Evolutionära Neurala Nätverk, ett Effektivitetsstudie

Tomass Wilson
thmwi@kth.se

5 oktober 2018

Sammanfattning

Sammanfattning

Innehåll

0.1 Inledning

Programmerare har alltid velat lösa problem, hellre med hjälp av en dator. När Artificiella Neurala Nätverk (ANN) började utvecklas kunde man applicera dem på problem som föredetta verkade omöjliga, såsom att urskilja ansikten eller kategorisera bilder (Hopfield 1988). Det blir en allt större och viktigare del av moderna program och då finns det stor anledning till att försöka korta ner tiden det tar för att bygga ett sådant nätverk. Inlärningsprocessen tar ofta flera timmar och bygger på att slumpa fram de kopplingarna mellan noder (neuroner) sådan att en viss korrekthet på resultatet nås för det specificerade problemet. Effektiviteten för ANN brukar definieras vid 2 variabler; inlärningstid och korrekthet, och detta mäts med hjälp av standardiserade tester. En viktig faktor som påverkar inlärningen av en ANN är meta-variabler, såsom antal noder per lager, antal lager, aktiveringsfunktionen, etcetera. Dessa meta-variabler kan ställas in manuellt, men detta brukar leda till att nätverket inte når sitt maximala precision, 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.

0.1.1 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.

0.1.2 Frågeställning

Vilken inställning av Matt Harveys neurala nätverk har kortast inlärningstid?

0.2 Bakgrund & Teori

0.2.1 Artificiella Neurala Nätverk

En Artificiell Neural Nätverk (ANN) är en samling sammankopplade noder sammanställda i flera lager. Varje nod har ett värde och en aktiveringsfunktion. En nod har flera inkommande länkar och utgående länkar till andra noder. 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 noder i input- och outputlagern bestäms i förhand beroende på hur nätverket ska användas.

0.2.2 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 test värden mellan 64 – 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, nadam* (Kingma och Ba 2014)

0.2.3 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, som borjar inlärningen tills den har nått en viss träffsäkerhet. Tiden detta tar mäts sedan , vilket då blir nätverkets *fitness*. Nätverket i populationen som hade 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 ska inte bli för homogen). Efter flera ”generationer” har förmodligen det bästa nätverket en sammanställning meta-variabler som är optimerade för användningsområdet. (Yao och Liu 1997)

0.2.4 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

0.3 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). All kod skrevs i python 3.6, och använder sig till huvuddel av tqdm, keras och tensorflow paketen för att optimera och skapa nätverken. Versionerna av samtliga paket ses i tabell 1.

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 det tog antecknades. Detta ger en basnivå som kan jämföras med den evolutionära processen. Sedan kördes den evolutionära processen genom att köra main.py filen, och tiden skrevs ned.

0.4 Resultat

0.5 Diskussion

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

Litteratur

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.