Projekt II – Neuralt nätverk i ett inbyggt system

Mål

- Kunna implementera neurala nätverk från grunden via mjukvara och använda för godtyckliga applikationer.
- Känna till och kunna beskriva begrepp såsom *feedforward, backpropagation* samt aktiveringsfunktioner.
- Erhålla fördjupad kunskap inom C++.

Uppgiftsbeskrivning

- Ni arbetar som embeddedutvecklare på ett företag i Göteborg. Er chef, Erik Pihl, har gett dig (och eventuellt din kollega) i uppgift att skapa ett POC *(Proof Of Concept),* med syftet implementera ett neuralt nätverk i ett inbyggt system, tillämpat på en Raspberry Pi med programkod skriven i C++ eller Python. Det finns två varianter av uppgiften. Ni väljer en av dessa.

Variant 1 – Egen applikation med neuralt nätverk

I denna variant ska ni själva definiera vad det neurala nätverket ska användas till, gärna kopplat till något ni själva är intresserade av (exempelvis spel, musik, sport, elektronik, IoT, hobbyprojekt och så vidare).

Krav

För G:

1. Nätverket ska implementeras från grunden i C++ eller Python (inga externa ML-bibliotek).
2. Koden ska vara skriven i enlighet med modern stil för det använda språket, med samtliga publika gränssnitt dokumenterade.
3. Nätverket ska inneha ett dolt lager. Antalet noder i respektive lager ska kunna väljas av användaren.
4. Nätverket ska vara statiskt, dvs. lärhastigheten samt antalet epoker som genomförs vid träning ska väljas av användaren.
5. Nätverket ska användas i ett inbyggt eller inbyggtslikt sammanhang, exempelvis på en Raspberry Pi.
6. Det ska finnas en tydlig koppling mellan insignaler, nätverkets prediktion och någon form av utsignal/visualisering, exempelvis lysdioder, motorer, displayer eller terminalutskrifter.
7. Projektet ska laddas upp till ett repo på GitHub, som er chef bjuds in till. Systemet ska också demonstreras live för chefen.

För VG:

8. Nätverket ska inneha ett godtyckligt antal dolda lager. Antalet noder i respektive lager ska kunna väljas av användaren.
9. Nätverket ska vara adaptivt till den grad att träning genomförs till prediktionen är mycket nära 100 %. Nätverket ska därmed utvärdera träningen regelbundet och ändra lärhastigheten utefter detta. Användaren ska därmed inte ange antalet epoker eller lärhastigheten som ska användas vid träning, utan detta hanteras internt.

Exempel på möjliga projektidéer:

- Ett neuralt nätverk som avgör om en spelare vinner eller förlorar i ett enkelt spel baserat på knappsekvenser.
- Ett system som klassificerar olika knappmönster eller sensordata och styr flera lysdioder.
- Ett enkelt ”smart hem”-scenario där nätverket avgör om en lampa ska tändas baserat på flera insignaler (knappar, timers, sensorer).
- Ett musik- eller rytmrelaterat projekt där knapptryckningar tolkas och klassificeras.

Variant 2 – Defaultuppgift

- Det neurala nätverket ska kunna prediktera en utsignal via insignaler från 4 - 5 tryckknappar (beroende på betygsnivå). Vid nedtryckning ska en given tryckknapp medföra insignal 1.0, annars 0.0. Nätverkets prediktion ska utgöras av en enda signal, som ska användas för att styra en lysdiod.

- Nätverket ska tränas till ett detektera ett 4- eller 5-bitars XOR-mönster. Om ett ojämnt antal tryckknappar är nedtryckta ska lysdioden tändas, annars ska den vara släckt. Därmed ska det neurala nätverket tränas till att prediktera en hög utsignal om en, tre eller fem knappar är nedtryckta, annars en låg utsignal.

- Samtidigt som lysdioden tänds eller släcks ska det neurala nätverkets predikterade utsignal skrivas ut i Linuxterminalen, tillsammans med information gällande om lysdioden tänds eller släcks. Prediktion ska enbart ske vid förändring av insignalerna. När modellen har tränats ska prediktion ske på använd träningsdata och skrivas ut i terminalen, så som visas i bilaga A (för prediktion av ett 5-bitars XOR-mönster).

- Se till träning sker direkt när systemet startar och att nedtryckning av någon eller några av tryckknapparna inte medför någon prediktion förrän träningen är slutförd.

Krav

För G:

1. Nätverket ska implementeras från grunden i C++ eller Python (inga externa ML-bibliotek).
2. Koden ska vara skriven i enlighet med modern stil för det använda språket, med samtliga publika gränssnitt dokumenterade.
3. Nätverket ska inneha ett dolt lager. Antalet noder i respektive lager ska kunna väljas av användaren.
4. Nätverket ska vara statiskt, dvs. lärhastigheten samt antalet epoker som genomförs vid träning ska väljas av användaren.
5. Nätverket ska tränas till att prediktera ett 4-bitars XOR-mönster.
6. Vid förändring av insignaler ska prediktionen samt information om lysdiodens tillstånd skrivas ut i Linux-terminalen.
7. Projektet ska laddas upp till ett repo på GitHub, som er chef bjuds in till. Systemet ska också demonstreras live för chefen.

För VG:

8. Nätverket ska inneha ett godtyckligt antal dolda lager. Antalet noder i respektive lager ska kunna väljas av användaren.
9. Nätverket ska vara adaptivt till den grad att träning genomförs till prediktionen är mycket nära 100 %. Nätverket ska därmed utvärdera träningen regelbundet och ändra lärhastigheten utefter detta. Användaren ska därmed inte ange antalet epoker eller lärhastigheten som ska användas vid träning, utan detta hanteras internt.
10. Nätverket ska via 3 – 5 dolda lager prediktera ett 5-bitars XOR-mönster. I systemet erhålls sedan insignaler från fem tryckknappar, där den predikterade utsignalen skrivs till en lysdiod.

- Träning ska ske direkt vid systemstart och ingen prediktion får ske innan träningen är avslutad.

- Drivrutiner från *libgpiod* (C++) tillhandhålles här och kan med fördel användas för avläsning av tryckknapparna samt styrning av lysdioden. Repot med drivrutinerna kan klonas med följande kommando:

git clone git@github.com:Yrgo-24/rpi-gpio-drivers.git

Genomförande och examination

- Ni får arbeta enskilt eller i grupper om två.
- Ni får rådfråga er chef, som är en gammal utvecklare, för handledning gällande konstruktionen.
- Ni är välkomna att ta inspiration från externa källor, vilket även innefattar kod från klassrepot, men ni ska göra en egen implementering. Kopiera alltså inte någon annans kod.
- Koden ska laddas upp på GitHub. Er ”chef” (eller rättare sagt lärare) kommer genomföra en review via denna plattform. Systemet ska också demonstreras live för chefen.
- **OBS!** Bjud in ”er chef” till repot så att han kan titta på koden.

Utvärdering

- I samband med inlämningen, vänligen lämna in en kort diskussion i en README-fil där ni beskriver följande:
  1. Vad lärde ni er av projektet?
  2. Vad var lätt/svårt?
  3. Vad hade ni velat ha lärt er mer innan projektet?
  4. Övriga kommentarer?

Bilaga A – Exempelutskrift av predikterade värden

- Nedan visas utskrift av predikterade värden från ett neuralt nätverk innefattande fem inportar samt en utport. Nätverket har tränats till att prediktera ett 5-bitars XOR-mönster. Efter träning är precisionen uppe i 100 %:

```
--------------------------------------------------------------------------------
Prediction accuracy: 100.0%
Input: [0.0, 0.0, 0.0, 0.0, 0.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [0.0, 0.0, 0.0, 0.0, 1.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [0.0, 0.0, 0.0, 1.0, 0.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [0.0, 0.0, 0.0, 1.0, 1.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [0.0, 0.0, 1.0, 0.0, 0.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [0.0, 0.0, 1.0, 0.0, 1.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [0.0, 0.0, 1.0, 1.0, 0.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [0.0, 0.0, 1.0, 1.0, 1.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [0.0, 1.0, 0.0, 0.0, 0.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [0.0, 1.0, 0.0, 0.0, 1.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [0.0, 1.0, 0.0, 1.0, 0.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [0.0, 1.0, 0.0, 1.0, 1.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [0.0, 1.0, 1.0, 0.0, 0.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [0.0, 1.0, 1.0, 0.0, 1.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [0.0, 1.0, 1.0, 1.0, 0.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [0.0, 1.0, 1.0, 1.0, 1.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [1.0, 0.0, 0.0, 0.0, 0.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [1.0, 0.0, 0.0, 0.0, 1.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [1.0, 0.0, 0.0, 1.0, 0.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [1.0, 0.0, 0.0, 1.0, 1.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [1.0, 0.0, 1.0, 0.0, 0.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [1.0, 0.0, 1.0, 0.0, 1.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [1.0, 0.0, 1.0, 1.0, 0.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [1.0, 0.0, 1.0, 1.0, 1.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [1.0, 1.0, 0.0, 0.0, 0.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [1.0, 1.0, 0.0, 0.0, 1.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [1.0, 1.0, 0.0, 1.0, 0.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [1.0, 1.0, 0.0, 1.0, 1.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [1.0, 1.0, 1.0, 0.0, 0.0], prediction: [1.0], reference: [1.0], error: 0.0
Input: [1.0, 1.0, 1.0, 0.0, 1.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [1.0, 1.0, 1.0, 1.0, 0.0], prediction: [0.0], reference: [0.0], error: 0.0
Input: [1.0, 1.0, 1.0, 1.0, 1.0], prediction: [1.0], reference: [1.0], error: 0.0
--------------------------------------------------------------------------------
```