Practicum Backpropagation

Peter Kleiweg

November 1995

1 Bestanden

In dit practicum wordt gebruik gemaakt van een eenvoudige implementatie van backpropagation in C. Resultaten van experimenten zullen worden bekeken met het programma `gnuplot`. In tabel 1 staat welke bestanden aanwezig moeten zijn voor dit practicum.[1] `bex.c` is alleen nodig om `bex` te compileren als het nog niet aanwezig is. Verder wordt het niet gebruikt. Onder Unix: `bp`, `bex`, `demo`, `parity` en `par4` moeten executie-permissie hebben. `gnuplot` moet geïnstalleerd zijn.

2 Demonstratie

Geef voor een demonstratie het commando `demo` zonder parameters. Door middel van backpropagation wordt de XOR-functie geleerd.

Als je, wanneer de demonstratie afgelopen is, de volgende bestanden bekijkt kun je vrij snel doorzien hoe het een en ander werkt:

```
demo
demo.plt
221.net
xor.pat
demo_out.*
demo_plt.*
```

Het programma `demo` werkt als volgt: Eerst wordt `bp` aangeroepen. Dit is het backpropagation programma. Wanneer parameters zijn meegegeven aan `demo`, dan worden deze doorgegeven aan `bp`. Je kunt `bp` intoetsen zonder parameters om een overzicht te krijgen van de mogelijkheden. `bp` maakt enkele bestanden aan, waarvan `demo_out.out` door enkele aanroepen van het programma `bex` wordt opgesplitst in de diverse bestanden waarvan de naam begint met `demo_plt`. Tot slot wordt `gnuplot` opgestart dat de instructies uit `demo.plt` uitvoert.

1. In de rest van de tekst wordt alleen naar de Unix-namen van de bestanden verwezen.

Unix	DOS
`Makefile`	`MAKEFILE`
`bp`	`BP.EXE`
`bp.c`	`BP.C`
`bex`	`BEX.EXE`
`bex.c`	`BEX.C`
`demo`	`DEMO.BAT`
`demo.plt`	`DEMO.PLT`
`221.net`	`221.NET`
`xor.pat`	`XOR.PAT`
`parity`	`PARITY.EXE`
`parity.c`	`PARITY.C`
`par4`	`PAR4.BAT`
`par4.plt`	`PAR4.PLT`

Tabel 1: Bestanden voor backpropagation practicum

3 Het programma `bp`

`bp` heeft minimaal twee parameters nodig om te kunnen werken: De naam van een bestand dat de definitie voor het netwerk bevat, en de naam van een bestand dat de te trainen invoer/uitvoer patronen bevat. Er kan een naam opgegeven worden voor de bestanden waarin de resultaten moeten worden bewaard. Onder DOS mag deze naam geen extensie bevatten. Wanneer geen naam wordt opgegeven wordt `bp_out` genomen als default.

3.1 Definitie van het netwerk

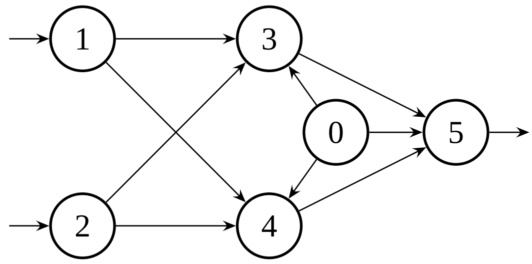
Het bestand `221.net` bevat de definitie voor het netwerk. Zo ziet het er uit:

```
# n_units n_in n_out
5 2 1

# biases
0 -> 3 4 5

# verbindingen
1 -> 3 4
2 -> 3 4
3 -> 5
4 -> 5
```

Lege regels en regels die beginnen met `#` worden genegeerd.

De eerste regel bevat het totaal aantal units, het aantal invoer-units, en het aantal uitvoer-units.

De tweede regel bevat de definitie voor de biases. Het programma maakt niet apart gebruik van biases voor units. Maar er is een unit nummer 0 waarvan



Figuur 1: Netwerk voor de XOR functie

de waarde altijd 1 is. Verbindingen die komen vanaf unit 0 fungeren daarom als bias. De keuze welke units een bias krijgen is vrij, maar invoer-units kunnen geen bias krijgen.

De overige regels geven de rest van de verbindingen aan. Figuur 1 geeft het netwerk weer dat door dit bestand wordt gedefinieerd.

Het gebruik van een netwerk-bestand laat vrij grillige netwerk-architecturen toe. De volgende beperkingen gelden:

- De invoer-units zijn genummerd vanaf 1.
- Uitvoerunits zijn de groep units in het netwerk met de hoogste nummers.
- Invoerunits kunnen geen bias hebben en geen inkomende verbindingen van andere units.
- Uitvoerunits kunnen geen uitgaande verbindingen hebben naar andere units.
- Verbindingen kunnen alleen gaan van een unit naar een unit met een hoger nummer. Deze laatste beperking maakt dat het een feedforward netwerk is dat door simpele backpropagation getraind kan worden.

Figuur 2 toont een voorbeeld van een grilliger netwerk.

3.2 De trainingsset

Het bestand `xor.pat` bevat de trainingsparen:

```
# XOR patterns
p00     0 0     0
p01     0 1     1
p10     1 0     1
p11     1 1     0
```

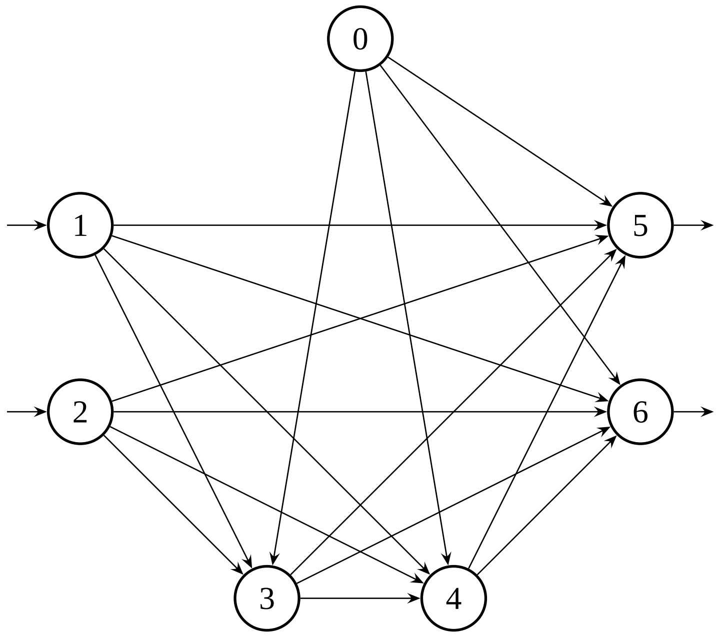
Elk patroon bevat

- een label bestaande uit een woord beginnend met een letter,
- de waarden voor de invoerunits,
- en de waarden voor de bijbehorende uitvoerunits.

In dit voorbeeld worden hele getallen gebruikt, maar dit is niet noodzakelijk.

3.3 Uitvoer van `bp`

Het programma `bp` maakt een aantal bestanden aan waarvan er twee van belang zijn.



Figuur 2: Non standaard feedforward netwerk

Het bestand `demo_out.fun` bevat het getrainde netwerk weergegeven als een stelsel functie-definities voor `gnuplot`. Hier is een voorbeeld:[2]

```
sigmoid (x) = 1.0 / (1.0 + exp (-x))
unit3(i1,i2)=sigmoid(-2.501253\
 +5.877529*i1\
 +5.920436*i2)
unit4(i1,i2)=sigmoid(-5.978926\
 +3.895354*i1\
 +3.904083*i2)
unit5(i1,i2)=sigmoid(-3.510484\
 +7.745664*unit3(i1,i2)\
 +-8.387618*unit4(i1,i2))
```

Het bestand `demo_out.out` bevat een overzicht van de werking van het netwerk tijdens het trainen. De regels in dit bestand zien er zo uit:

```
p00 5 0.491501 0.491501
```

In de eerste kolom staat de naam van het trainigspaar (zie `xor.pat`). De tweede kolom bevat het volgnummer van de huidige epoch. De derde kolom bevat de fout die het netwerk maakt voor dit trainingspaar. De fout is gedefinieerd als de

2. Versie 3.5 van `gnuplot` kan functies met maximaal vijf variabelen aan. Wanneer meer invoer-units gebruikt worden zijn deze functies dus niet te gebruiken.

Euclidean Distance:

$$\sqrt{\sum_i (out_i - target_i)^2}$$

Wanneer er maar 1 uitvoer-unit is zoals in dit geval is deze fout gelijk aan het absolute verschil tussen uitvoer en gewenste uitvoer. Na de eerste drie kolommen volgen de huidige uitvoerwaarden horend bij dit patroon, voor elke uitvoer-unit één waarde.

Verder komen er in het bestand regels voor zoals:

```
_MEAN_  5      0.500015
_WORST_ 5      0.578460
```

De eerste regel geeft het gemiddelde van de fouten van alle patronen in epoch 5. De tweede regel geeft de grootste fout in dit epoch weer.

4 Resultaten bekijken met `gnuplot`

Het programma `demo` start `gnuplot` om de resultaten te bekijken. Voordat dit gebeurt wordt het bestand `demo_out.out` met behulp van `bex` opgesplitst in de diverse bestanden waarvan de naam begint met `demo_plt`. Die bestanden bevatten op elke regel nog maar twee waarden, een epoch nummer en de uitvoer (`demo_plt.p00` etc) of de fout (`demo_plt.e00` etc). Er worden vier bestanden met uitvoer-waarden gemaakt en vier bestanden met fouten, voor elk patroon één. De eerste groep bestanden wordt gebruikt om de eerste grafiek weer te geven. De tweede groep bestanden wordt gebruikt voor de tweede grafiek. Ook worden uit `demo_out.out` de bestanden `demo_plt.em` en `demo_plt.ew` gefilterd die gebruikt worden voor de derde grafiek. Zie het bestand `demo.plt` waaruit `gnuplot` in deze demonstratie z'n instructies haalt. De vierde en vijfde grafiek worden weergegeven door de functies te gebruiken die gedefinieerd zijn in het bestand `demo_out.fun`.

5 Implementatie van backpropagation

Voor een aantal practicum-opgaven zal het nodig zijn om het programma `bp` aan te passen en te hercompileren. De implementatie is eenvoudig gehouden zodat aanpassing niet veel problemen kan opleveren. Hieronder volgt enige toelichting.

5.1 Belangrijkste globale data

De volgende variabelen zijn gedefinieerd:

```
int
    n_in,       /* aantal invoer-units   */
    n_hid,      /* aantal hidden units   */
    n_out,      /* aantal uitvoer-units  */
    n_units;    /* n_in + n_hid + n_out  */
```

unit [inA]	de eerste invoer-unit
unit [inZ]	de laatste invoer-unit
unit [hidA]	de eerste hidden unit
unit [hidZ]	de laatste hidden unit
unit [outA]	de eerste uitvoer-unit
unit [outZ]	de laatste uitvoer-unit

Tabel 2: Units met index

De units zijn gedefinieerd als een array van floats:

```
float
    *unit = (float *) malloc ((n_units + 1) * sizeof (float));
```

`unit [0]` heeft altijd de waarde 1. De overige units zijn invoer-, hidden en uitvoer-units, in die volgorde. Voor het gemak zijn enkele variabelen gedefinieerd die gebruikt worden als index in de array om te verwijzen naar de diverse soorten units. Zie tabel 2.

Voor het gewenste uitvoer-patroon is een aparte array gedefinieerd:

```
float
    *tar = (float *) malloc (n_out * sizeof (float));
```

De array `float *d;` wordt gebruikt voor de gradiënt zoals die berekend wordt tijdens backpropagation.

In de matrix `float **w;` staan de gewichten tussen de units. `w [i][j]` staat voor de verbinding van `unit [j]` naar `unit [i]`. De matrix `float **odw;` bevat de oude aanpassing van de gewichten zoals die gebruikt worden voor de toepassing van momentum.

De matrix `enum BOOL **connected;` geeft aan welke units met elkaar verbonden zijn.

Overige data:

```
float
    eta = 0.35,             /* leerconstante                    */
    alfa = 0.9,             /* momentum                         */
    init_weight = 1.0,      /* maximale beginwaarde gewichten   */
    quit_error = 0.05;      /* maximale fout voor succes        */
```

De default-waarden die je ziet werken redelijk voor het XOR-probleem. Deze waarden kunnen door *command line arguments* worden gewijzigd.

5.2 Feedforward

Het programma implementeert een standaard feedforward netwerk. De uitvoer a_i van een hidden of uitvoer-unit `unit [i]` wordt gegeven door:

$$net_i = \sum_j w_{ij} a_j$$

$$a_i = \frac{1}{1 + e^{-net_i}}$$

Dit gebeurt in de functie `feed_forward()`. Voor aanroep van deze functie staat een invoer-patroon klaar in `unit [inA]` tot en met `unit [inZ]`. Na afloop staat de uitvoer in `unit [outA]` tot en met `unit [outZ]`.

```
void feed_forward ()
{
    int
        i,
        j;
    float
        sum;

    for (i = hidA; i <= outZ; i++) {
        sum = 0.0;
        for (j = 0; j < i; j++)
            if (connected [i][j])
                sum += w [i][j] * unit [j];
        unit [i] = 1.0 / (1.0 + exp (-sum));
    }
}
```

5.3 Backpropagation

Backpropagation maakt gebruik van momentum. De gradiënt d_i van de uitvoer-units worden berekend door:

$$d_i = out_i(1 - out_i)(target_i - out_i)$$

De gradiënt d_i van de hidden units wordt berekend volgens:

$$d_i = out_i(1 - out_i)\sum_j w_{ji}d_j$$

Dit alles gebeurt in omgekeerde volgorde zodat wanneer d_i berekend wordt d_j bekend is voor elke $j > i$. Aanpassing van de gewichten volgt door middel van:

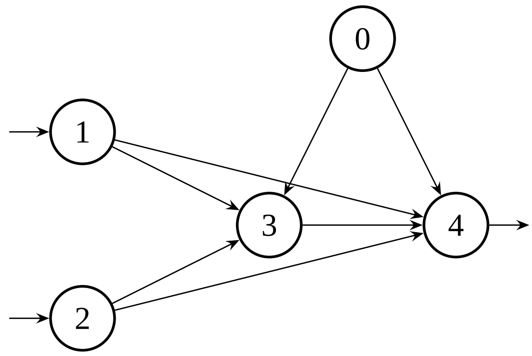
$$\Delta w_{ij}[t] = \eta d_i a_j + \alpha \Delta w_{ij}[t-1]$$

Als de functie `back_propagate ()` wordt aangeroepen staat een uitvoer klaar in `unit [outA]` tot en met `unit [outZ]`, en een *target* in `tar [0]` tot en met `tar [n_out - 1]`.

```
void back_propagate ()
{
    int
        i,
        j;
    float
        sum;

    for (i = outZ; i >= outA; i--)
        d [i] = unit [i] * (1.0 - unit [i])
              * (tar [i - outA] - unit [i]);

    for (i = hidZ; i >= hidA; i--) {
        sum = 0.0;
        for (j = i + 1; j <= outZ; j++)
            if (connected [j][i])
                sum += d [j] * w [j][i];
        d [i] = unit [i] * (1.0 - unit [i]) * sum;
    }

    for (j = 0; j < n_units; j++)
        for (i = j + 1; i <= n_units; i++)
            if (connected [i][j])
                w [i][j] +=
                    (odw [i][j] = eta * d [i] * unit [j]
                                + alfa * odw [i][j]);
}
```

5.4 Training en testen

De functie `main` roept functies aan voor het verwerken van parameters, inlezen van bestanden en initialisatie van het netwerk. Daarop volgt een lus waarin getraind en getest wordt, en tussentijdse resultaten in een bestand worden bewaard. Na training worden de eind-resultaten bewaard.



Figuur 3: Minimaal netwerk voor XOR functie

6 Opgaven (suggesties)

6.1 De XOR-functie

Het leren van de XOR-functie door `demo` gebruikt standaard-waarden voor de leerconstante, momentum en de maximale beginwaarden van de gewichten. Bestudeer het effect van andere waarden voor deze parameters. (Gebruik de command line arguments -e -a en -w.) Hoe leert het netwerk zonder momentum? Wat is de grootste waarde voor de leerconstante waarbij het netwerk nog leert?

Kan het netwerk de XOR-functie leren zonder bias? (Maak een kopie van `221.net` en verwijder de regel `0 -> 3 4 5`.)

Er zijn twee globale oplossingen die het netwerk voor de XOR-functie kan vinden. Beschrijf het verschil. Hoe liggen de desicion surfaces die voor deze oplossingen worden gevormd door de hidden units?

Het lukt niet altijd om de XOR-functie te leren: het netwerk komt in een lokaal minimum terecht. Wat gebeurt er als het mislukt? Kijk naar de desicion surfaces die de hidden units vormen.

Het netwerk dat gebruikt wordt in de demonstratie is niet het kleinste netwerk dat de XOR-functie kan leren. Figuur 3 toont het kleinste. Test dit netwerk. Behalve de netwerk-definitie moet ook `demo` en `demo.plt` aangepast worden. (Vier units in plaats van vijf.) Maak kopieën en vervang in de bestanden de tekst `demo` door een andere string. Raakt dit netwerk ooit in een lokaal minimum verstrikt?

Voeg aan de patronen in `xor.pat` het volgende patroon toe:

```
phalf  0.5  0.5    0.5
```

Hoeveel hidden units zijn er minimaal nodig om deze set van vijf patronen te leren?

6.2 Het Pariteits-probleem

Het pariteits-probleem houdt het volgende in: Wanneer het aantal 1-en in de invoer even is, geef 0 als uitvoer; is het aantal 1-en in de invoer oneven, geef 1 als uitvoer. Het pariteits-probleem is een van de moeilijkste types van problemen voor backpropagation, dat gelukkig in de 'vrije natuur' weinig voor komt. Probeer uit te leggen waarom dit probleem zo lastig is.

Train een backpropagation netwerk voor het pariteits-probleem met vectoren van lengte 4. Ga als volgt te werk:

1. Genereer een set trainingsparen door het commando:

   `parity 4 > par4.pat`

2. Creëer een netwerk-definitie (4 invoerunits, 1 uitvoerunit). Noem dit bestand `pat4.net`.
3. Train het netwerk met het programma `pat4`.

In principe kan de 4-bits pariteitsfunctie worden uitgevoerd door een feedforward netwerk met een verborgen laag met niet meer dan vier units. Met backpropagation lukt het echter meestal niet om zo'n netwerk de functie te leren. Probeer het uit. Wat gaat er mis?

Probeer andere netwerk-structuren uit. (Meer verborgen units, meer verborgen lagen, meer verbindingen). Met wat voor netwerk-architectuur is het je gelukt de taak wel te leren? Welke overige parameters heb je gebruikt? Welke vectoren zijn het moeilijkst te leren? Waarom?

Voor de ambitieuzen: Train een netwerk op het pariteits-probleem met vectoren van lengte groter dan 4.

6.3 Snellere leermethodes in backpropagation

Er zijn vele trucjes en trucs bedacht om het leren met backpropagation te versnellen. De toevoeging van momentum is zo'n truc. Een andere truc is de volgende: Gebruik invoer- en uitvoer-patronen met waarden tussen -1 en 1. Waarom zou dit een verbetering zijn? Om dit uit te testen moeten behalve de aanpassing van de trainingsparen ook het programma `bp` zelf worden aangepast.

Gebruik voor feedforward de volgende regel, die een uitvoer heeft tussen -1 en 1:

$$net_i = \sum_j w_{ij} a_j$$

$$a_i = \tanh(net_i)$$

(Wanneer je een programmeertaal gebruikt die geen tanh() kent: $\tanh(x) = \frac{2}{1+e^{-2x}} - 1$)

Backpropagation moet eveneens aangepast worden. Daarin komt de term $out_i(1 - out_i)$ voor. Deze term wordt gebruikt in verband met de afgeleide functie van de gebruikte sigmoid:

$$f'(x) = f(x)(1 - f(x))$$

Voor de afgeleide van tanh geldt:

$$f'(x) = 1 - (f(x))^2$$

Verwerk deze term in de functie `back_propagate` ($2\times$). Test dit programma en vergelijk de snelheid waarmee het leert met die van het oude programma. Je zult een kleinere waarde voor de leerconstante moeten gebruiken, ongeveer een kwart.

6.4 Project: Quickprop (4 weken)

Implementeer het quickprop algoritme (bijvoorbeeld [Has95]) in het programma `bp.c`. Voer een aantal van de eerste opdrachten opnieuw uit en vergelijk de prestaties van het nieuwe programma met die van het oude.

6.5 Project: Cascade-Correlation Learning (8 weken)

Backpropagation traint een meerlaags feedforward netwerk door gebruik te maken van de *generalized delta rule*. Deze methode is inefficiënt om twee redenen:

- Het aantal berekeningen dat in elk epoch moet worden uitgevoerd is groot.
- Met het aanpassen van de gewichten veranderen de activatie-waarden van de verborgen units. Omdat de aanpassing van de gewichten deels gebaseerd is op deze activatie-waarden moet het netwerk een *moving target* zien te bereiken.

Cascade-Correlation Learning [FL90] is een voorbeeld van een leermethode voor feedforward netwerken die sterk afwijkt van backpropagation. Verborgen units worden toegevoegd wanneer ze nodig zijn. Verbindingen naar verborgen units veranderen niet als ze eenmaal deel uitmaken van het netwerk, dus geen *hidden targets*. Deze methode is niet alleen sneller dan backpropagation, maar ook krachtiger. Een aantal taken die voor backpropagation te lastig zijn kunnen met Cascade-Correlation wel geleerd worden.

Implementeer Cascade-Correlation en herhaal enkele experimenten die je eerder met backpropagation hebt uitgevoerd.

Referenties

[FL90] Scott E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In Touretzky [Tou90], pages 524–532.

[Has95] Mohamad H. Hassoun. *Fundamentals of Artificial Neural Networks.* The MIT Press, Cambridge, MA, 1995.

[Tou90] David S. Touretzky, editor. *Advances in Neural Information Processing Systems 2*, San Mateo, CA, 1990. Morgan Kaufmann Publishers.