Kandidat 19 - Komma-igång guide

Kandidatgrupp 2018

20 januari 2019

1 Introduktion

Syftet med denna guide är att ge information till Håkans och Andreas kandidatarbetesgrupp 2019, så att de snabbt kan ta sig förbi några av de praktiska momenten i projeket och istället kunna spendera mer tid på problemlösningsaspekterna av projektet. Utöver att gå igenom rent praktiska moment i detalj, så som datagenerering och hur man använder datorfarmen Kebnekaise, så kommer även en viss genomgång göras av koden för de neurala nätverken som använts. Denna genomgång kommer framförallt att ske i separata tutorial dokument som ni hittar i den delade GitHub repositoryn.

2 Problemformulering

Anderas och Håkan kommer ge en grundlig beskrivning av problemet, men i korthet är problemet som man önskar lösa att rekonstruera vad som skedde vid en reaktion med hjälp av en omslutande detektor. Detta problem är särskilt komplicerat om partiklarna har en benägenhet att deponera sin energi i flera olika detektorkristaller, och en förhoppning är att det ska gå att överträffa dagens rekonstruktionsmodeller med hjälp av artificiella neurla nätverk.

Den detektor som önskas studeras är Califa som ska detektera γ-fotoner och protoner. En äldre och mindre komplex detektor än Califa är Crystall Ball (XB) detektorn som också har för uppgift att detektera γ-fotoner och protoner. Problemet med att partiklar deponerar sin energi i flera kristaller uppkommer i båda dessa detektorer, så det är ingen större inskränkning att studera Crystall Ball detekton istället för Califa. Under vårt projekt undersökte vi endast händelser med endast γ-fotoner och bortsåg alltså från de med protoner.

3 GitHub Repositories

Strukturen i vårt GitHub projekt KandidatarbeteSubatFys följer här. Projektet har två repositorys, TransferFrom2018To2019 och really-bad-data-collection-and-some-machine-learining. Det är framförallt TransferFrom2018To2019 som ni kommer ha användning av tror vi. Den innehåller tutorials, datagenererings-skript, skript för att lägga upp jobb på datorfarmen kebnekaise och även mappen FinalizedPythonPrograms. Den mappen innehåller några av våra slutgiltiga program som låg till grund för resultaten i rapporten, men det är tillskillnad från tutorialsen inte säkert att det är så trevligt att använda dem som utomstående. Repositoryn really-bad-data-collection-and-some-machine-learining var den vi använde under projektets gång, men i slutet var vi inte jätteduktiga på att lägga upp saker, allt är inte jätte användarvänligt och väldigt mycket förlegade program ligger där, men det är en resurs ni har till hands iallafall.

4 Datagenerering

För att generera data som de neurala nätverken kan tränas på används ett simuleringsverktyg för subatomärfysik kallat *GEANT4*. Detta program var inte installerat på datorena med de krafiga grafikkorten (pclab-110{225,232}), men finns på exempelvis datorn *vale* och *planck-o* (där vale är betydligt snabbare på att simulera). GEANT4 har ett komplicerat interface, så därför är det rekommenderat att använda det av Håkan utvecklade programmet *ggland* för att kommunicera med GEANT4. Så efter att ha ssh:at in på vale, be Håkan om information för att installera ggland. När ggland är installerat och du står i mappen du installerade det i, gå in i mappen ./land02/scripts/ggland och skriv sedan in . geant4.sh för att starta GEANT4. För att simulera skriver man nu ./land_geant4 följt av olika input-parametar. Se gglands hemsida för information om de olika specifikationerna som går att göra.

En typisk simulering som vi gjorde var:

```
./land_geant4 --gun=gamma,E=0.01:10MeV,isotropic
 --gun=gamma,E=0.01:10MeV,isotropic --gun=gamma,E=0.01:10MeV,isotropic
 --events=1000000 --xb=tree-gun-edep --tree=filename.root --np
```

som ska tolkas som ett kommando, det vill säga allt ovan är på samma rad. Termen --gun=gamma,E=0.01:10MeV,isotropic kommunicerar att en gamma foton

vid varje event ska skjutas isotropt (från origo eftersom inget annat anges) och dess energi är en slumpvariabel med likformig fördelning mellan 0.01 och 10 MeV. Eftersom vi har tre repetitioner av denna term så kommer tre gamma fotoner att avfyras vid varje event. --events=... specificerar antalet event, men eftersom en del av eventen inte resulterar i några excitationer så kommer antalet registrerade event med standardinställningarna att vara aningen lägre. Termen --xb specificerar att detektorn är Crystall Ball (XB) och utökar man den till --xb=tree-gun-edep så sparas även data om hur mycket energi varje enskild gamma foton deponerade i varje kristall (något som kan vara användbart för att sortera ut missvisande event för träningen av de neurala nätverken). --tree=gunlist,filename.root specificerar namnet på datafilen och gunlist-delen gör att man får ut *gunn*-variabeln som beskriver antalet guns, även när antalet guns är ett. --np är ett kommando som alltid bör vara med då det gör så att beräkningar görs parallellt och därmed förkortas processen avsevärt (är antal event väldigt litet så får man dock error med --np specificeringen).

Nästa steg är att extrahera den önskade datan från Root-filen, och spara den i textfiler, vilka i sin tur kan användas som indata till de neurala nätverken. Ett mellansteg man kan göra innan detta är att analysera root-filen direkt i programmet Root.

För att öppna Root skriver man bara root i terminalen (finns installerat på alla datorer på avdelningen verkar det som). Root är ett mångsidigt program och det finns mycket information på nätet om hur man använder det. Men vi presenterar det mest nödvändliga här. I Root-terminalen skriver man TFile *a = TFile::Open("filename.root") för att läsa in en root-datafil. Nu när datafilen är inläst har man tillgång till TTree:et h102, så det går till exempel att printa ut de olika grenarna med h102->Print() och plota grenen *XBe* med h102->Draw("XBe"), där XBe innhåller informationen om uppmätt energideponering i detektorns kristaller.

Vi har såklart redan gjort skript som automatiserar extraheringen av data från Root-filer som ni finner i mappen DataGeneration i repositoryn, men i detta stycke beskrivs den grundläggande processen som dessa skript utför. För att extrahera data från h102 TTree:et till en textfil så skriver man först h102->MakeClass() i Root. Detta skapar två filer, h102.h och h102.C. Filen h102.C är skriven i C++ och innehåller framförallt en for-loop som loopar över varje event. För att skriva över data till en textfil modifierar man denna fil i en editor och specificerar där

vad som ska göras med datan för varje loop (event) (t.ex. skriva ut vissa variabler till textfiler). I h102.h filen finns de tillgängliga grenarna man kan använda i loopen, som t.e.x XBe. När h102.C filen är färdigmodifierad läser man in den i Root-terminalen med .L h102.C, sedan skapar man ett objekt av denna klass med h102 t och när man därefter kör t.Loop() kommer ens modifierade for-loop att exekveras och ens önskade data skrivs till ens specificerade textfiler.

När ni fått ordning på Python så rekomenderas att konvertera textfilerna till *npa*-filer (alternativt till en zip av npa-filer, en npz-fil). Detta eftersom det kan ta lång tid att läsa in datan från textfiler i Python medan npa-filer läses in betydligt snabbare.

5 Information om TensorFlow och Python

För att använda TensorFlow så använder man sig förslagsvis av Python och importerar tensorflow modulen. Det verkar vara vanligt att programmera använda sig av high-level programmering i TensorFlow, t.ex. Keras biblioteket verkar vara mycket använt (t.ex. Christian Forssén använder det mycket verkar det som) och TensorFlow själva verkar förespåka Estimator klassen. I början av projektet testade vi olika approacher, och vi tyckte att det blev smidigare att programmera low-level.

För att få en bra genomgång över hur vi byggde upp våra nätverk, men också över datagenereringen, rekommenderas GeneralTutorial.pdf med tillhörande tutorial.py skript som ni hittar i TransferFrom2018To2019/Tutorials/GeneralTutorial mappen. GeneralTutorial.pdf innehåller även en del annan information som kanske skulle passat bättre i detta dokument, så ni bör titta igenom den när ni ni ska börja använda Python och TensorFlow. Vi hade också rekommenderad TensorFlows egna tutorial Deep MNIST for Experts som vi hade användning av, men den verkar inte ligga uppe längre. Efter att ha blivit välbekant med fully-connected lager har vi även en tutorial för nätverk med convolution tillämpat som ni hittar i mappen ConvolutionTutorial.

De slutgiltiga program python-program vi använde oss av finns i mappen TransferFrom2018To2019/FinalizedPythonPrograms. Allt i den mappen är inte användarvänligt för utomstående så vi rekommenderar nog att ni med hjälp av tutorialsen och kanske genom att studera programmen i FinalizedPythonPrograms mappen, gör egna program eftersom programmen har ganska få rader kod och det förmodligen

är lärorikt. En viktig detalj i våra gamla program är vi använde en tidsineffektiv och onödigt komplicerad metod för att hitta den permutation som minimerar kostnadsfunktionen (så ni kan hoppa över avsnitt 3.4.3 i rapporten). Istället är det bara att använda tf.min.

6 Utvärdering av de neurala nätverken

I GeneralTutorial presenteras ett enkelt sätt att utvärdera hur väl nätverket presterar. Två nackdelar med denna metod är att utvärderingsdatan är inte är realistisk, d.v.s. den är brusfri och icke-boostad (se teori-avsnittet i vår rapport för information om boosting). Dessutom så görs ingen jämförelse mot den konventionella metoden, Addback.

Att simulera realistisk data kan förstås göras med flera olika parameterval, men i mappen RealisticData finns ett skript som simulerar data med de parameterval vi använde. Använder ni denna data för utvärderingen, blir det blir tydligt när ni har utvecklat ett nätverk som presterar bättre än våra. I figur 4.1 i rapporten presenteras hur väl våra nätverk presterar på den realistiska datan jämfört med Addback-rutinen.

Nätverken vars prestation presenteras i rapporten var alla tränade på brusfri och icke-boostad data, d.v.s. inte på realistisk data. En poäng med ett sådant upplägg är bland annat att nätverket inte blir biased för vad vi anser vara realistisk data, men det finns inget som säger att man inte kan gräva djupare än vad vi hann göra med att t.ex. även träna nätverken på boostade fotoner.

7 Använda datorfarmen Kebnekaise

Efter ha att fått ett inlogg på Kebnekaise bör man följa instuktionerna på deras hemsida och göra en pfs mapp som man kör sina skript ifrån. Enklast är att lägga upp sina jobb med ett SBATCH skript. I mappen TransferFrom2018To2019/Kebnekaise ligger skiptet time.sbatch och jobbfilen som den kör är job_script.sh. Hur dessa tillämpas och modifieras för olika jobb står i readme-filen. Några användbara kommandos är när man loggat in på Kebnekaise är: squeue -u <user> (ser vilka aktiva jobb en viss användare har) och scancel <jobID> (avbryt ett jobb) och projinfo (ger info om de projekt du är med i). Alla dessa kommandoes (och fler

därtill) finns beskrivna på HCP2N:s hemsida.

Det finns tre huvudtyper av processorer på Kebnekaise farmen: Intel CPU:er, Nvidia GPU:er och Intel:s Knights Landing CPU:er (något slags försök att utmana Nvidia kring grafikintensiva beräkningar). Generellt är GPU:er bäst för att träna neurala nätverk och därför var det nästan uteslutande dem vi använde. Dock gjorde vi i slutet av projektet tester där vi tränade ett av nätverken på en vanlig Intel CPU, och just i det fallet så gick det ungefär lika fort som det hade gått på någon av GPU:erna. Men för Marcus Polleryds nätverk (han gjorde ett exjobb liknande detta projekt året innan oss) så var det betydligt snabbare att använda GPU:er, vilket kan ha berott på att hans nätverk innehöll convolution-lager tillskillnad från det vi testade. Timkostnaden är betydligt högre för GPU:erna än de vanliga Intel CPU:erna, så presterar CPU:erna tillräckligt bra är det ingen poäng att köra på GPU:erna. Något som vi definitivt kunde slå fast var att Knights Landing CPU:erna inte var värda sin höga timkostnad.