

# Bestemming av sekundærstrukturar i proteinstrukturar

av  
Vidar Lund

Hovudoppgåve for graden Cand. Scient.



Universitetet i Bergen  
Institutt for Informatikk  
22. Juni 2005



## Forord

Protein er ein type molekyl som spelar ei viktig rolle i korleis dei fleste organismar fungerer. Ein viktig faktor for kva funksjon proteinet skal ha er korleis det er oppbygd – dvs. kva struktur det har. Strukturen er oppbygd av sekundærstrukturelement (SSE). SSE gir proteinet eit rammeverk som er festepunkt for dei funksjonelle gruppene.

Det finnes i dag fleire program for bestemming av SSE i strukturar. Det mest brukte for bestemming av sekundærstruktur når strukturen er kjent, er Define Secondary Structure of Proteins (DSSP).

Oppgåva her går ut på å utvikla eit alternativt system, der avstandsmatriser og Kunstige Nevrale Nettverk (KNN) blir brukt. Utgangspunktet er då ei mengde av strukturar (testmengde) – representert som avstandsmatriser der SSE er bestemt. Avstandsmatriser viser alle avstandane mellom aminosyrene i eit protein. Samanhengen mellom verdiane i avstandsmatriser og SSE-typar må då bli funnet. Dette blir sannsynligvis best gjort ved bruk av KNN. KNN er ein måte å bruke datamaskinar på som er god til å klassifisere data.

## Organisering av oppgåva

Første del av oppgåva er kapittel som skal gi lesaren bakgrunnsinformasjon som er relevant for oppgåva.

Kapittel 1 tar for seg den biologiske bakgrunnen, Protein DataBase (PDB), samt ein lengre forklaring av kva oppgåva omhandlar.

Kapittel 2 omhandlar avstandsmatriser, som er ein såpass viktig del av oppgåva at eg syns dei treng ein lengre forklaring.

Kapittel 3 tar for seg Kunstige Nevrale Nettverk og dei paradigma eg har testa ut i løpet av oppgåva. I tillegg står det her ei lengre beskriving av Stuttgart Neural Net Simulator (SNNS).

Andre del beskriver kva og korleis eg har prøvd å løyse oppgåva.

Kapittel 4 beskriv dei tre forskjellige måtane å bygge opp KNN på som eg har testa ut. Dette kapitlet forklarar i praksis kva eg har gjort for å løysa oppgåva.

Kapittel 5 har ein meir inngående beskriving av dei programma eg har laga, og i tillegg ei beskriving av korleis eg brukte SNNS.

Kapittel 6 viser kva forsøk eg har kjørt og resultatata.

Til slutt samanfatar kapittel 7 oppgåva og gir nokre idear for vidare arbeid.

## Takk til:

Foreldra mine, Audny og Jan Ståle Lund, for gjennomlesing av oppgåva og framfor alt for mykje støtte gjennom heile studiet mitt.

Linda Ramsevik for verdifull hjelp med den biologiske bakgrunnen.

Frode Meland for gjennomlesing av oppgåva.

Rettleiar Ingvar Eidhammer for god rettleiing gjennom heile oppgåva.

# Innhold

Kapittel 1: Biologisk Bakgrunn .....	7
1.1 Frå DNA til Protein.....	7
1.1.1 Proteintypar og funksjon.....	8
1.1.2 Protein i celler .....	8
1.2 Proteinstruktur .....	9
1.2.1 Sekundærstrukturelement (SSE).....	10
1.2.1.1 Heliksstruktur.....	10
1.2.1.2 $\beta$ -tråd/flak-struktur.....	11
1.2.1.3 Loop-struktur .....	11
1.2.1.4 Supersekundære struktur.....	11
1.2.2 Bestemming av proteinstruktur.....	11
1.2.2.1 Røntgenstråle-krystallografi (X-ray Crystallography).....	11
1.2.2.2 NMR (Nuclear Magnetic Resonance).....	11
1.2.2.3 Prediksjon frå sekvens .....	12
1.3 Kvifor bestemme strukturen .....	12
1.4 Bestemming av SSE frå struktur.....	12
1.4.1 DSSP (Define Secondary Structure of Proteins) .....	13
1.5 PDB – The Protein DataBase.....	13
1.5.1 PDB-filer.....	14
1.6 Kva oppgåva omhandlar .....	14
Kapittel 2: Avstandmatriser .....	15
2.1 Samanhengen av avstandsmatriser og koordinatar .....	15
2.2 Korleis kan avstandsmatriser brukast til bestemming av SSE?.....	15
Kapittel 3: Kunstige Nevrale Nett.....	17
3.1 KNN, kva er det? .....	17
3.1.1 Framoverfasen.....	17
3.2 Læring .....	18
3.2.1 Backpropagation .....	18
3.2.2 Backprop Momentum .....	20
3.3 SNNS (Stuttgart Neural Net Simulator) .....	20
Kapittel 4: Kva har blitt gjort? .....	23
4.1 Tankar rundt inn og utdata i oppgåva .....	23
4.2 Kva KNN paradigme skal brukast? .....	23
4.3 Forskjellige topologiar som har blitt testa ut .....	24
4.3.1 Å mate inn heile avstandmatrisa direkte.....	24
4.3.2 Å mate inn $\frac{1}{4}$ av matrisa .....	25
4.3.3 Mange små matriser i kaskade langs diagonalaksen.....	26
4.3.3.1 Versjon 1: Isolerte subnettverk .....	27
4.3.3.2 Versjon 2: Slå saman for fleire r.....	29
4.3.3.3 Versjon 3: Bruk av ”nabonettverk”.....	31
Kapittel 5: Implementering .....	33
5.1 Tygg-programmet .....	33
5.1.1 Innlesing.....	33
5.1.2 Lag avstandsmatrise.....	33
5.1.3 Generer mønsterfil (.pat).....	33

5.2 SNNS .....	34
5.3 Analyseprogrammet .....	35
5.3.1 Outputfiler frå analyseprogrammet .....	36
5.3.1.1 Analysefila (.analyse) .....	36
5.3.1.2 Oversiktsfil (.kort.analyse).....	36
5.3.1.3 Samandragfil (.kort.kort.analyse).....	37
5.3.2 Endringar i analyseprogrammet i versjon 2 .....	37
5.4 Input-programmet .....	37
5.5 Del2-programmet .....	38
5.5.1 Outputfiler frå Del2-programmet.....	38
5.5.1.1 Analysefila (.analyse) .....	38
5.5.1.2 Mønsterfila (.pat) .....	39
5.5.1.3 Feilloggfil (.analysefeil.log).....	39
5.5.1.4 Samandragfila (sammendrag.log) .....	39
5.6 Sortsammendrag .....	39
5.7 Kva programmer blei gjenbrukt i versjon 3 .....	40
Kapittel 6: Kjøringar og resultat .....	41
6.1 Dei store nettverka som jobba på heile matrisa .....	41
6.1.1 300x300: .....	41
6.1.2 200x200: .....	42
6.1.3 Konklusjon .....	42
6.2 Dei store nettverka som jobba på ¼ av matrisa .....	43
6.2.1 Konklusjon .....	44
6.3 Kaskade.....	44
6.3.1 Versjon 1 .....	44
6.3.1.1 1doble-dim og 2doble-dim:.....	45
6.3.1.2 3doble-dim: .....	45
6.3.1.3 4doble-dim, 5doble-dim, 6doble-dim, 7doble-dim og 8doble-dim .....	45
6.3.1.4 Tabellar over kva verdisett eg jobba vidare med .....	46
6.3.1.5 Resultata eg tok med vidare til versjon 2.....	48
6.3.2 Resultata frå versjon 2 .....	49
6.3.3 Versjon 3 .....	50
6.3.3.1 Tabellar for netta i versjon 3 .....	51
6.3.3.2 Dei endelege resultata med alle residyar .....	51
6.3.3.3 Dei endelege resultata med bare korrekt bestemte residyar.....	52
6.3.3.4 Konklusjon over versjon 3 .....	53
6.3.4 Konklusjon.....	53
Kapittel 7: Konklusjonar og idear for vidare arbeid. ....	55
7.1 Vidare arbeid.....	55
7.2 Idear for vidare forsøk .....	55
Kapittel 8: Bibliografi .....	57
Appendix: Innhald på cdrom'en .....	59

# Kapittel 1: Biologisk Bakgrunn

Dette kapitlet er i hovudsak basert på følgjande kjelder: [1], [2], [3] og [4].

## 1.1 Frå DNA til Protein

Alle organismar har koda korleis dei skal sjå ut og fungere i eit molekyl som blir kalla DNA (DeoxyriboNucleicAcid). Dette molekylet finnes i alle cellene til organismen. Hos eukaryoter er det lokalisert i cellekjernen.

DNA består av to strengar med dei fire nukleotidene: Adenine (A), Guanine (G), Cytosine (C) og Thymine (T). Dei to strengane er tvinna saman i ein dobbel heliks. Dei to strengane er bundne til kvarandre ved at A dannar hydrogenbindingar til T, og G til C. Kvar enkelt nukleotid binder bare til ein annan, og dette medfører at den enkelte strengen alltid er komplementær til den andre. Ein kan derfor rekonstruere heile heliksen, sjølv om ein bare har ein av strengane. DNA'et lagrar all genetisk informasjon i ein organisme. Den delen av DNA-strengen som kodar for eit spesifikt protein, blir kalla eit gen, mens heile DNA-strengen blir kalla organismens genom. Genomet gis vidare til neste generasjon gjennom reproduksjon.

Protein spelar ei nøkkelrolle i cellene og deltar i nesten alle cellulære aktivitetar. Delemna i proteinet er polypeptidar som igjen er oppbygd av aminosyrer – ofte referert til som residyar. Aminosyresekvensen (primærstrukturen 1.2) blir koda frå ein DNA-streng.

- Først blir ein DNA-streng kopiert og danna om til pre mRNA (messenger). Dette blir kalla transkripsjonsfasen.
- I neste fase – spleising, blir denne igjen omdanna til mRNA. Her blir enkelte delar av RNA'et plukka bort (intron), mens andre delar (exons) blir beholdt for å danne mRNA'et. Akkurat kva deler som er kva, tilhøyrer eit forskingsfelt som blir kalla alternativ spleising. Det er framleis mye usikkerheit på dette feltet.
- Siste fasen blir kalla translasjon. Her blir mRNA'et oversatt til protein, etter den universelle genetiske koden (tabell 1.1).

RNA er ein tredje type molekyl. Det minner mykje om DNA. Forskjellen er at Thymine blir erstatta med Uracil, og RNA er vanlegvis ein lang enkel tråd.

Aminosyresekvensen blir koda frå mRNA-tråden etter den universelle genetiske koden (sjå tabell 1.1). Det finnes fire nukleotidar som skal kode 20 aminosyrer, og derfor bruker ein 3 nukleotidar pr aminosyre. Dette gir ein viss grad av redundans i kodane som vist i tabell 1.1.

Sjølve oversettinga skjer ved at mRNA'et dannar ei forbindelse med ribosom i cella. Denne forbindinga "les" mRNA'et, og binder til seg tRNA (transfer) som bærer med seg kvar si aminosyre og legger denne til i den voksande proteinstrengen.

A	Ala	Alanine
C	Cys	Cysteine
D	Asp	Aspartic acid
E	Glu	Glutamic acid
F	Phe	Phenylalanine
G	Gly	Glycine
H	His	Histidine
I	Ile	Isoleucine
K	Lys	Lysine
L	Leu	Leucine
M	Met	Methionine
N	Asn	Asparagine
P	Pro	Proline
Q	Gln	Glutamine
R	Arg	Arginine
S	Ser	Serine
T	Thr	Threonine
V	Val	Valine
W	Trp	Tryptophan
Y	Tyr	Tyrosine

	First position	Second position			Third position	
		U	C	A	G	
U		Phe	Ser	Tyr	Cys	U
		Phe	Ser	Tyr	Cys	C
		Leu	Ser	Stop	Stop	A
		Leu	Ser	Stop	Trp	G
C		Leu	Pro	His	Arg	U
		Leu	Pro	His	Arg	C
		Leu	Pro	Gln	Arg	A
		Leu	Pro	Gln	Arg	G
A		Ile	Thr	Asn	Ser	U
		Ile	Thr	Asn	Ser	C
		Ile	Thr	Lys	Arg	A
		Met	Thr	Lys	Arg	G
G		Val	Ala	Asp	Gly	U
		Val	Ala	Asp	Gly	C
		Val	Ala	Glu	Gly	A
		Val	Ala	Glu	Gly	G

**Tabell 1.1** Ein og tre bokstavskodane som representerer aminosyrene, og den universelle genetiske koden som blir brukt for å kode DNA om til protein. Henta frå [1].

### 1.1.1 Proteintypar og funksjon

Det finnes fleire forskjellige typar protein som har spesielle funksjonar i organismen. Nokon av desse er:

- **Strukturprotein:** organismens primære byggeblokker.
- **Enzym:** desse utfører eller katalyserer dei fleste av kroppens biokjemiske reaksjonar. Dei er som oftast veldig spesifikke og har bare ein spesiell funksjon, men enkelte kan delta i fleire reaksjonar.
- **Regulerande protein:** t.d. hormon, slik som veksthormonet i kroppen som regulerer veksten.
- **Lagringsprotein:** t.d. hemoglobin som transporterer oksygen i blodet.
- **Transmembrane protein:** Vedlikeheld og regulerer cellestorleik og miljø.

### 1.1.2 Protein i celler

Cellene i ein organisme dannar dei fleste protein etterkvart som dei har behov for dei. Det hadde ikkje vore nok plass om alle mulige protein skulle vore tilgjengelig i cella til ei kvar tid. Nokon protein vil det alltid finnast i cellene. Om eit protein finnes i ei celle er avhengig av om det er eit protein cella ofte har bruk for, og av kor fort den eventuelt kan lage proteinet. Proteinets sin livssyklus kan beskrivast som tre fasar: Danning, funksjon og nedbrytning.

## 1.2 Proteinstruktur

Når proteinet nettopp er danna er det forma som ein lang streng. Denne strengen blir kalla primærstrukturen til proteinet. Dette er ikkje ein struktur, men meir ein sekvens av aminosyrer. Den blir kalla primærstruktur, eller ofte ei poly-peptid-kjede. Lengda på ein proteinsekvens kan variere. To døme er: insulin som har 51 residyar, og titin som har ~28.000 residyar.

Protein folder seg saman i relativt stabile 3D strukturar, og denne strukturen er antatt å vere den forma der proteinet har sitt minimale energinivå.

Dei forskjellige aminosyrene i sekvensen har kvar sine spesielle eigenskapar som påverkar korleis dei bind seg til andre aminosyrer. Eigenskapane som varierer mest er om aminosyrene er positivt eller negativt lada, og om dei er hydrofobe eller hydrofile. Naturen vil utifrå sekvensen av aminosyrer folde proteinet slik at det får ein stabil struktur. Det blir danna bindingar mellom aminosyrene i proteinet når det foldar seg til sin struktur. Dette er fordi energi blir frigjort når det dannes bindingar, og at det då kan oppnå sin struktur med eit minimalt energinivå. Kvar sekvens har sine bestemte måtar å danne sin struktur på.

Strukturen til proteinet kan ofte vise oss kva funksjon det har. Til dømes så kan eit protein vere danna som ei saks, og ha eit aktivt punkt akkurat i krysset. Dette proteinet vil då ha ein kuttefunksjon. Andre protein kan vere forma som ei tønne, og vil då ofte ha ein porefunksjon, som kan regulere kva som slepper inn i cella eller ikkje.

Det som blir rekna som neste nivå i proteinet sitt strukturhierarki er sekundærstrukturelement (SSE). Der  $\alpha$ -heliks og  $\beta$ -flak er dei to mest vanlige. Dei blir igjen heldt saman av andre strukturar kalla Loop eller Turn/Coil.

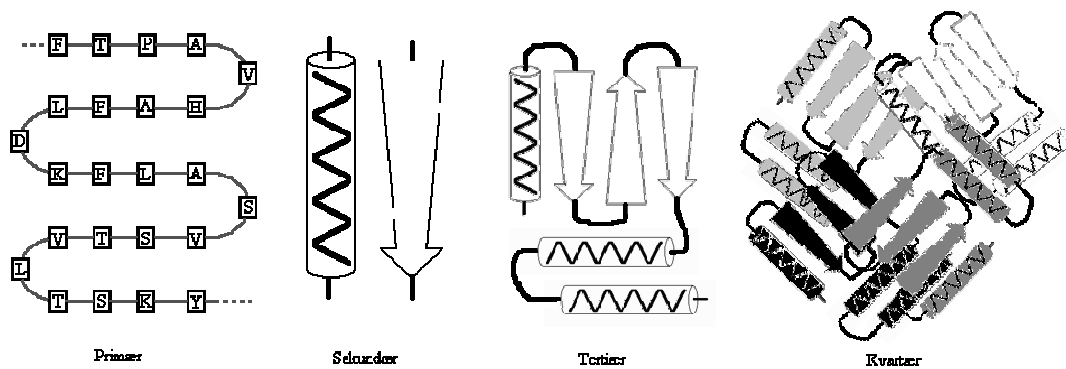
Ein proteinstruktur skal oppfylle ein del krav:

- Lav energikonformasjon av dei individuelle residyane
- Hydrogenbindingar mellom polare grupper
- Danning av ein god kompakt struktur.

Ein teknikk kalla Sasisekharan-Ramakrishnan-Ramachandran plot kan vise oss korleis proteinet oppfyller det første kravet. Kuleforma protein inneheld områder med heliksar og  $\beta$ -trådar som er forbundet med turns eller loops, og dette gjer at det kan danne ein kompakt struktur. Heliksane og  $\beta$ -trådane går gjennom midten av proteinet, mens turn- og loopstrukturane dannar overflata til proteinet. Heliksane blir danna ved at residyar dannar hydrogenbindingar til andre residyar som ligger forholdsvis nært i sekvensen. Residyane i  $\beta$ -trådane dannar som oftast hydrogenbindingar til andre residyar som ligger langt fram i sekvensen og dannar slik  $\beta$ -flak. Sekundærstrukturelement er betre beskrive i kapittel 1.2.1.

Neste nivå er tertiærstrukturen som er ein folding av heile aminosyrekjeden. Denne er ein relativt stabil og kompakt 3D struktur. Den er ofte eit resultat av alle dei forskjellige tiltrekkande og fråstøytande kreftene, som verkar mellom atoma/aminsyrene i proteinet (hydrogenbindingar, positive/negative ladningar, hydrofobe/file eigenskapar, osv).

Øvste nivå er kvartærstrukturen som er samansetningar av fleire aminosyrekjeder til ein eining. Til dømes hemoglobin som består av fire kjeder som kvar kan binde til seg eit jernmolekyl.



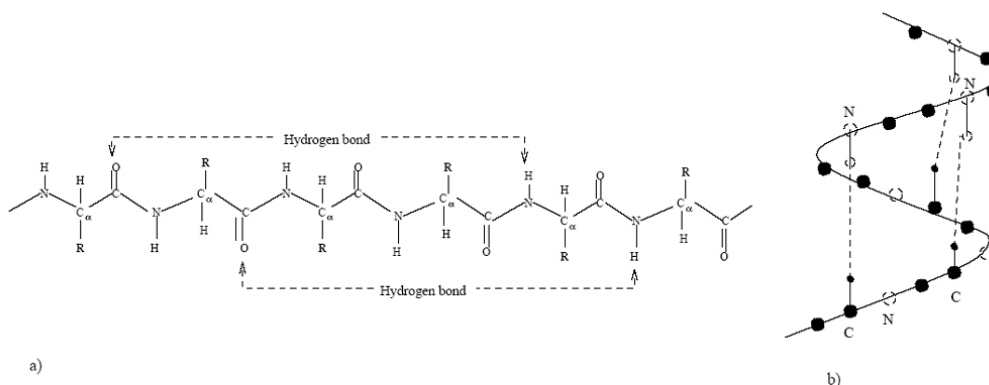
**Figur 1.1:** Typiske måtar å grafisk framstille dei 4 strukturnivåa til proteinet. Sekundærstrukturen er her vist med ein heliks og ein  $\beta$ -tråd. Tertiærstrukturen er vist med 3 heliksar og eit antiparallelt  $\beta$ -flak. Kvartærstrukturen er bygd opp av 4 utgåver av tertiærstrukturen.

### 1.2.1 Sekundærstrukturelement (SSE)

Primærstrukturen er sterkt polar og dermed hydrofil, med ein hydrogenbindingsdonor NH, og ein hydrogenbindingsmottakar  $C'=O$  i kvar peptideining. I hydrofobiske miljø må desse polare gruppene nøytraliserast ved danning av hydrogenbindingar. Dette blir løyst ved at det blir danna sekundærstrukturelement, som t.d.  $\alpha$ -heliks og  $\beta$ -flak. Sekundærstrukturelementa gir proteinet eit rigid og stabilt rammeverk, som dei funksjonelle gruppene til proteinet blir festa til. Eventuelt så vil dei funksjonelle gruppene feste seg til loop/turn strukturane i overflata som binder saman dei andre sekundærstrukturelementa.

#### 1.2.1.1 Heliksstrukturar

Den mest vanlige heliksstrukturen er  $\alpha$ -heliks.  $\alpha$ -heliksar dannes ved at ei aminosyre dannar ein hydrogenbinding til ei anna aminosyre som ligg 4 residyar lenger fram i sekvensen. Meir spesifikt så dannar  $C=O$  gruppa i residy  $i$  ein hydrogenbinding med HN gruppa til  $i+4$  som vist på figur 1.2. Med unntak av residyanne på endane så får alle residyanne i ein heliks brukt sitt potensiale for å binde seg i hydrogenbindingar. Gjennomsnittlig lengde på ein  $\alpha$ -heliks er 10 residyar. Mange  $\alpha$ -heliksar har ei hydrofil side vendt utover, og ei hydrofob side som vender innover – inn i heliksen. Rotasjonen til  $\alpha$ -heliksen er alltid hørehendt. Andre typar heliksar er  $\pi$ -heliks og  $3_{10}$ -heliks der bindingsavstandane er høvesvis 5 og 3.



**Figur 1.2:** a) Skjematisk diagram over hydrogenbindinga som formar ein  $\alpha$ -heliks. b) For at hydrogenbindinga skal kunne skje, må sekvensen formast som ein heliks i rommet. Henta frå [1].

### **1.2.1.2 $\beta$ -tråd/flak-strukturar**

$\beta$ -flak dannes av hydrogenbindingar mellom aminosyrer som kan ligge langt frå kvarandre i sekvensen.  $\beta$ -flak består av fleire  $\beta$ -trådar, enten ved at dei binder seg parallelt eller antiparallelt, eller ein kombinasjon av dei to. Når dei ligg parallelt får dei form som bølgeblikk-tak.

Det kan då vere fleire aminosyrer som forbinder kvar tråd, og desse kan bli klassifisert som å tilhøyre loop, turn eller coil strukturen.  $\alpha$ -heliksar kan òg ofte forbinde parallelle  $\beta$ -trådar. Eit antiparallelt flak kan dannes av residyar som ligg etter kvarandre i sekvensen. Desse er forbunde av det som blir kalla ein  $\beta$ -hairpin eller ein turn.

Av og til vil endetrådane i eit  $\beta$ -flak binde seg til kvarandre og gjere at flaket dannar ein struktur som blir kalla  $\beta$ -tønne.

### **1.2.1.3 Loop-strukturar**

Loop er dei delane av aminosyrekjeda som forbinder dei forskjellige sekundærstrukturelementa. Ca. ein tredjedel av aminosyrene i eit protein er loop-strukturar.

### **1.2.1.4 Supersekundære strukturar**

Det viser seg at SSE ofte binder seg saman på bestemte måtar. Slik som  $\alpha$ -heliksen som ofte ligger mellom to parallelle  $\beta$ -trådar. Akkurat den blir kalla ein  $\beta$ - $\alpha$ - $\beta$ -eining. Desse spesielle samanbindingane har fått samlenemninga supersekundære strukturar.

## **1.2.2 Bestemming av proteinstruktur**

Det er mulig å finne strukturen ved hjelp av Røntgenstråle-krystallografi (1.2.2.1) eller NMR-spektroskopi (1.2.2.2), men begge desse metodane er kostbare. Ofte blir desse to metodane brukt til å finne posisjonane til aminosyrene i proteinet, og så blir sekundærstrukturelementa bestemt ved hjelp av til dømes DSSP og andre metodar slik som beskrive i kapittel 1.4.

### **1.2.2.1 Røntgenstråle-krystallografi (X-ray Crystallography)**

Strukturbestemminga her begynner med at proteinet blir isolert, reinsa og krystallisert. Så blir krystallet plassert i ei røntgenstråle og ein kan då observere ein viss diffraksjon. Dette kommer av plasseringa av molekyla i krystallet. Det ein måler då er intensiteten til dei forskjellige diffraksjonerte strålane. Ein treng i tillegg informasjon om fasane til strålane før ein kan bestemme strukturen. Det finnes fleire måtar å finne fasane på, og den mest brukte er å samanlikne proteinet med eit kjent protein i same familie. Når ein då har diffraksjonen og fasane, kan ein finne strukturen ved bruk av eit dataprogram som utifrå dataa modellerer proteinet.

### **1.2.2.2 NMR (Nuclear Magnetic Resonance)**

NMR måler energinivået til dei magnetiske nukleona i atom. Desse energinivåa er veldig sensitive for miljøet rundt atomet. Atom som er forbundet til kvarandre påverkar frekvensen til signalet som blir sendt frå eit atom slik at NMR kan bruke det til å bestemme konformasjonsvinklane, og som igjen kan bestemme SSE. Frå interaksjonar mellom atom som ikkje er bunde til kvarandre kan NMR bestemme posisjonane til atom som ligger nær kvarandre i struktur, men ikkje nødvendigvis i sekvens.

Matematisk kunne ein ha kalkulert heile strukturen, om ein hadde eit komplett og eksakt sett av interatomiske avstandar. NMR gir eit tilnærma og ufullstendig sett av avstandar, slik at for å berekne dei nøyaktige koordinatane tar dei i tillegg inn kjemiske data.

### **1.2.2.3 Prediksjon frå sekvens**

Predikering av proteinets struktur frå sekvensen blir sett på som eit av dei største problema i bio-informatikk. Der prøver dei å føreseie korleis proteinet foldar seg bare ut frå sekvensen av residyar.

Eit av dei beste programma for predikering av SSE frå sekvens er PHD (Profile network from HeiDelberg). Dette programmet søker opp eit sett av sekvensar homologe til den som skal predikerast for å finne strukturen før den begynner predikeringa. Deretter lager den ein multippel samanstilling som den matar inn til dei kunstige nevrane nettverka på første nivå. Denne predikerer for kvar enkelt residy. Neste nivå tar omsyn til at residyar som ligger etter kvarandre ofte er i same sekundærstrukturelement. Tredje og siste nivå finner eit gjennomsnitt av resultatata frå fleire uavhengig treningsnettverk.

## **1.3 Kvifor bestemme strukturen**

Proteinets sin struktur blir bestemt av primærstrukturen (sekvensen), og miljøet det dannes i.

Grunnen til at ein ønskjer å finne strukturen til eit protein er at det er ein nær samheng mellom strukturen og det enkelte protein sin funksjon. Eit eksempel på dette er eit protein som dannar ei  $\beta$ -tønne og fungerer som ei pore i membranen til ei celle.

Proteinets sin struktur blir veldig godt bevart gjennom evolusjon, så ein annan grunn til at klassifisering er veldig interessant er for å kunne trekke evolusjonslinjer til andre kjente protein.

Målemetodar for å bestemme eit protein sin struktur er kostbare. Ein måte å prøve å gjere det billegare på er å lokalisere dei forskjellige aminosyrene sin posisjon ved hjelp av målemetodar. Dette er noko rimelegare enn dei for å bestemme heile strukturen. Deretter bestemmer ein kor strukturen er anten manuelt eller ved hjelp av dataprogram. Problemet med den bestemminga er at dei forskjellige forskarane er ueinige om kriterier for å bestemme SSE, og bestemminga blir derfor litt forskjellig. Dataprogramma blir laga for å prøve å lage ein felles måte å bestemme SSE på. Fordelen med dette er at alle protein får den same bestemminga. Ulempa kan bli at forskarane bruker dei programma heilt ukritisk. Så lenge ein ikkje har ein metode/program for bestemming som er 100% korrekt, kan ukritisk bruk vere uheldig. Ganske enkelt fordi det lett kan oppstå feil.

## **1.4 Bestemming av SSE frå struktur**

Identifisering av SSE frå struktur er ikkje eit enkelt problem. Dette fordi ingen ennå har klart å finne ein presis definisjon av kva aminosyrer som er med i eit SSE. Ein slit spesielt med å få ein god identifisering i endepunkta til sekundærstrukturelementa.

Dei vanligaste verktøya for bestemming er vinkelskjema, avstandsmatriser og hydrogenbindingar.

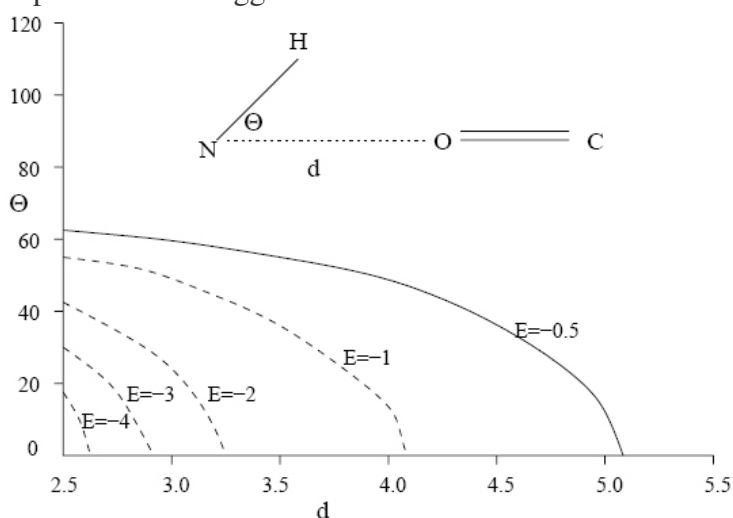
### 1.4.1 DSSP (Define Secondary Structure of Proteins)

Det vanligaste programmet för att identifiera SSE från strukturer är DSSP av Kabsch och Sander[6]. Detta programmet är baserat på mönster för vätebindningar.

DSSP använder en elektrostatisk värde för vätebindningar som den finner ut från avståndet mellan atomerna, och vinkeln mellan vätebindningen och bindningen från det ena atomet till det andra.

Till exempel vid en vätebinding mellan N i en N-H-grupp och O i en O=C-grupp, vilken vätebindning och då också avståndet mellan N och O, och vinkeln mellan vätebindningen och N-H-bindningen, som visas tydligt på Figur 1.3. En god (ideell) vätebinding har en energi på ca  $-3$  kcal/mol. DSSP tolkar alla bindningsenergiar mindre än  $-0,5$  kcal/mol som en vätebinding och tillägger därför ett tillräckligt stort avvik från den ideella bindningen.

Ut från den informationen DSSP finner om vätebindningarna kan den finna ut vilken SSE ligger, och vilken del av proteinet som ligger ut mot vätska.



**Figur 1.3:** Vätebinding mellan peptider beskrivs här av den dominerande elektrostatiska delen E av vätebindningsenergin. Här tecknas i stiplade linjer för olika konstanta funktioner av E som en funktion av avståndet,  $d$ , och sammanställningsvinkeln  $\theta$ . En ideell vätebinding har  $d = 2,9\text{\AA}$ ,  $\theta = 0$  och  $E = -3,0$  kcal/mol. Men det finns en vätebinding för  $E$  upp till  $-0,5$  kcal/mol (hel linje), så att man kan ha en sammanställningsvinkel på upp till  $63^\circ$ , vid ideell längd upp till  $d = 5,2\text{\AA}$  mellan N och O i en perfekt sammanställning. Å (Ångström) är definierat som  $10^{-10}$  m. Henta från [6].

### 1.5 PDB – The Protein DataBase

<http://www.rcsb.org/index.html>

Källa: [5].

PDB blev etablerat vid Brookhaven National Laboratory i 1971 som ett arkiv för biologisk makromolekylstruktur. I början bestod arkivet av 7 strukturer, och några få nya kom in kvart för kvart. I 1980-talet började nya strukturer komma in kvart för kvart i en snabbare takt. 13. november 2001 var antalet strukturer i PDB blivit 16546. Någon av orsakerna till den stora ökningen var att det har blivit gjort stora förbättringar i kristallografi-proceduren, i tillägg började de också ta in strukturer bestämt med NMR.

I början var användningen av PDB begränsad till en liten grupp forskare innan biologisk strukturforskning. I dag är den största delen av användarna forskare, föreläsare och studenter.

Sidan oktober i 1998 har PDB blitt administrert av 3 medlemmer av Research Collaboratory for Structural Bioinformatics (RCSB) – Rutgers, The State University of New Jersey, San Diego Supercomputer Center ved universitetet i California, San Diego og National Institute of Standards and Technology.

### **1.5.1 PDB-filer**

PDB lagrar alle proteina i databasen i egne filer som beskriver korleis proteinet er bygd opp. Fila inneheld detaljert informasjon om kva protein som er beskrevet i den enkelte fila, fullt namn, kven som har levert inn resultat om proteinet til PDB, når, osv. Hovudinformatjonen i fila er informasjon om sekvensen, kva SSE proteinet er bygd opp av, og ei opplisting av alle atoma med deira koordinatar og anna spesifikk informasjon som tilhører kvart enkelt atom. For å skilje dei forskjellige linjene frå kvarandre begynner kvar linje med eit merkenamn som fortel kva informasjon som finst på den linja. SEQRES viser at den linja fortel sekvensen til proteinet. HELIX og SHEET viser at linja har informasjon om SSE. ATOM viser at linja beskriver eit atom.

### **1.6 Kva oppgåva omhandlar**

Målet med denne oppgåva er å finne ein ny måte å kartlegge kor sekundærstrukturelementa er plassert, etter at posisjonane til dei forskjellige residyane er funnet ved hjelp av NMR eller X-stråle Krystallografi.

Eg har prøvd å bruke Kunstige Nevrale Nettverk (KNN, Kap.3) til å bestemme kor SSE ligger i proteinet ut frå ei avstandsmatrise (Kap.2). Ei avstandsmatrise er ein tabell med tal som viser avstandane mellom alle par av residyar i ein struktur. KNN er ein måte å bruke datamaskinar på som er god til å klassifisere data. Dei kan bli trena opp med mange eksempel, og vil då seinare kunne korrekt klassifisere reelle data av same type. Dette gjer at dei er gode på enkelte områder som er vanskelig å løyse ved hjelp av algoritmar. Det tar lang tid å trene opp nettverka, men etter at dei er ferdig trena bruker dei kort tid på å utføre oppgåva. Dette gjer at det kan ta mykje prosesseringstid å lage eit program som inneheld KNN, men når programmet skal kjøre bruker det lite prosesseringstid. Alternativet som er å lage programmet algoritmisk ville òg tatt lang tid å utvikle om det skulle løyse dei same problema. I tillegg ville der prosesseringstida ved kvar kjøring av programmet vore veldig lang.

Eg har laga eit program som deler opp fleire avstandsmatriser, slik at dei kan brukast som eksempel til trening, og som eit testsett for å sjå korleis nettverket klassifiserte data det ikkje hadde trena på. For å analysere data eg fekk frå nettverka laga eg fleire små program som er beskrevet i kapittel 5. Alle desse programma er skrivne i standard ANSI C++, som er eit vanlig objektorientert programmeringsspråk eg blei godt kjent med når eg gjekk på ingeniørhøgskulen.

Som referanse for kor gode resultatane mine er, har eg brukt bestemningar som er lagra i PDB.

## Kapittel 2: Avstandmatriser

Dette kapitlet er i hovudsak basert på [1].

Ei avstandsmatrise er ein tabell med tal som viser avstandane mellom alle par av residyar i ein struktur, slik som Figur 2.1. Avstandsmatriser kan bli sett på som ein 2D-presentasjon av ein 3D-struktur. Grunnen til at det går an å representere ein 3D-struktur i 2D er at når ein rekonstruerer 3D-strukturen kan ein bruke avstandane til å definere aksane. Eksempelvis kan ein definere første residy som origo, der y-aksen ligger på avstanden frå origo til andre residy. Ut i frå dei to punkta me då har og avstandane frå dei to første residyane til den tredje, kalkulerer ein y-koordinaten til den tredje residya som ein konstant, og x-koordinaten som ein verdi  $\pm k$ , der  $k$  er ein konstant. Dette er fordi avstandsmatriser manglar informasjon om kva retning strukturen har. Dvs at den strukturen ein får frå rekonstruksjonen kan vere speila om y-aksen i forhold til den du prøver å rekonstruere. X-aksen skal ligge  $90^\circ$  i forhold til y-aksen. Z-aksen ligger  $90^\circ$  på det planet som er definert av dei tre første punkta. Punkta til dei resterande residyane kan finnast på same måte som den tredje, og då vil avstanden mellom det nye punktet og det tredje definere om dei er på same side av y-aksen.

Dei vanlegaste kjeldene for avstandsinformasjon er NMR-eksperimenter og Røntgenstråle-krystallografi (X-ray Crystallography).

### 2.1 Samanhengen av avstandsmatriser og koordinatar

Når ein lager avstandsmatriser så tar ein primært utgangspunkt i  $C_\alpha$ -atoma eller  $C_\beta$ -atoma og kalkulerer avstandane mellom desse i euklidiske avstandar etter formelen:  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$  [7]. Dei fleste aminosyrer har bare eit  $C_\alpha$ -atom eller eit  $C_\beta$ -atom. Avstandane ein då får blir oppgitt i Ångstrøm ( $10^{-10}\text{m}$ ).

### 2.2 Korleis kan avstandsmatriser brukast til bestemming av SSE?

Avstandsmatriser kan brukast både manuelt og automatisk for å finne SSE.

I idealiserte  $\alpha$ -heliksar vil avstandane frå første  $C_\alpha$ -atom til dei andre  $C_\alpha$ -atoma vere omtrentlig: 3,8, 5,4, 5,1, 6,3, 8,7, 9,9, 10,6, 12,5Å... Desse kan ein finne utifrå idealiserte vinkelpar for  $\alpha$ -heliksar og avstandane mellom atoma i proteinet. Ein finn sjeldan idealiserte SSE, men verdiane er greie å ha som eit utgangspunkt.

Heliksar vises som oftast i avstandsmatriser som områder langs diagonalen med små avstandar mellom atoma, slik som residy 31-38 i Figur 2.1.

I idealiserte  $\beta$ -trådar er dei tilsvarande avstandane 3,8, 6,6, 10,3, 13,5, 16,9Å... Av og til kan ein finne nabotrådar rundt diagonalen til avstandsmatrisa, men dette blir fort vanskelig etterkvart som avstandane aukar. Områder med små avstandar rundt lokale diagonalar eller antidiagonalar kan ofte indikere at der kan ein finne forbindingane mellom trådar i eit flak. Parallele flak ligg som områder rundt ein subdiagonal, og antiparallele ligg rundt antisubdiagonalar.

Figur 2.1 har trådar i residyane 18-22 ( $\beta_1$ ), 26-29 ( $\beta_2$ ) og 52-56 ( $\beta_3$ ), med antiparallele forbindingar mellom  $\beta_1$  og  $\beta_2$ , og mellom  $\beta_1$  og  $\beta_3$ .

```

123456789A123456789B123456789C123456789D123456789E123456789
1 0469.....
2 40469.....
3 640479.....
4 9640479.....
5 .9740469.....778.....9.....
6 ..9740468..8.869.9.....8779.....
7 ...9640469857669.....989.....
8 ....964045547769.....9857..88.....99.....
9 .....8640466..9.....7..98..9.....967.....
10 .....9540469.....89..76.97.....79.....
11 .....856404688.....98..89.....
12 .....8546640456.....99.....
13 .....77.9640468.....9.....
14 .....7867..8540469.....8.....
15 ....76669.866404679.....8658.9.....
16 ....8999....864047.....968.....9.....
17 .....964047.....98559.....8868.....
18 .....9.....774046.....9564579.....98676.....
19 .....9.740469..8855645788.....66578.....
20 .....64046985546888.99.....85578.....
21 .....6404765678..9.78..9.....8.....9.676579.....
22 .....964034579.....9966568.....
23 .....9730469.....897779.....
24 .....86440469.....9.98.7.89.....
25 .....8555640469.....988.....
26 .....8.9.....98567964046.....9.....9.89.....
27 .....9798.....8.955479.96404789.9.....999.....
28 .....7857899..6986568...9640477669.....
29 .....997.98998565468.....7404657.....9.....
30 .....885548.....87404689.....99698.....
31 .....97589.....9764046579.....798.....
32 .....8978..9..97.....6564046679.....
33 .....8869.....897.....9967864045569...88.....9.....
34 .....898.....9.9564045569.9.....9.97.....
35 .....9.....7654046679.....
36 .....97.....97554046667.88.....
37 .....9.....9656404544768...88.....
38 .....966640466.....
39 .....97654046.....
40 .....9646404788.....
41 .....9.746640468..8867.....
42 .....7..74047874457.....
43 .....9.....8..9.9.....8..86..86404554577.....
44 .....967.....8..88..887404669.....
45 .....979.....9.8.....8540469.....
46 .....8.9.....75640469.....
47 .....8446640479.....
48 .....9987.....84599640469.....
49 .....99.....8..657..974047.....
50 .....6678.....9..8..777...9640468.....
51 .....7679.....974047.....
52 .....8657.9.....9.....64047.....
53 .....965569.8.9..97.97.....874047.....
54 .....86578..8.9..99.....740469.....
55 .....86579.....9.968.....740469.....
56 .....8778.....9.....64047.....
57 .....9668.....8.....964047.....
58 .....8.....97404.....
59 .....740.....

```

**Figur 2.1** Dei første 59 residyane i PDB-oppføringa til 1chc der kvar residy er representert ved sitt C<sub>α</sub> atom. Avstandane er runda av til heiltal, og avstandar større enn 9 er representert som ein prikk for å auke lesbarheita. Henta frå [1].

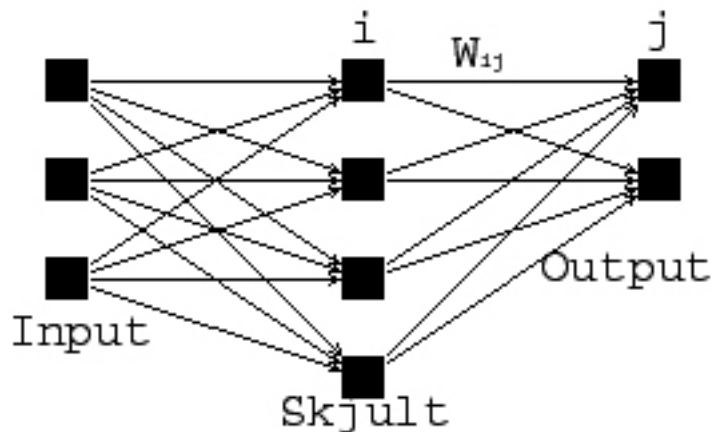
## Kapittel 3: Kunstige Nevrale Nett

Dette kapitlet er i hovudsak basert på følgjande kjelder: [8] og [9].

### 3.1 KNN, kva er det?

Kunstige Nevrale Nettverk (KNN) er ein veldig spesiell form for datastruktur. Framfor å bruke ei algoritme eller ein bestemt framgangsmåte så blir KNN trena opp til å utføre ei bestemt oppgåve. Dei blir trena etter modellar som er inspirert av korleis den menneskelege hjernen fungerer. Kwart enkelt nettverk er fullstendig spesialisert til den oppgåva det skal utføre, t.d. mønstergjenkjenning (som gjenkjenning av fingeravtrykk) eller dataklassifisering (som artsklassifisering av dyr).

Eit KNN består som oftast av fleire lag eller nivå med prosesseringseiningar kalla nodar med forbindingar mellom. Forbindingane blir kalla vektorer eller aktiveringsvektorar. Ofte er kvart enkelt lag fullstendig forbundet – det har alle-til-alle-forbindingar med dei laga som ligger framfor/bak det. Figur 3.1 viser eit enkelt lite KNN med 3 lag, eit inputlag, eit skjult og eit outputlag. Laga i Figur 3.1 har alle-til-alle-forbindingar.



**Figur 3.1:** Modifisert skjermbilde tatt frå SNNS (Kap 3.3) av eit enkelt nettverk med 3 lag. Inputlag med 3 nodar, skjult lag med 4 nodar, og outputlag med 2 nodar. Vektene er òg teikna inn på figuren som piler.

#### 3.1.1 Framoverfasen

Framoverfasen er når eit KNN skal utføre det som det er laga for å utføre – enten under trening eller i bruk. I framoverfasen får nodane i inputlaget ein verdi – ofte eit reelt tall. Deretter blir ein nettoinnverdi  $net_j$  rekna ut for kvar node, utanom dei i inputlaget, på bakgrunn av nodane i laget før som noden er forbunde med. Gjeldande node er her kalla  $j$ . Frå kvar node  $i$  brukar ein:

$o_i$ : utverdien frå node  $i$ .

$w_{ij}$ : vekt mellom node  $i$  og  $j$

Formelen for  $net_j$  er vanligvis  $\sum_i o_i w_{ij}$

Deretter blir ein aktiveringsverdi  $a_j$  rekna ut på bakgrunn av nettoinnverdi, den gamle aktiveringsverdien, og ein terskel  $\theta_j$ . Terskelverdien blir i SNNS (Kap 3.3) kalla bias, og blir i utgangspunktet satt til å vere 0, men blir av enkelte paradigme, deriblant Backpropagation, endra som ei vekt under trening. Ein vanlig funksjon er:  $a_j = 1/(1 + e^{-(net_j - \theta_j)})$  (gammel aktiveringsverdi er ikkje med her). Ein får då ein verdi i området  $[0,1]$ .

Til slutt blir ein utverdi  $o_j$  rekna ut frå aktiveringsverdien. Den er ofte lik  $a_j$ , men kan òg vera 0 eller 1.

Alt dette blir gjort for nodane i alle laga, utanom inputlaget, i stigande rekkefølge. Slik at når ein node blir oppdatert vil alltid alle nodane i laget før den som nå blir oppdatert allereie vere ferdig oppdatert. I inputlaget er denne utverdien lik den verdien som noden får tilordna.

Når alle nodane er oppdatert kan ein lese av resultatet i outputlaget.

## 3.2 Læring

Eit KNN lærer ved at det blir mata med ein del treningseksempel. Eit treningseksempel er bygd opp som eit mønster med data som skal tilordnast inputlaget og ein fasit. T.d. når nettverket skal klassifisere så blir nettverket mata med mange eksemplar på det som skal klassifiserast, og med ein fasit som seier kva den skal klassifisere eksempelet til. I mitt tilfelle var dei forskjellige mønstra ei liste med tall (avstandar), og ein fasit som sa om nettverket skulle klassifisere tala som ein  $\alpha$ -,  $\beta$ - eller  $\gamma$ -struktur. Slike mønstre blir ofte samla i ei eller fleire filer som blir kalla mønsterfiler.

Ein av dei mest vanlige klassane av KNN er "feed forward". Det vil seie at det har 2 eller fleire lag med nodar – helst minimum 3 (input, skjult og output), og at treningseksempla blir mata framover. Desse laga har alle-til-alle-forbindingar mellom nodane i inputlaget og det skjulte, og mellom det skjulte og outputlaget.

Det som skjer under trening av denne typen nettverk er at nettverket tar første inputmønster og kjører det gjennom framoverfasen. Deretter vil nettverket sjekke outputlaget mot eit fasitmønster og då finne ein feilverdi som blir sendt bakover og brukt av ein korreksjonsmekanisme til å korrigere vektene i heile nettverket. Målet med dette er at vektene skal bli endra slik at neste gong mønsteret blir presentert for inputlaget så vil outputlaget vise ein meir korrekt verdi. Fleire mønstre blir presentert for nettverket fleire gonger (syklusar), og etterkvart vil vektene i nettverket vere generaliserte slik at dei kan vise korrekt i outputlaget for alle inputmønstre. Dette blir kalla at nettverket generaliserer bra. Det blir utnytta vidare ved at ein presenterer nye ukjente mønstre, utan fasit, for nettverket, og målet er at ein då får ut korrekt verdi.

Om eit nettverk bestemmer veldig bra på treningssettet, og dårlig på ukjent sett, så seier ein at nettverket generaliserer dårlig, eller at det er overtrena. Det som skjer er at nettverket har blitt spesialisert til treningssettet.

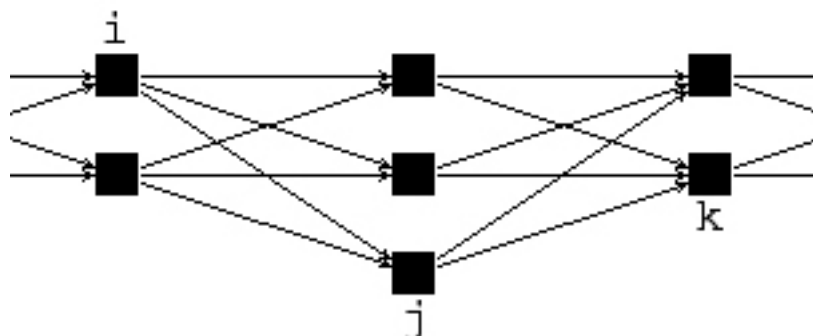
### 3.2.1 Backpropagation

Backpropagation er den vanlegaste læringsalgoritmen for kunstige nevrane nett. Den blei første gang presentert som eit paradigme i 1986 av Rumelhart og McClelland i boka "Parallell Distributing Processing"[10].

Backpropagation-nettverk har vanlegvis eit inputlag, eit eller fleire skjulte lag, og eit outputlag. Kvart enkelt lag er som oftast fullt forbundet med laget framfor. Backpropagation er eit feed forward-nettverk, og bruker den treningsmetoden som er beskrive over.

Korreksjonsmekanismen som blir brukt kan settes til å fungere på 2 forskjellige måtar:

- **Online trening** der vektene blir korrigerert etter kvart enkelt treningseksempel.
- **Batch trening** der alle eksempla blir presentert for nettverket, og feilverdien blir akkumulert slik at korreksjonsmekanismen rettar vektene ein gong for kvar gjennomgang av alle eksempla.



**Figur 3.2:** Figur for å vise samanhengen mellom indeksane  $i, j$  og  $k$  i formlane.

Vektene i nettverket blir oppdatert etter formelen:

$$\Delta w_{ij} = \eta \delta_j o_i$$

$$\delta_j = \begin{cases} f_j'(net_j)(t_j - o_j) & \text{om node } j \text{ er ein outputnode} \\ f_j'(net_j) \sum_k \delta_k w_{jk} & \text{om node } j \text{ er ein skjult node} \end{cases}$$

Der:

- $i$  er indeksen til noden før node  $j$  som har ein forbinding  $w_{ij}$  frå  $i$  til  $j$
- $j$  er indeksen til nåverande node
- $k$  er indeksen til noden som kommer etter node  $j$  som har forbindinga  $w_{jk}$  frå  $j$  til  $k$
- $\delta_j$  er feilen til node  $j$ . Dvs. forskjellen mellom outputverdien og fasit, eller summen av feil multiplisert med vektene som forbinder dei til node  $j$ , for alle nodane i laget etter.
- $t_j$  er fasiten til denne outputnoden
- $o_j$  er utverdien i denne outputnoden
- $o_i$  er utverdien i føregåande node  $i$

I programmet eg brukte til å trene KNN så har Standard (online) Backpropagation-lærefunksjonen 2 parameter:

- $\eta$ : Læreparameteren/Læringsraten. Denne indikerer kor "fort" nettverket skal lære. Merk: om denne er veldig høg så kan nettverket bli veldig ustabil og har derfor ein dårlig læring. Lave verdiar kan gi ein altfor treg læring.  $\eta$  må vanlegvis bestemast eksperimentelt. Ofte er det gunstig å la den begynne høgt, og så la den minke etterkvart. Vanlige verdiar ligger som oftast mellom 0,1 og 1,0 – men det kan òg vere at den fungerer bra på t.d. 2,0.
- $d_{max}$ : største differanse mellom outputverdien og fasitverdien som blir akseptert som korrekt. Typiske verdiar er 0, 0,1 eller 0,2. Den kan sjåast på som ei toleransegrense og er ikkje brukt direkte i formelen, men blir brukt av programmet.

### 3.2.2 Backprop Momentum

Backprop Momentum er ein spesialform av backpropagation nettverk der ein i tillegg bruker parameter for kor mykje av den førre vektforandringa ein skal dra med seg når ein kalkulerer neste vektforandring, og ein verdi (flat spot elimination) som blir brukt for å unngå at nettverket trenes til eit nivå (eit lokalt minima) som er dårlegare enn det optimale (det globale minima). Denne forma trener ein god del kjappare enn standard backpropagation.

Denne har to ekstra parameter i forhold til standard backpropagation:

- $\mu$ : moment, denne spesifiserer kor mykje av den førre vektforandringa som blir tatt med i kalkulering av den nye. Typiske verdiar ligger mellom 0 og 1,0.
- $c$ : Flat spot elimination value. Ein konstant verdi som blir brukt for å unngå at treninga av nettverket hamnar i lokale minima. Typiske verdiar for denne parameteren ligger mellom 0 og 0,25, som oftast blir 0,1 brukt.

Den nye vektendringa blir kalkulert etter nesten den same formelen som vanlig Backpropagation, men med følgjande endringar:

$$\Delta w_{ij}(t+1) = \eta \delta_j o_i + \mu \Delta w_{ij}(t)$$
$$\delta_j = \begin{cases} (f_j'(net_j) + c)(t_j - o_j) & \text{om node } j \text{ er ein outputnode} \\ (f_j'(net_j) + c) \sum_k \delta_k w_{jk} & \text{om node } j \text{ er ein skjult node} \end{cases}$$

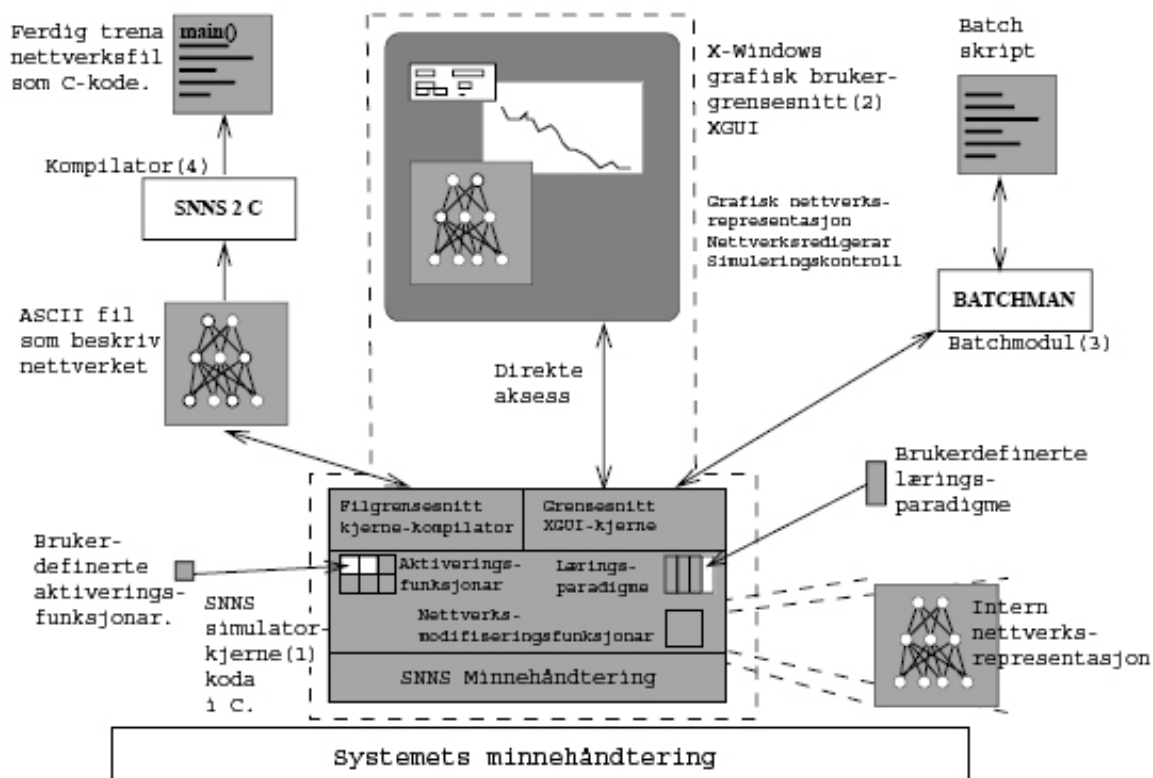
### 3.3 SNNS (Stuttgart Neural Net Simulator)

SNNS er ein simulator for nevrane nettverk som dei har jobba på ved Institute for Parallell and Distributed High Performance Systems, ved universitetet i Stuttgart sidan 1989. Målet med programmet er å lage eit effektivt og fleksibelt simuleringsmiljø for forskning på og bruk av nevrane nett.

Sjølve programmet består av fire delar: simulatorkjernen (1 i Figur 3.3), grafisk grensesnitt (2 i Figur 3.3), batchmodul (3 i Figur 3.3) og ein kompilator (4 i Figur 3.3) som oversetter ferdig trene nettverk til c-kode. Kjernen er den delen der all trening blir gjort. Det grafiske grensesnittet og batchmodulen jobbar mot kjernen, mens kompilatoren jobbar på dei ferdige nettverksfilene.

Det grafiske grensesnittet gjer det enkelt å:

- Lage nettverk: Grensesnittet har egne vindauge for å generere fleire forskjellige nettverkstypar. Eit kontrollvindauge der du kan justere dei fleste parametrar for nettverket. Slik som val av lærefunksjon, oppdateringsfunksjon og initieringsverdiar. I tillegg har SNNS eit eige vindauge for å gjere endringar på den enkelte noden i nettverket.
- Kjøre treningar: Via same kontrollvindauge som ein bruker for å justere parametrar for heile nettverket kan ein òg kjøre treningane direkte. Her kan ein initiere nettverket, velje tal på syklusar ein skal trene, velje kva treningsfil/valideringsfil ein skal bruke, kor ofte ein skal validere, osv. Merk at ein får her bare kjørt treningar med eit nettverk og eit sett med verdiar på parameterane i kvar kjøring.
- Studere nettverk: SNNS har fleire vindauge som er berekna på å studere nettverk. Det har eit vindauge for å sjå på heile nettverket i 2D, eit for 3D, og i tillegg egne vindauge for å vise forskjellige grafar som beskriver nettverket.



Figur 3.3: Figur som viser korleis dei forskjellige delane av SNNS jobbar saman. Figur tatt frå [9] og oversatt til norsk.

Batchmodulen jobbar med filer koda i deira eige batchspråk. Batchspråket følgjer same struktur som dei fleste programmeringsspråk. Batchspråket er meir beskrive i kap 5.2. Batchmodulen er god å bruke for å trene fleire nettverk med forskjellige parameter fortløpande – slik ein ofte må for å finne dei optimale parametrane. Altså at ein testar ut mange forskjellige verdiar for parametrane i ein og same kjøring.

Når ein har laga eller lasta eit nettverk i SNNS så treng ein deretter å laste ei eller fleire mønsterfiler. Mønsterfiler er filer som inneheld fleire sett med verdiar som skal tilordnas inputlaget, og eventuelt fasitverdiar om ein skal ha trening med fasit. Desse blir brukt som input til nettverka under trening og bruk. Ofte har ein eit treningssett som blir brukt under trening, og eit valideringssett som blir kjørt gjennom nettverket utan at vektene blir oppdatert, for å sjekke kor godt nettverket generaliserer.

Veldig mange nettverkspadigme er allereie fullstendig implementert i SNNS. Den er derfor eit godt verktøy for å teste ut forskjellige paradigme i rimelig tid. Ein treng lite tid til å sette seg inn dei forskjellige paradigme før ein begynner å bruke det.

Meir informasjon og sjølve SNNS finnes på Internett på følgjande adresse: <http://www.snns.org>



## Kapittel 4: Kva har blitt gjort?

### 4.1 Tankar rundt inn og utdata i oppgåva

Eg gjorde meg ein del tankar rundt kva og korleis inndata til nettet skulle vere. Dvs. korleis avstandsmatrisedata skulle vere formatert.

Inndata kom eg fram til at var eit greitt parsingsproblem. Det som verkeleg avgjorde korleis inndata skulle vere formatert var kva topologi eg valte å bruke i dei kunstige nevrane nettverka. For å generere inndata til nettverka laga eg eit program som tok inn PDB-filer og genererte mønsterfiler for SNNS automatisk. Dette programmet er beskrive i kapittel 5.1.

Utdata var definert til å vere ei liste over dei residiane eg har prøvd å bestemme med tilhørande opplisting av kva SSE programmet mitt bestemte dei til å vere. Planen var – om eg fekk ein god bestemming, å formatere resultata frå programmet mitt til ein prosentmessig fordeling over dei tre mulige SSE.

### 4.2 Kva KNN paradigme skal brukast?

Eit av dei første vala eg stod overfor var valet av KNN-paradigme for oppgåva. Eg enda opp med Backpropagation som er beskrive i kapittel 3.2.1. Backpropagation er det mest brukte paradigmet for KNN, og det er enkelt å forstå. I tillegg nyttar Backpropagation overvaka læring som eg såg det ville vere enkelt lage treningsfiler for med pdb-filene eg hadde fått tak i. Eg såg det som ein fordel å begynne denne oppgåva med eit paradigme som var enkelt å jobbe med, og så heller prøve dei vanskelige løysningane etterpå. Backpropagation såg òg ut til å passe betre til denne oppgåva enn dei to andre paradigma eg såg på.

Etterkvart som eg fant ut at Backpropagation trena veldig sakte i SNNS, leita eg etter alternativ. Den eg fant – og enda opp med – var Backprop Momentum. Dette er ein forbetra versjon av Backpropagation som eg fant beskrive i SNNS-manualen. Backprop Momentum er beskrive i kapittel 3.2.2.

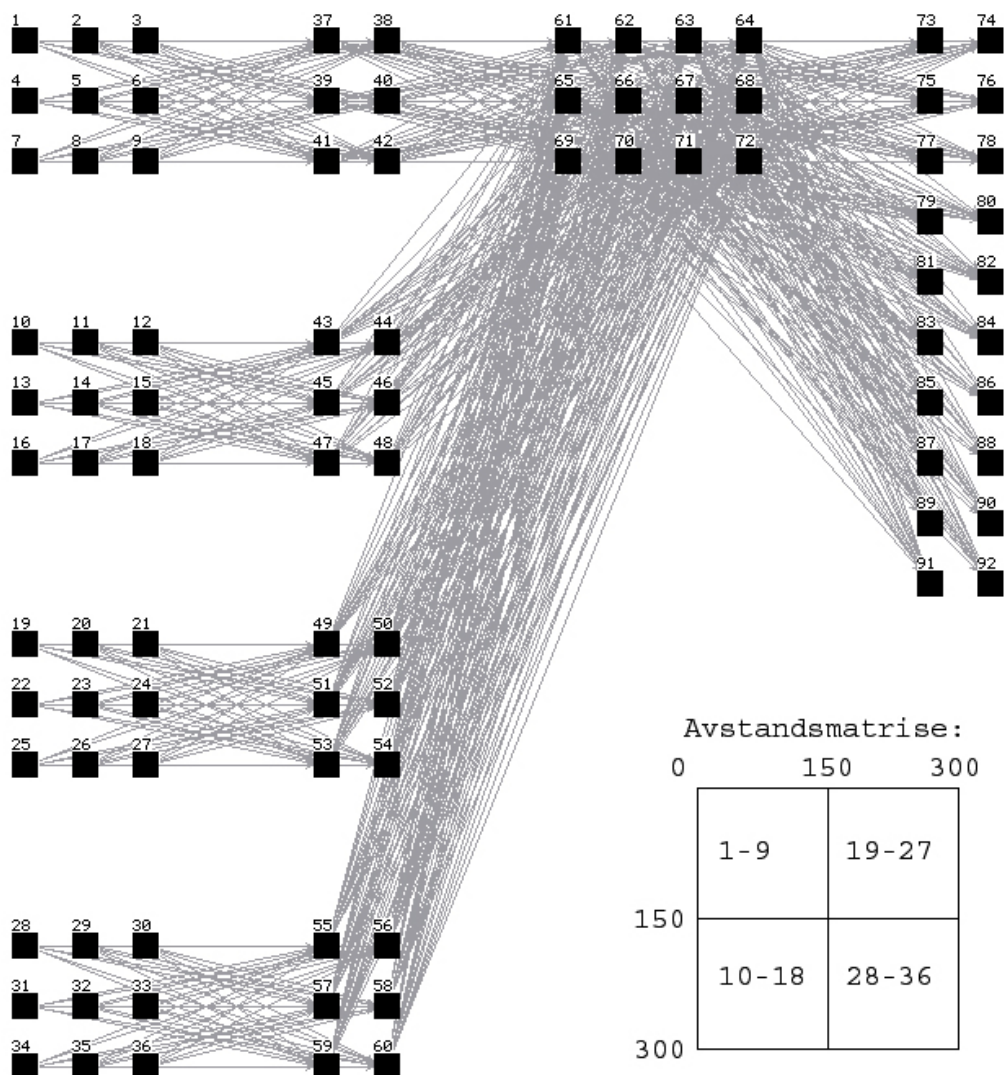
Andre paradigme eg meinte kunne vore aktuelle:

- Kohonen-nettverk: Dette er eit sjølvorganiserande nettverk, som består av eit inputlag og eit konkurrerande lag. Dette trenes med uovervaka læring, dvs læring utan fasit. Sidan eg hadde data der eg enkelt kunne trekke ut ein fasit, og eg føretrekk overvaka læring, valte eg å ikkje bruke denne.
- Counterpropagation: Dette er ein veldig spesiell form for KNN då den kombinerer eit Kohonen-nettverk med eit såkalla Grossberg-lag, og slik kombinerer fleire paradigme i eit nettverk. På meg virke dette paradigmet som noko meir avansert enn Backpropagation.

## 4.3 Forskjellige topologiar som har blitt testa ut

### 4.3.1 Å mate inn heile avstandsmatrisa direkte

I SNNS kan ein definere fleire "lag" av same type – inkludert inputlag. Vanlegvis har ein bare eit inputlag og eit outputlag, men i denne delen av oppgåva har eg laga til fleire inputlag. Ideen bak det å dele det i 4 var at det ville gjere det lettare seinare å parallelisere nettverket ved å flytte kvart "lag" til ein egen prosessor. For å unngå misforståingar vidare i oppgåva vil eg kalle kvar del av laget for ei nodeklynge. Merk at det første skjulte laget òg består av 4 nodeklynger.



**Figur 4.1:** Prinsippskisse for nettverket der eg mata inn heile avstandsmatrisa. Kvar node (1-36) i dei 4 nodeklyngene i inputlaget tilsvarer her 50 inputverdiar. Kvar node (37-72) i dei 2 skjulte laga tilsvarar 10 nodar. Outputlaget er her på 2x10, reelt var det 2x300. Nedst til høgre er det ei matrise som viser korleis verdiane i avstandsmatrisa blir tilordna dei forskjellige nodane i nettverket.

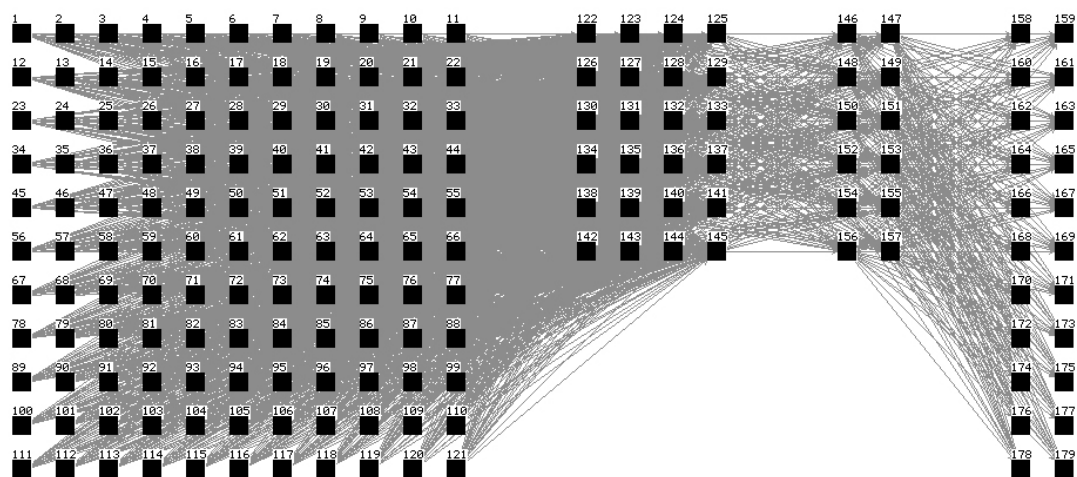
Dette har eg prøvd i forskjellige versjonar. Først med ei matrise på 300x300, som eg delte i 4 nodeklynger som vist i Figur 4.1. Teorien her var at alle residyane i proteinet skulle bestemmast samtidig. For å få dette til, fylte eg ut med nullar der det mangla avstandar når proteina var mindre enn 300 residyar. Deretter fordelte eg avstandane

på nodane i inputlaget som vist nedst til høgre i Figur 4.1. Eg prøvde dette både med matriser på 300x300 og 200x200.

Med 300x300 prøvde eg òg å lage opp nettverk med 2 ekstra skjulte lag. Der var nodeklynge 5 (node 37-42 på Figur 4.1) og nodeklynge 6 (node 43-48) forbundet med ei ny skjult nodeklynge, som igjen var forbundet med lag 9 (node 61-72). Nodeklynge 7 (49-54) og 8 (55-60) var tilsvarande forbundet. Sjå kapittel 6.1 for detaljar om korleis dei forskjellige nettverka var bygd opp.

Eg genererte dei nettverka eg hadde tenkt å teste ut før eg begynte å kjøre treningar. Så vidt eg kunne sjå ut i frå treningane blei alle desse nettverka altfor store for mine maskiner. For å i det heile tatt å klare å kjøre dei på mi maskin, måtte dei skjulte laga gjerast mykje mindre enn inputlaget. Dette medførte at nettverket generaliserte dårlig. Outputlaget her var ei liste binære verdiar med 2 tall, slik at det kunne klassifisere residiane som: 0 0 Gamma; 0 1 Alpha Heliks; 1 0 Beta flak; 1 1 kunne eventuelt bli brukt til å klassifisere Turn. For å kunne sjekke resultatata frå nettverka mot fasiten laga eg til eit analyseprogram som tok inn mønsterfilene frå både trenings- og valideringssetta, og sjekka dei mot alle resultatfilene nettverka hadde generert.

### 4.3.2 Å mate inn ¼ av matrisa



**Figur 4.2:** Prinsippskisse for å mate inn ¼ av matrisa. Inputlaget her (node 1-121) var 107x107, lag 2 (node 122-145) var 27x24, lag 3 (node 146-157) var 27x16, og outputlaget var 2x107.

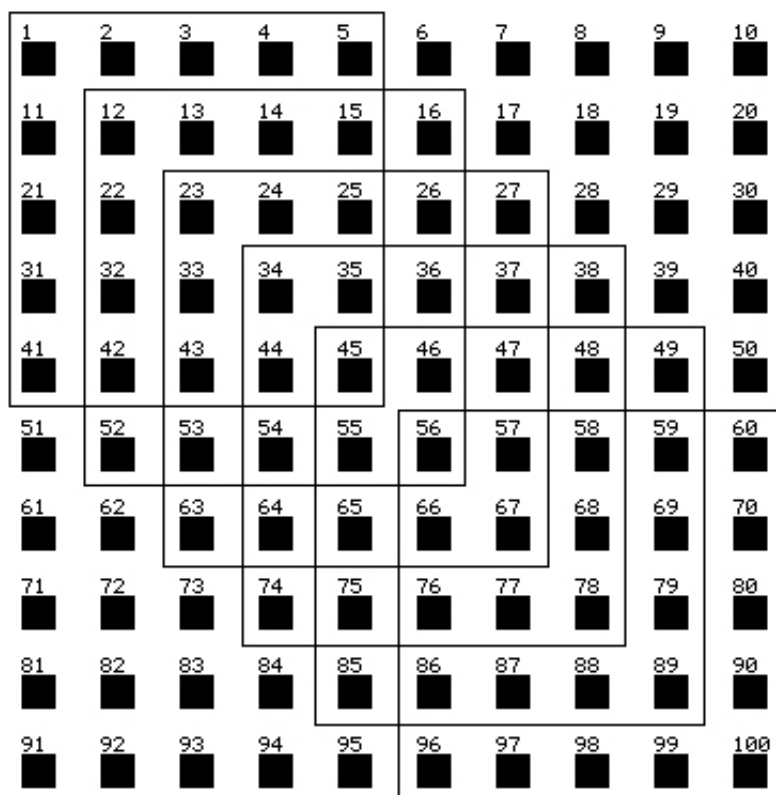
Eg brukte her ei total matrise på 200x200 som utgangspunkt, og hadde eit inputlag på 107x107 slik at eg fekk litt overlapping mellom dei forskjellige nettverka eg hadde tenkt å trene opp. Tanken bak dette var at eg skulle dele opp nettverket frå 4.3.1 og trene kvar enkelt nodeklynge, og så la eg inn ein overlap på 7 verdiar i tillegg. Eg testa ut 3 nettverk med 4 lag, og 4 nettverk der eg kutta vekk lag 3 (node 146-157 i Figur 4.2). Outputlaget her var tilsvarande outputlaget eg hadde når eg mata inn heile matrisa (kapittel 4.3.1), då bare tilpassa storleiken til inputlaget. Sjå kapittel 6.2 for detaljar om korleis dei forskjellige nettverka var bygd opp.

Dette blei framleis for stort slik at generaliseringa blei for dårlig.

### 4.3.3 Mange små matriser i kaskade langs diagonalaksen

Her bytta eg tilnæringsmåte for korleis eg skulle bygge opp nettverka. I staden for å bestemme fleire residyar samtidig prøvde eg her å lage eit (etterkvart fleire) nettverk som skulle trenes opp til å bestemme ein residy om gangen. Dette gjorde at nettverka blei mye mindre, men samtidig stilte det større krav til nettverket si generalisering. Sidan eg bestemte bare ein residy om gangen så fjerna denne løysninga avgrensinga i storleiken til proteinet som dei to første versjonane hadde.

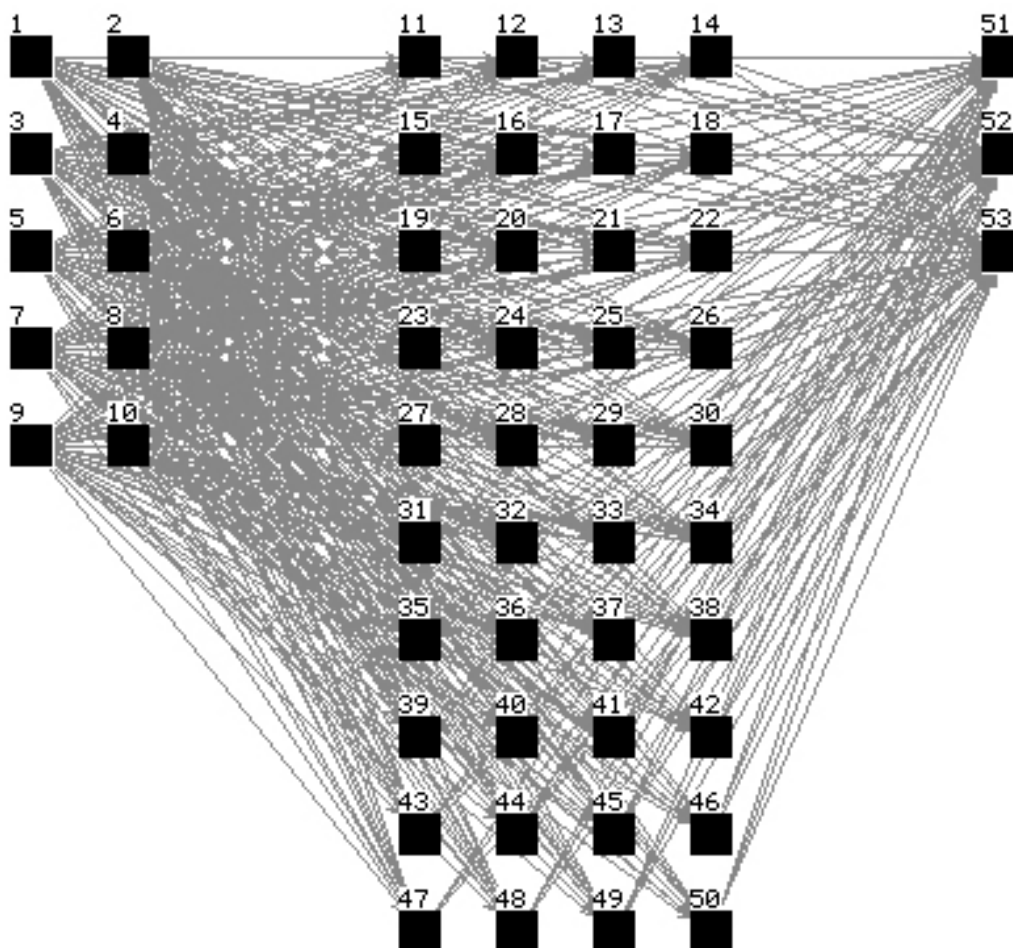
Utgangspunktet er her eit protein av lengde  $n$  og ei avstandsmatrise  $D$ . La  $D_i^r$  vera ei submatrise rundt diagonalen i  $D$ , med "sentrum" i residy  $i$ , og  $r$  residyar til kvar side. Det er då  $r(2r + 1)$  aktuelle avstandar i kvar  $D_i^r$  (p.g.a. symmetri).



**Figur 4.3:** Prinsippskisse for korleis avstandsmatrisa  $D$  blir delt opp med  $r=2$ . Døme: i første firkant vil residy 3 ( $i=3$ ) bli bestemt, då vil avstandane nummerert 2-5, 13-15, 24, 25 og 35 bli brukt. 1, 12, 23, 34 og 45 er lik 0, så dei trenger ein ikkje. 11, 21, 22, 31-33 og 41-44 er speilinga av dei på andre sida av diagonalen, så dei treng ein heller ikkje.

### 4.3.3.1 Versjon 1: Isolerte subnettverk

Det første eg måtte gjere her var å trene opp nettverk for å bestemme kvar enkelt residu basert på data frå avstandsmatriser. Desse nettverka har eg kalla Type I. Utdata her er ein verdi for kvar av dei tre strukturtypane. Input til desse nettverka var fleire submatriser  $D_i^r$  for forskjellige residuar  $i$ .



**Figur 4.4:** Skisse over eit nettverk med  $r=2$  i full storleik. Dette nettverket skal bestemme ein enkelt residu. Det skjulte laget (node 11-50) har dimensjonar som er det doble ( $2 \times 5 \rightarrow 4 \times 10$ ) av inputlaget (node 1-10). Outputlaget er på 3 nodar, ein for kvar SSE type ( $\alpha, \beta, \gamma$ ). Verdiane som det er oppført under Figur 4.3 at blir brukt blir tilordna nodane i inputlaget i stigande rekkefølge.

Alle nettverka i denne delen blei laga slik at det skjulte laget hadde doble dimensjonar av inputlaget. For nokre få av nettverka prøvde eg i tillegg å kvadrere dimensjonane til det skjulte ( $3 \times 7 \rightarrow 9 \times 49$ ), men sidan eg ikkje fant noko særlig betre bestemming på desse så brukte eg bare dei med doble dimensjonar vidare. Hovudgrunnane til at eg gjorde det skjulte laget forholdsmessig mykje større i denne topologien var at nettverka her i utgangspunktet blei veldig små. Frå dei første topologiane eg prøvde fant eg at generaliseringa blei veldig dårlig når dei skjulte laga var små i forhold til inputlaget. Sjå kapittel 6.3.1 for detaljar om korleis dei forskjellige nettverka var bygd opp.

For å få laga mønsterfiler til denne topologien måtte eg gjere ein del endringar i det programmet som laga mønsterfilene. Desse endringane er beskrive i kapittel 5.1.

Først trena eg her opp nettverk med verdiar for  $r$  som gjekk frå 3 til 8, men etter at det viste seg at  $r=3$  var den som gav best bestemming kjørte eg i tillegg treningar på  $r=1$  og  $r=2$ . For å sjekke kva resultat eg kunne jobbe vidare med skreiv eg om analyseprogrammet eg laga tidlegare. Dette programmet er beskrive i kapittel 5.3.

$r=1$  og  $r=8$  viste seg tidlig som lite gunstige å jobbe vidare med, så det var veldig få resultat frå dei som blei tatt med vidare til versjon 2, og desse igjen blei tidlig forkasta.

Etter at treningane var ferdige kunne då SSE av residy  $i$  bestemmas for  $r < i \leq n-r$  (nettverk  $I_i^r$ ) sekvensielt. Det måtte eventuelt ha blitt laga til egne nettverk for å bestemme residiane  $i \leq r$  og  $i > n-r$ , om det viste seg at denne forma for bestemming av SSE var noko ein kunne bruke vidare.

Den beste bestemminga eg hadde funnet etter versjon 1 var på eit nettverk med  $r=4$ ,  $\eta=0,1$ ,  $\mu=0,5$ ,  $c=0,25$ ,  $d_{max}=0,1$ , trena i 600 cyklar. Dette nettverket bestemte då 1098 residyar korrekt, 360 feil av totalt 1458, som tilsvarer 75,3 % korrekt. Fordelt på SSE ser det slik ut:

	# som skulle finnes	# korrekt funnet	#totalt funnet
$\alpha$	473	424 89,6%	536
$\beta$	329	198 60,2%	290
$\gamma$	656	476 72,6%	632

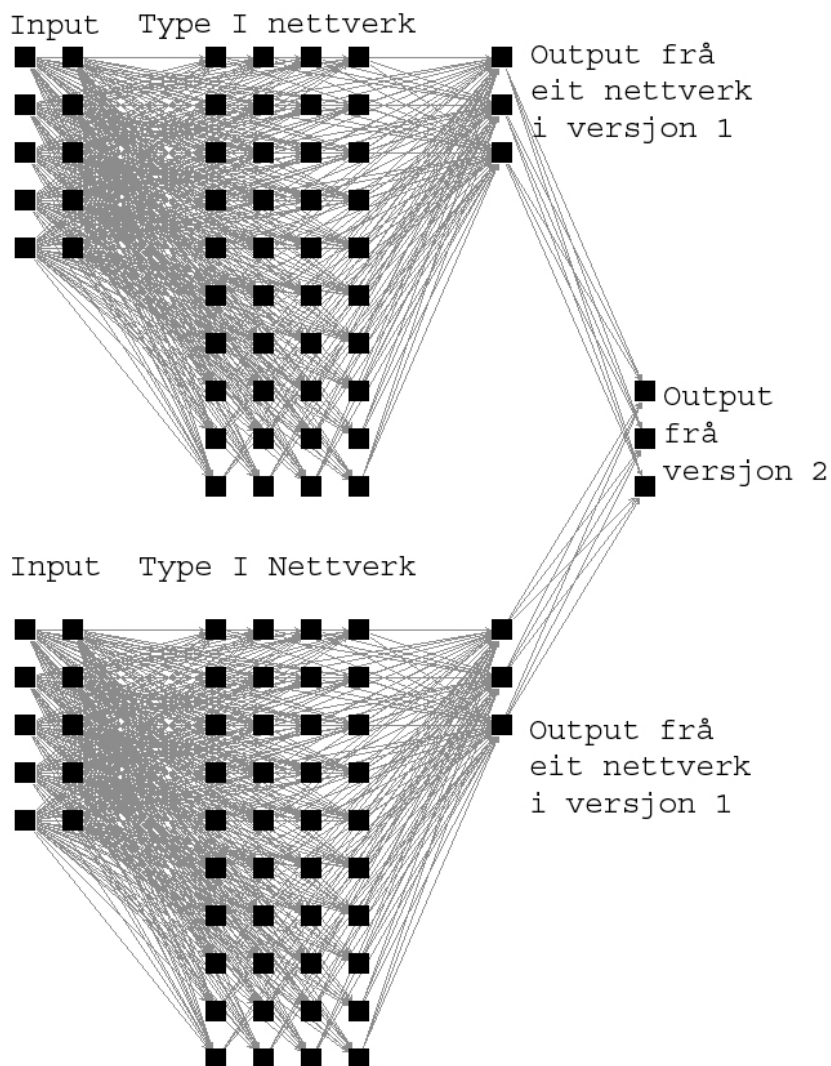
**Tabell 4.1:** Fordelinga over kor godt nettverket bestemte dei forskjellige SSE i versjon 1. Andre kolonne viser her kor mange residyar av kvar SSE-type som fans i proteina. Tredje kolonne viser kor mange som blei korrekt funnet, og siste kolonne viser kor mange som blei bestemt av programmet til å vere av den enkelte SSE-type.

Sjå kapittel 6.3.1.5 for fleire resultat.

### 4.3.3.2 Versjon 2: Slå saman for fleire $r$

SSE av residy  $i$  kan bestemast med bruk av fleire  $r$ . Her fant eg ein verdi for kvar av dei tre strukturtypane med bruk av verdiane funne for kvar  $I_i'$ .

Tanken bak denne versjonen er at resultatata frå fleire nettverk skulle slåast saman. Når eg jobba med resultatata frå versjon 1 fant eg i tillegg fleire forskjellige mellomresultat (resultat frå nettverk som bestemte godt ved forskjellig mengd treningscyklar, og med forskjellige verdier for treningsparameter) på dei enkelte  $r$  som det var interessant å jobbe vidare med. Det vil sei at eg gjekk vidare med både nettverk for forskjellige  $r$  og med fleire nettverk for kvar av den enkelte  $r$ . Dette medførte at eg gjekk vidare med forholdsvis mange mellomresultat.



Figur 4.5: Figur som viser korleis resultatata frå 2 Type I-nettverk med  $r=2$  blir slått saman.

Eg kombinerte her alle resultatata mot alle dei andre resultatata, og det igjen gjorde at eg fekk veldig mange kjøringar av versjon 2. Frå kvar kjøring tok eg vidare dei resultatata som hadde høgast tal korrekt bestemt, og samtidig hadde eit tal som var betre enn begge dei to filene som blei sendt inn. Programmet SortSammendrag blei skriva for å gjere ein grovsortering av resultatata, og resten av sorteringa gjorde eg manuelt. Eg enda opp med totalt 26796 kjøringar.

Måten eg faktisk gjorde dette på var å skriva eit program som leste inn fleire analysefiler frå versjon 1, og deretter tok snittet av resultatata for dei enkelte residyane.

Eg laga dette programmet slik at det henta inn analysefilene frå disk etter ei tekstfil som spesifiserte kva analysefiler som skulle slåast saman og kva resultatfila skulle heite. Dette gjorde at eg kunne lage ei lang tekstfil som først slo saman ei og ei analysefil, og deretter slo saman resultatfilene den allereie hadde generert. Slik fekk eg først slått saman ei og ei fil, deretter to og to, og fortsette slik inntil eg slo saman 8 og 8 analysefiler.

For å gjere desse kjøringane laga eg 4 småprogram. Ein meir inngåande forklaring av programma står i kapittel 5:

- Analyser: Same analyseprogrammet som blei brukt i versjon 1, men modifisert slik at det tar inn bare ei mønsterfil.
- Del2: Det programmet som faktisk slår saman analyseresultat og genererer i tillegg mønsterfilene for versjon 3. Programmet lager òg ei fil med alle dei residyanane som blir bestemt til å vere feil SSE. Denne fila var tenkt brukt for å sjå kor mykje feil bestemmingane til dei residyanane som var bestemt feil faktisk var, utan å måtte leite gjennom alle analysefilene.
- Input: Program for å generere inputfiler til Del2. Dette programmet blei laga for å enklare å lage inputfiler der alle analysefilene blei slått saman med alle dei andre analysefilene.
- SortSammendrag: Program for gå gjennom samandragfila frå Del2, og plukke ut alle resultat over ein viss terskelverdi. Dvs dei resultata som var verdt å jobbe vidare med.

Den beste bestemminga eg hadde etter versjon 2 var på eit resultat slått saman av resultata frå 6 forskjellige nettverk:

1. 5doble-dim-4:  $r=5$ ,  $\eta=0,1$ ;  $\mu=0,5$ ;  $c=0,1$ ; trenar i 2000 syklusar
2. 4doble-dim-3:  $r=4$ ,  $\eta=0,1$ ;  $\mu=0,5$ ;  $c=0,25$ ; trenar i 1300 syklusar
3. 4doble-dim-1:  $r=4$ ,  $\eta=0,1$ ;  $\mu=0,5$ ;  $c=0,25$ ; trenar i 600 syklusar
4. 2doble-dim-5:  $r=2$ ,  $\eta=0,5$ ;  $\mu=0,25$ ;  $c=0,25$ ; trenar i 300 syklusar
5. 2doble-dim-3:  $r=2$ ,  $\eta=0,1$ ;  $\mu=0,75$ ;  $c=0,15$ ; trenar i 400 syklusar
6. 2doble-dim-1:  $r=2$ ,  $\eta=0,1$ ;  $\mu=0,5$ ;  $c=0,2$ ; trenar i 300 syklusar

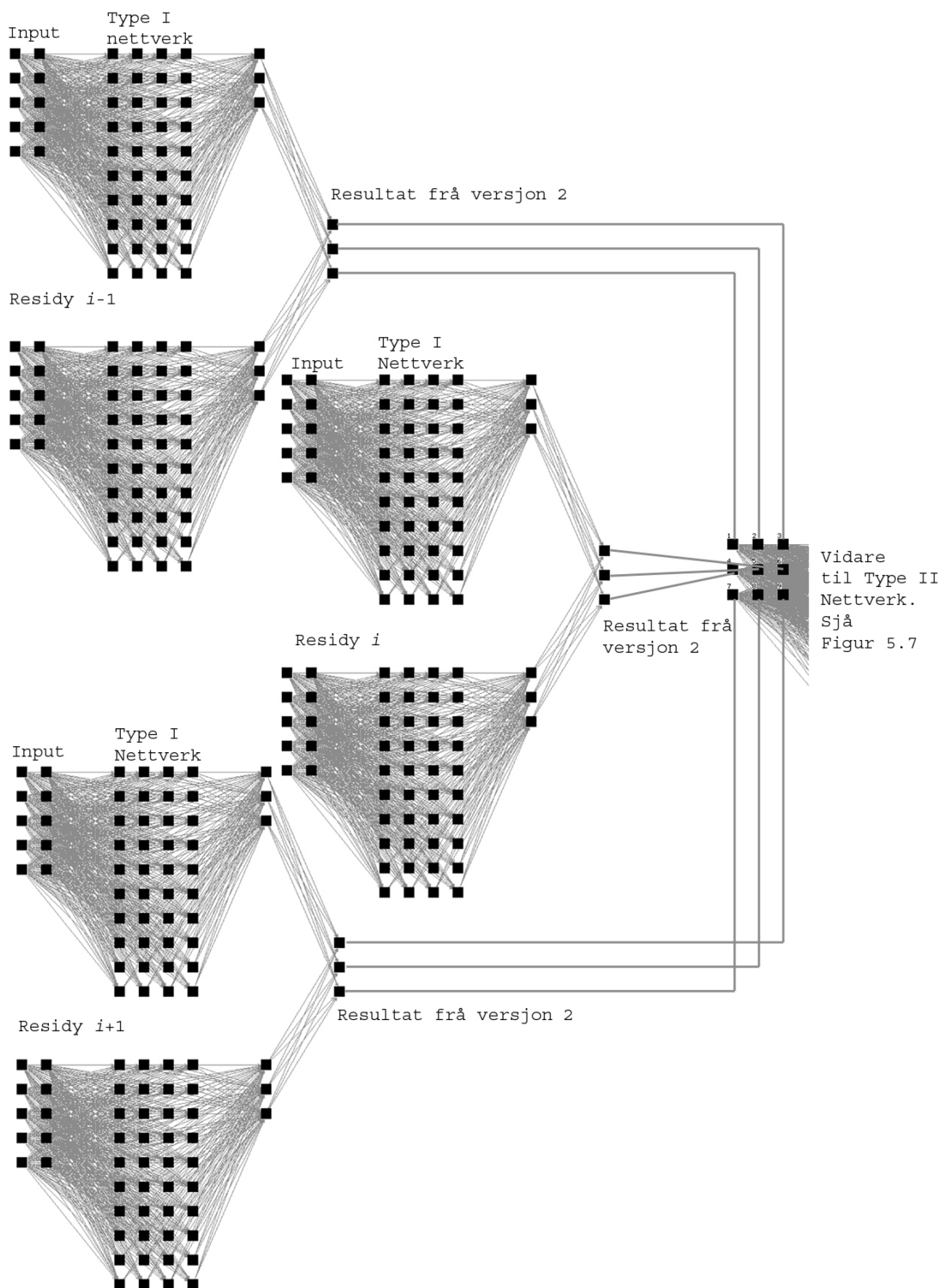
Dette nettverket bestemte då 1119 residyar korrekt, 319 feil av totalt 1438, som tilsvarer 77,8 % korrekt. Grunnen til at det her er færre residyar totalt enn i versjon 1 er at dei forskjellige  $r$  klipte vekk ulik mengde residyar i byrjinga av kvart protein. I versjon 1 blei det brukt  $r=4$ , her er høgaste  $r=5$  som klipper vekk ei residy meir i kvart protein. Fordelt på SSE ser det slik ut:

	# som skulle finnes	# korrekt funnet	#totalt funnet
$\alpha$	469	389 82,9%	436
$\beta$	321	183 57,0%	244
$\gamma$	648	547 84,4%	758

**Tabell 4.2:** Fordelinga over kor gode resultata var for dei forskjellige SSE i versjon 2. Andre kolonne viser her kor mange residyar av kvar SSE-type som fans i proteina. Tredje kolonne viser kor mange som blei korrekt funnet, og siste kolonne viser kor mange som blei bestemt av programmet til å vere av den enkelte SSE-type.

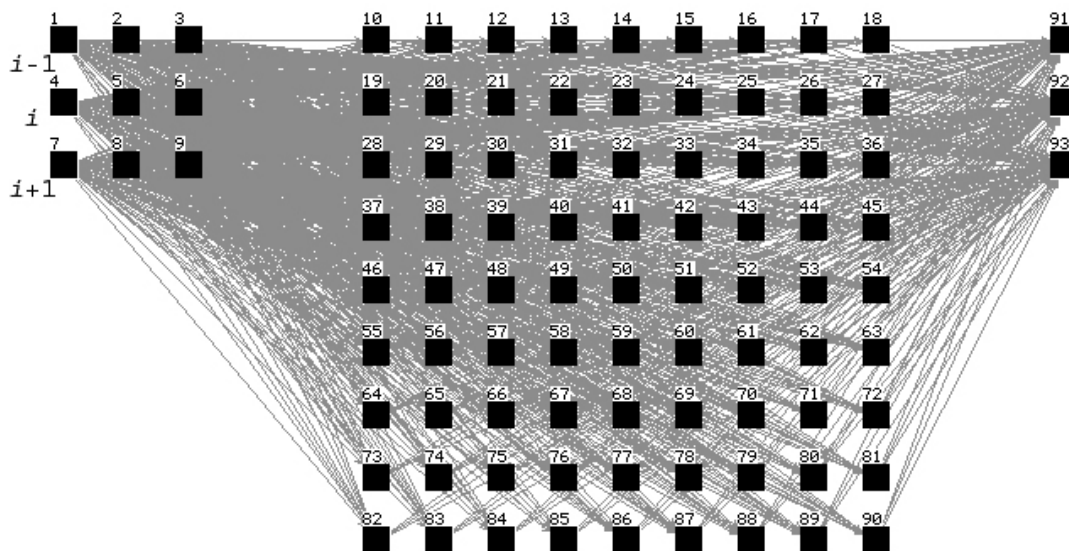
Det var her stor forskjell på resultata alt etter kva nettverk resultata, som blei slått saman, var i frå. På fleire var det bare 2 resultat som blei slått saman, og dei enda opp med eit nesten like bra resultat som det som er presentert her med resultat frå 6 forskjellige nettverk. Sjå kapittel 6.3.2 for desse resultata.

### 4.3.3.3 Versjon 3: Bruk av "nabonettverk"



Figur 4.6: Figur som viser korleis dei forskjellige versjonane av oppgåva er knytt saman.

Målet her er å bestemme kva SSE den enkelte residy tilhører ut i frå kva SSE den er funnet til å vere i versjon 2 og kva SSE som er funnet til å ligge på begge sider av den. Input til desse nettverka laga eg til i versjon 2, samtidig som Del2-programmet der slo saman resultatata frå versjon 1.



**Figur 4.7:** Skisse av nettverket som blei brukt til bestemming av ein residy, der eg tok som input bestemminga frå versjon 2 for den residya som skulle bestemmast med 1 nabo på kvar side. Dimensjonane i det skjulte laget er her kvadrert i forhold til inputlaget. Dette nettverket blei kalla Type II-nettverk.

Eg laga her 3 nettverk, eit med 1 nabo, eit med 2 og det siste med 3 naboar. Her prøvde eg 2 forskjellige måtar å trene nettverka på. Først trena eg dei med alle residyane, deretter prøvde eg å plukke vekk dei residyane som i versjon 2 var blitt bestemt til feil SSE, og trene dei beste nettverka om igjen. Når eg trena med setta utan dei som var blitt feilbestemt, blei bestemminga på ukjent sett (same settet som blei brukt heile tida – eg plukka ikkje vekk dei som var feilbestemt i valideringssettet) mykje meir stabil, men ikkje noko betre. Dette var mest sannsynlig fordi nettverket overtrena.

I denne delen kunne eg bruke om igjen det analyseprogrammet eg hadde skrive for versjon 2. Eg slapp å skrive nokon nye program for å behandle dataa i denne delen av oppgåva.

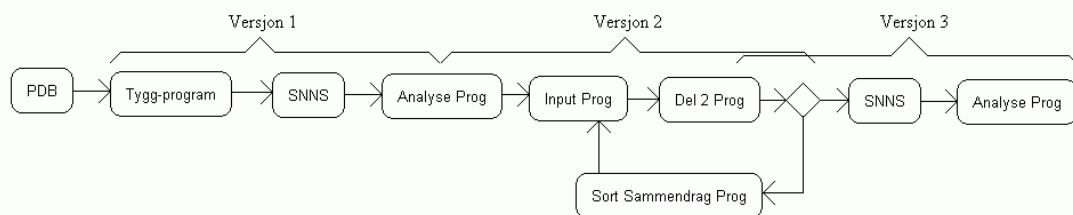
Den beste bestemminga eg hadde fant var på eit nettverk med 2 naboar,  $\eta=0,1$ ,  $\mu=0,5$ ,  $c=0,15$ ,  $d_{max}=0,1$ , trena i 1800 cyklar med alle resultatata frå versjon 2. Dette nettverket bestemte då 1141 residyar korrekt, 309 feil av totalt 1450, som tilsvarer 78,7 % korrekt. Fordelt på SSE ser det slik ut:

	# som skulle finnes	# korrekt funnet	#totalt funnet
$\alpha$	471	399 84,7%	446
$\beta$	329	203 61,7%	276
$\gamma$	650	539 82,9%	728

**Tabell 4.3:** Fordelinga over kor god bestemminga var for dei forskjellige SSE etter versjon 3. Andre kolonne viser her kor mange residyar av kvar SSE-type som fans i proteina. Tredje kolonne viser kor mange som blei korrekt funnet, og siste kolonne viser kor mange som blei bestemt av programmet til å vere av den enkelte SSE-type.

Sjå kapittel 6.3.3 for fleire resultat.

## Kapittel 5: Implementering



**Figur 5.1:** Diagram som viser korleis mine program har jobba mot kvarandre og mot SNNS og PDB.

### 5.1 Tygg-programmet

Dette er programmet eg har laga for å lage inputfilene til dei nevrale nettverka eg brukte i dei første forsøka (kap 4.3.1 og 4.3.2) og i versjon 1.

Dei første utgåvene av dette programmet laga i prinsippet 4 inputmatriser og ei outputmatrise (fasit) for heile proteinet. Tanken bak dette var at det skulle vere enklare å parallelisere dei nevrale nettverka. I den endelige versjonen lagar programmet nå ei inputmatrise og ei outputmatrise (fasit) for kvar enkelt residy i proteinet. Her blir ikkje lenger heile matrisa brukt, men bare ein del rundt diagonalen, sjå kapittel 4.3.3 for meir detaljar.

Programmet bruker ei inputfil som gir den verdiane til enkelte parameter og namna på pdb-filene det skal prosessere.

Dei tre hovuddelane går i ei while-løkke.

#### 5.1.1 Innlesing

Første del av programmet lesar gjennom pdb-filene enkeltvis og lagrar den informasjonen programmet trenger. Den er satt til å lagre informasjon frå pdb-linjer som begynner på "HELIX", "SHEET" og "ATOM" (PDB-filer kap 1.5.1). All informasjon blir lagra i tabellar og i posisjonar som reflekterer residynummeret eller residynummera som den gjelder for.

#### 5.1.2 Lag avstandsmatrise

Andre del lagar ei avstandsmatrise som ein tabell i minnet, etter formelen:  $\sqrt{(x1 - x2)^2 + (y1 - y2)^2 + (z1 - z2)^2}$ . Alle verdier blir korta ned til eit visst tal desimalar, som er angitt i byrjinga av inputfila. Eg brukte 3 desimalar når eg genererte mine matriser.

#### 5.1.3 Generer mønsterfil (.pat)

Denne delen av programmet generer mønsterfila i to omgangar. Først lagar den ei mellombels fil med alle input- og fasitmønstra.

Når den er ferdig med å gå gjennom alle pdb-filene

- går den ut av while-løkka som lesar inn data.
- generer så den informasjonen som skal stå i byrjinga av mønsterfila.
- og kopierer alle data over frå den mellombels fila til den som skal bli brukt vidare.

Dette er fordi starten av mønsterfila skal innehalde informasjon om kor mange mønstre som finnes i mønsterfila, og dette er ikkje mogleg å vite før ein har gått gjennom alle pdb-filene.

### Døme frå ei mønsterfil med $r=3$ :

```
SNNS pattern definition file V4.2
generated at ??

No. of patterns : 223
No. of input units : 21
No. of output units : 3

# input pattern 1
3.808 6.468 5.495 8.525 10.098 11.324
3.814 5.336 8.925 11.52 13.513
3.817 7.046 10.494 12.761
3.787 6.951 8.961
3.849 6.421
3.786

# filnavn: lsgt.pdb GLY 19 output pattern 1
0 0 1
```

Det som vises her er toppteksten til ei mønsterfil. Det står litt tekst for å angi kva versjon av SNNS fila er laga for, linje for dato, og informasjon om talet på mønstre ("No. Of patterns:"), samt talet input-/outputnodar den er berekna på. Etter det kommer første inputmønster, og til slutt første outputmønster (fasit) med informasjon om kva pdb-fil den er henta frå, kva aminosyre den er, og residynummer.

## 5.2 SNNS

Når eg brukte SNNS så nytta eg det grafiske grensesnittet til å generere opp nettverka og kjøre dei første treningane på dei store nettverka. Etterkvart som eg fann storleiken på dei nettverka eg ville jobbe med så byrja eg å kjøre treningane frå batchmodulen. Denne modulen er berekna på å kunne kjøre mange treningar av fleire forskjellige nettverk som bakgrunnsoperasjonar når systemet bruker lite resursar.

Batchmodulen er modellert etter programmeringsspråk som AWK, Pascal, Modula2 og C.

Dei fleste operatorane har fleire måtar dei kan bli skrive på, og variablar trenger ikkje å bli deklarerert før dei blir brukt. Språket er bygd opp med dei vanlegaste variabeltypene, standard operatorar (+, -, \*, osv), kontrollstrukturar (if, for, while, osv), og med ein del kall som går direkte mot SNNS-kjernen. Kalla lar ein utføre det meste av det ein kan gjere frå det grafiske grensesnittet, og ein kan då kjøre alle sine treningar fortløpande utan å måtte gå inn i det grafiske grensesnittet for å gjere endringar kvar gong ei trening er ferdig.

SNNS har nokre systemvariablar som brukaren bare har lesetilgang til. Eg brukte nokre av desse i dei første forsøka for å sjå om treninga av nettverka faktisk hadde noko effekt. Desse variablane blei brukt[9]:

- SSE (Sum Squared Error): Summen av dei kvadrerte forskjellane til den enkelte outputnode. Denne er definert av formelen:

$$SSE = \sum_{m \in \text{mønstre}} \sum_{j \in \text{output}} (t_{mj} - o_{mj})^2$$

der  $t_{mj}$  er fasiten til node  $j$  i mønster  $m$ , og  $o_{mj}$  er noden sin faktiske verdi.

- MSE (Mean Squared Error): SSE delt på talet på treningsmønstre.
- SSEPU: SSE delt på talet outputnodar i nettverket.
- CYCLES: Talet på syklusar som er trena til nå.

Slik eg brukte batchmodulen så lasta eg nettverket eg skulle trene inn i minnet og brukte nøsta FOR-løkker for å få trene det nettverket med forskjellige verdiar på parameterane. Eit sett av verdiar for læreparameter (kap 3.2.1), moment (kap 3.2.2), Flat Spot elimination value (kap 3.2.2) og  $d_{\max}$  (kap 3.2.1) vil eg vidare i oppgåva kalle for verdisett. I følgje SNNS-handboka[9] er typiske verdiar for  $d_{\max}$  0, 0,1 eller 0,2. Eg sette  $d_{\max}$  til å vere lik 0,1 i alle mine treningar. Eg følte det burde vere ein viss toleranse for feil under treninga – men ikkje altfor mykje.

Under trening bytta eg mellom trenings- og valideringsfilene etter kvar 100 syklusar, kjørte valideringsfila gjennom framoverfasen og skreiv ut ei resultatfil. Dette var for å ta ut mellombels resultat som kunne vise om nettverka trene godt eller dårlig.

Resultatfila var bygd opp med

- informasjon om dato
- kor mange mønstre den inneheldt
- kor mange input- og outputnodar som blei brukt
- og ei opplisting av resultata med fasit.

Dvs. at mesteparten av resultatfilene var opplistingar som såg slik ut:

```
#1.1
0 0 1
0.06395 0.11926 0.75761
#2.1
0 0 1
0.06395 0.11926 0.75761
```

Her vises bare opplistinga av resultata med fasit. 1. linje viser mønsternummer, 2. linje viser fasiten og 3. linje viser kva verdiar som outputnodane viste. 4.-6. linje er eit mønster tilsvarande det første. Det gjekk fint å skrive til den same resultatfila kvar gong eg skulle ha ut eit resultat. Slik fekk eg alle resultata for eit verdisett i ei fil, og kunne då enkelt prosessere den og deretter sjå utviklinga det spesifikke verdisettet hadde hatt under trening.

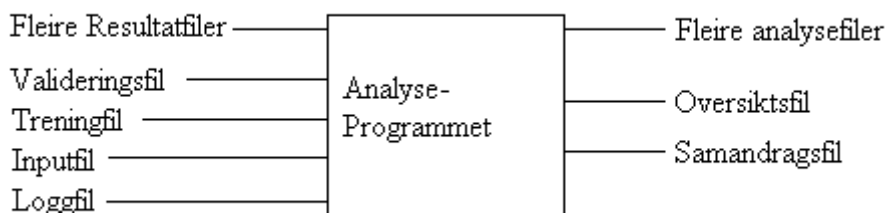
Under kjøring redirigerte eg output frå SNNS til ei loggfil som eg brukte vidare i analyseprogrammet.

### **5.3 Analyseprogrammet**

Dette programmet blei laga for å analysere resultatfilene frå SNNS.

Først leser det inn og lagrar informasjon frå både treningsmønsterfila og valideringsmønsterfila, då spesielt informasjon om kva residynummer den enkelte bestemming gjelder for, og kva aminosyre det er. I tillegg kopierer den under kjøring ein del informasjon frå loggfila til dei aktuelle treningane. Deretter leser den inn fasiten og bestemminga for alle residiane frå kvar enkelt resultatfil. Til slutt sjekkar den dei mot kvarandre.

### 5.3.1 Outputfiler frå analyseprogrammet



**Figur 5.2:** Figur som viser samanhengen mellom dei forskjellige filene brukt og generert av analyseprogrammet. Inputfila blei brukt til å angi kva filer som skulle bli behandla, og til parameterverdiar til programmet.

Under kjøring skriver programmet ut fleire filer:

#### 5.3.1.1 Analysefila (.analyse)

Ei outputfil blir laga for kvar enkelt resultatfil frå SNNS. Kvar linje der var bygd opp med residynummer, aminosyrenamn, 3 siffer fasit, 3 reele tall som er resultatet av bestemminga SNNS har gjort, og til slutt feil/KORREKT. Eg brukte store bokstavar på KORREKT for lettare å kunne sjå kva som var korrekt og feil bestemt når eg las gjennom fila manuelt. Døme:

```
1NCL-150.pdb
 9  GLU  0 0 1    0.10503    0.72321    0.20258  feil
10  ARG  0 1 0    0.10503    0.72321    0.20258  KORREKT
11  THR  0 1 0    0.10503    0.72321    0.20258  KORREKT
12  PHE  0 1 0    0.10503    0.72321    0.20258  KORREKT
13  LEU  0 1 0    0.10503    0.72321    0.20258  KORREKT
14  ALA  0 1 0    0.10503    0.72321    0.20258  KORREKT
15  VAL  0 1 0    0.10503    0.72321    0.20258  KORREKT
16  LYS  0 0 1    0.04169    0.18669    0.6462   KORREKT
17  PRO  1 0 0    0.04674     0.183     0.73692  feil
18  ASP  1 0 0    0.84038    0.04093    0.09618  KORREKT
19  GLY  1 0 0    0.5867     0.05035    0.16336  KORREKT
```

Etter alle residyane i ei pdb-fil blei det skrive ut eit samandrag for den fila:

```
1NCL-150.pdb #a:70:25^25 #b:26:22^38 #g:47:33^80 #t:143:80
```

Denne siste linja viser korleis bestemminga er for den pdb-fila, fordelt på SSE-type. Sjå 5.3.1.2 for ei nærmare forklaring.

#### 5.3.1.2 Oversiktsfil (.kort.analyse)

Ei outputfil som gir ut korleis bestemminga fordelt på kvar SSE er for kvar av pdb-filene som er gitt inn i trenings- og valideringsfilene.

Denne er bygd opp med ei linje som viser kva resultatfil resultata kommer frå. Ei linje henta frå loggfila, som viser dato og klokkeslett, talet på syklusar og til slutt 3 nøkkeltal frå SNNS: SSE, MSE og SSEPU (kap. 5.2).

Deretter kommer linjer for kvar enkelt pdb-fil. Først namnet på pdb-fila, så kommer #a: alpha, #b: beta, #g: gamma, desse 3 angir bestemminga for kvar SSE-type. Første tal etter type angir kor mange av den typen som var i fila, deretter kor mange som er korrekt bestemt og til slutt talet som totalt var bestemt til å vere av den typen. Så kjem talet på aminosyrer i fila, og kor mange som totalt har blitt bestemt korrekt. Til slutt skrives det ut informasjon om ein av SSE-typene si bestemming har ein korrelasjonskoeffisient over ein terskelverdi. Terskelverdien blir angitt i inputfila til

programmet. Korrelasjonskoeffisient er ein verdi som ligger mellom -1 og +1 og viser med eit enkelt tal om det finnes mange residyar som er blitt bestemt til å vere noko anna enn dei skulle vore bestemt å vere. Sjå eventuelt [1] for meir informasjon om korrelasjonskoeffisienten. Til slutt i kvar bolc kjem det ei linje som er samandraget av alle linjene over.

Under vises resultatata frå ei lita valideringsfil etter 50 syklusar.

```
NY FIL3-doble-dim-11-0.100000-0.750000-0.200000-val.res.analyse
13.11.2004 19:44 CYCLES:50 SSE:831.846 MSE:0.330754 SSEPU:277.282
 1KDK-177.pdb #a:42:33^115 #b:129:98^126 #g:165:66^210 #t:336:197
 1PTR-50.pdb #a: 6: 4^ 4 #b: 7: 6^ 16 #g: 30:20^ 27 #t: 43: 30
 1UCD-190.pdb #a:86:63^ 67 #b: 40:33^ 44 #g: 57:42^139 #t:183:138
 1X9G-200.pdb #a:72:58^ 67 #b: 37:30^ 36 #g: 76:61^149 #t:185:149
 2C2C-112.pdb #a:57:45^ 46 #b: 0: 0^ 13 #g: 48:34^ 92 #t:105: 79
FULL: #a:229:170^187 #b:168:132^181 #g:289:223^505 #t:686:525
```

### 5.3.1.3 Samandragstil (.kort.kort.analyse)

Den siste fila bruker bare den siste linja frå oversiktsfila – samandraget, og inneheld samandraga frå alle filene. Denne er laga for å fort kunne sjå gjennom resultatata frå mange forskjellige resultatfiler. På slutten av kvar linje blir det skrive ut ein kommentar. Den viser om talet korrekt bestemte residyar for ein SSE delt på det totale talet av den SSE, er over ein viss terskelverdi. Terskelverdien blir angitt i input. Desse kommentarane begynner med ALPHA, BETA eller GAMMA som viser kva type SSE som var over terskelverdien. Her vises resultatata frå ei lita valideringsfil, og resultatata frå den si tilhøyrande treningsfil.

```
3-doble-dim-11-0.100000-0.750000-0.200000-val.res.analyse
FULL: #a:229:170^187 #b:168:132^181 #g:289:223^505 #t:686:525
FULL: #a:229:215^362 #b:168:142^217 #g:289: 81^469 #t:686:438 ALPHA 0.93
FULL: #a:229:168^184 #b:168:120^154 #g:289:239^532 #t:686:527
FULL: #a:229:216^423 #b:168:113^142 #g:289: 96^544 #t:686:425 ALPHA 0.94
FULL: #a:229:186^237 #b:168:157^322 #g:289: 93^331 #t:686:436 BETA 0.93
FULL: #a:229:228^510 #b:168:130^176 #g:289: 0^510 #t:686:358 ALPHA 0.99
FULL: #a:229:147^153 #b:168:128^163 #g:289:248^523 #t:686:523
FULL: #a:229:191^230 #b:168:131^190 #g:289:191^496 #t:686:513
FULL: #a:229:183^222 #b:168:140^207 #g:289:185^479 #t:686:508
FULL: #a:229:229^524 #b:168:123^162 #g:289: 0^524 #t:686:352 ALPHA 1
```

```
3-doble-dim-11-0.100000-0.750000-0.200000-tre.res.analyse
FULL: #a:767:764^2001 #b:457:311^513 #g:1291: 1^ 1 #t:2515:1076 ALPHA 0.99
```

### 5.3.2 Endringar i analyseprogrammet i versjon 2

I versjon 2 trengte eg ei utgåve av analyseprogrammet som tok inn bare ei mønsterfil. Denne utgåva laga eg for å kjøre på resultatfiler som eg ville bruke vidare.

Etter at eg var ferdig å trene opp nettverka, kjørte eg mønstersetta gjennom dei ein gong og tok ut nye resultatfiler. Mønstersetta blei kjørt gjennom nettverka, slik dei ville ha blitt brukt i ein endeleg utgåve av programpakken som var målet med denne oppgåva. Dette for å få ut resultatfiler som inneheldt bare resultat som eg skulle jobbe vidare med. Resultatfilene eg allereie hadde etter versjon 1 inneheldt fleire mellombels resultat som eg hadde tatt ut under trening.

## 5.4 Input-programmet

Dette er eit enkelt parseprogram. Det tar inn ei fil med namna på dei analysefilene som skal bli brukt i versjon 2. Programmet lagar ut i frå namna ei ny fil som kan bli brukt som input til Del2-programmet. Først les det inn alle filnamna som skal bli brukt. Deretter skriver det ut første og andre namnet det leste inn, så skriver den ut eit utfilnamn som er ein samanslåing av dei to inputfilene. Deretter gjer den tilsvarende

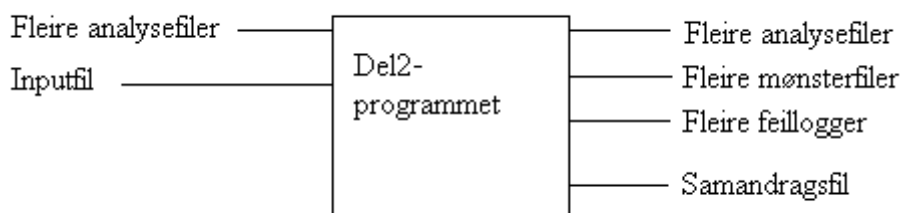
for første og tredje namn, og vidare slik inntil første filnamn er kombinert med alle dei andre filnamna. Etterpå gjer det tilsvarende med det andre filnamnet først, det tredje filnamnet først, osv. Programmet hadde to parameter som blei skrive direkte inn i programkoden ved kvar kjøring. Ein parameter for kor mange filer den skulle prosessere, og ein for kor lange filnamna var.

```
7-doble-2.res.analyse
7-doble-1.res.analyse
a7-2_7-1.analyse
7-doble-2.res.analyse
6-doble-3.res.analyse
a7-2_6-3.analyse
```

## 5.5 Del2-programmet.

Programmet leser inn ei analysefil og lagrar den i minnet. Deretter les det inn den eller dei andre analysefilene som det skal slå analysefila saman med. Kor mange analysefiler som skal bli slått saman i kvar kjøring blir angitt i inputfila. Den lagrar snittet av analysefilene i minnet etterkvart, slik at den enkelt kan slå saman resultatata frå fleire analysefiler. Til slutt skriver den ut ei mønsterfil som skal bli brukt i versjon 3, ei resultatfil og ei fil med alle residyar som ikkje blir korrekt bestemt. Siste var tenkt at kunne bli brukt til å finne ut i kor stor grad residyar blei bestemt feil.

### 5.5.1 Outputfiler frå Del2-programmet



**Figur 5.3:** Figur som viser samanhengen mellom dei forskjellige filene brukt og generert av Del2-programmet. Inputfila blei brukt for å angi kva analysefiler som skulle bli slått saman.

Under kjøring skriver programmet ut fleire filer:

#### 5.5.1.1 Analysefila (.analyse)

Denne fila er lik analysefila (kap 5.3.1.1) som blei laga av analyseprogrammet (kap 5.3) og er snitta av dei filene som er blitt slått saman.

```
1PTR-50.pdb
8 PRO 0 0 1 0.069875 0.426815 0.45961 KORREKT
9 VAL 0 0 1 0.01076 0.72879 0.23228 feil
```

### 5.5.1.2 Mønsterfila (.pat)

Denne fila er mønsterfila som skal bli brukt i versjon 3. Den er bygd opp av eit inputmønster som består av grupper på 3 reelle tal. Kvar gruppe er bestemminga til ei aminosyre, og kor mange grupper avhenger av kor mange naboar som skulle bli brukt i den enkelte treninga i versjon 3. Til slutt skrives outputmønsteret ut i lag med informasjon om residynummer og namn. Her vises toppteksten og eit mønster med 2 naboar:

```
SNNS pattern definition file V4.2
generated at ??

No. of patterns : HUSK Å SKRIVE INN KORREKT HER
No. of input units : 15
No. of output units : 3

#input pattern 2
0.0364075 0.445632 0.49581
0.0024775 0.91648 0.073325
0.0023725 0.935222 0.066675
0.0031125 0.943697 0.05142
0.0028075 0.969398 0.02951
#output pattern 2 11 THR
0 1 0
```

Her skreiv den ut talet på mønstre nederst i fila slik at eg måtte flytte det talet opp før eg brukte fila i SNNS. Siste linja i fila var slik:

```
DENNE SKAL FREMST: ANTALL PATTERNS: 1450
```

### 5.5.1.3 Feilloggfil (.analysefeil.log)

Denne var temmelig lik analysefila bortsett frå at den berre inneheld dei residiane som blei bestemt til å vere feil, og ingen informasjon om kva filer dei kom frå.

```
268 VAL 0 0 1 0.0109 0.794068 0.193403 feil
6 PRO 0 0 1 0.0493575 0.55915 0.324313 feil
8 TYR 0 0 1 0.479045 0.101217 0.321267 feil
```

### 5.5.1.4 Samandragsfila (sammendrag.log)

Denne fila viste kor mange som var korrekt/feil bestemt, etter at det var blitt tatt snittet av to filer. I tillegg vises kor mange som var korrekt/feil i dei to filene som det blei tatt snittet av. Merk at talet residyar totalt i siste fila aldri er høgare enn første. Dette er fordi det er bare dei residiane som blei tatt med i snittet som blir talt med.

```
a3-3_2-3.analyse Tot korr: 1121 tot feil: 357 Sum: 1478 :
3-doble-3.res.analyse # korrekt: 1103 #feil: 375 Sum: 1478
2-doble-3.res.analyse # korrekt: 1027 #feil: 451 Sum: 1478
```

## 5.6 Sortsammendrag

Dette er eit enkelt parseprogram som lesar gjennom samandragsfilene frå Del2-programmet og sorterar vekk alle resultat som ligger over ein viss terskelverdi. Terskelverdien blei skrive inn i samandragsfila før kvar kjøring. Programmet les inn den samandragsfila som blei danna av Del2-programmet. Deretter skriver det ut ei ny fil, som bare inneheld dei resultatata som ligg over terskelverdien. Eg kjørte dette programmet fleire gonger og auka terskelverdien mellom kvar gong, inntil eg hadde fått fjerna nok resultat til at eg følte eg kunne jobbe vidare med dei eg då hadde.

### ***5.7 Kva programmer blei gjenbrukt i versjon 3***

Forutan Del2-programmet som blei brukt til å generere mønsterfilene i versjon 3, var det bare analyseprogrammet eg hadde behov for å bruke. Eg brukte den utgåva som tar inn 2 mønsterfiler sidan det som skulle bli gjort her var å analysere alle resultata etter treningane av Type II-nettverka.

## Kapittel 6: Kjøringar og resultat

I utgangspunktet ønska eg å teste ut om forskjellige topologiar for KNN gav store forskjellar i bestemminga av SSE til proteina. I tillegg visste eg at eg måtte teste ut forskjellige verdiar for parameterane til læringsparadigmet.

Spesielt læringsparameteren må bestemast eksperimentelt. I dei første forsøka eg gjorde (6.1 og 6.2) var det viktigaste eg testa forskjellige topologiar, mens i dei andre forsøka (6.3) kom det inn eit element til – graden av overlapping. Dette medførte at i staden for å kjøre mange testar på forskjellige topologiar, blei det viktigare å teste om ulik grad av overlapping gav forskjellige resultat. Deretter jobba eg i versjon 2 med å slå saman alle dei resultatata eg hadde valt å behalde i versjon 1, før eg til slutt testa ut nettverk med forskjellig tal naboresidyar i versjon 3.

### 6.1 Dei store nettverka som jobba på heile matrisa

Eg laga fleire variantar av desse nettverkstopologiane og den topologien beskrive i kap 6.2. Dette var for å teste ut om ein variant hadde ei betre bestemming enn dei andre. På fart ville dei mindre variantane heilt klart vere betre enn dei store. Ganske enkelt fordi det var der færre forbindingar å prosessere.

Sidan eg genererte opp alle nettverka før eg testa dei, såg eg ikkje at dei var for store til å bli brukt før eg begynte å teste. Hadde eg laga og testa dei enkeltvis så hadde eg sloppe å generere opp alle. Eg kjørte få treningar av desse nettverka fordi eg såg tidleg i dei mellombelse resultatata at desse ville vere for store til å kunne bli brukt. Dei forsøka eg gjorde brukte så lang tid på å trene at eg ikkje klarte å få nokon som helst variasjon i resultatata dei gav ut, i ei rimeleg treningstid. Dette gjeld både for nettverka på 300x300 og 200x200.

#### 6.1.1 300x300:

Sidan desse nettverka skulle bestemme strukturen til eit heilt protein i ein kjøring, så ville eg her at dei skulle klare eit protein av ein grei storleik. Eg testa først med 300 residyar. Dette medførte at eg fekk eit stort inputlag – totalt 90000 nodar. Tanken bak nodeklyngene var at desse skulle gjere det enkelt å seinare dele opp nettverket og parallellisere det.

Dette blei laga i 3 hovudvariantar, og med 2 variantar som hadde eit ekstra skjult lag med 2 nodeklynger. Alle variantane hadde desse to laga:

- Inputlag med 4 nodeklynger 150x150 (1-4). 22500 nodar/nodeklynge.
- Eit outputlag 2x300 (10 eller 12). 600 nodar.

Variant	Skjult lag 4 nodeklynger (5-8)	Ekstra Skjult lag 2 n.k. (9-10)	Skjult lag (9 eller 11)	Fil (MB)
Minimalt	25x6(150/n.k.)		20x20(400)	264
Medium	25x12(300/n.k.)		25x24(600)	524
Medium+	25x12(300/n.k.)	30x15(450/n.k.)	25x24(600)	531
Stort	20x30(600/n.k.)		40x30(1200)	1068
Stort+	20x30(600/n.k.)	30x30(900/n.k.)	40x30(1200)	1095

**Tabell 6.1:** Tabellen viser korleis dei forskjellige nettverka var bygd opp. I tillegg viser den storleiken til nettverksfilene. Tala i parentes angir kor mange nodar totalt som finnes i det laget eller i kvar nodeklynge (n.k.).

I det første skjulte laget (5-8) behaldt eg minst like mange nodar som det skal bestemast residyar i nodeklynga før det skjulte laget, unntatt det minimale. Dette

laget er berekna på å lage nodeklyngene til nesten sjølvstendige nettverk. Det einaste dei manglar er outputlaget. Output frå dette laget blir samla saman i det siste skjulte laget, bortsett frå i dei to variantane med eit ekstra skjult lag. Det ekstra skjulte laget er lagt inn for å slå saman resultatane frå 2 og 2 sett med nodeklyngar før dei 2 nodeklyngene i dette laget blei slått saman. Det siste skjulte laget er laga stort nok til å dekke alle residyaner som skulle bestemast med 2 eller 4 nodar til kvar.

Det minimale nettverket er ikkje bygd opp på same måten som dei andre. Det er laga så lite som mogleg for å teste om eg fekk variasjon i output i det heile tatt. Eg hadde liten tro på at nettverket ville gi eit resultat som eg kunne bruke vidare, men eg ville teste det ut for å sjå om eg fekk eit resultat med variasjon i output. Det første skjulte laget her har ei node pr residy i diagonalen, og det siste skjulte laget har bare litt redundans.

Det var forbindingar mellom nodeklyngene 1 og 5, 2 og 6, 3 og 7, og 4 og 8. Dei 4 nodeklyngene (5-8) i det første skjulte laget hadde forbindingar til det andre skjulte laget (9). Til slutt var det andre skjulte laget (9) forbundet til outputlaget (10).

I dei to ekstra variantane av Medium og Stort blei outputlaget nummerert til 12. Her var det forbindingar mellom nodeklyngene 1 og 5, 2 og 6, 3 og 7, og 4 og 8. Nodeklynge 5 og 6 hadde forbindingar til nodeklynge 9, 7 og 8 hadde forbinding til 10. Nodeklyngene 9 og 10 var forbundet til det siste skjulte laget (11) Til slutt var det siste skjulte laget (11) forbundet til outputlaget (12).

### 6.1.2 200x200:

Her prøvde eg framleis å bestemme alle SSE i eit protein direkte, men reduserte talet på nodar i inputlaget til litt under halvparten, dvs 40000. Slik kunne eg gjere dei skjulte laga forholdsmessig større. Dette gjorde òg at maks storleik på proteina som eg kunne bestemme blei 200 residyar.

Dette blei laga i 3 variantar som alle hadde desse to laga:

- Inputlag med 4 nodeklyngar 100x100 (1-4). 10000 nodar/nodeklynge.
- Eit outputlag 2x200 (10). 400 nodar.

Variant	Skjult lag, 4 nodeklyngar (5-8)	Skjult lag (9)	Fil (MB)
Minimalt	20x20(400/n.k.)	20x20(400)	312
Medium	25x20(500/n.k.)	20x20(400)	380
Stort	40x40(1600/n.k.)	30x30(900)	1476

**Tabell 6.2:** Tabellen viser korleis dei forskjellige nettverka var bygd opp, og kor stor nettverksfila var. Tala i parentes angir kor mange nodar totalt som finnes i det laget eller i kvar nodeklynge (n.k.).

Dei skjulte laga i det minimale her er laga slik at kvar nodeklynge og det siste skjulte laget har 2 nodar for kvar residy som skal bestemast. Mediumnettverket er laga litt større enn det minimale ved å legge inn litt redundans i nodeklyngene. I det store har eg dobla dimensjonane til nodeklyngene i forhold til det minimale, og det siste skjulte laget er dobla og lagt til litt redundans.

Det var forbindingar mellom nodeklyngene 1 og 5, 2 og 6, 3 og 7, og 4 og 8. Dei 4 nodeklyngene (5-8) i det første skjulte laget hadde forbindingar til det andre skjulte laget (9). Til slutt var det andre skjulte laget (9) forbundet til outputlaget (10).

### 6.1.3 Konklusjon

Ut i frå treningane såg eg at alle desse nettverka blei altfor store for mine maskiner. At det skjulte laget var mykje mindre enn inputlaget medførte at nettverket

generaliserte dårlig. Nettverka brukte framleis veldig lang tid på å trene. Eg klarte faktisk ikkje å få trena desse nettverka nok til at det blei nokon som helst variasjon i output. Skulle desse nettverka kunne fungert så måtte dei sannsynlegvis ha blitt kjørt på ein stormaskin, og då med eit mykje større skjult lag. Nettverka hadde godt mogleg blitt for store sjølv for ein stormaskin. Sidan eg her prøver å bestemme fleire residyar samtidig er òg kompleksiteten til desse nettverka veldig høg. I tillegg er det ei ulempe at dei har eit avgrensa tal residyar dei kan bestemme. Eg vil ikkje anbefale å prøve meir testing av denne typen nettverk.

## 6.2 Dei store nettverka som jobba på 1/4 av matrisa

Her har eg i praksis delt dei største nettverka i 4, med ein liten overlapping slik at eg ikkje skulle miste bestemminga av SSE for residyar som ligg i midten av aminosyrekjeda.

Alle nettverka her var fullt forbunde. Eg laga 6 variantar som alle hadde desse to laga:

- Inputlag 107x107 (1). 11449 nodar.
- Eit outputlag 2x107 (3 eller 4 i dei med eit ekstra skjult lag). 214 nodar.

Variant	Skjult lag (2)	Ekstra Skjult lag (3)	Fil (MB)
Minimalt	27x8 (216)		47
Medium	27x24 (648)		140
Medium+	27x24 (648)	27x16 (432)	144
Stort	54x48 (2592)		558
Stort+	54x48 (2592)	54x32 (1728)	637
Ekstra stort	81x72 (5832)		1254

**Tabell 6.3:** Tabellen viser korleis dei forskjellige nettverka var bygd opp. Det skjulte laget (2) var det som hovudsaklig varierte på desse nettverka, men i tillegg står det her 2 nettverk med eit ekstra skjult lag (3). Siste kolonne viser kor stor nettverksfila var. Tala i parentes angir kor mange nodar totalt som finnes i det laget.

Her sjonglerte eg litt med tal for å få ei meir kvadratisk matrise enn outputlaget, samtidig som eg framleis hadde minst like mange nodar som outputlaget. Deretter multipliserte eg opp dimensjonane i dei andre nettverka. Det skjulte laget i medium har 3 gonger så mange nodar som det skjulte laget i det minimale, det store har doble dimensjonar av medium, og det ekstra store har triple dimensjonar av medium nettverket. Der det finnes er det ekstra skjulte laget er laga som 2/3 av det første skjulte laget.

Etter å ha gått over til Backprop Momentum, var den beste bestemminga eg fekk med 107-nettverka på rundt 70 rette av 107. Det var på eit av 5 valideringsprotein. Dei andre i valideringssettet bestemte då heller dårlig, totalt 255 av 535. Den totalt sett beste bestemminga på 107-nettverka var på treningssetta, noko som indikerer at dei nettverka vart veldig fort overtrena.

Her prøvde eg i tillegg å trene eit medium og eit minimalt nettverk med forskjellige høge læreparametrar, slik som 2, 4 og 5. Eg varierte då moment mellom 0,5 og 1,0, og Flat Spot mellom 0 og 0,25. Desse blei trena i 250 syklusar kvar.

Det minimale trena eg med læreparametrar på 5, 4, 3, 2, 1, 0,5 og 0,25, Moment på 0,25, 0,5 0,75 og 1, og Flat Spot på 0,1, 0,15, 0,2, 0,25. Totalt 75 sett som eg trena i 600 syklusar.

I tillegg kjørte eg eit par treningar på dei forskjellige nettverka der eg sette at læreparameteren skulle bli redusert under treninga.

## 6.2.1 Konklusjon

Dette blei framleis for stort slik at generaliseringa blei for dårlig. Inputlaget her var litt under dobbelt så stort som det største eg har i den endelige utgåva. Outputlaget derimot var ein del større, og kompleksiteten var ein del høgare enn i den endelige utgåva. Dette fordi eg prøvde å bestemme mange aminosyrer samtidig. Sidan dette i praksis var ¼-del av nettverka frå 6.1 og det framleis viste seg å ha ei dårlig bestemming så vil eg heller ikkje anbefale å teste meir på denne typen nettverk.

## 6.3 Kaskade

### 6.3.1 Versjon 1

I motsetnad til tidlegare variantar av nettverka så representerer dei forskjellige variantane her ei ulik grad av overlapping. 1doble-dim hadde ein overlapping på ein residy, 2doble-dim på 2, osv. opp til 8doble-dim som hadde ein overlapping av 8 residyar. Det viste seg å vere stor variasjon i kor god bestemminga av SSE var, alt etter kor stor overlappinga var.

Nettverka her er alle laga slik at det skjulte laget har doble ( $1 \times 3 \rightarrow 2 \times 6$ ) dimensjonar av inputlaget. Her var inputlaga såpass små at sjølv på dei største nettverka så gjekk det fint å gjere det skjulte laget mykje større enn inputlaget. Alle nettverka hadde eit outputlag på  $1 \times 3$  nodar, og var fullt forbundet. Dette er dei nettverka eg har brukt:

Namn	Input	Skjult	Fil (kB)
1doble-dim	1x3 (3)	2x6 (12)	3
2doble-dim	2x5 (10)	4x10 (40)	13
3doble-dim	7x3 (21)	14x6 (84)	41
4doble-dim	6x6 (36)	12x12 (144)	104
5doble-dim	11x5 (55)	22x10 (220)	223
6doble-dim	13x6 (78)	26x12 (312)	429
7doble-dim	15x7 (105)	30x14 (420)	759
8doble-dim	17x8 (136)	34x16 (544)	1241

**Tabell 6.4:** Tabellen viser namnet på dei forskjellige nettverka, storleiken på inputlaget og det skjulte laget, og til slutt kor stor nettverksfila var. Tala i parentes viser kor mange nodar det var i dei forskjellige laga.

I tillegg til dei som er nemnt ovanfor testa eg litt på to nettverk med overlapping på 9 og 10, og eit nettverk der eg kvadrerte dimensjonane i det skjulte laget. Dvs, med eit inputlag på  $7 \times 3$  så fekk eg eit skjult lag på  $49 \times 9$ . Overlapping med 9 og 10 blei store å jobbe med, og gav i utgangspunktet ei dårligare bestemming enn dei mindre nettverka. Nettverket med kvadrerte dimensjonar bestemte ikkje merkbar betre enn det med doble dimensjonar så eg gjorde ikkje fleire testar av den varianten.

Under treningane brukte eg 2 forskjellige mønstersett. Eit lite sett som blei brukt for å finne ut kva verdisett som var verdt å jobbe vidare med, og eit stort sett som blei brukt på dei verdisetta som skulle bli jobba vidare med.

Det minste mønstersettet bestod av:

- ei treningsfil bygd opp av 20 protein/pdb-filer som hadde t.d. 2515 mønstre ved overlapping på 3, og 2315 mønstre ved overlapping på 8
- ei valideringsfil bygd opp av 5 protein/pdb-filer som hadde t.d. 686 mønstre ved overlapping på 3, og 636 mønstre ved overlapping på 8.

Det store mønstersettet bestod av:

- ei treningsfil bygd opp av 40 protein/pdb-filer som hadde t.d. 6081 mønstre ved overlapping på 3, og 5681 mønstre ved overlapping på 8

- ei valideringsfil bygd opp av 10 protein/pdb-filer som hadde t.d. 1478 mønstre ved overlapping på 3, og 1378 mønstre ved overlapping på 8.

Grunnen til at eg brukte to mønstersetts var for å klare å teste ut mange forskjellige verdisetts utan å bruke altfor lang tid. Treningstida gjekk merkbar opp når eg gjekk over til dei store mønstersetta. Kva mønstersetts som blei brukt i lag med dei forskjellige verdisetta er angitt i teksten nedanfor.

Dei typiske verdiane for parametrane er oppgitt i kap 3.2.1 og 3.2.2, og var der igjen henta frå SNNS-handboka[9]. Når eg skulle finne parametrane eg skulle trene med valte eg det eg såg på som greie intervall av det som var oppgitt. Her følgjer i hovudsak dei parametrane eg har variert når eg har kjørt treningar, og kor langt eg har trena dei. Deretter kjem ein tabell for kvart nettverk med kva verdisetts eg fant ut var verdt å jobbe vidare med. Til slutt kjem ein tabell med resultatata eg tok med vidare.

### **6.3.1.1 1doble-dim og 2doble-dim:**

Desse nettverka blei bare trena med det store mønstersettet. Nettverka var så små at treningane ikkje tok veldig lang tid. Dei blei trena etter dei andre. Eg valte å trene desse etter å ha sett at det minste nettverket eg til då hadde, 3doble-dim, var det som bestemte best. Målet var å sjå om overlapping på bare 1 eller 2 òg ville bestemme bra. Når eg trena nettverka brukte eg fleire parameter her enn eg hadde brukt på 4doble-dim og oppover. Dette var fordi eg rekna på førehand med at desse treningane ikkje ville ta noko særlig lang tid.

Læreparameter: 0,1; 0,25; 0,5; 0,75; 1,0

Moment: 0,25; 0,5; 0,75; 1,0

Flat spot: 0,1; 0,15; 0,2; 0,25

Totalt 80 verdisetts for kvart nettverk, som alle blei trena i 2000 syklusar.

1doble-dim viste seg å ikkje vere brukande i det heile, men 2doble-dim fekk eg ut nokre gode resultat frå.

### **6.3.1.2 3doble-dim:**

Desse blei trena med det minste mønstersettet.

Læreparameter: 0,1; 0,25; 0,5; 0,75; 1,0; 2,0

Moment: 0,25; 0,5; 0,75; 1,0

Flat spot: 0,1; 0,15; 0,2; 0,25

Totalt 96 verdisetts som alle blei trena i 500 syklusar.

Etter å ha gått over resultatata frå desse treningane manuelt fant eg ut at å bruke ein læreparameter over 0,5 ikkje såg ut til å vere verdt å gjere i fleire testar. I dei vidare treningane tok eg med 1,0 i tilfelle det skulle vise seg at høgare parameter ville vere meir gunstig på eit av dei andre nettverka.

Når det gjelder Moment og flat spot såg eg liten variasjon i resultatata over desse parametrane, og tok derfor toppverdi, botnverdi og midtverdi. Eventuelt kunne eg ha prøvd parameterverdiar mellom dei eg valte om det såg ut til å vere noko å hente.

### **6.3.1.3 4doble-dim, 5doble-dim, 6doble-dim, 7doble-dim og 8doble-dim**

Desse nettverka blei trena med følgjande variasjonar på parametrar og med det minste mønstersettet:

Læreparameter: 0,1; 0,25; 0,5; 1,0

Moment: 0,1; 0,5; 1,0

Flat spot: 0,1; 0,12; 0,25

Totalt 36 verdisetts for kvart nettverk, som alle blei trena i 500 syklusar.

### 6.3.1.4 Tabellar over kva verdisettt eg jobba vidare med

Etter å ha manuelt gått gjennom resultatata frå treningane med det minste treningssettet fant eg dei verdisetta eg såg som gunstige å jobbe vidare med. Dette var dei verdisetta som eg fant hadde ein god utvikling i bestemminga, eller som hadde ei god bestemming allereie med det minste mønstersettet.

I teksten under tabellen vil det stå kva eg gjorde vidare med dei forskjellige verdisetta. I mange tilfelle så trena eg først nettverka i 1000 syklusar, gjekk gjennom resultatata frå dei treningane og plukka vekk eit par av nettverka som då ikkje såg ut til å gi eit resultat eg ønska å jobbe enda meir med. Deretter trena eg nettverka i enda 1000 syklusar og plukka så ut dei beste resultatata blant dei eg då hadde.

Her følgjer kva verdisettt som det blei gjort noko meir med for 2doble-dim, 3doble-dim, 4doble-dim, 5doble-dim, 6doble-dim, 7doble-dim og 8doble-dim:

#### 2doble-dim:

Sidan desse nettverka blei trena direkte med det store settet fører eg her bare opp dei verdisetta eg faktisk brukte.

#	$\eta$	$\mu$	c
1	0,1	0,5	0,2
2	0,1	0,75	0,1
3	0,1	0,75	0,15
4	0,1	0,75	0,2
5	0,5	0,25	0,25

Tabell 6.5

#### 4doble-dim:

Desse verdisetta blei trena med det store mønstersettet i 1000 syklusar.

#	$\eta$	$\mu$	c
1	0,1	0,1	0,1
2	0,1	0,5	0,25
3	0,25	0,5	0,12
4	0,25	0,5	0,25
5	0,5	0,1	0,1
6	0,5	0,1	0,12
7	1,0	0,5	0,12

Tabell 6.7

#### 3doble-dim:

Desse verdisetta blei trena opp i 10000 syklusar med det minste mønstersettet, for å sjå om det var ein merkbar forbetring når ein kom langt over 2000 syklusar. Sidan det ikkje var ei sterk merkbar forbetring trena eg seinare verdisettt bare opp i maks 2000 syklusar.

#	$\eta$	$\mu$	c
1	0,1	0,25	0,1
2	0,1	0,5	0,1
3	0,1	0,75	0,2
4	0,25	0,5	0,1
5	0,25	0,75	0,1
6	0,5	0,5	0,15

Tabell 6.6

Alle utanom det første blei deretter trena vidare til dei hadde totalt blitt trena i 2000 syklusar.

Deretter blei dei 6 same verdisetta trena i 2000 syklusar med det store treningssettet.

**5doble-dim:**

Desse verdisetta blei trenea med det store mønstersettet i 1000 syklusar.

#	$\eta$	$\mu$	c
1	0,1	0,1	0,1
2	0,1	0,5	0,1
3	0,1	0,5	0,25
4	0,25	0,1	0,1
5	0,25	0,1	0,12
6	0,25	0,1	0,25
7	0,5	0,1	0,1
8	1,0	0,1	0,1
9	1,0	0,1	0,12
10	1,0	0,5	0,1
11	1,0	0,5	0,12

Tabell 6.8

Deretter blei alle utanom 6 og 9 trenea i 1000 syklusar til. Nokon av desse verdisetta hadde best bestemming ved 2000 syklusar og blei då trenea i fleire syklusar for å sjekke om dei blei stadig betre.

**6doble-dim:**

Desse verdisetta blei trenea med det store mønstersettet i 1000 syklusar.

#	$\eta$	$\mu$	c
1	0,1	0,1	0,1
2	0,1	0,1	0,12
3	0,1	0,5	0,1
4	0,1	0,5	0,12
5	0,1	0,5	0,25
6	0,25	0,5	0,1
7	0,25	0,5	0,12
8	1,0	0,5	0,1
9	1,0	0,5	0,12

Tabell 6.9

Deretter blei alle utanom 1 og 2 trenea i 1000 syklusar til. Nokon av desse verdisetta hadde best bestemming ved 2000 syklusar og blei då trenea i fleire syklusar for å sjekke om dei blei stadig betre.

**7doble-dim:**

Desse verdisetta blei trenea med det store mønstersettet i 1000 syklusar.

#	$\eta$	$\mu$	c
1	0,1	0,1	0,1
2	0,1	0,1	0,12
3	0,1	0,1	0,25
4	0,1	0,5	0,1
5	0,1	0,5	0,25
6	0,25	0,1	0,1
7	0,25	0,5	0,12
8	0,5	0,5	0,1
9	0,5	0,5	0,12
10	0,5	0,5	0,25
11	1,0	0,1	0,12

Tabell 6.10

Deretter blei sett 2, 3, 5, 7 og 9 trenea i 1000 syklusar til.

**8doble-dim:**

Desse verdisetta blei trenea med det store mønstersettet i 1000 syklusar.

#	$\eta$	$\mu$	c
1	0,1	0,5	0,1
2	0,25	0,1	0,1
3	0,25	0,5	0,12
4	0,25	0,5	0,25
5	0,5	0,1	0,1

Tabell 6.11

Deretter blei sett 1, 2 og 4 trenea i 1000 syklusar til.

Når eg trena desse nettverka la eg inn ein tilfeldig støy på vektene. Støyen tilsvarte  $\pm 10\%$  av vektene, etter kvar 200 syklusar. Dette var for ytterligare å unngå at nettverket hamna i eit lokalt minima.

I tillegg har det vore nokre verdisett eg har kjørt for å teste ut spesifikke ting, og nokre sett som eg har måtte kjøre fleire gongar pga feil i batchfilene eller mønsterfilene.

Det er tydelig frå dei verdisetta eg har jobba vidare med at det er dei lave læreparametrane som gir best læring. Dette stemmer godt overens med det Terje Kristensen skriver om læreparameteren i si bok [8]. Han skriver der at for store verdiar kan føre til instabilitet i nettverket og ikkje tilfredstillande læring. Han skriver òg at for små læreparameter kan gi ein altfor langsam læring, men i mine kjøringar har det vore dei minste som oftast har gitt eit godt resultat. Eg trur forskjellen her ligger i at i si bok beskriv Kristensen bare vanlig Backpropagation, mens i mi oppgave enda eg opp med å bruke backpropagation med moment. Denne er ei sterk forbetring av det originale paradigmet – spesielt når det gjelder treningstid.

### 6.3.1.5 Resultata eg tok med vidare til versjon 2

Etter å ha gjennomgått alle resultata manuelt enda eg opp med at eg ville behalde resultata i tabell 6.12. Desse resultata var dei eg følte var dei beste frå kvar storleik av overlapping. Kor mange residyar i kjeda dei enkelte nettverka har hatt som overlapping vises med det første talet i namna deira.

Namn	V.sett	#Syklusar	Korrekt	Feil	Totalt	% korrekt
7doble-dim-2	3	1200	1023	375	1398	73,2
7doble-dim-1	3	900	1043	355	1398	74,6
6doble-dim-3	6	2400	1039	379	1418	73,3
6doble-dim-2	4	1500	1038	380	1418	73,2
6doble-dim-1	4	1300	1048	370	1418	73,9
5doble-dim-4	2	2000	1080	358	1438	75,1
5doble-dim-3	3	2200	1069	369	1438	74,3
5doble-dim-2	2	2900	1073	365	1438	74,6
5doble-dim-1	2	2000	1076	362	1438	74,8
4doble-dim-5	6	600	1088	370	1458	74,6
4doble-dim-4	3	1000	1078	380	1458	73,9
4doble-dim-3	2	1300	1095	363	1458	75,1
4doble-dim-2	2	1100	1078	380	1458	73,9
4doble-dim-1	2	600	1098	360	1458	75,3
3doble-dim-2	4	200	1103	375	1478	74,6
3doble-dim-1	1	2000	1100	378	1478	74,4
2doble-dim-5	5	300	1091	407	1498	72,8
2doble-dim-4	4	900	1060	438	1498	70,8
2doble-dim-3	3	400	1042	456	1498	69,6
2doble-dim-2	2	1200	1069	429	1498	71,4
2doble-dim-1	1	300	1052	446	1498	70,2

**Tabell 6.12:** Tabellen viser kva nettverk og verdisett eg fant ut at eg ville ta med vidare til versjon 2. Første kolonne viser namnet på nettverket. Neste kolonne viser nummeret på verdisettet. Nummeret er henta frå dei verdisetta eg valte å trene vidare (tabell 6.5-6.11). I tillegg står kor mange syklusar dei blei trena i, talet på residyar korrekt bestemt, feil bestemt og totalt. Tala som blir vist er frå valideringssettet. Til slutt vises kor mange prosent korrekte residyar nettverket fant.

Denne måten å bygge opp nettverka på såg ut til å vere lovande. I første versjon fekk eg opptil 75,3% korrekt. Noko som virka som eit godt resultat, sidan eg satsa på å få forbetra bestemminga ein god del i versjon 2 og 3.

Trening og analysing av nettverka i denne versjonen tok veldig lang tid ganske enkelt på grunn av mengda nettverk og verdisett eg testa ut. Eg føler eg har fått testa ut dei alternativa som har vore aktuelle å bruke, og vidare forsøk her bør eventuelt teste ut andre måtar å bygge opp liknande nettverk på.

### 6.3.2 Resultata frå versjon 2

Her er samandraga frå dei beste resultata etter at metoden beskrive i kapittel 4.3.3.2 er brukt. Det er eit resultat frå kvar storleik. Storleiken her viser kor mange residyar dei faktisk bestemmer. Dette på grunn av at det er forskjellig grad av overlapping på dei forskjellige nettverka. Resultata frå storleik 1478, 1458 og 1438 blei tatt med vidare til versjon 3. Desse resultata valte eg fordi dei beheld dei fleste residyane som er blitt bestemt i versjon 1, samtidig som dei har ei god bestemming. Prosentmessig stig talet på korrekte residyar inntil den bestemmer bare 1438 residyar, deretter minkar det. Det var òg ein av grunnane til at eg valte vekk dei resultata som hadde mindre enn 1438 residyar, sjølv om dei hadde ei betre prosentmessig bestemming enn to av dei eg faktisk brukte. Resultatet med 1498 residyar hadde den dårligaste prosenten over korrekte residyar, av dei setta eg tok vare på.

Her vises kva namn den samanslåtte fila fekk, kor mange korrekte den bestemte, kor mange feil, totalt tal med % korrekt og til slutt kva filer den er slått saman av. Den første fila som den er slått saman av angir kor mange residyar den endelige fila vil totalt få.

Eg såg ikkje noko poeng i å ta med bestemmingar av residyar frå dei andre filene når dei ikkje fanst i første fila. Dette var sidan målet med denne versjonen var å teste om eg faktisk fekk ei betre bestemming ved å slå saman fleire resultat. Residyar som ikkje fans i den første fila ville i dei fleste tilfelle ikkje ha blitt slått saman med nokon andre resultat. Enkelte filer er slått saman av filer som igjen er slått saman av andre. Desse står oppført bare med dei filene dei totalt sett er bygd opp av.

**a2-5\_2-4:** 1109 korrekt, 389 feil, av totalt 1498: 74,03% korrekt. Består av:

- 2doble-dim-5: 1091 korrekt, 407 feil, av totalt 1498: 72,8% korrekt.
- 2doble-dim-4: 1060 korrekt, 438 feil, av totalt 1498: 70,8% korrekt.

**a3-2\_2-3:** 1121 korrekt, 357 feil, av totalt 1478: 75,84% korrekt. Består av:

- 3doble-dim-2: 1103 korrekt, 375 feil, av totalt 1478: 74,6% korrekt.
- 2doble-dim-3: 1042 korrekt, 436 feil, av totalt 1498: 69,6% korrekt.

**a4-1\_3-1:** 1119 korrekt, 339 feil, av totalt 1458: 76,74% korrekt. Består av:

- 4doble-dim-1: 1098 korrekt, 360 feil, av totalt 1458: 75,3% korrekt.
- 3doble-dim-1: 1100 korrekt, 378 feil, av totalt 1478: 74,4% korrekt.

**ba5-4\_4-3a4-1\_2-5.a5-4\_4-1a2-3\_2-1:** 1119 korrekt, 319 feil, av totalt 1438: 77,81% korrekt. Består av:

- 5doble-dim-4: 1080 korrekt, 358 feil, av totalt 1438: 75,1% korrekt.
- 4doble-dim-3: 1095 korrekt, 363 feil, av totalt 1458: 75,1% korrekt.
- 4doble-dim-1: 1098 korrekt, 360 feil, av totalt 1458: 75,3% korrekt.
- 2doble-dim-5: 1091 korrekt, 407 feil, av totalt 1498: 72,8% korrekt.
- 2doble-dim-3: 1042 korrekt, 456 feil, av totalt 1498: 69,6% korrekt.
- 2doble-dim-1: 1052 korrekt, 446 feil, av totalt 1498: 70,2% korrekt.

**a6-1\_5-4a4-1\_2-5:** 1101 korrekt, 317 feil, av totalt 1418: 77,64% korrekt. Består av:

- 6doble-dim-1: 1048 korrekt, 370 feil, av totalt 1418: 73,9% korrekt.
- 5doble-dim-4: 1080 korrekt, 358 feil, av totalt 1438: 75,1% korrekt.
- 4doble-dim-1: 1098 korrekt, 360 feil, av totalt 1458: 75,3% korrekt.
- 2doble-dim-5: 1091 korrekt, 407 feil, av totalt 1498: 72,8% korrekt.

**a7-2\_4-1a4-5\_2-1:** 1082 korrekt, 316 feil, av totalt 1398: 77,39% korrekt. Består av:

- 7doble-dim-2: 1023 korrekt, 375 feil, av totalt 1398: 73,2% korrekt.
- 4doble-dim-5: 1088 korrekt, 370 feil, av totalt 1458: 74,6% korrekt.
- 4doble-dim-1: 1098 korrekt, 360 feil, av totalt 1458: 75,3% korrekt.
- 2doble-dim-1: 1052 korrekt, 446 feil, av totalt 1498: 70,2% korrekt.

Det var her stor forskjell på resultatata alt etter kva nettverk resultatata som blei slått saman var frå. Som ein kan sjå frå resultatata over kunne bestemminga med resultatata frå 2 nettverk vere omtrent like god som bestemminga med resultatata frå 6 nettverk.

Totalt sett vil eg seie at det var lite å hente i forhold til versjon 1. Om ein samanliknar dei to beste resultatata så gjekk bestemminga frå 75,3% korrekt til 77,81% korrekt. Sjølv om det var lite å hente i talet på korrekte så var det ingenting å tape i denne versjonen. Samanslåinga av resultat førte ikkje til at eg måtte redusere talet på residyar i bestemminga. Totalt sett vil eg anbefale å ta med denne versjonen i eventuelle vidare forsøk.

### 6.3.3 Versjon 3

Desse nettverka blei trena med totalt seks forskjellige mønstersett. Det var mønstersett i tre forskjellige storleikar, og med to forskjellige typar treningssett. Tre treningssett med alle residyane, og tre sett der alle dei residyane som var blitt bestemt feil var blitt plukka vekk. Det å plukke vekk alle som var blitt bestemt feil blei gjort for å teste om nettverka bestemte betre når dei blei trena med bare resultat som i utgangspunktet var korrekte. Valideringssettet var det same i begge mønstersetta av kvar storleik. Dei forskjellige storleikane reflekterer dei forskjellige resultatata eg tok med meg vidare frå versjon 2:

- a3-3\_2-3 der valideringssettet hadde totalt 1478 residyar.
- a4-1\_3-2 der valideringssettet hadde totalt 1458 residyar.
- ba5-4\_4-3a4-1\_2-5.a5-4\_4-1a2-3\_2-1 der valideringssettet hadde totalt 1438 residyar.

Settet med bare dei som var blitt bestemt korrekt kjørte eg bare nokre få treningar på, som stikkprøvar. Resultata frå desse treningane (kap 6.3.3.3) viste at nettverka ikkje bestemte noko særlig betre på valideringssettet, men derimot bestemte veldig mykje korrekt på treningssetta. Bestemminga der låg som oftast rundt 100% korrekt. Dette indikerte at nettverka blei rimelig raskt overtrena, og er derfor ikkje brukande i vidare arbeid.

Nettverka her var forholdsvis små, men dei måtte generalisere veldig bra. Derfor laga eg dei skjulte laga forholdsmessig veldig store. Totalt sett blei nettverka framleis ganske små. Det viste seg likevel at det største av desse nettverka brukte lang tid på å trene.

### 6.3.3.1 Tabellar for netta i versjon 3

Namn	Inputlag	Skjult lag	Fil (kB)
nabo1	3x3	9x9	22
nabo2	3x5	9x25	83
nabo3	3x7	9x49	204

**Tabell 6.13:** Tabell over dei nettverka som blei brukt i versjon 3. Alle nettverka her hadde eit skjult lag som var kvadrert i forhold til input, og eit outputlag på 3x1.

Variasjonar over parameterane som blei brukt under trening av nettverka med alle residiane:

Læreparameter: 0,1; 0,25; 0,5; 0,75; 1,0

Moment: 0,25; 0,5; 0,75; 1,0

Flat spot: 0,1; 0,15; 0,2; 0,25

Nabo1 og Nabo2 blei trena med alle mønstersetta opp til 2000 syklusar, mens nabo3 blei bare trena med mønstersettet med 1438 residyar av tidsmessige grunnar. Dei beste resultatata frå desse treningane står i kap 6.3.3.2.

Variasjonar over parameter som blei brukt under treningane der bare residiane som var bestemt korrekt var med:

Læreparameter: 0,1

Moment: 0,25; 0,5

Flat spot: 0,1; 0,15

Nabo1-nettverket blei trena med mønstersettet med 1458 residyar. Nabo2-nettverket blei trena med setta som hadde 1438 og 1478 residyar. Nabo3-nettverket blei trena med settet som hadde 1438 residyar. Alle netta blei trena i 1000 syklusar. Dei beste resultatata frå desse treningane står i kap 6.3.3.3.

### 6.3.3.2 Dei endelege resultatata med alle residyar

Tabell 6.14 viser dei nettverka og verdisetta som eg var nøgd med etter å ha gått gjennom resultatata frå treningane av desse nettverka.

Nr	Namn	Lære-param.	Mo-ment	Flat spot	# sykl-usrar	# res-idyar	Tot kor	% kor.
1	Nabo1-2	0,1	0,25	0,15	100	1442	1123	77,9
2	Nabo1-3	0,1	0,25	0,2	400	1422	1111	78,1
3	Nabo1-17	0,25	0,25	0,1	900	1462	1140	78
4	Nabo2-1	0,1	0,25	0,1	300	1410	1102	78,2
5	Nabo21458-1	0,1	0,25	0,1	100	1430	1115	78
6	Nabo21478-6	0,1	0,5	0,15	1800	1450	1141	78,7
7	Nabo31438-2	0,1	0,25	0,15	400	1398	1088	77,8

**Tabell 6.14:** Første kolonne er ei nummerering av desse som eg vil bruke å referere til i tabell 6.15. Namna er bygd opp med talet på naboar, eventuelt talet på residyar i valideringssettet som mønstersetta er henta frå, og ei nummerering. Deretter følgjer læreparameter, moment og flat spot-verdiane i verdisetet. Dei siste 4 kolonnane viser kor mange syklusar nettverket blei trena i, talet på residyar i valideringssettet, kor mange residyar nettverket bestemte korrekt og til slutt kor mange % korrekt nettverket bestemte på valideringssettet.

Her følgjer det ei oversikt over resultatata eg fekk frå dei forskjellige nettverka. Denne tabellen viser resultatata fordelt på dei forskjellige SSE-typane.

Nr	$\alpha$	$\alpha$ kor	$\alpha$ tot	$\beta$	$\beta$ kor	$\beta$ tot	$\gamma$	$\gamma$ kor	$\gamma$ tot
1	470	390	440	325	206	283	647	527	719
2	466	382	430	318	190	252	638	539	740
3	474	400	443	333	200	276	655	540	743
4	463	379	422	315	201	278	632	522	710
5	467	395	450	322	217	309	641	503	671
6	471	399	446	329	203	276	650	539	728
7	458	376	421	312	165	210	628	547	767

**Tabell 6.15:** Resultata frå Nabo-nettverka. Første kolonne viser kva nettverk resultatata er frå, nummeret viser til nettverk frå tabell 6.14. I dei andre kolonnane står tala på kor mange som er blitt bestemt til å vere av kvar SSE, kor mange som blei bestemt korrekt til å vere av kvar SSE, og kor mange som skulle vore av kvar SSE.

### 6.3.3.3 Dei endelege resultatata med bare korrekt bestemte residyar

Tabell 6.16 viser dei nettverka og verdisetta som eg var nøgd med etter å ha gått gjennom resultatata frå treningane av desse nettverka.

Nr	Namn	Lære- para- meter	Mo- ment	Flat spot	# sykl.	# res.	Tot kor.	% kor.	# res. i tr.sett
1	Nabo11458-2	0,1	0,25	0,15	100	1442	1108	76,8	4610
2	Nabo21438-1	0,1	0,25	0,1	1000	1410	1099	77,9	4631
3	Nabo21478-1	0,1	0,25	0,1	600	1450	1104	76,1	4655
4	Nabo31438-3	0,1	0,5	0,1	1000	1398	1093	78,1	4597

**Tabell 6.16:** Første kolonne er ei nummerering av desse som eg vil bruke å referere til i tabell 6.17 og 6.18. Namna er bygd opp med talet på naboar, talet på residyar i valideringssettet som mønstersetta er henta frå og ei nummerering. Deretter følgjer læreparameter, moment og flat spot-verdiane i verdisetten. Dei siste 5 kolonnane viser kor mange syklusar nettverket blei trena i, talet på residyar i valideringssettet, kor mange det bestemte korrekt, kor mange % korrekt nettverket bestemte på valideringssettet. Til slutt vises kor mange residyar det var i treningssettet.

Her følgjer det ei oversikt over resultatata eg fekk frå dei forskjellige nettverka. Denne tabellen viser resultatata fordelt på dei forskjellige SSE-typane.

Nr	$\alpha$	$\alpha$ kor	$\alpha$ tot	$\beta$	$\beta$ kor	$\beta$ tot	$\gamma$	$\gamma$ kor	$\gamma$ tot
1	470	413	491	325	194	273	647	501	678
2	463	382	424	315	177	234	632	540	752
3	471	393	463	329	201	281	650	510	706
4	458	379	421	312	177	233	528	537	744

**Tabell 6.17:** Resultata frå Nabo-nettverka. Første kolonne viser kva nettverk resultatata er frå, nummeret viser til nettverk frå tabell 6.16. I dei andre kolonnane står tala på kor mange som er blitt bestemt til å vere av kvar SSE, kor mange som blei bestemt korrekt til å vere av kvar SSE, og kor mange som skulle vore av kvar SSE.

Tabell 6.18 viser resultatata frå treningssetta fordelt på SSE og totalt for alle residyane. Resultata er tatt ut etter at alle treningane er ferdige, dvs etter 1000 syklusar. Som ein kan sjå er bestemminga på treningssetta tilnærma 100% korrekt, noko som tyder på at nettverka er overtrent.

Nr	$\alpha$	$\alpha$ kor	$\alpha$ tot	$\beta$	$\beta$ kor	$\beta$ tot	$\gamma$	$\gamma$ kor	$\gamma$ tot	Tot kor
1t.s.	1876	1876	1876	838	838	838	1896	1896	1896	4610
2t.s.	1766	1765	1766	799	799	799	2066	2065	2066	4629
3t.s.	1759	1759	1759	884	884	884	2012	2012	2012	4655
4t.s.	1747	1747	1750	795	795	795	2055	2052	2052	4594

**Tabell 6.18:** Resultata etter bestemming av treningssettet. Første kolonne viser kva nettverk resultatata er frå, nummeret viser til nettverk frå tabell 6.16. I dei andre kolonnane står tala på kor mange som er blitt bestemt til å vere av kvar SSE, kor mange som blei bestemt korrekt til å vere av kvar SSE, og kor mange som skulle vore av kvar SSE. Siste kolonne viser kor mange residyar i treningssettet nettverket har bestemt korrekt.

### 6.3.3.4 Konklusjon over versjon 3

Denne versjonen var det lite å hente i. Den beste bestemminga her var på 78,7% korrekt, noko som er under 1 prosentpoeng betre enn den beste bestemminga frå versjon 2. I tillegg reduserte denne versjonen ytterlegare talet på residyar som den totale programpakken kan bestemme.

Å trene nettverka med bare korrekt bestemte residyar såg ikkje ut til å vere meir gunstig enn å trene dei med alle residyane. Dette fordi resultatata ikkje var merkbar betre, samtidig som desse nettverka tydeleg overtrena.

Ved begge desse måtane å trene nettverka på viste det seg at det var dei lågaste lære- og flat spot-verdiane som gav best trening. For momentverdiane viste det seg her at 0,5 var den beste verdien ved dei fleste treningane, i motsetnad til versjon 1 der momentverdiane varierte mykje.

### 6.3.4 Konklusjon

Denne måten å bygge opp nettverk på var mykje betre enn dei andre eg testa ut. Sjølv om kravet til generaliseringa her var ein god del høgare viste dei seg å bestemme ein del betre. Eg trur mykje av dette ligger i at desse nettverka er betre bygd opp med skjulte lag som er ein del større enn inputmengda. Dette gjorde det mogleg for nettverka å få ei god generalisering. I tillegg var kompleksiteten her ein del mindre enn dei første nettverka eg testa ut, sidan det her skulle bare bestemmast ein residy ved kvar kjøring.

Versjon 1 viste seg å gi ein veldig god bestemming allereie frå starten, men versjon 2 og 3 var litt skuffande då dei viste seg å utgjere lite for det totale resultatet.

Bestemminga fordelt på SSE viste at  $\beta$ -strukturar var dei nettverka sleit mest med, mens  $\alpha$  og  $\gamma$  klarte dei ofte å bestemme bra. Ofte blei  $\alpha$  bestemt noko betre enn  $\gamma$ , men totalt sett bestemte nettverka begge bra.



## **Kapittel 7: Konklusjonar og idear for vidare arbeid.**

Å trene opp KNN til å bestemme SSE bare med data frå avstandsmatriser, har vist seg å vere lite gunstig. Eg trur dette var fordi informasjon frå avstandsmatriser ikkje er nok informasjon til å bestemme heile strukturen. Ut frå data eg har fått frå oppgåva ser det ut til at det ikkje er nok informasjon til å kunne bestemme alle residiane. Det største problemet er med residiane som ligger i endane av kvar SSE. Her klarer ikkje programmet å bestemme noko særlig bra. Eg trur at dette kan forbeistrast om ein gir inn meir informasjon til nettverka når dei skal trene og bli brukt.

### **7.1 Vidare arbeid**

Om programsystemet skal kunne bli brukt i praksis så trur eg ein fin løysning vil vere å lage ein web-motor som kan ta imot pdb-filer, eventuelt filer formatert mykje på same måten som pdb-filene. Denne motoren skal då returnere svar med liste over residiane og bestemmingane den har gjort for kvar residy.

Slik programsystemet er nå bestemmer det ikkje residiane som ligger i dei ytre grensene av kvart protein. Ein måte å løyse dette på kan vere å prøve å legge inn nullar i staden for avstandane som skulle ha vore framanfor grenseresidyane, og sjå om nettverka då vil klare å bestemme grenseresidyane.

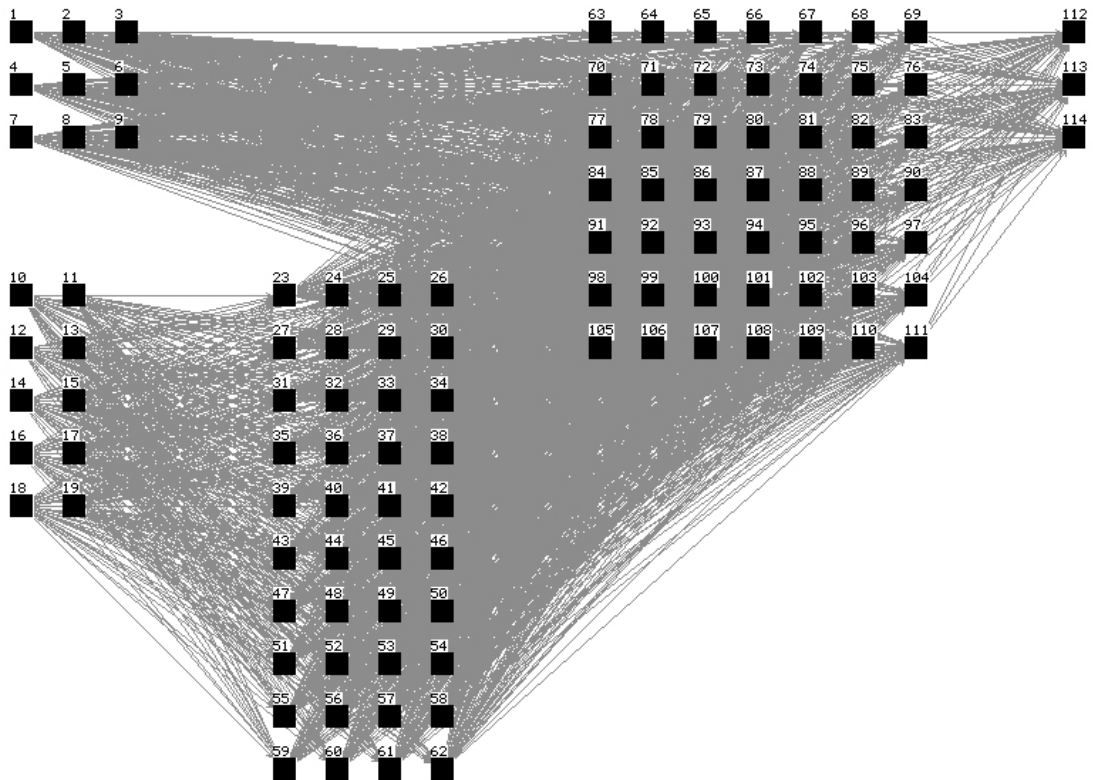
Om programsystemet skal bli brukt må ein endre litt på programma eg har laga slik at dei tar command line parameter. Ein må i tillegg endre talet på filer som programma skal behandle. Det endelege programmet skal bare ta inn ei fil. Slik fleire av programma er nå, tar dei inn både treningsfiler og valideringsfiler. Eit par av programma må i tillegg skrives om slik at dei ikkje lenger prøver å lage fasit til dei forskjellige inputdataene. Om eg hadde fått ei god bestemming så hadde heilt klart eit alternativ vore å bruke kompilatoren til SNNS og fått oversatt nettverka til C-kode, for deretter å implementerte heile programsystemet eg har laga som eit enkelt program.

Eg ville òg ha endra output til ein prosentfordeling over kva SSE dei forskjellige residiane tilhøyrer. Dette trur eg ville ha gjort resultata lettare å lese.

### **7.2 Idear for vidare forsøk**

Eg trur det ville vore gunstig å kombinere desse nettverka med informasjon om eigenskapane til dei forskjellige aminosyrene (hydrofob/hydrofil/polar/osb), og korleis det affekterer kva strukturelement dei vanlegvis blir funne i. I tillegg kan det vere aktuelt å legge inn informasjon om kva aminosyre det er, enten som eit enkelt tal til ei node eller ved hjelp av fleire nodar. I naturen blir det brukt 3 nukleotidar for å bestemme ein residy, så ein kunne t.d. gitt kvar nukleotid eit tal og så brukt 3 nodar i ei nodeklynge for å angi kva aminosyre det er. Om ein vil ta med seg litt av teorien bak min versjon 3 så kunne ein i tillegg gitt inn kva aminosyrer som ligger på kvar side av den som nå skal bestemmast.

Det kan vere gunstig å gi denne informasjonen inn utanom avstandsinformasjonen. Slik vil ein få ei bestemming utifrå avstandsinformasjonen først og deretter ei bestemming på grunnlag av resultata derifrå pluss den ekstra informasjonen. Ein måte å bygge opp eit slik nettverk på er vist i Figur 7.1.



**Figur 7.1:** Ein mulig konfigurasjon av eit nettverk med tilleggsinformasjon. Node 1-9 er for tilleggsinformasjon. Node 10-19 tilsvarer inputlaget i Type I nettverka og brukte i oppgåva. Node 23-62 tilsvarer det skjulte laget i Type I nettverka. Node 63-111 er eit nytt skjult lag som skal samanfatte bestemminga frå Type I og den ekstra informasjonen før resultatet går til output i node 112-114.

Skulle eg ha gjort forsøk med eit slikt nettverk ville eg lagt inn grenseresidyane med null. Dette for å sjå om tilleggsinformasjonen og dei få avstandane me har er tilstrekkelig til å bestemme kva SSE residyane tilhøyrer.

## Kapittel 8: Bibliografi

1. Ingvar Eidhammer, Inge Jonassen, William R. Taylor. Wiley 2004: Protein Bioinformatics: An algorithmic Approach to Sequence and Structure Analysis
2. Alvis Brazma, Helen Parkinson, Thomas Schlitt, Mohammadreza Shojatalab: A quick introduction to elements of biology - cells, molecules, genes, functional genomics, microarrays;  
[http://www.ebi.ac.uk/microarray/biology\\_intro.html](http://www.ebi.ac.uk/microarray/biology_intro.html)
3. Arthur M. Lesk. Oxford 2001: Introduction to Protein Architecture
4. Carl Branden & John Tooze. Garland 1999: Introduction to Protein Structure - 2. ed.
5. Berman m.fl. 2002 Protein Data Bank *Acta Cryst* **D58**, 899-907
6. Kabsch W. og Sander C. 1983 Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers* **22**, 2577-2637.
7. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. MIT Press 2001: Introduction to Algorithms, second edition.
8. Terje Kristensen. Cappelen 1997: Nevrale Nettverk, fuzzy logic og genetiske algoritmer.
9. SNNS User Manual v.4.2 frå University of Stuttgart og University of Tübingen
10. D. E. Rumelhart og J. L. McClelland MIT Press 1986: "Parallell Distributing Processing"



## Appendix: Innhald på cdrom'en

Alle katalogane som inneheld program, inneheld kjeldekoden til programmet samt ein katalog med nokre av testfilene eg har brukt.

- **Dei\_store\_netverka:** Denne katalogen inneheld komprimerte nettverksfiler for dei første topologiane eg testa ut. Ref. kapittel 4.3.1 og 4.3.2.
- **Versjon1:**
  - **Analyser:** Analyseprogrammet brukt i versjon 1. Kapittel 5.3.
  - **Batmanfiler:** Døme på Batchfiler eg brukte når eg kjørte treningar i SNNS.
  - **Mønsterfiler:** Døme på mønsterfiler eg brukte når eg kjørte treningar i SNNS.
  - **Nettverksfiler:** Nettverksfilene som blei brukt i versjon 1.
  - **Tygg:** Programmet som laga mønsterfilene. Kapittel 5.1.
- **Versjon2:**
  - **Analyser:** Den utgåva av analyseprogrammet som blei brukt i versjon 2. Kapittel 5.3.2.
  - **Del2:** Del2-programmet. Kapittel 5.5.
  - **Input:** Programmet som laga input til Del2-programmet. Kapittel 5.4.
  - **SortSammendrag:** Programmet som analyserte data frå Del2. Kapittel 5.6.
- **Versjon3**
  - **Bare\_korrekt\_bestemte\_residyar:** Filer som blei brukt i kjøringane som hadde bare dei korrekt bestemte residiane frå versjon 2. Kapittel 6.3.3.
    - **Batmanfiler:** Batchfilene eg brukte når eg kjørte treningar i SNNS.
    - **Mønsterfiler:** Mønsterfilene med bare korrekt bestemte residyar.
  - **Med\_alle\_residyar:** Filer som blei brukt i kjøringar av versjon 3. Kapittel 6.3.3.
    - **Batmanfiler:** Batchfilene eg brukte når eg kjørte treningar i SNNS.
    - **Mønsterfiler:** Mønsterfilene med alle residiane.
  - **Nettverksfiler:** Nettverksfilene som blei brukt i versjon 3.