Het backpropagation Kalman filter algoritme
Een reproduceerbaarheidstudie

Tibo Van den Eede[1] , **Sam Vervaeck**[1]
[1]KU Leuven
{tibo.vandeneede, sam.vervaeck}@student.kuleuven.be

Abstract

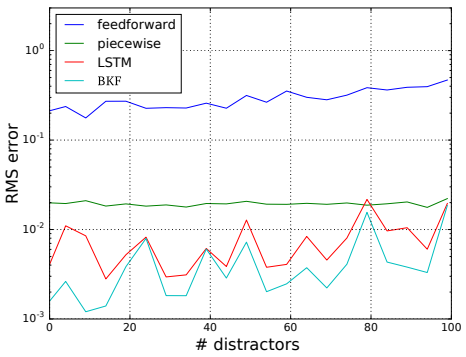
Deze paper behandelt een studie naar de reproductie van de paper *Backprop KF: Learning Discriminative Deterministic State Estimators* [Haarnoja *et al.*, 2016] waarin een discriminatief model naar voren wordt geschoven dat efficiënt een toestandschatting kan doen op een grote invoer. De methode wordt op basis van de beschrijvingen in die paper geïmplementeerd en het tracking experiment wordt gereproduceerd. Op basis daarvan wordt nagegaan of dezelfde resultaten bekomen kunnen worden.

1 Introductie

In recente jaren is de moeilijkheid om papers over machine learning te reproduceren steeds vaker ter sprake gekomen. Onder meer doordat de technieken minder algemeen toepasbaar blijken te zijn dan geclaimd wordt en door de complexe details van machine learning [Kapoor and Narayanan, 2021]. Er zijn namelijk veel aspecten, zoals welke hyperparameters er gebruikt worden, de grootte van de test en validatie datasets, hoe fouten exact berekend worden en nog veel meer die een grote impact kunnen hebben op de bekomen resultaten [Gundersen *et al.*, 2022]. In deze paper wordt een studie gedaan naar de reproduceerbaarheid van de *Backprop KF: Learning Discriminative Deterministic State Estimators* paper [Haarnoja *et al.*, 2016]. Daarin wordt een nieuwe methode voorgesteld om toestandschattingen, zoals locatie en snelheid, te maken door middel van de combinatie van een Kalman filter met een neuraal netwerk.

Nauwkeurige toestandschattingen zijn belangrijk in verschillende domeinen zoals de ruimtevaart, fabrieksrobots en zelfrijdende voertuigen [Thrun *et al.*, 2005]. Daarvoor bestaan verschillende klassen van algoritmes, waaronder generatieve en dicriminatieve modellen. Generatieve algoritmes zijn gelimiteerd bij gebruik met grote invoerdata zoals afbeeldingen, want ze moeten de kansverdeling over heel de invoer berekenen. Aangezien voor de resolutie van afbeeldingen over megapixels gesproken wordt, zouden miljoenen pixels verwerkt moeten worden voor elke invoer. Discriminatieve algoritmes hebben dat nadeel niet. Ze zijn echter complexer, aangezien ze meer parameters bevatten, die getraind moeten worden om al de eigenschappen te herkennen. Als het bestudeerde systeem beschreven kan worden door een lineair model en er witte, Gaussische ruis op de observaties voorkomt, dan kan er aangetoond worden dat het gebruik van een Kalman filter tot een optimale schatting leidt [Maybeck, 1979].

Het backpropagation Kalman filter algoritme (BKF) belooft eenvoudig te trainen te zijn door een deterministische berekeningsgrafe op te bouwen die eind-tot-eind geoptimaliseerd kan worden door middel van backpropagation en dalende gradiëntmethodes. De twee grote bouwblokken van BKF zijn een feedforward convolutioneel neuraal netwerk en een Kalman filter. De implementatie in de originele paper is erop gericht om een afbeelding als invoer te nemen en een toestandschatting, zoals de locatie in coördinaten, als uitvoer te geven. Door de invoer eerst door het neuraal netwerk te halen, voordat het aan de Kalman filter gegeven wordt, is het mogelijk om grote data zoals afbeeldingen te gebruiken. De auteurs testen BKF op een eigen synthetische tracking dataset, zie Figuur 5, en de *KITTI* dataset [Geiger *et al.*, 2013]. Voor de tracking dataset werd het resultaat in Figuur 1 behaald. Op basis van die resultaten wordt geclaimd dat BKF een nauwkeurigere voorspelling doet dan een feedforward netwerk.



Figuur 1: Figuur uit de originele paper met de RMSE voor de verschillende geteste modellen. Alle modellen waren getraind op 1000 sequenties van 100 afbeeldingen [Haarnoja *et al.*, 2016, p. 6]

Om de reproduceerbaarheid van die claim na te gaan, levert deze studie hoofdzakelijk de twee volgende bijdragen: een open source herimplementatie van BKF in PyTorch en een vergelijking met de originele resultaten door middel van die herimplementatie. Er zal blijken dat de originele paper ver-

schillende, belangrijke details mist voor het reproduceren van de resultaten.

1.1 Relevante links

Een andere studie naar de reproduceerbaarheid van de BKF paper behandelt het *KITTI* experiment [Chen and Lin, 2019].

De code van de originele paper is niet publiekelijk beschikbaar. De code van deze herimplementatie is dus geschreven op basis van de beschrijvingen in de originele paper en is te vinden op: https://github.com/tiboat/BackpropKF_Reproduction. Er wordt eveneens naar verwezen op de Papers With Code pagina van de originele paper: https://paperswithcode.com/paper/backprop-kf-learning-discriminative.

2 Probleemstelling

De originele paper beschrijft het BKF algoritme en voert er experimenten mee uit op twee verschillende datasets. In deze studie naar de reproduceerbaarheid wordt gefocust op het experiment met de synthetische tracking dataset. Daarbij wordt er een afbeelding van 128 pixels breed op 128 pixels hoog met rode, groene en blauwe kleurkanalen (RGB) als invoer gebruikt, zie Figuur 5. Elke afbeelding bevat een rode cirkel waarvan het middelpunt de toestand van het systeem is dat BKF moet volgen. De uitvoer is dus een vector met de schatting van de x- en y-coördinaat van dat middelpunt. Daarvoor leidt het feedforward netwerk eerst een tussentijdse schatting af uit de ruwe invoerdata van de afbeelding en geeft dat samen met de onzekerheid van die schatting door aan het Kalman filter gedeelte. Daarna maakt de Kalman filter gebruik van informatie over de toestand van het systeem in de vorige tijdstap om die schatting te verbeteren tot de uiteindelijke uitvoer.

Uit het bovenstaande zijn de volgende onderzoeksvragen af te leiden:

V1: Is het BKF netwerk duidelijk beschreven zodat het exact geherimplementeerd kan worden?

V2: Zijn de gebruikte datasets duidelijk beschreven zodat ze opnieuw gegenereerd kunnen worden?

V3: Komen de resultaten bij het heruitvoeren van de experimenten overeen met de originele resultaten?

V4: Zorgt het toevoegen van een Kalman filter aan een feedforward convolutioneel neuraal netwerk inderdaad voor een nauwkeurigere toestandschatting?

V5: Kan BKF inderdaad eind-tot-eind getraind worden?

V6: Geeft BKF een even nauwkeurige toestandschatting op een complexere dataset?

3 Achtergrondkennis

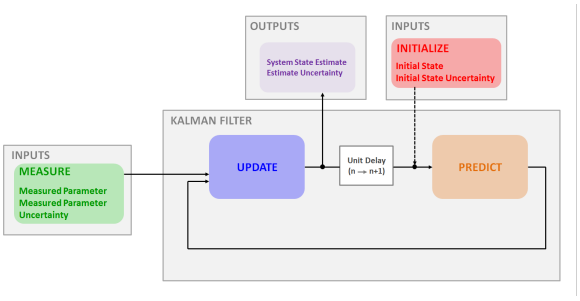
Om de onderzoeksvragen te kunnen beantwoorden, moet eerst de volgende theorie begrepen worden over de twee bouwblokken van BKF: de Kalman filter en het feedforward netwerk.

3.1 Kalman filter

Een Kalman filter (KF) is een recursief dataverwerkingsalgoritme, waarbij recursief inhoudt dat het niet alle voorgaande data opnieuw moet verwerken als er een nieuwe observatie bij komt. Het wordt gebruikt om de toestand $\mathbf{x}_t$ van een gemodelleerd systeem doorheen de tijd t optimaal te schatten, waarbij drie assumpties worden gemaakt: 1) het model is lineair, 2) de observatieruis is wit en 3) de observatieruis is Gaussisch. Het gebruik van een lineair model is meestal nauwkeurig genoeg voor de gewenste berekeningen. Zo niet, kan een deel van een niet-lineair model meestal met een acceptabele afwijking gelineariseerd worden. Witte ruis betekent dat de afwijking op één moment geen informatie geeft over de afwijking op een ander moment. Ze zijn niet gecorreleerd. De enige invloed van die assumptie valt buiten het systeem, waar het geen invloed heeft. Ten slotte is de Gaussische kansdichtheid een correcte verwachting, aangezien de som van verschillende onafhankelijke variabelen er nauwkeurig door benaderd wordt [Maybeck, 1979].

De toestand $\mathbf{x}_t$ is een vector. Bijvoorbeeld voor Figuur 5 kan $\mathbf{x}_t$ gelijk zijn aan $(x\ y\ v_x\ v_y)$ met x en y de x- en y-coördinaat van de locatie van de rode cirkel en v_x en v_y de snelheid in x- en y-richting. Hierbij maakt de KF gebruik van drie zaken:

- Een dynamisch model van het systeem. Dat is een verzameling van functies die de tijdsafhankelijkheid van de onderdelen van het systeem beschrijven.
- Als invoer een reeks van metingen $\mathbf{z}_t$ in de tijd met potentiële ruis en andere onnauwkeurigheden. Dat kunnen bijvoorbeeld metingen van de locatie zijn: $\mathbf{z}_t = (x_t\ y_t)$
- Als uitvoer een reeks vectoren $\boldsymbol{\mu}_{\mathbf{x}_t}$, de schattingen van de gezochte toestanden $\mathbf{x}_t$, en covariantiematrices $\boldsymbol{\Sigma}_{\mathbf{x}_t}$, de onzekerheden van die schattingen.



Figuur 2: Een schema van de verschillende stappen van een Kalman filter [Becker, 2022a]

Om van de invoer naar de uitvoer te gaan, gebruikt de KF een iteratief algoritme, gegeven in Algoritme 1. Elke iteratie is onder te verdelen in twee stappen, zie Figuur 2: voorspelling (predict) en correctie (update). In de voorspellingsstap wordt een voorspelling gemaakt van de toestand op een tijdstip t op basis van het dynamisch model. Meer specifiek houdt het regels 2 en 3 van Algoritme 1 in:

2: De toestand dynamische update: $\boldsymbol{\mu}_{\mathbf{x}_{t-1}}$, de toestandsvoorspelling van de vorige tijdstap $(t-1)$, wordt vermenigvuldigd met $\mathbf{A}$, de toestandsovergangsmatrix van het dynamisch model, om $\boldsymbol{\mu}'_{\mathbf{x}_t}$, een voorspelling voor de toestand op tijdstip t, te extrapoleren.

3: De covariantie dynamische update: $\boldsymbol{\Sigma}_{\mathbf{x}_{t-1}}$, de onzekerheid van de toestandsvoorspelling van de vorige tijdstap $(t-1)$, wordt vermenigvuldigd met $\mathbf{A}$, de toestandsovergangsmatrix van het dynamisch model, om $\boldsymbol{\Sigma}'_{\mathbf{x}_t}$, de onzekerheid van de voorspelling voor de toestand op tijdstip t, te extrapoleren. Daar wordt $\mathbf{Q}$, de covariantiematrix overeenkomend met de ruis van het dynamisch model, nog bij opgeteld.

In de correctiestap worden die voorspelling en een meting opgeteld met gewichten om een optimale toestandschatting te bekomen. Dat gewicht wordt bepaald door de Kalman gain: $\mathbf{K}_t = \frac{onzekerheid\ voorspelling}{onzekerheid\ voorspelling + onzekerheid\ meting}$. De verhouding tussen de onzekerheden zal dus bepalen of er meer gewicht aan de voorspelling of de meting gegeven wordt. Meer specifiek houdt het regels 4, 5 en 6 van Algoritme 1 in:

4: De Kalman gain berekening: $\mathbf{K}_t$, de Kalman gain, wordt berekend door middel van $\boldsymbol{\Sigma}'_{\mathbf{x}_t}$, de onzekerheid van de voorspelling, en $\mathbf{R}_t$, de onzekerheid van de meting.

5: De toestand observatie update: $\boldsymbol{\mu}_{\mathbf{x}_t}$, de toestandsschatting, is het resultaat van de gewogen som van $\boldsymbol{\mu}'_{\mathbf{x}_t}$, de voorspelling, $\mathbf{z}_t$, de observatie. Daarbij krijgt $\boldsymbol{\mu}'_{\mathbf{x}_t}$ als gewicht $1 - \mathbf{K}_t$ en $\mathbf{z}_t$ als gewicht $\mathbf{K}_t$. Als de onzekerheid van de voorspelling dus groot is ten opzichte van die van de observatie zal $\mathbf{K}_t$ dus dicht bij 1 aanzitten en $\boldsymbol{\mu}'_{\mathbf{x}_t}$ een klein gewicht hebben. $\mathbf{z}_t$ zal dan een groter gewicht hebben, omwille van de hogere zekerheid. $\mathbf{C_z}$ is de observatiematrix die $\boldsymbol{\mu}'_{\mathbf{x}_t}$ omzet naar de dimensie van $\mathbf{z}_t$. Zo kan $\mathbf{C_z} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$ genomen worden om enkel de x- en y-coördinaat over te houden.

6: De covariantie observatie update: uit de toestand dynamische update en de toestand observatie update kan de formule voor $\boldsymbol{\Sigma}_{\mathbf{x}_t}$ afgeleid worden [Becker, 2022b].

Algoritme 1 Kalman filter

Invoer: $\mathbf{z}_{1:T}, \mathbf{R}_{1:T}, \boldsymbol{\Sigma}_{\mathbf{x}_0}, \boldsymbol{\mu}_{\mathbf{x}_0}$
Parameters: $\mathbf{A}, \mathbf{C_z}, \mathbf{Q}$
Uitvoer: $\boldsymbol{\mu}_{\mathbf{x}_{0:T}}$

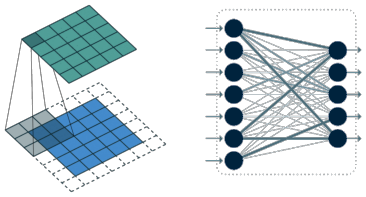
for $t \leftarrow 1$ **tot** T **do**
  $\boldsymbol{\mu}'_{\mathbf{x}_t} \leftarrow \mathbf{A}\boldsymbol{\mu}_{\mathbf{x}_{t-1}}$
  $\boldsymbol{\Sigma}'_{\mathbf{x}_t} \leftarrow \mathbf{A}\boldsymbol{\Sigma}_{\mathbf{x}_{t-1}}\mathbf{A}^\top + \mathbf{Q}$
  $\mathbf{K}_t \leftarrow \boldsymbol{\Sigma}'_{\mathbf{x}_t}\mathbf{C}_\mathbf{z}^\top \left(\mathbf{C_z}\boldsymbol{\Sigma}'_{\mathbf{x}_t}\mathbf{C}_\mathbf{z}^\top + \mathbf{R}_t\right)^{-1}$
  $\boldsymbol{\mu}_{\mathbf{x}_t} \leftarrow \boldsymbol{\mu}'_{\mathbf{x}_t} + \mathbf{K}_t\left(\mathbf{z}_t - \mathbf{C_z}\boldsymbol{\mu}'_{\mathbf{x}_t}\right)$
  $\boldsymbol{\Sigma}_{\mathbf{x}_t} \leftarrow \left(\mathbf{I} - \mathbf{K}_t\mathbf{C_z}\right)\boldsymbol{\Sigma}'_{\mathbf{x}_t}\left(\mathbf{I} - \mathbf{K}_t\mathbf{C_z}\right)^\top + \mathbf{K}_t\mathbf{R}_t\mathbf{K}_t^\top$
end for
return $\boldsymbol{\mu}_{\mathbf{x}_{0:T}}$

3.2 Feedforward netwerk

Een feedforward netwerk (FFN) is een soort artificieel neuraal netwerk (ANN) waarbij de uitvoer van het model niet opnieuw in een lus gebruikt wordt. Het doel van een ANN is om een bepaalde functie $f(x)$ te benaderen. Daarvoor simuleert het ANN een functie $f(x, \theta)$ met extra parameters θ bij de originele invoer x. Die parameters worden dus zo geleerd opdat $f(x, \theta)$ een zo goed mogelijke benadering is van $f(x)$ voor elke invoer x. Een ANN wordt een netwerk genoemd, omdat $f(x, \theta)$ bestaat uit een sequentie van verschillende functies $f^{(1)}, f^{(2)}, ..., f^{(n)}$, ook wel lagen genoemd, zodat $f = f^{(n)}(f^{(n-1)}(...f^{(1)}))$.

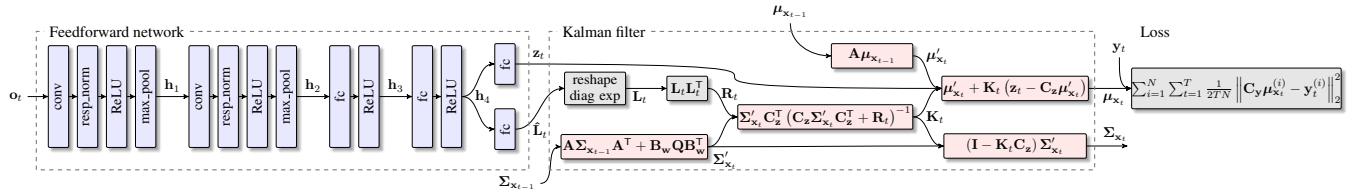
In wat volgt, worden de twee belangrijkste lagen verder uitgelegd: de convolutionele en lineaire laag. Voor de geïnteresseerde lezer is er in appendix A meer uitleg over de ReLU, Layer Normalization, maxpool en flatten laag te vinden [Goodfellow *et al.*, 2016] [Paszke *et al.*, 2019].

Een **convolutionele laag** heeft als doel om eigenschappen uit de ruwe data af te leiden. Het wordt bijvoorbeeld gebruikt om aan rand detectie te doen rond onderwerpen, zoals mensen, in afbeeldingen. Daarvoor wordt er een matrix K met gewichten, de kernel, over de afbeelding matrix A heen geschoven. Op elke positie worden de overlappende elementen vermenigvuldigd en opgeteld, zie Figuur 3. Die sommen komen vervolgens in een nieuwe matrix A^* te staan. Om er voor te zorgen dat A^* dezelfde dimensies heeft als A kan er een rand van elementen met waarde 0 toegevoegd worden, padding genaamd, zodat elk element de kans heeft om onder het midden van K te liggen. Tot slot kan K ook met meer dan 1 element per keer opgeschoven worden, wat de standaard stapgrootte is. Bij een grotere stap zijn de dimensies $n^* \times m^*$ van A^* gelijk aan de dimensies $n \times m$ van A gedeeld door de stapgrootte s: $n^* = (n/s)$, $m^* = (m/s)$.

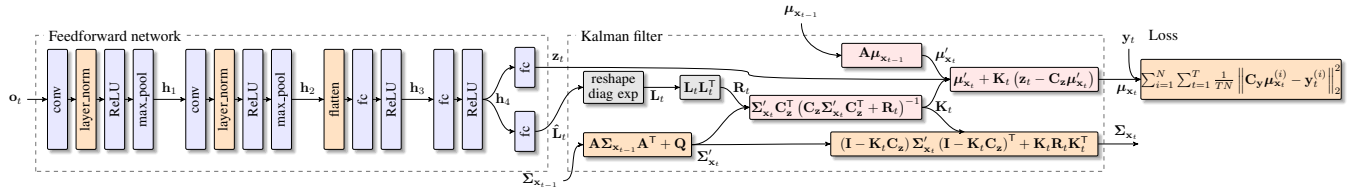


Figuur 3: Links: de werking van een convolutionele laag [Dumoulin and Visin, 2016]. Rechts: de werking van een lineaire fully-connected laag [Kost *et al.*, 2019]

De **lineaire fully-connected lagen** bevinden zich meestal aan het einde van een netwerk en voeren de uiteindelijk classificatie uit van de invoerdata naar een verzameling eigenschappen. Elk element van de invoer en uitvoer van een lineaire fully-connected laag bevat een eigenschap en wordt een neuron genoemd. Elke neuron heeft een gewichten matrix w en een matrix met biases b. Om de uitvoer te bekomen wordt de lineaire transformatie $y_{jk} = \sum_{i=1}^{n} w_{jk}x_i + b_{jk}$ op de invoer uitgevoerd. De grootte van de laatste uitvoer vector bepaalt dus het aantal eigenschappen dat het netwerk uit de originele invoer afleidt.



(a)



(b)

Figuur 4: (a) Illustratie van $\mathrm{BKF}_{\mathbf{o}}$, het BKF netwerk uit de originele paper. (b) Illustratie van $\mathrm{BKF}_{\mathbf{n}}$, het opnieuw geïmplementeerde BKF netwerk uit de originele paper. De aanpassingen zijn in het oranje gemarkeerd. [Haarnoja *et al.*, 2016]

3.3 Backpropagation Kalman filter

BKF is de combinatie van een feedforward neuraal netwerk en een Kalman filter. Eerst wordt een observatie $\mathbf{o}_t$ door het feedforward model gestuurd. Daarmee wordt een laagdimensionale, tussenliggende observatie $\mathbf{z}_t$ samen met een vector $\hat{\mathbf{L}}_t$ bepaald. $\hat{\mathbf{L}}_t$ wordt omgezet naar een observatie covariantiematrix $\mathbf{R}_t$. De $\mathbf{z}_t$ en $\mathbf{R}_t$ worden dan als observatie en observatie covariantiematrix in de KF ingevoerd. Uit de KF wordt een schatting van de toestand $\boldsymbol{\mu}_{\mathbf{x}_t}$ bekomen. Met een lineaire observatie matrix $\mathbf{C}_{\mathbf{y}}$ wordt dan de gewenste schatting $\hat{\mathbf{y}}_t$ bekomen met $\hat{\mathbf{y}}_t = \mathbf{C}_{\mathbf{y}}\boldsymbol{\mu}_{\mathbf{x}_t}$.

Om de nauwkeurigheid van die schatting te verbeteren, wordt het verschil berekent met de daadwerkelijke uitvoer door middel van een verliesfunctie. Hoe kleiner dat verschil gemaakt wordt, hoe hoger de nauwkeurigheid. Daarvoor loopt een optimalisatie functie in de omgekeerde richting doorheen de verschillende lagen van het model, van de uitvoer terug tot aan de invoer. In elke laag worden de parameters aangepast om meer in de richting van lager verlies te sturen. De originele paper maakt gebruik van een optimalisatie functie genaamd *Adaptive Moment Estimation* (Adam) [Kingma and Ba, 2014].

In Figuur 4 (a) wordt de originele architectuur van BKF weergegeven. [Haarnoja *et al.*, 2016]. Voor meer informatie over de inhoud uit de BKF paper kan de lezer zich verder verdiepen in feedforward convolutionele neurale netwerken [Albawi *et al.*, 2017] [O'Shea and Nash, 2015] en Kalman filters [Kalman, 1960] [Welch *et al.*, 1995].

4 Aanpassingen aan BKF

In deze sectie worden de voornaamste aanpassingen besproken van de geïmplementeerde versie van het BKF netwerk $\mathrm{BKF}_{\mathbf{n}}$ ten opzichte van het BKF netwerk van de originele paper $\mathrm{BKF}_{\mathbf{o}}$. De berekeningsgrafen van $\mathrm{BKF}_{\mathbf{o}}$ en $\mathrm{BKF}_{\mathbf{n}}$ zijn respectievelijk terug te vinden in figuur 4 (a) en (b).

4.1 Nieuwe bijdragen

Volgende nieuwe bijdragen werden gemaakt:

- $\mathrm{BKF}_{\mathbf{o}}$ maakt gebruik van een versimpelde formule om de covariantie in de Kalman filter te corrigeren: $(\mathbf{I} - \mathbf{K}_t\mathbf{C}_{\mathbf{z}})\,\boldsymbol{\Sigma}'_{\mathbf{x}_t}$. Die is echter numeriek instabiel als er afrondingsfouten in de berekening van de Kalman gain voorkomen. In $\mathrm{BKF}_{\mathbf{n}}$ is gebruik gemaakt van de geëxpandeerde formule: $(\mathbf{I} - \mathbf{K}_t\mathbf{C}_{\mathbf{z}})\,\boldsymbol{\Sigma}'_{\mathbf{x}_t}\,(\mathbf{I} - \mathbf{K}_t\mathbf{C}_{\mathbf{z}})^{\mathsf{T}} + \mathbf{K}_t\mathbf{R}_t\mathbf{K}_t^{\mathsf{T}}$ [Bucy and Joseph, 2005].
- Voor het trainen van $\mathrm{BKF}_{\mathbf{n}}$ worden de onzekerheden $\hat{\mathbf{L}}_t$ van $\mathbf{z}_t$ geleerd door het feedforward netwerk een eerste keer te trainen en daarbij op basis van de fouten tussen $\mathbf{z}_t$ en het label $\mathbf{y}_t$ een covariantiematrix $\mathbf{R}$ te berekenen voor de hele dataset. Die wordt dan verder gebruikt als label voor $\mathbf{R}_t$. Dat wordt verder uitgelegd in sectie 5.

4.2 Assumpties en aanpassingen aan originele implementatie

Voor de herimplementatie zijn enkele veronderstellingen en aanpassingen gemaakt.

- Appendix B.1 geeft een motivatie voor een aanpassing aan de gebruikte trainingstappen. De nieuwe implementatie komt in Sectie 5 aan bod.
- Voor het feedforward netwerk legt Appendix B.2 enkele veronderstellingen uit, zoals de grootte van de padding in de convolutionele lagen, welke normalisatie te gebruiken en de grootte van de fully-connected lagen.
- Appendix B.3 haalt veronderstellingen over de Kalman filter aan, zoals het omzetten van $\hat{\mathbf{L}}_t$ naar $\mathbf{R}_t$ en de afwezigheid van $\mathbf{B}_{\mathbf{w}}$ in $\mathrm{BKF}_{\mathbf{n}}$.
- In appendix B.4 komen de gebruikte hyperparameters aan bod die niet in de originele paper vermeld worden.

- Tot slot heeft de verliesfunctie van $\mathrm{BKF}_{\mathbf{n}}$ geen factor 2 in de noemer om de definitie van MSE te volgen.

5 Model trainen

In deze sectie wordt het trainingsproces van het nieuwe BKF netwerk $\mathrm{BKF}_{\mathbf{n}}$ (Figuur 4 (b)) voor het synthetische tracking experiment uitgewerkt in vier stappen.

5.1 Stap 1: Feedforward positie

De invoer $\mathbf{o}_t$ is een tensor van dimensie $3 \times 128 \times 128$ overeenkomstig met een afbeelding zoals in Figuur 5. Praktisch gezien wordt een batch van sequenties $\mathbf{o}_{batch}$ meegegeven aan feedforward. Met batch size N en een sequentie lengte T wordt dan een tensor van dimensie $NT \times 3 \times 128 \times 128$ ingevoerd. Als uitvoer wordt $\mathbf{z}_{batch}$ bekomen, welke bestaat uit NT vectoren met twee elementen overeenkomstig met de schattingen van de x- en y-waarde van het centrum van de rode cirkels. Daarmee wordt dan het verlies berekend met de effectieve positie $\mathbf{y}_{batch}$ en op basis daarvan wordt het verlies berekend met de Mean Squared Error Loss (MSE). Bij de laatste epoch worden de voorspellingen $\mathbf{z}_{batch}$ voor alle batches bijgehouden voor stap 2.

5.2 Stap 2: Variantie posities

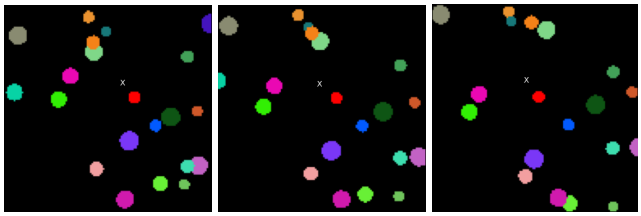
Op basis van de verschillen tussen de voorspellingen $\mathbf{z}_{batch}$ uit stap 1 en de labels $\mathbf{y}_{batch}$ voor alle batches wordt één 2×2 covariantiematrix $\mathbf{R}$ berekend.

5.3 Stap 3: Feedforward positie en variantie

Het feedforward netwerk wordt opnieuw van de grond af getraind analoog aan stap 1 op dezelfde training dataset. Echter worden in deze stap $\mathbf{z}_{batch}$ én $\hat{\mathbf{L}}_{batch}$ bekomen. $\hat{\mathbf{L}}_{batch}$ is een batch van vectoren $\hat{\mathbf{L}}_t$. Elke $\hat{\mathbf{L}}_t$ komt overeen met een afbeelding, heeft 3 elementen en vat de onzekerheid van de voorspelling $\mathbf{z}_t$. Die wordt omgevormd tot $\mathbf{R}_t$, zoals beschreven in Appendix B.3. $\mathbf{R}_t$ is positief semi-definitief (wegens de Cholesky-decompositie $\mathbf{R}_t = \mathbf{L}_t\mathbf{L}_t^{\mathsf{T}}$) [Haarnoja *et al.*, 2016, p. 10], wat noodzakelijk is voor een covariantiematrix. Verder wordt het verlies tijdens de training berekend met zowel $\mathbf{z}_t$ en $\mathbf{R}_t$. Daarbij wordt $\mathbf{y}_t$ gebruikt als label voor $\mathbf{z}_t$ en $\mathbf{R}$, uit stap 2, voor $\mathbf{R}_t$. In de verliesfunctie wordt voor $\mathbf{R}_t$ elk element met 100 vermenigvuldigd zodat de elementen van $\mathbf{R}_t$ ongeveer evenwaardig meetellen als de elementen van $\mathbf{z}_t$. Als verliesfunctie wordt wederom MSE gebruikt en ook wordt het model opgeslagen wanneer het laagste verlies wordt bereikt met de validatie dataset.

5.4 Stap 4: Backpropagation Kalman filter

In de laatste stap wordt het BKF model getraind, waarbij het feedforward gedeelte geïnitialiseerd wordt met de parameters van het getrainde feedforward model uit stap 3. De algemene werking werd besproken in Sectie 3.3. Voor de KF wordt de parameter $\mathbf{A} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ vast gekozen, opdat $\boldsymbol{\mu}'_{\mathbf{x}_t} = \mathbf{A}\boldsymbol{\mu}_{\mathbf{x}_{t-1}} = \mathbf{A}\begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} = \begin{pmatrix} x+v_x \\ y+v_y \\ v_x \\ v_y \end{pmatrix}$. $\mathbf{C}_{\mathbf{z}}$ wordt vast gekozen als $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$, opdat de positie wordt geselecteerd uit $\boldsymbol{\mu}'_{\mathbf{x}_t}$. Verder wordt $\mathbf{Q}$ geïnitialiseerd als diagonaalmatrix met willekeurige diagonaalelementen gegenereerd uit $U(0,1)$. Voor $\boldsymbol{\mu}_{\mathbf{x}_0}$ worden de x- en y-waarde gelijkgesteld aan het label $\mathbf{y}_0$ van de eerste afbeelding van de sequentie en de snelheden v_x en v_y geïnitialiseerd met een willekeurige waarde uit $U(-1,1)$.



Figuur 5: Afbeeldingen 40, 45 en 50 uit een sequentie van 50 van de synthetische tracking dataset met 1 rode cirkel en 24 afleidingscirkels. Het witte kruisje toont de schatting voor de locatie van de rode cirkel door BKF

5.5 Gebruikte datasets tijdens het feedforward en BKF trainen

In de originele paper wordt niet expliciet aangegeven welke datasets gebruikt worden voor de verschillende training-, validatie- en testfases. In de herimplementatie wordt voor het trainen van BKF een andere training dataset van dezelfde grootte gebruikt dan voor het feedforward model. De reden daarvoor is dat de parameters van het feedforward model gebruikt worden om het BKF model te initialiseren. Dezelfde training dataset gebruiken zou er voor zorgen dat BKF die dataset vanbuiten leert. Door een andere training dataset te gebruiken wordt dat vermeden.

6 Set up experimenten

Omwille van de lage instapdrempel voor nieuwe gebruikers is er gekozen voor het PyTorch kader voor Python [Paszke *et al.*, 2019] om BKF in uit te werken. De experimenten worden uitgevoerd op nieuw gegenereerde synthetische datasets. Elke dataset bestaat uit een opeenvolging van afbeeldingen van bewegende cirkels. Het doel is om de positie te bepalen van de rode cirkel, zie in het midden in Figuur 5. Andere cirkels krijgen een willekeurig andere kleur.

6.1 Generatie datasets

Elke afbeelding is opgeslagen als een `torch` tensor met dimensies $(3 \times 128 \times 128)$. Daarbij staat de eerste dimensie voor het aantal kleurkanalen: rood, groen en blauw. De tweede en derde dimensie staan respectievelijk voor de hoogte en breedte van een afbeelding. Een volledige sequentie van 100 afbeeldingen staat in eenzelfde map opgeslagen. Tussen elke opeenvolgende afbeelding is bovendien de tijdstap gelijk met als waarde 1. De cirkel parameters worden met behulp van de NumPy bibliotheek [Harris *et al.*, 2020] uit de volgende uniforme verdelingen gegenereerd:

- De straal r van de cirkel uit $U(3,6)$
- De x en y-coördinaat van het centrum van de cirkel komen beide uit $U(6, 128-6)$

- De snelheid van de cirkel volgens de x-as v_x en volgens de y-as v_y komen beide uit $U(-1, 1)$

In de originele paper worden de exacte verdelingen voor het genereren van de afbeelding niet vermeld. Daarom werden de grenzen van de verdelingen empirisch bepaald, opdat de geproduceerde afbeeldingen visueel overeenstemmen. Daarnaast wordt in de originele paper ook een lineaire veerkracht en weerstandskracht toegevoegd aan de beweging van de cirkels [Haarnoja *et al.*, 2016, p. 5]. Dat werd niet gedaan voor de nagemaakte synthetische datasets. Die maken gebruik van een snelheid die vanaf het begin vast ligt en niet meer verandert.

7 Resultaten nagebootste experimenten

In deze sectie worden de tracking experimenten van de originele paper opnieuw uitgevoerd door middel van een herimplementatie van het feedforward netwerk en BKF. Naast 100 cirkels worden ook 1, 25 en 50 cirkels geëvalueerd om de evolutie van de performantie te onderzoeken. Elk model werd op een training dataset met 100 sequenties van 100 afbeeldingen met 1, 25, 50 of 100 cirkels en een validatie dataset met 50 sequenties van 100 afbeeldingen met 1, 25, 50 of 100 cirkels getraind. Er werd getest op een andere dataset met dezelfde afmeting als die voor validatie. Vervolgens worden de gemiddelde test resultaten met de originele resultaten vergeleken.

7.1 Gebruikte hardware

De experimenten werden uitgevoerd met behulp van volgende hardware componenten:

- GPU: NVIDIA GeForce GTX 1080 Ti met 3584 CUDA kernen en 11 GB GDDR5X V-RAM
- CPU: 2 Intel Xeon E5-2630 processoren met elk 10 fysieke kernen of 20 hyperthreaded virtuele kernen
- RAM: 128 GB

Dankzij de grote hoeveelheid RAM kon de hele dataset in het geheugen geladen worden.

7.2 Originele resultaten

In de originele paper worden de resultaten in Tabel 1 bekomen voor de testen op de synthetische tracking dataset

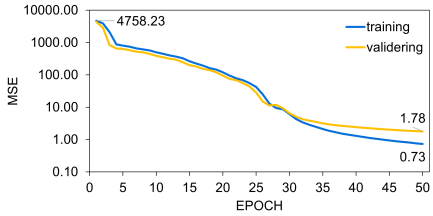
7.3 Nieuwe resultaten feedforward netwerk

1 Cirkel

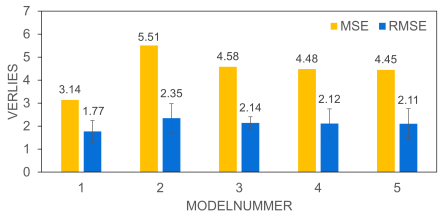
Voor 1 cirkel komt de RMSE het dichtst bij de originele resultaten voor 100 cirkels met een gemiddelde waarde van 2.10 over de vijf modellen heen. Dat is echter nog steeds tien keer slechter dan de originele 0.23 en voor maar één enkele cirkel. Ook de gemiddelde standaardafwijking van 0.53 ligt vijf keer hoger dan de originele 0.13.

Model	# Parameters	RMS test error $\pm\sigma$
Feedforward	7394	0.2322 ± 0.1316
BKF	7493	0.0537 ± 0.1235

Tabel 1: Tabel uit de originele paper met het aantal parameters, de RMSE en standaardafwijking voor de verschillende geteste modellen. Alle modellen waren getraind op 100 sequenties van 100 afbeeldingen [Haarnoja *et al.*, 2016, p. 6]



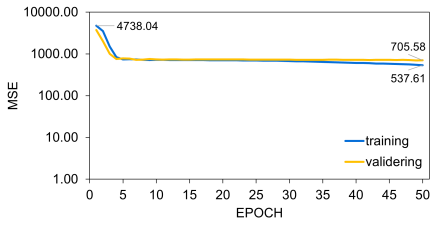
Figuur 6: De gemiddelde evolutie van MSE tijdens het trainen van het feedforward netwerk voor 1 cirkel over 5 trainingen heen.



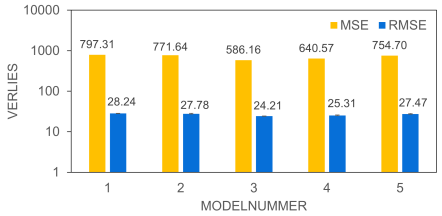
Figuur 7: De MSE en RMSE met standaardafwijking voor vijf verschillende feedforward modellen voor 1 cirkel. Voor alle vijf is dezelfde test dataset gebruikt.

25, 50 en 100 Cirkels

Vanaf 25 cirkels slaagt feedforward er niet meer in tot een nauwkeurige voorspelling te trainen. De resultaten en figuren voor 25 en 50 cirkels bevinden zich in Appendix C.1. Voor 100 cirkels ligt de RMSE 110 keer hoger dan de originele resultaten met een gemiddelde waarde van 26.60 over de vijf modellen heen. De standaardafwijking gaat er wel weer licht op vooruit ten opzichte van 50 cirkels met een gemiddelde van 0.85, maar ligt nog steeds zes keer hoger dan de originele 0.13.



Figuur 8: De gemiddelde evolutie van MSE tijdens het trainen van het feedforward netwerk voor 100 cirkels over 5 trainingen heen.

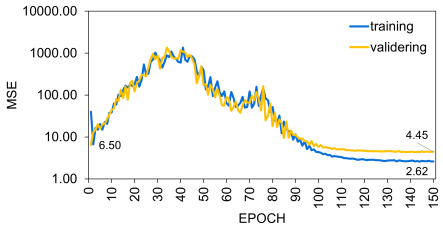


Figuur 9: De MSE en RMSE met standaardafwijking voor vijf verschillende feedforward modellen voor 100 cirkels. Voor alle vijf is dezelfde test dataset gebruikt.

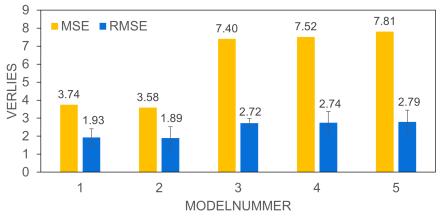
7.4 Nieuwe resultaten BKF

1 Cirkel

Voor 1 cirkel komt de RMSE het dichtst bij de originele resultaten voor 100 cirkels met een gemiddelde waarde van 2.42 over de vijf modellen heen. Dat is echter nog steeds 50 keer slechter dan de originele 0.05 en voor maar één enkele cirkel. Ook de gemiddelde standaardafwijking van 0.79 ligt vijf keer hoger dan de originele 0.12.



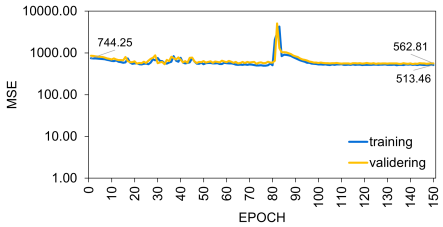
Figuur 10: De gemiddelde evolutie van MSE tijdens het trainen van BKF voor 1 cirkel over 5 trainingen heen.



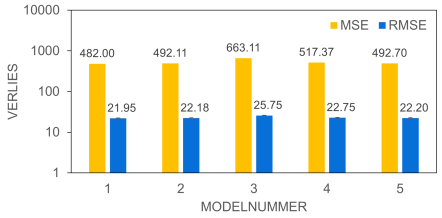
Figuur 11: De MSE en RMSE met standaardafwijking voor vijf verschillende BKF modellen voor 1 cirkel. Voor alle vijf is dezelfde test dataset gebruikt.

25, 50 en 100 Cirkels

Vanaf 25 cirkels slaagt BKF er niet meer in om richting een nauwkeurige voorspelling te trainen. De resultaten en Figuren voor 25 en 50 cirkels bevinden zich in Appendix C.2. Voor 100 cirkels ligt de RMSE 400 keer hoger dan de originele resultaten voor 100 cirkels met een gemiddelde waarde van 22.97 over de vijf modellen heen. De standaardafwijking ligt met een gemiddelde van 2.63 twintig keer hoger dan de originele 0.13.



Figuur 12: De gemiddelde evolutie van MSE tijdens het trainen van BKF voor 100 cirkels over 5 trainingen heen.



Figuur 13: De MSE en RMSE met standaardafwijking voor vijf verschillende BKF modellen voor 100 cirkels. Voor alle vijf is dezelfde test dataset gebruikt.

7.5 Vergelijking resultaten

De herimplementatie slaagt er niet in om de originele resultaten voor het tracking experiment te benaderen. Daarnaast valt ook op te merken dat er veel meer parameters te trainen zijn, ondanks dat er enkel een flatten laag aan het feedforward netwerk is toegevoegd. In de originele paper wordt verwezen naar Layer Normalization als de achterliggende techniek van de resp_norm laag. Die methode levert echter al 32.768 parameters op in de herimplementatie. In Tabel 2 wordt de vergelijking samengevat.

De RMSE waarden van de originele paper zouden echter ook geschaald kunnen zijn. Bijvoorbeeld van $[0, 128]$ tot het interval $[0, 1]$, aangezien er geen waarden buiten $[0, 1]$ in de originele paper aanwezig zijn. Door die schaling op de gegevens voor 100 cirkels toe te passen, wordt voor feedforward een RMSE van 0.2078 bekomen, wat zelfs een kleine verbetering is ten opzichte van 0.2322 van de originele paper. Daarentegen, wordt voor BKF een RMSE van 0.1794 bekomen, wat ongeveer het drievoudige is van de originele paper.

Model	**# Parameters**	**RMS test error $\pm\sigma$**
Feedforward (**o**)	7394	0.2322 ± 0.1316
Feedforward (**n**)	49357	26.6014 ± 0.8487
BKF (**o**)	7493	0.0537 ± 0.1235
BKF (**n**)	49361	22.9663 ± 2.6332

Tabel 2: Tabel met de originele (**o**) en nieuwe (**n**) resultaten met het aantal parameters, de RMSE en standaardafwijking voor de verschillende geteste modellen. Alle modellen werden getraind op 100 sequenties van 100 afbeeldingen [Haarnoja *et al.*, 2016, p. 6]

8 Resultaten nieuwe experimenten

De modellen zijn op dezelfde manier getraind als in Sectie 7.

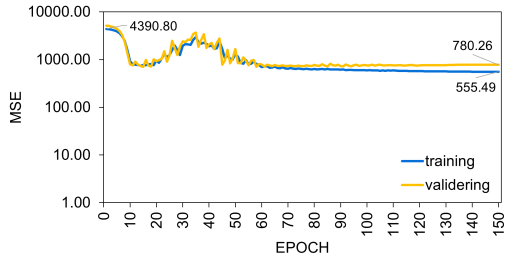
8.1 Eind-tot-eind training

In dit experiment wordt onderzocht of de claim dat BKF eind-tot-eind (e-t-e) te trainen is, standhoudt. De vergelijking wordt gemaakt met de training in vier stappen (4st). Voor e-t-e te trainen wordt een onder- en bovengrens gezet op de elementen van $\hat{\mathbf{L}}_t$. Zonder een grens op te leggen, werden er NaN waardes bekomen tijdens het trainen. De oorzaak daarvoor is dat het feedforward netwerk een nog zodanig slechte voorspelling $\mathbf{z}_t$ geeft waardoor de onzekerheid $\hat{\mathbf{L}}_t$ groot is. Door de

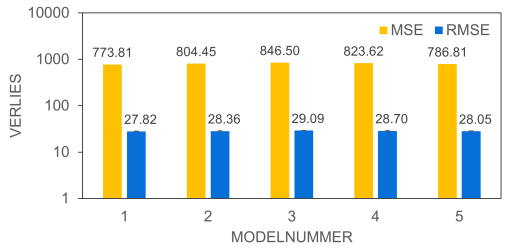
exponentiatie in de omzetting van $\hat{\mathbf{L}}_t$ naar $\mathbf{R}_t$ worden dan te grote getallen bekomen om opgeslagen te worden en dat leidt tot NaN waardes in verdere berekeningen.

De resultaten voor 100 cirkels starten met een veel grotere MSE waarde ten opzichte van 4st, aangezien er geen voorgetraind feedforward netwerk wordt gebruikt. Daarnaast wordt een gemiddelde RMSE van 28.40 bekomen wat ongeveer 25% hoger is dan de RMSE van 22.97 voor 4st.

Indien een herschaling wordt toegepast op de gemiddelde RMSE voor e-t-e van $[0, 128]$ tot $[0, 1]$, wordt 0.22 bekomen in vergelijking met 0.1794 voor 4st (zie Sectie 7.5).



Figuur 14: De gemiddelde evolutie van MSE tijdens het trainen van BKF voor 100 cirkels over 5 trainingen heen.



Figuur 15: De MSE en RMSE met standaardafwijking voor vijf verschillende BKF modellen voor 100 cirkels. Voor alle vijf is dezelfde test dataset gebruikt.

8.2 Vervormende cirkels

In Appendix D is nog een extra experiment terug te vinden, waarbij het feedforward netwerk en BKF op vervormende cirkels getest worden. Daaruit blijkt dat BKF niet slechter presteert in een iets complexere situatie. BKF presteert ook nog steeds beter dan het feedforward netwerk, dat meer moeite heeft met de verhoogde complexiteit door vervorming.

9 Conclusie

Uit de voorgaande resultaten van de experimenten zijn de volgende antwoorden op de onderzoeksvragen af te leiden:

V1: *Is het BKF netwerk duidelijk beschreven zodat het exact geherimplementeerd kan worden?*

A1: Nee, er ontbreken details die niet duidelijk zijn zonder de vereiste voorkennis. Zo zijn de hyperparameters niet beschreven en is de omvorming van $\hat{\mathbf{L}}_t$ tot $\mathbf{R}_t$ niet volledig uitgewerkt.

V2: *Zijn de gebruikte datasets duidelijk beschreven zodat ze opnieuw gegenereerd kunnen worden?*

A2: Nee, de vorm van de gebruikte datasets moet visueel uit de meegegeven figuren afgeleid worden. Ook is het niet duidelijk welke datasets de experimenten gebruiken om op te trainen, valideren en testen.

V3: *Komen de resultaten bij het heruitvoeren van de experimenten overeen met de originele resultaten?*

A3: Nee, voor het meest eenvoudige voorbeeld van één cirkel is de RMSE al 50 keer hoger. Vanaf 25 cirkels kon geen nauwkeurige voorspelling meer bekomen worden.

V4: *Zorgt het toevoegen van een Kalman filter aan een feedforward convolutioneel neuraal netwerk inderdaad voor een nauwkeurigere toestandschatting?*

A4: Niet altijd, voor 1 en 25 cirkels liggen de RMSE van feedforward en BKF binnen een foutmarge van elkaar. Voor 50 en 100 cirkels presteert BKF wel respectievelijk 20% en 15% beter dan feedforward.

V5: *Kan BKF inderdaad eind-tot-eind getraind worden?*

A5: Ja, mits een extra beperking op de onzekerheid van de voorspelling van het feedforward gedeelte. De RSME waarden zijn echter gemiddeld 25% hoger ten opzichte van BKF in 4 stappen.

V6: *Geeft BKF een even nauwkeurige nauwkeurige toestandschatting op een complexere dataset?*

A6: Ja, voor een licht complexere dataset waarbij de cirkels doorheen de tijd van vorm veranderen (naar ellipsen) worden even goede resultaten geboekt als bij niet vervormende cirkels.

De originele paper is niet zo goed reproduceerbaar door het ontbreken van verschillende belangrijke details. Zo mist er informatie rond de hyperparameters, de test dataset, hoe de fouten berekend worden en de implementatie van bepaalde lagen zoals resp_norm. Daardoor is het niet eenvoudig om een herimplementatie te produceren die de resultaten van de originele experimenten benadert.

9.1 Verder werk

Omwille van de tegenvallende resultaten is de hoofdauteur van de originele paper, Tuomas Haarnoja, gecontacteerd. Hij was echter niet beschikbaar voor feedback. In de toekomst kunnen de originele auteurs hun implementatie nog vrijgeven en de ontbrekende informatie aanvullen, opdat fouten in deze herimplementatie opgelost kunnen worden. Eenmaal dat de herimplementatie beter werkt voor meer dan 1 cirkel, kunnen nog complexere datasets getest worden, zoals niet-lineaire beweging of cirkels die van kleur veranderen.

10 Dankwoord

We bedanken dr. Pedro Zuidberg Dos Martires, de begeleider van deze bachelorproef, voor het vele advies. Ook bedanken we graag dr. Robin Manhaeve om ons wegwijs te maken in het gebruik van de departementale ssh server. Tot slot, dank aan prof. dr. Luc De Raedt voor het aanbieden van dit onderwerp.

Referenties

[Albawi *et al.*, 2017] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[Ba *et al.*, 2016] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[Becker, 2022a] Alex Becker. KalmanFilter.NET, 2022. https://www.kalmanfilter.net.

[Becker, 2022b] Alex Becker. KalmanFilter.NET, 2022. https://www.kalmanfilter.net/covUpdate.html.

[Bucy and Joseph, 2005] Richard S Bucy and Peter D Joseph. *Filtering for stochastic processes with applications to guidance*, volume 326. American Mathematical Soc., 2005.

[Chen and Lin, 2019] Claire Chen and Richard Lin. Learning-based Visual Odometry on Ground Robot with Monocular Camera. Technical report, Stanford University, 2019.

[Commons, 2018] Wikimedia Commons. Pictorial example of max-pooling, 2018. File: MaxpoolSample2.png.

[Dumoulin and Visin, 2016] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2016.

[Geiger *et al.*, 2013] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[Goodfellow *et al.*, 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[Gundersen *et al.*, 2022] Odd Erik Gundersen, Kevin Coakley, and Christine Kirkpatrick. Sources of irreproducibility in machine learning: A review. Technical report, San Diego Supercomputer Center, 2022.

[Haarnoja *et al.*, 2016] Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop KF: learning discriminative deterministic state estimators. *CoRR*, abs/1605.07148, 2016.

[Harris *et al.*, 2020] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[Kalman, 1960] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960.

[Kapoor and Narayanan, 2021] Sayash Kapoor and Arvind Narayanan. (Ir)Reproducible Machine Learning: A Case Study. Technical report, Princeton University, 2021.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

[Kost *et al.*, 2019] Alex Kost, Wael Altabey, Mohammad Noori, and Taher Awad. Applying neural networks for tire pressure monitoring systems. *Structural Durability & Health Monitoring*, 13:247–266, 01 2019.

[Maybeck, 1979] Peter S. Maybeck. *Stochastic models, estimation and control. Volume 1 [electronic resource] / Peter S. Maybeck.* Mathematics in science and engineering ; v. 141. Academic Press, New York, 1979.

[Nair and Hinton, 2010] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[O'Shea and Nash, 2015] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[Paszke *et al.*, 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[PyTorch, 2019] PyTorch. ReLU, 2019. https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html.

[Smith and Topin, 2017] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120, 2017.

[SuperDataScience, 2018] SuperDataScience. flatten, 2018. https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening.

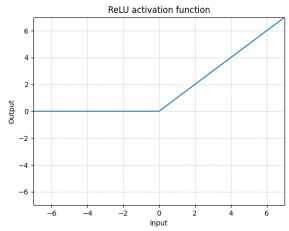
[Thrun *et al.*, 2005] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*, volume 1. The MIT Press, 01 2005.

[Welch *et al.*, 1995] Greg Welch, Gary Bishop, et al. An introduction to the Kalman filter. *Chapel Hill, NC, USA*, 1995.

[Wu and He, 2018] Yuxin Wu and Kaiming He. Group normalization, 2018.

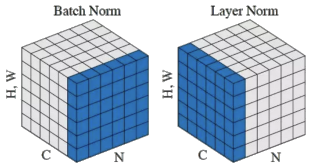
A Extra feedforward achtergrond

Een *Rectified Linear Activation Function* (ReLU) laag heeft als doel de training van het netwerk te versnellen. Daarvoor past ReLU de functie $f(x) = max(0, x)$ toe op elke invoer x. Bij de berekening van de gradiënt blijven zo enkel de resultaten 0 en 1 over voor respectievelijk negatieve en positieve invoer. [Nair and Hinton, 2010]



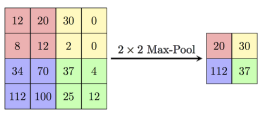
Figuur 16: De werking van een ReLU laag [PyTorch, 2019]

De *Layer Normalization* laag voert een normalisatie uit op een groep invoer, een batch. Het is een aanpassing van een andere techniek, genaamd *Batch Normalization* [Ioffe and Szegedy, 2015], die in bepaalde gevallen voor betere resultaten kan zorgen. Daarbij wordt een mini batch genomen van N training stappen waarvoor de variantie $\mathrm{Var}[\mathrm{X}]$ en het gemiddelde $\mathrm{E}[\mathrm{X}]$ worden berekend. Die worden dan gebruikt om tijdens het trainen in elke stap de invoer te normaliseren. De verbetering bestaat uit het feit dat $\mathrm{Var}[\mathrm{X}]$ en $\mathrm{E}[\mathrm{X}]$ voor elke training stap apart worden berekend uit de volledige invoer van de laag. Zo is er geen afhankelijkheid tussen verschillende training stappen wat tot een snellere trainingstijd en een betere generalisering leidt. [Ba *et al.*, 2016]



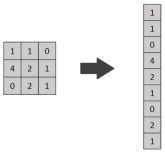
Figuur 17: De werking van een Layer Normalization laag. H, W, C en N zijn respectievelijk de hoogte, breedte, aantal invoerkanalen en aantal afbeeldingen in een batch [Wu and He, 2018]

Convolutionele lagen leren de exacte positie van de eigenschappen in de invoer. Een rotatie of translatie van de invoer levert dus een andere verdeling van eigenschappen op. Om ervoor te zorgen dat het netwerk niet de exacte positie van buiten leert, maar de globale vorm van de eigenschap, kan de resolutie (de dimensies) van de invoer verlaagd worden door middel van een *pooling laag*. Daarbij wordt er een, meestal vierkante, filter over de invoer geschoven waarbij een wiskundige formule op de elementen binnen de filter wordt uitgevoerd. Zo kan er bij max/min pooling enkel het hoogste/laagste element bijgehouden worden.



Figuur 18: De werking van een max pooling laag [Commons, 2018]

Het laatste deel van het netwerk bestaat uit fully-connected lagen waarbij elk knooppunt een eigenschap voorstelt. Aangezien die lagen dus als invoer een lijst van eigenschappen verwachten in de vorm van een één-dimensionale vector, moeten de matrices waarmee tot nu toe gewerkt zijn omgevormd worden tot een één-dimensionale vector. Daar komt een *flatten laag* bij kijken. Standaard worden alle rijen achter elkaar geplaatst in één lange vector.



Figuur 19: De werking van een flatten laag [SuperDataScience, 2018]

B Assumpties en aanpassingen

B.1 Trainen BKF

In appendix B van de originele paper worden de volgende drie stappen uitgelegd om BKF te trainen:

1. Het feedforward netwerk trainen om de positie van de rode cirkel te voorspellen.
2. Het feedforward netwerk nauwkeuriger maken om ook de covariantiematrix van de positie $\mathbf{R}_t$ te voorspellen "with a maximum-likelihood objective" [Haarnoja *et al.*, 2016, p. 10].
3. Het BKF netwerk trainen, geïnitialiseerd met de getrainde parameters uit stap 2.

Aangezien de beschrijving van stap 2 niet gedetailleerd genoeg is om herimplementatie mogelijk te maken zonder de benodigde voorkennis, zijn ter vervanging twee andere stappen gebruikt om de covariantiematrix voorspelling van het feedforward netwerk te trainen. Het resulterende trainingsplan gaat als volgt en wordt in Sectie 5 verder uitgewerkt:

1. Het feedforward netwerk trainen om de positie van de rode cirkel te voorspellen.
2. Voor de hele training dataset een vaste covariantiematrix $\mathbf{R}$ berekenen.
3. Het feedforward netwerk opnieuw vanaf nul trainen om de positie van de rode cirkel te voorspellen met de covariantie $\mathbf{R}$.
4. Het BKF netwerk trainen, geïnitialiseerd met de getrainde parameters uit stap 3.

B.2 Lagen feedforward netwerk

Tabel 3 uit de originele paper [Haarnoja *et al.*, 2016, p. 11] bevat technische details over de verschillende lagen van het feedforward netwerk. Er ontbreken echter een aantal details waarvoor de volgende assumpties zijn gemaakt:

- In "9x9 conv, 4, stride 2x2" en "9x9 conv, 8, stride 2x2" wordt niet gedefinieerd waarvoor de 4 en de 8 staan en wordt de grootte van de padding niet vermeld. De 4 en 8 worden geïnterpreteerd als het aantal uitvoerkanalen. De breedte van de rand met nullen rond de afbeeldingsmatrix

is op 4 gezet zodat de uitvoer van de convolutionele laag dezelfde dimensies heeft als de invoer.

- In "9x9 conv, 4, stride 2x2; ReLU; resp norm" wordt "ReLU" voor "resp norm" vernoemd terwijl ze in de omgekeerde volgorde in Figuur 4 (a) staan. Voor de herimplementatie is de volgorde uit de figuur gebruikt.
- In "fc, 16, ReLU" en "fc, 32, ReLU" wordt niet gedefinieerd waarvoor de 16 en 32 staan. Ze worden geïnterpreteerd als de grootte van de uitvoer van de fully-connected lagen en de grootte van de invoer is voor de herimplementatie afgeleid geweest uit de grootte van de uitvoer van de voorgaande laag.
- In "fc, 2 (output, $\mathbf{z}_t$); fc, 3 (output, $\hat{\mathbf{L}}_t$)" wordt niet vermeld hoe de splitsing naar twee fully-connected lagen verloopt en Figuur 4 (a) biedt ook geen uitsluitsel. Daarvoor is de aanname gemaakt dat beide lagen tegelijk de uitvoer van de laatste ReLU verwerken en dat $\mathbf{z}_t$ en $\hat{\mathbf{L}}_t$ als een tupel worden teruggegeven.

Figuur 3 uit de originele paper [Haarnoja *et al.*, 2016, p. 5] toont het ontwerp van BKF. Er is echter een onduidelijkheid waarvoor de volgende assumptie is gemaakt:

- Voor de resp_norm laag is het niet duidelijk welke implementatie er achter zit. In Sectie 5.2 van de originele paper wordt echter verwezen naar een onafhankelijke paper die de achterliggende implementatie bevat. Die paper bevat de Layer Normalization methode, dus die is gebruikt voor de herimplementatie.

B.3 Kalman filter

Volgende veronderstelling en aanpassing werd gemaakt in het Kalman filter gedeelte van $\text{BKF}_\mathbf{n}$ in Figuur 4 (b):

- In Figuur 4 (a) zijn de twee grijze blokken die de overgang van $\hat{\mathbf{L}}_t$ naar $\mathbf{R}_t$ voorstellen niet gedetailleerd genoeg om exacte herimplementatie mogelijk te maken. $\hat{\mathbf{L}}_t$ is een vector met drie elementen, dus "reshape diag exp" is herwerkt tot het opstellen van een benedendriehoeksmatrix $\mathbf{L}_t$ met op de diagonaal de exponentiële van het eerste en derde element en in de hoek het tweede element:

$$\mathbf{L}_t = f_{exp}(\hat{\mathbf{L}}_t) = f_{exp}(\begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix}) = \begin{bmatrix} e^{l_1} & 0 \\ l_2 & e^{l_3} \end{bmatrix}. \quad (1)$$

- De matrix $\mathbf{B}_\mathbf{w}$ werd in $\text{BKF}_\mathbf{o}$ gebruikt om ruis van het dynamisch model enkel toe te passen op een specifiek deel van de toestand (bv. enkel de snelheid, maar niet de positie). Omdat er geen specifieke ruis wordt toegepast in de eigen datasets, is $\mathbf{B}_\mathbf{w}$ weggelaten in de dynamische update van de covariantie van $\text{BKF}_\mathbf{n}$.

B.4 Hyperparameters

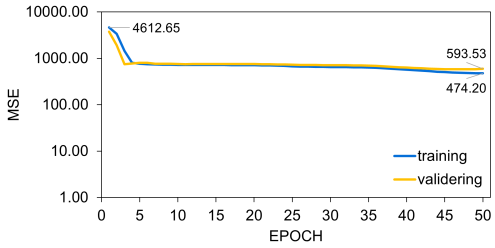
Verder worden de gebruikte hyperparameters niet in de originele paper vermeld. Dat zijn parameters die gebruikt worden om het leerproces te sturen en niet uit de dataset afgeleid kunnen worden. Daarvoor zijn de volgende assumpties gemaakt:

- Voor de hoeveelheid data die in één keer door het model behandeld wordt (batch size) is voor het feedforward netwerk telkens 20 gebruikt en voor BKF 10.
- Voor de grootte van de aanpassing aan de model parameters na één training stap (learning rate) is voor het feedforward netwerk een vaste learning rate van 0.005 gebruikt en voor BKF een variabele learning rate met een maximum van 0.01 door middel van de *1cycle learning rate policy* [Smith and Topin, 2017].

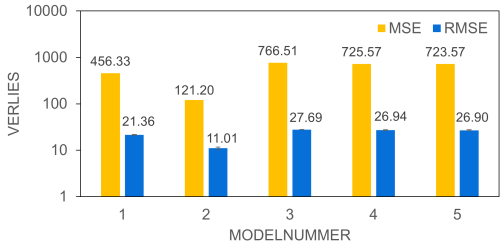
C Extra grafieken experimenten

C.1 Feedforward netwerk

Voor 25 cirkels ligt de RMSE 100 keer hoger dan de originele resultaten voor 100 cirkels met een gemiddelde waarde van 22.7785 over de vijf modellen heen. Ook de gemiddelde standaardafwijking van 1.1927 ligt tien keer hoger dan de originele 0.1316.

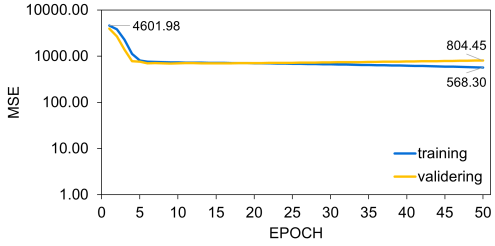


Figuur 20: De gemiddelde evolutie van MSE tijdens het trainen van het feedforward netwerk voor 25 cirkels over 5 trainingen heen.

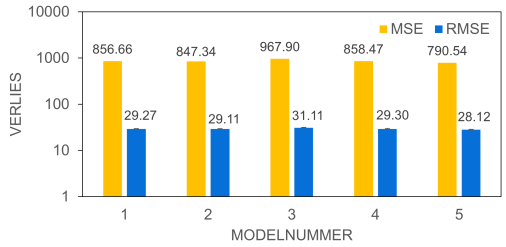


Figuur 21: De MSE en RMSE met standaardafwijking voor vijf verschillende feedforward modellen voor 25 cirkels. Voor alle vijf is dezelfde test dataset gebruikt.

Voor 50 cirkels ligt de RMSE ligt 125 keer hoger dan de originele resultaten voor 100 cirkels met een gemiddelde waarde van 29.3811 over de vijf modellen heen. De standaardafwijking gaat er wel licht op vooruit ten opzichte van 25 cirkels met een gemiddelde van 0.9694, maar ligt nog steeds zeven keer hoger dan de originele 0.1316.



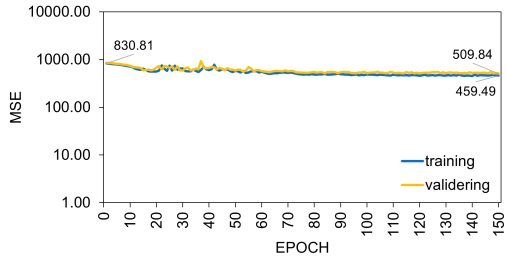
Figuur 22: De gemiddelde evolutie van MSE tijdens het trainen van het feedforward netwerk voor 50 cirkels over 5 trainingen heen.



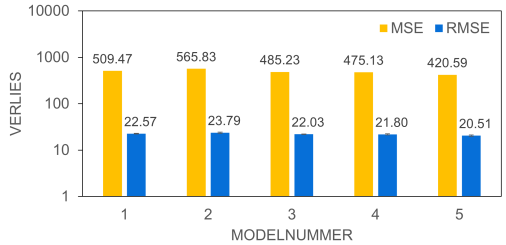
Figuur 23: De MSE en RMSE met standaardafwijking voor vijf verschillende feedforward modellen voor 50 cirkels. Voor alle vijf is dezelfde test dataset gebruikt.

C.2 BKF

Voor 25 cirkels ligt de RMSE 400 keer hoger dan de originele resultaten voor 100 cirkels met een gemiddelde waarde van 22.1385 over de vijf modellen heen. Ook de gemiddelde standaardafwijking van 2.7411 ligt twintig keer hoger dan de originele 0.1316.

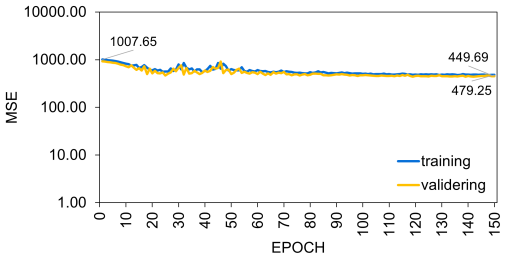


Figuur 24: De gemiddelde evolutie van MSE tijdens het trainen van BKF voor 25 cirkels over 5 trainingen heen.

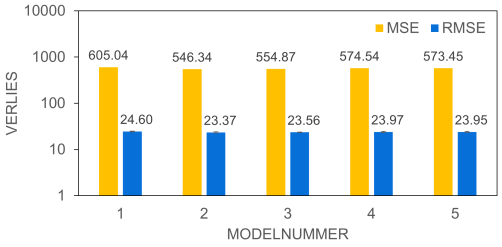


Figuur 25: De MSE en RMSE met standaardafwijking voor vijf verschillende BKF modellen voor 25 cirkels.Voor alle vijf is dezelfde test dataset gebruikt.

Voor 50 cirkels ligt de RMSE 425 keer hoger dan de originele resultaten voor 100 cirkels met een gemiddelde waarde van 23.8887 over de vijf modellen heen. De standaardafwijking ligt met een gemiddelde van 1.4227 elf keer hoger dan de originele 0.1316.



Figuur 26: De gemiddelde evolutie van MSE tijdens het trainen van BKF voor 50 cirkels over 5 trainingen heen.



Figuur 27: De MSE en RMSE met standaardafwijking voor vijf verschillende BKF modellen voor 50 cirkels. Voor alle vijf is dezelfde test dataset gebruikt.

D Vervormende cirkels

D.1 Vervormende dataset

Voor de vervormende cirkels worden ellipsen gegeneerd waarvan de verticale en horizontale straal wijzigt doorheen de tijd. Voor verdere details over de grenzen van deze stralen wordt verwezen naar de broncode.

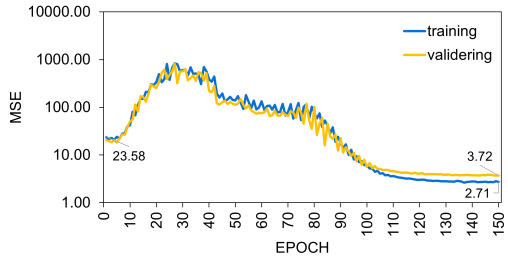
Figuur 28: Afbeeldingen 40, 45 en 50 uit een sequentie van 100 van de synthetische tracking dataset met 1 rode cirkel en 24 afleidingscirkels

D.2 Resultaten experimenten

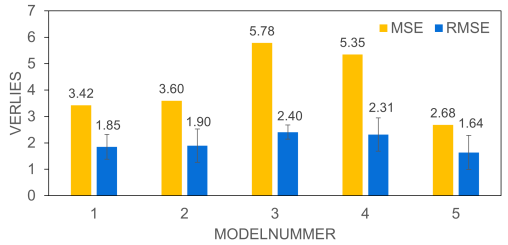
De modellen zijn op dezelfde manier getraind als in Sectie 7.

1 Cirkel

Voor 1 cirkel heeft BKF als RMSE een gemiddelde waarde van 2.02 over vijf modellen heen. Dat ligt lager dan de 2.42 bij de niet vervormende experimenten, maar is wel nog dezelfde grootteorde.



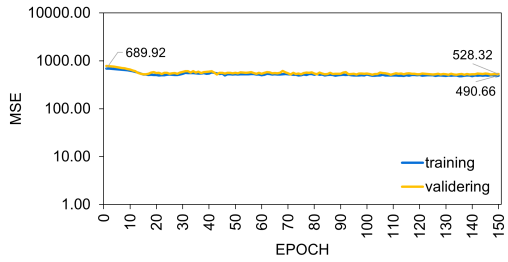
Figuur 29: De gemiddelde evolutie van MSE tijdens het trainen van BKF voor 1 cirkel over 5 trainingen heen.



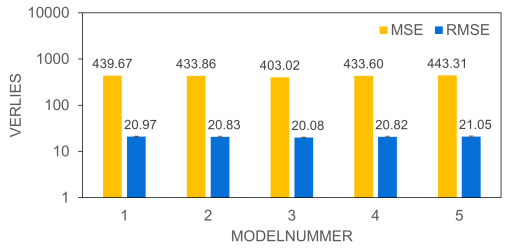
Figuur 30: De MSE en RMSE met standaardafwijking voor vijf verschillende BKF modellen voor 1 cirkels. Voor alle vijf is dezelfde test dataset gebruikt.

100 Cirkels

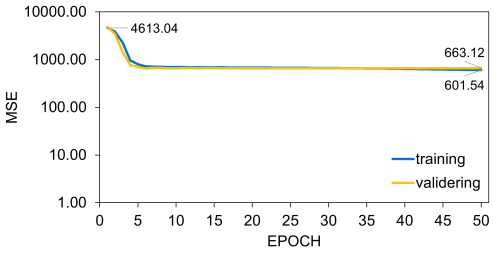
Voor 100 cirkels heeft BKF als RMSE een gemiddelde waarde van 20.75 over vijf modellen heen. Dat ligt lager dan de 22.97 bij de niet vervormende experimenten, maar is wel nog dezelfde grootteorde. Het feedforward netwerk heeft als RMSE een gemiddelde waarde van 28.34 over vijf modellen heen. Dat ligt hoger dan de 26.60 bij de niet vervormende experimenten, maar is wel nog dezelfde grootteorde.



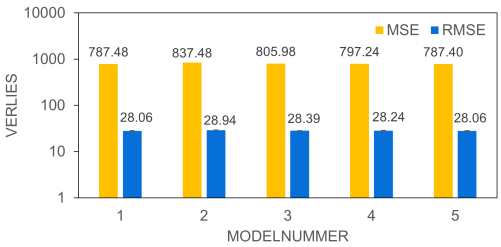
Figuur 31: De gemiddelde evolutie van MSE tijdens het trainen van BKF voor 100 cirkels over 5 trainingen heen.



Figuur 32: De MSE en RMSE met standaardafwijking voor vijf verschillende BKF modellen voor 100 cirkels. Voor alle vijf is dezelfde test dataset gebruikt.



Figuur 33: De gemiddelde evolutie van MSE tijdens het trainen van het feedforward netwerk voor 100 cirkels over 5 trainingen heen.



Figuur 34: De MSE en RMSE met standaardafwijking voor vijf verschillende feedforward modellen voor 100 cirkels. Voor alle vijf is dezelfde test dataset gebruikt.

D.3 Vergelijking resultaten

Voor BKF worden over het algemeen resultaten bekomen die in dezelfde grootteorde licht lager liggen dan bij die van niet vervormende cirkels. Dat wijst erop dat BKF niet slechter presteert in iets complexere situaties. BKF presteert ook nog steeds beter dan het feedforward netwerk. Voor het feedforward netwerk worden over het algemeen resultaten bekomen die in dezelfde grootteorde hoger liggen dan bij die van niet vervormende cirkels. Het feedforward netwerk heeft dus meer moeite met de verhoogde complexiteit door vervorming.