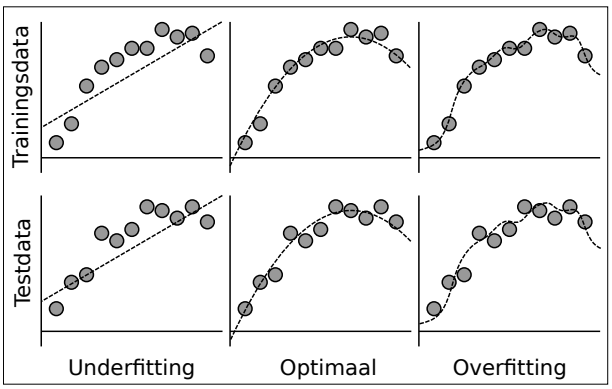
5. Regularisatie

Tijdens het trainen van neurale netwerken ligt het risico van overfitting op de loer. In dit hoofdstuk bekijken we een aantal methoden die erop gericht zijn om het aantal (of de grootte) van de fouten tijdens kruis-validatie te beperken, zelfs als dit ten koste gaat van meer (of grotere) fouten op de trainingsdata. Hieronder vallen in het bijzonder early stopping, data augmentatie, L_1- & L_2-regularisatie, en dropout.

5.1. Het bias-variance probleem

Afhankelijk van de complexiteit van het probleem in verhouding tot het model bestaat bij machine learning in het algemeen het risico op underfitting danwel overfitting.

We spreken van *underfitting* (of *undertraining*) wanneer het model niet voldoende flexibel is om de relaties die in de data aanwezig zijn te modelleren. Bijvoorbeeld, wanneer data niet lineair afhankelijk zijn (in het geval van een regressieprobleem) of niet lineair separabel (in het geval van een classificatieprobleem), en je zou deze data toch fitten met een lineair model, dan kan het model misschien wel een lineaire benadering vinden, maar het is niet in staat om de niet-lineariteiten goed te beschrijven. Daarvoor is het model in zekere zin te eenvoudig. Het bevat dan gewoonlijk te weinig parameters oftewel vrijheidsgraden om de ingewikkelde data voldoende nauwkeurig te benaderen. Er wordt dan ook wel gezegd dat het model een hoge *bias* heeft: het model maakt systematisch altijd hetzelfde soort fouten, ongeacht de steekproef aan data. (Let op: dit is een ander gebruik van het woord bias als de modelparameter b.) Dit komt overeen met de linker situatie in de figuur hieronder.



Omgekeerd spreken we van *overfitting* (of *overtraining*) wanneer het model juist té flexibel is en ook niet-bestaande toevallige effecten in de data gaat beschrijven. Wanneer

bijvoorbeeld een ingewikkeld model met tal van parameters wordt toegepast op een relatief simpele dataset met weinig datapunten, dan bestaat het risico dat het model niet alleen de systematische relaties gaat beschrijven die in de data aanwezig zijn, maar ook nog vrijheidsgraden "over heeft" om allerlei niet-reproduceerbare effecten die willekeurig in de data aanwezig zijn te fitten. In dit geval spreken we niet van bias maar van *variantie*: de kwaliteit van de fit varieert willekeurig met de toevallige effecten in de dataset. Dit wordt geïllustreerd in de rechter panelen van de vorige figuur.

In de praktijk wil je natuurlijk het liefst een model hebben dat net voldoende vrijheden heeft om alle systematische effecten in de data te beschrijven, maar geen extra vrijheden over heeft om ook de ruis te modelleren. Dit wordt de *bias-variance trade-off* genoemd. Als dat lukt dan noem je het model *parcimonieus*. In de figuur is op het oog redelijk te zien wat er aan de hand is, maar helaas is het in de praktijk meestal niet goed bekend wat de precieze vorm van het model zou moeten zijn of hoeveel parameters daarin zouden moeten worden opgenomen.

Om inzicht te krijgen in parcimonie, of dat je model lijdt aan under- of overfitting, wordt meestal gebruik gemaakt van meerdere afzonderlijke datasets.

1. De *trainingsdata* dienen om de modelparameters van de neuronen in het netwerk te optimaliseren. Dit betreft voor een neuraal netwerk de willekeurig geïnitialiseerde biases b_i en gewichten w_{ij} die automatisch door stochastic gradient descent met back-propagation worden bepaald.

2. De *validatiedata* dienen om de hyperparameters van het model geschikt te kiezen. Dit betreft instellingen die door de gebruiker handmatig worden gekozen, zoals de diepte van het netwerk, de breedte van de lagen, de gebruikte activatiefuncties en learning rate, of het aantal epochs waarover getraind wordt.

3. De *testdata* tenslotte dient om de kwaliteit van het uiteindelijke model betrouwbaar te evalueren. Deze data mag nooit worden gebruikt om model- of hyperparameters mee te optimaliseren; anders gezegd, zodra je de testdata gebruikt mag het model niet meer gewijzigd worden.

Overfitting is herkenbaar doordat het model op de trainingsdata significant beter presteert dan op de validatiedata. Dit duidt erop dat de modelparameters te specifiek zijn afgestemd op de trainingsdata, maar niet generaliseren naar andere steekproeven van soortgelijke data. Je kan overigens ook overfitting krijgen als je als onderzoeker te lang blijft tweaken aan je model en je hyperparameters te ver door optimaliseert. Dan presteert het model op de testdata slechter dan op de validatiedata. Het principe dat je een model dient te evalueren op een andere dataset dan waarmee je het optimaliseert wordt *kruis-validatie* genoemd.

In het geval van neurale netwerken speelt het bias-variance probleem een extra belangrijke rol omdat je als ontwerper van een neuraal netwerk de vrijheid hebt om zelf te kiezen hoeveel lagen je in een model opneemt, en hoeveel neuronen je in elk van die lagen stopt. Elk neuron heeft naast een bias-parameter b_i ook nog eens evenveel gewichten w_{ij} als het aantal inputs van de desbetreffende laag, dus daarmee loopt het totale aantal te

fitten parameters al gauw enorm op. Het aantal parameters bepaalt onder andere hoe goed het model in staat is om diverse soorten datasets te fitten. Dit wordt de *capaciteit* van het model genoemd. Een model met een te lage capaciteit zal underfitten; een model met een te hoge capaciteit neigt naar overfitten.

De capaciteit is niet in zijn eentje bepalend of er under- of overfitting plaatsvindt. Dit hangt ook samen met de hoeveelheid data en de complexiteit van het probleem. Een kleine hoeveelheid data met een eenvoudige structuur zal zelfs een eenvoudig model al gauw "van buiten" kunnen leren, terwijl grote hoeveelheden data met daarin ingewikkelde verbanden een complex model zullen vereisen. In de praktijk is het dan ook de verhouding tussen de capaciteit van het model enerzijds en de complexiteit van de data en het probleem anderzijds die bepaalt of er under- of overfitting optreedt.

		Model-capaciteit:	
		laag	*hoog*
Data-complexiteit:	*laag*	parcimonieus	overfitting
	hoog	underfitting	parcimonieus

In de praktijk ben je er niet zozeer in geïnteresseerd of je model het goed doet op de trainingsdata. Het gaat erom je model uiteindelijk toe te passen op nieuwe, onbekende data. Het eigenlijke doel is om de nauwkeurigheid op de testdata zo hoog mogelijk te maken, of de gemiddelde loss op de testdata zo laag mogelijk krijgen. Tot dusverre deden we dat door te proberen de loss op de trainingsdata te minimaliseren middels stochastic gradient descent en back-propagation. We gebruikten de loss op de trainingsdata als een meetbare surrogaatuitkomst of *proxy* voor de dan nog onbekende loss op de testdata. Er zijn echter methoden die weliswaar de prestaties op de trainingsdata verslechteren, maar toch de prestaties op de testdata typisch verbeteren. Deze heten *regularisatiemethoden* en zijn geschikt om overfitting tegen te gaan. In dit hoofdstuk zullen we er hiervan een aantal bekijken.

Opgave 80. *

Deel de volgende paren begrippen op in twee aparte groepjes van termen die allemaal bij elkaar passen: overfitting / underfitting; te simpel model / te complex model; hoge capaciteit / lage capaciteit; bias / variance; te veel vrijheidsgraden / te weinig vrijheidsgraden; eenvoudige datastructuur / ingewikkelde datastructuur; undertraining / overtraining; systematische fouten / willekeurige fouten; te veel modelparameters / te weinig modelparameters.

Opgave 81. *

Hoeveel modelparameters in totaal heeft een neuraal netwerk met een input layer met tien inputs, drie hidden layers met elk een breedte van honderd neuronen en een tanh-activatiefunctie, en een output layer met wederom tien neuronen en een softmax-activatiefunctie?

Opgave 82. *

Leg in je eigen woorden uit waarom je een model alleen maar in zijn definitieve uiteindelijke vorm mag toepassen op testdata, maar niet gaandeweg tijdens de ontwikkeling en optimalisatie.

Opgave 83. **
Een manier om overfitting tegen te gaan is door een grotere hoeveelheid trainingsdata te verzamelen. Denk na wat een dergelijke vergroting betekent voor de nauwkeurigheid op de trainingsdata en testdata in het geval van een model dat neigt naar overfitting. Is het vergroten van de trainingsdataset te beschouwen als een regularisatiemethode?

Opgave 84. **
Een andere manier om overfitting tegen te gaan is door de capaciteit van het gebruikte model te verlagen, bijvoorbeeld door minder neuronen in je model te verwerken. Denk na wat een dergelijke capaciteitsverlaging betekent voor de nauwkeurigheid op de trainingsdata en testdata in het geval van een model dat neigt naar overfitting. Is het verlagen van de capaciteit van het model te beschouwen als een regularisatiemethode?

Opgave 85. **
Anita beweert: "een model met een te *hoge* capaciteit vertoont *bias* als diens parameters onjuist geoptimaliseerd zijn", waarop Bob tegenwerpt: "een model met een te *lage* capaciteit vertoont *variantie* als diens parameters onjuist geoptimaliseerd zijn." Met wie ben je het eens?

Opgave 86. **
Regularisatiemethoden verbeteren de loss op testdata ten koste van een verslechtering in de loss op trainingsdata. Leg uit dat dit overfitting kan tegengaan.

Opgave 87. ***
Bij classificatie kan de nauwkeurigheid gemeten worden aan de hand van het percentage instances dat juist wordt geclassificeerd, dat wil zeggen de accuracy. Een hoge nauwkeurigheid en een lage loss zijn in de praktijk weliswaar gerelateerd, maar het verband is niet exact. Beschrijf een denkbeeldig voorbeeld van een classificatieprobleem waarin een (behoorlijk) hoge nauwkeurigheid toch kan samengaan met een (behoorlijk) hoge gemiddelde loss.

Opgave 88. ***
Motiveer welk van de volgende klassieke machine learning algoritmen in jouw ogen de allerlaagste capaciteit heeft, en welk de allerhoogste: J48-Tree, Naive Bayes, ZeroR, OneR, k-Nearest Neighbor, Logistische Regressie, Random Forest, Support Vector Machine.

5.2. Early stopping

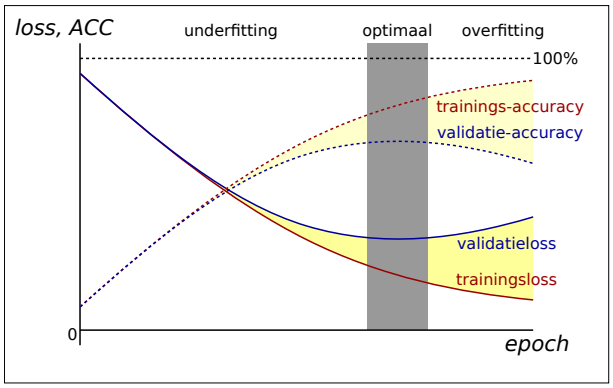
Overfitting treedt op wanneer een model te zeer is getraind op de fijne details van een trainingsdataset die niet generaliseren naar nieuwe data. Als je mag aannemen dat het model aanvankelijk een grove benadering vormt van de data en gaandeweg steeds fijner afgestemd raakt, dan is het aannemelijk dat het model tijdens de eerste epochs van de training vooral de algemene structuur van de trainingsdata leert die de trainingsdata wél gemeenschappelijk heeft met de testdata, en pas in latere epochs de kleine toevalligheden in de trainingsdata leert die niet reproduceerbaar hoeven te zijn in onafhankelijke data.

De training kan dan worden verdeeld in verscheidene fasen. Aanvankelijk is het model willekeurig geïnitialiseerd en ongetraind. Het benadert de data dan sowieso slecht, dus het vertoont bias, indicatief voor underfitting. Na verloop van een aantal epochs raakt het model echter getraind op de relevante structuur van de data en gaat het beter presteren op zowel de trainings- als testdata. Als het model een voldoende grote capaciteit heeft gaat het model uiteindelijk echter ook toevallige kenmerken van de trainingsdata leren. Het gaat dan geleidelijk lijden aan variance, indicatief voor overfitting. Het presteert dan nog wel steeds beter op de trainingsdata, maar verslechtert op de testdata.

Ergens daar tussenin ligt vermoedelijk een optimum waar het model een balans heeft tussen underfitting en overfitting. Dit zouden we graag bereiken, maar we weten natuurlijk niet van tevoren hoeveel epochs hiervoor nodig zijn. We kunnen dit echter doen door tijdens de training de loss op zowel de trainingsdata (waarmee we optimaliseren) als de validatiedata (waarmee we niet optimaliseren) bij te houden. De loss op de validatiedata gebruiken we als proxy voor de loss op de testdata, waarin we eigenlijk geïnteresseerd zijn.

Aanvankelijk zal de loss op zowel de trainings- als validatiedata gezamenlijk afnemen naarmate het model verbetert. Op een gegeven moment vlakt de validatieloss echter af, en gaat deze mogelijk zelfs toenemen, terwijl de loss op de trainingsdata langzaam verder afneemt. Het moment waar deze twee losses uit elkaar gaan lopen is het moment waarop overfitting een rol begint te spelen.

Een zinvol hulpmiddel om dit gedrag te visualiseren is de *validatiecurve*. Hierin zet je als functie van het volgnummer van de epoch (op de horizontale as) de trainings- en validatielosses (op de vertikale as) uit. Als alternatief kan ook de behaalde nauwkeurigheid worden uitgezet, of een andere relevante maat die iets zegt over de kwaliteit van het model (zoals de false positive rate of positive predictive value, de area under the ROC-curve, de F1-score, enzovoorts). Hieronder is een dergelijke validatiecurve geschetst met de gemiddelde loss in doorgetrokken lijnen, en in dezelfde figuur de nauwkeurigheid (accuracy, ACC) in stippellijnen. In de praktijk zullen deze grafieken overigens veel ruiziger en minder glad verlopen, deels omdat het model tijdelijk nabij lokale minima van de loss-functie kan blijven steken, en deels omdat de verschillen tussen de trainings- en testdata toevallig zijn.



Je kunt de informatie in deze validatiecurve op verschillende manieren gebruiken. Je

zou tijdens het trainen de modelparameters kunnen onthouden van het model met de laagste gemiddelde loss (of hoogste nauwkeurigheid) op basis van de validatiedata. Of je zou het model een aantal keren kunnen trainen met willekeurige beginwaarden en telkens kijken na hoeveel epochs ongeveer het optimum in de validatieloss wordt bereikt; je weet dit pas zodra het model reeds overfit is geraakt, maar daarna kun je een nieuw model precies zoveel epochs trainen als optimaal nodig is.

Het zal je niet verbazen dat deze methode *early stopping* wordt genoemd. Het is een regularisatiemethode. Immers, door vroegtijdig te stoppen wordt de trainingsloss niet zo klein als deze had kunnen zijn geweest door langer door te trainen; de validatieloss echter is wel beter dan wat je zou bereiken als je langer door was gegaan.

Opgave 89. *

Verwacht je dat de trainings-loss willekeurig dicht naar nul zou dalen, en de trainings-accuracy helemaal naar 100% zou naderen, als je "oneindig lang" zou kunnen door blijven trainen?

Opgave 90. **

Hoe hangt het antwoord op de vorige vraag af van de capaciteit van je model?

Opgave 91. **

Normaal neemt de loss op de trainingsdata geleidelijk aan af, in het begin snel en geleidelijk aan langzamer, soms met haperingen. De loss op de trainingsdata zou echter nooit systematisch mogen toenemen: immers, gradient descent probeert precies de loss op de trainingsdata stapsgewijs te minimaliseren. Toch komt het wel eens voor dat de loss op de trainingsdata zichtbaar wild heen en weer springt en gedurende sommige epochs flink toeneemt. Wat zou er dan aan de hand kunnen zijn?

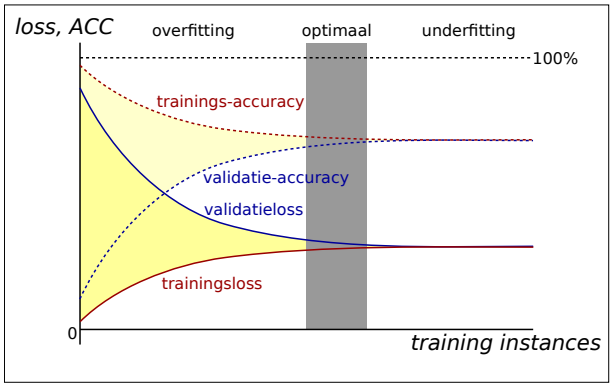
Opgave 92. **

Stel dat de validatiecurves nogal ruizig zijn, en je kiest als uiteindelijke modelparameters van je optimaal getrainde model die waarden die actueel waren toen de validatie-loss minimaal was. Is deze behaalde minimale validatie-loss dan een betrouwbare maat voor de kwaliteit van het model die generaliseert naar de testdataset, denk je? Motiveer je antwoord.

5.3. Data augmentatie

Het gedrag van een model omtrent over- en underfitting hangt ook af van de trainingsdata. Immers, overfitting treedt op wanneer een model in staat is om toevallige details in de trainingsdata te beschrijven. Wanneer de trainingsdataset echter groter en groter wordt, wordt het steeds onwaarschijnlijker dat die toevallige details voor álle trainingsdata blijven gelden. Met andere woorden, toevalligheden middelen steeds meer uit in de trainingsdata naarmate er hier meer van beschikbaar zijn. Hierdoor gaat de trainingsloss omhoog, in de richting van de validatieloss. Tegelijkertijd zal het model representatiever worden naarmate er meer trainingsdata is om van te leren. Dit zal ook tot uiting komen in de validatiedata: de prestaties hierop worden geleidelijk iets beter.

Ook dit gedrag kunnen we samenvatten in een grafiek. Dit keer zetten we op de horizontale as niet een hyperparameter zoals het aantal epochs uit, maar het aantal trainingsinstances. Langs de vertikale as zetten we wederom de loss of de nauwkeurigheid (of een soortgelijke kwaliteitsmaat) uit; in het voorbeeld hieronder zijn die weer respectievelijk met doorgetrokken en gestippelde lijnen geschetst. De resulterende grafiek wordt de *leercurve* of *trainingscurve* genoemd.



Wanneer er maar weinig trainingsdata beschikbaar is, is het risico op overfitting het grootst. Naarmate er meer data voorhanden is reduceert dat risico en lopen de curven geleidelijk naar elkaar toe. Er is dan geen sprake meer van overfitting. Wel zou er underfitting kunnen optreden. Dit hangt overigens ook af van de capaciteit van het model in relatie tot de complexiteit van de data: als de data nog structuur bevat die het model niet kan beschrijven, dan is er sprake van underfitting; echter, als het model in essentie alle structuur te pakken heeft die er in de data aanwezig is, dan blijft het model optimaal.

Vanaf het moment dat de curven niet meer significant van elkaar verschillen heeft het niet veel zin om nog meer trainingsdata toe te voegen. Omdat het in de praktijk vaak duur is om betrouwbare data te verzamelen kan de leercurve een goede indicatie vormen of je je energie het beste kan steken in het vergaren van meer trainingsdata of in het pogen te verbeteren van de opzet van het model.

Er is echter een goedkoop alternatief voor het verzamelen van meer trainingsdata. Vaak is het mogelijk om de attributen van trainingsinstances enigszins te wijzigen, waarna nieuwe instances ontstaan die eveneens mogelijk hadden kunnen zijn. Het hangt van de aard van het probleem af welk soort transformaties mogelijk zijn. Soms kan volstaan worden met het toevoegen van een hoeveelheid ruis, bijvoorbeeld door normaal verdeelde random waarden op te tellen bij numerieke attributen. Natuurlijk dient de hoeveelheid ruis niet dusdanig groot te zijn dat het karakter van de instance zodanig verandert dat deze bijvoorbeeld niet meer als een realistische instance kan worden herkend of zelfs tot een andere klasse zou kunnen gaan behoren. Soms is het mogelijk om attributen van op elkaar lijkende instances van eenzelfde klasse te mengen of te middelen.

Als instances bestaan uit afbeeldingen zijn transformaties met name nuttig. Zo kan het bijvoorbeeld toelaatbaar zijn om de afbeelding te spiegelen, roteren, transleren, schalen, of croppen, of om de kleur, contrast, helderheid, saturatie, of scherpte te veranderen. Een

microscopiefoto van cellen in een weefsel zou bijvoorbeeld prima op allerhande manieren getransformeerd kunnen worden, zolang hoogstens niet dusdanig ver in-/uitgezoomd wordt dat de celgrootte onrealistisch wordt; afbeeldingen van een handschrift kunnen daarentegen niet zomaar gespiegeld of gedraaid worden omdat dan p's, q's, b's, en d's in elkaar zouden worden omgezet, of zessen negens zouden worden. Hoe dan ook, de te leren uitkomsten dienen *invariant* te zijn onder de transformatie.

Het proces van het modificeren van data waarbij varianten van instances geproduceerd worden heet *data augmentatie*. Data augmentatie is niet zo effectief als het verzamelen van meer trainingsdata, omdat er niet werkelijk meer diversiteit in de trainingsdata ontstaat, maar het kan desalniettemin bijdragen tot het verminderen van overfitting. Het kan vooraf worden toegepast om de trainingsdataset kunstmatig te vergroten, maar het is gebruikelijker om tijdens het leren elke instance opnieuw willekeurig te transformeren. In dit laatste geval hoeft tijdens de training nooit twee keer exact dezelfde instance te worden aangeboden aan het model, maar dat is alleen haalbaar als het uitvoeren van de transformatie zelf niet een tijdrovende operatie is.

Data augmentatie is een vorm van regularisatie omdat het de prestaties op trainingsdata verslechtert, ten gunste van de prestaties op de test- en validatiedata. Dit komt duidelijk tot uitdrukking in de leercurve hierboven.

Opgave 93. *

De bekende iris-dataset bevat voor drie verschillende soorten irissen (*Iris setosa*, *Iris virginica* en *Iris versicolor*) de lengte- en breedte-afmetingen van kelk- en kroonbladeren. Het totale aantal instances (50 per klasse) is echter zeer beperkt. Hoe zou je deze dataset kunnen augmenteren?

Opgave 94. *

Welke vormen van data augmentatie acht je zoal geschikt wanneer je een model traint om aan de hand van kleurenpasfoto's gezichten te herkennen?

Opgave 95. *

Stel je wil de verschillende soorten terreinen in satellietfoto's classificeren; denk aan water, bos, heide, stedelijk gebied, weilanden, zandvlaktes, enzovoorts. Je hebt een beperkt aantal voorbeelden met de hand gelabeld, maar omdat dit intensief werk is overweeg je augmentatie toe te passen. Noem een aantal soorten transformaties die wel geschikt zijn, en een aantal die niet geschikt zijn.

Opgave 96. **

In plaats van ruis toe te voegen aan instances tijdens data augmentatie is het ook mogelijk om tijdens het trainen een klein beetje ruis toe te voegen aan de uitvoer van elk neuron in een netwerk zelf. De post-activatiewaarde wordt dan berekend als $h_j = \varphi\left(\sum_i w_{ij} \cdot x_i\right) + \varepsilon_j$, waarin ε_j voor elke instance opnieuw getrokken wordt uit een random verdeling. Wanneer je niet traint, tijdens het doen van voorspellingen op nieuwe data, voeg je echter geen ruis toe. Motiveer of je dit zou kunnen beschouwen als een regularisatiemethode.

5.4. Weight regularisation

Een eenvoudig idee om overfitting tegen te gaan is door het model simpeler te kiezen. Als we het aantal modelparameters verlagen reduceren we echter ook de capaciteit van het model om diverse verdelingen van data te fitten. Als we tevoren niet goed weten hoe de data eruitziet, of niet goed weten hoe dat te vertalen in een simpel model, dan lopen we het risico op underfitting.

Een benadering die hieraan tegemoet komt bestaat eruit weliswaar een model op te zetten dat veel modelparameters bevat, maar tijdens het trainen het model te bestraffen als het gebruik maakt van al deze parameters. Of, omgekeerd, we stimuleren dat het model de gewichten gelijk stelt aan nul. We doen dit door de bestaande berekening van de loss volgens $l = \sum_m \mathcal{L}\left(\hat{y}_m; y_m\right)$ uit te breiden met extra termen die een positieve loss toekennen als de gewichten w_{ij} ongelijk aan nul worden gekozen. We verkrijgen dan een uitdrukking van de vorm

$$l = \lambda_1 \sum_{i,j} |w_{ij}| + \sum_m \mathcal{L}\left(\hat{y}_m; y_m\right)$$

of

$$l = \lambda_2 \sum_{i,j} w_{ij}^2 + \sum_m \mathcal{L}\left(\hat{y}_m; y_m\right)$$

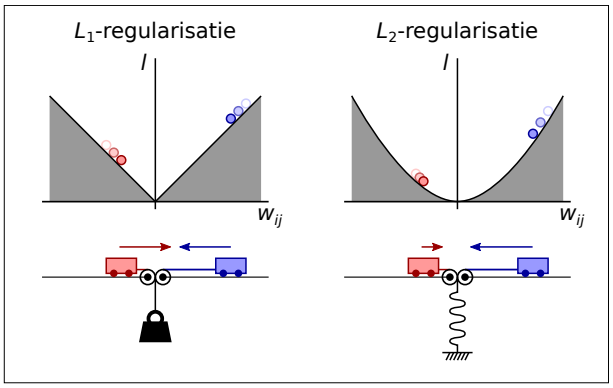
Hierin dienen we te sommeren over alle gewichten i van alle neuronen j. De coëfficiënten λ_1 en λ_2 bepalen hoe zwaar gewichten ongelijk aan nul bestraft worden. Biases worden meestal niet bestraft. De reden is dat gewichten overeenkomen met verbindingen tussen neuronen in het model, en als een verbinding een sterkte nul heeft kan deze effectief worden weggesnoeid waardoor het model eenvoudiger wordt; biases zijn meer een eigenschap van een neuron zelf, en deze snoei je niet weg.

Net als de loss-functie $\mathcal{L}$ dient de extra loss-term voor de gewichten strict positief te zijn, of eventueel nul als het gewicht nul is. In de eerste uitdrukking hierboven wordt dit bereikt door de absolute waarde te nemen van de gewichten. Deze aanpak wordt L_1-*regularisatie* genoemd, of *LASSO-regressie* als het een regressieprobleem betreft. In de tweede uitdrukking daaronder wordt voor kwadraten gekozen, hetgeen bekend staat als L_2-*regularisatie*, maar je kan ook de term *Tikhonov-regularisatie* tegenkomen of *ridge-regressie* in het geval van een regressieprobleem. De naamgeving komt van de L_1- en L_2-norm van een vector, waar we hier niet verder op ingaan, maar het cijfer 1 of 2 slaat op de macht waartoe een gewicht verheven wordt. Soms worden zowel de absolute waarden en de kwadraten tegelijkertijd gebruikt, elk met hun eigen λ-coëfficiënt, zodat

$$l = \lambda_1 \sum_{i,j} |w_{ij}| + \lambda_2 \sum_{i,j} w_{ij}^2 + \sum_m \mathcal{L}\left(\hat{y}_m; y_m\right)$$

Het resultaat wordt *elastic net regularisatie* genoemd. L_2-regularisatie is iets gangbaarder dan L_1-regularisatie, mede omdat het zich rekenkundig iets netter gedraagt doordat de kwadratische functie eenvoudig differentieerbaar is, maar de combinatie van beide is tegenwoordig steeds populairder.

De regularisatietermen kunnen denkbeeldig worden gezien als krachten die de gewichten w_{ij} naar nul trekken. L_1- en L_2-regularisatie gedragen zich in dit opzicht een beetje anders.



De absolute L_1-norm heeft altijd een even grote helling, hoe dicht je ook bij het minimum in de buurt komt. Hierdoor is ook de kracht waarmee deze de gewichten naar nul probeert te dwingen altijd even groot. Het is als het ware alsof een touwtje met een massa het gewicht naar beneden sleurt. De kwadratische L_2-norm zorgt echter voor een helling die geleidelijk afneemt naarmate je dichter bij het minimum in de buurt komt. Hierdoor is ook de kracht waarmee deze de gewichten naar nul trekt steeds kleiner naarmate de waarde dichter bij nul komt. Dit lijkt meer op een veer die steeds minder kracht uitoefent naarmate hij dichter bij de evenwichtsstand komt.

Hoe dan ook, het netto effect hiervan is dat L_2-regularisatie de gewichten wel in de buurt van nul probeert te trekken, maar deze nooit exact tot nul dwingt. Immers, hoe dichter een gewicht naar nul nadert, hoe zwakker de werking van de L_2-regularisatie wordt. L_1-regularisatie daarentegen probeert de gewichten exact op nul te krijgen. Beide methoden laten de gewichten echter in grootte krimpen, en worden dan ook wel *shrinkage methoden* genoemd; de term *weight decay* is ook gangbaar, met dezelfde betekenis. Gewichten gelijk aan nul leiden zoals gezegd tot een eenvoudiger interpreteerbaar model omdat er effectief minder verbindingen tussen neuronen aanwezig zijn. Echter, zelfs al schakelt L_2-regularisatie verbindigen nooit helemaal uit, beide vormen van regularisatie blijken overfitting tegen te gaan.

Het verkleinen van de waarde van de gewichten w_{ij} gebeurt overigens niet koste wat het kost. Immers, ook de fouten in de uiteindelijke voorspellingen leveren een loss op zoals voorgeschreven door de loss-functie $\mathcal{L}$, en als de sterkte van neurale verbindingen wordt verkleind vanwege L_1- of L_2-regularisatie gaat dit meestal ten koste van de juistheid van de voorspellingen. De gewichten kunnen dus alleen in de buurt komen van nul als de winst die geboekt kan worden vanwege het minimaliseren van de L_1- of L_2-norm opweegt tegen een slechts geringe verslechtering in de voorspellingen van het model voor de trainingsdata. Of, omgekeerd, de gewichten worden niet sterk naar nul gedwongen als dit in een te grote afname van de kwaliteit van de voorspellingen zou resulteren. Met andere woorden, verbindingen tussen neuronen worden alleen verzwakt of verwijderd als het model die specifieke verbindingen blijkbaar toch al niet zo hard nodig had.

Desalniettemin zal het introduceren van L_1- en/of L_2-regularisatie leiden tot slechtere prestaties op de trainingsdata, die echter vanwege de vereenvoudiging van het model minder snel tot overfitting zal leiden. Hiermee gedragen L_1- en L_2-regularisatie zich allebei als regularisatiemethoden, zoals de naam al aangeeft.

Opgave 97. *
Wat gebeurt er als je weight-regularisation uitvoert met coëfficiënten λ gelijk aan nul?

Opgave 98. *
Leg uit dat de loss van gewichten volgens L_2-regularisatie wel differentieerbaar is, maar die volgens L_1-regularisatie niet.

Opgave 99. **
Stel, we voeren lineaire regressie uit met een kwadratisch model $y = w_2x^2 + w_1x + w_0$, maar we hebben slechts twee datapunten: $(x, y) = (-1, +1)$ en $(x, y) = (2, 4)$. Er zijn verschillende modellen die exact door deze twee punten gaan, waaronder [i] $y = x + 2$, [ii] $y = x^2$, en [iii] $y = \frac{3}{4}x^2 + \frac{1}{4}x + \frac{1}{2}$. Welk van deze drie modellen zou de beste oplossing zijn (dat wil zeggen, de laagste loss opleveren) als je L_1-regularisatie zou toepassen, welk als je L_2-regularisatie zou toepassen. En welk wint als je géén regularisatie zou toepassen?

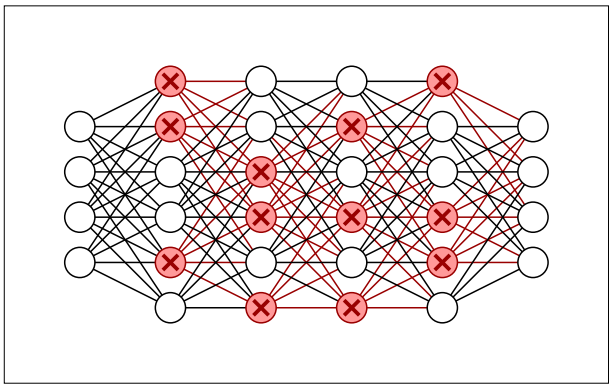
Opgave 100. ***
Stel we kijken alleen naar de loss-term van de L_2-regularisatie volgens $l = \lambda_2 \sum_{i,j} w_{ij}^2$. Laat zien dat dan de algemene updateregel van stochastic gradient descent $w_{ij} \leftarrow w_{ij} - \alpha \cdot \frac{\partial l}{\partial w_{ij}}$ leidt tot $w_{ij} \leftarrow \kappa w_{ij}$, waarbij $\kappa = 1 - 2\alpha\lambda_2$ typisch een getal is tussen nul en één. Oftewel, het effect van L_2-regularisatie in z'n eentje is dat de waarden van de gewichten tijdens elke trainingsstap worden gereduceerd met een constante vermenigvuldigingsfactor κ.

Opgave 101. ***
Stel we kijken alleen naar de loss-term van de L_1-regularisatie volgens $l = \lambda_1 \sum_{i,j} |w_{ij}|$. Laat zien dat dan de algemene updateregel van stochastic gradient descent $w_{ij} \leftarrow w_{ij} - \alpha \cdot \frac{\partial l}{\partial w_{ij}}$ leidt tot $|w_{ij}| \leftarrow |w_{ij}| - \kappa$, waarbij $\kappa = \alpha\lambda_1$ typisch een getal is tussen nul en één. Oftewel, het effect van L_1-regularisatie in z'n eentje is dat de grootten van de gewichten tijdens elke trainingsstap worden gereduceerd met een constante afname κ.

5.5. Dropout

De laatste regularisatiemethode die we bekijken in dit hoofdstuk is de minst intuïtieve. Deze is gebaseerd op het idee van ensemble learning. Dat wil zeggen, meerdere simpele machine learning modellen presteren tezamen vaak beter dan één ingewikkeld model. In het geval van neurale netwerken kunnen we opmerkelijk genoeg een simpeler model verkrijgen door willekeurig een aantal neuronen uit te schakelen. De output van deze neuronen wordt dan op nul gesteld: ze dragen niet bij tot de activatie van de neuronen in de volgende laag tijdens forward-propagation, en omdat ze geen invloed hebben op de uitkomst van de voorspelling worden ze ook niet bijgewerkt tijdens back-propagation.

Echter, afhankelijk van welke neuronen we precies uitschakelen en welke we intact laten kunnen we met dit recept een heleboel verschillende simpelere modellen genereren. Sterker nog, het gehele neurale netwerk waarin al deze neuronen wél intact zijn kun je zien als een ensemble van al deze simpelere neurale netwerkjes die elk voor zich bijdragen tot de voorspelling.



Met dit in gedachten is *dropout* een methode waarbij tijdens het trainen van een model willekeurig een zekere fractie p van de neuronen in een laag wordt behouden en de overige fractie $1-p$ wordt uitgeschakeld. Typisch wordt p gekozen tussen 0.5 en 0.8, waarbij dus een willekeurig gekozen 20 à 50% van de neuronen wordt uitgeschakeld. Andere waarden zijn ook mogelijk; er is vrij weinig bekend over wat in theorie het beste is.

Om te compenseren voor het wegvallen van een deel van de neuronen in de laag, wordt de uitvoerwaarde van alle andere neuronen vermenigvuldigd met een waarde $\frac{1}{p}$. In het geval $p = \frac{1}{2}$ bijvoorbeeld wordt de helft van de neuronen uitgeschakeld, maar wordt de uitvoer van de overlevende neuronen verdubbeld. Het idee achter deze *weight scaling rule* is dat de volgende laag in totaal evengoed "evenveel" invoer te verwerken krijgt. Hoewel dit in de praktijk goed blijkt te werken is er overigens geen theoretisch bewijs voor deze vuistregel.

In elke stap van de training wordt een andere subset aan neuronen uitgeschakeld. De neuronen in de output layer worden vanzelfsprekend onaangeroerd gelaten omdat anders niet alle voorspellingen gedaan worden. De neuronen in de input layer kunnen eventueel deels worden uitgeschakeld, maar noodzakelijk is dit niet; $p = 0.8$ is hier redelijk gebruikelijk. Verschillende hidden layers kunnen zonder bezwaar een uiteenlopende dropout rate hebben.

Wanneer tenslotte echter het model toegepast wordt op nieuwe data, dat wil zeggen de test- of validatiedata, dan worden alle neuronen wél ingeschakeld en worden hun uitvoerwaarden niet vermenigvuldigd. Oftewel, tijdens het trainen train je telkens afzonderlijke eenvoudigere subnetwerken met minder actieve neuronen erin, maar bij het toepassen van het model gebruik je wel het hele ensemble van al die subnetwerken tezamen. Het resultaat blijkt robuustere voorspellingen te geven.

Deze procedure lijkt uitermate onintuïtief. Immers, waarom zou een netwerk dat je tijdens het trainen moedwillig toetakelt beter presteren dan een intact netwerk? Dat zou bijna betekenen dat iemand met een herseninfarct beter functioneert dan iemand

met een gezond brein? Het is echter belangrijk om te realiseren dat je het model bij het toepassen op nieuwe data níet deels uitschakelt. Het is als het ware alsof je het neurale netwerk onder moeilijke omstandigheden traint om een taak uit te voeren. Dan is het wellicht niet zo vreemd dat het model het opeens verbazingwekkend goed doet als de omstandigheden niet meer moeilijk zijn zodra het ertoe doet. Vergelijk het met hoogtetraining van een atleet: de training vind plaats onder extra zware omstandigheden, maar daardoor presteer je beter als de omstandigheden normaal zijn, is het idee.

Dropout werkt omdat in een volledig neuraal netwerk neuronen soms heel delicaat op elkaar ingespeeld raken om precies de trainingsdata goed te verwerken. Dit samenspel genaamd *feature co-adaptation* blijkt niet altijd goed vertaalbaar naar nieuwe testdata. Dropout zorgt ervoor dat neuronen er niet altijd op kunnen vertrouwen dat andere neuronen in de laag ook aanwezig zijn. Daardoor worden neuronen gedwongen elk voor zich een uitvoer te produceren die erg robuust is. In zekere zin moeten de neuronen *redundante* informatie produceren, dat wil zeggen dat ze gedeeltelijk overlappende informatie doorgeven naar volgende lagen. En een dergelijke robuustere uitvoer blijkt een grotere kans te hebben om ook op nieuwe data goed te werken.

Een andere manier om tegen dropout aan te kijken is als het toevoegen van (multiplicatieve) maskeerruis, aangezien de uitvoerwaarden van sommige neuronen willekeurig op nul gesteld worden. We hebben al eerder gezien bij data augmentatie dat het toevoegen van ruis een gunstig effect kan hebben.

Een variatie van dropout is *dropconnect*; daarbij worden niet de neuronen maar de verbindingen ertussen willekeurig uitgeschakeld. Oftewel, tijdens het trainen (maar niet tijdens het toepassen) van het model wordt een willekeurige fractie aan gewichten w_{ij} gelijk aan nul gesteld. Dit lijkt een beetje op het idee achter L_1- of L_2-regularisatie.

Opnieuw zorgt het uitschakelen van neuronen ervoor dat de prestaties op de trainingsdata verminderen, maar het robuustere netwerk presteert dan vervolgens beter op de testdata. Daarmee kwalificeert het als een regularisatiemethode. Dropout heeft als grote voordelen dat het erg effectief kan zijn, eenvoudig te implementeren is, en zeer flexibel op allerlei soorten neurale netwerken kan worden toegepast.

Opgave 102. *

Waarom is het toevoegen van dropout niet zinvol als $p = 0$ of $p = 1$ gekozen wordt?

Opgave 103. **

In de figuur hierboven wordt een neuraal netwerk getoond met vier hidden layers met breedte zes waarbinnen per laag precies 50% van de neuronen wordt uitgeschakeld. Hoeveel verschillende "subnetwerken" van overlevende neuronen kunnen op die manier verkregen worden?

Opgave 104. **

Je zou dropout kunnen toepassen door de pre-activatiewaarden van een willekeurige fractie neuronen gelijk aan nul te stellen, of door dit te doen met de post-activatiewaarden. Wat zou in jouw ogen beter werken, of maakt het vermoedelijk niet veel uit? Hangt je antwoord af van de gebruikte activatiefunctie?