# 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 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. Effektiviteten för ANN brukar definieras vid 2 variabler; inlärningstid och träffsäkerhet, och detta mäts med hjälp av standardiserade tester. Industri-standarden för sådana test är CIFAR-10, en databas av 60 000 bilder som nätverket får kategorisera in i 10 grupper \parencite{krizhevsky2014cifar}. Med detta kan man okomplicerat jämföra olika ANN. 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, men detta brukar leda till att nätverket 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 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.

## Frågeställning

Vilken inställning av Matt Harveys neurala nätverk har kortast inlärningstid?

# 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

Detta är antal individuella neuroner i varje lager. Detta experiment kommer test värden mellan 64 – 1024 neuroner per lager.

1. # Lager

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

1. 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 \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

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, som borjar inlärningen tills den har nått en viss träffsäkerhet. Tiden detta tar mäts sedan och de nätverk i populationen som tog längst tid ”dödas” och ersätts av nätverk som är en kombination av 2 levande nätverk, ”föräldrarna” , och får sedan en slumpmässigt 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. \parencite(yao1997new)

## 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 nätverket har placerat bilden i korrekt kategori

# Metod

För att jämföra de två olika metoder för att bestämma bästa meta-variabler använder vi av CIFAR-10 databasen för materialet. Själva processen av populationskapande, inlärning, mätning och evolution är skriven av Matt Harvey \parencite{harvey2017}. All kod skrivs 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 är som följande:

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

# Resultat

# Diskussion