# 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 \parencite{hopfield1988artificial}. 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 sitt 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.

## Syfte

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

## Frågeställning

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

# Bakgrund

## Artificiella Neurala Nätverk

En Artificiell Neural Nätverk (ANN) är en samling sammankopplade neuroner sammanställda i flera \textit{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 \textit{aktiveringsfunktionen} och slutgiltiga resultatet är nodens \textit{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 \textit{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 \textit{optimeringsfunktion}.

## Meta-Variabler

För att ett ANN ska kunna skapas behövs några startparametrar. Dessa \textit{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

\newline

Detta är antal individuella neuroner i varje lager. Detta experiment kommer testa värdena 64, 128, 256, 512, 768 och 1024 neuroner per lager.

1. # Lager

\newline

Detta är antal lager exklusive input- och outputlagern, dessa heter \textit{gömda lager} och antalet varierar mellan 1 - 4

1. Aktiveringsfunktion

\newline

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 \parencite{jain1996artificial}. Detta experiment använder sig av aktiveringsfunktionerna: \textit{sigmoid, tanh} \parencite{karlik2011performance}, \textit{Exponential Linear Unit} (ELU) \parencite{clevert2015fast} och \textit{Rectified Linear Unit} (ReLU) \parencite{xu2015empirical}

1. Optimeringsfunktion

\newline

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 \parencite{TypesofO34:online}. Funktionerna som ska testas är: \textit{rmsprop, sgd, adam, adagrad, adadelta, adamax, nadam} \parencite{kingma2014adam}

## Evolutionär Algoritm

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 \textit{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 \textit{generationer} har förmodligen det bästa nätverket en sammanställning meta-variabler som är optimerade för användningsområdet. \parencite{yao1997new}

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, 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).

## CIFAR-10

Industri-standarden för att testa ANN är CIFAR-10, en databas av 60 000 bilder, som kategoriseras in i 10 grupper \cite{krizhevsky2014cifar}. 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 neuroner, med ett värde mellan 0 och 1 för att visa hur mycket bilden ”passar” in i en viss kategori. Träffsäkerheten mäts som andel gånger neuronnätet har placerat bilden i korrekt kategori.

# 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 \parencite{harvey2017} 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 \ref{paketversioner}. Hårdvaran som användes i denna undersökning ses i tabell \ref{hårdvara}.

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.

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

GPU: RTX 2080 @ 1950 MHz core & 7263 MHz mem

CPU: Intel i7 8700 @ 4.28 GHz

Minne: 32 GB @ 2666 MHz

# 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 \ref{evo}, och för den iterativa i tabell \ref{iterativ}. Totala tiden för den evolutionära algoritmen samt iterativa ses i tabell \ref{total}.

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 \ref{generation}.

# Diskussion

Enligt resultatet som redovisas i tabell \ref{total} så kan frågeställningen enkelt besvaras. En skillnad på 10h 10m visar på den stora fördelen med att använda den evolutionära processen. 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 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.

Genom att granska resultatet i figur \ref{generation} kan vi se 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%). Vid tillfällen med fler/olika meta-variabler kommer troligen den evolutionära processen kräva fler generationer och större populationer för att klara av samma förbättring, men det iterativa processen kommer alltid att behöva gå igenom alla kombination vilket då leder till att den evolutionära processen ger stora tidsfördelar med nästan samma resultat. I tabell \ref{evo}