Bruk av feed forward nevralt nettverk for prediksjoner av brystkreft

Siv Aalbergsjø

I denne oppgaven er det utviklet og testet kode for feed forward nevrale nettverk som kan benyttes til regresjon og klassifisering. Det er gjort uttestinger med ulike gradient-metoder for optimering, som igjen er benyttet til å trene de to nettverkene. De nevrale nettverkene er testet mot hhv. lineærregresjon og logistisk regresjon, og gir i begge tilfeller bedre resultater. Vi ser at for tilpasning av numeriske data fra et 2. grads polynom, så produserer det nevrale netteverket en funksjon som ligger svært nær datene, men ikke har form som et 2. grads polynom. For klassifisering av brystkreftdata klarer det nevrale nettverket å påvise alle krefttilfeller, og har ingen falske negative, men 1,4 prosent falske positive.

I. INTRODUKSJON

Nevrale nettverk er datamodeller som er bygget opp slik at de kan gjøre prediksjoner ved å trene på eksempler for det aktuelle problemet (Hjorth-Jensen, Morten, 2021). Modellene etterligner biologiske nevrale nettverk (hjernen) ved å sende informasjon mellom kunstige nevroner ulike lag, bestående av en eller flere noder. Feed forward nevrale nettverk (FFNN) er den enkleste formen for nevrale nettverk hvor informasjonen ikke sendes på tvers innad i lag, men kunn fremover gjennom nettverket (Hjorth-Jensen, Morten, 2021).

Hensikten med denne oppgaven er å utvikle kode for FFNN som kan brukes til regresjon og klassifisering og undersøke hvordan ulike oppsett av nettverkene og ulike metaparametere påvirker modelleringen. Regresjonsproblemet brukes som testcase for å bli kjent ulike optimeringsalgorimter som benyttes i de nevrale nettverkene og for å utvikle FFNN-kode på et problem hvor det er lett å sammenligne modelleringen med kjente løsninger. Deretter tilpasses det nevrale nettverket for å kunne brukes til klassifiseringsproblemer og testes ut og optimeres prediksjoner på brystkreft-data.

I denne rapporten vil jeg i det følgende gjøre rede for metodene som er benyttet i modelleringer og hva som kan være styker og svakheter ved disse. Deretter beskriver jeg implementeringen av metodene og resultater ved uttesting på regresjon av et 2. grads polynom. Til slutt beskriver jeg hvordan det nevrale nettverket må tilpasses for å brukes med klassifiseringsdata og resultatene jeg får når jeg benytter det nevrale nettverket til klassifising av brystkreft-tilfeller.

Programkode, vedlegg og ytterligere plott finnes på https://github.com/sivgaa/fys-stk4155/tree/main/Project2.

II. MODELLERING MED FFNN

I et FFNN gjøres en serie operasjoner på inndata for å produsere utdata gjennom flere lag (Goodfellow *et al.*, 2016). Inndataene til hvert lag kalles i det følgende for $\mathbf{z}$, mens utdataene fra hvert lag betegnes med $\mathbf{a}$. Inndatalaget utgjøres av dataene som leses inn, dette kan være enten enkle verdier (binære eller numeriske) eller et sett med verdier, såkalte features, dersom inndataene våre er mer komplekse, vi kaller uansett dise for $\mathbf{x}$, og disse er også utdata fra det første laget. Deretter gjøres det for hvert lag først en lineær operasjon på dataene når informasjonen føres inn i laget, på denne måten

$$z^{(i)} = W^{(i)}a^{(i-1)} + b^{(i-1)},$$

og deretter en ikke -lineær operasjon når det produseres utdata fra laget

$$a(z)^{(i)} = \sigma(z^{)}.$$

Tallene i parentes her betegner lagene i nettverket. Parameteren $\mathbf{b}$ er et konstantledd, og er vitkig for å hindre det nevrale nettverket å stoppe underveis. Funksjonen σ kalles en aktiveringsfunksjon, og valget av aktiveringsfunksjon påvirker modellens egenskaper. Til slutt er det et utdatalag, og input i dette laget utgjør modellens prediksjoner. Lagene mellom inndatalaget og utdatalaget kalles skjulte lag, og et nevralt nettverk kan ha mange eller få slike, og disse kan ha mange eller få noder (dimensjonen til z). I prinsippet skal det holde med ett lag (Goodfellow *et al.*, 2016). Men for å få til en god modell kan det da være nødvendig med et veldig høyt antall noder, slik at det i praksis fungerer bedre med flere lag med færre noder i hvert lag (Goodfellow *et al.*, 2016). Trening av nettverket består i å optimere $\mathbf{W}$ og $\mathbf{b}$ for at prediksjonene skal bli så gode som mulig.

Som alle maskinlæringsalgoritmer benyttes det en kostfunksjon for å estimere hvor gode prediksjonene er. Når et FFNN trenes, genereres det utdata basert på inndataene som prosesseres gjennom det nevrale nettverket. Deretter gjøres det en tilbakepropagering gjennom nettverket for å beregne den deriverte av kostfunksjonen mhp. $\mathbf{W}$ og $\mathbf{b}$ Disse deriverte benyttes til å oppdatere $\mathbf{W}$ og $\mathbf{b}$ og prosedyren gjentas til det oppnås konvergens eller et visst antall iterasjoner er gjennomført. Dette kalles for gradientmetoder.

A. Optimering med gradientmetoder

Optimering med gradientmetoder benyttes til å finne de parameterne **W** og **b** som minimerer kostfunksjonen til nettverket gjennom å iterativt oppdatere disse ved å ta et steg i motsatt regning av gradienten (Raschka *et al.*, 2022)

$$W = W + \Delta W$$

$$b = b + \Delta b.$$

Dette baserer seg på Newton-Rahpsons metode som i praksis går ut på taylorutvikle kostfunksjonen til 2. orden:

$$C(\theta) = C(\theta)\Big|_{\theta=\theta^0} + (\Delta\theta)\frac{dC}{d\theta}\Big|_{\theta=\theta^0} + (\Delta\theta)^2\frac{d^2C}{d\theta^2}\Big|_{\theta=\theta^0}.$$

Hvis vi så starter med en tilfeldig valgt verdi for θ og itererer, så kan vi oppdatere θ slik at

$$\begin{aligned}\theta^{n+1} &= \theta^n - (\frac{d^2C}{d\theta^2})^{-1}\Big|_{\theta=\theta^n}\frac{dC}{d\theta}\Big|_{\theta=\theta^n} \\ &= \theta^n - \eta^{(n)}g^{(n)}\end{aligned}$$

hvor g er gradienten og η kalles for læringsraten (learning rate) og representerer den andrederiverte av funksjonen. I tilfellet med to parametere (**W** og **b**) kan disse optimeres uavhengig, med samme metode.

I lineær regresjon er den andrederiverte uavhengig av parameterne, slik at den kan beregnes direkte. Men normalt så tilnærmer man den andrederiverte på ulike måter, og denne blir da en form for *metaparameter* i modellen. Denne metoden kalles for *gradient descent* (GD). Metoden kan bruke tid på å konvergere, er avhengig av at det ikke finnes lokale topper mellom verdien det initieres og bunnpunktet som skal finnes. Fordelen med metoden er at den går raskt (tar lange sted) der funksjonen er bratt, og kortere steg når funksjonen blir slakere, slik at det er mindre sannsynlig at man hopper forbi bunnpunktet.

B. Varianter av gradient descent

Det finnes variasjoner over gradient descent som baserer seg på ulike måter å estimere læringsraten η eller på annet vis effektivisere optimeringen.

GD med momentum (bevegelsesmengde) (GDM) benytter informasjon om endring fra forrige steg i tillegg til gradienten slik at hvis det var en stor endring i forrige steg, så gjør vi en større endring nå også, men hvis det var endring i motsatt retning enn gradienten er nå, så vil endringe i dette steget bli mindre enn ellers.

$$\theta^{(i+1)} = \theta^{(i)} - \eta g^{(i)} + \gamma(\beta^{(i)} - \theta^{(i-1)}) = \theta^{(i)} - v^{(i)}$$

hvor $v^{(i)} = -\eta g^{(i)} + \gamma(\beta^{(i)} - \beta^{(i-1)})$ og det siste leddet er "minnetfra forrige steg. Her er γ en ny metaparameter som innføres. I algoritmen for GDM må man 1) sette opp startgjett for $\theta^{(0)}$ 2) sette paramteren γ 3) sette paramteren η 4) initalisere $v^{(i)}$ eller f.eks. sette $\theta^{(-1)}$ 5) iterativt optimere θ

Adagrad-metoden er en måte å inkludere gradienten til høyere orden i beregningen av $\Delta\theta$. Slik får vi i praksis en adaptiv η. Her er $\mathbf{s}^{(i)}$ en funksjon av g^2.

$$\mathbf{s}^{(i)} = (\mathbf{g}^{(i)})^2$$

$$\theta^{(i+1)} = \theta^{(i)} - \eta_t\frac{\mathbf{g}^{(i)}}{\sqrt{\mathbf{s}^{(i)} + \delta}}$$

RMSprop og ADAM ligner på Adagrad i den forstand at de også tar inn gradienten til høyere orden, i parameteren η, men på litt andre måter. I RMSprop er ρ er en ny metaparameter:

$$\mathbf{s}^{(i)} = \rho\mathbf{s}^{(i-1)} + (1-\rho)(\mathbf{g}^{(i)})^2$$

$$\beta^{(i+1)} = \beta^{(i)} - \eta_t\frac{\mathbf{g}^{(i)}}{\sqrt{\mathbf{s}^{(i)} + \delta}}$$

Adam har enda flere metaparametere og vi ser at denne metoden benytter både gradienten i første potens (m), med minne fra forrige iterasjon, og gradienten i andre potens (s) for å justere steget i θ. Her er ρ_1 og ρ_2 nye metaparametere:

$$\mathbf{m}^{(i)} = \rho_1\mathbf{m}^{(i-1)} + (1-\rho_1)\mathbf{g}^{(i)}$$

$$\mathbf{s}^{(i)} = \rho_2\mathbf{s}^{(i-1)} + (1-\rho_2)(\mathbf{g}^{(i)})^2$$

$$\mathbf{m}^{(i)} = \frac{\mathbf{m}^{(i)}}{1-\rho_1^i}$$

$$\mathbf{s}^{(i)} = \frac{\mathbf{s}^{(i)}}{1-\rho_2^i}$$

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta^{(i)}\frac{\mathbf{m}^{(i)}}{\sqrt{\mathbf{s}^{(i)}} + \delta},$$

C. Stokastisk gradient descent

Med en stochastisk gradient descent (SGD) gjør man i praksis det samme som med GD, bare at i steden for å evaluere gradienten basert på hele datasettet, så velges det i hver iterasjon (tilfeldig) en undergruppe av datapunktene (en minibatch) å beregne gradienten på bagrunn av. Det er mye raskere å beregne gradienten basert på bare noen få punkter.

M brukes om størrelsen på en minibatch, og $m = n/M$ er da antall minibatcher.

$$\theta_{j+1} = \theta_j - \eta_j \sum_{i \in B_k}^{n} \nabla_\beta c_i(\mathbf{x}_i, \theta)$$

Her er B_k en minbatch (altså en undergruppe av datapunktene våre) og k plukkes tilfeldig i intervallet $[1, n/M]$.

Ved å velge tilfeldig hvilken minibatch som brukes for hver iterasjon, unngås bias og statisktisk sett vil man forvente å bruke hele datasettet likevel, fordi det gjøres mange iterasjoner.

En annen parameter som følger med SGD er antall époker". I løpet av en epoke løpes det gjennom alle batchene. Si at man har 10 minibatcher, og 20 epoker. For hver epoke så gjør man 10 iterasjoner, én for hver minibatch. MEN man løper ikke (nødvendigvis) systematisk gjennom alle minibatchene. Det trekkes en tilfeldig batch 10 ganger, slik at man *kunne* ha løpt gjennom alle. Dette gjentas 20 ganger. Etter å ha løpt gjennom alle epokene sine, må man vurdere om man synes det er bra nok dvs. om det er konvergert. Dette kalles en Minibatch stokastisk gradient descent og er den vanligste varianten av SGD (Raschka, nd).

SGD kan implementeres i kombinasjon med hver av de andre gradientmetodene beskrevet over.

D. Aktiveringsfunksjoner i et nevralt nettverk

Et FFNN settes opp ved at de ulike lagene i nettverket defineres, både hvor mange lag, antall noder (dimensjonen) til hvert av lagene og hvilke aktiveringsfunksjoner som benyttes for å gi utdata fra hvert lag som inndata til det neste

$$a = \sigma(z).$$

Det finnes ulike aktiveringsfunksjoner som brukes til ulike formål. Sigmoidfunksjonen er en slags utglattet stegfunksjon som tar verdier mellom 0 og 1, men ikke gir 0 og 1 i seg selv, og kalles også for den logistiske aktiveringsfunksjonen (Hjorth-Jensen, Morten, 2021):

$$a(z) = \sigma(z) = \frac{1}{1 + e^{-z}}.$$

Denne funksjonen varierer glatt og har en analytisk derivert, noe som er hensiktismessig i beregningene, og sigmoidfunksjonen er vanlig å bruke i nevronene i nevrale nettverk (Nielsen, 2015). En ulempe med sigmoidfunksjonen er at den ikke er robust (Glorot and Bengio, 2010) kan føre til at det ikke konvergerer.

ReLU-funksjonen (Rectified Linear Unit) gir 0 for negative inndata og er lineær for positive inndata. Ulempen med denne er at den lett kan gi 0, og i det området hvor den gir 0 har den heller ingen derivert, hvilket i praksis medfører at nodene dør ut og ikke bidrar i modelleringen.

Det er mulig å variere hvilke aktiveringsfunksjoner som benyttes i de skjulte lagene i et nevralt nettverk, men det er avgjørende at aktiveringsfunksjonen som benyttes i det siste lage (utdatalaget) produserer data av den typen man skal sammenligne med. For klassifiseringsproblemer, kan sigmoid-funksjonen benyttes, men dataene må videre konvertere til 0 og 1 for å kunne tolkes. Da kan man f.eks. si at alt over 0,5 blir satt til 1 og alt under 0,5 settes til 0. For regresjonsproblemer, må aktiveringsfunksjonen i det siste laget være en lineær funksjon.

E. Trening av nettverket gjennom tilbakepropagering

For å trene det nevrale nettet, dvs. optimere $\mathbf{W}$ og $\mathbf{b}$,

$$W^{(i)} = W^{(i-1)} - \eta \frac{dC}{dW}$$

$$b^{(i)} = b^{(i-1)} - \eta \frac{dC}{db}$$

må man finne gradientene til kostfunksjonen mhp. disse parameterne. For ett lag i det nevrale nettverket vil disse kunne beregnes ved bruk av kjerneregelen og uttrykkes som

$$\frac{dC}{dW} = \frac{dC}{da}\frac{da}{dz}\frac{dz}{dW}$$

og

$$\frac{dC}{db} = \frac{dC}{da}\frac{da}{dz}\frac{dz}{db}.$$

Dette regner vi ut gjennom å propagere bakover i nettverket igjen. I hvert lag (bakover") vil gradienten til kostfunksjonen avhenge av lagene lengere fremme:

$$\frac{dC}{da_n} = \frac{dC}{da_{n+1}}\frac{da_{n+1}}{dz_{n+1}}\frac{dz_{n+1}}{da_n}.$$

Vi ser at det kun er i det bakerste (første laget på vei tilbake) at det faktisk må beregnes en derivert av kostfunksjonen direkte.

Når de deriverte er beregnet i tilbakepropagerinen kan $\mathbf{W}$ og $\mathbf{b}$ oppdateres iterativt ved bruk av gradientene og valgt gradientmetode. Dette utgjør treningen av det nevrale nettverket.

F. Valg av aktiveringsfunksjoner og kostfunksjoner i nevrale nettverk

Både kostfunksjoner og aktiveringsfunksjoner må velges i det nevrale nettverket tilpasset hva nettverket skal modellere.

For lineær regresjon er det nødvendig at det siste laget har en lineær aktiveringsfunksjon. Vanlige kostfunksjoner som benyttes til lineær regresjon er mean square error (MSE) eller MSE med en regulariseringsparameter som f.eks. Ridge eller Lasso, som diskutert i prosjekt 1. Som et tillegssmål på hvor god modellen er, kan man også betrakte R^2-verdien, omtalt i prosjekt 1.

For klassifisering må det benyttes en aktiveringsfunksjon i siste lag som gir output-data som er 0 eller 1 eller kan tolkes som dette. Sigmoidfunksjonen (den logistiske aktiveringsfunksjonen) benyttes ofte til dette. Kan Cross-enropy kostfunksjonen

$$C(y,p) = -\frac{1}{n}[ylog(a) + (1-y)log(1-a)]$$

som benyttes i logistisk regresjon (Hjorth-Jensen, Morten, 2021). Her representrer y målverdiene, mens a er utdata fra det siste laget, altså prediksjonen modellen gjør. I tillegg kan nøyaktigheten estimeres fra en accuracy-score

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n},$$

som teller opp hvor mange ganger vi traff prediksjonen. For klassifiseringsmodeller er det også av interesse å lage en såkalt *Feilmatrise* som inneholder informasjon om antall rett og galt klassifiserte for hvert utfall.

G. Datasett som er benyttet

I denne oppgaven er det benyttet et enkelt 2. grads polynom tilsatt støy for uttesting av regresjonsmodellene.

$$y = 10 + 3x + x^2 + \varepsilon$$

Hensikten med dette er at det er enkelt å sammenligne modellen med både det man får ved bruk av OLS og de nominelle verdiene.

For klassifiseringen er Wisconsin Breas Cancer Data (Wolberg and Street, 1993) benyttet, importer fra scikit-learn (Pedregosa *et al.*, 2011). Dette datasettet inneholder informsjon om 569 bryst-tumorer, med 30 karakteriserende features og hvorvidt tumoren var malingn eller beningn.

III. IMPLEMENTERING AV GRADIENTMETODER OG UTTESTING MED POLYNOM

Alle modellene er implementert i python ved bruk av bibliotekene NumPy(Harris *et al.*, 2020) og Scikit-learn(Pedregosa *et al.*, 2011) og gjennomgående rapping av kodebiter fra ulike deler av forelesningsnotatene i kurset FYS-STK4155 (Hjorth-Jensen, Morten, 2021) og ukesoppgavene i uke 41-43 i kurset. Plot er laget med seaborn-pakken for visualisering av data (Waskom, 2021).

Gradientmetodene ble testet ut på lineær regresjon av et 2. ordens plynom. Det ble kun gjort sammenligninger med OLS, ikke med Ridge regresjon, rett og slett fordi tiden ikke strakk til. Det ville innebåret å gjennomføre de samme beregningene én gang til, men å bytte ut kostfunksjonen med den regulaiserte for Ridge.

Det ble generelt ikke benyttet skalering av datasett for denne uttestingen. Dette medførste at MSE-verdiene ble store. Men hensikten var bare å se hvilken metode som ga lavest MSE. For alle metodene ble beste MSE rett over 18, fordi dataene ikke er skalerte. Men ved å sammenligne med lineær regresjon, så vet vi at dette er så godt som metoden kan forventes å prestere. Det ble gjort én uttesting med skalerte data, som viste at da ble MSE-verdiene også mye lavere. Det ble gjort mest som en kontroll på at det ikke var noe galt med MSE-beregningen.

Modelleringen ble gjort med et datasett med 100 punkter, deretter ble MSE-verdier beregnet på et generert test-sett med 50 ytterligere punkter.

A. Gradient descent og gradient descent med momentum

For vanlig gradient descent ble det testet ut ulike verdier av læringsraten η, og det ble funnet at $10^-3 < \eta < 10^-1$ ga fornuftige verdier. Plott av MSE som funksjon av η er gitt i Vedlegg A. Det ble gjennomført en kjøring med $\eta = 0.1$ og MSE og de tilpassede paramterne i den lineære modellen ble tilsvarende til OLS, og modellen var konvergert i løpet av ca 1500 iterasjoner.

For gradient descent med momentum (GDM) ble samme learning rates testet som for GD, og γ-verdier mellom 10^-5 og 10^-1. $\eta = 0.005$, og $\gamma = 0.01$ ble funnet å være verdiene som ga lavest MSE, se heatmap i vedlegg A. Men når denne kjøringen ble gjentatt, for å se på konvergens, viste det seg at den ikke var konvergert på 10000 iterasjoner. Derfor ble det testet ut større verdier for å oppnå konvergens. $\eta = 0.1$, og $\gamma = 0.1$ ga tilsvarende verdier som OLS og konvergerte raskere.

Under beskrives implementering og uttesting som ble gjort ved bruk av analytiske uttrykk for gradientene. Det ble også gjort uttesting med autograd-funksjonen fra numpy. Disse ble bare testet i stikkprøver og ga samme resultat som de analytiske gradientene gjorde.

B. Implementering av stokastisk gradient descent

For den første implementeringen av SGD ble det benyttet vanlig gradient descent med $\eta = 0.1$, som hadde fungert godt for GD og en versjon med en η som avtok for hver iterasjon slik at stegene ble kortere og kortere etter hvert. Det ble gjort utprøvinger med både 5 og 10 minibatcher. Versjonen med 10 minibatcher konvergerte

best. Plott som viser konvergensen til disse kjøringene ligger i vedlegg A. Det ble tydelig at den med η som avtar gjennom kjøringene konvergerte til en løsning som lå nærmere den nominelle verdien enn OLS gjorde.

Jeg prøde også ut SGD med momentum med 10 minibatcher og samme verdier for metaparamterne som fungerte for vanlig GD med momentum: $\eta = 0.1$, og $\gamma = 0.1$. Denne utprøvingen konvergerte veldig sakte, og til samme verdier som OLS.

C. Implementering og utprøving av Adagrad, RMSprop og Adam

Adagrad, RMSprop og Adam er alle metoder som har learing rates som avhenger av gradienten og dermed ikke er konstante. Disse metodene ble kun testet ut med SGD med 10 minbatcher.

For Adagrad ble det testet ut η-verdier mellom 10^-3 og 2×10^-2 og 5×10^-3 ga den laveste MSE-verdien. Denne konvergerte ikke helt på 5000 iterasjner, men så ut til å være på veil til å konvergere til en verdi som lå tettere på den nominelle enn OLS. En større learing rate (som burde konvergert fortere) fungerte ikke.

For RMSprop ble det testet med samme η-verdier som for Adagrad, og med $10^-4 < \rho < 01^-1$. Beste verdier ble funnet å være $\eta = 0,05$ og $\rho = 0,01$, og denne kjøringen så ut til å konvergere. Men den divergerte faktisk etter ca 4000 steg! Konvergensplott er lagt ved i Vedlegg A.

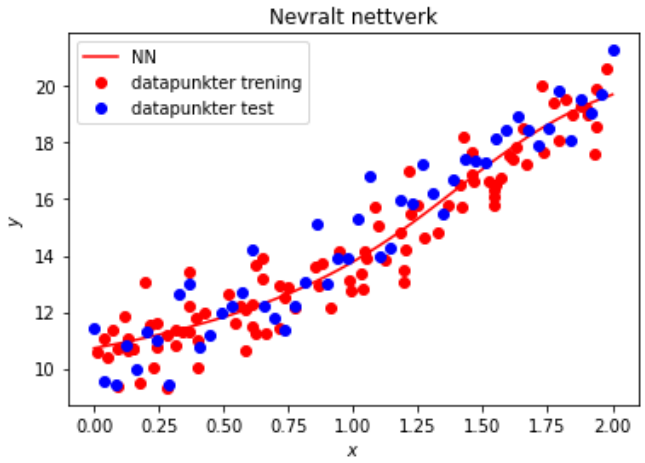
For utprøving med Adam ble det gjort utprøvinger med $\eta = 0.1$ siden det har fungert greit før og ρ_1 og ρ_2 mellom 10^-4 og 10^-1. Det så fra heatmap ut som at $\rho_1 = 0,001$ og $\rho_2 = 0,01$ ga beste verdier. Men heller ikke disse kjøringene konvergerte, og så heller ut til å divergere.

IV. Implementering av FFNN for regresjon og uttesting med polynom

Det nevrale nettverket for regeresjon er implementert og testet ut for det samme 2. grads polynomet som ble benyttet i testen av gradientmetodene med 100 datapunkter.

Sigmoidfunksjonen ble benyttet som aktiveringsfunksjon i alle lag unntatt det siste laget, hvor det ble benyttet en lineær funksjon. MSE ble benyttet som kostfunksjon. Det ble heller ikke her benyttet noen preprosessering av dataene. Testene etter trening av nettverket ble gjort på nygenererte data med 50 punkter.

Programmet ble satt opp slik at det kan genereres et vilkårlig antall lag med vilkårlig antall noder. Parameterne W og b ble initiert med tilfeldige tall. Det viktige er at de ikke er null til å begynne med, slik at nodene dør ut. Optimeringen ble gjort med stokastisk gradient descent med momentum. De mer avanserte metodene førte uansett ikke frem i uttestingen.



Figur 1 Modellering av 2. grads polynom med FFNN med ett skjult lag med 3 noder. Modellen som genereres av det nevrale nettverket er vist som en rød linje.

Vedlegg B inneholder resultater fra de ulike kjøringene. Det var tydelig at ett skjult lag ga bedre prediksjoner enn flere skjulte lag, og at 3 noder var tilstrekkelig i det skjulte laget. Færre eller flere noder i det skjulte laget ga dårligere modell. I Figur 1 ser vi modellen som en rød heltrukken linje, mens datapunktene benyttet til trening er røde og datapunktene i testsettet er blå. Vi ser tydelig at dette ikke er et 2. grads polynom. Men det er også tydelig at det er en linje som passer godt til dataene med denne mengden støy.

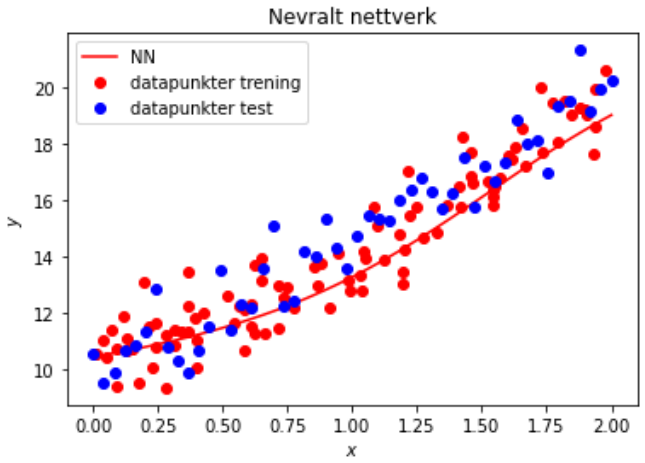
I figur 2 og 3 ser vi henholdsvis modellene som genereres i nettverk med 1 skjult lag med 10 noder og 3 skjulte lag med 3 noder i hvert lag. Modellen med ett skjult lag med 10 noder er tydelig, selv på øyemål en mye dårligere modell en den med ett skjult lag med 3 noder. Dette bekreftes også av MSE-verdiene, se Vedlegg B. For nettverket med 3 skjulte lag og 3 noder i hvert lag skjer en aldri så liten katastrofe.

Det ble også gjort en utprøving med ReLu aktiveringsfunksjon i det skjulte laget. Her ga 10 noder i det skjulte laget bedre resultater enn 3. Men modellen er klart dårligere enn med sigmoid-funksjonen i det skjulte laget. Se Vedlegg B for detaljer.

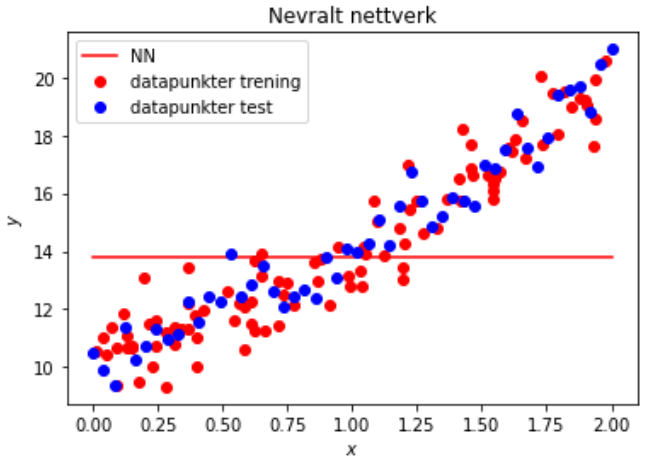
V. Implementering av FFNN for resultater for klassifisering av brystkreftdata

A. Implementering av FFNN for klassifisering

Det nevrale nettverket for klassifisering er implementert og testet ut for binær klassifisering av brystkreftdata. Sigmoidfunksjonen ble benyttet som aktiveringsfunksjon i alle lag, inkludert det siste laget. Cross-entropy ble benyttet som kostfunksjon, og accuracy score og confusion matrix ble brukt for å vurdere resultatet til slutt. For å



Figur 2 Modellering av 2. grads polynom med FFNN med ett skjult lag med 10 noder. Modellen som genereres av det nevrale nettverket er vist som en rød linje.



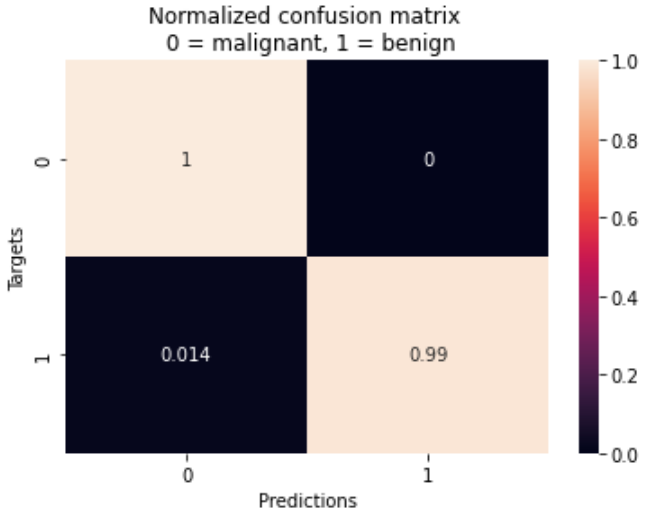
Figur 3 Modellering av 2. grads polynom med FFNN med 3 skjulte lag med 3 noder hvert lag. Modellen som genereres av det nevrale nettverket er vist som en rød linje.

sammenligne prediksjonene med datasettene, ble det satt en terskelverdi på 0,5, slik at alle verdier større enn dette ble satt til 1 og alle under ble satt til 0. I dette datasettet angir verdien 0 en malign tumor (altså positiv = syk) og verdien 1 angir beningn tumor (altså negativ = frisk).

Dataene ble preprosessert med MinMax scaler fra scikit-learn, og delt opp i test- og treningssett, med 20 prosent av dataene i testsettet. Testsettet ble benyttet som valideringssett i optimeringen av metaparameterne. Detaljer fra tesingen finnes i Vedlegg C.

Programmet ble satt opp slik at det kan genereres et vilkårlig antall lag med vilkårlig antall noder. Parameterne W og b ble initiert med tilfeldige tall. Optimeringen ble også her gjort med stokastisk gradient descent med momentum.

Det ble først gjort en systematisk utprøving med ulike



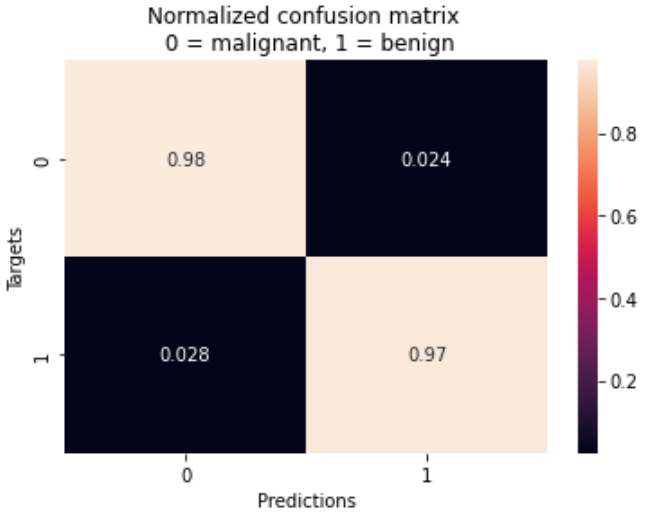
Figur 4 Feilmatrise for kjøring med klassifisering av brystkreftdata. Det er benyttet ett skjult lag med 5 noder, stokastisk gradient descent med momentum med 20 minibatcher og 3000 epoker, $\eta = 0,1$ og $\gamma = 10^-4$

antall minibatcher og epoker i optimeringen for kjøringer med ett skjult lag med 30 noder, og $\eta = 0,1$ og $\gamma = 10^-4$ basert på litt tilfeldig testing i starten. Det ble da funnet at 20 minibatcher og 10 000 epoker var det som ga best resultat. Ved å følge med på kjøringene etter hvert, ble det også tydelig at det ikke var nødvendig med så mange epoker for å nå konvergens.

Deretter ble det gjennomfør systematisk uttesting av verdier for η og γ. Det ble kjørt med 20 minibatcher og 10 000 epoker, men dette ble redusert til 3000 epoker. Det ble da funnet at de optimale paramterne var $\eta = 0,1$ og $\gamma = 10^-4$.

Med disse paramterne ble det gjort uttestinger med ulike antall skjulte lag og antall noder per lag. Det ble forsøkt med opp til 5 skjulte lag og 5, 10, 15 og 30 noder per lag. Tabeller med oversikt over resultater finnes i Vedlegg C. Det ble forsøkt med både 1000 og 3000 epoker, og dette var nok til å få gode resultater. Det så ikke ut til å være noe tydelig system for hvilke kjøringer som ble best. Men ett lag med 5 noder gjorde det svært godt, ed en accuracy på 0,99 for test-settet og 100 treffrate kreft og ingen falske negative. Feilmatrisen viser at modellen identifiserer 100 prosent av de maligne kreftsvlustene, men klassifiserer 1,4 prosent av de beningne (ikke kreft) svulstene som kreft.

Det ble også gjort utprøving med ReLU som aktiveringsfunksjon i de skjultelagene. Dette fungerte dårlig. Se vedlegg for detaljer.



Figur 5 Feilmatrise for logistisk regresjon for klassifisering av brystkreftdata.

VI. LOGISTISK REGRESJON FOR BRYSTKREFTDATA

Det ble også gjennomført en logistisk regresjon på brystkreftdataene for å gjøre prediksjoner. Dette er i praksis bare et nevralt nettverk helt uten skjulte lag. Dette ble gjennomført med 20 minibatcher og 5000 epoker i optimeringen. Det ble funnet at beste learning rate var 1 og beste gamma var igjen svært liten: $10^{-}5$. Dette resulterte i en accuracy på 0,9737. Feilmatrisen vises i figur 5. Vi ser at det nevrale nettet gjør en bedre jobb enn dette.

VII. OPPSUMMERING

Hensikten med denne oppgaven var å utvikle kode for FFNN som kan brukes til regresjon og klassifisering og bruke disse til å undersøke hvordan oppsett av nettverkene og metaparametere påvirker kjøringen. Det har blitt testet ulike gradientmetoder for optimering og ulike aktiveringsfunksjoner for både regresjonsdata og klassifiseringsdata.

Optimeringen de de nevrale nettverkene ble gjort med stokastisk gradient descent med momentum. Dette ble gjort fordi det var poengtert at vi skulle gjøre det i ukesoppgavene, og derfor tenkte jeg av en eller annen grunn at vi kom til å trenge den momentum-delen for å få fart nok påsakene i prosjektet. I ettertid ser jeg at resultatene mine tyder på at dette ikke er nødvendig. Resultatene med uttesting av gradientmetoder tyder ikke på at momentum gjør at det konvergerer raskere. Dessuten finner jeg i uttestingen med det nevrale nettet at γ bør være nesten så liten som mulig, noe som egentlig tilsvarer å ta vekk momentum-leddet. Jeg kunne antagelig spart meg for det.

I testene med det nevrale nettet for klassifiseringsdata var det ikke noe åpenbart system for hva som fungerte best eller dårligst i modelleringen. Men jeg vil si jeg fikk påfallende gode resultater! De fleste av kjøringene mine ga få falske negative, hvilket betyr at de som har kreft får vite det og kan behandles. Til gjengjeld var det oftere falske positive, dvs påvist kreft som ikke er kreft. Dette er en kjent problemstilling fra mammografiprogrammet her hjemme. Behandlingen av brystkreft har økt etter at man begynte med systematisk screening og man frykter at det nå bedrives overbehandling. Behandlingen er svært tung, men den er ikke dødelig, slik ubehandlet brystkreft er.

I mine kjøringer lag jeg inn en terskelverdi op 0,5 som styrer hva om tolkes som positiv og negativ. Ved å justere denne paramteren, kan man ytterligere tune modellen ved å gjøre den direkte mer eller mindre sensitiv. Det virket derimot ikke å være nødvendig i mitt tilfelle.

Både i tilfellet med tilpasning av data fra et polynom og klassifisering av brystkrefdata ser vi at med det beste oppsettet av de nevrale nettverkene og optimering av metaparamtere så er FFNN bedre enn lineær regresjon og logistisk regresjon. Men i tilfellet av lineær-regresjon, så ser vi også at det nevrale nettverket ikke gir en funksjon med samme fasong som dataene er generert med. Derfor må vi anta at det nevrale nettverket vil fungere svært dårlig på data som ligger utenfor området det er trent på.

A. Forbedringspotensialer

Det ble dessverre ikke tid til å gjennomføre beregninger med regulariseringsparameter (Ridge) hverken for lineær eller logistisk regresjon eller i de nevrale nettene.

Jeg skulle også gjerne hatt tid til å teste ut koden mer, både ved å prøve ut regresjonen f.eks. med Frankefunksjonen og å få litt bedre innsikt hvorfor det virker så tilfeldig hvilke paramtere for klassifiseringen som gir gode modeller. Jeg mistenker at det er en bug et sted, eller bare at optimeringsalgoritmen ikke er så god. Kanskje Adagrad, RMSprop eller Adam ville fungert bedre.

Heller ikke i dette prosjektet brukte jeg et testsett for klassifiseringsdataene som ble holdt utenfor optimeringen helt fra starten. Det gjør at jeg egentlig ikke vet hvor god modellen min blir. Ideelt sett burde jeg splittet dataene i trenings-, validerings- og test-sett (Raschka *et al.*, 2022). Det skyldes rett og slett at jeg har vært for opptatt med å få koden til fungere, til å huske på det, før jeg gjorde testkjøringene.

Jeg har heller ikke implementert noen form for bootstrapping eller cross-validation. Det kunne man jo gjort, siden datasettet for kreft-dataene er såpass lite.

Og jeg har en drøm om å objektorientere koden, og hadde håpet jeg skulle rekke det denne gangen. Men det kommer et prosjekt 3.

Men jeg vil si at jeg synes det har vært utrolig nyttig med måten ukesoppgavene er bygget opp på hvor vi

den første uken fikk erfaring med og tid til å implementere GD-metoder, så neste uke feed forward-algoritmen og tilbakepropageringen den siste uken. Dessverre var det fra jeg leverte siste ukesoppgave bare en uke igjen til prosjektet skulle være ferdig. Og det ble rett og slett litt for snaut for meg til å rekke å tilpasse fra ukesoppgavee til casene i prosjektet (det var jo en del andre funksjoner) og implemnetere de siste tingene, teste ut og skrive. Jeg hadde nok egentlig trengt en ekstra uke mellom å få til tilbakepropagering og innelvering av prosjekt :)

REFERANSER

X. Glorot, and Y. Bengio (2010), "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research, Vol. 9, edited by Y. W. Teh, and M. Titterington (PMLR, Chia Laguna Resort, Sardinia, Italy) pp. 249–256.

I. Goodfellow, Y. Bengio, and A. Courville (2016), *Deep Learning* (The MIT Press, Cambridge, Massachusetts).

C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant (2020), "Array programming with NumPy," Nature **585** (7825), 357–362.

Hjorth-Jensen, Morten (2021), "Applied data analysis and machine learning," .

M. A. Nielsen (2015), *Neural Networks and Deep Learning* (Determination Press).

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011), "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research **12**, 2825–2830.

S. Raschka (n.d.), "How is stochastic gradient descent implemented in the context of machine learning and deep learning?" .

S. Raschka, Y. H. Liu, V. Mirjalili, and D. Dzhulgakov (2022), *Machine Learning with Pytorch and Scikit-Learn: Develop Machine Learning and Deep Learning Models with Python*, 1st ed. (Packt Publishing, Limited, Birmingham).

M. L. Waskom (2021), "seaborn: statistical data visualization," Journal of Open Source Software **6** (60), 3021.

M. O. S. N. Wolberg, William, and W. Street (1993), "Breast Cancer Wisconsin (Diagnostic)," UCI Machine Learning Repository, DOI: https://doi.org/10.24432/C5DW2B.