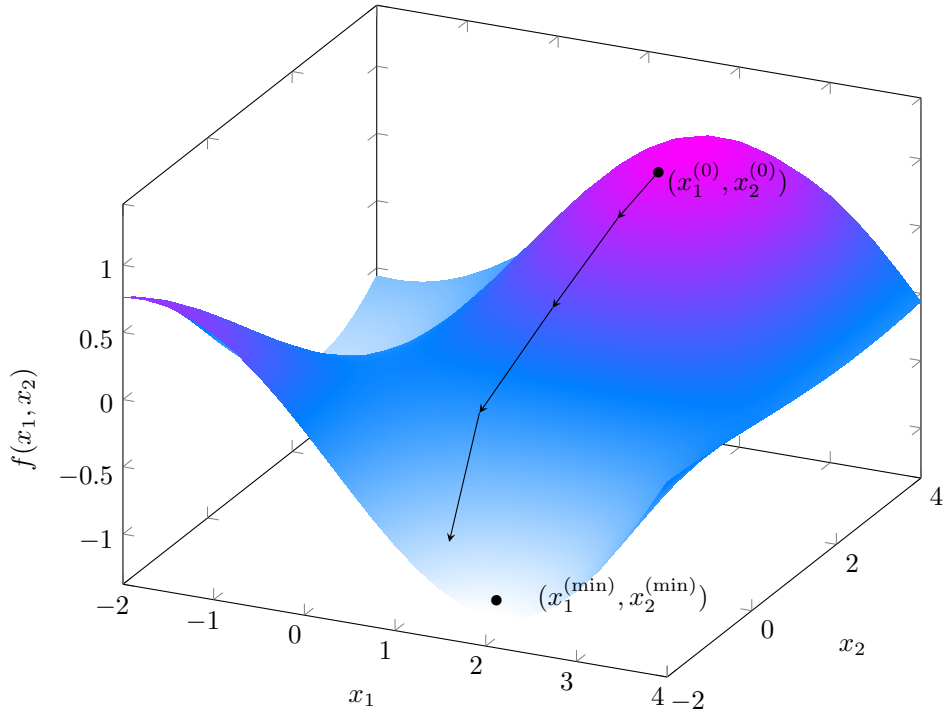
output: $(x_1^{(m+1)}, \ldots, x_k^{(m+1)}) \in \mathbb{R}^k$ sådant att
$f(x_1^{(m+1)}, \ldots, x_k^{(m+1)}) \leqslant f(x_1^{(0)}, \ldots, x_k^{(0)})$

1 **for** $j = 0, 1, \ldots, m$ **do**

2 $$\begin{bmatrix} x_1^{(j+1)} \\ \vdots \\ x_k^{(j+1)} \end{bmatrix} = \begin{bmatrix} x_1^{(j)} \\ \vdots \\ x_k^{(j)} \end{bmatrix} - \gamma \nabla f \left(\begin{bmatrix} x_1^{(j)} \\ \vdots \\ x_k^{(j)} \end{bmatrix} \right)$$

3 **end**

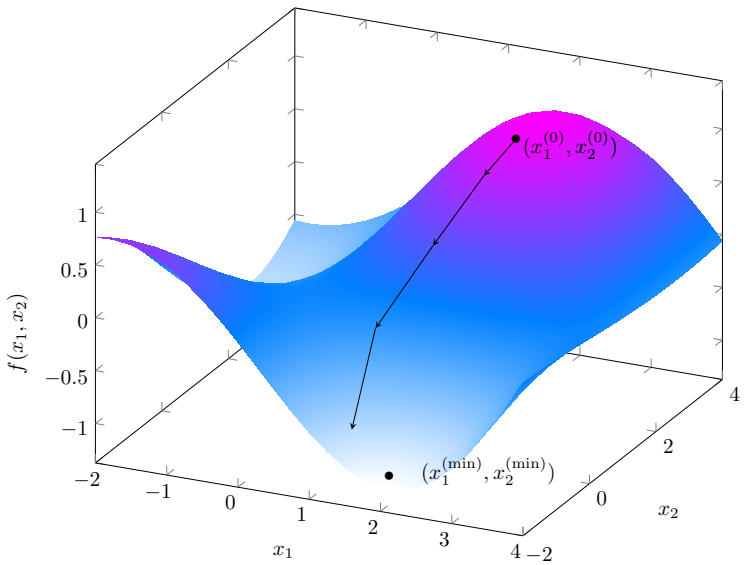
Att träna modellen (5)



Figur: Fyra steg av metoden gradientnedstigning för att approximera en minimipunkt på en funktion $f : \mathbb{R}^2 \to \mathbb{R}$

Att träna modellen (6)

- Hitta en minimipunkt på en funktion med 2 variabler
- startgissning: $(x_1^{(0)}, x_2^{(0)}) = (2, 2)$
- Funktionen f ges av $f(x_1, x_2) = \sin(0.8x_1)\sin(0.6x_2)e^{0.1x_1}$ där $D_f = \{(x_1, x_2) : -2 \leqslant x_1 \leqslant 3, -2 \leqslant x_2 \leqslant 2\}$
- Minimipunkten finns i $(x_1^{(\min)}, x_2^{(\min)}) \approx (2.11894, -2)$ och $f(x_1^{(\min)}, x_2^{(\min)}) \approx -1.14312$



Att träna modellen (7)

- Parametern γ i Algoritm 1 kallas för **inlärningshastighet**
- Vi tar $\gamma \in \mathbb{R}$ som en liten konstant
- Gradientnedstigning kommer att stanna om den når fram (eller kommer väldigt nära) till ett minimum eftersom gradienten är noll (eller nästan noll) där
- Vi ser att algoritmen därmed inte gör så mycket i varje steg, eftersom det j:te steg ges av $\gamma \nabla f(x^{(j)})$

Att träna modellen (8)

- Det är omöjligt att visualisera minimeringsproblemet om dimensionen är större än 3
  - t.ex. problemet att minimera förlustfunktionen som innan
  - X är en mängd med N bilder
- Processen densamma och gradientnedstigning fungerar fortfarande

Att träna modellen (9)

- Metoden **stokastisk gradientnedstigning** är en iterativ metod som ofta används för att minimera förlustfunktionen
- Metoden är ganska lik gradientnedstigning
- Skillnaden är att metoden stokastisk gradientnedstigning approximerar gradienten istället för att beräkna den exakt
- Vissa steg av stokastisk gradientnedstigning är inte så effektiva när det gäller att minimera förlustsfunktionen
- I djupinlärning **konvergerar** metoden stokastisk gradientnedstigning ofta bättre än gradientnedstigning
- Det betyder att metoden hittar en approximation på ett bättre sätt (t.ex. snabbare) eller hittar en approximation som är närmare sanningen