Neurala nätverk

Introduktion

- Via maskininlärning kan en given maskin bli kapabel till att lära sig regler utan att ha programmerats till det. Djupinlärning är en undergren av maskininlärning och innefattar tekniker som vagt påminner om hur våra hjärnor fungerar används. Namnet djupinlärning härstammar från att djupinlärning ursprungligen inspirerades av hur våra hjärnor fungerar. I praktiken fungerar dock hjärnor samt neurala nätverk väldigt olika.

- I djupinlärning används så kallade neurala nätverk innehållande flera steg, eller lager, mellan indata samt utdata, ofta hundratals, vilket medför att relevant information kan extraheras ur indata, medan icke-relevant information kan förkastas. Därmed är djupinlärning lämpligt för bilder och dylikt som innehåller en hög mängd information, varav det mesta vanligtvis är insignifikant.

- Ju fler lager som används i ett neuralt nätverk, desto djupare modell. Information kan tänkas gå igenom och behandlas i olika filter, som destillerar information. I varje lager filtreras en viss typ av irrelevant data, medan relevant data bibehålls. Stegvis ändras aktuell indata från sin ursprungliga form och innehåller i ökad grad information om utdatan.

Teori gällande träning av ett neuralt nätverk

- För att ett neuralt nätverk ska kunna prediktera med låg avvikelse måste nätverket tränas, så att de ingående nodernas parametrar (vikter samt bias) justeras till lämpliga värden. Befintliga träningsuppsättningar används därmed för att träna nätverket ett visst antal epoker. Vid varje epok randomiseras ordningen på träningsuppsättningarna för att nätverket inte ska vänja sig vid eventuella mönster som uppkommer i just den ordning som träningsuppsättningarna är lagrade.

- För varje träningsuppsättning som används för att träna modellen genomförs följande för ett enkelt neuralt nätverk bestående av ett ingångslager, ett dolt lager samt ett utgångslager:

1. Feedforward

- Noderna i ingångslagret utgör insignaler x på samtliga neuroner i det dolda lagret. Eftersom dessa insignaler x innehar olika stor betydelse, även kallat vikt, så multipliceras dessa med var sin vikt k i det dolda lagret. Dessa vikter bör vid start vara slumpmässigt valda, för att sedan justeras vid träning. För varje nod i det dolda lagret summeras bidraget k * x från varje insignal x från ingångslagret tillsammans med nodens vilopunkt / bias m till en summa s enligt nedan:

$$s = m + \sum_{i=0}^{j} k_i * x_i,$$

vilket är ekvivalent med

$$s = m + k_0 * x_0 + k_1 * x_1 \ldots + k_j * x_j,$$

där m är nodens bias, j är antalet insignaler och $k_i * x_i$ är produkten av respektive vikt x samt ansluten insignal x.

- Nodens predikterade utsignal y_p utgörs av ovanstående summa s, som behandlas via en aktiveringsfunktion. Nodens utsignal y erhålls därmed som ett flyttal enligt nedan:

$$y_p = \delta(s),$$

vilket är ekvivalent med följande:

$$y_p = \delta\left(m + \sum_{i=0}^{j} k_i * x_i\right) = \delta(m + k_0 * x_0 + k_1 * x_1 \ldots + k_j * x_j),$$

där y är nodens utsignal, δ är aktiveringsfunktionen, m är nodens bias, j är antalet insignaler och $k_i * x_i$ är produkten av respektive vikt k samt ansluten insignal x.

- Utsignalerna från det dolda lagret utgör sedan insignaler på det yttre lagret, där samma princip följs för att beräkna utsignaler y för noderna i det yttre lagret.

2. Backpropagation

- Avvikelsen för respektive nods utsignal beräknas från det yttre lagret och bakåt till det dolda lagret för att sedan genomföra justering av samtliga noders parametrar (bias m samt vikter k). För en given nod n i det yttre lagret gäller att

$$\delta = y_{ref} - y_p,$$

där δ är avvikelsen, y_{ref} är referensvärdet från träningsdatan och y_p är nodens utsignal.

- Beräknat fel Δe för en given nod n i det yttre lagret kan beräknas med följande formel:

$$\Delta e = \delta * y_p',$$

där δ är avvikelsen och y_p' är derivatan av nodens predikterade utsignal y_p.

- För en given nod n i ett givet dolt lagret gäller att

$$\delta = \sum_{i=0}^{j} [\Delta e_i * k_i],$$

där δ är avvikelsen, j är antalet noder i nästa lager, Δe_i är beräknat fel för respektive nod i nästa lager och utgör k_i vikten för aktuell nod i nästa lager. Vidare gäller att beräknat fel Δe för en given nod kan beräknas med formeln

$$\Delta e = \delta * y_p',$$

där y_p' är derivatan ur aktuell nods predikterade utsignal y_p.

- Därmed gäller att felet Δe för en given nod kan beräknas direkt via följande formel:

$$\Delta e = \sum_{i=0}^{j} [\Delta y_i * k_i] * y_p',$$

där δ är avvikelsen, j är antalet noder i nästa lager, Δy_i är beräknat fel för varje nod i nästa lager, k_i utgör vikten för noden i i nästa lager och y_p' är derivatan ur aktuell nods utsignal y_n.

3. Optimering

- Efter att fel för nodernas parametrar har beräknats, så sker optimering för att minska avvikelserna via en gradientalgoritm. För en given nod n i ett givet lager gäller att förändringshastigheten Δc_n kan beräknas med följande formel:

$$\Delta c_n = \Delta y_n * L,$$

där Δy_n är delta för aktuell nod och L är *learning rate* (lärhastigheten).

- Nodens bias m_n ska ökas med förändringshastigheten Δc_n:

$$m_n = m_n + \Delta c_n = m_n + \Delta y_n * L$$

- För respektive vikt k_j ansluten till noden n gäller att dessa ska ökas enligt nedan:

$$k_j = k_j + \Delta c_n * f_j(x),$$

där $f_j(x)$ är utsignal från ansluten nod j i föregående lager.

Aktiveringsfunktioner

- För varje nod i det dolda och yttre lagret används vanligtvis någon typ av aktiveringsfunktion, för att hålla utsignalerna inom ett visst intervall, såsom 0 – 1, som kan utgöra insignal till noder i nästa lager. Som vi såg tidigare gällande att summan en given nods bias m summeras med samtliga insignaler x multiplicerat med motsvarande vikt k till en summa s såsom tidigare beskrivet:

$$s = m + \sum_{i=0}^{j} k_i * x_i,$$

vilket är ekvivalent med

$$s = m + k_0 * x_0 + k_1 * x_1 \ldots + k_j * x_j,$$

där m är nodens bias, j är antalet insignaler och $k_i * x_i$ är produkten av respektive vikt k samt ansluten insignal x..

- Nodens utsignal y erhålls därmed som ett flyttal enligt nedan:

$$y = \delta(s),$$

vilket är ekvivalent med följande:

$$y = \delta\left(m + \sum_{i=0}^{j} k_i * x_i\right) = \delta\big(m + k_0 * x_0 + k_1 * x_1 \ldots + k_j * x_j\big),$$

där y är nodens utsignal, δ är aktiveringsfunktionen, m är nodens bias, j är antalet insignaler och $k_i * x_i$ är produkten av respektive vikt k samt ansluten insignal x.

- Några vanliga aktiveringsfunktioner är följande:

1. ReLU

- Den vanligaste aktiveringsfunktionen inom djupinlärning är ReLU *(Rectified Linear Unit)*, som fungerar lite som en neuron i hjärnan, där summan av samtliga insignaler måste övergå ett visst tröskelvärde för att denna ska passera, annars blir utsignalen noll enligt följande:

$$\begin{cases} s \geq 0 => y = s \\ s < 0 => y = 0 \end{cases}$$

- Som regel brukar ReLU utgöra den lämpligaste modellen att implementera när det inte finns något entydigt svar till alla problem.

- Leaky Relu utgör en förbättrad variant av ReLU, där signaler som understiger noll inte förloras, utan bibehålles som en svag signal som utgör en bråkdel k av summan s enligt nedan:

$$\begin{cases} s \geq 0 => y = s \\ s < 0 => y = s * k \end{cases}$$

- Som en tumregel kan ett k-värde på 0.01 användas, vilket innebär att

$$y = s * 0.01 \; då \; s < 0$$

- Leaky ReLU är används för att motverka att derivatan y'(s) = 0 för en reguljär ReLU då insignal x understiger 0. Leaky ReLU fungerar som regel bättre än en reguljär ReLU, men används tyvärr sällan i praktiken. Anledningen till att en leaky ReLU fungerar bättre än den reguljära varianten är att insignaler från noder som inkommer med negativa värden "fastnar" på värdet noll, vilket i praktiken innebär att dessa dör. Detta kan leda till att modellen inte tränas väl och därmed predikterar dåligt. När leaky ReLU används motverkas dock detta problem, då även neuroner som annars hade förlorats hålls aktiva.

- Derivatan dy av utsignal y från en ReLU-funktion definieras enligt nedan:

$$\begin{cases} y > 0 => dy = 1 \\ y \leq 0 => dy = 0 \end{cases}$$

- I praktiken är derivatan av ReLU inte definierad för y = 0. Dock är det högst osannolikt att utsignal y blir exakt noll, så ifall detta sker antas ett litet fel ha uppstått och derivatan dy sätts då till noll.

2. Sigmoid

- Sigmoid är en vanlig aktiveringsfunktion ifall enbart två klasser / kategorier används, där utsignalen sätts till ett tal mellan 0 – 1. Utsignalen y för en nod med summan s av dess parametrar kan vid användning av aktiveringsfunktionen sigmoid beräknas enligt nedan:

$$y = \frac{1}{1 + e^{-s}}$$

- Derivatan dy av utsignalen y från en aktiveringsfunktion bestående av en sigmoid kan sedan beräknas enligt nedan:

$$dy = y * (1 - y)$$

3. Softmax

- Softmax är en vanlig aktiveringsfunktion i det yttre lagret, där summan av samtliga utsignalers sannolikheter blir ett. Därmed skaleras sannolikheterna mot varandra. Detta är mycket fördelaktigt vid bildigenkänning, då utsignalerna y indikerar vilken kategori som det är mest sannolikt att aktuell bild innehar ett element, där kategorierna jämförs inbördes.

- Utsignalen y för en nod med summan s av dess parametrar i ett lager bestående av j noder kan vid användning av aktiveringsfunktionen softmax beräknas enligt nedan:

$$y = \frac{e^{s}}{\sum_{i=0}^{j} e^{s_i}} = \frac{e^{s}}{e^{s_0} + e^{s_1} \ldots + e^{s} + e^{s_j}},$$

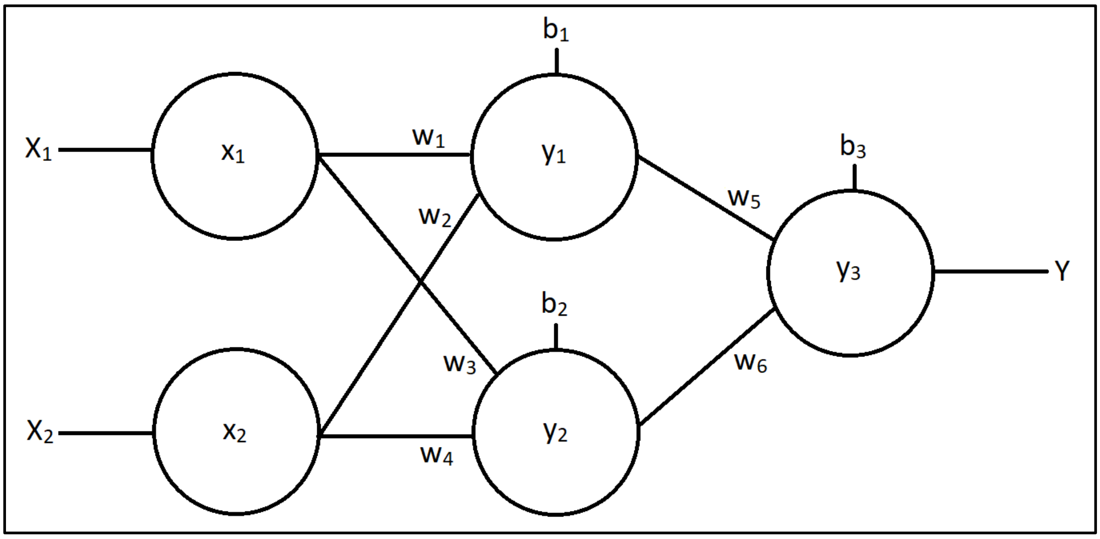
  vilket motsvarar bidraget e^{ws} från aktuellt nods summa dividerat på summan av bidraget från samtliga noder i lagret. Softmax kommer inte behandlas ytterligare i detta avsnitt.

Träning av ett neuralt nätverk för hand – Exempel 1

- Det neurala nätverket i nedanstående figur ska tränas till att kunna prediktera en hög utsignal Y vid udda antal höga insignaler X_1 och X_2, annars ska utsignal Y bli noll.

- Antag att träning nu ska genomföras och den första träningsuppsättningen som nätverket utsätts för innehar insignaler $X_1X_2 = 01$ samt Y = 1.

- Nodernas parametrar har vid start har randomiserats till följande:
  b1 = 0.2, b2 = 0.4, b3 = 0.6
  w1 = 0.5, w2 = 0.6, w3 = 0.3
  w4 = 0.8, w5 = 0.1, w6 = 0.9



Figur 1 – Litet neuralt nätverk.

- Antag att lärhastigheten LR är satt till 0.1 och att aktiveringsfunktionen ReLU används för alla noder.
- Genomför feedforward, backpropagation samt optimering.
- Beräkna sedan utsignalen igen. Blev felet mindre? Om inte, vad hade du kunnat ändra för att erhålla mindre fel?

Lösning – Exempel 1

- Träning sker genom feedforward (beräkning av nya utsignaler för varje nod), backpropagation (beräkning av aktuellt fel genom att jämföra nätverkets utsignaler mot referensvärden från träningsdatan) samt optimering (justering av nätverkets parametrar i syfte att minska aktuellt fel).

1. Feedforward

- Beräkna utsignal y_n från varje nod. Beräkning sker framåt.

- Utsignal x_1 och x_2 ur respektive nod i ingångslagret är samma som nätverkets insignaler X_1 respektive X_2:

$$x_1 = X_1 = 0$$

$$x_2 = X_2 = 1$$

- För en given nod n i ett dolt eller yttre lager gäller att utsignal y_n är lika med summan s_n av nodens bias b_n samt bidraget $x_n w_n$ från respektive insignal, filtrerat genom aktiveringsfunktionen ReLU. För k antal noder gäller därmed att

$$y_n = \sigma(s_n) = \sigma(b_n + x_n w_n + x_{n+1} w_{n+1} \dots + x_{n+1} w_{n+1} + x_k w_k)$$

- Ifall summan s_n överstiger noll, så blir utsignal y_n samma som summan s_n, vilket motsvarar att noden aktiveras. Annars blir utsignalen noll, vilket innebär att noden inte aktiveras, vilket framöver innebär noden att ingen justering av nodens parametrar genomförs vid optimeringen (nodens fel beräknas till noll):

$$\begin{cases} s_n > 0 \Rightarrow y_n = s_n \\ s_n \leq 0 \Rightarrow y_n = 0 \end{cases}$$

- Utsignaler y_1 samt y_2 ur respektive nod i det dolda lagret kan därmed beräknas enligt följande:

$$y_1 = \sigma(s_1) = \sigma(b_1 + x_1 w_1 + x_2 w_2)$$

samt

$$y_2 = \sigma(s_2) = \sigma(b_2 + x_1 w_3 + x_2 w_4)$$

- Genom att sätta in värden i ovanstående formler kan då utsignaler y_1 samt y_2 beräknas:

$$y_1 = \sigma(0{,}2 + 0 * 0{,}5 + 1 * 0{,}6) = \sigma(0{,}8) = 0{,}8$$

samt

$$y_2 = \sigma(0{,}4 + 0 * 0{,}3 + 1 * 0{,}8) = \sigma(1{,}2) = 1{,}2$$

- Utsignal y_3 beräknas på samma sätt, där nodens insignaler utgörs av utsignaler y_1 samt y_2 i det dolda lagret:

$$y_3 = \sigma(s_3) = \sigma(b_3 + y_1 w_5 + y_2 w_6)$$

- Genom att sätta in värden i ovanstående formel kan sedan utsignal y_3 beräknas:

$$y_3 = \sigma(0{,}6 + 0{,}8 * 0{,}1 + 1{,}2 * 0{,}9) = \sigma(1{,}76) = 1{,}76$$

- Utsignal y_3 ur noden i det yttre lagret utgör hela nätverkets enda utsignal Y:

$$Y = y_3 = 1{,}76$$

2. Backpropagation

- Beräkna aktuellt fel på varje nod utefter uppmätt avvikelse på utgången. Beräkning sker bakåt.

- Avvikelsen för respektive nods utsignal beräknas från det yttre lagret och bakåt till det dolda lagret för att sedan genomföra justering av samtliga noders parametrar (bias b samt vikter w).

- För en given nod n i det yttre lagret gäller att avvikelsen δ_n kan beräknas enligt nedan:

$$\delta_n = Y_{train,n} - y_n,$$

där δ_n är avvikelsen, $Y_{train,n}$ är referensvärdet från träningsdatan och y_n är nodens aktuella utsignal.

- Aktuellt fel e_n för en given nod n i det yttre lagret kan sedan beräknas med följande formel:

$$e_n = \delta_n * \sigma'(y_n),$$

där δ_n är avvikelsen och $\sigma_n'(x)$ är derivatan av nodens utsignal y_n.

- Derivatan $\sigma_n'(x)$ av nodens utsignal y_n är lika med 1 för samtliga aktiverade noder (samtliga noder vars utsignal y_n överstiger noll). För inaktiverade noder gäller i stället att derivatan $\sigma_n'(x)$ av nodens utsignal y_n är lika med 0:

$$\begin{cases} y_n > 0 => \sigma'(y_n) = 1 \\ y_n \leq 0 => \sigma'(y_n) = 0 \end{cases}$$

- Därmed gäller att för aktiverade noder så beräknas aktuellt fel e_n till uppmätt avvikelse δ_n, samtidigt som aktuellt fel för inaktiverade noder beräknas till 0:

$$\begin{cases} y_n > 0 => \delta_n * 1 = \delta_n \\ y_n \leq 0 => \delta_n * 0 = 0 \end{cases}$$

- Eftersom beräknat fel är proportionerligt med hur mycket som nodens parametrar justeras vid optimering, så justeras inte inaktiverade noders parametrar överhuvudtaget. Detta är också logiskt, då inaktiverade noder inte har bidraget med data till efterföljande noder och därmed inte har orsakat eventuell avvikelse.

- Avvikelsen δ_3 för noden det yttre lagret kan beräknas direkt via referensvärdet Y till -0,76, då

$$\delta_3 = Y_{train} - y_3 = 1 - 1{,}76 = -0{,}76$$

- Eftersom denna nod är aktiverad, så beräknas felet e_3 också till -0,76, eftersom

$$e_3 = \delta_3 * \sigma'(y_3) = -0{,}76 * 1 = -0{,}76$$

- För en given nod n i ett det dolda lagret gäller att avvikelsen δ_n för k antal noder i efterföljande lager kan beräknas enligt nedan:

$$\delta_n = e_0 w_0 + e_1 w_1 + e_2 w_2 \ldots + e_k w_k,$$

där δ_n är avvikelsen, e_k är beräknas fel för varje nod i efterföljande lager och w_k vikten mellan de två noderna.

- Eftersom aktuell utsignal y_n ur en given nod i aktuellt lager multipliceras med en vikt w_n i nästa lager, så gäller att ju lägre vikten w_n är, desto mindre beror aktuell avvikelse på utsignal y_n från föregående nod. Därmed beräknas avvikelsen för denna nod vara lägre, vilket innebär att dess parametrar justeras mindre.

- Avvikelser δ_1 samt δ_2 i det dolda lagret kan därmed beräknas enligt nedan:

$$\delta_1 = e_3 w_5 = -0{,}76 * 0{,}1 = -0{,}076$$

samt

$$\delta_2 = e_3 w_6 = -0{,}76 * 0{,}9 = -0{,}684$$

- Eftersom vikten w_6 ansluten till den nedre noden är avsevärt högre än motsvarande vikt w_5 ansluten till den nedre noden, så beror felet e_3 på noden i det yttre lagret avsevärt mer på fel i den nedre noden. Därmed blir det beräknade felet också högre, vilket också kommer medföra att dess parametrar kommer justeras avsevärt mer, i detta fall nedåt för att sänka utsignal y_3.

- Eftersom båda noder i det dolda lagret är aktiverade (utsignaler y_1 samt y_2 överstiger 0), så beräknas fel e_1 samt e_2 för respektive nod vara samma som uppmätt avvikelse:

$$e_1 = \delta_1 * \sigma'(y_1) = -0{,}076 * 1 = -0{,}076$$

$$e_2 = \delta_2 * \sigma'(y_2) = -0{,}684 * 1 = -0{,}684$$

3. Optimering

- Justera bias samt vikter på noderna i det yttre samt dolda lagret. Optimeringen sker bakåt.

- För en given nod n i ett givet dolt eller yttre lager gäller att förändringsfaktor Δc_n som ska användas för nodens bias samt vikter kan beräknas med följande formel:

$$\Delta c_n = e_n * LR,$$

där e_n är beräknat fel för aktuell nod och LR är lärhastigheten *(learning rate)*. Via lärhastigheten justeras därmed parametrarna med en bråkdel av felet, vilket för många epoker möjliggör finjustering för ett mycket lågt fel.

- Nodens bias b_n ska ökas med förändringshastigheten Δc_n:

$$b_n = b_n + \Delta c_n = b_n + e_n * LR$$

- För respektive vikt w_k ansluten mellan aktuell nod n samt nod k i föregående lager gäller att denna ska ökas enligt nedan:

$$w_k = w_k + \Delta c_n * y_k,$$

där y_k är utsignalen från nod k i föregående lager.

- Förändringsfaktor Δc_3 för noden i det yttre lagret kan därmed beräknas enligt nedan:

$$\Delta c_3 = e_3 * LR$$

- För en lärhastighet LR på 0.1 kan därmed förändringsfaktor Δc_3 beräknas enligt nedan:

$$\Delta c_3 = -0{,}76 * 0{,}1 = -0{,}076$$

- Nodens bias b_3 ökas sedan med beräknad förändringshastighet Δc_3. Eftersom nätverket predikterade för högt, så har samtliga avvikelser samt fel beräknats till negativa värden, vilket i praktiken medför att bias samt vikterna i detta fall kommer förminskas:

$$b_3 = b_3 + \Delta c_3 = 0{,}6 + (-0{,}076) = 0{,}524$$

- Vikterna justeras sedan utefter ansluten insignal y_1 respektive y_2:

$$w_5 = w_5 + \Delta c_3 * y_1 = 0{,}1 + (-0{,}076) * 0{,}8 = 0{,}0392$$

$$w_6 = w_6 + \Delta c_3 * y_2 = 0{,}9 + (-0{,}076) * 1{,}2 = 0{,}8088$$

- Förändringsfaktor Δc_1 samt Δc_2 för noderna i det dolda lagret beräknas enligt nedan:

$$\Delta c_1 = e_1 * LR = -0{,}076 * 0{,}1 = -0{,}0076$$

$$\Delta c_2 = e_2 * LR = -0{,}684 * 0{,}1 = -0{,}0684$$

- Den övre nodens bias b_1 samt vikter w_1 samt w_2 justeras sedan enligt nedan:

$$b_1 = b_1 + \Delta c_1 = 0{,}2 + (-0{,}0076) = 0{,}1924$$

$$w_1 = w_1 + \Delta c_1 * x_1 = 0{,}5 + (-0{,}0076) * 0 = 0{,}5$$

$$w_2 = w_2 + \Delta c_1 * x_2 = 0{,}6 + (-0{,}0076) * 1 = 0{,}5924$$

- Notera att vikten w_1 inte förändrades alls, då ansluten signal x_1 är lika med 0. Därmed har denna vikt inte bidragit till aktuellt fel och justeras därmed inte.

- Likartad justering genomförs för bias b_2 samt vikter w_1 samt w_2 för den nedre noden i det dolda lagret:

$$b_2 = b_2 + \Delta c_2 = 0{,}4 + (-0{,}0684) = 0{,}3316$$

$$w_3 = w_3 + \Delta c_2 * x_1 = 0{,}3 + (-0{,}0684) * 0 = 0{,}3$$

$$w_4 = w_4 + \Delta c_2 * x_2 = 0{,}8 + (-0{,}0684) * 1 = 0{,}7316$$

- Notera även här att vikten w_3 inte förändrades alls, då ansluten signal x_1 är lika med 0. Därmed har inte heller denna vikt bidragit till aktuellt fel och justeras därmed inte.

4. Kontroll av resultat (via feedforward)

- Nu utsignal Y = y_3 beräknas via en ny feedforward, med samma insignaler X_1X_2 = 01.

- Utsignal x_1 och x_2 ur respektive nod i ingångslagret är samma som nätverkets insignaler X_1 respektive X_2:

$$x_1 = X_1 = 0$$

$$x_2 = X_2 = 1$$

- Utsignaler y_1 samt y_2 ur respektive nod i de dolda lagren beräknas sedan med de nya parametrarna enligt nedan:

$$y_1 = \sigma(b_1 + x_1 w_1 + x_2 w_2) = \sigma(0{,}1924 + 0 * 0{,}5 + 1 * 0{,}5924) = \sigma(0{,}7848) = 0{,}7848$$

samt

$$y_2 = \sigma(b_2 + x_1 w_3 + x_2 w_4) = \sigma(0{,}3316 + 0 * 0{,}3 + 1 * 0{,}7316) = \sigma(1{,}0632) = 1{,}0632$$

- Utsignal y_3 ur noden i utgångslagret beräknas på samma sätt, där nodens insignaler utgörs av utsignaler y_1 samt y_2 i det dolda lagret:

$$y_3 = \sigma(b_3 + y_1 w_5 + y_2 w_6) = \sigma(0{,}524 + 0{,}7848 * 0{,}0392 + 1{,}0632 * 0{,}8088) = \sigma(1{,}41) = 1{,}41$$

- Utsignal y_3 ur noden i det yttre lagret utgör hela nätverkets enda utsignal Y:

$$Y = y_3 \approx 1{,}41$$

Referensvärdet Y_{train} från träningsdatan är lika med 1, vilket medför en avvikelse δ_3 runt -0,41, då

$$\delta_3 = Y_{train} - y_3 \approx 1 - 1{,}41 = -0{,}41$$

- Notera att avvikelsen är avsevärt lägre än förut, då utsignal y_3 tidigare beräknades till 1,76, vilket innebar en avvikelse på -0,76. Via ytterligare träningsomgångar hade avvikelsen kunnat hamnar mycket nära noll.

- Dock vore en lägre lärhastighet vara lämpligare, såsom 0,01, vilket möjliggör bättre finjustering. I detta fall förminskades avvikelsen avsevärt, men samtidigt medför detta att det kan vara svårt att finjustera när felet minskat närmaste noll. I kombination via ett lämpligt antal epoker hade sedan avvikelsen kunnat hamna mycket nära noll.

- I detta exempel minskade som sagt avvikelsen i utsignalen, vilket är positivt. Däremot om avvikelsen i stället hade ökat, fast åt andra hållet (så att nätverket predikterade alldeles för lågt, såsom att Y = y_3 = 0,2), så är lärhastigheten för hög, vilket kommer medföra ökad avvikelse vid träning, där nätverket kontinuerligt justeras så att prediktionerna kommer alternera mellan för högt och för lågt i växande takt.

Träning av ett neuralt nätverk för hand – Exempel 2

- Det neurala nätverket i figuren nedan ska tränas till att kunna prediktera så att utsignaler Y_1Y_2 utgör inversen till insignaler X_1X_2. För att träna nätverket ska de fyra träningsuppsättningarna i nedanstående tabell användas:

X_1X_2	Y_1Y_2
00	11
01	10
10	01
11	00

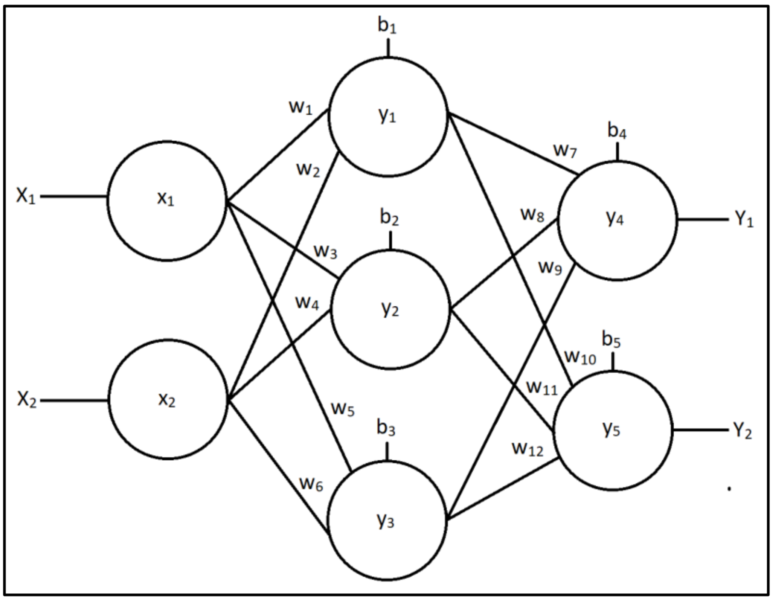
Tabell 1 – Träningsuppsättningar för det neurala nätverket.

- Antag att träning nu ska genomföras och den första träningsuppsättningen som nätverket utsätts för innehar insignaler $X_1X_2 = 10$ samt $Y_1Y_2 = 01$.

- Nodernas parametrar har vid start har randomiserats till följande:

Dolda lagret:
$b_1 = 0.1$, $b_2 = 0.1$, $b_3 = 0.7$
$w_1 = 0.2$, $w_2 = 0.9$, $w_3 = 0.3$
$w_4 = 0.8$, $w_5 = 0.5$, $w_6 = 0.4$

Utgångslagret:
$b_4 = 0.1$, $b_5 = 0.6$
$w_7 = 0.1$, $w_8 = 0.1$, $w_9 = 1.0$
$w_{10} = 0.3$, $w_{11} = 0.0$, $w_{12} = 0.6$



Figur 2 – Litet neuralt nätverk.

- Antag att lärhastigheten LR är satt till 0.1 och att aktiveringsfunktionen ReLU används för alla noder.
- Genomför feedforward, backpropagation samt optimering.
- Beräkna sedan utsignalen igen. Blev felet mindre? Om inte, vad hade du kunnat ändra för att erhålla mindre fel?

Lösning – Exempel 2

- Träning sker i tre steg, feedforward backpropagation samt optimering för att beräkna nya utsignaler, beräkna aktuellt fel via jämförelse med träningsdatan samt justera nätverkets parametrar i syfte att minska aktuellt fel.

1. Feedforward

- Beräkna utsignal y_n från varje nod. Beräkning sker framåt.

- Utsignal x_1 och x_2 ur respektive nod i ingångslagret är samma som nätverkets insignaler X_1 respektive X_2:

$$\begin{cases} x_1 = X_1 = 1 \\ x_2 = X_2 = 0 \end{cases}$$

- För en given nod n i ett dolt eller yttre lager gäller att utsignal y_n är lika med summan s_n av nodens bias b_n samt bidraget $x_n w_n$ från respektive insignal, filtrerat genom aktiveringsfunktionen ReLU. För k antal noder gäller därmed att

$$y_n = \sigma(s_n) = \sigma(b_n + x_n w_n + x_{n+1} w_{n+1} \dots + x_{n+1} w_{n+1} + x_k w_k)$$

- Ifall summan s_n överstiger noll, så blir utsignal y_n samma som summan s_n, vilket motsvarar att noden aktiveras. Annars blir utsignalen noll, vilket innebär att noden inte aktiveras, vilket framöver innebär noden att ingen justering av nodens parametrar genomförs vid optimeringen (nodens fel beräknas till noll):

$$\begin{cases} s_n > 0 => y_n = s_n \\ s_n \leq 0 => y_n = 0 \end{cases}$$

- Utsignaler y_1, y_2 samt y_3 ur respektive nod i det dolda lagret kan därmed beräknas enligt följande:

$$y_1 = \sigma(s_1) = \sigma(b_1 + x_1 w_1 + x_2 w_2),$$

$$y_2 = \sigma(s_2) = \sigma(b_2 + x_1 w_3 + x_2 w_4)$$

samt

$$y_3 = \sigma(s_2) = \sigma(b_3 + x_1 w_5 + x_2 w_6)$$

- Genom att sätta in värden i ovanstående formler kan då utsignaler y_1 samt y_2 beräknas:

$$y_1 = \sigma(0{,}1 + 1 * 0{,}2 + 0 * 0{,}9) = \sigma(0{,}3) = 0{,}3$$

$$y_2 = \sigma(0{,}1 + 1 * 0{,}3 + 0 * 0{,}8) = \sigma(0{,}4) = 0{,}4$$

$$y_3 = \sigma(0{,}1 + 1 * 0{,}5 + 0 * 0{,}4) = \sigma(0{,}6) = 0{,}6$$

- Utsignaler y_4 samt y_5 beräknas på samma sätt, där nodernas insignaler utgörs av utsignaler y_1, y_2 samt y_3 i det dolda lagret:

$$y_4 = \sigma(s_4) = \sigma(b_4 + y_1 w_7 + y_2 w_8 + y_3 w_9)$$

samt

$$y_5 = \sigma(s_5) = \sigma(b_5 + y_1 w_{10} + y_2 w_{11} + y_3 w_{12})$$

- Genom att sätta in värden i ovanstående formel kan sedan utsignal y_3 beräknas:

$$y_4 = \sigma(s_4) = \sigma(0{,}1 + 0{,}1 * 0{,}3 + 0{,}1 * 0{,}4 + 1{,}0 * 0{,}6) = \sigma(1{,}73) = 1{,}73$$

$$y_5 = \sigma(s_5) = \sigma(0{,}6 + 0{,}1 * 0{,}3 + 0{,}1 * 0{,}0 + 0{,}1 * 0{,}6) = \sigma(1{,}41) = 1{,}41$$

- Utsignaler y_5 samt y_6 ur noderna i det yttre lagret utgör hela nätverkets utsignaler Y_1 samt Y_2:

$$\begin{cases} Y_1 = y_4 = 1{,}73 \\ Y_2 = y_5 = 1{,}41 \end{cases}$$

2. Backpropagation

- Beräkna aktuellt fel på varje nod utefter uppmätt avvikelse på utgången. Beräkning sker bakåt.

- Avvikelsen för respektive nods utsignal beräknas från det yttre lagret och bakåt till det dolda lagret för att sedan genomföra justering av samtliga noders parametrar (bias b samt vikter w).

- För en given nod n i det yttre lagret gäller att avvikelsen δ_n kan beräknas enligt nedan:

$$\delta_n = Y_{train,n} - y_n,$$

där δ_n är avvikelsen, $Y_{train,n}$ är referensvärdet från träningsdatan och y_n är nodens aktuella utsignal.

- Aktuellt fel e_n för en given nod n i det yttre lagret kan sedan beräknas med följande formel:

$$e_n = \delta_n * \sigma'(y_n),$$

där δ_n är avvikelsen och $\sigma_n'(x)$ är derivatan av nodens utsignal y_n.

- Derivatan $\sigma_n'(x)$ av nodens utsignal y_n är lika med 1 för samtliga aktiverade noder (samtliga noder vars utsignal y_n överstiger noll). För inaktiverade noder gäller i stället att derivatan $\sigma_n'(x)$ av nodens utsignal y_n är lika med 0:

$$\begin{cases} y_n > 0 => \sigma'(y_n) = 1 \\ y_n \leq 0 => \sigma'(y_n) = 0 \end{cases}$$

- Därmed gäller att för aktiverade noder så beräknas aktuellt fel e_n till uppmätt avvikelse δ_n, samtidigt som aktuellt fel för inaktiverade noder beräknas till 0:

$$\begin{cases} y_n > 0 => \delta_n * 1 = \delta_n \\ y_n \leq 0 => \delta_n * 0 = 0 \end{cases}$$

- Eftersom beräknat fel är proportionerligt med hur mycket som nodens parametrar justeras vid optimering, så justeras inte inaktiverade noders parametrar överhuvudtaget. Detta är också logiskt, då inaktiverade noder inte har bidraget med data till efterföljande noder och därmed inte har orsakat eventuell avvikelse.

- Avvikelsen δ_4 samt δ_5 för noderna i det yttre lagret kan beräknas direkt via referensvärdet $Y_{train,1}$ samt $Y_{train,2}$:

$$\delta_4 = Y_{train,1} - y_4 = 0 - 1{,}73 = -1{,}73$$

$$\delta_5 = Y_{train,2} - y_5 = 1 - 1{,}41 = -0{,}41$$

- Eftersom båda noder är aktiverade, så beräknas motsvarande fel e_4 samt e_5 till att vara ekvivalent med avvikelsen, då

$$e_4 = \delta_4 * \sigma'(y_4) = -1{,}73 * 1 = -1{,}73$$

$$e_5 = \delta_5 * \sigma'(y_5) = -0{,}41 * 1 = -0{,}41$$

- För en given nod n i ett det dolda lagret gäller att avvikelsen δ_n för k antal noder i efterföljande lager kan beräknas enligt nedan:

$$\delta_n = e_0w_0 + e_1w_1 + e_2w_2 \dots + e_kw_k,$$

där δ_n är avvikelsen, e_k är beräknas fel för varje nod i efterföljande lager och w_k vikten mellan de två noderna.

- Eftersom aktuell utsignal y_n ur en given nod i aktuellt lager multipliceras med en vikt w_n i nästa lager, så gäller att ju lägre vikten w_n är, desto mindre beror aktuell avvikelse på utsignal y_n från föregående nod. Därmed beräknas avvikelsen för denna nod vara lägre, vilket innebär att dess parametrar justeras mindre.

- Avvikelser δ_1, δ_2 samt δ_3 i det dolda lagret kan därmed beräknas enligt nedan:

$$\delta_1 = e_4w_7 + e_5w_{10} = -1{,}73 * 0{,}1 + (-0{,}41) * 0{,}3 = -0{,}246$$

$$\delta_2 = e_4w_8 + e_5w_{11} = -1{,}73 * 0{,}1 + (-0{,}41) * 0{,}0 = -0{,}173$$

$$\delta_3 = e_4w_9 + e_5w_{12} = -1{,}73 * 1{,}0 + (-0{,}41) * 0{,}6 = -1{,}976$$

- Beloppet av en given vikt är proportionerligt med aktuell avvikelse, då ansluten insignal medför ett större bidrag till ansluten nod i nästa lager. Därmed har ansluten vikt högre betydelse för aktuell utsignal och därmed aktuellt fel. Därmed tas ansluten vikt i åtanke vid beräkningen.

- Eftersom samtliga noder i det dolda lagret är aktiverade (utsignaler y_1, y_2 samt y_3 överstiger 0), så beräknas felet e_1, e_2 samt e_3 för respektive nod vara samma som uppmätt avvikelse:

$$e_1 = \delta_1 * \sigma'(y_1) = -0{,}246 * 1 = -0{,}246$$

$$e_2 = \delta_2 * \sigma'(y_2) = -0{,}173 * 1 = -0{,}173$$

$$e_3 = \delta_3 * \sigma'(y_3) = -1{,}976 * 1 = -1{,}976$$

3. Optimering:

- Justera bias samt vikter på noderna i det yttre samt dolda lagret. Optimeringen sker bakåt.

- För en given nod n i ett givet dolt eller yttre lager gäller att förändringsfaktor Δc_n som ska användas för nodens bias samt vikter kan beräknas med följande formel:

$$\Delta c_n = e_n * LR,$$

där e_n är beräknat fel för aktuell nod och LR är lärhastigheten *(learning rate)*. Via lärhastigheten justeras därmed parametrarna med en bråkdel av felet, vilket för många epoker möjliggör finjustering för ett mycket lågt fel.

- Nodens bias b_n ska ökas med förändringshastigheten Δc_n:

$$b_n = b_n + \Delta c_n = b_n + e_n * LR$$

- För respektive vikt w_k ansluten mellan aktuell nod n samt nod k i föregående lager gäller att denna ska ökas enligt nedan:

$$w_k = w_k + \Delta c_n * y_k,$$

där y_k är utsignalen från nod k i föregående lager.

- Förändringsfaktor Δc_4 samt Δc_5 för noderna i det yttre lagret kan därmed beräknas enligt nedan:

$$\Delta c_4 = e_4 * LR$$

$$\Delta c_5 = e_5 * LR$$

- För en lärhastighet LR på 0.1 kan därmed förändringsfaktorer Δc_4 samt Δc_5 beräknas enligt nedan:

$$\Delta c_4 = -1{,}73 * 0.1 = -0{,}173$$

$$\Delta c_5 = -0{,}41 * 0.1 = -0{,}041$$

- Nodernas respektive bias b_4 samt b_5 ökas sedan med motsvarande beräknad förändringshastighet Δc_5 respektive Δc_5. Eftersom nätverket predikterade för högt, så har samtliga avvikelser och därmed fel beräknats till negativa värden, vilket i praktiken medför att bias samt vikterna i detta fall kommer reduceras:

$$b_4 = b_4 + \Delta c_4 = 0{,}1 + (-0{,}174) = -0{,}074$$

$$b_5 = b_5 + \Delta c_5 = 0{,}6 + (-0{,}041) = 0{,}559$$

- Vikterna justeras sedan utefter anslutna insignaler y_1, y_2 samt y_3:

$$w_7 = w_7 + \Delta c_4 * y_1 = 0{,}1 + (-0{,}174) * 0{,}3 = 0{,}0478$$

$$w_8 = w_8 + \Delta c_4 * y_2 = 0{,}1 + (-0{,}174) * 0{,}4 = 0{,}0304$$

$$w_9 = w_9 + \Delta c_4 * y_3 = 1{,}0 + (-0{,}174) * 0{,}6 = 0{,}8956$$

$$w_{10} = w_{10} + \Delta c_5 * y_1 = 0{,}3 + (-0{,}041) * 0{,}3 = 0{,}2877$$

$$w_{11} = w_{11} + \Delta c_5 * y_2 = 0{,}0 + (-0{,}041) * 0{,}4 = -0{,}0164$$

$$w_{12} = w_{12} + \Delta c_5 * y_3 = 0{,}6 + (-0{,}041) * 0{,}6 = 0{,}5754$$

- Förändringsfaktorer Δc_1, Δc_2 samt Δc_3 för noderna i det dolda lagret beräknas enligt nedan:

$$\Delta c_1 = e_1 * LR = -0{,}246 * 0{,}1 = -0{,}0246$$

$$\Delta c_2 = e_2 * LR = -0{,}173 * 0{,}1 = -0{,}0173$$

$$\Delta c_3 = e_3 * LR = -1{,}976 * 0{,}1 = -0{,}1976$$

- Bias b_1 – b_3 för respektive nod i det dolda lagret justeras sedan enligt nedan:

$$b_1 = b_1 + \Delta c_1 = 0{,}1 + (-0{,}0246) = 0{,}0754$$

$$b_2 = b_2 + \Delta c_2 = 0{,}1 + (-0{,}0173) = 0{,}0827$$

$$b_3 = b_3 + \Delta c_3 = 0{,}7 + (-0{,}1976) = 0{,}5024$$

- Vikter $b_1 - b_6$ för respektive nod i det dolda lagret justeras sedan enligt nedan:

$$w_1 = w_1 + \Delta c_1 * x_1 = 0{,}2 + (-0{,}0246) * 1 = 0{,}1754$$

$$w_2 = w_2 + \Delta c_1 * x_2 = 0{,}9 + (-0{,}0246) * 0 = 0{,}9$$

$$w_3 = w_3 + \Delta c_2 * x_1 = 0{,}3 + (-0{,}0173) * 1 = 0{,}2827$$

$$w_4 = w_4 + \Delta c_2 * x_2 = 0{,}8 + (-0{,}0173) * 0 = 0{,}8$$

$$w_5 = w_5 + \Delta c_3 * x_1 = 0{,}5 + (-0{,}1976) * 1 = 0{,}3024$$

$$w_6 = w_6 + \Delta c_3 * x_2 = 0{,}4 + (-0{,}1976) * 0 = 0{,}2024$$

4. Kontroll av resultat (via feedforward):

- Nu utsignal $Y = y_3$ beräknas via en ny feedforward, med samma insignaler $X_1X_2 = 10$.

- Utsignal x_1 och x_2 ur respektive nod i ingångslagret är samma som nätverkets insignaler X_1 respektive X_2:

$$x_1 = X_1 = 1$$

$$x_2 = X_2 = 0$$

- Utsignaler $y_1 - y_3$ ur respektive nod i det dolda lagret beräknas sedan med de nya parametrarna enligt nedan:

$$y_1 = \sigma(b_1 + x_1w_1 + x_2w_2) = \sigma(0{,}0754 + 1 * 0{,}1754 + 0 * 0{,}9) = \sigma(0{,}2508) = 0{,}2508$$

$$y_2 = \sigma(b_2 + x_1w_3 + x_2w_4) = \sigma(0{,}0827 + 1 * 0{,}2827 + 0 * 0{,}8) = \sigma(0{,}3654) = 0{,}3654$$

$$y_3 = \sigma(b_3 + x_1w_5 + x_2w_6) = \sigma(0{,}5024 + 1 * 0{,}3024 + 0 * 0{,}2024) = \sigma(0{,}8048) = 0{,}8048$$

- Utsignal $y_4 - y_5$ ur noderna i utgångslagret beräknas på samma sätt, där nodernas respektive insignaler utgörs av utsignaler y_1, y_2 samt y_3 i det dolda lagret:

$$y_4 = \sigma(b_4 + y_1w_7 + y_2w_8 + y_3w_9) = \sigma(0{,}074 + 0{,}2508 * 0{,}0478 + 0{,}3654 * 0{,}0304 + 0{,}8040 * 0{,}8956),$$

vilket är ekvivalent med att

$$y_4 \approx \sigma(0{,}82) = 0{,}82$$

$$y_4 = \sigma(b_5 + y_1w_{10} + y_2w_{11} + y_3w_{12}) = \sigma(0{,}559 + 0{,}2508 * 0{,}2877 + 0{,}3654 * (-0{,}0164) + 0{,}8048 * 0{,}5754),$$

vilket är ekvivalent med att

$$y_5 \approx \sigma(1{,}09) = 1{,}09$$

- Utsignal y_4 samt y_5 ur noderna i det yttre lagret utgör nätverkets utsignaler $Y_1 - Y_2$:

$$\begin{cases} Y_1 = y_4 \approx 0{,}83 \\ Y_2 = y_5 \approx 1{,}09 \end{cases}$$

- Avvikelser δ_4 samt δ_5 för noderna i det yttre lagret kan beräknas direkt via referensvärdet $Y_{train,1}$ samt $Y_{train,2}$:

$$\delta_4 = Y_{train,1} - y_4 \approx 0 - 0{,}82 = -0{,}82$$

$$\delta_5 = Y_{train,2} - y_5 \approx 1 - 1{,}09 = -0{,}09$$

Maskininlärning

- Notera att avvikelserna är avsevärt lägre än förut, då utsignaler y_4 samt y_5 tidigare beräknades till 1,73 respektive 1,41, vilket innebar en avvikelse på -1,73 respektive -0,41. Via ytterligare träningsomgångar hade avvikelsen kunnat hamna mycket nära noll.

- Dock vore en lägre lärhastighet vara lämpligare, såsom 0,01, vilket möjliggör bättre finjustering. I detta fall förminskades avvikelsen avsevärt, men samtidigt medför detta att det kan vara svårt att finjustera när felet minskat närmaste noll. I kombination via ett lämpligt antal epoker hade sedan avvikelsen kunnat hamna mycket nära noll.