4. Multinomiale classificatie

In dit hoofdstuk zetten we de uitvoer van het multi-layer perceptron ten behoeve van classificatie om in probabilistische waarden die de kans op elk klasselabel aangeven. We leiden de feed-forward en back-propagation formules af voor een softmax-layer en combineren deze met de cross-entropy lossfunctie om de uitkomst robuuster te maken tegen saturatie. Het resultaat blijkt een multinomiale generalisatie van logistische regressie op te leveren, waarmee we eveneens een formulering van het oorspronkelijke perceptron in termen van activatie- en lossfuncties kunnen afleiden.

4.1. Softmax

In het vorige hoofdstuk zijn we gekomen tot een multi-layer perceptron dat in staat is om classificatie danwel regressie uit te voeren door middel van meerdere neurale lagen waarin neuronen parallel aan elkaar functioneren om uiteindelijk te komen tot een voorspelling $\hat{\boldsymbol{y}}$. In het geval van classificatie kan de uitvoer gezien worden als een vector van waarschijnlijkheden voor de verschillende mogelijke klasselabels. In dat geval wordt de gewenste uitkomst $\boldsymbol{y}$ gegeven door een one-hot encoding. Immers, de kans op het juiste klasselabel zou idealiter gelijk moeten zijn aan 100%, oftewel 1, en de kans op alle andere labels gelijk aan 0%, oftewel 0.

Tot dusverre hebben we geen bijzondere eisen gesteld aan de uitkomst van het model, behalve dan dat deze zo goed mogelijk met de gewenste uitkomsten dient overeen te komen. Wanneer de voorspellingen geïnterpreteerd dienen te worden als kansen kunnen echter een aantal randvoorwaarden worden opgelegd waaraan de voorspelling van de output layer sowieso móet voldoen. Deze omvat de volgende kenmerken:

- Omdat kansen nooit een negatieve waarde kunnen aannemen dienen de voorspellingen $\hat{y}_n$ non-negatief te zijn; dat wil zeggen, positief of eventueel gelijk aan nul.

$$\hat{y}_n \geq 0$$

- Omdat de mogelijke klasselabels elkaar uitsluiten maar er wel altijd één label juist is, dienen de voorspellingen voor alle mogelijke klasselabels $\hat{y}_n$ tezamen op te tellen tot 100%, oftewel 1.

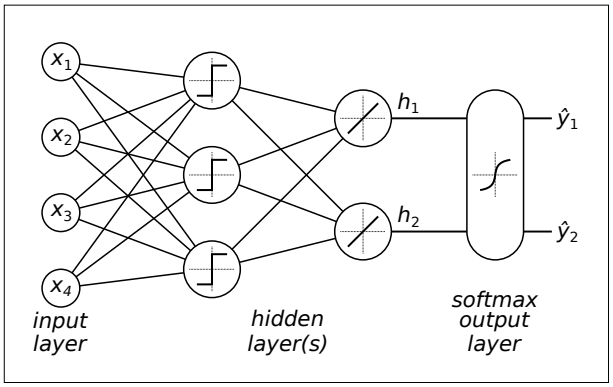
$$\sum_n \hat{y}_n = 1$$

- Omdat de uitkomst van de output neuronen rechtstreeks verband dient te houden met de voorspelde kans, eisen we dat de kans *monotoon* toeneemt met de invoerwaarden h_n van de output neuronen; dat wil zeggen, hoe hoger de invoer van een

neuron in de output layer, hoe hoger de corresponderende kans op het bijbehorende klasselabel.

$$\frac{\partial \hat{y}_m}{\partial h_m} > 0$$

Om te garanderen dat aan deze eisen wordt voldaan kan de output layer aan het einde van het neurale netwerk worden aangepast. Om redenen die nog nader zullen worden toegelicht noemen we deze een *softmax* layer. In de figuur hieronder wordt deze getoond.



Duiden we de invoer naar deze laatste laag aan met de vector $\boldsymbol{h}$, dan transformeert de softmax layer deze naar voorspellingen $\hat{\boldsymbol{y}}$ die voldoen aan de eerder genoemde drie eigenschappen. Merk op dat de softmax layer niet bestaat uit meerdere functies die parallel en onafhankelijk op één invoerwaarde elk worden toegepast. Dit komt omdat elke uitvoer $\hat{y}_n$ af zal hangen van alle invoerwaarden h_m tezamen. Om dit te benadrukken beschouwen we de softmax layer als een *vectorfunctie* die wordt toegepast op een complete vector tegelijkertijd ($\boldsymbol{h}$) en ook weer een nieuwe vector ($\hat{\boldsymbol{y}}$) oplevert.

Kansen moeten zoals gezegd non-negatief zijn. Positieve uitkomsten kunnen worden verkregen door diverse soorten transformaties toe te passen. We zijn dat eerder tegengekomen bij de lossfunctie, waarbij bijvoorbeeld gekozen werd voor het kwadraat. In dit geval zouden deze mogelijkheden echter niet tot een monotone functie leiden omdat een parabool ook een neergaande flank heeft, dus dit is in strijd met het derde genoemde kenmerk hierboven. Een functie die wel monotoon toeneemt en altijd positieve uitkomsten oplevert is bijvoorbeeld de exponentiële functie $f(x) = e^x$. Als we voor elke invoer h_n een uitvoer berekenen volgens $\hat{y}_n = e^{h_n}$ is echter nog niet automatisch voldaan aan de eis dat de som van alle kansen opgeteld 1 moet opleveren. De exponentiële functie kan immers willekeurig grote uitkomsten opleveren die het gewenste totaal van 1 ver overtreffen. Dit kan worden gecorrigeerd door de uitkomsten te *normaliseren*. Je deelt in dit geval elke individuele uitkomst door het totaal, om te komen tot onderstaande formule voor de *softmax-functie*.

$$\hat{y}_n = \frac{e^{h_n}}{\sum_j e^{h_j}}$$

De naam van deze functie is eigenlijk ietwat ongelukkig gekozen. De welbekende *max*-functie retourneert de hoogste waarde die voorkomt of mogelijk is; de *argmax*-functie geeft

de index of het argument waarvoor die maximale waarde wordt bereikt. Voor de vector $\boldsymbol{v} = [-1, 4, 2, 0]$ is bijvoorbeeld $\max(\boldsymbol{v}) = 4$, maar $\operatorname{argmax}(\boldsymbol{v}) = 2$ omdat het tweede element van de vector de hoogste waarde heeft. Wanneer indices vanaf nul geteld worden, zoals gebruikelijk in de informatica, zou gesteld kunnen worden dat $\operatorname{argmax}(\boldsymbol{v}) = 1$, of wanneer een one-hot encoding wordt gebruikt is $\operatorname{argmax}(\boldsymbol{v}) = [0, 1, 0, 0]$, waarbij de positie van het maximum als het ware met booleans wordt aangegeven. Deze laatste definitie van $\operatorname{argmax}(\boldsymbol{v})$ is geschikt om als laatste laag van het netwerk te fungeren, want deze produceert precies het type uitvoer dat we wensen. Maar deze heeft als nadeel dat deze functie discontinu is. Dat wil zeggen, een willekeurig kleine verandering in de invoer kan tot een grote verandering in de uitvoer leiden. Dergelijke discontinue functies zijn niet gewenst omdat deze een helling opleveren die enerzijds oneindig groot kan zijn, en anderzijds vaak gelijk is aan nul. Vergelijk dit met het gedrag van de signum-functie. Om dit probleem op te lossen kan de functie worden afgevlakt. De signum-functie gaf daarbij aanleiding tot bijvoorbeeld de softsign-functie; de argmax-functie leidt soortgelijk tot de softmax-functie. Deze laatste had dus eigenlijk beter de "softargmax-functie" kunnen heten, maar helaas is de naam anders ingeburgerd geraakt.

Opgave 59. *

Gegeven een vector $\boldsymbol{x} = [-1, 0, +1]$. Bereken $\operatorname{softmax}(\boldsymbol{x})$. Controleer dat de uitkomst aan de drie hierboven genoemde vereisten voldoet.

Opgave 60. *

Gegeven een vector $\boldsymbol{x} = \left[-10^6, \frac{1+\sqrt{5}}{2}, \ln(2), \pi, e\right]$. Bepaal $\max(\boldsymbol{x})$, $\operatorname{argmax}(\boldsymbol{x})$, en $\operatorname{softmax}(\boldsymbol{x})$.

Opgave 61. **

In het lijstje met vereisten voor kanswaarden wordt wel genoemd dat kansen niet negatief mogen zijn, te weten $\hat{y}_n \geq 0$, maar er wordt niet genoemd dat kansen niet groter dan 100% mogen zijn, te weten $\hat{y}_n \leq 1$. Leg uit dat dit laatste kenmerk toch reeds vanzelf uit de drie genoemde vereisten voortvloeit.

Opgave 62. **

Laat zien dat de uitkomst van de softmax-functie niet verandert als je alle invoerwaarden h_m met dezelfde waarde verhoogt of verlaagt. Dat wil zeggen, gegeven een vector $\boldsymbol{x} = [x_1, x_2, \ldots, x_m]$ en een aangepaste vector $\boldsymbol{x}' = [x_1 + c, x_2 + c, \ldots, x_m + c]$ met een willekeurige offset c, dan is $\operatorname{softmax}(\boldsymbol{x}) = \operatorname{softmax}(\boldsymbol{x}')$.

Opgave 63. ***

Laat zien dat voor alle vectoren $\boldsymbol{v}$ geldt dat $\max(\boldsymbol{v}) = \boldsymbol{v} \cdot \operatorname{argmax}(\boldsymbol{v})$, waarbij de argmax-functie een vectoriële uitvoer geeft volgens een one-hot encoding en de punt staat voor het inproduct van twee vectoren.

Opgave 64. ***

Leg uit hoezo de argmax-functie met een one-hot encoding als resultaat noch continu noch differentieerbaar is.

4.2. Optimalisatie

Zojuist is de formule afgeleid waarmee de uitvoer van de softmax layer kan worden afgeleid uit diens invoer. Dit is nodig tijdens de feed-forward fase tijdens de evaluatie van een neuraal netwerk. Tijdens de optimalisatie van een dergelijk netwerk met behulp van back-propagation dient echter ook de gradiënt van de loss naar de invoer $\boldsymbol{\nabla_h} l$ te kunnen worden bepaald als de gradiënt naar de uitvoer $\boldsymbol{\nabla_{\hat{y}}} l$ bekend is. Dit kan in principe numeriek gedaan worden door de waarde van een invoer h_m een klein beetje te veranderen en te bezien wat de invloed op alle uitvoeren $\hat{y}_n$ is. Er kan dan gesteld worden dat

$$\frac{\partial l}{\partial h_m} = \sum_n \frac{\partial l}{\partial \hat{y}_n} \frac{\partial \hat{y}_n}{\partial h_m}$$

Hierbij mag $\frac{\partial l}{\partial \hat{y}_n}$ recursief bekend worden verondersteld uit eerdere stappen van de back-propagation fase. De afgeleide $\frac{\partial \hat{y}_n}{\partial h_m}$ die aangeeft hoe sterk uitvoer nummer n van de softmax layer verandert als invoer nummer m een beetje verandert zou nu numeriek kunnen worden bepaald. Dit is vergelijkbaar met hoe de gradiënt van de loss door een neurale laag wordt teruggepropageerd. Echter, in dit geval is de functie altijd gelijk aan de softmax-functie, terwijl we bij neurale lagen met diverse soorten activatiefuncties rekening moesten houden. Daarnaast blijkt de softmax-functie een elegante afgeleide te hebben. Daarom kiezen we er hier voor om deze afgeleide eenmalig analytisch af te leiden door middel van differentiëren, in plaats van dit telkens numeriek door de computer te moeten laten doen.

We maken gebruik van de regel omtrent het differentiëren van quotiënten die zegt dat

$$\frac{\partial}{\partial x} \frac{f}{g} = \frac{g \cdot \frac{\partial f}{\partial x} - f \cdot \frac{\partial g}{\partial x}}{g^2}$$

Passen we dit toe op de softmax-functie $\hat{y}_n = \frac{e^{h_n}}{\sum_j e^{h_j}}$ dan dienen we eerst de afgeleiden van de teller en de noemer afzonderlijk te kennen.

- Voor de exponent in de teller geldt dat $\frac{\partial}{\partial h_m} e^{h_n}$ gelijk is aan nul indien $n \neq m$ omdat in dat geval de exponent helemaal niet afhangt van h_m, en gelijk is aan e^{h_n} als $m = n$ omdat de afgeleide van de exponentiële functie gelijk is aan zichzelf. Dit kan korter worden genoteerd door gebruik te maken van de *Kronecker-delta* notatie

  $$\delta_{mn} = \begin{cases} 0 \text{ voor } m \neq n \\ 1 \text{ voor } m = n \end{cases}$$

  wat dan oplevert $\frac{\partial}{\partial h_m} e^{h_n} = \delta_{mn} \cdot e^{h_n}$.

- Voor de afgeleide van de som in de noemer geldt dat $\frac{\partial}{\partial h_m} \sum_j e^{h_j} = e^{h_m}$ omdat er precies één term e^{h_m} in de som zit die afhangt van h_m: diens afgeleide is e^{h_m}, en de afgeleide van elke andere term is weer nul.

Voegen we dit allemaal samen dan vinden we

$$\begin{aligned}
\frac{\partial \hat{y}_n}{\partial h_m} &= \frac{\left(\sum_j e^{h_j}\right) \cdot \left(\delta_{mn} \cdot e^{h_n}\right) - \left(e^{h_n}\right) \cdot \left(e^{h_m}\right)}{\left(\sum_j e^{h_j}\right)^2} \\
&= \frac{\left(\sum_j e^{h_j}\right) \cdot \left(\delta_{mn} \cdot e^{h_n}\right)}{\left(\sum_j e^{h_j}\right) \cdot \left(\sum_j e^{h_j}\right)} - \frac{\left(e^{h_n}\right) \cdot \left(e^{h_m}\right)}{\left(\sum_j e^{h_j}\right) \cdot \left(\sum_j e^{h_j}\right)} \\
&= \delta_{mn} \cdot \frac{e^{h_n}}{\sum_j e^{h_j}} - \frac{e^{h_n}}{\sum_j e^{h_j}} \cdot \frac{e^{h_m}}{\sum_j e^{h_j}} \\
&= \frac{e^{h_n}}{\sum_j e^{h_j}} \left(\delta_{mn} - \frac{e^{h_m}}{\sum_j e^{h_j}}\right)
\end{aligned}$$

Dit kan worden vereenvoudigd door te herkennen dat hierin twee maal de softmax-formule $\hat{y}_n = \frac{e^{h_n}}{\sum_j e^{h_j}}$ voorkomt. We krijgen dan

$$\frac{\partial \hat{y}_n}{\partial h_m} = \hat{y}_n \cdot (\delta_{mn} - \hat{y}_m)$$

De resulterende formule beschrijft de verandering in elke voorspelling $\hat{y}_n$ als één van de invoeren h_m verandert. Hieruit blijkt dat elke uitvoer afhangt van elke invoer. Er kan worden aangetoond dat de voorspelde kans $\hat{y}_n$ op een zeker klasselabel n geleidelijk toeneemt als de daarmee overeenkomende invoer h_n toeneemt, maar dat deze kans afneemt als de andere invoeren h_m (met $m \neq n$) toenemen.

Opgave 65. *

Laat zien dat de afgeleide van de softmax-functie geschreven kan worden als

$$\frac{\partial \hat{y}_n}{\partial h_m} = \begin{cases} \hat{y}_n \cdot (1 - \hat{y}_m) & \text{voor } m = n \\ -\hat{y}_n \cdot \hat{y}_m & \text{voor } m \neq n \end{cases}$$

Opgave 66. **

Leg uit dat afgeleide van de softmax-functie ook geschreven kan worden als

$$\frac{\partial \hat{y}_n}{\partial h_m} = \hat{y}_m \cdot (\delta_{mn} - \hat{y}_n)$$

Opgave 67. **

Laat zien dat voor de softmax-functie geldt dat $\frac{\partial \hat{y}_n}{\partial h_m} > 0$ voor $m = n$ en $\frac{\partial \hat{y}_n}{\partial h_m} < 0$ voor $m \neq n$. Leg uit dat dit aantoont dat aan de derde voorwaarde omtrent de monotoniciteit van de kansverdelingsfunctie wordt voldaan.

Opgave 68. **

Gegeven een neuraal netwerk dat gebruikt wordt om data te classificeren met N verschillende klasselabels. Stel dat alle invoerwaarden naar de softmax laag identiek zijn, dat wil zeggen alle h_m zijn aan elkaar gelijk. Toon aan dat dan $\hat{y}_n = \frac{1}{N}$ voor alle klassen. Hoe groot is in dit geval $\frac{\partial \hat{y}_n}{\partial h_m}$ voor alle combinaties van n en m?

Opgave 69. ***

Toon aan dat voor de softmax-functie $\sum_n \frac{\partial \hat{y}_n}{\partial h_m} = 0$. Wat betekent dit in woorden? Hint: je kan schrijven $\sum_n \frac{\partial \hat{y}_n}{\partial h_m} = \frac{\partial}{\partial h_m}\left(\sum_n \hat{y}_n\right)$; waarvoor staat de grootheid $\sum_n \hat{y}_n$ en wat moet diens waarde dus wel zijn en blijven, ongeacht de waarde van h_m?

4.3. Dichotome classificatie

Als er maar twee mogelijke klassen zijn dan vereenvoudigt de softmax-functie tot

$$\hat{y}_1 = \frac{e^{h_1}}{e^{h_1} + e^{h_2}} = \frac{1}{1 + e^{-(h_1 - h_2)}}$$

$$\hat{y}_2 = \frac{e^{h_2}}{e^{h_1} + e^{h_2}} = \frac{1}{1 + e^{-(h_2 - h_1)}}$$

Hierin herken je misschien de logistische functie $\sigma(a) = \frac{1}{1+e^{-a}}$. We vinden dan $\hat{y}_1 = \sigma(\Delta h)$ en $\hat{y}_2 = \sigma(-\Delta h)$, met $\Delta h = h_1 - h_2$. De kans op het eerste klasselabel neemt dus geleidelijk toe volgens een sigmoïde curve naarmate de uitvoer h_1 van de output layer verder uitstijgt boven de waarde h_2. Dit is ook precies wat je zou verwachten, want h_1 codeert in zekere zin de kans op het eerste klasselabel. Tegelijkertijd is wel gegarandeerd dat $\hat{y}_1 + \hat{y}_2 = 1$, wat ook vereist is om de voorspellingen als kansen te kunnen interpreteren.

Voor de afgeleiden geldt op basis van het voorgaande dat

$$\frac{\partial \hat{y}_1}{\partial h_1} = \hat{y}_1 \cdot (1 - \hat{y}_1) = +\hat{y}_1 \hat{y}_2$$

$$\frac{\partial \hat{y}_1}{\partial h_2} = -\hat{y}_1 \cdot \hat{y}_2 = -\hat{y}_1 \hat{y}_2$$

$$\frac{\partial \hat{y}_2}{\partial h_1} = -\hat{y}_2 \cdot \hat{y}_1 = -\hat{y}_1 \hat{y}_2$$

$$\frac{\partial \hat{y}_2}{\partial h_2} = \hat{y}_2 \cdot (1 - \hat{y}_2) = +\hat{y}_1 \hat{y}_2$$

Deze hebben dus allemaal dezelfde grootte, alleen niet hetzelfde teken. Omdat $\hat{y}_1$ en $\hat{y}_2$ allebei tussen de 0 en 1 liggen volgt dat $\frac{\partial \hat{y}_1}{\partial h_1}, \frac{\partial \hat{y}_2}{\partial h_2} > 0$ en $\frac{\partial \hat{y}_1}{\partial h_2}, \frac{\partial \hat{y}_2}{\partial h_1} < 0$. Oftewel, als h_1 toeneemt dan neemt ook $\hat{y}_1$ toe maar daalt $\hat{y}_2$, terwijl als h_2 toeneemt dan stijgt $\hat{y}_2$ maar daalt $\hat{y}_1$.

De mate waarin de voorspelde kansen veranderen is klein als hetzij $\hat{y}_1$ of $\hat{y}_2$ in de buurt ligt van 0, want dan is immers ook het product $\hat{y}_1 \cdot \hat{y}_2$ klein. In dat geval is het model overtuigd van de juistheid van de classificatie en is de kans op het ene klasselabel ongeveer gelijk aan 100% en die op het andere ongeveer gelijk aan 0%. Dan is de verandering die in het model zal optreden klein. Als het model het inderdaad bij het juiste eind heeft dan is dit wenselijk, want dan wil je niet dat het model zou veranderen. Echter, als het model ernaast zit (maar het wel zeker "denkt" te weten) dan leidt dit ook nauwelijks tot

veranderingen, waardoor het model moeilijk leert van zijn fouten. Dit is een stuk minder wenselijk.

Het model leert in dit geval het snelst als het getraind wordt met instances die twijfelgevallen vormen. Dat wil zeggen, als zowel $\hat{y}_1$ en $\hat{y}_2$ ergens in de buurt liggen van $\frac{1}{2}$. In hoofdstuk 2 zagen we ook al dat uit de updateregel van het logistische regressiemodel bleek dat instances die nabij de scheidingslijn tussen de twee klassen liggen het zwaarst worden meegewogen, en instances die hier ver vandaan liggen een lagere weging krijgen. Dit is exact hetzelfde gedrag.

Kortom, een enkel neuron met een logistische functie als activatiefunctie komt in essentie op precies hetzelfde neer als een neuraal netwerk met een softmax layer als output layer. Een softmax layer heeft echter als voordeel dat deze ook toepasbaar is op multinomiale classificatie met meer dan twee klassen.

Opgave 70. *

Laat zien dat voor de logistische functie geldt dat $\sigma(a) + \sigma(-a) = 1$.

Opgave 71. ***

Toon aan dat voor de afgeleide van de logistische functie geldt dat $\frac{\partial}{\partial a}\sigma(a) = \sigma(a) \cdot (1 - \sigma(a))$.

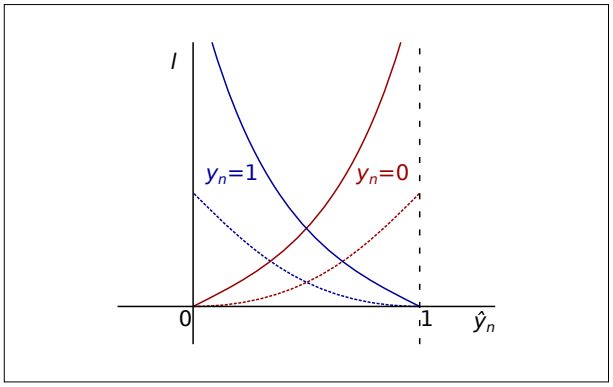
4.4. Cross-entropy

We hebben net een belangrijk nadeel ontdekt van de softmax functie zoals we die tot nu toe gebruiken. Als het model overtuigd is van de juistheid van de klasselabels, dan zullen de modelparameters maar weinig worden bijgewerkt. Dat is hiervoor aangetoond voor gevallen met twee klasselabels, maar dit geldt ook in het algemeen. Voor een deel is dit gewenst. Immers, als het model terecht een hoge kans toekent aan de juiste klasse, dan duidt dit op een goede classificatie, en dan hoef je het model helemaal niet aan te passen. Echter, als het model het weliswaar zeker meent te weten maar het eigenlijk mis heeft, dan is het ongewenst dat de gewichten niet of nauwelijks worden aangepast. Dit probleem heeft te maken met de *verzadiging* of *saturatie* die optreedt in de softmax-functie (en evenzo de logistische functie). Hiermee wordt bedoeld dat de helling uitermate klein wordt in de "staarten" van de functie. Dit betekent dat je maar weinig kan veranderen aan de uitkomsten van de functie door de parameters enigszins aan te passen, en het principe van gradient descent dicteert dan dat de modelparameters ook maar weinig aangepast zullen worden. Het model blijft dan steken in een slechte oplossing. Dit is in elk geval onwenselijk als de voorspelling onjuist is.

We zijn dit probleem eerder tegengekomen bij het perceptron, waar de signum-functie evenzo een kleine helling had. In dat geval trad het probleem in zijn meest extreme vorm op: de helling van de signum-functie is vrijwel overal exact gelijk aan nul. Voor de softmax- en logistische functies geldt dit in afgezwakte vorm. Weliswaar wordt de helling nooit helemaal nul, en is deze ook niet letterlijk overal klein, maar als een instance per ongeluk in de staart van de functie terecht komt en verkeerd geclassificeerd wordt, dan is het heel moeilijk voor het model om hiervan te herstellen.

Het probleem zit eigenlijk niet alleen in de softmax- en logistische functie. Het heeft ook te maken met de lossfunctie die we gebruiken. Stel bijvoorbeeld dat een instance eigenlijk tot klasse nummer één behoort. Dus $y_1 = 1$, en alle andere $y_n = 0$. Dan bestraffen we het model als het komt tot een voorspelling met $\hat{y}_1 \neq 1$. Als het model bijvoorbeeld een kans van slechts 10% toekent aan klasse één, in plaats van 100%, dan is $\hat{y}_1 = 0.1$ en vinden we een kwadratisch loss $l = \mathcal{L}(0.1; 1) = 0.9^2 = 0.81$. Als het model echter een kans van 0% toekent aan die instance geeft dat een loss $l = 1.00$. In het eerste geval is het model weliswaar niet nauwkeurig, maar het geeft wel nog de mogelijkheid aan dat de instance tot klasse één behoort. In het tweede geval beweert het model absoluut zeker te zijn dat het onmogelijk klasse één kan zijn. Hoewel beide modellen ernaast zitten is dat tweede model qua interpretatie veel slechter dan het eerste. Dat komt niet goed tot uiting in de loss, die nauwelijks 25% gestegen is. We zouden voor classificatie-problemen eigenlijk beter een lossfunctie hanteren die een steeds sterker toenemende loss toekent naarmate de toegekende kans op een klasselabel dat eigenlijk juist is nadert naar nul.

In de onderstaande grafiek is een dergelijk gedrag geïllustreerd. In blauw is de loss geplot van een instance die eigenlijk een hoge gewenste kans dient te worden toegekend, dat wil zeggen $y_n = 1$. Gestippeld is de waarde van de kwadratische lossfunctie. Als de voorspelling exact juist is zodat $\hat{y}_n = 1$, dan wordt een loss $l = 0$ toegekend, zoals het hoort. Als echter $\hat{y}_n$ daalt naar nul, dan stijgt de loss geleidelijk naar 1. Belangrijk is dat voor alle $\hat{y}_n$ in de buurt van nul de loss in de buurt ligt van 1.0; er wordt weinig onderscheid gemaakt tussen een ingeschatte kleine kans $\hat{y}_n > 0$ en een nulkans $\hat{y}_n = 0$. We zouden mogelijk liever een andere lossfunctie kiezen die steeds steiler loopt naarmate $\hat{y}_n = 0$ wordt genaderd, en dus *asymptotisch* tot de x-as nadert. Dit betere gedrag is geschetst met een doorlopende blauwe lijn.



Voor de rode curven geldt iets soortgelijks. Als een instance eigenlijk niet tot een zekere klasse behoort zodat $y_n = 0$, dan is het model dat $\hat{y}_n = 0$ voorspelt natuurlijk het beste. Maar een model dat absoluut zeker denkt te zijn dat het wél tot die klasse behoort met $\hat{y}_n = 1$ zou eigenlijk veel sterker moeten worden bestraft dan een ander model dat het alleen waarschijnlijk acht met bijvoorbeeld $\hat{y}_n = 0.9$.

Opnieuw zijn er tal van functies te bedenken met deze vorm. Een functie die we al eerder zijn tegengekomen (bij het vak *Introduction Datamining*) is de hoeveelheid *informatie* die een gebeurtenis met kans p met zich meebrengt. Deze is gelijk aan $H(p) =$

$-\lg(p)$. In dit geval werd de binaire logaritme met grondtal 2 gebruikt, waarbij de uitkomst kan worden uitgedrukt in *bits*. Het is ook mogelijk om de natuurlijke logaritme $-\ln(p)$ met grondtal e te kiezen of de decimale logaritme $-\log(p)$ met grondtal 10; die schelen slechts een constante factor die niet tot een andere optimale oplossing zal leiden. We zullen vanaf hier werken met de natuurlijke logaritme.

De logaritme heeft precies een staart nabij de x-as die gebruikt kan worden om het gedrag in de bovenstaande figuur te modelleren. We vinden dan:

$$\mathcal{L}(\hat{y}_n; y_n) = \begin{cases} -\ln(1-\hat{y}_n) & \text{voor } y_n = 0 \\ -\ln(\hat{y}_n) & \text{voor } y_n = 1 \end{cases}$$

Ga voor jezelf na dat dit kan worden samengevat in één uitdrukking van de volgende vorm die klopt voor zowel $y_n = 0$ als $y_n = 1$:

$$\mathcal{L}(\hat{y}_n; y_n) = -y_n \cdot \ln(\hat{y}_n) - (1-y_n) \cdot \ln(1-\hat{y}_n)$$

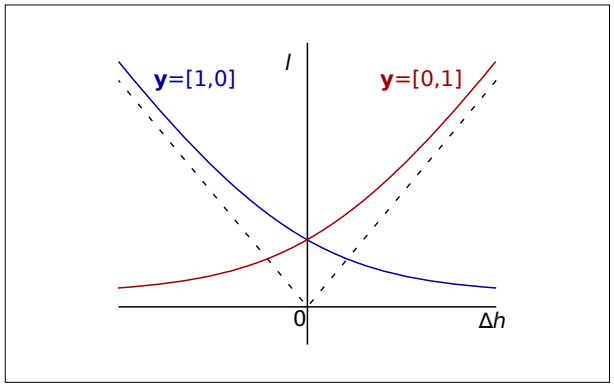
Deze formule heeft enige gelijkenis met de *entropie* $H = \sum -p \cdot \lg(p)$, zij het dat hier twee kansgrootheden $\hat{y}_n$ en y_n door elkaar worden gebruikt. De bovenstaande lossfunctie staat bekend als de *binary cross-entropy*. De cross-entropy geeft een maat voor hoe sterk de werkelijke en de geschatte kansverdelingen y_n en $\hat{y}_n$ van elkaar afwijken.

Als we meerdere uitvoeren $\hat{y}_n$ hebben wordt de loss reeds gesommeerd over al die uitvoeren: $l = \sum_n \mathcal{L}(\hat{y}_n; y_n)$. Het volstaat dan eigenlijk om de loss-functie te definiëren als

$$\mathcal{L}(\hat{y}_n; y_n) = -y_n \cdot \ln(\hat{y}_n)$$

Immers, de andere uitvoeren hebben samen een kans $1 - y_n$, dus de tweede term in de binary cross-entropy wordt al verdisconteert doordat de loss ook op die andere uitvoeren van de softmax-functie wordt toegepast. Deze vereenvoudigde versie van de cross-entropy heet de *categorical cross-entropy*. Als je de softmax-functie gebruikt en elk mogelijk klasselabel een eigen voorspelde kans heeft, dan is de categorische variant geschikt; als je maar twee klassen hebt en alleen de kans op de ene klasse voorspelt met een logistische activatiefunctie (en de andere klasse impliciet laat), dan dien je de binaire variant te kiezen.

Passen we deze lossfunctie toe op de uitkomst van de softmax layer dan krijgen we voor het eerdere vereenvoudigde dichotome geval met twee klassen $l = \sum_n \mathcal{L}(\hat{y}_n; y_n) = \mathcal{L}\left(\frac{1}{1+e^{-(h_1-h_2)}}; y_1\right) + \mathcal{L}\left(\frac{1}{1+e^{-(h_2-h_1)}}; y_2\right)$. Noemen we het verschil tussen de twee klassen $\Delta h = h_1 - h_2$, en passen we dit toe op een instance met het gewenste klasselabel één, oftewel de one-hot encoding $\boldsymbol{y} = [1, 0]$, dan leidt dit tot $l = \mathcal{L}\left(\frac{1}{1+e^{-\Delta h}}; 1\right) + \mathcal{L}\left(\frac{1}{1+e^{+\Delta h}}; 0\right)$ hetgeen in een aantal stappen vereenvoudigd kan worden tot $l = \ln\left(1 + e^{-\Delta h}\right)$. Het verloop van deze functie is hieronder geschetst in blauw.



Het gedrag van deze functie kan uitgelegd worden. Als h_1 groter is dan h_2 leidt dit tot een voorspelling waarin de hoogste kans aan het eerste klasselabel wordt gegeven. Dit is correct. In dit geval is $h_1 - h_2$ positief, en voor positieve Δh daalt de loss volgens de blauwe curve naar nul. Dat is juist, want een correcte classificatie dient een lage loss toegekend te worden. Omgekeerd wordt voor een model dat een incorrecte classificatie toekent een negatieve Δh gevonden. Hierbij hoort een toenemende loss, dus dit model wordt bestraft. Belangrijk is dat de helling van de blauwe curve aan de linkerkant niet satureert. Kortom, naarmate het model het slechter doet blijft de loss steeds verder toenemen en gaat gradient descent ervoor zorgen dat de modelparameters worden aangepast. Dit is precies het gewenste gedrag waar we naar op zoek waren. Wat er in feite gebeurt is dat de steeds vlakker lopende softmax-functie wordt gecompenseerd met een steeds steiler lopende cross-entropy. Deze twee werken elkaar precies zodanig tegen dat voor foutief geclassificeerde instances de gradiënt naar een constante nadert, maar niet langer naar nul gaat zoals bij de kwadratische lossfunctie. Voor correct geclassificeerde instances vlakt de curve wel af, maar dat is niet erg want die worden reeds juist geclassificeerd dus het model hoeft die niet nog beter te leren.

Voor een instance met het andere klasselabel $\boldsymbol{y} = [0, 1]$ geldt min of meer hetzelfde verhaal, maar dan gespiegeld. Je verkrijgt daarvoor de rode curve. De vorm van deze curve volgt nauw de *softplus-functie* $S^+(x) = \ln(1 + e^x)$. Deze heeft twee asymptoten, $S^+(x) = 0$ voor $x \downarrow -\infty$ en $S^+(x) = x$ voor $x \uparrow +\infty$. Deze curve satureert voor negatieve Δh, waarvoor het model een juiste voorspelling doet, maar satureert nooit voor positieve Δh waar het model het mis heeft. Hierdoor kan het model van zijn fouten blijven leren en is het probleem van de verzadiging van de softmax-functie verholpen.

Opgave 72. *

Stel we hebben een dichotoom classificatieprobleem met twee klassen: A en B. Schets in de vorm van een diagram twee verschillende neurale netwerken die in staat zijn om een dergelijk probleem te modelleren: eentje met één output die alleen de kans op klasse A voorspelt; en een ander met twee outputs die zowel de kans op A als de kans op B voorspelt. Hoe dien je je output layer in te richten en hoe dien je de loss-functie te kiezen in die beide gevallen?

Opgave 73. **

Gegeven is een classificatiemodel dat een voorspelling $\hat{\boldsymbol{y}} = [0.1, 0.4, 0.2, 0.3]$ toekent aan een instance die, van vier mogelijk labels A tot en met D, eigenlijk het ware label B heeft. (Merk op dat het model inderdaad de hoogste kans van 40% toekent aan dit tweede label B.) Bereken de loss als je gebruik zou maken van de kwadratische loss-functie en doe hetzelfde voor de cross-entropy loss-functie. Stel vervolgens dat het model deze uitkomst zou voorspellen voor een instance met waar label A. (Merk op dat het model dan de laagste kans van 10% toekent aan label A.) Bepaal ook dan weer de loss voor beide genoemde loss-functies.

Opgave 74. ***

Stel, we hebben een classificatieprobleem met honderd identieke instances, waarvan er 60 tot klasse A behoren en 40 tot klasse B. Laat zien dat de behaalde cross-entropy loss daadwerkelijk minimaal is als het model voor deze instances een waarschijnlijkheid $\hat{y}_A = 0.6$ toekent aan klasse A en $\hat{y}_B = 0.4$ aan klasse B.

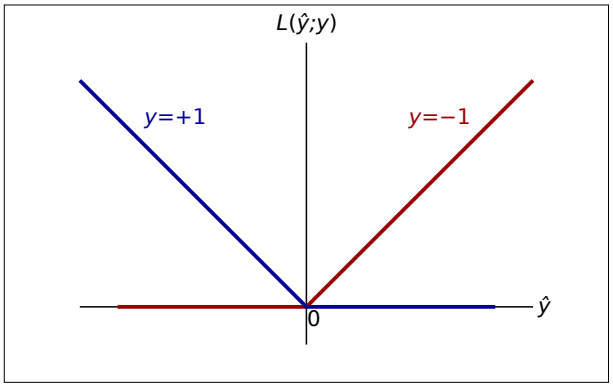
Opgave 75. ***

Leid af dat voor een instance met de gewenste one-hot classificatie $\boldsymbol{y} = [0, 1]$, gebruikmakend van een model met een hidden layer met twee uitvoerwaarden h_1 en h_2, een softmax layer, en een categorical cross-entropy lossfunctie, de totale loss gelijk is aan $l = S^+(h_1 - h_2)$.

4.5. Tot slot

Hoewel we oorspronkelijk zijn begonnen met het perceptron model en dat al doende hebben gegeneraliseerd tot algemene neuronen die veel meer kunnen dan wat het perceptron kan, zijn we er tot dusverre niet in geslaagd om het perceptron zelf te formuleren in termen van lossfuncties. Dit had ermee te maken dat de signum-functie die het perceptron gebruikt niet differentieerbaar is rond nul en helling nul heeft daarbuiten. Met het resultaat van de vorige paragraaf kunnen we echter toch komen tot een model van het perceptron op basis van lossfuncties.

Herinner je je dat we het logistische model bedacht hebben als een afgevlakte versie van het perceptron dat een te "harde" stapvormige activatiefunctie had? Het ligt voor de hand dat je omgekeerd dus het perceptron model kan verkrijgen door een afgevlakt logistisch regressie model weer "hard" te maken. Hierboven hebben we gezien dat als we de softmax-functie en de cross-entropy combineren, dat we dan een logistisch regressiemodel verkrijgen dat niet satureert. De combinatie van softmax en cross-entropy kwam neer op een softplus-functie die aan één zijde naar nul gaat en aan de andere zijde blijft hellen. Voor de softplus-functie worden deze twee regimes verbonden door een gladde overgang. Het lijkt aannemelijk dat voor het perceptron dan hetzelfde soort gedrag gekozen moet worden, maar met een harde overgang.



Hierboven staat een dergelijke "harde" functie geschetst. Hierbij zijn tijdelijk weer klasselabels van de vorm $y = \pm 1$ gebruikt omdat het perceptron hiermee werkte. Het functievoorschrift bij deze grafiek luidt $\mathcal{L}(\hat{y}; y) = \max(-\hat{y} \cdot y, 0)$. Het perceptron kan dus worden verkregen door deze lossfunctie toe te passen op een enkel neuron met de identiteitsfunctie als activatiefunctie. Merk op dat nu niet langer de signum-functie als activatiefunctie optreedt, vanwege diens genoemde problemen. De "harde" overgang die eerst in de activatiefunctie zat is nu ingebakken in de lossfunctie. In tegenstelling tot de signum-functie is echter de bovenstaande perceptron-lossfunctie wél numeriek en satureert deze niet. Wel moeten we nu de voorspellingen $\hat{y}$ iets ruimer interpreteren. Dit model produceert namelijk niet strict voorspellingen met waarden $\hat{y} = \pm 1$. Als de voorspelling $\hat{y} > 0$ dan dient dit geïnterpreteerd te worden als klasselabel $\hat{y} = +1$ en als $\hat{y} < 0$ dan duidt dit op $\hat{y} = -1$.

Voor instances met het klasselabel $y = -1$ is de lossfunctie in rood afgebeeld. Voor $\hat{y} < 0$ (links van de y-as) is de loss gelijk aan nul. Dit is wel te begrijpen, omdat een negatieve waarde van $\hat{y}$ zoals zojuist gezegd als een voorspelling van het klasselabel -1 moet worden gezien. Deze voorspelling is correct voor deze instances, dus deze mogen loss nul toegewezen krijgen. Als de voorspelling het verkeerde teken heeft (rechts van de y-as) loopt de lossfunctie echter steeds verder op. De helling van de lossfunctie is dan constant, wat ertoe leidt dat de updateregel de parameters altijd op dezelfde wijze bijwerkt, ongeacht hoe ver het punt aan de verkeerde kant van de scheidingslijn ligt. Dit is precies hoe het perceptron werkt: punten aan de verkeerde kant van de lijn leiden altijd tot eenzelfde update; punten aan de goede kant van de lijn leiden niet tot een update. Voor instances met het klasselabel $y = +1$ komt de lossfunctie neer op de blauwe lijn. Dan worden voorspellingen met $\hat{y} > 0$ niet bestraft, en is de helling constant voor $\hat{y} < 0$.

Hierboven hebben we laten zien dat het probleem van saturatie van de softmax-functie kan worden opgelost door een andere lossfunctie te kiezen. Zo leidde het gebruik van de cross-entropy ertoe dat het model samen met de softmax laag effectief een softplus-functie optimaliseert. Deze heeft als prettig kenmerk dat deze niet satureert; althans, niet voor instances die verkeerd worden geclassificeerd. Tijdens de optimalisatie van het model leidt dit ertoe dat foutief geclassificeerde instances blijven zorgen voor aanpassingen in de modelparameters.

Het genoemde probleem betreffende saturatie treedt echter niet alleen op voor de softmax-functie. Ook activatiefuncties in eerdere lagen van het neurale netwerk zijn hier

gevoelig voor. Zo heeft de tanh-functie net als de logistische functie twee staarten die geleidelijk vlak gaan lopen. Als het model per toeval in die staarten terecht komt zorgt dit ervoor dat de gewichten en biases van de neurale lagen nauwelijks meer aangepast zullen worden. Als dit voor vrijwel alle instances geldt wordt ook wel gezegd dat het neuron is *overleden*, aangezien het niet meer bijdraagt aan het leerproces.

Saturatie kan voor hidden layers niet tegengegaan worden door een andere lossfunctie te kiezen. Immers, de lossfunctie wordt pas achteraan het model toegepast en kan in de praktijk daardoor hoogstens de saturatie van de output layer beïnvloeden. Wel is het mogelijk om simpelweg een andere activatiefunctie te nemen. Jaren geleden was de tanh-functie de meest gebruikte activatiefunctie omdat deze wel enige gelijkenis toont met de manier waarop neuronen in de hersenen actief worden als ze voldoende geëxciteerd worden. Recentelijk wordt echter steeds vaker gekozen voor activatie-functies die minder gevoelig blijken te zijn voor saturatie, zoals de hierboven reeds genoemde softplus-functie.

Nog gebruikelijker is een vereenvoudigde vorm van de softplus-functie die geen gladde overgang heeft rondom nul, maar bestaat uit twee lineaire functies die met een knik aan elkaar vast zitten. Deze zou je de "hardplus-functie" kunnen noemen, maar staat beter bekend als de *rectified linear unit*, oftewel de *relu-functie*. Deze wordt gedefinieerd door $f(x) = x$ voor $x \geq 0$ en $f(x) = 0$ voor $x < 0$. Een korte manier om dit te schrijven is $f(x) = \max(x, 0)$ voor alle x. Hij werkt vergelijkbaar met een gelijkrichter diode in de electronica of een terugslagklep in een waterleiding aangezien deze signalen met een positief teken ongewijzigd doorlaat maar negatieve invoerwaarden totaal blokkeert op nul. De relu is daarmee de meest typische vertegenwoordiger van een familie *rectifier* activatiefuncties. De relu heeft als sterke punt dat zowel de functie zelf als diens afgeleide zeer eenvoudig numeriek is te bepalen, wat het een efficiënte keuze maakt.

Samengevat zijn tegenwoordig relu-functies populair als activatiefuncties in de hidden layer(s), hoewel je ook nog wel tanh-functie tegenkomt of andere varianten die hierboven genoemd zijn. Het is soms een kwestie van experimenteren, maar het kan meestal geen kwaad om te beginnen met relu-functies. Als je classificatie uitvoert worden die standaard gevolgd door een softmax layer en een loss layer gebaseerd op cross-entropy; als je regressie uitvoert zijn de laatste lagen gewoonlijk een lineaire output layer met de identiteitsfunctie als activatiefunctie en een kwadratische loss layer. Van deze keuze voor de output en loss layers wordt maar zelden afgeweken.

Hoeveel hidden layers er nodig zijn en hoe breed deze lagen dienen te zien hangt af van de complexiteit van het op te lossen probleem. Dit vereist enige ervaring om goed in te kunnen schatten, en blijft ook dan altijd een kwestie van uitproberen.

Opgave 76. *

Leg uit dat de afgeleide van de relu-functie geschreven kan worden in termen van de signum functie als $\frac{\mathrm{d}}{\mathrm{d}a}\mathrm{relu}(a) = \frac{1+\mathrm{sgn}(a)}{2}$.

Opgave 77. *

Het perceptron kan enerzijds geformuleerd worden door als activatie-functie de signum-functie $\hat{y} = \mathrm{sgn}(a)$ te nemen, of anderzijds door als loss-functie de functie $\mathcal{L}(\hat{y}; y) = \max(-\hat{y} \cdot y, 0)$ te kiezen. Bespreek van beide keuzes een voor- en nadeel.

Opgave 78. **

Zoek zelf informatie over de ELU-functie (exponential linear unit) en de SiLU-functie (Sigmoid Linear Unit). Plot hun vorm in een grafiek samen met de relu- en softplus-functies en ga na dat het allemaal varianten van rectifier-functies zijn.

Opgave 79. **

Druk de lossfunctie van het perceptron $\mathcal{L}(\hat{y}; y) = \max(-\hat{y} \cdot y, 0)$ uit in termen van de relu-functie in plaats van de max-functie.