Djupinlärning av Neurala nätverk

John Paul Edward Möller

9 april 2020

Innehåll

1 Anteckningar

1.1 Förklara orden/förklara:

Modul, dimension av vektor, vektorkomponent, list (python), parametrar till en funktion (python), funktionparameter på följande vis: f(arg1, arg2) Rectifier, Sigmoid, objekt orienterade termer (objekt och metod), softmax, Stokastiskt gradient nedgång

1.2 Fixa:

Att jag har skrivit numpy.funktion men bör bara konsekvent med np.funktion instället

1.3 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

4.2.1 Vektor

4.2.2 Matris

4.2.3 Matris vektor multiplikation

4.3 Deep learning

4.3.1 Bildlig förklaring

4.3.2 Linjär algebra

4.3.3 Flervariabel analys

1. Funktioner av flera variabler
2. Paritalderivata
3.

4.4 Grunderna till python och programering

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 Simple-Sat

Simple-sat är en 3-SAT lösare kodad av Sahand Saba, och används i detta arbete under MIT lisens. Den tar in en input i string format där varje konjuktion är separerad med ett komma och disjuktioner är separerade med ett mellanslag.Så en variabel kan vara vilken kombination av bokstäver och siffror så länge det inte innehåller ett mellanslag. Det innebär att till skillnad från programspråk som python så kan en enkel siffra vara en unik variabel. Detta är i fördel till omvandlingen i denna metod eftersom varje siffra kodar en viss variabel. För att beteckna negationen används tecknet tilde (˜) framför variabeln utan mellanslag. Betrakta exempelA.

Problemet med detta program är att det kan endast bli kallad i en terminal. För att ka

1. fig stuff exempelA

5.1.4 Algoritm

1. Algoritmen på en allmän nivå
   (a) Skapa en tom lista för all träningsdata
   (b) Skapa en tom lista för all utvärderingsdata
   (c) Låt n = 0
   (d) Omvandla n till en lista med 5 värden
   (e) Skapa en lista med 67 slumpmässiga värden
   (f) Sätt ihop listorna från 2 och 3 till en ny lista
   (g) Lägg till listan från 8 i input träningsdata listan (1)
   (h) Omvandla listan från 8 till en formel
   (i) Kolla om formeln från 10 är satisferbar
   (j) Om den är satiferbar lägg till en etta i output träningsdatan
   (k) Om den inte är det lägg till en nolla i output träningsdatan
   (l) Repetera steg 3 till 13 där n ökar med ett varje gång, tills n=69999
   (m) Exportera datan
2. Algortimen i python kod
   (a) Numpy arrays 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.
kod2 (eller 1?)

```
import numpy


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

Fotnot till kod2: Det är två paranteser p.g.a att man kan sätta in flera parametrar och de arrays man vill lägga ihop sätter man in i första parametern.
En sak man måste ta i hänsyn med numpy är att man måste specifiera vid skapandet av en array vad för typ alla element kan vara. Man kan därmed inte ha olika typer. Detta i kontrast till vanliga listor som man kan ha flera olika typer. Betrakta exempel6 där listan av sig själv fungerar men om man omvandlar till en numpy array skapas det ett felmedelande. Man kan dock skapa en tom arrray och lägga till värden, så detta är inte ett problem för ouputdatan. Men för input datan så har man en sammling av listor av storleken 72. Om man skulle skapa en tom numpy array och en numpy array med 72 element och sedan försöka använda numpy.concatenate, visas ett felmedelande. Man är då tvungen att vid skapandet av numpy arrayen för träningsdata redan ha ett element som är en numpy array på storleken 72. Ett enkelt sätt att göra detta är med funktionen numpy.zeros. Numpy.zeros tar in en storlek som argument och skapar en numpy array fylld med nollor med den specifierade storleken. För att skapa en array med 5 nollor skriver man numpy.zeros(5). För att skapa en array fylld med 3 arrays där varje array har 2 nollor skriver man numpy.zeros((3,2)) (se kod3).
Eftersom träningsdatan kommer vara en array där varje element är en array på 72 element, så skriver man numpy.zeros((1,72)). Ett problem som uppstår på grund av detta är att första träningsexemplet kommer vara på plats två och det andra på plats 3 o.s.v (se figur1). Detta löses dock enkelt genom att ta bort det första elementet i slutet av algoritmen. Man kan göra detta med funktionen numpy.delete. En sista detalj är att koden så betecknas input med x och output med y. Notationen är inte för någon speciell anledning, för att det ska bli enklare att skriva variabeln. Så steg 1 och två kan översättas till följande kod:

```
import numpy as np


x_train = np.zeros((1,72))
y_train = np.array([])
```

(b) Loopen steg 3 till 7 Nästa del i koden är att påbörja loopen från steg 3 till 12. Första

steget i loopen är att omvandla n till en lista med fem värdesiffror. För det första så är det mer korrekt enlight pythons rekomederade stilformat att ha ord istället för bokstäver som variabler, så n kommer reffereras som num i python koden. Omvandlingen av num delades upp i fyra delar. Först omvandlades num till en String. Sedan lades det eventuellt på ledande nollor för att få Stringen att ha 5 värdesiffror. Sedan omvandlades den till en vanlig lista, och tillsist en numpy array. Steg 1 och 2 blev en rad och steg 3 och 4 blev också en rad. Koden såg ut som följande:

kod4

```
num_string = str(num).zfill(5)
head_array = np.array(int_list(num_string))
```

Steg 5 skrevs på följande vis:

kod5

```
tail_array = np.random.randint(10, size=67)
```

Första argumentet anger då att alla heltal under det givna nummret får slumpmässigt skapas, och size parametern talar om för storleken av arrayen som skapas.

För att sedan lägga ihop de två arraysen används np.concatenate.

kod6

```
array_to_append = np.concatenate((head_array,tail_array))
```

Sedan för att lägga till denna array till träningsdatan använder np.concatenate för att updatera x_{train}.

kod7

```
x_train = np.concatenate((x_train, [array_to_append]))
```

i. Figurer Kod1, Kod2, exempel6, kod3, figur1, kod4, kod5, kod6, kod7

(c) Loppen steg 8 till 11 För att se om formeln är satisferbar eller inte används en 3-SAT lösare programerad av Sahand Saba under MIT licens. Sabas program körs egentligen via terminalen, men den är tillgänglig med modulen satsolver.py som finns beskriven i appendix I. Modulen innehåller två funktioner: satsolver.generate$_{formula}$ och satsolver.check. Satsolver.generate$_{formula}$ tar en numpy array och omvandlar den enligt metoden beskriven i omvandling av 3-sat, till en formel satsolver.check kan ta som argument. Satsolver.check ger då värdet noll om formeln är ej satisferbar och värdet ett om den är satisferbar. Så koden blir som följande:

kod8

```
formula = sat.generate_formula(array_to_append)
value_to_append = sat.check(formula)
y_train = np.append(y_train, value_to_append)
```

Ett steg som inte står beskrivet i den allmänna algortimen, är att radera det elementet i x_{train}. Detta var som sagt p.g.a vid skapandet av x_{train} behövdes ett element fylld av nollor som specifierade storleken på arrayen. Så för att radera det första elementet i x_{train} används följande kod:

kod9

```
x_train = np.delete(x_train, 0, 0)
```

i. Figurer, kod och referatmarkörer Sahanda Saba, referens till omvandling av 3-sat, kod8, kod9

(d) Export av datan För att exportera numpy arrays används funktionen np.save. Funktionen tar in namnet på en filplats och numpy arrayen man vill spara. I detta projekt så sparades filerna i mapp kallad 'data'. Så exporterings koden såg ut som följande:

```
np.save('data/x_train', x_train)
np.save('data/y_train', y_train)
```

i. Kod saker: ko10

(e) Anpassning för utvärderingsdatan Koden för utvärderingsdata är i princip likadan. Enda skillnaden är att num fick variera mellan 60 000 och 69 999 och datan sparades som x_{test} och y_{test} för input datan respektive output datan.

3. Pythonkoden actually

5.2 Skapandet av modellen

5.2.1 Representation av modellen

Modellen består alltså av 10 lager med 128 noder var, där Rectifier är aktiveringsfunktionen. Det finns dessutom ett till lager i slutet som består av endast en nod och har Sigmoid aktiveringsfunktionen (se figur2).

5.2.2 Tensorflow och Keras

För att skapa den här modellen i python används modulerna Tensorflow och Keras. Tensorflow är en modul som tillåter en att bygga neurala nätverk från grunden. Keras är en modul som som förenklar byggandet och träningen av neurala nätverk. I koden kommer dessa moduler och delar modulerna importeras:

kod11

```
import numpy as np
from tensorflow import keras
from tensorflow.keras.layers import Activation, Dense
```

För att skapa en tom modell som man sedan kan senare tillägga fler lager till skriver man:
kod12

```
model = keras.models.Sequential()
```

Variabeln model blir då ett objekt som symboliserar det neurala nätverket. För tillägg av lager samt träning och utvärdering av modellen används assosierade objekt metoder. För att lägga till ett lager (till höger om man tänker som fig2 där input går in till vänster och output går ut från höger) så använder man objektmetoden add. Som argument tar den vad för sorts lager. Den sortens lager som har hittils gått igenom i texten kallas i keras för 'Dense'. Det är alltså ett lager som kan symboliseras med en matris, eller noder på rad, samt en associerad aktivationsfunktion. Så för att lägga till ett lager med 128 noder, och Rectifier som aktiveringsfunktionen skriver man:
kod13

```
model.add(Dense(128, activation='relu'))
```

relu symbolisera då Recitifier funktionen. Den här raden av kod repeteras då 10 gånger så att det blir totalt tio lager. Sista lagret är två noder där första noden är sannolikheten att det är noll och andra är sannolikheten att det är 1. Detta görs med aktiveringsfunktionen softmax". Följande kod motsvarar detta lager:
kod14

```
model.add(Dense(2, activation='softmax'))
```

1. Saker kod11, kod12, figur2, kod13

5.3 Träning av nätverket

För träning av nätverket används binary cross entropysom loss funktion, 'accuracy' som metric och stokastisk gradient nedgång som optimizer. 'Binary crossentropy' och 'accuracy' förklaras ej i detta arbete men dokumentaion om dessa funktioner finns på [ref2]. För att träna nätverket och sedan utvärdera det skrivs följande kod:
kod15

```
model.compile(optimizer = 'SGD', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test, verbose=2)
```

Epochs är hur många gånger den går igenom all data. Så epoch 1 är första gången den går igenom datan epoch 2 andra gången o.s.v. Verbose variabeln specifierar bara hur mycket information som printas ut.

5.3.1 Stuff ref kod och sånt

ref2, kod15

6 Resultat

6.1 Model 1

Epoch	tid	resultat
1	8 sekunder	50 %
2	7 sekunder	50 %
3	7 sekunder	50 %
4	7 sekunder	50 %
5	7 sekunder	50 %

Och för träningsdatan var resultatet 50 %.

7 Diskussion och slutsats

7.1 Sannolikhet att liknande (och därmed samma) sat formler fanns

7.2 Presicionen var bara 50% som är lika med en gissning.

Som man ser så var resultatet på samtliga delar av utvärderingsprocessen 50 %. Detta tyder med säkerhet att nätverket inte har hittat något mönster då det är lika med chansen för en gissning. Detta stödjer kontentan bland forskare att P är skiljt från NP (ref3). Eftersom om det finns en tydligt mönster som neurala nätverk kan se så skulle det öka chansen att det finns en enklare algortim för 3-SAT problem av allmänn storlek, som är tillräckligt enkel så att den är i polynomisk tid. Detta utesluter såklart dock inte att neurala nätverk kan tränas att lösa 3-SAT problem. På grund av att koden behövde köras på en persondator, så var antalet lager, noder, epochs, och träningsexempel begränsade till processorkraften och minnet tillgängligt.

7.2.1 Ref och sånt

ref3

7.3 Att man använder neurala nätverk för tydliga mönster och inte resonemang

7.4 Samtidigt att det finns ingen teori om hur man känner igen människo ansikten

7.5 Vad innebär att det finns en polynomisk algoritm för 3-sat

7.5.1 Banksystem faller

7.6 Vidare forskning

7.6.1 Djupare nätverk

1.

8 Appendix

8.1 I