Prosjekt 2 - FYS-STK4155

Jonas Asperud og Magnus Børsting

12. november 2018

Lineære, Ridge og Lasso regresjon blir brukt på tilstandene for den en dimensjonale Ising modellen med tilsvarende energier for å finne den lineære modellen som best finner koblingskonstantene mellom tilstandene. Neste del blir logistisk regresjon med SGD brukt til å klassifisere den to dimensjonale Ising modellen, med resultat at den ikke klarte å klassifisere tilstandene. Deretter blir nevralt nettverk brukt til å tilpasse den en dimensjonale Ising modellen. Til slutt blir nevralt nettverk brukt til å klassifisere om de to dimensjonale Ising tilstandene er ordnet eller uordnet. Den beste modellen for de kritiske tilstandene har 97 prosent nøyaktighet med læringsfrekvens 0.0001 og regulariseringen var på 1.0.

I. Introduksjon

Ising modellen gir mange mulige utforskninger av hva slags påvirkning partikler med forskjellig spinn har på hverandre og hele systemet av partikler. Det blir først brukt lineær, Ridge og Lasso regresjon for å finne koblingskonstantene mellom 40 tilstander og hvor mye de påvirker systemets energi. Denne typen regresjon forutsetter at forholde mellom observerte verdier og tilstander er lineære, så det systemet som blir analysert kan ikke være for komplisert. Men siden de er relativt mindre kompliserte, har de mye mindre sjanse for overfitting. Med effekt at de er bedre til å lage modeller med god prediksjon.

Deretter blir det brukt logistisk klassifisering på den to dimensjonale Ising modellen. Dette er en modell som får fase skifte mellom ordnet og uordnet tilstand ved en hvis temperatur. Så hvis modeller kan klare å klassifisere mellom de to tilstandene, kan den kritiske temperaturen bli funnet. Logistisk regresjon er en lineær metode av klassifisering, altså området mellom ordnete

og uordnete tilstand må være lineært. For å løse logistisk regresjon ble gradient descent brukt, en metode for å minimere kost funksjonen.

I neste del går det fra lineære modeller til nevralt nettverk, en modell som kan finne flere ikke lineære forhold. Denne modellen blir først brukt til å tilpasse den en dimensjonale Ising modellen. Så til slutt blir nevralt nettverk brukt til å forutsi fasen til den to dimensjonale Ising modellen. Selve koden for oppgave b og d kan bli funnet i Jonas sin github [1]. Mens oppgave c og e kan bli funnet i Magnus sin github [2].

II. Teori

Det fokuseres ikke på Ordinary least squares, Lasso- eller Ridge-regresjon. Heller ikke MSE eller R2 score. Dette ble gjort i prosjekt 1. For mer informasjon les W.N. van Wieringen [10] og prosjektoppgaven vår [9].

A. Ising modellen

Ising modellen er en matematisk modell av ferromagnetisme. Det er et binært system, hvor variablene består av de magnetiske dipol momentene av atomisk spinn. De beskrives av enten spinn opp +1 eller spinn ned -1. Spinn variablene er plassert i et gitter og kan vekselvirke med naboene. Styrken på vekselvirkningen er beskrevet av en koblingskonstant, **J** [7]. Energien i systemet er beskrevet som [7]

$$E = -j \sum_{<jk>}^{N} s_j s_k \tag{1}$$

$s_j \pm 1$, N er antall spinn vektorer og S er koblings konstanten. Summen over $< jk >$ er kun summen over nærmeste naboer.

For 1D tilfellet defineres s^i som er et sett av spinntilstander, hvor $i = 1, 2, \ldots, n$ antall sett. Det kan f.eks. være $s^1 = [1, -1]$. Og hvor **J** er en kvadratisk matrise. Energien i sytemet er da beskrevet av [7]

$$E[s^i] = -\sum_{j}^{N} \sum_{k}^{N} J_{j,k} s_j^i s_k^i \tag{2}$$

Hvis matrisen $\mathbf{J} = \begin{vmatrix} 0 & -1 \\ -1 & 0 \end{vmatrix}$ har vi at energien $E[s^i] = 2$ for $s^i = [1, -1]$. Alle vekselvirknin-

gene mellom nærmeste nabo kan skrives som en matrise $\mathbf{X}^i$ som består av settet $\{s_j^i s_k^i\}_{k,j=1}^N$. Modellen som brukes i eksperimentet er da [7]

$$E^i = \mathbf{X}^i \mathbf{J} \tag{3}$$

Det endimensjonale tilfellet har ingen faseovergang, men i et todimensjonalt gitter kan et ferroelektrisk materiale med alle spinnvektorer i samme retning bli til en uordnet fase ved den kritiske temperaturen $\frac{T_c}{J} \approx 2.26$ [7].

B. Gradient Descent

Gradient Descent (GD) er en algoritme for minimering av en kostfunksjon. Dette oppnår den ved å starte i et tilfeldig punkt. Så tar den gradienten i punktet den står, så går den i retningen med mest nedgang. Steget den tar er en funksjon av læringsfrekvens. Dette er en variabel som må bli vurdert i henhold til problemet som skal bli minimert, med hensikt at GD ikke sitter seg fast på et platå eller bruker for lang tid siden steget er for langt eller for kort. En måte å forbedre GD på er å ikke bruke hele trening settet, men en tilfeldig undergruppe. Denne versjonen av GD heter Stokastisk GD (SGD). Siden gradienten for et gitt punkt vil være annerledes, klarer SGD bedre å komme seg unna lokale minima og platåer. Men dette betyr også at den vil gå vekk fra det globale minima, altså vil det miste noe av nøyaktigheten. Hele trenings sett blir heller ikke brukt, så det vil være lettere å behandle store data sett.

En annen populær måte å forbedre GD på er å bruke den partiell deriverte med hensyn på alle dimensjonene i problemet istedenfor gradienten. Så når retningen som går oppover er funnet, går Batch SD motsatt vei. Når Batch SD og SGD blir kombinert heter det Mini-batch GD. Dette blir gjort ved at gradienten for hvert steg blir regnet ut fra flere små tilfeldige sett. Det at tilfeldighet blir lagt vil gjøre at det ikke blir for mye tilpasning til modellen. Mens det at det er en undergruppe som blir brukt, øker hastigheten til algoritmen.

C. Logistisk Regresjon

Lineær regresjon finner hvor mye modellen skal forandre seg gitt en forandring i en eller flere variabler. Dette er en modell som gir svar over et intervall. Men hvis modellen man bruker skal forutsi to forskjellige konsekvenser, som problemet fra bioinformatikk om en gitt forandring i hjerterytmen betyr at man får hjerteinfarkt. Regresjons modellen som løser denne typen problemer er logistisk regresjon. For å oppnå dette, er alltid summen av kategoriene til

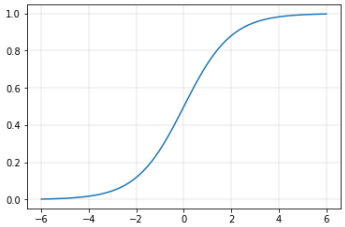
modellen lik 1. For å oppnå dette er det den logistiske funksjonen

$$\sigma(t) = \frac{1}{1 + \exp(-t)} \tag{4}$$

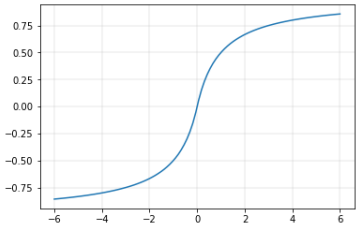
som estimerer modellens sannsynlighet. Dette er en funksjon som alltid er mellom null og en.

D. Aktiverings-funksjoner

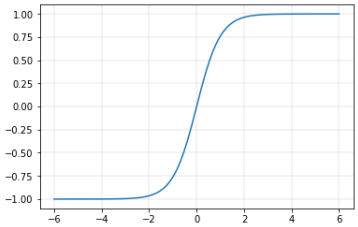
I nevrale nettverk brukes det **Aktiverings-funksjoner** til å skalere nodene, dette er forklart i mer detalj i teorien under i nevralt nettverk. Det finnes mange forskjellige aktiverings-funksjoner, med forskjellige egenskaper. Sigmoid funksjonen er illustrert i figur 1a, denne har en myk aktivering, men gir kun positive output. To andre som ble brukt i dette eksperimentet er tanh og softsign, illustrert i figur 1c og figur 1b. Disse ligner på hverandre, men tanh er brattere enn softsign rundt 0 og gir derfor en bråere overgang. Begge disse funksjonene kan gi negative output. Funksjonene og flere andre, med deres deriverte er listet i [11].



(a) Sigmoid-funksjonen.



(b) Softsign-funksjonen



(c) Tanh-funksjonen.

Når det forsøkes å tilpasse en måling eller et resultat ønsker man å minimere **cost funksjonen**. Denne funksjonen forteller hvor stor feil man har gjort i forhold til målingen. Dette gjøres også i vanlig regresjon. En cost funksjon som brukes i denne oppgaven er f.eks.

$$C = \frac{1}{2} \sum_j (y_j - a_j^L)^2 \tag{5}$$

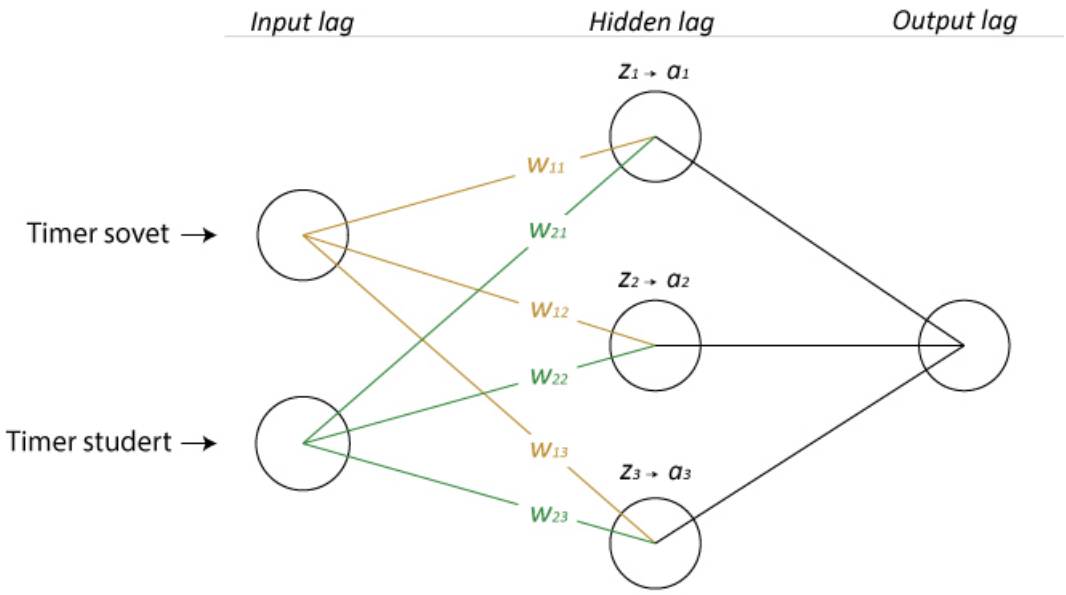
Hvor y er målingene som tilpasses i rad j og a er aktiverings-funksjonen i rad j i output laget nummerert L.

E. Nevralt nettverk

Nevralt nettverk fungerer slik at nevroner eller noder sender et signal mellom lag av noder. Det første laget kalles et input lag, illustrert til venstre i figur 2. Dette laget har like mange noder

som antall input. Hvis man ser på en modell hvor antall timer man sover er 5, antall timer man studerer er 7 og med dette fikk en person en prosent karakter på 83% [6]. Prøver vi å estimere karakteren gitt antall timer sovet og antall timer 7 er antall noder i input laget 2.

Til høyre i figuren er output laget. Det er kun 1 node i output laget i vårt tilfelle, siden vi kun vil estimere 83% med metoden. Mellom output og input laget er det ett eller flere hidden lag. Hvert hidden lag kan ha ønsket antall noder, trenger ikke være de samme som noen av de andre lagene, og det er opp til brukeren å definere antall lag. I dette eksempelet er det et hidden lag med 3 noder. Hver node er forbundet med en vekt w. Input fra hver node multipliseres med hver sin vekt w. Summen av begge multiplikasjonene summeres og kalles aktiviteten, z. Aktiviteten sendes deretter gjennom en aktiverings-funksjon som skalerer aktiviteten og symboliseres ved a.



Figur 2. Viser basisen på hvordan et nevralt nettverk fungerer. Alle nodene blir multiplisert med en vekt, w, videre til neste lag. Summen av dette, z, sendes til en aktiveringsfunksjon , a, se tekst for nærmere detaljer.

Det tilsettes også en liten bias-verdi i hver node slik at hvis vektene er null gir ikke det kontinuerlig null verdier tilbake. Metoden kan beskrives når alle vekter, aktiveringer og aktiveringsfunksjoner beskrives av matriser, matrisene er definert nærmere i [5]. For et hvert trenings input x definer den første aktiveringsfunksjonen som $a^{x,1} = x$. Hvor l er laget, i den første aktiveringsfunksjonene er $l = 1$. Først utregnes alle z-verdier og deretter a-verdier **suksessivt forover**.

For $l = 2, 3, \ldots, L$, L er output laget [8]

$$z^{x,l} = w^l a^{x,l-1} + b^l \quad for \quad a^{x,l} = f z^{x,l} \tag{6}$$

f er en vilkårlig aktiveringsfunksjon. Deretter regnes feilen som gjøres ut i output laget [8]

$$\delta^{x,l} = \nabla_a C_x \odot f'(z^{x,L}) \tag{7}$$

Hvor f' er aktiveringsfunksjonen derivert, $\odot$ er Hadamard produktet [3] og $\nabla_a C_x$ er gradienten til cost-funksjonen i output noden. Dette kan brukes til å gjøre backpropagation av feilen man gjør [3]. For hver $l = L - 1, L - 2, \ldots, 2$ beregn [8]

$$\delta^l = (w^{l+1})^T \delta^{l+1} \odot f'(z^l) \tag{8}$$

For at det nevrale nettverket skal lære kan man bruke feilen man har funnet til å oppdatere vektene. En måte er å anvende Gradient decent. Dette er en metode hvor gradienten til cost funksjonen, C, brukes til å bevege seg mot et mindre avvik. Denne metoden beskrives som [8]

$$\frac{dC}{dw^x_{jk}} = a^{l-1}_k \delta^l_j \quad og \quad \frac{dC}{db^l_j} = \delta^l_j \tag{9}$$

Når j er raden og k er kolonnen. Gradientene brukes til å oppdatere vektene hvor η er læringsraten [4], som beskriver hvor mye vekten skal bevege seg i retningen til gradienten. De nye vektene er da [8]

$$w^l = w^l - \eta \sum_x \delta^{x,l} (a^{x,l-1})^T \tag{10}$$

og de nye biasene er [8]

$$b^l = b^l - \eta \sum_x \delta^{x,l} \tag{11}$$

I tillegg blir det også ofte brukt **batch gradient decent**. Istedenfor å bruke hele datasettet kalkuleres gradienten i et subset kalt minibatch. Med et data sett med N punkter og M minibatcher blir antall batcher $\frac{M}{N}$. Gradienten blir da [8] for $B_k = 1, 2, 3..\frac{M}{N}$

$$\nabla C(w) = \frac{1}{M} \sum_{i \in B_k} \nabla C_{jk}(w) \tag{12}$$

Siden denne gradienten legger et stokastisk element til, så er sjansen for minima mye lavere. Samtidig kan dette øke hastigheten på utregningene.

Det blir også ofte lagt til en **regulerings parameter**, λ til cost funksjonen. Denne parameteren gjør det slik at vektene ikke kan spinne ut av kontroll, akkurat som for ridge regresjon. Denne beskrives som [5]

$$\nabla C(w) = \frac{1}{N}\sum_{j=1}^{N} \nabla C_{jk}(w) + \lambda ||w||_2^2 \tag{13}$$

III. Applikasjon

A. OLS, Ridge og Lasso

Ising modellen i en dimensjon anvendes, hvor det blir produsert en kjede av spinntilstander med en størrelse på $L = 40$. Deretter dannes det 1e4 forskjellige tilfeldig valgte sett av spinntilstander. Disse settene blir brukt til å regne ut energien til systemene, se ligning 3, hvor koblingskonstanten, $\mathbf{J}$, er en LxL matrise med -1 langs en diagonal som starter i $\mathbf{J}_{12}$. I første kolonne er $J_{L1} = -1$ mens resten av matrisen er 0. $\mathbf{X}^i$ matrisen er interaksjonen mellom naboene av de tilfeldige spinntilstandene. Det anvendes deretter OLS, Ridge og Lasso regresjon med grad 1 for å finne koblingskonstanten. Det gjøres også en bootstrap av de tre forskjellige metodene. Bootstrap MSE og R2 score anvendt på test- og trenings-settet bruktes til å analysere modellen. Antall bootsraps er 10 og det gjøres også en bias-variance dekomposisjon. All iterasjon foregår med en lambda fra 10^{-5} til 10^5 i 10 steg, hvor trening-/test-splitting av dataen var 2/3 trening og 1/3 testing. OLS, Ridge og Lasso ble testet mot scikit-learn sine funksjoner og reproduserte resultatene hver gang. Det ble også gjort en visuell sammen ligning med Metha et al. [7] Som stemmer godt overens.

B. Logistisk Regresjon

Det ble tatt utgangspunkt i koden for Stochastic gradient descent og Batch gradient descent fra boken til Heron [3] for å løse logistisk regresjon. Med de to metodene som mal, ble Stochastic gradient descent med mini batches lagd. Stochastic gradient descent med mini batches ble testet mot de to andre metodene, for å sjekke at svaret var i nærheten. Så ble det lagd to logistisk regresjons funksjoner. Den ene brukte Stochastic gradient descent, den andre Stochastic gradient descent med mini batches. Funksjonene startet med å ta ut 70 prosent til trening og 30 prosent til test settet. Deretter ble dimensjonene til testsettet tatt. De dimensjonene

ble brukt til å finne hvor mange mini batches det blir når de hadde størrelse på 32 og hvor mange vekter det er i modellen.

Hvor mange ganger datasettet ble gått over ble satt av inngangsparameteret n_epoch når funksjonen ble kalt. For SGD ble den andre løkken satt av hvor mange tilstander det var i trenging settet. SGD med mini batches gikk over så mange mini batches at det tilsvarte lengden til settet det gikk over, hvor det ble rundet opp når det ikke gikk opp. Deretter ble det gjort n_epoch ganger. Gradienten ble så forandret til å løse logistisk regressjon. Det to dimensjonale Ising settet ble så gått over med SGD, både uten og med mini batches. Det ble brukt elleve læringsfrekvenser i potens av ti fra minus ti til null. Det ble brukt 50 epochs, så ble nøyaktigheten til modellen målt mot trenings sett, test settet og på settet med tilstander i kritisk fase. Det til slutt testet om det ble en forskjell om antall epocher ble forandret til 25 eller 300, med resultat at det ble ikke observert en forandring.

C. Nevrale nettverk ising modell regresjon

Input dataen er den samme som i eksperimentet med regresjon. Dataen er en 600 x 1600 matrise som blir fordelt i et trenings-sett på 2/3 av størrelsen og test-sett på 1/3. Trenings-settet er da en 400 x 1600 matrise som brukes til å trene modellen. Til å evaluere modellen brukes et test-settet som er en 200 x 1600 matrise. Det nevrale nettverket pga. Aktiverings-funksjonene gir ikke output større enn 1 og mindre enn -1. Derfor normaliseres input før det brukes til beregningene.

I dette eksperimentet brukes nevrale nettverk med cost funksjonen i ligning 5, men med en regulerings parameter gitt ved ligning 13. Dette programmeres ved ligningene (6-11) i samme rekkefølge. Det brukes også batch gradient decent gitt ved ligning 12. Softsign-funksjonen blir brukt som aktiveringsfunksjon. Først plottes alle kombinasjoner av $\eta = 10^{-5} - 10^{-1}$ og $\lambda = 10^{-3} - 10^{1}$, med en batch størrelse på 50, epoke størrelse på 10^3, dvs. Antall ganger man går igjennom det fulle trenings-settet, og med 3 hidden lag med 50, 75 og 30 noder. I denne rekkefølgen. Mean squared error, MSE, anvendes på test settet og plottes.

Det avgjøres så manuelt ut ifra den minimale MSE og fra MSE plottet hvilken 3 kombinasjoner som gir best resultat. Disse tre brukes til å finne MSE til alle kombinasjoner av antall hidden lag, hvor hidden lag størrelsene er 50, 75, 30, 85 og 110. I denne rekkefølgen. Den beste kombinasjonen brukes så til å plotte trenings-sett MSE og test-sett MSE med epoke størrelsen på 10^4 alt gjort med batch størrelse på 50. Det testes så med de forskjellige aktiveringsfunksjonene nevnt i teoridelen.

De fulle test kodene finnes på github [1]. De er ganske maskinkrevende og kan ta litt tid. Koden ble testet mot regresjons resultatene for å se om det ga et realistisk resultat.

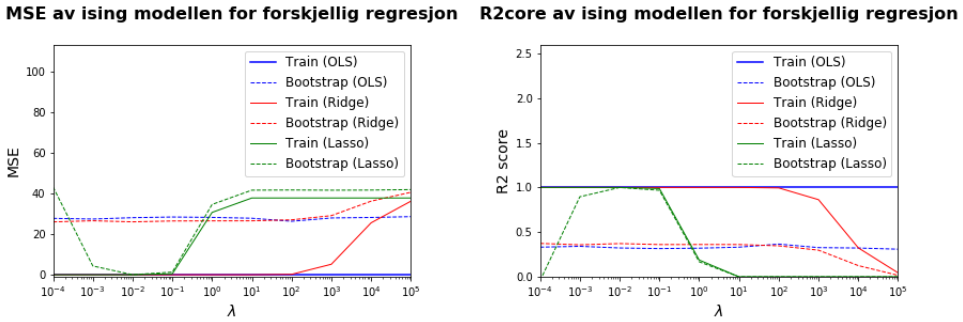
D. Nevralt Nettverk For To Dimensjonalt Ising

Denne delen baserte seg på forelesningsnotatene, altså et grid search med regularisering og læringsfrekvens i tier potenser fra minus fem til en på Python klassen NeuralNetwork. Det ble tatt ut et trenings sett fra 80 prosent av de uordnete og ordnete tilstandene, de resterende 20 prosent ble brukt som test sett. For hver av de parametrene ble nøyaktigheten målt mot trening settet, test settet og de kritiske fasene. Det ble brukt 20 Nevroner over 25 epocher med batch størrelse på 32.

IV. Resultater

A. OLS, Ridge og Lasso

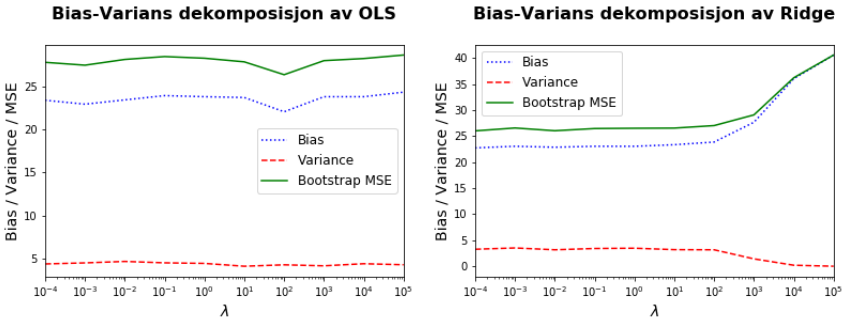
Den 1 dimensjonale Ising modellen ble anvendt, hvor tilpasningen av beta verdiene for de forskjellige ble beregnet. Prediksjonen av modellen ble evaluert med MSE og R2-score er plottet i figur (3a,3b). Disse er plottet med den direkte estimeringen av trening/test settet og via bootstrapping.



(a) MSE av OLS, Ridge og Lasso. Det er anvendt MSE på treningssettet direkte og via bootstrapping.

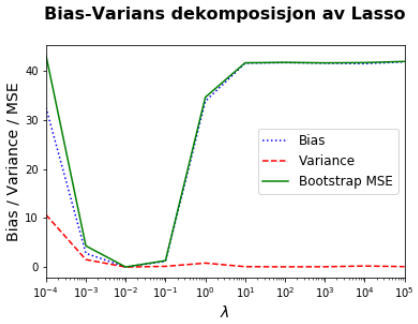
(b) MSE av OLS, Ridge og Lasso. Det er anvendt MSE på treningssettet direkte og via bootstrapping.

Bias-varaians dekomposisjon ble også plottet for de forskjellige regresjonsmetodene.



(a) Bias-varians dekomposisjon av OLS.

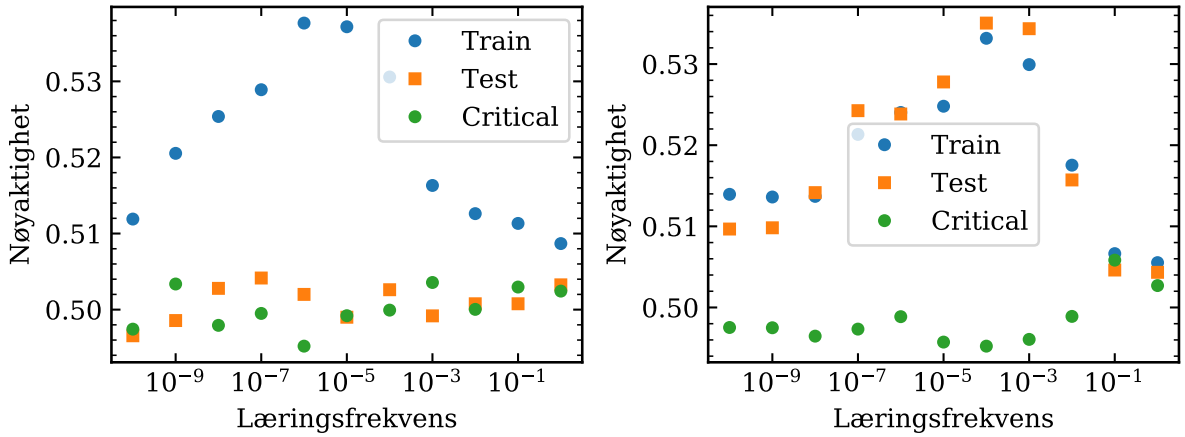
(b) Bias-varians dekomposisjon av Ridge regresjon..



(c) Bias-varians dekomposisjon av Lasso regresjon.

B. Logistisk Regresjon

Fra Figur 5 kan man se at logistisk regresjon med SGD hadde en nøyaktighet for den kritiske fasen rundt 50 prosent, altså modellen klarte ikke å tilpasse seg. De beste modellene for trenings settet hadde en læringsfrekvens fra 10^{-5} til 10^{-3}. For vanlig SGD, ga forbedringen på fire prosent fra ren tilfeldig gjetting ingen forbedring i test settet. Men SGD med mini batches så en tilsvarende nøyaktighet for båre trenings settet og test settet.



(a) Stochastic gradient descent.

(b) Stochastic gradient descent med mini batches.

Figur 5. Logistisk regresjon på 2D Ising modell som funksjon av læringsfrekvensen. Gradienten ble funnet med Stochastic gradient descent med og uten mini batches.

C. Nevrale nettverk ising modell regresjon

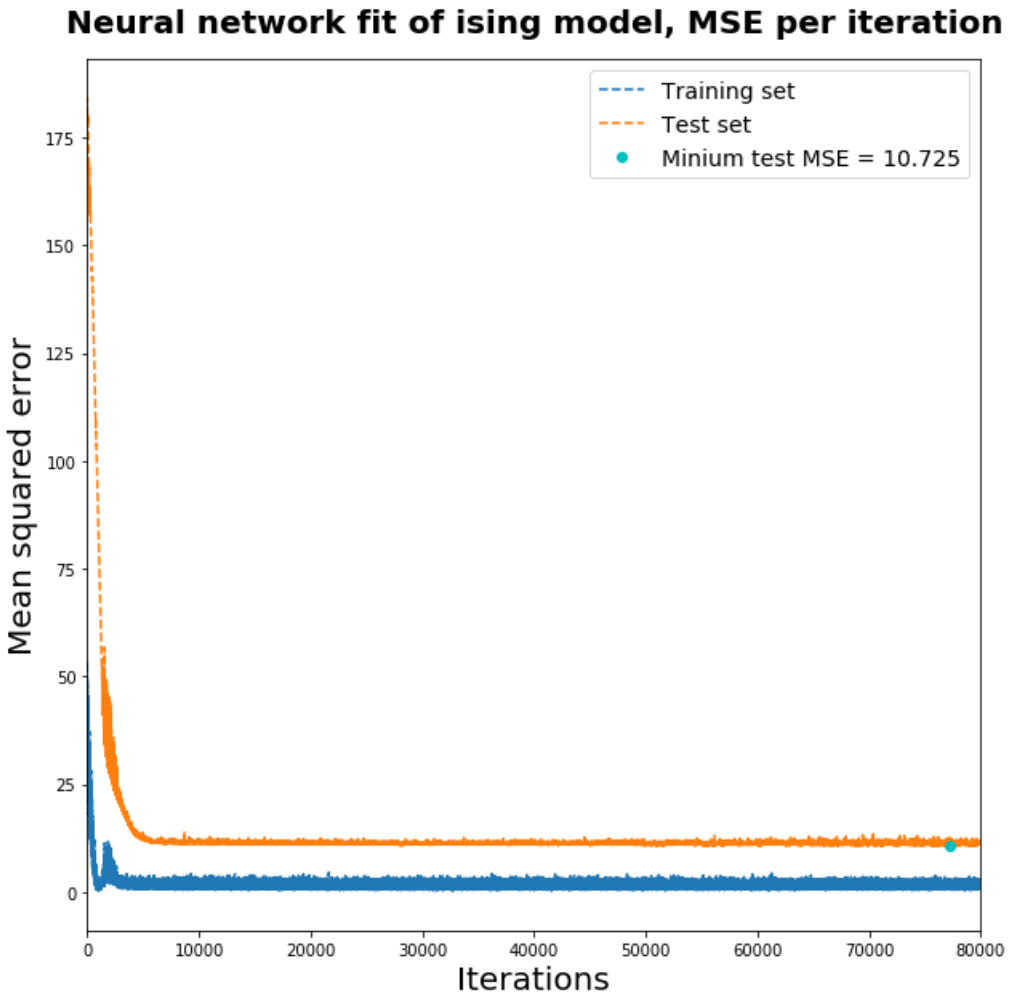
I dette eksperimentet ble det brukt de samme input av 1D ising modellen som for regresjons eksperimentet. Det beregnes først MSE for alle mulige kombinasjoner av $\eta = 10^{-5} - 10^{-1}$ og $\lambda = 10^{-3} - 10^{1}$. Test MSE blir plottet, hvor epoke størrelsen er 10^3, batch størrelse på 50, og hidden lagene er 50, 75, 30 noder. Softsign-funksjonen brukes som aktiveringsfunksjon. Plottet er veldig stort og det må zoomes inn for å få en god oversikt, plottet ligger på github [1] med navn *Eta_lambda_testing.png*. Y-aksen venstre er mean squared error av test-settet, x-aksen er antall iterasjoner som er blitt gjort og tegnforklaringen er den minste MSE til test settet av alle iterasjonene.

De beste kombinasjonene blir vurdert til å være rad 3 kolonne 4 $\eta = 10^{-3}; \lambda = 10^{0}$, rad 4 kolonne 3 $\eta = 10^{-2}; \lambda = 10^{-1}$ og rad 5 kolonne 2 $\eta = 10^{-1}; \lambda = 10^{-2}$. Disse hadde de laveste MSE verdiene og var relativt stabile, altså ikke fullstendig støy.

Kombinasjonene blir videre brukt til å beregne MSE som funksjon av antall hidden lag, fra 1-5 lag med antall noder 50, 75, 30, 85 og 110. Dette ble plottet, men dette plottet er også ganske stort og ligger på github [1] med navn *Layer_size_testing.png*. I plottet er de forskjellige kombinasjonene plottet på i hver kolonne, hvor det legges til et ekstra hidden lag for hver rad. Y-aksen venstre er mean squared error av test-settet, x-aksen er antall iterasjoner som er blitt gjort og tegnforklaringen er den minste MSE til test settet av alle iterasjonene. Den beste kombinasjonen blir vurdert til å være rad 4 kolonne 1, $\eta = 10^{-3}; \lambda = 10^{0}$, antall hidden

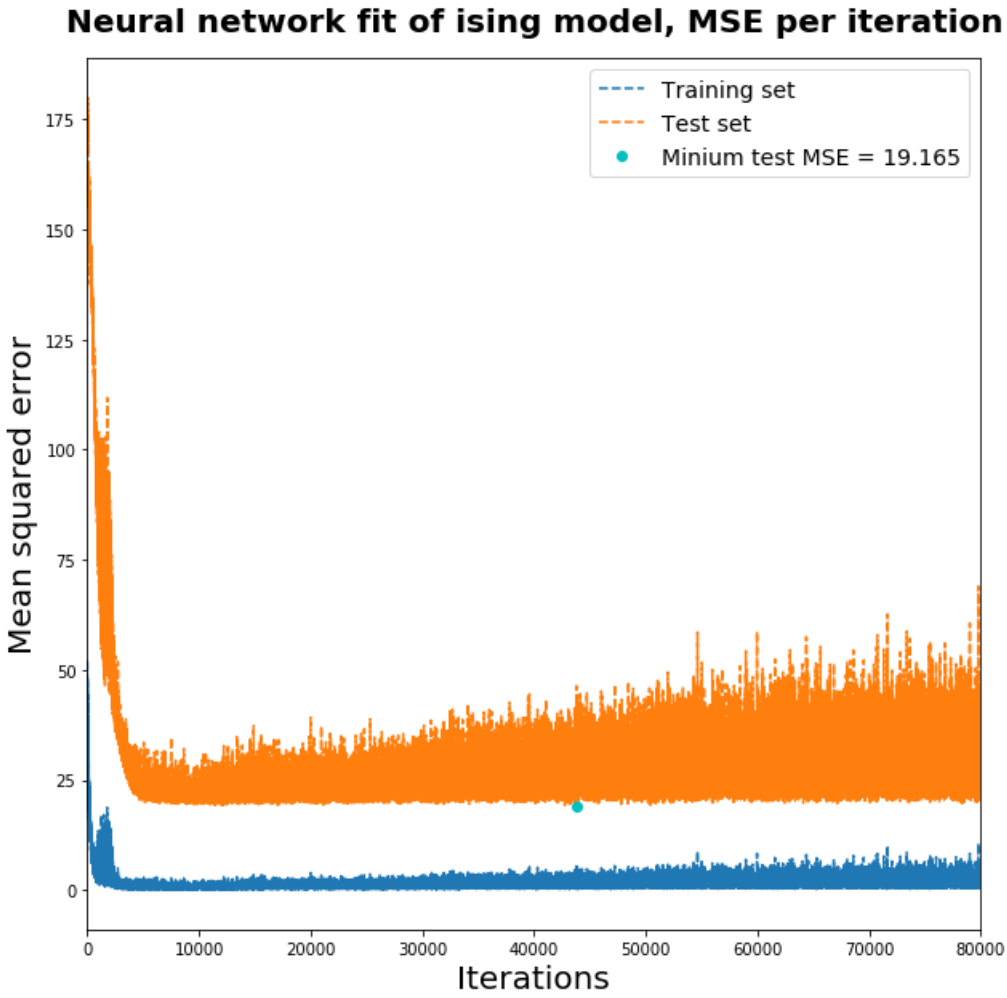
lag er 4 med antall noder fra 1-4 er 50, 75, 30, 85.Dette var ikke resultatet med lavest MSE, men det var blant de laveste og det var litt støy men ikke for mye.

Dette resultatet ble brukt til å plotte MSE til test- og trenings-settet med epoke størrelse på 10^4, vist i figur 6. Det samme ble beregnet med epoke størrelse på 10^5, hvor det viser seg at MSE støyen øker rundt 10^4 iterasjoner. Plottet ligger på github [1] med navn $Best\_hyperparameter.png$.



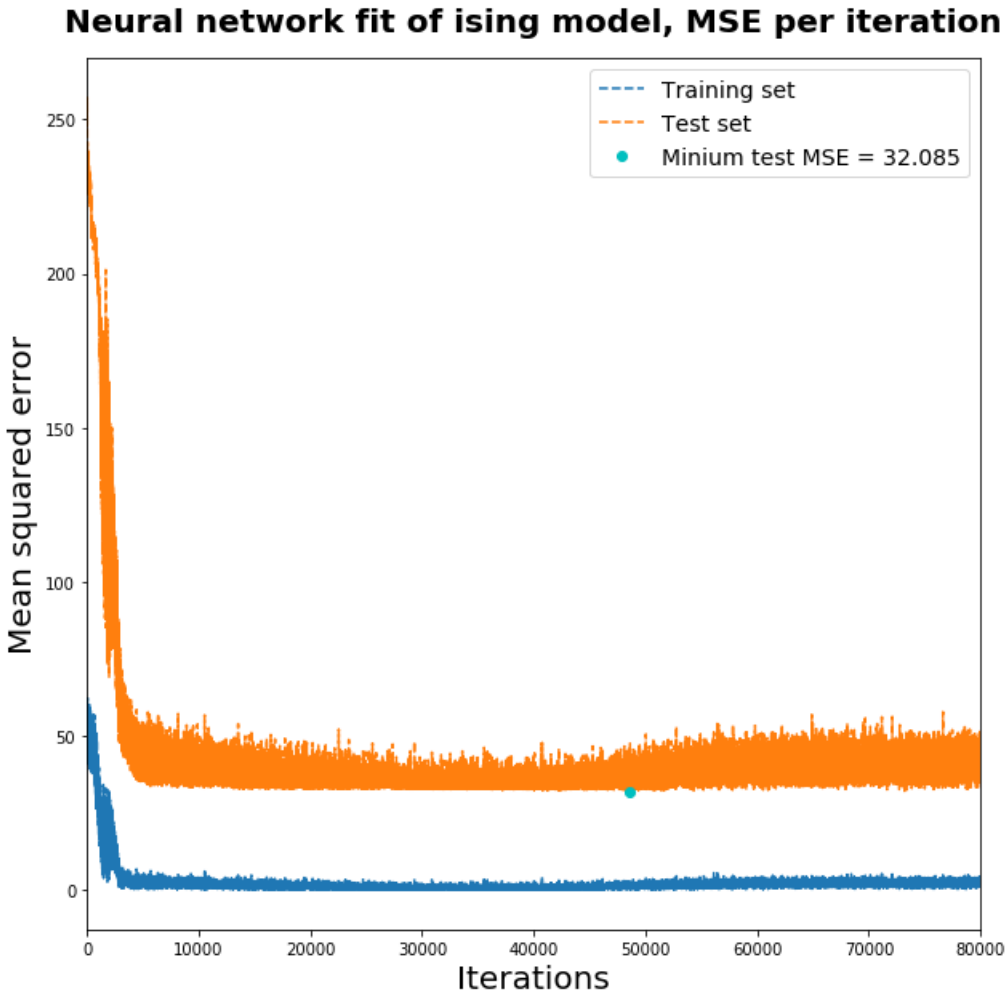
Figur 6. viser MSE til trening- og test-settet med de beste hyperparameterene funnet fra tidligere testingen. Se tekst for detaljer. Aktiveringsfunksjonen brukt er softsign-funksjonen.

Det beregnes også MSE med kombinasjonen fra rad 4 kolonne 2 fra github [1] med navn $Layer\_size\_testing.png$. $\eta = 10^{-2}; \lambda = 10^{-1}$, med samme antall hidden lag. Denne har mer MSE støy enn de beste parameteren. Plottet er vist i figur 7.



Figur 7. viser MSE til trening- og test-settet med $\eta = 10^{-2}; \lambda = 10^{-1}$ funnet fra tidligere testingen. Se tekst for detaljer. Aktiveringsfunksjonen brukt er softsign-funksjonen

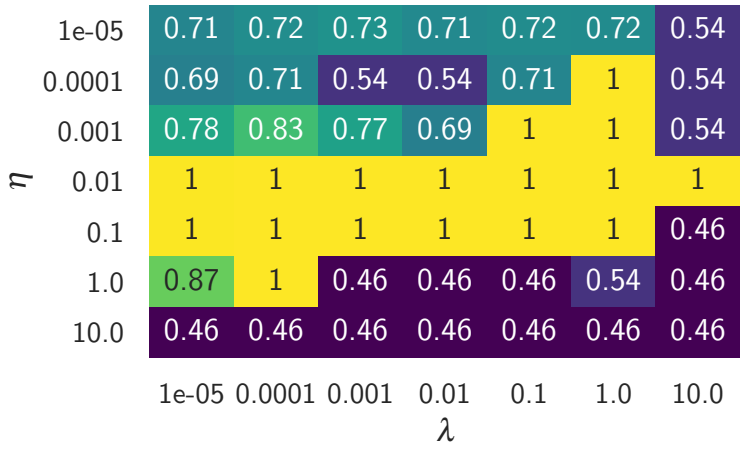
De beste parameterene brukes til å beregne det samme, men med tanh-funksjonen som aktiveringsfunksjon. Vist i figur 8.



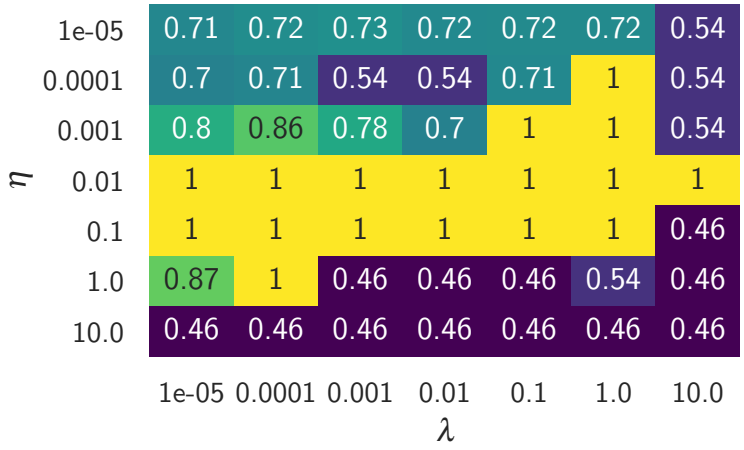
Figur 8. viser MSE til trening- og test-settet med $\eta = 10^{-3}; \lambda = 10^{0}$ funnet fra tidligere testing. Se tekst for detaljer. Aktiveringsfunksjonen brukt er softsign-funksjonen

D. Nevralt Nettverk For To Dimensjonalt Ising

Fra Figur 9 klarte modellen å tilpasse seg til trenings tilstandene med 100 prosent nøyaktighet for 17 av de 49 mulige kombinasjonen. For modellene med 100 prosent nøyaktighet hadde 13 av 17 en læringsfrekvens på enten 0.1 eller 0.01. Nøyaktigheten er gyldig til vist oppløsning. Det var bare to modeller som hadde 100 prosent nøyaktighet for hele settet. Det var med læringsfrekvens 0.1 og regulariseringskonstanten på 0.001. Den andre modellen hadde læringsfrekvens på 0.01 og regulariseringskonstanten 0.01. Det samme resultatet kan bli sett i Figur 10, altså mot test settet.

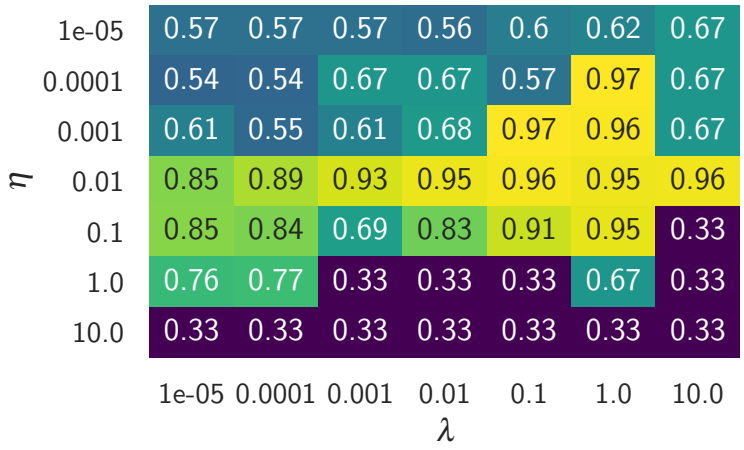


Figur 9. Nevralt nettverk modellens nøyaktighet på to dimensjonalt Ising test settet for parametrene læringsfrekvens (η) og regulariseringskonstant (λ).



Figur 10. Prediksjonsfeilen til det nevrale nettverket som funksjon av læringsfrekvens (η) og regulariseringskonstant (λ).

Hva nøyaktigheten ble for modellen mot de kritiske tilstandene kan bli sett i Figur 11. Her var det for ingen av parameterne at nøyaktigheten ble 100 prosent. Men området med de beste parameterne for trening/test sett gir også her de beste verdiene. Den beste nøyaktigheten er på 97 prosent, med de to samme verdiene for læringsfrekvens og regulariseringskonstanter som trening/test settet.



Figur 11. De nevrale nettverksmodellene nøyaktighet mot kritiske faser, trent mot de uordnete og ordnete fasene med forskjellige læringsfrekvenser (η) og regulariseringskonstanter (λ).

V. Diskusjon

A. OLS, Ridge og Lasso

Det som observeres med OLS er at den er relativt jevnt med økende lambda verdi. Dette bør den være, siden den i utgangspunktet er uavhengig av den. Ridge er avhengig av lambda verdien, hvor straffeparameteren er en L_2 norm. Dette setter så godt som alle beta verdiene til noe som ikke er null. Vi ser dermed at Ridge er relativt stabil frem til lambda verdien blir veldig stor. Ved en stor lambda verdi blir straffeparameteren så stor at den setter alle regressorene til null. Lasso derimot har den egenskapen at straffeparameteren er en L_1 norm og som konsekvens av dette kan noen av regressorene settes til null. Den beste $\lambda = 10^{-1}$. Hvor metoden gjør at nesten alle regressorene som skal være null, settes til null. Akkurat på samme måte som koblingskonstanten **J** ble definert.

B. Logistisk Regresjon

Siden de beste SGD modellene hadde en nøyaktighet på 53 prosent, betyr dette at modellen ikke klarte å tilpasse seg de observerte verdiene. Det lille SGD uten mini batches klarte å tilpasse seg trenings tilstandene, så ingen effekt på test eller de kritiske tilstandene. Altså modellen må ha tilpasset seg på en for spesifikk måte, som ikke fant sammenhengen mellom de organiserte og uordnede tilstandene. En annen effekt av at modellen ikke klarte å tilpasse seg, var at antall epocher ikke hadde en effekt. Hvis modellen hadde klart å tilpasse seg, ville det blitt problemer

med at modellen var for tilpasset. Når Mehta et al [7] brukt scikit learn til å bestemme fasen til det to dimensjonale Ising modellen, var det L2 regularisering som gjorde at de fikk nøyaktighet over 60 prosent. For regularisering konstant på 10^{-5} ble resultatene det samme som vi fikk, altså med nøyaktighet rundt 50 prosent. Men med økt regularisering opptil 1, ble nøyaktigheten bedre for de kritiske tilstandene. Denne regulariseringen som fungerer på samme måte som Ridge, klarer å gjøre modellen mer generell. Dette resulterer i at nøyaktigheten kommer i nærheten av 70 prosent. Uten denne regulariseringen klarer ikke logistisk regresjon å finne de riktige vektene. Altså for ordnete og uordnete tilstander, er årsak og virkning for komplisert for ren logistisk regresjon.

C. Nevrale nettverk ising modell regresjon

Det ble først avlet frem gode η- og $lambda$-kombinasjoner $\eta = 10^{-3}; \lambda = 10^{0}$, $\eta = 10^{-2}; \lambda = 10^{-1}$ og $\eta = 10^{-1}; \lambda = 10^{-2}$. Hvis man ser på graden av $\lambda * \eta$ ser vi at den er konstant. Det viser at for funksjonen som det tilpasses så er dette den gyldne kombinasjonen. Med en lav, men ikke for lav, η-verdi går gradienten til vektene mot et den sanne verdien sakte men sikkert. Derimot når λ-verdien økes så hindres vektene å bli for store. Måten det gjøres på er, ligning 13, å skyte gradienten mer i samme retning ved økende λ. Dette observerer vi i plottet på github [1] med navn $Eta\_lambda\_testing.png$ som økende støy, fordi vektene fluktuerer hardere frem og tilbake. Dette kan virke positivt ved at MSEen ikke blir stående i et lokalt minimum, men fluktuerer nok til å gå forbi barrieren. I tillegg til at λ-verdien bidrar til dette, så gjør også det at implementeres batch gradient decent at sjansen for å bli stående i et lokalt minimum mye mindre.

Hyperparameter kombinasjonene brukes deretter til å finne antall hidden lag som gir best resultat. Dette viser seg å være 4, hvor den valgte kombinasjonen var $\eta = 10^{-3}$ og $\lambda = 10^{0}$. Dette var ikke resultatet med lavest MSE, alle var ganske like med små variasjoner. Grunnen til at denne ble valgt var at MSE kjapt går mot et minimum og at den ikke har for mye støy. Hvis man velger å stoppe et sted for å bruke vektene får man da et stabilt resultat hver gang. Men det er derimot ikke for lite støy slik at den kan overvinne eventuelle lokale minimum. Plottet i figur 6.

For undersøke om det var et lokalt minimum, ble det beregnet MSE med gode hyperparametere, men som har mer støy enn de beste parameterene. Vist i figur 7.

Det viser at de beste parameterene faktisk er de beste. For å undersøke om det kan hjelpe med en funksjon som har brattere overgang brukes tanh-funksjonen som aktiverings-funksjon. Denne funksjonen går fra -1 til 1, softsign funksjonen går derimot kun til 0.7. Det kunne tenkes

at siden input har verdier fra -1 til 1 så vil dette kunne forbedre resultatet. Plottet i figur 8 viser derimot at minimum MSE er høyere enn med softsign og det gir økt støy. Dette kan komme av at energiene/verdiene som tilpasses inneholder 0. Hverken softsign eller tanh klarer å sette verdier til 0 fordi det er en bias og en helning rundt null, som betyr at de vil fluktuere frem og tilbake. Med tanh vil de fluktuere hardere siden helningen er brattere og kanskje er grunnen til at softsign-funksjonen fungerer bedre for denne funksjonen.

D. Diskusjon av de forskjellige metodene

Når det kommer til tilfellet med regresjon observeres det OLS og Ridge-regresjon ikke fungerer spesielt bra for funksjonen som skal tilpasses. Dette er fordi begge disse metodene setter alle regressorene til en verdi, men som vi allerede vet består **J** matrisen av mange 0 verdier. Lasso fungerer spesielt bra for denne funksjonen siden den har egenskapen til å sette noen regressorer til 0 for en spesifikk λ-verdi.

Når det samme gjøres med et Nevralt Nettverk ser vi at den utkonkurrerer både OLS og Ridge. Den setter derimot heller aldri noen verdier til 0 grunnet at det er positive og negative verdier som skal tilpasses. Ridge fungerer spesielt bra i dette tilfellet og er den metoden man ville brukt. Men det nevrale nettverket viser en allsidighet ved at den er veldig justerbar til å tilpasses forskjellige funksjoner. I dette tilfellet ga funksjonen som skulle tilpasses nettverket en ulempe, men kan fungere mye bedre på andre funksjoner.

E. Nevralt nettverk for to dimensjonalt Ising

De optimale parameterne grupperte seg rundt de samme verdiene. Men de som hadde best verdi, var de med minst mulig læringsfrekvens og høyest grad av regularisering. Altså det å bruke SGD med mini batches var ikke nok for å forhindre overfitting, den trenger også regulariseringskonstant fra 0.1-1.0. Fra Figur 41 i Mehta et al [7] kan man se at de fikk en nøyaktighet på 99.6 prosent for 10 nevroner, men de trengte 1000 nevroner og 100 epocher for å få en nøyaktighet på 96.2 prosent for de kritiske tilstandene. For våre resultater ble det brukt 20 nevroner og 25 epocher for å få en nøyaktighet på 97 prosent. Altså med regularisering, gir det bedre resultater med en fjerdedel av epochene. Dette illustrer hvor stort problem overfitting er for maskinlæring. Men når problemet blir for avansert for regresjon, som i det to dimensjonale Ising modellen. Kan de mer kompliserte modellene som nevralt nettverk finne svaret.

VI. Konklusjon

SGD med og uten mini batches over 100 epocher og elleve forskjellige læringsfrekvenser ble brukt til å løse logistisk regresjon for klassifisering trening og test settet av ordnete og uordnete tilstander fra den to dimensjonale Ising modellen. Trenings sett utgjorde 70 prosent og test settet 30 prosent. Den beste modellen hadde en nøyaktighet på 53 prosent på test settet og 50 prosent for de kritiske tilstandene. Altså den to dimensjonale Ising modellen er for komplisert for regresjon. Deretter ble et nevralt nettverk med 20 nevroner over 20 epocher brukt til å klassifisere den todimensjonale Ising modellen. Det ble testet syv forskjellige verdier av læringsfrekvenser, og for hver av de syv forskjellige grader av regularisering. Altså tilsammen 49 forskjellige modeller. Hver av de modellene sin nøyaktighet ble testet mot test settet, trening settet og de kritiske tilstandene. Det resulterte i en nøyaktighet på 97 prosent for to modeller. Den første hadde læringsfrekvens på 0.001 og regularisering grad på 0.1, den andre hadde læringsfrekvens på 0.0001 og regularisering på 1.0.

Den 1 dimensjonale ising modellen ble tilpasset med OLS, Lasso og Ridge regresjon, hvor koblingskonstanten var satt opp på en måte som gjorde den spesielt egnet for Ridge. De fleste verdiene i koblingskonstant-matrisen var satt til null, hvor L_1 normen har egenskapen at den ofte setter lave parametere til null. Den samme ising modellen ble tilpasset via nevrale nettverk, hvor det ble testet forskjellig verdier av η, λ og antall hidden lag. De beste kombinasjonene for 1D ising modellen var $\eta = 10^{-3}$ og $\lambda = 10^{0}$ med 4 hidden lag. Disse ga en lav MSE som var lavere enn både Lasso og OLS. Derimot passet denne modellen spesielt bra for Ridge som hadde lavere MSE enn det nevrale nettverket og er metoden som bør velges til å tilpasse denne modellen. Nevrale nettverk viser derimot en fleksibilitet, hvor mengden støy antall hidden lag og aktiverings-funksjon kan varieres. Dette er derfor en metode som kan brukes til er bredt spekter av modell tilpasninger.

Referanser

[1] Jonas Asperud. Prosjekt 1/tilleggs figurer. 8.11.2018. https://github.com/Jonas-Asp/FYS-STK4155/tree/master/Prosjekt%202/Bilder/Prosjekt%20bilder.

[2] Magnus Børsting. Prosjekt 2/kode/. 07.10.2018. https://github.com/magnbo/FYS4155/tree/master/Prosjekt%202/Kode/.

[3] Aurelien Geron. *Hands On Machine Learning With Scikit Learn & TensorFlow*. O Reilly Media Inc., 2017.

[4] Morten Hjorth-Jensen. Data analysis and machine learning lectures: Optimization and gradient methods. *FYS-STK4155*, 2018. Senest 11 November 2018.

[5] Morten Hjorth-Jensen. Data analysis and machine learning: Neural networks, from the simpleperceptron to deep learning and convolutional networks. *FYS-STK4155*, 2018. Senest 11 November 2018.

[6] Welch labs. Neural networks demystified. `https://www.youtube.com/watch?v=bxe2T-V8XRs`. Senest 8 November 2018.

[7] P. Mehta, M. Bukov, C.-H. Wang, A. G. R. Day, C. Richardson, C. K. Fisher, and D. J. Schwab. A high-bias, low-variance introduction to Machine Learning for physicists. *ArXiv e-prints*, March 2018.

[8] Michael A. Nielsen. Neural networks and deep learning. *Determination press*, 2018. Senest 11 November 2018.

[9] Jonas Asperud og Magnus Børsting. Prosjekt 1 - fys-stk4155. `https://v2.overleaf.com/project/5ba8d84c4f229946972d48ec`. Senest 11 November 2018.

[10] W. N. van Wieringen. Lecture notes on ridge regression. *ArXiv e-prints*, September 2015.

[11] Wikipedia. Activation function. `https://en.wikipedia.org/wiki/Activation_function#Comparison_of_activation_functions`. Senest 8 November 2018.