Bestämmande av optimal arkitektur för artificiella neuronnät med evolution

Tomass Wilson
thmwi@kth.se
Grupp B:7

27 oktober 2018

Sammanfattning

Maskininlärning används mer och mer i dagens industrier och programutveckling. Utvecklingen av artificiella neuronnät bör optimeras för att underlätta applikationen av denna viktiga teknologi till allt fler och fler områden. Vid körning av Matt Harveys python program (Harvey 2017) syns ett tydligt sätt tiden detta tar kan förkortas, genom användning av en evolutionär algoritm som fungerar på principerna av naturligt urval. Denna process är byggd för att bestämma det optimala neuronnätsarkitektur, då olika kombinationer av meta-variabler såsom antal neuroner och optimeringsfunktion kan starkt påverka nätverkets träffsäkerhet och inlärningstid. Undersökningen visar på förbättringar vid nästan en hel storleksordning gentemot iterativ sökning, då processtiden vid denna studie gick från 18 till 3 timmar. Detta kan appliceras på flera olika implementeringar av artificiella neuronnät för att förenkla och effektivisera deras utveckling.

Innehåll

1 Inledning

Programmerare har alltid velat lösa problem, hellre med hjälp av en dator. När Artificiella Neuronnät (ANN) började utvecklas kunde man applicera dem på problem som före-detta verkade omöjliga, såsom att urskilja ansikten eller kategorisera bilder (Hopfield 1988). Inlärningsprocessen tar ofta flera timmar och bygger på att slumpa fram de kopplingarna mellan neuroner sådan att en viss träffsäkerhet på resultatet nås för det specificerade problemet. En viktig faktor som påverkar inlärningen av en ANN är meta-variabler, såsom antal neuroner per lager, antal lager, aktiveringsfunktionen, etcetera. Dessa meta-variabler kan ställas in manuellt av användaren, men detta brukar leda till att neuronnätet inte når sin maximala träffsäkerhet, eller förlänger inlärningstiden märkbart. Lösningen till detta har tidigare varit att iterera genom alla möjliga combinationer meta-variabler och testa dem var-för-sig. En föreslagen lösning till detta är att istället evolutionärt optimera slumpmässigt valda meta-variabler för att undvika onödiga tester av ineffektiva kombinationer av dessa.

1.1 Syfte

Syftet med denna studie är att undersöka om tiden för inlärandeprocessen för ett neuralt nätverk kan förkortas med hjälp av en evolutionär process. Precisionen för nätverket ska vara helst den samma som vid en iterativ inlärningsprocess, för att kunna ge en komparabel alternativ.

1.2 Frågeställning

Vilken inställning av Matt Harveys pythonprogram uppskattar de optimala meta-variabler på kortast tid?

2 Bakgrund & Teori

2.1 CIFAR-10

Industri-standarden för att testa ANN är CIFAR-10, en databas av 60 000 bilder, som kategoriseras in i 10 grupper (Krizhevsky, Nair och Hinton 2014). Med detta kan man okomplicerat jämföra olika ANN, genom att mäta hur snabbt och väl de kan identifiera vilken kategori en bild tillhör. Som input får nätverket en bild av 32 x 32 pixlar, som mäts in i Input lagern som en matrix. Output lagern har 10 noder, med ett värde mellan 0 och 1 för att

visa hur mycket bilden ”passar” in i en viss kategori. Korrektheten mäts som andel gånger nätverket har placerat bilden i korrekt kategori

2.2 Artificiella Neurala Nätverk

En Artificiell Neural Nätverk (ANN) är en samling sammankopplade neuroner sammanställda i flera *lager*. Varje nod har ett värde och en aktiveringsfunktion, och befinner sig i ett lager. En nod har flera inkommande länkar och utgående länkar till andra neuroner i lager under och över den respektive. Inkommande värden multipliceras med en slumpmässig vikt och sedan multipliceras alla inkommande värden tillsammans. Detta nya värde kläms sedan ner till ett värde mellan 0 och 1. Denna process kallas *aktiveringsfunktionen* och slutgiltiga resultatet är nodens *värde*. Antal neuroner i input- och outputlagern bestäms i förhand beroende på hur nätverket ska användas. I denna studie är inputlagern en matrix med 32x32 neuroner som representerar en svartvit bild, där ett högre värde (1) representerar vit och ett lägre (0) svart. Outputlagern uppbyggs av 10 neuroner, där neuronen med högsta värde representerar nätverkets *gissning* för vilket kategori bilden passar in i. Till en början är alla neuronkopplingar och deras vikter slumpmässigt valda, vilket gör att den ger helt slumpmässiga svar. För att optimera nätverket krävs en *optimeringsfunktion*.

2.3 Meta-variabler

För att ett ANN ska kunna skapas behövs några startparametrar. Dessa *meta-variabler* bestämmer övergripande hur nätverket ser ut, dess storlek och konstruktion. Undersökningen som beskrivs i denna artikel kommer att behandla dessa meta-variabler:

1. # Neuroner
   Detta är antal individuella neuroner i varje lager. Detta experiment kommer testa värderna 64, 128, 256, 512, 768, 1024 neuroner per lager.

2. # Lager
   Detta är antal lager exklusive input- och outputlagern, dessa heter *gömda lager* och antalet varierar mellan 1 - 4

3. Aktiveringsfunktion
   Aktiveringsfunktioner kommer i flera olika sorter, som brukar likna s kurvor. Vilken aktiveringsfunktion man använder påverkar hur komplex ett nätverk kan bli och dess maximala träffsäkerhet (Jain, Mao och

Mohiuddin 1996). Detta experiment använder sig av aktiveringsfunktionerna: *sigmoid, tanh* (Karlik och Olgac 2011), *Exponential Linear Unit* (ELU) (Clevert, Unterthiner och Hochreiter 2015) och *Rectified Linear Unit* (ReLU) (Xu m. fl. 2015)

4. Optimeringsfunktion
   Optimeringsfunktionen är den process som nätverket använder för att optimeras. Optimeringsfunktioner förbättrar nätverket genom att ändra på kopplingar och dess vikter mellan neuroner och mäta förändringen i träffsäkerhet, och behåller bara de ändringar som är gynnsamma (Walia 2017). Funktionerna som ska testas är: *rmsprop, sgd, adam, adagrad, adadelta, adamax* och *nadam* (Kingma och Ba 2014)

2.4 Evolutionär Inlärningsprocess

Som i naturen fungerar den evolutionära algoritmen som ett slags naturligt urval. Vid första början skapas en population med ett visst antal neurala nätverk med slumpmässigt valda meta-variabler. Varje nätverk får sedan påbörja inlärningsprocessen, tills inlärningen når ett lokalt maximum och kan inte lära sig något mer. Tiden inlärningen tar och nätverkets träffsäkerhet mäts och kombineras, vilket då blir nätverkets *fitness*. Nätverket i populationen som har lägst fitness, ”dödas” och ersätts av nätverk som är en kombination av 2 levande nätverk, ”föräldrarna”, och får sedan en slumpmässig mutation (detta för att populationen inte ska bli för homogen). Efter flera *generationer* har förmodligen det bästa nätverket en sammanställning meta-variabler som är optimerade för användningsområdet. (Yao och Liu 1997)

I denna undersökning kommer systemet utföra 10 generationer, med 20 neuronnät var. Varje generation kommer undersökas och sedan kommer det 12 sämsta neuronnät (de med lång inlärningstid eller dålig träffsäkerhet) raderas (+- några slumpmässignt valda ANN, för att populationen inte ska bli homogen). Nya nät skapas för att få populationen upp till 20 genom att slumpmässigt välja meta-variabler från två existerande nät. Sist så ändras en meta variabel helt slumpmässigt (en mutation).

3 Metod

För att jämföra de två olika metoder för att bestämma bästa meta-variabler användes CIFAR-10 databasen för input material. Själva processen av populationskapande, fitness evaluering och evolution är skriven av Matt Harvey (Harvey 2017) med några modifieringar för att spara minne gjord av Tomass

Wilson. All kod skrevs i python 3.6 och kördes på Windows 10, och använder sig till huvuddel av keras och tensorflow paketen för att optimera och skapa neuronnäten. Versionerna av samtliga paket ses i tabell 1. Hårdvaran som användes i denna undersökning ses i tabell 2.

Till först så testades den iterativa processen genom att köra brute.py filen. Alla 672 möjliga kombinationer meta-variabler testades och en slutsiffra på tiden hela processen tog antecknades. Detta ger en basnivå som kan jämföras med den evolutionära processen. Sedan kördes den evolutionära processen med 10 generationer av 20 neuronnät var genom att köra main.py filen, och tiden skrevs ned.

Paket	Version
keras	2.2.2
keras-applications	1.0.4
keras-preprocessing	1.0.2
numpy	1.15.1
scipy	1.1.0
six	1.11.0
tensorflow	1.10.1
tqdm	4.25.0
werkzeug	0.14.1
wheel	0.31.1

Tabell 1: Paketversioner

Komponent	Specifikation
GPU:	RTX 2080 @ 1950 MHz core, 7263 MHz mem
CPU:	Intel i7 8700 @ 4.28 GHz
Minne:	32 GB @ 2666 MHz

Tabell 2: Hårdvara

4 Resultat

Dessa resultat insamlades över flera dagar, och datorn programmet kördes på användes lätt under den tiden.

Programmet slutar med att presentera de fem bästa neuronnät som den kunde hitta. Resultatet från den evolutionära processen visas i tabell 3, och för den iterativa i tabell 4. Totala tiden för den evolutionära algoritmen samt iterativa ses i tabell 5.

Träffsäkerhet	#Neuroner	#Lager	Akt-funk.	Opt-funk.
56.12%	256	4	ELU	adamax
55.75%	256	4	ELU	adamax
55.65%	512	4	ELU	adamax
55.64%	256	4	ELU	adamax
55.57%	256	4	ELU	adamax

Tabell 3: 5 bästa neuronnät efter evolutionär sökning.

Träffsäkerhet	#Neuroner	#Lager	Akt-funk.	Opt-funk.
55.52%	768	2	ELU	adamax
55.47%	1024	2	ELU	adamax
55.33%	512	3	ELU	adamax
55.30%	768	2	sigmoid	adamax
55.21%	768	4	sigmoid	adamax

Tabell 4: 5 bästa neuronnät efter iterativ sökning.

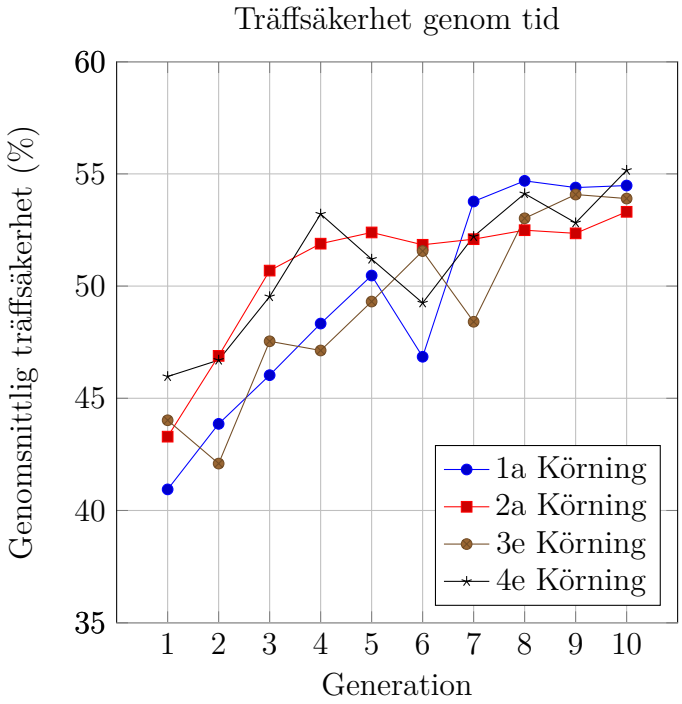
För att få ett närmare inblick i den evolutionära processen så kördes den i flera omgångar och träffsäkerhet för varje omgång/generation presenteras i figur 1.

5 Diskussion

Enligt resultatet som redovisas i tabell 5 så kan frågeställningen enkelt besvaras. En skillnad på 10h 10m visar på den stora fördelen med att använda den evolutionära processen med inga förluster av träffsäkerhet. Däremot måste implementationen av systemet diskuteras samt hur applicerbart detta är på andra datamängder/maskininlärningsproblem. Studien som genomfördes riktar sig in på bara ett av flera andra problem som maskininlärning appliceras på. Vissa användningar, där neuronnätets arkitektur är given eller inlärningstiden är väldigt kort behövs inte sådana metoder, då den bara försvårarar för användaren. Åt andra sidan så kan detta system underlätta för sökandet av meta-variabler då dessa är okända, och kommer troligen ge nästan lika bra output som en iterativ process.

Inställning	Tid	Maximal Träffsäkerhet
Evolutionär:	3h 43m 14s	56.12%
Iterativ:	18h 53m 40s	55.52%

Tabell 5: Total tid för hela processen



Figur 1: Träffsäkerhet för varje generation

Bästa nätverket som hittades i första evolutionära körning (tabell 3) är inte den samma som den som hittades av den iterativa processen (tabell 4). Detta är inte ett stort problem, då de fick nästan samma maximala träffsäkerhet, men de kan ha olika inlärningstid. Totala antalet neuroner är tillräckligt lika, $256 * 4 = 1024$ medan $768 * 2 = 1536$, för att tyda på olika lösningar till samma problem. Dessutom, eftersom optimeringsprocessen utgår ifrån att slumpmässigt försöka förbättra varje neuronnät, så kan man få olika träffsäkerhet vid varje optimeringsomgång för *samma* sammanställning meta-variabler. Däremot kommer detta avvikning vara lite, och inte spela mycket roll även om det leder till att man väljer än eller annan arkitektur för neuronnätet. Om det absolut bästa sammanställning meta variabler är nödvändigt, kan processen köras flera gånger och det neuronnät som har konsekvent bäst träffsäkerhet kan användas.

Genom att granska resultatet i figur 1 kan det synas hur vid varje omgång så förbättras populationen från en slumpmässig sammanställning neuronnät vid första generation till en population där nästa alla individer är av samma, effektiva design (från 45% till 55%). Även om vid tillfällen med fler/olika meta-variabler så kräver den evolutionära processen fler generationer och större populationer för att klara av samma förbättring, så kommer den itera-

tiva processen alltid att behöva gå igenom alla kombinationer vilket då leder till att den evolutionära processen ger stora tidsfördelar med nästan samma resultat i det flesta fall.

5.1 Felkällor

En fällkälla som uppstår vid denna undersökning är att datorn kan inte alltid leverera samma prestanda vid alla tillfällen, särskilt om den används under samma tid. Dock är detta systematiskt för båda processen, och skillnaden är troligen så liten att den inte kunde påverka resultaten.

5.2 Fortsatta studier

Fler studier bör göras med andra sammanställningar meta-variabler och fler omgångar, för att testa den evolutionära processen med svårare starttillstånd. Det är troligt att vid denna studie fanns ett få antal optimala neuronnät (träffsäkerhet på 54–55%) existerande redan i populationen för första generationen av ren slump. En intressant undersökning skulle kunna testa om det evolutionära systemet kan utveckla det optimala kombination bara med hjälp av naturligt urval, istället för att bara ”sprida” det effektivast neuronnät genom populationen.

5.3 Slutsats

Slutsatsen för denna studie är att en evolutionär process för urval av meta-variabler kan kraftigt förkorta tiden det krävs för en sådan sökning. Vid tillfällen med långa inlärningstider/stort antal meta-variabler lönar sig metoden mest, och den kan öppna dörrar för nya appliceringar av maskininlärning där förut utvecklingstiden har varit ett hinder för programmerare.

Referenser

Clevert, Djork-Arné, Thomas Unterthiner och Sepp Hochreiter (2015). ”Fast and accurate deep network learning by exponential linear units (elus)”. I: *arXiv preprint arXiv:1511.07289*.

Harvey, Matt (april 2017). *Let's evolve a neural network with a genetic algorithm—code included*. `https://blog.coast.ai/lets-evolve-a-neural-network-with-a-genetic-algorithm-code-included-8809bece164`. (Accessed on 10/05/2018).

Hopfield, John J (1988). "Artificial neural networks". I: *IEEE Circuits and Devices Magazine* 4.5, s. 3–10.

Jain, Anil K, Jianchang Mao och K Moidin Mohiuddin (1996). "Artificial neural networks: A tutorial". I: *Computer* 29.3, s. 31–44.

Karlik, Bekir och A Vehbi Olgac (2011). "Performance analysis of various activation functions in generalized MLP architectures of neural networks". I: *International Journal of Artificial Intelligence and Expert Systems* 1.4, s. 111–122.

Kingma, Diederik P och Jimmy Ba (2014). "Adam: A method for stochastic optimization". I: *arXiv preprint arXiv:1412.6980*.

Krizhevsky, Alex, Vinod Nair och Geoffrey Hinton (2014). "The CIFAR-10 dataset". I: *http://www.cs.toronto.edu/kriz/cifar*.

Walia, Anish S (juni 2017). *Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent*. `https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f`. (Accessed on 10/05/2018).

Xu, Bing m. fl. (2015). "Empirical evaluation of rectified activations in convolutional network". I: *arXiv preprint arXiv:1505.00853*.

Yao, Xin och Yong Liu (1997). "A new evolutionary system for evolving artificial neural networks". I: *IEEE transactions on neural networks* 8.3, s. 694–713.