ImageRecognition toteutus

Simo Korkolainen

27. kesäkuuta 2016

Projektin tarkoituksena on tehdä ohjelma, joka opettaa neuroverkon tunnistamaan kuvia backpropagation-algoritmin avulla. Neuroverkon opetuksessa verkon painoja muutetaan liikuttamalla niitä virhefunktion gradientin vastaiseen suuntaan, kunnes virhefunktio on minimoitunut ja neuroverkko on oppinut tunnistamaan kuvat. Derivoinnin ketjusääntöön perustuva backpropagation-algoritmi mahdollistaa gradientin nopean laskemisen.

1 Neuroverkko

Neuroverkko koostuu kerroksista, joissa on neuroneita. Olkoon L neuroverkon kerroksien lukumäärä. Olkoon l_k kerroksen $k = 1, ..., L$ neuronien lukumäärä. Merkitään kerroksen k aktivaatiota vektorina $z_k \in \mathbb{R}^{l_k}$. Ensimmäisen kerroksen aktivaatio z_1 on neuroverkon syöte ja viimeisen kerroksen aktivaation z_L on neuroverkon antama tuloste.

Jokaisen kerroksen $k > 1$ aktivaation voidaan ajatella laskettavan parametrisoidun funktion $f_k : \mathbb{R}^{l_{k-1}} \times A_k \to \mathbb{R}^{l_k}$ avulla. Tässä A_k on verkon kerroksien $k-1$ ja k yhteyksien painoina toimivien parametrien joukko. Kerroksen aktivaatio lasketaan rekursiivisesti kaavan

$$z_k = f(z_{k-1}, a_k)$$

avulla, missä $a_k \in A_k$. Parametrit $a_1, ..., a_L$ on tarkoitus oppia backpropagation-algoritmia käyttäen.

2 Aineisto ja kokonaisvirhe

Olkoon $x_1, x_2, \ldots, x_n \in \mathbb{R}^{l_1}$ neuroverkon opetussyötteitä ja $y_1, y_2, \ldots, y_n \in \mathbb{R}^{l_L}$ syötteitä vastaavia tavoitetuloksia. Olkoon $E : \mathbb{R}^{l_L} \times \mathbb{R}^{l_L} \to \mathbb{R}$ virhefunktio. Merkitään syötteen x_j aiheuttamaa aktivaatiota kerroksessa k merkinnällä z_k^j, jolloin syötekerroksessa pätee $z_1^j = x_j$. Määritellään neuroverkon kokonaisvirhe kaavalla

$$E_{tot} = \sum_{j=1}^{n} E(z_L^j, y_j).$$

Backprobagation-algoritmin tarkoituksena on valita parametrien $a_1, ..., a_L$ arvot siten, että kokonaisvirhe E_{tot} minimoituu.

Jos neuroverkkoa käytetään syötteiden luokittelemiseen erillisiin luokkiin $1, ..., S = l_L$, viimeisen kerroksen $z_L = (z_{L1}, \ldots, z_{LS})$ aktivaation z_{Lm} tulkitaan tarkoittavan todennäköisyyttä, että syöte x_j kuuluu luokkaan m. Halutuksi tulokseksi valitaan $y_{jm} = 1$, kun x_j kuuluu luokkaan m ja $y_{jm} = 0$ muulloin. Luokittelun tapauksessa on luontevaa käyttää viimeisessä kerroksessa softmax-funktiota $f_L : \mathbb{R}^S \times A_L \to (0,1)^S$, missä

$$f_{Lm}(z_{L-1}) = \frac{e^{\sum_{i=1}^{l_{L-1}} a_{mi} z_{(L-1)i}}}{\sum_{k=1}^{S} e^{\sum_{i=1}^{l_{L-1}} a_{ki} z_{(L-1)i}}}.$$

Softmax-funktion kuva-alkion komponenttien summa on yksi.

Luokittelussa virhefunktiona käytettään yleensä logloss-virhettä. Tässä tapauksessa virhefunktio määritellään kaikilla $s \in \{0,1\}^S$ ja $t \in (0,1)^S$ kaavalla

$$E(s,t) = \sum_{m=1}^{S} s_m \log(t_m).$$

3 Derivaatta ja ketjusääntö

Tässä kappaleessä käytetään yleisiä differentiaalilaskennan merkintöjä aiemmista merkinnöistä huolimatta. Olkoon funktio $f = (f_1, \ldots, f_n) : \mathbb{R}^m \to \mathbb{R}^n$ derivoituva. Derivaatalla pisteessä $x_0 \in \mathbb{R}^m$ tarkoitetaan matriisia

$$f'(x_0) = \begin{bmatrix} \frac{\partial f_1(x_0)}{\partial x_1} & \cdots & \frac{\partial f_1(x_0)}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(x_0)}{\partial x_1} & \cdots & \frac{\partial f_n(x_0)}{\partial x_m} \end{bmatrix},$$

missä osittaisderivaatat $\frac{\partial f_i(x_0)}{\partial x_j}$ on määritelty kaavalla

$$\frac{\partial f_i(x_0)}{\partial x_j} = \lim_{h \to 0} \frac{f_i(x_0 + h e_j) - f_i(x_0)}{h}.$$

Olkoon lisäksi funktion $g : \mathbb{R}^n \to \mathbb{R}^p$ derivoituva. Tällöin yhdistetty kuvaus $g \circ f : \mathbb{R}^m \to \mathbb{R}^p$ on myös derivoituva ja yhdistetyn kuvauksen derivaatan pisteessä x_0 on mahdollista laskea kaavalla

$$(g \circ f)'(x_0) = g'(f(x_0))g'(x_0).$$

4 Backpropagation-algoritmi

Backpropagation algoritmi perustuu edellä mainittuun differentiaalilaskennan ketjusääntöön. Merkitään

$$\frac{\partial z_{i+1}}{\partial z_i} = \begin{bmatrix} \frac{\partial z_{(i+1)1}}{\partial z_{i1}} & \cdots & \frac{\partial z_{(i+1)1}}{\partial z_{il_i}} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_{(i+1)l_{i+1}}}{\partial z_{i1}} & \cdots & \frac{\partial z_{(i+1)l_{i+1}}}{\partial z_{il_i}} \end{bmatrix},$$

$$\frac{\partial z_i}{\partial a_i} = \begin{bmatrix} \frac{\partial z_{i1}}{\partial a_i} \\ \vdots \\ \frac{\partial z_{il_{i+1}}}{\partial a_i} \end{bmatrix}, \text{ ja}$$

$$\frac{\partial E_{tot}}{\partial z_i} = \begin{bmatrix} \frac{\partial E_{tot}}{\partial z_{i1}} & \cdots & \frac{\partial E_{tot}}{\partial z_{il_i}} \end{bmatrix}.$$

Ketjusäännön perusteella eri kerrosten aktivaatioiden virhegradientti voidaan laskea rekursiivisesti

$$\frac{\partial E_{tot}}{\partial z_i} = \frac{\partial E_{tot}}{\partial z_{i+1}} \frac{\partial z_{i+1}}{\partial z_i}$$

Eli, jos tiedämme ylemmän kerroksen virhegradientin $\frac{\partial E_{tot}}{\partial z_{i+1}}$ arvon, voimme laskea alemman kerroksen virhegradientin kertomalla gradienttia $\frac{\partial E_{tot}}{\partial z_{i+1}}$ oikealta matriisilla $\frac{\partial z_{i+1}}{\partial z_i}$. Parametrien virhegradientti voidaan laskea vastaavasti käyttäen aktivaatioiden virhegradienttia

$$\frac{\partial E_{tot}}{\partial a_i} = \frac{\partial E_{tot}}{\partial z_i} \frac{\partial z_i}{\partial a_i}$$

Seuraavaksi on esitettynä pseudokoodi backpropagation-algoritmille.

Algorithm 1 Backpropagation-algoritmi

procedure BACKPROPAGATION(x, y, α)

  for $j = 1, \ldots, n$ **do**

    FORWARD PROPAGATION(x_j)
    BACKWARD PROPAGATION(y_j, α)

  end for

end procedure

procedure FORWARD PROPAGATION(x)

  Aseta syöte $z_1 = x$.
  for jokaiselle kerrokselle $k = 2, \ldots, L$ **do**
    Laske aktivaatio $z_k := f_k(z_{k-1}, a_k)$
  end for

end procedure

procedure BACKWARD PROPAGATION(y, α)

  Laske virhegradientti $\frac{\partial E}{\partial z_L}$ arvojen y ja z_L avulla
  for jokaiselle kerrokselle $k = L - 1, \ldots, 2$ **do**

    Laske ensiksi $\frac{\partial z_{k+1}}{\partial z_k}$ arvon z_k avulla
    Laske toiseksi $\frac{\partial E}{\partial z_k} := \frac{\partial E}{\partial z_{k+1}} \frac{\partial z_{k+1}}{\partial z_k}$
    Laske kolmanneksi $\frac{\partial E_{tot}}{\partial a_i} := \frac{\partial E_{tot}}{\partial z_i} \frac{\partial z_i}{\partial a_i}$

    Päivitä painot $a_i := a_i - \alpha \frac{\partial E_{tot}}{\partial a_i}$

  end for

end procedure

5 Aikavaativuus

Tarkastellaan seuraavaksi Backpropagation-algoritmin aikavaativuutta. Neuroverkoon liittyvät aikavaativuudet riippuvat paljon neuroverkon rakenteesta. Tarkastellaan ensiksi eteenpäinvirtauksen, eli neuroverkon aktivaatioden laskemisen aikavaativuutta. Ohjelmassa käytetään vain eteenpäin kytkettyjä neuroverkkoja. Neuroneiden aktivaatio z_k kerroksessa k lasketaan täsmälleen edellisen kerroksen aktivaatioiden perusteella eli $z_k = f(z_{k-1}, a_k)$ missä f on aktivaatiofunktio.

Kuten aikasemmin olkoon L neuroverkon kerroksien lukumäärä ja olkoon l_k kerroksen $k = 1, ..., L$ neuronien lukumäärä. Jos jokainen kerroksen k neuroni on kytketty kaikkiin edellisen kerroksen solmuihin ja neuronipariin liittyvän laskennan aikavaativuus on luokkaa $O(1)$, yhden kerroksen k neuronin aktivaation laskemisen aikavaativuus on luokkaa $O(l_{k-1})$. Koska kerroksessa k on l_k neuronia, koko kerrokseen liittyvän laskennan aikavaativuus on $O(l_{k-1}l_k)$. Ensimmäisen kerroksen eli syötekerroksen aktivaatioden asettamisen aikavaativuus on $O(l_1)$.

Koko neuroverkon aktivaatioden laskennan aikavaativuus T_{act} on kerrosten aikavaativuuksien summa eli

$$T_{act} = O(l_1 + \sum_{k=2}^{L} l_{k-1}l_k)$$

Edellisen summan ymmärtämiseksi tarkastellaan erikoistapausta, jossa kerrosten neuronien lukumäärä pienee eksponentiaalisesti eli $l_k = \alpha^{k-1}l_1$, missä $0 < \alpha < 1$. Tällöin

$$\begin{aligned}
l_1 + \sum_{k=2}^{L} l_{k-1}l_k &= l_1 + \sum_{k=2}^{L} \alpha^{k-2}l_1\alpha^{k-1}l_0 \\
&= l_1 + l_1^2 \sum_{k=2}^{L} \alpha^{2k-3} \\
&= l_1 + l_1^2\alpha \sum_{k=0}^{L-2} (\alpha^2)^k \\
&\leq l_1 + l_1^2\alpha \sum_{k=0}^{\infty} (\alpha^2)^k \\
&= l_1 + l_1^2 \frac{\alpha}{1-\alpha^2}
\end{aligned}$$

Saamme, että $T_{act} = O(l_1^2)$, koska $\frac{\alpha}{1-\alpha^2}$ on positiivinen vakio.

Vaikka takaisinvirtaus vaikuttaa vaikeammalta, soittautuu, että takaisinvirtausvaiheen laskenta on yhtä haastastavaa kuin eteenpäinvirtauksen laskenta. Ohjelmassa jokaiseen kerroksen k neuroniin m liittyy painovektori a_{km} ja yksittäisen neuronin aktivaatio lasketaan tyypillisesti kaavan

$$z_{km} = f_{km}(a_{km}^T z_{(k-1)m})$$

avulla. Tässä kuvaus $f_{km} : \mathbb{R} \to \mathbb{R}$ on neuronin aktivaatiofunktio, joka on derivoituva. Osittaisderivaatta $\frac{\partial z_{km}}{\partial z_{(k-1)m}}$ voidaan laskea ketjusääntöä käyttäen seuraavan kaavan avulla.

$$\frac{\partial z_{km}}{\partial z_{(k-1)j}} = a_{kmj} f'_{km}(a_{km}^T z_{(k-1)m})$$

Pistetulon $a_{km}^T z_{(k-1)m}$ laskemisen aikavaativuus on $O(l_{k-1})$, koska kerroksessa $k-1$ on l_{k-1} neuronia. Nyt voisi ajatella, että vaakavektorin

$$\frac{\partial z_{km}}{\partial z_{(k-1)}} = \begin{bmatrix} \frac{\partial z_{km}}{\partial z_{(k-1)1}} & \cdots & \frac{\partial z_{km}}{\partial z_{(k-1)l_{k-1}}} \end{bmatrix}$$

laskennan aikavaativuus olisi $O(l_{k-1}^2)$, koska $O(l_{k-1})$ operaatiota toistetaan l_{k-1} kertaa. Kuitenkin huomaamme, ettei skalaaria $f'_{km}(a_{km}^T z_{(k-1)m})$ tarvitse laskea useita kertoja. Tätä ajatusta jalostaen pätee kaava

$$\frac{\partial z_{km}}{\partial z_{(k-1)}} = f'_{km}(a_{km}^T z_{(k-1)m}) a_{km}^T,$$

jossa vektorin ja skalaarin kertolaskun aikavaativuus on $O(l_{k-1})$. Yhteensä vektorin $\frac{\partial z_{km}}{\partial z_{(k-1)}}$ laskemisen aikavaativuus on $O(l_{k-1} + l_{k-1}) = O(l_{k-1})$. Matriisin $\frac{\partial z_k}{\partial z_{k-1}} \in \mathbb{R}^{l_{k-1} \times l_k}$ laskemiseen kuluu kaikkien rivien laskemiseen tarvittavien aikojen summa, joten aikavaativuus on $O(l_{k-1} l_k)$. Kaikkien kerrosten matriisien $\frac{\partial z_k}{\partial z_{k-1}}$ laskemisen aikavaatimus on

$$O(\sum_{k=2}^{L} l_{k-1} l_k).$$

Kerroksen virhegradientin $\frac{\partial E}{\partial z_{k-1}}$ on mahdollista laskea kaavalla

$$\frac{\partial E}{\partial z_{k-1}} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial z_{k-1}}.$$

Tällöin lasketaan $1 \times l_{k-1}$-vektorin ja $l_{k-1} \times l_k$-matriisin tulo, jonka aikavaativuus on $O(l_{k-1} l_k)$. Viimeisen kerroksen virhegradientti $\frac{\partial E}{\partial z_L}$ pitää laskea

erikseen. Laskennan aikavaativuus on $O(l_L)$, koska virhegradienttin laskeminen vie vakioajan jokaista vektorin $\frac{\partial E}{\partial z_L}$ komponenttia kohden. Vastaavasti kaikkien kerroksien virhegradienttien $\frac{\partial E}{\partial z_L}$ laskemisen aikavaativuus on

$$O(l_L + \sum_{k=2}^{L} l_{k-1}l_k) = O(\sum_{k=2}^{L} l_{k-1}l_k).$$

Virhegradientin parametrivektorin a_{km} suhteen pystyy laskemaan kaavalla

$$\frac{\partial E}{\partial a_{km}} = \frac{\partial E}{\partial z_{km}} \frac{\partial z_{km}}{\partial a_{km}} \quad = \frac{\partial E}{\partial z_{km}} f'_{km}(a_{km}^T z_{(k-1)m}) z_{(k-1)}^T.$$

Tämän laskennan aikavaatimukseksi tulee $O(l_{k-1})$. Kaikkien gradienttien $\frac{\partial E}{\partial a_{km}}$ laskennan aikavaativuus kerroksessa k on $O(l_{k-1}l_k)$. Koko neuroverkoon littyvien gradienttien $\frac{\partial E}{\partial a_{km}}$ laskennan aikavaativuus on

$$O(\sum_{k=2}^{L} l_{k-1}l_k).$$

Edellisten päättelyiden perusteella koko takaisinvirtauksen aikavaativuus on

$$O(3\sum_{k=2}^{L} l_{k-1}l_k) = O(\sum_{k=2}^{L} l_{k-1}l_k).$$

Kokonaisuudessaan yhden backpropagation-algoritmin iteraation aikavaativuus on myös $O(\sum_{k=2}^{L} l_{k-1}l_k)$. Kun iteraatiota on K kappaletta ja kuvia on n kappaletta, aikavaativuus on yhteensä $O(nK\sum_{k=2}^{L} l_{k-1}l_k)$.

Algoritmin aikavaativuutta on vaikeaa parantaa lisäämällä tilavaativuutta, koska aikavaativuus aiheutuu neuroverkon rakenteesta. Tilavaativuutta lisäämällä on kuitenkin mahdollista vaikuttaa aikavaativuuden vakiokertoimeen. Tämä tapahtuu tallentamalla laskennan välitulokset taulukkoihin sen sijaan, että samat laskut suoritettaisiin aina uudelleen.

6 Ohjelman toimivuus ja syötteiden vaikutus

Jos opetusdatassa on vähän kuvia ohjelma oppii tunnistamaan opetusdatan kuvat täydellisesti. Neuroverkkojen teorian perusteella neuroverkon on mahdollista saada vastaamaan mitä tahansa hyvin käyttäytyvä funktiota, kunhan neuroneja ja kerroksia on tarpeeksi. Hyvästä opetusdatan oppimistuloksesta huolimatta ohjelma ei opi tunnistamaan testidatan kuvia kovinkaan hyvin.

Jos opetusdatan kuvien määrä on suuri, opetukseen kuluu paljon aikaa. Oppimistulos opetusdatassa ei ole niin hyvä kuin siinä tapauksessa, että neuroverkko pääsee ylisovittamaan painonsa pieneen määrään opetuskuvia. Testidatan kuvia ohjelma luokittelee paremmin kuin silloin kuin opetusdatassa oli vähän kuvia.

Jos oppimisnopeutta säätelevä parametri (learning rate) valitaan väärin tulokset eivät ole kovinkaan hyviä, koska painot eivät muutu alkuperäisistä satunnaisista painoista. Jos learning rate on liian suuri painot rupeavat värähtelemään voimakkaasti. Esimerkiksi painot voivat tässä tilanteessa saada arvoja $0, -10, 100, -1000, 10000, -100000 \ldots$.

Pahimmassa tapauksessa painot muuttuvat NaN arvoisiksi, koska e^{-x} pyöristyy nollaksi, kun x on suuri. Tällöin ohjelmassa yritetään laskea jakolaskun $\frac{e^{-x}}{e^{-x}+e^{-x-1}} \approx \frac{0}{0}$ tapainen lasku, mikä ei ole määritelty. Ohjelma ei kaadu vaikka painot muuttusivat NaN arvoisiksi, mutta neuroverkko pitää alustaa uudelleen, jos haluaa käyttää kaikkea ohjelman toiminnallisuutta.

7 Parannusehdotuksia

Yksi tapa parantaa ohjelman toimintaa olisi datan esikäsittely pääkomponenttianalyysin avulla. Pääkomponenttianalyysin avulla kuvista olisi saanut valmiiksi hyviä piirteitä, jotka olisivat helpottaneet neuroverkon oppimisprosessia. Pääkomponenttianalyysin toteuttaminen on sen verran haastavaa, että siitä olisi voinut tehdä oman kurssityön.

Toinen tapa parantaa ohjelman toimintaa olisi ollut konvoluutioden lisääminen neuroverkkoon. Konvoluutiot ovat hyödyllisiä kuvantunnistuksessa, koska kuvan kaikki pikselit käyttäytyvät melko samalla tavalla riippumatta pikselin sijainnista kuvasta. Jos jostakin kohtaa kuvaa opitaan hyvä piirre, kuten reuna, tästä piirteestä olisi mahdollisesti hyötyä myös muualla. Konvoluutiot eivät nykyisyydellään sovellu ohjelman rakenteeseen ilman merkittävää laskennallista haittaa, joka vältettäisiin erilaisella ohjelman rakenteella. Ohjelmakoodissa on oletettu, että neuroverkon kerrokset ovat kokonaan kytkettyjä edelliseen kerrokseen. Lisäksi on oletettu, että painot liittyvät vain yhteen neuroniin. Nämä oletukset yksinkertaistavat differentiaalilaskennan matematiikkaa. Konvoluutiot lisättäessä molemmat oletukset jouduttaisiin hylkäämään, minkä seurauksena suuri osa ohjelmakoodia pitäisi kirjoittaa ja testata kokonaan uudelleen.