ANHANG C

Das duale Problem bei SVMs

Um *Dualität* zu verstehen, müssen Sie zunächst die Methode der *Lagrange-Multiplikatoren* verstehen. Der generelle Ansatz ist, die Zielgröße einer Optimierung mit Nebenbedingungen in eine Optimierung ohne Nebenbedingungen zu überführen, indem die Nebenbedingungen in die Zielfunktion verschoben werden. Betrachten wir ein einfaches Beispiel. Angenommen, Sie möchten die Werte von x und y finden, die die Funktion $f(x,y) = x^2 + 2y$ minimieren, wobei *Gleichheit als Nebenbedingung* vorliegt: $3x + 2y + 1 = 0$. Bei den Lagrange-Multiplikatoren definieren wir zunächst eine neue Funktion, die sogenannte *Lagrange-Funktion* (oder *Lagrangian*): $g(x, y, \alpha) = f(x, y) - \alpha(3x + 2y + 1)$. Jede Nebenbedingung (in diesem Fall eine) wird vom ursprünglichen Zielterm subtrahiert und mit einer neuen Variablen, dem Lagrange-Multiplikator, multipliziert.

Joseph-Louis Lagrange hat Folgendes gezeigt: Falls $(\hat{x},\hat{y})$ eine Lösung zum Optimierungsproblem mit Nebenbedingungen ist, muss ein $\hat{\alpha}$ existieren, für das $(\hat{x},\hat{y},\hat{\alpha})$ ein *stationärer Punkt* des Lagrange-Terms ist. Ein stationärer Punkt liegt vor, wenn sämtliche partiellen Ableitungen gleich null sind. Anders ausgedrückt, können wir Punkte finden, an denen alle diese Ableitungen gleich null sind, indem wir die partiellen Ableitungen von $g(x, y, \alpha)$ nach x, y und α; bilden; die Lösungen dieses Optimierungsproblems mit Nebenbedingungen (falls sie existieren) müssen unter diesen stationären Punkten zu finden sein.

In unserem Beispiel sind die partiellen Ableitungen: $\begin{cases} \frac{\partial}{\partial x} g(x,y,\alpha) = 2x - 3\alpha \\ \frac{\partial}{\partial y} g(x,y,\alpha) = 2 - 2\alpha \\ \frac{\partial}{\partial \alpha} g(x,y,\alpha) = -3x - 2y - 1 \end{cases}$

Wenn alle diese partiellen Ableitungen gleich 0 sind, gilt

$$2\hat{x} - 3\hat{\alpha} = 2 - 2\hat{\alpha} = -3\hat{x} - 2\hat{y} - 1 = 0,$$

woraus wir leicht nachweisen können, dass $\hat{x} = \frac{3}{2}$, $\hat{y} = -\frac{11}{4}$ und $\hat{\alpha} = 1$ gilt. Dies ist der einzige stationäre Punkt, und da er der Nebenbedingung genügt, muss dies die Lösung des Optimierungsproblems mit Nebenbedingung sein.

Glücklicherweise lässt sich dieses Verfahren nicht nur auf Nebenbedingungen mit Gleichheitsoperator anwenden, sondern unter bestimmten Umständen (die in einer SVM vorliegen) auch auf *Nebenbedingungen mit Ungleichheit* (z. B. $3x + 2y + 1 \geqslant 0$). Die *allgemeine Lagrange-Funktion* für das Hard-Margin-Problem ist in Formel C-1 angegeben, wobei man die Variablen $\alpha^{(i)}$ als *Karush-Kuhn-Tucker*-(KKT-)Multiplikatoren bezeichnet. Diese müssen gleich oder größer null sein.

Formel C-1: Allgemeine Lagrange-Funktion für das Hard-Margin-Problem

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^\intercal\mathbf{w} - \sum_{i=1}^{m}\alpha^{(i)}(t^{(i)}(\mathbf{w}^\intercal\mathbf{x}^{(i)} + b) - 1)$$
$$\text{mit} \quad \alpha^{(i)} \geq 0 \quad \text{für } i = 1, 2, \cdots, m$$

Wie bei der Methode mit den Lagrange-Multiplikatoren können Sie die partiellen Ableitungen berechnen und die stationären Punkte ermitteln. Wenn es eine Lösung gibt, muss diese unter den stationären Punkten $(\hat{\mathbf{w}}, \hat{b}, \hat{\alpha})$ zu finden sein, die den *KKT-Bedingungen* entsprechen:

- Sie berücksichtigen die Nebenbedingungen des Problems: $t^{(i)}(\hat{\mathbf{w}}^\intercal\mathbf{x}^{(i)} + \hat{b}) \geq 1$ für $i = 1, 2, \ldots, m$.
- Für sie gilt $\hat{\alpha}^{(i)} \geq 0$ mit $i = 1, 2, \ldots, m$.
- Entweder gilt $\hat{\alpha}^{(i)} = 0$, oder die i^{te} Nebenbedingung muss eine *aktive Nebenbedingung* sein, was bedeutet, dass für sie Gleichheit gilt: $t^{(i)}(\hat{\mathbf{w}}^\intercal\mathbf{x}^{(i)} + \hat{b}) = 1$. Diese Bedingung nennt man die *Complementary-Slackness*-Bedingung. Sie impliziert, dass entweder $\hat{\alpha}^{(i)} = 0$ gilt oder der i^{te} Datenpunkt auf der Grenze liegt (also ein Stützvektor ist).

Beachten Sie, dass die KKT-Bedingungen notwendige Bedingungen dafür sind, dass ein stationärer Punkt eine Lösung des Optimierungsproblems mit Nebenbedingungen darstellt. Unter bestimmten Umständen sind diese Bedingungen auch hinreichend. Glücklicherweise ist das beim Optimierungsproblem in einer SVM der Fall, sodass jeder stationäre Punkt, der den KKT-Bedingungen entspricht, garantiert eine Lösung des Optimierungsproblems mit Nebenbedingungen darstellt.

Wir können die partiellen Ableitungen der allgemeinen Lagrange-Funktion nach $\mathbf{w}$ und b mit Formel C-2 berechnen.

Formel C-2: Partielle Ableitungen der allgemeinen Lagrange-Funktion

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}, b, \alpha) = \mathbf{w} - \sum_{i=1}^{m}\alpha^{(i)}t^{(i)}\mathbf{x}^{(i)}$$
$$\frac{\partial}{\partial b}\mathcal{L}(\mathbf{w}, b, \alpha) = -\sum_{i=1}^{m}\alpha^{(i)}t^{(i)}$$

Wenn diese partiellen Ableitungen gleich 0 sind, so gilt Formel C-3.

Formel C-3: Eigenschaften der stationären Punkte

$$\hat{\mathbf{w}} = \sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \mathbf{x}^{(i)}$$
$$\sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} = 0$$

Wenn wir dieses Ergebnis in die Definition der allgemeinen Lagrange-Funktion einfließen lassen, verschwinden einige Terme, und wir erhalten Formel C-4.

Formel C-4: Duale Form des SVM-Problems

$$\mathcal{L}(\widehat{\mathbf{w}}, \widehat{b}, \alpha) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} t^{(i)} t^{(j)} \mathbf{x}^{(i)\intercal} \mathbf{x}^{(j)} \quad - \quad \sum_{i=1}^{m} \alpha^{(i)}$$
$$\text{mit} \quad \alpha^{(i)} \geq 0 \quad \text{für } i = 1, 2, \cdots, m$$

Das Ziel ist nun, den Vektor $\hat{\alpha}$ zu finden, der diese Funktion minimiert, wobei für alle Datenpunkte $\hat{\alpha}^{(i)} \geq 0$ gilt. Dieses Optimierungsproblem mit Nebenbedingungen ist das gesuchte duale Problem.

Sobald Sie das optimale $\hat{\alpha}$ gefunden haben, können Sie $\hat{\mathbf{w}}$ berechnen, indem Sie die erste Zeile von Formel C-3 verwenden. Um $\hat{b}$ zu berechnen, können Sie sich zunutze machen, dass für einen Supportvektor $t^{(i)}(\mathbf{w}^\intercal \cdot \mathbf{x}^{(i)} + b) = 1$ gilt. Wenn also der k^{te} Datenpunkt ein Stützvektor ist (also $\alpha_k > 0$), können Sie ihn verwenden, um $\widehat{b} = t^{(k)} - \widehat{\mathbf{w}}^\intercal \mathbf{x}^{(k)}$ zu berechnen. Allerdings zieht man oft die Berechnung des Mittelwerts über alle Stützvektoren vor, um einen stabileren und genaueren Wert zu erhalten, wie in Formel C-5 angegeben.

Formel C-5: Abschätzung des Bias-Terms über die duale Form

$$\widehat{b} = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} [t^{(i)} - \widehat{\mathbf{w}}^\intercal \mathbf{x}^{(i)}]$$

ANHANG E

Weitere verbreitete Architekturen neuronaler Netze

In diesem Anhang möchten wir einen kurzen Überblick über einige historisch wichtige Architekturen neuronaler Netze geben, die heute deutlich seltener als mehrschichtige Perzeptrons (Kapitel 10), Convolutional Neural Networks (Kapitel 14), Recurrent Neural Networks (Kapitel 15) oder Autoencoder (Kapitel 17) eingesetzt werden. Sie werden aber häufig in der Fachliteratur erwähnt und in manchen Gebieten noch immer eingesetzt, sodass es sich lohnt, sie zu kennen. Wir werden außerdem *Deep Belief Networks* kennenlernen, die bis in die frühen 2010er der letzte Schrei waren. Sie werden noch immer sehr aktiv erforscht. Es ist daher möglich, dass sie in Zukunft zurückkehren, um sich zu rächen.

Hopfield-Netze

Hopfield-Netze wurden erstmalig von W. A. Little im Jahr 1974 entwickelt und erfuhren durch J. Hopfield im Jahr 1982 größere Beliebtheit. Es sind *assoziative Speicher*: Man bringt ihnen zuerst einige Muster bei, und wenn sie dann ein neues Muster sehen, geben sie (hoffentlich) das ähnlichste erlernte Muster aus. Dadurch waren sie insbesondere zur Erkennung von Buchstaben nützlich, bevor sie von anderen Verfahren abgelöst wurden. Das Netz lässt sich trainieren, indem Sie ihm Bilder von Buchstaben zeigen (jedes binäre Pixel wird an ein Neuron geleitet). Wenn Sie ihm anschließend ein neues Bild eines Buchstabens zeigen, gibt es den ähnlichsten erlernten Buchstaben aus.

Hopfield-Netze sind vollständig verbundene Graphen (siehe Abbildung E-1); das heißt, jedes Neuron ist mit jedem anderen verbunden. Die Bilder im Diagramm sind 6 × 6 Pixel groß, daher sollte das neuronale Netz 36 Neuronen (und 630 Verbindungen) enthalten. Der Klarheit wegen ist ein deutlich kleineres Netz dargestellt.

Der Trainingsalgorithmus folgt der hebbschen Lernregel (siehe »Das Perzeptron«): Bei jedem Trainingsbild wird das Gewicht zwischen zwei Neuronen erhöht, wenn

jeweils beide korrespondierenden Pixel an oder aus sind, und gesenkt, falls ein Pixel an und das andere aus ist.

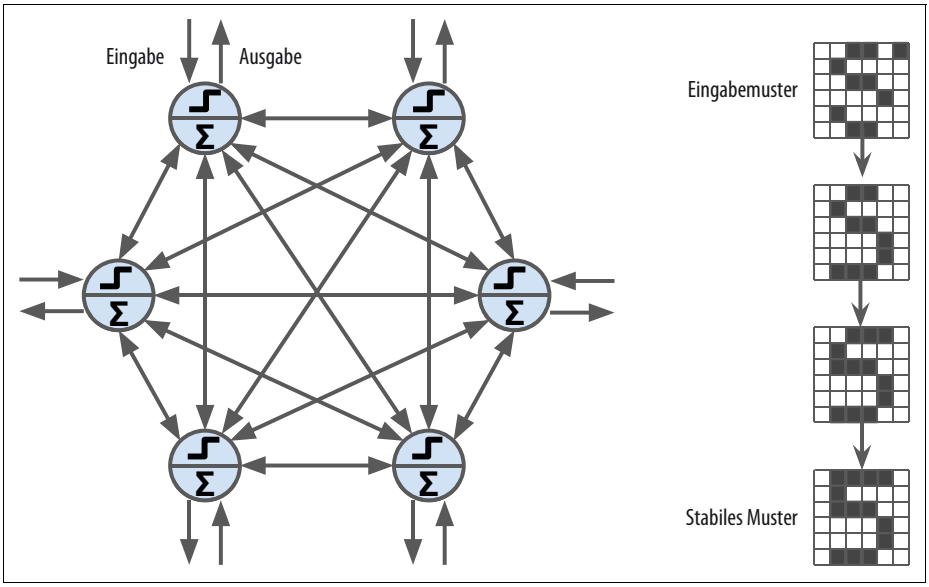

Abbildung E-1: Hopfield-Netz

Um dem Netz ein neues Bild zu zeigen, aktivieren Sie einfach die mit aktiven Pixeln korrespondierenden Neuronen. Das Netz berechnet dann die Ausgabe für jedes Neuron, und Sie erhalten ein neues Bild. Anschließend können Sie dieses Bild nehmen und den Prozess wiederholen. Nach einer Weile erreicht das Netz einen stabilen Zustand. Im Allgemeinen entspricht dieser dem Trainingsbild, das dem Eingabebild am ähnlichsten ist.

Bei Hopfield-Netzen gibt es eine sogenannte *Energiefunktion*. Die Energie wird von Iteration zu Iteration verringert, sodass das Netzwerk garantiert irgendwann einen stabilen niedrigen Energiezustand erreicht. Der Trainingsalgorithmus ändert die Gewichte derart, dass die Energiestufe der Trainingsmuster geringer ist. Damit wird es wahrscheinlich, dass sich das Netz in einer dieser energetisch günstigen Anordnungen stabilisiert. Leider können auch im Trainingsdatensatz nicht enthaltene Muster zu einem niedrigen Energiezustand führen, sodass das Netz sich bisweilen in einem nicht erlernten Zustand stabilisiert. Dies bezeichnet man auch als *unechte Muster* oder *spurious Patterns*.

Ein weiterer Nachteil von Hopfield-Netzen ist, dass sie nicht sehr gut skalieren – ihre Speicherkapazität entspricht grob 14% der Anzahl Neuronen. Um beispielsweise Bilder der Größe 28 × 28 zu klassifizieren, würden Sie ein Hopfield-Netz mit 784 vollständig verbundenen Neuronen und 306.936 Gewichten benötigen. Solch ein Netz wäre lediglich in der Lage, etwa 110 Zeichen zu unterscheiden (14% von 784). Das sind eine Menge Parameter für ein kleines Gedächtnis.

Boltzmann Machines

Boltzmann Machines wurden im Jahr 1985 von Geoffrey Hinton und Terrence Sejnowski eingeführt. Wie Hopfield-Netze sind es vollständig verbundene ANNs, basieren aber auf *stochastischen Neuronen*: Anstatt einer deterministischen Aktivierungsfunktion zur Bestimmung des Ausgabewerts geben diese Neuronen mit einer bestimmten Wahrscheinlichkeit 1 und andernfalls 0 aus. Die von diesen ANNs verwendete Wahrscheinlichkeitsfunktion beruht auf der Boltzmann-Verteilung (bekannt aus der statistischen Mechanik), daher der Name. In Formel E-1 finden Sie die Wahrscheinlichkeit, dass ein Neuron eine 1 ausgibt.

Formel E-1: Wahrscheinlichkeit, dass das i^{te} Neuron eine 1 ausgibt

$$p\left(s_i^{(\text{nächster Schritt})} = 1\right) = \sigma\left(\frac{\sum_{j=1}^{N} w_{i,j}s_j + b_i}{T}\right)$$

- s_j ist der Zustand des *j*. Neurons (0 oder 1).
- $w_{i,j}$ ist das Gewicht der Verbindung zwischen dem *i*. und *j*. Neuron. Beachten Sie, dass $w_{i,i} = 0$ gilt.
- b_i ist der Bias-Term des *i*. Neurons. Dieser lässt sich implementieren, indem wir dem Netz ein Bias-Neuron hinzufügen.
- N ist die Anzahl Neuronen im Netz.
- T ist die *Temperatur* des Netzes; je höher die Temperatur, desto zufälliger ist die Ausgabe (umso näher liegt die Wahrscheinlichkeit an 50%).
- σ ist die logistische Funktion.

Neuronen in Boltzmann Machines lassen sich in zwei Gruppen einteilen: *Visible Units* und *Hidden Units* (siehe Abbildung E-2). Alle Neuronen arbeiten auf die gleiche stochastische Weise, aber die Visible Units sind diejenigen, die die Eingabe erhalten und von denen das Ergebnis ausgelesen wird.

Aufgrund seiner stochastischen Eigenschaften stabilisiert sich eine Boltzmann Machine nicht in einer bestimmten Anordnung, sondern wechselt ständig zwischen vielen Konfigurationen hin und her. Wenn man es lange genug laufen lässt, hängt die Wahrscheinlichkeit, eine bestimmte Konfiguration zu beobachten, lediglich von den Gewichten der Verbindungen und Bias-Termen ab, nicht von der ursprünglichen Konfiguration (wenn Sie analog dazu einen Kartenstapel lange genug mischen, hängt die Anordnung der Karten nicht vom Ausgangszustand ab). Sobald das Netz diesen Zustand erreicht, in dem die ursprüngliche Konfiguration »vergessen« wurde, bezeichnet man dies als *thermisches Gleichgewicht* (auch wenn sich die Konfiguration ständig verändert). Durch entsprechendes Einstellen der Parameter, Erreichen des thermischen Gleichgewichts und anschließendes Beob-

achten des Zustands können wir ein breites Spektrum an Wahrscheinlichkeitsverteilungen simulieren. Da bezeichnet man als *generatives Modell*.

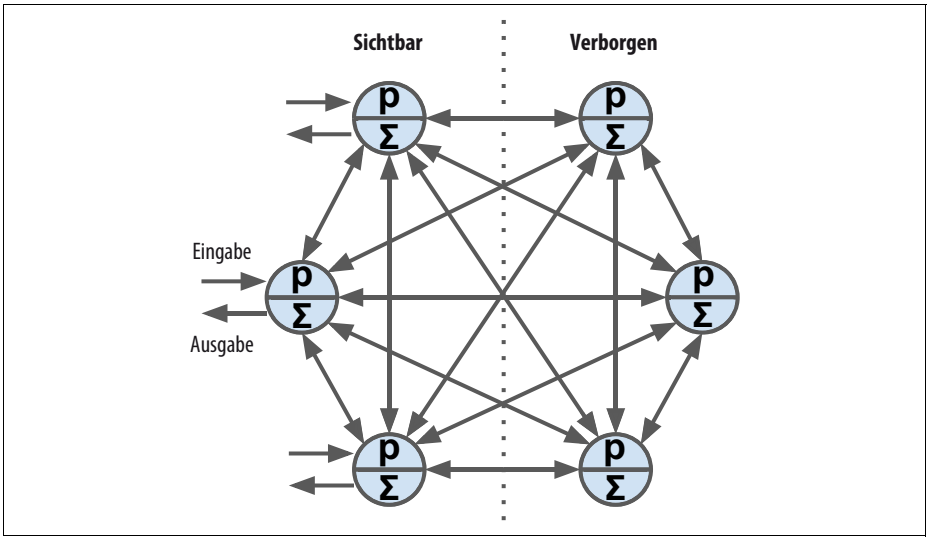

Abbildung E-2: Boltzmann Machine

Das Trainieren einer Boltzmann Machine besteht im Finden der Parameter, für die das Netz die Wahrscheinlichkeitsverteilung der Trainingsdaten approximiert. Wenn es beispielsweise drei Visible Units gibt und die Trainingsdaten 75 % (0, 1, 1)-Tripel, 10 % (0, 0, 1)-Tripel und 15 % (1, 1, 1)-Tripel enthalten, können Sie die Boltzmann Machine nach dem Trainieren einsetzen, um zufällige Tripel mit in etwa der gleichen Wahrscheinlichkeitsverteilung zu generieren. Zum Beispiel wäre die Ausgabe in etwa 75 % der Fälle ein (0, 1, 1)-Tripel.

Solch ein generatives Modell lässt sich unterschiedlich einsetzen. Wenn Sie es z. B. mit Bildern trainieren und dann ein unvollständiges oder verrauschtes Bild in das Netz einspeisen, wird es das Bild automatisch »reparieren«. Sie können ein generatives Modell auch zur Klassifikation einsetzen. Fügen Sie einfach einige Visible Units hinzu, die die Kategorie des Trainingsbilds codieren (z. B. fügen Sie zehn Visible Units hinzu und aktivieren nur das 5. Neuron, wenn das Trainingsbild die Ziffer 5 enthält). Mit einem neuen Bild aktiviert das Netz dann automatisch die entsprechenden Visible Units, die der Kategorie des Bilds entsprechen (z. B. wird das 5. Neuron aktiviert, wenn das Bild eine 5 enthält).

Leider gibt es keine effizienten Techniken, um Boltzmann Machines zu trainieren. Es gibt aber recht effiziente Algorithmen, die entwickelt wurden, um *Restricted Boltzmann Machines* (RBMs) zu trainieren.

Restricted Boltzmann Machines

Eine RBM ist nichts weiter als eine Boltzmann Machine, bei der es keine Verbindungen zwischen Visible Units untereinander oder zwischen Hidden Units untereinander gibt, nur zwischen Visible und Hidden Units. Beispielsweise stellt Abbildung E-3 eine RBM mit drei Visible Units und vier Hidden Units dar.

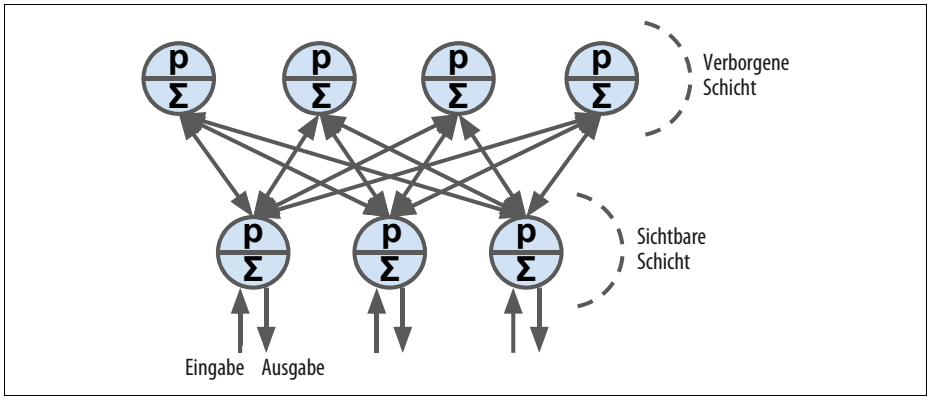

Abbildung E-3: Restricted Boltzmann Machine

Im Jahr 2005 wurde von Miguel Á. Carreira-Perpiñán und Geoffrey Hinton ein sehr effizienter Trainingsalgorithmus namens *Contrastive Divergence* vorgestellt. (*https://homl.info/135*)[1] Er funktioniert folgendermaßen: Der Algorithmus speist jeden Trainingsdatenpunkt **x** in das Netz ein, indem er den Zustand der Visible Units auf $x_1, x_2, \ldots, x_n$ setzt. Anschließend wird der Zustand der Hidden Units über die oben beschriebene stochastische Gleichung berechnet (siehe Formel E-1). Damit erhalten Sie den Hidden-Vektor **h** (wobei h_i dem Zustand des *i.* Neurons entspricht). Anschließend berechnen Sie den Zustand der Visible Units mit derselben stochastischen Gleichung. Damit erhalten Sie einen Vektor **x′**. Danach berechnen Sie wieder den Zustand der Hidden Units, wodurch Sie den Vektor **h′** erhalten. Nun können Sie mit der in Formel E-2 angegebenen Formel die Gewichte aktualisieren, wobei η die Lernrate ist.

Formel E-2: Aktualisieren der Gewichte im Contrastive-Divergence-Algorithmus

$$w_{i,j} \leftarrow w_{i,j} + \eta\left(\mathbf{x}\mathbf{h}^{\mathsf{T}} - \mathbf{x}'\,\mathbf{h}'^{\mathsf{T}}\right)$$

Der wesentliche Vorteil dieses Algorithmus ist, dass Sie nicht darauf warten müssen, dass das Netz ein thermisches Gleichgewicht erreicht: Es geht einfach nur vor, zurück, wieder vor, und das ist alles. Dadurch ist dieser Algorithmus früheren Algo-

1 Miguel Á. Carreira-Perpiñán und Geoffrey E. Hinton, »On Contrastive Divergence Learning«, *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics* (2005): 59–66.

rithmen deutlich überlegen, und er war ein wichtiger Beitrag zu den ersten Erfolgen von Deep Learning mit multiplen gestapelten RBMs.

Deep Belief Networks

RBMs lassen sich in mehreren Schichten übereinanderstapeln; die Hidden Units des ersten RBM dienen als Visible Units für das RBM der zweiten Schicht und so weiter. Solch einen Stapel RBMs nennt man auch ein *Deep-Belief-Netz* (DBN).

Yee-Whye Teh, ein Schüler Geoffrey Hintons, hatte beobachtet, dass sich DBNs mit dem Contrastive-Divergence-Algorithmus Schicht für Schicht trainieren lassen. Dabei beginnt man mit den unteren Schichten und arbeitet sich dann schrittweise zu den höheren Schichten vor. Dies führte zu dem bahnbrechenden Artikel (*https://homl.info/136*)[2], der 2006 eine Flutwelle im Deep-Learning-Bereich auslöste.

Wie RBMs lernen auch DBNs, die Wahrscheinlichkeitsverteilung der Eingabedaten ohne Überwachung zu reproduzieren. Sie sind aber viel besser darin, und zwar aus dem gleichen Grund, aus dem auch Deep Neural Networks den einfacheren Netzen überlegen sind: Realistische Daten bestehen häufig aus hierarchisch organisierten Mustern, und DBNs machen sich dies zunutze. Die niedrigeren Schichten erlernen einfachere Merkmale in den Eingabedaten, und die höheren Schichten erlernen Merkmale auf einer höheren Ebene.

Wie RBMs sind auch DBNs grundsätzlich unüberwacht. Sie lassen sich aber auch überwacht trainieren, indem Sie einige Visible Units zum Repräsentieren der Labels hinzufügen. Eine weitere herausragende Eigenschaft von DBNs ist, dass sie sich in einem halbüberwachten Modus trainieren lassen. In Abbildung E-4 ist ein für halbüberwachtes Lernen konfiguriertes DBN dargestellt.

Zunächst wird RBM 1 ohne Überwachung trainiert. Es erlernt die Merkmale der Trainingsdaten auf niedriger Ebene. Dann wird RBM 2 mit den Hidden Units von RBM 1 als Eingabe trainiert, auch diesmal ohne Überwachung: Es erlernt Merkmale auf höherer Ebene (beachten Sie, dass die Hidden Units in RBM 2 nur aus den drei Neuronen auf der rechten Seite bestehen, nicht aus den Labelneuronen). Weitere RBMs ließen sich auf diese Weise hintereinanderschalten, aber Sie verstehen das Grundprinzip. Bisher war das Training zu 100 % unüberwacht. Schließlich wird RBM 3 sowohl mit den Hidden Units von RBM 2 als Eingabe als auch mit zusätzlichen Visible Units trainiert, die die Labels repräsentieren (z. B. ein One-Hot-Vektor mit den Zielkategorien). Das Netz lernt, die abstrakteren Merkmale mit den Trainingslabels zu assoziieren. Dies ist der überwachte Schritt.

Wenn Sie RBM 1 nach dem Training einen neuen Datenpunkt vorstellen, pflanzt sich dessen Signal zu RMB 2 und danach zu RBM 3 fort und erreicht schließlich die

2 Geoffrey E. Hinton et al., »A Fast Learning Algorithm for Deep Belief Nets«, *Neural Computation* 18 (2006): 1527–1554.

Neuronen mit den Labels; mit etwas Glück leuchtet dann das richtige Label auf. Auf diese Weise lässt sich ein DBN zur Klassifikation einsetzen.

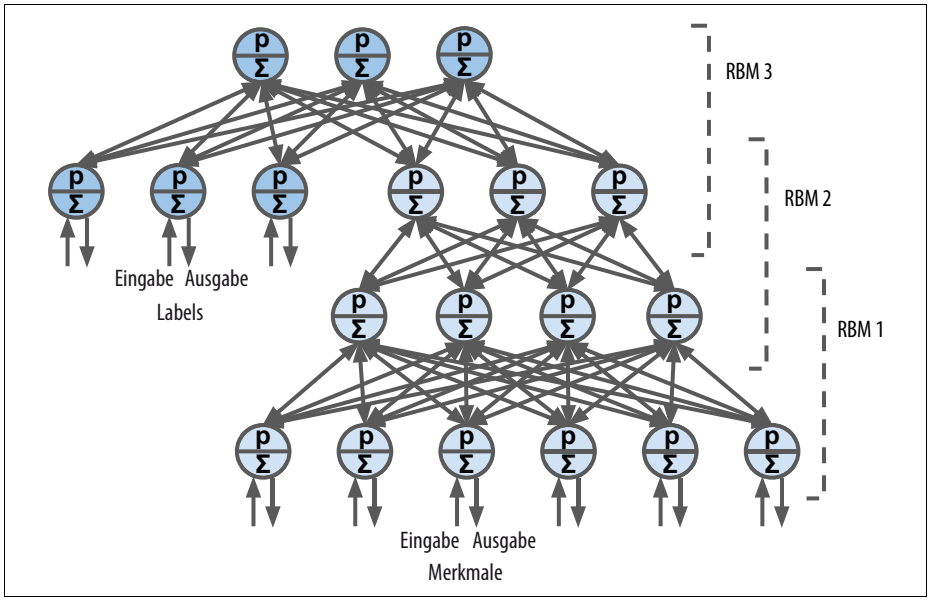

Abbildung E-4: Ein für halbüberwachtes Lernen konfiguriertes Deep Belief Network

Ein wesentlicher Vorteil dieses halbüberwachten Ansatzes ist, dass Sie nicht viele gelabelte Trainingsdaten benötigen. Wenn die unüberwachten RBMs gut funktionieren, ist nur eine geringe Anzahl gelabelter Trainingsdatenpunkte pro Kategorie notwendig. Genauso lernt ein Baby, Gegenstände ohne Überwachung zu erkennen: Wenn Sie auf einen Stuhl zeigen und »Stuhl« sagen, kann das Baby das Wort »Stuhl« mit einer ganzen Klasse von Objekten verbinden, die es bereits selbst erkennen kann. Sie müssen nicht auf jeden einzelnen Stuhl zeigen und »Stuhl« sagen; einige Beispiele reichen aus (gerade genug, somit das Baby sicher sein kann, dass Sie tatsächlich den Stuhl meinen und nicht dessen Farbe oder einen seiner Bestandteile).

Interessanterweise funktionieren DBNs auch umgekehrt. Wenn Sie eines der gelabelten Neuronen aktivieren, pflanzt sich das Signal bis zu den Hidden Units von RMB 3 fort, dann weiter zu RMB 2 und RBM 1. Am Ende geben die Visible Units von RBM 1 einen neuen Datenpunkt aus. Dieser Datenpunkt sieht in der Regel aus wie ein gewöhnlicher Vertreter der Kategorie, dessen Label Sie aktiviert haben. Diese generative Fähigkeit von DBNs ist ausgesprochen mächtig. Sie wurde z.B. eingesetzt, um Bilder automatisch zu beschriften: Ein DBN wird (ohne Überwachung) trainiert, um die Eigenschaften von Bildern zu erlernen, und ein zweites DBN wird (ebenfalls unüberwacht) trainiert, um Eigenschaften in Bildbeschriftungen zu erlernen (z.B. tritt »Auto« oft gemeinsam mit »Fahrzeug« auf). Anschließend wird auf beiden DBNs ein RBM aufgesetzt und mit einem Satz Bildern und

deren Beschriftungen trainiert; es lernt, abstrakte Merkmale der Bilder mit abstrakten Merkmalen der Beschriftungen zu assoziieren. Wenn Sie dem DBN anschließend das Bild eines Autos präsentieren, pflanzt sich das Signal durch das Netz bis zum RBM an der Spitze fort. Dann dringt es bis zum Beginn des DBN für die Beschriftungen vor, sodass Sie eine Beschriftung für das Bild erhalten. Wegen der stochastischen Eigenschaften von RBMs und DBNs ändert sich die Beschriftung zufällig, wird aber im Allgemeinen passend zum Bild sein. Wenn Sie einige Hundert Beschriftungen generieren, sind die am häufigsten erzeugten normalerweise gut.[3]

Selbstorganisierende Karten

Selbstorganisierende Karten (SOM) unterscheiden sich stark von allen anderen bisher betrachteten Arten neuronaler Netze. Sie erzeugen aus einem höher dimensionalen Datensatz eine Repräsentation mit wenigen Dimensionen, die allgemein zur Visualisierung, zum Clustern oder zur Klassifikation dient. Die Neuronen sind, wie in Abbildung E-5 gezeigt, über eine Karte verteilt (zur Visualisierung typischerweise in 2-D, es ist aber jede gewünschte Anzahl Dimensionen möglich). Jedes Neuron besitzt eine gewichtete Verbindung zu jedem Eingabewert (das Diagramm zeigt nur zwei Eingabewerte, aber deren Anzahl ist normalerweise sehr groß, da der Zweck einer SOM in der Dimensionsreduktion besteht).

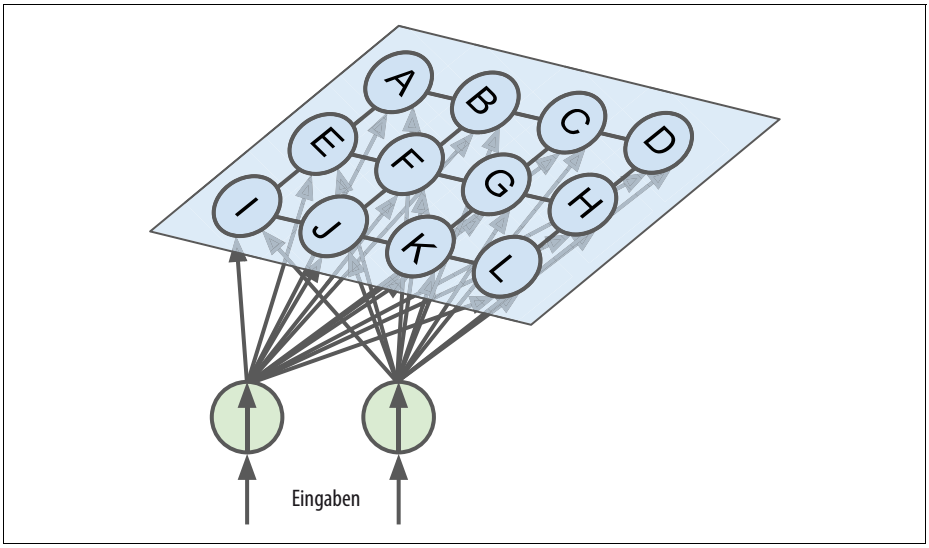

Abbildung E-5: Selbstorganisierende Karten

3 Details und eine Demonstration finden Sie in diesem Video von Geoffrey Hinton: *https://homl.info/137*.

Ist das Netz erst einmal trainiert, können Sie einen neuen Datenpunkt eingeben. Dieser aktiviert nur ein Neuron (also einen Punkt auf der Karte): das Neuron, dessen Gewichtsvektor dem Eingabevektor am ähnlichsten ist. Im Allgemeinen aktivieren im Eingaberaum ähnliche Datenpunkte nahe beieinanderliegende Neuronen auf der Karte. Das macht SOMs für die Visualisierung (insbesondere können Sie Cluster auf der Karte leicht identifizieren), aber auch für Anwendungen wie Spracherkennung so nützlich. Wenn beispielsweise jeder Datenpunkt die Tonaufnahme eines Vokals ist, werden unterschiedlich betonte Vokale »a« Neuronen im gleichen Gebiet der Karte aktivieren, während Aufnahmen des Vokals »e« Neuronen in einem anderen Gebiet aktivieren, und Geräusche dazwischen aktivieren in der Mitte gelegene Neuronen auf der Karte.

Ein wichtiger Unterschied zu den anderen in Kapitel 8 vorgestellten Verfahren zur Dimensionsreduktion ist, dass sämtliche Datenpunkte auf eine diskrete Anzahl Punkte in einem niedriger dimensionierten Raum projiziert werden (ein Punkt pro Neuron). Wenn es sehr wenige Neuronen gibt, sollte man diese Technik eher Clustering als Dimensionsreduktion nennen.

Der Trainingsalgorithmus ist unüberwacht. Er arbeitet, indem sämtliche Neuronen miteinander konkurrieren. Zuerst werden sämtliche Gewichte mit Zufallswerten initialisiert. Dann wird ein zufällig ausgewählter Datenpunkt in das Netz eingespeist. Alle Neuronen berechnen den Abstand zu ihren Gewichtsvektoren (dies ist anders als bei allen bisher betrachteten künstlichen Neuronen). Das Neuron mit dem kürzesten Abstand gewinnt und ändert sein Gewicht ein wenig in Richtung des Eingabevektors, sodass es zukünftige Wettbewerbe um ähnliche Datenpunkte noch wahrscheinlicher gewinnt. Es ermuntert auch seine Nachbarneuronen, deren Gewichte in Richtung des Eingabevektors zu ändern (diese ändern ihre Gewichte jedoch nicht so stark wie das Gewinnerneuron). Anschließend wählt der Algorithmus einen weiteren Datenpunkt aus und wiederholt den Vorgang wieder und wieder. Dieser Algorithmus führt dazu, dass sich nahe beieinandergelegene Neuronen nach und nach auf ähnliche Eingabedaten spezialisieren.[4]

4 Stellen Sie sich eine Klasse Kinder mit ähnlichen Fähigkeiten vor. Ein Kind ist vielleicht ein wenig besser im Basketball als die anderen. Das motiviert es, mehr mit seinen Freunden zu üben. Nach einer Weile wird diese Gruppe so gut im Basketball, dass ihnen die anderen Kinder nicht das Wasser reichen können. Das ist aber in Ordnung, weil sich die anderen Kinder auf andere Gebiete spezialisieren. Nach einer Weile besteht die gesamte Klasse aus kleinen Gruppen von Spezialisten.