Juul Arthur Ribe Rudihagen og Hans Kristian Henriksen

TDT4171 - Øving 5

Introduksjon

Denne rapporten beskriver oppbygningen av koden for backwardspropagation-algoritmen som er implementert i øving 5.

Implementasjon

Det er gjort endringer i begge de utdelte klassene. Under er disse endringene beskrevet. I tillegg til disse forklaringene kan man se på kommentarene i koden.

`dataLoaderSkeelton.py`

Metoden runRanker() er her endret for å legge til dataene fra trenings- og testsettet i de riktige listene. Vi starter med å gå igjennom treningsdataene. Her sammenligner vi hver datainstance med alle de andre. Dersom verdien til de to instansene vi sammenligner er ulik legges de til i listen slik at den med høyest verdi ligger først.

Nøyaktig den samme fremgangsmåten benyttes for testsettet.

Etter at listene med trenings- og testdata er satt opp kalkuleres en initiell verdi for feilen i nettverket. Deretter starter en løkke hvor vi først trener nettverket, og deretter tester det på nytt. Når dette er utført det ønskede antall ganger plottes en graf som viser utviklingen i feilprosent over tid.

For å lage grafen med snitt av 5 kjøringer er det skrevet en mengde kode i slutten av filen som kaller runRanker() 5 ganger, registrerer resultatene, og tar gjennomsnittet av disse. Det finnes egne kommentarer i denne delen av koden.

`Backprop_skeleton.py`

Metoden computeOutputDelta() benytter formelen fra oppgaveteksten til å beregne outputdeltaet. O_a er her prevOutputActivation, mens O_b er OutputActivation.

For computeHiddenDelta() benyttes også funksjonene fra boken. Vi oppdaterer det gjeldende punktet i prevDeltaHidden for A, og deltaHidden for B.

Funksjonen updateWeights følger oppgavetekstens formel i den første doble for-løkken. Her oppdateres inputvektene i matrisen basert på inputActivations. I den andre for-løkken oppdateres vektenes output med bakgrunn i hiddenActivations.

backpropagate() kaller ganske enkelt computeOutputDelta(), computeHiddenDelta() og updateWeights() i korrekt rekkefølge.

Metoden train() tar inn alle parene med resultater som vi skal sammenligne, og går igjennom dem en etter en. For hvert par propageres nettverket, før backpropagate() kalles.

countMisorderedPairs() sjekker hvilket dokument vårt nettverk mener er det beste, og kontrollerer deretter hvilket som faktisk var det beste. Bom og treff telles opp, og en andel feil returneres.

Resultater

Etter å ha kjørt koden 5 ganger med 25 iterasjoner i hver kjøring har vi oppnådd følgende resultater. Merk at grafene beskriver feilprosent.

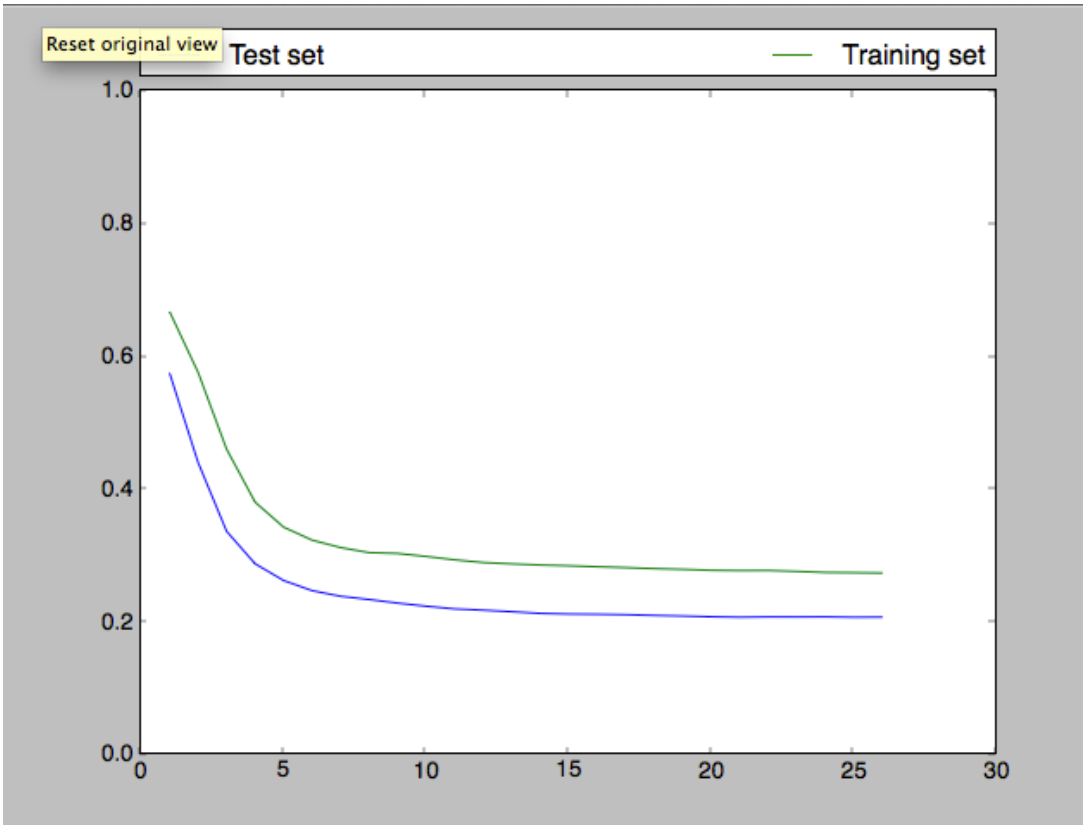

Fig 1

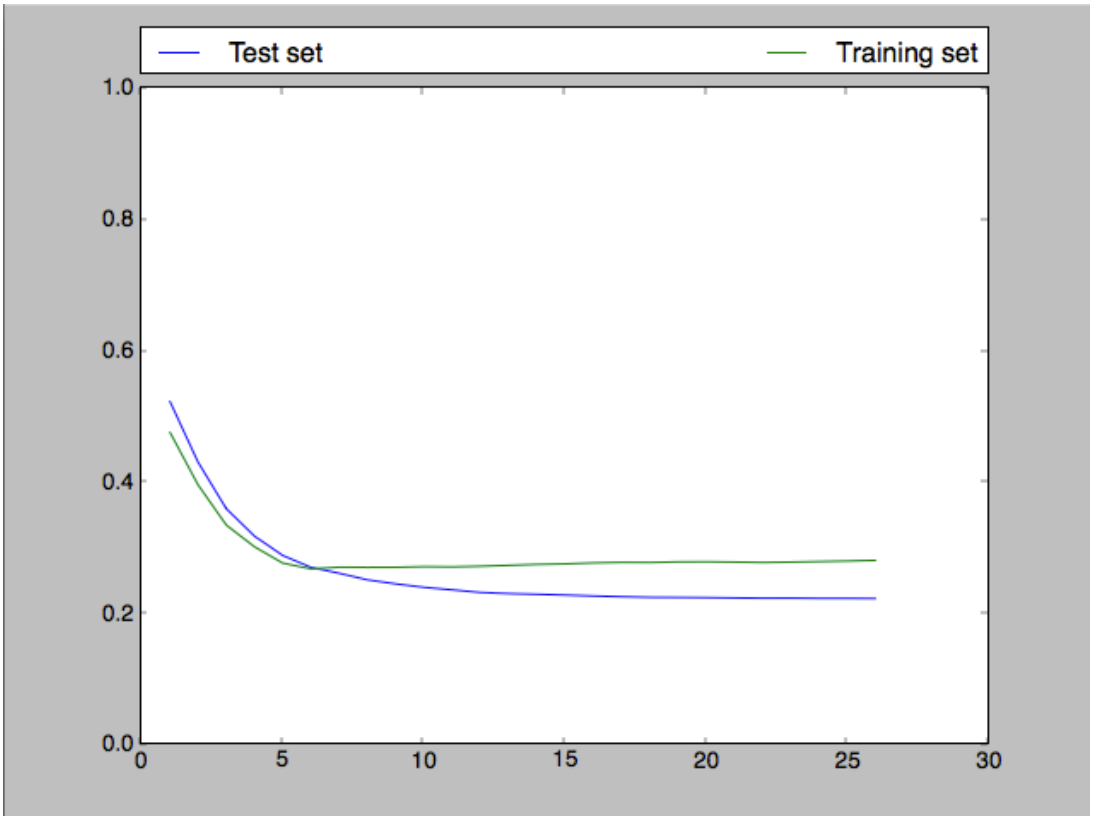

Fig 2

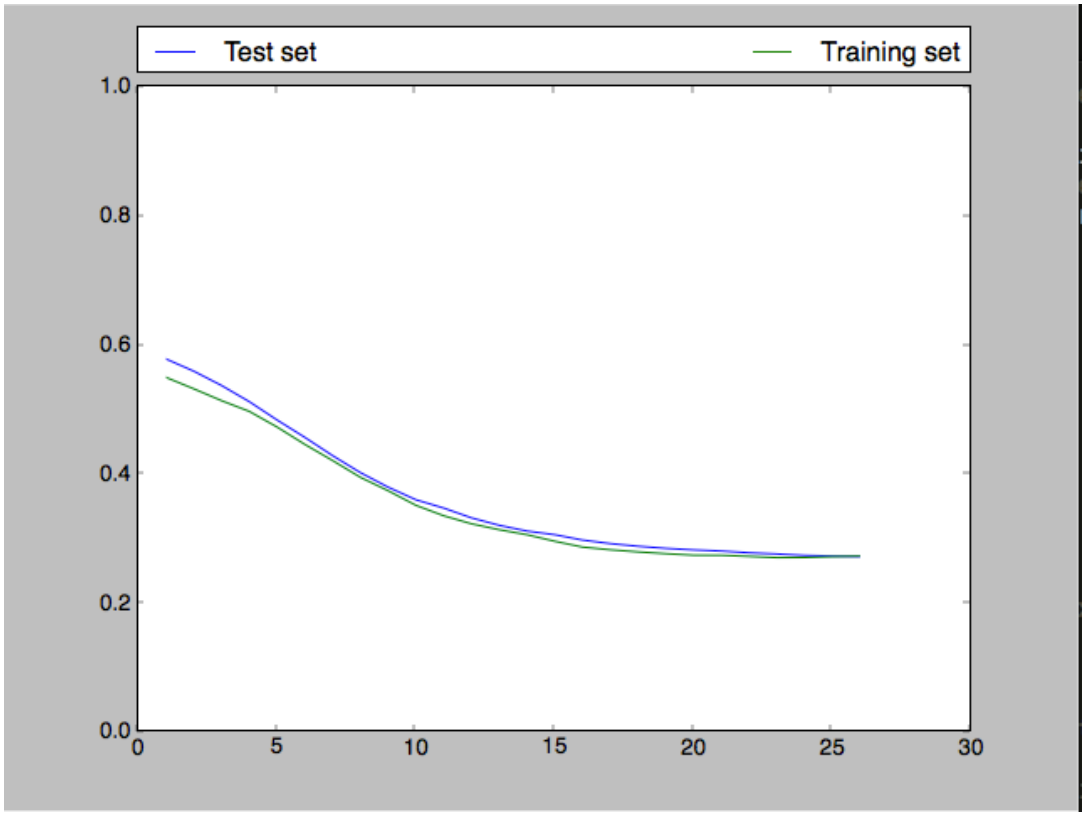
Test set
Training set
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30

Fig 3

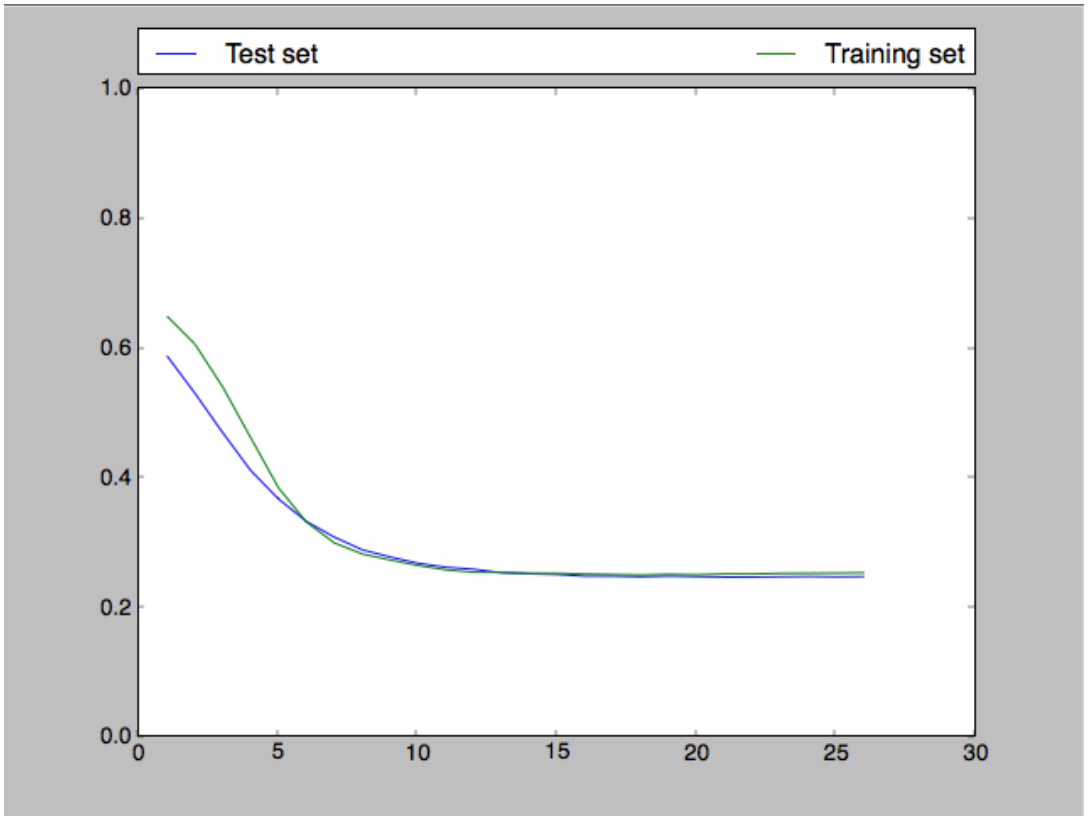
Test set
Training set
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30

Fig 4

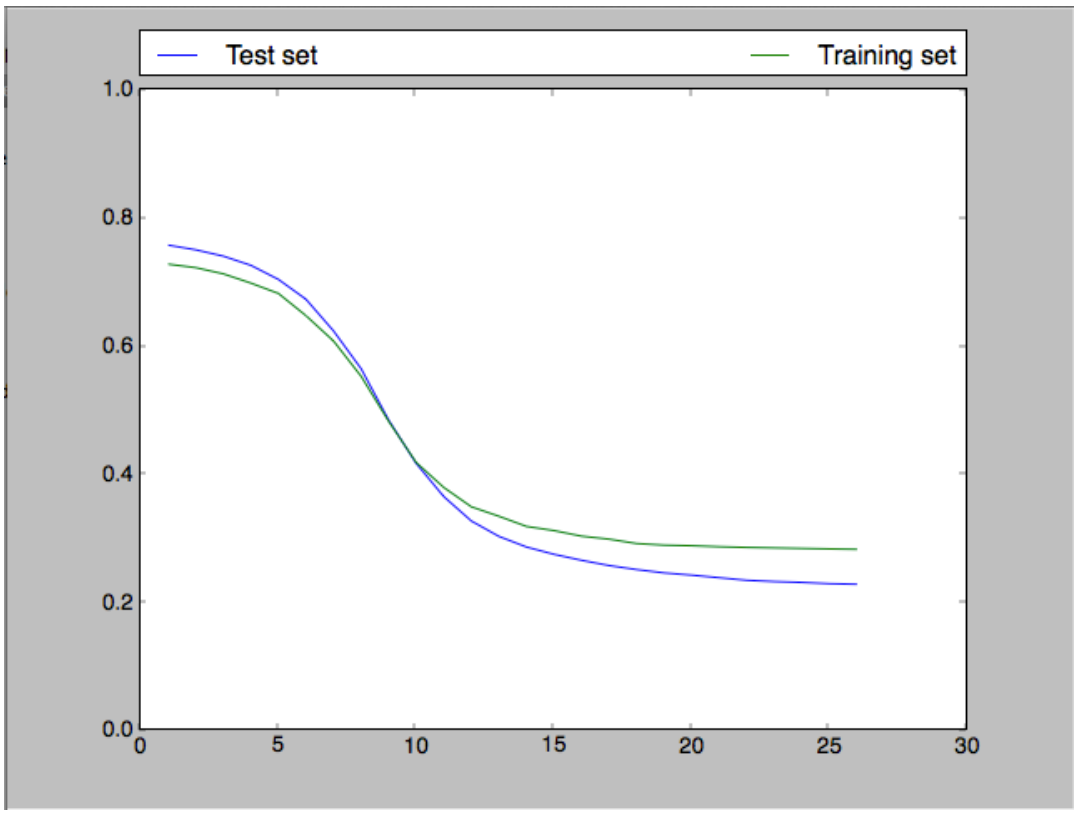

Fig 5

Vi ser at det stort sett er små forskjeller mellom de to settene. Der det er forskjeller er det gjerne slik at treningsettet kommer dårligere ut enn testsettet. Vi ser at det i Fig 1 er slik at treningsettet starter vesentlig dårligere enn testsettet, og også ender dårligere, mens det i Fig 2 og Fig 5 starter bedre enn testsettet, men ender dårligere. I Fig 3 og Fig 4 blir settene så godt som like gode etter 25 iterasjoner.

Vi ville ha forventet at treningsettet skulle ha kommet bedre ut, da det er dette settet nettverket trenes på, og dermed burde være bedre rustet til å gjøre gode antagelser for. Vi er dermed overrasket over at testsettet kommer så godt ut som det gjør.

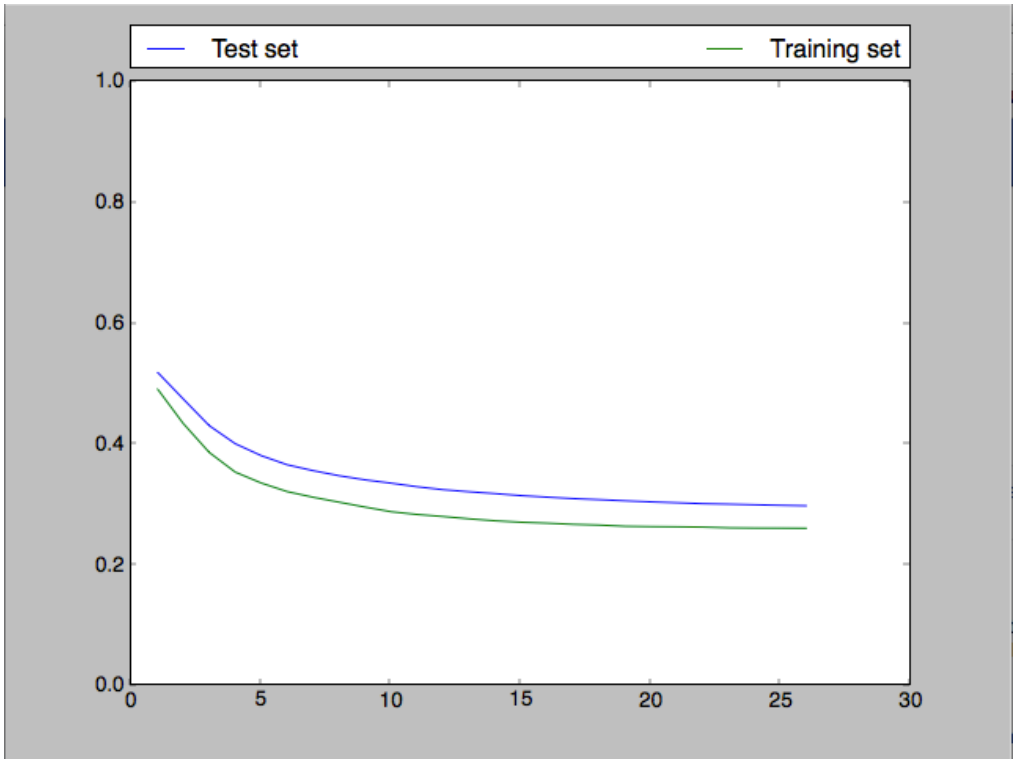

Fig 6 - Gjennomsnitt

I Fig 6 ser vi gjennomsnittet av 5 kjøringer (merk, dette er 5 andre kjøringer enn de i Fig 1 til Fig 5). Her ser vi at utflatingen skjer som vi forventer. I dette tilfellet er treningsettet bedre enn testsettet, slik vi ville forventet, men ikke så i de 5 opprinnelige kjøringene. Dette kan være et utfall av de tilfeldige verdiene som velges på starten, og kan bare avgjøres ved å gjøre et statistisk tilstrekkelig utvalg.

Dersom vi hadde kjørt programmet i langt flere iterasjoner tror vi at resultatene ville ha holdt seg på cirka det nivået vi ser nå. Det virker som at vi flater kraftig ut etter ca 20 iterasjoner. Vi regner med at det ville ha blitt noe oscillasjon rundt dette punktet, da vi ser at vi ved utflating har tendenser til noe forverring før vi igjen faller.

Implementasjonsutfordringer

Under implementasjonen hadde vi noen utfordringer med å se hvilke variabler fra formlene som tilsvarte variabler i koden, og pseudokoden i boken, da disse bruker relativt ulike navngivningskonvensjoner.

Formlene som stod i boken var også ganske forskjellige fra de som var oppgitt i oppgaveteksten. Formelen for å oppdatere weightsOutput var heller ikke oppgitt i oppgaveteksten, selv om oppgaveteksten ga inntrykk av å være et komplett sett av formler.

Vi opplever noen ganger noe merkelig oppførsel fra koden. I enkelte tilfeller opplever vi at feilraten ikke kommer under det oppgitte nivået på 25%. Vi har også opplevd at vi, når feilraten kommer svært lavt, får en liten økning i feil. Vi har snakket med studass, og avgjort at koden skal være riktig. Dette virker å være en iboende egenskap ved algoritmen.

Gruppemedlemmenes bidrag

Vi har i hovedsak jobbet sammen med oppgaven, og par-/paralellprogrammert. Vi har ikke valgt å tildele noen spesielle oppgaver, men heller arbeidet felles med alle aspekter av innleveringen.