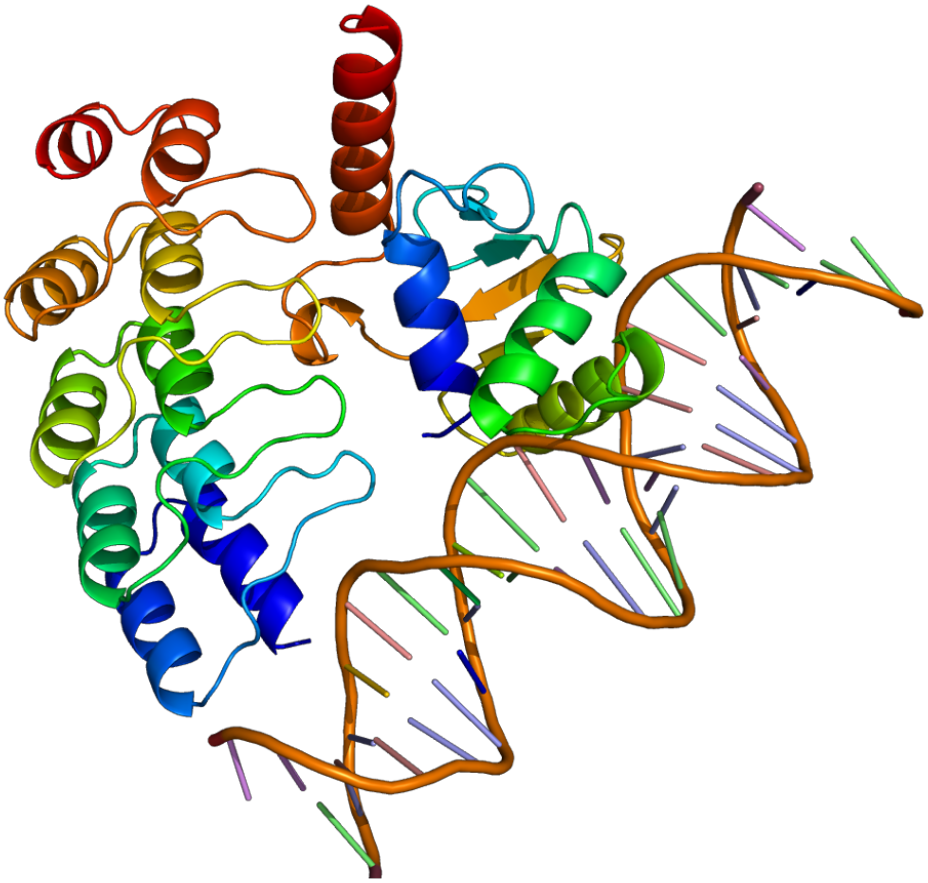
Profielwerkstuk 6 vwo

Bindingsplekken eiwitten op DNA voorspellen met machine learning

Door Bram van Heest en Erik van der Plas



13 DECEMBER, 2018

EERSTE CHRISTELIJK LYCEUM

Inhoudsopgave

Sectie 1 – Inleiding ***3***

1.1 – Introductie **3**

1.2 – Onderzoek **3**

Sectie 2 – Het DNA ***4***

2.1 – Wat is DNA? **4**

2.1.1 – De bouw van DNA 4

2.1.2 – De functie van DNA 5

2.2 – Regeling uiting van genen **5**

2.2.1 – Transcriptiefactoren 6

2.2.2 – Histonmodificaties 6

2.2.3 – Motieven 7

2.3 – ChIP-seq **7**

Sectie 3 – Neurale netwerken ***9***

3.1 – Machine learning algoritme **9**

3.1.1 – Neurale netwerken 9

3.1.2 – Werking neurale netwerken 10

3.1.3 – Gradient descent 11

3.1.4 – Recurrent neural networks 13

Sectie 4 – Methode ***15***

4.1 – Toepassing RNN voor transcriptiefactor hechtingslocaties **15**

4.1.1 – Bidirectionality 15

4.1.2 – Gated Recurrent Unit 16

4.1.4 – One-hot encoding 16

4.1.5 – Binary log loss 16

4.1.6 – Optimalisatie 17

4.1.7 – Dropout 17

4.1.8 – Volledige architectuur 18

4.2 – Training **19**

4.2.1 – Datasets 20

4.2.2 – Gebruikte rekenkracht 21

Sectie 5 – Resultaten ***21***

5.1 – Resultaten training **21**

5.1.1 – Verloop training 21

5.1.2 – Statistische scores 22

5.2 – Visualisaties **24**

Sectie 6 – Conclusie ***27***

6.1 – Verwerking resultaten **27**

6.2 – Vergelijking met bestaande technieken **28**

6.2.1 – Vergelijking met ChIP-seq 28

6.2.2 – Vergelijking met bestaande statistische modellen 28

6.2.3 – Vergelijking met bestaande machine learning modellen 29

6.3 – Vervolgonderzoek **30**

Sectie 7 – Discussie ***31***

7.1 – Terugblik **31**
7.2 – Samenwerking **31**
7.3 – Dankwoord **32**
Bronvermelding ***33***
Appendix 1 – Wiskundige toelichtingen ***36***
1.1 – Standaard neuraal netwerk **36**
1.2 – Gradient descent **36**
1.3 – Recurrent neural network **37**
1.4 – Gated Recurrent Unit **37**
1.4.1 – Logistische sigmoïde en de hyperbolische tangens 38
1.5 – Binary log loss cost function **39**
1.6 – Wiskundige toelichting Adam Gradient Descent **41**

Sectie 1 – Inleiding

1.1 – Introductie

Het genetisch materiaal beïnvloedt hoe organismen ontwikkelen. Ook is het genetisch materiaal, met uitzondering van lokale mutaties, in het gehele organisme gelijk. Toch doen niet alle cellen hetzelfde. Dit komt doordat de genen (bepaalde gebieden op het DNA die zich laten vertalen tot bepaalde erfelijke eigenschappen) niet in iedere cel op dezelfde wijze tot uiting komen. Twee belangrijke systemen die de transcriptie van DNA bepalen zijn histonmodificaties en transcriptiefactoren. Beide systemen zullen in dit onderzoek verder toegelicht worden, maar het gaat in principe in beide gevallen om eiwitten die aan het erfelijk materiaal hechten om de genexpressie in dat gebied op het genetisch materiaal te beïnvloeden. De systemen zijn tekenend voor de uiting van alle genen, en zo ook voor bijvoorbeeld het ontstaan van ziektes. Men kan met behulp van geavanceerde technieken aflezen waar op het genetisch materiaal zulke eiwitten zitten, maar niet nauwkeurig genoeg om de complexe systemen volledig te begrijpen. Als sneller, nauwkeuriger en goedkoper alternatief lichten we in dit onderzoek een probabilistisch model op basis van *machine learning* toe dat de locaties van specifieke eiwitten op basis van enkel de DNA-sequentie kan voorspellen.

1.2 – Onderzoek

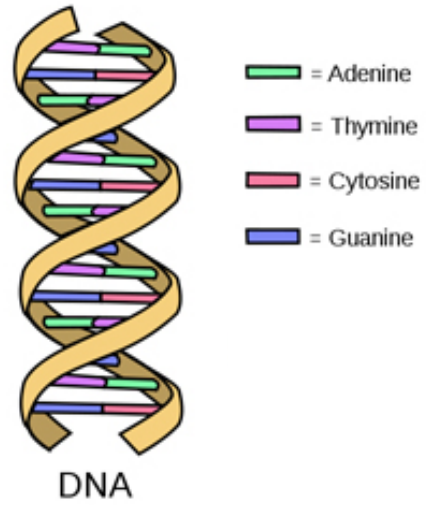
In dit onderzoek willen we kijken of we het eerdergenoemde model kunnen ontwikkelen en implementeren. Ook willen we uiteindelijk het algoritme op verscheidene manieren testen om de kwaliteit van de techniek vast te stellen. We zullen de verschillende mogelijkheden voor ons onderzoek uitlichten en ook over de biologische achtergrond van dit onderzoek uitweiden. Uiteindelijk willen we antwoord geven op de vraag: is met behulp van een computer algoritme te voorspellen waar op het DNA specifieke transcriptiefactoren zullen hechten?
Op basis van het literatuur- en vooronderzoek dat we aan het einde van 5 vwo hebben gedaan, vermoeden we dat we tijdens ons onderzoek voldoende tijd hebben het model goed te kunnen ontwikkelen en dat we na validatie van ons algoritme positieve resultaten zullen krijgen. Ook denken we dat verdere ontwikkelingen in de bioinformatica (het onderzoeksveld waar ons onderzoek deel van uitmaakt), specifiek met machine learning, buiten de mogelijkheden en het tijdsbestek van ons profielwerkstuk, essentieel zullen zijn om genexpressie volledig te begrijpen.
In dit onderzoek zullen wij ons beperken tot de binding van één specifiek eiwit in één specifieke cel, om te kijken of ons model functioneel is en later schaalbaar is voor meer toepassingen. Hiervoor zullen we eerst onderzoeken waar het genetisch materiaal precies uit bestaat en hoe de uiting ervan gereguleerd wordt. Dan zullen we kijken naar hoe *machine learning* precies werkt en hoe we dat kunnen toepassen in dit onderzoek. Ten slotte zullen we antwoord geven op de onderzoeksvraag, door te kijken in hoeverre ons model in staat is te voorspellen waar bepaalde eiwitten op het DNA zullen binden, enkel op basis van het DNA zelf.

Sectie 2 – Het DNA

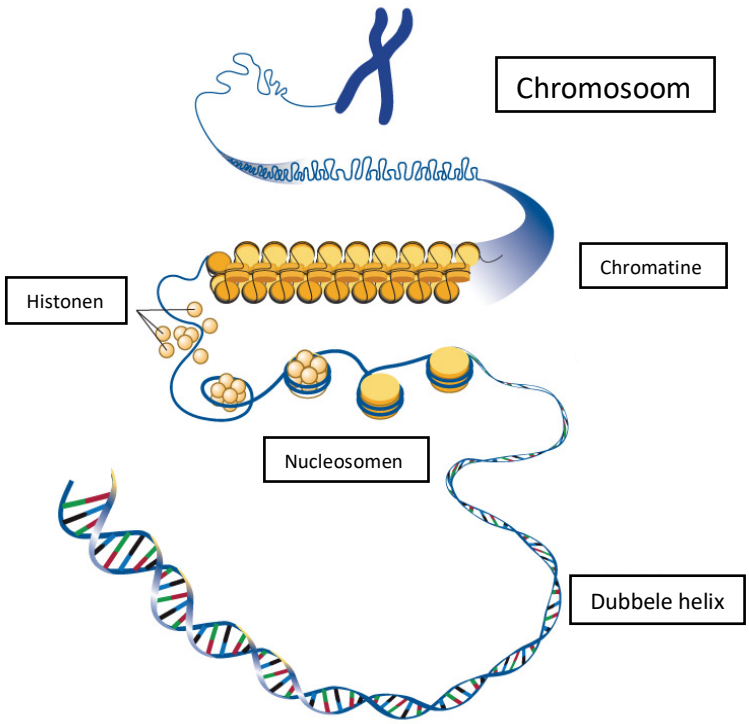
2.1 – Wat is DNA?

2.1.1 – De bouw van DNA

Desoxyribonucleïnezuur (“DNA”) bestaat uit twee lange strengen die zijn opgebouwd uit nucleotiden. Elke nucleotide bevat een stikstofbase. Deze basen van de nucleotiden van de ene streng vormen baseparen met de stikstofbasen van de nucleotiden van de andere streng. Er zijn vier soorten stikstofbasen in het DNA: guanine (G), cytosine (C), thymine (T) en adenine (A). T kan alleen met A een basenpaar vormen en G alleen met C. De twee strengen nucleotiden zitten aan elkaar vast en zijn in een dubbele helix opgevouwen, zie Figuur 1 (VIDA, 2010). In alle cellen zit de hele DNA-sequentie, de volledige volgorde van de basenparen in het DNA. Er wordt geschat dat in elke cel zo’n 2 meter DNA zit opgerold, bestaande uit ongeveer 3 miljard baseparen. Dit past allemaal in de celkern door eiwitten genaamd histonen. De dubbele DNA-helix zit namelijk gewikkeld rondom histonen die per 8 een pakketje vormen om op die manier zo genaamde nucleosomen te vormen. Het DNA met de nucleosomen en verschillende andere eiwitten wordt chromatine genoemd. De functie van de chromatine is om het DNA efficiënt te verpakken zodat het DNA in de celkern past. Ook heeft het controle over de expressie van genen. Genen zijn stukjes DNA die allemaal een andere code bevatten voor bepaalde kenmerken. Er zijn genen voor uiterlijk, zoals oogkleur en haarkleur, maar er zijn ook genen die beïnvloeden hoe de mens van binnen werkt. De chromatine spiraliseert nog meer waarna uiteindelijk chromosomen ontstaan, de compactste vorm waarin DNA in het menselijk lichaam zit, zie Figuur 2. Mensen hebben per cel 46 chromosomen in de celkern en in geslachtscellen 23 (Annunziato, 2008) (abcam, 2014).



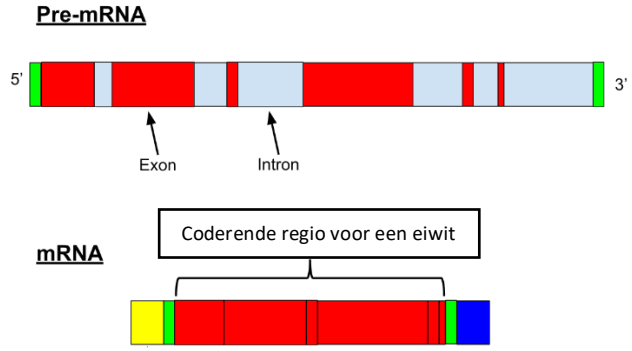
Figuur 1. Basenparen van het DNA



Figuur 2. Verpakking van het DNA in chromosomen

2.1.2 – De functie van DNA

Het DNA heeft meerdere functies, waaronder het coderen voor eiwitten. Eiwitten zijn belangrijke grote moleculen die veel uiteenlopende belangrijke functies voor het lichaam hebben. Dit coderen van eiwitten wordt mogelijk gemaakt door de volgorde van de nucleotiden. In de chromosomen liggen namelijk genen, stukjes DNA die een code bevatten voor eiwitten. De genen zijn opgedeeld in introns en exons. Introns zijn de stukken van het DNA die niet coderen voor een eiwit, dit is bijna 98,5% van het DNA. Exons zijn de sequenties in het DNA die wel coderen voor een eiwit. Eiwitten worden niet in de celkern gevormd. Het DNA wordt eerst opengeritst in de celkern en er wordt een kopie van een streng gemaakt. Dit kopie heet het pre-mRNA. Dit pre-mRNA vormt mRNA dat de celkern kan verlaten. Dit hele proces heet de transcriptie. Het mRNA wordt gemaakt door de introns van de genen weg te knippen uit het pre-mRNA en de exons achter elkaar te plakken, zie Figuur 3. Deze rij exons wordt dan buiten de cel afgelezen en daar wordt ook een eiwit gevormd. Dit proces wordt translatie genoemd (10voorBiologie, 2013). De mate waarin stukken DNA worden overgeschreven naar mRNA om eiwitten te vormen is variabel en wordt onder andere geregeld door transcriptiefactoren die aan het DNA binden (zie Sectie 2.2.1).
DNA is zo ook de drager van erfelijke informatie. Iedere ouder geeft 23 chromosomen aan zijn of haar kind: de helft van hun DNA. Deze twee helften bij elkaar vormen samen het hele DNA van 46 chromosomen en mede door gen-dominantie wordt bepaald welke eigenschappen tot uiting komen en welke niet.



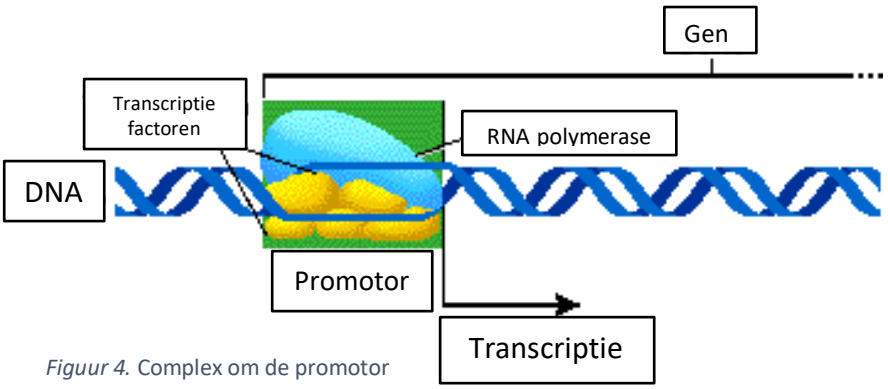
Figuur 3. Vorming mRNA vanuit pre-mRNA

2.2 – Regeling uiting van genen

Het DNA codeert voor eiwitten om zo allerlei processen te regelen. Eiwitten zijn overal in het lichaam en ook op het DNA zelf. Op het DNA hebben eiwitten verschillende functies waaronder ook de regeling van de expressie van genen. Ook zijn er bepaalde groepen moleculen zoals ethyl en acetyl die aan histonen binden. Dit worden histonmodificaties genoemd. Ook histonmodificaties hebben invloed op de uiting van genen door de mogelijkheid die zij hebben om het DNA van structuur te laten veranderen.

2.2.1 – Transcriptiefactoren

Wanneer RNA-polymerase, een enzym, bindt aan een promotor, een stuk DNA-sequentie aan het begin van een gen, start de transcriptie. Dit is dus de activatie van het gen. Deze activatie wordt vooral geregeld door de transcriptiefactoren. Deze eiwitten binden zich namelijk aan de promotor en zorgen er dan voor dat het makkelijk of juist moeilijk is voor RNA-polymerase om te binden aan de promotor. Voor het werken van RNA-polymerase bij transcriptie is er bij eukaryoten een basale transcriptiefactor, door deze transcriptiefactoren is het mogelijk om complexen te vormen van transcriptiefactoren met RNA-polymerase. Er zijn andere transcriptiefactoren die een complex vormen met de basale transcriptiefactor en daarmee de binding van RNA-polymerase aan de promotor beïnvloeden, zie Figuur 4. De transcriptiefactoren die ervoor zorgen dat RNA-polymerase makkelijk aan de promotor bindt worden activators genoemd. De transcriptiefactoren die de binding van RNA-polymerase juist tegen gaan worden repressors genoemd. Er zijn heel veel transcriptiefactoren met allemaal hele specifieke functies en soms werken ze in complexe verbanden met elkaar samen (Khan Academy, 2018).



Figuur 4. Complex om de promotor

2.2.2 – Histonmodificaties

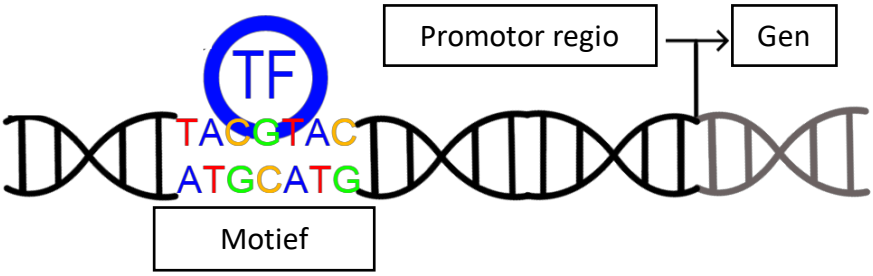
Histonen zijn de eiwitten waaromheen het DNA zit gebonden in nucleosomen. Wanneer histonen modificeren kan de bindingen tussen het DNA en de histonen versterken of verzwakken. Dit kan zorgen voor het aan en uitzetten van de transcriptie, waardoor er dus wel of geen eiwit wordt gevormd. Genen kunnen door histonmodificatie dus aan en uit staan. Er zijn verschillende vormen van histonmodificatie, onder andere methylering en acetylering. Methylering treedt op wanneer er één of meerdere methyl groepen binden aan histonen. Dit binden van methylgroepen aan histonen kan de transcriptie van genen laten toenemen en laten afnemen, afhankelijk van de plaats waar de methylering plaatst vindt en de hoeveelheid methylgroepen die bindt aan het histon. Wanneer de methylering zorgt voor een lossere structuur in het DNA wordt de uiting van een gen bevorderd, aangezien transcriptiefactoren en RNA-polymerase dan beter bij het DNA kunnen komen. Acetylering is een histonmodificatie waarbij er een acetylgroep ($COCH_3$) aan een histon bindt. Dit zorgt ervoor dat de structuur van het DNA losser wordt, waardoor transcriptiefactoren

makkelijker aan het DNA kunnen binden. Wanneer histon acetylering plaatsvindt is dit bevorderlijk voor de uiting van een gen (What is epigentics, 2013).

Het weten van de plaatsen op het genoom waar bepaalde transcriptiefactoren en histonmodificaties zitten is erg belangrijk. Hiermee zijn namelijk heel veel dingen te voorspellen over hoe het genotype, de genetische informatie, zich uiteindelijk zal uiten in het fenotype, het waarneembare van een organisme. Welke genen actief zijn en welke niet en of deze nog actief worden is hele belangrijke informatie, ook bijvoorbeeld bij ziektes zoals kanker. Bij kanker is er namelijk sprake van ongecontroleerde deling van cellen, deze deling wordt normaal gecontroleerd maar in dit geval is deze controle inactief. Dit is het gevolg van deacetylering, het verwijderen van acetyl groepen van histonen. Zonder deze acetylering wordt deze deling van cellen dus niet meer gecontroleerd. Hoe meer informatie over de transcriptiefactoren en histonmodificaties wordt verkregen, des te beter bepaalde ziektes kunnen worden bestreden of voorkomen (What is epigentics, 2013).

2.2.3 – Motieven

Een motief is een reeks basen op het DNA waaraan een bepaald eiwit vaak bindt, zie Figuur 5. Per eiwit is er meestal niet slechts één motief bekend, maar zijn er meerdere motieven die een klein beetje afwijken van elkaar: als er enkele basen afwijken van een bepaald motief dat vaak voorkomt kan het zo zijn dat het eiwit toch nog bindt aan dit stuk DNA. Ook kunnen er meerdere volkomen verschillende motieven zijn voor een bepaald eiwit, maar vaak is er dan wel een bepaald soort motief waar het eiwit het vaakst aan bindt. Deze motieven worden tegenwoordig ontdekt door bijvoorbeeld ChIP-seq onderzoeken (zie Sectie 2.3), hierbij wordt namelijk duidelijk waar een bepaalde transcriptiefactor vaak op het DNA bindt (D'haeseleer, 2006).

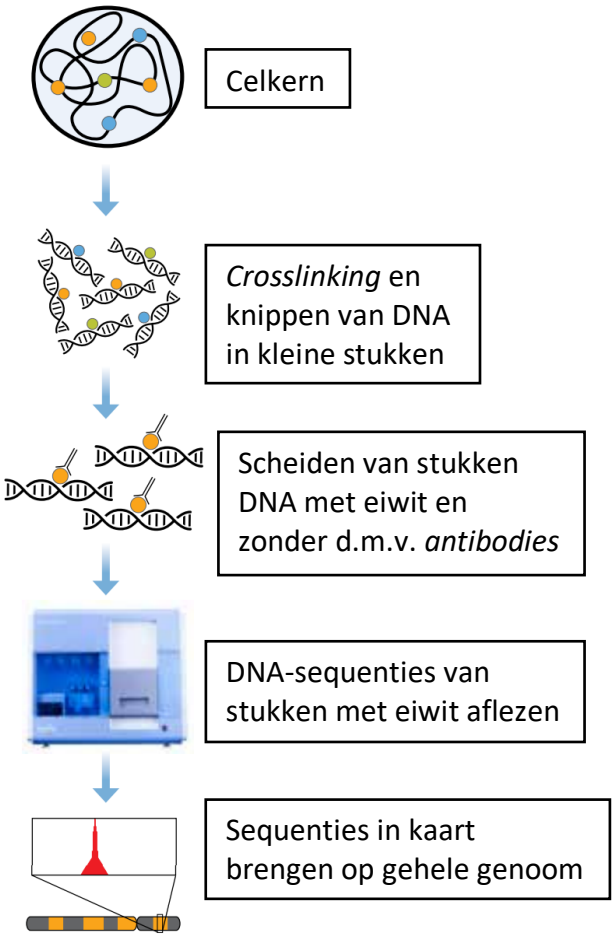


Figuur 5. Transcriptiefactor gebonden aan motief

2.3 – ChIP-seq

Chromatin Immunoprecipitation Sequencing ("ChIP-seq") is een methode om de bindingsplekken van transcriptiefactoren en andere eiwitten aan het DNA te vinden, zie Figuur 6. ChIP-seq onderzoekt honderden cellen tegelijk. Eerst worden de eiwitten die op het DNA zitten, bijvoorbeeld een transcriptiefactor, van deze cellen permanent vastgemaakt aan het DNA, wat *crosslinking* wordt genoemd. Daarna wordt het DNA in kleine stukken van ongeveer gelijke grootte geknipt, deze stukken zijn rond de 100 tot 1000 basenparen ("bp") lang. De stukken DNA waar de eiwitten in werkelijkheid aan binden zijn enkele tientallen bp

breed. Het knippen van het DNA gebeurt zodat de stukken daarna van elkaar gescheiden kunnen worden. Dit is mogelijk doordat sommige stukken de te lokaliseren eiwitten bevatten. Voor deze eiwitten bestaan zogenaamde *antibodies*. Dit zijn andere eiwitten die zich precies om het eiwit kunnen sluiten waar het voor bedoeld is. Deze *antibodies* zijn Y-vormige eiwitten, die ook worden gebruikt door het immuunsysteem om zo bepaalde bacteriën en virussen te neutraliseren. Omdat ChIP-seq ook gebruik maakt van deze eiwitten maar dan in de chromatine is de naam ***Chromatin Immuno****precipitation Sequencing* te verklaren. Doordat de *antibodies* om de eiwitten op het DNA sluiten, kunnen de stukken DNA met de eiwitten gescheiden worden van de stukken zonder eiwitten. Deze stukken DNA met eiwitten eraan zijn erg belangrijk, ze laten namelijk zien waar de eiwitten op het DNA binden. Ze worden in kaart gebracht op het gehele genoom, dit is het gehele DNA. Dit in kaart brengen van waar deze stukken DNA dan op het DNA zaten is te vergelijken met woorden zoeken op een computer met Ctrl + F**.** De stukken DNA kunnen namelijk worden gezocht op het gehele genoom. Omdat er honderden cellen worden gebruikt kunnen er pieken worden opgemaakt, die hoog zijn op de stukken DNA waar het bepaalde eiwit altijd of vaak aanwezig is en laag of afwezig zijn waar bijna nooit of nooit een eiwit bindt (EpiGenie, 2013) (illumina, 2010).



Figuur 6. ChIP-seq processen

Sectie 3 – Neurale netwerken

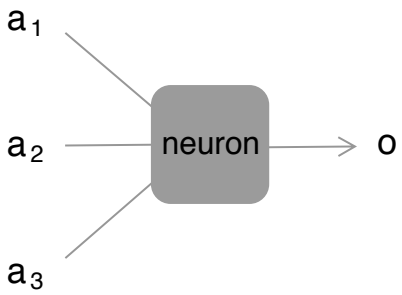
3.1 – Machine learning algoritme

Machine learning is de studie die zich bezighoudt met het ontwikkelen van algoritmes waarmee computers leren om de oplossingen van, voor de computer gedefinieerde, problemen te benaderen. Om het onderzoek naar de hechtingslocaties van transcriptiefactoren uit te voeren is een specifiek algoritme ontworpen dat gebruikt maakt van *recurrent neural networks* ("RNN"). Het algoritme is gebaseerd op een grote verzameling voorgaand werk met betrekking tot *machine learning* algoritmes dat in dit hoofdstuk samengevat is.

3.1.1 – Neurale netwerken

In de afgelopen jaren hebben kunstmatige neurale netwerken ("NN"), waarvan het RNN een specifieke vorm is, bij verschillende patroonherkenningstaken en andere *machine learning* doeleinden bewezen de beste optie te zijn. NN'en zijn mathematische modellen die geïnspireerd zijn op het zenuwstelsel, hun biologische tegenhanger. (Schmidhuber, Deep Learning in Neural Networks: An Overview, 2014)

De modellen bestaan uit een vast aantal zogenaamde neuronen. Deze neuronen representeren individueel een simpele functie, maar vormen in verbinding met elkaar een complex netwerk. De neuronen zijn in lagen opgestapeld. De eerste laag neuronen verwerkt bepaalde bekende reële waarden als 'invoer' of 'prikkel'. Hierbij wordt het neuron in een bepaalde mate 'geactiveerd' zoals gedefinieerd door zijn interne functie. De 'activatie' van een neuron is dus de uitkomst van zijn interne functie gegeven zijn invoer. Deze invoer bestaat vrijwel altijd uit meerdere reële waarden, en de functie neemt dus meerdere variabelen, zie ook Figuur 7. Alle andere lagen neuronen hebben de 'activatie' van voorafgaande neuronen weer als invoer, en worden vervolgens ook zelf volgens een eigen functie geactiveerd. De laatste laag kan als de 'uitkomst' van het gehele NN kan worden beschouwd. Alle tussenliggende lagen hebben geen directe betekenis en worden daarom 'verborgen lagen' genoemd. De functies van alle neuronen worden gekenmerkt door parameters die tijdens het leerproces van het NN worden aangepast om de uitkomst van het NN zo gewenst mogelijk te maken. (Schmidhuber, Deep Learning in Neural Networks: An Overview, 2014) Dit gaat met een proces genaamd *gradient descent* ("GD") dat later verder zal worden toegelicht.

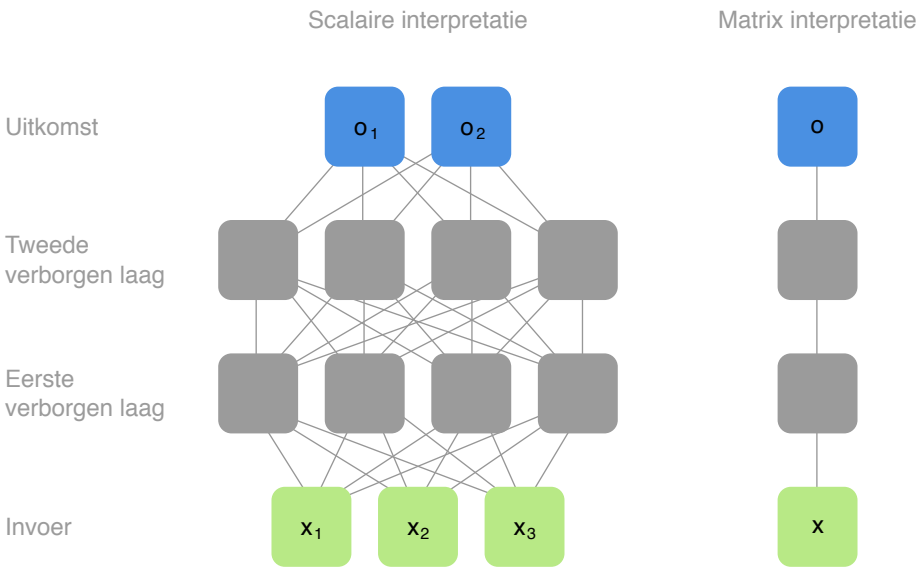


Figuur 7. Voorbeeldneuron dat drie reële waarden verwerkt

Een voorbeeld van een toepassing van een NN, om tevens een beter beeld te krijgen van het hiervoor omschreven leerproces, is de volgende: het NN krijgt als invoer bepaalde meteorologische meetwaarden, zoals getallen met betrekking tot regenval, zonneschijn, zonsopgang en -ondergang, etc. Als gewenste uitvoer geeft het NN een voorspelling van het komende weer. Het NN heeft een vaste vorm (een vast aantal lagen met ieder een vast aantal neuronen), maar het kan wel voor iedere neuron de parameters aanpassen, die allemaal in het gehele netwerk ieder een complexe invloed hebben op de uiteindelijke uitkomst. Deze worden herhaaldelijk zo aangepast dat het NN bepaalde bekende voorspellingen (bijvoorbeeld historische gegevens, waarvoor het komende weer al bekend is) de juiste uitkomst geeft. Mits er genoeg data is waar het NN van kan 'leren' kan het NN een gegeneraliseerd model worden dat ook voor de toekomst nieuwe voorspellingen kan doen.

3.1.2 – Werking neurale netwerken

Bovenstaande uitleg gaat uit van meerdere neuronen per laag, waarbij iedere neuron zowel voor de invoer als voor de uitkomst scalaire waardes (gewone getallen) gebruikt, maar vaak worden matrices gebruikt. Dit zorgt ervoor dat een mathematische omschrijving van een neuraal model een stuk compacter kan worden genoteerd, maar rekenen met matrices is vaak ook efficiënter voor computers omdat er dan verscheidene optimalisaties gedaan kunnen worden. Een hele laag van neuronen gebruikt één matrix aan parameters, en ook de invoer, activaties en uitvoer van het NN worden weergegeven in matrices (waarbij meerdere invoeren in rijen staan in de matrix, idem voor de activaties en uitvoer). (Zie Figuur 8). Hierna zal alleen nog maar worden uitgegaan van de matrix interpretatie van een NN, en ook figuren zullen deze conventie afbeelden.



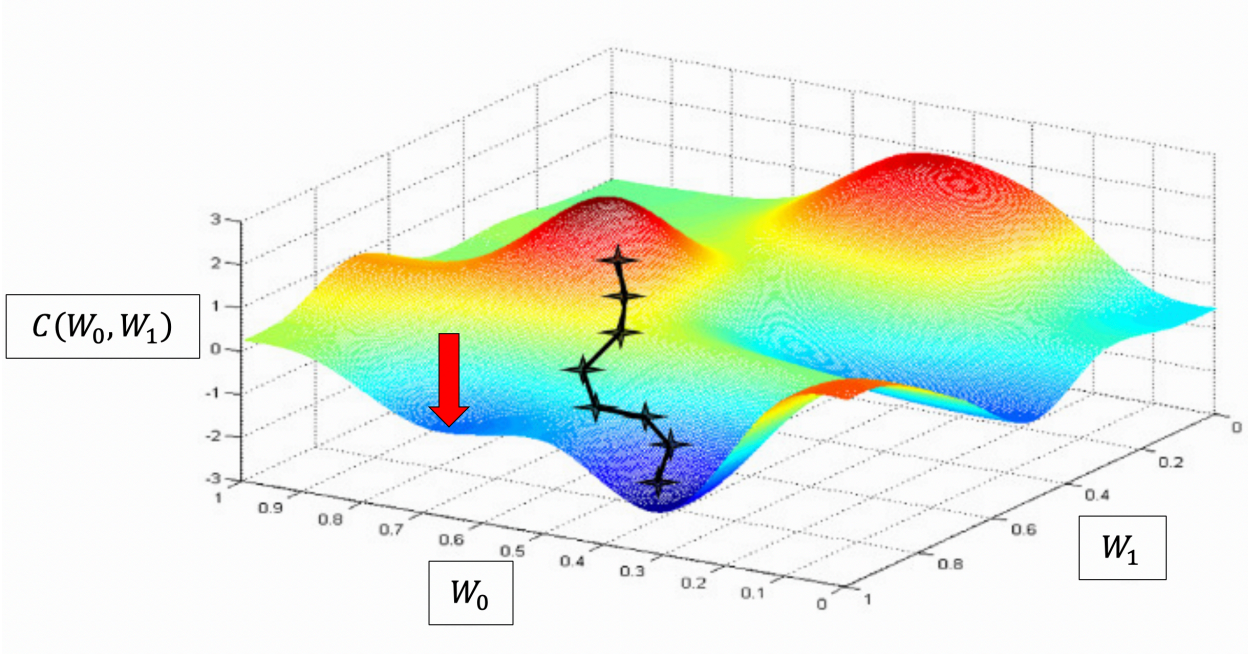
Figuur 8. NN, gevisualiseerd met scalaire en matrix interpretatie

De invoer wordt allereerst lineair getransformeerd: er worden in feite gewogen sommen berekend, waarbij een groot deel van de parameters van de cel als gewichten dient, maar sommige andere parameters, samen de zogenaamde bias, termen vormen die aan de gewogen sommen worden toegevoegd. Daarna wordt voor alle waarden in de ontstane matrix met een niet-lineaire functie, de activatiefunctie, de activatie van het neuron berekend (omdat het anders niet mogelijk is voor het NN om niet-lineaire verbanden überhaupt vast te leggen). De daaropvolgende lagen doen precies hetzelfde, maar hebben als invoer de activatie van de vorige laag. Theoretisch kan met deze relatief simpele structuur een NN iedere continue reëelwaardige functie modelleren. Anders gezegd betekent dat dat bewezen is dat NN'en theoretisch de oplossing voor ieder mogelijk probleem kunnen benaderen, al betekent dat niet dat een dergelijke oplossing ook altijd haalbaar is omdat het formaat van het NN hierbij geen limiet kent. (Nielsen, 2015) Zie Appendix 1.1 voor een diepgaandere wiskundige toelichting.

3.1.3 – Gradient descent

Om een NN te leren zo accuraat mogelijk bepaalde voorspellingen te doen gegeven bepaalde invoer, wordt gebruik gemaakt van GD. Tijdens GD wordt eerst bepaald in welke mate gedane voorspellingen door het NN correct zijn, door ze te vergelijken met de waarheid. Dit wordt gedaan met een zogenaamde *cost function*, die als argumenten de voorspellingen van het NN en de waarheid volgens de training data heeft, en als uitkomst de *cost*, ofwel, een mate van hoe 'slecht' het NN presteert. Het 'optimaliseren' van het NN door het aanpassen van de parameters van alle neuronen, gebeurt door de uitkomst van de *cost function* te minimaliseren. Dit wordt gedaan door met behulp van afgeleiden van de individuele parameters, waarmee je de benodigde aanpassing kan vinden om de uitkomst van de *cost function* te verkleinen. Een positieve afgeleide van de *cost function* naar een van de parameters betekent immers dat als je die parameter verkleint de *cost* ook verkleint; het

omgekeerde geldt voor een negatieve afgeleide.[1] Als deze aanpassingen voor alle parameters herhaaldelijk (iteratief) gemaakt worden wordt uiteindelijk een minimum gevonden in het 'oppervlak' van de *cost function*.[2] Vaak wordt de analogie getrokken met een bal die van een soort *cost* berg rolt. Alhoewel GD niet helemaal overeenkomt met een fysisch model van een rollende bal, is het een erg toegankelijke manier om GD te begrijpen. Zie Figuur 9 voor een visualisatie van het *cost* 'oppervlak' en het pad dat de parameters tijdens GD volgen voor een hypothetisch model met slechts twee scalaire parameters. Zie Appendix 1.2 voor een diepgaandere wiskundige toelichting.



Figuur 9. Optimalisatie met GD gevisualiseerd voor twee parameters

Om de eerdergenoemde (partiële) afgeleiden efficiënt te berekenen wordt gebruik gemaakt van *backpropagation*. Dit houdt simpelweg in dat de volgorde waarop de afgeleiden van de *cost function* naar de parameters berekend worden zo is dat er zo min mogelijk 'schakels' voor de kettingregel onnodig of opnieuw worden berekend. Dit komt er in de praktijk op neer dat eerst de afgeleiden voor de laatste laag van het NN worden berekend, daarna die van de op één na laatste laag, etc. Dit verklaart ook de naam *backpropagation*.

Soms wordt GD zo geïmplementeerd dat niet alle parameters even snel worden bijgesteld (*cf.* Appendix 1.2: zo dat α niet voor alle parameters gelijk is), zoals bij *Adagrad* GD (Duchi, Hazan, & Singer, 2011), of zo dat de aanpassingen 'momentum' houden om convergentie in lokale minima te voorkomen (Qian, 1999), of zo dat GD ook rekening houdt met afgeleiden

[1] Formeel heten deze afgeleiden partiële afgeleiden, omdat het gehele NN een multivariabele functie is. Bij partieel afleiden naar een bepaalde variabele, beschouw je tijdelijk de andere variabelen als constanten, waarna je in wezen een gewone afgeleide bepaalt.

[2] Feitelijk gaat het niet om een oppervlak, maar om een hyperoppervlak, omdat het gehele NN praktisch altijd gekenmerkt wordt door $\gg 2$ parameters. Een hyperoppervlak is een generalisatie van een oppervlak in een willekeurig aantal dimensies. De visualisatie in Figuur 9 van een oppervlak (dus in drie dimensies) is intuïtiever te begrijpen.

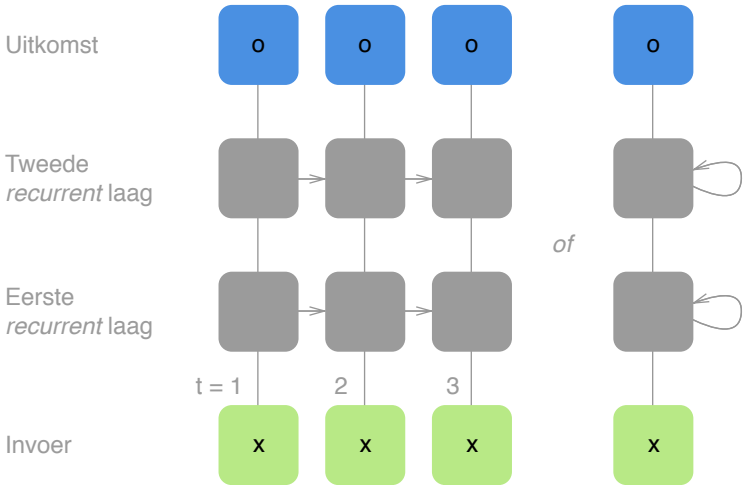
van hogere orde(s) (Battiti, 1992). Dit laatste is echter minder gebruikelijk vanwege de veel minder efficiënte benodigde berekeningen. Bij GD met momentum klopt de eerdergenoemde analogie met de 'rollende bal' iets beter.

Daarnaast wordt vaak niet alle training data tegelijk gebruikt voor GD, maar gaat de training per individueel voorbeeld. Deze wordt willekeurig geselecteerd uit alle voorbeelden waarop getraind kan worden. Als alle voorbeelden bij iedere trainingsupdate gebruikt worden betekent dat namelijk dat het relatief lang duurt voordat er ook maar een enkele aanpassing aan de parameters plaatsvindt. Deze aanpassing omvat dan weliswaar alle training data, maar als er in plaats daarvan veel meer updates op basis van losse voorbeelden waren gedaan was het totale NN naar alle waarschijnlijkheid beter geconvergeerd (geoptimaliseerd naar een ideale oplossing). In de praktijk blijkt dit goed te werken. Deze methode is bekend onder de naam *Stochastic Gradient Descent* ("SGD"). De training kan ook met groepjes voorbeelden worden gedaan (*cf.* Appendix 1.2: waarbij willekeurig b sets gegevens worden gebruikt per iteratie, met $0 < b < m$). Deze techniek staat bekend onder de naam *Mini-batch Gradient Descent* ("MBGD") (Robbins & Monro, 1951).

Met GD daalt de uitkomst van de *cost function* uiteindelijk richting een (al dan niet lokaal) minimum voor de training data door aanpassingen aan de parameters van het NN. Als er sprake is van een lokaal minimum, is de uitkomst van de *cost function* niet compleet geminimaliseerd, maar zitten deze parameters 'vast' in een lokale 'put' op het 'oppervlak' van de *cost function*. Bij de rode pijl in Figuur 9 is een voorbeeld van zo'n lokaal minimum te zien. Afhankelijk van de variant van het GD-algoritme dat is gebruikt, kan dit gebeuren (want in lokale minima is de afgeleide van de *cost* naar de parameters immers ook 0). Zelfs in lokale minima is het NN onder de juiste condities geschikt om voorspellingen te doen op nieuwe data, maar ook geldt logischerwijs dat zelfs bij een globaal minimum het NN niet altijd in staat is om waardevolle voorspellingen te doen op nieuwe data, omdat het bijvoorbeeld té specifiek is aangepast naar alleen de training data (dit heet *overfitting*). Het NN heeft dan niet goed kunnen generaliseren.

3.1.4 – Recurrent neural networks

Waar bij een gewoon NN per uitvoering van het model geen gegevens worden opgeslagen of 'onthouden' doet een RNN dat wel. Sommige neuronen worden zo gemaakt dat ze informatie van voorgaande uitvoeringen opslaan die dan weer in een volgende uitvoering kunnen worden meegenomen in de berekening van zijn nieuwe activatie. Ze zijn daardoor uiterst geschikt voor het verwerken van sequentiële data. Ondanks dat de manier waarop dit geheugen gebruikt wordt lang niet altijd betrekking heeft op hoe bepaalde gegevens over de tijd verlopen, wordt de volgorde waarop sequentiële informatie in een RNN wordt gevoed vaak de tijdsdimensie genoemd. Grafisch is de tijddimensie voor te stellen zoals in Figuur 10 is weergegeven.

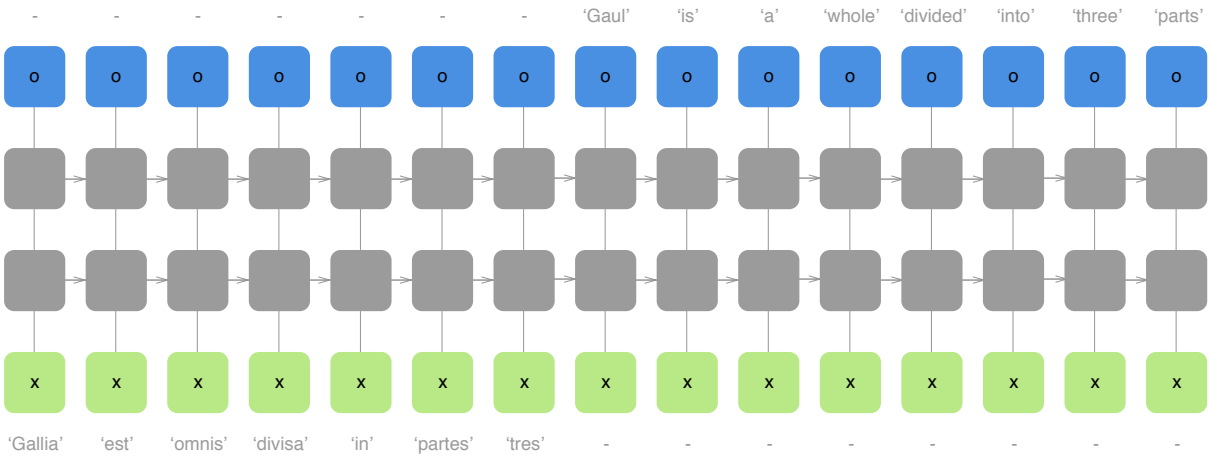


Figuur 10. RNN op twee verschillende manieren gevisualiseerd

De simpelste vorm van een RNN-cel staat beschreven in Appendix 1.3, maar in de praktijk blijkt het lastig om met zulke neuronen afhankelijkheden over lange tijdstermijn te leren (Hochreiter, 1991). Hiervoor zijn alternatieve cellen bedacht zoals de LSTM (Schmidhuber & Hockreiter, Long Short Term Memory, 1996), de GRU (Cho, et al., 2014), en het CW-RNN (Koutník, Greff, Gomez, & Schmidhuber, 2014).

Een voorbeeld van een toepassing van een RNN, om tevens een beter beeld te krijgen van het bovenstaande, is de volgende: moderne vertaalalgoritmen maken gebruik van RNNs om niet alleen woorden individueel te kunnen vertalen, maar ook sequenties van woorden met correcte grammatica. De woorden in de oorspronkelijke taal, als vectoren gerepresenteerd[3], vormen ieder een invoer van individuele uitvoeringen van het RNN. Een dergelijk getraind RNN heeft bijvoorbeeld eerst de invoeren voor de losse woorden ‘Gallia est omnis divisa in partes tres’ gevolgd door de uitvoeren overeenkomend met de vectoren voor de losse woorden ‘Gaul is a whole divided into three parts’. Dat ziet er zo uit als in Figuur 11 is weergegeven (het netwerk is uiteraard sterk versimpeld, en vandaag de dag zijn er al sterk verbeterde neurale vertaalalgoritmen, die overigens wel nog altijd met RNNs werken).

[3] Om dat te kunnen doen wordt vaak ook weer van *machine learning* gebruik gemaakt zodat deze vectoren de verbanden tussen verschillende woorden kunnen omvatten, om zo van enige betekenis te kunnen zijn voor het NN. Hoe dit precies werkt is niet belangrijk om dit voorbeeld te begrijpen.



Figuur 11. Voorbeeld toepassing RNN

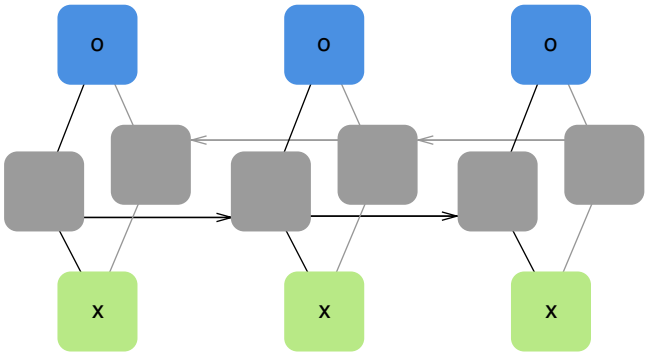
Sectie 4 – Methode

4.1 – Toepassing RNN voor transcriptiefactor hechtingslocaties

Ons model maakt gebruik van de eerder beschreven RNNs om gegeven de DNA-sequentie een voorspelling te doen over de aanwezigheid van specifieke transcriptiefactoren in bepaalde cellen. In dit hoofdstuk wordt de exacte implementatie toegelicht en over specifieke onderdelen van het gebruikte model verder uitgeweid.

4.1.1 – Bidirectionality

Een belangrijke toevoeging in dit model is dat de *recurrent cells bidirectional* zijn. Dit betekent dat er *recurrent cells* zijn die een functie zijn van (o.a.) een gegeven uit de vorige tijdsstap, maar óók cellen met een afhankelijkheid van de volgende tijdsstap. In Figuur 12 is dit concept (met slechts één verborgen laag, puur ter demonstratie) weergegeven.



Figuur 12. Bidirectional RNN met drie tijdsstappen

De reden dat gebruik gemaakt wordt van *bidirectional recurrent cells* is als volgt: de aanwezigheid of juist afwezigheid van transcriptiefactoren kan bepaald worden door beide flankerende stukken DNA-sequentie. Beide stukken kunnen immers bepaalde chemische eigenschappen hebben die de binding van een transcriptiefactor kunnen bewerkstelligen.

4.1.2 – Gated Recurrent Unit

Er wordt gebruik gemaakt van een *recurrent cell* die langere verbanden kan 'onthouden' dan de standaard RNN-cel van Elman die in Sectie 3.1.4 werd genoemd. Dit is noodzakelijk omdat belangrijke invloeden van de sequenties mogelijk tamelijk ver van de daadwerkelijke binding van een transcriptiefactor liggen. 'Tamelijk ver' betekent in dit geval verder dan waar een RNN-cel van Elman in de praktijk van kan leren, zo rond de enkele tientallen basenparen. De RNN-cel van Elman wordt eigenlijk nog maar zelden gebruik vanwege deze beperking. De cel die wel gebruikt wordt is de zogenaamde *gated recurrent unit* ("GRU") (Cho, et al., 2014). Deze lijkt sterk op de *long short-term memory* cel ("LSTM") (Schmidhuber & Hockreiter, Long Short Term Memory, 1996), maar heeft minder parameters en een netwerk van gelijke grote met deze cellen is dus sneller te optimaliseren. De GRU vertoont echter wel resultaten die vergelijkbaar zijn met die van de LSTM bij andere *sequence-modeling* taken (Chung, Gulcehre, Cho, & Bengio, 2014). Zie Appendix 1.4 voor een diepgaandere wiskundige toelichting.

4.1.4 – One-hot encoding

De nucleotiden worden gecodeerd in het model ingevoerd in een zogenaamde *one-hot encoding*. Dit houdt in dat de vier mogelijke nucleotiden in een 4D-vector ieder een eigen index hebben, waarin de desbetreffende nucleotide als het ware 'aangevinkt' wordt door op die index gelijk aan 1 en op alle andere indices gelijk aan 0 te zijn. Zo wordt bijvoorbeeld $G = \{1,0,0,0\}$, $C = \{0,1,0,0\}$, $T = \{0,0,1,0\}$ en $A = \{0,0,0,1\}$. Omdat de nucleotiden geen onderling verband hebben, is het tevens terecht dat alle waarden in de vector onafhankelijk van elkaar zijn: waar bijvoorbeeld G op $x_1 = 1$ geldt voor alle andere nucleotiden dat $x_1 = 0$. Dit lijkt wellicht voor de hand liggend, maar als we bijvoorbeeld met woorden zouden werken, zoals in het voorbeeld in Sectie 3.1.4, is een *one-hot encoding* niet erg praktisch, omdat het dan wenselijk is dat woorden die op elkaar lijken ook als vector dichter bij elkaar liggen.

4.1.5 – Binary log loss

Als *cost function* wordt gebruik gemaakt van de *binary log loss* tussen de voorspelde kans dat een bepaalde transcriptiefactor op een bepaald punt in de DNA-sequentie zit en de gemeten waarschijnlijkheid volgens de training data. De *binary log loss* geeft een beeld van

in hoeverre een voorspelling over de aan- of afwezigheid van een eiwit[4] representatief is voor de waarlijke kans (in dit geval zoals waargenomen in historische data). (Collier, 2015) Zie Appendix 1.5 voor een diepgaandere wiskundige toelichting.

4.1.6 – Optimalisatie

Voor dit onderzoek wordt een variant van GD gebruikt die zowel gebruik maakt van momentum, als een benadering van het tweede moment van de afgeleiden, genaamd Adam (een acroniem voor Adaptive Moment Estimation) (Kingma & Ba, 2015). Met het laatstgenoemde krijgt iedere parameter een eigen snelheid waarmee ze worden aangepast door GD, zodanig dat parameters die het *cost* 'oppervlak' sterk doen oscilleren minder snel worden aangepast, en parameters die rustiger een consistente richting volgen tijdens GD juist sterker worden aangepast. De keuze voor het GD-variant is ook op Adam gevallen omdat deze vorm in de praktijk bij veel *machine learning* problemen het best werkt. Soms kan het uiteraard zo zijn dat een andere of specifiek ontwikkelde optimalisatie beter werkt, maar vanwege tijdsbeperkingen hebben we bewust besloten om hiermee niet te experimenteren. Adam wordt in Appendix 1.6 verder wiskundig toegelicht.

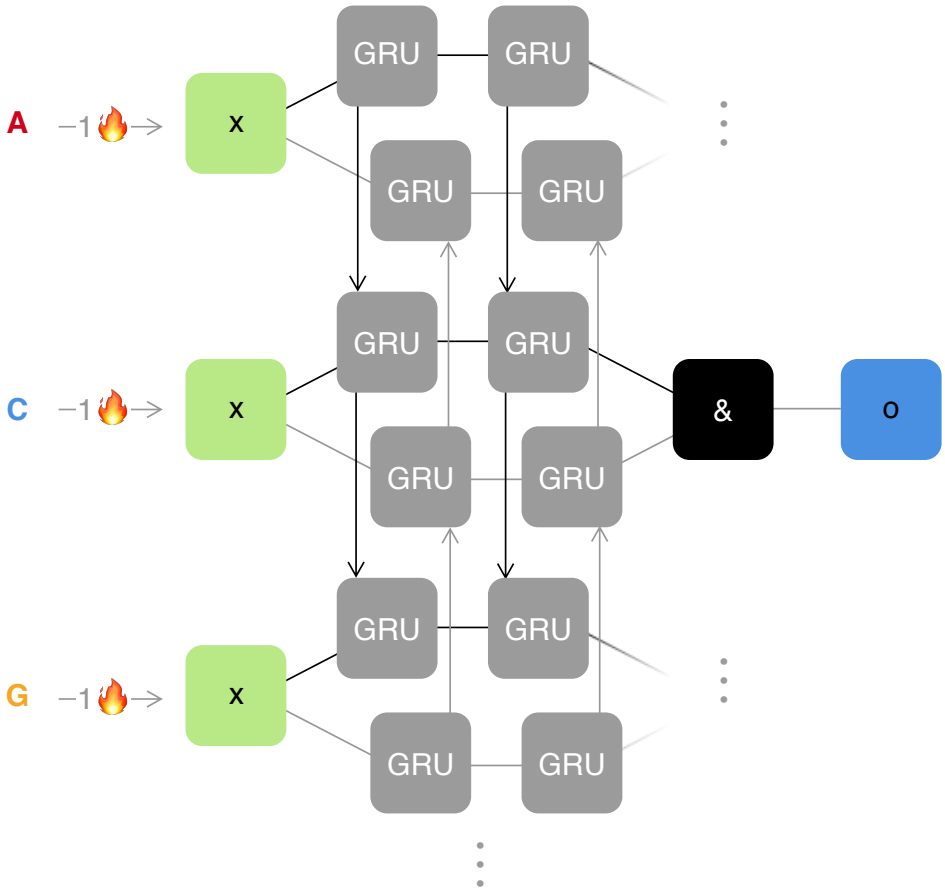
4.1.7 – Dropout

Dropout is een hele simpele techniek om *overfitting* bij het trainen van NN'en te voorkomen (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). De term *overfitting* wordt uitgelegd in Sectie 3.1.3. Praktisch komt het erop neer dat tijdens het trainen willekeurig neuronen worden 'uitgezet' volgens een bepaalde kans p, waardoor neuronen niet dezelfde taak op zich nemen om dat telkens vergelijkbare updates aan ze worden gedaan (zogenaamde co-adaptatie). Anders gezegd: het forceert de lagen waar *dropout* wordt toegepast om alle neuronen op een unieke wijze te benutten om tot een gegeneraliseerde oplossing te komen voor de training data. Op het moment dat het netwerk niet getraind wordt, maar gebruikt wordt voor voorspellingen (predictie), vallen de neuronen niet willekeurig weg, maar worden de parameters van de lagen met *dropout* zo bijgesteld dat er dan wel tijdens het sommeren resultaten zijn van vergelijkbare grootte. De parameters worden dus kleiner gemaakt met een factor $(1 - p)$. De uitkomst tijdens predictie is dus gelijk aan de verwachte uitkomst tijdens het trainen, maar volledig deterministisch.

[4] In het algemeen is de *binary log loss* bruikbaar voor iedere voorspelde Bernoulli-toevalsvariabele. Een Bernoulli-toevalsvariabele is de kans dat een bepaald evenement gebeurt (bijvoorbeeld de kans op kop bij het opgooien van een munt) en ligt dus altijd in $[0,1]$.

4.1.8 – Volledige architectuur

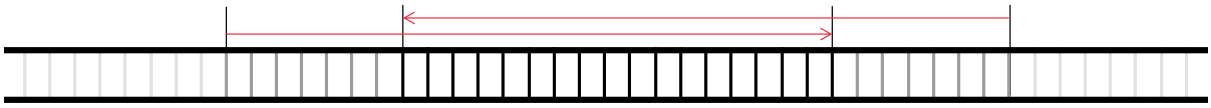
Een streng DNA wordt als in Figuur 13 is weergegeven verwerkt. Omdat er twee verschillende experimenten gedaan worden (zie Sectie 4.2.1) zijn bepaalde eigenschappen verschillend. De twee netwerken zullen vanaf hier worden aangeduid als 'klein' en 'groot'. Het zal zo duidelijk worden waarom dat zo is.



Figuur 13. Weergave van de volledige architectuur van ons model

De individuele nucleotiden worden na *one-hot encoding* in het *bidirectional* RNN met GRUs gevoed. Het RNN heeft twee lagen, en iedere GRU bevat in het grote netwerk 128 neuronen en in het kleine netwerk 32 neuronen. Dit formaat is vastgesteld op basis van de complexiteit van vergelijkbare netwerken, zoals die van J.M. Zhang en G.M. Kamath (Zhang & Kamath, 2016). De GRU-lagen maken ook gebruik van *dropout*. Vervolgens wordt een voorspelling over een regio van baseparen gedaan. In Figuur 13 wordt een regio van slechts één basepaar gebruikt, maar tijdens de training doen we voorspellingen over regio's van 200 bp bij het grote netwerk en 40 bp bij het kleine netwerk. Deze regio's voor het grote netwerk komen overeen met de resolutie van de dataset die we gebruiken, waarbij telkens regio's van 200 bp gelabeld zijn, steeds om de 50 bp. In werkelijkheid zijn bindingsplekken maar enkele tientallen baseparen breed, maar op deze resolutie kan niet of nauwelijks worden gemeten met ChIP-seq. Zelfs met een algoritme is het onhaalbaar om direct op het

niveau van baseparen voorspellingen te maken, o.a. omdat het trainen op het volledige genoom dan significant langer zou duren. De laatste uitkomsten van de RNNs binnen zo'n regio in beide richtingen worden aan elkaar geplakt (concatenatie), in Figuur 13 aangegeven met het symbool '&'. De beide RNNs starten tijdens de training echter ook buiten deze regio, bij flankerende stukken sequentie van maximaal 100 bp bij het grote netwerk, en exact 20 bp bij het kleine netwerk**.** De manier waarop het DNA gescand wordt, staat ook in Figuur 14 schematisch afgebeeld: de flankerende regio's worden alleen bekeken door het RNN dat van die richting komt. Daarna is er nog één gewone NN laag, die haar invoer van de geconcateneerde uitkomsten van de RNNs (dus van 256 waarden voor het grote netwerk en 64 waarden voor het kleine netwerk) uiteindelijk omzetten naar één reële waarde tussen 0 en 1. Dit is de voorspelling over de aanwezigheid van het eiwit, gegeven de DNA-sequentie.



Figuur 14. Manier waarop het DNA 'gescand' wordt, inclusief flankerende stukken

4.2 – Training

We hebben het voorgestelde algoritme ontwikkeld in de Python programmeertaal op het PyTorch framework (Facebook, 2018). Met een programmeertaal kun je een computer instrueren om bepaalde commando's en berekeningen uit te voeren. Op die manier kon ons model ook geïmplementeerd, getraind en getest worden. Python is een relatief gemakkelijk te leren programmeertaal die daarom ook erg veel in de academische wereld gebruikt wordt. Het *machine learning* framework PyTorch biedt een laag bovenop Python waarin een hoop vaak voorkomende berekeningen zijn geabstraheerd, waardoor bijvoorbeeld een GRU, inclusief de berekeningen die voor *backpropagation* nodig zijn, met een regel code kunnen worden omschreven:

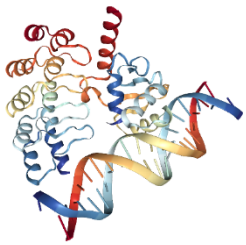
```
nn.GRU(input_size=4, output_size=32, num_layers=2, dropout=0.1)
```

Hetzelfde geldt voor bijvoorbeeld optimalisatie met Adam (zie Sectie 4.1.6) en een heleboel andere onderdelen van het model. Dit maakte het gemakkelijk om dingen snel te ontwikkelen en waar nodig snel aan te passen. Er waren ook enkele onderdelen die voor dit onderzoek zijn gebouwd, zoals de *binary log loss*, maar ook dit wordt met behulp van PyTorch redelijk gemakkelijk gemaakt. PyTorch is bovendien in staat berekeningen te optimaliseren en afgeleiden automatisch te bepalen. Verdere details over de training van het algoritme staan in deze Sectie. Verder is de broncode van het algoritme te vinden op GitHub (https://github.com/erikvdplas/binding-site-predictor).

4.2.1 – Datasets

De gebruikte data voor het grote netwerk is afkomstig van het ENCODE project (Stanford University, 2018) en omvat zowel de DNA-sequentie als de ChIP-seq pieken in een speciaal afgekapt format, waarbij de aan- en afwezigheid van eiwitten alleen door 0 en 1 worden gerepresenteerd. De experimenten die we hebben gebruikt komen uit een subset die reeds geselecteerd was voor de ENCODE-DREAM *in vivo Transcription Factor Binding Site Prediction Challenge*. (Stanford University, 2016) Om de data programmatisch toegankelijk te maken hebben we met behulp van de publieke code van een van de deelnemers van die wedstrijd, team BlueWhale, de ruwe data voorbereid (Team BlueWhale, 2017). Uiteindelijk hebben we van de beschikbare data één transcriptiefactor uitgekozen binnen één soort cel om de haalbaarheid van ons onderzoek te vergroten. De keuze is gevallen op het GABPA-eiwit in de H1-hESC stamcel.

Er zijn meerdere bekende motieven van de transcriptiefactor GABPA in de cel H1-hESC. Hierdoor kan na de training van het algoritme ook worden gekeken of het algoritme de echte motieven van dit eiwit kan herkennen. GABPA is een transcriptiefactor die invloed heeft op de expressie van cytochroom oxidase, dit is een enzym dat een belangrijke rol speelt in oxidatieve fosforylering, de ademhalingsketen. Deze transcriptiefactor heeft tevens invloed op andere functies van de mitochondriën. Ook kan deze transcriptiefactor invloed hebben op het syndroom van Down door zijn chromosomale ligging en doordat hij de mogelijkheid heeft om van polypeptiden (ketens van aminozuren) hetero-dimeren te maken: biologische complexen bestaande uit twee verschillende eiwitten (NCBI, 2010). GABPA is, gebonden aan het DNA, in Figuur 15 weergegeven.



Figuur 15. GABPA transcriptiefactor, gebonden aan het DNA

Omdat er motieven van het GABPA-transcriptiefactor bekend zijn, kunnen we ook willekeurig DNA genereren dat deze motieven bevat, wat als voordeel heeft dat we op kleinere schaal kunnen testen of het algoritme überhaupt bepaalde sequenties kan herkennen die verantwoordelijk zijn voor de binding van transcriptiefactoren. Dit is ook waar het kleine netwerk voor gemaakt is. We trainen het kleine netwerk op totaal willekeurige sequenties van verschillende baseparen, waarbij 1 op de 10 wél een bekend motief bevat, en de andere 9 op 10 wel willekeurig worden gegenereerd, maar wel zo dat het in ieder geval geen enkel motief bevat. In de data van ENCODE-DREAM is de verhouding van sequenties waaraan gebonden is ten opzichte van waaraan niet gebonden is veel kleiner, namelijk ongeveer 1 op de 1000. De training voor deze ‘neppe’ gegenereerde dataset is een stuk sneller omdat er een stuk minder data per trainingsupdate wordt

bekeken en het netwerk een stuk kleiner is en dus zelfs op CPUs (zie Sectie 4.2.2) snel genoeg functioneert.

4.2.2 – Gebruikte rekenkracht

Voor het trainen van het grote netwerk is gebruik gemaakt van de high performance computer ("HPC") van het bioinformatics lab van het Universitair Medisch Centrum Utrecht ("UMCU"). Deze HPC beschikt over een onderdeel met twee NVIDIA Tesla P100 graphical processing units ("GPU"). Deze GPUs kunnen de berekeningen die het algoritme maakt parallelliseren (dus waar mogelijk tegelijk uitvoeren), veel beter dan normale processors, central processing units ("CPU"). Deze node biedt 64 gigabyte aan werkgeheugen, waarmee we vrij veel data tegelijk kunnen verwerken: we kunnen hiermee theoretisch tot 10,000 training sequenties tegelijk in het geheugen hebben.
Voor het trainen van het kleine netwerk is slechts gebruik gemaakt van Erik's laptop: een MacBook Pro uit 2017, met een 2,9 gigahertz dual-core Intel Core i5 CPU en 16 gigabyte aan werkgeheugen.

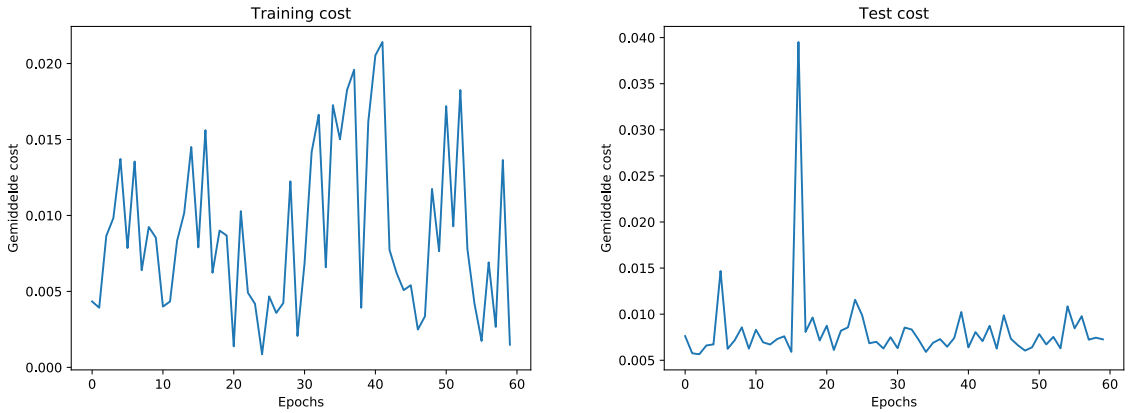
Sectie 5 – Resultaten

5.1 – Resultaten training

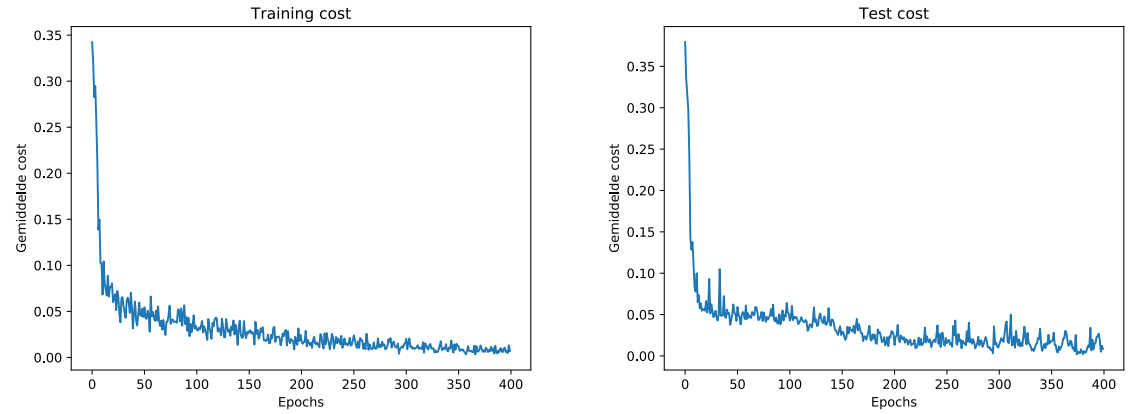
5.1.1 – Verloop training

De training van het grote netwerk op de HPC van het UMCU hebben wij niet zelf kunnen monitoren, dat werd voor ons gedaan door werknemers van het UMCU (zie Sectie 7.3). Het model is een aantal keer kort getraind, om te testen of het volledig werkte, en hiervoor zijn er zeker in het begin nog verscheidene aanpassingen aan de code gedaan. Vanwege tijdsbeperkingen en miscommunicatie is het slechts gelukt om één langere training te doen, waarvoor in totaal 52,428,800 sequenties in batches van 512 zijn gebruikt. Tijdens het trainen werden ook tests gedaan op steeds dezelfde 262,144 test sequenties, die uiteraard niet óók gebruikt werden voor de training. De training duurde bijna 40,5 uur.

Het trainen van het kleine model verliep een stuk vlotter, en heeft in ongeveer een uur op 921,600 sequenties getraind in batches van 16. De tests werden gedaan op telkens dezelfde 256 test sequenties, die ook weer niet gebruikt werden voor training.
De *binary log losses* die tijdens de training worden gerapporteerd geven inzicht in hoeverre het model over de tijd heen leert van wat het gezien heeft, en de *binary log losses* die tijdens het testen worden gerapporteerd geven inzicht in hoeverre het model over de tijd heen generaliseerd voor onbekende voorbeelden. Grafieken van beide *costs* over de tijd heen staan in onderstaande figuren: Figuur 16 voor het grote netwerk en Figuur 17 voor het kleine netwerk. Epochs, vermeld onderaan de grafieken, zijn de intervallen waarin tests worden uitgevoerd tijdens het trainen.



Figuur 16. Training en test cost van grote netwerk



Figuur 17. Training en test cost van kleine netwerk

5.1.2 – Statistische scores

Het model retourneert als voorspelling telkens een waarde tussen de 0 en de 1. De zogenaamde discriminatiedrempel, ook tussen de 0 en de 1, bepaalt vanaf welke waarde een voorspelling als een definitieve binding wordt beschouwd. Voor de test sequenties is vast te stellen in hoeverre voorspellingen, waarop een bepaalde discriminatiedrempel is toegepast, accuraat zijn. Hiervoor worden verschillende statistische scores gehanteerd:

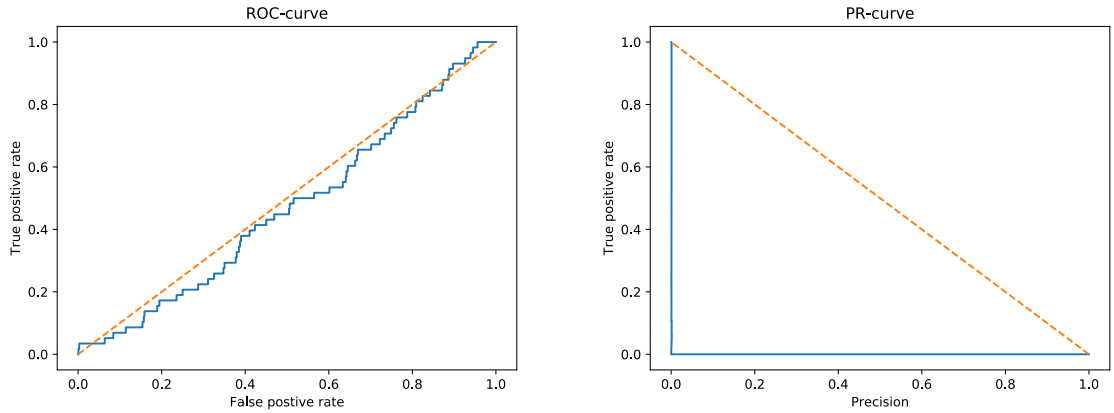
- De *true positive rate*, ook wel bekend als de *recall* of sensitiviteit is het aantal goed voorspelde bindingen gedeeld door het werkelijke aantal bindingen. In andere woorden is het een benadering van de kans dat het model een binding juist weet te voorspellen.
- De *false positive rate* is het aantal incorrect voorspelde bindingen (dus voorspelde bindingen, die in werkelijkheid helemaal niet bestaan) gedeeld door het werkelijke aantal niet-bindingen. In andere woorden is het een benadering van de kans dat niet-binding toch als binding wordt geclassificeerd.
- De precisie (in de literatuur dikwijls *precision*) is het aantal correct voorspelde bindingen gedeeld door het totale aantal voorspelde bindingen. In andere woorden

is het een benadering van de kans dat een voorspelde binding ook werkelijk een binding zou zijn.

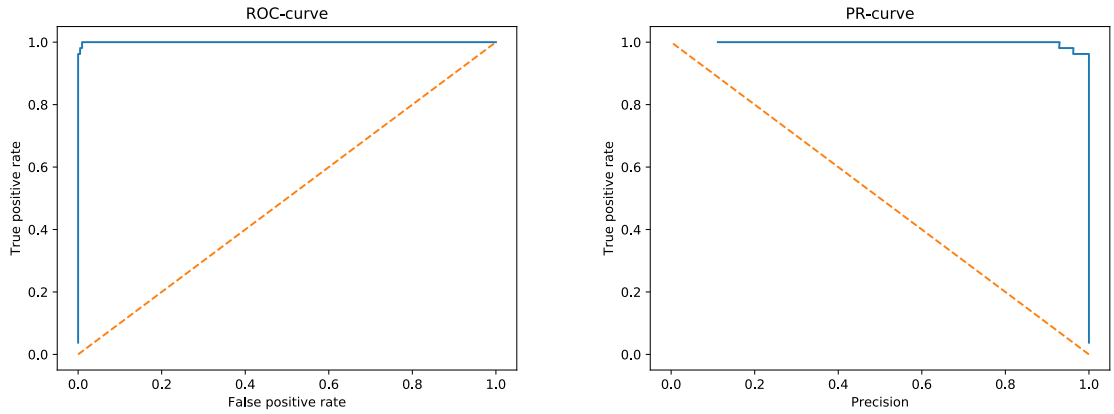
De waarden van deze statistische scores verandert als de discriminatiedrempel wijzigt. De relatie tussen de scores als de discriminatiedrempel tussen 0 en 1 alle mogelijke waarden aanneemt is in een grafiek weer te geven. Een veel gebruikte curve is de Receiver Operating Characteristics ("ROC") curve, waarbij de *true positive rate* tegen de *false positive rate* staat uitgezet. De oppervlakte onder de curve is ook een goede indicatie van hoe goed ons binaire classificatiemodel functioneert, en heet in de literatuur de auROC (voor *area under the ROC curve*). (Fawcett, 2004) Een andere curve die in tegenstelling tot de ROC-curve geen waarde toekent aan veel correct voorspelde niet-bindingen[5], is de *precision-recall* curve ("PR"). *Recall* is dus een ander woord voor de *true positive rate*. De *recall* wordt in deze curve uitgezet tegen *precision*, en ook hier is de oppervlakte onder de curve, die auPR wordt genoemd een indicatie van de kwaliteit van het netwerk. In het vergelijkbare, maar iets geavanceerdere model FactorNet (zie ook Sectie 6.2.3) worden ook deze scores gerapporteerd, en is ook het belang van deze scores vastgesteld (Quang & Xie, 2017). Omdat het extreem lang duurt om de ROC- en PR-curves voor het grote netwerk te genereren op de MacBook Pro van Erik en de HPC van het UMCU inmiddels niet meer beschikbaar was hebben we voor het grote netwerk alleen de curves, en niet de auROC en auPR exact kunnen vaststellen (overigens is aan de grafiek makkelijk te zien dat deze scores niet erg hoog zijn). Beide curves zijn voor het grote en het kleine netwerk geplot en respectievelijk in Figuur 18 en Figuur 19 weergegeven. Bij de ROC-curve staat de oranje stippellijn voor een hypothetisch model dat compleet willekeurige voorspellingen doet (wel volgens de juiste verhouding van *positives* en *negatives*).

N.B.: Bij de PR-curve is ook een oranje stippellijn zichtbaar, maar die was incorrect geplot door ons Python script: dat moest een horizontale lijn zijn ter hoogte van $\frac{1}{1000} = 0.001$ voor het grote netwerk en $\frac{1}{10} = 0.1$ voor het kleine netwerk, maar we konden de grafiek niet gemakkelijk opnieuw genereren met de gecorrigeerde baseline.

[5] Dit is immers niet heel knap: zeker niet voor het grote netwerk, waarbij 999 op de 1000 van de sequenties geen binding heeft, en dus zelfs een random model hier 999 op de 1000 sequenties goed classificeert. Bij het kleine netwerk is dit ook het geval alleen is de eerdergenoemde verhouding 9 op 10. Per definitie is voor onze modellen de auROC score dus hoger dan de auPR score.



Figuur 18. ROC- en PR-curve van grote netwerk op bijhorende test dataset



Figuur 19. ROC- en PR-curve van kleine netwerk op bijhorende test dataset

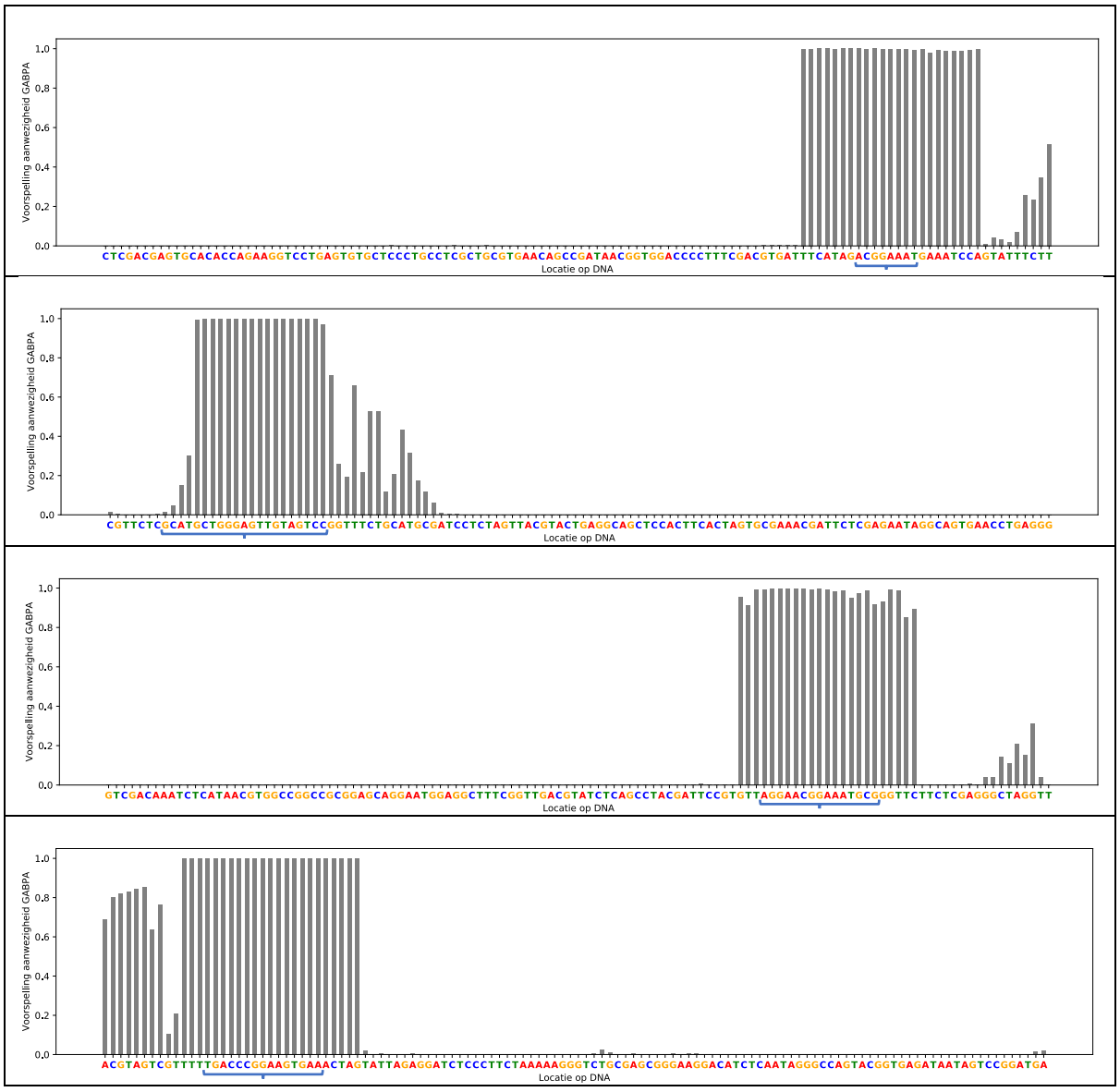
Voor het kleine netwerk zijn er wel auROC en auPR scores bepaald, voor het grote netwerk zijn schattingen gemaakt op basis van de grafiek:

	auROC	auPR
Grote netwerk	~0.5	~0.0
Kleine netwerk	0.9997310984627795	0.99794029395311

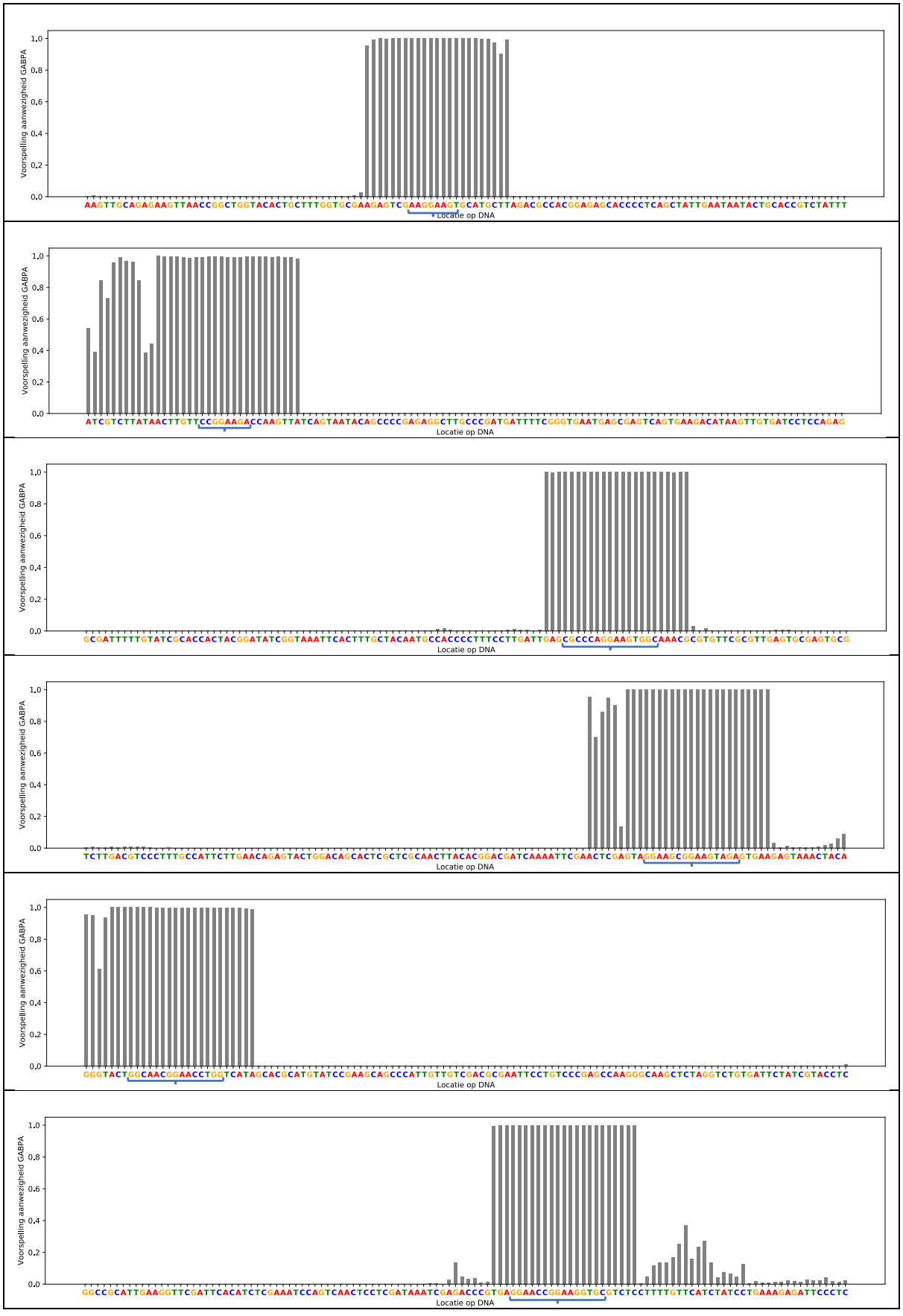
5.2 – Visualisaties

Naast statistische scores, zijn ook makkelijker te interpreteren tests gedaan: voor het kleine netwerk zijn visualisaties gemaakt voor willekeurige sequenties (zonder motieven) van 120 bp, waarin op een willekeurige plek een stukje voor een motief van GABPA wordt vervangen. Vervolgens is voor ieder basepaar voor de omvangende regio van 30 bp (waarop het model getraind is) een voorspelling gedaan over het binden van GABPA in die regio. Ook zijn de flankerende sequenties van 15 baseparen, op dezelfde manier als tijdens het trainen,

weer bekeken.[6] De voorspelling is in een staafdiagram per basepaar weergegeven en de locatie van het motief is aangegeven met een haakje onder het bijhorende motieven. In totaal zijn er 949 motieven gebruikt. Dit is inclusief herhalingen, maar die zijn bewust geconserveerd om een echte DNA-sequentie zo goed mogelijk te kunnen simuleren. De lijst is samengesteld op basis van de gegevens over experimenteel gevonden GABPA-transcriptiefactoren in H1-hESC op Factorbook (Myers, 2011). In totaal zijn er 10 visualisaties willekeurig gegenereerd om in dit verslag te verwerken, zie hieronder. We hebben besloten om dergelijke visualisaties niet voor het grote netwerk te maken, omdat de in Sectie 5.1 gerapporteerde resultaten sterk suggereren dat dat geen kwalitatief waardevolle resultaten zou hebben opgeleverd.



[6] Aan de uiteinden van de sequenties zijn de regio en haar flankerende stukken van 15 bp ingekort, omdat daar gegevens missen. Omdat het netwerk (met zijn RNN-architectuur) sequenties van variabele lengte kan verwerken, is dit mogelijk.


Voorspelling aanwezigheid GABPA
1.0
0.8
0.6
0.4
0.2
0.0
AAGTTGCAGAGAAGTTAACCGGCTGGTACACTGCTTTGGTGCGAAGAGTCGAAGGAAGTGCATGCTTAGACGCCACGGAGAGCACCCCTCAGCTATTGAATAATACTGCACCGTCTATTT
Locatie op DNA
Voorspelling aanwezigheid GABPA
1.0
0.8
0.6
0.4
0.2
0.0
ATCGTCTTATAACTTGTTCCGGAAGACCAAGTTATCAGTAATACAGCCCCGAGAGCCTTGCCCGATGATTTTCGGGTGAATGAGCGAGTCAGTGAAGACATAAGTTGTGATCCTCCAGAG
Locatie op DNA
Voorspelling aanwezigheid GABPA
1.0
0.8
0.6
0.4
0.2
0.0
GCGATTTTTGTATCGCACCACTACGGATATCGGTAAATTCACTTTGCTACAATGCCACCCCTTTCCTTGATTGAGCGCCCAGGAAGTGGCAAACGCGTGTTCGCGTTGAGTGCGAGTGCG
Locatie op DNA
Voorspelling aanwezigheid GABPA
1.0
0.8
0.6
0.4
0.2
0.0
TCTTGACGTCCCTTTGCCATTCTTGAACAGAGTACTGGACAGCACTCGCTCGCAACTTACACGGACGATCAAAATTCGAACTCGAGTAGGAAGCGGAAGTAGAGTGAAGAGTAAACTACA
Locatie op DNA
Voorspelling aanwezigheid GABPA
1.0
0.8
0.6
0.4
0.2
0.0
GGGTACTGGCAACGGAACCTGGTCATAGCACGCATGTATCCGAAGCAGCCCATTGTTGTCGACGCGAATTCCTGTCCCGAGCCAAGGGCAAGCTCTAGGTCTGTGATTCTATCGTACCTC
Locatie op DNA
Voorspelling aanwezigheid GABPA
1.0
0.8
0.6
0.4
0.2
0.0
GGCCGCATTGAAGGTTCGATTCACATCTCGAAATCCAGTCAACTCCTCGATAAATCGAGACCCGTGAGGAACCGGAAGGTGCGTCTCCTTTTGTTCATCTATCCTGAAAGAGATTCCCTC
Locatie op DNA

Sectie 6 – Conclusie

6.1 – Verwerking resultaten

In Sectie 5.1.1 zijn gegevens gerapporteerd over het verloop van de training. Om een vast aantal trainingsupdates is zowel voor het kleine als het grote netwerk over die periode telkens de gemiddelde *training cost* (de uitkomst van *binary log loss* op de voorspellingen die tijdens het trainen worden gemaakt) en de *test cost* over voor een aantal vooraf vastgestelde test sequenties bepaald en opgeslagen. Na afloop hebben we van de progressie van de beide *costs* grafieken gemaakt. Idealiter daalt de *training cost*, omdat het model van de training sequenties immers moet leren, maar ook de *test cost*, omdat dat aangeeft dat het model leert te generaliseren voor sequenties die hij nog niet gezien heeft. In de grafieken is te zien dat de *training cost* voor het grote netwerk met vrij grote oscillaties maar nauwelijks daalt. Blijkbaar is het niet gelukt om in de 102,400 trainingsupdates die het netwerk gedaan heeft goed van de training sequenties te leren. Men kan dan ook niet verwachten dat de *test cost* daalt, en dat is dan ook niet het geval: die blijft nagenoeg gelijk, met enkele oscillaties.
Enkele oscillaties waren verwacht, omdat er een vorm van MBGD gebruikt wordt, wat ervoor zorgt dat de afgeleiden voor trainingsupdates niet consistent zijn, maar door de variëteit van batches soms sterk kan verschillen in opeenvolgende updates. De extreme oscillaties zijn in ieder geval deels te verklaren door een tweetal fouten die zijn gemaakt tijdens de training: allereerst liep de training de training data lineair door (d.w.z.: van het begin tot het einde van het genoom, en zo vaak opnieuw als nodig), in de veronderstelling dat de GABPA-bindingsplekken bij benadering uniform over het genoom verdeeld zouden zijn. Na nadere inspectie bleek dit echter verre van waar te zijn: de bindingsplekken zitten vaak redelijk geclusterd binnen bepaalde chromosomen, of zelfs in de buurt van specifieke genen. Hierdoor kwam het maar zelden, maar dan wel redelijk vaak achter elkaar, voor dat een bindingsplek werd verwerkt tijdens het trainen. Tussendoor werd dan telkens getraind op niet-bindingen, wat er waarschijnlijk voor zorgde dat de vooruitgang voor het bepalen bindingsplekken weer teniet werd gedaan (het model begint dan al immers weer te leren dat er ‘vrijwel’ nooit een binding is). De tweede verklaring is een verkeerd ingestelde *dropout* in het grote netwerk. De *dropout* was ingesteld op $0{,}9$, wat betekent dat 90% van de neuronen willekeurig uitstaat tijdens de training. Dit, in combinatie met het feit dat het verwerken van een bindingsplek redelijk zeldzaam is (1 op 1000 voor het grote netwerk), resulteert in zeer beperkte invloed van de trainingsupdates voor positieve voorbeelden (voorbeelden waar wel een binding is, in tegenstelling tot negatieve voorbeelden, voor niet-bindingen). Waar *dropout* gebruikt is om *overfitting* te voorkomen, zorgt het in zulke grote mate voor extreme *underfitting*. De auROC en auPR scores bevestigen dit: het model doet het zelfs iets slechter dan een willekeurig model.
Het kleine netwerk heeft een aantal van de genoemde problemen met het grote netwerk aangepakt. De training sequenties bevatten nu uniform verdeeld positieve voorbeelden, een lagere concentratie negatieve voorbeelden en de dropout is aangepast naar 0,1. Ook kon de training beter gemonitord worden, dus bij problemen of matige resultaten konden we op tijd ingrijpen om aanpassingen te maken. Ook de schaal van het netwerk maakte dit makkelijker in de beperkte tijd waarin het onderzoek kon worden uitgevoerd. Zowel de *training cost* als de *test cost* dalen al vrij snel, dus het netwerk leert niet alleen van de training data, maar leert ook goed te generaliseren voor nieuwe sequenties. De auROC en

auPR scores zijn dan ook extreem hoog, bijna gelijk aan 1, wat een perfect model zou scoren. Hoewel dit netwerk niet getraind is op het echte genoom bewijst deze test wel dat het mogelijk is om met onze architectuur bepaalde patronen in sequenties van redelijke lengte kan herkennen. Vergelijkbare resultaten behalen op het echte genoom is een kwestie van het juist schalen van de techniek. Ook de visualisaties uit Sectie 5.2 leveren een bevestigend beeld van de prestatie van het model: de motieven worden in de 10 voorbeelden altijd goed herkent, en alle nucleotiden met een minimale afstand van 30 bp (de resolutie waar het model op getraind is) van het motief vertonen nauwelijks een uitslag die zou duiden op een binding. Het is duidelijk, ook uit andere visualisaties die ten behoeve van bondigheid niet zijn opgenomen in dit verslag, dat er goed een discriminatiedrempel is vast te stellen die optimaal de bindingen van niet-bindingen kan onderscheiden.

6.2 – Vergelijking met bestaande technieken

6.2.1 – Vergelijking met ChIP-seq

ChIP-seq is in staat om van écht DNA de bindingsplekken van specifieke eiwitten vast te stellen op enkele honderden baseparen nauwkeurig. De bindingen nemen in werkelijkheid echter enkele tientallen baseparen in beslag, die een stuk lastiger zijn vast te stellen. De onderzoeken die nauwkeuriger dan ChIP-seq de bindingsplekken vaststellen, kosten veel meer moeite en zijn dus duurder, en zijn bovendien nog niet voldoende uitgevoerd met het menselijke genoom om de resultaten toe te passen in pragmatischer onderzoek. Daarom is het naast ChIP-seq noodzakelijk om met computeralgoritmes voorspellingen te maken over exactere bindingsplekken. Het is bovendien noodzakelijk om snel en op grote schaal bindingsplekken te vinden om beter inzicht te krijgen in de effecten van de transcriptieregulatie door transcriptiefactoren, de effecten van mutaties in het DNA op de binding van eiwitten en uiteindelijk de gevolgen van bindingen in het fenotype (Jayaram, Usvyat, & Martin, 2016). De techniek is eigenlijk niet met onze techniek te vergelijken omdat ChIP-seq direct benaderde bindingsgegevens af kan lezen, maar het wel ondersteund moet worden met computeralgoritmes zoals de onze omdat het geen inzicht geeft in exacte bindingsplekken.

6.2.2 – Vergelijking met bestaande statistische modellen

Er bestaan al verscheidene technieken om de bindingsplekken van eiwitten te voorspellen, waarvan de eerste die we hebben kunnen vinden al uit 1989 komt, van het onderzoek van Stormo en Hartzell. Dit model en vergelijkbare modellen zonder machine learning gaan uit van klassieke statistiek en vaak worden alleen de positieve voorbeelden bekeken, om daar de kenmerkende baseparen te vinden ten opzichte van ‘gemiddeld DNA’ (bij veel organismen is dat met de baseparen nagenoeg uniform verdeeld) (Storno & Hartzell, 1989) (Bailey & Elkan, 1994). Hieruit volgt vaak een zogenaamd *position weight matrix* (“PWM”) die aan ieder basepaar op een bepaalde plek in een sequentie een waarde hangt in hoeverre die verantwoordelijk kan worden gerekend voor een uiteindelijke binding. Visueel wordt dit vaak weergegeven met een zogenaamd motieflogo zoals weergegeven in Figuur 20. De relatieve hoogte van iedere letter zegt iets over haar frequentie ten opzichte van de

andere letters in dat ‘stapeltje’ letters en de totale hoogte van het ‘stapeltje’ zegt iets over het belang voor een binding van die plek in de sequentie (D’haeseleer, 2006).

Figuur 20. Voorbeeld van een motieflogo

PWMs blijken in de praktijk enorm handig omdat ze met voldoende data tamelijk accuraat bepaald kunnen worden, en dan als ‘filter’ kunnen dienen op nieuwe sequenties om daar de aan- of afwezigheid van een bepaalde binding te kunnen vaststellen. Er zijn echter nadelen aan PWMs en de manier waarop ze met statistiek gevonden kunnen worden. Zo is het noodzakelijk om alle waarnemingen te reduceren tot een vast aantal PWMs, en kan een eventueel causaal verband tussen de nucleotiden niet worden bepaald (bijvoorbeeld: als guanine niet op locatie 4 in de sequentie zitten, dan moet die wel op locatie 5 zitten). Ook de lengte van PWMs staat altijd vast, terwijl in werkelijkheid omliggende nucleotiden best de binding kunnen beïnvloedden, maar dit wordt niet meegenomen. Ook statistische modellen die niet met PWMs werken adresseren deze problemen meestal niet, laat staan al die problemen tegelijk.

6.2.3 – Vergelijking met bestaande machine learning modellen

Er zijn in de afgelopen jaren enkele initiatieven geweest om *machine learning* te gebruiken in de genetische biologie in het algemeen. *Machine learning* is namelijk een stuk toegankelijker geworden, en is goed toepasbaar op problemen waar veel data voor beschikbaar is. Daarnaast kunnen modellen op basis van neurale netwerken de complexe verbanden ontrafelen die in de biologie voorkomen. Zo wordt *machine learning* inmiddels ook al voor de voorspelling van bindingsplekken van eiwitten op het DNA gebruikt, het idee van dit onderzoek is dus niet compleet nieuw. DeepBind is naar alle waarschijnlijkheid de eerste poging van deze toepassing. Het DeepBind model maakt geen gebruik van RNNs maar van een ander type complex NN genaamd *convolutional neural networks* (“CNN”). Ook CNNs zijn goed in het verwerken van grote hoeveelheden gegevens die een bepaald sequentieel verband hebben, maar verwerken wel altijd sequenties van vaste lengtes. DeepBind rapporteert een auROC van $0{,}71$ op een dataset waar een statistische methode op basis van PWMs slechts een score van $0{,}63$ rapporteert. De voordelen van *machine learning* zijn hier dus evident (Alipanahi, Delong, Weirauch, & Frey, 2015). De scores vallen niet direct te vergelijken met onze resultaten aangezien er niet dezelfde datasets en trainingprocedures zijn gebruikt. Wel valt te beargumenteren dat onze architectuur veelzijdiger is: door gebruik te maken van RNNs valt de training uit te breiden over langere sequenties, en er kunnen dus verbanden over langere afstanden ontdekt worden, die per definitie niet met de CNN-architectuur van DeepBind kunnen worden gevonden. Overigens doet DeepBind het naar alle waarschijnlijkheid in de getrainde staat beter, maar dat komt omdat er enkele fouten zijn gemaakt bij de training van ons grote netwerk. Dat RNNs waarde kunnen toevoegen aan *machine learning* in dit veld blijkt uit een nieuwer model dat

naast CNNs ook van RNNs gebruikmaakt, genaamd DeeperBind. De auteurs van DeeperBind rapporteren auROC-scores voor twee verschillende datasets, $0{,}92$ en $0{,}90$, waar DeepBind auROC-scores van respectievelijk $0{,}90$ en $0{,}81$ behaalt (Hassanzadeh & Wang, 2016). Een model uit 2017, FactorNet, dat ook gebruikmaakt van zowel CNNs als RNNs, geeft geen vergelijking met andere modellen, maar rapporteert wel louter auROC-scores ruim boven de $0{,}9$ (Quang & Xie, 2017). Het is duidelijk dat de toepassing van *machine learning* grote sprongen teweeg heeft gebracht voor deze specifieke taak. Helaas is het lastig om de prestaties van ons model te vergelijken met die van deze modellen, omdat we uiteindelijk geen goede training en ook geen goede tests hebben kunnen doen op het echte genoom. Wel hebben we aangetoond dat onze architectuur bepaalde patronen in sequenties van redelijke lengte kan ontdekken, dus als in later onderzoek nog eens kritisch naar bepaalde punten wordt gekeken waardoor het netwerk wél goed kan schalen, is het wellicht mogelijk om vergelijkbare resultaten te halen.

6.3 – Vervolgonderzoek

Omdat in dit onderzoek is aangetoond dat met een neuraal model met RNNs de patroonherkenningstaak voor het detecteren van bindingsplekken van eiwitten op het DNA goed te doen is, kan in volgende onderzoeken verder worden uitgezocht hoe het model is te schalen naar praktische toepassingen op echt DNA. In dit onderzoek is wel een poging gedaan om dat te doen, maar ondanks het feit dat dat niet heel goed gelukt is, is er wel een hoop duidelijk geworden over hoe dit in volgende onderzoeken beter aangepakt kan worden. Allereerst is het belangrijk om nog even goed te kijken naar de architectuur van het gepresenteerde model. Er zijn verscheidene andere succesvolle pogingen gedaan om met neurale modellen het probleem aan te pakken, wellicht kunnen we met meer tijd geavanceerdere technieken in het model verwerken, zoals bijvoorbeeld de CNNs die in Sectie 6.2.3 genoemd werden, die goed gecombineerd kan worden met het bestaande model, zoals ook in DeeperBind en FactorNet. Tijdens dit onderzoek was dat niet mogelijk vanwege een beperking op de complexiteit van ons model om het voor onze doelgroep begrijpelijk te houden. Ook het optimalisatieproces kan verbeterd worden. Hoewel de keuze voor Adam een goede lijkt te zijn (aangezien daarmee het kleine netwerk goed wist te convergeren), kan de manier waarop getraind worden nog worden aangepast: zo kan de verhouding van positieve voorbeelden ten opzichte van negatieve voorbeelden aan het begin van de trainisng vergroot worden om in het begin sneller te leren. Ook moeten uiteraard de problemen die in Sectie 6.1 werden genoemd verholpen worden. *Dropout* heeft in veel gevallen goed geholpen met de generalisatie van neurale modellen, maar werd in dit onderzoek in eerste instantie veel te hoog ingesteld. Voor het beste resultaat zouden met verschillende *dropouts* meerdere experimenten gedaan worden, maar bijvoorbeeld een *dropout* van $0{,}1$, zoals gebruikt voor het kleine netwerk, levert vermoedelijk betere resultaten op. De data moet daarnaast in willekeurige volgorde worden verwerkt, of in ieder geval op zo'n manier dat positieve voorbeelden om ongeveer gelijke intervallen gezien worden, in tegenstelling tot bij de volgorde waarin de sequenties in het genoom liggen. Als een training op het volledige (menselijke) genoom, bijvoorbeeld weer voor het GABPA-eiwit in de H1-hESC-cel, vervolgens wel succesvol is, kan worden gekeken in hoeverre het model geoptimaliseerd kan worden voor andere eiwit-cel paren of één enkel model kan worden

ontwikkeld dat meerdere verschillende eiwitten kan ontdekken, iets wat FactorNet bijvoorbeeld ook ondersteunt (Quang & Xie, 2017).

Sectie 7 – Discussie

7.1 – Terugblik

Vooraf aan het onderzoek hebben wij literatuuronderzoek gedaan over het onderwerp om zoveel mogelijk te weten te komen over ons onderwerp. Gaandeweg in het onderzoek kwamen wij steeds meer nieuwe biologische aspecten tegen in andere literatuur die wij nog niet vooraf hadden gelezen, wat achteraf wel bevorderlijk was geweest, vooral voor het ontwikkelen van ons grote netwerk. Voortaan zou het verstandig zijn om nog meer literatuuronderzoek te doen voordat we daadwerkelijk eigen onderzoek gaan doen. Ook zou het in het vervolg handig zijn om zelf toegang te hebben tot een HPC, waardoor wanneer er een fout is deze meteen kan worden opgelost. We zijn er pas te laat achter gekomen dat de training data enkel sporadisch een aantal GABPA-bindingen bevatte, waardoor het model niet goed heeft kunnen leren. Als we zelf direct toegang hadden gehad tot een HPC hadden we dit eerder kunnen inzien en de trainingsprocedure adequaat kunnen aanpassen.
Het probleem bij het grote netwerk met *dropout* dat in Sectie 6 meermaals naar voren kwam is het gevolg van een inconsistentie uit de literatuur, waarbij soms de *dropout* wordt gerapporteerd als de kans dat neuronen *blijven*, en soms als de kans dat neuronen juist *verdwijnen* tijdens de training. We hebben klakkeloos de waarde van $0{,}9$ van een model overgenomen dat was gebouwd met TensorFlow, een framework dat redelijk lijkt op PyTorch, maar wel precies de andere standaard met betrekking tot *dropout* hanteert. Deze onoplettendheid is waarschijnlijk voor een redelijk deel verantwoordelijk geweest voor de tegenvallende prestaties van het grote netwerk.

7.2 – Samenwerking

Het vinden van een onderzoeksvraag ging vrij voorspoedig. Al snel was het duidelijk dat we iets wilden doen met *machine learning* en biologie of natuurkunde. Na wat brainstormen is toen de keuze gevallen op een onderzoek met DNA en *machine learning*. Vervolgens hebben we na het lezen van literatuuronderzoek gekeken of ons onderzoek haalbaar zou zijn en hebben we een taakverdeling gemaakt. Erik heeft zich vooral beziggehouden met de *machine learning* en Bram met de biologie achter het onderzoek. Toen de voorkennis zo goed als geheel beschreven was en wij zelf ook genoeg kennis hadden om het onderzoek uit te voeren, zijn wij op zoek gegaan naar een manier om het door Erik gebouwde algoritme te kunnen laten draaien. Dit was mogelijk bij het Universitair Medisch Centrum Utrecht (UMCU).
Na de conceptversie hebben we onze stukken verbeterd en vooral de gedeeltes van elkaar nog nagekeken. De samenwerking in de loop van het onderzoek verliep goed en we hebben allebei de kans gekregen om onze eigen kwaliteiten te benutten.

7.3 – Dankwoord

In de zoektocht naar voldoende rekenkracht voor ons grote netwerk zijn wij de met mensen van de Universiteit van Utrecht van de bioinformatica faculteit in contact gekomen. We hebben gepraat dr. Jeroen de Ridder bij het onderzoeksgebouw van het UMCU over onze plannen, en naar aanleiding van dit gesprek enkele wijzigingen gemaakt aan ons plan: zo wilden we eerst meerdere eiwitten in verschillende cellen tegelijk onderzoeken, maar hebben we het onderzoek uiteindelijk beperkt tot één eiwit-cel paar. Ook hebben we door zijn opmerkingen een beter inzicht gekregen in bepaalde biologische aspecten van ons onderzoek. Daarnaast hebben we uiteindelijk toestemming gekregen om voor het trainen van ons algoritme gebruik te maken van de rekenkracht van de high performance computer ("HPC") van het lab. Er scheen pas recentelijk aan de HPC een node toegevoegd te zijn met GPUs, en die mochten we voor een redelijke tijd gebruiken. We konden zelf geen toegang krijgen tot de HPC, omdat er gevoelige patiëntgegevens op het systeem stonden, maar via dr. Joep de Ligt en Rene Janssen, HPC-administrator, konden we onze code van GitHub op de GPU HPC-node draaien. Naast deze hulp met ons onderzoek hebben we ook een kijkje kunnen nemen in het lab waar wij de nieuwste generatie DNA-sequencers met nanopores hebben kunnen zien, die op dat moment actief waren. Hoewel dit niet per se iets te maken had met ons onderzoek, was het wel erg interessant om dit te kunnen zien. Met dergelijke apparatuur is namelijk ook ooit de data voor ons onderzoek verkregen.
Ondanks dat we met het grote netwerk helaas niet de gewenste resultaten hebben gekregen, willen wij Jeroen de Ridder en Joep de Ligt heel erg bedankt voor de ondersteuning en de tijd die zij hebben genomen om ons verder te helpen met ons onderzoek.
Daarnaast willen wij ook onze profielwerkstukbegeleider Dan van de Poll bedanken voor het meedenken aan ons onderzoek en voor de gerichte feedback waardoor wij dit profielwerkstuk zo goed mogelijk konden maken.

Bronvermelding

10voorBiologie. (2013). *Introns en exons.* Opgeroepen op september 2018, van 10voorbiologie: https://www.10voorbiologie.nl/index.php?cat=9&id=393&par=395&sub=1627

abcam. (2014). *Chromatin structure and function: a guide.* Opgeroepen op september 2018, van abcam: https://www.abcam.com/epigenetics/chromatin-structure-and-function-a-guide

Alipanahi, B., Delong, A., Weirauch, M. T., & Frey, B. J. (2015). *Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning.* Opgeroepen op december 2018, van Nature Biotechnology: https://www.nature.com/articles/nbt.3300#ref32

Annunziato, A. T. (2008). *DNA Packaging: Nucleosomes and Chromatin.* Opgeroepen op september 2018, van Nature: https://www.nature.com/scitable/topicpage/dna-packaging-nucleosomes-and-chromatin-310

Bailey, T. L., & Elkan, C. (1994). *Fitting a mixture model by expectation maximization to discover motifs in biopolymers.* Opgeroepen op december 2018, van UCSD: https://www.sdsc.edu/~tbailey/papers/memeplus.tech.pdf

Battiti, R. (1992). *First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method.* Opgeroepen op augustus 2018, van ResearchGate: https://www.researchgate.net/profile/Roberto_Battiti/publication/2498372_First-_and_Second-Order_Methods_for_Learning_Between_Steepest_Descent_and_Newton%27s_Method/links/5537b11c0cf2058efdeae0b6/First-and-Second-Order-Methods-for-Learning-Between-Steepes

Cho, K., van Merriënboer, B., Gulcehre, C., Bougares, F., Schwenk, H., Bahdanau, D., & Bengio, Y. (2014, september 3). *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation.* Opgeroepen op september 2018, van arXiv: https://arxiv.org/pdf/1406.1078.pdf

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014, december 11). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.* Opgeroepen op september 2018, van arXiv: https://arxiv.org/pdf/1412.3555.pdf

Collier, A. B. (2015, 12). *Making Sense of Logarithmic Loss.* Opgeroepen op september 2018, van datawookie: https://datawookie.netlify.com/blog/2015/12/making-sense-of-logarithmic-loss/

D'haeseleer, P. (2006, april). *What are DNA sequence motifs?* Opgeroepen op november 2018, van NATURE BIOTECHNOLOGY: https://www.researchgate.net/publication/7177552_What_are_DNA_sequence_motifs

Duchi, J., Hazan, E., & Singer, Y. (2011). *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.* Opgeroepen op augustus 2018, van JMLR: http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

Elman, J. (1990). *Finding Structure in Time.* Opgeroepen op september 2018, van Wiley Online Library: https://onlinelibrary.wiley.com/doi/epdf/10.1207/s15516709cog1402_1

EpiGenie. (2013, februari 11). *EpiGenie Guide: ChIP Antibody Validation.* Opgeroepen op september 2018, van Epigenie: https://epigenie.com/recognizing-a-great-chip-antibody/

Facebook. (2018). Opgeroepen op september 2018, van PyTorch: https://pytorch.org

Fawcett, T. (2004, maart 16). *ROC Graphs: Notes and Practical Considerations for Researchers.* Opgeroepen op december 2018, van School of Systems Biology: http://binf.gmu.edu/mmasso/ROC101.pdf

Hassanzadeh, H. R., & Wang, M. D. (2016). *DeeperBind: Enhancing Prediction of Sequence Specificities of DNA Binding Proteins.* Opgeroepen op december 2018, van arXiv: https://arxiv.org/pdf/1611.05777.pdf

Hochreiter, S. (1991, juni 15). *Untersuchungen zu dynamischen neuronalen Netzen.* Opgeroepen op september 2018, van IDSIA: http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf

illumina. (2010). *Whole-Genome Chromatin IP Sequencing.* Opgeroepen op september 2018, van illumina: https://www.illumina.com/Documents/products/datasheets/datasheet_chip_sequence.pdf

Jayaram, N., Usvyat, D., & Martin, A. C. (2016). *BMC Bioinformatics.* Opgeroepen op december 2018, van Evaluating tools for transcription factor binding site prediction: https://bmcbioinformatics.biomedcentral.com/track/pdf/10.1186/s12859-016-1298-9

Khan Academy. (2018). *Transcription factors.* Opgeroepen op september 2018, van Khan Academy: https://www.khanacademy.org/science/biology/gene-regulation/gene-regulation-in-eukaryotes/a/eukaryotic-transcription-factors

Kingma, D. P., & Ba, J. L. (2015). *Adam: A Method for Stochastic Optimization.* Opgeroepen op september 2018, van arXiv: https://arxiv.org/pdf/1412.6980.pdf

Koutník, J., Greff, K., Gomez, F., & Schmidhuber, J. (2014, februari 14). *A Clockwork RNN.* Opgeroepen op september 2018, van arXiv: https://arxiv.org/pdf/1402.3511

Myers, R. (2011, juli 18). *Factorbook.* Opgeroepen op december 2018, van GABPA motifs H1-hESC: http://www.factorbook.org/human/chipseq/tf/GABPA

NCBI. (2010, oktober). *GABPA GA binding protein transcription factor subunit alpha [Homo sapiens (human)].* Opgeroepen op november 2018, van NCBI: https://www.ncbi.nlm.nih.gov/gene/2551

Nielsen, M. A. (2015). *A visual proof that neural nets can compute any function.* Opgeroepen op september 2018, van Neural Networks and Deep Learning: http://neuralnetworksanddeeplearning.com/chap4.html

Qian, N. (1999). *On the Momentum Term in Gradient Descent Learning Algorithms.* Opgeroepen op augustus 2018, van http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.5612&rep=rep1&type=pdf

Quang, D., & Xie, X. (2017, juni 18). *BioXiv.* Opgeroepen op december 2018, van FactorNet: a deep learning framework for predicting cell type specific transcription factor binding from nucleotide-resolution sequential data: https://www.biorxiv.org/content/biorxiv/early/2017/06/18/151274.1.full.pdf

Robbins, H., & Monro, S. (1951). *A Stochastic Approximation Method.* Opgeroepen op augustus 2018, van Project Euclid: https://projecteuclid.org/euclid.aoms/1177729586

Schmidhuber, J. (2014, april 30). *Deep Learning in Neural Networks: An Overview.* Opgeroepen op augustus 2018, van ArXiv: https://arxiv.org/pdf/1404.7828.pdf

Schmidhuber, J., & Hockreiter, S. (1996). *Long Short Term Memory.* Opgeroepen op september 2018, van CiteSeerX: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.634&rep=rep1&type=pdf

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014, juni). *Journal of Machine Learning Research.* Opgeroepen op december 2018, van Dropout: A Simple Way to Prevent Neural Networks from Overfitting: http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Stanford University. (2016). *ENCODE-DREAM in vivo Transcription Factor Binding Site Prediction Challenge.* Opgeroepen op september 2018, van Synapse: https://www.synapse.org/#!Synapse:syn6131484/wiki/402026

Stanford University. (2018). *ENCODE*. Opgeroepen op september 2018, van ENCODE: Encyclopedia of DNA Elements: https://www.encodeproject.org

Storno, G. D., & Hartzell, G. (1989, maart). *Identifying protein-binding sites from unaligned DNA fragments.* Opgeroepen op december 2018, van ResearchGate: https://www.researchgate.net/publication/20237426_Identifying_protein-binding_sites_from_unaligned_DNA_fragments

Team BlueWhale. (2017). *BlueWhale deep neural network that was used in the ENCODE-DREAM challenge 2016.* Opgeroepen op september 2018, van GitHub: https://github.molgen.mpg.de/wkopp/BlueWhale

VIDA. (2010). *Wat is DNA?* Opgeroepen op september 2018, van http://www.ziekdoorvoeding.nl/watisdna.php

What is epigentics. (2013). *Histone Modifications.* Opgeroepen op oktober 2018, van What is epigenetics: https://www.whatisepigenetics.com/histone-modifications/

Zhang, J. M., & Kamath, G. M. (2016, juni). *Learning the Language of the Genome using RNNs.* Opgeroepen op december 2018, van Stanford CS224D: https://cs224d.stanford.edu/reports/jessesz.pdf

Appendix 1 – Wiskundige toelichtingen

1.1 – Standaard neuraal netwerk

Mathematisch wordt een neuraal netwerk ("NN") vaak als volgt gedefinieerd:

$$a_l = \begin{cases} x, & l = 1 \\ act(a_{l-1} * W_l + b_l), & anders \end{cases}$$

Waarbij x de invoer van het NN is (een matrix, met iedere rij één set gegevens) en a_l de activatie van laag l. act is een 'activatiefunctie' (zonder een dergelijke functie zou het NN enkel een lineaire transformatie zijn van zijn invoer) die verschillende vormen aan kan nemen. Met $*$ wordt het matrixproduct aangeduid. W_l zijn de parameters van de neuronen in laag l (in een matrix) en b_l is de zogenaamde bias van laag l (een vector, deze wordt ook 'aangeleerd' tijdens *gradient descent* ("GD")). De uiteindelijke uitkomst van het NN wordt als volgt beschreven:

$$o = f_{out}(a_L)$$

Hierbij is o de uitkomst van het NN (een matrix, met in iedere rij één set voorspellingen voor dezelfde rij van invoer x). f_{out} is een specifieke functie die alleen in de laatste laag van het NN wordt toegepast (L is het totale aantal lagen van de NN, dus a_L is de laatste activatie van het NN) (dit om bijvoorbeeld een multinomiale verdeling als uitkomst te krijgen, maar ook vaak is $f_{out}(x) = x$, de identiteitsfunctie).

1.2 – Gradient descent

Als het NN leert om bepaalde voorspellingen te doen wordt gebruik gemaakt van GD. De uitkomsten van het NN worden hierbij vergeleken met de daadwerkelijke uitkomsten van training data (data waarvan de uitkomsten al bekend zijn) in een zogenaamde *cost function*. Deze geeft aan in welke mate het NN afwijkt van de werkelijkheid en neemt verschillende vormen aan. Hij straft als het ware het NN af voor fouten. Hij ziet er bijvoorbeeld zo uit (dit is de Mean Squared Error ("MSE") functie):

$$C(x) = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}\left(o_{i,j} - y_{i,j}\right)^2$$

Waarbij m het aantal rijen in matrix x is (en dus het aantal sets van gegevens waar voorspellingen over worden gedaan) en y_i de daadwerkelijke uitkomst voor de set gegevens op rij i van x (x_i). o_i is de voorspelde uitkomst van het NN voor x_i. n is het aantal voorspelde scalaire waardes per set gegevens, en dus het aantal kolommen in matrix o en y. De meest eenvoudige versie van GD werkt dan als volgt:

$$W_{l,i,j} := W_{l,i,j} - \alpha * \frac{\partial C(x)}{\partial W_{l,i,j}}, \qquad b_{l,i} := b_{l,i} - \alpha * \frac{\partial C(x)}{\partial b_{l,i}}$$

Alle parameters W en b worden herhaaldelijk op bovenstaande wijze aangepast, waardoor de waarde van $C(x)$ met vrijwel (er zijn een paar uitzonderingen) iedere iteratie verkleind wordt. α heet de *learning rate*, en regelt de snelheid waarmee de parameters veranderen. De waarde van α moet vaak met *trial and error* worden bepaald. Als de *learning rate* te hoog is veranderen de parameters te snel en kunnen ze voorbij minima gaan, waardoor de *cost* zelfs toe kan gaan nemen. Als de *learning rate* te laag is passen de parameters zich niet snel genoeg aan waardoor het leren overbodig lang duurt. Ook kan GD dan gemakkelijker vastraken in kleine lokale minima.

1.3 – Recurrent neural network

Recurrent neural networks ("RNN") behouden gegevens van vorige doorlopen van het NN om als het ware geheugen te simuleren. Activaties krijgen hiervoor een tijdsindex (bijv. $a_{l,t}$). De simpelste RNN-cel (Elman, 1990), hier de cellen met laagindices in set R, ziet er dan mathematisch ook zo uit:

$$a_{l,t} = act\left(a_{l-1,t} * W_l + a_{l,t-1} * U_l + b_l\right) \qquad l \in R$$

Hierbij zijn W_l en U_l twee verschillende parametermatrices. Beide paramatermatrices worden dan weer met GD geoptimaliseerd. *Backpropagation* werkt hier in principe hetzelfde, maar wordt toch vaak *backpropagation through time* ("BPTT") genoemd.

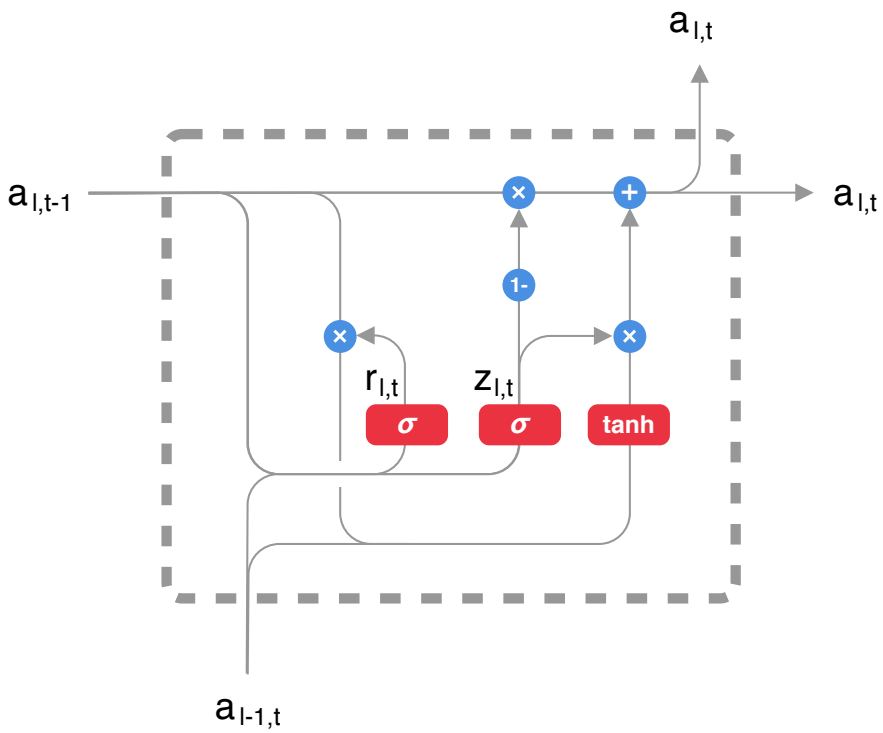
1.4 – Gated Recurrent Unit

De *gated recurrent unit* ("GRU") is een RNN-cel die in staat is relatief lange tijdsverbanden vast te leggen. Mathematisch ziet de GRU er als volgt uit:

$$z_{l,t} = act_g(a_{l-1,t} * W_{l,z} + a_{l,t-1} * U_{l,z} + b_{l,z})$$
$$r_{l,t} = act_g(a_{l-1,t} * W_{l,r} + a_{l,t-1} * U_{l,r} + b_{l,r})$$
$$a_{l,t} = \left(1 - z_{l,t}\right) \circ a_{l,t-1} + z_{l,t} \circ act_h(a_{l-1,t} * W_{l,a} + \left(r_{l,t} \circ a_{l,t-1}\right) * U_{l,a} + b_{l,a})$$

Hierbij zijn z en r respectievelijk *update* en *reset gates*. Het symbool $\circ$ staat voor het Hadamardproduct (waarbij van twee matrices van gelijke vorm de individuele elementen op dezelfde plaats worden vermenigvuldigt). De *gates* zelf worden gedefinieerd door een functie van de activatie uit de vorige laag in dezelfde tijdsstap en de activatie van dezelfde laag uit de vorige tijdsstap. Deze functie lijkt duidelijk sterk op de functie die de activatie van een gewoon neuron bepaalt. De *update gate* bepaalt hoe groot het deel is dat van de 'oude' activatie moet blijven, en in hoeverre deze door een vernieuwde activatie moet worden vervangen. Deze vernieuwde activatie is ook weer een activatie functie, maar hierbij wordt de invloed van de activatie van dezelfde laag uit de vorige tijdsstap beperkt door de *forget gate* (vandaar ook de naam). act_g en act_h zijn weer activatiefuncties, en act_g moet logischerwijs zo gedefinieerd zijn dat $act_g(x) \in [0; 1]$. Vaak worden voor deze activatiefuncties respectievelijk de logistische sigmoïde en de hyperbolische tangens gebruikt, en deze gebruiken wij ook in ons algoritme. In Figuur 21 staat een schematische

weergave van hoe de verschillende variabelen in een GRU-cel worden verwerkt. Ten behoeve van overzichtelijkheid zijn de parameters (inclusief bias) weggelaten.



Figuur 21. Schematische weergave van een GRU-cel

1.4.1 – Logistische sigmoïde en de hyperbolische tangens

De logistische sigmoïde en de hyperbolische tangens zijn als volgt gedefinieerd:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Beide functies worden erg vaak als activatiefunctie gebruikt. De functies zijn gemakkelijk af te leiden en kunnen allebei de uitkomsten van de oorspronkelijke functie gebruiken om de afgeleide op dat punt te berekenen, wat de uiteindelijke *backpropagation* erg efficiënt maakt:

$$Laat\ f(x) = \frac{1}{\sigma(x)} = 1 + e^{-x}$$
$$f'(x) = -\frac{\sigma'(x)}{\sigma(x)^2}$$
$$ook\ geldt\ f'(x) = -e^{-x} = 1 - f(x) = 1 - \frac{1}{\sigma(x)} = \frac{\sigma(x)-1}{\sigma(x)}$$

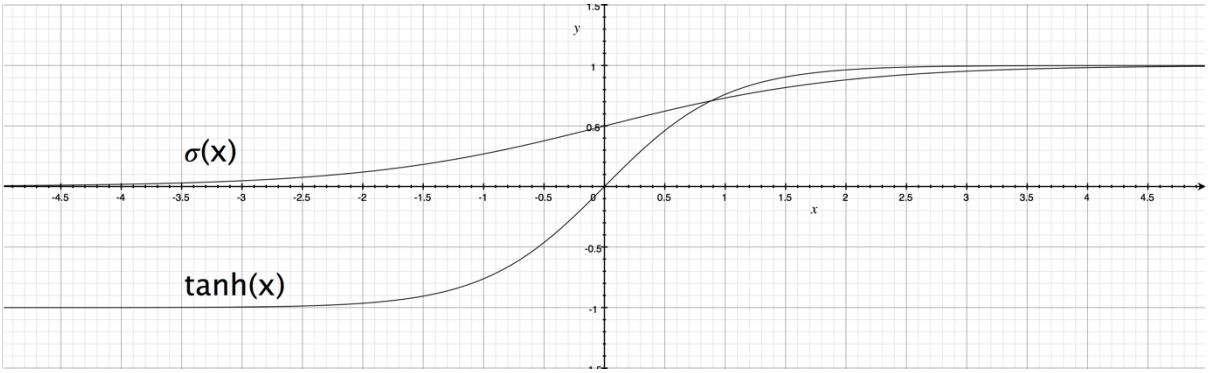
$$-\frac{\sigma'(x)}{\sigma(x)^2} = \frac{\sigma(x)-1}{\sigma(x)} \; dus \; \sigma'(x) = \sigma(x) * (1 - \sigma(x))$$

$$tanh'(x) = \frac{(e^x+e^{-x})^2-(e^x-e^{-x})^2}{(e^x+e^{-x})^2} = 1 - \tanh(x)^2$$

Je kunt met substitutie gemakkelijk aantonen dat de logistische sigmoïde enkel een getransformeerde versie is van de hyperbolische tangens en vice versa:

$$\sigma(x) = \frac{\tanh\left(\frac{x}{2}\right)+1}{2}$$

De logistische sigmoïde heeft twee horizontale asymptoten bij 0 en 1, de hyperbolische tangens dus bij -1 en 1 (zie ook Figuur 22).



Figuur 22. Plot van logistische sigmoïde en hyperbolische tangens

1.5 – Binary log loss cost function

De *binary log loss* is een simplificatie van de *cross entropy* bij multinomiale verdelingen, waarbij de kans op verschillende zogenaamde klassen wordt voorspeld. Dit is bijvoorbeeld het geval bij een neuraal model dat aan de hand van een foto kan bepalen welk dier er op de foto staat: hij voorspeld voor alle mogelijke dieren de kans dat dat dier op de foto staat. De kansen worden zo berekend dat ze bij elkaar altijd gelijk zijn aan 1 (dat is een eigenschap van een multinomiale verdeling).

De *cross entropy cost function* is (voor een RNN) als volgt gedefinieerd:

$$C(x) = -\frac{1}{m}\sum_{t=1}^{m}\sum_{i=1}^{v} y_{i,t} \ln\left(o_{i,t}\right)$$

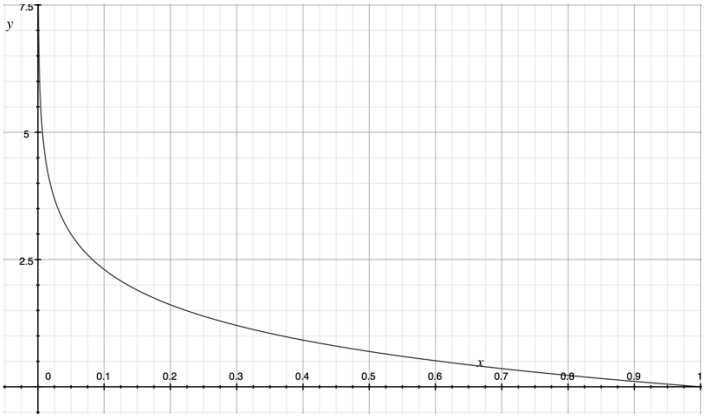
Hierbij is m het totaal aantal geëvalueerde tijdsstappen voor de trainingsupdate (dus bijvoorbeeld 1 bij SGD), v het aantal verschillende klassen waarover het model

voorspellingen doet en $o_{i,t}$ de voorspelde kans voor klas met index i op tijdstap t. $y_{i,t}$ is de waarschijnlijkheid volgens de gegeven labels van de training data.

Ons model zal geen multinomiale verdeling retourneren, maar slechts één waarde die iets zegt over de kans van de aanwezigheid van een transcriptiefactor. Je kan dit natuurlijk wel voorstellen als een multinomiale verdeling met twee klassen: de kans voor de aan- en afwezigheid, waarbij geldt dat de kans voor de een exact 1 minus de kans van de ander is. De *binary log loss* is dan ook zo versimpeld:

$$C(x) = -\frac{1}{m}\sum_{t=1}^{m}(y_t \ln(o_t) + (1 - y_t)\ln(1 - o_t))$$

$y_{i,t}$ is in onze dataset altijd 0 of 1, wat betekent dat voor iedere individuele voorspelling maar één van de termen binnen de bovenstaande sommatie meetelt. Deze vergelijking straft zekerheid over een foute voorspelling extra af, zoals duidelijk te zien is in een plot van $f(x) = -\ln(x)$ (wat gelijk is aan de *binary log loss* voor één enkele voorspelling x gegeven dat in werkelijkheid deze kans gelijk aan 1 had moeten zijn), zie Figuur 23.



Figuur 23. Plot van -ln(x)

De afgeleide van deze *cost function* is makkelijk te bepalen als de *non-linearity function* van de laatste laag gelijk is aan de logistische sigmoïde, waarbij â staat voor de uitkomst van de laatste laag, voordat deze *non-linearity* is toegepast:

$$\sigma(\hat{a}_t) = \frac{1}{1 + e^{-\hat{a}_t}}, \quad \left(1 - \sigma(\hat{a}_t)\right) = \frac{e^{-\hat{a}_t}}{1 + e^{-\hat{a}_t}}$$

$$\frac{\partial C(x)}{\partial \hat{a}_t} = -\frac{1}{m}\frac{d}{d\hat{a}_t}\left(y_t \ln\left(\sigma(\hat{a}_t)\right) + (1 - y_t)\ln\left(\frac{e^{-\hat{a}_t}}{1 + e^{-\hat{a}_t}}\right)\right)$$

$$= -\frac{1}{m}\left(y_t\left(1 + e^{-\hat{a}_t}\right)\sigma(\hat{a}_t)\left(1 - \sigma(\hat{a}_t)\right) - (1 - y_t)\left(\frac{e^{-\hat{a}_t}}{1 + e^{-\hat{a}_t}}\right)^{-1}\sigma(\hat{a}_t)\left(1 - \sigma(\hat{a}_t)\right)\right)$$

$$= -\frac{1}{m}\left(y_t\left(1 - \sigma(\hat{a}_t)\right) - (1 - y_t)\sigma(\hat{a}_t)\right)$$

$$= -\frac{1}{m}\left(y_t - \sigma(\hat{a}_t)\right)$$

$\sigma(\hat{a}_t)$ is al bekend, want dat is de uitkomst van de laatste laag, *inclusief* de toepassing van de logistische sigmoïde *non-linearity* (Collier, 2015).

1.6 – Wiskundige toelichting Adam Gradient Descent

Adam updates voor een willekeurige parameter W worden als volgt berekend:

$$m_i = \beta_1 m_{i-1} + (1 - \beta_1)\frac{\partial C(x)}{\partial W}$$

$$v_i = \beta_2 v_{i-1} + (1 - \beta_2)\left(\frac{\partial C(x)}{\partial W}\right)^2$$

$$\widehat{m}_i = \frac{m_i}{1 - \beta_1^i}$$

$$\hat{v}_i = \frac{v_i}{1 - \beta_2^i}$$

$$W := W - \frac{\alpha}{\sqrt{\hat{v}_i} + \epsilon}\widehat{m}_i$$

β_1, β_2, ϵ en α zijn constanten van het Adam optimalisatiemodel. ϵ dient enkel om delen door 0 te voorkomen. m_i en v_i (waarbij i de iteratie van het optimalisatieproces aangeeft) starten bij 0, waardoor ze met name in het begin van de training hier ook sterk naartoe blijven leunen. $\widehat{m}_i$ en $\hat{v}_i$ zijn hiervoor gecorrigeerd (Kingma & Ba, 2015).