Antwoorden Floris Verheijen bij ML22- tentamen

Antwoorden vraag 1

1a

De gekozen architectuur is minder geschikt is voor dit type data. De data betreffen namelijk tijdseries. Het gebruikte model is een lineair model en lineaire modellen zijn niet goed in staat om temporele patronen in data te analyseren. Recurrent Neural Networks (RNNs) zijn beter geschikt zijn om modellen te creëren met een goede voorspellende waarde, omdat dit type neurale netwerken zich onderscheidt van andere neurale netwerken door het wel informatie uit eerdere input kan gebruiken om huidige input en output te beïnvloeden. Ondanks dat het huidige model minder geschikt is voor de huidige data, is het voordeel van de eenvoud van dit type modellen dat ze minder snel zullen overfitten, dat ze snel zijn en dat je geen zorgen hoeft te maken om in een lokaal minimum vast te blijven zitten (dan denkt de computer dat het meest optimale algoritme heeft gevonden, terwijl er nog betere mogelijk zijn). Daarnaast biedt een dergelijk model het voordeel dat het als een baseline-model kan dienen. Daarmee kan de prestatie van andere, betere modellen in perspectief geplaatst worden en weet je dus beter hoeveel hoe goed een ander model presteert ten opzichte van een eenvoudig model.

Van h2=10 weer naar output=20 is vreemd: informatie wordt samengevat (je gooit informatie weg) en vervolgens moet uit de samengevatte informatie weer meer details gemaakt worden. Dus dit model zou beter werken als het van breed naar smal zou zijn opgebouwd. Als de output=20 is, dan moet h2 sowieso groter het vijf- of tienvoudige zijn van de output (100 a 200) en h1 daar weer een veelvoud van zijn. Een dropout-laag is bedoeld om overfitten van het model tegen te gaan. Wanneer een dergelijke laag niet aanwezig is, dan heeft de eerste batch met trainingsdata dermate veel invloed op het model dat het leren van features van die in latere training batches zitten tegengaat. Dat wil je voorkomen. Als te veel neuronen tijdens het trainen uit worden gezet (een te hoge dropout) dan leert het model ook niet goed meer. Ook dat is niet wenselijk.

1b

Als je in de forward methode van het Linear model kijkt (in tentamen/model.py) dan kun je zien dat het eerste dat hij doet x.mean(dim=1) is. x.mean(dim=1) heeft als functie de grootte van de tensor langs een dimensie te verminderen door de gemiddelde waarde te berekenen van die dimensie. De tensor die overblijft heeft een dimensie minder. Deze oplossing is nodig omdat tijdserie data bestaan uit drie dimensies en de lineaire lagen in het gebruikte model werken met slechts twee dimensies. Dit had ook opgelost kunnen worden door een pooling layer of een flatten-layer toe te voegen aan het model. Beiden kunnen gebruikt worden om de dimensionaliteit terug te brengen.

Los van hoe je de overgang van 3 naar 2-dimensionale data probeert te ondervangen hebben deze oplossingen allemaal het nadeel dat je informatie die beschikbaar is over een van de dimensies niet ten volle gebruikt, waardoor de voorspellende waarde van je model beperkt zal zijn.

Pooling layers werken volgens verschillende methoden. Twee gangbare methoden zijn de Max pool en de Average pool methoden. Average pool werkt met de gemiddelde waarden. Patronen in data zijn daardoor minder uitgesproken waardoor het langer duurt voordat modellen getraind zijn. Max pool werkt met de hoogste waarden in data. Het nadeel daarvan is dat zwakke patronen niet of minder snel opgepikt worden, wat ook beperkend is voor de voorspellende waarde van je model.

1c

Gezien het bovenstaande kies ik voor een model dat GRU-lagen bevat én Attention-lagen omdat het beste van twee werelden combineert. RNN, GRU, LSTM & Attention & NLP-architecturen zijn beter geschikt om tijdserie data te verwerken: al deze architecturen zijn in staat om temporele aspecten van data mee te nemen. Bij data met meer tijdstappen wordt het belangrijker dat de elementen een beter geheugen hebben. Dan heeft een

LSTM-unit vanwege de extra gate een voordeel boven een GRU-model. Dit vraagt wel meer rekenkracht waardoor LSTM-modellen minder snel zijn dan GRU-modellen. In dit geval bevatten de data slechts 13 tijdstappen, daarom zal een GRU-architectuur waarschijnlijk prima voldoen. Verder zijn RNN's (GRU & LSTM) gebaseerd op de gedachte dat metingen dicht bij elkaar ook aan gerelateerd zijn. Attention-modellen laten laat deze gedachte los. Bij attention is volgorde niet meer relevant en daarom zijn attention-layers beter in staat om de relatie met context ver(-der) weg te leggen. Het kenmerkende aan NLP-modellen is deze in staat zijn complexe data te verwerken. De inschatting is dat een NLP-model te zwaar is voor de huidige data.

Te tunen variabelen:

Input size (13): Dat is het aantal features van de input. In dit geval is dat 13. Attention handelt alle data in een keer af en daarom moet input altijd van gelijke lengte zijn. Bij de huidige data is dat het geval en daarom is er geen aparte embeddings laag nodig in een model om GRU en Attention op elkaar aan te laten sluiten.

Num layers (2): Om te starten zou ik met 2 lagen beginnen. RNNs zijn geheugen-intensief en hoe meer lagen, hoe meer geheugen dit vergt. Mocht de prestatie met 2 lagen tegenvallen, dan hoog ik het aan tal op naar 3 of 4.

Hidden size (64): Om mee te starten: 64. Gezien de opmerking tijdens les 3 dat een hidden size van 32 laag is om een model op de Gestures-dataset te trainen. Dit model bevat 20 te classificeren bewegingen, wat vergelijkbaar is met de huidige dataset. Wanneer het resultaat tegenvalt kan ik het verhogen naar 128 of 256. Al lijkt dat laatste veel voor een tijdreeks van 13 getallen.

Dropout (0.1): Gebruik maken van de dropout draagt bij aan het voorkomen van het overfitten van een model op de data. Ik start met een lage waarde: 0.1. 10% van de units in het model worden dan per keer niet getraind. Mocht het model bij deze waarde overfitten, dan vergroot ik de waarde naar 0.3 vervolgens 0.5.

Batch size (256): Diverse posts bevelen een batch size aan tussen de 64 en 512. Het Lineair model werkt met een batch size van 128. Ik ga uit van 256 als een eerste uitgangspunt. Eerst ga ik dit ophogen (256, 512 & 1024) om te kijken wat dit met de resultaten doet.

Epochs: Het aantal epochs benodigd om te trainen hangt samen met de batch size. Is de batch size klein, dan heeft het model langer nodig om getraind te raken. 50 Epochs om mee te starten geeft voldoende ruimte om een indicatie te krijgen van de prestatie van het model zonder dat het gelijk heel veel tijd vraagt om uit te voeren.

Optimizer: Adam, omdat het in de Gestures-dataset gebruikt wordt en dit een vergelijkbare dataset is. En deze methode werkt bij classificatieproblemen.

learning rate: 1e-3 is prima om mee te starten, omdat bij eerdere opdrachten met deze learning rate tot goede resultaten kwamen.

loss functie: Bij classificatie-vraagstukken werkt Cross Entropy Loss goed. Het geeft namelijk de kans dat een voorspelde output past bij elk van de vooraf ingestelde categorieën waarin output te classificeren is. Vervolgens wordt de categorie met de hoogste waarschijnlijkheid gekozen als de voorspelde uitvoer.

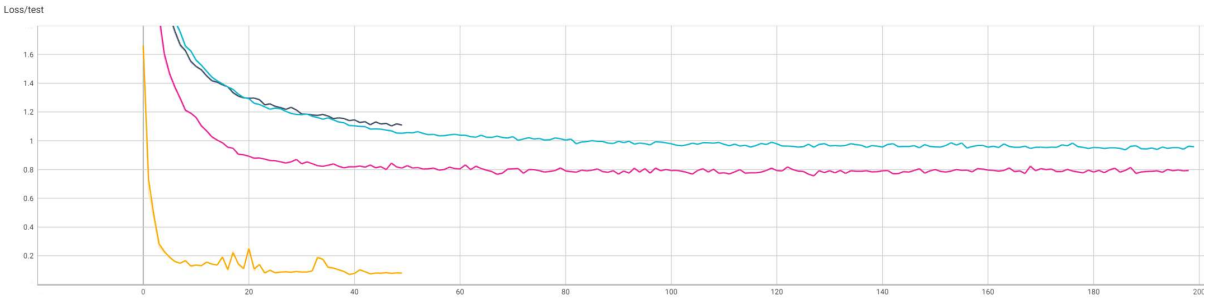
1d

Eerst heb ik het lineaire model getest en vergeleken met AttentionGRU-model (tabel 1 & figuur 1). Vervolgens heb ik enkele parameters van het AttentionGRU-model handmatig getuned. Als eerste heb ik gevarieerd met de dropout-waarden (tabel 2 & figuur 2), vervolgens met de batch_size (tabel 3 & figuur 3). Aansluitend met de hidden_size (tabel 4 & figuur 4). Tot slot heb ik gevarieerd met het aantal num_layers (tabel 5 & figuur 5).

Tabel 1: verschil in resultaat tussen lineaire modellen en een RNN-model

Model		**Kleur lijn**	**Epochs**	**H1**	**H2**	**Num_ layers**	**Hidden_ size**	**Dropout**	**Batch_size**	**Accuracy**
Model 1	Lineair	Grijs	50	100	10	-	-	-	128	65,6%
Model 2	Lineair	Lichtblauw	200	100	10	-	-	-	128	69,4%
Model 3	Lineair	Magenta	200	10	100	-	-	-	128	74,9%
Model 4	Att_GRU	Licht oranje	50	-	-	2	64	0.1	256	97,1%

Figuur 1: Tabel 1: verschil in test-loss tussen de lineaire modellen en een RNN-model

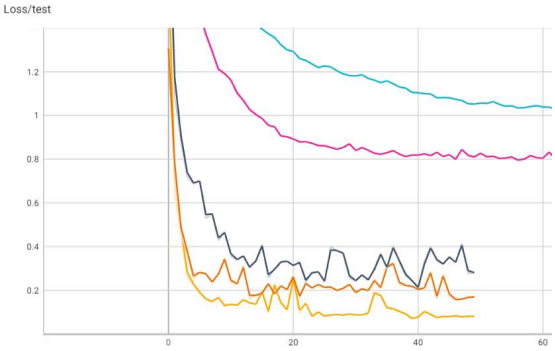


Bovenstaande tabel en grafiek onderbouwen de antwoorden bij 1b. Het onderbouwt de argumentatie dat een h1 > h2 onlogisch is: de modellen 1 (grijs) & 2 (lichtblauw) presteren duidelijk minder dan model 3 (magenta): Model 1 en Model 2 hebben een grotere test_loss & lagere accuracy dan model 3. Ten tweede blijkt eruit dat de keuze voor een lineair model om tijdreeksdata mee te voorspellen minder goede resultaten oplevert. Model 4 is een Recurrent Neural Network (RNN) met Attention- & GRU-lagen met verder willekeurig gekozen hyperparameters. Wanneer de resultaten van de modellen 1,2 en 3 (allen lineair) vergeleken worden met de resultaten van model 4 dan valt op dat duidelijk beter presteert. Daarmee dient Model 1 t/m 3 verder als baseline om de overige resultaten van de overige modellen tegen af te zetten.

Tabel 2: verschil in resultaat met verschillende dropout-waarden

Model	**Kleur lijn**	**Num_ layers**	**Hidden_ size**	**Dropout**	**Batch_size**	**Accuracy**
Model 4	Licht oranje	2	64	0.1	256	97,8%
Model 7	Oranje	2	64	0.5	256	96,7%
Model 8	Grijs	2	64	0.9	256	96,4%

Figuur 2: verschil in resultaat met verschillende dropout-waarden

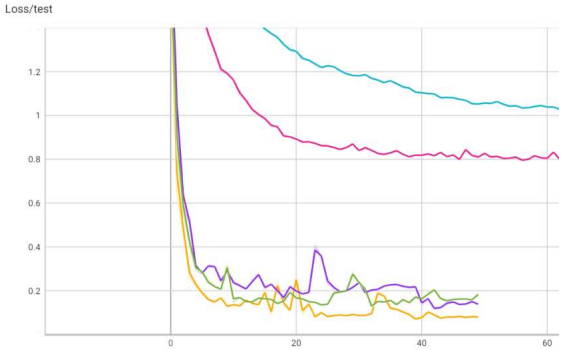


De licht blauwe lijn en de magenta lijn geven de resultaten weer van Model 2 & 3 en zijn bedoeld om de overige resultaten in perspectief te plaatsen. Het verschil in *test loss* van de modellen 4 (licht oranje), 7 (oranje) & 8 (grijs) dat hoe hoger de *dropout* is, hoe grilliger de loss verloopt. Voor het tunen ga ik uit van de range 0.1 t/m 0.9. Bekend is dat naarmate modellen complexer zijn (meer num_layer en een grotere hidden_size), de dropout ook hoger moet zijn om tot dezelfde resultaten te komen. Daarom kies ik ervoor om binnen een brede range te tunen: (0.1 - 0.9).

Tabel 3: verschil in resultaat met verschillende batch_sizes

Model	**Kleur lijn**	**Num_ layers**	**Hidden_ size**	**Dropout**	**Batch_size**	**Accuracy**
Model 4	Licht oranje	2	64	0.5	256	97,8%
Model 5	Paars	2	64	0.5	512	97,3%
Model 6	Groen	2	64	0.5	1024	96,5%

Figuur 3: verschil in resultaat met verschillende batch_sizes

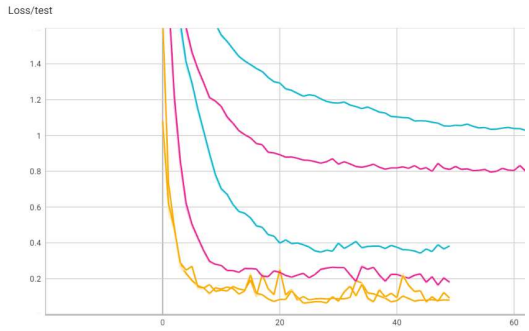


De resultaten van de verschillende *batch_size* zijn lastig te interpreteren. De verwachting is dat bij gelijkblijvende parameters een oplopende batch_size ook een op- of aflopende patroon zichtbaar zou moeten zijn ten aanzien van de accuracy en de test_loss. Er is echter geen duidelijk patroon te ontdekken in de gesteste configuraties. De accuracies en test-loss van de modellen 4, 5 & 6 zitten dicht bij elkaar. Bij het Tunen ga ik daarom uit van een vrij brede range van batch_size waarden: 128-1024.

Tabel 4: verschil in resultaat met verschillende hidden_sizes

Model	Kleur lijn	Num_ layers	Hidden_ size	Dropout	Batch_size	Accuracy	Benodigde tijd
Model 9	Licht blauw	2	16	0.5	256	90,7%	2 m 29 s
Model 10	Magenta	2	32	0.5	256	95,9%	3 m 01 s
Model 4	Licht oranje	2	64	0.5	256	97,8%	4 m 17 s
Model 11	Licht oranje	2	128	0.5	256	98,3%	7 m 14 s

Figuur 4: verschil in resultaat met verschillende hidden_sizes

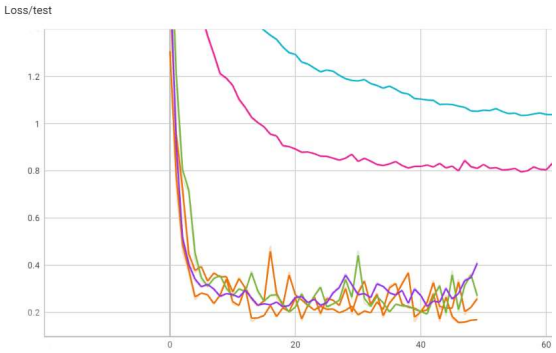


Uit tabel 4 & figuur 4 valt op te maken dat hoe hoger de hidden_size is, hoe hoger de accuracy is, maar ook hoe langer het duurt voordat een model getraind is. Ook valt te concluderen dat de hidden_size niet te laag moet zijn (model 9, onderste licht blauwe lijn): dit gaat ten kostte van de accuracy (90,7%) en de snelheid waarmee het model leert. Het automatisch tunen voer ik uit met waardes tussen de 32 en 256.

Tabel 5: verschil in resultaat met verschillende num_layers

Model	Kleur lijn	Num_ layers	Hidden_ size	Dropout	Batch_size	Accuracy	Benodigde tijd
Model 12	Paars	1	64	0.5	256	94,0%	2 m 47 s
Model 7	Oranje	2	64	0.5	256	96,7%	4 m 18 s
Model 13	Groen	3	64	0.5	256	96,6%	5 m 55 s
Model 14	Oranje	4	64	0.5	256	96,0%	7 m 26 s

Figuur 5: verschil in resultaat met verschillende num_layers



Uit de resultaten van tabel 5 en figuur 5 valt moeilijk op te maken hoeveel lagen tot betere resultaten leidt. Twee zaken vallen wel op: hoe meer lagen, hoe langer het trainen van een model duurt. Daarnaast lijkt de paarse lijn een lichte trend omhoog te hebben wat inhoudt dat de loss niet afneemt naarmate een model langer getraind wordt. Een model met te weinig lagen (1) is dus ook niet goed. Bij het automatisch tunen laat modellen proberen met 1 t/m 4 num_layers

Antwoorden vraag 2

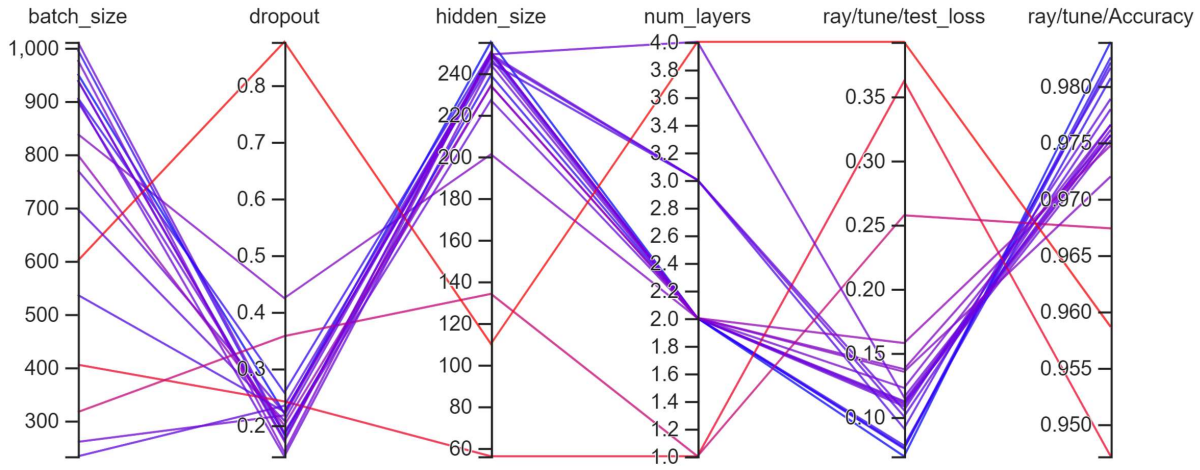
2a

Zie voor de toelichting op mijn keuzes en in welke range ik ga hypertunen het antwoord bij 1d. Daarnaast maak ik gebruik van search algoritme (bobh_search) en een scheduler (bobh hyperband) om te hypertunen. Het zoekalgoritme regelt hoe hyperparameterruimtes worden gesampled en geoptimaliseerd. Het algoritme kiest middels bayesian optimalisatie slim het aantal te testen configuraties van hyperparameters. Het voordeel van deze optimalisatie is dat het niet random naar een optimale hyperparameterconfiguratie zoekt. De scheduler zorgt ervoor dat slecht presterende/lerende modellen vroegtijdig gestopt worden, en rekenkracht niet wordt verspild. Beiden zorgen er samen voor dat het hypertunen efficiënt verloopt en dat de optimale configuratie van parameters gevonden wordt.

2b

In eerste instantie heb ik het attention layer van het GruAttModel ingesteld met *num_heads=4*. Dit gaf vele foutmeldingen. Trials werden stopgezet omdat de attention layer niet werkte. Na veel zoekwerk bleek dat de *embed_dim* deelbaar moet zijn door de *num_heads*. Door de *num_heads* op 1 te zetten heb ik dit opgelost.

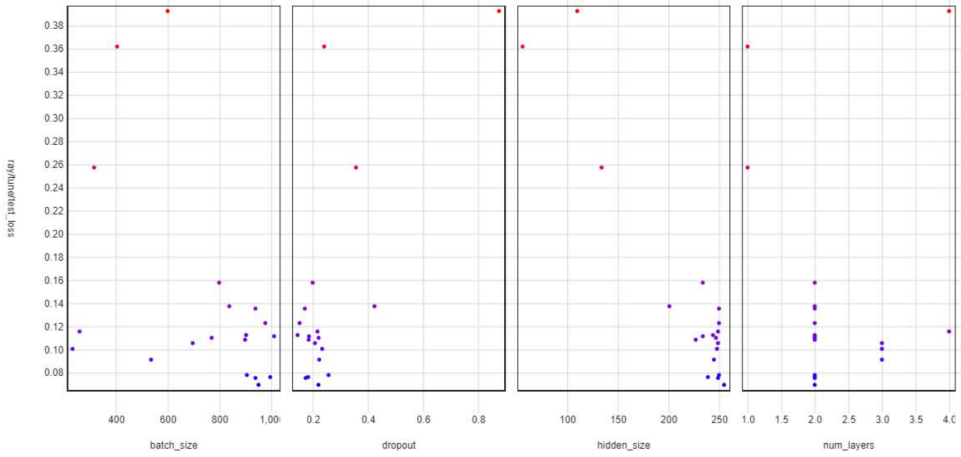
Figuur 6: parallel coordinates view



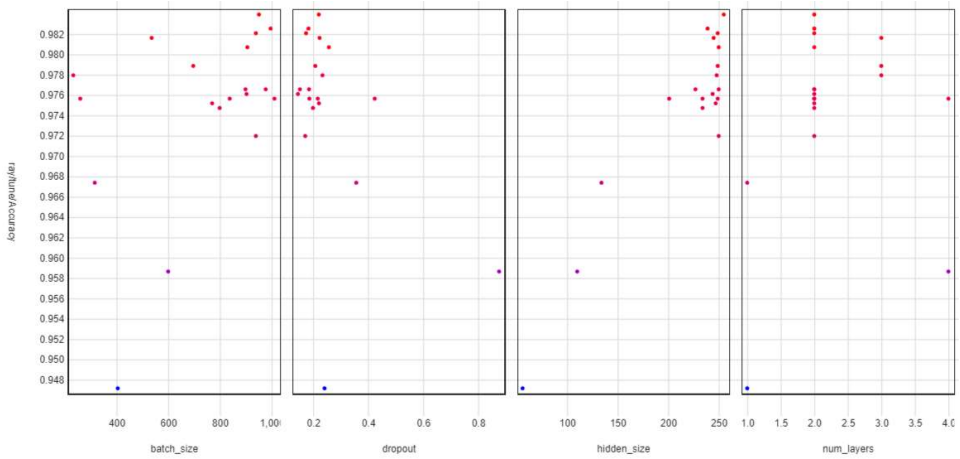
Ray Tune heeft 100 trials gedraaid van maximaal 50 epoch per trial. Om de resultaten overzichtelijk te houden laat de bovenstaande parallel coordinates view alleen de resultaten zien van de trials die het volledige aantal van 50 epochs gerund hebben. Wat opvalt is dat van alle mogelijke configuraties dat de combinatie met batch_sizes >900, een dropout <0.3, een hidden_size van >220, en 2 num_layers tot de laagste test_loss en de hoogste accuracy leiden. De output geeft aan dat de *current best trial* (16be981d) een test_loss van 0.069 en accuray van 98,4% heeft met de parameters hidden_size=255, dropout 0.22, num_layers: 2 en een batch_size van 951. Dit zijn dan ook de instellingen waarmee het finale model wordt getraind bij 2c.

Wat opvalt uit de *parallel coordinates view* (figuur 6) en de scatter plot matrix vies (figuur 7 & 8) is dat de hidden_size van alle volledig gedraaide trials allemaal aan de bovenste grens liggen van de range waarbinnen is getest (hidden_size: 32-256). Ook hier zijn alleen de trials getoond die alle 50 epochs gedraaid hebben. Dit geldt in mindere mate ook voor de dropout en de batch_size. Het kan dus zijn dat het meest optimale model in theorie dus een hidden_size heeft groter dan de bovengrens van 256. Dit test ik verder niet gezien twee redenen: de huidige *best trial* levert al goede resultaten. Het ophogen van de *hidden_size* zou het model alleen maar meer complex zonder dat de resultaten daar nog zoveel beter van kunnen worden. Daarnaast kost het teveel tijd om dit verder uit te zoeken.

Figuur 7: scatter plot matrix view: test_loss



Figuur 8: parallel coordinates view: accuracy



2c

In eerste instantie heb ik het model met 200 epoch gerund. Bleek dat dit natuurlijk heel lang duurde, maar dat de test_loss niet meer beter werd na 30 epochs. Daarom heb ik besloten het model opnieuw te trainen met slechts 35 epochs.

Antwoorden vraag 3

Vraag 3 bestaat uit diverse opdrachten. Deze heb ik uitgevoerd en daar hoeft niet op gereflecteerd te worden.