Neurale netværk til multinomial klassifikation af aktieafkast

Bachelorprojekt

Mads Salinas (S151817)

Mikkel Hansen (S153541)

Studie: HA(Mat)

Vejleder: Peter Dalgaard

Afleveret: 23. maj 2024

Opgavens omfang i sider: **50**

Antal sider i alt: **64**

Antal anslag: **95040**

Abstract

This project aims to build neural networks for multinomial classification of daily stock returns as well as use **L**ocal **I**nterpretable **M**odel-agnostic **E**xplanations Technique (LIME) as a technique to solve the black-box problem. We will construct three types of networks, Artificial Neural Networks (ANN), Recursive Neural Networks (RNN), and Long Short-Term Memory Neural Networks (LSTM). The networks will model the daily returns of the Shell stock. They build upon 20 years of data of underlying momentum variables, the stock price of industry peers, and a commodity index. We find that none of the three types of neural networks are able to achieve significantly higher precision than 50% and that not one performs significantly better than the others. Furthermore, we find that LIME can help solve the black-box problem on neural networks. However, it is possible that the modelling requirements in this scenario are so complex that LIME fails to make reliable local models.

Abstract - Dansk

Dette projekt har til formål at bygge neurale netværk til multinomial klassifikation af aktieafkast samt benytte ***L**ocal **I**nterpretable **M**odel-agnostic **E**xplanations Technique* (LIME) som metode til at løse black-box -problemet. Vi vil opstille tre forskellige typer af neurale netværk, simple neurale netværk (ANN), rekursive neurale netværk (RNN) og *Long Short-Term Memory* neurale netværk (LSTM). Netværkene modellerer Shell-aktiens daglige afkast og bygger på 20 års data af underliggende momentum variable, aktiepriser for industrikonkurrenter og et handelsvareindeks. Vi finder, at ingen af de tre netværkstyper er i stand til at opnå betydelig højere præcision end 50%, og at der ikke er en netværkstype, som udviser klart højere præcision end de andre. Dertil kommer, at LIME er i stand til at hjælpe med at løse black-box-problemet på neurale netværk. Dog er det muligt, at modelleringsbehovet i denne situation er så komplekst, at LIME fejler i at skabe pålidelige lokale modeller.

Indhold

1 Notation

I følgende opgave benytter vi os af følgende notation fra (Ng Andrew, 2019).

Dimensioner

- n_x : Antallet af observationer for variablen x
- n_y : Antallet af observationer for output-vektoren y
- $n_h^{[l]}$: Antallet af noder i lag l
- L : Antallet af lag i det neurale netværk

Objekter

- $x_i \in \mathbb{R}^{n_x}$: Repræsenterer den i'te input-vektor
- $W^{[l]} \in \mathbb{R}^{n_h^{[l]} \times n_h^{[l+1]}}$: Vægtmatrix for lag l
- $\hat{y} \in \mathbb{R}^{n_y}$: Den prædikterede output-vektor

Ligninger

- $g^{[l]}$: Repræsenterer *activation funktionen* for lag l
- $E(\hat{y}, y)$: Repræsenterer tabsfunktionen
- $\xi(x)$: Repræsenterer LIME-forklaringen af prædiktion x

2 Begrebsoversættelse

Nedenstående liste giver overblik over, hvorledes engelske machine learning begreber er oversat og forkortet i opgaven. Visse begreber bliver ikke oversat, da de ikke eksisterer på dansk eller har en anden betydning på dansk.

- *Activation function*: *Activation function*
- *Artificial Neural Network*: Simpelt neuralt netværk (ANN)
- *Back-propagation*: Back-propagation
- *Batch / Batch size*: Batch / Batch-størrelse
- *Bias*: Bias
- *Cross-entropy*: *Cross-entropy*
- *Epoch*: Epoke
- *Forward pass*: Forward pass
- *Gate(s)*: Port(e)
- *Gradient descent algorithm*: Gradient descent algoritme
- *Hidden layer*: Skjult lag
- *Hidden Unit*: Skjult node
- *Learning rate*: Læringsrate
- *Long Short-Term Memory Neural Network*: Long Short-Term Memory Neuralt Netværk (LSTM)
- *Loss function*: Tabsfunktion
- *Optimizer*: *Optimizer*
- *Recurrent Neural Network*: Rekursivt Neuralt Netværk (RNN)
- *Unit*: Node
- *Vanishing/Exploding gradient problem*: Forsvindende/Eksploderende gradient problem
- *Weights*: Vægte

3 Introduktion og problemformulering

Den finansielle sektor forvalter hver dag milliarder af kroner ved at handle med et utal af finansielle instrumenter såsom aktier, derivater og obligationer. I forsøget på at maksimere afkastet ved hver handel undersøger finansielle institutioner konstant nye metoder, der kan give dem en fordel i forhold til markedet. Inden for aktieverdenen er det for eksempel specielt interessant på forhånd at vide, hvad det daglige afkast for en aktie potentielt er. Hvis dette lykkes, vil en sådan institution kunne tjene mange penge og opnå store profitter. Teorier, såsom *random walk*, siger dog, at aktier bevæger sig tilfældigt og uforudsigeligt, hvilket udelukker muligheden for at kende det fremtidige afkast for en aktie.

I takt med øget digitalisering har adgangen til enorme datamængder og mere computerkraft gjort det muligt at bygge mere og mere komplekse modeller. Hér har machine learning modeller, såsom neurale netværk, fundet sit eksistensgrundlag, da de er i stand til at lære komplekse sammenhænge fra data uden eksplicitte instruktioner.

Dog kan neurale netværk blive så komplekse, at de kommer til at fremstå som black-box-modeller. I sådanne situationer har brugeren af modellen lille eller slet ingen forståelse for, hvad der ligger til grund for netværkets beslutninger. Dette kan svække pålideligheden af modellens prædiktioner og dermed sænke incitamentet til at benytte den. Dette problem forsøger ***L**ocal **I**nterpretable **M**odel-agnostic **E**xplanations Technique* (LIME) at løse.

På baggrund af disse overvejelser vil vi i denne opgave fokusere på brugen af neurale netværk til klassifikation af aktieafkast. Dette leder os til problemformuleringen:

Hvordan kan neurale netværk benyttes til klassifikation af Shells aktieafkast, og hvordan kan black-box-problemet løses ved hjælp af LIME?

For at besvare problemformuleringen vil vi:

- Konstruere tre typer af neurale netværk, simple neurale netværk (ANN), rekursive neurale netværk (RNN) og *Long Short-Term Memory* (LSTM) neurale netværk og vurdere deres præcision
- Udføre tuning af hyperparametre for at øge præcisionen af de tre netværkstyper
- Benytte *Local Interpretable Model-agnostic Explanations Technique* (LIME) til at skabe sammenhæng mellem inputvariable og det neurale netværks prædiktioner

Opgaven vil indlede med metodeovervejelser og en beskrivelse af datasættet. Herefter vil vi i afsnit 5 - 11 gennemgå de underliggende metoder som neurale netværk bygger på samt en introduktion til LIME. I afsnit 12 og 13 bygger vi neurale netværk ud fra de beskrevne metoder og analyserer samt diskuterer vores resultater. Afslutningvis vil vi afsnit 15 foreslå tilføjelser til lignende projekter.

4 Metodeovervejelser

Vi har valgt at afgrænse os til at lave multinomial klassifikation af aktieafkast. Motivationen for at lave multinominal klassifikation fremfor binær klassifikation omhandler at øge vores viden om afkastet, som en model prædikterer. I en binær model er det kun muligt at konstatere, om en aktie vil gå op eller ned, men ikke hvor meget. Eksempelvis kan en model klassificere aktieafkastet som positivt, men hvor afkastet i realiteten ikke opvejer for transaktionsomkostningerne. Dette gør det mindre attraktivt i et finansielt henseende. Ved at inddele aktieafkastet i flere kategorier tydeliggøres det, hvorvidt det potentielle afkast opvejer omkostningerne.

Neurale netværk er også i stand til at lave regression, hvor man forsøger at prædiktere den eksakte aktiepris i stedet for en kategori. Vi fravælger regression, da klassifikation gør mulighederne for at fortolke på sammenhænge mellem inputvariable og modellens prædiktioner bedre. Dette skyldes, at LIME bygger på tærskler af inputvariable fremfor de eksakte værdier, hvilket vores metode tillader.

Det er alment kendt, at man ikke kan lave inferens på black-box-modeller såsom neurale netværk (Molnar, 2020, s. 3). Dette er en åbenlys ulempe ved brugen af neurale netværk. Med aktieafkast kan det således være svært at gennemskue, hvilke faktorer der har indflydelse på, hvilken kategori modellen prædikterer. Manglen på gennemskuelighed kan mindske brugerens opfattede pålidelighed af modellen, da man ikke er i stand til at verificere grundliggende antagelser. Dette problem forsøger LIME at løse ved at lave inferens baseret på lokale områder af modellen. Brugeren kan således bekræfte sine grundlæggende antagelser og dermed øge pålideligheden af resultaterne. Samtidigt er brugeren også i stand til at opdage potentielle fejl, som kan være forekommet i databehandlings- eller træningsfasen.

Kode

I forbindelse med opbygning af de neurale netværk har vi primært benyttet `Keras`-pakken. Derudover har vi valgt at "sætte seed". Dette fjerner tilfældigheden, der normalt opstår ved neurale netværk. Dette gør os i stand til at isolere effekten af ændringer i for eksempel hyperparametre. Derudover skærer det drastisk ned på tiden, det tager at producere resultater.

Økonomisk data

Vi har valgt at bruge Shell (TICKER: SHELL.AS) som underliggende aktie i vores machine learning modeller. Vi har valgt Shell, da det er let at kategorisere Shell og dens konkurrenter i ener-

gisektoren. Energisektoren kan opdeles i fire kategorier henholdsvis olie og naturgasser, raffinering og pipeline, minedrift (især udvinding af kul og uran) og vedvarende energi (Rodeck, 2024). Her ligger Shell primært i de to første kategorier, hvilket betyder, at konkurrenterne også skal findes her. Vi har valgt Chevron (TICKER: CVX), British Petroleum (TICKER: BP) og Exxon Mobile (TICKER: XOM) som nærtliggende konkurrenter i energisektoren. Vi benytter de konkurrerende virksomheders aktiekurser for at undersøge, om det kan forbedre mulighederne for at klassificere afkastet for Shell-aktien, da man kunne forvente en vis korrelation på industriniveau („Understanding Relationships Between Industry Peers with MarketReader - MarketReader“, 2023).

Da olie er en handelsvare (*commodity*), inkluderer vi også S&P GSCI, som er et handelsvareindeks. S&P GSCI-indekset er en samling af forskellige typer af handelsvarer. De energirelaterede handelsvarer, såsom olie, repræsenterer 54% af indekset, hvor de resterende 46% dækker metaller og landbrug (Hayer, 2022). Indekset tillader modellerne at fange, om udsving i råvaremarkedet har indflydelse på det daglige afkast af Shell-aktien.

Vi har også inkluderet *relative strength index* (RSI) og *moving average convergence/divergence* (MACD), som er to momentum variable for Shell-aktien. RSI måler hastigheden og størrelsen på en akties nylige prisændringer for at vurdere om aktien er værdiansat højt eller lavt (Fernando, 2024). MACD bruges til at identificere trends i priser, til at måle disse trends' momentum og til at identificere indgange på markedet for at købe eller sælge aktier. RSI og MACD bliver ofte brugt sammen af analytikere til at danne et mere komplet billede af aktien og dens momentum (Dolan, 2022).

Samlet set har vi dermed data, der forsøger at fange udsving på industriniveau, råvaremarkedet såvel som på Shell-aktien selv.

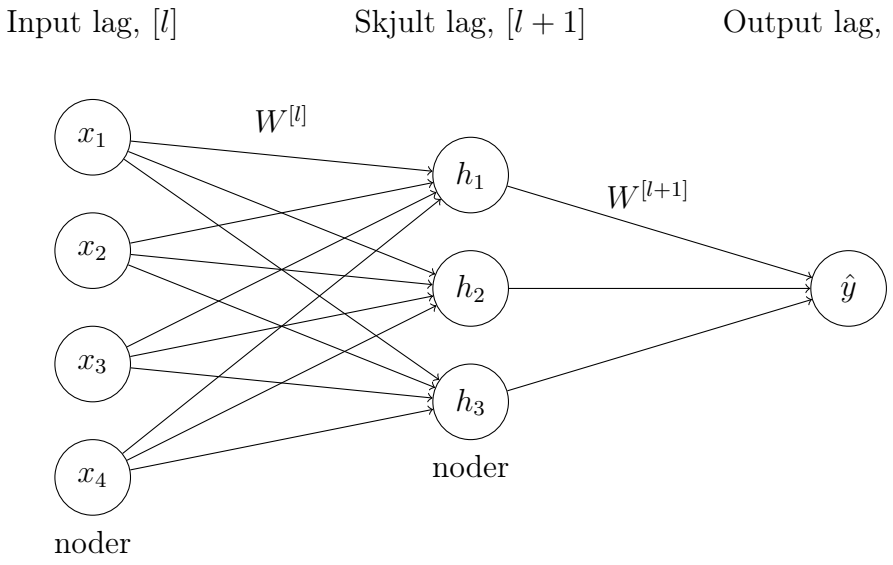
Alt data i opgaven er hentet igennem Eikon Datastream.

5 Introduktion til simple neurale netværk og den underliggende arkitektur

Afsnit 5 er baseret på (Müller & Guido, 2017).

Et neuralt netværk er en deep learning algoritme, som kan være designet til regression eller klassifikation. Neurale netværk kan ses som en generalisering af lineære modeller, hvor udregningsprocessen i stedet består af flere stadier. Oftest bliver deep learning algoritmer "skræddersyet" til en opgave. Man kan skræddersy netværket ved at justere på arkitekturen, hvilket blandt andet sker gennem valg af forskellige hyperparametre.

Et neuralt netværk er designet ud fra princippet om en animalsk hjerne, hvor synapser sender signaler igennem hjernen. Dette imiteres i form af forbindelser mellem skjulte lag og noder i netværket. Et simpelt neuralt netværk (ANN) kan bygges op som figur 1. Dette netværk indeholder 4 noder i inputlaget, 1 skjult lag med 3 noder og en outputnode.



Figur 1: Illustration af et neuralt netværk med 4 inputnoder, ét skjult lag med 3 skjulte noder, og ét output.

Neurale netværk kan også tolkes ved matrix-notation. Hvert skjult lag l består af et antal skjulte noder, $n_h^{[l]}$. I forbindelserne mellem hvert lag tilføjes koefficienter, som indenfor neurale netværk

kaldes vægte. Vægtene i hvert lag kan repræsenteres ved en matrix, $W^{[l]}$, som kan defineres således:

$$W^{[l]} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n_h^{[l+1]}} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n_h^{[l+1]}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_h^{[l]},1} & w_{n_h^{[l]},2} & \cdots & w_{n_h^{[l]},n_h^{[l+1]}} \end{bmatrix}$$

Her er hvert w en vægt på en forbindelse mellem lag l og $l+1$. Der opstår typisk mange vægte, som skal trænes på grund af kompleksiteten af neurale netværk

Netværket udregner noderne i det næste lag som:

$$h_j^{[l+1]} = \sum_{i=1}^{n_h^{[l]}} h_i^{[l]} w_{i,j}^{[l]} \qquad j = 1, ..., n_h^{[l+1]} \tag{1}$$

Hvilket i matrix-notation kan skrives som:

$$h^{[l+1]^T} = h^{[l]^T} W^{[l]}$$

I det simple netværk som ses på figur 1, vil man udregne noderne i det skjulte lag, $h_i^{[l+1]}$, og outputnoden, $\hat{y}^{[l+2]}$, som henholdsvis:

$$h_j^{[l+1]} = \sum_{i=1}^{4} x_i w_{i,j}^{[l]} \qquad j = 1, 2, 3 \qquad\qquad \hat{y}^{[l+2]} = \sum_{i=1}^{3} h_i^{[l+1]} w_i^{[l+1]}$$

Det betyder, at outputnoden kan udregnes som en serie af vægtede summer, hvilket matematisk set er det samme som at udregne én vægtet sum, hvilket svarer til en simpel lineær model. For at modellen kan finde komplekse sammenhænge i data, tilføjes *activation functions*, som transformerer data fra et lag til det næste ved hjælp af en ikke-lineær funktion.

6 *Activation functions*

Afsnit 6 er baseret på (Brownlee, 2021) og (Ghatak, 2019).

En *activation function*, $g(\cdot)$, er en funktion, som aktiverer en vægtet sum fra det forrige lag.

$$h_j^{[l+1]} = g^{[l+1]}\left(\sum_{i=1}^{n_h^{[l]}} h_i^{[l]} w_{i,j}^{[l]}\right) \qquad j = 1, ..., n_h^{[l+1]} \tag{2}$$

Transformationen gør algoritmen i stand til at lære mønstre i data, der ikke nødvendigvis er lineære, hvilket gør den mere anvendelig i komplicerede datastrukturer. *Activation functions* kan bruges i både det skjulte lag og i outputlaget. I de skjulte lag benyttes ofte *Rectified Linear Activation* (ReLU) og tangens hyperbolsk (tanh). I outputlaget benyttes der derimod ofte lineære funktioner, logistiske funktioner eller *softmax*. I vores opgave vil vi i de skjulte lag benytte ReLU og tanh, som kan skrives som henholdsvis:

$$g(x) = \max\{0, x\} \tag{3}$$

$$g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4}$$

ReLU er den mest normale *activation function*, da den er let at implementere og ikke er lige så modtagelig for det forsvindende/eksploderende gradient problem (afsnit 8.3). ReLU-funktionens form medfører, at den er følsom overfor problemer såsom døde noder. Døde noder beskriver fænomenet, hvor noder forsvinder fra træningsprocessen. Tanh funktionen bygger ovenpå den logistiske funktion. Den har omtrent samme s-formet kurveforløb og samme definitionsmængden givet ved $x \in \mathbb{R}$. Dertil er tanhs værdimængde defineret til at ligge i $[-1, 1]$. Tanh er en af de *activation functions*, hvor det forsvindende/eksploderende gradient problem kan opstå under kørsel af algoritmen.

I outputlaget benytter vi softmax som *activation function*, da denne anvendes til klassifikationsproblemer, når man har n kategorier med $n > 2$. Softmax er defineret som:

$$P(y \mid x) = \hat{y} = \frac{e^{x_j}}{\sum_{k=1}^{n} e^{x_k}} \forall j \in 1, \ldots, n \tag{5}$$

Værdimængden for softmax funktionen er intervallet [0, 1].

7 Træning af simple neurale netværk

Afsnit 7 er baseret på (Aggarwal, 2023), (Ghatak, 2019), (Goodfellow m.fl., 2016), (Liu, 2020) og (Müller & Guido, 2017).

Træningen af et neuralt netværk er en iterativ tovejsproces, der forsøger at optimere vægtene i

netværket sådan, at datastrukturen modelleres bedst muligt. Her er "bedst muligt" defineret ved det sæt af vægte, som minimerer netværkets prædiktionsfejl.

Inden træningen begynder, inddeler man data i henholdsvis træningsdata, valideringsdata og testdata. Træningsdata er, som ordet indikerer, dét data, man træner sit neurale netværk på. Valideringsdata er den del af data, man benytter til at teste sin model på i tuningen af hyperparametrene. Hyperparametre vil blive forklaret i afsnit 10. Testdata er den del af data, man tester sin endelige model på, hvormed man opnår et mål for dens præcision.

Den første del af træningsprocessen består af en vægt-initialisering, som kan følge forskellige strategier (afsnit 7.3). Herefter følger et *forward pass*, hvor inputdata sendes forlæns igennem det utrænede netværk. Netværket benytter inputdata til at lave en prædiktion, som omdannes til en prædiktionsfejl ved hjælp af tabsfunktionen. Valget af tabsfunktion kan variere betydeligt afhængig af modelleringsbehovet. Ved multinomial klassifikation benyttes *cross-entropy*, som er givet ved:

$$E(\hat{y}, y) = -\sum_{i=1}^{k} y_i \cdot log\big(P(y_i \mid x)\big) \tag{6}$$

Her er $P(y_i \mid x)$ netværkets prædikterede sandsynlighed for, at en given observation tilhører en bestemt kategori, k. y_i repræsenterer den sande observerede kategori og kan antage værdien 1 eller 0.

Log-transformationen af netværkets prædiktion giver *cross-entropy* særligt gode egenskabet til at straffe misklassificeringer, hvilket skyldes logaritmens natur. Hvis netværket fejlagtigt tildeler en lav sandsynlighed til en sand kategori, vil det resultere i en stor fejl. Dette skyldes logaritmers asymptote omkring $x = 0$, der skaber høje negative værdier for x tæt på 0.

Efter at have fuldendt et *forward pass* begynder den anden del af tovejsprocessen, back-propagation, hvor man bevæger sig baglæns gennem netværket.

7.1 Back-propagation & gradient descent

Back-propagation er processen, hvormed prædiktionsfejl sendes baglæns gennem det neurale netværk. Back-propagation virker i samarbejde med gradient descent algoritmen til at opdatere vægtene i det neurale netværk. Ved brug af kædereglen beregner back-propagation gradienten af tabsfunktionen med hensyn til hver vægt. Den begynder med at beregne gradienterne i slutningen af netværket og bevæger sig herefter baglæns til begyndelsen af netværket. Kompleksiteten af gradienter øges markant jo længere tilbage i netværket, man befinder sig. Dette skyldes den indbyrdes

afhængighed, der opstår mellem lagene i det neurale netværk samt brugen af kædereglen i udregningerne af gradienterne. Man kan opstille et generelt udtryk for den afledte af tabsfunktionen med hensyn til vægtene, således:

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial P(y \mid x)} \frac{\partial P(y \mid x)}{\partial x} \frac{\partial x}{\partial W} \tag{7}$$

Gradienterne, samt hyperparameteren λ, kaldt læringsraten, benyttes nu i optimeringsalgoritmen, gradient descent. Denne algoritme står for at opdatere vægtene. Opdateringsreglen kan opskrives således:

$$W^{(t+1)} = W^{(t)} - \lambda \frac{\partial E}{\partial W} \tag{8}$$

Her er $W^{(t)}$ vægtmatrixen til iteration t af gradient descent algoritmen. Gradienten af tabsfunktionen repræsenterer den retning, hvori den største stigning af tabsfunktionen sker. Vægtene opdateres således ved at bevæge sig i den negative retning af den største stigning i tabsfunktionen. Hermed mindskes fejlen for den specifikke vægt, hvormed ydeevnen for netværket forbedres.

Efter gradient descent algoritmen har opdateret vægtene, gentages processen med et *forward pass* som start. I løbet af træningsprocessen gennemløber netværket træningsdata flere gange. En hel gennemløbning af alt træningsdata kaldes en epoke. Typisk itereres der over flere epoker i løbet af træningsprocessen.

Træningsprocessen fortsættes indtil, et stopkriterie nås. Dette kunne for eksempel være:

- Når netværksfejlen til en given epoke er under en brugerdefineret grænseværdi.
- Når netværksfejlen mellem to fortløbende epoker ikke reduceres betydeligt.
- Når opdateringen af vægtene mellem to fortløbende epoker er under en brugerdefineret grænseværdi.

7.2 Typer af gradient descent algoritmer

Der findes mange typer af gradient descent algoritmer, som hovedsageligt afviger fra hinanden i mængden af data, der benyttes i gennemløbningen af det neurale netværk. Den mest basale gradient descent algoritme kaldes batch gradient descent. Den benytter hele træningssættet i udregningen af tabet forbundet med et *forward pass*. Da hele træningssættet benyttes, kan denne variation blive udregningsmæssig tung, og den vælges derfor sjældent som den mest optimale.

En anden variation er stokastisk gradient descent (SGD), hvor man benytter en tilfældig observation i udregningen af tabet forbundet med et *forward pass*. Da der kun benyttes en observation, løser SGD udregningsbyrden ved batch gradient descent. Dog kan SGD ofte opføre sig ustabilt for eksempel på grund af outliers i datasættet, hvormed konvergens til et minimum kan blive vanskeligt.

Ofte vælger man derfor at benytte mini-batch gradient descent. Her benytter man en delmængde af træningssættet til udregningen af tabet forbundet med et *forward pass*. Denne variation er fordelagtig, da den både løser udregningsbyrden ved batch gradient descent samt stabilitetsproblemet ved SGD. Dette gør den ved at dele træningssættet op i mindre dele, sådan at algoritmen skal iterere færre gange for at gennemgå hele træningssættet. Samtidig sørger opdelingen for, at man undgår høj usikkerhed ved enkelt-observationer.

Foruden *gradient descent* metoderne findes der også adaptive læringsrate optimerende algoritmer. Disse algoritmer finder de optimale vægte ved at ændre på læringsraten løbende, så man undgår at skulle tune den. Der findes mange adaptive algoritmer for eksempel RMSprop. Empirisk er det vist, at RMSProp er en af de adaptive algoritmer, som benyttes oftest inden for deep learning.

7.3 Initialisering af vægte i neurale netværk

Som nævnt begynder træningsprocessen af et neuralt netværk ved vægt-initialisering. Det betyder, at hver vægt tildeles en tilfældig startværdi, som benyttes som udgangspunkt for træningen. Valget af startværdi kan have stor betydning for de endelige vægte i det neurale netværk, tiden og ressourcerne, der benyttes i træningsprocessen.

En vigtig del af initialiseringen er, at man ikke vælger de samme værdier for alle vægte. Hvis dette var tilfældet, ville det betyde, at hver node ville modtage den samme information. Som konsekvens ville gradient-descent-algoritmen ikke være i stand til at konvergere til et minimumspunkt, hvormed netværksvægtene ikke ville blive optimale. At initialisere vægtene til forskellige værdier beskrives også som at bryde symmetri i ens vægte. Der findes en lang række initialiseringsstrategier, der formår at opfylde dette kriterie, heriblandt "tilfældig initialisering" og "Xavier initialisering".

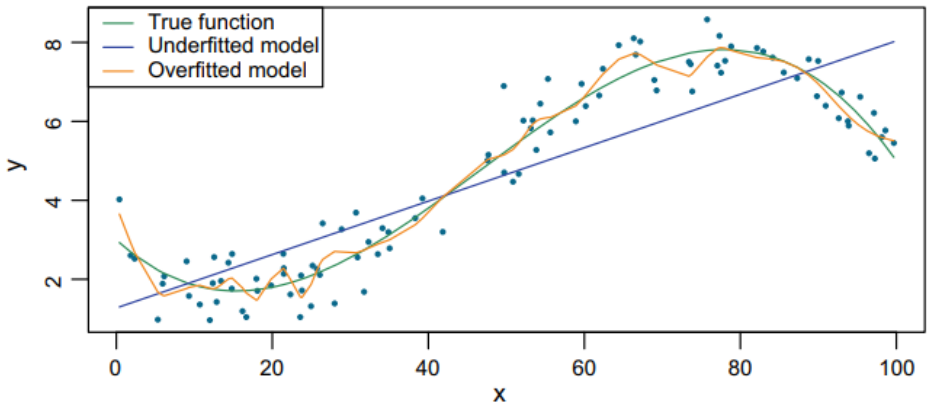
Ved tilfældig initialisering initialiseres vægte ved at trække tilfældigt fra en normalfordeling med middelværdi 0 og en endelig varians. Xavier initialisering bygger videre på dette princip og definerer variansen for normalfordelingen. Dette skyldes, at man ønsker, at variansen på tværs af netværkets lag forbliver konstant, da dette modvirker det forsvindende/eksploderende gradient problem. Det kan vises, at variansen skal sættes til $Var(W^{[l]}) = \frac{1}{n_h^{[l-1]}}$ i normalfordelingen.

8 Ulemper ved simple neurale netværk

Afsnit 8 er baseret på (Aggarwal, 2023), (Bishop, 2006) og (Ghatak, 2019).

8.1 Overfitting

Overfitting beskriver fænomenet, hvor modeller lærer støjen i data fremfor de underliggende mønstre. Inden for machine learning kan dette typisk observeres ved en væsentligt bedre ydeevne på træningsdatasættet end på testsættet. Overfitting forekommer typisk som konsekvens af en model, der er for kompleks i forhold til modelleringsbehovet. Dette resulterer i, at modellen "passer" perfekt til træningsdata, hvormed man observerer en høj varians. Da modellen passer perfekt til data, betyder det samtidig, at modellen har lav bias. Denne afvejning mellem varians og bias leder til det velkendte *bias-variance-tradeoff*, hvor man forsøger at balancere mængden af bias og varians i modellen. En model der overfitter er ikke ønskelig, da det betyder, at dens ydeevne på ny data vil være ringe, og derfor er modellen til lille nytte.



Figur 2: Den orange kurve overfitter til data og formår hermed ikke at fange det sande underliggende mønster i data (Ghatak, 2019, fig. 1.2).

8.2 Regularisering til afhjælpning af overfitting

En metode til modvirkning af overfitting er ved hjælp af regularisering. Her forsøger man at reducere kompleksiteten af modellen ved at straffe vægte, som er uhensigtsmæssigt store. Dette opnås ved at tilføje et ekstra led til tabsfunktionen, sådan at man opnår den ønskede straf. Der findes mange former for regulariseringsmetoder, dog benyttes ofte ℓ_2-regularisering i forbindelse

med neurale netværk. ℓ_2-regularisering kaldes også Ridge-regression og kan skrives op således:

$$\ell_2 = \frac{1}{2}\lambda \sum w^2$$

Denne tilføjes til tabsfunktionen, hvilket giver:

$$E(\hat{y}, y) = -\sum_{i=1}^{k} y_i \cdot log\big(P(y_i \mid x)\big) + \ell_2 \tag{9}$$

ℓ_2 tilføjer de kvadrerede vægtværdier til tabsfunktionen. Det samlede udtryk, E, benyttes som bekendt i gradient-descent algoritmen. Tilføjelsen af det ekstra strafled vil føre til, at høje vægtværdier straffes hårdere. Algoritmen vil som konsekvens foretrække mindre vægtværdier. Dette betyder, at vægte vil konvergere mod nul, medmindre deres eksistens er understøttet af data.

Udover ℓ_2-regularisering findes der også dropout-regularisering, som forklares i afsnit 10

8.3 Det forsvindende/eksploderende gradient fænomen

Det forsvindende/eksploderende gradient fænomen beskriver problemet, hvor størrelsen af gradienterne af tabsfunktionen med hensyn til vægtene i forskellige lag forhindrer gradient descent algoritmen at konvergere til et minimum. Problemet opstår som konsekvens af et netværks mange lag, kædereglens multiplikation af gradienter (ligning 7) og valget af *activation functions*.

Hvis et neuralt netværk indeholder mange lag, vil den konstante multiplikation af gradienterne kunne medføre, at de forsvinder eller eksploderer, når back-propagation algoritmen bevæger sig gennem netværket. Særligt vil vægte i starten af netværket have uendelig små/store opdateringer, hvormed de ikke konvergerer til en optimal værdi. Valget af Sigmoid-lignende *activation functions* i de skjulte lag forstærker problemet yderligere, da værdien af deres gradienter ikke overstiger 1. Problemet kan således forklares ved, at den konstante multiplikation af gradienter, vis værdier er under/over 1, vil lede til henholdsvis uendelige små eller uendelige store værdier. Dette forhindrer gradient descent algoritmen i at konvergere til et minimum, og dermed opnår man ikke optimale vægtværdier.

8.4 Mangel på hukommelse

Simple neurale netværk lider desuden af ikke have indbygget et element af hukommelse. Det betyder, at hvis man står overfor et problem med tidsafhængige data, bliver modellen bygget med

antagelsen om, at datapunkt t ikke afhænger af datapunkt $t-1$. Dette resulterer i, at simple neurale netværk præsterer dårligt ved sådanne dataformater. For at kunne anvende neurale netværk til tidsafhængige data skal modellen typisk udbygges, så den får en hukommelsesegenskab. Hukommelsen bliver inkorporeret i modellen ved at tillade, at den deler vægte igennem de skjulte lag.

9 Rekursive neurale netværk (RNN)

Afsnit 9 er baseret på (Aggarwal, 2023), (Bianchi m.fl., 2017) og (Ghatak, 2019).

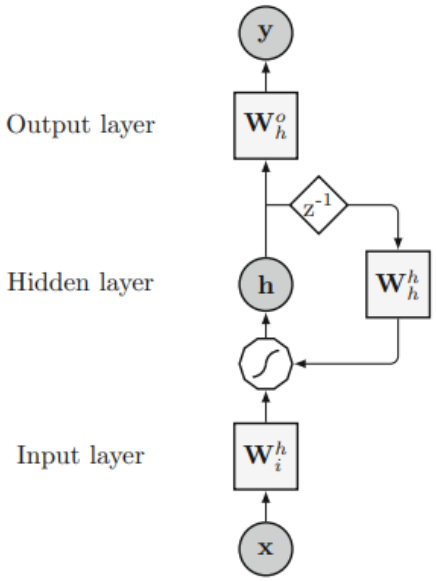
Som nævnt i forrige afsnit lider simple neurale netværk af manglen på indbygget hukommelse. En måde hvorpå man kan imødekomme dette, er ved hjælp af rekursive neurale netværk (RNN). Sådanne netværk indbygger et element af hukommelse ved at dele vægtmatrix gennem tidsintervaller. Dette gør dem i stand til, ikke blot at kunne lære fra hvad der lige er sket, men også at tage højde for længere tidseffekter, hvilket gør dem optimale til behandling af tidsseriedata.

9.1 Arkitekturen bag RNN

Et simpelt RNN består, ligesom det simple neurale netværk, af et inputlag, et eller flere skjulte lag samt et outputlag. Det der adskiller det simple RNN fra det simple neurale netværk, er det rekursive element. Figur 3 viser, hvordan RNN adskiller sig fra simple neurale netværk i form af at benytte outputtet fra et tidspunkt til det næste. Dog bliver det også tydeligt, hvordan et RNN ligner et simpelt neuralt netværk ved at udfolde netværket i længden af tidsintervaller, hvilket kan ses på figur 4. På den måde skabes der tilnærmelsesvis den samme struktur som ved et simpelt neuralt netværk, hvilket gør, at man kan benytte de velkendte træningsalgoritmer som tidligere beskrevet. I praksis udfolder man ikke netværket i den totale længde af tidsintervallet, men derimod afkortes det. Dette skyldes ønsket om at minimere risikoen for det forsvindende/eksploderende gradient problem.

9.2 Træning af RNN

Træningen af et RNN foregår i store træk på samme måde som ved et simpelt neuralt netværk. Dog afviger træningen af et RNN på den måde, at man skal tage højde for, at vægtmatrixen bliver delt mellem tidslag. Dette gør man ved først at udfolde det rekursive netværk, hvorefter man benytter standard back-propagation og gradient descent til at opdatere vægtmatricerne under antagelsen af, at de er unikke. Efter færdiggørelsen af standard back-propagation benyttes princippet om, at

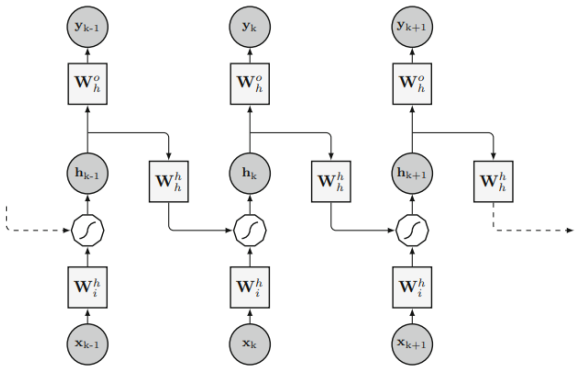


Figur 3: Arkitektur af RNN. Fra (Bianchi m.fl., 2017, fig. 2.1).

de individuelle bidrag fra hver vægtmatrix kan summeres sådan, at man opnår et samlet bidrag til gradient descent algoritmen. Denne variantion af back-propagation kaldes back-propagation gennem tid (*Back-Propagation Through Time (BPTT)*), da den tager højde for det temporale element ved netværket.

Udover variationen af den klassiske back-propagation algoritme, laves der endnu en tilføjes til BPTT, som kaldes *Truncated Backpropagration Through Time (TBTT)*. Denne metode, benytter mindre dele af tidsserien, ligesom mini-batch gradient descent, således at den beregningsmæssige byrde lettes, og dermed sikres der hurtigere konvergens. Hér skal det pointeres, at på grund af den naturlige afhængighed mellem observationer i tidsserien, skal der vælges batches, der ligger i forlængelse af hinanden. På den måde bibeholder man afhængigheden mellem observationerne.

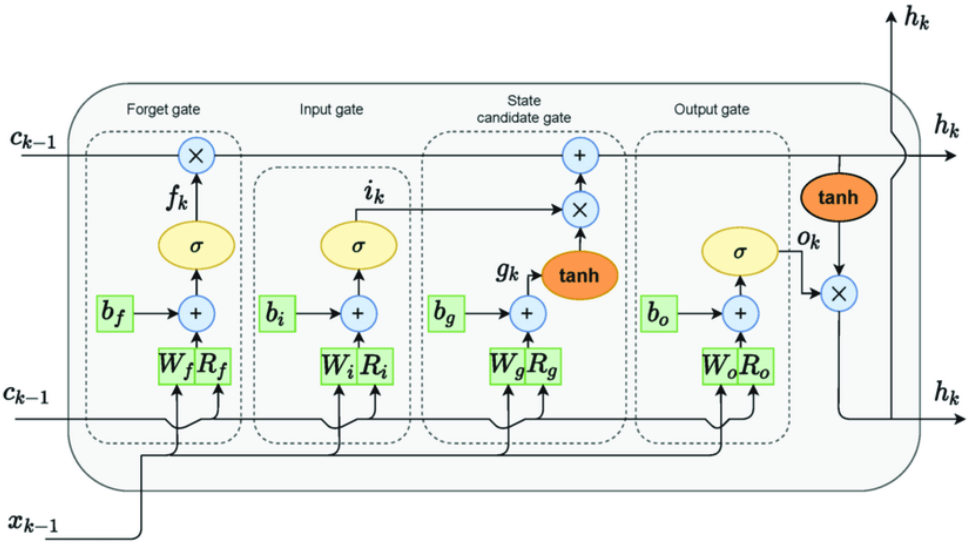
Som nævnt benyttes RNN typisk, når man forsøger at modellere længere tidsperioder. Dog vil man ofte opleve, at jo længere tidsperioden bliver, desto mere sandsynligt er det forsvindende/eksploderende gradient problem. RNN er derfor ikke nyttige på tidsserier, hvor man også vil fange langtidseffekter. Til dette blev Long Short-Term Memory (LSTM) neurale netværk konstrueret.



Figur 4: Udfoldet RNN. Fra (Bianchi m.fl., 2017, fig. 2.2).

9.3 Opbygning af Long Short-Term Memory (LSTM) neurale netværk

Et LSTM neuralt netværk er grundlæggende opbygget på samme vis som det rekursive neurale netværk. Afvigelsen fra RNN sker dog i aktiveringen af de forskellige noder i netværket. Hvor man i det simple RNN har en enkelt ikke-lineær komponent, udtrykt ved en *activation function* $g(\cdot)$, benytter man i LSTM fem forskellige ikke-lineære komponenter. Samlingen af komponenterne kalder man en celle. Cellerne er tekniske i deres opbygning, idet de består af følgende tre porte, *forget gate*, *update gate* og *output gate*. Dertil kommer *candidate state* samt *cell state*, som er vigtige i at holde på både lang- og korttidshukommelse. På grund af cellens kompleksitet beskrives kun de overordnede funktioner ved de enkelte komponenter, og ikke de matematiske argumenter.



Figur 5: Illustration af en celle i et LSTM neuralt netværk (Brand m.fl., 2022, fig. 3).

Cell state fungerer som langtidshukommelse for netværket. Den opdateres hver gang, data passerer igennem en LSTM-celle via *forget gaten* og *candidate state/gaten*. *Forget gaten* bestemmer hvor stor vægt, der lægges på netværkets nuværende langtidshukommelse. *Candidate state/gaten* har til opgave at opdatere netværkets langtidshukommelse. Den bestemmer altså hvor meget af korttidsinformationen, der skal tilføjes til langtidshukommelsen. Til sidst udregner *output gaten* netværkets prædiktion. Prædiktionen benyttes i de næste celler samt i træningen af netværket i form af prædiktionsfejl. Samlet set formår LSTM netværk at reducere risikoen for det forsvindende/eksploderende gradient problem ved dynamisk at opdatere netværkets langtidshukommelse. Dette gør dem i stand til at modellere længere tidsperioder og dermed også fange langtidseffekter.

10 Hyperparametre

Afsnit 10 er baseret på (Aggarwal, 2023), (Brownlee, 2021), (Bronwlee, 2022), (Chowdhury, 2021), (Ghatak, 2019), (Goodfellow m.fl., 2016), (Gupta, 2023) og (Shen, 2018).

Der findes to typer af parametre indenfor machine learning algoritmer. Den ene type af parametre er vægtene mellem lagene, som er med til at minimimere tabet i tabsfunktionen (afsnit 7). Denne type af parametre bliver optimeret af algoritmen, og brugeren kan derfor ikke direkte bestemme slut-værdien. Den anden type parametre er hyperparametre. Hyperparametre er parametre, som brugeren kan bestemme og ændre på for at tune algoritmen. Der findes mange forskellige typer af hyperparametre, og sommetider kan et specifikt valg af en hyperparameter medføre, at der kommer flere hyperparametre, der skal tunes. Et eksempel på dette er, når man vælger RMSProp som *optimizer*. Her bliver *decay*-faktoren, ρ, tilføjet, som er en ny hyperparameter. En vigtig ting at have i baghovedet, når man arbejder med neurale netværk, er, at "Neurale netværk kan nogle gange virke meget godt ved kun at benytte et lille antal tunede hyperparametre, men ofte bliver netværket markant bedre ved at tune 40 eller flere hyperparametre"(Goodfellow m.fl., 2016, s.427, vores oversættelse). Det er dermed vigtigt at tune på netværkene.

10.1 Typer af hyperparametre

Activation functions, som blev introduceret i afsnit 6, er også en hyperparameter. Det er kun i de skjulte lag, hvor man tester forskellige *activation functions*. Dette skyldes, at valget af *activation function* i outputlaget er bestemt af modelleringsbehovet. I praksis benyttes ReLU ved ANN og tanh til RNN og LSTM i de skjulte lag.

Antal lag er ofte med til at øge modellens evne til at fange komplekse sammenhænge i data. Man

ser i praksis, at ét skjult lag kan opnå en god præcision til simple problemer, og med 2 eller flere lag får man en relativ god præcision på selv komplekse problemer.

Antal noder i hvert enkelt lag af netværket kan også defineres af brugeren. Et stort antal noder i et lag vil medføre, at der vil være flere vægte at optimere for netværket, da hver node i ét lag skal forbindes til hver node i det næste lag. Flere vægte er med til at forbedre muligheden for at opfange de underliggende mønstre i data for enhver statistisk model men kan samtidigt skabe overfitting på grund af modellens øgede kompleksitet.

Batch-størrelse er den delmængde af træningssættet, som bliver brugt i henholdsvis *forward pass* og back-propagation. Når der vælges værdier af batch-størrelser, skal man vælge værdier såsom $16, 32, 64, 128, 256, ...$, hvilket er værdier af 2^n. Dette skyldes computerens arkitektur. Når batch-størrelsen stiger, vil man ofte se, at præcisionen falder for netværket. Det betyder, at det kan svare sig at have små værdier i batch-størrelsen.

Dropout-raten beskriver, hvor meget modellen skal huske fra det ene skjulte lag til det næste. Værdien er et tal i intervallet $[0, 1]$, som beskriver sandsynligheden for, at en node vil blive ignoreret under træningen. Dette er en nem måde at gardere sig mod, at modellen overfitter, da man træner netværket ved at "slukke" og "tænde" noder. Da noder er forbundet med vægte, vil dét at slukke individuelle noder også slukke vægte, hvormed man kan regularisere sit netværk.

Læringsraten bestemmer hvor stor en indflydelse gradienten har på den nye opdaterede vægtmatrix i ligning 8. Dette kan tolkes som, at læringsraten bestemmer, hvor store skridt gradient descent algoritmen tager mod det lokale minima, og dermed hvor hurtigt man konvergerer til et minimumspunkt. En for stor eller for lille læringsrate kan medføre, at man ikke kan finde et minimum. For store skridt kan medføre, at man springer forbi minimummet, hvorimod for små skridt kan medføre, at man aldrig når til minimummet.

10.2 Tuning af hyperparametre

Grid search benyttes i tuningen af hyperparametrene. Grid search benytter alle kombinationer af brugerdefinerede hyperparameterværdier og danner et net (*grid*), hvor alle punkterne i nettet er kombinationerne af hyperparameterværdier. Herefter tester den alle punkterne i nettet og finder den bedste kombination af hyperparametre. Denne fremgangsmåde vil medføre, at antallet af kombinationer stiger eksponentielt. Grid search benyttes, når der tunes på tre eller færre hyperparametre.

Random search vælger indledningsvis et tilfældigt punkt (punkt 1) i rummet udspændt af hyperparametrene og udregner tabet. Derefter vælges et nyt tilfældigt punkt (punkt 2) i en given radius af punkt 1, hvorefter tabet udregnes. Hvis tabet i punkt 2 er større end i punkt 1, går algoritmen tilbage til punkt 1. Hvis det derimod er bedre, findes et nyt tilfældigt punkt (punkt 3) i en given radius af punkt 2. Algoritmen fortsætter på denne måde, indtil antallet af brugerdefineret iterationer er opbrugt. Random search konvergerer hurtigere mod gode hyperparameterværdier end grid search, hvilket gør den mere tidseffektivt. Dog undersøger den ikke alle værdier, hvorfor det er muligt, at den kan overse det bedste sæt af hyperparametre. Derudover udviser random search bedre resultater end grid search, når antallet af hyperparametre, der skal tunes, er stort.

11 *Local Interpretable Model-agnostic Explanaitions Technique* (LIME)

Afsnit 11 er baseret på (Ribeiro m.fl., 2016).

LIME bygger på antagelsen om, at alle komplekse modeller kan modelleres pålideligt og stabilt i en lokal omegn af en prædiktion. Det vil sige, man er i stand til at danne en surrogatmodel, som omkring en enkelt prædiktion af den komplekse model kan fortolkes pålideligt ved hjælp af inputvariablene til den oprindelige model.

Der skal gælde, at inputvariablene til den nye model er fortolkbare, altså skal en bruger være i stand til at forstå værdien, de antager. Dette gøres ofte ved at transformere tidligere komplekse (for eksempel numerisk) inputvariable om til binære inputvariable, sådan at fortolkningen baserer sig på tærskler af inputvariablene fremfor den eksakte værdi.

Når disse antagelser er opfyldt, er forklaringen af en prædiktion, $\xi(x)$, blot et minimeringsproblem givet ved

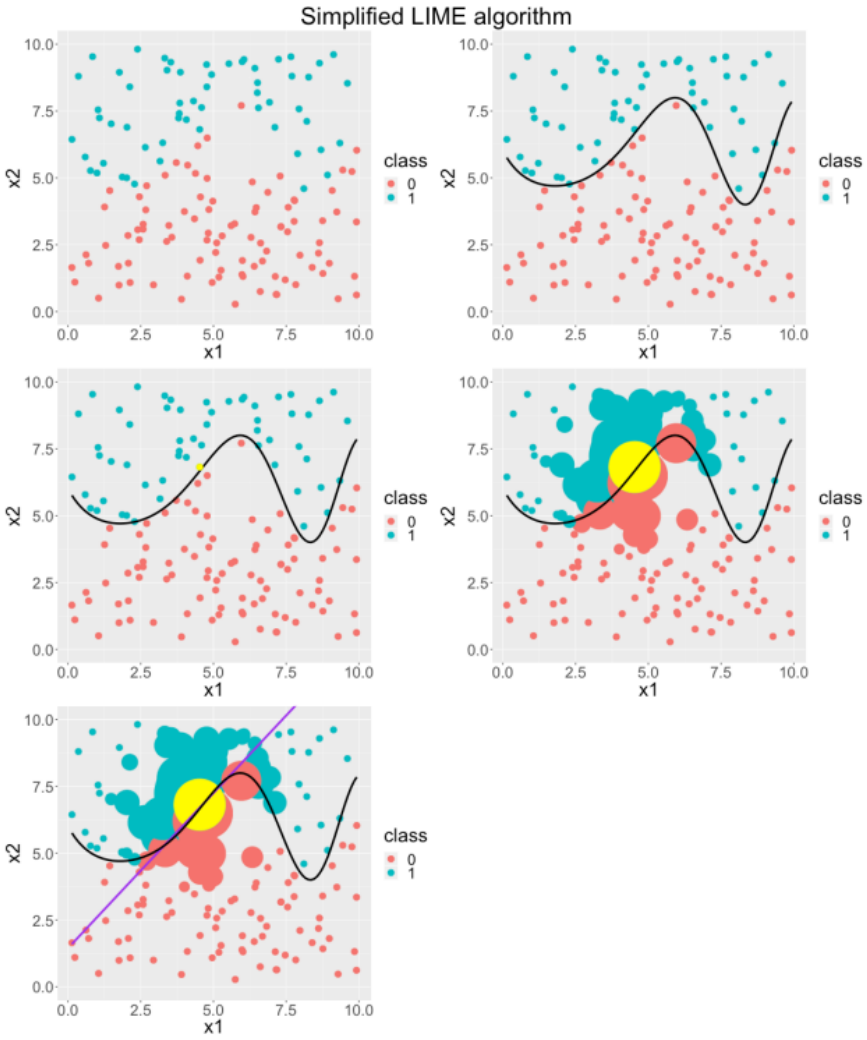
$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \; \mathcal{L}\left(f, g, \pi_x\right) + \Omega(g) \tag{10}$$

Der gælder, at man skal finde en model $g \in G$, hvor G er mængden af mulige modeller i en familie, for eksempel lineære eller logiske modeller. Da man skal kunne fortolke på den lokale model, inkluderes $\Omega(g)$ som et mål for kompleksiteten af modellen g. For eksempel øges kompleksiteten af en lineær model med mængden af kovariater. En lineær model med mange kovariater er således høj i kompleksitet men kan som konsekvens lide af nedsat mulighed for fortolkning. Der opstår altså en afvejning mellem kompleksitet og forklarlighed af surrogatmodellen. f er den komplekse

model, man forsøger at forklare med valget af g.

Da man forsøger at modellere én prædiktion, x, af den komplekse model, er det nødvendigt at danne nye observationer. LIME danner nye observationer omkring x ved at trække tilfældigt fra en uniform fordeling. De nye observationer vægtes ved π_x i surrogatmodellen. Her er π_x et mål for afstanden mellem en observation og prædiktionen x. Der lægges på den måde højere vægt på observationer, der er tæt på x fremfor dem, der befinder sig langt fra x.

LIME-processen kan illustreres som på figur 6. Her bygges først en binær klassifikationsmodel. Herefter forsøger man at forklare det gule punkt ved at danne en lokal lineær model baseret på vægtede observationer i omegnen af punktet.



Figur 6: En simpel binær klassifikationsmodel (sort linje) forsøger at blive forklaret ved hjælp af LIME (lilla linje) i en omegn af det gule punkt (Molnar, 2020, fig. 12.1).

12 Analyse

12.1 Databehandling

Som en indledning til analysen giver vi en gennemgang af vores metodik i forbindelse med databehandling. Denne kommer i forbindelse med analysen, da datagrundlaget for modellerne er essentielt i forståelsen af deres ydeevne.

Som nævnt i afsnit 4 dækker vores data følgende variable:

Variable
SHELL - OPEN
SHELL - MACD
SHELL - RSI
CHEVRON - OPEN
BP - OPEN
EXXON - OPEN
S&P GSCI Total Return - OPEN

Tabel 1: Der benyttes 1 responsvariable, SHELL - OPEN, samt 6 inputvariable i modelleringsprocessen.

Her repræsenterer SHELL - OPEN aktiekursen for Shell-aktien. Den vil fungere som vores responsvariabel i vores neurale netværk. De resterende 6 variable vil fungere som inputvariable.

For at tilpasse data til vores modelleringsbehov laver vi en række transformationer af data. Først og fremmest er hver variabel omdannet til procent-ændring, sådan at vi fanger relationen mellem ændringer i inputvariablene og responsvariablen. Det vil sige, at hver observation af de overstående variable symboliserer ændringen i åbningsværdi fra den ene dag til den næste.

Det skal desuden nævnes, at "SHELL-MACD" variablen symboliserer ændringen i forskellen i to eksponentielt glidende gennemsnit (*exponential moving average (EMA)*) henholdsvis 12-dags EMA og 26-dags EMA. Hvis 12-dags EMA-værdien er større end 26-dags EMA-værdien, betyder det, at en aktie har positivt momentum. Tilsvarende har en aktie negativt momentum, hvis det

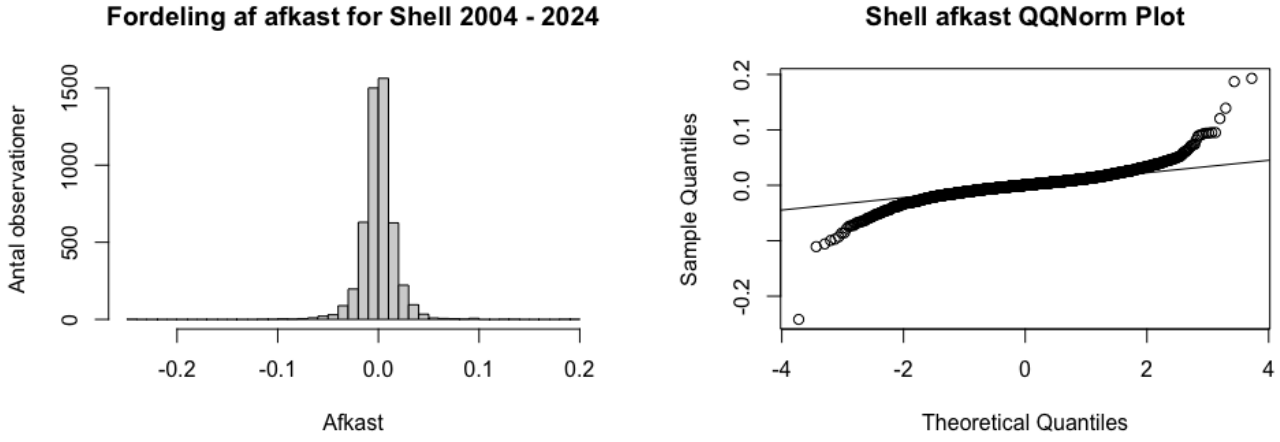
modsatte er tilfældet. At tage procent-ændringen af forskellen i disse to EMA'er fanger derfor den inkrementelle ændring i momentum for aktien.

Vores data strækker sig fra 1. marts 2004 til 28. februar 2024, hvilket er en periode på 20 år. I vores dataindsamlingsproces var det muligt at indsamle data tilbage til 1980. Dog var data fra denne periode inkonsistent med en lang række manglende observationer. Da data er det essentielle grundlag for machine learning, blev data før 1. marts 2004 derfor frasorteret.

NA-værdier blev renset fra data i forbindelse med databehandlingen. Disse værdier optrådte primært på børs-helligdage på både den amerikanske, hollandske og engelske børs, hvor de forskellige olieselskabers aktier handles.

Efter at have renset data og transformeret det til procent-ændring skal det kategoriseres på en sådan måde, at det kan benyttes til multinominal klassifikation.

På grund af den måde aktieafkast fordeler sig på, skal opdelingen behandles med forsigtighed. Det er alment kendt, at aktieafkast har en tendens til at fordele sig omtrent efter en normalfordeling på lang sigt. Dette var også tilfældet for Shells aktieafkast over de sidste 20 år, hvilket ses på figur 7. Fordelingen er omtrent normal dog med tunge haler som konsekvens af dage, der har haft store udsving i aktieprisen. Da en stor del af afkastene koncentrerer sig omkring 0%, kan der forekomme risiko for at danne et ubalanceret datasæt ved opdelingen af afkast i kategorier.



(a) Histogram af Shells aktieafkast

(b) QQplot for Shells aktieafkast

Figur 7: Histogram og QQplot for Shell-aktien illustrerer at aktieafkast er omtrent normalfordelt over længere sigt.

Laver man for eksempel en inddeling af kategorier som i tabel 2a, ses det på figur 8a og tabel 3a at

Kategori	Afkast (r)
0	$r \leq -3\%$
1	$-3\% < r \leq -1.5\%$
2	$-1.5\% < r \leq 0\%$
3	$0\% < r \leq 1.5\%$
4	$1.5\% < r \leq 3\%$
5	$r > 3\%$

(a) Før ændring af grænser

Kategori	Afkast (r)
0	$r \leq -1.5\%$
1	$-1.5\% < r \leq -0.5\%$
2	$-0.5\% < r \leq 0\%$
3	$0\% < r \leq 0.5\%$
4	$0.5\% < r \leq 1.5\%$
5	$r > 1.5\%$

(b) Efter ændring af grænser

Tabel 2: Grænseværdier for kategorisering af aktieafkast henholdsvis før og efter ændringen af grænseværdierne.

Kategori	Andel
0	2.44%
1	8.07%
2	39.03%
3	40.55%
4	7.54%
5	2.37%

(a) Den første kategorisering

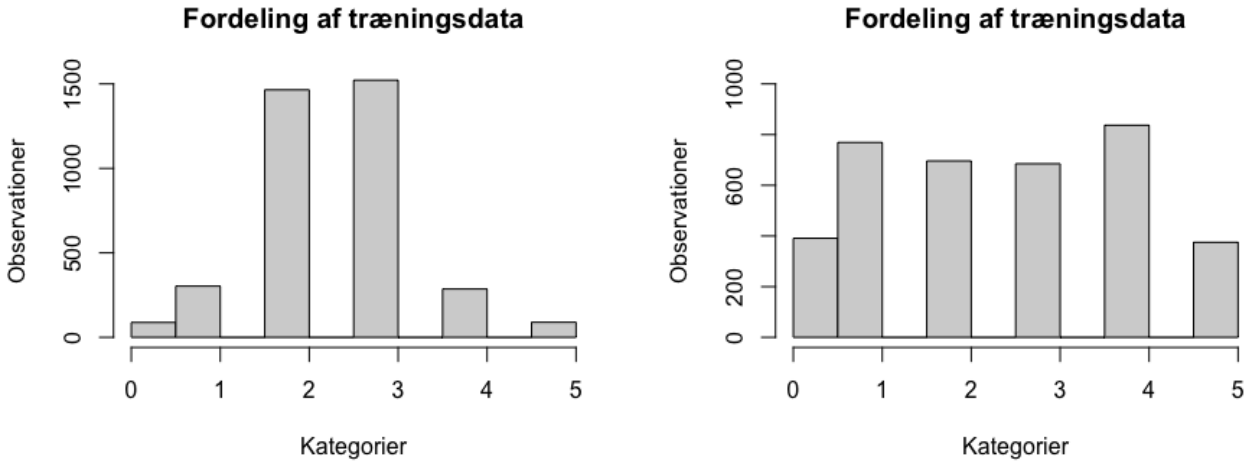
Kategori	Andel
0	10.40%
1	20.49%
2	18.54%
3	18.27%
4	22.3%
5	10%

(b) Den anden kategorisering

Tabel 3: Den første kategorisering (a) giver anledning til et ubalanceret datasæt, karakteriseret ved to dominante kategorier, 2 og 3. Den anden kategorisering (b) viser ikke tegn på tydelige dominante kategorier.

størstedelen af afkastene vil ende i kategori 2 og 3. Disse to kategorier benævnes som dominante kategorier, mens de resterende benævnes som minoritets kategorier. At have ubalance i sit data er uhensigtsmæssigt, da det betyder, at modellen primært bliver eksponeret til de dominante kategorier i træningsprocessen. Som konsekvens vil modellen have en tendens til at lægge for meget vægt på dem og dermed primært prædiktere dem (Canuma, 2023). Dette er ikke attraktivt, da vi ønsker en model, der er nuanceret i dens prædiktioner.

For at afhjælpe ubalancen mellem kategorierne skal vi justere på grænseværdierne, der definerer kategorierne. Inddeler vi i stedet afkastene som i tabel 2b, ser vi på figur 8b og tabel 3b, at der ikke længere er nogen tydelig dominante kategorier. Dette er positivt, da modellen nu vil blive eksponeret for et mere nuanceret billede af kategorier i træningsfasen. Dermed vil variationen i prædiktioner også blive større, hvilket vil forbedre modellens ydeevne.



(a) Den initiale opdeling af klasser

(b) Den nye opdeling af klasser

Figur 8: (a) viser, at den initiale opdeling af aktieafkast leder til ubalancerede klasser karakteriseret ved dominans af kategori 2 og 3. (b) viser, at den nye opdeling af klasser ledte til et mere balanceret datasæt, hvor der ikke er samme grad af dominans af kategori 2 og 3.

Samlet set betyder det, at vi har været i stand til at afhjælpe problemet ved ubalancerede kategorier ved blot at ændre, hvordan vi inddeler vores aktieafkast. Dette er en simpel metode, men vigtig, da balancen mellem kategorier vil have stor betydning for vores models ydeevne.

Efter vi har transformeret vores responsvariabel til procent-ændring og kategoriseret afkastene efter grænseværdierne i tabel 2b ender vi med én søjle, der indeholder de seks potentielle afkastkategorier fra 0 til 5. Dog skal vi lave en yderligere transformation af vores responsvariabel for at kunne benytte den i vores model. Dette kaldes *one-hot-encoding* (OHE). OHE består af processen hvor den oprindelige søjle transformeres til 6 søjler, der indeholder 0'er og 1'er. Hver ny søjle repræsenterer således én kategori. Hvis den pågældende afkastkategori var observeret i den oprindelige søjle, ville den tilhørende nye søjle antage værdien 1 og 0 ellers. OHE transformationen illustreres på figur 9.

Transformationen er nødvendig, da modellens output kommer til at være sandsynligheden for, at en given observation tilhører en af de seks kategorier.

Afslutningsvis skal det nævnes, at de overstående transformationer med kategorisering og OHE også benyttes på inputvariablene. Det betyder, at vi udvider vores inputdata fra at have 6 søjler til nu at have 36 søjler. Hver af de 6 inputvariable, som defineret i tabel 1, kan altså antage 6 forskellige kategorier repræsenteret ved OHE. Denne transformation er nødvendig, da det vil forbedre mulighederne for fortolkning af betydningsfulde variable ved brug af LIME i afsnit 12.5.

Traditionel opsætning		One-Hot-Encoding af kategorier					
Kategori - observation		**Kategori 0**	**Kategori 1**	**Kategori 2**	**Kategori 3**	**Kategori 4**	**Kategori 5**
0		1	0	0	0	0	0
4		0	0	0	0	1	0
5		0	0	0	0	0	1
2		0	0	1	0	0	0
3		0	0	0	1	0	0
1		0	1	0	0	0	0

Figur 9: Den oprindelige søjle med observationer af afkastkategorier transformeres ved hjælp af OHE, således at hver kategorier har en søjle, der kan antage værdierne 1 eller 0.

12.2 Opbygning af ANN, RNN og LSTM som basismodeller

Før vi tuner vores modeller og laver fortolkningsanalyser af de forskellige variable, vil vi teste hvilket slags neuralt netværk, der resulterer i den højeste præcision. Vi starter derfor med at bygge tre simple netværk, henholdsvis et ANN, et RNN og et LSTM-netværk.

Vi bygger alle modeller op med et inputlag. Dertil bruger alle netværkerne Xavier initialisering som initialiseringsmetode. I ANN netværket benytter vi en ReLU *activation function* i det skjulte lag og i RNN og LSTM-netværkene benytter vi tanh som *activation function* i det skjulte lag. Vi benytter disse activation functions, da dette er normal praksis for disse netværkstyper. I alle netværkene benytter vi en softmax *activation function* i outputlaget, da denne bruges til multinominal klassifikation. Vi vælger disse hyperparamtre med grundlag i teorien fra tidligere afsnit. Dertil vælger vi at have 3 noder i det skjulte lag, en batch-størrelse på 128 og 50 epoker at træne over.

Vi vælger `RMSProp`-optimizeren, da denne normalt benyttes. Vi sætter derefter vores tabsfunktion til at være *cross-entropy*, da denne benyttes til multinominal klassifikation som forklaret i afsnit 7. Vi bruger *categorical accuracy* som mål for vores modellers ydeevne. *Categorical accuracy* vil fremadrettet benævnes "præcision". Her defineres præcision som andelen af korrekt klassificerede kategorier ud af den samlede mængde af kategorier i testdata. Vi træner vores model på træningsdata og evaluerer det på testdata for at opnå en præcision. Resultaterne af de ikke-tunede netværk kan ses i tabel 4. Her ses det tydeligt, at vores simple neurale netværk præsterer bedst med en præcision på 47%. En ting, som er værd at lægge mærke til, er, at vores RNN og LSTM-netværk ikke er særlig gode til at prædiktere den korrekte kategori. Indledningsvis viser basismodellerne, at det simple neurale netværk præsterer bedst. Som bekendt kræver neurale netværk dog typisk

Netværkstype	Tab	Præcision
ANN	1.31	47%
RNN	1.63	29%
LSTM	1.83	21%

Tabel 4: Resultater af kørsel med ikke-tunet hyperparametre for de tre netværkstyper.

tuning for at opnå deres fulde potentiale (afsnit 10).

12.3 Tuning af hyperparametre i de tre netværkstyper

Efter at have opbygget basismodellerne og opnået en præcision for hver kan vi nu begynde at tune hyperparametrene i de individuelle netværk. Dette tillader os, at se om RNN og LSTM-netværkene kan opnå samme præcision, som det simple neurale netværk kan. Ved tuningen af hyperparametrene benytter vi *grid search* (afsnit 10.2), hvor alle kombinationer af hyperparametre testes. Da mængden af kombinationer øges eksponentielt med antallet af hyperparametre, starter vi med batch-størrelse og antal noder. Når vi har tunet de tre netværk på disse hyperparametre, vil vi udvælge netværket med højest præcision og tune videre på dette.

Hyperparameter valg

Vi har valgt værdierne 16, 32, 64 og 128 for batch-størrelse. Dertil varierer vi over fire forskellige antal noder i det skjulte lag henholdsvis 5, 10, 15 og 20. Dette producerer 16 forskellige par af hyperparametre, som skal testes i de tre netværkstyper.

Resultater af basismodeller efter initial hyperparameter-tuning

Efter tuningen ses det i tabel 5, at præcisionerne for hver netværkstype er omtrent ens. Vi ser

Netværkstype	Antal noder	Batch-størrelse	Præcision
ANN	10	64	50.2%
RNN	15	16	50.1%
LSTM	20	16	47.40%

Tabel 5: Efter hyperparameter-tuning har de tre netværkstyper omtrent ens præcision.

fortsat, at det simple neurale netværk har den højeste præcision med 50.2%. Samtidigt ser vi også, at alle tre netværk har en præcision omkring 50%. Dette er ikke umiddelbart højt, dog er det væsentligt bedre end det "naive" gæt. Det naive gæt benyttes ofte som en minimumsgrænse for en

acceptabel præcision i et neuralt netværk. I dette tilfælde er det naive gæt at vælge en af de seks kategorier tilfældigt, hvilket har en succesrate på $1/6 \approx 16.66\%$. Hvis ikke modellerne havde haft højere præcision end dette, ville de være overflødige, og tilfældigt at gætte på dagens aktieafkast ville virke som en bedre metode. Det betyder, at vores modeller, på trods af kun at have præcision omkring 50%, stadig har værdi.

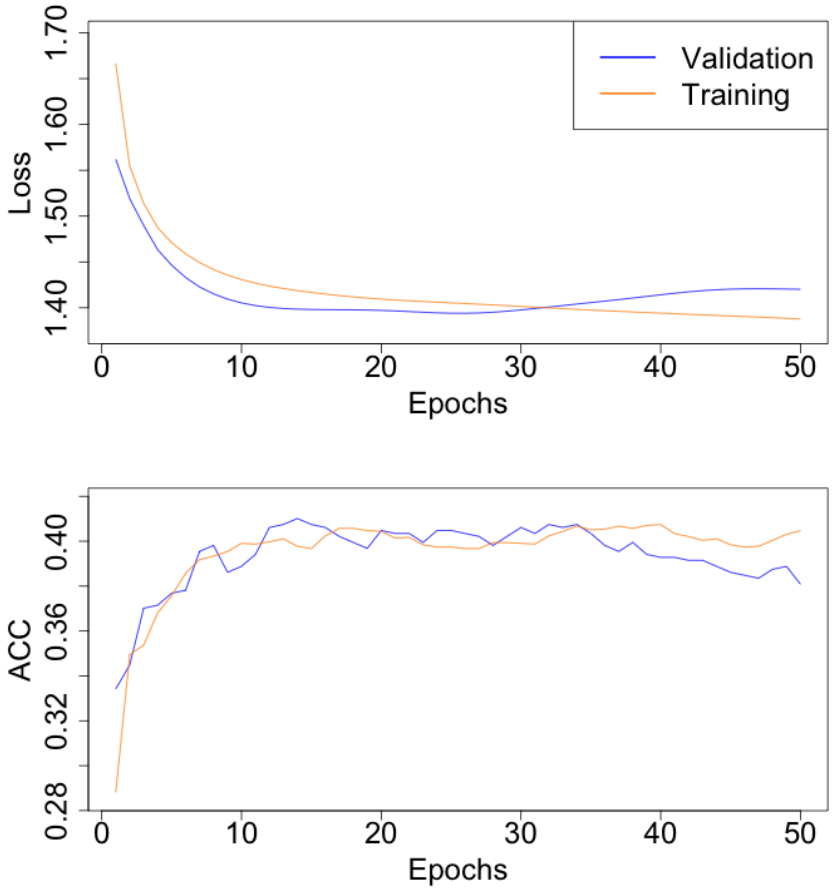
Ser man nærmere på resultaterne for de enkelte netværk, er det også interessant at observere, at når antallet af noder holdes konstant, er der en tendens til faldende præcision ved større batches. Dette kan observeres ud fra resultattabellerne i bilag A.1 eller figurerne i bilag B.1. Denne tendens er primært tydelig i RNN og LSTM-netværkene, hvor ANN-netværket kun udviser det i et af tilfældene. Tendensen giver god mening i forhold til teorien om batch-størrelse (afsnit 10.1). En mindre batch-størrelse tillader modellen at "se" flere individuelle observationer, hvormed den bedre kan lære nuancerne i data. Ligeledes vil en større batch-størrelse betyde, at netværket ikke fanger de samme underliggende strukturer i data, og på den måde ikke er i stand til at identificere de enkelte kategorier korrekt. Ulempen ved mindre batch-størrelser er som bekendt, at træningstiden for netværket øges markant. Dog spillede det en mindre rolle i dette tilfælde, da datamængden er forholdsvis begrænset og derfor var forskellen i træningstid på tværs af batch-størrelser et spørgsmål om sekunder.

Udover at præcisionen er faldende i batch-størrelsen, ser vi også en tendens til, at præcisionen er stigende, jo større antallet af noder er i det skjulte lag. Denne tendens er igen primært tydelig i RNN og LSTM-netværkene. Dette giver også god mening i forhold til teorien om noder i et neuralt netværk. En større mængde af noder i inputlaget vil medføre, at der vil være flere vægte at optimere for netværket. Flere vægte er, som nævnt i afsnit 10.1, med til at øge muligheden for at opfange de underliggende mønstre i data. Det stemmer derfor godt overens med, at vi ser, at præcisionen stiger, når vi øger antallet af noder i det skjulte lag.

Som bekendt kan et større antal vægte dog også lede til overfitting, som beskrevet i afsnit 8.1. Det betyder, at modellen har sværere ved at generalisere fra træningsdata til testdata, hvilket medfører lavere præcision. Tegn på overfitting forekommer også i en af varianterne af modellerne, hvilket illustreres på figur 10. For RNN-netværket med 20 noder i det skjulte lag og en batch-størrelse på 16 er det tydeligt, at der forekommer overfitting. Efter omkring 25 epoker ses det, hvordan valideringstabet (*loss*) begynder at stige, samtidigt med træningstabet fortsat falder. Dette indikerer, at modellen overfitter til træningsdata og derfor generaliserer dårligt. Den forventede lavere præcision (ACC) bekræftes også i valideringskurven for præcision, der topper omkring 25 epoker og begynder at falde lidt efter 30 epoker. Hér ses det også, hvordan valideringstabet overstiger

træningstabet, hvilket opstår som konsekvens af overfittingen.

Samlet set observerer vi, at netværkene falder i præcision med en større batch-størrelse, hvilket er forventeligt i forhold til teori. Samtidigt observerer vi, at et større antal noder leder til højere præcision, hvilket igen er i overensstemmelse med teori. Dog kan antallet af noder ikke øges uendeligt, da det leder til overfitting.



Figur 10: Efter 25 epoker begynder RNN-netværket at overfitte til data, hvilket resulterer i lavere præcision.

12.4 Yderligere tuning af ANN

Da det simple neurale netværk har højest præcision, vælger vi at tune videre på dette, for at undersøge om præcisionen kan forbedres yderligere. Vi fortsætter med at benytte *grid search* til at finde den kombination af hyperparametre, der giver den højeste præcision. Vi vælger at tune på noder, batch-størrelsen, læringsraten, dropout-rate og antal lag i modellen. Antallet af noder i

tuningen ændres til 10, 15, 20 og 25. Vi benytter de samme batch-størrelser som i den forrige tuning. Som det nye har vi tilføjet både læringsrate, dropout-rate og antal skjulte lag som hyperparametre. Vi har valgt de følgende læringsrate værdier til vores *grid search*, 0.0001, 0.0026, 0.0051 og 0.0076. Dertil har vi valgt vores dropout-rate værdier til at være henholdsvis 0.2 og 0.4. Endeligt har vi valgt at teste vores ANN med henholdsvis 1, 2 og 3 skjulte lag.

Når vi vælger at tune på 5 hyperparametre med flere forskellige værdier, vil der opstå mange kombinationer. Den præsenterede tuning producerer $4 \cdot 4 \cdot 4 \cdot 2 \cdot 3 = 384$ forskellige kombinationer af hyperparametre og dermed lige så mange modeller, der skal testes. Efter at have kørt disse modeller vælger vi derfor at sortere resultaterne efter præcision og kun undersøge de 5 bedste.

Ud fra tabel 6 ses det, at det ikke er lykkedes at få tunet vores model bedre end vores basismodel. Den højeste præcision, 49.57% blev opnået med 1 lag, 10 noder, en batch-størrelse på 64, en læringsrate på 0.0051 og en dropout-rate på 0.2.

En ting, som er værd at lægge mærke til, er, at det kun er modeller med 1 lag, som er i topfem for præcision. Dette kan muligvis skyldes, at vi får for mange vægte i vores netværk, når vi har flere lag. Dette vil resultere i, at modellen overfitter til data, hvilket giver en lavere præcision på testdata. Så i dette tilfælde kan det ikke svare sig at bygge mere komplicerede netværk. Derudover er det værd at bemærke at blandt de 384 forskellige ANN netværk, har netværket med højest præcision præcis den samme kombination af noder og batch-størrelse (10 og 64) som i den indledende tuning (afsnit 12.3). Ændringen af læringsrate og tilføjelsen af dropout har altså reduceret præcisionen fra 50.2% til 49.57% af det bedste ANN netværk fra afsnit 12.3.

	Antal lag	Antal noder	Batch-størrelse	Læringsrate	Dropout-rate	Præcision
1	1	10	64	0.0051	0.2	49.57%
2	1	15	64	0.0026	0.4	49.36%
3	1	20	64	0.0026	0.4	49.36%
4	1	20	128	0.0026	0.4	49.25%
5	1	15	64	0.0051	0.4	49.04%

Tabel 6: Hyperparametre og præcision for de 5 bedste ANN netværk efter yderligere tuning.

Benytter vi vores tunede ANN netværk på vores testdata, kan vi i tabel 7 se modellens prædiktioner sammenlignet med de reelle observationer. Her repræsenterer diagonalen, de kategorier som modellen har prædikteret korrekt. Tabellen giver anledning til flere interessante observationer omkring vores model og dens tendenser i sine prædiktioner.

Som beskrevet er diagonalen de prædiktioner, som modellen har været i stand til at få korrekte.

Alle andre prædiktioner er altså misklassificeringer. Ser man bort fra de korrekte klassificeringer og bevæger sig et trin uden for diagonalen, danner der sig en tendens af, at modellen i mange situationer ikke begår store klassifikationsfejl. Det ses, hvordan prædiktionerne har det med at koncentrere sig omkring diagonalen enten som en kategori under eller over den korrekte. For eksempel har modellen korrekt klassificeret 27 tilfælde, hvor kategorien var 2. Derudover har den misklassificeret 15 tilfælde som kategori 2, der skulle have været 1 og 26 tilfælde, der skulle have været 3. Dog ses det også, at den kun fejlagtigt har klassificeret 3 tilfælde, der skulle have været 0 og 5 tilfælde, der skulle have været 5. Det samme er tilfældet for flere af de andre prædikterede kategorier. Det lader altså til, at modellen er god til ikke at være drastisk forkert. Den prædikterer sjældent en af de ydre kategorier, når kategorien i sandhed var en af de indre kategorier. Ligeledes prædikterer den sjældent en af de indre kategorier, når kategorien i sandhed var en af de ydre kategorier. Denne tendens tydeliggør sig også ved at inkludere fejlkategoriseringer af ± 1 kategori som korrekte. Havde dette været tilfældet, ville præcisionen på modellen øges til 82.3%. Det gør altså en stor effekt på modellens præcision at inkludere disse mindre fejlkategoriseringer. Samlet set viser det, at modellen er god til ikke at være drastisk forkert, eller med andre ord, er modellen omtrent korrekt.

		Faktisk kategori					
		0	1	2	3	4	5
Prædikteret kategori	0	121	43	10	3	1	1
	1	52	76	42	30	18	2
	2	3	15	27	26	16	5
	3	1	7	12	11	7	1
	4	3	12	32	35	99	38
	5	1	1	4	9	41	133

Tabel 7: Confusion matrix efter at have tunet yderligere på ANN-netværk.

Ser man på fordelingen af prædiktionerne i tabel 8, er det også tydeligt, at vores model favoriserer kategori 1 og 4 med henholdsvis 23.45% og 20.15% af prædiktionerne. Det giver god mening, at modellen foretrækket kategori 1 og 4, når man sammenholder med fordelingen af vores træningsdata, som ses i tabel 8 og figur 8b. Her ses det, at kategori 1 og 4 også er dem der fylder mest, hvilket betyder, at modellen i træningsfasen vil lære at lægge mere vægt på disse kategorier. Dette resulterer i, at modellen vil have en tendens til at favoriserer disse kategorier og dermed prædiktere dem oftere. Dette fænomen var som bekendt også berørt i databehandlingen (afsnit 12.1). Her forsøgte

vi netop at mitigere denne effekt ved at balancere træningsdata. Det viser sig dog, at vi ikke har formået at balancere det fuldkomment. Dog er det sandsynligt, at den ubalancerede kategorisering, som ses på figur 8a, havde produceret væsentligt værre resultater, hvorfor den nuværende fordeling af prædiktioner ikke virker alarmerende.

Kategori	0	1	2	3	4	5
Kategori-specifik præcision	67.6%	34.54%	29.35%	28.2%	45.2%	70.37%
Fordeling af prædiktion	19.08%	23.45%	9.8%	4.17%	23.35%	20.15%
Fordeling af træningsdata	10.4%	20.49%	18.54%	18.27%	22.3%	10%

Tabel 8: Den kategori-specifikke præcision og fordeling af prædiktioner af vores ANN netværk samt fordeling af træningsdata.

I forlængelse af dette ser vi også, at kategori 2 og 3 repræsenterer en væsentligt mindre del af modellens prædiktioner sammenlignet med deres respektive andele i træningsdata. Underrepræsentationen er kommet på bekostning af en overrepræsentation af kategori 0 og 5, vis andele af prædiktionerne er næsten dobbelt så store som i træningssættet. Samtidigt ser vi også, at den klassespecifikke præcision er lavest for disse to indre kategorier og højest for de to ydre kategorier 0 og 5.

At de klassespecifikke præcisioner fordeler sig på denne måde, er interessant. Som beskrevet i afsnit 12.1 vil en machine learning model have tendens til at favorisere kategorier, der er højt repræsenteret i træningsdata. Dog ser vi det omvendte i denne situation. En årsag til dette kan være det, der netop gør kategori 0 og 5 unikke nemlig deres ekstremhed. Disse kategorier er ydre kategorier af aktieafkast, som er større end $\pm 1.5\%$. Det er derfor muligt, at inputvariablene, som tilhører disse ydre kategorier, også er ekstreme i deres kategorier. Dette gør modellen i stand til bedre at kunne adskille disse kategorier fra de indre kategorier. De indre kategoriers inputvariable er muligvis ikke distinkte nok, hvilket kan gøre det svært for modellen at adskille en kategori 2 fra en kategori 1 eller en kategori 3 fra 4.

At de ydre kategorier er mere entydige i deres inputvariable, bliver tydeligt, når vi undersøger vores træningsdata. I tabel A.2 er der for hver kategori af Shell-aktiens afkast (kolonner) specificeret, hvordan kategorierne af inputvariablene fordeler sig (rækker). Ser man nærmere på udsnittet i tabel 9, er det tydeligt, at der opstår en tendens, hvor BP variablen har det med at befinde sig i de ydre kategorier (0 og 5) samtidigt med, Shell-aktien befinder sig her. Derimod ser vi, at når Shell-aktien for eksempel befinder sig i kategori 2 eller 3, har BP-aktien det med at fordele sig på tværs af flere

	Kategori 0	Kategori 1	Kategori 2	Kategori 3	Kategori 4	Kategori 5
BP Kategori 0	242	155	52	14	11	3
BP Kategori 1	95	313	197	113	57	9
BP Kategori 2	25	138	164	172	83	11
BP Kategori 3	13	90	143	166	153	23
BP Kategori 4	11	59	114	188	367	105
BP Kategori 5	5	14	26	32	166	244

Tabel 9: Udsnit af tabel A.2.

kategorier (1, 2, 3 eller 4). Denne tendens går igen for Exxon, Chevron og S&P GSCI variablene. Dog ser vi ikke samme mønster for MACD og RSI variablene, som befinder sig mest i kategori 0 eller 5 lige meget, hvor Shell-aktien befinder sig. De førstnævnte observationer understøtter altså, at modellen bedre kan adskille de ydre kategorier fra de indre kategorier, hvormed den opnår højere præcision hér.

Samlet set betyder det altså, at machine learning modellen er i stand til at lave prædiktioner, der ikke er drastisk forkerte. Modellen vægter som forventet kategori 1 og 4 højt grundet deres større repræsentation i træningsdata. Samtidigt er kategori 2 og 3 underrepræsenteret på bekostning af kategori 0 og 5, hvilket går imod forventningen om, at kategoriernes andel af prædiktionerne bør følge deres andel i træningsdata. Afslutningsvis udviser modellen øget præcision, jo mere ekstreme kategorierne bliver, hvilket højest sandsynlig skyldes deres tilhørerende inputvariables entydighed.

12.5 Inferens af black-box-model ved brug af *Local Interpretable Model-agnostic Explanations Technique* (LIME)

Lokal model af black-box-model

Vi vil nu forsøge at forklare lokale områder af vores ANN neurale netværk ved hjælp af LIME. Da vi som nævnt i afsnit 12.4 ikke har været i stand til at forbedre vores basismodeller fra afsnit 12.3, vil vi benytte det bedst præsterende netværk til LIME-forklaringen. Det betyder, vi har et ANN netværk med følgende hyperparametre som underliggende black-box-model:

- Batch-størrelse: 64
- Epoker: 50
- Læringsrate: 0.001
- Antal noder: 10 i det skjulte lag + 6 i outputlaget

- Antal lag: 1 inputlag + 1 skjult lag + 1 outputlag

Dette netværk resulterede i en præcision på 50.2%. Vi vil forsøge at forklare den sidste observation i vores testdata.

LIME vælger de fem mest indflydelsesrige variable ved hjælp af *forward selection*, sådan, at man opnår en surrogatmodel med lavest mulig fejl. Målet for LIME-modellen, er altså at forsøge at forklare hvilke fem faktorer, der har haft den største indflydelse på den komplekse models prædiktion.

Resultater - LIME

Ud fra figur 11 er der flere interessante observationer at gøre. Først ses det at vores ANN netværk har prædikteret, at den sidste observation af vores testdata tilhører kategori 3. Dette ses ved at "Probability: 1" ud fra "Label: Kategori 3". Ud fra vores testdata viser det sig, at den sidste observation i sandhed også tilhører kategori 3. Vores model har prædikteret rigtigt i denne situation.

Desuden ser vi, at vores LIME-model er kommet frem til fem variable, der har haft den største indflydelse på, at prædiktionen netop skulle tilhøre denne kategori:

- BP OPEN - Kategori 2
- BP OPEN - Kategori 3
- EXXON OPEN - Kategori 3
- CHEVRON OPEN - Kategori 4
- BP OPEN - Kategori 5

I denne situation har fire af variablene haft en positiv indflydelse på, at prædiktionen skulle tilhøre kategori 3 indikeret ved den blå farve "supports". Kun én variabel, BP OPEN - Kategori 5, modsiger, at prædiktionen skulle tilhøre kategori 3 indikeret ved den røde farve "contradicts".

Vi observerer, at kategorierne, som understøtter, at prædiktionen skulle tilhøre kategori 3, er henholdsvis kategori 2, 3 og 4. Ligeledes ser vi, at LIME-modellen fremhæver inputvariable af kategori 0 til at forklare, at prædiktionen bør tilhøre kategori 0. Denne tendens fortsætter i mange af de andre forklaringer og indikerer, at der er en sammenhæng mellem kategorien af prædiktionen og kategorien af de betydningsfulde inputvariable. At Shell-aktiens afkast for en given dag for eksempel er i kategori 3, er altså understøttet af, at dens industrikonkurrenter også befinder sig i omtrent denne kategori. Denne tendens giver god mening, da man må forvente, at der er en vis

korrelation mellem aktier i samme industri. De lokale forklaringer indikerer dermed, at modellen fanger en relation mellem kategorien af inputvariablene og kategorien af Shell-aktien.

Samtidigt kan figur 11 også understøtte ideen om, at inputvariable for de ydre kategorier er mere distinkte end de indre kategorier (som forklaret i afsnit 12.4). Som det ses på figur 11, lader det til, at for forklaringer af de indre kategorier er der er flere kategorier af inputvariable, der understøtter prædiktionen. Dette ses for eksempel for forklaringen af kategori 3, hvor inputvariable af kategori 2, 3 og 4 understøtter. Derimod ses det for forklaringerne af kategori 0 og 5, at det udelukkende er inputvariable af samme kategori, som understøtter hér. Denne observation indikerer altså også, at præcisionen for de ydre kategorier er højest, fordi inputvariablene er mere entydige i deres design, hvilket vores model formår at fange.

Ser vi på figur 11, er det også interessant at se på R^2 værdierne for de forskellige lokale modeller. Vi ser, at de er lavest for kategori 2 og 3 på henholdsvis 0.1 og 0.17, mens de er højest for kategori 0 og 5 med henholdsvis 0.34 og 0.38. Selvom alle fire værdier generelt set kan anses som lave, er det interessant, at LIME er i stand til bedre at kunne modellere de ydre kategorier. Denne observation kan også ses i tråd med tidligere observation omkring det neurale netværks højere præcision omkring de ydre kategorier. Vi ser på samme måde her, at LIME er i stand til bedre at kunne modellere disse kategorier. Dette indikerer igen, at karakteristikaene for de ydre kategorier er lettere at identificere for modellen i kontrast til de indre kategorier.

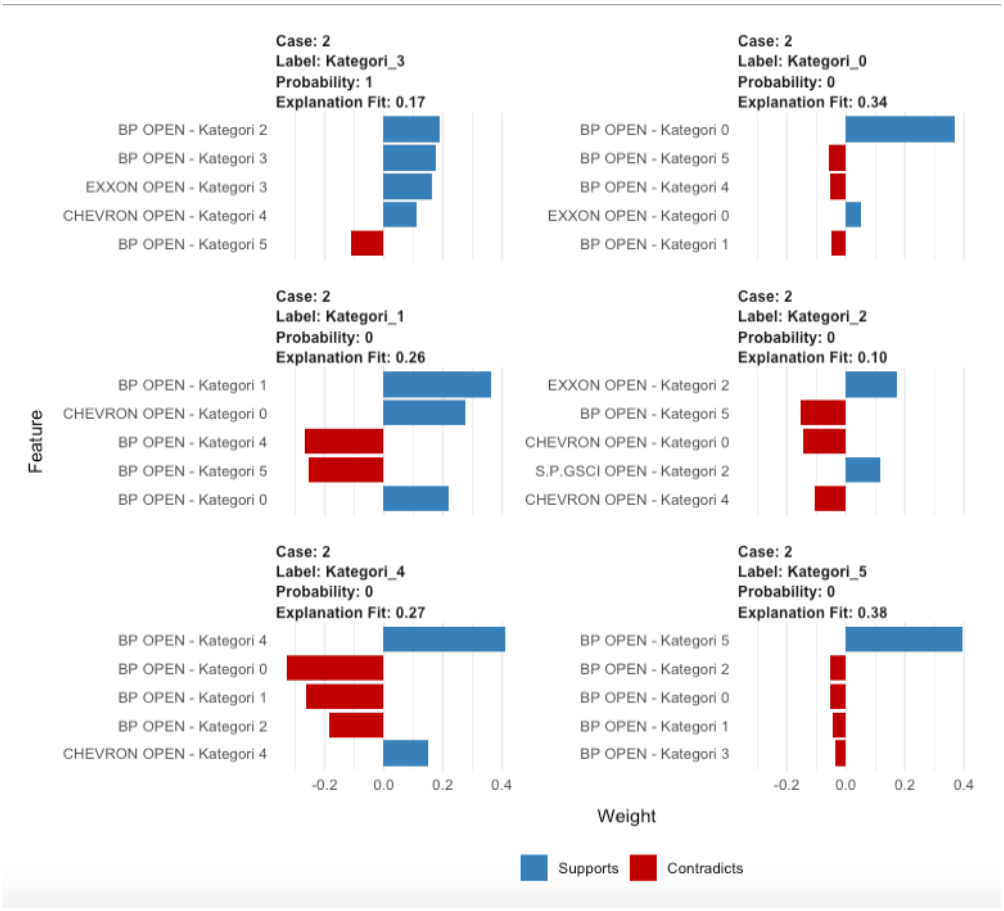
Efter at have undersøgt de lokale forklaringer kan man også brede analysen ud på et globalt plan. Her vil det blive tydeligt, om der er en eller flere af de seks overordnede inputvariable (MACD, RSI, CHEVRON, BP, EXXON, S&P GSCI), som LIME-modellen hyppigt vægter som værende blandt de mest indflydelserige variable. På den måde får man et bedre indtryk af vigtigheden af hver af de overordnede variable på black-box-modellens prædiktioner.

Dette gøres ved at benytte LIME på et større antal observationer end blot én. Undersøger vi nu i stedet de sidste 50 observationer i vores testdata og aflæser, hvor mange gange de enkelte overordnede variable bliver valgt til at være blandt de fem mest indflydelsesrige, får vi fordelingen som ses i tabel 10.

Variabel	BP	MACD	RSI	CHEVRON	EXXON	S.P. GSCI
Antal	1068	2	4	164	166	96

Tabel 10: Blandt de seks overordnede inputvariable er BP oftest blandt de fem mest indflydelsesrige inputvariable.

Vi ser, at BP bliver valgt størstedelen af gangene. Dette kunne indikere, at BP har en stor indfly-



Figur 11: LIME-modellen fremhæver de fem mest indflydelsesrige variable i black-box-modellens prædiktion.

delse på, hvordan Shells aktiekurs bevæger sig i kontrast til konkurrenter såsom Exxon og Chevron. Dertil ses det, at de resterende overordnede variable vælges relativt få gange. Dette kunne indikere, at de ikke har en stor global signifikans for black-box-modellen.

Selvom formålet med LIME netop er at forklare vores neurale netværk i en lokal omegn, kan dette overblik give os en god indikation af, hvilke af vores inputvariable har en "global" indflydelse på vores models prædiktion.

Samlet set indikerer vores lokale analyse, at modellen lader til at fange en sammenhæng mellem kategorien for prædiktionen og kategorien for inputvariablene. Derudover indikerer LIME-modellernes forklaringer også, at karakteristikaene for de ydre kategorier er mere distinkte og dermed lettere for det neurale netværk at identificere. Afslutningsvis viser vores globale analyse, at BP oftest bliver fremhævet som en betydningsfuld variabel af LIME-modellen fremfor de fem andre.

12.6 Konklusion på analyse

Analysen af vores tre netværkstyper viser, at ANN netværket opnår klart højest præcision blandt de tre netværkstyper uden tuning. Dog kan en simpel tuning af noder og batch-størrelse øge RNN og LSTM netværkenes præcision, hvormed de opnår en sammenlignelig præcision til ANN netværket på omkring 50%. Efter den simple tuning er der dermed ingen af netværkene, der udviser klart højere præcision end andre. Dertil vil yderligere tuning ved brug af *grid search* ikke øge præcisionen af ANN netværket. Vi finder også at ANN netværket ikke prædikterer drastisk forkert og vægter kategori 1 og 4 højt i dens prædiktioner. Samtidigt undervægter den kategori 2 og 3 på bekostning af 0 og 5, hvilket kan skyldes de ydre kategoriers ekstreme design. I brugen af LIME til analysen af ANN netværket viser den lokale analyse, at ANN netværket kan finde sammenhænge i data mellem input- og responsvariablerne. Dertil viser den globale analyse, at BP-variablen oftest vælges blandt de 5 mest betydningsfulde variable til en forklaring.

13 Diskussion

13.1 Grid search vs. Random search

Vi har i analysen benyttet os af *grid search* til tuning af vores neurale netværk. Denne metode har medført, at antallet af kombinationer af hyperparametre stiger eksponentielt (afsnit 10.2). Da det tager lang tid at teste alle kombinationer, valgte vi at inkludere færre værdier i tuningen af hyperparametrene. Dette medfører en større risiko for, at vi overser de bedste kombinationer af hyperparametre. Det er altså muligt, at vi ved brugen af *grid search* ikke har fundet den optimale kombination af hyperparametre. En anden, og måske bedre, mulighed havde været at benytte *random search*. *Random search* vælger tilfældige kombinationer af hyperparametre i det rum, hyperparametrene udspænder, og udvælger den bedste kombination ud af alle mulige. *Random search* konvergerer hurtigere mod optimale hyperparameterværdier end *grid search*, hvorfor det kunne have været tidsbesparende at vælge denne metode. Dertil er det også standard at benytte *grid search*, når der er tre eller færre hyperparametre, som skal tunes (afsnit 10.2). Ud fra karakteristikaene ved de to metoder vil det teoretisk set, godt kunne svare sig at lave *grid search* i den første del af vores analyse (afsnit 12.3), da vi kun tuner på to hyperparametre. Derimod vil teorien sige, at vi skal benytte os af *random search*, når vi laver yderligere tuning (afsnit 12.4), da vi her tuner 5 hyperparametre. Set i retrospekt burde vi have ændret metoden til at tune vores hyperparametre undervejs i vores analyse.

13.2 Potentielle årsager til ANN opnår højere præcision end RNN og LSTM

Som beskrevet i afsnit 9 udviser RNN og LSTM-netværk typisk et bedre resultat, når man arbejder med tidsseriedata som vores. Dette står i kontrast til ANN netværk der, som beskrevet i afsnit 8.4, lider af ikke at have indbygget hukommelse. Derfor ville man umiddelbart have forventet, at resultaterne i afsnit 12.3 havde illustreret ANN netværkets svagheder i form af en lavere præcision. Dette var dog ikke tilfældet.

Som vi fandt i afsnit 12.3, opnåede ANN netværket den højeste præcision efter at have tunet på noder og batch-størrelse. At ANN netværket præsterede bedst, kan have flere årsager heriblandt de følgende to:

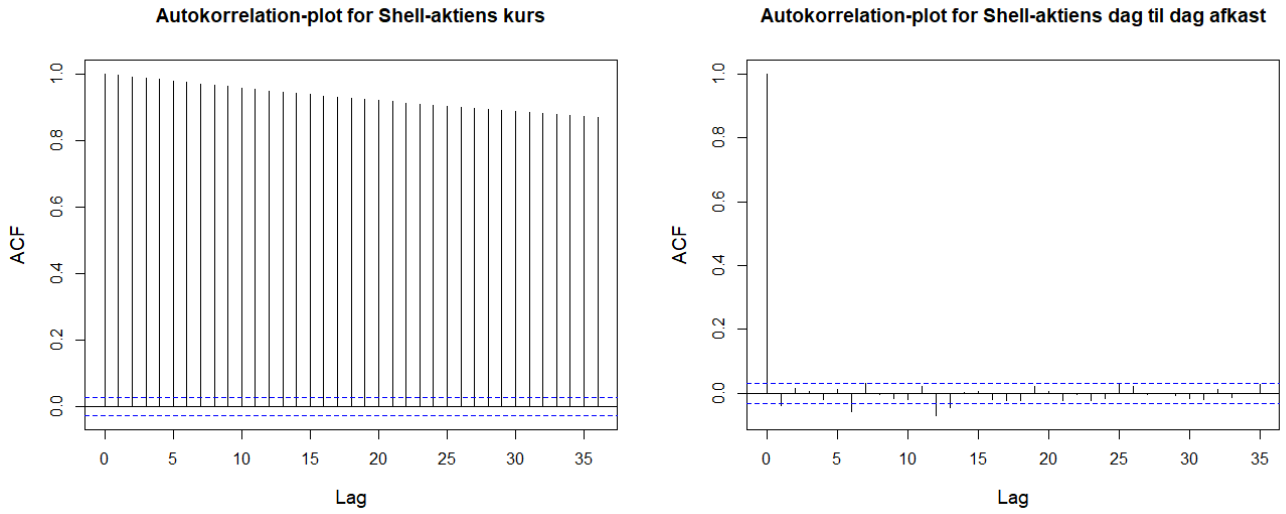
ANN netværkets højere præcision kan først og fremmest skyldes, at vi begrænsede os til to hyperparametre i den indledende tuningsproces (batch-størrelse og antal noder). Som resultat af vores

indledende tuning (afsnit 12.3) så vi to tendenser opstå. Den første tendens var, at større batch-størrelse ledte til faldende præcisionen, hvilket var tydeligst for RNN og LSTM netværkene. Det vil sige, at det potentielt kunne have givet en højere præcision for disse to netværkstyper, hvis vi havde haft lavere batch-størrelser med i tuningsprocessen. Den anden tendens, vi så, var, at et stigende antal noder ville forbedre præcisionen, hvilket igen var særlig tydeligt for RNN og LSTM netværkene. Det er muligt, at vi også hér kunne have forbedret præcisionen ved at have inkluderet et større antal af noder i tuningsprocessen. Som det dog også fremgår af afsnit 12.3, kan et stort antal noder også føre til overfitting, hvilket sænker præcisionen. Der opstår altså en grænse for, hvor mange noder vi kunne have i det skjulte lag. Samlet set er det altså muligt, at vores ANN netværk udviste højere præcision end de andre netværkstyper på grund af afgrænsningen af den indledende tuningsproces. Vi kunne eventuelt have udvidet denne proces ved at have inkluderet flere hyperparametre såsom antallet af skjulte lag. Derudover kunne vi også have inkluderet flere værdier for hver hyperparameter men med det in mente, at antallet af kombinationer stiger eksponentielt.

Som tidligere bemærket udviste ANN netværket ikke betydeligt højere præcision end de to andre efter den indledende tuning. Tværtimod fandt vi, at RNN netværkets præcision var blot 0.1%-point lavere og LSTM netværkets præcision 2.8%-point lavere. Det er altså ikke fordi, ANN netværket præsterede betydeligt bedre. Dog så vi i tabel 4, at ANN netværket udviste betydeligt højere præcision inden tuningen, hvilket leder os til den anden potentielle årsag til dette resultat.

Som del af vores databehandling transformerede vi vores responsvariabel, Shell-aktien, til procentændring. Dette har visse implikationer for modellernes ydeevne. Typisk vil man med tidsseriedata observere en høj grad af korrelation mellem observationer fra et tidspunkt til det næste. Tendensen kan bekræftes i et autokorrelationsplot. Hvis en tidsserie udviser høj autokorrelation, vil et sådant plot vise en let faldende tendens. En tidsserie uden vil derimod have størstedelen af observationerne, på nær den første, omkring 0. Hvis vi havde bibeholdt vores responsvariable som den reelle aktiekurs, er der tydelig autokorrelation, hvilket kan ses på figur 12a. Dog fjernes denne autokorrelation, når vi transformerer responsvariablen til den daglige procentændring i aktien, som ses på figur 12b. Det vil altså sige, at afkastene fra dag til dag er stort set uafhængige. Denne observation er også understøttet af økonomisk teori, der siger, at aktiemarkedet følger en *random walk* (Fama, 1965). *Random walk* beskriver aktiebevægelser som tilfældelige og uforudsigelige. Der er altså ingen sammenhæng mellem afkastet i går og i dag, hvilket teoretisk set gør det umuligt at prædiktere afkastet for enkeltaktier. Ved at fjerne tidsafhængigheden fjerner vi dermed historikken i data. Som beskrevet i afsnit 9.1 og afsnit 9.3 er det netop egenskaben med at kunne opbygge hukommelse, der gør RNN og LSTM-netværkene unikke. Vores transformation af responsvariablen

gør det altså irrelevant for en model at have denne egenskab, hvilket kan forklare, hvorfor ANN netværket opnår en betydeligt højere præcision inden tuning.



(a) ACF-plot for aktiekursen

(b) ACF-plot for afkast

Figur 12: Den reelle aktiekurs, (a), udviser høj autokorrelation, hvilket ses ved den let faldende tendens. Når responsvariablen transformeres til dag til dag afkast, (b), udviser tidsserien lille autokorrelation, hvilket ses ved at størstedelen af observationerne befinder sig omkring 0.

13.3 Mulighed for yderligere forbedring af netværk

Som tidligere nævnt blev det klart, at hverken basismodellerne eller de tunede modeller var i stand til at opnå en præcision betydeligt over 50%. Som fremhævet er det dog stadig væsentligt bedre end det naive gæt på 1/6, som typisk tjener som en minimumsgrænse for en machine learning algoritmes eksistensgrundlag. At vores modeller ikke yder bedre end 50%, stemmer godt overens med den førnævnte teori om *random walk*. Det var derfor muligvis forventeligt, at vi ikke kunne opnå en præcision, der var betydeligt bedre. Man kunne dog gøre sig overvejelser over hvilke tiltag, der kunne have bidraget til at forbedre modellernes præcision.

Først og fremmest er det muligt, at man i det indledende arbejde kunne have valgt et større antal inputvariable. I og med at Shell befinder sig inden for olieindustrien, ville det have været oplagt at inkludere olieprisen som inputvariabel. Dertil kunne man have medbragt andre makro-variable såsom renteniveauet i form af for eksempel den amerikanske 10-årige statsrente. Man kunne også have medbragt flere firmaspecifikke faktorer, såsom udviklingen i indtjeningen per aktie, som en slags dummy-variabel. Denne kunne muligvis have fanget særligt store udsving i aktien. Disse

tilføjelser kunne muligvis have gjort vores machine learning modeller bedre til at fange afkastmønstrene for aktien. Dog er det sandsynligt, at sådanne tilføjelser ikke havde tilført yderligere værdi til modellen, da man i stedet havde stødt på andre udfordringer i modelleringsprocessen. Ved at inkludere flere inputvariable ville man muligvis skulle have øget kompleksiteten af det neurale netværks arkitektur. Dette ville føre til et netværk med flere vægte. Som beskrevet i 8.1 kan dette lede til overfitting, da modellen lærer støjen i data fremfor de underliggende mønstre. Derudover ville den øgede kompleksitet også betyde, at modellen skulle optimere flere vægte, hvilket kunne have ført til længere træningstider. Derfor kan man altså ikke konkludere, at flere inputvariable i modellerne havde forbedret deres præcision.

Endnu en måde, hvorpå modellernes præcision kunne have været forbedret, ville være ved hjælp af en modificeret tabsfunktion i træningsprocessen. Som vi observerede i tabel 8, havde vores model den laveste kategori-specifikke præcision på kategori 2 og 3. Dette var indre kategorier, som sandsynligvis var svære for modellen at identificere på grund af inputvariablenes mangel på distinkthed. Hvis vi havde været i stand til at forbedre præcisionen for disse kategorier, havde det forbedret den samlede præcision for modellen betydeligt. Dette kunne have været opnået ved hjælp af en modificeret tabsfunktion, der ville have straffet modellen betydeligt hårdere for at prædiktere kategori 2 og 3 forkert. En måde at opbygge denne modificerede tabsfunktion er at tilføje et ekstra led til den oprindelige tabsfunktion. Dette ekstra led kunne se ud som på følgende måde:

$$-1_{\{y_i=k|k=2,3\}} \log(\hat{y_i}) \tag{11}$$

Her har vi en indikatorfunktion som er 1, når y_i er i kategori 2 eller 3 og 0 ellers. Dertil bruger vi logaritmen af prædiktionen, da dette vil straffe tabsfunktionen, hvis modellen har prædikteret forkert, hvilket er samme princip som *cross entropy* benytter sig af. Dermed vil modellen lægge mere vægt på disse kategorier og derigennem potentielt øge præcisionen for dem. Dog kunne det også have medført, at andre kategorier ville have blevet nedprioriteret, hvilket havde sænket modellens generelle præcision.

13.4 Overvejelser ved brugen af LIME

Motivationen for at bruge LIME er som bekendt ønsket om at løse black-box-problemet. Som bruger ønsker man at få et bedre indblik i hvilke mønstre, der ligger bag modellens beslutningsproces. I vores brug af LIME er det blevet tydeligt, at på trods visse svagheder ved de lokale modeller er LIME stadig i stand til at løse dette problem.

Som nævnt i afsnit 11 bygger LIME på antagelsen om, at komplekse modeller kan modelleres af lineære funktioner i en lokal omegn. Dog ser vi også fra figur 11, at LIME-modellen har en lav R^2 på blot 0.17 for kategori 3. Den lave R^2 observeres også for LIME-modeller for andre kategorier. Dette kan indikere, at en lineær model ikke er den bedste model til at modellere denne form for kompleks klassifikationsmodel. (Molnar, 2020, s. 224) finder også, at jo mere komplekse de sande sammenhæng i modellen er, desto svære bliver det for LIME at lave en pålidelig lokal model. Hvis vi havde implementeret en mere kompleks model, som foreslået i forrige afsnit, er det muligt vi kunne have forbedret netværkets præcision. Dog havde dette betydet, at vi havde gjort det endnu svære for LIME at lave pålidelige lokale modeller. For at kompensere for den øgede kompleksitet af black-box-modellen kunne vi i stedet have benyttet mere komplekse lokale modeller, såsom logistiske eller træ-baserede modeller. Disse kunne muligvis have været i stand til at opnå en højere R^2 værdi i den lokale omegn. Dog ville en sådan model gå imod motivationen ved LIME, som netop bygger på, at surrogatmodellen skal være fortolkbar. Havde vi implementeret en mere kompleks model, havde vi gået på kompromis med dette princip. Afvejningen mellem forståelsen af surrogatmodellen og dens evne til at modellere pålideligt i en lokal omegn bliver altså også tydelig i vores brug af LIME.

Derudover har metoden, vi vælger, som den LIME skal benytte til at vælge de mest betydningsfulde forklaringer, stor effekt på hvilke variable, der bliver valgt oftest. Til analysen i afsnit 12.5 benyttede LIME en *forward selection* metode, der tilføjer variable til en ridge-regressionsmodel, sådan at fittet forbedres mest muligt. Hvis i stedet *highest weights* metoden var benyttet, vælges de variable med den højest absolutte vægt i ridge-regressionsmodellen. I forhold til den lokale analyse har det ikke stor betydning for kvaliteten af de lokale modeller, som ses på bilag B.2. Dog har det stor betydning for den globale analyse, hvor vi aflæste, hvor ofte hver af de 6 overordnede inputvariable blev valgt blandt de 5 mest betydningsfulde for en forklaring. Tabel 11 viser fortsat, at BP vælges oftest. Dog vælges MACD og RSI nu henholdsvis 256 og 80 gange, hvilket står i kontrast til blot at blive valgt henholdsvis 2 og 4 gange med *forward selection*. Valget af metode har altså stor påvirkning på forståelsen af hvilke variable, der påvirker modellen på overordnet plan.

Variabel	BP	MACD	RSI	CHEVRON	EXXON	S.P. GSCI
Antal	891	256	80	101	135	37

Tabel 11: Med *highest weights* vælges variable såsom MACD og RSI markant oftere end ved *forward selection*.

Det er også muligt, at der kunne justeres yderligere på LIME-modellen således, at den potentielt vil

opnå en bedre lokal model. Som beskrevet i den teoretiske indledning til LIME (afsnit 11) vægtes observationer baseret på deres afstand til prædiktionen, man forsøger at forklare. Vi kunne have ændret på måden afstanden blev målt på, samt hvor stor en omegn modellen skulle modellere. Havde vi for eksempel øget omegnen, som LIME modellerede, kunne det have medført en mere stabil model, da den havde vægtet observationerne mere ligeligt. Dog havde det igen betydet, at vi var gået på kompromis med teorien bag LIME, som netop fokuserer på at modellere pålideligt i et lokalt område. Modsat havde en for lille omegn resulteret i, at de lokale forklaringer ikke var stabile. Dette betyder, at LIME-modellens forklaringer havde varieret betydeligt afhængigt af prædiktionen, vi forsøgte at forklare. Dette ville igen gå på kompromis med principperne bag LIME, som også fokuserer på stabiliteten af forklaringerne. Det er altså muligt, vi kunne have justeret yderligere på LIME-modellerne men ikke nødvendigvis have tilføjet mere værdi til indblikket, LIME gav os.

13.5 LIMEs egenskab til at løse black-box-problemet

Som fremhævet i analysen (afsnit 12.5) var LIME i stand til at tydeliggøre, at modellen fangede sammenhænge mellem kategorien af inputvariablene og kategorien af responsvariablen. Ligeledes fandt vi, at BP havde en stor betydning for LIME-modellens forklaringer. Disse observationer bør dog ikke være nogen overraskelse for os, da det blot bekræfter korrelation. Havde vi indledt dataanalysen med en korrelationsmatrix mellem variablene, havde vi på forhånd kunne forvente disse sammenhænge (se figur 13 for korrelationsmatrix). Derfor er det ikke overraskende, at vi observerer, at modellen er i stand til at fange korrelationen i variablene. På trods af dette bekræfter det dog netop, hvad LIME er skabt til at gøre. LIME har gjort os i stand til at bekræfte, at det neurale netværk fanger de mønstre, vi som brugere af modellen måtte forvente. Hvis vi ikke havde benyttet LIME, eller hvis LIME havde udvist, at modellen ikke havde fanget dette mønster, havde vi ikke haft den samme tillid til den. På trods af de forventelige observationer har LIME været i stand til at opnå dens formål ved netop at give et indblik i black-box-modellens beslutningsproces.

	SHELL OPEN	SHELL MACD	SHELL RSI	CHEVRON OPEN	BP OPEN	EXXON OPEN	S&P GSCI TR OPEN
SHELL OPEN	1	0.01	0.01	0.52	0.8	0.5	0.36
SHELL MACD	0.01	1	0.01	0.01	0	0.01	0.01
SHELL RSI	0.01	0.01	1	0	0	−0.01	−0.01
CHEVRON OPEN	0.52	0.01	0	1	0.5	0.85	0.39
BP OPEN	0.8	0	0	0.5	1	0.49	0.34
EXXON OPEN	0.5	0.01	−0.01	0.85	0.49	1	0.38
S&P GSCI TR OPEN	0.36	0.01	−0.01	0.39	0.34	0.38	1

Figur 13: Korrelationsmatrix af variable. NB TR står for total return.

13.6 Konklusion på diskussion

Baseret på overstående kan vi konstatere, at vi i tuningsprocessen skulle have varieret tuningsmetoden til at benytte *random search* og *grid search*. Derudover er det muligt, at vi i vores tuningsproces ikke har været i stand til at opnå højere præcision end 50% på grund af afgrænsningen i valget af hyperparametre samt teorien om *random walk*. Man kunne muligvis have forsøgt sig med et større antal hyperparametre samt en modificeret tabsfunktion, selvom førstnævnte havde kommet på bekostning af længere træningstid. Det er muligt, at transformationen af data til procent-ændring har gjort hukommelsesegenskaberne ved RNN og LSTM overflødige, hvorfor ANN netværket opnår højest præcision. Afslutningsvis er det muligt at vores modelleringsproblem er for komplekst til, at LIME kan skabe gode lokale modeller. Dog har LIME stadig skabt værdi, da den giver et indblik i det neurale netværks beslutningsproces og dermed løser black-box-problemet.

14 Konklusion

Ud fra vores resultater kan vi konkludere, at man kun i nogen grad kan benytte de tre typer af neurale netværk ANN, RNN og LSTM til at modellere aktieafkast. Dette skyldes, at ingen af de tre netværkstyper opnåede betydeligt højere præcision end 50%. Dertil kan vi konkludere, at LIME kan løse black-box-problemet ved at give et indblik i neurale netværks beslutningsproces. Dog fejler den i at skabe gode lokale modeller, da vores modelleringsbehov sandsynligvist er for komplekst.

Efter indledende opbygning og test af de tre netværkstyper uden tuning fandt vi, at ANN netværket opnåede den højeste præcision på 47%, RNN netværket en præcision på 29% og LSTM netværket en præcision på 21%. Efter tuning af to hyperparametre, batch-størrelse og antal noder, kom vi frem til, at RNN og LSTM-netværkene forbedrede deres præcision betydeligt til henholdsvis 50.1% og 47.4%. Præcisionen for ANN netværket steg også til 50.2%. ANN netværket forblev dermed det netværk med højest præcision, dog med få %-point til forskel. Efter yderligere tuning af fem hyperparametre i vores ANN netværk fandt vi, at ANN netværket opnåede en præcision på 49.57%. ANN netværket præsterede altså værre end tidligere. Efter at have testet vores ANN netværk på testdata finder vi, at det er godt til ikke at prædiktere drastisk forkert. Netværket vægter kategori 1 og 4 højt i dens prædiktioner, og samtidig undervægter den kategori 2 og 3 på bekostning 0 og 5. Brugen af LIME på ANN netværket viste, at netværket var i stand til at skabe en lokal sammenhæng mellem kategorien af inputvariablene og kategorien, som netværket prædikterede. Derudover blev det klart, at LIME oftest vælger British Petroleum (BP) variablen i sine forklaringer.

Efter at have evalueret vores resultater og metoder fandt vi, at vi i tuningsprocessen skulle have vekslet mellem *grid search* og *random search* afhængigt af antallet af hyperparametre. Dertil er det muligt, at vi kunne have opnået højere præcision af vores netværk ved at have testet et større antal værdier for hver hyperparameter, øget antallet af inputvariable eller implementeret en modificeret tabsfunktion. Førstnævnte ville dog være på bekostning af længere tidsforbrug i tuningsprocessen. Dertil siger *random walk* teorien, at aktiebevægelser er uforudsigelige, hvorfor det er muligt, at vi ikke kunne have opnået betydeligt højere præcision end 50%. Transformationen af data til procent-ændring har sandsynligvis medført, at hukommelsesegenskaberne, som RNN og LSTM-netværkene besidder, bliver overflødig. Derudover er det for vores LIME model muligt, at vores modelleringsbehov er så komplekst, at LIME ikke er i stand til at skabe gode lokale modeller. Dertil har valget af *feature selection* metode en stor betydning for, hvilke variable LIME benytter i sine forklaringer. Slutteligt har LIME vist, at vores netværk kan finde sammenhænge mellem vores inputvariable og responsvariabel, hvilket øger tilliden til *black-box*-modellen.

15 Perspektivering

Vores projekt bygger på modeller, der benytter dagens udvikling i inputvariablene til at prædiktere dagens afkast for Shell-aktien. Det har ikke meget værdi i den virkelige verden, da det er klart, at hvis man kender dagens afkast for inputvariablene, vil man også kende Shell-aktiens afkast. Hvis modellerne skal have finansiel betydning, skal man være i stand til at profitere ved at benytte dem. Set i dette lys havde det været oplagt, at vi havde tidsforskudt vores inputvariable, så dagens afkast bliver prædikteret ud fra gårsdagens udvikling i inputvariablene. Dertil kan det også være interessant at udbygge opgaven, med en undersøgelse af relationen mellem *Efficient Market Hypothesis* (EMH) og neurale netværk. Teorien siger, at hvis alle kender morgendagens pris på en aktie, vil alle skynde sig at købe/sælge den, hvormed profitmulighederne forsvinder. Hvis enhver finansiel institution er i stand til at bygge et neuralt netværk, der pålideligt kan prædiktere morgendagens afkast, kan man diskutere hvorvidt der rent faktisk er mulighed for at profitere, eller om EMH holder, og alt profit derfor handles væk.

Tilføjelsen af andre machine learning algoritmer så som Support Vector Machines eller Random Forests for at sammenligne deres ydeevne med neurale netværk kan også være en interessant videreførelse. Dette vil tydeliggøre, om der er bedre alternativer til neurale netværk i forsøget på at prædiktere aktieafkast. Foruden at tilføje andre macine learning algoritmer kan det også være interessant at sammenligne LIME med andre metoder som forsøger at løse *black-box*-problemet. Man kunne for eksempel se på *Shapley Additive Explanations* (SHAP), som inkluderer spilteori til at forklare, hvilke inputvariable der har en indflydelse på prædiktionen.

Selvom LIME bekræftede simple antagelser i vores projekt, kan det vise sig at være mere kritisk i andre sammenhænge. Hvis det neurale netværk i stedet havde været brugt til patient-diagnosticering, havde LIME gjort brugeren (lægen/sygeplejersken) i stand til at bekræfte, om modellen havde opfanget grundlæggende antagelser om et sygdomsforløb. Hvis modellen havde opfanget mønstrene korrekt, ville det have øget pålideligheden ved modellens prædiktion og dermed gjort brugeren mere sikker. Modsat havde brugeren også været i stand til at tage sagen i egen hånd, hvis modellen baserede sin prædiktion på forkerte antagelser. I en sådan situation er det muligt, at modellen skulle justeres yderligere. Set i dette lys kan det være interessant at diskutere black-box-problemet i et bredere perspektiv. I nogen situationer kan det have stor betydning, at forstå hvad der ligger til grund for modellens beslutning, og i andre situationer har det mindre betydning. For eksempel i et sundhedsmæssigt perspektiv i kontrast til et finansielt henseende. Black-box-problemet har således forskellig betydning afhængig af, hvordan den endelige model skal benyttes. Dette kunne være interessant at diskutere yderligere.

16 Litteraturliste

Aggarwal, C. (2023). *Neural Networks and Deep Learning, A Textbook* (2. Edition). Springer.

Bianchi, F., Maiorino, E., Rizzi, M., & Jenssen, R. (2017). *Recurrent Neural Networks for Short-Term Load Forecasting*. Springer.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning* (1.). Springer.

Brand, Y. E., Schwartz, D., Gazit, E., Buchman, A. S., Gilad-Bachrach, R., & Hausdorff, J. M. (2022). Gait Detection from a Wrist-Worn Sensor Using Machine Learning Methods: A Daily Living Study in Older Adults and People with Parkinson's Disease. *Sensors*, *22*(18). https://doi.org/10.3390/s22187094

Bronwlee, J. (2022). Dropout Regularization in Deep Learning Models with Keras - MachineLearningMastery.com. https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

Brownlee, J. (2021). How to Choose an Activation Function for Deep LearningHow to Choose an Activation Function for Deep Learning. https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

Canuma, P. (2023). How to Deal With Imbalanced Classification and Regression Data. https://neptune.ai/blog/how-to-deal-with-imbalanced-classification-and-regression-data

Chowdhury, K. (2021). 10 Hyperparameters to keep an eye on for your LSTM model — and other tips | by Kuldeep Chowdhury | Geek Culture | Medium. https://medium.com/geekculture/10-hyperparameters-to-keep-an-eye-on-for-your-lstm-model-and-other-tips-f0ff5b63fcd4

Dolan, B. (2022). What Is MACD? https://www.investopedia.com/terms/m/macd.asp

Fama, E. F. (1965). Random Walks in Stock-Market Prices. *Financial Analysts Journal*, *21*(5), 55–60. https://www.jstor.org/stable/4469865

Fernando, J. (2024). Relative Strength Index (RSI) Indicator Explained With Formula Learn how to measure the magnitude of price changes in 11 minutes What Is the Relative Strength Index (RSI)? https://www.investopedia.com/terms/r/rsi.asp

Ghatak, A. (2019). *Deep learning with R*. Springer Singapore. https://doi.org/10.1007/978-981-13-5850-0

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org

Gupta, B. (2023). Random Search in Machine Learning - Scaler Topics. https://www.scaler.com/topics/machine-learning/random-search-in-machine-learning/

Hayer, A. (2022 april). S&P GSCI Definition, Commodity Types Listed, Potential Drawbacks. https://www.investopedia.com/terms/g/gsci.asp#:~:text=The%20S%26P%20GSCI%20is%20made,metals%2C%20agriculture%2C%20and%20livestock.

Liu, W. (2020). Neural Networks. I *SAGE Research Methods Foundations* (1.). SAGE Publications Ltd. https://doi.org/10.4135/9781526421036888177

Molnar, C. (2020). *Limitations of Interpretable Machine Learning Methods*. https://slds-lmu.github.io/iml_methods_limitations/lime.html

Müller, A. C., & Guido, S. (2017). *Introduction to Machine Learning with Python* (1.). O'Reilly Media, Inc.

Ng Andrew. (2019). *Standard notations for Deep Learning* (tekn. rapp.). Stanford University. https://cs230.stanford.edu/files/Notation.pdf

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should i trust you?"Explaining the predictions of any classifier. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13-17-August-2016*, 1135–1144. https://doi.org/10.1145/2939672.2939778

Rodeck, D. (2024 marts). Best Energy Stocks – Forbes Advisor. https://www.forbes.com/advisor/investing/best-energy-stocks/

Shen, K. (2018). Effect of batch size on training dynamics | by Kevin Shen | Mini Distill | Medium. https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e#

Understanding Relationships Between Industry Peers with MarketReader - MarketReader. (2023). https://marketreader.com/understanding-relationships-between-industry-peers-with-marketreader/

Bilag

A Tabeller

Resultater af de tre netværk

Hyperparametre		ANN		RNN		LSTM	
Antal noder	Batch-størrelse	Tab	Præcision	Tab	Præcision	Tab	Præcision
5	16	1.2693	0.49254	1.5032	0.37846	1.3740	0.42217
5	32	1.2720	0.49467	1.5298	0.37313	1.4490	0.33582
5	64	1.2811	0.48827	1.5768	0.37420	1.5239	0.30384
5	128	1.3243	0.47015	1.6297	0.36141	1.6436	0.27186
10	16	1.2716	0.48294	1.2974	0.47015	1.3212	0.46055
10	32	1.2746	0.48934	1.3055	0.47335	1.3272	0.45096
10	64	1.2756	0.50213	1.3252	0.44136	1.3740	0.41578
10	128	1.2922	0.50000	1.3610	0.43497	1.4825	0.32090
15	16	1.2807	0.48614	1.2739	0.50107	1.3238	0.46908
15	32	1.2744	0.49147	1.2680	0.48507	1.3356	0.44243
15	64	1.2744	0.48507	1.2878	0.48614	1.3634	0.40618
15	128	1.2775	0.49147	1.3157	0.47548	1.4775	0.32303
20	16	1.2834	0.47655	1.2924	0.46375	1.3157	0.47441
20	32	1.2798	0.48294	1.2721	0.47761	1.3329	0.44136
20	64	1.2792	0.48827	1.2759	0.47548	1.3854	0.41578
20	128	1.2808	0.48614	1.2753	0.48188	1.4699	0.34861

Tabel A.1: Et netværk med 10 noder og en batch-størrelse på 64 giver den højeste præcision for et simpelt neuralt netværk på 50.2%.Et netværk med 15 noder og en batch-størrelse på 16 giver den højeste præcision på 50.1% for et rekursivt neuralt netværk. Et netværk med 20 noder og en batch-størrelse på 16 giver den højeste præcision på 47.4% for et LSTM neuralt netværk.

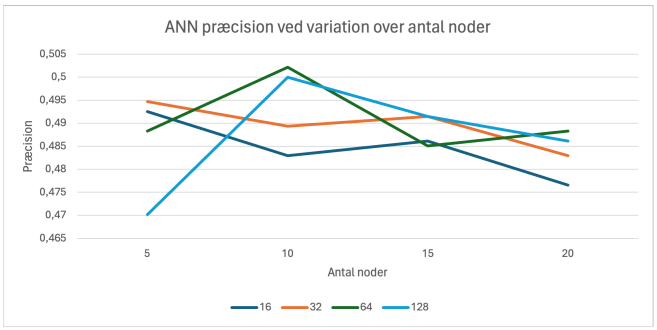
Fordeling af inputvariable for kategorier i træningsdata

	Kategori 0	Kategori 1	Kategori 2	Kategori 3	Kategori 4	Kategori 5
MACD Kategori 0	202	411	375	354	416	189
MACD Kategori 1	3	8	4	9	15	7
MACD Kategori 2	7	4	1	6	10	2
MACD Kategori 3	4	4	6	4	7	4
MACD Kategori 4	4	12	8	9	14	5
MACD Kategori 5	171	330	302	303	375	168
RSI Kategori 0	188	352	305	294	355	158
RSI Kategori 1	10	19	23	27	28	13
RSI Kategori 2	7	26	17	24	25	11
RSI Kategori 3	10	10	10	18	16	6
RSI Kategori 4	10	32	21	21	36	19
RSI Kategori 5	166	330	320	301	377	168
Chevron Kategori 0	156	149	71	41	47	16
Chevron Kategori 1	96	219	172	126	104	34
Chevron Kategori 2	55	129	136	124	118	35
Chevron Kategori 3	34	101	119	127	139	48
Chevron Kategori 4	26	121	141	201	273	103
Chevron Kategori 5	24	50	57	66	156	139
BP Kategori 0	242	155	52	14	11	3
BP Kategori 1	95	313	197	113	57	9
BP Kategori 2	25	138	164	172	83	11
BP Kategori 3	13	90	143	166	153	23
BP Kategori 4	11	59	114	188	367	105
BP Kategori 5	5	14	26	32	166	224
Exxon Kategori 0	153	127	56	35	40	20
Exxon Kategori 1	92	237	160	141	130	34
Exxon Kategori 2	43	128	158	115	124	37
Exxon Kategori 3	43	114	122	157	159	48
Exxon Kategori 4	45	119	147	173	254	108
Exxon Kategori 5	15	44	53	64	130	128
S.P.GSCI Kategori 0	102	96	72	56	49	26
S.P.GSCI Kategori 1	103	177	135	119	138	60
S.P.GSCI Kategori 2	72	163	127	142	150	39
S.P.GSCI Kategori 3	57	156	168	167	178	56
S.P.GSCI Kategori 4	42	119	141	140	227	98
S.P.GSCI Kategori 5	15	58	53	61	95	96

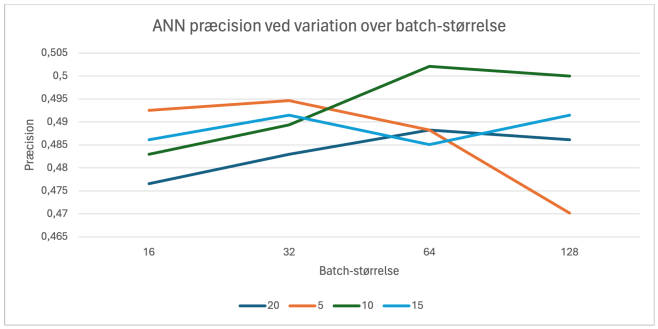
Tabel A.2: Fordeling af inputvariable for kategorier i træningsdata.

B Figurer

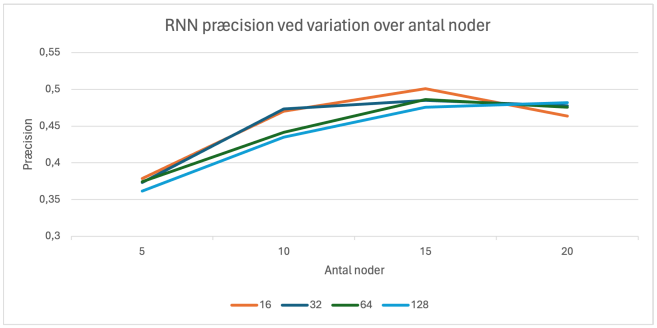
Variation over batch-størrelser og antal noder i de tre netværkstyper



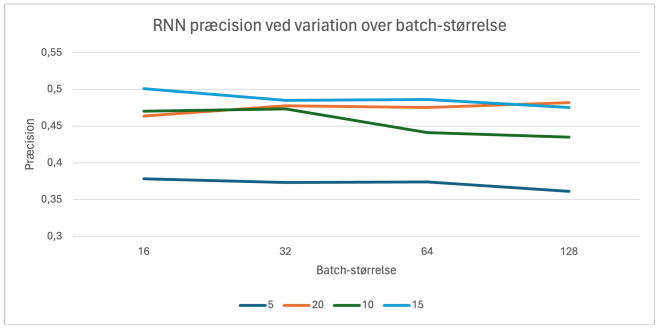
(a) ANN præcision ved variation over antal noder



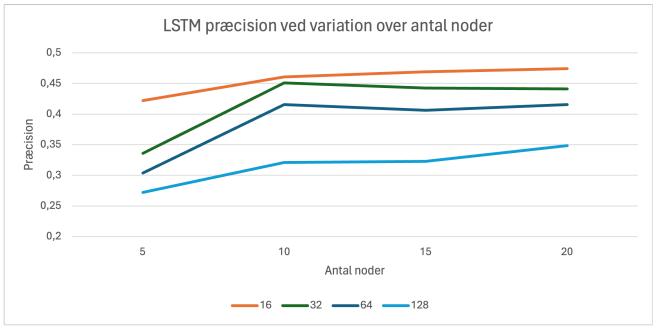
(b) ANN præcision ved variation over batch-størrelse



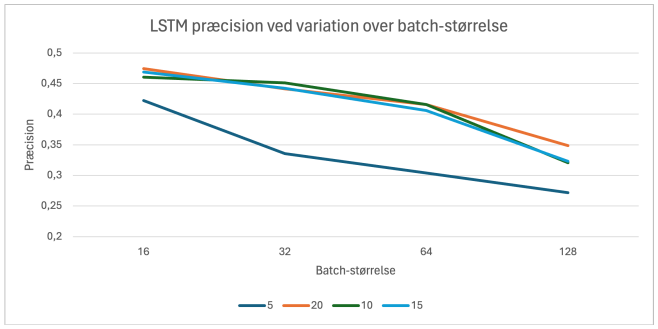
(c) RNN præcision ved variation over antal noder



(d) RNN præcision ved variation over batch-størrelse



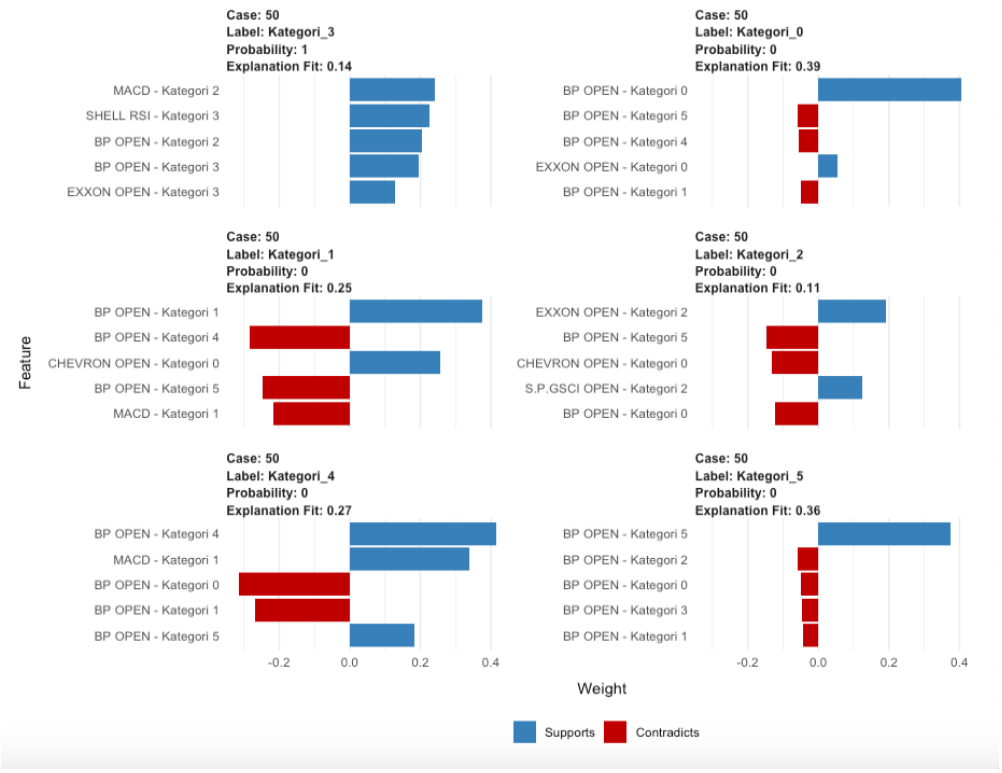
(e) LSTM præcision ved variation over antal noder



(f) LSTM præcision ved variation over batch-størrelse

Figur B.1: (a), (c) og (e) er variationen over antal noder i de respektive netværk. (b), (d) og (f) er variationen over batch-størrelsen i de respektive netværk.

LIME forklaringer med *highest weights*



Figur B.2: LIME forklaring ved *highest weights* metode til valg af variable.

C Kode

Hoveddokument

```
## DATABEHANDLING
PROCESS_DATA <- function(DF, breaks){
  DF$Date <- as.Date(DF$Date, "%d.%m.%Y"); DF <- DF[1:8]
  colnames(DF) <- c("DATE", "SHELL OPEN", "MACD", "SHELL RSI", "CHEVRON OPEN",
                    "BP OPEN", "EXXON OPEN", "S.P.GSCI.TOTAL.RETURN OPEN")
  DF <- na.omit(DF)
  # Categorize the data into multiple categories
  breaks_in_fun <- c(-Inf,breaks, Inf)
  for (i in 2:ncol(DF)) {
    column_name <- colnames(DF)[i]
    new_column_name <- paste(column_name, "MultiCat", sep = "_")
    DF[[new_column_name]] <- cut(DF[[column_name]],
                                 breaks = breaks_in_fun,
                                 labels = c("0", "1", "2", "3", "4", "5"),
                                 include.lowest = TRUE)
    DF[[new_column_name]] <- as.numeric(DF[[new_column_name]]) -1}
  ##PICK DATA
  data <- DF[-(1:8)]
  data <- as.data.frame(data)
  # Apply custom categories
  scaled_data <- data
  scaled_data$MultiCat <- data$MultiCat
  scaled_data <- relocate(scaled_data,"SHELL OPEN_MultiCat", .before = 1)
  scaled_data <- na.omit(scaled_data)
  ##TRAINING AND TEST DATA
  X <- scaled_data[2:ncol(scaled_data)]
  y <- scaled_data[[1]]
  n <- dim(X)[1]
  training_index <- round(n - (n * 0.2))
  temp_df <- matrix(nrow = n)
  for (i in 1:ncol(X)){
    OHE <- to_categorical(X[,i])
    OHE <- as.data.frame(OHE)
    temp_df <- cbind(temp_df, OHE) }
  temp_df <- temp_df[,-1]
  for (i in 1:ncol(X)){
    for(j in 1:6){
      col_index <- (j + (i - 1) * 6)
```

```
      colnames(temp_df)[col_index] <- paste(colnames(X)[i], paste0("cat", j-1)) } }
  X <- temp_df
  X_train <<- (X[1:training_index,])
  y_train <<- to_categorical(y[1:training_index])
  X_test <<- (X[(training_index + 1): n,])
  y_test <<- to_categorical(y[(training_index + 1): n])
  y_testtable <<- y[(training_index + 1): n]
  #LABELS
  col_names <- colnames(X_train)
  X_train <<- array(as.matrix(X_train), dim = c(nrow(X_train), ncol(X_train), 1))
  colnames(X_train) <<- col_names
  X_test <<- array(as.matrix(X_test), dim = c(nrow(X_test), ncol(X_test), 1))
  colnames(X_test) <<- col_names
  colnames(y_train) <<- c(1:6)
  colnames(y_test) <<- c(1:6)      } #End of function
## PLOT GRAFER
PLOT_METRICS <- function(val_value, org_value, legends, ylab){
  #MIN_VALUES
  min_value_val_value <- min(val_value)
  min_value_org_value <- min(org_value)
  #DETERMINE WHICH LIST CONTAINS THE SMALLEST VALUE
  if(min_value_val_value <= min_value_org_value){
    min_list <- val_value
  } else {
    min_list <- org_value}
  #MAX VALUE
  max_value_val_value <- max(val_value); max_value_org_value <- max(org_value)
  #DETERMINE WHICH LIST CONTAINS THE LARGEST VALUE
  if(max_value_val_value >= max_value_org_value){
    max_list <- val_value
  } else {
    max_list <- org_value }
  plot(val_value, type = "l", col = "blue", xlab = "Epochs", ylab = ylab,
       ylim = c(min(min_list) * 0.99, max(max_list) * 1.02))
  lines(org_value, type = "l", col = "darkorange")        } #End of function
##LIBRARIES
library(tidyverse)
library(caret)
library(dplyr)
library(stringr)
library(timetk)
```

```
library(reticulate)
library(tensorflow)
library(keras)
library(tfruns)
library(tfestimators)
library(lime)
##SET SEED IN TENSORFLOW/KERAS
set.seed(123); tensorflow::set_random_seed(123)
## DATA
wd <- dirname(rstudioapi::getSourceEditorContext()$path); setwd(wd)
DF <- read.csv("FINALFINAL.csv", sep = ";", header = TRUE)
breaks <- c(-0.015, -0.005, 0, 0.005, 0.015)
PROCESS_DATA(DF = DF, breaks = breaks)
## TUNING
#LOOP VARIABLE
nodes <- c(5, 10, 15, 20); batch_size <- c(16, 32, 64, 128)
loop1_NODES <- length(nodes); loop2_BATCH <- length(batch_size)
total_epochs <- 5
RESULTS_ANN <- matrix(data = NA, nrow = (loop1_NODES*loop2_BATCH) , ncol = 3)
colnames(RESULTS_ANN) <- c("PARAM_CHOICES", "ANN_LOSS", "ANN_ACC")
RESULTS_RNN <- matrix(data = NA, nrow = (loop1_NODES*loop2_BATCH) , ncol = 3)
colnames(RESULTS_RNN) <- c("PARAM_CHOICES", "RNN_LOSS", "RNN_ACC")
RESULTS_LSTM <- matrix(data = NA, nrow = (loop1_NODES*loop2_BATCH) , ncol = 3)
colnames(RESULTS_LSTM) <- c("PARAM_CHOICES", "LSTM_LOSS", "LSTM_ACC")
for (i in 1:loop1_NODES) {for (j in 1:loop2_BATCH) {
    print(nodes[i]); print(batch_size[j])
    # ANN
    my_model_ANN <- keras_model_sequential()
    my_model_ANN %>%
      layer_dense(units = nodes[i], activation = "relu", input_shape = dim(X_train)[2],
                  kernel_initializer=initializer_glorot_uniform(seed = 123)) %>%
      layer_dense(units = 6, activation = 'softmax',
                  kernel_initializer = initializer_glorot_uniform(seed=123))
    my_model_ANN %>% compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy',
                             metrics = 'categorical_accuracy')
    mymodel_ANN <- my_model_ANN %>% fit(x = X_train, y = y_train, batch_size = batch_size[i],
      epochs = total_epochs, shuffle = FALSE, validation_split = 0.20)
    RESULTS_ANN[(i*loop2_BATCH+j-loop2_BATCH), 1] <- paste(nodes[i], batch_size[j], sep = ",")
    RESULTS_ANN[(i*loop2_BATCH+j-loop2_BATCH), 2:3] <- my_model_ANN %>% evaluate(X_test, y_test)
    path <- paste0(wd, "/", "Nodes_", nodes[i], "_Batch_size_", batch_size[j],
                                              "_Plot_", i, "_", j, "_ANN.png")
```

```
print(path); png(path, width = 690, height = 745)
#PLOTS ALL THE METRICS INTO A GRID
par(mfrow = c(2,1), mar=c(5, 5, 3, 2)); par(cex.lab=2, cex.axis=2)
PLOT_METRICS(val_value = mymodel_ANN$metrics$val_loss, org_value = mymodel_ANN$metrics$loss,
             legends = c("Validation Loss", "Loss"), ylab = "Loss")
legend("topright", legend = c("Validation", "Training"), col = c("blue", "darkorange"),
       lwd = 2, cex = 2)
PLOT_METRICS(val_value = mymodel_ANN$metrics$val_categorical_accuracy,
             org_value = mymodel_ANN$metrics$categorical_accuracy,
             legends = c("Validation ACC", "ACC"), ylab = "ACC")
dev.off()
Sys.sleep(1)
#RNN
my_model_RNN <- keras_model_sequential()
my_model_RNN %>%
  layer_simple_rnn(units = nodes[i], activation = "tanh", input_shape = dim(X_train)[2:3],
            kernel_initializer=initializer_glorot_uniform(seed = 123),
            recurrent_initializer = initializer_orthogonal(seed = 123)) %>%
  layer_dense(units = 6, activation = 'softmax',
            kernel_initializer = initializer_glorot_uniform(seed=123))
my_model_RNN %>% compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy',
                         metrics = 'categorical_accuracy')
mymodel_RNN <- my_model_RNN %>% fit(x = X_train, y = y_train, batch_size = batch_size[i],
  epochs = total_epochs, shuffle = FALSE, validation_split = 0.20)
RESULTS_RNN[(i*loop2_BATCH+j-loop2_BATCH), 1] <- paste(nodes[i], batch_size[j], sep = ",")
RESULTS_RNN[(i*loop2_BATCH+j-loop2_BATCH), 2:3] <- my_model_RNN %>% evaluate(X_test, y_test)
path <- paste0(wd, "/", "Nodes_", nodes[i], "_Batch_size_", batch_size[j],
                                            "_Plot_", i, "_", j, "_RNN.png")
print(path); png(path, width = 690, height = 745)
#PLOTS ALL THE METRICS INTO A GRID
par(mfrow = c(2,1), mar=c(5, 5, 3, 2)); par(cex.lab=2, cex.axis=2)
PLOT_METRICS(val_value = mymodel_RNN$metrics$val_loss, org_value = mymodel_RNN$metrics$loss,
             legends = c("Validation Loss", "Loss"), ylab = "Loss")
legend("topright", legend = c("Validation", "Training"), col = c("blue", "darkorange"),
       lwd = 2, cex = 2)
PLOT_METRICS(val_value = mymodel_RNN$metrics$val_categorical_accuracy,
             org_value = mymodel_RNN$metrics$categorical_accuracy,
             legends = c("Validation ACC", "ACC"), ylab = "ACC")
dev.off()
Sys.sleep(1)
#LSTM
```

```
    my_model_LSTM <- keras_model_sequential()
    my_model_LSTM %>%
      layer_lstm(units = nodes[i], activation = "tanh", input_shape = dim(X_train)[2:3],
                 kernel_initializer=initializer_glorot_uniform(seed = 123),
                 recurrent_initializer = initializer_orthogonal(seed = 123)) %>%
      layer_dense(units = 6, activation = 'softmax',
                  kernel_initializer = initializer_glorot_uniform(seed=123))
    my_model_LSTM %>% compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy',
                              metrics = 'categorical_accuracy')
    mymodel_LSTM <- my_model_LSTM %>% fit(x = X_train, y = y_train, batch_size = batch_size[i],
      epochs = total_epochs, shuffle = FALSE, validation_split = 0.20)
    RESULTS_LSTM[(i*loop2_BATCH+j-loop2_BATCH), 1] <- paste(nodes[i], batch_size[j], sep = ",")
    RESULTS_LSTM[(i*loop2_BATCH+j-loop2_BATCH),2:3] <- my_model_LSTM %>% evaluate(X_test,y_test)
    path <- paste0(wd, "/", "Nodes_", nodes[i], "_Batch_size_", batch_size[j],
                                              "_Plot_", i, "_", j, "_LSTM.png")
    print(path); png(path, width = 690, height = 745)
    #PLOTS ALL THE METRICS INTO A GRID
    par(mfrow = c(2,1), mar=c(5, 5, 3, 2)); par(cex.lab=2, cex.axis=2)
    PLOT_METRICS(val_value=mymodel_LSTM$metrics$val_loss, org_value=mymodel_LSTM$metrics$loss,
                 legends = c("Validation Loss", "Loss"), ylab = "Loss")
    legend("topright", legend = c("Validation", "Training"), col = c("blue", "darkorange"),
           lwd = 2, cex = 2)
    PLOT_METRICS(val_value = mymodel_LSTM$metrics$val_categorical_accuracy,
                 org_value = mymodel_LSTM$metrics$categorical_accuracy,
                 legends = c("Validation ACC", "ACC"), ylab = "ACC")
    dev.off()
    Sys.sleep(1)  } #End of first loop     } #End of second loop
RESULTS_ANN; RESULTS_RNN; RESULTS_LSTM
## YDERLIGERE TUNING  # OBS YOU NEED THE FILE "TESTER HYPER.R" TO RUN THIS CODE
runs <- tuning_run(paste0(wd,"/","KerasTuneR test.R"),runs_dir = "ANN_tuning_1_layers",
  flags = list("NODES" = c(10, 15, 20, 25), "BATCHSIZE" = c(16, 32, 64, 128),
    "LEARNING_RATE" = seq(0.0001, 0.01, 0.0025), "DROPOUT" = c(0.2, 0.4) ) )
### LIME
# Function to replace feature codes (e.g., "V23") with corresponding feature names
replace_feature_names <- function(desc_vector, feature_vector) {
  # Extracting the number after "V" in each element of desc_vector
  num <- as.numeric(gsub("V", "", desc_vector))
  # Using the number as an index to get corresponding feature name
  replaced_names <- feature_vector[num]
  return(replaced_names) }
#MAKE INFERENCE
```

```
model_type.keras.engine.sequential.Sequential <- function(x, ...){"classification"}
predict_model.keras.engine.sequential.Sequential <- function(x, newdata, type, ...){
  if(class(newdata) == "data.frame"){
    newdata <- as.matrix(newdata)
    newdata <- apply(newdata ,2, function(x) ifelse(x < 0.5, 0, 1))
    dim(newdata) <- c(dim(newdata)[1], dim(newdata)[2], 1)
    newdata <<- newdata}
  pred <- predict(object = x, x = newdata) #NEW
  predicted_cats <- as.vector(k_argmax(pred))
  df <- data.frame(Kategori_0 = as.integer(predicted_cats == "0"),
                   Kategori_1 = as.integer(predicted_cats == "1"),
                   Kategori_2 = as.integer(predicted_cats == "2"),
                   Kategori_3 = as.integer(predicted_cats == "3"),
                   Kategori_4 = as.integer(predicted_cats == "4"),
                   Kategori_5 = as.integer(predicted_cats == "5")  ) }
#CHANGE COL-NAMES TO V1,V2,ETC. AS lime::explain CANNOT TAKE REAL COL-NAMES
feature_desc <- paste0("V", 1:36)
features <- c(
  "MACD - Kategori 0","MACD - Kategori 1","MACD - Kategori 2","MACD - Kategori 3",
  "MACD - Kategori 4","MACD - Kategori 5","SHELL RSI - Kategori 0",
  "SHELL RSI - Kategori 1","SHELL RSI - Kategori 2","SHELL RSI - Kategori 3",
  "SHELL RSI - Kategori 4","SHELL RSI - Kategori 5","CHEVRON OPEN - Kategori 0",
  "CHEVRON OPEN - Kategori 1","CHEVRON OPEN - Kategori 2","CHEVRON OPEN - Kategori 3",
  "CHEVRON OPEN - Kategori 4","CHEVRON OPEN - Kategori 5","BP OPEN - Kategori 0",
  "BP OPEN - Kategori 1","BP OPEN - Kategori 2","BP OPEN - Kategori 3",
  "BP OPEN - Kategori 4","BP OPEN - Kategori 5","EXXON OPEN - Kategori 0",
  "EXXON OPEN - Kategori 1","EXXON OPEN - Kategori 2","EXXON OPEN - Kategori 3",
  "EXXON OPEN - Kategori 4","EXXON OPEN - Kategori 5","S.P.GSCI OPEN - Kategori 0",
  "S.P.GSCI OPEN - Kategori 1","S.P.GSCI OPEN - Kategori 2",
  "S.P.GSCI OPEN - Kategori 3","S.P.GSCI OPEN - Kategori 4","S.P.GSCI OPEN - Kategori 5")
X_test <- array(as.matrix(X_test), dim = c(nrow(X_test), ncol(X_test), 1))
X_train <- array(as.matrix(X_train), dim = c(nrow(X_train), ncol(X_train), 1))
#OBS: CHANGE 'my_model_ANN' to the appropriate name
predict_model(x = my_model_ANN, newdata = X_test, type = "prob") %>% tibble::as.tibble()
explainer <- lime::lime(x = as.data.frame(X_train), model = my_model_ANN,
                        bin_continuous = FALSE, quantile_bins = FALSE)
explanation <- lime::explain(as.data.frame(X_test[889:938,,]), explainer = explainer,
                     n_labels = 6, n_features = 5, feature_select = "highest_weights")
#REPLACE THE VXX WITH THE CORRESPONDING FEATURE-NAME FROM THE DATA
replaced_names <- replace_feature_names(explanation$feature_desc, features)
explanation$feature_desc <- replaced_names
```

```
#PLOT
plot_features(explanation, case = "50") #CASE-BY-CASE PLOT
plot_explanations(explanation) #CLASS-BY-CLASS PLOT
tail(y_test) #test-data
#SHOW HOW MANY TIMES A VARIABLE GROUP (SUBGROUP) HAS BEEN TOP 5 INFLUENTIAL FEATURES
phrases <- c("BP", "MACD", "RSI", "CHEVRON", "EXXON", "S.P.GSCI")
counts_df <- data.frame(matrix(nrow = 1, ncol = length(phrases)))
counts_df2 <- data.frame(matrix(nrow = 1, ncol = length(features)))
colnames(counts_df2) <- features; colnames(counts_df) <- phrases
for (j in seq_along(phrases)) {
  counts_df[,j] <- sum(str_count(explanation$feature_desc, fixed(phrases[j])))   }
counts_df
for (j in seq_along(features)) {
  counts_df2[,j] <- sum(str_count(explanation$feature_desc, fixed(features[j]))) }
t(counts_df2)
```

Ekstra kodedokument til "TESTER HYPER.R"

```
FLAGS <- flags(flag_numeric("NODES", 20), flag_numeric("BATCHSIZE", 128),
        flag_numeric("LEARNING_RATE", 0.0026), flag_numeric("DROPOUT", 0.4))
my_model_ANN <- keras_model_sequential()
my_model_ANN %>% layer_dense(
              units = FLAGS$NODES, activation = "relu", input_shape = dim(X_train)[2],
              kernel_initializer=initializer_glorot_uniform(seed = 123)) %>%
  layer_dropout(FLAGS$DROPOUT) %>%
# layer_dense(units = FLAGS$NODES, activation = "relu", input_shape = dim(X_train)[2],
#      kernel_initializer=initializer_glorot_uniform(seed = 123)) %>% #Adds extra layer
# layer_dropout(FLAGS$DROPOUT) %>% # Adds extra dropout
# layer_dense(units = FLAGS$NODES, activation = "relu", input_shape = dim(X_train)[2],
#      kernel_initializer=initializer_glorot_uniform(seed = 123)) %>% #Adds extra layer
# layer_dropout(FLAGS$DROPOUT) %>% #Adds extra dropout
  layer_dense(units = 6, activation = 'softmax',
              kernel_initializer = initializer_glorot_uniform(seed=123)) %>%
  compile(optimizer = optimizer_rmsprop(learning_rate = FLAGS$LEARNING_RATE),
    loss = 'categorical_crossentropy',metrics = 'categorical_accuracy') %>%
  fit(x = X_train, y = y_train, batch_size = FLAGS$BATCHSIZE,
           epochs = total_epochs, shuffle = FALSE, validation_split = 0.20)
temp <- my_model_ANN %>% evaluate(X_test, y_test)
name <- paste0("Nodes_",FLAGS$NODES,"_Batch_size_",FLAGS$BATCHSIZE,"_LEARNING_RATE_"
               ,FLAGS$LEARNING_RATE,"_Dropout_",FLAGS$DROPOUT,"_ANN")
write.csv(temp, name)
```