Djupinlärning av Neurala nätverk

John Paul Edward Möller

8 april 2020

Innehåll

1 Anteckningar

1.1 Förklara orden:

Modul, dimension av vektor, vektorkomponent, list (python)

1.2 Referenser:

Att 3-sat är ett NP problem och alla NP problem är ekvivalenta.

2 Abstract

3 Inledning

3.1 Bakgrund

3.2 Syfte och Frågeställning

4 Teori

4.1 Propositionell logik

4.2 Linjär algebra

4.3 Deep learning

4.3.1 Bildlig förklaring

4.3.2 Linjär algebra

4.3.3 Flervariabel analys

4.4 Introduktion till python

5 Metod

5.1 Skapandet av datan

5.1.1 Krav på datan

Oavsett hur man bygger upp det neurala nätverket så behövs det data med exempel som är redan lösta. I detta arbete är syftet se om ett neuralt nätverk kan avgöra om ett 3-SAT problem är satisferbart eller inte. Alltså ska inputen vara ett 3-SAT problem och outputen vara en indikation på om det är satisferbart eller inte. För konstruktion av de neurala nätverk kommer modulen Tensorflow användas. Med den modulen krävs det att man har två listor: en för inputen och en för outputen. Alltså kommer första exemplets formell vara första elementet i inputlistan och svaret kommer vara första elementet i outputlistan. I detta arbete valdes det 60 000 träningsexempel och 10 000 utvärderingsexempel. Träningsexempelerna är de exempel det neurala nätverket kommer analysera och anpassa sig efter. Utvärderingsexemplerna är de som kommer användas i betygsättning av hur bra nätverket fungerar. Anledningen varför dessa måste vara separata är att det är väldigt enkelt för neurala nätverk att endast memorera exemplerna som användes vid träningen. Därför krävs det att exempelerna är unika så att inget exempel i träningsdatan finns dessutom i utvärderingsdatan.

Ett annat krav är att inputen och outputen är i form av en vektor av reella värden. Detta är inte ett problem för outputen då man kan ha en 2 dimensionell vektor där första värdet är sannolikheten att formeln är falsk och andra värdet är sannolikheten att den är sann (se exempel1). En lika tydlig

lösning för inputen finns dock inte. Ett förslag är att associera ett värde till varje tecken och sedan låta antalet tecken beteckna dimensionen på vektorn (se exempel2). Som man dock ser i exempel (k0lla om det är två eller 2) två så är dimensionen av vektorn onödigt stor då vissa tecken har alltid samma värde. De enda tecknen som kan vara olika är variablerna. I denna text kommer detta faktum utnyttjas, då antalet variabler i uttrycket kommer motsvara dimensionen av vektorn.

1. Figurer till parent sektion: exempel1, exempel2

5.1.2 Omvandling av 3-SAT formler

Låt variablerna i en formel ordnas i viss order, exempelvis alfabetsordning och sedan stigande i nummer index. Den första variabeln kommer motsvara värdet 1, den andra 2, den tredje 3 o.s.v. Sedan när man har kodat för den sista variabeln med värdet n så låter man $n + 1$ symbolisera negationen av den första variabeln och $n + 2$ negationen för den andra variabeln o.s.v. Sedan låtes det första variabeltecknet i formeln koda för första värdet i vektorn och den andra den andra värdet o.s.v. (se exempel3).

I denna text kommer det dessutom begränsas till max 5 olika variabler i varje formel. Därmed så kan varje heltal (om man tar hänsyn till ledande nollor) symbolisera en unik formel (se exempel4). Detta löser nästan problemet om att skapa 60 000 träningsexempel och 10 000 utvärderingsexempel som är alla unika. Eftersom varje heltal motsvarar en unik formel så kan man låta de talen från 0 till och med 59 999 koda för träningsexempelarna och talen 60 000 till 69 999 koda för utvärderingsexempelarna. Dock, somsagt löser detta nästan problemet. Första problemet är att om man ska ha fem stycken värdesiffror så kommer formelerna vara fem variabler långa. Men ett 3-SAT problem är på formen av en konjuktion av disjuktioner av längden tre variabler. Därför måste antalet variabler vara lika med en multipel av tre. Det andra problemet är att formeln är kort och när en formel är kortare så är det mindre chans att det blir en falsk formel (om man antar att antalet olika variabler håller sig konstant). Anledningen är att fler konjuktioner begränsar antalet möjliga lösningar (om antalet variabler håller sig konstant). Ett exempel på detta ges i (exempel5).

Lösningen på detta är att införa slumpmässiga siffror efter. På detta vis kommer alla exempel vara unika, och längden kan anpassas till en multipel av tre. I denna text kommer längden vara 72 variabler d.v.s fem bestämda siffror och 67 slumpmässiga.

1. Figurer till parent paragraf: exempel3, exempel4, exempel5

5.1.3 Algoritm

1. Algoritmen på en allmän nivå
   (a) Skapa en tom lista för all input träningsdata
   (b) Skapa en tom lista för all input utvärderingsdata
   (c) Skapa en tom lista för all output träningsdata

(d) Skapa en tom lista för all output utvärderingsdata

(e) Låt n = 0

(f) Omvandla n till en lista med 5 värden

(g) Skapa en lista med 67 slumpmässiga värden

(h) Sätt ihop listorna från 2 och 3 till en ny lista

(i) Lägg till listan från 8 i input träningsdata listan (1)

(j) Omvandla listan från 8 till en formel

(k) Kolla om formeln från 10 är satisferbar

(l) Om den är satiferbar lägg till en etta i output träningsdatan

(m) Om den inte är det lägg till en nolla i output träningsdatan

(n) Repetera steg 5 till 13 där n ökar med ett varje gång, tills n=69999

(o) Repetera steg 5 till 13 där n ökar med ett varje gång, från 69999 till 70000

2. Algortimen i python kod

   (a) Numpy För att översätta den allmänna koden till python så skulle det tekniskt gå att ha vanliga listor. Men p.g.a att det är så stora värden samt att vi vill senare kunna exportera dessa listor, så är det mer praktiskt att använda en modul som heter Numpy. Listor i numpy heter numpy arrays". Numpy arrays fungerar i omfattningen av denna text i princip som vanliga lists. Enda skillnaden är att det inte går att direkt modifera dem. Om man vill ändra en variabel som innehåller en numpy array så måste man skapa en ny array baserad på den gamla och sedan spara den nya i variabeln (betrakta kod1). För att skapa en numpy array använder man funktionen numpy.array. Som argument sätter man i en vanlig list som numpy arrayen kommer efterlikna. Betrakta kod2.

```
import numpy

numpy_array = numpy.array([4,2,0])
```

      i. Figurer Kod1, Kod2

3. Pythonkoden actually