Kandidat 19 - Kort Convolution Tutorial

Kandidatgrupp18

18 januari 2019

1 Introduktion

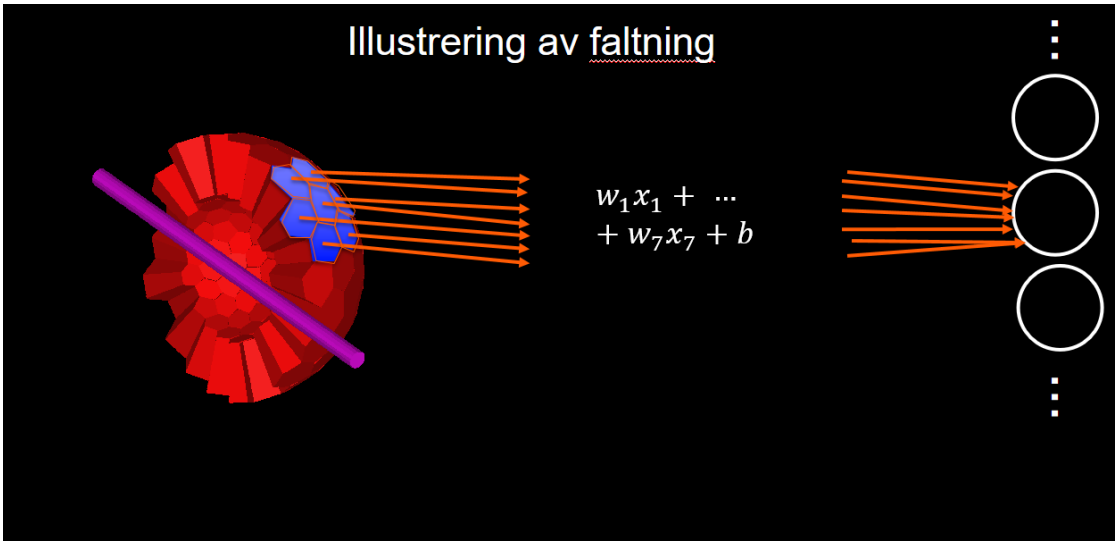
Denna tutorial är gjord av kandidatgruppen med motsvarande projekt 2018 och syftet med den är att ge en kort och konkret genomgång av en enkel tillämpning av convolution på Crystal Ball (XB) detektorn som vi gjorde. Vi provade en hel del tillvägagångssätt, men för att hålla tutorialen så överblickbar som möjligt så presenteras bara ett av sätten. Gör GeneralTutorial med skriptet tutorial.py innan du gör denna.

I tutorialen kommer vi använda två convolution-lager på samma nätverk som användes i den generella tutorialen i tutorial.py skriptet. Under vårt projekt så utvärderade vi främst convolution på nätverk som hade som uppgift att bestämma antal partiklar (vi hade skäl att göra detta och hann inte göra en riktig utvärdering på de vanliga nätverken, dvs de som skulle förutsäga energi och vinklar). Så vi lyckades bara visa positiva effekter av convolution-lager vid multiplicitets fallet, men förhoppningsvis kan ni, efter att ha bekantat er med de vanliga fully-connected nätverken, göra framsteg mha convolution tillämpat på nätverk för rekonstruktion av vinklar och energier. Dock behöver er convolutions-approach inte nödvändigtvis följa den vi presenterar här, en nackdel med vår är att t.ex. att vissa kristaller har mindre än 6 kristaller, så filtren tränas på olika formationer.

2 Beskrivning av convolution och vår metod

I tuturial.py skriptet i den generella tutorialen så ges kristallenergierna som en radvektor till nätverket och så får nätverket lista ut hur de olika kristallerna förhåller sig tillvarandra. Syftet med convolutions-lager är att utnyttja att vi vet om geometrin hos XB detektorn och på så sätt göra nätverket mer effektivt.

Figuren nedan försöker illustrera vad vi försöker åstadkomma. Den blåa markeringen vandrar runt över alla kristaller och spottar ut sig ett värde för varje område. Värdet som spottas ut beror på w (w_1 ... w_7) och b variablerna, och dessa uppdateras under träningsprocessen. Så vi går alltså från 162 energier till 162 nya värden efter convolution-lagret. Den blå markeringen som vandrar runt och producerar värden m.h.a. w och b variablerna kallas för ett filter, och man brukar ha flera filter i varje lager, och varje filter kommer förhoppningsvis att tränas att känna igen olika typer mönster. Eftersom ett convolutions-lager med ett filter gav 162 värden så kommer ett lager med t.ex. 8 lager ge 162x8 värden som tensorflow blir på dimensionen 1x162x8.



Sättet vi implementerar att filter vandrar över XB-detektorn är att vi först skapar en 162·7 lång radvektor, där de första sju elementen är kristall nummer 1 (se t.ex. pappfigur av XB-detektorn för numreringen) följt av de sex grannarna till kristall 1. Grannarna är ordnade så att den första grannen är den närmast strålutgången och övriga följer i moturs riktning. De sju elementen efter det innehåller motsvarande värden, fast centrerade runt kristall 2 o.s.v. För att gå från de 162 energivärdena till den 162·7 långa rad vektorn använde vi matrismultiplikation, eftersom denna operation saktar ned träningsprocessen så lite som möjligt.

Med den 162·7 långa radvektorn uppbyggd så låter man en 1x7 variabel-vektor vandra över den med steglängd 7 (dvs inget överlapp) och vid varje steg fås

skalärprodukten av det variabel-vektor och det den täcker som output (plus en bias också). På så sätt har vi lyckats implementera det vi ville.

Som sagt, om vi hade 8 filter i första convoltion-lagret blev vår output på formen 1x162x8. Vill vi ha ytterligare ett convolutions-lager omvandlar vi först matrisen med samma procedur som ovan, och vi får då den på formen 1x162·7x8. Vanligt när man har fler än ett lager är att utöka filtrenas storlek till att även inkludera z-dimensionen. Så i vårt fall kommer filtrena i det andra convolution-lagret att ha storlek 1x7x8, och kommer att vandra över 1x162·7x8 outputen från lager 1 med steglängd 7 i y-riktingen, så att outputen från varje filter i lager 2 blir på formen 1x162. Med t.ex. 16 filter i lager två blir outputen från convolutions-lager två: 1x162x16. Vill man ha ytterligare convolutions-lager gör man samma procedur som för convolutions-lager 2.

Det sista steget är att ta outputen från convolutions-lager 2 och platta ut det till en $1 \cdot 162 \cdot 16$ vektor, som sedan använda denna som input till ett fully-connected nätverk, som det i tutorial.py.

3 Själva tutorialen

Tutorialen fås framförallt av att läsa kommentarerna till varje steg i convolution_tutorial.py.

För att köra convolution_tutorial.py skriptet, behövs matrisen som omvandlar en 162 lång kristallenergi-vektor till den $162 \cdot 7$ långa grann-energi-vektorn. Den fås genom att köra skriptet make_tutorial_convolution_matrix.py:

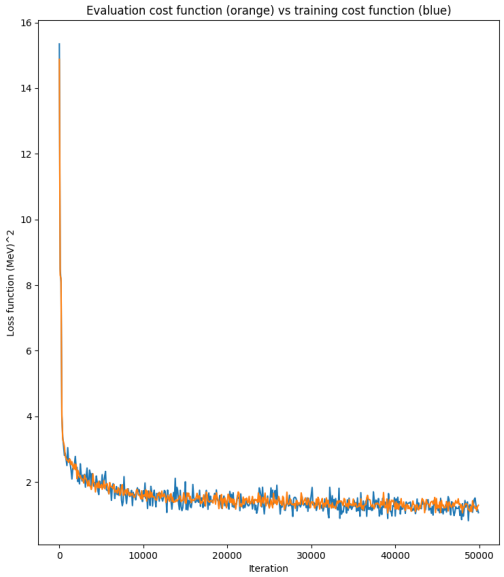
```
python3 make_tutorial_convolution_matrix.py
```

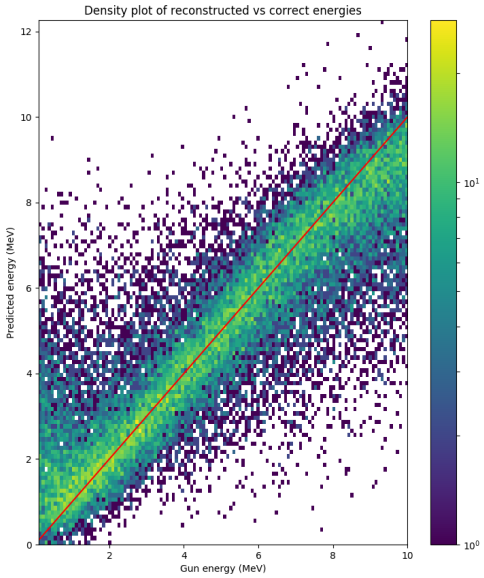
vilket kommer att ge tutorial_conv_mat.npy filen. För det ska fungera att köra skriptet ovan måste make_tutorial_convolution_matrix.py filen ligga i samma mapp som geom_xb.txt textfilen. geom_xb.txt innehåller information om geometrin för XB detektorn, bland annat om varje kristalls grannar.

För att köra huvud-skriptet, så gör på samma sätt som i GeneralTutorial (använd samma data som där, kom ihåg att nätverket bara klarar 3 guns) och kör programmet på likande sätt:

```
python3 convolution_tutorial.py XB_det_data.txt XB_gun_vals.txt .
```

där tutorial_conv_mat.npy filen måste ligga i samma mapp för att skriptet ska fungera. Du bör nu få upp en figur liknande denna efter en stund:





Resultatet med convolution ser bättre ut än det som ficks utan convolution i den generella tutorialen med tutorial.py skriptet. Dock så har vi många fler variabler i nätverket med convolution pga av att outputen från convolutions-lager 2 är 162x16 vilket gör viktmatrisen från convolutionslagren till de fullt kopplade lagren, stor.